Manual de dictado de Laboratorio de computación
Autoformación
En una tecnicatura universitaria de Energías Renovables
Pensado para la formación docente, puede ser usado por estudiantes como complemento

Diego Saravia Alía.
email:Diego.Saravia@gmail.com
ORCID: 0000–0002–1868–0409
Con asistencia de ChatGPT, GPT–5, Claude y Kimi (2026).
Salta, Argentina.

Versión 0.3

30 de agosto de 2026

Índice general

1. Hardware, software y estándares **35**

1.1. La Computación 35

1.1.1. Función de la computadora 35

1.1.2. Reseña histórica 35

1.1.3. Computadoras analógicas 36

1.1.4. Computadoras digitales: Arquitectura de Von Neumann 37

1.1.5. Redes neuronales e Inteligencia Artificial 40

1.2. Definiciones: hardware, software, máquinas de IA 40

1.2.1. Hardware 42

1.2.2. Arquitecturas de dispositivos de computación 52

1.2.3. Arquitecturas de procesadores 55

1.2.4. Software: Sistemas Operativos 58

1.2.5. Tareas de un SO 63

1.2.6. Instalación de un SO 66

1.3. Archivos: la gran abstracción 66

1.3.1. Archivos en el árbol de directorios 67

1.3.2. Archivos propiamente dichos 68

1.3.3. Archivos y directorios 68

1.3.4. Archivos de dispositivos 69

1.3.5. Rutas y nombre de archivos 69

1.3.6. Ejemplo de 'ruta' en GNU/Linux 69

1.3.7. Ejemplo de 'ruta' en un sistema Windows 70

1.3.8. Extensiones 70

1.3.9. Tipo de sistemas de archivos 70

1.3.10. Capacidades de los sistemas de archivos 70

1.3.11. El árbol genérico 71

1.3.12. El árbol estándar . . . 73
1.4. Comandos y sus Intérpretes . . . 76
1.4.1. Terminal y comandos . . . 76
1.4.2. Comandos . . . 77
1.4.3. Comandos compuestos . . . 78
1.4.4. ¿Qué es el shell? . . . 78
1.4.5. Ejemplo, comando date . . . 78
1.4.6. Algunos Comandos . . . 79
1.4.7. Redirección . . . 79
1.4.8. Man e info . . . 80
1.5. Sistema gráfico X - escritorio, gestión de ventanas . . . 80
1.5.1. Puntos básicos . . . 80
1.5.2. Cliente -– servidor, eventos . . . 81
1.5.3. Entornos gráficos de escritorio . . . 81
1.5.4. KDE . . . 81
1.5.5. Gnome . . . 81
1.5.6. Interfaz gráfica de usuario (Graphical shells), Unity . . . 82
1.5.7. Windows . . . 82
1.6. Estándares . . . 82
1.6.1. Representación de la información . . . 83
1.7. Instalación de aplicaciones . . . 85
1.7.1. Repositorios y paquetería . . . 85
1.8. Computación y sociedad . . . 89
1.8.1. Código fuente y código ejecutable . . . 89
1.8.2. Software libre . . . 90
1.8.3. Desarrollo colaborativo . . . 91
1.8.4. Ciencia abierta . . . 92
1.8.5. Transparencia y oscuridad en el software . . . 93
1.9. Referencias . . . 93
1.10. Ejercicios . . . 93
1.10.1. Instrucciones generales para los prácticos de la materia: . . . 93
•Ej. 1 Placa madre. . . . 94
•Ej. 2 Instalar un segundo SO. . . . 94

•Ej. 3 Terminal . . . 95
•Ej. 4 Man . . . 95
•Ej. 5 Procesos . . . 95
•Ej. 6 ls, grep, wc . . . 95
•Ej. 7 Permisos . . . 96
•Ej. 8 Redirección . . . 96
•Ej. 9 Directorios . . . 96
•Ej. 10 Tuberías (pipes) . . . 96
•Ej. 11 Tuberías 2 . . . 97
•Ej. 12 find . . . 97
•Ej. 13 find y grep . . . 97
•Ej. 14 Encodings: Ascii y unicode . . . 97
•Ej. 15 Encodings 2 . . . 97
•Ej. 16 Encodings 3 . . . 97
•Ej. 17 Montar pendrives USB . . . 98
•Ej. 18 tar . . . 98
•Ej. 19 gz . . . 98
•Ej. 20 xz . . . 99
•Ej. 21 Paquetes y repositorios . . . 99
•Ej. 22 Recompilar un paquete desde las fuentes del repositorio . . . 102
•Ej. 23 Licencias . . . 105
•Ej. 24 Licencias y paquetes . . . 106
•Ej. 25 Software y diferentes licencias . . . 106
•Ej. 26 4 libertades . . . 106
1.11. Laboratorio . . . 106
•Lab. Armado y desarmado de computadoras . . . 106

2. Computadoras y personas en red **109**
2.1. Redes e Internet . . . 109
2.1.1. Protocolos y el modelo TCP/IP . . . 109
2.1.2. Puertos y paquetes . . . 110
2.1.3. Direcciones IP y redes locales . . . 110
2.1.4. Ruteo . . . 111
2.1.5. Nombres y direcciones de Internet: el DNS . . . 111

2.1.6. Configuración de redes locales . . . 112
2.1.7. Comandos de red . . . 113
2.1.8. HTTP y HTTPS . . . 115
2.1.9. Correo electrónico . . . 116
2.1.10. Herramientas colaborativas en red . . . 117
2.1.11. Redes sociales . . . 117
2.1.12. Bitcoin . . . 118
2.1.13. VPN . . . 118
2.1.14. Voto electrónico . . . 119
2.2. Internet y sociedad . . . 120
2.2.1. El código es ley . . . 120
2.2.2. Libertad versus seguridad en la red . . . 121
2.2.3. Vigilancia y privacidad . . . 121
2.2.4. Anonimato . . . 121
2.2.5. Neutralidad de la red . . . 121
2.2.6. Derechos de autor y acceso al conocimiento . . . 122
2.3. Ejercicios . . . 122
•Ej. 1 Capas. TCP/IP. . . . 122
•Ej. 2 HTTP. . . . 122
•Ej. 3 TCP–UDP. . . . 122
•Ej. 4 TTL y routers. . . . 122
•Ej. 5 traceroute. . . . 123
•Ej. 6 Puertos. . . . 123
•Ej. 7 Números UO y máscaras. . . . 123
•Ej. 8 Números y máscaras 2. . . . 123
•Ej. 9 192.168. . . . 123
•Ej. 10 Ip pública. . . . 123
•Ej. 11 IPv6. . . . 124
•Ej. 12 comando ip. . . . 124
•Ej. 13 ping. . . . 124
•Ej. 14 Tabla de ruteo. . . . 124
•Ej. 15 DHCP. . . . 124
•Ej. 16 MAC address. . . . 124

- Ej. 17 dig. . . . 125
- Ej. 18 Dominios. . . . 125
- Ej. 19 Resolución inversa. . . . 125
- Ej. 20 Delegación DNS. . . . 125
- Ej. 21 /etc/hosts. . . . 125
- Ej. 22 Claves ssh. . . . 125
- Ej. 23 Clave pública. . . . 125
- Ej. 24 Servidor remoto. . . . 126
- Ej. 25 scp. . . . 126
- Ej. 26 telnet. . . . 126
- Ej. 27 wget. . . . 126
- Ej. 28 Captura de tráfico DNS. . . . 126
- Ej. 29 Captura HTTP y HTTPS. . . . 126
- Ej. 30 nmap. . . . 127
- Ej. 31 nmap propio con servidor 8080. . . . 127
- Ej. 32 Desarrollo web. . . . 127
- Ej. 33 TLS. . . . 127
- Ej. 34 curl. . . . 128
- Ej. 35 301–302. . . . 128
- Ej. 36 Ataques WiFi. . . . 128
- Ej. 37 MX. . . . 128
- Ej. 38 IMAP - POP3. . . . 128
- Ej. 39 mail. . . . 128
- Ej. 40 SPF, DKIM, DMARC. . . . 128
- Ej. 41 Correo cifrado. . . . 129
- Ej. 42 Redes sociales. . . . 129
- Ej. 43 Privacidad vs. seguridad. . . . 129
- Ej. 44 Mastodon. . . . 129
- Ej. 45 Código es ley. . . . 130
- Ej. 46 Tor. . . . 130
- Ej. 47 BitCoin. . . . 130
- Ej. 48 Neutralidad de la red. . . . 130
- Ej. 49 Sci-hub. . . . 130

•Ej. 50 V.P.N. . . . 131
2.4. Laboratorio . . . 131
•Lab. Configuración de red local, transferencia segura e interceptación de paquetes. . . . 131

3. Edición de textos **137**
3.1. Edición de textos, estándares . . . 137
3.1.1. Editores de texto plano . . . 137
3.1.2. WYGIWYW, WYSIWYM: procesadores de texto basados en comandos . . . 139
3.1.3. Editores WYSIWYG . . . 140
3.1.4. Fortalezas y debilidades de ambos modelos . . . 141
3.1.5. LaTeX . . . 141
3.1.6. Matemáticas en LaTeX . . . 143
3.1.7. BibTeX y BibLaTeX . . . 146
3.1.8. Presentaciones con Beamer . . . 150
3.1.9. Enlaces en LaTeX . . . 153
3.1.10. HTML . . . 153
3.1.11. XML . . . 154
3.1.12. PDF . . . 156
3.1.13. Markdown . . . 156
3.1.14. Conversión de formatos . . . 158
3.1.15. Planilla de cálculo . . . 158
3.1.16. Gráficos vectoriales con Inkscape . . . 160
3.1.17. Edición de imágenes con GIMP . . . 162
3.2. Estándares abiertos y cerrados . . . 165
3.3. Formatos de documentos científicos . . . 165
3.3.1. Documentos reproducibles . . . 165
3.3.2. Reproducibilidad: el problema de la publicación científica . . . 167
3.3.3. Guía rápida: qué formato usar . . . 168
3.4. Método de cuadrados mínimos . . . 168
3.5. Control de versiones . . . 170
3.5.1. Control de versiones con Git . . . 171
3.6. Git: instalación, flujo y referencia . . . 181
3.6.1. Operaciones fundamentales . . . 181

3.7. Control de versiones con Git . . . 182
3.7.1. Instalación y configuración . . . 182
3.7.2. Configuracion inicial . . . 182
3.7.3. Crear un repositorio local . . . 183
3.7.4. El flujo de trabajo basico . . . 183
3.7.5. Linux y Windows . . . 184
3.7.6. Resumen de comandos . . . 185
3.7.7. Control de versiones con Subversion . . . 185
3.8. Trabajo colaborativo entre humanos e inteligencia artificial . . . 188
3.8.1. Qué son y qué hacen . . . 188
3.8.2. Cómo pedir . . . 189
3.8.3. Cómo verificar . . . 189
3.8.4. Qué herramientas hay . . . 190
3.8.5. Ciclo de trabajo . . . 191
3.8.6. Adaptación de metodologías ágiles . . . 191
3.8.7. Definition of Done en trabajo humano–IA . . . 191
3.8.8. Roles humanos durante el uso de IA . . . 192
3.8.9. Autoría, ética y cuidado de los datos . . . 192
3.9. Ejercicios . . . 193
•Ej. 1 Instale LibreOffice y LaTeX . . . 193
•Ej. 2 Carpeta personal . . . 193
•Ej. 3 Documentos con LO . . . 193
•Ej. 4 Movimiento Rectilíneo Uniforme (MRU) . . . 193
•Ej. 5 Instale LaTeX . . . 194
•Ej. 6 HTML . . . 194
•Ej. 7 PDF . . . 195
•Ej. 8 Conversiones . . . 195
•Ej. 9 nano, vim, ed y sed . . . 195
•Ej. 10 LaTeX básico . . . 195
•Ej. 11 Matemática . . . 196
•Ej. 12 LaTeX varios . . . 196
•Ej. 13 LaTeX, beamer . . . 196
•Ej. 14 HTML . . . 196

•Ej. 15 XML . . . 197
•Ej. 16 Markdown . . . 197
•Ej. 17 Pandoc . . . 197
•Ej. 18 Formato Calc . . . 197
•Ej. 19 CSV . . . 198
•Ej. 20 CSV, cálculos . . . 198
•Ej. 21 Artículo Científico . . . 198
•Ej. 22 PDF – HTML . . . 198
•Ej. 23 WYSIWYG – WYGIWYW . . . 199
•Ej. 24 bash y Markdown . . . 199

4. Simulación **201**

4.1. Simulación de sistemas físicos . . . 201
4.1.1. El ciclo modelo–EDO–método–programa . . . 202
4.1.2. Modelos de parámetros concentrados . . . 203
4.1.3. Variables generalizadas y analogías . . . 203
4.1.4. Sentido del método . . . 204
4.2. Ciencia, filosofía, modelos y simulación: resumen . . . 204
4.3. Las variables de la física . . . 206
4.3.1. Por qué empezar por las variables . . . 206
4.3.2. Sistema material . . . 206
4.3.3. Estado, cambio, predicción . . . 206
4.3.4. Qué es una variable . . . 207
4.3.5. Tipos de variables . . . 207
4.3.6. Funciones, variables independientes y dependientes . . . 208
4.3.7. De la variable física a la variable del programa . . . 209
4.4. Herramientas de cálculo científico . . . 210
4.5. Introducción a Scilab . . . 210
4.5.1. Versiones y arquitecturas . . . 210
4.5.2. Instalación y directorio de Scilab . . . 211
4.5.3. La ventana de trabajo . . . 211
4.5.4. Entrada, ejecución y salida . . . 212
4.5.5. Directorio de trabajo y registro de sesión . . . 213
4.5.6. Uso de Scilab desde la terminal bash, extensiones .sce, .sci . . . 214

4.5.7. Ayuda de Scilab . . . 215
4.6. Conceptos de Scilab: constantes, variables, operadores . . . 216
4.6.1. Constantes . . . 216
4.6.2. Variables: nombre, dirección, tipo, tamaño . . . 216
4.6.3. Variables y operaciones básicas . . . 219
4.6.4. Números reales en la computadora . . . 220
4.6.5. Números complejos . . . 220
4.6.6. Operadores . . . 221
4.7. Vectores y matrices . . . 222
4.7.1. Definición de matrices y operaciones básicas . . . 223
4.7.2. El generador doble punto . . . 225
4.7.3. Operaciones vectoriales y matriciales; indexación . . . 226
4.8. Cadenas de caracteres . . . 228
4.8.1. Cadenas o tiras de caracteres: operaciones . . . 228
4.8.2. Evaluación de expresiones en cadenas . . . 232
4.9. Funciones definidas por el usuario . . . 234
4.9.1. `deff` . . . 235
4.9.2. Aplicar una función a un vector con `feval` . . . 236
4.10. Estructuras, contenedores y diccionarios . . . 236
4.10.1. Estructuras y objetos en Scilab . . . 236
4.10.2. list, cell, tlist y mlist . . . 237
4.10.3. Diccionarios o “hashes” en Scilab . . . 239
4.10.4. `find` . . . 240
4.10.5. Uso de vectores de caracteres como diccionarios . . . 240
4.11. Funciones predefinidas de Scilab . . . 242
4.11.1. Funciones elementales . . . 242
4.11.2. Norma de un vector . . . 242
4.11.3. Número de elementos de un vector . . . 242
4.11.4. Traza de una matriz . . . 242
4.11.5. Dimensión de una matriz . . . 243
4.11.6. Determinante de la matriz . . . 243
4.11.7. Inversa de la matriz . . . 243
4.11.8. Valores y vectores propios de una matriz . . . 243

4.11.9. Matrices especiales 244
4.11.10.Creación o extracción de una matriz diagonal 244
4.11.11.Matriz triangular 245
4.11.12.Ordenamiento de vectores 245
4.11.13.Máximo y mínimo de un vector 246
4.11.14.Uso de funciones matemáticas predefinidas en Scilab 246
4.12. Cálculo numérico 248
4.12.1. Sistema de ecuaciones lineales 248
4.12.2. Polinomios 250
4.12.3. Resolución de ecuaciones no lineales 251
4.12.4. Ajuste lineal 252
4.13. Cálculo en Scilab: derivación, integración, ecuaciones diferenciales 253
4.13.1. `diff` 253
4.13.2. Derivada 254
4.13.3. Integrales 255
4.13.4. Ecuaciones diferenciales ordinarias 259
4.14. Simulación de procesos físicos 260
4.14.1. Modelo, EDO, método, programa: el recorrido completo 260
4.14.2. Errores y comprobaciones 262
4.15. Ver y comunicar los resultados 262
4.15.1. Visualización rápida con `plot` 262
4.15.2. De Scilab a LaTeX 264
4.16. Python interactivo como calculadora vectorial 264
4.16.1. Por qué mirar Python después de Scilab 264
4.16.2. Tres consolas 264
4.16.3. Variables: nombre, tipo, tamaño, dirección 265
4.16.4. Escalares, vectores y matrices con NumPy 265
4.16.5. Operaciones relacionales y lógicas 266
4.16.6. Cadenas 266
4.16.7. Funciones predefinidas 266
4.16.8. Ayuda y espacio de trabajo 267
4.16.9. Tabla de equivalencias 267
4.17. GNU Octave interactivo como calculadora vectorial 267

4.17.1. Por qué mirar Octave después de Scilab 267
4.17.2. Dos consolas 268
4.17.3. Variables: nombre, tipo, tamaño, dirección 268
4.17.4. Escalares, vectores y matrices 269
4.17.5. Operaciones relacionales y lógicas 269
4.17.6. Cadenas 270
4.17.7. Funciones predefinidas 270
4.17.8. Ayuda y espacio de trabajo 270
4.17.9. Tabla de equivalencias 271
4.18. Perl interactivo como calculadora 271
4.18.1. Qué se puede esperar 271
4.18.2. Instalación 272
4.18.3. Consolas 272
4.18.4. Variables: nombre, tipo, tamaño, dirección 273
4.18.5. Escalares y arreglos sin PDL 273
4.18.6. PDL: la calculadora vectorial 274
4.18.7. Álgebra lineal, polinomios y ajuste 275
4.18.8. Cálculo numérico: los módulos GSL de PDL 275
4.18.9. Ecuaciones diferenciales: un módulo aparte 276
4.18.10. Cadenas 276
4.18.11. Salida a LaTeX 276
4.18.12. Ayuda 277
4.18.13. Tabla de equivalencias 277
4.19. JavaScript interactivo como calculadora 278
4.19.1. Dónde vive JavaScript 278
4.19.2. Dos consolas 278
4.19.3. Variables: nombre, tipo, tamaño, dirección 278
4.19.4. Escalares y arreglos sin biblioteca 279
4.19.5. math.js: la calculadora vectorial 279
4.19.6. Gráficos 280
4.19.7. Cadenas 280
4.19.8. Ayuda 280
4.19.9. Tabla de equivalencias 281

4.20. Instalación de software para la materia, automatizado 281
4.21. Instalación de software para el practico 4 descripto 313
4.22. Secciones expandidas, no leer en la primera lectura 317
4.22.1. Sistemas y predicción 317
4.22.2. Ciencia y filosofía 327
4.22.3. Simulación 339
4.22.4. Modelos de parámetros concentrados, circuitos 345
4.22.5. Matemáticas 357
4.23. Ejercicios 365
4.23.1. Criterio general para la resolución. 365
4.23.2. Ejercicios básicos 366
•Ej. 1 Sistema operativo y calculadora. 366
•Ej. 2 Entorno interactivo básico. 367
•Ej. 3 Consulta de ayuda y documentación. 370
•Ej. 4 Operaciones numéricas básicas. 373
•Ej. 5 Manejo de fechas y cadenas. 379
•Ej. 6 Tipos de datos y dimensiones. 380
•Ej. 7 Unidades y conversiones. 383
4.23.3. Vectores, matrices y estructuras de datos 385
•Ej. 8 Generación de vectores. 385
•Ej. 9 Producto escalar y norma. 386
•Ej. 10 Suma y resta de vectores. 387
•Ej. 11 Multiplicación matricial. 389
•Ej. 12 Cadenas de caracteres. 394
•Ej. 13 Rangos y secuencias. 397
•Ej. 14 Recta paramétrica. 400
•Ej. 15 Plano en el espacio. 401
•Ej. 16 Elipse parametrizada. 402
•Ej. 17 Indexación de matrices. 403
•Ej. 18 Álgebra matricial. 406
•Ej. 19 Independencia lineal. 408
•Ej. 20 Transformaciones lineales. 409
•Ej. 21 Autovalores de matrices. 409

•Ej. 22 Triangularización de matrices. 411
•Ej. 23 Ordenamiento de datos. 413
•Ej. 24 Resumen matricial. 415
•Ej. 25 Estructuras de mediciones. 416
•Ej. 26 Contenedores heterogéneos. 420
•Ej. 27 Diccionarios de datos. 423
•Ej. 28 Vectores paralelos. 425
4.23.4. Evaluación de cadenas 427
•Ej. 29 Expresiones dinámicas. 427
4.23.5. Lógica y máscaras 428
•Ej. 30 Máscaras de selección. 428
•Ej. 31 Umbral y filtrado. 429
•Ej. 32 Operadores lógicos. 430
•Ej. 33 Funciones definidas por tramos. 432
4.23.6. Ecuaciones lineales 433
•Ej. 34 Sistemas lineales. 433
•Ej. 35 Combinaciones lineales. 436
•Ej. 36 Cuatro incógnitas. 437
4.23.7. Ecuaciones no lineales 439
•Ej. 37 Ecuación no lineal. 439
4.23.8. Polinomios 441
•Ej. 38 Polinomios y coeficientes. 442
•Ej. 39 Producto polinomial. 444
•Ej. 40 Raíces de polinomios. 447
4.23.9. Cálculo numérico. Funciones de una línea matemática 451
•Ej. 41 Mapeo de funciones. 451
•Ej. 42 Vectorización de funciones. 454
•Ej. 43 Ajuste lineal de datos. 457
•Ej. 44 Límites numéricos. 459
•Ej. 45 Derivadas numéricas. 460
•Ej. 46 Tangente a una curva. 462
•Ej. 47 Series de Taylor. 463
•Ej. 48 Integración numérica. 465

•Ej. 49 Integrales dobles . . . 466
•Ej. 50 Ecuaciones diferenciales ordinarias . . . 468
4.23.10. LaTeX . . . 472
•Ej. 51 Exportación a LaTeX . . . 472
4.23.11. Expresiones regulares . . . 474
•Ej. 52 Dominios en expresiones regulares . . . 474
•Ej. 53 Extracción de patrones . . . 477
4.23.12. Simulación e integradores avanzados . . . 481
•Ej. 54 Enfriamiento de Newton . . . 481
•Ej. 55 Tiro parabólico sin resistencia . . . 487
•Ej. 56 Proyectil con resistencia del aire . . . 489
•Ej. 57 Oscilador armónico simple . . . 497
•Ej. 58 Crecimiento logístico poblacional . . . 503
•Ej. 59 Modelo depredador–presa de Lotka–Volterra . . . 508
•Ej. 60 Descarga de capacitor . . . 513
•Ej. 61 Caída libre con rozamiento . . . 520

5. Programación **529**

5.1. Programación . . . 529
5.1.1. De la orden suelta al programa . . . 529
5.1.2. Algoritmos y programas . . . 530
5.1.3. Interpretación de comandos y compilación . . . 530
5.1.4. Estructuras de control . . . 531
5.1.5. Funciones predefinidas . . . 533
5.1.6. Funciones definidas por el usuario . . . 534
5.1.7. Variables: alcance y tiempo de vida . . . 535
5.1.8. Paradigmas de programación . . . 537
5.1.9. Buenas prácticas como integración de los paradigmas . . . 540
5.1.10. Ingeniería del software científico . . . 540
5.1.11. Uso de inteligencia artificial para programar . . . 543
5.2. Programación, temas avanzados . . . 544
5.2.1. Funciones que recuerdan . . . 544
5.2.2. Evaluación dinámica . . . 546
5.2.3. Procesamiento de texto . . . 547

5.2.4. De texto a estructura: análisis léxico y sintáctico 548
5.2.5. Estructuras de datos más allá del arreglo 549
5.2.6. Tipos definidos por el usuario y sobrecarga 550
5.2.7. Cómo corre el programa: entorno de ejecución 550
5.3. Programación en Scilab 551
5.3.1. Por qué Scilab 551
5.3.2. La consola y el archivo 552
5.3.3. Entrada y salida 553
5.3.4. Estructuras de control 553
5.3.5. Funciones definidas por el usuario 558
5.3.6. Variables 559
5.3.7. Los paradigmas en Scilab 560
5.3.8. Buenas prácticas aplicadas 560
5.3.9. Módulos y reutilización 561
5.3.10. Control de versiones 561
5.4. Scilab: temas avanzados 561
5.4.1. Estado que sobrevive a la llamada 562
5.4.2. Currificación y aplicación parcial 563
5.4.3. Estructuras como objetos 563
5.4.4. Sobrecarga de operadores 564
5.4.5. Evaluación dinámica 564
5.4.6. Expresiones regulares 564
5.4.7. De texto a estructura 565
5.4.8. Qué no tiene Scilab 567
5.5. Panorama de lenguajes 568
5.5.1. Criterios para clasificar un lenguaje 568
5.5.2. Comparación 569
5.5.3. Un recorrido por varios lenguajes 570
5.6. Programación en Python 571
5.6.1. Qué tipo de lenguaje es 571
5.6.2. Lo que se escribe distinto 572
5.6.3. Módulos y entorno 572
5.6.4. El ecosistema científico 573

5.6.5. Buenas prácticas . . . 573
5.7. Programación en Perl . . . 574
5.7.1. Qué tipo de lenguaje es . . . 574
5.7.2. Contexto . . . 574
5.7.3. Estructuras de datos . . . 575
5.7.4. Expresiones regulares . . . 575
5.7.5. Lo que la teoría llamó avanzado . . . 575
5.7.6. Buenas prácticas . . . 576
5.7.7. Cuándo conviene y cuándo no . . . 576
5.8. Programación en el intérprete de comandos: bash . . . 576
5.8.1. Qué tipo de lenguaje es . . . 576
5.8.2. Lo mismo, escrito distinto . . . 577
5.8.3. El problema de las comillas . . . 577
5.8.4. El entorno de ejecución . . . 578
5.8.5. Buenas prácticas y cuándo dejar el shell . . . 578
5.9. Programación en C . . . 579
5.9.1. Qué tipo de lenguaje es . . . 579
5.9.2. Lo que acá se ve y en los otros no . . . 579
5.9.3. Lo que hay que emular . . . 580
5.9.4. Compilación separada y construcción . . . 581
5.9.5. Buenas prácticas . . . 581
5.9.6. Cuándo conviene . . . 581
5.10. Programación en JavaScript . . . 582
5.10.1. Qué tipo de lenguaje es . . . 582
5.10.2. Eventos y asincronía . . . 582
5.10.3. Clausuras . . . 583
5.10.4. Estructuras de datos . . . 583
5.10.5. Dentro y fuera del navegador . . . 583
5.10.6. Buenas prácticas . . . 584
5.10.7. Cuándo conviene . . . 584
5.11. Ejercicios . . . 584
5.11.1. Criterio general para la resolución. . . . 584
5.11.2. El entorno interactivo o REPL . . . 585

5.11.3. Programas, y estructuras de control 585
•Ej. 1. REPL y archivos de comandos. 585
•Ej. 2. Variables en el REPL. 591
•Ej. 3. Tiro parabólico con ciclos. 597
•Ej. 4. Calendario con meses y bisiestos. 607
•Ej. 5. Días laborables y clases. 618
•Ej. 6. Ciclos `for` y `while`. 621
•Ej. 7. Paridad de libreta. 624
•Ej. 8. Mediciones con validación de entrada. 626
•Ej. 9. Selección múltiple y ternario. 649
5.11.4. Funciones, alcance y modularización 652
•Ej. 10. Doble con validación de tipos. 652
•Ej. 11. Conversión de temperaturas. 656
•Ej. 12. Módulo de paridad. 663
•Ej. 13. Alcance de variables. 670
•Ej. 14. Normalización con contrato. 677
5.11.5. Estructuras de datos 689
•Ej. 15. Procesamiento de líneas de texto. 689
•Ej. 16. Representaciones de datos. 707
•Ej. 17. Colecciones heterogéneas. 724
•Ej. 18. Estudiantes con estructuras. 740
•Ej. 19. Alias y copias. 762
5.11.6. Calidad, pruebas y rendimiento 772
•Ej. 20. Depuración de normalización. 772
•Ej. 21. Reescritura de código. 783
•Ej. 22. Documentación con JSDoc. 791
•Ej. 23. Pruebas unitarias. 802
•Ej. 24. Rendimiento de vectorización. 825
•Ej. 25. Revisión de código de IA. 842
5.11.7. Temas avanzados de programación 848
•Ej. 26. Contador con clausura. 848
•Ej. 27. Currificación de funciones. 862
•Ej. 28. Medición con operadores. 878

•Ej. 29. Analizador de expresiones 910
5.11.8. Módulos, bibliotecas y reutilización 958
•Ej. 30. Biblioteca en módulos 958
5.11.9. Programas compilados 983
•Ej. 31. Hola mundo compilado 983

6. Entradas y salidas **985**

6.1. Entrada y salida de datos 985
6.1.1. Introducción 985
6.1.2. Panorama comparativo 986
6.1.3. Entrada y salida interactiva 987
6.1.4. Archivos de texto: lectura, escritura y actualización 989
6.1.5. Datos tabulares, persistencia e intercambio 992
6.1.6. Representación binaria y estándares de archivos 997
6.1.7. Entrada, salida y error estándar 998
6.1.8. Redirección y tuberías 999
6.1.9. Ejecución de programas externos 999
6.1.10. El sistema de archivos como interfaz del sistema operativo 1000
6.1.11. Gráficos, imágenes e informes como salida 1007
6.1.12. Flujo integrado de procesamiento 1007
6.1.13. Métodos de entrada y salida en Scilab 1008
6.1.14. Métodos de entrada y salida en Octave 1012
6.1.15. Métodos de entrada y salida en Python 1013
6.1.16. Métodos de entrada y salida en Perl 1016
6.1.17. Métodos de entrada y salida en JavaScript con Node.js 1018
6.1.18. Criterios para elegir un formato 1020
6.1.19. Resumen del flujo de trabajo 1021
6.2. Ejercicios 1022
6.2.1. Entrada y salida interactiva 1022
•Ej. 1. Carga matricial elemento por elemento 1022
•Ej. 2. Calificaciones con ingreso dinámico 1027
•Ej. 3. Separación de pares e impares 1031
•Ej. 4. Diagonal de una matriz cuadrada 1033
•Ej. 5. Norma y extremos de un vector 1035

•Ej. 6. Ensamble matricial y propiedades . . . 1038
•Ej. 7. Polinomio con raíces y gráfico . . . 1041
•Ej. 8. Saludo interactivo con nombre y apellido . . . 1044
•Ej. 9. Formato de salida numérica . . . 1045
•Ej. 10. Presentación de matrices y vectores . . . 1047
6.2.2. Archivos de texto: lectura, escritura y actualización . . . 1049
•Ej. 11. Escritura de resultados en archivo de texto . . . 1049
•Ej. 12. Interacción entre entrada, cálculo y archivos . . . 1051
•Ej. 13. Enumeración de distancias en archivo . . . 1053
•Ej. 14. Graficación de datos almacenados . . . 1055
•Ej. 15. Bitácora de operaciones . . . 1058
•Ej. 16. Monitoreo de un sensor en tiempo real . . . 1061
6.2.3. Datos tabulares, persistencia e intercambio . . . 1064
•Ej. 17. Persistencia matricial en texto, CSV y formato nativo . . . 1064
•Ej. 18. Verificación de matrices guardadas . . . 1066
•Ej. 19. Ordenamiento de registros por DNI . . . 1068
•Ej. 20. Tabulación de registros con ancho fijo . . . 1071
•Ej. 21. Correspondencia generada desde CSV . . . 1073
•Ej. 22. Normalización de datos tabulares en archivos CSV . . . 1088
•Ej. 23. Consultas relacionales sobre dos archivos CSV . . . 1092
•Ej. 24. Interoperabilidad entre formatos de matrices . . . 1103
6.2.4. Representación y estándares de archivos . . . 1106
•Ej. 25. Binarios frente a archivos de texto . . . 1106
•Ej. 26. Acceso aleatorio y actualización de un archivo . . . 1108
•Ej. 27. Firmas mágicas y tipos de archivo . . . 1109
•Ej. 28. Codificación de texto y finales de línea . . . 1111
6.2.5. Entrada y salida estándar, redirección y tuberías . . . 1112
•Ej. 29. Redirección de entrada y salida estándar . . . 1112
6.2.6. El sistema de archivos como interfaz del sistema operativo . . . 1114
•Ej. 30. Archivos especiales de GNU/Linux . . . 1114
•Ej. 31. Hardware representado por el sistema de archivos . . . 1116
•Ej. 32. Propietarios, grupos y permisos de archivos . . . 1119
6.2.7. Integración de entrada, procesamiento y salida . . . 1120

•Ej. 33. Simulación térmica de un tanque solar . . . 1120
•Ej. 34. Descarga y análisis de datos climáticos . . . 1125
•Ej. 35. Integración de lectura, análisis e informe . . . 1128

7. Gráficos – Gnuplot 1133

7.1. Programa . . . 1133
7.2. Gráficos, visualización e imágenes . . . 1134
7.2.1. Visualización como parte del cálculo científico . . . 1134
7.2.2. Dos arquitecturas para producir gráficos . . . 1135
7.2.3. Principios comunes de representación bidimensional . . . 1136
7.2.4. Escalas y sistemas de coordenadas . . . 1138
7.2.5. Datos experimentales y estadísticos . . . 1139
7.2.6. Gnuplot como motor gráfico externo . . . 1141
7.2.7. Invocación de Gnuplot desde un programa . . . 1166
7.2.8. Grillas, superficies, contornos y campos vectoriales . . . 1184
7.2.9. Series temporales, paneles, trayectorias y retratos de fase . . . 1186
7.2.10. Animación . . . 1186
7.2.11. Objetos gráficos y animación . . . 1188
7.2.12. Gráficos directos en Scilab . . . 1194
7.2.13. Gráficos directos en GNU Octave . . . 1214
7.2.14. Gráficos directos en Python . . . 1217
7.2.15. Gráficos directos en Perl . . . 1223
7.2.16. Gráficos directos en JavaScript/Node.js . . . 1225
7.2.17. Matrices representadas como imágenes . . . 1228
7.2.18. Exportación, formatos e interoperabilidad . . . 1228
7.2.19. Comparación entre Gnuplot y la vía directa . . . 1233
7.2.20. Equivalencias conceptuales entre interfaces . . . 1233
7.2.21. Un mismo problema, dos soluciones . . . 1235
7.2.22. Criterios para elegir el camino . . . 1235
7.2.23. Buenas prácticas para figuras científicas . . . 1236
7.2.24. Flujo de trabajo recomendado . . . 1237
7.3. Ejercicios . . . 1237
7.3.1. Problemas de representación gráfica . . . 1237
7.4. Ejercicios . . . 1239

7.4.1. Familiarización con Gnuplot 1239
•Ej. 1. Instalación de la herramienta gráfica externa. 1239
•Ej. 2. Familiarización con Gnuplot. 1239
7.4.2. Sección 1: Herramientas de generación de gráficos y automatización 1240
•Ej. 1. Interacción con Gnuplot: limpieza y muestreo. 1240
•Ej. 2. Ejecución de Gnuplot y exportación. 1243
•Ej. 3. Importación de archivos de datos. 1246
•Ej. 4. Invocación de Gnuplot desde lenguajes. 1248
•Ej. 5. Construcción automática de órdenes para Gnuplot. 1252
•Ej. 6. Comunicación persistente con Gnuplot mediante una tubería. 1257
7.4.3. Sección 2: Representación gráfica de funciones y datos 1265
•Ej. 7. Muestreo y estilos de una curva racional. 1265
•Ej. 8. Comparación de dos funciones racionales. 1275
•Ej. 9. Anotación de puntos notables y funciones especiales. 1282
•Ej. 10. Trazado de una función racional simple. 1286
•Ej. 11. Discontinuidades y separación de ramas. 1287
•Ej. 12. Definición de funciones y curvas múltiples. 1293
7.4.4. Sección 3: Escalas, coordenadas y datos experimentales 1299
•Ej. 13. Escalas lineales y logarítmicas. 1299
•Ej. 14. Transformación y curvas polares. 1309
•Ej. 15. Incertidumbre experimental y barras de error. 1316
•Ej. 16. Regresión lineal de datos experimentales. 1319
•Ej. 17. Distribución normal e histograma. 1323
•Ej. 18. Comparación entre gráfico de barras e histograma. 1327
•Ej. 19. Dispersión con una variable codificada por color. 1330
7.4.5. Sección 4: Composición de figuras y representaciones múltiples . . . 1332
•Ej. 20. Composición de gráficos de un proyectil. 1332
•Ej. 21. Objetos gráficos, propiedades y paneles. 1335
7.4.6. Sección 5: Representación tridimensional 1337
•Ej. 22. Superficies tridimensionales elementales. 1337
•Ej. 23. Curva paramétrica tridimensional. 1347
•Ej. 24. Perspectivas de una superficie sinc. 1350
•Ej. 25. Mapas de nivel con relleno. 1353

•Ej. 26. Refinamiento de grilla en contornos . . . 1355
•Ej. 27. Interacción con curvas de nivel mediante el ratón . . . 1370
•Ej. 28. Gradiente sobre superficie y contornos . . . 1373
•Ej. 29. Comparación de una grilla en tres representaciones . . . 1376
7.4.7. Sección 6: Visualización de modelos numéricos y animaciones . . . 1379
•Ej. 30. Oscilador amortiguado y retrato de fase . . . 1379
•Ej. 31. Animación de un péndulo simple . . . 1383
•Ej. 32. Animación mediante actualización de objetos . . . 1389
•Ej. 33. Ajuste interactivo de curvas experimentales . . . 1391
7.4.8. Sección 7: Exportación, persistencia y reproducibilidad . . . 1393
•Ej. 34. Reproducibilidad y formatos de salida . . . 1393

8. GUI-Web 1399

8.1. Interfaces Gráficas de Usuario (GUI) . . . 1399
8.1.1. Introducción . . . 1399
8.1.2. Concepto de control, evento y callback . . . 1400
8.1.3. Diálogos predefinidos . . . 1400
8.1.4. Paso de parámetros en interfaces . . . 1402
8.1.5. Interfaces de usuario . . . 1404
8.2. Formularios web y el protocolo CGI . . . 1411
8.2.1. Cómo funciona un formulario web . . . 1411
8.2.2. El protocolo CGI . . . 1414
8.2.3. Comparación de arquitecturas . . . 1417
8.2.4. Seguridad básica . . . 1417
8.3. Sistema de formularios web con motores de cálculo externos . . . 1418
8.3.1. Introducción . . . 1418
8.3.2. Instalación y uso rápido . . . 1419
8.3.3. Instalación del ejemplo Scilab como paquete . . . 1420
8.3.4. Manual de usuario . . . 1420
8.3.5. Documentación técnica . . . 1421
8.3.6. Adaptar el sistema a otra aplicación . . . 1453
8.3.7. Código fuente del ejemplo: simulador de proyectil . . . 1456
8.4. Interfaz nativa de Scilab . . . 1462
8.4.1. Uso de diálogos predefinidos . . . 1465

8.4.2. Paso de parámetros 1465
8.4.3. UI Controls 1466
8.4.4. Animación 1468
8.4.5. Procesamiento de eventos 1470
8.5. Interfaces gráficas habituales en otros lenguajes 1473
8.5.1. Cómo incorporar una gráfica a cada interfaz 1474
8.5.2. Octave: figure y uicontrol 1475
8.5.3. Python: tkinter y ttk 1476
8.5.4. Perl: Perl/Tk 1477
8.5.5. JavaScript: el DOM como GUI 1477
8.5.6. C: GTK 1478
8.5.7. Bash: Zenity 1479
8.5.8. PowerShell: Windows Forms 1480
8.5.9. Una misma lógica, distintas interfaces 1480
8.6. Jupyter como idea: documentos interactivos 1481
8.6.1. Estado, orden de ejecución e historial 1481
8.6.2. Eventos para simular fenómenos físicos en Jupyter 1483
8.7. Ejercicios 1484
8.8. Ejercicios para interfaz terminal 1484
•Ej. 1. Programa ejecutado desde la terminal. 1484
8.9. Ejercicios para interfaz web 1487
•Ej. 1. Puesta en marcha del servidor multilenguaje. 1487
•Ej. 2. Parámetros enviados al motor. 1488
•Ej. 3. Campos nuevos del formulario web. 1491
•Ej. 4. Masa en la simulación del proyectil. 1492
•Ej. 5. Color de la gráfica desde el formulario. 1494
•Ej. 6. Energía del proyectil. 1496
•Ej. 7. Descarga de un capacitor con servidor web. 1498
•Ej. 8. Validación y manejo de errores. 1501
•Ej. 9. Tiempo de respuesta del sistema. 1504
•Ej. 10. Diseño de una aplicación web científica. 1506
•Ej. 11. Oscilador web con motor de cálculo y Gnuplot. 1508
•Ej. 12. Archivos estáticos entregados por el servidor. 1511

•Ej. 13. Preguntas y problemas . . . 1512
8.10. Ejercicios para interfaces gráficas . . . 1512
•Ej. 1. Diálogos gráficos . . . 1513
•Ej. 2. Ventana para seno y coseno . . . 1514
•Ej. 3. Visualizador interactivo de funciones . . . 1515
•Ej. 4. Simulador interactivo de proyectiles . . . 1517
•Ej. 5. Regresión lineal o exponencial con interfaz gráfica . . . 1520
•Ej. 6. Corrientes de un circuito con interfaz gráfica . . . 1521
•Ej. 7. Sistema masa-resorte con interfaz gráfica . . . 1522
8.11. Ejercicios para interfaz Jupyter . . . 1523
•Ej. 1. Cuaderno interactivo tipo Jupyter en Scilab . . . 1523

9. Laboratorio de toma de datos 1527
9.1. Medición y control . . . 1527
9.1.1. El proceso de medida . . . 1527
9.1.2. Unidades, patrones y trazabilidad . . . 1528
9.1.3. Magnitudes físicas como señales eléctricas: sensores . . . 1529
9.1.4. Señales eléctricas analógicas y digitales . . . 1530
9.1.5. Conversión analógica-digital . . . 1532
9.1.6. Simulación del circuito de acondicionamiento . . . 1534
9.1.7. Errores en la medición . . . 1535
9.1.8. Calibración . . . 1538
9.1.9. Sistemas multicanales . . . 1539
9.1.10. Adquisición con la computadora . . . 1540
9.1.11. Registro, transmisión y visualización . . . 1548
9.1.12. Control . . . 1549
9.2. Hardware de adquisición y control . . . 1552
9.2.1. LM35: sensor analógico de temperatura . . . 1554
9.2.2. LDR y divisor resistivo . . . 1554
9.2.3. BH1750 y VEML7700: iluminancia – Sensores digitales integrados . 1555
9.2.4. DHT11 y DHT22 . . . 1555
9.2.5. SHT31, BMP280 y BME280: sensores ambientales digitales . . . 1556
9.2.6. DS18B20 y el bus 1-Wire – Sensores digitales integrados . . . 1556
9.2.7. Medición de distancia y tiempo de vuelo por ultrasonido . . . 1556

9.2.8. Arduino – Microcontrolador 1557
9.2.9. ESP32 – microcontrolador 1559
9.2.10. Raspberry Pi – computadora embebida 1561
9.2.11. Módulos ADAM – Adquisición industrial distribuida 1563
9.2.12. TEMPer2 – Instrumento digital completo 1566
9.2.13. Comparación de plataformas 1570
9.3. Sistema completo de adquisición y control 1571
9.4. Laboratorios 1573
•Ej. 1. Adquisicion de audio y analisis espectral 1573
•Ej. 2. Calibracion y errores de medicion 1576
•Ej. 3. Adquisicion de temperatura con LM35 1579
•Ej. 4. Adquisicion multicanal de temperatura y luz 1582
•Ej. 5. Control PID de temperatura 1586
•Ej. 6. ESP32 como servidor de sensores 1590
•Ej. 7. Sensores digitales con microcontrolador 1594
•Ej. 8. Medicion de la velocidad del sonido 1595
•Ej. 9. Raspberry Pi como centro de adquisicion y servidor 1599
•Ej. 10. Instrumentacion USB con TEMPer2 1603
•Ej. 11. Adquisicion industrial con modulos ADAM 1619
•Ej. 12. Sistema de adquisicion de datos multicanal 1623

10. Integración **1629**
10.1. Programa 1629
10.2. Proyecto final integrador 1629
10.2.1. Descripcion general 1629
10.2.2. Tipos de proyecto 1629
10.2.3. Seleccion del proyecto 1631
10.2.4. Estructura del repositorio 1632
10.2.5. Metodologia de desarrollo 1634
10.2.6. Informe tecnico 1636
10.2.7. Citando software 1638
10.2.8. Presentacion y defensa 1638
10.2.9. Criterios de evaluacion 1640
10.2.10. Consejos varios 1641

10.2.11. Cronograma sugerido 1643
10.2.12. Lista de verificacion final 1643
10.3. Ejercicios 1644
•Ej. 1. Ecuaciones diferenciales ordinarias. 1644
•Ej. 2. Enfriamiento de Newton. 1644
•Ej. 3. Modelo depredador–presa de Lotka–Volterra. 1645
•Ej. 4. Crecimiento logístico poblacional. 1645
•Ej. 5. Modelo de proyecto integrador. 1645

Referencias **1646**

Prefacio

Éste documento está conformado por material de formación docente en la materia "Laboratorio 1" (mejor nombre: de computación), carrera TUES, en el departamento de Física de la Facultad de Ciencias Exactas, Universidad Nacional de Salta.

Estoy desarrollando un software para un sitio web que permitira a los estudiantes sbir respuestas a los ejercicios y ser evaluados por dos vías:

1. una ejecutando lo que propongan, teniendo ya entornos con todos los progrmas necesarios para todos los lenguajes.-
2. otra consultando a una IA externa, para aprobar los ejercicios y relizar consultas.

Así será posible para un estudiante el cursar toda la materia en forma autónoma, salvo los laboratorios. Ya es un clasico que los primeros ejercicios se entreguen con respuestas para que los estudiantes puedan avanzar por su cuenta. Ahora se profundizará ésta forma de pensar la materia.

Espero el software esté terminado para la próxima cursada, toda las dificultades técnicas han sido resueltas.

Ya todos los ejercicios resueltos de estas guías han sido automáticamente contrastados con su ejecución automática a partir de las extracciones de las fuentes de éste documento.

Esto es particularmente importante para el dictado en Cafayate donde las dificultades económicas de la Universidad impiden pagar viáticos a los profesores.

Programa

Este programa es mucho mas exténso que lo que se puede dar en un curso habitual de estas materias. Es una guía para armar programas. El curso es para docentes que dan la materia, aunque puede serivr de consulta para estuidantes que lo necesiten.

Una forma habitual de dar este programa es dar las unidades:

- 1 uso básico y armado de la computadora,
- 2 redes con comandos del sistema operativo,
- 3 editores, planillas y presentaciones: con Libre Office y un poco de LaTeX.
- 4 calculadora, simulación con scilab.
- 5 programación con scilab.
- 6 entrada – salida y archivos con scilab.
- 7 gráficos con gnuplot y scilab acompañando. Lo que hace que los gráficos sean mucho mejores, aptos para publicación y mas simples de hacer. Además se aprende una sola vez.
- 8 interfaces gráficas con el servidor web desarrollado para esta materia como GUI y scilab (o cualquier otro).
- 9 con scilab.
- 10 es para un proyecto integrador.

Muchas veces comenzamos por la 4.

Pienso que el mejor lenguaje para un curso de este tipo es Julia, seguramente lo agregaremos a lapróxima versión.

Más allá de ésto, es casi irrelevante, lo que mejor hace la IA es traducir un progrtama que funciona a otro lenguaje.

Otra posibilidad habilitada por la IA es dar simusol en el curso, es trivial hacer modelos de simusol con IA, sobre todo si se hace un agente específico.

CONTENIDOS MÍNIMOS (actual, según el plan de estudios)

Módulo 1: Manejojo de sistemas operativos. Planillas de cálculo. Procesadores de texto. Introducción al uso de Internet. Introducción a los experimentos de Física controlados por

computadora. Módulo 2: Introducción a la simulación computarizada de procesos físicos mediante un sistema de lenguaje de programa

CONTENIDOS MÍNIMOS: propuestos

Dispositivos. hardware y software, archivos. Estándares abiertos. Control de versiones. IA. Sistemas operativos. Edición de textos, planillas de cálculo y presentaciones. Redes e Intnernet, seguridad, libertad. Medicción y control de experimentos y simulación de procesos físicos por computadora. Creación de gráficas y representación de datos y fenómenos. Interfaces de usuario. Eventos. Buenas prácticas en el software y calculadoras científicas.

Siempre pense que lo ideal sería usar éstos conocimientos para cursar álgebra. Yo no invertiria el orden y las correlatividades, las pondría como cursado simultaneo y sin correlativas entre sí.

Debiera requerir como regular Introducción a la matemática e Introduccion a la física.

Notas sobre la elección del lenguaje

Los dos bloques que siguen son notas de discusión sobre por qué se empieza con Scilab. Se conservan tal como estaban.

Una observación personal

Yo reforzaría todavía más el argumento a favor de Scilab diciendo algo que me parece importante desde el punto de vista pedagógico:

Scilab funciona como una çalculadora programable", mientras que Python es un lenguaje de programación de propósito general.

En primer año, el objetivo no es enseñar ingeniería de software, sino enseñar a pensar algoritmos y resolver problemas numéricos. En Scilab, el estudiante puede escribir en la primera clase:

A = rand(5,5); b = A ones(5,1); plot(x,sin(x))

sin haber aprendido todavía conceptos como paquetes, entornos virtuales, instalación de bibliotecas, importaciones o gestión de dependencias. Esa simplicidad reduce la carga cognitiva inicial y permite que el foco esté en la física y la matemática.

Hay un argumento adicional que me parece muy fuerte

Yo lo mencionaría explícitamente porque responde a la discusión que seguramente tienen en la Facultad.

En Ingeniería y en Ciencias no se enseña un lenguaje; se enseñan algoritmos y métodos de resolución de problemas.

El lenguaje cambia con los años.

Hace 30 años era Fortran.

Después apareció Matlab.

Hoy domina Python.

Dentro de diez años probablemente aparezca otro.

En cambio:

plantear un algoritmo; discretizar un problema; resolver un sistema lineal; integrar una ecuación diferencial; procesar datos experimentales;

son habilidades que permanecen durante toda la vida profesional y que pueden implementarse prácticamente en cualquier lenguaje.

Ese último punto es el que retoma `tu05otros.tex` al comparar lenguajes, y el que sostiene el criterio de toda la unidad Saravia Alía 2005.

UNIDAD 1: Hardware, software y estándares

La computadora. Otros dispositivos. Reseña histórica. Computadoras analógicas. Arquitectura de Von Neumann. Redes neuronales e IA. Partes y dispositivos de una computadora personal. Datos y direcciones, almacenamiento permanente.

Sistemas Operativos. Terminal y comandos. Memoria. Archivos. Directorios. Código Fuente y ejecutable. Conceptos de Software libre. Desarrollo colaborativo. Ciencia abierta. Transparencia vs. Oscuridad. Representación de la información: bit, ASCII, Unicode, UTF-8/16.

UNIDAD 2: Redes

Redes. protocolos TCP/IP . Puertos, paquetes, y capas. Números IP y redes locales, ruteo. Nombres DNS. Comandos: ssh, scp y sus llaves, wget, tcpdump, nmap; Uso y protocolos de internet. Navegadores. Http y https. Correo electrónico. Herramientas colaborativas en red. Redes sociales. El código es ley. Redes Sociales. Libertad vs, seguridad.

UNIDAD 3: Edición de textos y estándares

Editores de texto plano. Procesadores de textos WYGIWYW basados en comandos. Edición de matemáticas. Editor WYSIWYG. Fortalezas y debilidades de ambos modelos. Planilla de Cálculo: fórmulas y funciones, gráficos, importación de datos, ajuste de curvas. Documentos reproducibles. Creación, edición y uso de archivos xml, tex (LaTeX), pdf y html, con sus comandos y tags. Usos de Markdown, conversión de formatos. Textos científicos. Reproducibilidad. Control de versiones. Repositorios de trabajo colaborativo: update, commit, version. Interaccion humano – IA: Chat y agentes. Local y en la nube. Discernimiento, criterios, alucinaciones, ética.

Uso de IA para programar: pedir, verificar, depurar. Cierre con IA: usarla, no delegarle el criterio; alucinaciones, verificación, ética.

UNIDAD 4: Sistemas, simulación y predicción, cálculo numérico

Simulación y modelado de sistemas físicos. Leyes de la ciencia. Variables en la física. Uso de un software de cálculo científico. Constantes y variables en el software: dirección, tamaño, tipo, nombre, alcancem tiempo de vida: locales, globales, parámetros. Dinámicas y estáticas. Escalares, vectores y matrices, cadenas de caracteres, operaciones. Reales y complejos. Operaciones relacionales y lógica. Cálculo numérico: raíces de polinomios, álgebra lineal y sistemas de ecuaciones, derivación e integración numérica, ecuaciones diferenciales. Simulación de procesos físicos resolviendo ecuaciones diferenciales (EDO) en forma numérica. Funciones matemáticas, y estadísticas predefinidas.

Uso de Scilab, Octave, Python, Perl, Javascript, Julia como terminales REPL. Comprender las diferencias con la terminal bash. Se acentuará sobre uno, el estudiante podra optar

por el de su preferencia, incluso otros, si la cáterda autoriza. Recorrido rápido por algunos de Prolog, Datalog, Clingo, Z3, cvc5, Lean, Coq, Isabelle, HOL, Mathematica, Maple, SageMath, SymPy, Maude, MiniZinc, CLIPS, Jess, Drools, Elisa en Emacs.

UNIDAD 5: Programación

Comandos. Secuencia en un archivo. Interpretación de comandos y programación. Estructuras de control. Funciones definidas por el usuario.

Algoritmos y programas. Intérpretes y compiladores. Variables: parámetros, punteros, dinámica, alcance, vida y duración, recolección de memoria. Tipado.

Estructuras de control, funciones, clausuras. Estructuras de datos. Hashes. Objetos. Módulos. Eventos.

Paradigmas de programación: funcionales, estructurados, objetos, lógico, eventos, literatdo, por tests. Validación. Buenas prácticas o ingeniería del software científico. Modularidad, clases y reuso. Procesamiento de texto. Uso de IA para programar.

Uso de los lenguajes ya vistos en el práctico anterior junto a sus librerías de funciones científicas.Parsers. Análisis sintáctico y léxico. Introducción a c, fortran, rust. BLAS, LAPACK, GSL. Make. TEXcomo lenguaje de macros.

UNIDAD 6: Entrada, salida y archivos. Estándares

Entrada y salida de datos. Entrada interactiva por teclado y salida por pantalla. Conversión y validación de datos de entrada. Formato de lectura y de presentación de resultados.

Archivos y persistencia de datos. Apertura y cierre.Manipulación. Modos de acceso. Lectura, escritura, agregado y actualización. Procesamiento línea por línea y entrada y salida de datos formateados. Comandos bash para intervenir archivos.

Archivos de texto y binarios. Lectura y escritura de tablas y matrices. Archivos delimitados y formatos tabulares. Formatos nativos de almacenamiento. Recuperación de datos y variables. Intercambio de información entre programas.

Representación y estándares de archivos. Formato, codificación y estructura de los datos. Identificación de tipos de archivo por su contenido. Portabilidad e interoperabilidad.

Entrada, salida y error estándar. Redirección, composición de programas y tuberías.

El sistema de archivos como interfaz del sistema operativo. Archivos especiales, dispositivos e información del sistema. Principio “todo es un archivo”.

Operaciones de E/S (entrada/salida) y manipulación de archivos. Lectura y escritura de datos.Todo es un archivo.

UNIDAD 7: Gráficos, visualización e imágenes

Representación gráfica de funciones y datos. Gráficos bidimensionales. Discretización y muestreo. Curvas múltiples con dominios y resoluciones diferentes. Estilos de líneas y marcadores. Títulos, ejes, leyendas, cuadrículas, anotaciones, límites y relación de aspecto. Tratamiento gráfico de discontinuidades, singularidades y valores no representables.

Escalas y sistemas de coordenadas. Escalas lineales, semilogarítmicas y logarítmicas. Transformaciones de coordenadas y curvas polares. Representación de datos experimentales. Barras de error. Histogramas y distribuciones. Ajuste de datos, regresión lineal y análisis de residuos.

Herramientas de generación de gráficos. Uso de Gnuplot como programa gráfico externo: instalación y verificación, ejecución interactiva y no interactiva, scripts, archivos de datos, generación automática de órdenes e invocación desde programas. Uso de las capacidades gráficas nativas de Scilab y de los demás lenguajes del curso. Creación y manipulación de objetos gráficos. Propiedades y parámetros de los gráficos. Composición de figuras y paneles.

Representación tridimensional. Grillas de datos. Superficies, mallas, mapas de color y perspectivas. Curvas de nivel, mapas con relleno, contornos sobre superficies y proyecciones sobre planos. Resolución de grillas. Campos vectoriales y gradientes.

Visualización de resultados de modelos numéricos. Series temporales, trayectorias y retratos de fase. Representaciones múltiples de un mismo fenómeno. Generación de secuencias de imágenes y animaciones.

Exportación, persistencia e interoperabilidad gráfica. Formatos raster y vectoriales. PNG, PDF, SVG y formatos equivalentes. Reproducibilidad de los gráficos a partir de datos y scripts. Separación entre cálculo, datos, especificación gráfica y archivo de salida.

Resolución de los problemas mediante dos enfoques complementarios: uso de Gnuplot como motor gráfico externo y uso de las capacidades gráficas propias de Scilab, GNU Octave, Python, Perl y JavaScript/Node.js.

UNIDAD 8 Interfaces gráficas de usuarios (GUI) y eventos

a) Terminal y Web. Terminal como interfaz. Entrada, salida y error estándar. Paso de parámetros y argumentos.

Web. Modelo cliente-servidor. HTTP. State-less. Formularios web. Métodos GET y POST. Paso de parámetros entre cliente y servidor. Estado. Uso de diálogos predefinidos. CGI y servidores de aplicaciones. Integración de una interfaz web con programas de cálculo.

b) Nativa de diferentes programas. Ventanas, controles y widgets. Creación y manipulación de objetos gráficos. Callbacks y procesamiento de eventos. Paso de parámetros y mantenimiento del estado. Uso de diálogos predefinidos. Eventos para modificar parámetros de una simulación. Separación entre modelo, interfaz y representación gráfica.

c) Jupyter como idea. Documentos interactivos: código, texto y resultados. Widgets y callbacks. Eventos para simular fenómenos físicos.

UNIDAD 9: medición y control

El proceso de medida. Magnitud física como una señal eléctrica: sensores. Señales eléctricas analógicas y digitales. Medida de una señal analógica: adquisición, digitalización, trasmisión, visualización y registro de señales digitales. Conversión A/D, Errores en general y errores de conversión. Sistemas multicanales.

Los temas anterires pueden tener superposición con otra materia sobre Medición y electrónica. Según sea el contenido real deberá cpprdinarse.

Adquisición de datos experimentales, y control. Programación y actuación. Conceptos de uso de Arduino, ESP32, Raspberry Pi.

UNIDAD 10: No es unidad, es un contenedor para hacer el proyecto

Como cierre de la asignatura se debe completar un proyectoque integre los contenidos

desarrollados durante el curso.

No tiene unidad temática, pues es un práctico integrador.

Capítulo 1

Hardware, software y estándares

1.1. La Computación

1.1.1. Función de la computadora

La función principal de una computadora es la de procesar información.

Tanto el ingreso como la presentación de los resultados correspondientes y la interacción humano–máquina, o máquina–máquina, requieren de diversos dispositivos.

Así existen:

- los dispositivos de entrada, como el escáner, el mouse, el teclado;
- los dispositivos de salida, tales como el monitor y la impresora;
- y los dispositivos de entrada – salida, como la memoria permanente (antes llamados discos duros, hoy de varios tipos como SSD).

1.1.2. Reseña histórica

La historia de la computación puede dividirse en generaciones determinadas por la tecnología empleada en la construcción de los equipos.

Antes de la aparición de los circuitos electrónicos, existieron máquinas mecánicas capaces de realizar cálculos. En el siglo XVII, Blaise Pascal construyó una calculadora mecánica capaz de sumar y restar. Gottfried Wilhelm Leibniz extendió ese diseño para multiplicar y dividir. A mediados del siglo XIX, Charles Babbage concibió la *Máquina Analítica*, un dispositivo de propósito general que anticipó los principios de las computadoras modernas; Ada Lovelace, colaboradora de Babbage, es considerada la primera programadora de la historia.

La primera generación (1945–1955): estuvo dominada por las *válvulas de vacío*. La ENIAC (1945), desarrollada en la Universidad de Pensilvania, fue una de las primeras computadoras electrónicas de propósito general: ocupaba una sala entera,

consumía enormes cantidades de energía y se programaba mediante conexiones físicas de cables.

La segunda generación (1955–1965): se caracterizó por la adopción del *transistor*, inventado en los Laboratorios Bell en 1947. Los transistores reemplazaron a las válvulas de vacío: eran más pequeños, consumían menos energía y eran más confiables. En esta época surgieron los primeros lenguajes de programación de alto nivel como FORTRAN y COBOL.

La tercera generación (aprox. 1964–1971): introdujo los *circuitos integrados*, que permitieron colocar decenas y luego miles de transistores en una sola pastilla de silicio. Los equipos se hicieron más pequeños, rápidos y económicos. En 1964 IBM presentó la familia System/360, un hito por mantener una arquitectura compatible entre modelos de distintas prestaciones. System/360 utilizó la tecnología híbrida SLT de IBM, no circuitos integrados monolíticos como los que terminarían imponiéndose posteriormente.

La cuarta generación (desde 1971): está definida por el *microprocesador*: todos los circuitos de la CPU integrados en un único chip. Intel lanzó el 4004 en 1971; en 1981 IBM presentó la IBM PC, cuya arquitectura tuvo una enorme influencia en la estandarización posterior de la computadora personal. Durante la década de 1990 la expansión de Internet transformó radicalmente el uso de las computadoras. A partir de los años 2010 la proliferación de dispositivos móviles, la computación en la nube y la inteligencia artificial marcaron nuevas etapas de la computación.

A partir de la cuarta generación deja de existir una periodización universalmente aceptada basada en un único componente físico dominante. A efectos pedagógicos, pueden distinguirse etapas posteriores según los cambios en el uso y la organización de los sistemas:

Etapa de computación en red (aprox. 1990–2010): Internet, la Web, el correo electrónico, los servidores, las bases de datos distribuidas y luego los dispositivos móviles hicieron que la computadora dejara de ser principalmente una máquina aislada y pasara a funcionar como nodo de una infraestructura mundial de comunicación, información y servicios.

Etapa de computación ubicua y acelerada (aprox. 2010–): se caracteriza por la nube, los grandes datos, las GPU, los aceleradores especializados y la inteligencia artificial. La potencia de cálculo se distribuye entre computadoras personales, centros de datos, teléfonos inteligentes, sensores y servicios en red. Históricamente, la expresión "quinta generación" también estuvo asociada al proyecto japonés de los años ochenta orientado a inteligencia artificial y lenguajes lógicos.

1.1.3. Computadoras analógicas

Las computadoras analógicas operan con señales continuas en lugar de señales discretas. Mientras que una computadora digital representa la información mediante dígitos binarios (0 y 1), una computadora analógica la representa mediante magnitudes físicas continuas: voltajes, corrientes, ángulos de rotación o presiones.

En una computadora analógica electrónica, las variables matemáticas del problema se corresponden con voltajes en el circuito. Las operaciones matemáticas se realizan mediante componentes electrónicos analógicos: los amplificadores operacionales pueden sumar, restar e integrar señales; los multiplicadores analógicos realizan productos; los potenciómetros ajustan coeficientes. Conectando estos elementos de manera apropiada es posible construir un circuito cuyo comportamiento eléctrico es *análogo* al sistema físico que se desea estudiar.

Su principal ventaja histórica fue la velocidad de respuesta: el circuito evolucionaba continuamente y podía representar en tiempo real la dinámica de ciertos sistemas descritos por ecuaciones diferenciales. Por eso fueron muy utilizadas hasta la década de 1970 para simulación de sistemas de control, trayectorias de proyectiles y dinámica de aeronaves.

Sus desventajas frente a las computadoras digitales son la precisión limitada (los voltajes están afectados por ruido y tolerancias de los componentes), la dificultad de programación (requería recablear el circuito para cada problema) y la dificultad de reproducir y almacenar resultados con una precisión comparable a la de los sistemas digitales.

Hoy las computadoras analógicas de propósito general prácticamente no se fabrican. Sin embargo, los principios analógicos persisten en: circuitos de procesamiento de señales, conversores analógico-digitales (ADC) y digital-analógicos (DAC), sistemas de control embebidos, y en investigación sobre computación neuromórfica, que busca reproducir en hardware principios de organización inspirados en sistemas nerviosos. Las implementaciones neuromórficas pueden ser analógicas, digitales o de señal mixta.

1.1.4. Computadoras digitales: Arquitectura de Von Neumann

Desde la aparición del microprocesador en la década de 1970, la arquitectura básica de las computadoras personales ha permanecido prácticamente inalterada. Aunque los componentes evolucionaron considerablemente, todas las PC modernas continúan organizándose alrededor de una placa madre, un procesador, memoria principal y dispositivos de almacenamiento y entrada/salida.

La arquitectura asociada al modelo de programa almacenado formulado en 1945 constituye una referencia fundamental para comprender la mayoría de las computadoras de propósito general. Las implementaciones modernas, sin embargo, incorporan variantes de tipo Harvard, cachés separadas, múltiples niveles de memoria y aceleradores especializados. Su principio fundamental es el *programa almacenado*: las instrucciones del programa y los datos que procesa se guardan juntos en la misma memoria, lo que permite modificar el programa como si fuera un dato más.

El modelo se compone de cuatro unidades funcionales:

Unidad Central de Procesamiento (CPU): ejecuta las instrucciones. Contiene la Unidad de Control, que dirige el flujo de operaciones, y la Unidad Aritmético-Lógica (ALU), que realiza los cálculos.

Memoria principal: almacena tanto las instrucciones del programa como los datos que se están procesando. Es de acceso aleatorio (RAM) y su contenido se pierde al apagar la computadora.

Dispositivos de entrada y salida: permiten la comunicación entre la computadora y el mundo exterior. El teclado y el mouse son ejemplos de entrada; el monitor y la impresora, de salida.

Bus del sistema: canal de comunicación que conecta las unidades anteriores. Se divide en bus de datos (transporta la información), bus de direcciones (indica la posición en memoria) y bus de control (coordina las operaciones).

El ciclo básico de funcionamiento se denomina *ciclo fetch-decode-execute* (buscar, decodificar, ejecutar): la CPU busca la próxima instrucción en memoria, la decodifica para saber qué operación realizar y finalmente la ejecuta, repitiendo el ciclo continuamente mientras el equipo está en funcionamiento.

Una limitación conocida de esta arquitectura es el llamado *cuello de botella de Von Neumann*: como instrucciones y datos comparten el mismo bus, el procesador debe esperar a que lleguen por el mismo canal, lo que limita el rendimiento. Las arquitecturas modernas mitigan este problema mediante memorias caché y técnicas de ejecución especulativa y fuera de orden.

En el modelo de Von Neumann se distinguen varias partes principales para una computadora:

CPU: Unidad Central de Proceso. Ejecuta instrucciones y realiza operaciones sobre datos.

Controlador de interrupciones: Combina varias fuentes de interrupción sobre una o más líneas de la CPU, y permite asignar prioridades a las interrupciones.

Memoria: Retiene, memoriza o almacena datos informáticos durante algún intervalo de tiempo. Puede pensarse como una gran tabla de palabras o bytes, cada una identificada mediante una dirección única. Tipos básicos: RAM y ROM.

Entrada–salida: Comunicación con periféricos y dispositivos. Muchas operaciones de entrada–salida se realizan mediante direcciones de memoria o registros especiales; por el bus de direcciones el procesador indica a qué posición o dispositivo se refiere.

Buses

Los buses son conjuntos de cables o líneas de comunicación que transportan información entre los componentes de la computadora.

Bus de direcciones: Transporta la dirección del dato, posición de memoria o periférico al que se quiere acceder.

Bus de control: Indica la operación a realizar, por ejemplo lectura, escritura, modificación o atención de interrupciones.

Bus de datos: Transporta los datos propiamente dichos. Su ancho puede ser, por ejemplo, de 32 o 64 bits.

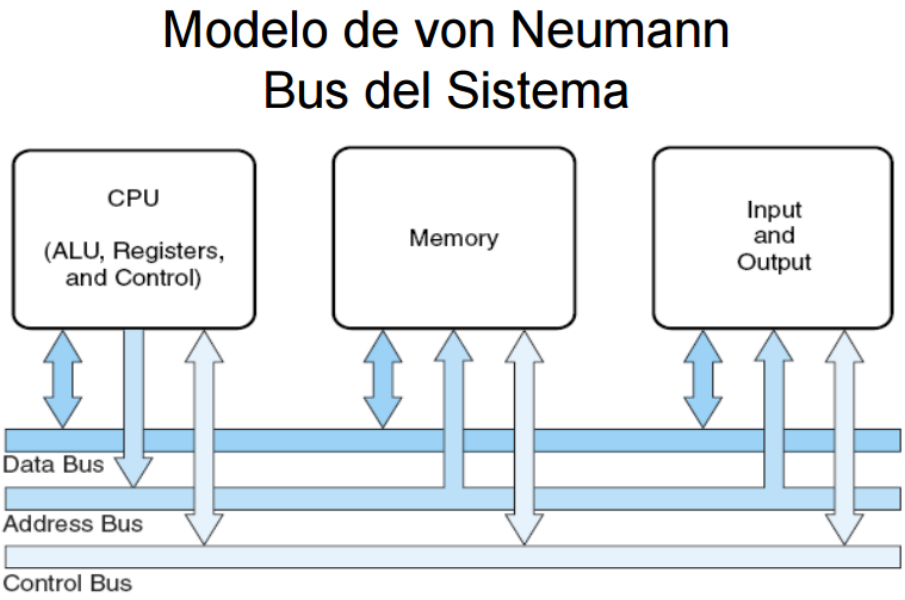


Figura 1.1: Buses Von Neumann

Procesos e interrupciones

Al encenderse o reiniciarse, el procesador comienza la ejecución desde un estado y una dirección de arranque definidos por su arquitectura. Allí se encuentra código de firmware que inicializa el hardware y posteriormente carga un gestor de arranque y el sistema operativo. Sólo una vez iniciado el sistema operativo tiene sentido hablar de procesos. Durante la ejecución, el procesador puede recibir interrupciones de hardware (IRQ), por ejemplo ante eventos de dispositivos o temporizadores, ejecutar la rutina correspondiente y luego continuar la tarea interrumpida.

Interrupciones de hardware

Las señales de interrupción tienen como propósito proveer una manera al procesador de manejar código que está fuera del flujo normal de ejecución. Cuando llega una señal de interrupción, la CPU tiene que parar lo que está haciendo en ese momento y cambiar a una nueva actividad. Para poder regresar después al flujo interrumpido preserva el estado arquitectónico necesario; el código de atención guarda además los registros adicionales que necesite.

Dispositivos

Una computadora tiene periféricos que realizan operaciones de entrada, salida, comunicación o almacenamiento. El procesador accede a sus controladores mediante mecanismos definidos por la arquitectura: en muchos sistemas se utiliza entrada/salida mapeada en memoria y, en otros casos, espacios de E/S específicos. Ejemplos: impresoras, pantallas, teclados, unidades de almacenamiento y adaptadores de red.

Arquitecturas de hardware del modelo Von Neumann

Las arquitecturas de conjunto de instrucciones se estudian más adelante de forma específica. Conviene distinguirlas de los sistemas operativos: i386 y amd64 son arquitecturas x86, ARM es una familia de arquitecturas de procesador, mientras que Android es un sistema operativo que puede ejecutarse sobre distintas arquitecturas.

1.1.5. Redes neuronales e Inteligencia Artificial

La inteligencia artificial (IA) es la rama de la informática que estudia el desarrollo de sistemas capaces de realizar tareas que, cuando las realiza un ser humano, se consideran propias de la inteligencia: reconocer imágenes, comprender texto, traducir idiomas o tomar decisiones.

Una **red neuronal artificial** es un modelo computacional inspirado en la estructura del cerebro. Se organiza en capas de nodos o *neuronas*: una capa de entrada, una o más capas ocultas y una capa de salida. Cada conexión entre neuronas tiene asociado un *peso* numérico que se ajusta durante el entrenamiento. El proceso de aprendizaje consiste en presentar al modelo grandes cantidades de ejemplos y ajustar los pesos para minimizar el error entre la salida producida y la salida esperada, mediante un algoritmo llamado *retropropagación del error*.

Las **redes neuronales profundas** (*deep learning*) tienen muchas capas ocultas y son capaces de aprender representaciones complejas de los datos. Su auge en la última década se debe a tres factores combinados: la disponibilidad de grandes volúmenes de datos, el aumento de la capacidad de cómputo, especialmente mediante unidades de procesamiento gráfico (GPU), y el desarrollo de nuevas arquitecturas como las redes convolucionales (para imágenes) y los transformadores (para texto).

Los **modelos de lenguaje** actuales, como los que alimentan los asistentes de IA generativa, son redes de transformadores entrenadas con cantidades masivas de texto. Son capaces de generar texto coherente, traducir, resumir, responder preguntas y escribir código, aunque no "comprenden" el lenguaje en el sentido humano del término.

El hardware requerido por la IA moderna es exigente. Las GPU, originalmente diseñadas para gráficos, resultaron ideales para las operaciones matriciales masivas que requieren las redes neuronales. Además de las GPU, distintos fabricantes desarrollaron aceleradores especializados. Google diseñó las TPU (*Tensor Processing Units*), mientras que numerosos SoC incorporan NPU (*Neural Processing Units*); NVIDIA integra unidades especializadas, como los Tensor Cores, dentro de sus GPU.

1.2. Definiciones: hardware, software, máquinas de IA

Se designa como:

Hardware: al conjunto de componentes físicos que forman el sistema de computación, sean reemplazables o estén integrados o soldados: procesadores, memorias, placas,

circuitos, sensores, dispositivos de almacenamiento, teclado, monitor y otros periféricos.

El hardware se apoya en dos disciplinas fundamentales:

- Tecnología electrónica, responsable del diseño y construcción de los componentes físicos.

  La electrónica propone el estudio de los componentes: integrados, placas, transistores, resistencias, elementos de memoria, etc.

- Lógica digital, encargada del diseño de circuitos y de la organización de los sistemas de procesamiento.

  En una perspectiva "lógica" vinculada al hardware se aborda el estudio del ensamblaje de aquellos componentes para constituir circuitos tales como registros, sumadores, circuitos de selección en memoria o unidades como la unidad de control, la unidad aritmética-lógica, etc.

En este apartado interesa distinguir los componentes físicos de la computadora, el software que los controla y los programas de uso común.

Software: es la información que dirige el funcionamiento de la computadora: instrucciones para ejecutar, o aquello que se traduce en esas instrucciones que interpreta el procesador y que se almacena en su memoria, sea momentánea o permanente.

Estudiaremos:

- Lenguajes y técnicas de programación.

  La computación, el desarrollo e ingeniería del software, el análisis de sistemas, la programación, etc., cubre los problemas ligados a los lenguajes, paradigmas de programación, sus buenas prácticas, etc.

- Programas destinados a administrar el funcionamiento del sistema informático.

  Otro estudio que haremos será el del software de los programas encargados de la gestión del funcionamiento de un sistema informático.

  Desde el sistema operativo hasta las aplicaciones comunes utilizadas por casi todos los usuarios, o las especializadas que usan algunos, según su función.

Modelos de inteligencia artificial: En inteligencia artificial aparece una categoría que conviene distinguir tanto del hardware como del software: el **modelo computacional**. Una red neuronal entrenada no es software. Es una estructura matemática cuyos parámetros han sido determinados mediante un proceso de entrenamiento.

En una red neuronal pueden distinguirse:

- la **arquitectura**, que establece cómo se relacionan matemáticamente las distintas unidades o capas;
- los **parámetros o pesos**, que son valores numéricos obtenidos durante el entrenamiento;
- el **software**, que ejecuta las operaciones necesarias para evaluar o entrenar el modelo;

- el **hardware**, como CPU, GPU o NPU, que realiza físicamente esas operaciones.

Un modelo de lenguaje, por ejemplo, puede representarse conceptualmente como

$$\boxed{\text{modelo} = \text{arquitectura matemática} + \text{parámetros}}$$

Los parámetros de un modelo son datos. Pueden ser miles de millones de números almacenados en archivos. El software de inferencia lee esos parámetros y ejecuta las operaciones matemáticas definidas por la arquitectura.

Por lo tanto,

$$\boxed{\text{modelo de IA} \neq \text{software que ejecuta el modelo}}$$

Esta diferencia resulta especialmente clara durante el entrenamiento:

$$\text{software de entrenamiento} + \text{datos de entrenamiento} \longrightarrow \text{modelo entrenado.}$$

El resultado del entrenamiento no es un nuevo programa, sino un nuevo conjunto de parámetros que determina el comportamiento de una estructura matemática. De este modo, en un sistema moderno de inteligencia artificial deben distinguirse al menos cuatro elementos: **hardware**, **software**, **datos** y **modelos**.

1.2.1. Hardware

La Motherboard

La motherboard o placa madre constituye la placa principal del sistema. En ella se ubican y conectan los componentes que hacen al funcionamiento de la computadora: el microprocesador, la memoria, los buses de datos, control y direcciones, el chipset, el firmware BIOS o UEFI y los distintos conectores internos y externos. Desde allí se organiza la comunicación entre la CPU, la memoria, las placas de expansión, los dispositivos de almacenamiento y los periféricos.

En una motherboard actual conviene reconocer, al menos, los siguientes elementos:

- Socket del procesador
- VRM
- Memoria RAM (DIMM)
- Ranuras PCI Express
- Conector ATX de 24 pines
- Conector EPS de CPU

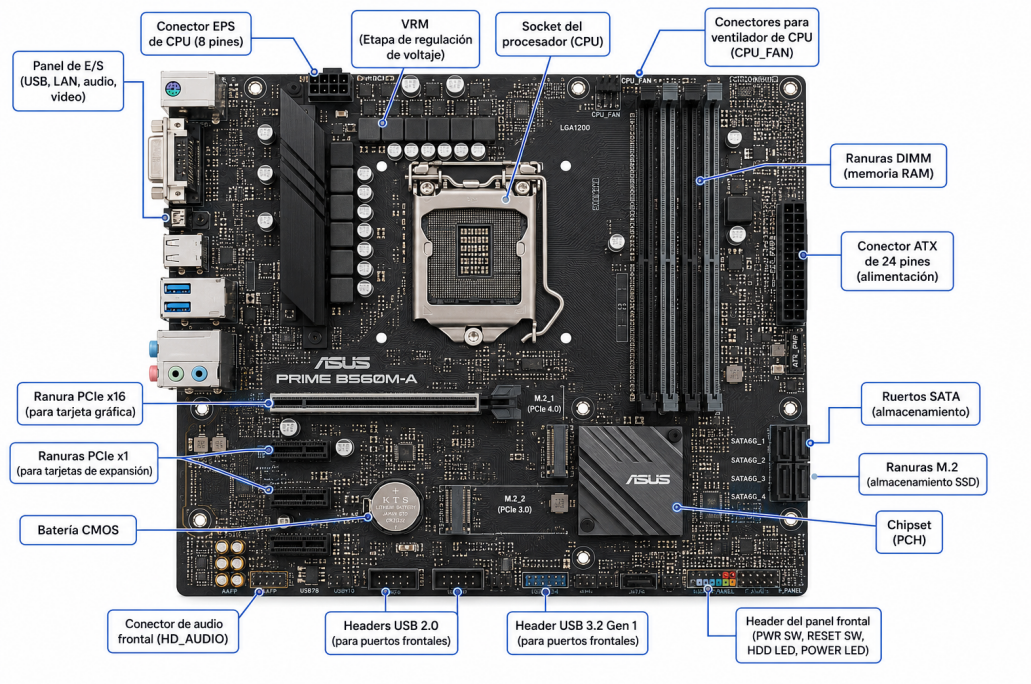


Figura 1.2: ASUS PRIME B560M-A (micro-ATX para procesadores Intel de décima y undécima generación).

- Puertos SATA
- Ranuras M.2
- Headers USB
- Header del panel frontal
- Batería CMOS
- Puertos de E/S (USB, LAN, audio, video)
- Chipset
- Conectores para ventiladores
- Conectores RGB/ARGB
- Antenas o módulo WiFi/Bluetooth, si la placa lo incluye

El Microprocesador o CPU

El microprocesador o CPU (*Central Processing Unit*) recibe las instrucciones almacenadas en memoria, las interpreta y ejecuta las operaciones correspondientes mediante la unidad de control y la unidad aritmético-lógica (ALU).

Algunos llaman CPU a la caja.

El funcionamiento del microprocesador está asociado a un bloque llamado unidad de control que recibe la información proveniente de la memoria a través del bus de datos,

decodifica cada instrucción y la ejecuta, enviando datos a cualquiera de los dispositivos que la instrucción indique o efectuando distintos procesos con los datos. Existe otro bloque, también importante llamado unidad aritmética y lógica (ALU) que es la que realiza las operaciones lógicas y aritméticas con los datos recibidos. Una característica importante es que el ancho del bus de direcciones de un microprocesador determina la cantidad de memoria que puede ser direccionada. Los procesadores disipan energía en forma de calor y necesitan un sistema de refrigeración adecuado a su potencia. En equipos de alto consumo suele utilizarse disipador y ventilador, mientras que procesadores de bajo consumo pueden funcionar con refrigeración pasiva.

Datos, direcciones y control: buses

El bus vincula el procesador con los dispositivos de entrada y salida, y a su vez a éstos con la memoria. El bus de datos es la vía por donde el procesador transporta los datos desde y hacia los dispositivos de entrada y salida. Como estos datos deben ser colocados en algún lugar asignado, existe el bus de direcciones. Otro bus es el de control que administra la forma en cómo se realizan los procesos, es decir, el procesador lo utiliza para enviar comandos.

En el modelo clásico se distinguen conceptualmente bus de datos, bus de direcciones y bus de control. Las computadoras modernas pueden implementar estas funciones mediante varios enlaces e interconexiones diferentes, no necesariamente mediante un único bus físico compartido.

El **bus de datos** transporta información entre los componentes. Su ancho indica cuántos bits forman una transferencia elemental del modelo; por ejemplo, 64 bits corresponden a ocho bytes por transferencia. La cantidad real transferida por ciclo depende del protocolo, la frecuencia, el número de canales y otras características de la interconexión.

El **bus de direcciones** identifica posiciones de memoria o recursos. Con n bits de dirección pueden representarse 2^n direcciones distintas. Si cada dirección identifica un byte, un espacio de 32 bits abarca 2^{32} bytes, es decir 4 GiB. Los procesadores de 64 bits no necesariamente implementan físicamente los 64 bits posibles de dirección.

El **bus de control** coordina las operaciones: señales como lectura/escritura, petición de interrupción o confirmación de transferencia viajan por este bus.

Internamente, la CPU dispone de *registros*: pequeñas memorias de altísima velocidad integradas en el propio chip. El registro más importante es el *contador de programa* (PC, *Program Counter*), que guarda la dirección de la próxima instrucción a ejecutar. El *registro de instrucción* (IR) contiene la instrucción en curso. Los *registros de propósito general* almacenan operandos y resultados intermedios.

La frecuencia de reloj del procesador se mide en hercios (Hz) e indica cuántos ciclos de reloj ocurren por segundo. No determina por sí sola el rendimiento: también importan las instrucciones ejecutadas por ciclo (IPC), la cantidad y tipo de núcleos, la jerarquía de memoria, el paralelismo y la naturaleza de la carga.

Controlador de interrupciones

Cuando un dispositivo de entrada o salida necesita la atención del procesador, no puede simplemente esperar a que éste lo consulte: eso sería ineficiente. En su lugar, envía una señal especial llamada *interrupción*.

El **controlador de interrupciones** (en sistemas x86 modernos se utiliza la familia APIC, *Advanced Programmable Interrupt Controller*) recibe esas señales de los distintos dispositivos, las prioriza y las presenta al procesador en el momento adecuado. Cuando el procesador acepta una interrupción, suspende temporalmente lo que estaba haciendo, guarda su estado, ejecuta una rutina de atención específica para ese dispositivo y luego retoma la tarea interrumpida exactamente donde la había dejado.

Cada tipo de interrupción tiene asignado un número llamado *vector de interrupción*, que apunta a la rutina de atención correspondiente en una tabla mantenida por el sistema operativo. Así, una pulsación de teclado, la llegada de un paquete de red o el fin de una operación de disco generan interrupciones distintas que se atienden con código diferente.

Conviene distinguir:

Interrupciones de hardware: eventos asincrónicos generados por dispositivos o controladores, como temporizadores, adaptadores de red o almacenamiento.

Excepciones: eventos síncronos generados por la ejecución de una instrucción, como división por cero, fallo de página o violación de protección.

Llamadas al sistema: peticiones explícitas de un programa al núcleo mediante la interfaz definida por la arquitectura y el sistema operativo. Históricamente algunas arquitecturas utilizaron interrupciones o *traps*; procesadores modernos suelen disponer de instrucciones específicas para realizarlas.

El Chipset.

En una PC moderna, muchas funciones que históricamente pertenecían al chipset —como el controlador de memoria— están integradas dentro de la CPU. El chipset o PCH concentra principalmente funciones de entrada/salida y expansión, como puertos USB, SATA y carriles PCIe adicionales. Junto con la CPU y la motherboard determina parte de las posibilidades de ampliación y conectividad del sistema.

Ranuras de expansión.

Son conectores de la motherboard que permiten agregar placas internas, como placas gráficas, placas de red, placas de sonido o controladoras adicionales. En las computadoras actuales, la interfaz de expansión más común es PCI Express.

La Memoria.

Memoria: jerarquía y gestión

La memoria de una computadora no es un recurso homogéneo sino una jerarquía de tecnologías con distintas velocidades, capacidades y costos. A medida que se sube en la jerarquía, la memoria es más rápida pero más pequeña y costosa.

Registros: memorias internas de la CPU accesibles directamente por las instrucciones. Su número, tamaño y función dependen de la ISA; pueden incluir registros escalares, vectoriales, de control y otros. Su acceso es normalmente de muy baja latencia.

Caché L1, L2, L3: integrada en el chip del procesador, de kilobytes a megabytes, tiempos de acceso de 1 a 30 nanosegundos.

RAM: memoria principal del sistema, de gigabytes, tiempo de acceso de decenas de nanosegundos.

Almacenamiento permanente: SSD y HDD, de cientos de gigabytes a terabytes, tiempos de acceso de microsegundos a milisegundos.

El sistema operativo implementa **memoria virtual**: cada proceso ve un espacio de direcciones propio y continuo, independientemente de cómo esté distribuida la memoria física. El SO mantiene una *tabla de páginas* que traduce las direcciones virtuales del proceso a direcciones físicas reales. Cuando un proceso realiza un acceso que no puede resolverse con la traducción actual ocurre un *fallo de página* (*page fault*). El núcleo analiza la causa: puede asignar una página nueva, realizar una copia por escritura (*copy-on-write*), cargar datos de un archivo o del área de intercambio, o rechazar el acceso si viola las protecciones.

La memoria virtual proporciona a cada proceso un espacio de direcciones propio, aislamiento y protección, y permite mapear y compartir memoria de manera controlada. Además puede utilizar almacenamiento secundario como respaldo de ciertas páginas, pero el uso de disco o *swap* es sólo uno de los mecanismos asociados.

Principal

(DIMM, Dual Inline Memory Module – Módulo de memoria dual en línea)

La memoria principal es uno de los componentes centrales del sistema. Permite almacenar temporalmente los datos e instrucciones que la CPU necesita mientras ejecuta programas. En una PC actual se implementa como RAM, habitualmente en módulos DIMM. Es una memoria de lectura y escritura, pero volátil: al apagar la computadora, su contenido se pierde.

Cuando se necesita conservar información de forma permanente se utilizan dispositivos de almacenamiento no volátil. Históricamente se usaron disquetes, CD-ROM y discos rígidos; hoy son comunes los HDD, las unidades de estado sólido SSD —que pueden utilizar SATA o NVMe— y las memorias USB. La cantidad de RAM necesaria depende de los programas y datos con los que se trabaje. La ampliación se realiza agregando o reemplazando módulos compatibles con los zócalos disponibles de la motherboard.

ROM, Firmware, BIOS y UEFI.

Históricamente se utilizaron memorias ROM para almacenar firmware que debía conservarse sin alimentación. En equipos actuales el firmware suele residir en memoria flash no volátil, que puede actualizarse.

BIOS y UEFI son firmware de plataforma: inicializan el hardware y participan en el arranque del sistema. Una vez cargado el sistema operativo, éste accede a los dispositivos principalmente mediante sus propios controladores. La configuración del firmware moderno suele almacenarse en memoria no volátil; la batería de la placa mantiene además el reloj en tiempo real y, según la plataforma, ciertos parámetros.

Así, entre el hardware físico y el sistema operativo existe una capa de software muy básica llamada *firmware*. El firmware está grabado en memorias no volátiles de la placa madre o de los dispositivos, y permite inicializar, configurar y controlar componentes aun antes de que cargue el sistema operativo.

En las computadoras personales clásicas, esta función fue cumplida por el *BIOS* (*Basic Input/Output System*). Al encender la computadora, el BIOS realiza pruebas iniciales del hardware, detecta dispositivos básicos, configura parámetros elementales y busca desde dónde arrancar el sistema operativo, por ejemplo un disco rígido, un pendrive o una unidad óptica.

En los equipos modernos, el BIOS fue reemplazado en gran medida por *UEFI* (*Unified Extensible Firmware Interface*). UEFI cumple una función semejante, pero es más flexible: puede trabajar con discos grandes, maneja tablas de particiones GPT, ofrece una interfaz más moderna, permite cargar controladores y programas previos al sistema operativo, y puede incluir mecanismos de seguridad como *Secure Boot*.

Así, al encender una computadora, la secuencia general es:

1. el hardware recibe energía;
2. el firmware se ejecuta primero;
3. BIOS o UEFI inicializa y prueba componentes básicos;
4. se selecciona un dispositivo de arranque;
5. se carga el gestor de arranque;
6. el gestor de arranque carga el sistema operativo;
7. el sistema operativo toma control del hardware y lanza los servicios y programas iniciales.

En resumen, el firmware es el primer software que corre en la máquina. BIOS y UEFI son formas históricas de esa capa inicial: BIOS corresponde al modelo tradicional de PC; UEFI es su reemplazo moderno.

La memoria caché.

La memoria caché sigue existiendo y es fundamental en las computadoras actuales, pero ya no suele aparecer como un componente separado de la placa madre, sino integrada dentro del procesador. Su función es guardar temporalmente instrucciones y datos usados con frecuencia para que la CPU no tenga que esperar cada vez el acceso a la memoria RAM.

En los procesadores modernos la caché se organiza en niveles: caché L1, muy pequeña y muy rápida; caché L2, de mayor tamaño; y caché L3, generalmente compartida entre

varios núcleos. La cantidad de caché se mide hoy en megabytes y afecta el rendimiento, especialmente cuando los programas reutilizan muchas veces los mismos datos.

Interfaces de red, módems

Una interfaz de red permite conectar la computadora con otras computadoras o con Internet. Puede estar integrada en la motherboard, instalada como placa de expansión o conectada externamente, por ejemplo mediante USB. Las interfaces de red más comunes son Ethernet, para redes cableadas, y WiFi, para redes inalámbricas de área local.

El equipo de acceso depende de la tecnología utilizada por el proveedor. En DSL o cable se emplean módems; en fibra óptica suele utilizarse una ONT/ONU; en redes celulares se usa un módem celular. El equipo entregado por el proveedor puede integrar además router, punto de acceso WiFi, switch Ethernet y otras funciones.

Bluetooth no se usa normalmente para conectar una computadora a Internet, sino para enlaces de corto alcance con periféricos y dispositivos cercanos, como teclados, mouse, auriculares, teléfonos o parlantes.

Conexiones, puertos, potencia

Algunos son internos al gabinete de la computadora, otros se usan para conectar dispositivos exteriores.

Los del exterior pueden ir al frente o la parte trasera.

Los puertos e interfaces de comunicación sirven para transportar datos entre la computadora y otros dispositivos. Una distinción técnica importante, que sigue vigente aunque hayan cambiado los conectores, es la diferencia entre transmisión paralela y transmisión serie.

- **Transmisión paralela.** Transmite varios bits al mismo tiempo por varios conductores. Fue común en puertos antiguos de impresora, como PRN o puerto paralelo. Su ventaja era enviar varios bits por ciclo, pero resultaba más difícil mantener la sincronización y la calidad de señal a altas velocidades o largas distancias.

- **Transmisión serie.** Transmite los bits en secuencia por uno o pocos pares de conductores, usando velocidades muy altas y técnicas de codificación y sincronización más robustas. En las computadoras actuales predominan interfaces serie o serializadas: USB, SATA, PCI Express, DisplayPort, HDMI y Ethernet.

Por eso, aunque casi no se usen ya los conectores clásicos COM y PRN, la idea de comunicación serie y paralela sigue siendo importante: muchas interfaces modernas son serie porque permiten mayores velocidades, cables más simples y mejor confiabilidad.

Puertos VGA, HDMI, DisplayPort: para los monitores.

Conexiones para sonido: para micrófono (MIC IN), auriculares (LINE OUT, LINE IN), 5.1, 7.1, etc. Para subwoofers, canales frontales y traseros, estéreo, etc.

Puertos USB. USB significa *Universal Serial Bus*. Es una interfaz serie estándar que permite conectar periféricos de distinta naturaleza mediante un conector común. Además de datos, muchas versiones de USB permiten alimentar o cargar dispositivos. Han aparecido distintas generaciones del protocolo, como USB 3.x y USB4, y distintos conectores. USB-C es un tipo de conector y no determina por sí solo la velocidad, la potencia disponible ni la capacidad de transportar video. Hoy se usan para el ratón (antes serie común, PS/2 MOUSE), y para el teclado (antes PS/2 KBD), para impresoras (antes paralelo). Hoy también puede intercambiarse sonido por estos puertos.

Conexiones de discos y almacenamiento permanente.

A lo largo del tiempo se usaron distintas tecnologías de almacenamiento permanente. Entre ellas estuvieron los discos flexibles o *floppy disks*, los discos rígidos magnéticos y, más recientemente, las unidades de estado sólido o *SSD*.

También cambiaron las interfaces, protocolos y transportes utilizados para acceder a esos dispositivos. En computadoras personales fueron comunes ATA/IDE y luego SATA; en servidores, SCSI y SAS. En muchos SSD actuales se utiliza **NVMe**, un protocolo de comandos para almacenamiento no volátil que normalmente funciona sobre PCI Express. El dispositivo puede presentarse en distintos formatos físicos, entre ellos M.2 y tarjetas PCIe.

En síntesis, conviene distinguir varios niveles:

- el **tipo de dispositivo**: disco rígido, SSD, unidad óptica, memoria USB;
- el **protocolo o interfaz lógica**: ATA, SCSI, NVMe;
- el **transporte o enlace**: SATA, SAS, PCIe, USB;
- el **formato físico o conector**: por ejemplo M.2 o una tarjeta de expansión PCIe.

El sistema operativo abstrae esas diferencias y presenta los dispositivos de almacenamiento como unidades, particiones, archivos y directorios.

De energía eléctrica y de contacto.

La motherboard necesita corriente y la toma de la fuente de alimentación de la caja. Los enchufes tienen la particularidad de que presentan una muesca que los hacen asimétricos de manera de no equivocar la conexión.

Almacenamiento permanente

A diferencia de la memoria RAM, los dispositivos de almacenamiento permanente conservan la información aunque la computadora esté apagada. Constituyen el nivel más lento pero de mayor capacidad en la jerarquía de memoria.

Los **discos rígidos** (HDD, *Hard Disk Drive*) almacenan la información en platos magnéticos giratorios. Un cabezal de lectura y escritura se desplaza sobre los platos para acceder a los datos. Su capacidad puede superar los varios terabytes, pero la presencia de partes mecánicas los hace lentos y vulnerables a golpes.

Las **unidades de estado sólido** (SSD, *Solid State Drive*) no tienen partes móviles: utilizan memoria flash para almacenar la información. Son significativamente más rápidas que los HDD, más resistentes y silenciosas, aunque históricamente más costosas por unidad de capacidad.

NVMe (*Non-Volatile Memory Express*) es un protocolo diseñado para acceder eficientemente a almacenamiento no volátil. En las PC se utiliza habitualmente sobre PCIe y es común en SSD de alto rendimiento. **M.2**, en cambio, describe un formato físico y un conector: una unidad M.2 puede utilizar SATA o PCIe/NVMe, según el dispositivo y la ranura.

Algunos conceptos que conviene separar son:

SATA (*Serial ATA*): interfaz y enlace utilizado por HDD y SSD; SATA III alcanza una tasa nominal de 6 Gb/s.

M.2: formato físico y conector compacto que puede transportar, según la implementación, señales SATA, PCIe u otras.

PCIe: interconexión de alta velocidad utilizada, entre otros usos, como transporte habitual de NVMe en computadoras personales.

NVMe: protocolo de comandos para almacenamiento no volátil, normalmente transportado sobre PCIe en una PC.

Una jerarquía típica de memoria y almacenamiento local, de más rápida y costosa a más lenta y económica, puede representarse esquemáticamente como: registros → caché L1/L2/L3 → RAM → SSD (SATA o NVMe) → HDD. El almacenamiento remoto no ocupa una posición fija en esta escala: su rendimiento depende de la red y del sistema remoto.

Placas gráficas

La **GPU** (*Graphics Processing Unit*) es el procesador especializado en gráficos y cálculo altamente paralelo. Una **tarjeta gráfica** discreta es una placa que contiene una GPU, memoria, alimentación e interfaces; también existen GPU integradas dentro de una CPU o un SoC, sin tarjeta separada. Las CPU están optimizadas para baja latencia, control complejo y rendimiento general, mientras que las GPU priorizan un gran caudal de cálculo sobre muchas operaciones paralelas.

Arquitectura

Una GPU moderna contiene numerosas unidades de ejecución relativamente simples y está diseñada para explotar paralelismo de datos a gran escala. La cantidad y organización de esas unidades depende de la arquitectura y no es directamente comparable entre fabricantes.

Una **tarjeta gráfica discreta** suele conectarse a la motherboard mediante PCIe, frecuentemente con un enlace x16. Una **GPU integrada**, en cambio, forma parte de la CPU o del SoC y se comunica mediante las interconexiones internas del sistema.

Memoria de video: VRAM

Las tarjetas gráficas discretas suelen disponer de memoria propia, denominada VRAM (*Video RAM*), separada de la RAM principal. Las GPU integradas normalmente comparten memoria del sistema. La memoria accesible por la GPU almacena texturas, buffers, geometría y, en aplicaciones de IA, pesos, activaciones y otros datos de trabajo.

En tarjetas gráficas discretas de consumo se utilizan distintas generaciones de memoria **GDDR** (*Graphics Double Data Rate*), como GDDR6, GDDR6X y GDDR7, según la generación y el fabricante. Estas memorias están optimizadas para obtener un gran ancho de banda, normalmente a costa de una latencia mayor que la de las cachés internas del procesador.

Para aplicaciones de inteligencia artificial y cálculo de alto rendimiento, algunos aceleradores profesionales y de centro de datos utilizan memoria **HBM** (*High Bandwidth Memory*). La memoria HBM se apila verticalmente en capas (tecnología 3D) y se conecta a la GPU mediante un bus de ancho extraordinario, logrando anchos de banda de varios TB/s. La serie H100 de NVIDIA, diseñada para IA y cálculo de alto rendimiento, utiliza HBM3 con hasta 3,35 TB/s de ancho de banda, lo que le permite alimentar sus núcleos de cómputo sin interrupciones.

Las tarjetas gráficas para consumidores disponen actualmente de modelos con 8 a 32 GB de VRAM; las GPU profesionales para centros de datos pueden tener 80 GB o más de HBM.

Uso en Inteligencia Artificial

El entrenamiento de redes neuronales consiste fundamentalmente en multiplicaciones de matrices de grandes dimensiones, operaciones que se repiten millones de veces sobre enormes volúmenes de datos. Esta estructura matemática es exactamente la que mejor aprovecha la arquitectura paralela de la GPU.

Cuando se entrena una red neuronal, los datos se organizan en lotes (*batches*) que se procesan simultáneamente. La GPU calcula en paralelo grandes conjuntos de operaciones de cada capa, propaga el error hacia atrás y actualiza los pesos explotando el paralelismo disponible. Una operación que tardaría horas en una CPU puede completarse en minutos en una GPU.

NVIDIA desarrolló CUDA (*Compute Unified Device Architecture*), una plataforma de programación que permite usar la GPU para cálculo de propósito general. Sobre CUDA se construyeron bibliotecas como cuDNN y TensorRT, optimizadas específicamente para redes neuronales, que utilizan los *Tensor Cores*: unidades de hardware dentro de la GPU diseñadas para multiplicar matrices de forma ultrarrápida operando en precisión reducida (FP16, BF16 o INT8).

AMD ofrece una alternativa con su plataforma ROCm y sus GPU de la serie Instinct. Google diseñó sus propias TPU (*Tensor Processing Units*), chips especializados exclusivamente en IA que se utilizan en sus centros de datos y están disponibles a través de Google Cloud.

La cantidad de memoria disponible es un factor crítico en IA. Para obtener el máximo rendimiento es conveniente que los pesos y los datos activos residan en memoria accesible directamente por el acelerador, pero también es posible repartir el modelo entre varias GPU, utilizar memoria unificada o descargar parte del modelo a RAM o CPU a costa de rendimiento. Un modelo con 7.000 millones de parámetros en precisión de 16 bits ocupa

aproximadamente 14 GB sólo en pesos.

Memoria unificada en Apple Silicon

Las computadoras Apple con chips de la serie M (M1, M2, M3, M4 y M5) adoptan una arquitectura radicalmente diferente a la del esquema tradicional CPU + GPU discreta. En lugar de organizar CPU y GPU discretas con memorias independientes, Apple Silicon integra estrechamente CPU, GPU, motor neuronal, controladores y otros bloques dentro del mismo sistema en paquete, compartiendo un espacio de memoria unificada. La implementación física concreta puede utilizar uno o varios troqueles según el modelo.

Esta arquitectura se denomina **memoria unificada** (*Unified Memory Architecture*, UMA). La memoria LPDDR de alta velocidad se encuentra estrechamente integrada con el paquete del SoC y es accesible por CPU, GPU y otros aceleradores dentro del modelo de memoria unificada, evitando muchas de las copias explícitas que requiere una GPU discreta con VRAM separada.

En el esquema tradicional, cuando la CPU prepara datos para que los procese la GPU, debe copiarlos desde la RAM del sistema hacia la VRAM de la tarjeta gráfica a través del bus PCIe, lo que consume tiempo y energía. En la arquitectura UMA de Apple, CPU y GPU leen y escriben en la misma memoria física, eliminando esa copia.

Las ventajas son significativas para un dispositivo de bajo consumo: menor latencia en operaciones que involucran a ambas unidades, menor consumo energético al eliminar transferencias redundantes, y flexibilidad para asignar dinámicamente la memoria según lo requiera cada tarea. Una Mac Studio con M5 Ultra puede configurarse con hasta 512 GB de memoria unificada accesible por CPU, GPU y otros aceleradores.

La desventaja respecto a los aceleradores de centro de datos de alta gama sigue siendo, entre otros factores, el ancho de banda: la memoria unificada del M5 Ultra puede alcanzar alrededor de 1,2 TB/s, mientras aceleradores de centro de datos con HBM pueden alcanzar anchos de banda muy superiores. Las Mac con Apple Silicon son por ello especialmente relevantes para inferencia y desarrollo local de modelos que entran en su gran memoria unificada; los trabajos de entrenamiento de gran escala requieren clusters o servidores con aceleradores dedicados.

1.2.2. Arquitecturas de dispositivos de computación

Los dispositivos de computación pueden organizar sus componentes de maneras muy diferentes. Una computadora personal, un teléfono celular, un servidor o un microcontrolador poseen procesador, memoria y mecanismos de entrada y salida, pero difieren considerablemente en la forma en que estos componentes se integran y se comunican.

Esta arquitectura general del dispositivo no debe confundirse con la arquitectura del conjunto de instrucciones del procesador, que se estudiará en la sección siguiente.

Tipo de dispositivo	**Organización característica**

Computadora personal de escritorio	Sistema de propósito general tradicionalmente modular. CPU, memoria RAM, almacenamiento, GPU, interfaces y periféricos pueden encontrarse en componentes diferentes conectados mediante buses. Generalmente permite ampliar o sustituir componentes.
Notebook o laptop	Computadora personal portátil. Integra en poco espacio CPU, memoria, almacenamiento, pantalla, batería, teclado y dispositivos de comunicación. Puede conservar una organización semejante a la PC tradicional o integrar gran parte del sistema en uno o pocos chips.
Computadora basada en SoC	Gran parte de los componentes principales se integra en un *System on Chip* (SoC): CPU, GPU, controladores de memoria, procesadores especializados e interfaces. Este modelo es utilizado en computadoras Apple Silicon y también en numerosas computadoras ARM.
PC con aceleradores de IA	Computadora personal que incorpora, además de CPU y GPU, uno o más aceleradores destinados al cálculo de inteligencia artificial, habitualmente una NPU (*Neural Processing Unit*). Puede estar basada en ARM64 o x86-64. Los equipos denominados Copilot+ PC de Microsoft y sus fabricantes asociados son ejemplos actuales.
Teléfono inteligente	Sistema altamente integrado, normalmente basado en un SoC. Puede contener CPU, GPU, DSP, NPU, controladores de memoria, procesadores de imagen, seguridad, interfaces inalámbricas y módem celular dentro de uno o pocos circuitos integrados.
Tablet	Arquitectura semejante a la de un teléfono inteligente, con fuerte integración de componentes y bajo consumo de energía. Generalmente utiliza un SoC ARM64 y puede incorporar GPU, NPU y otros aceleradores.
Servidor	Sistema diseñado para ejecutar servicios de manera continua y atender numerosos procesos, usuarios o conexiones. Puede utilizar uno o varios procesadores con muchos núcleos, grandes cantidades de memoria, múltiples unidades de almacenamiento e interfaces de red de alta velocidad.
Estación de trabajo	Computadora de altas prestaciones destinada a ingeniería, ciencia, simulación, diseño, producción audiovisual u otras tareas intensivas. Suele disponer de procesadores potentes, gran cantidad de memoria y una o más GPU o aceleradores especializados.
Estación de trabajo para IA	Sistema de escritorio especializado en ejecutar, desarrollar o entrenar modelos de inteligencia artificial. Puede integrar CPU y GPU mediante interconexiones de muy alta velocidad y utilizar grandes cantidades de memoria compartida o coherente. NVIDIA DGX Spark y DGX Station son ejemplos de esta clase de sistemas.

Computadora de placa única (SBC)	Computadora completa contenida en una pequeña placa. Integra procesador, memoria, interfaces y conectores de entrada y salida. Raspberry Pi es un ejemplo conocido.
Sistema embebido	Computadora incorporada dentro de otro dispositivo y destinada a realizar una función específica. Se encuentra en automóviles, electrodomésticos, instrumentos científicos, máquinas industriales, routers y numerosos equipos electrónicos.
Microcontrolador	Integra en un único circuito CPU, memoria, temporizadores, interfaces, conversores y entradas y salidas. Está orientado principalmente al control de dispositivos y normalmente utiliza muchos menos recursos que una computadora de propósito general.
Dispositivo IoT	Sistema embebido o microcontrolador dotado de comunicaciones de red. Puede utilizar WiFi, Bluetooth, Ethernet, redes celulares u otros medios para comunicarse con otros dispositivos o servidores.
Consola de videojuegos	Sistema especializado para gráficos y multimedia. Normalmente integra CPU, GPU y memoria dentro de una arquitectura definida por el fabricante y optimizada para una plataforma de hardware relativamente uniforme.
Supercomputadora o cluster	Sistema compuesto por numerosos nodos de cálculo conectados mediante redes de muy alta velocidad. Cada nodo puede contener CPU, GPU y otros aceleradores. El cálculo se distribuye entre gran cantidad de procesadores trabajando en paralelo.
Mainframe	Sistema de gran escala orientado a alta disponibilidad, procesamiento de grandes cantidades de transacciones, virtualización, seguridad y ejecución simultánea de numerosos trabajos y usuarios.

Una diferencia importante es la existente entre la computadora modular tradicional y las arquitecturas altamente integradas.

Una PC tradicional puede representarse esquemáticamente como:

$$\text{CPU} + \text{RAM} + \text{GPU} + \text{controladores} + \text{almacenamiento}.$$

Los componentes pueden encontrarse en circuitos separados y comunicarse mediante distintos buses.

En cambio, un SoC puede integrar gran parte de estos elementos:

$$\boxed{\text{CPU} + \text{GPU} + \text{NPU} + \text{DSP} + \text{controladores} + \text{interfaces}}$$

La tendencia moderna consiste además en incorporar procesadores especializados. Por ello una computadora ya no debe imaginarse necesariamente como una única CPU que realiza

todo el cálculo, sino como un sistema heterogéneo en el cual diferentes unidades procesan distintos tipos de operaciones.

Es interesante notar que las mac modernas comparten memoria entre el procesador y la unidad de IA, lo cual las puede hacer relevantes para ejecutar IA local.

1.2.3. Arquitecturas de procesadores

La arquitectura de un procesador puede describirse a distintos niveles. El más importante desde el punto de vista del software es la **arquitectura del conjunto de instrucciones**, o ISA (*Instruction Set Architecture*).

La ISA define las instrucciones que puede ejecutar el procesador, los registros visibles para los programas, los tipos básicos de datos, los modos de direccionamiento y otros aspectos de la interfaz entre hardware y software.

Una misma ISA puede ser implementada por procesadores internamente muy diferentes. Esa implementación concreta recibe el nombre de **microarquitectura**.

Las principales arquitecturas de instrucciones utilizadas en computación son:

ISA	**Ejemplos**	**Usos característicos**
x86-64 / AMD64	Intel, AMD	Arquitectura de 64 bits derivada de la familia x86. Ha sido dominante en computadoras personales, notebooks, estaciones de trabajo y gran cantidad de servidores. Intel y AMD producen numerosas implementaciones diferentes de esta misma ISA.
ARM64 / AArch64	Apple, Qualcomm, MediaTek, NVIDIA, Ampere y otros	Arquitectura de 64 bits de la familia ARM. Es dominante en teléfonos celulares y tablets y se utiliza ampliamente en sistemas embebidos. Actualmente también tiene una presencia importante en computadoras personales, estaciones de trabajo y servidores.
RISC-V	SiFive y numerosos diseños académicos, abiertos y comerciales	Arquitectura de instrucciones abierta y extensible. Se utiliza en microcontroladores, sistemas embebidos, investigación y enseñanza, y se está extendiendo hacia procesadores de propósito general y sistemas de mayores prestaciones.
POWER	IBM POWER	Arquitectura RISC utilizada principalmente en servidores, sistemas de alto rendimiento y algunas supercomputadoras.

PowerPC	IBM, Motorola, Freescale	Arquitectura RISC utilizada históricamente en estaciones de trabajo, computadoras Apple, consolas y sistemas embebidos.
MIPS	MIPS y múltiples implementaciones históricas	Arquitectura RISC utilizada históricamente en estaciones de trabajo, routers, sistemas embebidos y enseñanza de arquitectura de computadoras.
SPARC	Sun Microsystems, Oracle	Arquitectura RISC utilizada principalmente en estaciones de trabajo y servidores Unix. Actualmente tiene una presencia mucho menor que en décadas anteriores.
IBM Z	IBM	Arquitectura utilizada por los mainframes IBM Z. Está orientada a procesamiento masivo de transacciones, virtualización, confiabilidad, seguridad y compatibilidad entre generaciones de sistemas.

No debe confundirse la arquitectura con el fabricante.

Por ejemplo, Intel y AMD producen procesadores que ejecutan la misma arquitectura x86-64. De manera semejante, procesadores diseñados por Apple, Qualcomm, NVIDIA y otros fabricantes pueden ejecutar la arquitectura ARM64.

Tampoco debe confundirse la ISA con la microarquitectura. Dos procesadores pueden ejecutar exactamente las mismas instrucciones y, sin embargo, organizar internamente sus unidades de ejecución, cachés, predicción de saltos y procesamiento paralelo de maneras muy diferentes.

SoC y plataformas heterogéneas

Los nombres de procesadores y plataformas comerciales no constituyen necesariamente nuevas arquitecturas de instrucciones.

Un SoC puede combinar una CPU que implementa una ISA conocida con GPU, NPU, DSP y numerosos controladores especializados.

Algunos ejemplos actuales son:

Plataforma	**Componentes principales**	**Características**

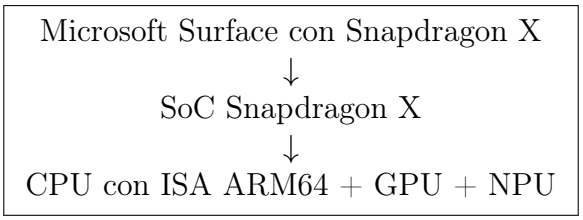
Apple Silicon	CPU ARM64 + GPU + Neural Engine	SoC utilizado en computadoras Macintosh modernas. Integra CPU, GPU, aceleradores especializados y controladores alrededor de un sistema de memoria unificada compartido por diferentes unidades de procesamiento.
Qualcomm Snapdragon X / X2	CPU ARM64 + GPU + NPU	Familia de SoC destinada a computadoras personales. Integra procesamiento general, gráficos y aceleración de inteligencia artificial y se utiliza, entre otros equipos, en computadoras Windows de la categoría Copilot+ PC.
Intel Core Ultra	CPU x86-64 + GPU + NPU	Familia de procesadores para computadoras personales que conserva la ISA x86-64 pero incorpora diferentes tipos de núcleos, gráficos integrados y una NPU dedicada al procesamiento de inteligencia artificial.
AMD Ryzen AI	CPU x86-64 + GPU + NPU	Familia de procesadores para computadoras personales que combina CPU x86-64, gráficos integrados y aceleradores dedicados a inteligencia artificial dentro del mismo sistema.
NVIDIA Grace Blackwell	CPU ARM64 + GPU Blackwell	Plataforma heterogénea orientada a inteligencia artificial y cálculo de altas prestaciones. Combina una CPU basada en ARM64 con una GPU NVIDIA y una interconexión coherente de alta velocidad. Se utiliza en sistemas de la familia DGX.

Por lo tanto, expresiones como *Snapdragon X*, *Apple Silicon* o *Grace Blackwell* no deben colocarse al mismo nivel que x86-64, ARM64 o RISC-V.

Por ejemplo:

NVIDIA DGX Spark
↓
Grace Blackwell
↓
CPU ARM64 + GPU Blackwell

y, de manera semejante,



Esto permite distinguir claramente cuatro conceptos:

- el **tipo de dispositivo**: PC, notebook, celular, servidor, etc.;
- la **plataforma o SoC**: Apple Silicon, Snapdragon, Grace Blackwell, etc.;
- la **arquitectura de instrucciones**: x86-64, ARM64, RISC-V, etc.;
- la **microarquitectura**: la organización interna concreta utilizada para implementar esa ISA.

1.2.4. Software: Sistemas Operativos

El sistema operativo (SO) es el software fundamental que administra los recursos del hardware y proporciona servicios a los programas de aplicación. Sin sistema operativo, cada programa tendría que encargarse por sí mismo de controlar el disco, la memoria, el teclado y la pantalla, lo cual sería impracticable. El SO actúa como intermediario entre el hardware y los programas del usuario.

Sus funciones principales son:

Gestión de procesos: crea, planifica y termina los procesos en ejecución. Decide qué proceso usa la CPU en cada momento mediante algoritmos de planificación, dando la ilusión de que varios programas corren simultáneamente aunque haya un solo procesador.

Gestión de memoria: asigna y libera memoria para los procesos y aplica mecanismos de protección y aislamiento. Implementa la *memoria virtual*, que proporciona espacios de direcciones propios y permite mapear memoria física, archivos y, cuando corresponde, áreas de intercambio en almacenamiento secundario.

Gestión de archivos: organiza la información en archivos y directorios sobre los dispositivos de almacenamiento, controla los permisos de acceso y mantiene la coherencia de los datos.

Gestión de dispositivos: proporciona una interfaz uniforme para acceder al hardware a través de *controladores* (drivers), de modo que los programas no necesitan conocer los detalles de cada dispositivo.

Seguridad y control de acceso: gestiona usuarios, grupos y permisos, garantizando que cada usuario solo pueda acceder a los recursos autorizados.

La importancia relativa de los sistemas operativos depende del tipo de dispositivo. En computadoras personales son comunes Microsoft Windows, macOS y GNU/Linux; en dispositivos móviles predominan Android (basado en Linux) e iOS. Linux tiene además una presencia dominante en numerosos servidores, sistemas de nube y supercomputadoras.

El núcleo o *kernel* es la parte central del sistema operativo: se carga al iniciar la computadora, controla directamente el hardware y permanece en memoria mientras el equipo está encendido. El resto del sistema operativo, las bibliotecas y las aplicaciones se apoyan en los servicios que provee el kernel.

Software: conjunto de programas y bibliotecas formados por instrucciones ejecutables o por código que puede traducirse en instrucciones que ejecuta el procesador. Los datos que esos programas leen, escriben o utilizan —incluidos los datos de configuración— no son software.

- Es un conjunto de programas.
- Que gestiona los recursos de hardware y provee servicios a los demás programas (aplicaciones), en los que abstrae las especificidades del hardware.
- El núcleo se ejecuta en modo privilegiado respecto de los programas de usuario; otras partes del sistema operativo pueden ejecutarse en espacio de usuario.
- Junto con el manejo de memoria, el manejo de interrupciones es una de las tareas más sensibles y complejas que tiene que manejar el núcleo. Las interrupciones, en este caso las asincrónicas, pueden aparecer en cualquier momento, y el kernel puede que quiera finalizar algo más antes de atenderla. El objetivo del kernel es atenderla lo más rápido posible sin dejar de procesar tanto como puede, es una tarea milimétrica.

GNU/Linux

En el caso del GNU/Linux:

- núcleo o kernel Linux.
- herramientas y bibliotecas del sistema GNU, por ejemplo `bash`, `coreutils`, `glibc`, GCC y otras utilidades de desarrollo y administración.

El núcleo y algunos controladores ejecutan código en modo privilegiado. Las aplicaciones y la mayor parte de los programas de sistema se ejecutan en espacio de usuario (*userland*), dentro de espacios de direcciones virtuales protegidos. El hardware y el sistema operativo impiden que un proceso de usuario acceda directamente a memoria del núcleo o de otros procesos salvo mediante mecanismos explícitamente autorizados.

Fuera del SO: aplicaciones o programas de sistema, que son aplicaciones de utilidad que se suministran con el SO pero no forman parte de él. Ofrecen un entorno útil para el desarrollo y ejecución de programas, siendo algunas de las tareas que realizan:

- Manipulación y modificación de archivos.

Figura 1.3: GNU y Linux, Daniel Colleti, Linus Torvalds, Mad Dog Hall

- Información del estado del sistema.
- Soporte a lenguajes de programación.
- Comunicaciones.

Distribuciones: Debian/Ubuntu, Red Hat/Fedora, openSUSE y muchas otras, mantenidas por empresas, fundaciones y comunidades. Una distribución integra y mantiene el kernel, bibliotecas, herramientas, repositorios, gestor de paquetes, configuración, actualizaciones y políticas de compatibilidad. El código fuente suele ser más portable entre distribuciones que los binarios precompilados, pero ni la compatibilidad fuente ni la binaria son universales: dependen de arquitecturas, bibliotecas, versiones y ABI.

Figura 1.4: GNU/Linux, iconos

El siguiente gráfico es muy denso para verlo directamente en un pdf. Visualice en la web.

`https://es.wikipedia.org/wiki/Archivo:Linux_Distribution_Timeline.svg`

Aquí una imagen de baja resolución al solo efecto de mostrar de qué se trata.

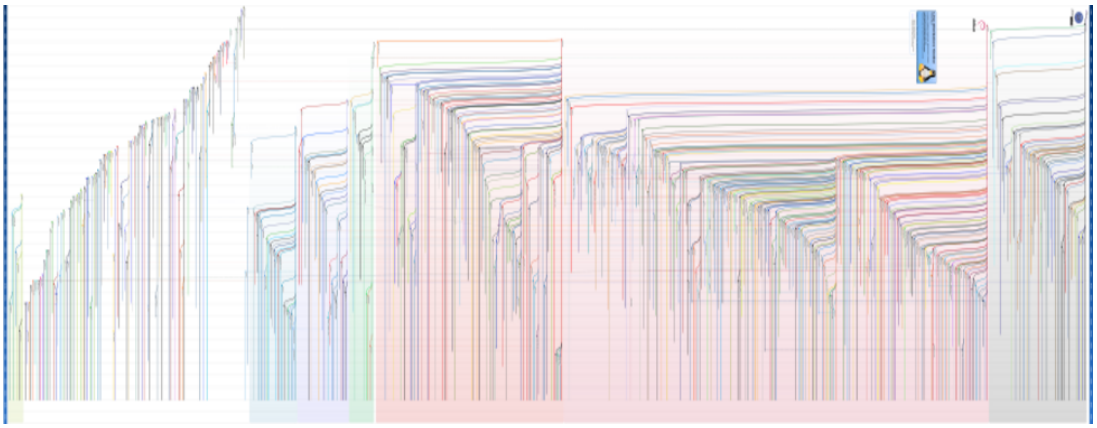


Figura 1.5: GNU/Linux, línea de tiempo

Windows

En el caso de Windows: diversas versiones, todas de Microsoft.

macOS

En el caso de macOS: diversas versiones, todas desarrolladas por Apple para sus computadoras Macintosh.

macOS combina componentes de software libre con componentes propietarios de Apple. Su base es Darwin, que incluye:

- núcleo XNU;
- componentes derivados de Mach;
- componentes provenientes de BSD y otros sistemas Unix;
- controladores y mecanismos de administración de procesos, memoria, archivos y dispositivos.

Sobre esta base Apple incorpora las bibliotecas, servicios, interfaces gráficas y aplicaciones propias que caracterizan a macOS.

El núcleo XNU es un núcleo híbrido. Combina ideas provenientes del micronúcleo Mach con componentes de BSD y mecanismos propios de Apple.

Como en otros sistemas operativos modernos, se distingue entre espacio de núcleo y espacio de usuario. Los procesos normales se ejecutan en espacios de memoria virtual protegidos y no pueden acceder directamente a la memoria del núcleo ni a la de otros procesos.

Fuera del núcleo se encuentran numerosos servicios y programas de sistema que proporcionan el entorno utilizado por las aplicaciones. Entre ellos:

- sistema de archivos;
- interfaces de usuario;

Figura 1.6: Versiones de Windows: 1.0, 3.1, 95, XP, Vista, 7, 8, 10, 11

- administración de dispositivos;
- comunicaciones y redes;
- herramientas de desarrollo;
- servicios gráficos y multimedia.

A diferencia de GNU/Linux, macOS no se organiza en distribuciones desarrolladas por diferentes organizaciones. Apple controla la integración del sistema operativo con el hardware sobre el cual se ejecuta oficialmente.

Históricamente pueden distinguirse varias etapas:

- Mac OS clásico, utilizado en las primeras computadoras Macintosh;
- Mac OS X, basado en tecnologías provenientes de NeXT y de Unix;
- OS X, denominación utilizada posteriormente;
- macOS, denominación actual de la misma familia de sistemas.

Entre las versiones modernas pueden mencionarse Mac OS X Cheetah, Puma, Jaguar, Panther, Tiger, Leopard y Snow Leopard; posteriormente Lion, Mountain Lion, Mavericks, Yosemite, El Capitan, Sierra, High Sierra, Mojave y Catalina; y luego macOS Big Sur, Monterey, Ventura, Sonoma y las versiones posteriores.

Una característica importante de macOS es su origen Unix. Esto permite utilizar una gran cantidad de herramientas tradicionales de Unix desde la línea de comandos. El intérprete de comandos y numerosas utilidades conviven con la interfaz gráfica desarrollada por Apple.

Desde 2020 Apple comenzó además la transición desde procesadores Intel hacia procesadores propios basados en la arquitectura ARM, denominados Apple Silicon. El sistema operativo mantiene una fuerte relación con la arquitectura del hardware, aunque gran parte del software puede conservar compatibilidad a nivel de código fuente y, mediante mecanismos de traducción, también ejecutar programas compilados originalmente para otra arquitectura.

1.2.5. Tareas de un SO

Gestión de procesos

Un proceso es un programa en ejecución que necesita recursos para realizar su tarea: tiempo de CPU, memoria, archivos y dispositivos de E/S. Tanto GNU/Linux como Windows (hoy) son sistemas operativos multitarea, capaces de ejecutar varios procesos al "mismo tiempo", sea secuencialmente o incluso en distintos procesadores o núcleos. El tiempo de CPU se reparte entre los procesos listos para ejecutar; la memoria puede permanecer asignada simultáneamente a muchos procesos y los dispositivos se comparten mediante controladores, colas y otros mecanismos. El SO es el responsable de:

- Parar y reanudar procesos.
- Crear y destruir procesos. Conviene distinguir mecanismos diferentes: una señal como `SIGTERM` o `SIGKILL` es un mecanismo del sistema operativo para afectar a un proceso; una división por cero genera una excepción síncrona del procesador; y una interrupción de hardware es un evento asincrónico originado por un dispositivo o controlador.
- Ofrecer mecanismos para que los procesos puedan comunicarse y se sincronicen (señales, pipes, sockets, semáforos).

La planificación de procesos mantiene información sobre las tareas que están listas para ejecutarse, las que están ejecutando y las que están bloqueadas esperando algún evento. Un planificador moderno puede desalojar un proceso para ceder la CPU a otro y aplica políticas de reparto de tiempo, prioridades, afinidad y distintas clases de planificación. Algunas políticas pueden producir inanición de tareas de baja prioridad; una técnica clásica para evitarla es aumentar gradualmente su prioridad. Cada proceso tiene además un identificador llamado *PID* (*process identifier*), que no debe confundirse con su prioridad.

Gestión de recursos

El sistema operativo administra los recursos del sistema, como los ya vistos:

- La unidad central de procesamiento (donde está alojado él o los microprocesadores).
- La memoria principal (RAM), las memorias caché y el almacenamiento secundario, que ocupan niveles diferentes de la jerarquía de memoria.
- Los dispositivos de entrada y salida: teclados, pantallas, impresoras, discos duros, dvds, pendrives, etc. El sistema accede a ellos mediante controladores: algunos son específicos, otros pertenecen a clases genéricas, y pueden estar integrados en el núcleo, cargarse como módulos o ejecutarse parcialmente en espacio de usuario.

Gestión de usuarios

Tanto GNU/Linux como Windows (hoy) son sistemas operativos multiusuarios. Varios usuarios ejecutan simultáneamente sus programas, accediendo a la vez a los recursos de la computadora. Normalmente estos sistemas operativos utilizan métodos de protección de datos, de manera que un programa no pueda usar o cambiar los datos de otro usuario. Al acceder a un sistema cada usuario puede autenticarse mediante contraseña u otros mecanismos. En GNU/Linux pueden existir varias consolas virtuales accesibles con combinaciones como `Ctrl+Alt+Fn`, pero las teclas concretas y la ubicación de la sesión gráfica dependen de la distribución y de su configuración. Todo esto implica la existencia de múltiples programas para administrar a los usuarios y sus derechos sobre diferentes recursos.

Aspectos:

- seguridad, derechos o permisos (-rw),
- listas de control de acceso (ACLs),
- UGO (Usuario, Grupo, Otros, o por sus siglas en inglés, User, Group, Others),
- Capacidades granuladas,
- Atributos extendidos y, en algunos sistemas de archivos, banderas especiales del inode, como *append-only* o *immutable*. El SO se encarga de:
- Distinguir entre uso autorizado y no autorizado.
- Especificar los controles de seguridad a realizar.
- Forzar el uso de estos mecanismos de protección.

Gestión de la memoria

El SO es el responsable de:

- Conocer qué partes de la memoria están siendo utilizadas y por quién.

- Decidir qué procesos se cargarán en memoria cuando haya espacio disponible.
- Asignar y reclamar espacio de memoria cuando sea necesario.

El sistema puede utilizar un área de intercambio (*swap*) en almacenamiento secundario para desalojar temporalmente ciertas páginas de memoria. No es una caché y no existe una regla general que exija que su tamaño sea el doble de la RAM: depende de la carga prevista, de la cantidad de memoria, del uso de hibernación y de la política del sistema.

Gestión de dispositivos

En sistemas Unix/Linux, muchos nodos de dispositivo expuestos a los procesos se clasifican en dos tipos tradicionales:

Dispositivos de caracteres: presentan al proceso una interfaz orientada a un flujo de bytes y normalmente no permiten direccionar bloques arbitrarios como un disco. Ejemplos tradicionales son terminales y puertos serie.

Dispositivos de bloques: permiten acceso a datos organizados en bloques direccionables y son la interfaz típica para discos y otros dispositivos sobre los que pueden construirse sistemas de archivos.

Como veremos en Unix, todo es un archivo. Se puede saber si una entrada representa un archivo común, un directorio, un dispositivo u otro tipo mirando el primer carácter de la salida de `ls -l`.

`-`: Archivo regular, presentado al programa como una secuencia de bytes asociada a metadatos.

`d`: Directorio: objeto que asocia nombres con otros objetos del sistema de archivos.

`b`: Dispositivo de bloques.

`c`: Dispositivo de caracteres.

`l`: Enlace simbólico, es decir, un nombre que apunta a otro archivo.

`p`: Cañería con nombre, o *named pipe*, usada para comunicar procesos.

`s`: Socket local conectado a un programa.

Gestión de módulos

Muchos núcleos permiten cargar y descargar módulos en tiempo de ejecución. Linux, por ejemplo, es un núcleo monolítico modular: algunos controladores pueden compilarse dentro del propio kernel y otros distribuirse como módulos cargables. Esta modularidad es independiente de la distinción entre núcleo monolítico y microkernel.

Gestión de archivos

El SO es responsable de:

- Construir y eliminar archivos y directorios.
- Ofrecer funciones para manipular archivos y directorios.
- Establecer la correspondencia entre archivos y unidades de almacenamiento.
- Proporcionar las operaciones de archivos y metadatos que utilizan las herramientas de copia, sincronización y respaldo. Las copias de seguridad suelen ser responsabilidad de programas o servicios específicos, no una función obligatoria del núcleo.

1.2.6. Instalación de un SO

Para instalar un sistema operativo hay que ejecutar un instalador adecuado para ese sistema. Una forma habitual consiste en arrancar la computadora desde un medio de instalación, actualmente por lo general un pendrive USB. Ese medio puede contener el instalador de GNU/Linux, Windows u otro sistema; algunas distribuciones GNU/Linux permiten además iniciar un sistema *live* antes de instalar.

También puede instalarse un sistema operativo dentro de una máquina virtual, pero en ese caso se instala sobre hardware virtual y no directamente sobre la computadora física.

Uno de los pasos críticos de una instalación sobre hardware real es lograr que la computadora arranque desde el medio elegido. En equipos actuales esto suele seleccionarse desde el menú de arranque o la configuración UEFI/BIOS.

Si el equipo ya contiene otro sistema operativo, el particionado puede requerir reducir particiones existentes para liberar espacio. Esta operación modifica estructuras fundamentales del almacenamiento y puede provocar pérdida de información si falla o se realiza incorrectamente, por lo que debe hacerse un respaldo previo.

Muchos sistemas y herramientas permiten redimensionar particiones y sistemas de archivos, incluso en ciertos casos mientras están montados; las posibilidades concretas dependen del sistema de archivos, el tipo de volumen y la herramienta utilizada.

En sistemas con varios sistemas operativos pueden prepararse particiones separadas y configurar un gestor de arranque. En equipos UEFI cada instalador puede registrar su propia entrada de arranque en el firmware, por lo que no existe una regla general que obligue a instalar GNU/Linux en último lugar. La estrategia concreta depende de los sistemas que se quieran combinar y del gestor de arranque elegido.

Particionado

1.3. Archivos: la gran abstracción

Un sistema operativo puede ofrecer abstracciones comunes para recursos diferentes. En los sistemas Unix, muchos recursos se exponen mediante descriptores y operaciones seme-

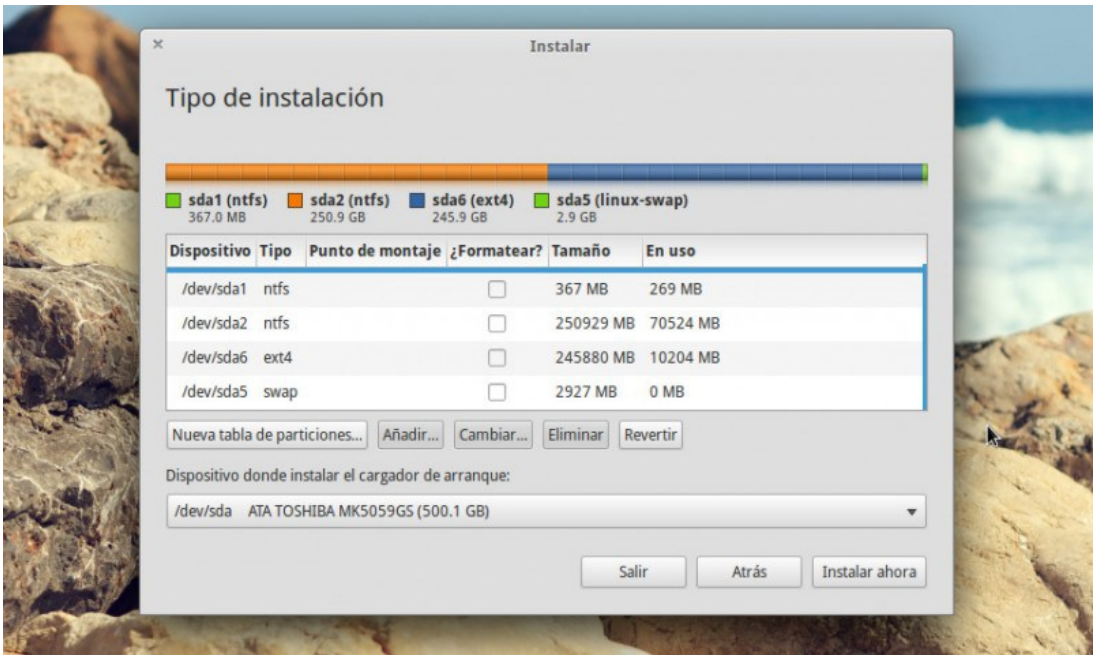


Figura 1.7: Particionado en la instalación

jantes a las usadas con archivos. Un archivo regular puede modelarse, desde el punto de vista del programa, como una secuencia finita de bytes asociada a metadatos; dispositivos, tuberías y sockets son objetos distintos que pueden compartir parte de esa interfaz.

1.3.1. Archivos en el árbol de directorios

Un archivo es la unidad básica de almacenamiento de información en un sistema operativo. Desde el punto de vista del usuario, un archivo es una secuencia de bytes con un nombre. El sistema de archivos se encarga de gestionar cómo esos bytes se almacenan físicamente en el disco.

Todo archivo tiene asociados **metadatos**: información que no forma parte de su contenido, como tamaño, propietario, permisos y marcas de tiempo. En sistemas Unix el nombre se almacena principalmente en la entrada de directorio que referencia al objeto del sistema de archivos, por lo que un mismo objeto puede tener más de un nombre mediante enlaces duros.

Los **permisos** en sistemas GNU/Linux se organizan en tres categorías: propietario, grupo y otros. Para cada categoría existen tres tipos de permiso:

Lectura (r): permite ver el contenido del archivo.

Escritura (w): permite modificarlo.

Ejecución (x): permite ejecutarlo como programa.

El comando `ls -l` muestra los permisos con una notación como `-rwxr-xr-`, que indica que el propietario puede leer, escribir y ejecutar; el grupo puede leer y ejecutar; y el resto solo puede leer.

Los archivos pueden clasificarse según su contenido:

Archivos de texto: contienen caracteres legibles por humanos codificados en ASCII, UTF-8 u otros esquemas. Los programas fuente, los archivos de configuración y los documentos en formato LaTeX son archivos de texto.

Archivos binarios: contienen datos en formato que solo los programas entienden directamente: ejecutables, imágenes, audio, video, documentos Office.

1.3.2. Archivos propiamente dichos

Los archivos son secuencias de bytes que interpretamos mediante un código, como ASCII (ver el editor de `mc`).

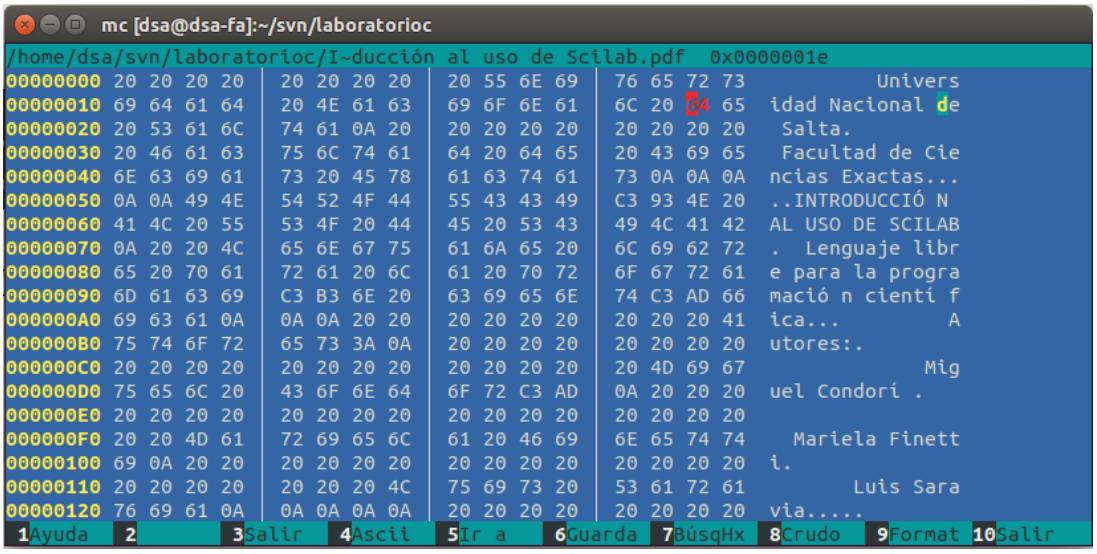


Figura 1.8: Editando un archivo

Pero pueden ser cualquier conjunto de información relacionada, definida por sus creadores. Éstos almacenan programas (en código fuente y objeto) y datos tales como imágenes, textos, información de bases de datos, etc.

Procesos y dispositivos

En Unix muchos dispositivos se exponen mediante nodos especiales que pueden abrirse y utilizarse con operaciones semejantes a las de archivos. Por ejemplo, un disco puede tener un nodo de dispositivo de bloques y cada partición puede aparecer como otro nodo.

1.3.3. Archivos y directorios

Un directorio es un objeto del sistema de archivos que mantiene asociaciones entre nombres y otros objetos del sistema de archivos, formando una jerarquía. Un sistema de archivos puede residir sobre una partición, un volumen lógico, un dispositivo o incluso una imagen, y puede montarse en un punto de esa jerarquía.

En Unix/Linux aparecen además sistemas de archivos y directorios especiales:

- `/dev` contiene nodos de dispositivo que permiten acceder a muchos dispositivos mediante controladores del núcleo;
- `/proc` es un sistema de archivos virtual que expone información de procesos y del núcleo;
- `/sys` expone información y objetos relacionados con dispositivos, buses y el modelo de dispositivos del kernel.

1.3.4. Archivos de dispositivos

Los procesos solicitan servicios al núcleo mediante llamadas al sistema. Para trabajar con dispositivos y con información del kernel pueden intervenir distintos mecanismos, entre ellos descriptores de archivos, nodos de `/dev`, `ioctl`, `mmap`, sockets, `netlink` y sistemas de archivos virtuales como `/proc` y `/sys`. Los nodos de `/dev` permiten que muchas operaciones sobre dispositivos usen interfaces semejantes a las de archivos regulares. En Windows, letras como `C:` o `D:` identifican volúmenes o unidades lógicas montadas; no constituyen nombres generales para todos los dispositivos.

1.3.5. Rutas y nombre de archivos

Normalmente los archivos y carpetas se organizan jerárquicamente. La estructura de directorios suele ser jerárquica, ramificada o "en árbol", aunque en algún caso podría ser plana. En algunos sistemas de archivos los nombres de archivos son estructurados, con sintaxis especiales para extensiones de archivos y números de versión. En otros, los nombres de archivos son simplemente cadenas de texto y los metadatos de cada archivo son alojados separadamente. En los sistemas de archivos jerárquicos, usualmente, se declara la ubicación precisa de un archivo con una cadena de texto llamada "ruta", (path). La nomenclatura para rutas varía ligeramente de sistema en sistema, pero mantienen por lo general una misma estructura. Una ruta viene dada por una sucesión de nombres de directorios y subdirectorios, ordenados jerárquicamente de izquierda a derecha y separados por algún carácter especial que suele ser una diagonal ('/') o diagonal invertida ('\') y puede terminar en el nombre de un archivo presente en la última rama del directorio. El punto representa el directorio "corriente" y los dos puntos consecutivos (`..`) representan el directorio padre. Una barra inicial indica la raíz del árbol, su ausencia que se parte del directorio corriente.

1.3.6. Ejemplo de 'ruta' en GNU/Linux

Así, por ejemplo, en un sistema tipo Unix como GNU/Linux, la ruta para la canción llamada "La canción.ogg" del usuario "Pedro" sería algo como: "/home/Pedro/Mi música/La canción.ogg" Donde: el primer '/' representa el directorio raíz donde está montado todo el sistema de archivos. 'home/Pedro/Mi música/' es la ruta del archivo. 'La canción.ogg' es el nombre del archivo. Los espacios son válidos en nombres de archivos. En un shell deben protegerse mediante comillas o caracteres de escape porque el espacio separa palabras en la línea de comandos.

1.3.7. Ejemplo de 'ruta' en un sistema Windows

Un ejemplo análogo en un sistema de archivos de Windows (específicamente en Windows 8) se vería como: "C:\Users\Pedro\Music\canción.mp3" donde: "C:" es la unidad de almacenamiento en la que se encuentra el archivo. "\Users\Pedro\Music\" es la ruta del archivo. "canción.mp3" es el nombre completo del archivo y ".mp3" es su extensión.

1.3.8. Extensiones

*.?, *.bat, *.exe, *.com, *.txt, *.sh, *.mp3 '.mp3' es una extensión, es decir, una convención incluida en el nombre del archivo. En sistemas como Windows se utiliza habitualmente para asociar nombres con aplicaciones y sugerir el formato del contenido, pero la extensión no constituye por sí sola una identificación segura del tipo real de archivo. Sin embargo muchas veces Windows oculta la extensión del nombre y representa el tipo con un icono. Asteriscos y preguntas para búsquedas. En los listados de archivos y caminos se pueden reemplazar letras por comodines y con esto se incluye a todos los que coinciden.

1.3.9. Tipo de sistemas de archivos

Un sistema de archivo de disco está diseñado para el almacenamiento de archivos en una unidad de disco, que puede estar conectada directa o indirectamente a la computadora. Existen diferentes sistemas de archivos, es decir, existen diferentes formas de organizar la información que se almacena en las memorias (normalmente discos) de los ordenadores. Por ejemplo, existen FAT32, exFAT, NTFS, ext4, XFS, Btrfs y muchos otros. Sus diferencias afectan límites de tamaño, permisos, integridad, journaling o copy-on-write, recuperación y otras funciones. FAT32, muy usado por su amplia compatibilidad entre sistemas, limita un archivo individual a $2^{32} - 1$ bytes (aproximadamente 4 GiB); NTFS y los sistemas de archivos Unix modernos permiten archivos mucho mayores. Un sistema de archivos se crea al formatear una partición, volumen, dispositivo o imagen; esto puede hacerse durante la instalación del sistema operativo o posteriormente.

1.3.10. Capacidades de los sistemas de archivos

- Estrategias para reducir, administrar o reparar la fragmentación; ningún sistema de archivos está universalmente libre de ella,
- Capacidad de enlaces simbólicos o duros,
- Integridad del sistema de archivos (Journaling),
- Soporte para archivos dispersos,
- Soporte para cuotas de discos,
- Soporte de crecimiento del sistema de archivos nativo.

1.3.11. El árbol genérico

En los sistemas Unix no existe un único prefijo en el que se instale todo un paquete. Un mismo paquete de la distribución puede repartir sus archivos por varios directorios según su función: ejecutables en `/usr/bin`, bibliotecas en `/usr/lib`, configuración en `/etc`, datos variables en `/var`, etc.

Los programas instalados localmente por el administrador, fuera del gestor de paquetes de la distribución, suelen utilizar `/usr/local` como prefijo. `/` no es un prefijo alternativo equivalente a `/usr` o `/usr/local`: es la raíz de toda la jerarquía. En muchos sistemas actuales, además, `/bin`, `/sbin` y `/lib` están integrados mediante *usr-merge* con sus equivalentes bajo `/usr`.

Dentro de árboles como `/usr` y `/usr/local` aparecen subdirectorios genéricos. No todos los paquetes usan todos ellos: cada archivo se ubica según su función.

`bin/`: Programas ejecutables destinados al uso normal de los usuarios.

`sbin/`: Programas ejecutables de administración del sistema. Históricamente se separaban de `bin/` porque estaban destinados al administrador o a tareas de mantenimiento.

`lib/`: Bibliotecas, módulos, complementos y archivos binarios usados por los programas. En sistemas de 64 bits puede aparecer también `lib64/`.

`libexec/`: Programas auxiliares usados internamente por otros programas, no pensados para ser ejecutados directamente por el usuario.

`etc/`: Archivos de configuración. En el sistema general suelen estar en `/etc`; en instalaciones locales pueden aparecer bajo `/usr/local/etc`.

`include/`: Encabezados de las librerías.

`share/`: Datos independientes de la arquitectura de la máquina: documentación, páginas de manual, archivos de ayuda, iconos, traducciones, ejemplos, plantillas y otros recursos compartibles.

`share/man/`: Páginas de manual.

`share/doc/`: Documentación del paquete o programa.

`share/info/`: Documentación en formato `info`, habitual en el entorno GNU.

`share/locale/`: Archivos de traducción e internacionalización.

`src/`: Código fuente. No todos los paquetes instalan aquí sus fuentes; se usa sobre todo para fuentes del sistema, del núcleo o de software compilado localmente.

`var/`: Datos variables producidos por programas: estado, cachés, colas, registros o información que cambia durante la ejecución. En el sistema general estos datos suelen ubicarse bajo `/var`.

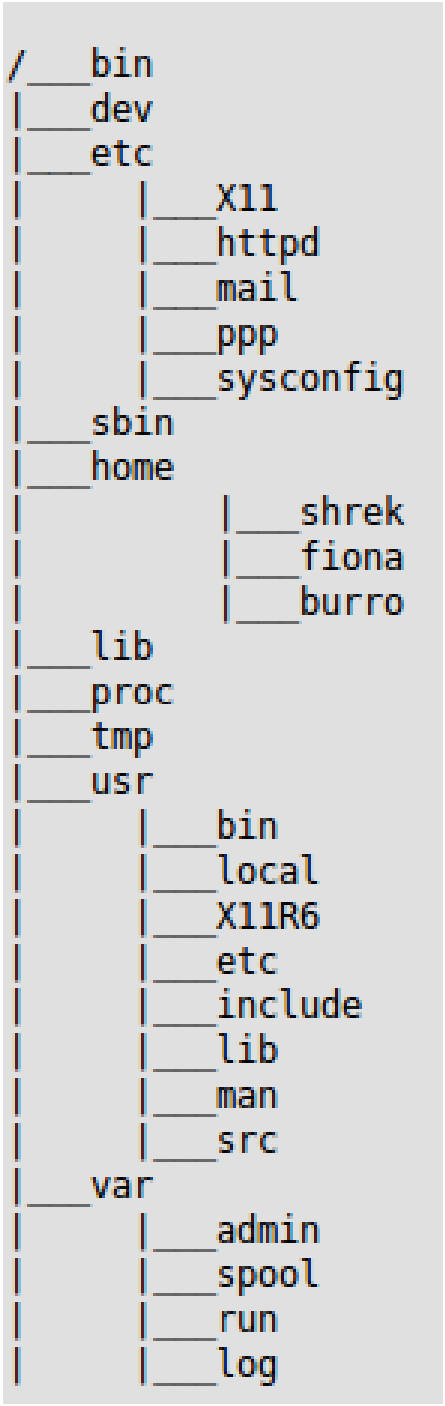


Figura 1.9: Sistema de archivos

1.3.12. El árbol estándar

Estructura del sistema de archivos de los GNU/Linux.

Nos referimos con esto a los directorios reales montados en un sistema GNU/Linux.

Como verán los directorios genéricos se encuentran repetidos en diferentes árboles, según lo visto.

/: Raíz del sistema de archivos.

/bin: Contiene programas ejecutables básicos disponibles para todos los usuarios. En muchas distribuciones actuales `/bin` es un enlace simbólico a `/usr/bin`, pero conserva su sentido histórico: ubicar comandos esenciales como `cp`, `ls`, `mv`, `cat` o `sh`.

/dev: Contiene nodos o archivos especiales de dispositivo que ofrecen a los procesos una interfaz hacia controladores del núcleo. Allí aparecen, por ejemplo, dispositivos de almacenamiento, terminales y puertos serie. No son los controladores mismos: el controlador reside en el kernel y el nodo de `/dev` es una forma de acceder a él.

/dev/console: Representa el dispositivo de consola del sistema. Puede estar asociado a una consola virtual, una terminal serie u otro dispositivo configurado como consola; no equivale simplemente al monitor.

/dev/ttyS*, /dev/ttyUSB*, /dev/ttyACM*: Representan puertos serie físicos o adaptadores serie modernos conectados por USB. Los antiguos dispositivos `/dev/cua*` pertenecen a una convención histórica ya abandonada.

/dev/hda, /dev/hda1: Nombres históricos usados para discos IDE antiguos. Hoy casi no se usan en sistemas modernos; se conservan aquí para entender documentación vieja.

/dev/sda: Los nombres de dispositivo que comienzan con `sd` identifican muchos discos vistos por el núcleo como dispositivos de bloque SCSI o compatibles: discos SATA, USB, virtuales y otros. Las particiones se numeran como `/dev/sda1`, `/dev/sda2`, etc. En discos NVMe se usan nombres como `/dev/nvme0n1` y `/dev/nvme0n1p1`.

/dev/lp0: Los nombres que comienzan por `lp` se utilizan para impresoras o puertos paralelos. `/dev/lp0` es el primer dispositivo paralelo de esa clase en Linux; es funcionalmente comparable al histórico `LPT1` de DOS, pero no es el mismo nombre de dispositivo.

/dev/null: Es usado como "agujero negro": cualquier dato enviado a este dispositivo desaparece. Si desea suprimir la salida por pantalla de una orden, podría enviar la salida a `/dev/null`.

/dev/tty[0-9]*: Hace referencia a las consolas virtuales del sistema. `/dev/tty1` corresponde a la primera, `/dev/tty2` a la segunda, etc. El cambio entre consolas puede hacerse mediante combinaciones de teclas de función cuya forma concreta depende de la distribución y de la configuración.

/dev/pty*, /dev/pts/*: Representan pseudo-terminales. En sistemas actuales se usan normalmente entradas bajo `/dev/pts/`, por ejemplo cuando se abre una terminal gráfica, una sesión SSH o una consola remota.

`/etc`: Contiene archivos de configuración del sistema, entre ellos `/etc/passwd` y configuraciones de servicios. Los mecanismos de inicio dependen de la distribución; muchos sistemas actuales utilizan systemd, mientras que `/etc/rc*` corresponde a convenciones históricas o a otros sistemas de inicio.

`/sbin`: Contiene programas de administración del sistema. En muchas distribuciones actuales `/sbin` puede ser un enlace simbólico a `/usr/sbin`. Históricamente reunía comandos necesarios para arrancar, reparar o administrar el sistema.

`/home`: Contiene los directorios "home" de los usuarios. Por ejemplo, `/home/shrek` es el directorio del usuario shrek.

`/lib`: Contiene bibliotecas compartidas necesarias para ejecutar programas básicos del sistema. En sistemas actuales puede aparecer como enlace a `/usr/lib` o dividirse según arquitectura, por ejemplo `/lib64`.

`/boot`: Contiene archivos necesarios para iniciar el sistema, como el núcleo Linux, imágenes `initramfs`, cargadores de arranque y archivos asociados al arranque.

`/root`: Directorio personal del usuario administrador `root`. No debe confundirse con `/`, que es la raíz del sistema de archivos.

`/run`: Contiene información variable creada desde el arranque del sistema: identificadores de procesos, sockets, archivos de bloqueo y datos temporales usados por servicios en ejecución.

`/sys`: Sistema de archivos virtual que expone información del núcleo sobre dispositivos, controladores, buses y parámetros del sistema. No contiene archivos ordinarios guardados en disco.

`/media`: Punto habitual donde se montan automáticamente medios removibles, como pendrives, discos externos, tarjetas de memoria o unidades ópticas.

`/mnt`: Punto de montaje usado tradicionalmente para montajes manuales o temporales realizados por el administrador.

`/opt`: Contiene paquetes de software adicionales instalados fuera de la estructura principal de la distribución. Suele usarse para aplicaciones grandes o de terceros.

`/srv`: Contiene datos servidos por el sistema, por ejemplo archivos de un servidor web, FTP, repositorios o servicios de red.

`/proc`: Es un sistema de archivos virtual generado por el núcleo al ser consultado. Expone información sobre procesos y numerosos parámetros del sistema; sus entradas no son archivos regulares almacenados ni en disco ni simplemente en RAM.

`/tmp`: Muchos programas tienen la necesidad de generar información temporal y guardarla en un fichero temporal. El lugar habitual para esos ficheros es `/tmp`.

`/usr`: Contiene la mayor parte del software instalado por la distribución: programas, bibliotecas, documentación, páginas de manual, archivos compartidos y código de apoyo. En sistemas modernos muchos directorios históricos de la raíz, como `/bin`, `/sbin` y `/lib`, pueden estar integrados bajo `/usr`.

`/usr/X11R6`: Directorio histórico usado por el sistema X Window. En distribuciones actuales casi todo el sistema gráfico se integra bajo `/usr/bin`, `/usr/lib` y `/usr/share`; además, muchos escritorios modernos pueden usar Wayland además de X.

`/usr/bin`: Contiene gran parte de los programas ejecutables instalados por el sistema. En distribuciones con *usr merge*, `/bin` suele ser un enlace hacia `/usr/bin`.

`/usr/include`: Contiene los ficheros de cabecera para el compilador de C.

`/usr/share`: Contiene datos independientes de la arquitectura: documentación, páginas de manual, iconos, archivos de configuración de ejemplo, traducciones y otros recursos compartibles.

`/usr/lib`: Contiene bibliotecas y otros archivos auxiliares, a menudo dependientes de la arquitectura o privados de determinados paquetes. La configuración global del sistema se encuentra principalmente bajo `/etc`.

`/usr/local`: Contiene programas específicos de su sistema.

`/usr/share/man`: Contiene las páginas de manual. Reemplaza en la práctica al uso histórico de `/usr/man`.

`/usr/src`: Puede contener código fuente instalado localmente o fuentes del núcleo y otros paquetes; su contenido es opcional y depende de la distribución.

`/var`: Contiene directorios que a menudo cambian su tamaño o tienden a crecer.

`/var/lib`: Contiene datos persistentes modificados por programas y servicios, como bases de datos internas, estado de paquetes, contenedores o información mantenida por aplicaciones.

`/var/cache`: Contiene datos cacheados que pueden regenerarse: paquetes descargados, índices, miniaturas u otros archivos usados para acelerar operaciones.

`/var/tmp`: Contiene archivos temporales que, a diferencia de muchos contenidos de `/tmp`, suelen preservarse entre reinicios.

`/var/log`: Contiene históricos del sistema, errores, problemas, sesiones de presentación e intentos fallidos.

`/var/spool`: Contiene ficheros que van a ser pasados a otro programa.

`/var/mail`: En algunas distribuciones contiene los buzones locales o *mail spools* donde se entrega correo recibido para usuarios del sistema.

Directorios

Un directorio, también llamado carpeta, es un contenedor que puede alojar archivos y otros directorios, formando una estructura jerárquica en árbol. En los sistemas Unix y GNU/Linux, el punto de partida de esta jerarquía es el directorio raíz, representado por `/`.

La estructura estándar de directorios en GNU/Linux incluye:

`/`: raíz del sistema, punto de partida de toda la jerarquía.

`/home`: contiene los directorios personales de los usuarios. El directorio del usuario `diego` sería `/home/diego`.

`/bin`, `/usr/bin`: programas ejecutables disponibles para todos los usuarios.

`/etc`: archivos de configuración del sistema.

`/var`: datos variables: registros del sistema, bases de datos, correo.

`/tmp`: archivos temporales; su limpieza depende de la política del sistema y no se garantiza que todos se eliminen en cada reinicio.

`/dev`: archivos especiales que representan dispositivos de hardware.

La posición de un archivo en la jerarquía se describe mediante su *ruta* (*path*). Una **ruta absoluta** parte desde la raíz: `/home/diego/documentos/informe.tex`. Una **ruta relativa** parte desde el directorio actual: si estamos en `/home/diego`, la ruta relativa al mismo archivo es `documentos/informe.tex`.

El símbolo `.` representa el directorio actual y `..` representa el directorio padre. En los shells habituales, `~` es expandido por el propio shell al directorio personal del usuario actual.

1.4. Comandos y sus Intérpretes

1.4.1. Terminal y comandos

Una terminal es una interfaz de texto para interactuar con programas que esperan un dispositivo terminal. En un escritorio gráfico suele utilizarse un emulador de terminal; la consola del sistema es un caso particular de terminal. Dentro de una terminal suele ejecutarse un *shell*, que interpreta comandos. Durante décadas fue una de las principales formas de interacción directa con computadoras, junto con otros mecanismos históricos como procesamiento por lotes, teletipos, tarjetas y cintas. Hoy sigue siendo la herramienta más potente y eficiente para administrar sistemas, automatizar tareas y desarrollar software.

El programa que interpreta los comandos escritos por el usuario se llama *shell*. Los más comunes en sistemas GNU/Linux son `bash` (*Bourne Again Shell*) y `zsh`. En Windows existe PowerShell.

Un comando tiene la forma general:

```
comando [opciones] [argumentos]
```

Algunos comandos esenciales en sistemas GNU/Linux:

`ls`: lista el contenido de un directorio. `ls -la` muestra todos los archivos, incluyendo ocultos, con sus permisos y tamaños.

`cd`: cambia el directorio actual. `cd /home/usuario` va a esa ruta absoluta; `cd ..` sube un nivel.

`pwd`: muestra la ruta del directorio actual (*print working directory*).

`cp`, `mv`, `rm`: copian, mueven y eliminan archivos respectivamente.

`mkdir`: crea un directorio.

`cat`, `less`, `head`, `tail`: muestran el contenido de archivos de texto.

`grep`: busca texto dentro de archivos o en la salida de otro comando. `grep 'error' registro.txt` muestra todas las líneas que contienen la palabra "error".

`man`: muestra páginas de manual de comandos y otros temas que dispongan de una. `man ls` muestra la página de manual de `ls`.

`chmod`: cambia los permisos de un archivo.

`ps`, `top`, `htop`: muestran los procesos en ejecución y su uso de recursos.

`ssh`: establece una conexión segura con otra computadora a través de la red.

Una característica poderosa de la terminal es el *encadenamiento* de comandos mediante tuberías (*pipes*, símbolo `|`): la salida de un comando se convierte en la entrada del siguiente. Por ejemplo, `ls -la | grep '[.]pdf$'` lista las entradas mostradas por `ls -la` y conserva las líneas cuyos nombres terminan en `.pdf`.

Los *scripts de shell* permiten automatizar secuencias de comandos guardándolas en un archivo de texto ejecutable, lo que hace posible realizar tareas repetitivas con una sola instrucción.

1.4.2. Comandos

Un "comando" puede invocar:

- un programa ejecutable,
- un comando interno del *shell*.

En un shell, un comando se introduce de forma interactiva o aparece dentro de un script. El shell puede entonces ejecutar un programa externo o una orden interna. Una interfaz gráfica también puede lanzar programas, pero hacerlo mediante un icono no implica necesariamente ejecutar un comando de shell. De igual modo, un evento del sistema puede provocar acciones o iniciar procesos sin que exista una línea de comandos.

Conviene distinguir opciones y argumentos. En `ls -l`, `-l` es una opción que modifica el comportamiento de `ls`. En `cp origen destino`, `origen` y `destino` son argumentos posicionales. A nivel del proceso todos forman parte de la lista de argumentos, pero la sintaxis de la línea de comandos distingue normalmente estas funciones.

1.4.3. Comandos compuestos

Una de las características de los sistemas *nix es la capacidad de combinar comandos simples y complejos para obtener comandos compuestos. Un comando compuesto puede consistir en una lista de comandos separados por el carácter de punto y coma (`;`).

```
$ comando1 ; comando2 ; comando3 ; ...
```

En múltiples lenguajes de programación, como C, Perl o Ruby, el punto y coma se utiliza como terminador o separador de sentencias, no como final físico de línea: una sentencia puede ocupar varias líneas y una línea puede contener varias sentencias. En el shell, el punto y coma permite separar comandos en una misma línea.

1.4.4. ¿Qué es el shell?

Se utiliza un programa especial llamado shell para interpretar los comandos. El shell provee una interfaz al sistema y suele correr en una terminal. Obtiene la información del usuario y ejecuta programas en base a esa información. Cuando el programa finaliza su ejecución, despliega la salida del programa. Por esta razón el shell es conocido como el intérprete de comandos. En GNU/Linux suele usarse `bash` u otros shells como `zsh`. En Windows existen el Símbolo del sistema y, como entorno moderno, PowerShell; además WSL permite ejecutar entornos GNU/Linux dentro de Windows. Cygwin sigue siendo una alternativa para disponer de muchas herramientas de estilo Unix. El shell es un lenguaje de programación completo basado en comandos con variables de entorno, funciones y estructuras de control.

1.4.5. Ejemplo, comando date

Al introducir el comando:

```
$ date
```

el sistema puede responder, por ejemplo:

```
Fri Mar 19 21:34:59 CST 2004
```

El comando `date` despliega el nombre del día, el mes, el número de día del mes, la hora, la zona horaria y el año del sistema. Después de ejecutar el comando, el sistema vuelve a mostrar el carácter `$`. Ese carácter indica el *prompt*, o línea de comando. Cuando el sistema está en el *prompt*, se puede teclear el nombre de un comando y presionar Enter. Mientras el comando está siendo ejecutado, el *prompt* no se muestra. Cuando el comando finaliza, el *prompt* aparece nuevamente.

1.4.6. Algunos Comandos

Equivalencias entre comandos de DOS/Windows y de Unix (DOS → Unix):

- `copy` → `cp`
- `ren` → `mv`
- `type` → `cat`
- `md` → `mkdir`
- `del` → `rm`
- `dir` → `ls`
- `rd` → `rmdir`
- `cd` → `cd`

1.4.7. Redirección

Las redirecciones y cañerías permiten modificar la entrada y la salida de los comandos.

`>`: Redirige la salida estándar a un archivo, reemplazando su contenido.

`>>`: Agrega la salida estándar al final de un archivo.

`<`: Toma la entrada estándar desde un archivo.

`<<`: Permite entregar texto al comando hasta encontrar una marca de finalización.

`2>`: Redirige la salida de error.

`|`: Conecta la salida de un comando con la entrada de otro.

Ejemplos:

```
$ ls > guardado
```

envía el listado al archivo `guardado`.

```
$ ls | sort
```

ordena la salida de `ls`.

1.4.8. Man e info

El comando `man` muestra las páginas de manual del sistema y es la referencia autoritativa: cuando no sepa qué hace un comando o qué opciones acepta, consúltelo antes que a Google.

Dentro de la página use `Space`/`b` para avanzar y retroceder, `/patrón` para buscar, `n` para siguiente coincidencia, `q` para salir.

Las páginas de manual siguen convenciones de estructura, pero no todas contienen exactamente las mismas secciones. En `man ls` identifique las que estén presentes, por ejemplo `NAME`, `SYNOPSIS`, `DESCRIPTION`, `OPTIONS`, `EXAMPLES` o `SEE ALSO`, y anote qué información aporta cada una.

El manual está dividido en secciones numeradas: 1 = comandos de usuario, 2 = syscalls, 3 = funciones de biblioteca, 4 = dispositivos, 5 = formatos de archivo, 6 = juegos, 7 = miscelánea, 8 = administración. Compare:

Así:

```
man 1 passwd        # el comando passwd
man 5 passwd        # el formato del archivo /etc/passwd
```

Cuando no recuerde el nombre exacto de un comando, use `apropos` (equivalente a `man -k`) para buscar por palabra clave. Ejemplo: para buscar comandos relacionados con procesos:

```
apropos process
man -k copy
```

El sistema `info`, utilizado especialmente por el proyecto GNU, ofrece documentación hipertextual. Los documentos Info suelen generarse a partir de fuentes Texinfo, de las que también pueden producirse otros formatos como HTML o PDF. No debe suponerse que la documentación Info se genera normalmente a partir de páginas `man`.

1.5. Sistema gráfico X - escritorio, gestión de ventanas

El sistema X Window es un sistema gráfico con arquitectura cliente–servidor y capacidad de funcionamiento distribuido. Fue durante décadas la base dominante de los escritorios Unix y GNU/Linux. En sistemas actuales convive con Wayland, que es el protocolo predeterminado en numerosos entornos modernos; X continúa siendo importante por compatibilidad, incluso mediante XWayland. El diseño de X permite que clientes y servidor se encuentren en máquinas distintas, aunque hoy el acceso remoto se realiza normalmente mediante mecanismos seguros como el reenvío X sobre SSH.

1.5.1. Puntos básicos

Una sesión de X contiene ventanas que son recursos gestionados por el servidor X y pertenecen a programas cliente. Un mismo proceso puede crear varias ventanas y una ventana no equivale a un proceso. Al igual que los programas en cualquier otro sistema,

los programas que se ejecutan en X varían en cuanto a su funcionalidad. Algunos necesitan de la interacción con el usuario y otros solo despliegan datos en la pantalla. A la ventana de fondo se la conoce como ventana raíz (root window). Las ventanas de aplicación se despliegan encima de la ventana raíz. Las ventanas forman una jerarquía: la ventana raíz ocupa la cima y es ancestro de todas las demás; una ventana puede ser hija de otra ventana intermedia.

1.5.2. Cliente -– servidor, eventos

X utiliza una arquitectura cliente–servidor: el servidor X administra recursos de pantalla y entrada y los clientes son las aplicaciones. X define un protocolo independiente del transporte; una conexión local suele usar sockets Unix y también puede transportarse por red cuando la configuración y los controles de acceso lo permiten. Por ello clientes y servidor pueden incluso ejecutarse en máquinas o plataformas distintas. El servidor actúa como una interfaz entre los programas clientes y el hardware físico. Cuando se proporcionan datos, ya sea por medio del teclado o el mouse, el servidor acepta la entrada y es responsable de pasarla al cliente. Esta información se pasa al cliente como un evento. Una tecla apretada o el movimiento del mouse generan un evento ante el cual reaccionará el cliente.

1.5.3. Entornos gráficos de escritorio

Existen entornos de escritorios, los cuales proveen un manejador de ventanas, utilidades varias y un framework de desarrollo que nos permitirán desarrollar aplicaciones que se integren al entorno perfectamente, compartiendo el look&feel del mismo y aprovechando servicios como arrastrar y soltar, servicio de impresión centralizado, un portapapeles común, etc. Algunos de estos entornos son KDE, y Gnome.

1.5.4. KDE

KDE ofrece para GNU/Linux un entorno gráfico de configuración sencillo y con una apariencia homogénea entre las distintas aplicaciones. KDE es un proyecto de un grupo de programadores que se formó en 1996 y que siguen trabajando en el proyecto junto con muchos otros que fueron incorporándose. Ofrece su propio gestor/compositor de ventanas llamado KWin y aplicaciones integradas; el administrador de archivos habitual de KDE Plasma es Dolphin, mientras que Konqueror continúa existiendo como navegador y gestor de archivos multipropósito. KDE está desarrollado con las bibliotecas gráficas Qt.

1.5.5. Gnome

GNOME nació con el nombre “GNU Network Object Model Environment”, pero el proyecto dejó hace tiempo de usar esa expansión como definición de su nombre. GNOME es hoy un proyecto y entorno de escritorio de software libre vinculado históricamente al ecosistema GNU. Su interfaz actual se organiza alrededor de GNOME Shell, que integra la barra superior, la vista de Actividades, el lanzador de aplicaciones, la gestión de ventanas

y los espacios de trabajo. El entorno incluye además aplicaciones y servicios integrados, configuración del sistema, notificaciones, búsqueda, accesibilidad, internacionalización y mecanismos comunes de interacción entre aplicaciones.

1.5.6. Interfaz gráfica de usuario (Graphical shells), Unity

Las interfaces gráficas permiten abrir, cerrar, mover o cambiar el tamaño de ventanas y cambiar el foco entre aplicaciones. Pueden formar parte de un entorno de escritorio o componerse de elementos separados. Unity fue desarrollado por Canonical y fue el entorno predeterminado de Ubuntu durante varios años; hoy tiene principalmente interés histórico frente a los escritorios actuales basados en GNOME, KDE Plasma y otros.

1.5.7. Windows

Windows dispone de un sistema gráfico y un shell de escritorio propios que han evolucionado entre versiones. Conviene distinguir estos componentes de la arquitectura del núcleo: la interfaz gráfica no es simplemente una parte "monolítica e inseparable" del sistema operativo.

1.6. Estándares

Un estándar es una especificación técnica acordada que permite que componentes y dispositivos de distintos fabricantes sean compatibles entre sí. En informática, los estándares son fundamentales: garantizan que un teclado USB funcione en cualquier computadora o que un monitor HDMI pueda conectarse a cualquier equipo que tenga ese puerto.

USB (*Universal Serial Bus*): familia de interfaces serie para conectar periféricos y transportar datos y energía. USB-C designa el conector físico reversible, pero no garantiza por sí solo una velocidad, capacidad de video o potencia determinada: esas funciones dependen de la versión USB, del puerto, del cable y de los modos alternativos implementados. USB4 admite distintas velocidades según generación e implementación, incluyendo enlaces de hasta 80 Gb/s en USB4 Version 2.0.

HDMI (*High Definition Multimedia Interface*): transmite audio y video digital de alta definición en un único cable. Utilizado para conectar computadoras a monitores y televisores.

DisplayPort: interfaz digital para video y audio, muy utilizada en monitores de PC. Según la versión puede ofrecer distintas combinaciones de resolución, frecuencia y profundidad de color, y algunas implementaciones permiten conectar múltiples monitores en cadena (*daisy chain*). Su capacidad no puede compararse en general con HDMI sin especificar las versiones concretas.

PCIe (*PCI Express*): bus de alta velocidad para tarjetas de expansión internas: tarjetas gráficas, controladoras de red, unidades NVMe. Se organiza en carriles (*lanes*); un enlace PCIe x16 utiliza 16 carriles. Una ranura física de tamaño x16 puede estar cableada con menos carriles.

SATA (*Serial ATA*): interfaz estándar para discos duros y unidades SSD dentro del gabinete.

ATX (*Advanced Technology eXtended*): estándar que define el tamaño físico de las motherboards, la disposición de los conectores y la forma del gabinete. El conector ATX de 24 pines provee energía a la placa; el conector EPS de 8 pines alimenta específicamente a la CPU.

WiFi (IEEE 802.11): estándar para redes inalámbricas de área local. Las versiones más recientes son WiFi 6 (802.11ax) y WiFi 7 (802.11be), con velocidades teóricas de varios gigabits por segundo.

1.6.1. Representación de la información

Toda la información que procesa una computadora digital está representada en forma binaria: secuencias de ceros y unos. Esta elección no es arbitraria: los circuitos electrónicos son más simples y confiables cuando solo deben distinguir dos estados (tensión alta / tensión baja, magnetizado / no magnetizado, refleja luz / no refleja luz).

El bit y el byte

La unidad mínima de información es el **bit** (*binary digit*): puede valer 0 o 1. Con n bits es posible representar 2^n valores distintos. Ocho bits forman un **byte**, que puede representar 256 valores diferentes (del 0 al 255).

Las unidades de medida se expresan en múltiplos del byte:

Kilobyte (kB): $10^3 = 1{,}000$ bytes; **kibibyte (KiB)**: $2^{10} = 1{,}024$ bytes.

Megabyte (MB): 10^6 bytes; **mebibyte (MiB)**: 2^{20} bytes.

Gigabyte (GB): 10^9 bytes; **gibibyte (GiB)**: 2^{30} bytes.

Terabyte (TB): 10^{12} bytes; **tebibyte (TiB)**: 2^{40} bytes.

ASCII

Para representar texto es necesario asignar un número a cada carácter. El estándar ASCII (*American Standard Code for Information Interchange*), definido en 1963, asigna valores del 0 al 127 (7 bits) a los caracteres del alfabeto inglés, los dígitos, los signos de puntuación y algunos caracteres de control. Por ejemplo, la letra ‘A’ corresponde al valor 65, ‘a’ al 97 y el espacio al 32.

ASCII fue diseñado principalmente para el inglés y resulta insuficiente para representar de forma general muchos otros idiomas: no contiene caracteres acentuados, la ñ, el alfabeto griego, el cirílico, ni los caracteres asiáticos. La extensión a 8 bits (256 valores) dio lugar a numerosas codificaciones incompatibles para distintos idiomas, lo que generó décadas de problemas de incompatibilidad: un archivo creado en un sistema con una codificación era ilegible en otro sistema con una codificación diferente.

Fin de línea (CRLF vs LF) diferentes sistemas DOS/Windows vs. Unix representan de diferente forma el fin de línea/retorno de carro. Al pasar archivos de uno a otro sistema hay que tener eso en cuenta.

Unicode

Unicode es un estándar internacional que asigna un número único (*code point*) a cada carácter de todos los sistemas de escritura del mundo. La cantidad crece con cada versión del estándar, por lo que conviene evitar fijar aquí una cifra que envejece rápidamente; incluye alfabetos latinos, griegos, cirílicos, árabes, hebreos, chinos, japoneses, coreanos, jeroglíficos egipcios, emojis y muchos más.

Los *code points* se expresan en notación hexadecimal con el prefijo U+. La letra 'A' es U+0041, la 'ñ' es U+00F1, el símbolo del euro '€' es U+20AC y el carácter chino (*zhōng*, "centro") 中 corresponde al code point U+4E2D.

Unicode define qué carácter corresponde a cada número, pero no cómo esos números se almacenan en bytes. Para eso existen distintas *codificaciones*.

UTF-8 y UTF-16

UTF-8 (*Unicode Transformation Format, 8 bits*) es la codificación dominante en la web y en sistemas GNU/Linux. Usa entre 1 y 4 bytes para representar cada carácter según su valor:

- Los 128 primeros caracteres (U+0000 a U+007F, exactamente el conjunto ASCII) se codifican en un solo byte, con el mismo valor que en ASCII. Esto garantiza compatibilidad total con ASCII: cualquier texto ASCII es UTF-8 válido.
- Los puntos de código U+0080 a U+07FF usan 2 bytes.
- Los puntos de código U+0800 a U+FFFF usan 3 bytes; este rango contiene gran parte del plano multilingüe básico.
- Los puntos de código U+10000 a U+10FFFF usan 4 bytes; allí se encuentran muchos caracteres suplementarios, incluidos numerosos emojis y escrituras antiguas.

Esta codificación de longitud variable hace que UTF-8 sea muy eficiente para texto principalmente en inglés (un byte por carácter) y razonablemente eficiente para otros idiomas.

UTF-16 usa unidades de código de 16 bits. Los caracteres del plano multilingüe básico se representan normalmente con una unidad; los puntos de código suplementarios requieren dos unidades de 16 bits (un *par subrogado*). Las interfaces de cadenas de Java y JavaScript se expresan en términos de unidades UTF-16, y las API "wide" de Windows utilizan UTF-16, aunque las implementaciones internas puedan aplicar optimizaciones de almacenamiento.

Una complicación adicional de UTF-16 es el orden de los bytes dentro de cada unidad de 16 bits: el estándar admite *big-endian* (el byte más significativo primero) y *little-endian* (el menos significativo primero). Cuando el contexto no especifica el orden de bytes, puede utilizarse una marca de orden de byte (BOM, *Byte Order Mark*), U+FEFF, al comienzo del texto. No todos los archivos UTF-16 llevan BOM: también puede declararse explícitamente UTF-16LE o UTF-16BE.

Hoy UTF-8 es la codificación recomendada para todo intercambio de información. Más del 98 % de las páginas web la utilizan. Conocer la codificación de un archivo es fundamental en programación: leer un archivo UTF-8 como si fuera Latin-1, o viceversa, produce

el clásico problema de los caracteres 'incorrectos' en los textos con tildes o caracteres especiales.

Ahora, si realmente quiere trabajar en chino instale ibus–pinyin.

```
sudo apt install ibus ibus-libpinyin
```

`ibus-setup  # agregue "Chinese - Intelligent Pinyin"` Después Ctrl+Espacio alterna entre ES y chino. Escriba zhong, aparece un menú de candidatos, elija el número. Es el método "de verdad" si va a tipear más que ocasionalmente.

1.7. Instalación de aplicaciones

La forma de administrar el software actual con repositorios de software es probablemente la mayor contribución de Ian Murdock, fundador de Debian.

Hoy computadoras, tablets, celulares, etc. usan estos mecanismos con sus almacenes de aplicaciones.

Una posible crítica es la creación de una capa extra de software entre sus creadores y los empaquetadores, vinculados a una distribución particular o grupo de ellas, con software para usarlos específicos, nombres y subpaquetes diferentes. (Manifiesto antidistros)

Existen dos formatos ampliamente usados en el universo del software libre. El formato deb, de Debian y Ubuntu y el rpm de Red Hat, usado por openSUSE y muchas otras distribuciones. Hay que tener en cuenta que no todos los paquetes que usen el mismo formato de paquete pueden ser instalados en una máquina en particular. Deben estar construidos para la misma distribución y versión, para tener compatibilidad binaria y corresponder las librerías que se requieren con la de los paquetes de los que dependen. Aquí el concepto clave es el de dependencia. Un paquete depende de otros y esto debe funcionar armónicamente. Por ello cada distribución tiene repositorios de paquetes coherentes. Esta es probablemente una de las contribuciones más importantes del mundo del software libre, al software en general. Las actualizaciones del sistema operativo y sus aplicaciones se hacen automáticamente a partir de los repositorios, mediante aplicaciones a estos fines. En Windows conviven instaladores distribuidos por cada desarrollador con mecanismos centralizados como Microsoft Store y Windows Package Manager (`winget`).

1.7.1. Repositorios y paquetería

Un **paquete** es un archivo empaquetado (`.deb`, `.rpm`, `.pkg`, etc.) que contiene binarios, librerías, archivos de configuración, documentación y *metadatos*: nombre, versión, dependencias y otros datos de control. Un **repositorio** es un servidor con un catálogo indexado de miles de paquetes al que el gestor local se suscribe.

El sistema resuelve tres problemas simultáneamente: **(a)** instalar software sin compilarlo cada vez; **(b)** resolver automáticamente cadenas de dependencias entre cientos de librerías; **(c)** verificar criptográficamente que los paquetes descargados pertenecen a repositorios autenticados y no fueron modificados por un atacante. Ian Murdock, creador de Debian, explicó que el sistema de paquetes no fue diseñado para gestionar software sino para *coordinar la colaboración* entre miles de mantenedores voluntarios distribuidos por el

mundo (Murdock 2007).

Ecosistemas principales

Cada familia de distribuciones tiene su formato y su gestor:

Familia	Formato	Bajo nivel	Alto nivel	Distros
Debian	`.deb`	`dpkg`	`apt`	Debian, Ubuntu, Mint
Red Hat	`.rpm`	`rpm`	`dnf`/`yum`	Fedora, RHEL, CentOS
Arch	`.pkg.tar.zst`	`pacman`		Arch, Manjaro
Alpine	`.apk`	`apk`		Alpine
Universal	`.flatpak`	`flatpak`		(transversal)
Universal	`.snap`	`snap`		(transversal)

La distinción **bajo/alto nivel** es importante: `dpkg` opera sobre paquetes `.deb` locales, conoce y comprueba sus relaciones de dependencia, pero no descarga automáticamente los paquetes necesarios ni realiza la resolución de alto nivel; `apt` habla con los repositorios remotos, descarga, ordena, y termina invocando `dpkg` para hacer el trabajo local. Análogamente `rpm` vs. `dnf`.

apt — operaciones cotidianas

En una Debian/Ubuntu típica alcanza con conocer estos comandos:

```
sudo apt update                          # refresca índice de repos
sudo apt upgrade                         # actualiza sin quitar
   paquetes instalados
sudo apt install paquete                 # instala + dependencias
sudo apt remove paquete                  # desinstala (conserva
   config)
sudo apt purge paquete                   # desinstala + borra
   config
sudo apt autoremove                      # elimina huérfanos de
   deps
apt search patron                        # busca por
   nombre/descripción
apt show paquete                         # metadatos, deps, tamaño
apt list --installed | wc -l             # cuántos paquetes hay
apt policy paquete                       # versiones disponibles y
   origen
sudo apt full-upgrade                    # actualiza y puede
   instalar o quitar paquetes para resolver cambios de
   dependencias
```

`apt` apareció como una interfaz interactiva que reúne operaciones de herramientas como `apt-get` y `apt-cache`, pero no son comandos idénticos ni meros sinónimos de sintaxis vieja. Para scripts se prefiere normalmente `apt-get` por su interfaz más estable; para uso interactivo suele resultar más cómodo `apt`.

dpkg — nivel local

Cuando baja un `.deb` suelto (no viene de un repo), lo maneja con `dpkg`:

```
sudo dpkg -i archivo.deb                # instala paquete local
sudo dpkg -r paquete                    # desinstala
dpkg -l                                 # lista todo lo instalado
dpkg -l | grep patron                   # filtra por patrón
dpkg -L paquete                         # lista archivos del
   paquete
dpkg -S /ruta/al/archivo                # ¿qué paquete instaló
   esto?
dpkg -s paquete                         # estado y metadatos
dpkg-deb -c archivo.deb                 # contenido sin instalar
dpkg-deb -I archivo.deb                 # metadata sin instalar
```

Si `dpkg -i` deja el sistema con dependencias rotas (típico al instalar un `.deb` suelto), se arregla con:

```
sudo apt install -f                     # fix broken deps
```

Fuentes de APT y repositorios adicionales

APT puede leer el formato tradicional de una línea en `/etc/apt/sources.list` y `/etc/apt/sources.li` pero las versiones modernas también admiten y prefieren el formato *deb822* en archivos `.sources`. Un ejemplo genérico es:

```
Types: deb
URIs: https://deb.debian.org/debian
Suites: stable
Components: main
Signed-By: /usr/share/keyrings/debian-archive-keyring.gpg
```

El formato tradicional equivalente sería una línea `deb ...`; sigue siendo ampliamente reconocido, pero conviene conocer `.sources` para configuraciones actuales y más estructuradas.

Para repositorios externos, la clave debe limitarse al repositorio correspondiente mediante `Signed-By`. Las claves agregadas manualmente pueden almacenarse, por ejemplo, en `/etc/apt/keyrings/`, en lugar de otorgarles confianza global colocándolas indiscriminadamente en el conjunto de claves confiables. La URL, suite, componentes y clave concreta deben obtenerse de la documentación del proveedor.

La firma del repositorio autentica sus metadatos; a partir de ellos APT verifica por hashes los índices y paquetes descargados.

Confianza y verificación

APT autentica criptográficamente los metadatos del repositorio, normalmente mediante archivos `InRelease` o `Release` firmados. Esos metadatos contienen hashes de los índices,

y los índices contienen a su vez hashes de los paquetes. De este modo se verifica una cadena de confianza hasta los archivos `.deb`; no es correcto describir el mecanismo como una firma independiente de cada paquete por la clave del repositorio. Esto protege contra:

- **MITM** en la descarga (aunque uses HTTP, no HTTPS, la firma detecta la modificación).
- **Compromiso del mirror** (los mirrors no tienen la clave privada, no pueden refirmar paquetes modificados).
- **Repositorios no autorizados** (agregar un repo hostil sin importar su clave da error, no instalación silenciosa).

Un repositorio sin firma se puede forzar con `apt install --allow-unauthenticated`, pero es una mala práctica salvo debug.

Paquetes universales: flatpak, snap, AppImage

Los formatos por distro (`.deb`, `.rpm`, ...) requieren mantener paquetes compatibles con distintos ecosistemas. En la última década se extendieron alternativas transversales que reducen esa dependencia, aunque requieren soporte compatible en el sistema y no garantizan funcionar literalmente en cualquier distribución:

- **Flatpak** — sandbox + runtime compartido; es ampliamente utilizado en escritorios Linux y en muchos entornos basados en GNOME. Un repositorio muy usado es Flathub.
- **Snap** — impulsado por Canonical (Ubuntu); ofrece mecanismos de confinamiento y actualizaciones automáticas administradas por el sistema Snap. Repositorio central y centralizado (snapcraft.io) — controversial.
- **AppImage** — un solo archivo ejecutable con todo dentro; no requiere instalación, no hay repositorio central. Ideal para distribuir a usuarios finales sin infraestructura.

Ventaja: reducen la necesidad de crear paquetes distintos para cada distribución y facilitan distribuir la misma aplicación en varios sistemas compatibles. Esto no elimina todos los requisitos de arquitectura, kernel, runtime o integración. Como desventaja, pueden ocupar más espacio y se actualizan por vías paralelas al gestor nativo de la distribución.

Gestores por lenguaje

Además del gestor del sistema, cada lenguaje tiene el suyo para librerías específicas de programación:

Lenguaje	Gestor
Python	`pip`, `pipx`, `uv`
JavaScript/Node	`npm`, `pnpm`, `yarn`
Rust	`cargo`
Go	`go get` / `go mod`
Perl	`cpan`, `cpanm`
Ruby	`gem`, `bundler`
Haskell	`cabal`, `stack`
LaTeX	`tlmgr`

Regla general: usar el gestor del sistema (`apt`) para aplicaciones y herramientas de propósito general (editores, browsers, compiladores, servidores); usar el gestor de lenguaje para librerías que va a importar en su propio código. Mezclarlos causa conflictos (`pip install` en `/usr/` puede pisar paquetes del sistema. En distribuciones que marcan su instalación de Python como `EXTERNALLY-MANAGED` según PEP 668, `pip` evita modificar directamente el entorno administrado por el sistema y recomienda entornos virtuales o herramientas como `pipx`).

Higiene del sistema

Con el tiempo se acumulan paquetes que ya no se usan, versiones de kernel viejas, cachés grandes. Limpieza periódica:

```
sudo apt autoremove --purge          # borra deps huérfanas
sudo apt clean                       # vacía cache de .deb
   bajados
sudo apt autoclean                   # borra solo .deb
   obsoletos
dpkg -l 'linux-image-*' | grep ^ii   # kernels instalados
apt list --upgradable                # paquetes con updates
   pendientes
```

Y para auditar qué ocupa espacio:

```
dpkg-query -Wf '${Installed-Size} ${Package}\n' \
  | sort -n | tail -20               # top 20 paquetes más
     pesados
```

1.8. Computación y sociedad

1.8.1. Código fuente y código ejecutable

El **código fuente** es el texto escrito por el programador en un lenguaje de programación legible por humanos: C, Python, Java, Scilab, etc. Es una secuencia de instrucciones que describen lo que el programa debe hacer, pero que el procesador no puede ejecutar directamente.

El **código máquina nativo** está formado por instrucciones codificadas para una arquitectura de instrucciones (ISA, *Instruction Set Architecture*) y puede ser ejecutado directamente por el procesador. Un **archivo ejecutable** es un archivo que el sistema operativo puede cargar para iniciar un programa y puede contener código máquina, datos, tablas y metadatos. El término **binario** es más amplio: un archivo objeto, una biblioteca, un ejecutable o un archivo de bytecode pueden ser binarios sin que todos ellos sean ejecutables directamente por la CPU.

Existen dos grandes modelos para realizar esa transformación:

Compilación: un programa llamado *compilador* traduce código fuente a código máquina u otras representaciones antes de la ejecución. En muchos sistemas el proceso produce archivos objeto que luego se enlazan con bibliotecas para formar un ejecutable; ese ejecutable puede depender todavía de bibliotecas dinámicas del sistema. C, C++, Fortran y Rust suelen implementarse mediante compilación a código nativo.

Interpretación y ejecución dinámica: una implementación puede analizar el programa y ejecutarlo sin producir previamente un ejecutable nativo permanente. Esto no implica necesariamente trabajar "línea por línea": Python suele compilar a bytecode y los motores JavaScript modernos combinan interpretación y compilación JIT. Por ello "compilado" e "interpretado" describen sobre todo estrategias de implementación, no categorías absolutas de lenguajes.

Algunos lenguajes combinan ambos enfoques: Java compila el código fuente a un formato intermedio llamado *bytecode*, que luego es ejecutado por una máquina virtual (JVM). Esto permite que el mismo bytecode corra en cualquier sistema que tenga la JVM instalada.

La distinción entre código fuente y ejecutable tiene consecuencias importantes para la libertad del usuario. Quien recibe solo el ejecutable binario puede usar el programa pero no puede entender cómo funciona, modificarlo ni corregir errores. Quien recibe el código fuente tiene acceso pleno al conocimiento incorporado en el programa.

1.8.2. Software libre

El software libre es aquel que respeta la libertad de los usuarios para ejecutarlo, estudiarlo, modificarlo y distribuirlo. El concepto fue formalizado por Richard M. Stallman, quien fundó el Proyecto GNU en 1983 con el objetivo de crear un sistema operativo completamente libre.

Las cuatro libertades esenciales del software libre, según la definición de la Free Software Foundation, son:

Libertad 0: libertad de ejecutar el programa para cualquier propósito, sin restricciones.

Libertad 1: libertad de estudiar cómo funciona el programa y modificarlo para adaptarlo a las propias necesidades. Para ejercer esta libertad es necesario tener acceso al código fuente.

Libertad 2: libertad de redistribuir copias del programa para ayudar a otros.

Libertad 3: libertad de distribuir copias de las versiones modificadas, de modo que la comunidad se beneficie de las mejoras. Igualmente requiere acceso al código fuente.

Es importante distinguir *software libre* de *software gratuito*. Un programa puede ser gratuito sin ser libre (no se paga pero no se puede modificar ni redistribuir), y puede ser libre sin ser gratuito (se puede cobrar por distribuirlo). La palabra inglesa *free* significa tanto "libre" como "gratuito", lo que genera confusión; por eso Stallman insiste en que "free software" se refiere a libertad, no a precio: "piensen en free speech, no en free beer".

Las licencias de software libre más importantes son:

GNU GPL (*General Public License*): la más utilizada. Garantiza las cuatro libertades y exige que cualquier obra derivada se distribuya bajo la misma licencia. Esta característica, llamada *copyleft*, garantiza que el software libre permanezca libre.

GNU LGPL: similar a la GPL pero permite que programas propietarios enlacen con bibliotecas libres sin verse afectados por el copyleft.

MIT y BSD: licencias permisivas que garantizan las cuatro libertades pero no exigen que las obras derivadas sean libres. Permiten incorporar el código en software propietario.

Las licencias **Creative Commons** forman otra familia, pensada principalmente para obras culturales, documentación, imágenes y otros contenidos, no para software. Creative Commons recomienda utilizar licencias específicas de software para programas. Además, no todas las licencias CC son licencias libres: las variantes con restricciones *NC* o *ND* limitan usos o modificaciones.

El sistema operativo GNU/Linux es el ejemplo más conocido de software libre: el kernel Linux fue creado por Linus Torvalds en 1991 y distribuido bajo GPL; combinado con las herramientas del proyecto GNU, puede formar la base de sistemas operativos completamente libres. Una distribución GNU/Linux concreta, sin embargo, puede incluir firmware, controladores o aplicaciones no libres. GNU/Linux se utiliza ampliamente en servidores y supercomputadoras.

1.8.3. Desarrollo colaborativo

El desarrollo colaborativo de software es una metodología en la que múltiples personas contribuyen al mismo proyecto, típicamente de forma distribuida y asincrónica. Las herramientas de control de versiones son el eje central de este modelo.

Un **sistema de control de versiones** (VCS, *Version Control System*) registra los cambios que se incorporan explícitamente a su historial. En Git, cada *commit* fija una instantánea y permite saber qué se registró, quién lo registró y cuándo; el mensaje del *commit* puede dejar registrado por qué se hizo. El historial permite revertir cambios y combinar el trabajo de múltiples contribuidores.

Git es el sistema de control de versiones distribuido más utilizado en la actualidad. Fue creado por Linus Torvalds en 2005 para gestionar el desarrollo del kernel Linux. En Git, cada desarrollador tiene una copia completa del historial del proyecto en su computadora. Los conceptos fundamentales son:

Repositorio: directorio que contiene el proyecto y su historial completo de cambios.

Commit: instantánea del estado del proyecto en un momento dado, acompañada de un mensaje que describe el cambio.

Rama (*branch*): línea de desarrollo independiente. Permite trabajar en una nueva funcionalidad sin afectar el código principal hasta que esté listo.

Fusión (*merge*): integración de los cambios de una rama en otra.

Pull request / merge request: mecanismo ofrecido por plataformas como GitHub, GitLab o Codeberg para proponer y revisar cambios antes de integrarlos. No es una primitiva del sistema Git en sí mismo.

GitHub, **GitLab** y **Codeberg** son plataformas web que alojan repositorios Git y añaden herramientas de colaboración: seguimiento de errores (*issues*), revisión de código, integración continua y documentación.

El modelo de desarrollo colaborativo abierto, popularizado por proyectos como el kernel Linux, Mozilla Firefox y LibreOffice, ha demostrado que equipos distribuidos de voluntarios pueden producir software de calidad comparable o superior al desarrollado por equipos comerciales. Eric Raymond describió este fenómeno en su ensayo *La catedral y el bazar* (1997): mientras el software propietario se desarrolla en secreto como una catedral, el software libre se construye en la plaza pública como un bazar caótico pero productivo.

1.8.4. Ciencia abierta

La ciencia abierta (*open science*) es un movimiento que propone aplicar a la investigación científica los mismos principios de transparencia y colaboración que el software libre aplica al desarrollo de programas. Otra rama del conocimiento libre. Sus pilares son:

Acceso abierto (*open access*): publicación de artículos científicos en repositorios de libre acceso, sin barreras de pago para los lectores. Iniciativas como arXiv (física, matemáticas, computación) y PubMed Central (biomedicina) alojan millones de artículos de libre acceso.

Datos abiertos (*open data*): publicación de los conjuntos de datos utilizados en la investigación, con documentación suficiente para que otros puedan reutilizarlos y verificar los resultados.

Código abierto en investigación: publicación de los programas y scripts utilizados para el análisis de datos, de modo que los experimentos computacionales sean reproducibles.

Cuadernos de laboratorio abiertos: registro público del proceso de investigación en tiempo real, no solo de los resultados finales.

La reproducibilidad es un valor central de la ciencia. Un resultado científico que no puede ser reproducido por otros investigadores no es confiable. La publicación del código y los datos es una condición necesaria para la reproducibilidad computacional: si el análisis se realizó con un script que no se publica, nadie puede verificar que el script hace lo que el artículo describe.

1.8.5. Transparencia y oscuridad en el software

La dicotomía entre transparencia y oscuridad atraviesa el debate sobre el software libre y tiene implicaciones que van más allá de la informática.

El **software libre** hace posible la inspección y auditoría del código fuente por personas con los conocimientos necesarios. Esto permite estudiar su comportamiento, buscar errores y proponer correcciones. Esta posibilidad de auditoría es una forma de rendición de cuentas.

El **software propietario** es opaco: el fabricante distribuye únicamente el binario ejecutable. El usuario no puede saber qué hace el programa con sus datos, si registra su actividad, si contiene vulnerabilidades de seguridad o si implementa comportamientos no declarados. Debe confiar en las afirmaciones del fabricante.

Esta diferencia tiene consecuencias prácticas importantes:

Seguridad: la seguridad por oscuridad (*security through obscurity*) es un principio débil. Ocultar el código no impide que los atacantes encuentren vulnerabilidades; simplemente dificulta que los usuarios y la comunidad de seguridad las encuentren y corrijan. El software de código abierto permite que miles de expertos revisen el código buscando errores.

Privacidad: con software opaco es imposible saber si un programa recopila y transmite datos personales del usuario. Con software libre, esa conducta sería visible en el código fuente.

Soberanía tecnológica: los gobiernos y las instituciones que usan software propietario dependen de los fabricantes para obtener actualizaciones y correcciones, y no pueden adaptar el software a sus necesidades específicas. El software libre les permite mantener control sobre su infraestructura tecnológica.

Educación: el software libre permite que estudiantes e investigadores estudien cómo están implementados los algoritmos reales, no solo las descripciones teóricas.

El principio de Kerckhoffs, formulado en criptografía en 1883, establece que la seguridad de un sistema no debe depender del secreto de su diseño, sino únicamente del secreto de las claves. Este principio, hoy universalmente aceptado en seguridad informática, es una aplicación específica de la idea más general de que la transparencia es más robusta que la oscuridad.

1.9. Referencias

1.10. Ejercicios

1.10.1. Instrucciones generales para los prácticos de la materia:

Se recomienda ejecutar cada instrucción de manera individual y analizar cuidadosamente el resultado obtenido antes de continuar con el siguiente ejercicio. En aquellos casos en que

se solicite una explicación, la respuesta deberá fundamentarse a partir de la observación del comportamiento del programa.

No se pretende memorizar comandos, sino comprender la forma de trabajo de cada software y adquirir soltura en el uso de sus herramientas básicas.

Para entregar los trabajos practicos o parciales de la materia vaya copiando lo que muestre la terminal en su consola, si corresponde, en un archivo de texto plano (`.txt`) o un documento `.odt`.

Al comienzo del archivo indique fecha, su nombre, DNI y número de práctico. Antes de cada problema indique Problema NÚMERO:

Use masivamente la ayuda de cada software, las teorías comprendidas en el práctico, otro material de la bibliografía, consulte internet y las inteligencias artificiales de las que disponga. Consulte con sus compañeros y profesores. Use adicionalmente el grupo de la materia en Whatsapp, para sus consultas.

Los ejercicios de terminal en adelante deben realizarse en GNU/Linux, en la distribución que prefiera. Los ejercicios de paquetes están formulados en forma general; las notas al final de cada uno dan los comandos concretos para Debian (y derivados como Ubuntu), Fedora, Arch, openSUSE y Gentoo. Puede usar una instalación nativa, una máquina virtual o WSL2 (ver ejercicio 2).

1. **Ejercicio:** Placa madre

   Observando la foto de la placa madre, identifique y describa brevemente la función de:

   - Socket de CPU (¿qué familia de procesadores acepta?)
   - Ranuras de memoria RAM (¿cuántas? ¿qué tecnología?)
   - Ranuras de expansión PCIe (¿para qué se usan?)
   - Ranuras M.2 (¿qué se conecta ahí?)
   - Puertos SATA (¿qué se conecta ahí?)
   - Chipset (¿qué rol cumple?)

   Consulte la serigrafía impresa sobre la placa para identificar modelo y fabricante. Escriba en cada cuadro vacio de la figura el nombre de cada componente señalado.

2. **Ejercicio:** Instalar un segundo SO

   Instale un segundo sistema operativo junto al que ya usa.

   Use **Dual-boot real**: GNU/Linux junto a Windows en el mismo disco.

   Advertencia: haga backup previo, el particionado puede destruir datos. Alternativamente podría usar una **Máquina virtual**: instale VirtualBox o VMware y monte el segundo OS ahí. Más seguro, no toca el disco físico. Pero esta vía no le permitirá aprender varios aspectos importantes.

   En Windows instale además una terminal POSIX; hoy la vía recomendada es **WSL2** (`wsl -install` desde PowerShell como administrador).

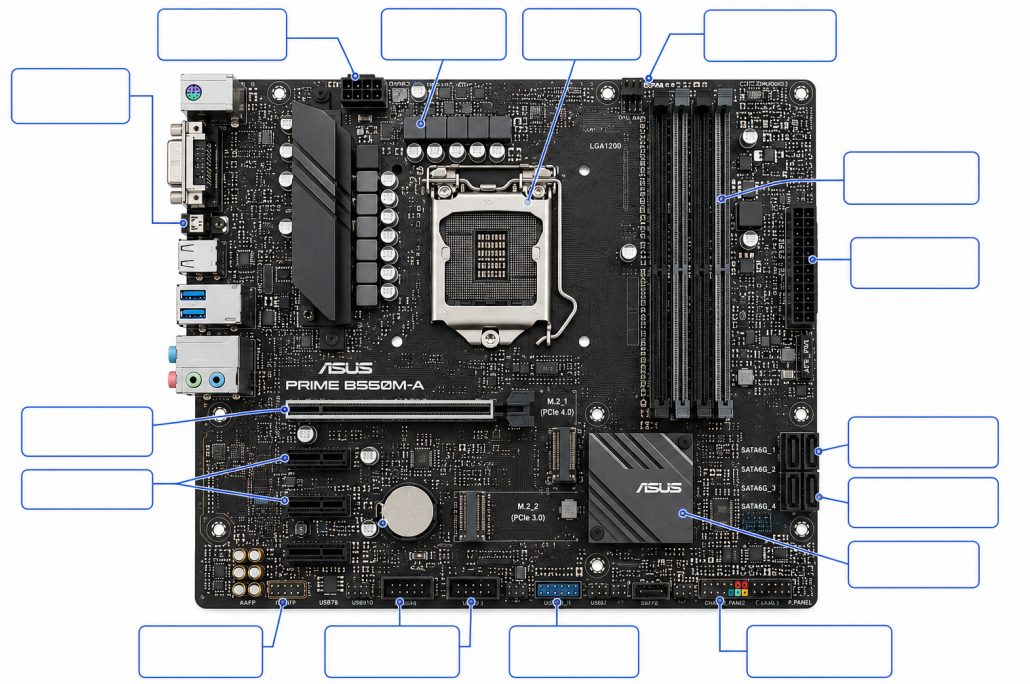


Figura 1.10: Placa madre: componentes principales a identificar.

3. **Ejercicio:** Terminal

   Abra una terminal e investigue los siguientes comandos ejecutando cada uno y observando su salida. Para cada uno escriba una línea describiendo qué hace: `pwd`, `whoami`, `uname -a`, `df -h`, `free -h`, `uptime`.

4. **Ejercicio:** Man

   I. Lea la sección teórica correspondiente a `man`. Anote tres comandos que descubrió y no conocía.

   II. Abra la página de manual de `ls`:

   ```
   man ls
   ```

   III. Explique la diferencia entre usar `man 1 passwd` y `man 5 passwd`.

   IV. Localice qué hacen las opciones de man `-a`, `-l`, `-h`, `-R`, `-S`.

   V. Consulte `man man`. ¿Qué comando lee?

5. **Ejercicio:** Procesos

   Muestre los diez procesos que más memoria RAM están consumiendo en este momento. ¿Qué comando utilizó?

6. **Ejercicio:** ls, grep, wc

   Liste el contenido del directorio `/etc` mostrando permisos, propietario y tamaño de cada archivo. ¿Cuántos archivos contiene? Use `ls` con las opciones adecuadas y `wc -l` para contar.

Bash. `ls /etc` para contar archivos: `ls | wc -l` no cuenta ocultos ni maneja bien nombres raros. Explicar `-A` o usar `find /etc -maxdepth 1 | wc -l`. También: `grep -r -l root /etc 2>/dev/null >resultados.txt`.

7. **Ejercicio:** Permisos

   Los archivos Unix tienen tres grupos de permisos (dueño, grupo, otros) y tres acciones (leer, escribir, ejecutar).

   I. Cree un archivo `secreto.txt` con `echo hola >secreto.txt`.

   II. Vea sus permisos con `ls -l`. ¿Cuáles son por defecto?

   III. Quítele el permiso de lectura al grupo y a otros: `chmod go-r secreto.txt`.

   IV. Intente leerlo desde otra cuenta o con `sudo -u nobody cat secreto.txt`.

   Bash. Si su directorio personal no es legible por otros (default en Ubuntu recientes), pruebe en `/tmp`.

   V. Devuelva los permisos originales.

8. **Ejercicio:** Redirección

   En la terminal, los símbolos `>>` `<` y `2>` redirigen los flujos estándar (stdout, stderr, stdin), abiertos en casi toda aplicación pensada para la consola.

   Ejecute y explique la diferencia:

```
ls /etc > lista.txt           # sobrescribe
ls /home >> lista.txt         # agrega al final
ls /noexiste 2> errores.txt   # solo stderr
ls /noexiste > /dev/null 2>&1  # descarta todo
wc -l < lista.txt             # entrada desde archivo
```

9. **Ejercicio:** Directorios

   Usando únicamente comandos de terminal, cree la siguiente estructura de directorios en su carpeta personal:

```
practico/
    ejercicio1/
    ejercicio2/
    datos/
        entrada/
        salida/
```

   Verifique el resultado con `tree practico` o con `ls -R practico`.

10. **Ejercicio:** Tuberías (pipes)

    Ejecute el siguiente comando y explique qué hace cada parte:

```
ls /usr/bin | grep "^py" | wc -l
```

¿Qué representa el símbolo `|` (tubería o *pipe*)? ¿Qué significa `^py` en este contexto?

11. **Ejercicio:** Tuberías 2

    El comando `grep` permite buscar texto dentro de archivos. Busque en el directorio `/etc` todos los archivos que contengan la palabra `root`. Use la opción `-r` para búsqueda recursiva y `-l` para mostrar solo los nombres de los archivos. Redirija la salida a un archivo llamado `resultados.txt`.

12. **Ejercicio:** find

    El comando `find` permite buscar archivos por distintos criterios. Realice las siguientes búsquedas y anote el comando utilizado en cada caso:

    I. Todos los archivos con extensión `.pdf` en `/usr/share/doc`.
    II. Todos los directorios dentro de su carpeta personal.
    III. Todos los archivos modificados en las últimas 24 horas en `/var/log`.
    IV. Todos los archivos mayores de 10 MB en `/usr`.
    V. Archivos cuyo nombre contenga la palabra `readme` (sin distinguir mayúsculas de minúsculas).

13. **Ejercicio:** find y grep

    Combine `find` con `grep` mediante una tubería para buscar archivos de texto que contengan la palabra `copyright` dentro del directorio `/usr/share/doc`. ¿Cuántos archivos encontró?

14. **Ejercicio:** Encodings: Ascii y unicode

    El texto ASCII de 7 bits puede representar 128 caracteres. ¿Puede representar la letra ñ? ¿Y la á? ¿Y el carácter chino 中?

    Explique por qué y qué estándar resuelve esta limitación.

    Verifique si su terminal muestra correctamente caracteres ascii, latin-1, UTF-8. Verifique lo mismo para un editor de texto de su preferencia.

15. **Ejercicio:** Encodings 2

    Un archivo de texto contiene la palabra `hola`. Indique cuántos bytes ocupa si está codificado en:

    I. ASCII puro.
    II. UTF-8.
    III. UTF-16.

    Repita el análisis para la palabra `niño`. ¿Cambia el tamaño? ¿Por qué?

16. **Ejercicio:** Encodings 3

    Ejecute en la terminal y anote los bytes hexadecimales de cada carácter:

```
echo -n "A" | xxd

echo -n "ñ" | xxd
```

`echo -n "中" | xxd`

Puede intentar copiar y pegar el caracter chino desde este pdf. o ejecutar

`printf '\u4e2d\n'`

Verifique que coinciden con la codificación UTF-8 explicada en la teoría: 1 byte para ASCII, 2 bytes para caracteres latinos extendidos, 3 bytes para caracteres del plano básico multilingüe.

17. **Ejercicio:** Montar pendrives USB

    Conecte el pendrive USB y detecte qué nombre le asigna el sistema (por ejemplo con `lsblk` o mirando la salida de `dmesg | tail`). Móntelo en el árbol de directorios de su PC y liste su contenido. ¿Qué sistema de archivos utiliza? ¿Puede escribir en él? ¿Por qué?

```
lsblk
sudo mkdir -p /mnt/usb
sudo mount /dev/sdb1 /mnt/usb
ls -lh /mnt/usb
df -T /mnt/usb
```

    Copie tres archivos del USB a su directorio `practico/datos/entrada/`.

    Opcionalmente, extraiga una imagen del pendrive completo:

```
sudo umount /mnt/usb
sudo dd if=/dev/sdb of=pendrive.img bs=4M status=progress
```

18. **Ejercicio:** tar

    El comando `tar` (*tape archive*) empaqueta varios archivos en uno solo sin comprimir. Cree un archivo `practico.tar` que contenga todo el directorio `practico/` creado en el ejercicio anterior:

```
tar -cvf practico.tar practico/
```

    I. ¿Qué significa cada letra de la opción `-cvf`?

    II. Liste el contenido del `.tar` sin desempaquetarlo.

    III. ¿Es el archivo `.tar` más pequeño, igual o más grande que la suma de los archivos originales? ¿Por qué?

19. **Ejercicio:** gz

    Comprima el archivo `practico.tar` usando `gzip`:

```
gzip practico.tar
```

I. ¿Qué extensión tiene el archivo resultante?

II. ¿Desapareció el `practico.tar` original? ¿Por qué?

III. Compare el tamaño del `.tar.gz` con el del `.tar`. ¿Cuánto espacio ahorró la compresión?

IV. Descomprima el archivo con `gunzip` y verifique que el contenido es idéntico al original.

Bash. `md5sum` es una utilidad de línea de comandos que genera una huella digital única de 128 bits (tipo MD5), para cualquier archivo. Se usa comúnmente para verificar la integridad de los datos y asegurar que un archivo no haya sido alterado durante una descarga o transferencia. Hay alternativas como `sha256sum`, `xxhsum` y `b2sum`. Estas opciones ofrecen mayor seguridad criptográfica o velocidad superior al calcular hashes de archivos.

20. **Ejercicio:** xz

El comando `tar` soporta varios compresores:

```
tar -czf practico.tar.gz  practico/    # gzip  (rápido,
   ratio medio)
tar -cjf practico.tar.bz2 practico/    # bzip2 (lento,
   buen ratio)
tar -cJf practico.tar.xz  practico/    # xz    (lento,
   mejor ratio)
```

Compare los tres tamaños con `ls -lh`. ¿Cuál comprime más? ¿Cuál tarda más? Mida los tiempos con `time tar ...`. En sistemas modernos, ¿por qué xz suele preferirse sobre bzip2 (hoy obsoleto)?

21. **Ejercicio:** Paquetes y repositorios

Practique el ciclo completo de instalación, actualización y remoción de paquetes con el gestor de paquetes de su distribución. Ejecute los pasos en orden y anote el resultado o el mensaje de error de cada uno. Los comandos concretos de cada distribución están en las notas al final del ejercicio.

I. **Refrescar el índice de repositorios.** Antes de instalar nada, sincronice la lista local de paquetes con los servidores. ¿Cuántos paquetes aparecen con actualizaciones pendientes?

II. **Buscar antes de instalar.** Vamos a instalar `tree` (un utilitario chico que dibuja árboles de directorios). Primero búsquelo e inspecciónelo sin instalarlo. Anote tamaño instalado, versión disponible y quién lo mantiene.

III. **Instalar.** Antes de confirmar, observe qué otros paquetes se van a instalar como dependencia (si es que hay). Después de instalar, verifique la ruta del ejecutable (`which tree`), la versión (`tree -version`), que funciona (`tree /etc`), y liste los archivos que el paquete trajo.

IV. **¿A qué paquete pertenece un archivo?** Consulta inversa: pregunte al gestor qué paquete instaló `/usr/bin/tree`. Debería devolver `tree`. Pruebe también con `/etc/hostname`, `/bin/ls` y `/usr/bin/python3` para ver qué paquetes trajeron esos archivos.

V. **Instalar un segundo paquete con dependencias.** Elija `cowsay` (una vaca ASCII que dice frases) o `figlet` (arte ASCII con texto grande). Al instalarlos observe qué dependencias arrastran. Pruébelos:

```
cowsay "Hola mundo"
figlet Linux
```

VI. **Actualizar todo el sistema.** Aplique las actualizaciones pendientes que vio en (i). Anote el número de paquetes actualizados. ¿Su gestor distingue entre actualizaciones simples y actualizaciones que requieren agregar o quitar paquetes?

VII. **Ver versiones disponibles.** Elija un paquete instalado y consulte qué versiones existen y de dónde vienen. ¿Ve varias versiones? ¿De qué repositorio viene la instalada?

VIII. **Remover.** Quite `tree` y `cowsay`, y elimine las dependencias huérfanas que quedaron. Verifique con `which tree` y `which cowsay` que ya no están. ¿Su gestor distingue entre borrar solo los binarios y borrar también la configuración? Pista: buscar en `/etc/`.

IX. **Limpieza del cache.** El gestor guarda los paquetes descargados en un directorio de cache que con el tiempo crece varios GB. Averigüe dónde está, mida cuánto ocupa con `du -sh`, límpielo y mida de nuevo. ¿Cuánto espacio recuperó?

X. **Instalar un paquete suelto.** Baje del sitio web de su distribución un paquete cualquiera en el formato nativo (`.deb`, `.rpm`, etc.) e instálelo directamente desde el archivo, sin pasar por el gestor de repositorios. ¿Qué pasa con las dependencias?

Debian. El gestor de alto nivel es `apt` (resuelve dependencias contra los repositorios) y el de bajo nivel es `dpkg` (instala y consulta paquetes `.deb` sueltos). El ciclo completo, en el orden de los pasos:

```
sudo apt update                    # i. refrescar índice
apt search tree                    # ii. buscar
apt show tree                      #     inspeccionar
   (campo Maintainer)
sudo apt install tree              # iii. instalar
dpkg -L tree                       #     archivos que trajo
dpkg -S /usr/bin/tree              # iv. ¿a qué paquete
   pertenece?
sudo apt install cowsay figlet     # v. segundo paquete con
   deps
sudo apt upgrade                   # vi. actualizar todo
sudo apt full-upgrade              #     idem, permitiendo
   agregar/quitar paquetes
apt policy tree                    # vii. versiones y
   repositorio de origen
```

```
apt policy python3
sudo apt remove tree                # viii. borra binarios,
   deja config
sudo apt purge cowsay               #      borra todo,
   incluyendo config
sudo apt autoremove                 #      elimina deps
   huérfanas
du -sh /var/cache/apt/archives/     # ix. el cache está ahí
sudo apt clean                      #      limpiarlo
sudo dpkg -i archivo.deb            # x. paquete suelto
   (packages.debian.org)
sudo apt install -f                 #      corregir deps
   faltantes tras dpkg -i
```

¿Cuál es la diferencia entre `upgrade` y `full-upgrade`? ¿Qué queda de `tree` después de `remove` que no queda después de `purge`? Explique por qué `dpkg` solo no resuelve dependencias y `apt install -f` sí.

Fedora. El gestor es `dnf` y el formato de paquete es `.rpm`; las consultas a paquetes instalados se hacen con `rpm`. No hay separación instalar/purgar: `remove` borra también la configuración. Equivalentes de cada paso:

```
sudo dnf check-update               # refrescar índice y ver
   actualizables
dnf search tree                     # buscar
dnf info tree                       # inspeccionar
sudo dnf install tree               # instalar
rpm -ql tree                        # archivos que trajo
rpm -qf /usr/bin/tree               # ¿a qué paquete
   pertenece?
sudo dnf upgrade                    # actualizar todo
dnf list --showduplicates tree      # versiones disponibles
sudo dnf remove tree                # remover
sudo dnf autoremove                 # deps huérfanas
sudo dnf clean all                  # limpiar cache
sudo dnf install ./archivo.rpm      # paquete suelto,
   resuelve deps
```

Arch. El gestor es `pacman` y el formato es `.pkg.tar.zst`. Importante: nunca use `-Sy` solo para instalar (actualización parcial); Arch es *rolling release* y lo correcto es `-Syu`. Equivalentes:

```
sudo pacman -Syu                    # refrescar índice y
   actualizar todo
pacman -Ss tree                     # buscar
pacman -Si tree                     # inspeccionar
sudo pacman -S tree                 # instalar
pacman -Ql tree                     # archivos que trajo
pacman -Qo /usr/bin/tree            # ¿a qué paquete
   pertenece?
sudo pacman -R tree                 # remover
```

```
sudo pacman -Rns cowsay          # remover con config y
   deps
sudo pacman -Sc                  # limpiar cache
sudo pacman -U archivo.pkg.tar.zst  # paquete suelto
```

openSUSE. El gestor es `zypper`, también sobre `.rpm`, así que las consultas con `rpm` son iguales que en Fedora. Equivalentes:

```
sudo zypper refresh              # refrescar índice
zypper search tree               # buscar
zypper info tree                 # inspeccionar
sudo zypper install tree         # instalar
rpm -ql tree                     # archivos que trajo
rpm -qf /usr/bin/tree            # ¿a qué paquete
   pertenece?
sudo zypper update               # actualizar todo
sudo zypper remove tree          # remover
sudo zypper clean                # limpiar cache
sudo zypper install ./archivo.rpm  # paquete suelto
```

Gentoo. No hay paquetes binarios precompilados por defecto: `emerge` baja las fuentes y compila en el momento según la receta (*ebuild*) del árbol de Portage. Las consultas usan `equery` (paquete `gentoolkit`). Equivalentes:

```
sudo emerge --sync               # refrescar el árbol de
   Portage
emerge --search tree             # buscar
equery meta app-text/tree        # inspeccionar
sudo emerge app-text/tree        # instalar (compila)
equery files tree                # archivos que trajo
equery belongs /usr/bin/tree     # ¿a qué paquete
   pertenece?
sudo emerge -avuDN @world        # actualizar todo
sudo emerge --deselect tree      # remover (y luego:)
sudo emerge --depclean           # deps huérfanas
sudo eclean distfiles            # limpiar cache de fuentes
```

22. **Ejercicio:** Recompilar un paquete desde las fuentes del repositorio

    Bajar el código fuente de un paquete tal como lo empaqueta su distribución (no las fuentes del sitio original del software), recompilarlo desde cero, y reinstalar el paquete resultante. Este ejercicio muestra por qué un sistema binario no es suficiente: para *construir* un paquete hace falta un ecosistema entero de herramientas de desarrollo que en instalaciones típicas no vienen. Los comandos concretos de cada distribución están en las notas al final del ejercicio.

    1. **Verificar que las herramientas de desarrollo no están.** En un sistema recién instalado:

    ```
    which gcc make
    ```

Probablemente devuelvan nada. **Ese es el punto**: un usuario final tiene el compilador *ausente* por defecto — su sistema puede correr programas compilados pero no compilarlos.

II. **Instalar las herramientas de desarrollo** y de empaquetado de su distribución. Verifique que ahora `gcc` y `make` aparecen, con `which` y `gcc -version`.

III. **Habilitar el acceso al código fuente**, si su distribución lo requiere: los repositorios binarios traen paquetes listos para instalar; los de fuentes traen lo necesario para reconstruirlos.

IV. **Bajar el código fuente de un paquete.** Elegimos `hello` — el programa GNU minimalista pensado justamente para enseñar empaquetado. Trabaje en un directorio nuevo e identifique en lo descargado dónde está el código original del autor (*upstream*) y dónde los cambios y metadatos propios de la distribución.

V. **Instalar las dependencias de compilación.** Un paquete puede necesitar librerías o herramientas específicas para *compilar* que no son necesarias para *ejecutar*. Por ejemplo, para compilar necesita `gettext` o `autoconf`, pero para correr el binario final no. Su gestor las resuelve automáticamente. ¿Qué paquetes agregó?

VI. **Compilar** con la herramienta de empaquetado de su distribución.

VII. **Ver los artefactos generados.** ¿Qué archivos aparecieron y qué contiene cada uno?

VIII. **Inspeccionar el paquete recompilado.** Antes de instalarlo, vea su metadata y la lista de archivos que instalará.

IX. **Instalar el paquete recompilado.** Reemplaza al que vino del repositorio. Verifique que `hello` y `hello -help` funcionan.

X. **Modificar y rebuildear (opcional, avanzado).** Edite `src/hello.c` cambiando el saludo, incremente la versión del paquete según la convención de su distribución, recompile, reinstale y verifique el nuevo saludo.

Preguntas de discusión.

- ¿Por qué la instalación mínima de una distribución no incluye el compilador por defecto?
- ¿Qué diferencia hay entre las dependencias de ejecución (*Depends*) y las de compilación (*Build-Depends*)? Dé un ejemplo de cada tipo.
- Si distribuye su paquete modificado a otro usuario, ¿cómo puede el destinatario verificar que no fue alterado en el camino? Pista: firmas GPG.
- ¿Qué gana la distribución al mantener el código del autor intacto y separar sus parches en un archivo aparte?

Una distribución sin herramientas de desarrollo es una distribución amputada. Esto conecta con la discusión de software libre (se necesita el fuente *y* las herramientas, no solo el fuente) y con licencias: una GPL sin compilador es libertad teórica.

Debian. Las herramientas mínimas están agrupadas en el meta-paquete `build-essential` (`gcc`, `make`, `libc-dev`, `dpkg-dev`); `devscripts` trae `debuild`,

`debchange` y otros helpers; `fakeroot` simula permisos de root para el empaquetado sin necesitarlos realmente:

```
which gcc make dpkg-buildpackage debuild fakeroot      # i. ¿están?
sudo apt install build-essential devscripts fakeroot   # ii. instalarlas
```

Para (iii), los repositorios de fuentes son las líneas `deb-src`. En Debian 12 o posterior edite `/etc/apt/sources.list.d/debian.sources` y agregue `deb-src` en la línea `Types:` (queda `Types: deb deb-src`); en el formato viejo, descomente las líneas `deb-src` de `/etc/apt/sources.list`. Luego `sudo apt update`.

```
mkdir -p ~/build && cd ~/build
apt source hello                    # iv. bajar el fuente
ls -la
```

Debería ver tres archivos y un directorio: `hello_X.Y.orig.tar.gz` — código upstream original; `hello_X.Y-N.debian.tar.xz` — parches y metadata específicos de Debian; `hello_X.Y-N.dsc` — descriptor del paquete fuente, firmado; y `hello-X.Y/` — árbol desempacado, listo para compilar.

```
sudo apt build-dep hello            # v. deps de compilación
cd hello-*/
dpkg-buildpackage -us -uc -b        # vi. compilar
```

Las flags: `-us` no firmar el fuente, `-uc` no firmar el changelog, `-b` construir solo binarios (sin regenerar el `.dsc`). El proceso corre `./configure`, `make` y `make install` dentro de un entorno `fakeroot` que arma la estructura de directorios del paquete. El resultado aparece *en el directorio padre* (convención Debian):

```
cd ..
ls -la *.deb *.changes *.buildinfo    # vii. artefactos
dpkg-deb -I hello_*_amd64.deb         # viii. metadata
dpkg-deb -c hello_*_amd64.deb         #       archivos que instalará
sudo dpkg -i hello_*_amd64.deb        # ix. instalar
hello
hello --help
```

El `.changes` es el resumen firmable de lo construido (lo que se subiría a un repositorio, con `debsign`) y el `.buildinfo` la metadata reproducible (versiones exactas de todas las deps de compilación). Para (x):

```
cd hello-*/
export DEBFULLNAME="Su Nombre" DEBEMAIL="su@email"
dch -i "Mensaje modificado por el estudiante"
dpkg-buildpackage -us -uc -b
sudo dpkg -i ../hello_*_amd64.deb
hello
```

`dch -i` incrementa la versión Debian (`X.Y-N` → `X.Y-N+1`) y agrega una entrada

al `debian/changelog`. Es la manera formal de "yo hice cambios sobre el paquete oficial".

Fedora. El equivalente del fuente Debian es el `.src.rpm`, y la receta de construcción es el archivo `.spec`. Los artefactos quedan bajo `~/rpmbuild/`:

```
sudo dnf install rpm-build rpmdevtools    # herramientas
dnf download --source hello               # baja
   hello-X.Y.src.rpm
sudo dnf builddep hello                   # deps de
   compilación
rpmbuild --rebuild hello-*.src.rpm        # compila
ls ~/rpmbuild/RPMS/x86_64/
sudo dnf install ~/rpmbuild/RPMS/x86_64/hello-*.rpm
```

Arch. Las recetas son archivos `PKGBUILD` (scripts de shell legibles, buen material para inspeccionar). Se obtienen del repositorio de empaquetado de Arch con `pkgctl` y se compilan con `makepkg`:

```
sudo pacman -S base-devel devtools        # herramientas
pkgctl repo clone --protocol=https hello
cd hello
makepkg -s                                # -s instala deps
   de compilación
sudo pacman -U hello-*.pkg.tar.zst
```

openSUSE. Igual que Fedora (`.src.rpm` y `rpmbuild`), con `zypper` para las dependencias:

```
sudo zypper install rpm-build
zypper source-install -d hello            # deps de
   compilación
zypper source-install hello               # baja el fuente
rpmbuild -bb /usr/src/packages/SPECS/hello.spec
sudo zypper install
   /usr/src/packages/RPMS/x86_64/hello-*.rpm
```

Gentoo. Este ejercicio pierde sentido como excepción: en Gentoo compilar desde fuentes *es* el mecanismo normal de instalación. Cada `emerge` hace exactamente lo que este ejercicio pide a mano. Para ver los pasos separados de una receta:

```
equery which sys-devel/gcc                # ubicar el ebuild
sudo ebuild $(equery which app-misc/hello) fetch unpack
   compile
ls /var/tmp/portage/                      # árbol de trabajo
```

23. **Ejercicio:** Licencias

Clasifique las siguientes licencias de software indicando si son libres o propietarias, y si son copyleft o permisivas. Justifique brevemente cada respuesta: GNU GPL, MIT, BSD, software de Windows, macOS, LibreOffice, Google Chrome, Chromium.

Nota: Chromium es BSD-3, Chrome es propietaria pero deriva de Chromium — buena pregunta trampa.

LibreOffice es MPL-2.0 (copyleft débil), no pura GPL — vale la pena distinguir çopyleft fuerte / débil / permisivo".

24. **Ejercicio:** Licencias y paquetes

    Baje un paquete de software de algún repositorio o directamente del sitio del paquete, inspecciónelo, e indique que licencia tiene.

    Debian. `apt show <paquete>` y `/usr/share/doc/<paquete>/copyright` es donde se lee la licencia.

    Fedora. `dnf info <paquete>` o `rpm -qi <paquete>` muestran el campo `License`.

    Arch. `pacman -Si <paquete>` muestra el campo `Licenses`.

    openSUSE. `zypper info <paquete>` o `rpm -qi <paquete>`, campo `License`.

    Gentoo. `equery meta <paquete>` o la variable `LICENSE` del ebuild.

25. **Ejercicio:** Software y diferentes licencias

    Un programa distribuido bajo licencia MIT puede ser incorporado en un producto propietario sin publicar el código fuente resultante. ¿Puede hacerse lo mismo con un programa bajo GNU GPL? Explique la diferencia entre ambas licencias en este aspecto.

26. **Ejercicio:** 4 libertades

    Según las cuatro libertades del software libre definidas por la Free Software Foundation, ¿que distingue al software libre del software gratuito? ¿Por qué es necesario el acceso al código fuente para ejercerlas?

1.11. Laboratorio

Laboratorio: Armado y desarmado de computadoras

Preparación:

1. **Atención trabaje sin energía** y cuidado con condensadores cargados.
2. Use herramientas de electricista, especialmente destornilladores imantados con aislante. Habitualmente tendrá tornillos Philips (punta en cruz). Puede necesitar de varios tamaños.
3. Use zapatillas con suela plástica o de goma.
4. Tenga recipientes para los tornillos y recuerde donde iban.

5. Tenga superficies y recipientes adecuados para poner las piezas.
6. En climas secos puede necesitar bandas conductoras a tierra.

Ejecución:

1. Pruebe que la computadora que recibe está funcionando. Esto sirve para que cuando la vuelva a armar, si no funciona, sepa que antes funcionaba.
2. Saque la tapa de la PC.
3. Inspeccione visualmente, fije una estrategia para el desarmado.
4. Desarme la PC, desmonte todos los componentes del motherboard
5. Vuelva a armar.
6. Verifique que todo funciona correctamente.

Capítulo 2

Computadoras y personas en red

2.1. Redes e Internet

2.1.1. Protocolos y el modelo TCP/IP

Un **protocolo** es un conjunto de reglas que define cómo dos o más entidades se comunican: qué formato tienen los mensajes, en qué orden se envían, cómo se detectan y corrigen errores y cómo se establece y termina una conexión. En redes de computadoras, los protocolos son el equivalente a los idiomas: dos sistemas solo pueden comunicarse si hablan el mismo protocolo.

La arquitectura de Internet se organiza en capas. Cada capa resuelve un problema específico y ofrece servicios a la capa superior, ocultando los detalles de implementación. El modelo utilizado en Internet se llama **TCP/IP** y tiene cuatro capas:

Capa de acceso a la red: se ocupa de transmitir bits a través del medio físico: cable de cobre, fibra óptica o radio. Incluye protocolos como Ethernet (redes por cable) y WiFi (802.11). Define cómo se codifican los bits en señales eléctricas, luminosas o electromagnéticas, y cómo se accede al medio compartido sin colisiones.

Capa de Internet: se ocupa del direccionamiento y el encaminamiento de paquetes entre redes distintas. El protocolo fundamental es **IP** (*Internet Protocol*). Cada paquete IP contiene la dirección de origen y la de destino; los routers leen esas direcciones y deciden por dónde reenviar el paquete para que llegue a destino.

Capa de transporte: proporciona comunicación extremo a extremo entre aplicaciones. Los dos protocolos principales son:

- **TCP** (*Transmission Control Protocol*): orientado a conexión, garantiza que los datos lleguen en orden y sin errores. Si un paquete se pierde, lo retransmite. Adecuado para transferencia de archivos, correo, navegación web.
- **UDP** (*User Datagram Protocol*): no orientado a conexión, no garantiza entrega ni orden. Es más rápido y adecuado para aplicaciones donde la latencia importa más que la fiabilidad: videoconferencia, streaming, videojuegos.

Capa de aplicación: donde viven los protocolos que usan directamente los usuarios: HTTP (web), SMTP (correo), DNS (nombres de dominio), SSH (terminal remota), FTP (transferencia de archivos).

La comunicación entre dos aplicaciones funciona así: la aplicación emisora pasa sus datos a la capa de transporte, que los divide en segmentos y añade información de control. La capa de Internet encapsula cada segmento en un paquete IP con las direcciones de origen y destino. La capa de acceso a la red encapsula el paquete en una trama y la transmite por el medio físico. En el receptor, el proceso se invierte: cada capa elimina su cabecera y pasa los datos a la capa superior hasta que la aplicación recibe el mensaje original. Este proceso se llama *encapsulamiento*.

2.1.2. Puertos y paquetes

Una dirección IP identifica una computadora en la red, pero una computadora puede estar ejecutando simultáneamente un servidor web, un servidor de correo y una sesión SSH. Los **puertos** permiten distinguir a qué aplicación va destinado cada paquete.

Un puerto es un número de 16 bits (del 0 al 65535) que identifica un proceso o servicio dentro de una computadora. Cuando un cliente se conecta a un servidor, especifica tanto la dirección IP del servidor como el puerto del servicio. Los puertos del 0 al 1023 están reservados para servicios estándar:

Puerto	Protocolo	Servicio
20, 21	FTP	Transferencia de archivos
22	SSH	Terminal segura
25	SMTP	Envío de correo
53	DNS	Resolución de nombres
80	HTTP	Web sin cifrar
110	POP3	Recepción de correo
143	IMAP	Correo (acceso remoto)
443	HTTPS	Web cifrada

Un **paquete IP** tiene dos partes: la *cabecera* (*header*) y los *datos* (*payload*). La cabecera contiene, entre otros campos: versión del protocolo (IPv4 o IPv6), dirección IP de origen, dirección IP de destino, tiempo de vida del paquete (TTL, *Time To Live*) y el protocolo de transporte utilizado (TCP, UDP u otro). El TTL es un contador que se decrementa en cada router; cuando llega a cero, el paquete se descarta y el emisor recibe una notificación, lo que evita que los paquetes circulen indefinidamente por la red.

2.1.3. Direcciones IP y redes locales

Una **dirección IPv4** es un número de 32 bits representado como cuatro grupos de decimales separados por puntos, cada uno entre 0 y 255. Por ejemplo: `192.168.1.100`.

Toda dirección IP tiene dos partes: la *dirección de red*, que identifica la red a la que pertenece el host, y la *dirección de host*, que identifica al dispositivo dentro de esa red.

La **máscara de subred** indica cuántos bits corresponden a la red y cuántos al host. Una máscara `255.255.255.0` (también escrita como `/24` en notación CIDR) significa que los primeros 24 bits identifican la red y los últimos 8 identifican el host: permite hasta 254 hosts en la misma red.

Existen rangos de direcciones reservados para redes privadas, no enrutables en Internet:

`10.0.0.0/8`**:** rango de clase A, hasta 16 millones de hosts.

`172.16.0.0/12`**:** rango de clase B, hasta un millón de hosts.

`192.168.0.0/16`**:** rango de clase C, hasta 65.000 hosts. El más común en redes domésticas y de oficina.

La dirección `127.0.0.1` se llama *loopback* o *localhost*: siempre apunta a la propia computadora y se usa para probar servicios localmente sin usar la red.

El agotamiento de las direcciones IPv4 (solo $2^{32} \approx 4{,}000$ millones) llevó al desarrollo de **IPv6**, que usa direcciones de 128 bits representadas en hexadecimal. Una dirección IPv6 típica es `2001:0db8:85a3:0000:0000:8a2e:0370:7334`. El espacio de 2^{128} direcciones es prácticamente inagotable.

2.1.4. Ruteo

El **ruteo** es el proceso por el cual los paquetes IP son encaminados desde el origen hasta el destino, posiblemente pasando por múltiples redes intermedias. Los dispositivos que realizan esta tarea son los **routers**.

Cada router mantiene una *tabla de ruteo*: una lista de redes de destino y el *salto siguiente* (*next hop*) al que debe reenviar los paquetes para cada destino. Cuando llega un paquete, el router busca en su tabla la entrada que mejor coincide con la dirección de destino y reenvía el paquete por la interfaz correspondiente.

La entrada más general de la tabla es la *ruta por defecto* (*default route*), representada como `0.0.0.0/0`: si ninguna otra entrada coincide, el paquete se envía al *gateway* predeterminado. En una red doméstica, el gateway es el router que conecta la red local con el proveedor de Internet.

El comando `ip route` (en Linux) o `netstat -r` muestra la tabla de ruteo de la computadora local. El comando `traceroute` (o `tracepath`) muestra la secuencia de routers que atraviesa un paquete hasta llegar a su destino, con el tiempo de cada salto.

Los protocolos de ruteo dinámico como OSPF y BGP permiten a los routers intercambiar información sobre la topología de la red y actualizar automáticamente sus tablas. BGP (*Border Gateway Protocol*) es el protocolo que mantiene la conectividad entre los grandes sistemas autónomos que componen Internet.

2.1.5. Nombres y direcciones de Internet: el DNS

Las personas recuerdan nombres como `www.unsa.edu.ar` mucho más fácilmente que direcciones IP como `200.3.141.170`. El **DNS** (*Domain Name System*) es el sistema que

traduce nombres de dominio en direcciones IP.

El espacio de nombres de Internet es jerárquico. Un nombre de dominio como `softwarelibre.unsa.edu.ar` se lee de derecha a izquierda: `ar` es el dominio de primer nivel (país Argentina), `edu` indica institución educativa, `unsa` es la Universidad Nacional de Salta, y `softwarelibre` es un subdominio dentro de esa institución.

Cuando un navegador necesita resolver `www.unsa.edu.ar`, consulta a su *servidor DNS recursivo* (generalmente provisto por el proveedor de Internet o configurado manualmente). Si ese servidor no tiene la respuesta en caché, sigue la jerarquía: consulta a uno de los 13 *servidores raíz*, que lo redirige al servidor autoritativo para `.ar`, que a su vez lo redirige al servidor de `edu.ar`, y así sucesivamente hasta obtener la dirección IP. El resultado se guarda en caché durante el tiempo indicado por el campo TTL del registro DNS.

Los tipos de registros DNS más importantes son:

A: asocia un nombre a una dirección IPv4.

AAAA: asocia un nombre a una dirección IPv6.

MX: indica los servidores de correo de un dominio.

CNAME: alias: un nombre que apunta a otro nombre.

NS: indica los servidores DNS autoritativos de un dominio.

TXT: texto libre, usado para verificación de dominio y configuración de correo (SPF, DKIM).

El comando `dig` permite consultar el DNS manualmente: `dig www.unsa.edu.ar` muestra la dirección IP asociada. `nslookup` es una alternativa disponible en todos los sistemas operativos.

2.1.6. Configuración de redes locales

En GNU/Linux, la configuración de red se realiza con los comandos del paquete `iproute2`:

`ip addr`**:** muestra las interfaces de red y sus direcciones IP asignadas.

`ip link`**:** muestra el estado de las interfaces (activa/inactiva, dirección MAC).

`ip route`**:** muestra la tabla de ruteo.

`ping`**:** verifica la conectividad con otro host enviando paquetes ICMP. `ping 8.8.8.8` verifica la conectividad con un servidor de Google.

`ip addr add`**:** asigna una dirección IP a una interfaz.

En la mayoría de las redes, la dirección IP se asigna automáticamente mediante **DHCP** (*Dynamic Host Configuration Protocol*). El servidor DHCP (usualmente el router) asigna

a cada dispositivo que se conecta una dirección IP, la máscara de subred, la dirección del gateway y la dirección del servidor DNS.

La dirección **MAC** (*Media Access Control*) es un identificador único de 48 bits grabado en la placa de red por el fabricante. Se usa en la capa de acceso a la red para identificar dispositivos dentro de la misma red local; a diferencia de la IP, no es enrutable.

NAT (*Network Address Translation*) es la técnica que permite que múltiples dispositivos de una red privada compartan una única dirección IP pública. El router reemplaza la IP privada del emisor por su propia IP pública al reenviar los paquetes hacia Internet, y realiza la traducción inversa al recibir las respuestas. NAT es la razón por la que todos los dispositivos de una red doméstica pueden acceder a Internet con una sola IP pública.

2.1.7. Comandos de red

SSH y SCP

SSH (*Secure Shell*) establece una conexión cifrada con otro sistema, permitiendo ejecutar comandos remotamente como si se estuviera en una terminal local. Reemplazó a `telnet`, que transmitía todo en texto claro incluyendo contraseñas.

```
ssh usuario@servidor.ejemplo.com
ssh -p 2222 usuario@192.168.1.10   # puerto no estándar
```

La autenticación puede hacerse con contraseña o con **par de claves**: una clave privada que permanece en la computadora del usuario y una clave pública que se instala en el servidor. Cuando el usuario se conecta, el servidor cifra un desafío con la clave pública; solo quien tenga la clave privada correspondiente puede descifrarlo. Este mecanismo es más seguro que las contraseñas porque la clave privada nunca viaja por la red.

Para generar un par de claves:

```
ssh-keygen -t ed25519 -C "mi_comentario"
```

Esto crea dos archivos: `~/.ssh/id_ed25519` (clave privada, nunca compartir) y `~/.ssh/id_ed25519.pub` (clave pública, se copia al servidor). Para instalar la clave pública en un servidor:

```
ssh-copy-id usuario@servidor.ejemplo.com
```

SCP (*Secure Copy*) copia archivos entre sistemas usando SSH como transporte:

```
scp archivo.txt usuario@servidor:/ruta/destino/
scp usuario@servidor:/ruta/archivo.txt .
scp -r directorio/ usuario@servidor:/ruta/
```

wget

`wget` descarga archivos desde la web o servidores FTP. Es ideal para scripts de automatización porque no requiere interacción:

```
wget https://ejemplo.com/archivo.pdf
wget -O nuevo_nombre.pdf https://ejemplo.com/archivo.pdf
wget -r -np https://ejemplo.com/docs/   # descarga recursiva
wget -c https://ejemplo.com/grande.iso  # continúa descarga interrumpida
```

curl

curl

`curl` transfiere datos hacia o desde un servidor y habla muchos protocolos (HTTP, HTTPS, FTP, SFTP, SMTP, ...). A diferencia de `wget`, por defecto imprime a stdout —lo que lo hace ideal para pipelines— y su especialidad es interactuar con APIs REST:

```
curl https://ejemplo.com/                          # GET, salida a stdout
curl -o archivo.pdf https://ejemplo.com/f.pdf    # guardar en archivo
curl -L https://acortador.com/xyz                  # sigue redirecciones
curl -X POST -d '{"k":"v"}' \
       -H "Content-Type: application/json" \
       https://api.ejemplo.com/endpoint            # POST con JSON
curl -H "Authorization: Bearer TOKEN" \
       https://api.ejemplo.com/privado             # con header de auth
```

rsync

`rsync` sincroniza archivos y directorios entre dos ubicaciones (local o remota vía SSH) transfiriendo solo las diferencias, lo que lo hace muy eficiente para backups y mirrors:

```
rsync -av origen/ destino/                         # local, preserva metadata
rsync -av --delete origen/ destino/                # espejo exacto (borra sobrantes)
rsync -av -e ssh dir/ usuario@host:/ruta/          # copia remota vía SSH
rsync -av --progress --exclude '*.tmp' src/ dst/ # con barra y exclusiones
```

Cuál elegir

Vale mencionar cuándo elegir cuál

- wget para "bajar un archivo grande o un sitio de una vez" (soporta reanudar y recursivo),
- rsync para "mantener dos árboles sincronizados" (idempotente, deltas),
- curl para "hablar con una API" (headers, métodos HTTP, pipes). Los tres se solapan en el caso simple "bajar un archivo" pero divergen apenas el caso se complica.

tcpdump

`tcpdump` captura y muestra los paquetes que pasan por una interfaz de red. Es una herramienta fundamental para el diagnóstico de problemas de red y para entender cómo funcionan los protocolos:

```
tcpdump -i eth0                    # captura en la interfaz eth0
tcpdump -i any port 80             # solo paquetes del puerto 80
tcpdump -i any host 8.8.8.8        # solo paquetes de/hacia esa IP
tcpdump -i any -w captura.pcap     # guarda en archivo para analizar
tcpdump -r captura.pcap            # lee un archivo capturado
```

La salida muestra para cada paquete: la hora, el protocolo, las direcciones y puertos de origen y destino, y la longitud. Esta herramienta permite ver en tiempo real la conversación entre dos aplicaciones, lo que hace evidente por qué los protocolos sin cifrado como HTTP son inseguros en redes no confiables.

nmap

`nmap` (*Network Mapper*) explora redes y sistemas, detectando qué hosts están activos, qué puertos tienen abiertos y qué servicios están corriendo:

```
nmap 192.168.1.1              # escanea un host
nmap 192.168.1.0/24           # escanea toda la red local
nmap -sV 192.168.1.1          # detecta versiones de servicios
nmap -p 22,80,443 servidor    # escanea puertos específicos
nmap -A 192.168.1.1           # detección agresiva (OS, versiones)
```

`nmap` es una herramienta legítima de administración y auditoría de seguridad. Escanear redes propias para verificar qué servicios están expuestos es una práctica recomendada. Escanear redes ajenas sin autorización es ilegal en la mayoría de los países.

2.1.8. HTTP y HTTPS

HTTP (*Hypertext Transfer Protocol*) es el protocolo de la web. Define cómo los navegadores solicitan recursos y cómo los servidores los entregan. Es un protocolo de petición-respuesta sin estado: cada petición es independiente de las anteriores.

Una petición HTTP tiene la forma:

```
GET /index.html HTTP/1.1
Host: www.ejemplo.com
Accept: text/html
```

El servidor responde con un código de estado y el recurso solicitado. Los códigos más importantes son:

200 OK: la petición fue exitosa.

301, 302: redirección a otra URL.

404 Not Found: el recurso no existe.

403 Forbidden: acceso denegado.

500 Internal Server Error: error en el servidor.

HTTP transmite todo en texto claro: cualquiera que pueda interceptar el tráfico puede ver el contenido de las páginas, los formularios enviados y las contraseñas ingresadas. Esta vulnerabilidad es crítica en redes WiFi públicas.

HTTPS añade una capa de cifrado mediante **TLS** (*Transport Layer Security*). Antes de intercambiar datos, el navegador y el servidor realizan un *handshake*: acuerdan algoritmos de cifrado, el servidor presenta su certificado digital (que prueba su identidad), y derivan una clave de sesión para cifrar la comunicación. Todo el contenido de la conversación posterior es ilegible para cualquier observador externo.

Los certificados TLS son emitidos por *autoridades de certificación* (CA) de confianza. El navegador verifica que el certificado fue emitido por una CA reconocida y que corresponde al dominio que se está visitando. El servicio **Let's Encrypt**, lanzado en 2016, ofrece certificados gratuitos y automatizados, lo que llevó a la adopción masiva de HTTPS.

Hoy HTTPS es el estándar: los navegadores marcan los sitios HTTP como "no seguros", y los motores de búsqueda penalizan en el ranking a los sitios sin cifrado.

2.1.9. Correo electrónico

El correo electrónico funciona mediante varios protocolos que se ocupan de distintas etapas del proceso:

SMTP (*Simple Mail Transfer Protocol*, puerto 25/587): se usa para *enviar* correo. Cuando se envía un mensaje, el cliente lo entrega al servidor SMTP del remitente, que lo reenvía al servidor SMTP del destinatario consultando los registros MX del DNS.

IMAP (*Internet Message Access Protocol*, puerto 143/993): permite *acceder* al correo almacenado en el servidor desde múltiples dispositivos, manteniendo los mensajes sincronizados. Es el protocolo recomendado actualmente.

POP3 (*Post Office Protocol*, puerto 110/995): descarga los mensajes al dispositivo local y opcionalmente los elimina del servidor. Adecuado cuando se usa un único dispositivo.

Un problema histórico del correo electrónico es que SMTP no tiene mecanismos de autenticación del remitente: cualquiera puede enviar un correo afirmando ser cualquier persona. Para mitigar esto se desarrollaron tres mecanismos complementarios:

SPF (*Sender Policy Framework*): un registro DNS TXT que declara qué servidores están autorizados a enviar correo en nombre de un dominio.

DKIM (*DomainKeys Identified Mail*): el servidor añade una firma digital al correo; el receptor puede verificarla con la clave pública publicada en el DNS del remitente.

DMARC: política que indica qué hacer con los correos que no superan las verificaciones SPF y DKIM.

El correo cifrado de extremo a extremo se implementa con **PGP/GPG**: el emisor cifra el mensaje con la clave pública del receptor; solo este puede descifrarlo con su clave privada. También firma digitalmente los mensajes para que el receptor pueda verificar que no fueron alterados.

2.1.10. Herramientas colaborativas en red

La red ha transformado radicalmente la forma en que las personas colaboran. Las principales categorías de herramientas son:

Control de versiones centralizado: Subversion (svn).

Control de versiones distribuido (también centralizado): Git con plataformas como GitHub, GitLab o Codeberg. Permiten que equipos distribuidos trabajen simultáneamente en el mismo código, manteniendo un historial completo de cambios y facilitando la revisión.

Comunicación en tiempo real: IRC fue el primer sistema de chat en red (1988) y sigue en uso en comunidades técnicas. Matrix es un protocolo federado y descentralizado moderno. Herramientas como Mattermost (software libre) o Slack (propietario) son equivalentes corporativos.

Edición colaborativa de documentos: sistemas wiki como MediaWiki (la base de Wikipedia) permiten que múltiples personas editen el mismo documento con control de versiones. Etherpad y CryptPad permiten edición simultánea en tiempo real con diferentes grados de privacidad.

Gestión de proyectos: sistemas de seguimiento de incidencias (*issue trackers*) como los integrados en GitLab o el Redmine independiente permiten registrar errores, planificar tareas y coordinar equipos.

Videoconferencia: Jitsi Meet es una plataforma de videoconferencia de código abierto que puede instalarse en servidores propios. BigBlueButton está orientado a la educación. Zoom y Meet son alternativas propietarias ampliamente utilizadas.

2.1.11. Redes sociales

Las redes sociales son plataformas que permiten a los usuarios crear perfiles, conectarse con otros y compartir contenido. Surgieron a principios de los años 2000 y transformaron profundamente la comunicación pública y privada.

Las plataformas centralizadas dominantes (Meta/Facebook, X/Twitter, Instagram, TikTok) son servicios propietarios financiados por publicidad. Su modelo de negocio consiste

en captar la atención del usuario el mayor tiempo posible y vender esa atención a anunciantes. Para maximizar el tiempo de uso, emplean algoritmos que priorizan el contenido que genera más reacciones emocionales, lo que tiene consecuencias documentadas sobre la salud mental, especialmente en adolescentes, y sobre la polarización política.

El **fediverso** es una alternativa descentralizada basada en el protocolo abierto **ActivityPub**. En lugar de una sola empresa controlando todos los datos, existen miles de servidores independientes (*instancias*) que se comunican entre sí. Mastodon (equivalente a Twitter/X), PeerTube (video), Pixelfed (fotos) y Lemmy (foros) forman parte del fediverso. Un usuario en una instancia puede seguir y comunicarse con usuarios de cualquier otra instancia, igual que el correo electrónico permite comunicación entre servidores distintos.

2.1.12. Bitcoin

Bitcoin es una **moneda digital descentralizada** propuesta por Satoshi Nakamoto (pseudónimo) en 2008. No depende de bancos ni gobiernos: las transacciones se firman con criptografía de clave pública y se registran en un libro contable público —la *blockchain*— mantenido por miles de nodos que validan bloques mediante

No es una moneda basada en deuda. Si está basada en la confianza. Algunos la reputan como una estafa Ponzi.

proof-of-work.

- **Unidad mínima**: 1 satoshi = 10^{-8} BTC.
- **Suministro fijo**: máximo 21 millones de BTC (por diseño).
- **Wallet**: par de claves criptográficas (pública = dirección, privada = firma). Perder la privada = perder los fondos, no hay recuperación.
- **Tiempo de bloque**: ∼10 minutos; una transacción se considera confirmada tras 6 bloques (∼1 hora).

Existen implementaciones libres del nodo completo (`bitcoin-core`) y clientes ligeros que consultan servidores externos (`electrum`). Correr un nodo completo requiere descargar la blockchain entera (cientos de GB y creciendo).

2.1.13. VPN

Una **VPN** (*Virtual Private Network*) construye un *túnel cifrado* entre su computadora y un servidor remoto: todo el tráfico sale por ese servidor, no por su proveedor de Internet local.

- **Privacidad**: su ISP solo ve tráfico cifrado hacia el servidor VPN, no los sitios que visita.
- **Cambio de ubicación aparente**: los servicios remotos ven la IP del servidor VPN, no la suya.

- **Evadir censura o filtros**: cuando la red local bloquea contenidos, la VPN los enruta desde otro punto.
- **Acceso a redes privadas**: conectarse a la LAN de la oficina desde casa como si estuviera físicamente ahí.

Protocolos habituales: **WireGuard** (moderno, rápido, simple), **OpenVPN** (maduro, muy configurable), **IPsec** (estándar empresarial). En Linux se configuran vía `wg-quick`, `openvpn` o el gestor de red del entorno gráfico.

Cuidado: la VPN traslada la confianza de su ISP al proveedor del servicio VPN, que ve todo el tráfico saliendo del túnel. Elegir proveedor con política de *no-logs* auditada, o mejor: autohospedar en un servidor propio.

Cada sección da la definición, tres o cuatro puntos operativos, y una advertencia práctica al final — mismo esqueleto que usaste para wget/rsync/curl. Encajan bien en un bloque más amplio de ïnternet y seguridad"junto con esos comandos, o después de licencias como cierre conceptual del práctico.

2.1.14. Voto electrónico

Bajo el nombre de *voto electrónico* se agrupan tecnologías muy distintas: desde urnas electrónicas de registro directo (DRE) que reemplazan al padrón papel completo, hasta el escaneo óptico de boletas físicas, hasta el voto remoto por Internet. El consenso académico entre criptógrafos e investigadores en seguridad es que **cualquier sistema donde el registro primario del voto sea digital es incompatible con los requisitos básicos de una elección democrática** (Simons y D. W. Jones 2012; Rivest 2008).

Los peligros

- **No verificabilidad por el votante.** Con una boleta papel el ciudadano ve físicamente lo que emitió. Con una urna electrónica ve una *pantalla que dice* lo que emitió — pero el bit que quedó registrado en memoria puede ser otro. No hay forma de que un no-experto verifique la coherencia entre lo mostrado y lo grabado (A. J. Feldman, Halderman y Felten 2007).
- **Imposibilidad de recuento independiente.** En elecciones con boleta papel, un escrutinio manual con veedores de todos los partidos siempre es posible. Con voto electrónico puro el "recuento" es re-ejecutar el mismo software que ya emitió el resultado — círculo vicioso. Recuentos manuales solo son posibles si existe un *Voter-Verified Paper Audit Trail* (VVPAT), y estos se descartaron en varias jurisdicciones por costo.
- **Código cerrado y proveedores privados.** Los sistemas comerciales (ES&S, Dominion, Smartmatic, Vot.ar) son propietarios. Los organismos electorales dependen de una empresa para garantizar la integridad del acto más importante de una democracia — *trust the vendor*. Auditorías externas se permiten con restricciones que impiden publicar resultados (Busaniche 2014; Smaldone 2017).

- **Superficie de ataque enorme.** Una urna electrónica es una computadora completa: firmware, OS, drivers, red, USB, tarjetas de memoria. Cada uno de esos puntos ha sido comprometido en demostraciones académicas: Diebold AccuVote-TS en 2006, Sequoia AVC en 2007, ES&S iVotronic en 2010, Dominion ImageCast en 2019 (Wolchok et al. 2010; Blaze et al. 2019).

- **Ataques masivos silenciosos.** Con boleta papel un fraude requiere manipular físicamente miles de urnas — operativamente detectable. Con voto electrónico un único cambio de código propaga el fraude a todas las urnas simultáneamente, sin huella física (Schneier 2004).

- **Falsa neutralidad tecnológica.** El discurso oficial presenta al voto electrónico como "modernización". En realidad es un traslado de la confianza pública (ejercida por veedores humanos observando papeles) a la confianza privada (ejercida por la palabra de una empresa proveedora) (Lessig 2006).

Recomendación técnica

El consenso entre organizaciones como *Verified Voting* (EEUU), *Fundación Vía Libre* (Argentina), *Chaos Computer Club* (Alemania) y las academias de ciencia de varios países es converger a un modelo mixto: **boleta papel escaneada ópticamente para tabulación rápida, con la boleta papel como registro autoritativo para auditoría y recuento**. Nunca voto por Internet en elecciones vinculantes.

El caso argentino de la Ciudad de Buenos Aires (Vot.ar, 2015) es paradigmático: el sistema fue comprometido semanas antes de la elección por Joaquín Sorianello, se demostraron múltiples vías de manipulación, y las autoridades respondieron con denuncia penal contra los investigadores en vez de con revisión del sistema (Smaldone 2017; Libre 2015).

2.2. Internet y sociedad

2.2.1. El código es ley

En 1999, el jurista Lawrence Lessig acuñó la frase *"el código es ley"* (*code is law*) para describir un fenómeno fundamental de la era digital: en el entorno de Internet, la arquitectura técnica del sistema determina qué es posible hacer, de manera más efectiva y directa que las leyes escritas.

Una ley prohíbe algo pero no impide físicamente hacerlo; su cumplimiento depende de la detección, el juicio y la sanción. El código, en cambio, puede hacer que algo sea técnicamente imposible: si una plataforma no tiene botón para copiar un contenido, la mayoría de los usuarios simplemente no lo copiarán, independientemente de si la copia sería legal.

Esta observación tiene consecuencias en múltiples direcciones. Las empresas que controlan las plataformas toman decisiones técnicas que tienen efectos sociales y políticos: qué contenidos son visibles, qué voces son amplificadas, qué comportamientos son posibles. Esas decisiones no son aprobadas por ningún parlamento ni revisadas por ningún tribunal; son simplemente parte del diseño del sistema.

El software libre y los protocolos abiertos son una respuesta a este problema: si el código de una plataforma es público y el protocolo es estándar, cualquiera puede auditar el comportamiento del sistema, crear clientes alternativos o levantar su propio servidor compatible. La arquitectura técnica se vuelve transparente y modificable, en lugar de ser una caja negra que gobierna sin rendir cuentas.

2.2.2. Libertad versus seguridad en la red

La tensión entre libertad y seguridad es uno de los debates más profundos de la era digital. Se manifiesta en múltiples dimensiones:

2.2.3. Vigilancia y privacidad

Las revelaciones de Edward Snowden en 2013 documentaron que agencias de inteligencia como la NSA realizaban vigilancia masiva de las comunicaciones digitales de millones de personas en todo el mundo, incluidas personas que no eran objeto de ninguna investigación. Esto reavivó el debate sobre los límites de la vigilancia estatal.

El cifrado de extremo a extremo protege las comunicaciones de personas comunes ante la vigilancia masiva. Periódicamente, gobiernos de distintos países proponen obligar a las empresas de tecnología a introducir *puertas traseras* (*backdoors*) en los sistemas de cifrado para que las fuerzas de seguridad puedan acceder a las comunicaciones con una orden judicial. Los expertos en criptografía argumentan que es técnicamente imposible crear una puerta trasera que solo puedan usar las autoridades legítimas: una vulnerabilidad introducida deliberadamente puede ser descubierta y explotada también por actores maliciosos.

2.2.4. Anonimato

El anonimato en Internet tiene usos legítimos: periodistas que protegen sus fuentes, activistas en regímenes autoritarios, víctimas de acoso que buscan apoyo. Herramientas como **Tor** (*The Onion Router*) permiten navegar de forma anónima enrutando el tráfico a través de múltiples nodos cifrados.

Al mismo tiempo, el anonimato facilita comportamientos negativos: acoso, desinformación, actividad criminal. La eliminación del anonimato resolvería algunos problemas pero destruiría la privacidad y la libertad de expresión de millones de personas. No existe una solución técnica que permita anonimato para los "buenos" y trazabilidad para los "malos": el mismo mecanismo sirve a ambos.

2.2.5. Neutralidad de la red

La **neutralidad de la red** es el principio según el cual los proveedores de acceso a Internet deben tratar todos los paquetes de datos de la misma manera, sin discriminar por origen, destino, tipo de contenido o protocolo. Sin neutralidad, un proveedor podría

favorecer sus propios servicios o los de socios comerciales, ralentizar a los competidores o bloquear contenidos legales.

Sus defensores argumentan que la neutralidad es condición necesaria para la innovación y la libertad de expresión: cualquiera puede crear un servicio que compita en igualdad de condiciones con los grandes actores. Sus detractores argumentan que permite que algunos servicios consuman ancho de banda desproporcionado sin pagar más.

La regulación varía según el país. La Unión Europea estableció la neutralidad de la red por ley en 2016. En los Estados Unidos ha sido objeto de debates regulatorios recurrentes.

2.2.6. Derechos de autor y acceso al conocimiento

El sistema de derechos de autor fue diseñado para un mundo de libros y discos físicos. En la era digital, donde la copia tiene costo cero y la distribución es instantánea y global, genera tensiones profundas. Las industrias culturales buscan extender las restricciones tecnológicas (*DRM*, *Digital Rights Management*) para controlar el uso de los contenidos. Los defensores del acceso abierto argumentan que el conocimiento humano debería ser accesible para todos.

Proyectos como Sci-Hub, que provee acceso gratuito a millones de artículos científicos bloqueados por editoriales comerciales, o la Biblioteca de Babel digital, ilustran la tensión entre los derechos de los creadores y distribuidores, el acceso al conocimiento como bien común y las posibilidades técnicas de la red.-

2.3. Ejercicios

1. **Ejercicio:** Capas TCP/IP

   Indique a qué capa del modelo TCP/IP pertenece cada uno de los siguientes protocolos y explique brevemente su función: HTTP, TCP, IP, Ethernet, DNS, UDP, SSH, WiFi.

2. **Ejercicio:** HTTP

   Un navegador solicita la página `https://www.unsa.edu.ar`. Describa paso a paso qué ocurre desde que el usuario presiona Enter hasta que la página aparece en pantalla, mencionando qué protocolos intervienen en cada etapa.

3. **Ejercicio:** TCP–UDP

   Explique la diferencia entre TCP y UDP. Para cada uno de los siguientes servicios, indique cuál de los dos protocolos es más adecuado y justifique: transferencia de un archivo, videollamada, consulta DNS, descarga de una página web, transmisión de audio en vivo.

4. **Ejercicio:** TTL y routers

Un paquete IP tiene un TTL inicial de 64. Dibuje el recorrido del paquete si pasa por 5 routers antes de llegar a destino, indicando el valor de TTL en cada salto. ¿Qué ocurre si hay un bucle de ruteo y el paquete nunca llega a destino?

5. **Ejercicio:** traceroute

   Ejecute en la terminal:

   ```
   traceroute www.google.com
   ```

   ¿Cuántos saltos hay hasta el destino? ¿Puede identificar en qué punto el tráfico sale de la red de su institución? ¿Qué significa cuando aparece `* * *` en una línea?

6. **Ejercicio:** Puertos

   Complete la siguiente tabla indicando el protocolo de aplicación y el servicio correspondiente a cada puerto:

Puerto	Protocolo	Servicio
22		
25		
53		
80		
443		
110		
143		

7. **Ejercicio:** Números UO y máscaras

   Dada la dirección IP `192.168.10.45` con máscara `255.255.255.0`:

   I. ¿Cuál es la dirección de red?
   II. ¿Cuántos hosts puede tener esa red?
   III. ¿Está `192.168.10.200` en la misma red?
   IV. ¿Está `192.168.11.1` en la misma red?

8. **Ejercicio:** Números y máscaras 2

   Repita el análisis anterior para la dirección `10.0.3.17` con máscara `255.255.0.0`.

9. **Ejercicio:** 192.168

   ¿Por qué las direcciones del rango `192.168.0.0/16` no son enrutables en Internet? ¿Qué mecanismo permite que dispositivos con esas direcciones accedan a Internet? ¿Qué ventaja tiene ese mecanismo para la seguridad?

10. **Ejercicio:** Ip pública

    Investigue la dirección IP pública de la red en la que se encuentra con el comando:

```
curl ifconfig.me
```

Compare con la dirección IP local que muestra `ip addr`. ¿Son iguales? ¿Por qué?

11. **Ejercicio:** IPv6

    Explique qué es IPv6 y por qué fue necesario crearlo. ¿Cuántas direcciones IPv6 existen? Exprese el resultado como potencia de 2 y como número decimal aproximado.

12. **Ejercicio:** comando ip

    Ejecute los siguientes comandos y anote e interprete la salida de cada uno:

    ```
    ip addr
    ip route
    ip link
    ```

    ¿Cuántas interfaces de red tiene la computadora? ¿Cuál es la dirección IP de cada una? ¿Cuál es el gateway predeterminado?

13. **Ejercicio:** ping

    El comando `ping` envía paquetes ICMP y mide el tiempo de respuesta. Realice las siguientes pruebas y anote los resultados:

    I. `ping -c 4 127.0.0.1`: ¿qué host es este?
    II. `ping -c 4 <gateway>`: ¿llega al router local?
    III. `ping -c 4 8.8.8.8`: ¿hay conectividad a Internet?
    IV. `ping -c 4 www.google.com`: ¿funciona la resolución DNS?

    Si alguna prueba falla, ¿qué indica sobre la configuración de red?

14. **Ejercicio:** Tabla de ruteo

    Analice la tabla de ruteo obtenida con `ip route`. Para cada entrada indique: la red de destino, la interfaz de salida y el siguiente salto. ¿Cuál es la ruta por defecto?

15. **Ejercicio:** DHCP

    Explique qué es DHCP y qué información configura automáticamente en un dispositivo cuando se conecta a una red. ¿Qué ventaja tiene respecto a la configuración manual?

16. **Ejercicio:** MAC address

    Investigue qué es una dirección MAC. ¿Puede cambiarse? ¿En qué se diferencia de una dirección IP? ¿En qué situaciones importa la dirección MAC y en cuáles importa la IP?

17. **Ejercicio:** dig

    Resuelva los siguientes nombres de dominio con el comando `dig` y anote la dirección IP correspondiente: `www.unsa.edu.ar`, `www.gnu.org`, `www.kernel.org`, `wikipedia.org`.

18. **Ejercicio:** Dominios

    Para el dominio `unsa.edu.ar`, consulte los registros MX con `dig unsa.edu.ar MX`. ¿Cuántos servidores de correo tiene? ¿Qué indica el número que aparece antes del nombre del servidor?

19. **Ejercicio:** Resolución inversa

    Realice una *resolución inversa*: dado una dirección IP, obtener el nombre de dominio asociado:

    ```
    dig -x 8.8.8.8
    dig -x 8.8.4.4
    ```

    ¿A quién pertenecen esas IPs?

20. **Ejercicio:** Delegación DNS

    Trace la jerarquía de delegación DNS para el dominio `softwarelibre.unsa.edu.ar`: ¿cuál es el servidor raíz? ¿cuál el servidor de `.ar`? ¿cuál el de `edu.ar`? ¿cuál el de `unsa.edu.ar`? Use `dig +trace softwarelibre.unsa.edu.ar` para verlo en acción.

21. **Ejercicio:** /etc/hosts

    El archivo `/etc/hosts` permite resolver nombres sin consultar al DNS. Examine su contenido. ¿Qué entradas tiene? ¿Para qué se usaba históricamente antes de que existiera el DNS?

22. **Ejercicio:** Claves ssh

    Genere un par de claves SSH con el algoritmo Ed25519:

    ```
    ssh-keygen -t ed25519 -C "nombre_apellido@unsa"
    ```

    I. ¿Dónde se guardan los archivos generados?
    II. ¿Cuál es la clave pública y cuál la privada?
    III. ¿Qué pasaría si alguien obtuviera su clave privada?
    IV. ¿Para qué sirve la *passphrase*?

23. **Ejercicio:** Clave pública

    Copie su clave pública a un servidor en donde tenga acceso ssh con `ssh-copy-id` y verifique que puede conectarse sin contraseña. Explique qué ocurrió en el servidor: ¿en qué archivo se guardó su clave pública?

24. **Ejercicio:** Servidor remoto

    Una vez conectado por SSH al servidor remoto:

    I. Liste los usuarios del sistema consultando `/etc/passwd`.

    II. ¿Cuántos procesos hay en ejecución? Use `ps aux | wc -l`.

    III. ¿Cuánto espacio libre hay en disco? Use `df -h`.

    IV. Cree un archivo con su nombre en su directorio personal y verifique que otro estudiante puede verlo.

25. **Ejercicio:** scp

    Transfiera con `scp` un archivo de su computadora al servidor y viceversa. Verifique la integridad comparando los checksums:

    ```
    md5sum archivo.txt
    sha256sum archivo.txt
    ```

    ¿Para qué sirve verificar el checksum de un archivo descargado?

26. **Ejercicio:** telnet

    Explique por qué `telnet` fue reemplazado por `ssh`. ¿Qué puede ver un atacante que intercepte una sesión `telnet`? Use `tcpdump` para capturar y examinar el tráfico.

27. **Ejercicio:** wget

    Use `wget` para descargar los siguientes recursos y anote el tiempo de descarga y el tamaño de cada uno:

    I. La página principal de `www.gnu.org` (`wget www.gnu.org`).

    II. El manual de GNU Coreutils en PDF desde `https://www.gnu.org/software/coreutils/ma`

    III. Interrumpa la descarga anterior con Ctrl+C y reanúdela con `wget -c`. ¿Desde dónde continuó?

28. **Ejercicio:** Captura de tráfico DNS

    Capture el tráfico de red mientras realiza una consulta DNS con dos terminales simultáneamente:

    - Terminal 1: `tcpdump -i any port 53 -n`
    - Terminal 2: `dig www.wikipedia.org`

    Identifique en la captura: la petición DNS (qué pregunta), la respuesta (qué responde) y las IPs involucradas.

29. **Ejercicio:** Captura HTTP y HTTPS

    Capture el tráfico HTTP de una conexión no cifrada y el de una conexión HTTPS:

- `tcpdump -i any -A port 80` mientras accede a un sitio HTTP.
- `tcpdump -i any -A port 443` mientras accede a un sitio HTTPS.

¿Puede leer el contenido de la página en el tráfico HTTP? ¿Y en el HTTPS? Explique la diferencia.

30. **Ejercicio:** nmap

    Escanee la red local con `nmap`:

    ```
    nmap -sn 192.168.1.0/24
    ```

    ¿Cuántos dispositivos activos encontró? Luego escanee el gateway de la red:

    ```
    nmap -sV <ip_del_gateway>
    ```

    ¿Qué puertos tiene abiertos? ¿Qué servicios están corriendo?

31. **Ejercicio:** nmap propio con servidor 8080

    Escanee su propia computadora con `nmap localhost`. ¿Qué puertos tiene abiertos? ¿Esperaba esos resultados? Inicie un servidor web simple con:

    ```
    python3 -m http.server 8080
    ```

    Vuelva a escanear. ¿Apareció el puerto 8080?

32. **Ejercicio:** Desarrollo web

    Abra las herramientas de desarrollo del navegador (F12), vaya a la pestaña *Red* y acceda a `www.wikipedia.org`. Examine la primera petición HTTP y responda:

    I. ¿Qué método HTTP se usó (GET, POST)?
    II. ¿Cuál fue el código de respuesta?
    III. ¿Cuántos recursos adicionales (CSS, JS, imágenes) se descargaron?
    IV. ¿Cuánto tiempo tardó en cargar la página completa?
    V. ¿El sitio usa HTTP o HTTPS? ¿Cómo lo verifica?

33. **Ejercicio:** TLS

    Examine el certificado TLS de `www.unsa.edu.ar`: haga clic en el candado del navegador y luego en "Ver certificado". Responda:

    I. ¿Quién emitió el certificado?
    II. ¿Cuándo vence?
    III. ¿Qué dominios cubre?
    IV. ¿Qué algoritmo de firma usa?

34. **Ejercicio:** curl

    Realice una petición HTTP desde la terminal usando `curl`:

    ```
    curl -v http://www.example.com 2>&1 | head -50
    ```

    Identifique en la salida: la línea de petición, los headers enviados, el código de respuesta y los headers recibidos.

35. **Ejercicio:** 301–302

    ¿Por qué un código de respuesta 301 es diferente de un 302? Verifíquelo accediendo con `curl -v http://wikipedia.org` (sin www). ¿Adónde redirige?

36. **Ejercicio:** Ataques WiFi

    Explique qué información puede obtener un atacante que intercepte tráfico HTTP en una red WiFi pública. ¿Cambia la situación con HTTPS? ¿Qué datos siguen siendo visibles aunque se use HTTPS?

37. **Ejercicio:** MX

    Consulte los registros MX de su dominio universitario con `dig unsa.edu.ar MX`. ¿A qué servidores de correo llegan los mensajes dirigidos a ese dominio?

38. **Ejercicio:** IMAP –POP3

    Explique la diferencia entre IMAP y POP3. Si usa el correo desde su teléfono y su computadora simultáneamente, ¿cuál de los dos protocolos conviene configurar? ¿Por qué?

39. **Ejercicio:** mail

    Envíe un correo con `mail` desde la terminal al docente y copie a un compañero:

    ```
    echo "Mensaje de prueba" | mail -s "Asunto" \
      docente@unsa.edu.ar -c compañero@unsa.edu.ar
    ```

    Verifique los encabezados del correo recibido. ¿Qué información revelan los campos `Received`?

40. **Ejercicio:** SPF, DKIM, DMARC

    Explique qué son SPF, DKIM y DMARC. ¿Por qué son necesarios si SMTP ya existe desde 1982? ¿Qué problema de SMTP intentan resolver?

41. **Ejercicio:** Correo cifrado

    ¿Qué es el correo cifrado de extremo a extremo? ¿En qué se diferencia de simplemente usar HTTPS para acceder al webmail? ¿Puede el proveedor de correo leer sus mensajes en cada caso?

    Nota: Piense que tipo de acciones pueden hacer los fabricantes de teléfonos para leer todo lo que hace con su movil, más allá de la encriptación de extremo a extremo, considerando que los estados lo ordenan en muchos casos. Investigue sobre leyes en tal sentido en diferentes países. ¿Puede la IA darle información completa sobre como evadir éstas cuestiones?

42. **Ejercicio:** Redes sociales

    Investigue el modelo de negocios de una red social centralizada de su elección (Facebook, Instagram, TikTok, X). Responda:

    I. ¿Cómo genera ingresos?
    II. ¿Qué datos recopila sobre sus usuarios?
    III. ¿Cómo usa esos datos?
    IV. ¿Qué ocurre con sus datos si decide dejar de usar la plataforma?
    V. ¿Tiene elmicrófono o la cámara del celular abiertos ahora?
    VI. ¿Puede realmente cerrarlos?
    VII. ¿Funciona apagar el celular? Pruebe apagarlo y referirse en voz alta a un tema como "horóscopos", ¿le llegan en los próximos días publicidades sobre éste tema?. Pruebe lo mismo con Whatsapp, signal, telegram, matrix. Para cada prueba use palabras claves diferentes, con texto y con sonido. ¿Y el método Snowden de poner el celular en un termo metálico o una coctelera adentro de una heladera?
    VIII. Investigue sobre Chat–control.

43. **Ejercicio:** Privacidad vs. seguridad

    El cifrado de extremo a extremo protege la privacidad de los usuarios pero dificulta la investigación de actividades criminales. Argumente dos posiciones:

    I. A favor de exigir que los sistemas de mensajería permitan acceso de las fuerzas de seguridad con orden judicial.
    II. En contra de esa exigencia, desde el punto de vista técnico y de derechos civiles.

    ¿Existe una solución técnica que satisfaga ambas posiciones simultáneamente? Justifique.

44. **Ejercicio:** Mastodon

    Compare el modelo de una red social centralizada con el del fediverso (Mastodon, por ejemplo). Complete la siguiente tabla:

Aspecto	Centralizada	Fediverso
¿Quién controla los datos?		
¿Puede cambiarse de proveedor?		
¿El protocolo es abierto?		
¿Cómo se financia?		
¿Puede auditarse el algoritmo?		

45. **Ejercicio:** Código es ley

    Lawrence Lessig afirmó que "el código es ley". Elija una decisión técnica de una plataforma digital (por ejemplo: el algoritmo de recomendación de YouTube, el límite de caracteres de X, la ausencia de cifrado en los mensajes de Facebook Messenger hasta cierta fecha) y analícela como si fuera una norma jurídica: ¿a quién beneficia? ¿a quién perjudica? ¿quién la decidió? ¿puede apelarse?

46. **Ejercicio:** Tor

    Investigue qué es Tor y cómo funciona. Enumere tres usos legítimos del anonimato en Internet y tres usos problemáticos. ¿Debería prohibirse Tor? ¿Podría prohibirse efectivamente?

47. **Ejercicio:** BitCoin

    Investigue BitCoin, ¿es realmente dinero anónimo como el oro o el papel?. ¿Es una moneda sustentable ecológicamente?. ¿Que es el block chain?

48. **Ejercicio:** Neutralidad de la red

    Analice el siguiente escenario: un proveedor de Internet ralentiza el tráfico de Netflix pero no el de su propio servicio de video.

    I. ¿Viola la neutralidad de la red?
    II. ¿Qué efectos tiene sobre la competencia?
    III. ¿Qué efectos tiene sobre los usuarios?
    IV. ¿Debería regularse? ¿Quién debería hacerlo?

49. **Ejercicio:** Sci-hub

    Reflexione sobre el acceso a artículos científicos. Un investigador en una universidad con recursos limitados no puede pagar los aranceles de acceso a las revistas académicas donde se publican los resultados de investigaciones financiadas con fondos públicos. Analice este escenario desde las perspectivas de:

    I. Las editoriales académicas.
    II. Los investigadores autores de los artículos.
    III. Los investigadores que necesitan leer los artículos.
    IV. La sociedad que financia la investigación con impuestos.

¿Qué soluciones existen? ¿Cuáles son sus limitaciones?

50. **Ejercicio:** VPN

    Investigue sobre VPNs, especialmente gratuitas. ¿Que logran? ¿Pueden ser comprometidas por estados extranjeros?

2.4. Laboratorio

El objetivo de esta práctica es configurar manualmente una red local, operar herramientas de transferencia segura e insegura de archivos, y demostrar de forma concreta por qué el cifrado es necesario: se transferirá un archivo de audio sin cifrado, se capturará el tráfico con un analizador de paquetes y se reconstruirá el archivo original a partir de los paquetes capturados.

Laboratorio: Configuración de red local, transferencia segura e interceptación de paquetes

Usar un router y conectar al menos dos computadoras configurando a mano la red

ssh eject

scp

usar trazadores de paquetes

intenrar copiar de una maquia a otra en forma plana, sin encriptar un archivo de audio, Reconstruir con los paquetes un wav similar al original, que se escuche

a) **Configuración manual de la red.** Conecte al menos dos computadoras a un router mediante cable Ethernet. Deshabilite el DHCP del router y asigne manualmente las siguientes direcciones:

Máquina	IP	Máscara	Gateway
PC-A (servidor)	`192.168.10.1`	`255.255.255.0`	`192.168.10.254`
PC-B (cliente)	`192.168.10.2`	`255.255.255.0`	`192.168.10.254`

   En cada computadora ejecute:

```
ip addr add 192.168.10.X/24 dev eth0
ip link set eth0 up
ip route add default via 192.168.10.254
```

   Verifique la conectividad con `ping` entre ambas máquinas. Documente la salida de `ip addr` e `ip route` en cada una.

b) **SSH: acceso remoto seguro.** En PC-A, verifique que el servidor SSH está activo:

```
systemctl status ssh
```

Desde PC-B, conéctese a PC-A:

```
ssh usuario@192.168.10.1
```

Una vez dentro, ejecute `who` para confirmar que está conectado remotamente. Cierre la sesión con `exit`.

Genere un par de claves en PC-B y configure la autenticación sin contraseña:

```
ssh-keygen -t ed25519
ssh-copy-id usuario@192.168.10.1
ssh usuario@192.168.10.1
```

¿Pidió contraseña esta vez? Explique qué ocurrió.

c) **SCP: copia segura de archivos.** Cree en PC-B un archivo de texto con su nombre:

```
echo "Práctica de redes - $(whoami)" > identificacion.txt
```

Cópielo a PC-A con SCP y verifique que llegó:

```
scp identificacion.txt usuario@192.168.10.1:/tmp/
ssh usuario@192.168.10.1 "cat /tmp/identificacion.txt"
```

Luego copie un directorio completo:

```
scp -r practico/ usuario@192.168.10.1:/tmp/
```

d) **Captura de tráfico con tcpdump.** En PC-A, inicie una captura de todo el tráfico de la red local:

```
tcpdump -i eth0 -w captura.pcap
```

Desde PC-B, realice las siguientes acciones mientras la captura está activa:

- Un `ping` a PC-A.
- Una conexión SSH.
- Una consulta DNS: `dig www.gnu.org`.

Detenga la captura con Ctrl+C. Examine el resultado:

```
tcpdump -r captura.pcap -n | head -40
tcpdump -r captura.pcap -n port 22
```

¿Puede leer el contenido de la sesión SSH en los paquetes capturados? ¿Por qué?

e) **Transferencia sin cifrado e interceptación.** Esta parte demuestra por qué el cifrado es indispensable. Se transferirá un archivo de audio WAV sin ningún cifrado usando `netcat`, se capturará el tráfico y se reconstruirá el archivo original a partir de los paquetes.

Preparación del archivo. En PC-A, genere o copie un archivo WAV de al menos 10 segundos de duración. Para generar uno sintético:

```
sox -n -r 44100 -c 1 audio_prueba.wav \
    synth 10 sine 440
```

Verifique que es audible:

```
aplay audio_prueba.wav
```

Anote el tamaño exacto del archivo:

```
wc -c audio_prueba.wav
md5sum audio_prueba.wav
```

Captura previa al envío. En PC-B (el interceptor), inicie la captura en el puerto que se usará para la transferencia (por ejemplo, puerto 9000) y guárdela en un archivo:

```
tcpdump -i eth0 port 9000 -w intercepcion.pcap
```

Transferencia sin cifrado con netcat. En PC-A (receptor), ponga `netcat` a escuchar y redirigir lo recibido directamente a un archivo:

```
nc -l -p 9000 > recibido.wav
```

En PC-B (emisor), envíe el archivo:

```
nc 192.168.10.1 9000 < audio_prueba.wav
```

Cuando termine la transferencia, detenga la captura en el interceptor con Ctrl+C.

Verifique que el archivo llegó íntegro a PC-A:

```
md5sum recibido.wav
aplay recibido.wav
```

¿Coincide el MD5 con el original?

Reconstrucción desde los paquetes capturados. En el interceptor (PC-B), extraiga los datos de los paquetes capturados. `tcpdump` con la opción `-A` muestra el payload en ASCII; para extraer los bytes crudos se usa `tshark` o se procesa el pcap directamente:

```
tshark -r intercepcion.pcap \
       -Y "tcp.port == 9000" \
       -T fields -e data \
       | tr -d '\n' \
       | xxd -r -p > reconstruido.wav
```

Alternativamente, con `tcpflow` que reconstruye automáticamente los flujos TCP:

```
tcpflow -r intercepcion.pcap port 9000
ls -lh
```

El archivo generado por `tcpflow` contiene exactamente los bytes del flujo TCP, que en este caso son los bytes del WAV. Renómbrelo:

```
mv 192.168.010.002.????-192.168.010.001.09000 interceptado.wav
aplay interceptado.wav
md5sum interceptado.wav
```

¿Se escucha el audio? ¿Coincide el MD5 con el archivo original?

f) **Análisis y comparación.** Repita la transferencia del mismo archivo WAV pero esta vez usando SCP:

```
scp audio_prueba.wav usuario@192.168.10.1:/tmp/seguro.wav
```

Capture el tráfico simultáneamente en PC-B:

```
tcpdump -i eth0 port 22 -w scp_captura.pcap
```

Intente extraer el archivo de audio de esta captura usando el mismo procedimiento anterior. ¿Obtiene un WAV audible? ¿Por qué?

Guía para la resolución

Requisitos de software

Instale en ambas computadoras antes de comenzar:

```
apt install openssh-server openssh-client netcat-openbsd \
    tcpdump tshark tcpflow sox alsa-utils iproute2
```

- **Parte A: configuración manual de red**

  La configuración manual de direcciones IP es temporal: se pierde al reiniciar. Para hacerla persistente habría que editar los archivos de configuración de red, pero para una práctica de laboratorio la configuración temporal es suficiente.

  Si `eth0` no es el nombre de la interfaz en su sistema, encuéntrelo con `ip link show`. En sistemas modernos los nombres pueden ser `enp3s0`, `ens33` u otros.

  Si el router no tiene opción para deshabilitar el DHCP, puede igualmente asignar IPs estáticas; simplemente asegúrese de que las direcciones elegidas no estén en el rango que entrega el DHCP (consulte la configuración del router).

  Verifique la conectividad en orden:

  1. `ping 127.0.0.1`: verifica que la pila TCP/IP funciona localmente.
  2. `ping 192.168.10.X`: verifica que la interfaz tiene la IP correcta.
  3. `ping 192.168.10.Y`: verifica que la otra máquina es alcanzable.

Si alguno falla, no avance: diagnostique y resuelva antes de continuar.

- **Parte B y C: SSH y SCP**

  Si SSH da el error `Connection refused`, el servidor no está corriendo. Inícielo con:

  ```
  systemctl start ssh
  ```

  La primera vez que se conecta a un host por SSH, el sistema pregunta si confía en su clave pública (*host key*). Responda `yes`. Esa clave queda registrada en `~/.ssh/known_hosts`; si la clave del servidor cambia, SSH advertirá de una posible suplantación.

  Si `ssh-copy-id` falla por restricciones de contraseña, puede copiar la clave manualmente:

  ```
  cat ~/.ssh/id_ed25519.pub | ssh usuario@192.168.10.1 \
    "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
  ```

- **Parte D: captura de tráfico**

  `tcpdump` requiere privilegios de superusuario o pertenencia al grupo `pcap`. Use `sudo` si es necesario.

  Para ver el tráfico en tiempo real mientras se captura a archivo, use dos terminales: una para la captura (`-w`) y otra para la visualización en tiempo real:

  ```
  # Terminal 1: captura a archivo
  sudo tcpdump -i eth0 -w captura.pcap

  # Terminal 2: visualización en tiempo real
  sudo tcpdump -i eth0 -n -A
  ```

- **Parte E: interceptación del audio**

  La transferencia con `netcat` usa TCP sin ninguna capa de cifrado. El flujo de bytes del archivo WAV viaja tal cual por la red. Esto significa que:

  - Un formato WAV tiene una cabecera fija de 44 bytes al inicio. Si la reconstrucción está bien hecha, los primeros bytes del archivo interceptado serán `52 49 46 46` (`RIFF` en ASCII), que es la firma de los archivos WAV. Verifíquelo:

    ```
    xxd interceptado.wav | head -3
    ```

  - Si el MD5 del archivo interceptado coincide con el original, la interceptación fue perfecta: se obtuvo una copia exacta bit a bit sin que ni el emisor ni el receptor lo supieran.
  - Si el audio se escucha correctamente, queda demostrado que cualquier persona en la red local con acceso al medio puede capturar y reconstruir cualquier dato transmitido sin cifrado: contraseñas, documentos, conversaciones, archivos de cualquier tipo.

  Si `tshark` no está disponible, puede reconstruir el flujo usando solo `tcpdump` y `dd`:

```
tcpdump -r intercepcion.pcap -A -s 0 port 9000 \
  | grep -v "^[0-9]" \
  | grep -v "^$" > payload_ascii.txt
```

Este método es menos preciso. Se recomienda `tcpflow`, que hace la reconstrucción automáticamente y con mayor fidelidad.

Si el archivo WAV reconstruido tiene algunos segundos de silencio al principio o no empieza exactamente igual que el original, es porque los primeros paquetes no fueron capturados. Inicie siempre la captura *antes* de comenzar la transferencia.

- **Parte F: comparación con SCP**

  Al intentar extraer el WAV de la captura de SCP, el archivo resultante no será audible: contendrá datos cifrados que parecen ruido aleatorio. Este contraste visual y auditivo es la demostración más directa de qué cambia el cifrado: los mismos bytes del archivo de audio viajan por la red en la transferencia con `netcat`, pero en la transferencia con SCP viajan cifrados y son indistinguibles del ruido.

  Puede visualizar el contraste examinando el contenido hexadecimal de ambas capturas:

  ```
  tcpdump -r intercepcion.pcap -A port 9000 | head -30
  tcpdump -r scp_captura.pcap  -A port 22   | head -30
  ```

  En la primera captura se verán fragmentos reconocibles del archivo de audio. En la segunda, solo caracteres aparentemente aleatorios.

- **Preguntas de cierre**

  Al terminar la práctica, reflexione y responda:

  1. ¿Qué protocolos de los que usa cotidianamente transmiten datos sin cifrado? ¿Qué riesgos implica usarlos en una red WiFi pública?
  2. El ejercicio usó `netcat` para la transferencia sin cifrado. ¿Conoce algún protocolo de uso común que haga algo similar? ¿FTP? ¿HTTP? ¿Telnet?
  3. Si en lugar de un archivo de audio se hubiera transferido una contraseña o un número de tarjeta de crédito usando un protocolo sin cifrado, ¿habría podido recuperarlo igual? Demuéstrelo.
  4. ¿Qué habría ocurrido si las dos computadoras estuvieran conectadas por WiFi en lugar de cable? ¿Sería más o menos fácil interceptar el tráfico?
  5. La práctica se realizó en una red de laboratorio controlada. ¿Es legal hacer lo mismo en una red ajena? ¿Y en una red propia?

Capítulo 3

Edición de textos

3.1. Edición de textos, estándares

3.1.1. Editores de texto plano

Un **editor de texto plano** es un programa que crea y modifica archivos que contienen únicamente caracteres codificados en algún estándar como ASCII o UTF-8, sin ninguna información de formato embebida en el archivo. El contenido es exactamente lo que se ve: no hay fuentes, colores, estilos ni metadatos ocultos.

Esta simplicidad es una fortaleza fundamental. Un archivo de texto plano puede abrirse con cualquier programa en cualquier sistema operativo, puede procesarse con herramientas de línea de comandos (`grep`, `sed`, `awk`), puede controlarse con sistemas de control de versiones como Git, y seguirá siendo legible dentro de décadas sin importar qué aplicaciones existan en el futuro.

Los editores de texto plano más utilizados en entornos GNU/Linux son:

nano: editor simple orientado a principiantes, con los comandos más importantes visibles en la parte inferior de la pantalla.

vi / vim: editor modal extremadamente poderoso, presente en prácticamente todos los sistemas Unix. Tiene dos modos: *normal* (para navegar y ejecutar comandos) e *inserción* (para escribir texto). La curva de aprendizaje es pronunciada pero los usuarios expertos lo consideran el editor más eficiente.

emacs: editor extensible mediante Lisp, con funcionalidades que van mucho más allá de la edición de texto: correo, agenda, depurador, intérprete de comandos. Es el editor emblemático del Proyecto GNU. Excelente para LaTeX

gedit, kate, geany: editores gráficos con interfaz visual para escritorio, con resaltado de sintaxis para múltiples lenguajes de programación.

Visual Studio Code: editor gráfico moderno, extensible, con soporte para centenares de lenguajes y un ecosistema enorme de extensiones. Aunque el editor base es de código abierto (MIT), la distribución oficial de Microsoft incluye componentes propietarios.

ed: es el editor de líneas original de Unix (Ken Thompson, 1969), presente en cualquier sistema POSIX. Es el ancestro directo de `sed`, `grep`, `vi` y de la sintaxis de expresiones regulares.

Además de valor histórico, puede servir en condiciones de trabajo muy frugales.

Es **modal**: en modo comando cada línea es una orden; con `a`, `i` o `c` se entra al modo inserción, del que se sale tipeando un `.` solo. Es **silencioso**: no muestra el archivo salvo que lo pidas explícitamente; los errores son un simple `?`.

```
$ ed -p '*' archivo.txt      # -p '*' activa un prompt visible
*a                            # entrar en modo inserción
Primera línea.
Segunda línea.
.                             # fin de inserción
*1,$p                         # imprimir de la 1 a la última
*s/Segunda/2da/               # sustituir en la línea actual
*w                            # guardar
*q                            # salir
```

Comandos básicos: `p` imprimir, `d` borrar, `a`/`i`/`c` agregar/insertar/cambiar, `s/patron/reemplazo/` sustituir, `/patron/` buscar, `w` guardar, `q` salir. Los rangos se dan como prefijo: `2,5d` borra líneas 2 a 5; `,p` o `%p` imprime todo.

Hoy sigue usándose principalmente en dos casos: en sistemas de rescate donde no anda una terminal completa, y en scripts como alternativa portable a `sed -i`:

```
ed -s archivo.txt <<'EOF'
%s/color/colour/g
w
q
EOF
```

sed: (*stream editor*) es `ed` aplicado a un flujo: lee su entrada línea por línea, ejecuta comandos tipo ed sobre cada una, y escribe el resultado en la salida.

No es interactivo como losotros editores de ésta lista. Es la herramienta estándar para transformaciones de texto en pipelines Unix.

```
sed 's/foo/bar/' archivo.txt          # primera ocurrencia por línea
sed 's/foo/bar/g' archivo.txt         # todas las ocurrencias
sed 's|/usr/local|/opt|g' arch.txt    # separador alternativo
sed -i 's/foo/bar/g' archivo.txt      # edición in-place
sed -i.bak 's/foo/bar/g' arch.txt     # in-place con backup .bak

sed -n '5,10p' archivo.txt            # imprime solo líneas 5 a 10
sed '/^#/d' config                    # borra líneas que empiezan con #
sed '$d' archivo                      # borra la última línea
```

```
sed '10q' archivo                    # sale después de la línea 10
                                      %   (equivalente a head -10)
```

Se pueden encadenar varios comandos con `-e` o con `;`:

```
sed -e 's/rojo/red/g' -e 's/azul/blue/g' archivo.txt
sed 's/rojo/red/g; s/azul/blue/g' archivo.txt
```

Los rangos y direcciones funcionan igual que en `ed`:

```
sed '10,20s/foo/bar/g' archivo       # sustituye solo en líneas 10-20
sed '/inicio/,/fin/d' archivo        # borra desde línea que dice
                                      % inicio hasta la que dice fin
```

Referencias hacia atrás en el reemplazo con `\1`, `\2`, etc., y captura con paréntesis *escapados* (`\(` y `\)`) en modo BRE, o directos con `-E` para ERE:

```
sed -E 's/([a-z]+)@([a-z]+)/\2 punto \1/' emails.txt
# convierte "juan@empresa" en "empresa punto juan"
```

`sed` tiene un lenguaje completo con etiquetas, saltos y un buffer secundario (*hold space*), pero para cualquier cosa más compleja que sustituciones y borrado por rangos conviene pasarse a `awk` o a un script en Python/Perl.

3.1.2. WYGIWYW, WYSIWYM: procesadores de texto basados en comandos

El acrónimo **WYGIWYW** (*What You Get Is What You Want*, "obtenés lo que querés") o **WYSIWYM** (*What You See Is What You Mean*) describe un modelo de edición en el que el autor escribe el contenido del documento junto con marcas o comandos que describen su estructura y formato, y un programa separado procesa esas marcas para producir el documento final. El autor nunca ve directamente el resultado mientras escribe: escribe un archivo fuente que luego se compila o procesa.

Este modelo se contrapone al WYSIWYG (*What You See Is What You Get*), donde el documento en pantalla durante la edición es idéntico al resultado final.

Las ventajas del modelo WYGIWYW son múltiples. El autor se concentra en el contenido y la estructura, no en la apariencia visual. El formato es consistente porque está definido por reglas, no por decisiones manuales ítem por ítem. El archivo fuente es texto plano, controlable con Git, procesable con scripts y legible sin software especializado. La separación entre contenido y presentación permite producir múltiples salidas (PDF, HTML, ebook) desde el mismo fuente.

Los sistemas WYGIWYW más importantes son:

históricos: El modelo WYGIWYW no es nuevo: troff / nroff / roff son de 1964, mm y me son macros de esa era.

TEX/LATEX/TypST: para documentos científicos y técnicos con alta calidad tipográfica. Se detalla en la sección siguiente.

TypST: nueva alternativa a LaTeX (2023-), vale la pena mencionar como tendencia

Nota: LATEXes el sucesor moderno de una tradición Unix de 60 años. Lo que contextualiza políticamente contra la impresión de que WYSIWYG es "lo natural" y lo otro es geek.

Markdown: lenguaje de marcas ligero diseñado para ser legible como texto plano y procesable a HTML u otros formatos.

reStructuredText: usado en la documentación de Python y otros proyectos de software.

DocBook: formato XML para documentación técnica extensa, capaz de generar PDF, HTML, ebook y otros formatos desde el mismo fuente.

Org-mode: formato de texto plano integrado en emacs, usado para notas, documentación, planificación y generación de documentos en múltiples formatos.

3.1.3. Editores WYSIWYG

Un editor **WYSIWYG** (*What You See Is What You Get*, "lo que ves es lo que obtenés") muestra el documento durante la edición tal como aparecerá en el resultado final. El usuario no escribe comandos de formato sino que aplica formatos directamente mediante menúes, botones y atajos de teclado, y ve el resultado de inmediato.

El paradigma WYSIWYG fue popularizado por el sistema Bravo en el PARC de Xerox en los años 70 y llegó al público masivo con el Macintosh y el procesador de texto MacWrite en 1984. Su ventaja principal es la baja barrera de entrada: cualquier usuario puede producir un documento con formato sin aprender ningún lenguaje de marcas.

Los procesadores de texto WYSIWYG más utilizados son:

LibreOffice: procesador múltiple: texto, planillas, presentaciones, matemáticas, gráficos, etc. libre y gratuito, compatible con el formato DOCX de Microsoft. Un solo ejectuable, muchos formatos y tipos de contenido.

OnlyOffice: alternativa también libre a LibreOffice, con muy alta compatibilidad DOCX y edición colaborativa en tiempo real integrada.

Microsoft Word: el procesador de texto más extendido comercialmente. Formato propietario DOCX basado en XML.

Google Docs: procesador de texto en línea, colaborativo en tiempo real, sin instalación. Los documentos se almacenan en la nube de Google. El servicio es propietario.

Etherpad, HedgeDoc y CryptPad: alternativas libres a Google Docs con edición colaborativa en tiempo real, sin depender de la nube de una empresa privada.

3.1.4. Fortalezas y debilidades de ambos modelos

Aspecto	WYGIWYW (LaTeX, Markdown)	WYSIWYG (Word, Writer)
Calidad tipográfica	Excelente, especialmente en matemáticas y textos científicos	Aceptable para uso general, inferior en matemáticas
Curva de aprendizaje	Alta: requiere aprender comandos y sintaxis	Baja: intuitivo para usuarios de computadora
Consistencia de formato	Perfecta: el estilo se define una vez y se aplica globalmente	Propensa a inconsistencias por aplicación manual de formatos
Control de versiones	Nativo: el fuente es texto plano, ideal para Git	Difícil: los formatos binarios o XML complejo no se comparan bien
Colaboración	Posible mediante control de versiones HedgeDoc, HackMD, Overleaf ; herramientas especializadas	Nativa en Google Docs; en Word con funciones de revisión y en LibreOffice mediante Collabora Online
Portabilidad	Alta: el fuente es texto plano, legible en cualquier sistema	Depende del software; los formatos propietarios pueden variar
Matemáticas	Superior: diseñado específicamente para composición matemática	Limitada: los editores de ecuaciones son difíciles de usar
Automatización	Alta: el documento puede generarse programáticamente	Limitada: requiere macros o acceso a la API del programa
Visualización inmediata	No: requiere compilar para ver el resultado	Sí: el resultado es visible en tiempo real
Idoneidad	Tesis, artículos científicos, documentación técnica, libros	Cartas, informes de oficina, documentos con colaboración no técnica
Búsqueda/grep	Trivial (texto plano)	Requiere abrir la app
Longevidad	Décadas garantizadas	Depende de que el software siga existiendo
Tamaño de archivo	KB típico	MB típico (incluye metadata + estilos)
Diff/merge	Línea por línea, natural	Difícil o imposible sin herramientas dedicadas

3.1.5. LaTeX

LaTeX es un sistema de preparación de documentos creado por Leslie Lamport en 1984, basado en el sistema TeX de Donald Knuth (1978). TeX fue diseñado por Knuth cuando quedó insatisfecho con la calidad tipográfica de la segunda edición de su obra *The Art of Computer Programming*. El resultado es un sistema que produce documentos de calidad tipográfica profesional, particularmente en matemáticas.

Un documento LaTeX es un archivo de texto plano con extensión `.tex`. Tiene la siguiente

estructura básica:

```
\documentclass[12pt,a4paper]{article}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[spanish]{babel}
\usepackage{amsmath}

\title{Título del documento}
\author{Nombre del autor}
\date{\today}

\begin{document}

\maketitle

\section{Introducción}
El texto del documento va aquí.

\subsection{Una subsección}
Más texto con una fórmula: $E = mc^2$.

\end{document}
```

El flujo de trabajo es: se escribe el `.tex`, se compila con `pdflatex miarchivo.tex` y se obtiene un `.pdf`.

Los comandos fundamentales de LaTeX son:

Estructura: `\section`, `\subsection`, `\subsubsection`, `\paragraph`.

Texto: `\textbf{}` (negrita), `\textit{}` (cursiva), `\texttt{}` (monoespaciado). El subrayado con `\underline{}` se desaconseja en tipografía moderna – usar `\emph{}` para dar énfasis (se renderiza en cursiva pero respeta el contexto). `\texttt{}` (monoespaciado).

Listas: entornos `itemize` (viñetas), `enumerate` (numerada), `description` (definiciones).

Tablas: entorno `tabular`.

Figuras: entorno `figure` con `\includegraphics`.

Referencias cruzadas: `\label{}` y `\ref{}` para referenciar secciones, figuras y tablas; `\cite{}` para bibliografía.

Bibliografía: entorno `thebibliography` o sistema BibTeX con archivos `.bib`.

Los paquetes amplían las capacidades de LaTeX. Los más utilizados son: `amsmath` y `amssymb` (matemáticas avanzadas), `graphicx` (imágenes), `babel` (idiomas), `geometry` (márgenes), `hyperref` (hipervínculos en el PDF), `listings` (código fuente), `booktabs` (tablas profesionales).

3.1.6. Matemáticas en LaTeX

LaTeX fue diseñado por Donald Knuth con un objetivo prioritario: componer matemática con calidad tipográfica profesional. Sigue siendo insuperable en esa tarea, y es la razón principal por la que la comunidad científica lo adoptó. Antes de escribir cualquier fórmula seria, incluir en el preámbulo:

```
\usepackage{amsmath, amssymb, amsthm, mathtools}
```

Modos matemáticos

LaTeX distingue matemática *en línea* (dentro del párrafo) y *en presentación* (aislada, centrada, opcionalmente numerada):

```
Matemática en línea: la fórmula $a^2 + b^2 = c^2$ va incrustada.

Presentación sin numerar:
\[
  \int_0^1 x^2 \, dx = \frac{1}{3}
\]

Presentación numerada:
\begin{equation}
  e^{i\pi} + 1 = 0
  \label{eq:euler}
\end{equation}

Se puede referenciar con Ecuación~\ref{eq:euler}
o \eqref{eq:euler} que agrega los paréntesis.
```

Los delimitadores viejos

...

funcionan pero se desaconsejan porque no ajustan el espaciado vertical según amsmath y pueden dar problemas de alineación. Usar

...

en su lugar, que se integra mejor con amsmath.

Ecuaciones alineadas y sistemas

Para varias ecuaciones alineadas por un símbolo (típicamente `=`):

```
\begin{align}
  (x+y)^2 &= x^2 + 2xy + y^2 \\
  (x-y)^2 &= x^2 - 2xy + y^2 \\
  x^2 - y^2 &= (x+y)(x-y)
\end{align}

\begin{align*}    % sin numeración
  f(x) &= x^2 \\
  f'(x) &= 2x
\end{align*}
```

El `&` marca el punto de alineación. `align*` no numera; `align` numera cada línea; para numerar solo algunas usar `\nonumber` o `\notag` en las que no se quieran.

Sistemas de ecuaciones con llave:

```
\begin{equation}
  \begin{cases}
    2x + 3y = 5 \\
    x - y = 1
  \end{cases}
\end{equation}
```

Fracciones, raíces, exponentes

```
\frac{a}{b}          \dfrac{a}{b}     % \dfrac fuerza tamaño display
\sqrt{x}              \sqrt[n]{x}      % raíz n-ésima
x^{2n+1}              x_{ij}           % llaves si es más de un carácter
\binom{n}{k}          \dbinom{n}{k}    % coeficiente binomial (\dbinom display)
```

Operadores grandes

```
\sum_{i=1}^{n} i^2
\prod_{k=1}^{\infty} \left(1 - \frac{1}{k^2}\right)
\int_a^b f(x)\, dx
\iint \iiint \oint            % múltiples y de contorno
\lim_{x \to 0} \frac{\sin x}{x}
```

Los delimitadores auto-ajustables `\left(` y `\right)` crecen según su contenido:

```
\left( \frac{a}{b} \right)^n
\left[ \int_0^1 f(x)\,dx \right]
\left| \vec{v} \right|
```

Matrices y arreglos

```
\begin{pmatrix}     % paréntesis ()
  a & b \\
  c & d
\end{pmatrix}

\begin{bmatrix}...\end{bmatrix}    % corchetes []
\begin{vmatrix}...\end{vmatrix}    % barras |...| (determinante)
\begin{Vmatrix}...\end{Vmatrix}    % dobles ||...||
```

Símbolos frecuentes

```
\alpha \beta \gamma \delta \Delta \epsilon \varepsilon
\theta \Theta \lambda \Lambda \mu \pi \Pi \sigma \Sigma
\phi \varphi \psi \Psi \omega \Omega

\leq \geq \neq \approx \equiv \sim \propto
\in \notin \subset \subseteq \supset \cup \cap \emptyset
\rightarrow \Rightarrow \leftrightarrow \Leftrightarrow \mapsto
\forall \exists \nexists \therefore
\nabla \partial \infty \hbar

\mathbb{R} \mathbb{N} \mathbb{Z} \mathbb{Q} \mathbb{C}
\mathcal{F} \mathcal{L}         % script (transformada de Fourier)
\mathfrak{g}                     % fraktur (álgebras de Lie)
\mathbf{v}                       % vector en negrita
\vec{v}                          % vector con flecha
\hat{n} \tilde{f} \bar{x} \dot{q} \ddot{q}
```

Texto dentro de matemática

Dentro del modo matemático, el texto normal se pone con `\text{...}` (requiere `amsmath`):

```
\begin{equation}
  f(x) = \begin{cases}
    x^2 & \text{si } x \geq 0 \\
    -x^2 & \text{si } x < 0
  \end{cases}
```

```
\end{equation}
```

Los operadores con nombre (`sin`, `cos`, `log`, `lim`, `max`, `min`, `exp`, `det`, `tr`, `gcd`...) tienen comandos propios que los formatean en romana:

```
\sin(2\pi x) \quad \log_{10} 100 \quad \lim_{n \to \infty}
```

Para definir uno nuevo: `\DeclareMathOperator{\rank}{rank}`.

Teoremas

Con el paquete `amsthm`:

```
\newtheorem{teorema}{Teorema}[section]
\newtheorem{lema}[teorema]{Lema}
\newtheorem{definicion}{Definición}
\theoremstyle{remark}
\newtheorem{observacion}{Observación}

% Uso:
\begin{teorema}[Pitágoras]
  En un triángulo rectángulo, $a^2 + b^2 = c^2$.
\end{teorema}

\begin{proof}
  ... la demostración ...
\end{proof}
```

3.1.7. BibTeX y BibLaTeX

BibTeX es el sistema tradicional de gestión de bibliografía en LaTeX, creado por Oren Patashnik en 1985. Separa el contenido bibliográfico (una base de datos de referencias en un archivo `.bib`) del estilo de citación (definido por un archivo `.bst`). El mismo archivo `.bib` sirve para muchos documentos y muchos estilos.

BibLaTeX (2006, Philipp Lehman) es el reemplazo moderno, más flexible, con soporte nativo para UTF-8, ordenamiento configurable y múltiples bibliografías por documento. Usa **biber** como procesador en vez de `bibtex`. Para trabajo nuevo, BibLaTeX es lo recomendado.

Archivo `.bib`

Un archivo BibTeX contiene entradas del tipo:

```
@article{einstein1905,
  author  = {Einstein, Albert},
  title   = {Zur Elektrodynamik bewegter K\"orper},
  journal = {Annalen der Physik},
  volume  = {17},
  number  = {10},
  pages   = {891--921},
  year    = {1905},
  doi     = {10.1002/andp.19053221004}
}

@book{knuth1986texbook,
  author    = {Knuth, Donald E.},
  title     = {The {\TeX}book},
  publisher = {Addison-Wesley},
  address   = {Reading, MA},
  year      = {1986},
  isbn      = {0-201-13448-9}
}

@inproceedings{ramirez2019voto,
  author    = {Ram{\'\i}rez Morales, Jorge F. and Saravia Alía, Diego},
  title     = {El eterno retorno del voto electr{\'o}nico},
  booktitle = {48 JAIIO --- Simposio de Inform{\'a}tica en el Estado},
  pages     = {66--72},
  year      = {2019},
  publisher = {SADIO},
  url       = {http://sedici.unlp.edu.ar/handle/10915/89721}
}

@misc{nakamoto2008,
  author = {Nakamoto, Satoshi},
  title  = {Bitcoin: A peer-to-peer electronic cash system},
  year   = {2008},
  url    = {https://bitcoin.org/bitcoin.pdf}
}

@online{eff,
  author       = {{Electronic Frontier Foundation}},
  title        = {Defending your rights in the digital world},
  urldate      = {2026-07-23},
  url          = {https://www.eff.org/}
}
```

Los ejemplos usan escapes como í, Körper y ó por compatibilidad con BibTeX clásico. Con BibLaTeX + biber podés escribir los caracteres UTF-8 directamente: Ramírez, Körper, electrónico.

Cada entrada tiene un **tipo** (`@article`, `@book`, `@inproceedings`, `@techreport`, `@phdthesis`, `@misc`, `@online`, `@inbook`, ...) y una **clave de cita** (la primera palabra, aquí `einstein1905`) que se usa después con `\cite{einstein1905}`.

Uso con BibTeX clásico

En el documento `.tex`:

```
\documentclass{article}
\begin{document}

Como demostró Einstein~\cite{einstein1905}, la simultaneidad
es relativa.

Varias fuentes coinciden~\cite{knuth1986texbook, nakamoto2008}.

\bibliographystyle{plain}     % o alpha, abbrv, acm, apalike, ...
\bibliography{mis_referencias}  % lee mis_referencias.bib

\end{document}
```

El flujo de compilación es de **cuatro pasos**:

```
pdflatex tesis.tex     # primera pasada, genera .aux con las \cite
bibtex tesis           # procesa .aux + .bib, genera .bbl
pdflatex tesis.tex     # incorpora la bibliografía
pdflatex tesis.tex     # resuelve referencias cruzadas
```

Herramientas como `latexmk` automatizan este ciclo: `latexmk -pdf tesis.tex` lo hace todo y solo recompila lo necesario.

Uso con BibLaTeX + biber (recomendado)

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[
  backend=biber,
  style=apa,            % o numeric, alphabetic, authoryear, ieee, ...
  sorting=nyt,          % nombre-año-título
  language=spanish
]{biblatex}
\addbibresource{mis_referencias.bib}

\begin{document}
```

```
En \textcite{einstein1905} se demostró que...
Otros autores han argumentado esto \parencite{knuth1986texbook}.
Cita con página: \parencite[p.~42]{nakamoto2008}.

\printbibliography

\end{document}
```

Se puede eliminar: `\usepackage[utf8]{inputenc}` (desde LaTeX 2018 el default es UTF-8 y con biber es contraproducente).

Compilación:

```
pdflatex tesis.tex
biber tesis
pdflatex tesis.tex
pdflatex tesis.tex
```

Comandos de cita en BibLaTeX

`\cite{clave}` cita según el estilo default.

`\parencite{clave}` cita entre paréntesis: "(Einstein, 1905)".

`\textcite{clave}` cita textual dentro del párrafo: "Einstein (1905) demostró...".

`\footcite{clave}` cita en nota al pie.

`\cite{k1, k2, k3}` múltiples citas en una.

`\cite[p. 42]{clave}` con número de página.

Estilos de citación

BibTeX clásico: `plain`, `alpha`, `abbrv`, `acm`, `ieeetr`, `apalike`, `unsrt` (sin ordenar, en orden de aparición).

BibLaTeX: `numeric`, `alphabetic`, `authoryear`, `authortitle`, `verbose`, `apa`, `chicago-authordate`, `ieee`, `iso-numeric`. La revista destino suele especificar cuál requiere.

Gestores de bibliografía

Escribir a mano las entradas `.bib` es tedioso y propenso a errores. Los **gestores de referencias** extraen los datos directamente del DOI, del ISBN o de la página web del artículo:

Zotero: libre y gratuito, plug-in para navegador que captura la referencia con un clic. Exporta a `.bib` directamente o mantiene el archivo sincronizado con *Better BibTeX*.

JabRef: cliente de escritorio en Java, especializado en editar archivos `.bib`. Libre.

Mendeley: propietario, de Elsevier. Bueno hasta que Elsevier lo compró; hoy es preferible Zotero.

BibDesk: nativo de macOS, libre.

Cualquier artículo con DOI se importa a Zotero desde el navegador con un clic; la entrada `.bib` resultante suele ser correcta y solo hay que revisar caracteres especiales (acentos, `{...}` para preservar mayúsculas en títulos).

Makefile

Una makefile automatiza ejecuciones de comandos en una terminal. Esta makefile es bastante buena para LaTeX ya que decide por si cuando hay que ejecutar cada comando

```
curl -O
    https://raw.githubusercontent.com/tueda/makefile4latex/v0.13.0/Makefile
echo 'BUILDDIR = build' >> latex.mk
make opa.tex
```

si no hay mas de un archivo make solo funciona

Consejos prácticos

- Mantener **un solo archivo** `.bib` para toda la carrera — crecerá con vos y evita duplicados.
- Elegir **claves de cita consistentes**: convención habitual `autor_año_palabraclave` (`einstein_1905_relatividad`).
- Poner los nombres propios y siglas del título entre llaves para que el estilo no las convierta a minúsculas: `title = {The {TeX}book}`.
- Usar `-` para rangos de páginas: `pages = {100-115}`.
- Los caracteres UTF-8 son directos si usás BibLaTeX + biber; con BibTeX clásico requieren escapes: `{\'o}` para ó.

3.1.8. Presentaciones con Beamer

Beamer es una clase de LaTeX para producir presentaciones en formato PDF, equivalente a PowerPoint o Impress pero con toda la potencia tipográfica y matemática de LaTeX. Fue creada por Till Tantau en 2003 y hoy es el estándar de facto para presentaciones académicas en ciencias exactas.

La estructura básica de una presentación Beamer es:

```
\documentclass{beamer}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\usepackage{amsmath}

\usetheme{Madrid}          % tema visual
\usecolortheme{beaver}     % paleta de colores

\title{Título de la charla}
\subtitle{Un subtítulo opcional}
\author{Diego Saravia}
\institute{Universidad Nacional de Salta}
\date{\today}

\begin{document}

\begin{frame}
  \titlepage
\end{frame}

\begin{frame}{Índice}
  \tableofcontents
\end{frame}

\section{Introducción}

\begin{frame}{Motivación}
  El problema que queremos resolver es:
  \begin{itemize}
    \item Primer punto.
    \item Segundo punto.
    \item Con matemática: $E = mc^2$.
  \end{itemize}
\end{frame}

\begin{frame}{Una figura}
  \begin{figure}
    \centering
    \includegraphics[width=0.7\textwidth]{diagrama.pdf}
    \caption{Un diagrama.}
  \end{figure}
\end{frame}

\end{document}
```

Se compila con `pdflatex charla.tex` y se obtiene un PDF donde cada `frame` es una diapositiva.

Elementos característicos

frame: cada diapositiva es un entorno `frame`. El título de la diapositiva va como argumento: `\begin{frame}{Título}`.

Temas: `\usetheme{...}` con opciones como `Madrid`, `Berlin`, `Warsaw`, `Copenhagen`, `default`, `Boadilla`, `metropolis` (moderno, muy usado). Combinar con `\usecolortheme{...}`.

Overlays y transiciones: revelar contenido progresivamente:

```
\begin{frame}{Progresivo}
  \begin{itemize}
    \item<1-> Aparece desde el principio.
    \item<2-> Aparece en la segunda pulsación.
    \item<3-> Aparece en la tercera.
  \end{itemize}
  \pause
  Ese \textbackslash{}pause es equivalente a <2->.
\end{frame}
```

Bloques resaltados: `block`, `alertblock`, `exampleblock`:

```
\begin{block}{Definición}
  Un espacio vectorial es...
\end{block}

\begin{alertblock}{Atención}
  Este teorema requiere axioma de elección.
\end{alertblock}

\begin{exampleblock}{Ejemplo}
  Consideremos $\mathbb{R}^3$...
\end{exampleblock}
```

Columnas: partir la diapositiva verticalmente:

```
\begin{frame}{Comparación}
  \begin{columns}
    \begin{column}{0.5\textwidth}
      Contenido izquierdo.
    \end{column}
    \begin{column}{0.5\textwidth}
      Contenido derecho.
    \end{column}
  \end{columns}
\end{frame}
```

Handout: la opción `\documentclass[handout]{beamer}` colapsa los overlays — útil para imprimir la presentación como PDF sin las transiciones.

Notas del orador: `\note{texto}` agrega notas visibles solo al presentador con `pdfpc` o `Beamer notes`.

Alternativas web

Para presentaciones en HTML/JS existen **reveal.js** (usado por muchos investigadores) e **impress.js**, que permiten publicar la charla en una URL. Son útiles cuando la audiencia necesita ver la presentación en el navegador sin descargar nada. **Quarto** permite generar directamente presentaciones reveal.js desde Markdown con matemática integrada.

3.1.9. Enlaces en LaTeX

Para enlaces clicables en el PDF: `\usepackage{hyperref}` al final del preámbulo (después del resto de packages) – evita conflictos con otros paquetes que redefinen comandos internos.

Después:

```
\url{https://www.gnu.org/}                    % muestra la URL
\href{https://www.gnu.org/}{sitio del GNU}   % texto clicable
\href{mailto:diego@ejemplo.com}{escribime}   % enlace mailto
```

`hyperref` además convierte automáticamente las referencias internas (`\ref`, `\cite`, tabla de contenidos) en enlaces navegables dentro del PDF.

3.1.10. HTML

HTML (*HyperText Markup Language*) es el lenguaje de marcas estándar para crear páginas web. Fue creado por Tim Berners-Lee en 1991 junto con el protocolo HTTP y el concepto de la World Wide Web. La versión actual es HTML5.

Un documento HTML es un archivo de texto con extensión `.html` que el navegador interpreta para mostrar la página. Su estructura básica es:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Título de la página</title>
    <link rel="stylesheet" href="estilo.css">
  </head>
  <body>
    <h1>Encabezado principal</h1>
```

```
 <p>Un párrafo de texto con <strong>importante</strong>              y <em>énfasis</em>

    <h2>Una lista</h2>
    <ul>
      <li>Primer ítem</li>
      <li>Segundo ítem</li>
    </ul>

    <a href="https://www.gnu.org">Enlace al sitio GNU</a>

    <img src="imagen.png" alt="Descripción de la imagen">
  </body>
</html>
```

Los elementos HTML fundamentales son:

Estructura del documento: `<html>`, `<head>`, `<body>`, `<meta>`, `<title>`.

Encabezados: `<h1>` a `<h6>`.

Texto: `<p>` (párrafo), `<strong>` (negrita semántica), `<em>` (énfasis/cursiva), `<code>` (código en línea), `<pre>` (texto preformateado).

Listas: `<ul>` (no ordenada), `<ol>` (ordenada), `<li>` (ítem), `<dl>`, `<dt>`, `<dd>` (lista de definiciones).

Hipervínculos: `<a href='URL'>texto</a>`.

Multimedia: `<img>`, `<video>`, `<audio>`.

Tablas: `<table>`, `<tr>`, `<th>`, `<td>`.

Semántica HTML5: `<header>`, `<nav>`, `<main>`, `<article>`, `<section>`, `<footer>`.

Formularios: `<form>`, `<input>`, `<button>`, `<select>`, `<textarea>`.

HTML define la *estructura* del contenido. El aspecto visual se define con **CSS** (*Cascading Style Sheets*) y el comportamiento interactivo con **JavaScript**.

Para matemáticas en entornos web, el estándar es **MathJax** o **KaTeX**: bibliotecas JavaScript que interpretan notación LaTeX y la renderizan en el navegador. Esto permite usar la misma notación matemática de LaTeX en páginas web y documentos Jupyter.

3.1.11. XML

XML (*eXtensible Markup Language*) es un metalenguaje de marcas definido por el W3C en 1998. A diferencia de HTML, que tiene un conjunto fijo de etiquetas, XML permite definir cualquier conjunto de etiquetas según las necesidades de la aplicación.

XML es un formato estándar para el intercambio de datos entre sistemas. Muchos formatos de archivos modernos están basados en XML: el formato DOCX de Microsoft Word,

el formato ODT de LibreOffice, el formato SVG para gráficos vectoriales y el formato XHTML son todos XML.

Un documento XML tiene la siguiente estructura:

```
<?xml version="1.0" encoding="UTF-8"?>
<biblioteca>
  <libro id="001">
    <titulo>El nombre de la rosa</titulo>
    <autor>
      <nombre>Umberto</nombre>
      <apellido>Eco</apellido>
    </autor>
    <año>1980</año>
    <editorial>Lumen</editorial>
  </libro>
  <libro id="002">
    <titulo>Ficciones</titulo>
    <autor>
      <nombre>Jorge Luis</nombre>
      <apellido>Borges</apellido>
    </autor>
    <año>1944</año>
    <editorial>Sur</editorial>
  </libro>
</biblioteca>
```

Las reglas fundamentales de XML son:

- Todo documento debe tener exactamente un elemento raíz.
- Toda etiqueta de apertura debe tener su etiqueta de cierre correspondiente, o ser una etiqueta vacía con barra al final: `<br/>`.
- Las etiquetas deben estar correctamente anidadas: si se abrió `<a>` dentro de `<b>`, debe cerrarse `</a>` antes de `</b>`.
- Los valores de los atributos siempre van entre comillas.
- XML distingue mayúsculas de minúsculas.

Un documento que cumple estas reglas se dice *bien formado* (*well-formed*). Un documento es *válido* si además cumple las reglas de una gramática definida en un DTD o un esquema XSD.

Para procesar XML desde la línea de comandos se usa `xmllint`:

```
xmllint --format archivo.xml        # formatea el XML
xmllint --valid archivo.xml         # valida contra el DTD declarado
xmllint --xpath "//titulo" arch.xml # extrae nodos con XPath
```

El formato **MathML** es un estándar XML del W3C para representar matemáticas en la web, aunque en la práctica es generado automáticamente por herramientas y rara vez escrito a mano.

3.1.12. PDF

PDF (*Portable Document Format*) es un formato de archivo creado por Adobe Systems en 1993 y convertido en estándar ISO en 2008 (ISO 32000). Su objetivo es representar documentos de forma independiente del software, el hardware y el sistema operativo con el que fueron creados.

Un archivo PDF encapsula una descripción completa del documento: fuentes (incrustadas), imágenes, gráficos vectoriales, texto y las instrucciones de disposición en cada página. El resultado es idéntico en cualquier visor y en cualquier impresora.

El PDF tiene dos variantes importantes para uso académico:

PDF/A: estándar para archivado a largo plazo. Prohíbe elementos que dependen de software externo (multimedia, JavaScript, cifrado) y exige que todas las fuentes estén incrustadas.

PDF/UA: estándar para accesibilidad. Requiere que el PDF esté etiquetado de forma que los lectores de pantalla puedan interpretar correctamente la estructura del documento.

En GNU/Linux, el PDF puede manipularse con varias herramientas:

```
pdflatex documento.tex        # genera PDF desde LaTeX
pdftk a.pdf b.pdf cat output combinado.pdf  # combina PDFs
pdftotext documento.pdf       # extrae texto
convert -density 300 doc.pdf imagen.png     # convierte a imagen
qpdf --decrypt entrada.pdf salida.pdf       # elimina contraseña
gs -dNOPAUSE -sDEVICE=pdfwrite \
   -sOUTPUTFILE=comprimido.pdf \
   -dBATCH entrada.pdf        # comprime con Ghostscript
```

Se puede firmar un documento PDF (pdfsig, signature), lo que es relevante para trámites/informes; y se puede usar PDF/X para usar con una imprenta profesional.

3.1.13. Markdown

Markdown es un lenguaje de marcas ligero creado por John Gruber en 2004, diseñado con un objetivo específico: que el texto fuente sea legible como texto plano, incluso antes de ser procesado. A diferencia de HTML o LaTeX, un archivo Markdown es cómodo de leer directamente en un editor de texto.

La sintaxis básica de Markdown es:

````
# Encabezado 1
## Encabezado 2
### Encabezado 3

Párrafo normal. **negrita** y *cursiva* y `código en línea`.

- Ítem de lista no ordenada
- Otro ítem
  - Ítem anidado

1. Lista numerada
2. Segundo ítem

[Texto del enlace](https://www.ejemplo.com)

![Texto alternativo](imagen.png)

> Texto de una cita en bloque.

```python
def saludo(nombre):
    print(f"Hola, {nombre}")
```

| Columna 1 | Columna 2 |
|-----------|-----------|
| Celda 1   | Celda 2   |

---
````

Markdown tiene múltiples variantes (sabores): CommonMark (especificación estandarizada), GitHub Flavored Markdown (GFM, usado en GitHub y GitLab), Pandoc Markdown (con extensiones para tablas, notas al pie, matemáticas y más).

La herramienta más potente para trabajar con Markdown es **Pandoc**, que convierte entre una enorme cantidad de formatos:

```
pandoc informe.md -o informe.pdf
pandoc informe.md -o informe.docx
pandoc informe.md -o informe.html
pandoc informe.tex -o informe.md
pandoc informe.docx -o informe.md
```

Markdown es el formato estándar para la documentación de proyectos de software (archivos `README.md` en GitHub), para blogs estáticos (Jekyll, Hugo), para cuadernos Jupyter y para tomar notas técnicas.

Existe también MyST (Markdown científico con LaTeX embebido), estándar emergente en documentación técnica y proyectos que combinan texto con notación matemática.

3.1.14. Conversión de formatos

La posibilidad de convertir entre formatos es fundamental para la interoperabilidad y la longevidad de los documentos. Las herramientas más importantes son:

Pandoc: la navaja suiza de la conversión de documentos. Convierte entre Markdown, LaTeX, HTML, DOCX, ODT, EPUB, reStructuredText, PDF y docenas de formatos más.

LibreOffice en modo línea de comandos: puede convertir documentos de Office a PDF y otros formatos:

```
libreoffice --headless --convert-to pdf documento.docx
libreoffice --headless --convert-to odt documento.docx
```

ImageMagick (`convert`): convierte entre formatos de imagen y puede rasterizar PDFs.

Ghostscript (`gs`): manipula y convierte archivos PostScript y PDF.

Calibre (`ebook-convert`): convierte entre formatos de libros electrónicos: EPUB, MOBI, PDF, HTML.

html2text, lynx -dump: convierten HTML a texto plano.

La conversión de formatos rara vez es perfecta: cada formato tiene capacidades distintas y algunos elementos no tienen equivalente en el formato destino. Por ejemplo, al convertir un documento LaTeX complejo a DOCX se perderán las ecuaciones matemáticas avanzadas. Al convertir de DOCX a Markdown se perderán las personalizaciones de estilo finas.

3.1.15. Planilla de cálculo

Una **planilla de cálculo** o hoja de cálculo es una aplicación organizada en una cuadrícula de filas y columnas. Cada celda puede contener un valor (número, texto, fecha) o una **fórmula** que calcula su valor a partir de otras celdas. Cuando se modifica una celda, todas las celdas que dependen de ella se recalculan automáticamente.

Las planillas de cálculo más utilizadas son Microsoft Excel, LibreOffice Calc y Google Sheets. El formato estándar de intercambio es `.xlsx` (basado en XML), que puede abrirse en todas las aplicaciones mencionadas.

Un archivo CSV bajo Git es diffable línea a línea; las planillas binarias (.xlsx, .ods) no. Esto refuerza la preferencia por texto plano cuando el seguimiento de cambios importa. Por diffable se entiende la posibilidad de usar `diff`, lo cual se dificulta en archivos que tienen codificación compleja en particular binaria.

Fórmulas y funciones

Una fórmula comienza con el signo `=` y puede contener operadores aritméticos, referencias a celdas y llamadas a funciones:

```
=A1+B1           suma el contenido de A1 y B1
=A1*B1           producto
=(A1+A2)/2       promedio manual de dos celdas
=SUMA(A1:A10)    suma el rango A1 a A10
=PROMEDIO(B1:B20)
=MAX(C1:C100)
=MIN(C1:C100)
=CONTAR(A1:A50)  cuenta las celdas con números
=CONTARA(A1:A50) cuenta las celdas no vacías
=SI(A1>0,"positivo","no positivo")
=BUSCARV(valor,tabla,columna,0)
```

Las referencias a celdas pueden ser:

Relativas: `A1`. Se ajustan al copiar la fórmula a otra celda.

Absolutas: `$A$1`. No cambian al copiar.

Mixtas: `$A1` o `A$1`. Solo se fija la fila o la columna.

Gráficos

Las planillas de cálculo pueden generar gráficos automáticamente a partir de los datos. Los tipos más comunes son:

Gráfico de líneas: evolución de valores a lo largo del tiempo o de otra variable continua.

Gráfico de barras / columnas: comparación de valores entre categorías.

Gráfico de dispersión (XY): representa pares de valores (x, y). Fundamental para visualizar relaciones entre variables.

Gráfico de sectores (torta): proporciones de un total.

Histograma: distribución de frecuencias.

Importación de datos

Las planillas pueden importar datos de múltiples fuentes:

CSV (*Comma Separated Values*): formato de texto plano donde cada línea es una fila y los campos están separados por comas (o punto y coma, o tabulaciones). Es el formato más universal para intercambio de datos tabulares.

Bases de datos: conexión directa a bases SQL para importar los resultados de consultas.

Servicios web: Google Sheets puede importar datos desde URLs con la función `=IMPORTDATA()`.

Desde la terminal, los archivos CSV pueden procesarse con `csvkit`, `awk` o `python`:

```
csvstat datos.csv          # estadísticas del CSV
csvcut -c 1,3 datos.csv    # selecciona columnas 1 y 3
csvsort -c 2 datos.csv     # ordena por columna 2
```

Ajuste de curvas

El ajuste de curvas (*curve fitting*) consiste en encontrar la función matemática que mejor describe un conjunto de datos experimentales. Las planillas de cálculo ofrecen ajuste automático en los gráficos de dispersión: al añadir una *línea de tendencia*, se puede elegir el tipo de ajuste (lineal, polinomial, exponencial, logarítmico, potencial) y mostrar la ecuación y el coeficiente de determinación R^2.

Un R^2 cercano a 1 indica que el modelo explica bien la variación de los datos; un R^2 cercano a 0 indica que el modelo no es adecuado.

Para ajustes más sofisticados se usan herramientas especializadas: Scilab (`regress`, `polyfit`), Python (NumPy, SciPy) o R.

3.1.16. Gráficos vectoriales con Inkscape

Inkscape es un editor de gráficos vectoriales libre y multiplataforma, equivalente a Adobe Illustrator o CorelDraw. A diferencia de los editores de mapa de bits (GIMP, Photoshop), que trabajan con píxeles, Inkscape trabaja con **trayectorias matemáticas**: los dibujos se escalan a cualquier tamaño sin pérdida de calidad. Su formato nativo es **SVG** (*Scalable Vector Graphics*), estándar del W3C.

Cuándo usar vectorial vs. bitmap

- **Vectorial (Inkscape, SVG):** diagramas, esquemas, logos, íconos, gráficos científicos, ilustraciones geométricas — todo lo que se define por líneas, formas y texto.
- **Bitmap (GIMP, PNG/JPG):** fotografías, capturas de pantalla, texturas y cualquier imagen con degradados complejos.

Un diagrama de un circuito hecho en Inkscape se ve nítido tanto en un póster A0 como en una diapositiva; el mismo diagrama escaneado como PNG se pixela al ampliarlo.

Herramientas básicas

Al abrir Inkscape, la barra izquierda contiene las herramientas esenciales:

Selección (`S`): mover, escalar, rotar objetos.

Rectángulos (`R`), círculos (`E`), estrellas (`*`): formas primitivas.

Bezier / pluma (`B`): dibujar trayectorias arbitrarias clic a clic; doble clic termina.

Nodos (N): editar los puntos de una trayectoria existente — fundamental para retocar cualquier forma.

Texto (T): agregar rótulos.

Cuentagotas (F7): copiar el color de un objeto a otro.

Zoom (Z): o con +/- y rueda del ratón + Ctrl.

Flujo típico para un diagrama científico

1. **Configurar el documento:** Archivo → Propiedades del documento para elegir tamaño (A4, A5) y orientación.
2. **Dibujar componentes:** usar rectángulos, círculos y flechas para los bloques del diagrama.
3. **Alinear:** Objeto → Alinear y distribuir (Shift+Ctrl+A) — alinea bordes o distribuye equitativamente objetos seleccionados. Es lo que separa un diagrama profesional de uno torcido.
4. **Agrupar:** Ctrl+G agrupa; Ctrl+Shift+G desagrupa. Un diagrama complejo debe estar organizado en grupos.
5. **Capas:** Capa → Capa nueva (Shift+Ctrl+N). Poner fondo, elementos y rótulos en capas separadas facilita editar sin romper todo.
6. **Guardar como SVG:** formato editable.
7. **Exportar a PNG:** Archivo → Exportar para uso web o embebido en documentos que no soportan SVG.
8. **Exportar a PDF:** Archivo → Guardar como → PDF. Es lo ideal para incluir en LaTeX con \includegraphics.

Integración con LaTeX

Tres opciones, de menor a mayor sofisticación:

PDF directo: \includegraphics{fig.pdf} con el PDF exportado desde Inkscape. Simple y funciona siempre.

SVG con svg package: \usepackage{svg} más \includesvg{fig.svg} convierte automáticamente al compilar (requiere Inkscape instalado en el sistema que compila).

PDF+LaTeX (texto nativo): al exportar como PDF marcar "Omitir texto en PDF y crear archivo LaTeX". Inkscape genera fig.pdf (dibujo) y fig.pdf_tex (texto). Se incluye con:

```
\input{fig.pdf_tex}
```

El texto queda tipeado con la fuente y las fórmulas del propio LaTeX, lo que da un resultado impecable en publicaciones científicas.

Alternativas libres

- **Krita:** orientado a ilustración digital y pintura; también hace vectorial pero es su rol secundario.
- **LibreOffice Draw:** más simple, integrado con LO — útil para diagramas rápidos si ya tenés LibreOffice.
- **TikZ / PGF** (dentro de LaTeX): programación de diagramas como código, sin GUI. Mayor curva pero ideal cuando el diagrama es lógico (grafos, árboles, autómatas).
- **draw.io / diagrams.net:** web, especializado en diagramas de flujo, UML y arquitectura.

Para el trabajo científico general, la combinación **Inkscape (dibujo) + LaTeX (documento) + PDF (distribución)** cubre todos los casos con calidad profesional y usando exclusivamente software libre.

3.1.17. Edición de imágenes con GIMP

GIMP (*GNU Image Manipulation Program*) es el editor de imágenes de mapa de bits (raster) libre y multiplataforma, equivalente a Adobe Photoshop. Trabaja con **píxeles**, a diferencia de Inkscape que trabaja con vectores. Está pensado para retoque fotográfico, composición, ajuste de color, capturas de pantalla y cualquier imagen que no se pueda describir con trayectorias geométricas.

Formato nativo: **XCF** (preserva capas, canales, máscaras). Exporta a PNG, JPG, TIFF, WebP y decenas de otros.

Cuándo usar GIMP y cuándo Inkscape

- **GIMP:** fotografías, capturas de pantalla, texturas, retoque, edición de PNG/JPG existentes.
- **Inkscape:** diagramas, esquemas, logos, ilustraciones geométricas donde la escalabilidad importa.

Regla práctica: si al ampliar la imagen 10× querés que siga nítida, usá Inkscape. Si estás partiendo de una foto o captura, usá GIMP.

Herramientas básicas

Selección rectangular (`R`), elíptica (`E`), libre (`F`): delimitar áreas para copiar, recortar o aplicar efectos solo a esa zona.

Selección por color (`Shift+O`) / varita mágica (`U`): seleccionar píxeles similares.

Mover (`M`): mover la selección o capa activa.

Recortar (`Shift+C`): cortar la imagen al rectángulo elegido.

Pincel (`P`), lápiz (`N`), aerógrafo (`A`), goma (`Shift+E`): pintar.

Cuentagotas (`O`): tomar color de la imagen.

Texto (`T`): agregar rótulos.

Clonar (`C`): copiar región de la imagen a otra parte (retoque de imperfecciones).

Zoom (`+`/`-`): o `Shift+Ctrl+J` para encajar a ventana.

Capas: el concepto clave

Toda imagen no trivial se construye con **capas** apiladas (`Ventanas → Diálogos empotrables → Capas`, o `Ctrl+L`):

- Cada capa se edita de forma independiente sin afectar las otras.
- Se pueden mover, rotar, cambiar opacidad y modo de fusión (multiplicar, pantalla, superponer...).
- Las **máscaras de capa** permiten ocultar partes de una capa sin borrar píxeles — reversible.
- El orden importa: la de arriba tapa la de abajo (según opacidad).

Trabajar sin capas es lo que separa una edición amateur de una profesional.

Operaciones frecuentes

Cambiar tamaño: `Imagen → Escalar la imagen`. Reduce o amplía en píxeles; ampliar mucho pixela (limitación intrínseca del raster).

Recortar: `Imagen → Recortar al contenido` o usar la herramienta de recorte.

Ajuste de color: `Colores → Brillo/Contraste`, `Curvas`, `Niveles`, `Balance de color`, `Tono/Saturación`. `Niveles` es el más útil para arreglar fotos mal expuestas.

Convertir a escala de grises: `Imagen → Modo → Escala de grises`.

Rotar / voltear: `Imagen → Transformar`.

Filtros: menú `Filtros` — desenfoque, ruido, realzado, artísticos. `Filtros → Realzar → Enfocar (unsharp mask)` es el estándar para dar nitidez tras redimensionar.

Eliminar fondo: combinar selección por color + `Capa → Transparencia → Añadir canal alfa` + `Supr`.

Exportar: importante: `Archivo → Guardar` guarda en formato XCF (interno). Para PNG/JPG usar `Archivo → Exportar como` (`Shift+Ctrl+E`). Esta distinción confunde a muchos principiantes.

Formatos de salida: cuál elegir

- **PNG:** compresión sin pérdida, soporta transparencia, ideal para capturas de pantalla, diagramas y web con transparencia.
- **JPG:** compresión con pérdida, sin transparencia, ideal para fotografías (archivos mucho más chicos).
- **WebP:** más moderno, mejor compresión que ambos, soportado por navegadores actuales.
- **TIFF:** sin pérdida, muy pesado, usado en imprenta profesional.
- **XCF:** solo GIMP. Guardar acá el "fuente" con capas para poder seguir editando; exportar a otro formato para uso final.

GIMP desde la línea de comandos

Para tareas repetitivas (redimensionar 100 fotos, convertir formatos en lote), suele ser más eficiente usar **ImageMagick** en la terminal:

```
convert foto.jpg -resize 800x600 foto_chica.jpg
convert *.png -quality 85 -format jpg out.jpg
mogrify -resize 50% *.png       # in-place
identify foto.jpg               # dimensiones y metadata
```

GIMP también acepta scripting en Script-Fu (dialecto de Lisp) o Python-Fu para automatización, pero para uso ocasional ImageMagick alcanza y sobra.

Alternativas libres

- **Krita:** orientado a ilustración digital y pintura (más pinceles, mejor para dibujar que para retocar fotos).
- **darktable** / **RawTherapee:** especializados en fotografía RAW (revelado no destructivo). Complementan a GIMP.
- **Pinta:** clon simplificado de Paint.NET, para tareas triviales.

Para el trabajo científico, la combinación **GIMP (raster)** + **Inkscape (vectorial)** + **LaTeX (documento)** cubre casi todas las necesidades gráficas de un informe o publicación.

3.2. Estándares abiertos y cerrados

Un **formato abierto** tiene su especificación publicada libremente y puede implementarse sin permiso ni pago. Un **formato cerrado** es propiedad de una empresa que controla quién puede leerlo o escribirlo. La distinción es crítica para la libertad, la interoperabilidad y la longevidad de los documentos.

Formato	Abierto/Cerrado	Organismo/Empresa
HTML, CSS	abierto	W3C
PDF, PDF/A	abierto	ISO 32000
ODF (ODT, ODS)	abierto	OASIS, ISO 26300
OOXML (DOCX)	abierto pero de facto controlado	Microsoft / ISO 29500
DOC (Word 97)	cerrado (ingeniería inversa)	Microsoft
XML, JSON	abierto	W3C, IETF
Markdown	abierto	CommonMark
EPUB	abierto	IDPF/W3C
LaTeX	abierto	LaTeX Project

La preferencia por formatos abiertos no es capricho ideológico: un formato cerrado deja los datos rehenes del proveedor. Cuando Microsoft dejó de soportar el viejo formato `.wri` (Windows Write, años 90), miles de documentos personales se volvieron inaccesibles. Los archivos `.txt` de esa misma época se leen perfectamente hoy.

3.3. Formatos de documentos científicos

Durante el desarrollo de la carrera los estudiantes producirán informes, artículos, presentaciones, manuales, páginas web y documentación técnica. Actualmente existen numerosos formatos para representar información, cada uno con ventajas y limitaciones según el uso previsto.

En la práctica profesional es habitual combinar varios formatos dentro de un mismo proyecto. Por ejemplo, un informe puede escribirse en LaTeX, publicarse como PDF, resumirse en una página HTML y mantenerse bajo control de versiones mediante archivos Markdown.

La Tabla 3.1 resume algunas de las características de los formatos de documentos más utilizados en la actualidad.

Otros formatos:

- AsciiDoc: alternativa a Markdown más rica, usado por O'Reilly
- RTF: histórico pero todavía aparece

3.3.1. Documentos reproducibles

Un **documento reproducible** integra en un único archivo el texto, el código que realiza el análisis y los resultados (tablas, gráficos, estadísticas). Cuando los datos cambian o se

Formato	Editable	Matemática	Aplicación principal
TXT	Sí	No	Texto plano, archivos de configuración, programas fuente y almacenamiento de información simple.
Markdown	Sí	Básica (LaTeX)	Documentación técnica, archivos README, repositorios Git, manuales y páginas web estáticas.
HTML	Sí	Mediante MathJax u otras bibliotecas	Publicación de documentos y páginas web accesibles desde cualquier navegador.
ODT (LibreOffice)	Sí	Sí	Documentos de oficina, informes, cartas, trabajos prácticos y tesis con edición visual WYSIWYG.
docx (Word de Office)	Sí	Sí	Documentos de oficina, informes, cartas, trabajos prácticos y tesis con edición visual WYSIWYG.
LaTeX	Sí	Excelente	Artículos científicos, libros, tesis, informes técnicos y documentos con gran cantidad de fórmulas matemáticas.
PDF	No (normalmente)	Sí	Distribución, impresión y preservación de documentos conservando exactamente su formato original.

Cuadro 3.1: Comparación entre algunos formatos de documentos de uso habitual.

corrige el código, el documento se regenera automáticamente con los resultados actualizados.

El concepto fue formalizado por Donald Knuth como **programación literaria** (*literate programming*) en 1984: el código y su documentación se escriben juntos, como un texto coherente dirigido tanto a humanos como a computadoras.

Las herramientas modernas de documentos reproducibles son:

Jupyter Notebook / JupyterLab: cuadernos interactivos que combinan celdas de texto en Markdown y celdas de código ejecutable en Python, R, Julia u otros lenguajes. Al ejecutar el cuaderno, los resultados (incluyendo gráficos) aparecen embebidos en el documento. El archivo `.ipynb` es JSON y puede convertirse a HTML, PDF o diapositivas con `nbconvert`.

R Markdown / Quarto: sistema similar a Jupyter pero integrado en el ecosistema de R. Los bloques de código R se ejecutan durante la compilación del documento y sus resultados se insertan en el PDF o HTML de salida. Quarto es la evolución moderna y soporta Python, R, Julia y Observable.

Sweave y knitr: integración de código R directamente en documentos LaTeX. Los blo-

ques de código R (llamados *chunks*) se ejecutan durante la compilación de LaTeX y sus resultados, tablas y figuras se insertan automáticamente.

Org-mode con Babel: permite embeber y ejecutar código en docenas de lenguajes dentro de documentos Org-mode en emacs.

También existen Binder y Google Colab como plataformas donde correr un notebook sin instalar nada.

La reproducibilidad tiene tres niveles:

Reproducible computacionalmente: con el mismo código y los mismos datos, cualquier persona puede obtener exactamente los mismos resultados.

Replicable: con métodos similares y datos independientes, se obtienen resultados consistentes.

Robusto: los resultados son consistentes ante variaciones razonables en los métodos o los datos.

Los documentos reproducibles son fundamentales en la ciencia abierta: un artículo cuyo análisis viene acompañado de un cuaderno Jupyter ejecutable permite a cualquier lector verificar cada cálculo, modificar los parámetros y explorar los datos directamente, en lugar de tener que confiar ciegamente en los resultados publicados.

Para garantizar la reproducibilidad a largo plazo es necesario además fijar las versiones del software utilizado. Herramientas como `conda`, `virtualenv` (Python), `renv` (R) y contenedores Docker permiten encapsular el entorno de software completo junto con el documento.

Otras herramientas no mencionadas que se pueden necesitar:

- Overleaf / TeXworks / TeXstudio Editores LaTeX
- Grammarly, LanguageTool — corrección gramatical
- Diff/merge para documentos (diff, wdiff, latexdiff), se usan mucho con subversion o git.
- TikZ / PGF — diagramas dentro de LaTeX
- Graphviz, diagramas como texto a imagen

3.3.2. Reproducibilidad: el problema de la publicación científica

La **crisis de reproducibilidad** es un problema documentado en múltiples disciplinas científicas: un porcentaje significativo de los resultados publicados en revistas de prestigio no puede ser reproducido por otros investigadores.

Las causas son múltiples: errores en el análisis estadístico, sesgos en la publicación (se publican resultados positivos pero no los negativos), manipulación inconsciente de los

datos para obtener resultados significativos (*p-hacking*), y falta de acceso al código y los datos originales.

Los documentos reproducibles no resuelven todos estos problemas, pero abordan directamente el último: si el código y los datos se publican junto con el artículo, los lectores pueden verificar exactamente qué se hizo y detectar errores. Plataformas como **Zenodo**, **OSF** (*Open Science Framework*) y **Figshare** permiten publicar datos y código con un DOI citable, igual que un artículo.

El estándar emergente en varias disciplinas es el **artículo ejecutable** (*executable article*): un documento que combina el texto del artículo, el código de análisis y los datos en un contenedor que cualquier lector puede ejecutar en la nube con un clic, sin instalar nada localmente.

Cada uno de estos formatos fue desarrollado para resolver necesidades diferentes. En consecuencia, no existe un formato que sea el mejor para todas las aplicaciones. La elección dependerá del tipo de documento, del público al que está destinado y del proceso de trabajo utilizado.

Dos problemas transversales cruzan la elección de formato. El primero es la **longevidad digital**: los archivos de texto plano producidos hace 40 años se leen perfectamente hoy, mientras que un `.doc` de 1998 muchas veces no abre en Word actual. El segundo es la **accesibilidad**: los estándares WCAG (para HTML) y PDF/UA (para PDF) especifican requisitos de etiquetado para que lectores de pantalla puedan interpretar el documento correctamente.

3.3.3. Guía rápida: qué formato usar

Habiendo revisado los formatos principales, la Tabla 3.2 sintetiza qué herramienta conviene según la tarea. La regla general es: **formato de texto plano siempre que sea posible**, formatos binarios solo cuando el destino final lo exija (impresión, distribución cerrada).

Esta guía no es dogma: en cada caso pesan también las restricciones del entorno (¿los colaboradores usan Word?, ¿la revista pide Word?), la infraestructura disponible y el tiempo de aprendizaje. Pero cuando podés elegir, elegí texto plano.

3.4. Método de cuadrados mínimos

Así como estudiamos la edición de textos y la creación de planillas de cálculo, el práctico pide resolver un problema de cuadrados mínimos. Con ese objetivo se proporcionan estos apuntes teóricos. La parte gráfica se cubrirá con gnuplot más adelante.

En muchas experiencias de laboratorio es necesario ajustar una función matemática a un conjunto de datos experimentales.

Uno de los métodos más utilizados es el método de los cuadrados mínimos, que permite obtener la recta que mejor representa los datos medidos.

Tarea	**Formato recomendado**
Tesis o artículo con fórmulas	LaTeX
Documentación de código	Markdown
Página web propia	HTML + CSS
Datos tabulares	CSV
Configuración de programas	YAML o JSON
Distribución final del documento	PDF
Colaboración en tiempo real	Google Docs, HedgeDoc u Overleaf
Presentación con matemática	Beamer (LaTeX)
Análisis de datos reproducible	Jupyter, R Markdown, Quarto
Nota rápida personal	Markdown en editor plano
Diagramas técnicos versionables	TikZ, Graphviz, PlantUML, Mermaid
Libro electrónico	EPUB
Bibliografía citable	BibTeX / BibLaTeX (`.bib`)

Cuadro 3.2: Formato recomendado según el tipo de tarea.

En el ajuste de curvas en 2D se dan n pares de puntos (parejas de números)

$$(x_1, y_1), \ldots, (x_n, y_n)$$

y se desea determinar una función $f(x)$ tal que

$$f(x_j) \approx y_j, \qquad j = 1, \ldots, n.$$

El tipo de función (por ejemplo polinomios, funciones exponenciales, funciones senoidales y cosenoidales) puede ser sugerida por la naturaleza del problema (la ley física) y, en muchos casos, resultará apropiado un polinomio de cierto grado.

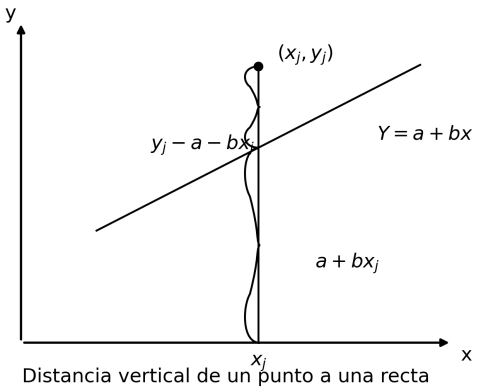
La recta $y = a + bx$ debe ajustarse por los puntos dados $(x_1, y_1), \ldots, (x_n, y_n)$ de modo que la suma de los cuadrados de las distancias de esos puntos a la recta sea mínima, en donde la distancia se mide en la dirección vertical (la dirección y).

El punto sobre la recta con abscisa x_j tiene la ordenada $a + bx_j$. De donde su distancia a (x_j, y_j) es $|y_j - a - bx_j|$ y esa suma de cuadrados es:

$$q = \sum_{j=1}^{n} (y_j - a - bx_j)^2$$

q depende de a y b. Una condición necesaria para que q sea mínima es que

$$\frac{\partial q}{\partial a} = -2 \sum (y_j - a - bx_j) = 0,$$
$$\frac{\partial q}{\partial b} = -2 \sum x_j (y_j - a - bx_j) = 0.$$



Distancia vertical de un punto a una recta

(en donde se suma j desde 1 hasta n). Por lo tanto,

$$an + b\sum x_j = \sum y_j,$$
$$a\sum x_j + b\sum x_j^2 = \sum x_j y_j.$$

Estas ecuaciones reciben el nombre de ecuaciones normales del problema propuesto. De las ecuaciones anteriores:

$$a = \frac{\sum y_j - b\sum x_j}{n},$$
$$b = \frac{n\sum x_j y_j - \sum x_j \sum y_j}{n\sum x_j^2 - (\sum x_j)^2}.$$

3.5. Control de versiones

El **control de versiones** registra la historia de los cambios en el código: quién cambió qué, cuándo y por qué. Permite volver a cualquier versión anterior, trabajar en paralelo en distintas líneas de desarrollo y colaborar sin sobrescribir el trabajo de nadie.

Reemplaza una práctica que todos conocemos y que no funciona: los archivos llamados `informe_final`, `informe_final_2`, `informe_final_definitivo_corregido`. Lo que falla ahí no es el desorden sino la pérdida de información: esos nombres no dicen qué cambió entre uno y otro, ni por qué, ni cuál es el bueno.

Los conceptos, independientes de la herramienta que se use, son:

Repositorio: el directorio del proyecto más la base de datos con toda su historia.

Versión o revisión: un estado completo del proyecto en un momento dado, identificado de manera unívoca. No es un archivo sino una foto del conjunto.

Confirmar (*commit*): registrar un conjunto de cambios como una nueva versión, acompañado de un mensaje. El mensaje debe explicar *por qué* se hizo el cambio; qué archivos se tocaron ya lo registra el sistema.

Actualizar (*update*): traer al espacio de trabajo local los cambios que otros incorporaron.

Diferencia (*diff*): la comparación entre dos versiones, que muestra exactamente qué líneas cambiaron.

Rama: una línea de desarrollo paralela, que permite trabajar en algo nuevo sin afectar la versión que funciona.

Fusión (*merge*): la incorporación de los cambios de una rama en otra.

Conflicto: la situación en que dos personas modificaron las mismas líneas en paralelo. El sistema no puede decidir cuál vale y pide que se resuelva a mano. No es una falla de la herramienta: es información que sólo las personas involucradas tienen.

Hay dos arquitecturas. En los sistemas **centralizados** existe un único repositorio en un servidor y las copias locales son sólo espacios de trabajo; hace falta conexión para confirmar y ver la historia. En los **distribuidos**, cada copia local es un repositorio completo con toda la historia, se puede trabajar y confirmar sin conexión, y los repositorios se sincronizan entre sí.

Un consejo que se repite y conviene tomar en serio: confirmar seguido y con cambios chicos y coherentes. Una versión que contiene un solo cambio con sentido es fácil de entender, de revertir y de trasladar a otra rama. Una que contiene tres semanas de trabajo no sirve para nada de eso.

Por último, no todo va al repositorio. Los archivos generados –resultados intermedios, ejecutables, figuras producidas por el programa– se pueden volver a generar y sólo ensucian la historia; lo que va es lo que no se puede reconstruir: el código, los datos primarios y la documentación.

3.5.1. Control de versiones con Git

La teoría presentó los conceptos sin herramienta: repositorio, versión, confirmar, actualizar, diferencia, rama, fusión y conflicto, más la distinción entre arquitectura centralizada y distribuida. Git es una implementación distribuida, y hoy la más usada Git Project 2026; Chacon y Straub 2014.

Nació de una necesidad concreta: coordinar a mucha gente trabajando en paralelo sobre el mismo código sin que nadie tuviera que pedir permiso ni estar conectado Torvalds 2007. De ahí sus dos decisiones centrales: cada copia local es un repositorio completo con toda la historia, y las ramas son baratas.

Instalación y configuración

```
sudo apt update
sudo apt install git

# verificar la instalacion
git --version
# debe mostrar: git version 2.x.x
```

```
git --version
```

La configuración de nombre y correo no es un trámite: queda registrada en cada versión que se confirma, y es lo que después permite saber quién hizo qué.

```
git config --global user.name "Nombre Apellido"
git config --global user.email "correo@unsa.edu.ar"

# editor para mensajes de commit
git config --global core.editor nano    # o vim, o gedit

# verificar la configuracion
git config --list
```

```
git config --global user.name "Nombre Apellido"
git config --global user.email "correo@unsa.edu.ar"

# editor (elegir uno instalado)
git config --global core.editor notepad
# o para VS Code:
git config --global core.editor "code --wait"

# en Windows, configurar el manejo de fin de linea
git config --global core.autocrlf true

# verificar
git config --list
```

Crear un repositorio

```
# crear el directorio del proyecto
mkdir ~/proyectos/mi_proyecto
cd ~/proyectos/mi_proyecto

# inicializar el repositorio Git
git init

# verificar que se creo la carpeta .git
ls -la
# debe aparecer: drwxr-xr-x ... .git
```

```
# crear el directorio en Documentos
mkdir /c/Users/TuNombre/Documentos/mi_proyecto
cd /c/Users/TuNombre/Documentos/mi_proyecto

git init
```

```
# en cmd o PowerShell
```

```
cd C:\Users\TuNombre\Documentos
mkdir mi_proyecto
cd mi_proyecto
git init
```

```
cd /ruta/al/proyecto/existente
git init

# agregar todos los archivos existentes al primer commit
git add .
git commit -m "estado inicial del proyecto"
```

Lo que crea el repositorio es un directorio oculto con la base de datos de la historia. Mientras exista, el proyecto está versionado; si se lo borra, quedan los archivos y se pierde toda la historia.

El ciclo de trabajo

Git introduce un paso que la teoría no menciona porque no es común a todos los sistemas: el **área de preparación**. Un archivo modificado no entra automáticamente en la próxima versión; primero se lo agrega al área de preparación y después se confirma lo que hay ahí.

Eso permite algo útil: haber tocado cinco archivos por dos motivos distintos y confirmar dos versiones separadas y coherentes, en vez de una sola mezcla. Es la contrapartida práctica del consejo de la teoría de confirmar seguido y con cambios chicos.

```
git status
```

```
# agregar un archivo especifico
git add archivo.py

# agregar varios archivos
git add archivo1.sce archivo2.html

# agregar todos los archivos modificados y nuevos
git add .

# agregar todos los archivos .sce del directorio src/
git add src/*.sce

# agregar interactivamente (elegir que cambios incluir)
git add -p archivo.py
```

```
# commit con mensaje en la linea de comandos
git commit -m "descripcion clara de que se cambio y por que"

# commit abriendo el editor configurado para escribir
# un mensaje mas largo
git commit
```

```
# agregar todos los archivos modificados y commitear
# en un solo paso (solo para archivos ya rastreados)
git commit -am "descripcion del cambio"
```

El mensaje

```
# mal: no dice nada
git commit -m "cambios"
git commit -m "fix"
git commit -m "wip"

# bien: describe el cambio real
git commit -m "implementar metodo RK4 para la EDO del pendulo"
git commit -m "agregar validacion: angulo debe estar entre 1 y
   89"
git commit -m "corregir error: division por cero cuando v0=0"
git commit -m "documentar la funcion calibrar() con ejemplos"
```

Vale insistir en lo que ya dijimos: el mensaje debe explicar *por qué* se hizo el cambio. Qué archivos se tocaron y qué líneas cambiaron ya lo registra el sistema, y se recupera cuando haga falta. Lo que no se puede reconstruir después es la intención.

Ver la historia y las diferencias

```
# historia completa
git log

# historia compacta (una linea por commit)
git log --oneline

# historia con grafico de ramas
git log --oneline --graph --all

# historia de un archivo especifico
git log --oneline calcular.sce

# los ultimos 5 commits
git log --oneline -5
```

```
# diferencias entre el directorio de trabajo y el ultimo commit
git diff

# diferencias de un archivo especifico
git diff calcular.sce

# diferencias en el area de preparacion
git diff --staged
```

```
# diferencias entre dos commits
git diff abc1234 def5678

# diferencias entre el commit actual y el anterior
git diff HEAD~1 HEAD

# que archivos cambiaron entre dos commits (sin el detalle)
git diff --name-only HEAD~3 HEAD
```

La diferencia es la operación que da sentido a todo lo demás. Es lo que convierte una sucesión de copias en una historia legible: en vez de ver dos archivos completos, se ven las líneas que cambiaron. Por eso el control de versiones funciona bien con texto plano —código, LaTeX, datos en columnas— y mal con formatos binarios, donde la diferencia no se puede mostrar.

Deshacer

Ésta es la razón principal por la que conviene usar control de versiones desde el primer día: permite probar cosas sabiendo que se puede volver.

```
# descartar los cambios en un archivo (volver al ultimo commit)
# ATENCION: esta operacion no se puede deshacer
git checkout -- archivo.sce

# en Git moderno (2.23+), mas claro:
git restore archivo.sce

# descartar todos los cambios
git restore .
```

```
# quitar un archivo del staging sin descartar los cambios
git restore --staged archivo.sce

# version antigua del comando
git reset HEAD archivo.sce
```

```
# deshacer el ultimo commit manteniendo los cambios
# en el directorio de trabajo
git reset --soft HEAD~1

# deshacer el ultimo commit y sacar los cambios del staging
git reset HEAD~1

# deshacer el ultimo commit y descartar los cambios
# ATENCION: se pierden los cambios definitivamente
git reset --hard HEAD~1
```

```
# ver la historia para encontrar el commit al que volver
git log --oneline
```

```
# crear un nuevo commit que revierte los cambios de uno anterior
# (la forma segura: no reescribe la historia)
git revert abc1234

# ver el estado del proyecto en un commit pasado (solo lectura)
git checkout abc1234

# volver al estado actual
git checkout main
```

Conviene distinguir dos situaciones. Deshacer algo que todavía no salió de la máquina propia es libre. Reescribir una versión que ya compartieron otros crea problemas, porque la historia que ellos tienen deja de coincidir. La regla práctica: se puede corregir lo que no publicaste; lo publicado se corrige agregando, no borrando.

Qué no versionar

```
# crear el .gitignore
cat > .gitignore << 'EOF'
# archivos temporales de LaTeX
*.aux
*.log
*.toc
*.out
*.synctex.gz
*.fdb_latexmk
*.fls

# archivos temporales de Scilab
*.bin

# archivos temporales del sistema
*~
*.swp
.DS_Store         # Mac
Thumbs.db         # Windows

# datos grandes que no deben estar en el repo
datos/mediciones/*.csv
resultados/graficas/*.png

# archivos de configuracion local
config_local.sce
/tmp/

# archivos de Python
__pycache__/
*.pyc
*.pyo
```

```
# no ignorar el archivo de calibracion aunque sea CSV
!datos/calibracion.csv
EOF

# agregar al repositorio
git add .gitignore
git commit -m "agregar .gitignore"
```

```
git check-ignore -v nombre_archivo.csv
```

Esto realiza lo dicho en la teoría: entra lo que no se puede reconstruir —código, datos primarios, documentación— y no entra lo generado. Un archivo de resultados intermedios, una figura que el programa produce o un ejecutable ensucian la historia y hacen crecer el repositorio sin aportar nada, porque se obtienen otra vez ejecutando el programa.

Hay un caso que merece atención propia: los archivos de datos grandes. El sistema guarda cada versión completa de todo archivo binario que cambie, de modo que un archivo de cien megas modificado veinte veces son dos gigas de historia. Para eso hay extensiones específicas, o conviene dejar esos datos fuera del repositorio y versionar sólo el guión que los baja o los genera.

Ramas

```
# ver las ramas existentes (* indica la rama actual)
git branch

# crear una rama nueva
git branch nueva_funcionalidad

# cambiar a esa rama
git checkout nueva_funcionalidad

# crear y cambiar en un solo comando
git checkout -b nueva_funcionalidad

# version moderna
git switch -c nueva_funcionalidad

# hacer cambios y commits en la rama nueva
# (no afectan a main)
git add ...
git commit -m "..."

# volver a la rama principal
git checkout main
# o:
git switch main

# integrar los cambios de la rama en main
```

```
git merge nueva_funcionalidad

# eliminar la rama una vez integrada
git branch -d nueva_funcionalidad
```

La rama es lo que permite trabajar en algo nuevo sin poner en riesgo la versión que funciona. En cálculo científico el uso típico no es tanto la colaboración como la exploración: probar otro método numérico, otra discretización, otro modelo, con la posibilidad de volver a lo anterior si no mejora, y de conservar las dos líneas mientras se comparan.

El conflicto de fusión, que la teoría presentó como concepto, aparece acá cuando dos ramas modificaron las mismas líneas. El sistema marca la zona en disputa y se detiene. No es una falla: es información que sólo quien programó puede resolver, porque hay que decidir cuál de las dos versiones vale, o si vale una tercera.

Trabajo diario

```
# al comenzar una sesion de trabajo:
git status            # ver el estado actual
git log --oneline -5 # recordar donde se quedo

# durante el trabajo:
# ... editar archivos ...
git diff              # revisar los cambios antes de commitear
git add archivo.sce   # preparar los cambios
git status            # verificar que esta preparado
git commit -m "descripcion clara"

# al terminar la sesion:
git log --oneline -5 # verificar que todo quedo commitado
git status            # debe decir "nothing to commit"
```

```
# agregar el archivo olvidado al staging
git add archivo_olvidado.py

# modificar el ultimo commit para incluirlo
# (solo si el commit no fue compartido con nadie)
git commit --amend --no-edit
```

```
# guardar el trabajo en progreso en un stash
git stash

# hacer lo que necesite (cambiar de rama, etc.)
# ...

# recuperar el trabajo guardado
git stash pop

# ver los stashes guardados
```

```
git stash list
```

```
# corregir el mensaje del ultimo commit
# (solo si no fue compartido)
git commit --amend -m "mensaje correcto"
```

```
# ver el contenido de un archivo en un commit especifico
git show abc1234:src/calcular.sce

# comparar el archivo actual con una version anterior
git diff HEAD~3 -- src/calcular.sce
```

```
# bisect: busqueda binaria en la historia
git bisect start
git bisect bad            # el estado actual tiene el bug
git bisect good abc1234   # este commit funcionaba bien

# Git va mostrando commits intermedios
# para cada uno, verificar si tiene el bug:
git bisect good   # si este commit esta bien
git bisect bad    # si este commit tiene el bug

# al terminar, Git indica el primer commit con el bug
git bisect reset  # volver al estado actual
```

La búsqueda binaria en la historia merece un comentario, porque es la que más sorprende cuando se descubre. Si algo funcionaba hace un mes y hoy no, y hay doscientas versiones en el medio, el sistema propone versiones intermedias y con cada respuesta descarta la mitad: unas ocho pruebas para encontrar exactamente la versión que introdujo el problema. Es la misma bisección que se usa para aislar un error en un archivo de datos, aplicada a la historia del proyecto.

Repositorios remotos

```
# agregar el repositorio remoto con el nombre 'origin'
git remote add origin https://github.com/usuario/proyecto.git

# verificar
git remote -v

# subir la historia local al remoto por primera vez
git push -u origin main
# pedira usuario y contrasena (o token de acceso personal)

# en versiones recientes de GitHub se usa token en lugar
# de contrasena. Crear en:
# GitHub -> Settings -> Developer settings ->
# Personal access tokens -> Generate new token
# Seleccionar scope: repo
```

```
# subir los commits locales al remoto
git push

# bajar los cambios del remoto al local
git pull

# ver el estado del remoto sin descargar
git fetch
git status    # muestra si el local esta adelante o atras
```

El remoto cumple dos funciones distintas que conviene no confundir: sincronizar el trabajo entre varias personas o varias máquinas, y servir de resguardo. La segunda es la que más importa en trabajos individuales: un repositorio local es una sola copia, y un disco que falla se lleva la historia completa.

Para el trabajo en investigación, las convenciones sobre qué versionar, cómo escribir los mensajes y cómo organizar la colaboración están resumidas en Perez-Riverol et al. 2016a; Blischak, Davenport y Wilson 2016.

Fin de línea entre sistemas

```
# .gitattributes: define el manejo de fin de linea
# por tipo de archivo
cat > .gitattributes << 'EOF'
# normalizar todos los archivos de texto a LF
* text=auto

# archivos que siempre deben ser LF
*.py     text eol=lf
*.sce    text eol=lf
*.html   text eol=lf
*.css    text eol=lf
*.sh     text eol=lf

# archivos que siempre deben ser CRLF (solo Windows)
*.bat    text eol=crlf

# archivos binarios: no tocar los fines de linea
*.png    binary
*.pdf    binary
*.wav    binary
*.csv    binary
EOF

git add .gitattributes
git commit -m "agregar .gitattributes para fin de linea"
```

Éste es un problema que aparece apenas hay dos sistemas operativos involucrados y que

desconcierta porque no es un error del programa: Windows termina las líneas con dos caracteres y los sistemas de tipo Unix con uno. Sin configuración, cada vez que un archivo pasa de un sistema al otro, el sistema de versiones ve todas las líneas modificadas. La solución es declarar el tratamiento deseado en la configuración del repositorio, para que la conversión sea automática y no aparezca como cambio.

3.6. Git: instalación, flujo y referencia

Esta sección completa `tu05git.tex` con la parte operativa:

cómo se instala en cada sistema, cómo se configura, el esquema de las tres áreas y la tabla de comandos Git Project 2026; Chacon y Straub 2014.

3.6.1. Operaciones fundamentales

Las operaciones fundamentales son:

`git init`: inicializa un repositorio en el directorio actual.

`git add`: prepara los cambios para ser confirmados (*staging area*).

`git commit`: confirma los cambios preparados, creando un punto en la historia con un mensaje descriptivo. El mensaje debe explicar *por qué* se hizo el cambio, no qué archivos se modificaron (eso Git ya lo registra).

`git push` / `git pull`: sincroniza el repositorio local con un repositorio remoto (GitHub, GitLab, Codeberg).

`git branch`: crea una rama de desarrollo paralela para trabajar en una nueva funcionalidad sin afectar la rama principal.

`git merge`: integra los cambios de una rama en otra.

`git log`: muestra la historia de commits.

`git diff`: muestra las diferencias entre el estado actual y el último commit, o entre dos commits.

`git clone`: copia un repositorio remoto al sistema local, incluyendo toda la historia.

En repositorios colaborativos, el flujo de trabajo típico es: `pull` para actualizar el repositorio local con los cambios del remoto, modificar los archivos, `add` y `commit` para confirmar los cambios, y `push` para compartirlos. Cuando dos personas modifican el mismo archivo en paralelo se produce un *conflicto* que debe resolverse manualmente.

****

3.7. Control de versiones con Git

3.7.1. Instalación y configuración

Linux (Debian/Ubuntu)

```
sudo apt update
sudo apt install git

# verificar la instalacion
git --version
# debe mostrar: git version 2.x.x
```

Windows

Descargar el instalador de `https://git-scm.com/download/win` y ejecutarlo. Durante la instalacion:

- En *Choosing the default editor*: elegir Notepad++ o VS Code si estan instalados, o Notepad como alternativa simple.
- En *Adjusting your PATH environment*: seleccionar **Git from the command line and also from 3rd-party software**.
- En *Choosing HTTPS transport backend*: dejar la opcion por defecto (OpenSSL).
- En *Configuring the line ending conversions*: seleccionar **Checkout Windows-style, commit Unix-style line endings**.
- El resto de las opciones pueden dejarse por defecto.

Al terminar la instalacion aparece **Git Bash** en el menu de inicio: una terminal Linux emulada dentro de Windows. Se puede usar Git Bash o el simbolo del sistema (cmd) o PowerShell. Los comandos de Git son identicos en todos.

Verificar en Git Bash o cmd:

```
git --version
```

3.7.2. Configuracion inicial

Antes de usar Git por primera vez, configurar el nombre y el correo que apareceran en cada commit. Esta configuracion se hace una sola vez por computadora.

Linux (terminal):

```
git config --global user.name "Nombre Apellido"
git config --global user.email "correo@unsa.edu.ar"
```

```
# editor para mensajes de commit
git config --global core.editor nano    # o vim, o gedit

# verificar la configuracion
git config --list
```

Windows (Git Bash o cmd):

```
git config --global user.name "Nombre Apellido"
git config --global user.email "correo@unsa.edu.ar"

# editor (elegir uno instalado)
git config --global core.editor notepad
# o para VS Code:
git config --global core.editor "code --wait"

# en Windows, configurar el manejo de fin de linea
git config --global core.autocrlf true

# verificar
git config --list
```

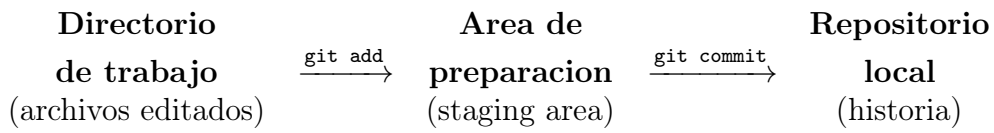
La configuracion se guarda en:

- Linux: `~/.gitconfig`
- Windows: `C:\Users\TuNombre\.gitconfig`

3.7.3. Crear un repositorio local

3.7.4. El flujo de trabajo basico

El ciclo de trabajo con Git tiene tres areas:



Directorio de trabajo: los archivos del proyecto tal como estan en el disco. Aqui se edita el codigo.

Area de preparacion (*staging area*): zona intermedia donde se eligen que cambios van al proximo commit. Permite hacer commits parciales.

Repositorio local: la base de datos de la historia del proyecto, guardada en `.git/`.

3.7.5. Linux y Windows

Aspecto	Linux	Windows
Terminal	Bash nativa	Git Bash, cmd o PowerShell
Separador de path	`/`	`\` (Git Bash acepta `/`)
Fin de linea	LF	CRLF
Config autocrlf	`false`	`true`
Home del usuario	`~/`	`C:\Users\Nombre\`
Archivo .gitconfig	`~/.gitconfig`	`%USERPROFILE%\.gitconfig`

El problema de fin de linea entre Linux y Windows es el mas frecuente en proyectos mixtos. La configuracion `core.autocrlf = true` en Windows hace que Git convierta automaticamente los finales de linea al hacer checkout (CRLF) y al commitear (LF), manteniendo los archivos en el repositorio siempre con LF (estandar Unix).

Si trabaja en un proyecto que comparten Linux y Windows, agregar un archivo `.gitattributes` al repositorio:

```
# .gitattributes: define el manejo de fin de linea
# por tipo de archivo
cat > .gitattributes << 'EOF'
# normalizar todos los archivos de texto a LF
* text=auto

# archivos que siempre deben ser LF
*.py     text eol=lf
*.sce    text eol=lf
*.html   text eol=lf
*.css    text eol=lf
*.sh     text eol=lf

# archivos que siempre deben ser CRLF (solo Windows)
*.bat    text eol=crlf

# archivos binarios: no tocar los fines de linea
*.png    binary
*.pdf    binary
*.wav    binary
*.csv    binary
EOF

git add .gitattributes
git commit -m "agregar .gitattributes para fin de linea"
```

3.7.6. Resumen de comandos

Comando	Que hace
`git init`	Inicializar un repositorio
`git status`	Ver el estado actual
`git add archivo`	Preparar un archivo para commit
`git add .`	Preparar todos los cambios
`git commit -m "msg"`	Guardar los cambios preparados
`git log -oneline`	Ver la historia resumida
`git diff`	Ver cambios no preparados
`git diff -staged`	Ver cambios preparados
`git restore archivo`	Descartar cambios en un archivo
`git restore -staged`	Sacar del staging
`git reset -soft HEAD~1`	Deshacer el ultimo commit
`git branch nombre`	Crear una rama
`git switch nombre`	Cambiar de rama
`git merge nombre`	Integrar una rama
`git stash`	Guardar trabajo en progreso
`git stash pop`	Recuperar trabajo guardado
`git remote add origin URL`	Conectar con remoto
`git push`	Subir al remoto
`git pull`	Bajar del remoto
`git clone URL`	Copiar un repositorio remoto

3.7.7. Control de versiones con Subversion

La teoría distinguió dos arquitecturas: en la centralizada hay un único repositorio en un servidor y las copias locales son sólo espacios de trabajo; en la distribuida cada copia es un repositorio completo. Subversion es el ejemplo vigente de la primera Collins-Sussman, Fitzpatrick y Pilato 2011; Apache Software Foundation 2026.

Conviene conocerlo por dos razones prácticas. Muchos proyectos científicos y muchas instituciones lo siguen usando, de modo que tarde o temprano hay que trabajar sobre un repositorio así. Y el contraste con el modelo distribuido hace entender mejor a los dos: qué gana y qué pierde cada uno.

El modelo

Hay un repositorio, que vive en un servidor y es la única copia de la historia. Cada persona tiene una **copia de trabajo**: los archivos en su estado actual más la información mínima para saber de qué revisión provienen. La copia de trabajo no contiene la historia.

De ahí se siguen las diferencias con el modelo distribuido:

La numeración es global y creciente. Cada confirmación produce una **revisión** numerada, y ese número identifica el estado completo del repositorio, no de un archivo. La revisión 428 es una foto de todo el proyecto. Es más fácil de nombrar y de ci-

tar que los identificadores de un sistema distribuido, y por eso los proyectos que necesitan referirse a versiones concretas suelen apreciarlo.

Hace falta conexión. Confirmar, ver la historia y comparar con revisiones anteriores requieren hablar con el servidor. En el modelo distribuido todo eso es local.

No hay área de preparación. Se confirma lo que está modificado en la copia de trabajo, eventualmente restringido a ciertos archivos. El paso intermedio que introduce Git no existe acá.

Se puede bajar sólo una parte. Como el repositorio está en el servidor, se puede obtener un subdirectorio en lugar del proyecto entero. En un sistema distribuido eso no es natural, porque la copia local trae toda la historia.

Operaciones básicas

Los conceptos son los de la teoría, con estos nombres:

`checkout` obtiene por primera vez una copia de trabajo a partir de la dirección del repositorio. Se hace una sola vez.

`update` trae al espacio local los cambios que otros confirmaron. Es lo primero que conviene hacer al empezar a trabajar.

`status` informa qué cambió en la copia local. Con la opción que consulta al servidor, además marca qué hay de nuevo allá sin modificar nada local.

`diff` muestra las diferencias, por omisión entre la copia de trabajo y la revisión de la que proviene.

`commit` confirma los cambios locales al repositorio, con su mensaje. A diferencia del modelo distribuido, acá confirmar *es* publicar: no hay paso separado.

`log` muestra la historia de un archivo o del proyecto.

`add` **y** `delete` registran altas y bajas de archivos, que se hacen efectivas en la próxima confirmación.

`revert` descarta los cambios locales de un archivo y lo devuelve al estado de la revisión de la que proviene.

Dos que conviene tener presentes y que ya usamos en la práctica: `cat` muestra el contenido de un archivo en una revisión determinada sin tocar la copia de trabajo, e `info` informa de qué revisión proviene el archivo local.

Revisiones

La revisión se puede indicar de varias formas, y ésa es una comodidad del modelo centralizado: por número, por la palabra que designa la última del repositorio, por la que corresponde a la copia local antes de modificarla, o por fecha.

Un punto que confunde: `log` sin más muestra la historia hasta la revisión que se tiene bajada, no hasta la última del repositorio. Para ver lo que hay más adelante hay que pedirlo explícitamente. Es la consecuencia directa de que la copia local no tenga la historia completa.

Ramas y etiquetas

Acá aparece la diferencia de diseño más llamativa. En Subversion una rama no es un objeto especial: es una **copia** de un directorio dentro del mismo repositorio. Copiar es una operación barata, porque el servidor no duplica el contenido sino que registra la relación.

De ahí surge la convención de tres directorios en la raíz del repositorio: uno para la línea principal de desarrollo, uno que contiene las ramas y otro que contiene las etiquetas. Nada obliga a usarla; es una costumbre que el sistema no impone pero que todos siguen, y es lo que hace que un repositorio ajeno resulte familiar.

Una etiqueta es también una copia, con el acuerdo de no modificarla: sirve para marcar el estado exacto con el que se produjo un resultado. Para trabajo científico eso es directamente útil, porque permite volver años después a la versión con la que se hizo un cálculo publicado.

La fusión existe y es la operación que más ha cambiado entre versiones del sistema; conviene verificar el comportamiento de la versión que se esté usando Apache Software Foundation 2026.

Propiedades

Subversion tiene una característica sin equivalente directo en Git: cada archivo o directorio puede llevar **propiedades**, que son pares de nombre y valor almacenados junto al objeto y versionados como él.

Las más usadas resuelven problemas que ya nombramos. Una declara qué archivos ignorar en un directorio, que es el equivalente del listado de exclusiones. Otra fija el tratamiento del fin de línea, que es el problema entre sistemas operativos distintos. Otra marca un archivo como ejecutable. Y otra registra que un directorio incorpora contenido de otro repositorio.

Es un mecanismo elegante: en lugar de archivos de configuración especiales, los metadatos viajan con el objeto al que se refieren y tienen su propia historia.

Conflictos

Cuando dos personas modifican las mismas líneas, el sistema marca el conflicto al actualizar y deja la copia de trabajo en un estado que hay que resolver antes de poder confirmar. Genera archivos auxiliares con las tres versiones —la propia, la del repositorio y el ancestro común— y espera que quien trabaja decida.

Vale lo dicho en la teoría: el conflicto no es una falla del sistema sino información que sólo las personas involucradas tienen. Lo que sí es propio de este modelo es que aparece al actualizar y no al fusionar ramas, porque acá el flujo normal es que todos trabajen sobre la misma línea.

Cuándo conviene cada modelo

El centralizado tiene a favor la numeración global, que hace trivial citar una versión; el control de acceso por directorio, que permite que alguien vea o modifique sólo una parte; la posibilidad de bajar sólo un subdirectorio de un proyecto grande; y una curva de aprendizaje más corta, porque hay menos conceptos.

El distribuido tiene a favor que se puede trabajar y confirmar sin conexión, que cada copia es un resguardo completo de la historia, y que las ramas son suficientemente baratas como para usarlas todo el tiempo Torvalds 2007.

Para el trabajo de la materia, lo importante no es cuál usar sino haber entendido los conceptos comunes: que la historia se guarda, que cada versión tiene autor, fecha y motivo, que las diferencias son la unidad de lectura, y que lo generado no se versiona. Eso vale igual en los dos, y es lo que se traslada a cualquier sistema que aparezca después.

3.8. Trabajo colaborativo entre humanos e inteligencia artificial

Se plantea muchas veces para programación, pero sirve para todo trabajo intelectual: escribir un informe, buscar bibliografía, entender un artículo, armar un gráfico, resolver un ejercicio, depurar un programa, traducir, resumir. En todos esos casos la situación es la misma: una herramienta produce texto o código plausible a partir de lo que se le pide, y la persona decide qué hacer con eso. Esta unidad trata de cómo organizar ese trabajo para que la herramienta ayude sin que la persona deje de entender, verificar y responder por lo que entrega.

3.8.1. Qué son y qué hacen

Las herramientas actuales de inteligencia artificial para texto y código son modelos de lenguaje de gran tamaño (LLM, *large language models*): redes neuronales entrenadas sobre cantidades enormes de texto para predecir la continuación más probable de una secuencia Vaswani et al. 2017; Brown et al. 2020. Sobre esa base se construyen los asistentes de conversación, los completadores de código en el editor y los agentes que ejecutan tareas de varios pasos. Lo que producen es una continuación plausible, no una consulta a una base

de datos ni un cálculo: por eso aciertan mucho en lo frecuente y bien documentado, y por eso también inventan, con la misma fluidez, funciones que no existen, resultados numéricos que no calcularon y citas bibliográficas que nadie escribió. A ese comportamiento se lo llama alucinación Ji et al. 2023, y es la razón central de todo lo que sigue: la salida de una IA es una propuesta, no un resultado.

Lo que hacen bien, y conviene aprovechar: explicar un fragmento de código o un texto, proponer alternativas, traducir entre lenguajes o idiomas, generar borradores, casos de prueba, esqueletos de programas, comandos de la terminal, fórmulas de LaTeX, y señalar errores en algo que se les muestra. Lo que hacen mal: aritmética y cálculo numérico sin una herramienta que los ejecute, memoria de hechos precisos (fechas, versiones, nombres de funciones poco usadas), citas, y cualquier cosa que dependa de información posterior a su entrenamiento o privada.

3.8.2. Cómo pedir

La calidad de la respuesta depende de la calidad del pedido. Reglas que funcionan en cualquier herramienta:

- Dar contexto: qué se está haciendo, en qué lenguaje, con qué versión, para qué materia, qué se probó ya. Pegar el código o el fragmento de texto real, y el mensaje de error completo.
- Pedir una cosa por vez, y decir qué forma se espera de la respuesta: "un programa de Scilab que…", "una explicación de dos párrafos", "una tabla con…", "sólo el comando".
- Pedir que explique, no sólo que resuelva: "explicame por qué", "qué hace cada línea", "qué pasa si…". La explicación es lo que permite después verificar.
- Pedir pruebas y casos límite junto con el código, y pedir que diga qué supuestos hizo.
- Iterar: corregir, acotar, volver a preguntar. La primera respuesta rara vez es la última.
- Cuando se le pide bibliografía o datos, pedir la fuente y comprobarla afuera; una cita que la herramienta no puede mostrar no existe hasta que se la encuentra.

3.8.3. Cómo verificar

Todo lo que produce la herramienta se comprueba antes de aceptarlo, y el método de comprobación depende de qué es. Un programa se ejecuta con los casos de prueba, los normales, los límite y los inválidos, y se compara con un resultado conocido, como se hace en la unidad de simulación con la solución analítica o la energía conservada. Un cálculo se rehace con la calculadora del lenguaje. Un comando se lee antes de correrlo, y si toca archivos se prueba primero en un directorio de descarte. Una explicación se contrasta con la documentación oficial (`help`, la ayuda del lenguaje, el manual). Una cita se busca en el catálogo o en el DOI. Un texto se lee entero, porque la fluidez esconde errores de

contenido. Y en todos los casos vale la prueba de la explicación: si la persona no puede explicar con sus palabras por qué la solución es correcta, la solución no está terminada.

3.8.4. Qué herramientas hay

Los nombres cambian cada pocos meses; lo que se mantiene son las categorías. Al momento de escribir, con ejemplos:

Asistentes de conversación en el navegador o en una aplicación: ChatGPT (OpenAI), Claude (Anthropic), Gemini (Google), Copilot (Microsoft), Le Chat (Mistral). Sirven para preguntar, explicar, redactar, traducir, revisar. La mayoría tiene un nivel gratuito y puede ejecutar código Python en un entorno propio, lo que corrige en parte su debilidad para el cálculo: conviene pedir explícitamente que calcule ejecutando y no "de memoria".

Asistentes de código en el editor que completan mientras se escribe y responden sobre el archivo abierto: GitHub Copilot en VS Code, Cursor, Codeium, JetBrains AI. Son cómodos y peligrosos por lo mismo: proponen código plausible en el lugar exacto donde va, y es fácil aceptar sin leer.

Agentes de programación que reciben una tarea, leen el repositorio, escriben, ejecutan, ven el error, corrigen, y entregan un cambio: Codex (OpenAI) Chen et al. 2021, Claude Code (Anthropic), Gemini CLI, Aider, Cursor en modo agente. Trabajan desde la terminal o el editor, corren los comandos del proyecto (`make`, las pruebas, `git`) y hacen en minutos lo que a una persona le lleva horas. Exigen más control, no menos: hay que leer el *diff* completo antes de aceptarlo, tenerlos en un repositorio con control de versiones para poder volver atrás, y no darles acceso a nada que no se quiera que toquen. Un agente que corrió las pruebas y dice que pasan sigue siendo una propuesta hasta que la persona lo verifica.

Búsqueda con IA que responde citando páginas: Perplexity, los modos de búsqueda de los asistentes anteriores, Google con resúmenes. Útiles para orientarse; las fuentes hay que abrirlas.

Modelos locales que corren en la propia computadora sin enviar datos a nadie: Ollama, LM Studio, llama.cpp, con modelos abiertos como Llama, Mistral, Qwen o Gemma. Son más lentos y menos capaces que los grandes, pero no tienen costo por uso ni problema de privacidad, y para explicar código o redactar alcanzan; en una máquina sin GPU conviene un modelo chico.

Herramientas específicas para tareas del laboratorio: asistentes de LaTeX (Overleaf lo integra), generación de figuras y esquemas (los asistentes producen SVG, TikZ o código de gnuplot), extracción de datos de tablas o gráficos en PDF, y transcripción.

Para el curso, cualquiera de las tres primeras categorías sirve, y la recomendación es usar al menos dos distintas en el mismo problema y comparar: donde discrepan hay algo que entender.

3.8.5. Ciclo de trabajo

Las herramientas de inteligencia artificial pueden incorporarse al proceso de desarrollo de software, y de cualquier trabajo intelectual, como asistentes para generar código o texto, proponer alternativas, explicar, detectar errores, generar casos de prueba y revisar soluciones. No deben considerarse un reemplazo del programador ni un integrante autónomo del equipo: la responsabilidad sobre el producto continúa siendo humana, y una respuesta generada es una propuesta que debe ser comprendida, comprobada y, si es necesario, modificada antes de ser aceptada. Un posible ciclo de trabajo es:

especificar → IA propone → humano comprende → humano prueba → humano corrige → IA revisa →

Este esquema permite utilizar la capacidad de generación y análisis de la IA sin transferirle la responsabilidad sobre el resultado. Con un agente el ciclo es el mismo, sólo que "IA propone" incluye ejecutar y corregir por su cuenta; el "humano comprende" y "humano prueba" no se saltean.

3.8.6. Adaptación de metodologías ágiles

Las metodologías ágiles de desarrollo Schwaber y Sutherland 2020b pueden adaptarse al trabajo humano–IA. Por ejemplo, en una adaptación de Scrum la IA puede intervenir como herramienta de asistencia en distintas actividades, pero los roles y las decisiones continúan bajo responsabilidad humana.

Actividad	**Participación posible de la IA**
Product Backlog	Analizar requisitos, proponer tareas y detectar ambigüedades.
Sprint Planning	Proponer alternativas de implementación, identificar dependencias y ayudar a estimar la complejidad.
Desarrollo	Generar fragmentos de código, explicar algoritmos, proponer refactorizaciones y buscar posibles errores.
Pruebas	Proponer casos normales, casos límite, entradas inválidas y pruebas de regresión.
Revisión	Comparar la implementación con los requisitos y señalar posibles problemas.
Retrospectiva	Ayudar a resumir dificultades, errores frecuentes y posibles mejoras del proceso.

En todos los casos, las decisiones y la aceptación de los resultados corresponden a las personas.

3.8.7. Definition of Done en trabajo humano–IA

Cuando se utiliza IA para desarrollar un programa, la generación del código no implica que la tarea esté terminada. Puede establecerse una *Definition of Done* que exija, como mínimo:

- que el programa se ejecute correctamente;
- que satisfaga la especificación original;
- que el programador comprenda el algoritmo utilizado;
- que pueda explicar las partes relevantes del código;
- que existan casos de prueba;
- que se hayan considerado casos normales, casos límite y entradas incorrectas;
- que todo código producido por una IA haya sido revisado;
- que los errores o limitaciones encontrados hayan sido corregidos;
- que no se hayan utilizado funciones que resuelvan directamente un algoritmo cuya implementación formaba parte del problema;
- que el programador pueda justificar la solución final independientemente de quién haya generado inicialmente cada fragmento de código.

Para un informe o un texto la lista es análoga: que la persona haya leído todo, que pueda defender cada afirmación, que las citas existan y digan lo que se les atribuye, y que lo que se entrega como propio lo sea.

3.8.8. Roles humanos durante el uso de IA

En un trabajo grupal pueden distinguirse temporalmente diferentes actividades:

- **programador:** desarrolla y modifica la solución;
- **operador de IA:** formula consultas y analiza las respuestas obtenidas;
- **revisor:** examina el código sin asumir que una propuesta generada automáticamente es correcta;
- **tester:** diseña pruebas destinadas a verificar la solución y encontrar errores.

Estos roles pueden rotarse. El objetivo es evitar que la interacción con una IA se reduzca a solicitar un programa y copiar su respuesta.

3.8.9. Autoría, ética y cuidado de los datos

Tres cuestiones que no son técnicas y hay que decidir antes de empezar. Autoría: en esta materia se puede usar IA en cualquier trabajo, pero se declara qué herramienta se usó y para qué, y lo que se entrega tiene que poder ser explicado y defendido por quien lo entrega; copiar una salida sin entenderla no es aprender y no se acepta como propio. Datos: no se pegan en una herramienta en línea datos personales, contraseñas, claves de acceso, ni resultados de investigación que no se quieran publicar; para eso están los modelos locales.

Y sesgo y error: la herramienta reproduce lo que hay en sus datos de entrenamiento, con sus errores y sus sesgos, y responde con la misma seguridad cuando acierta y cuando no; la confianza no es un indicador de nada. Las guías institucionales sobre uso de IA en la enseñanza Miao y Holmes 2023 van en la misma dirección: transparencia, verificación y responsabilidad humana.

El uso de IA modifica algunas actividades del desarrollo de software y del trabajo intelectual en general, pero no el principio fundamental de responsabilidad: quien utiliza un programa, o firma un texto, debe poder comprenderlo, verificarlo y justificar sus resultados.

3.9. Ejercicios

1. **Ejercicio:** Instale LibreOffice y LaTeX

   Instale LibreOffice y LaTeX en GNU/Linux y en Windows. En GNU/Linux use TeX Live (`sudo apt install texlive-full`); en Windows, MiKTeX.

2. **Ejercicio:** Carpeta personal

   Cree una carpeta con su nombre, que será su directorio personal, en el que guardará todos los archivos que genere durante el curso.

   Puede hacerlo en su computadora personal o en un dispositivo portátil conocido como pen-drive, usalmente conectado vía USB.

3. **Ejercicio:** Documentos con LO

   Inicie LO y abra un documento de texto nuevo y copie tal cual el texto que figura como introducción teórica en este práctico.

4. **Ejercicio:** Movimiento Rectilíneo Uniforme (MRU)

   A continuación se plantea un problema de Movimiento Rectilíneo Uniforme (MRU). En la tabla siguiente se indican medidas de la posición de un automóvil que viaja a velocidad constante a distintos tiempos.

t(s)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
x(m)	100	109	118	130	135	140	153	163	170	178	190	196	206	215	223

   a) En Calc, abra una planilla de cálculo nueva, calcule la velocidad del auto empleando el método de cuadrados mínimos descrito en el punto 3, generando los campos necesarios. Compare los resultados obtenidos al emplear las funciones *slope* (pendiente) e *intercept* (ordenada al origen) del programa.

   b) Grafique x vs t superponiendo en el mismo gráfico los datos experimentales y la recta ajustada.

Después del Punto 3

Un dibujo.

Datos

Recta

Error

Cuadrados mínimos

Muchos estudiantes entienden mejor viendo el procedimiento.

En el Punto 4 Agregaría una pregunta.

No solamente

calcule

sino

¿Qué representa físicamente la pendiente obtenida? y ¿Qué representa la ordenada al origen?

Realice el informe de la resolución del problema anterior, mediante una presentación Impress usando LO. Describa el método empleado y los resultados obtenidos.

5. **Ejercicio:** Instale LaTeX

   a) Instale una distribución de LaTeX (por ejemplo TeX Live) y un editor adecuado, o utilice un servicio en línea como Overleaf.
   b) Compile el siguiente documento de ejemplo y observe el resultado.
   c) Modifique el título, el autor y la fecha.
   d) Agregue una sección denominada "Introducción" y escriba un breve texto describiendo el objetivo del informe.
   e) Inserte una ecuación utilizando el entorno matemático de LaTeX.
   f) Genere el archivo PDF correspondiente.

6. **Ejercicio:** HTML

   HTML es el lenguaje por defecto para estructurar las páginas web.

   Escriba un archivo denominado `index.html` que contenga:

   a) un título;
   b) un encabezado con el nombre de la asignatura;
   c) un párrafo describiendo el problema de Movimiento Rectilíneo Uniforme;
   d) una tabla con los datos experimentales;
   e) una imagen correspondiente al gráfico obtenido;
   f) un enlace al archivo PDF generado en el punto anterior.
      Abra el archivo utilizando un navegador web y compruebe el resultado.

7. **Ejercicio:** PDF

   Generalatexción de documentos PDF. con LaTeX.

   Exporte a formato PDF los documentos tipo:

   a) LibreOffice Writer;
   b) LibreOffice Calc;
   c) LibreOffice Impress;

   Compare el tamaño de los archivos generados y analice las diferencias observadas.

8. **Ejercicio:** Conversiones

   Los documentos científicos suelen intercambiarse entre diferentes programas.

   Realice las siguientes conversiones:

   a) Documento Writer → PDF.
   b) Documento Writer → HTML.
   c) Documento Markdown → HTML.
   d) Documento Markdown → PDF.
   e) Documento LaTeX → PDF.

   Analice las ventajas y limitaciones de cada formato para su distribución por correo electrónico o publicación en Internet.

9. **Ejercicio:** nano, vim, ed y sed

   Abra un archivo de texto plano con `nano`, escriba tres líneas de contenido, guárdelo y ciérrelo. Luego ábralo con `vim`: entre al modo inserción con `i`, agregue una línea, salga del modo inserción con `Esc` y guarde con `:wq`. ¿En qué se diferencian los dos editores? ¿Cuál le resultó más intuitivo?

   Maneje mínimamente **ed**y **sed** .

10. **Ejercicio:** LaTeXbásico

    Cree el siguiente documento LaTeX mínimo, compílelo con `pdflatex` y verifique que genera un PDF correcto:

```
\documentclass[12pt,a4paper]{article}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\begin{document}
\section{Mi primer documento}
Este es mi primer documento en \LaTeX{}.
\end{document}
```

    Agregue su nombre como autor, la fecha actual y un segundo párrafo. Recompile y verifique los cambios.

11. **Ejercicio:** Matemática

    Escriba en LaTeX las siguientes expresiones matemáticas y compílelas para verificar el resultado:

    a) La fórmula cuadrática: $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

    b) La integral de Gauss: $\displaystyle\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}$

    c) La serie de Taylor de e^x: $e^x = \sum_{n=0}^{\infty} \dfrac{x^n}{n!}$

    d) Un sistema de dos ecuaciones usando el entorno `align`.

12. **Ejercicio:** LaTeX varios

    Cree un documento LaTeX que contenga: un título, autor y fecha; tres secciones con al menos un párrafo cada una; una lista de viñetas y una lista numerada; una tabla de al menos tres columnas y cuatro filas con el entorno `tabular`; una referencia cruzada de una sección a otra usando `\label` y `\ref`; y una entrada bibliográfica citada con `\cite`.

13. **Ejercicio:** LaTeX, beamer

    Cree un documento LaTeX en formato presentación, practique presentarlo en un proyector que lo permita, como pdf, mediante un pendrive USB, sin computadora.

14. **Ejercicio:** HTML

    Escriba la siguiente página HTML, ábrala en un navegador y verifique que se muestra correctamente:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Mi página</title>
</head>
<body>
  <h1>Título principal</h1>
  <p>Un párrafo con <strong>negrita</strong> y <em>cursiva</em>.</p>
  <ul>
    <li>Primer ítem</li>
    <li>Segundo ítem</li>
  </ul>
  <a href="https://www.gnu.org">Sitio GNU</a>
</body>
</html>
```

    Agregue una tabla HTML con los datos de tres compañeros: nombre, correo y carrera. Valide el HTML con `tidy -errors archivo.html`.

15. **Ejercicio:** XML

    Cree un archivo XML que represente el catálogo de al menos cinco libros, con los campos título, autor, año, editorial e ISBN. Verifique que está bien formado con `xmllint -format catalogo.xml`. Luego extraiga solo los títulos usando XPath: `xmllint -xpath //titulo/text()catalogo.xml`.

16. **Ejercicio:** Markdown

    Escriba el siguiente archivo en Markdown, procéselo con Pandoc y genere tres salidas: HTML, PDF y DOCX:

```
# Informe de práctica

## Introducción

Texto de introducción con **negrita** y *cursiva*.

## Resultados

| Medición | Valor |
|----------|-------|
| Prueba 1 | 3.14  |
| Prueba 2 | 2.71  |

## Conclusiones

Lista de conclusiones:

- Primera conclusión
- Segunda conclusión
```

    ¿Qué diferencias observa entre las tres salidas?

17. **Ejercicio:** Pandoc

    Convierta los siguientes formatos usando Pandoc y examine el resultado en cada caso:

    a) Un archivo `.docx` a Markdown.
    b) Un archivo `.html` a LaTeX.
    c) Un archivo Markdown a EPUB.
    d) Un archivo LaTeX a HTML.

    En cada conversión identifique qué elementos se perdieron o se transformaron de forma imperfecta.

18. **Ejercicio:** Formato Calc

    Usando LibreOffice u otro, cargue la siguiente tabla de datos de temperatura medida cada hora, en una planilla de cálculo:

Hora	Temperatura (°C)
0	12.3
3	11.1
6	10.8
9	15.4
12	21.7
15	23.2
18	19.8
21	14.5

Calcule en celdas separadas: el promedio, el máximo, el mínimo y el desvío estándar usando las funciones de la planilla. Grafique los datos como gráfico de líneas con los ejes correctamente etiquetados. Agregue una línea de tendencia polinomial de grado 2 y anote la ecuación y el valor de R^2.

19. **Ejercicio:** CSV

    Descargue un archivo CSV de datos abiertos (por ejemplo del portal datos.gob.ar o cualquier fuente pública) con al menos 100 filas y 4 columnas. Ábralo en LibreOffice Calc e investigue: ¿cuántas filas tiene? ¿Qué columnas hay? Calcule estadísticas básicas de al menos dos columnas numéricas. Genere un gráfico de dispersión entre dos variables y agregue la línea de tendencia más adecuada. ¿Existe correlación entre esas dos variables?

20. **Ejercicio:** CSV, cálculos

    Procese el mismo archivo CSV descargado en el ejercicio anterior desde la terminal usando `csvkit`:

    ```
    csvstat datos.csv
    csvcut -c 1,3 datos.csv | head -10
    csvsort -c 2 -r datos.csv | head -5
    csvgrep -c 3 -m "valor" datos.csv
    ```

    Compare la eficiencia de trabajar desde la terminal con la de trabajar en la planilla de cálculo para cada tarea. ¿Cuándo conviene cada enfoque?

21. **Ejercicio:** Artículo Científico

    Elija un artículo científico publicado en arXiv o en cualquier revista de acceso abierto. Busque si los autores publicaron el código y los datos junto con el artículo. Si lo hicieron: descargue el repositorio, intente reproducir al menos uno de los resultados o figuras del artículo, y documente si lo logró y qué obstáculos encontró. Si no lo hicieron: reflexione sobre qué necesitaría para reproducir los resultados y por qué la ausencia del código dificulta la verificación independiente.

22. **Ejercicio:** PDF – HTML

    Compare los formatos PDF y HTML como formatos de publicación de documentos. Para cada uno indique: si es editable, si preserva el formato independientemente del

dispositivo, si es accesible para lectores de pantalla, si permite búsqueda de texto, si puede ser indexado por motores de búsqueda y si es adecuado para impresión. ¿En qué situaciones elegiría uno sobre el otro?

23. **Ejercicio:** WYSIWYG – WYGIWYW

    Examine la diferencia entre un editor WYSIWYG y un editor WYGIWYW realizando la misma tarea en ambos: escriba un documento de una página con título, autor, tres secciones numeradas, una lista y una fórmula matemática. Use LibreOffice Writer para la versión WYSIWYG y LaTeX para la versión WYGIWYW. Compare el tiempo dedicado, la facilidad del proceso, la calidad del resultado y el tamaño del archivo generado. ¿Cuál elegiría para un trabajo práctico? ¿Y para una tesis? Justifique.

24. **Ejercicio:** bash y Markdown

    Escriba un script de `bash` que genere automáticamente un informe en formato Markdown a partir de datos del sistema: fecha y hora actuales, nombre del usuario, nombre de la máquina, uso de disco (`df -h`), memoria disponible (`free -h`) y los diez procesos que más CPU consumen (`ps aux -sort=-| head -10`). Luego convierta el informe a PDF con Pandoc. El script debe poder ejecutarse cada día y generar un informe distinto con la información actualizada.

Capítulo 4

Simulación

Hemos resumido las secciones completas de simulación por un resumen:

4.1. Simulación de sistemas físicos

Un sistema físico se estudia mediante un **modelo**: una representación formal que conserva las variables y relaciones necesarias para describir su comportamiento. En una simulación no se trabaja directamente sobre el sistema material, sino sobre ese modelo. Por eso conviene distinguir:

Sistema material: parte del universo físico que se quiere estudiar.

Modelo: representación formal del sistema, con variables, parámetros, ecuaciones, condiciones iniciales y condiciones de borde.

Método numérico: procedimiento que permite resolver aproximadamente las ecuaciones del modelo.

Programa: implementación concreta del método en una herramienta de cálculo, por ejemplo Scilab.

El dominio del modelo es el conjunto de estados posibles del sistema, o más generalmente el conjunto de objetos sobre los que el modelo habla. Sobre ese dominio se interpretan las variables, relaciones y ecuaciones. Las proposiciones no son el dominio: son enunciados que pueden resultar verdaderos o falsos al evaluarse en un modelo y, si corresponde, en un estado determinado.

La simulación consiste en ejecutar el modelo para obtener su evolución o sus estados posibles. Cuando el modelo contiene ecuaciones diferenciales, la simulación calcula aproximaciones discretas de una solución continua. En lugar de conocer una función completa, se obtienen valores en una sucesión de tiempos:

$$t_0, t_1, t_2, \ldots, t_n.$$

Si el estado del sistema se escribe como un vector $Y(t)$, una forma habitual de representar el problema es:

$$\frac{dY}{dt} = F(t, Y), \qquad Y(t_0) = Y_0.$$

El vector Y contiene las **variables de estado**. Las condiciones iniciales fijan desde dónde parte la simulación. La función F expresa las leyes físicas, restricciones y relaciones constitutivas que determinan cómo cambia el sistema.

En Scilab, este tipo de problema se resuelve con `ode`. El esquema conceptual es:

```
y = ode(y0, t0, t, f)
```

Aquí `y0` es el estado inicial, `t0` el tiempo inicial, `t` el vector de tiempos en los que se calcula la solución, y `f` la función que representa $F(t, Y)$.

4.1.1. El ciclo modelo–EDO–método–programa

La simulación científica puede pensarse como un ciclo:

1. Se identifica el sistema y se seleccionan las magnitudes relevantes.
2. Se construye un modelo matemático mediante leyes físicas y supuestos simplificadores.
3. Se escribe el modelo como ecuaciones, preferentemente como un sistema de ecuaciones diferenciales ordinarias de primer orden.
4. Se elige un método numérico: Euler, Runge–Kutta, métodos adaptativos, métodos implícitos, etc.
5. Se implementa el cálculo en un programa y se comparan los resultados con la teoría o con datos medidos.

El método de Euler muestra la idea elemental de avance por pasos. Si

$$\frac{dy}{dt} = f(t, y),$$

entonces, para un paso Δt:

$$y_p = y_i + \Delta t\, f(t_i, y_i).$$

El subíndice i indica el estado inicial del paso y p el próximo estado. Métodos más precisos evalúan la función varias veces dentro del intervalo o ajustan automáticamente el tamaño del paso. La idea de fondo, sin embargo, es la misma: calcular el próximo estado a partir del estado actual y de las reglas de cambio.

4.1.2. Modelos de parámetros concentrados

Un modelo de **parámetros concentrados** reemplaza un sistema espacialmente distribuido por una red de componentes discretos. En vez de describir cada punto del espacio mediante ecuaciones en derivadas parciales, se representa el sistema mediante un número finito de variables y ecuaciones diferenciales ordinarias.

La hipótesis central es que cada componente concentra ciertas propiedades físicas: resistencia, capacidad, inductancia, masa, elasticidad, capacidad calorífica, etc. El sistema completo se obtiene conectando esos componentes en una red. Esa red impone relaciones entre variables, de modo similar a lo que ocurre en los circuitos eléctricos.

En el caso eléctrico aparecen componentes típicos:

Resistor: relaciona tensión y corriente mediante una ecuación de transporte, por ejemplo $V = RI$. No tiene memoria: no almacena una variable de estado.

Condensador: almacena carga Q. Su ecuación de estado es $V = Q/C$ y su ecuación de cambio es $\dot{Q} = I$.

Inductor: almacena flujo magnético Φ. Su ecuación de estado es $I = \Phi/L$ y su ecuación de cambio es $\dot{\Phi} = V$.

Fuentes: imponen tensión o corriente y representan intercambio de energía con el exterior del circuito.

Las conexiones se describen mediante nodos, ramas, lazos y conjuntos de corte. Bajo las hipótesis de parámetros concentrados, las leyes de Kirchhoff resumen las restricciones de red:

- la suma de corrientes en un nodo es cero;
- la suma de tensiones en un lazo es cero.

Así se combinan dos tipos de ecuaciones: las ecuaciones propias de cada componente y las ecuaciones impuestas por la red.

4.1.3. Variables generalizadas y analogías

Los circuitos sirven como modelo para otros dominios físicos porque muchas ramas de la física tienen estructuras formales semejantes. Las variables pueden nombrarse de manera general:

Desplazamiento: magnitud acumulada o extensiva, como carga Q o posición x.

Flujo: derivada temporal del desplazamiento, como corriente I o velocidad v.

Fuerza: variable intensiva que impulsa el cambio, como tensión V o fuerza mecánica F.

Momento: integral temporal de la fuerza, como flujo magnético Φ o momento mecánico.

Esta nomenclatura permite comparar dominios eléctricos, mecánicos y térmicos. Por ejemplo, un sistema masa–resorte puede escribirse como:

$$\dot{x} = v, \qquad \dot{v} = \frac{k}{m}x - g.$$

Un circuito RLC serie puede escribirse con dos variables de estado, por ejemplo la corriente del inductor I y la carga del condensador Q:

$$RI + L\dot{I} + \frac{Q}{C} - E = 0, \qquad \dot{Q} = I.$$

Reordenando:

$$\dot{I} = \frac{E - RI - Q/C}{L}, \qquad \dot{Q} = I.$$

Estas ecuaciones ya están en forma adecuada para simulación numérica: dado el estado actual (I, Q) se calcula el próximo estado. En cada paso se actualizan las variables según las reglas de cambio. Si se usa voltaje del condensador en lugar de carga, $Q = CV$ y queda:

$$\dot{V} = \frac{I}{C}.$$

4.1.4. Sentido del método

La ventaja de los modelos de parámetros concentrados es que permiten estudiar sistemas complejos con herramientas algebraicas y numéricas manejables. La red organiza las interacciones; los componentes aportan las ecuaciones físicas; el método numérico calcula la evolución temporal.

Por eso, programas como Scilab, Sceptre, Spice, Simusol o Xcos pueden resolver modelos de dominios distintos usando ideas semejantes: variables de estado, relaciones constitutivas, conexiones de red y avance temporal por integración numérica.

Debe notarse que la siguiente materia en la Licenciatura de Energías Renovables, no en la Tecnicatura, es Laboratorio II. En esa materia se ven métodos numéricos. Por lo que en ésta materia sólo enseñamos como usar las funciones de Scilab a éstos efectos, dejando de lado como trabajan esas funciones.

4.2. Ciencia, filosofía, modelos y simulación: resumen

Qué llamamos ciencia. Ciencia es la capacidad de predecir en forma contrastable, o la actividad que produce avances en esa capacidad. Es experimental y es empírica, y no tiene sentido decir "ciencia empírica" porque no hay otra. Una teoría científica debe poder ser negada por al menos un experimento que pueda fallar Popper 1959; un experimento que no puede fallar no sirve a este efecto. La ciencia avanza reduciendo el error de sus

predicciones, ampliando el ámbito de aplicación y unificando campos en un mismo marco. El consenso de la comunidad llega después: lo que lo impulsa es la capacidad de predecir. En sistemas con azar, sea por errores aleatorios o por resultados intrínsecamente aleatorios como en la mecánica cuántica, la predicción es una distribución de probabilidad y no un número.

Qué llamamos filosofía. Filosofía es lo que la ciencia no ha cubierto: temas donde no hay consenso, donde no hay predicciones o no son razonablemente precisas. Se reconoce una posición filosófica cuando sobre un mismo tema coexisten ideas contradictorias y no hay forma racional y universalmente aceptada de cerrar la disputa. Esas disputas suelen expresar intereses distintos, y cuando esos intereses pasan a la acción colectiva se entra en la política. Es otro tipo de conocimiento, no uno inferior; sólo distinto.

La diferencia, en una frase. En ciencia hay un criterio de cierre, la predicción contrastada; en filosofía no lo hay. Por eso todas las filosofías de la ciencia comparten el mismo mínimo operativo (medir, formalizar, calcular, predecir, contrastar) y difieren sólo en lo que agregan encima: mecanismos reales y ontología material Bunge 1967, reconstrucción lógica del lenguaje y adecuación empírica Carnap 1937; Hempel 1965; Fraassen 1980, práctica social transformadora Marx 1845; Lenin 1909, marcos históricos y lingüísticos Kuhn 1962; Gadamer 1975, o familias de modelos idealizados Cartwright 1983; Giere 1988; Suppes 1960. Ese mínimo es lo que aquí se estudia; lo demás se reconoce y se deja para otro ámbito.

El mínimo: instrumentalismo predictivo y operacional. Para estudiar un sistema material se construye una estructura formal que funciona como modelo cuando sus símbolos se vinculan con procedimientos de medición y sus resultados pueden compararse con observaciones. Hay dos planos: el sistema material, abordado por magnitudes medibles, y el sistema formal, construido con variables, parámetros, relaciones, ecuaciones, condiciones iniciales y de borde y, si corresponde, reglas de evolución. Un modelo es el sistema formal interpretado para describir, calcular o predecir magnitudes del sistema material. Interpretar una variable es saber qué mide, cómo se mide y qué predicciones permite calcular. Esta idea reúne el falsacionismo Popper 1959, el operacionalismo Bridgman 1927, el convencionalismo Poincare 1905; Duhem 1954 y el pragmatismo Peirce 1878; Dewey 1938. La matemática, en este marco, es lenguaje: auxiliar de la razón y de la ciencia, no ciencia; "sistema material" es sólo lo que se mide, sin compromiso con el debate materialismo–idealismo, que se pospone.

Modelos y simulación. Si la ciencia es predicción y para predecir sobre casos concretos se hacen modelos y se los simula, entonces ciencia y capacidad de simular son parte del mismo concepto: la ciencia es el trabajo de buscar, para cada sistema material, cómo hacer un modelo y simularlo. Un modelo que sólo relaciona magnitudes es estático; uno que describe cambios de estado mediante reglas de evolución es dinámico, y el equilibrio no es ausencia de dinámica sino un posible resultado de ella. Las leyes de la ciencia (Newton, termodinámica, enfriamiento, crecimiento exponencial, circuitos, osciladores) son relaciones matemáticas entre magnitudes verificadas experimentalmente, casi siempre ecuaciones diferenciales; son la materia prima del modelo pero no el modelo completo, que

además necesita hipótesis, parámetros, condiciones e interpretación física de las variables. El proceso es: identificar el sistema y sus variables, aplicar las leyes, simplificar con hipótesis, resolver analítica o numéricamente, validar contra datos y usar para predecir o diseñar Cellier y Kofman 2006; Saravia Alía 2019. La simulación numérica de ese modelo, que ocupa el resto de la unidad, es la forma concreta en que la definición de ciencia se convierte en un programa.

4.3. Las variables de la física

4.3.1. Por qué empezar por las variables

Si la ciencia es predecir, y para predecir hacemos modelos y los simulamos, la variable es la bisagra entre los dos planos: del lado del sistema material es lo que se mide; del lado del sistema formal es el símbolo con el que se calcula; y del lado del programa es la celda de memoria con un nombre. Un modelo no es más que un conjunto de variables y las relaciones entre ellas. Por eso, antes de las leyes y antes de Scilab, conviene tener claro qué es una variable, de qué tipos hay y cómo se relacionan.

4.3.2. Sistema material

Un sistema material o físico es una parte del universo en el que vivimos, tomada como objeto de estudio DeVoe 2020; Callen 1985. Qué parte es una decisión: se traza una frontera y todo lo que queda afuera es el ambiente. Las divisiones pueden ser arbitrarias, pero según lo que se quiera estudiar tiene sentido decidir qué integra el sistema y cómo eso cambia en el tiempo. Un sistema puede considerarse formado por subsistemas o partes, y a la inversa varios sistemas pueden verse como uno solo. Hay dos formas típicas de definirlo: por masa de control, siguiendo un conjunto fijo de materia en el tiempo, o por volumen de control, mirando una región fija del espacio y viendo qué entra y qué sale.

Un sistema está aislado cuando sus cambios no provocan cambios en otros sistemas ni los de otros lo afectan a él. Dos sistemas interactúan cuando no están aislados entre sí. El entorno de un sistema es el conjunto de los sistemas con los que interactúa, y el conjunto sistema más entorno está, por construcción, aislado. Muchas veces conviene pensar la interacción como una acción: un sistema acciona y el otro responde. Al que se estudia se lo llama el sistema a secas; al que acciona, el accionante, y en un experimento suele ser algo que el experimentador controla directamente: una persona, un motor, una computadora Ashby 1956. Esta distinción entre lo que se controla y lo que responde reaparece enseguida como la diferencia entre variables independientes y dependientes.

4.3.3. Estado, cambio, predicción

El cambio, la diferencia y el estado se definen experimentalmente para cada sistema: deben poder observarse, detectarse y medirse Fermi 1937. Si se percibe el tiempo a saltos, un sistema que cambia muestra diferencias entre los estados de dos momentos; si se lo percibe continuo, en cada instante está en un estado que incluye su movimiento. En

cualquier caso hay que poder decir experimentalmente en qué situación está el sistema, y eso puede incluir posiciones, velocidades, temperaturas o lo que corresponda.

Cuando el espacio de estados puede pensarse como geométrico, cada estado es un punto, las variables son sus coordenadas, y un cambio es un segmento entre el estado inicial y el final. Una secuencia de cambios es un camino; un camino que termina donde empieza es un ciclo. El objetivo, el de la ciencia en general, es predecir el futuro. Si el sistema es determinista, podemos, o pensamos que podemos, predecir cómo evolucionará: para eso hace falta toda la información pertinente que reproduzca esa misma información en el futuro, paso a paso. Esa información es el estado, y las reglas que lo llevan de un paso al siguiente son el modelo dinámico Khalil 2002; Cellier y Kofman 2006. Un sistema dinámico es uno en el que se estudian estados con movimiento; uno estático, en el que sólo se estudian estados de equilibrio, es decir, estados en los que un sistema aislado no cambia. Un estado estacionario es distinto: el sistema no cambia, pero si se lo aísla, cambia; hay flujos que lo atraviesan. Los tres casos aparecen en las simulaciones del laboratorio: el sistema dinámico es lo que se integra en el tiempo, el equilibrio y el estacionario son adonde tiende, y distinguirlos es parte de interpretar el resultado.

4.3.4. Qué es una variable

Una variable o magnitud física representa o describe una propiedad de un sistema DeVoe 2020: de configuración, de estado o de cambio. Casi siempre es numérica, real o a veces entera. Pueden requerirse muchas variables para describir completamente el estado de un sistema o sus cambios. En un sistema material una variable es medible, o calculable a partir de otras medibles; sabemos que una variable cambió porque podemos medir, percibir o ver el cambio. Interpretar una variable es saber qué mide, cómo se mide y qué predicciones permite calcular.

Tres cosas acompañan siempre a una variable física y conviene decirlas explícitamente porque la computadora no las lleva por sí sola. La unidad: el valor se expresa en relación a una unidad, y las unidades del Sistema Internacional Bureau International des Poids et Mesures 2019 son la referencia. El error: en física una variable va acompañada de un intervalo de error, aunque sea implícito, sea de medida o de predicción. Y el nombre: cada variable tiene un símbolo, y ese símbolo se usa siempre igual; que una magnitud se llame T o x es una convención, pero cambiarla a mitad de un razonamiento es la fuente más común de error.

Constantes y parámetros. Una constante es una variable que no cambia en el problema, sea porque es una constante de la naturaleza (g, c, k_B) o porque se la fija para ese caso (la masa del péndulo, la longitud del hilo). A las segundas se las suele llamar parámetros: son lo que se cambia entre una simulación y otra, y no dentro de una simulación. La distinción es práctica, no de fondo: en el modelo del péndulo la longitud es un parámetro; en el modelo del hilo que se estira es una variable.

4.3.5. Tipos de variables

Por su forma matemática, una variable es escalar cuando es un número real; vectorial cuando vive en un espacio vectorial de dimensión fija, habitualmente 3 o 2, con producto

escalar; y puede ser también compleja, tensorial o de otros tipos. Casi todo lo que se simula en el laboratorio es escalar o vectorial de pocas componentes.

Por su relación con el tamaño del sistema. Una variable es aditiva si para un sistema dividido en dos partes vale la suma de lo que vale en cada parte: $\psi_{Tot} = \psi_1 + \psi_2$, para cualquier división y cualquier estado Callen 1985. Una variable de cantidad es la que expresa cuánto de algo hay dentro de una región: masa, volumen, energía, carga, cantidad de sustancia; es escalar y aditiva. Si el sistema es homogéneo, es decir, si dos partes iguales en volumen y forma tomadas de cualquier manera son idénticas, las variables cuyo valor es proporcional al volumen se llaman extensivas (masa, volumen, energía) y las que valen lo mismo en cualquier porción se llaman intensivas (presión, temperatura, densidad). El cociente de dos variables de cantidad, como la masa sobre el volumen, es una variable tipo densidad: es intensiva y no es aditiva.

Por su relación con el proceso. Una variable de estado es la que participa en la descripción de un estado: vale algo en cada punto del espacio de estados, y su cambio en un proceso depende sólo del estado inicial y del final, $\Delta\psi = \psi_{final} - \psi_{inicial}$. Una variable de proceso depende de todo el camino, no de un estado en particular: un promedio, una suma a lo largo del recorrido, el trabajo o el calor en termodinámica Fermi 1937. La notación sigue esa diferencia: Δ para un cambio finito, d para el diferencial de una variable de estado, δ para el de una de proceso. Los d y δ son Δ pensados en el límite de cambios cada vez menores, y esa idea de límite es la que después usa el método numérico al elegir un paso.

Por su papel en un balance. Cuando una variable de cantidad X cambia en una región, el cambio se puede contabilizar: lo que aumenta adentro es lo que entró menos lo que salió por la frontera, más lo que se generó menos lo que se destruyó adentro. La variable que expresa lo que atraviesa una superficie es un desplazamiento, vectorial; su cambio por unidad de tiempo es un flujo. Cuando la generación y la destrucción son cero la variable es conservativa: si baja en un sistema sube en el de al lado. Masa, energía, carga y cantidad de movimiento son conservativas; la entropía sólo se genera. La ecuación de balance

$$\frac{dX}{dt} = \text{entra} - \text{sale} + \text{genera}$$

es la forma en que casi todas las leyes de conservación entran en un modelo, y es también la primera comprobación que se le hace a una simulación: si la variable debía conservarse y no se conserva, el error está en el programa o en el método.

4.3.6. Funciones, variables independientes y dependientes

Una función asigna a cada elemento de un conjunto origen un elemento de un conjunto destino; en física hablamos de funciones de unas variables a otras DeVoe 2020. Las variables que determinan el elemento del conjunto origen se llaman independientes; la del destino, dependiente. Cuando la función tiene más de una variable independiente, el elemento de origen queda fijado por varias a la vez.

En un modelo la elección de qué es independiente y qué es dependiente es una decisión, no un hecho de la naturaleza. Las independientes típicas son el tiempo y la posición: no dependen de nada del sistema, las fija el observador, y son las que se recorren en la simulación. Las dependientes son las que el sistema decide: la temperatura en función del tiempo, la altura en función del tiempo, la presión en función del volumen. Una ley de

la ciencia es una función, o una relación entre funciones, entre variables: $F = m\,a$ dice que la aceleración es función de la fuerza y la masa. Y una ecuación diferencial es una ley que en vez de dar la dependiente da su derivada respecto de la independiente: la regla de evolución. Con la regla de evolución y el estado inicial, el modelo dinámico predice el estado en cualquier momento; sin el estado inicial, la regla sola no predice nada. Modelo, ecuación diferencial, condición inicial: los tres son necesarios y los tres se traducen en el programa Saravia Alía 2019.

4.3.7. De la variable física a la variable del programa

Cada variable del modelo se convierte en una variable del programa, y las cuatro cosas que definen a la segunda, nombre, tipo, tamaño y dirección, salen directamente de la primera. El nombre es el símbolo de la variable física, o algo que lo recuerde: T se llama `T`, θ se llama `theta`, y no `x1`. El tipo es casi siempre el real de doble precisión, que es como la computadora aproxima a $\mathbb{R}$; los enteros aparecen en los contadores y en las cantidades discretas. El tamaño viene del papel de la variable en el modelo: un parámetro es un escalar; una variable dependiente de una sola independiente, como $T(t)$, se convierte en un vector, con un elemento por cada valor de t en que se calculó; una variable vectorial en el espacio, como una velocidad, es un vector de tres componentes; y una variable dependiente de dos independientes, como la temperatura en función de la posición y el tiempo, es una matriz. La variable independiente misma es un vector, el de los tiempos o las posiciones en que se evalúa. La dirección es donde el programa la guarda, y sólo importa cuando dos nombres apuntan al mismo lugar o cuando el tamaño es grande.

Lo que no pasa al programa por sí solo, y hay que llevarlo a mano, es lo que acompaña a la variable física: la unidad, que conviene anotar en un comentario junto a la declaración y respetar en todo el cálculo, porque la computadora suma metros con segundos sin quejarse; el error, que en el programa se convierte en el error de redondeo del tipo real y en el error del método numérico, y que hay que estimar aparte; y la interpretación, es decir, qué mide esa variable, que está en el nombre y en el comentario o no está en ninguna parte.

En el modelo	En el programa	Ejemplo
constante de la naturaleza	escalar, definido una vez	`g = 9.81`
parámetro	escalar, se cambia entre corridas	`L = 0.5`
variable independiente	vector de valores	`t = 0:0.01:10`
variable dependiente $y(t)$	vector, mismo tamaño que `t`	`theta`
variable vectorial $\vec{v}$	vector de 2 o 3 componentes	`v = [vx; vy]`
dependiente de dos independientes $T(x,t)$	matriz	`T(i,j)`
estado del sistema dinámico	vector de estado	`[x; v]`
ley, regla de evolución	función	`function dy = f(t,`
condición inicial	valor inicial del vector de estado	`y0 = [1; 0]`
variable de proceso	acumulador o integral numérica	`W = W + p * dV`
balance, conservación	comprobación después de correr	`E($) - E(1)`

La tabla es el resumen de la unidad y el puente hacia la siguiente: lo que en la consola de Scilab aparece como nombre, tipo, tamaño y dirección de una variable, es lo que aquí se decidió mirando el sistema material.

4.4. Herramientas de cálculo científico

Existen varias alternativas para hacer cálculo científico, cada una con distintas características.

Herramienta	Uso principal
Scilab	Cálculo numérico y simulación
GNU Octave	Compatible con gran parte de MATLAB
Python (NumPy, SciPy, Matplotlib), otras con Perl	Programación científica
GNUplot	Gráficos científicos
SageMath	Matemática simbólica y numérica
Maxima	Álgebra computacional
LibreOffice Calc	Hojas de cálculo

Hay librerías en fortran, c y c++ y otros lenguajes para muchas funciones matemáticas y físicas.

En este curso se utilizará Scilab por ser software libre, de fácil instalación y especialmente orientado al cálculo científico.

4.5. Introducción a Scilab

Veremos la información básica sobre el funcionamiento del lenguaje y los comandos más comunes a fin de poder comenzar a utilizar el programa. Como con cualquier otra aplicación, la destreza se irá adquiriendo con el uso. Scilab Scilab Group 2000; Baudin y Kroeter 2011; Scilab Enterprises 2026c es fundamentalmente numérico, no simbólico: para expresiones simbólicas (derivadas, integrales sin resolver) no alcanza y conviene usar **Maxima** u otro programa. Los tutoriales introductorios Gomez 2003; Calvo Rolle 2003; Scilab 2024; Scilab 2026; Bassi y Venturin 2026; Echevarría Líbano 2013; Mora Escobar 2001, los apuntes de la cátedra de los que se tomaron textos y ejemplos Condorí, Finetti y Saravia 2011, el manual de referencia y el sitio del proyecto Scilab Group 2026; INRIA 2026, y los libros de Urroz Urroz 2001 y de Goyal y Dhingra Goyal y Dhingra 2019 cubren con más detalle todo lo que sigue.

4.5.1. Versiones y arquitecturas

Los ejemplos desarrollados para este libro se realizaron usando la versión 2024.0.0 o versiones anteriores de Scilab para GNU/Linux, La última, al momento de la edición, es la 2026.1.0, pero las nociones básicas son comunes, siendo las diferencias entre ellas fundamentalmente el agregado de nuevas funciones y aplicaciones. En el sitio de Scilab se pueden ver los cambios entre versiones. En las versiones modernas no existen diferencias sustanciales entre el funcionamiento en GNU/Linux, MAC Intel, MAC ARM y Windows. La sintaxis de comandos y sentencias son las mismas y se ejecutan igual en todas las versiones. Sólo se observan ligeras diferencias de aspecto y traducción al castellano de los menús.

4.5.2. Instalación y directorio de Scilab

En GNU/Linux Scilab se instala desde los paquetes de la distribución (`apt install scilab` en Debian y derivadas) o descargando desde el sitio de Scilab un archivo comprimido que se desempaqueta donde se quiera; se recomienda `/usr/local` u `/opt`. En Windows y Mac se usa el instalador del sitio. En el directorio de instalación los subdirectorios más significativos son:

- **bin** (binarios): los ejecutables, `scilab`, `scilab-cli` y `scilab-adv-cli`, y los scripts[1] que arrancan Scilab con el entorno correcto.
- **modules** (módulos): un subdirectorio por cada módulo de Scilab (`elementary_functions`, `linear_algebra`, `polynomials`, `differential_equations`, `graphics`, `scinotes`, `xcos`, etc.). Dentro de cada uno están `macros` (el código fuente en lenguaje Scilab de las funciones), `help` (la ayuda en línea, subdividida por módulo), `demos` (los ejemplos y demostraciones que se ven con el botón **Demos**), `tests` y `src` (el código fuente en C y Fortran de las rutinas numéricas). Es la versión actual de lo que en Scilab antiguo eran los directorios `routines`, `man` y `demos`. Recorrer `demos` y leer `macros` es muy útil: los ejemplos incluidos inspiran al nuevo usuario y las macros muestran cómo está escrito Scilab en Scilab.
- **contrib**: los módulos externos instalados con ATOMS, el gestor de paquetes de Scilab.
- **share** y **thirdparty**: iconos, traducciones, bibliotecas de terceros. En Scilab antiguo existía además `examples`, con ejemplos de cómo conectar Scilab con programas externos mediante enlaces dinámicos; hoy eso está en la ayuda de `ilib_build` y `link`.

4.5.3. La ventana de trabajo

En la figura 4.1 se observa la ventana de trabajo de Scilab. En las versiones actuales la ventana está dividida en paneles: la **consola** en el centro, donde se escriben los comandos; el **navegador de archivos** a la izquierda; el **navegador de variables** y el **historial de comandos** a la derecha. Cada panel se puede cerrar, mover o volver a abrir desde el menú **Applications**. En la barra de menús se tienen las siguientes opciones:

File (archivo): *Execute* (ejecutar un archivo de comandos o de funciones, con terminación `.sce` o `.sci`), *Open a file* y *Open file with SciNotes*, *Load environment* y *Save environment* (cargar y grabar las variables de la sesión), *Change current directory* y *Display current directory* (cambiar y mostrar el directorio de trabajo), *Page setup* y *Print* (imprimir), y *Quit* (salir, termina la sesión).

Edit (editar): cortar, copiar y pegar mediante el "clipboard" (portapapeles), un sector de memoria donde se guarda el contenido copiado hasta que se "pega" en otro texto; seleccionar todo; borrar el historial; limpiar la consola; y *Preferences*.

[1] Un script es un programa sencillo que contiene una secuencia de instrucciones que el programa principal (en este caso Scilab) deberá interpretar. Ver la unidad de programación.

Preferences (preferencias): opciones para cambiar la apariencia y el comportamiento, como el idioma de la interfaz, colores, tipo y tamaño de letra, la codificación de archivos y las opciones de SciNotes y de la ayuda.

Control (control): *Resume*, *Abort* e *Interrupt* (reanudar, abortar, interrumpir), que permiten detener un cálculo largo o moverse entre niveles de ejecución cuando una función se detuvo con `pause`.

Applications (aplicaciones): **SciNotes**, el editor de textos con el cual se escriben los guiones y funciones que se ejecutarán en Scilab (reemplaza al antiguo SciPad); **Xcos**, el paquete de simulación de sistemas dinámicos por diagramas de bloques (antes Scicos) Scilab Enterprises 2026d; el navegador de variables, el historial de comandos, el navegador de archivos, el gestor de módulos ATOMS y el traductor de MATLAB a Scilab.

? (ayuda): **Scilab Help**, que abre un navegador con la descripción de cada uno de los comandos, agrupados por módulo, la misma información que la ayuda en línea; **Scilab Demonstrations**, que abre una lista de demostraciones de las posibilidades de Scilab (se elige una y se ejecuta; se recomienda probarlas); enlaces al sitio y a la comunidad; y *About*.

Recorriendo la ayuda y las demostraciones se aprecia la gran variedad de funciones disponibles.

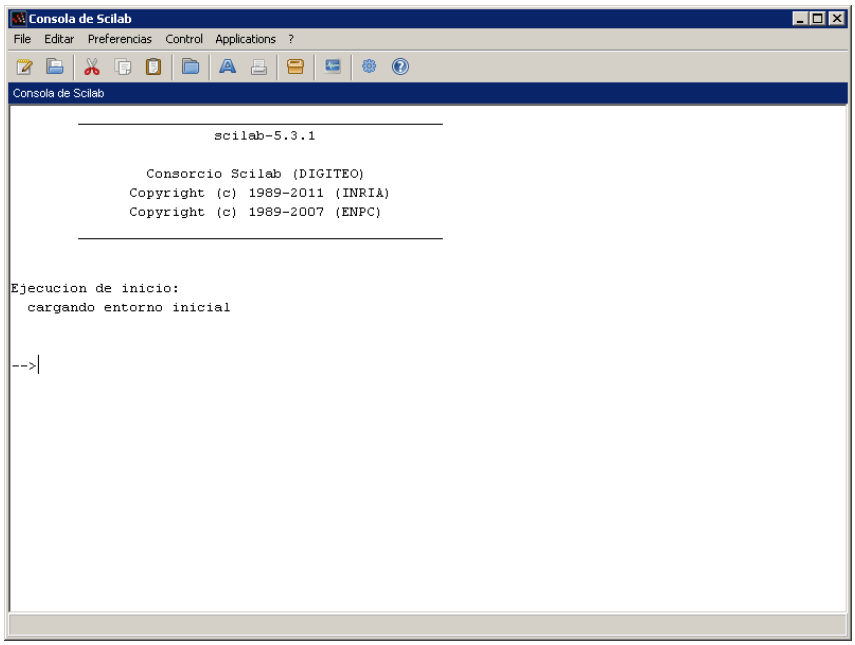


Figura 4.1: Consola de Scilab.

4.5.4. Entrada, ejecución y salida

Comencemos viendo cómo se ejecuta el programa y cómo se cierra al dejar de usarlo. Se inicia Scilab haciendo clic en el icono correspondiente en el escritorio o, en GNU/Linux,

abriendo una terminal y escribiendo `scilab`. Lo primero que se muestra es la ventana de trabajo con la consola:

```
   Startup execution:
     loading initial environment

--> t = 2+3
 t  =

    5.

-->
```

Scilab es una aplicación interactiva: se introducen las operaciones que se desean ejecutar ("los comandos") y el programa las ejecuta. Veremos después que también es posible ejecutar una serie de comandos escritos en un archivo (programas) mediante un editor de texto; para eso Scilab trae un editor incorporado, **SciNotes**, al cual se accede desde el menú **Applications** o con el botón de la barra de herramientas.

En la parte superior está el menú principal desde donde se pueden ejecutar las operaciones principales desplegando los menús. Debajo se presenta una barra de herramientas con iconos para la ejecución rápida de comandos específicos, como llamar a SciNotes, abrir un archivo, cortar, copiar, pegar, cambiar el directorio actual, abrir Xcos o la ayuda.

Más abajo, en la consola, Scilab muestra el siguiente símbolo, que se conoce como *prompt*, indicando que el programa está listo para recibir la instrucción:

```
->
```

Después del prompt se comienzan a introducir los comandos a ejecutar. En el ejemplo se le ha ordenado que sume $2 + 3$ y almacene el resultado en la variable `t`. El comando se ejecuta presionando la tecla `ENTER`. Scilab imprime el contenido de la variable `t`, con el resultado de la operación, y a continuación vuelve a mostrar el prompt indicando que está listo para recibir otra instrucción.

Cuando se escribe solamente el nombre de una variable y se presiona `ENTER`, Scilab muestra en pantalla el contenido actual de esa variable. Por ejemplo, si antes se definió `t = 2.56`, al escribir `t` Scilab imprime su valor.

Para salir de Scilab se utiliza el comando `exit`, escribiéndolo a continuación del prompt y presionando `ENTER`, o seleccionando la opción *Quit* desde el menú **File**, con lo cual se vuelve al escritorio del sistema operativo.

```
--> exit
```

4.5.5. Directorio de trabajo y registro de sesión

Scilab trabaja siempre en un directorio actual. Allí busca archivos y allí guarda, por defecto, los archivos que se generan durante la sesión. El comando `pwd` muestra ese directorio:

```
   pwd
```

Para crear un directorio se usa `mkdir`, y para cambiar el directorio de trabajo se usa `chdir`:

```
mkdir("practico4");
chdir("practico4");
```

La función `diary` permite guardar en un archivo de texto lo que aparece en la consola durante la sesión. Se abre indicando el nombre del archivo y se cierra con `diary(0)`:

```
diary("guia4-nombreestudiante.txt");

// comandos de trabajo

diary(0);
```

El archivo queda guardado en el directorio de trabajo. Es útil para conservar el registro de comandos, resultados y errores producidos durante una práctica.

4.5.6. Uso de Scilab desde la terminal bash, extensiones .sce, .sci

Scilab puede ejecutarse con interfaz gráfica, pero también desde una terminal. Esto es útil para correr programas en forma automática, repetir cálculos, probar ejercicios o integrar Scilab con otros programas.

El comando habitual es:

```
scilab-cli
```

Este comando se escribe en la terminal del sistema operativo, no dentro del prompt `->` de Scilab.

El comando abre Scilab en modo texto, sin la interfaz gráfica completa. Desde allí se pueden escribir comandos igual que en la consola de Scilab.

También se puede ejecutar directamente un archivo `.sce`.

Se podrá argumentar que esto es entrar a programación y en cierta forma lo es, sin embargo hacemos apenas un uso marginal de la cuestión al igual que usar funciones. Es util introducrilo aquí.

“X.sce” es un script. Se ejecuta línea por línea, como si escribieras comandos en la consola.

```
scilab-cli -f programa.sce
```

Por ejemplo, si el archivo `programa.sce` contiene:

```
x = 3;
y = x^2 + 1;
y
```

al ejecutar:

```
scilab-cli -f programa.sce
```

se obtiene en pantalla:

```
10
```

Esta forma de trabajo permite usar Scilab en scripts de shell, en trabajos por lotes o en procesos de corrección automática.

También puede usarse para ejecutar un programa y guardar la salida en un archivo:

```
scilab-cli -f programa.sce > salida.txt
```

Si se quiere cargar un archivo de funciones y luego ejecutar un programa, se puede hacer desde el propio archivo `.sce`:

```
exec("funciones.sci", -1);

r = cuadrado(5);
r
```

En este caso, el archivo `funciones.sci` contiene las definiciones de funciones, y el archivo `programa.sce` las carga y las usa.

En resumen:

`scilab`: abre Scilab con interfaz gráfica.

`scilab-cli`: abre Scilab en modo consola.

`scilab-cli -f archivo.sce`: ejecuta un archivo de comandos desde la terminal.

4.5.7. Ayuda de Scilab

Ya hemos visto que en el menú principal se encuentra un botón "**?**" para invocar la ayuda de Scilab desde donde se accede a un navegador de ayuda y al detalle de todos los comandos agrupados por temas.

También es posible obtener la información de un comando específico directamente dentro de la ventana de trabajo con el comando `help`. Si, por ejemplo, se escribe después del prompt `help tan`, sin las comillas, obtendremos directamente la información detallada sobre la función tangente. Si no se dispone del nombre específico de la función, se puede solicitar información de todas las funciones de un tema dado utilizando el comando `apropos`. Por ejemplo, se puede pedir `apropos polynomial` para obtener todas las funciones relacionadas con los polinomios.

Si bien la ayuda de Scilab está en inglés, es sumamente útil para confirmar la sintaxis o ver todas las posibilidades de un comando, y para obtener ejemplos de aplicación y una lista de comandos asociados. Su utilización es altamente recomendable cada vez que se invoca un nuevo comando.

4.6. Conceptos de Scilab: constantes, variables, operadores

Antes de introducir los primeros programas conviene presentar los conceptos de constantes; variables: nombre, ubicación, valor, tipo (espacio ocupado); dinámicas y estáticas; memoria; tipos de datos; expresiones; y operadores. Qué es una variable en el modelo físico se vio en la primera parte de la unidad; aquí se ve cómo aparece en Scilab.

4.6.1. Constantes

Es habitual utilizar en matemática algunas constantes que tienen una importancia especial, como ser el número **pi**, el número **e**, etc. Estos números se encuentran predefinidos en Scilab y pueden ser utilizados directamente en cualquier cálculo, se los llama "constantes". Estos números están protegidos de modo que no se pueden cambiar. Se distinguen mediante una nomenclatura especial, colocando el signo `%` delante de su nombre. En el cuadro siguiente se muestran algunos de ellos. En el ejemplo se ve que al intentar cambiar el número pi, dándole el valor 0 el programa marca un error ya que una constante no se puede cambiar. En el cuadro se observa el número e y también el número imaginario `i` que se utiliza para escribir números complejos.

```
-->%pi
 %pi  =
    3.1415927

-->%pi=0
    !--error   13
 Redefiniendo la variable permanente

-->%e
 %e  =
    2.7182818

-->2+3*%i
 ans  =
    2. + 3.i
```

4.6.2. Variables: nombre, dirección, tipo, tamaño

Una variable tiene varias características simultáneas que es importante distinguir:

Nombre: el identificador simbólico con el que quien programa se refiere a la variable en el código fuente. El compilador o intérprete lo traduce a una dirección de memoria.

Dirección: la posición en memoria donde se almacena el valor. Normalmente no se trabaja con direcciones directamente, pero es útil entender que detrás de cada nombre hay una dirección concreta. Los lenguajes que exponen ese nivel lo hacen mediante punteros.

Tipo: determina qué clase de valores puede almacenar la variable y cuánta memoria ocupa: un entero de 32 bits, un real de doble precisión de 64 bits, un carácter. El tipo también determina qué operaciones son válidas.

Tamaño: la cantidad de bytes que ocupa en memoria. Depende del tipo y, en arreglos y estructuras, de sus dimensiones.

Valor: el dato almacenado en esa dirección en un momento dado.

En Scilab se pueden definir variables en las que se almacenan valores numéricos o alfanuméricos. Estas variables quedan guardadas en la memoria RAM de la computadora y pueden utilizarse en diferentes cálculos mientras dure la ejecución del programa o no se redefina la variable. Una variable es creada simplemente asignándole un valor. Por ejemplo, anteriormente se definió la variable `t` introduciendo en ella la suma $2 + 3$.

En computación suelen usarse distintos tipos de datos: numérico entero, numérico flotante, texto alfanumérico, etc. Scilab determina el tipo de la variable a partir del contenido asignado a la variable, por lo tanto se puede cambiar de tipo simplemente cargando un nuevo valor. Como ejemplo, a continuación se carga en la variable `t` un número flotante y luego un texto alfabético.

```
-->t=2.56
 t  =

    2.56

-->t="Scilab"
 t  =

 Scilab
```

Los tipos de datos básicos más importantes que maneja Scilab son enteros, punto flotante (reales en sus distintas notaciones) y cadenas de caracteres. Como ejemplos tenemos:

`4532`	entero
`3.5`	punto flotante
`-4.1234`	punto flotante
`-31.45e+11`	punto flotante
`"scilab"`	cadena de caracteres

Como se aprecia, los textos alfanuméricos (cadenas de caracteres) deben introducirse entre comillas para que Scilab los reconozca como tales y no los confunda con identificadores de variables.

Más adelante veremos que Scilab utiliza tipos más complejos de variable que permiten manejar números complejos, matrices y polinomios. La posibilidad de almacenar matrices en una variable y realizar diversas operaciones con ellas permite que Scilab pueda llevar a cabo operaciones complejas con suma facilidad, convirtiéndolo en un poderoso lenguaje de cálculo. Las mayúsculas y minúsculas son distinguidas por Scilab. Esto significa que `t` y `T` son dos variables diferentes, es decir se guardan por separado en la memoria. Por ejemplo:

```
-->t = 2.56
 t  =

    2.56

-->T = 3.1e3
 T  =

    3100.

-->t, T
 t  =

    2.56
 T  =

    3100.
```

Como se aprecia, `t` y `T` se guardan por separado y pueden ser utilizadas como tales a posteriori.

Es posible pedir en cualquier momento un listado de las variables disponibles en la memoria. Para ello se ejecuta el comando `who`, que muestra en pantalla el nombre de todas las variables que está utilizando Scilab durante la sesión de trabajo, tanto las definidas por defecto como por el usuario.

Con el comando `who_user` se muestra un listado de todas las variables definidas por el usuario, lo cual es muy útil para no redefinir variables. En el ejemplo vemos que la lista contiene a las variables `t` y `T`, que se definieron en el ejemplo anterior.

```
-->who_user
 Las variables de usuario son:

 T    t    WSCI

 Usando 24 elementos de 4990713
```

Al comenzar su trabajo, Scilab define un conjunto de variables (variables predeterminadas o del entorno Scilab). No deben usarse identificadores de variables similares a las predeterminadas por Scilab ya que su uso está reservado.

Los nombres (identificadores) de las variables pueden tener hasta 24 caracteres de longitud. Deben comenzar con una letra o con el símbolo `$`. El resto de los símbolos pueden ser letras, números, `#`, `_`, `$`, `!`. Por ejemplo, los siguientes nombres son válidos:

`Primer_valor`, `b33`, `$signo`, `$123`, `masaInicial`

Scilab posee la variable predeterminada `ans` que almacena el último valor calculado. El uso de la misma permite realizar operaciones posteriores empleando este valor.

```
-->4.5+0.23
```

```
 ans  =

    4.73

-->ans*2
 ans  =

    9.46
```

Como se aprecia, la variable `ans` siempre almacena el último valor calculado. En el ejemplo se realiza un segundo cálculo operando con el resultado anterior, luego de esta segunda instancia `ans` adopta el nuevo valor.

Para facilitar el trabajo, Scilab guarda en memoria todas las operaciones sucesivas que se están realizando. Si se quiere utilizar una de ellas a posteriori, basta presionar las teclas del cursor (flecha arriba o abajo) para recorrer la lista. Los comandos utilizados van apareciendo y uno puede seleccionar y ejecutar aquel que se quiere utilizar por segunda vez.

Scilab también permite realizar varias operaciones en una misma orden, separando cada orden o comando con una coma.

```
-->a1=5, a2=4.35, a3=0.0023
 a1  =

    5.
 a2  =

    4.35
 a3  =

    .0023
```

En la ejecución de una orden, si ésta se encuentra finalizada por un punto y coma, no se muestra el resultado de la ejecución en la ventana de Scilab, pero la operación se lleva adelante. (Esta forma de ejecutar órdenes se utiliza mucho en la confección de programas).

```
-->a1=5; a2=4.35; a3=0.023;
-->
```

4.6.3. Variables y operaciones básicas

En esta sección se presentan las operaciones elementales que pueden realizarse con Scilab. Observe atentamente el resultado obtenido luego de ejecutar cada instrucción e identifique el tipo de dato involucrado en cada operación.

Cuando una instrucción produzca un resultado inesperado, utilice la ayuda del programa y discuta la causa antes de continuar.

4.6.4. Números reales en la computadora

Un real de doble precisión tiene unos 16 dígitos significativos y un rango de $10^{\pm 308}$. Eso alcanza para casi todo, pero no es $\mathbb{R}$: `0.1 + 0.2 == 0.3` da falso, restar dos números casi iguales pierde dígitos (cancelación), y sumar muchos números chicos a uno grande los pierde. Las consecuencias prácticas son dos: nunca comparar reales con `==` sino con una tolerancia (`abs(a - b) <1e-9`), y desconfiar de un resultado que depende de la diferencia de dos cantidades grandes Higham 2002; Goldberg 1991a; Baudin 2010. Scilab también tiene enteros y complejos (`2 + 3*%i`), y opera con complejos sin aviso: `sqrt(-1)` da `i`.

4.6.5. Números complejos

Scilab maneja los números complejos en forma sencilla. Para eso define internamente una variable, `%i`, que contiene el número imaginario i. Un complejo cualquiera se expresa usando la parte real más la imaginaria multiplicada por `%i`. Por ejemplo:

$$a = 5 + 7 \cdot \%\mathrm{i}$$

Las funciones de Scilab admiten números complejos como variables del cálculo y pueden expresar sus resultados como números. Por ejemplo:

```
-->b = sqrt(-4)     //se obtiene la raíz cuadrada
 b  =

    2.i

-->c=sqrt(1+%i)
 c  =

    1.0986841 + .4550899i

-->c^2
 ans  =

    1. + i
```

Con más detalle:

Para el complejo $i = \sqrt{-1}$, Scilab utiliza `%i`, y las operaciones con números complejos (suma, resta, multiplicación, división) utilizan los mismos símbolos que para los reales (`+, -, *, /`).

```
-->z=3+4*%i
 z  =
    3. + 4.i

-->real(z)
 ans  =
    3.
```

```
-->imag(z)
 ans  =
    4.

-->conj(z)
 ans  =
    3. - 4.i

-->abs(z)
 ans  =
    5.

-->[r,tita]=polar(z)
 tita  =
    0.9272952 - 8.327D-17i
 r  =
    5.

-->atan(imag(z)/real(z))
 ans  =
    0.9272952
```

Las funciones de Scilab usadas para funciones reales elementales que tienen generalizaciones en complejos, se pueden usar también para los complejos, por ejemplo, `sin, cos, log, ...`

4.6.6. Operadores

Scilab incluye numerosos operadores. Los operadores son símbolos que actúan sobre uno o más datos y producen un resultado. Algunos operadores se usan para calcular, otros para construir vectores o matrices, otros para extraer partes de una matriz y otros para comparar valores.

Aritméticos básicos

Los operadores aritméticos básicos son:

Operador	Significado
+	suma
-	resta o cambio de signo
*	multiplicación
/	división
^ o **	potencia

Además se dispone de los paréntesis `( )` para agrupar operaciones. Las reglas de prioridad son las habituales en matemática: primero se evalúa lo que está entre paréntesis, luego las potencias, después multiplicaciones y divisiones, y finalmente sumas y restas.

```
2 + 3*4
(2 + 3)*4
2^3
2**3
```

Relacionales

Los operadores relacionales comparan valores y producen resultados lógicos o booleanos. El valor lógico es un tipo de dato en sí mismo, distinto del número y de la cadena: sus dos valores se escriben `%t` (verdadero) y `%f` (falso), también `%T` y `%F`, y `typeof` devuelve `boolean`. Un valor lógico se convierte en número si se lo usa en aritmética (`%t + 1` da 2), pero no al revés: un número no es verdadero ni falso hasta que se lo compara. Los operadores relacionales son:

Operador	Significado
==	igual
<>	distinto
<	menor
>	mayor
<=	menor o igual
>=	mayor o igual

```
x = [1, 5, 8];

x > 4
x == 5
```

Lógicos

Los operadores lógicos permiten combinar condiciones. Los más usados son `&` para la conjunción, `|` para la disyunción y `~` para la negación.

```
x = [1, 5, 8];

(x > 2) & (x < 7)
(x < 2) | (x > 7)
~(x == 5)
```

Estas operaciones también pueden aplicarse a vectores y matrices; en ese caso actúan elemento a elemento y producen un vector o matriz de valores lógicos.

4.7. Vectores y matrices

Las matrices constituyen la estructura de datos fundamental para el cálculo numérico en Scilab.

En los ejercicios siguientes se estudiarán distintas formas de construir matrices, acceder a sus elementos y efectuar operaciones entre ellas. Preste especial atención a la diferencia entre las operaciones matriciales y las operaciones elemento a elemento.

4.7.1. Definición de matrices y operaciones básicas

Uno de los aspectos más importantes del trabajo con matrices consiste en acceder únicamente a los elementos que resultan de interés.

En esta sección se utilizará el operador dos puntos (`:`) y la indexación mediante vectores para extraer filas, columnas y elementos individuales de una matriz.

Scilab está preparado para trabajar con arreglos de variables (vectores y matrices) sin dificultad alguna. Esto convierte al Scilab en un lenguaje de cálculo matricial, ya que están definidas todas las operaciones que se aplican a matrices en forma de funciones. Las matrices se definen utilizando corchetes `[ ]`. Existen tres formas para definir una matriz, las que se ilustran en los ejemplos que siguen:

```
-->c = [11,12,13;21,22,23]
 c  =

    11.    12.    13.
    21.    22.    23.

-->d = [11 12 13;21 22 23]
 d  =

    11.    12.    13.
    21.    22.    23.

-->f = [11 12 13
-->21 22 23]
 f  =

    11.    12.    13.
    21.    22.    23.
```

Como se aprecia, los elementos de la matriz se pueden separar con comas o con un espacio en blanco. Las filas sucesivas se separan con punto y coma o escribiéndolas en renglones sucesivos. En todos los casos la expresión debe comenzar y terminar con corchetes.

El elemento de una matriz `c` que ocupa la fila b y la columna d se indica como `c(b,d)`. Por ejemplo, en la matriz `c` definida anteriormente:

```
-->c(2,3)
 ans  =

    23.
```

A continuación expondremos las operaciones más sencillas. Las restantes se verán en el capítulo cuatro.

La matriz transpuesta se obtiene con el símbolo ' colocado después de la variable. Por ejemplo, refiriéndonos a la matriz c definida anteriormente:

```
-->c'
 ans  =

    11.    21.
    12.    22.
    13.    23.
```

El producto de una constante por una matriz se expresa con el signo habitual de producto *, con el que la constante se multiplica por cada elemento de la matriz.

```
-->2*c
 ans  =

    22.    24.    26.
    42.    44.    46.
```

El producto entre dos matrices se define mediante el operador para la multiplicación "*". La operación A*B indica el producto matricial entre las matrices A y B. El producto de dos matrices se puede definir sólo si el número de columnas de la matriz izquierda es el mismo que el número de filas de la matriz derecha. Por ejemplo, para matrices de 2×2:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} * \begin{pmatrix} f & g \\ h & i \end{pmatrix} = \begin{pmatrix} af + bh & ag + bi \\ cf + dh & cg + di \end{pmatrix}$$

```
-->A=[1,1;2,2];B=[2,2]';C=[3,3;1,1];

-->A*C      // producto de matrices
 ans  =

    4.     4.
    8.     8.
```

Scilab también puede efectuar el producto entre los elementos de dos matrices anteponiendo un punto al signo asterisco, es decir: .*. En el ejemplo siguiente se ve la diferencia entre ambas operaciones.

```
-->A=[1,1;2,2];B=[2,2]';C=[3,3;1,1];

-->A*B
 ans  =

    4.
    8.

-->A.*C   // producto elemento por elemento
 ans  =

    3.     3.
```

```
    2.    2.
```

4.7.2. El generador doble punto

Resulta muy útil en el entorno de Scilab la generación de vectores formados por una sucesión de valores igualmente separados. A esos efectos dispone de una forma abreviada de generación de vectores que se escribe con el siguiente formato:

```
valor_inicial : incremento : valor_final
```

Esto genera un vector cuyo primer valor es el valor "inicial" y los valores siguientes están igualmente espaciados en la cantidad "incremento" hasta llegar al valor "final". Se puede omitir el incremento y en este caso se considera por defecto el valor 1.

```
-->x=3:0.5:7
 x  =

    3.   3.5   4.   4.5   5.   5.5   6.   6.5   7.
```

Debido al uso del doble punto en la notación al operador se le conoce con el nombre de **generador doble punto** y también se utiliza en otras circunstancias. Por ejemplo, si se dispone de una matriz `c` y se escribe `d = c(:,2:3)`, el primer doble punto significa que se está considerando todas las filas, y luego las columnas de 2 a 3 con saltos de 1, es decir:

```
-->c=[-1  0  0;1  1  0;-1  1  0]
 c  =

  - 1.    0.    0.
    1.    1.    0.
  - 1.    1.    0.

-->d=c(:,2:3)
 d  =

    0.    0.
    1.    0.
    1.    0.
```

Otra forma de generar un vector linealmente espaciado es mediante la sentencia *linspace*, que genera un vector fila de n puntos igualmente espaciados entre `x1` y `x2`. `v` será un vector fila, real o complejo de acuerdo al valor de `x1` y `x2`; `n` es un entero. El valor por defecto es 100.

```
[v] = linspace(x1, x2 [, n])
```

```
-->linspace(1,2,5)
 ans  =
```

```
    1.    1.25    1.5    1.75    2.

-->linspace(1+%i,2+2*%i,5)
 ans  =

    1. + i    1.25 + 1.25i    1.5 + 1.5i    1.75 + 1.75i    2.
       + 2.i
```

4.7.3. Operaciones vectoriales y matriciales; indexación

Cuando se trabaja con vectores y matrices debe distinguirse entre operaciones matriciales y operaciones elemento a elemento. La multiplicación `*` es multiplicación matricial. Para multiplicar, dividir o elevar elemento por elemento se antepone un punto al operador: `.*`, `./`, `.^`.

```
x = [1, 2, 3];

x * x'       // producto matricial
x .* x       // producto elemento a elemento
x .^ 2       // potencia elemento a elemento
```

El operador dos puntos `:` permite generar rangos y también seleccionar partes de una matriz:

```
1:5          // de 1 a 5, con paso 1
1:2:9        // de 1 a 9, con paso 2
```

En una matriz, los índices se escriben entre paréntesis. El primer índice indica la fila y el segundo la columna. El símbolo `:` significa "todos los índices" en esa posición. El símbolo `$` representa el último índice disponible.

Extracción de bloques de una matriz

Se puede fragmentar una matriz extrayendo bloques de la misma.

```
-->A=rand(4,4)
 A  =
    0.2113249   0.6653811   0.8782165   0.7263507
    0.7560439   0.6283918   0.0683740   0.1985144
    0.0002211   0.8497452   0.5608486   0.5442573
    0.3303271   0.6857310   0.6623569   0.2320748

-->a=A(1:2,3:4)  // Intersección de las filas 1 hasta 2 y
   las columnas 3 a 4
 a  =
    0.8782165   0.7263507
    0.0683740   0.1985144

-->c1=A(:,1)     // Extrae de todas las filas (operador :)
   la columna 1
```

```
 c1  =
    0.2113249
    0.7560439
    0.0002211
    0.3303271

-->f1=A(2,:)      // Extrae de todas las columnas la fila 2
 f1  =
    0.7560439   0.6283918   0.0683740   0.1985144

-->a2=A(2:4,$)    // "$" indica la última fila o columna de
   una matriz
 a2  =
    0.1985144
    0.5442573
    0.2320748
```

También podemos extraer los elementos de una matriz mediante un vector que indique sus posiciones dentro de la misma. Una matriz puede leerse con dos índices, ej. `A(i,j)`, o con un índice que varía de 1 a $m \times n$.

```
-->A
 A  =
    0.2113249   0.6653811   0.8782165   0.7263507
    0.7560439   0.6283918   0.0683740   0.1985144
    0.0002211   0.8497452   0.5608486   0.5442573
    0.3303271   0.6857310   0.6623569   0.2320748

-->b=[1 3 7 9 14];

-->A(b)
 ans  =
    0.2113249
    0.0002211
    0.8497452
    0.8782165
    0.1985144

-->A(1,3)   // El elemento A(1,3) es el elemento A(11)
 ans  =
    0.8782165
```

Transpuesta de una matriz

```
-->A=[1+%i 2+2*%i; 3+3*%i 4-4*%i]   // se introduce la
   matriz A
 A  =
    1. + i         2. + 2.i
    3. + 3.i       4. - 4.i

-->A'              // matriz transpuesta conjugada de A
```

```
 ans  =
    1. - i         3. - 3.i
    2. - 2.i       4. + 4.i

-->A.'            // matriz transpuesta sin conjugar (se
   agrega el punto)
 ans  =
    1. + i         3. + 3.i
    2. + 2.i       4. - 4.i
```

Los corchetes permiten concatenar valores. El espacio o la coma agregan elementos en una misma fila; el punto y coma separa filas.

```
a = [1 2 3];
b = [4 5 6];

[a b]        // concatenacion horizontal
[a; b]       // concatenacion vertical
```

4.8. Cadenas de caracteres

4.8.1. Cadenas o tiras de caracteres: operaciones

Además de realizar cálculos numéricos, Scilab permite trabajar con información textual.

En esta sección se estudiarán las operaciones básicas sobre cadenas de caracteres, incluyendo concatenación, búsqueda, extracción de subcadenas y conversión entre mayúsculas y minúsculas.

Una **cadena de caracteres** es una secuencia de símbolos de texto. En cálculo científico se usan principalmente para etiquetas de gráficos, mensajes al usuario, nombres de archivos y formatos de entrada/salida.

Una cadena de caracteres (*string*) es básicamente una variable que contiene un texto que se coloca entre comillas simples o dobles. Una cadena de caracteres puede ser manipulada a través de comandos específicos. En esta sección veremos algunos que pueden ser de gran utilidad cuando se trabaja con archivos de texto.

Dos o más cadenas de caracteres pueden unirse en sucesión mediante la operación de **concatenación**, que se realiza directamente empleando el símbolo +:

```
-->t1="El resultado es "
 t1  =

 El resultado es

-->t2="positivo"
 t2  =

 positivo
```

```
-->t3=t1+t2
 t3  =

 El resultado es positivo
```

La función `date()` devuelve la fecha actual como una cadena de caracteres. Como el resultado es texto, puede concatenarse con otras cadenas:

```
fecha = date();
frase = "Hoy es " + fecha;
frase
```

Esto permite construir mensajes que incluyen la fecha de la sesión.

Las matrices de caracteres se construyen como matrices ordinarias, es decir utilizando corchetes. Un uso muy importante de las matrices de caracteres es la manipulación y creación de funciones. El siguiente ejemplo ilustra la creación de una matriz de caracteres.

```
-->A=['x' 'y';'z' 'w+v']
 A  =

 !x   y     !
 !           !
 !z   w+v   !
```

La función **length** calcula la cantidad de caracteres en la tira. Para matrices de tiras de caracteres retorna la longitud de cada uno de los elementos de la matriz.

```
-->length('123')
 ans  =

    3.

-->M=['a','bb';'ccc','dddd'];

-->length(M)
 ans  =

    1.    2.
    3.    4.
```

La función **part** permite extraer caracteres de una tira, para lo cual debemos indicar la posición de los caracteres a extraer. Para extraer una serie de caracteres, la posición se indica como una secuencia de valores.

```
-->part("abcd",1)
 ans  =

 a

-->part("abcd",[1,3])
```

```
 ans  =

 ac

-->t3="El resultado es positivo";

-->part(t3,[1:12])
 ans  =

 El resultado
```

La función **strindex(str1, str2)** proporciona la posición que la tira `str2` tiene en `str1`. Una vez que se determina la posición de `str2` se puede utilizar la función `part` para extraerlo de la tira.

```
-->t3="El resultado es positivo"

-->strindex(t3,"es")
 ans  =

    5.    14.
```

La función **strsubst(str1, str2, str3)** reemplaza `str2` con `str3` en la tira `str1`.

```
-->strsubst(t3,"positivo","negativo")
 ans  =

 El resultado es negativo
```

La función **strcat(str1, str2)** genera una tira insertando `str2` entre los caracteres de `str1`. Este último tiene que ser un vector de caracteres.

```
-->s=['a' 'b' 'c']
 s  =

 !a   b   c   !

-->strcat(s,'-')
 ans  =

 a-b-c
```

La función **ascii** permite pasar de caracteres a sus códigos numéricos y de códigos numéricos a caracteres. Por ejemplo, el código ASCII del guion es 45, y el carácter correspondiente al código 45 es nuevamente el guion.

```
-->ascii("-")
 ans  =

 45.
```

```
-->ascii(45)
 ans  =

 -
```

La función **convstr** cambia una cadena o una matriz de cadenas a mayúsculas o minúsculas. Con "u" convierte a mayúsculas y con "l" convierte a minúsculas.

```
-->convstr("laboratorio", "u")
 ans  =

 LABORATORIO

-->convstr("LABORATORIO", "l")
 ans  =

 laboratorio
```

La función **string(x)** convierte la matriz x en una matriz de caracteres:

```
-->s2=string(7)
 s2  =

 7

-->s3='la suma es '+s2
 s3  =

 la suma es 7

-->s4=string(1:5)
 s4  =

 !1   2   3   4   5   !

-->s5=s4(1)+s4(5)
 s5  =

 15

-->s5+s4
 ans  =

 !151   152   153   154   155   !
```

Los caracteres numéricos pueden convertirse a datos numéricos mediante la función **evstr** (*evaluate string*).

```
-->n1=evstr(s4)
 n1  =
```

```
 !   1.    2.    3.    4.    5.    !

-->n1(1)+n1(5)
 ans  =

    6.

-->evstr('2+2')
 ans  =

    4.
```

Como se observa en el último ejemplo, la función puede usarse para evaluar cualquier operación expresada como caracteres, incluyendo las que tienen variables que guardan números. También pueden ejecutarse sentencias expresadas como tiras de caracteres mediante la función **execstr** (*execute string*).

```
-->execstr(['a=1' 'b=sqrt(9)'])

-->a b

-->a
 a  =

    1.

-->b
 b  =

    3.
```

El resultado no se muestra pero al llamar las variables de salida se verifica que se ha realizado. Lo mismo se puede hacer con varias sentencias en un vector, como en el caso del último ejemplo. El ejemplo siguiente ejecuta varias asignaciones escritas como tiras de un vector:

```
-->execstr(['a=2' 'x1=a^1' 'x2=a^2' 'x3=a^3' 'x=[x1 x2
   x3]'])

-->x
 x  =

    2.    4.    8.
```

4.8.2. Evaluación de expresiones en cadenas

Scilab permite evaluar expresiones guardadas como texto. Esto es útil cuando una fórmula se lee desde un archivo, se escribe por teclado o se construye durante la ejecución del programa.

La función principal para esto es `evstr`. Recibe una cadena y devuelve el resultado de evaluarla como expresión de Scilab.

```
expr = "3 + 4*5";

r = evstr(expr)
```

El resultado es:

```
r = 23
```

La expresión puede usar variables ya definidas:

```
x = 10;

expr = "x^2 + 3*x + 1";

r = evstr(expr)
```

También puede usar funciones de Scilab:

```
expr = "sin(%pi/2) + sqrt(9)";

r = evstr(expr)
```

El resultado es:

```
r = 4
```

Para ejecutar instrucciones completas, como asignaciones, se usa `execstr`. A diferencia de `evstr`, que devuelve un valor, `execstr` ejecuta una sentencia.

```
cmd = "x = 3 + 4";

execstr(cmd);

x
```

Esto crea o modifica la variable `x`.

También se puede construir el nombre de una variable a partir de una cadena:

```
nombre = "temperatura";
valor = 23.5;

execstr(nombre + " = valor");
```

Luego existe la variable:

```
temperatura
```

Este mecanismo es poderoso, pero debe usarse con cuidado. Evaluar texto como código

puede producir errores difíciles de detectar y también puede ser peligroso si el texto proviene de usuarios o archivos no confiables.

Cuando sólo se necesita guardar valores asociados a nombres, suele ser más claro usar una estructura:

```
datos = struct();

nombre = "temperatura";
datos(nombre) = 23.5;
```

En resumen:

`evstr`: evalúa una expresión y devuelve su valor.

`execstr`: ejecuta una o más instrucciones escritas como texto.

4.9. Funciones definidas por el usuario

Trataremos mejor el tema funciones en el próximo tema: programación, sin embargo debemos presentarla ahora para poder avanzar en temas que dependen de las mismas.

Una función es un bloque de instrucciones que recibe datos de entrada, realiza un cálculo o una transformación, y devuelve uno o más resultados. Permite dar nombre a una operación y reutilizarla sin escribir nuevamente las mismas instrucciones.

En Scilab una función se define con `function` y termina con `endfunction`:

```
function salida = nombre(entrada)
    salida = entrada^2 + 1;
endfunction
```

Luego se la usa llamándola por su nombre:

```
y = nombre(3)
```

En este ejemplo, `entrada` es el argumento que recibe la función y `salida` es la variable que devuelve. El valor que retorna la función es el valor final asignado a la variable de salida.

Una función puede recibir varios argumentos:

```
function z = suma_cuadrados(x, y)
    z = x^2 + y^2;
endfunction

r = suma_cuadrados(3, 4)
```

También puede devolver varios resultados:

```
function [s, p] = suma_producto(a, b)
    s = a + b;
```

```
    p = a * b;
endfunction

[x, y] = suma_producto(2, 5)
```

Las funciones deben estar definidas antes de usarse. Pueden escribirse directamente en la consola, dentro de un archivo `.sce`, o en un archivo `.sci` cargado mediante `exec`.

Para que un guion reciba un dato desde el teclado y muestre un resultado se usan `input` y `disp`: `x = input(Ïngrese un valor: ")` espera un valor numérico (con el argumento `"string"` espera texto), y `disp(x)`, `disp(typeof(x))`, `disp(size(x))` lo muestran; con `mprintf("%g
n", x)` se controla el formato. Cuando el dato viene como argumento de una función no hace falta `input`.

4.9.1. deff

Scilab también permite definir funciones breves mediante `deff`. Esta forma construye una función a partir de cadenas de caracteres.

La forma general es:

```
deff("[salida]=nombre(entrada)", "salida = expresion")
```

Por ejemplo:

```
deff("[y]=f(x)", "y = x^2 + 1")

f(3)
```

El resultado es:

```
10
```

El primer argumento de `deff` indica el nombre de la función, sus variables de entrada y sus variables de salida. El segundo argumento contiene las instrucciones que se ejecutan cuando se llama a la función.

También puede usarse con varias instrucciones:

```
deff("[y]=g(x)", ["a = x^2"; "y = a + 1"])

g(4)
```

Para funciones simples, `deff` es cómodo. Para programas más largos o funciones que serán reutilizadas, suele ser más claro escribirlas con `function` y `endfunction`.

4.9.2. Aplicar una función a un vector con `feval`

Scilab no usa habitualmente una función llamada `map` como en otros lenguajes. Entre otras cosas porque scilab tiene directamente operadores y funciones vectoriales.

Lo más parecido, para cálculo numérico, es definir una función y aplicarla a todos los valores de un vector mediante `feval`.

```
deff("[y]=cuadrado(x)", "y = x.^2")
x = 1:5;
y = feval(x, cuadrado)
```

En este ejemplo, `feval` evalúa la función `cuadrado` en cada elemento de `x` y devuelve el vector de resultados. La función debe estar definida antes de usarla y se pasa como función, no como cadena de texto.

También puede aplicarse a funciones definidas por el usuario para transformar una malla de puntos antes de graficar o integrar.

Map no es estándar en Scilab.

4.10. Estructuras, contenedores y diccionarios

4.10.1. Estructuras y objetos en Scilab

Una estructura permite agrupar varios datos bajo un mismo nombre. Cada dato se guarda en un **campo**. Es útil cuando una entidad tiene varias propiedades: por ejemplo, una persona con nombre y edad, una medición con valor y unidad, o una simulación con parámetros y resultados.

En Scilab se puede crear una estructura con `struct`:

```
persona = struct();

persona.nombre = "Ana";
persona.edad = 20;
persona.notas = [8 9 7];
```

Los campos se consultan usando el punto:

```
persona.nombre
persona.edad
persona.notas
```

También se pueden modificar:

```
persona.edad = persona.edad + 1;
```

Las estructuras permiten escribir programas más ordenados, porque reúnen datos relacionados en un solo objeto. En lugar de tener muchas variables sueltas:

```
nombre = "Ana";
edad = 20;
notas = [8 9 7];
```

podemos usar una única estructura:

```
persona = struct("nombre", "Ana", "edad", 20, "notas", [8 9
   7]);
```

También es posible acceder a un campo cuyo nombre está guardado en una cadena:

```
campo = "edad";
persona(campo)
```

Esto permite construir programas más flexibles, donde los nombres de los campos pueden venir de un archivo, de una tabla o de una entrada del usuario.

Una estructura puede pasarse a una función:

```
function p = cumplir_anios(p)
    p.edad = p.edad + 1;
endfunction

persona = cumplir_anios(persona);
```

De esta forma una estructura funciona como un objeto simple: contiene datos y puede ser procesada por funciones que conocen sus campos.

4.10.2. list, cell, tlist y mlist

Scilab tiene varios contenedores para agrupar datos. Los más usados en este nivel son `list`, `cell`, `tlist` y `mlist`.

`list`: lista general. Puede guardar datos de distinto tipo y se accede por posición.

`cell`: arreglo de celdas. También puede guardar datos de distinto tipo, pero se accede con llaves.

`tlist`: lista tipada. Además de guardar datos, declara un tipo y nombres de campos.

`mlist`: lista tipada orientada a matrices y objetos más complejos. Permite definir comportamiento especial, por ejemplo para impresión, acceso u operaciones.

Una `list` es el contenedor más simple. Sirve para reunir valores heterogéneos:

```
L = list("Ana", 20, [8 9 7]);

L(1)
L(2)
L(3)
```

En una `list` no hay nombres de campos: el acceso se hace por número de posición. Por eso es útil cuando importa el orden de los elementos, pero no cuando se quiere acceder por nombre.

Una `cell` es un arreglo de celdas. Cada celda puede contener un dato de tipo diferente:

```
C = cell(2, 1);

C{1} = "Ana";
C{2} = 20;

C{1}
C{2}
```

La diferencia visible es el acceso con llaves: `C{1}` devuelve el contenido de la primera celda. Es una forma cómoda de armar arreglos heterogéneos.

Una `tlist` agrega un tipo y campos nombrados:

```
p = tlist(["persona", "nombre", "edad"], "Ana", 20);

p.nombre
p.edad
```

El primer elemento, `["persona", "nombre", .edad"]`, indica que el objeto es de tipo `persona` y que sus campos son `nombre` y `edad`.

Una `mlist` se escribe de modo parecido, pero se usa cuando se necesita un comportamiento más especializado:

```
p = mlist(["persona", "nombre", "edad"], "Ana", 20);

p.nombre
p.edad
```

La diferencia importante es que `mlist` permite asociar funciones especiales al tipo. Con ellas se puede controlar cómo se muestra el objeto, cómo se accede a sus datos o cómo responde a ciertas operaciones.

Resumen:

```
list  -> lista general, acceso por posicion
cell  -> arreglo de celdas, acceso con llaves
tlist -> lista tipada, acceso por campos
mlist -> lista tipada con posible sobrecarga de comportamiento
```

Para datos simples con campos conviene comenzar con `struct`. Para una secuencia heterogénea de datos conviene `list` o `cell`. Para objetos con tipo y campos conviene `tlist`. Para objetos con comportamiento propio conviene `mlist`.

4.10.3. Diccionarios o "hashes" en Scilab

En lenguajes como Perl, Python o JavaScript existen estructuras llamadas **hashes**, **diccionarios** o **mapas**. Permiten asociar una clave con un valor:

```
temperatura -> 23.5
presion     -> 1013
masa        -> 2.4
```

Scilab no tiene un tipo hash nativo equivalente al de Perl, pero se puede usar una estructura `struct` como diccionario simple. Cada campo de la estructura funciona como una clave.

```
datos = struct();

datos.temperatura = 23.5;
datos.presion = 1013;
datos.masa = 2.4;
```

Luego se accede a cada valor usando el nombre del campo:

```
datos.temperatura
datos.presion
datos.masa
```

También se puede usar una cadena como nombre de campo:

```
clave = "temperatura";

datos(clave)
```

Esto permite construir código donde la clave se decide durante la ejecución:

```
clave = "energia";
datos(clave) = 1500;

datos.energia
```

En este sentido, una estructura puede funcionar como una tabla de asociación entre nombres y valores.

```
datos("velocidad") = 12.8;
datos("tiempo") = 5;

distancia = datos("velocidad") * datos("tiempo");
```

La limitación principal es que las claves deben ser nombres válidos de campo. Por ejemplo, conviene usar:

```
datos.temperatura_media = 25;
```

y no claves con espacios o caracteres extraños. Para casos simples, `struct` alcanza como

reemplazo práctico de un hash.

4.10.4. `find`

La función `find` devuelve las posiciones donde se cumple una condición. Se usa cuando no interesa solamente calcular una condición lógica, sino saber en qué lugares de un vector o una matriz esa condición resulta verdadera.

Por ejemplo:

```
x = [4, 7, 2, 7, 9];

i = find(x == 7)
```

El resultado es:

```
i =

   2.    4.
```

Esto significa que el valor `7` aparece en las posiciones 2 y 4 del vector `x`. Luego esas posiciones pueden usarse para extraer valores:

```
x(i)
```

También puede usarse con desigualdades:

```
x = [4, 7, 2, 7, 9];

i = find(x > 5)
```

En este caso `i` contiene las posiciones de los elementos mayores que `5`. Si no hay ningún elemento que cumpla la condición, `find` devuelve la matriz vacía `[]`.

Con cadenas de caracteres, `find` permite ubicar la posición de una clave:

```
claves = ["temperatura", "presion", "masa"];
clave = "presion";

i = find(claves == clave)
```

Ese índice puede usarse después para buscar el valor asociado en otro vector.

4.10.5. Uso de vectores de caracteres como diccionarios

Otra forma simple de simular un hash o diccionario en Scilab es usar dos vectores paralelos: uno con las claves y otro con los valores. La clave es una cadena de texto y el valor se guarda en la misma posición del segundo vector.

Por ejemplo:

```
claves  = ["temperatura", "presion", "masa"];
valores = [23.5, 1013, 2.4];
```

La clave `"temperatura"` está en la primera posición, por lo tanto su valor está también en la primera posición de `valores`.

Para buscar un valor se puede usar `find`:

```
clave = "presion";

i = find(claves == clave);

valor = valores(i)
```

El resultado es:

```
valor = 1013
```

Esta técnica permite usar claves que no serían válidas como campos de una estructura, por ejemplo claves con espacios:

```
claves  = ["temperatura media", "presion atmosferica"];
valores = [25.1, 1013];
```

También se puede agregar una nueva clave:

```
claves($+1) = "velocidad";
valores($+1) = 12.8;
```

Y modificar un valor existente, suponiendo que la clave está en el vector:

```
clave = "temperatura";

i = find(claves == clave);
valores(i) = 24.0;
```

Para hacer el acceso más cómodo, se puede definir una función. En esta versión simple suponemos que la clave existe:

```
function valor = obtener(claves, valores, clave)
    i = find(claves == clave);
    valor = valores(i);
endfunction
```

Uso:

```
v = obtener(claves, valores, "masa");
```

Esta solución es menos elegante que un hash verdadero, porque la búsqueda recorre el vector de claves. Pero para tablas pequeñas es simple, clara y suficiente.

4.11. Funciones predefinidas de Scilab

4.11.1. Funciones elementales

Se dispone de una gran cantidad de funciones, que prácticamente cubren la mayor parte de las necesidades. Algunas de las funciones elementales disponibles son:

```
abs, acos, acosh, asin, asinh, atan, atanh
ceil, cos, cosh, cotg, coth, sin, sinh, tan, tanh
sqrt, exp, fix, floor, int, round
log, log10, log2
max, min, modulo(a,b)
```

La mayoría son reconocibles por su nombre. Por ejemplo `ceil` es la parte entera superior, `floor` es la parte entera inferior, `fix` redondea hacia cero (igual a `int`), `modulo(a,b)` proporciona el residuo de a dividido b donde ambos son enteros.

Además de las funciones elementales existe una gran cantidad de funciones especiales que se irán viendo a lo largo del curso, muchas de ellas de interés por sus aplicaciones en física e ingeniería.

Algunas operaciones ya se vieron anteriormente.

Ahora haremos un resumen de las funciones más importantes mediante ejemplos de aplicación.

4.11.2. Norma de un vector

```
-->x=rand(1,4)    // genera un vector aleatorio de 1x4
 x  =
    0.1121355   0.6856896   0.1531217   0.6970851

-->norm(x)        // norma euclídea
 ans  =
    0.9960514
```

4.11.3. Número de elementos de un vector

```
-->x=1:500;

-->n=length(x)
 n  =
    500.
```

4.11.4. Traza de una matriz

```
-->A=[2 3 1;-3 4 0;1 -1 5]
 A  =
    2.    3.    1.
  - 3.    4.    0.
    1.  - 1.    5.

-->t=trace(A)    // devuelve la suma de los elementos de la
   diagonal
 t  =
    11.
```

4.11.5. Dimensión de una matriz

```
-->[m,n]=size(A)    // devuelve el número de filas m y de
   columnas n
 n  =
    3.
 m  =
 3.
```

4.11.6. Determinante de la matriz

```
-->d=det(A)         // devuelve el determinante de la matriz
   A
 d  =
    84.
```

4.11.7. Inversa de la matriz

```
-->b=inv(A)          // equivale a 1/A
 b  =
    0.2380952  -0.1904762  -0.0476190
    0.1785714   0.1071429  -0.0357143
  - 0.0119048   0.0595238   0.2023810
```

Solamente tienen inversa las matrices cuadradas cuyo determinante es distinto de cero.

4.11.8. Valores y vectores propios de una matriz

```
-->[X,D]=spec(A)
 D  =

    2.8386444 + 2.7791238i   0                          0
```

```
    0                          2.8386444 - 2.7791238i   0
    0                          0
       5.3227111

 X  =

    0.6915854                   0.6915854
       0.0959153
    0.2655926 + 0.6355632i     0.2655926 - 0.6355632i
       -0.2175426
  - 0.2167837 + 0.0153119i   - 0.2167837 - 0.0153119i
     0.9713267
```

La matriz `A` es cuadrada, y `spec(A)` calcula los valores propios (diagonal de `D`) y vectores propios (columnas de `X`). Con frecuencia el resultado es complejo, si `A` no es simétrica.

4.11.9. Matrices especiales

```
-->zeros(3,3)     // matriz de ceros
 ans  =
    0.    0.    0.
    0.    0.    0.
    0.    0.    0.

-->eye(3,3)       // matriz identidad
 ans  =
    1.    0.    0.
    0.    1.    0.
    0.    0.    1.

-->ones(3,3)      // matriz de unos
 ans  =
    1.    1.    1.
    1.    1.    1.
    1.    1.    1.
```

4.11.10. Creación o extracción de una matriz diagonal

```
-->v=rand(1,4)
 v  =
    0.6283918   0.8497452   0.6857310   0.8782165

-->a=diag(v)  // creación de matriz con los elementos del
   vector v en la diagonal
 a  =
    0.6283918   0.           0.           0.
    0.          0.8497452    0.           0.
    0.          0.           0.6857310    0.
```

```
    0.          0.          0.          0.8782165

-->x=diag(a)  // extracción de los elementos diagonales de
   una matriz
 x  =
    0.6283918
    0.8497452
    0.6857310
    0.8782165
```

4.11.11. Matriz triangular

```
-->v=rand(4,4)
 v  =
    0.0683740   0.1985144   0.2164633   0.9329616
    0.5608486   0.5442573   0.8833888   0.2146008
    0.6623569   0.2320748   0.6525135   0.312642
    0.7263507   0.2312237   0.3076091   0.3616361

-->triu(v)    // triangular superior
 ans  =
    0.0683740   0.1985144   0.2164633   0.9329616
    0.          0.5442573   0.8833888   0.2146008
    0.          0.          0.6525135   0.312642
    0.          0.          0.          0.3616361

-->tril(v)    // triangular inferior de A
 ans  =
    0.0683740   0.          0.          0.
    0.5608486   0.5442573   0.          0.
    0.6623569   0.2320748   0.6525135   0.
    0.7263507   0.2312237   0.3076091   0.3616361
```

4.11.12. Ordenamiento de vectores

Los elementos de vectores y matrices pueden ordenarse empleando la instrucción `gsort`.

```
-->c=int(10*rand(1,10))
 c  =
    1.   6.   6.   3.   0.   5.   3.   2.   5.   4.

-->y=gsort(c)    // ordena los elementos en forma
   decreciente
 y  =
    6.   6.   5.   5.   4.   3.   3.   2.   1.   0.

-->[y,k]=gsort(c) // k vector con los índices que los
   elementos
```

```
                    // ocupan en el vector original
 k  =
    2.    3.    6.    9.    10.    4.    7.    8.    1.    5.
 y  =
    6.    6.    5.    5.    4.    3.    3.    2.    1.    0.
```

4.11.13. Máximo y mínimo de un vector

```
-->y=[6 6 5 5 4 3 3 2 1 0]
 y  =
    6.    6.    5.    5.    4.    3.    3.    2.    1.    0.

-->max(y)        // calcula el máximo del vector
 ans  =
    6.

-->[m,k]=max(y) // k es un vector que indica la posición
   donde
                 // está el máximo
 k  =
    1.
 m  =
    6.

-->min(y)        // el elemento de valor mínimo
 ans  =
    0.
```

4.11.14. Uso de funciones matemáticas predefinidas en Scilab

Funciones suma, producto y promedio

```
-->y=[6 6 5 5 4 3 3 2 1 0]
 y  =
    6.    6.    5.    5.    4.    3.    3.    2.    1.    0.

-->sum(y)
 ans  =
    35.

-->prod(y)       // producto de los elementos del vector
 ans  =
    0.

-->mean(y)       // calcula el promedio entre los elementos
 ans  =
    3.5
```

Funciones matemáticas elementales

Raíz cuadrada, valor absoluto, potenciación, logaritmo natural y en base 10, exponencial, funciones trigonométricas (seno, coseno, tangente y sus inversas), funciones hiperbólicas y sus inversas, redondeo (hacia el entero más cercano, hacia arriba, hacia abajo, hacia cero).

Cuando se aplican a vectores y matrices, estas funciones actúan elemento a elemento y producen un resultado del mismo tamaño que el argumento.

Funciones estadísticas

Suma de elementos, producto de elementos, máximo y mínimo (y sus índices), promedio o media aritmética, varianza, desvío estándar, mediana, moda, correlación entre vectores, ordenamiento de elementos.

Cuando el argumento es una matriz, estas funciones suelen operar por columnas, devolviendo un vector fila con el resultado de cada columna.

Funciones de generación de datos

Generación de vectores de valores igualmente espaciados entre un valor inicial y un final con un número determinado de puntos o con un paso determinado; generación de matrices de ceros, de unos, de la identidad; generación de números aleatorios con distribución uniforme o normal; generación de vectores y matrices a partir de funciones matemáticas.

Estas funciones se aplican a escalares y matrices, y actúan sobre cada elemento de una matriz como si se tratase de un escalar. Comprenden funciones matemáticas trascendentales y otras funciones básicas, como por ejemplo las siguientes:

sin(x) : seno	**asin(x)** : arco seno
cos(x) : coseno	**acos(x)** : arco coseno
tan(x) : tangente	**atan(x)** : arco tangente
sinh(x) : seno hiperbólico	**asinh(x)** : arco seno hiperbólico
cosh(x) : coseno hiperbólico	**acosh(x)** : arco coseno hiperbólico
tanh(x) : tangente hiperbólica	**atanh(x)** : arco tangente hiperbólica
log(x) : logaritmo natural	**sqrt(x)** : raíz cuadrada
log10(x) : logaritmo decimal	**gcd(x)** : máximo común divisor
exp(x) : función exponencial	**lcm(x)** : mínimo común múltiplo
round(x): redondeo a entero más próximo	**fix(x)**: redondea al entero más próximo a 0
floor(x) : valor entero más próximo hacia $-\infty$	**ceil(x)** : valor entero más próximo hacia $+\infty$
sign(x): signo	**abs(x)** : valor absoluto

En estas funciones, los argumentos de entrada y salida pueden ser matrices, lo que permite calcular un conjunto de datos con mayor facilidad, sin necesidad de utilizar bucles o ciclos. En el siguiente ejemplo se comprueba la relación pitagórica $\cos^2(x) + \sin^2(x) = 1$.

```
-->a=1:10
```

```
 a  =
    1.    2.    3.    4.    5.    6.    7.    8.    9.    10.

-->cos(a).^2 + sin(a).^2
 ans  =
    1.    1.    1.    1.    1.    1.    1.    1.    1.    1.
```

4.12. Cálculo numérico

Lo que sigue son las funciones de Scilab para el cálculo numérico que plantea el programa. Cada una encapsula un método que Scilab aplica por nosotros. En esta materia se aprende a usarlas, con criterio sobre qué esperar y cómo comprobar; cómo funcionan por dentro es tema de Laboratorio II y de los libros de métodos numéricos Chapra 2007; Chapra y Canale 2015; Burden y Faires 2011; Quarteroni, Sacco y Saleri 2007; Press et al. 2007; Nakamura 1997.

4.12.1. Sistema de ecuaciones lineales

La función `linsolve` calcula todas las soluciones posibles a la ecuación $\mathbf{A} \cdot \mathbf{X} + \mathbf{b} = 0$, donde $\mathbf{A}$ es la matriz de los coeficientes del sistema de ecuaciones:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n + b_1 &= 0 \\ \vdots \quad \vdots \quad \vdots \qquad \vdots \quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n + b_n &= 0 \end{aligned}$$

$\mathbf{X}$ es el vector de las incógnitas a encontrar y $\mathbf{b}$ el vector que contiene los términos independientes. Por ejemplo, se resolverá el sistema:

$$\begin{aligned} 4x + 2y + z + 3 &= 0 \\ 3x - y + 3z &= 0 \\ x + 0{,}5y - z - 4 &= 0 \end{aligned}$$

```
-->A=[4 2 1; 3 -1 3; 0.5 -1]
 A  =
    4.     2.     1.
    3.    -1.     3.
    1.     0.5   -1.

-->b=[3; 0; -4]     // b tiene que introducirse como vector
   columna
 b  =
    3.
    0.
   -4.
```

```
-->X=linsolve(A,b)
 X  =
    2.36
   -4.32
   -3.8

-->round(A*X+b)      // comprobación de la solución
 ans  =
    0.
    0.
    0.
```

El mismo tipo de sistema, escrito en la forma usual $AX = b$, puede resolverse con el operador de división izquierda: `X = A \ b`. Este operador es la forma habitual y numéricamente conveniente de resolver sistemas lineales cuadrados en Scilab. La función `linsolve`, en cambio, usa la convención $AX + b = 0$; por eso debe cuidarse el signo del vector de términos independientes al comparar ambos métodos.

```
A = [2 3 -1; 1 -1 2; 3 1 1];
b = [1; 8; 6];

x1 = A \ b;
x2 = linsolve(A, -b);
```

En ambos casos se obtiene la solución de $AX = b$.

No debe confundirse `A \ b` con `A / b`. Para un sistema escrito como $AX = b$ se usa división izquierda, `A \ b`; la división derecha corresponde a otra operación matricial.

La función `cond(A)` calcula el número de condición de una matriz. Este número indica cuán sensible puede ser la solución de $AX = b$ frente a pequeños cambios en los datos. Si `cond(A)` es cercano a 1, el sistema está bien condicionado; si es muy grande, pequeños errores en A o en b pueden producir cambios grandes en la solución.

```
c = cond(A)
```

Complemento. Un sistema lineal $Ax = b$ se resuelve con `x = A\b`, la barra invertida, que elige la factorización adecuada (eliminación gaussiana con pivoteo para cuadradas, mínimos cuadrados para rectangulares) Golub y Loan 2013; Trefethen y Bau 1997. Como se vio arriba, `linsolve(A, b)` resuelve $Ax+b = 0$, ojo con el signo, y devuelve además el núcleo si hay infinitas soluciones. `inv(A)*b` da lo mismo pero es más caro y menos preciso; se usa `inv` sólo cuando la inversa misma interesa. `det`, `rank`, `cond` dicen si el sistema tiene solución única y cuán sensible es a los datos: un número de condición grande avisa que la solución puede tener pocos dígitos buenos aunque Scilab no dé error. `spec` da autovalores y autovectores, que aparecen en las oscilaciones y en la estabilidad de los modelos lineales.

```
--> A = [4 2 1; 3 -1 3; 1 0.5 -1];  b = [-3; 0; 4];
--> x = A \ b
--> A * x - b              // comprobacion: casi cero
--> cond(A)
--> [V, D] = spec(A)       // autovectores y autovalores
```

```
--> A = [4 2 1; 3 -1 3; 1 0.5 -1];  b = [-3; 0; 4];
--> x = A \ b
--> A * x - b              // comprobacion: casi cero
--> cond(A)
--> [V, D] = spec(A)       // autovectores y autovalores
```

4.12.2. Polinomios

Un polinomio se puede definir por sus coeficientes o por sus raíces. Se define un polinomio en base a sus coeficientes con la función `poly(v, 'x', 'coef')` donde `v` es un vector que contiene los coeficientes en orden creciente y `'x'` indica la variable simbólica para el polinomio.

```
-->p=poly([2 3 5 7],"x","coef")
 p  =
                    2        3
   2 + 3x + 5x + 7x
```

Para definir un polinomio en base a sus raíces se utiliza la orden `poly(r, 'x', 'roots')`, donde `r` es un vector que contiene las raíces del polinomio.

```
-->q=poly([2 3 5],"x")
 q  =
                          2    3
  - 30 + 31x - 10x  + x
```

Escribir `q = poly([2 3 5], "x")` produce exactamente el mismo resultado, o sea, "`roots`" es el tipo de definición por defecto. Se puede reemplazar `'coeff'` por `'c'` y `'roots'` por `'r'`.

Se puede utilizar la función `roots` para hallar las raíces de un polinomio, sean éstas reales o complejas:

```
-->raices=roots(q)
 raices  =
    2.
    3.
    5.
```

Con polinomios se pueden hacer sumas, multiplicaciones, restas, multiplicación por un número. Deben ser polinomios en la misma variable. También se puede elevar un polinomio a una potencia. Por ejemplo:

```
-->v=p+q+p*q-3.1*q
 v  =
                              2        3       4        5      6
   5 - 90.1x - 51x  - 78.1x  + 170x  - 65x  + 7x
```

La función `coeff` se utiliza para obtener los coeficientes del polinomio.

```
-->p
 p  =
                        2        3
    2 + 3x + 5x + 7x

-->k=coeff(p)
 k  =
    2.    3.    5.    7.

-->c=coeff(p,2)
 c  =
    5.
```

Para evaluar un polinomio en un valor dado se utiliza la función `horner(polinomio, x)`:

```
-->p
 p  =
                        2        3
    2 + 3x + 5x + 7x

-->r=horner(p,1)
 r  =
    17.
```

Si `q` es un polinomio, es lícito utilizar la función `horner(p, q)` para obtener $p(q(x))$.

En Scilab la función estándar para evaluar polinomios es `horner`. En otros entornos de cálculo, como Matlab u Octave, suele usarse `polyval`; por eso ese nombre puede aparecer en bibliografía o ejemplos externos. En esta guía usaremos `horner`.

```
r = horner(p, 1);
```

4.12.3. Resolución de ecuaciones no lineales

Para resolver una ecuación o un sistema de ecuaciones no lineales, se puede usar el comando `fsolve`:

```
[x [,v [,info]]] = fsolve(x0, fct [,fjac] [,tol])
```

donde:

`x0:` vector de valores reales que contiene valores iniciales, aproximados, de la función,

`fct:` nombre de la definición de la función cuya solución se desea encontrar,

`fjac:` definición del jacobiano de la función a resolver,

`tol:` tolerancia del proceso iterativo de búsqueda de la solución. El proceso iterativo termina cuando el error relativo de la solución es menor o igual que `tol` (1×10^{-10} es el valor por defecto).

Hay tres parámetros de salida, únicamente el primero es obligatorio:

`x:` vector real que contiene el valor final de la solución aproximada.

`v:` vector real que contiene el valor de la función en `x`.

`info:` indicador de culminación, que puede tener 5 valores:

- 0: número inapropiado de parámetros de entrada.
- 1: se obtuvo solución con error relativo que satisface condiciones de tolerancia.
- 2: número máximo de iteraciones alcanzado. Demasiados llamados a la función.
- 3: la tolerancia (`tol`) permitida es muy pequeña. No se puede mejorar la aproximación a la solución.
- 4: la iteración no progresa o avanza muy lentamente.

El ejemplo resuelve el siguiente sistema de ecuaciones:

$$x + 7y + 10 = 0$$
$$2x + 8y + 11 = 0$$

```
-->a=[1,7;2,8]; b=[10;11];

-->deff('[y]=fsol1(x)','y=a*x+b');

-->[xres]=fsolve([100;100],fsol1)
 xres  =
    0.5
   -1.5

-->a*xres+b      // prueba
 ans  =
    0.
    0.
```

Complemento. Los sistemas no lineales, y las raíces de funciones que no son polinomios, se resuelven con `fsolve(x0, f)`, que itera desde un punto inicial con un método de Newton modificado: necesita un `x0` razonable, puede converger a una raíz que no era la buscada o no converger, y devuelve un indicador `info` que hay que mirar. Para ajustes no lineales existen `datafit` y `leastsq`.

```
function y = f(x)
    y = x - cos(x);
endfunction
--> [xr, fr, info] = fsolve(1, f)      // xr = 0.739..., info = 1
```

4.12.4. Ajuste lineal

Scilab tiene una gran variedad de funciones estadísticas, con las que es posible operar sobre variables escalares y vectoriales.

Ajustar una recta significa buscar la recta que aproxima mejor una nube de puntos. La diferencia entre cada punto medido y el valor calculado por la recta se llama residuo; la desviación estándar de esos residuos mide cuánto se apartan los datos de la recta ajustada.

`reglin` permite obtener una regresión lineal entre pares de puntos (x, y) del problema $y = a \cdot x + b$ por el método de los cuadrados mínimos:

```
[a, b, sig] = reglin(x, y)
```

`sig` es la desviación estándar del residuo. `x` e `y` son dos vectores con los datos de la muestra. El estimador `a` es la pendiente de la recta de ajuste y `b` es la ordenada.

```
-->x=[1 2 3 4 5]; y=[1.1 1.9 3.1 3.984 5.05];

-->[a,b,sig]=reglin(x,y)
 sig  =
    0.0763403
 b  =
    0.0316
 a  =
    0.9984

-->yy=a*x+b
 yy  =
    1.03    2.0284    3.0268    4.0252    5.0236
```

4.13. Cálculo en Scilab: derivación, integración, ecuaciones diferenciales

4.13.1. `diff`

La función `diff` calcula diferencias entre elementos consecutivos de un vector o una matriz. No calcula la derivada simbólica de una función; calcula diferencias discretas.

Si:

$$x = (x_1, x_2, x_3, \ldots, x_n)$$

entonces:

$$\texttt{diff}(x) = (x_2 - x_1,\ x_3 - x_2,\ \ldots,\ x_n - x_{n-1})$$

Por ejemplo:

```
x = [10 13 20 21];

dx = diff(x)
```

El resultado es:

```
dx =
   3.    7.    1.
```

Esto es útil para estudiar variaciones entre datos consecutivos. Por ejemplo, si `t` representa tiempos y `x` posiciones, entonces una aproximación de la velocidad media entre puntos consecutivos puede calcularse como:

```
t = [0 1 2 3];
x = [0 2 8 18];

v = diff(x) ./ diff(t)
```

El operador `./` indica división elemento a elemento. En este caso se divide cada cambio de posición por el cambio de tiempo correspondiente.

4.13.2. Derivada

La derivada de una función se calcula mediante el comando *derivative*, que realiza una aproximación numérica de la primera y segunda derivada de una función $F : \mathbb{R}^n \to \mathbb{R}^m$ en el punto x. En la forma más simple, la secuencia de llamado es la siguiente:

```
derivative(F, x)
[J [,H]] = derivative(F, x [,h ,order])
```

Parámetros

`F:` una función Scilab o una lista $(F, p_1, \ldots, p_k)$, donde F es una función de la forma $y = F(x, p_1, \ldots, p_k)$, donde $p_1, \ldots, p_k$ son cualquier objeto (matrices, listas, ...).

`x:` vector columna real de dimensión n.

`h:` (opcional) real, el tamaño del paso utilizado en la aproximación de diferencias finitas.

`order:` (opcional) entero, el orden de la fórmula de diferencia finita utilizada para aproximar las derivadas (`order = 1, 2` o `4`, por defecto el orden es 2).

La derivada de un polinomio se realiza mediante la función `derivat` que calcula la derivada de una matriz racional o polinómica de variable muda. Tiene la siguiente secuencia de llamado:

```
pd = derivat(p)
```

`p`: es una matriz de polinomios o racionales.

```
-->s=poly(0,'s')
 s  =
    s

-->derivat(1/s)      // -1/s^2;
 ans  =
     -1
     --
      2
      s
```

Nota: en las versiones actuales `derivative` está marcada como obsoleta y se recomienda `numderivative(F, x [, h, order])`, con los mismos argumentos y resultados. El error de la diferencia finita baja con `h` hasta que el redondeo lo hace subir de nuevo, por lo que un `h` muy chico es peor, no mejor Higham 2002.

4.13.3. Integrales

Una integral definida se escribe:

$$\int_a^b f(x)\,dx$$

El término dx, referido como el diferencial de x, indica la variable de integración.

Integración por método trapezoidal (`inttrap`)

La función **inttrap** calcula la integral numérica de una relación de pares de puntos utilizando interpolación lineal por el método trapezoidal. Cuando se trabaja con datos experimentales, comúnmente se obtiene una serie de puntos que se correlacionan con una coordenada creciente, por ejemplo el tiempo. Con los pares de puntos $(x(i), y(i))$ se ajusta la función $y(i) = f(x(i))$ donde $i = 1, 2, \ldots, n$ es un índice creciente.

La descripción del comando es:

```
[v] = inttrap([x,] y)
```

`x` es un vector con las coordenadas x con valores crecientes. Los paréntesis rectos indican que el término es opcional, siendo su valor por defecto $x = [1, 2, \ldots, m]$, donde m es la medida del vector y.

`y` es un vector de coordenadas de la ordenada que debe tener igual dimensión que x.

`v` es el valor de la integral.

En el siguiente ejemplo se realiza la integral de la función $g(x) = \sin(x) + \sin(2x)$ en el intervalo $[0, \pi]$. Se genera el vector con incremento $\pi/5$, $\pi/20$ y $\pi/100$, es decir el intervalo

discretizado en forma creciente para mejorar la precisión del cálculo. Luego se muestran distintas formas de utilizar la función.

```
-->deff('[y]=g(x)','y=sin(x)+sin(2*x)')

-->x = [0:%pi/20:%pi]; inttrap(x,g(x))
 ans  =
    1.995886

-->x = (1.0:0.1:2.0);

-->y=x^(-1);

-->inttrap(x,y)
 ans  =
    .6937714
```

Integración por cuadratura (`integrate`)

Proporciona la integral de una función externa, calculada con métodos que determinan automáticamente el intervalo para tener un error máximo dado: $|f - x| \leq \text{máx}(ea, er \cdot |f|)$ donde f es el valor exacto de la integral. El llamado general es:

```
[x] = integrate(f, v, a, b [,ea [,er]])
```

`x` es el resultado de la integral.

`f` es la función definida externamente como matriz de caracteres.

`v` es la variable de integración ingresada como una tira de caracteres.

`a` el límite inferior (número).

`b` el límite superior (número).

`ea` es el error absoluto (0 por defecto).

`er` es el error relativo (1×10^{-8} por defecto).

```
-->v=integrate('sin(x)','x',0,%pi/2)
 v  =
    1.

-->integrate('x^2','x',0,3)
 ans  =
    9.

-->deff('[y]=f(x)','y=sin(x)+sin(2*x)')

-->integrate('f(x)','x',0,%pi)
```

```
 ans  =
    2.
```

Integración mediante spline (`intsplin`)

Es una integración por interpolación, parecida a *inttrap*, solo que la función entre los valores discretos se interpola utilizando splines, pedazos discretos de curva empleados para ajustar datos entre puntos consecutivos. Se utiliza la función **intsplin**, que tiene el siguiente llamado general:

```
I = intsplin([x,] y)
```

`x` es un vector con las coordenadas x (tiene que ser creciente).

`y` es el vector de las coordenadas y, es decir los valores de la función $f(x)$.

```
-->x = 0:0.1:%pi;

-->y = sin(x) + sin(2*x);

-->intsplin(x,y)
 ans  =
    2.0008637

-->x = (-4:0.1:0);

-->y = exp(-x^2/2)/sqrt(2*%pi);

-->intsplin(x,y)
 ans  =
    .4999685
```

Integrales definidas con la función (`intg`)

La función **intg** calcula la integral definida desde a a b de la función $f(t)\,dt$ que debe ser continua. La descripción general es:

```
[I, err] = intg(a, b, f [,ea [,er]])
```

`a, b` son los límites de integración.

`f` la función externa a integrar. Puede ser una función, una lista o una cadena de caracteres.

`ea` es el error absoluto requerido en el resultado (por defecto 1×10^{-14}).

`er` error relativo requerido (por defecto 1×10^{-8}).

`I` valor de la integral.

`err` error absoluto estimado del resultado.

Esta función es similar a **integrate**, pero permite más opciones para definir la función externa f.

- Si f es una función su definición debe ser como sigue $y = f(t)$.
- Si f es una lista, esta debe ser como sigue *list(f, x1, x2, ...)*, donde f es una función con llamado de secuencia $f(t, x1, x2, \ldots)$.
- Si f es una cadena de caracteres, esta es el nombre para llamar una función definida en Fortran o C.

```
-->deff('[y]=f(x)','y=cos(x)+0.5*cos(2*x)+1.5*cos(3*x)')

-->[I,err] = intg(0,1,f)
 err   =
    1.668E-14
 I   =
    1.1393553
```

Integrales dobles

La función `int2d` puede utilizarse para calcular la integral de la función $z = f(x, y)$ sobre una región R definida por un número n de triángulos. Utiliza el método de cuadratura o curvatura. El llamado más simple es:

```
[I, err] = int2d(X, Y, f)
```

`I` es el valor de la integral.

`err` es el error estimado para el cálculo.

`X, Y` son matrices de 3 filas y n columnas representando las abscisas y las ordenadas, respectivamente, de los n triángulos en los cuales se ha dividido la región R.

`f` es la función externa (función, lista o caracteres).

Como ejemplo consideremos la integral de la función $f(x, y) = \cos(x + y)$ en la región $R = \{0 \leq x \leq 5, 0 \leq y \leq 5\}$. La figura muestra las formas posibles de triangular la región.

Figura 4.2: Triangulación de la región de integración.

```
-->deff('z=f(x,y)','z=cos(x+y)')

-->//Triángulos ABC-ACD:

-->X = [0,0;1,1;1,0]; Y = [0,0;0,1;1,1];

-->[Int,er] = int2d(X,Y,f)
 er   =
    3.569E-11
 Int   =
    .4967514
```

4.13.4. Ecuaciones diferenciales ordinarias

La resolución de un problema de valor inicial requiere resolver una ecuación diferencial de primer orden con condición inicial. Generalmente se escribe en la forma:

$$\frac{dy}{dx} = f(x, y)$$
$$y(x_o) = y_o$$

Scilab permite obtener soluciones numéricas a problemas de valor inicial. El comando a ser utilizado es `ode` (ecuación diferencial ordinaria):

```
[y] = ode([type], y0, x0, x, f)
```

`type:` es un argumento opcional que indica el método numérico de resolución que se utilizará y puede ser `'adams'` (Adams), `'rk'` (Runge-Kutta), `'rkf'` (Runge-Kutta modificado), `'fix'`, `'discrete'`, `'roots'`, `'stiff'`.

`y0:` es un vector o matriz real conteniendo las condiciones iniciales de la función.

`x0:` es un escalar real que corresponde al valor inicial de la variable independiente.

`x:` es un vector real. Contiene los valores de la variable independiente para los cuales se calcula la solución.

`f:` definición de la función a ser integrada.

`y:` es la matriz de los vectores solución. $y = [y(x(0)), y(x(1)), \ldots]$.

Por ejemplo, el programa que se muestra a continuación resuelve el siguiente problema:

$$\frac{dy}{dt} = y^2 - y\sin(t) + \cos(t)$$
$$y(0) = 0$$

Antes de utilizar la función `ode`, es necesario crear en Scilab la función f y cargarla. La función `ode` evalúa aproximaciones del valor de y en valores $t_1, t_2, \ldots, t_p$.

```
-->deff('ydot=f(t,y)','ydot=y^2-y*sin(t)+cos(t)')

-->y0=0; t0=0; t=0:0.1:%pi;

-->y=ode(y0,t0,t,f);

-->plot(t,y)
```

Se puede resolver un sistema de varias ecuaciones diferenciales ordinarias con un solo llamado de `ode`. Para ello se debe utilizar como entrada de la función f una matriz. Es decir, se resuelve $dY/dt = F(t, Y)$ donde Y es una matriz $p \times q$, la condición inicial Y_0 debe ser también una matriz $p \times q$ y el resultado de `ode` es la matriz $[Y(t_0), Y(t_1), \ldots, Y(t_T)]$.

Los métodos por defecto ajustan el paso interno para cumplir una tolerancia; el vector `x` sólo dice dónde se quiere la salida Hairer, Norsett y Wanner 1993; Ascher y Petzold 1998. Las tolerancias se pueden pasar como argumentos opcionales `rtol` y `atol`.

Una ecuación de segundo orden se convierte en un sistema de dos de primer orden definiendo el vector de estado con la variable y su derivada. Es siempre el mismo truco y conviene automatizarlo: el oscilador $\ddot{x} = -\frac{k}{m}x - \frac{b}{m}\dot{x}$ se escribe con $Y = [x; v]$ y $\dot{Y} = [v;\ -\frac{k}{m}x - \frac{b}{m}v]$.

```
k = 4; m = 1; b = 0.2;
function Ydot = osc(t, Y)
    x = Y(1); v = Y(2);
    Ydot = [v; -k/m*x - b/m*v];
endfunction
--> t = 0:0.05:20;
--> Y = ode([1; 0], 0, t, osc);
--> x = Y(1, :); v = Y(2, :);
```

Dos observaciones. La función `F` puede usar variables definidas en la consola (aquí `k`, `m`, `b`) porque Scilab las ve como globales de sólo lectura; para pasar parámetros en forma explícita se usa `list(F, p1, p2)` en lugar de `F`. Y `ode` no dice si el resultado es bueno: la comprobación es del usuario, con la energía, con un caso de solución conocida, o repitiendo con otro método o tolerancia y viendo que no cambia.

4.14. Simulación de procesos físicos

4.14.1. Modelo, EDO, método, programa: el recorrido completo

Todo lo anterior se junta en la simulación. El recorrido es siempre el mismo Cellier y Kofman 2006; Saravia Alía 2019; Campbell, Chancelier y Nikoukhah 2006: identificar el sistema y sus variables; aplicar las leyes y escribir el modelo como ecuaciones diferenciales de primer orden, con sus parámetros y condición inicial; elegir el método, que en esta materia es dejarle la elección a `ode`; escribir el programa; comparar con la teoría o con datos; usar. Se muestra con el enfriamiento de Newton, que tiene solución analítica para comparar, y luego con el péndulo, que no.

Enfriamiento. Un cuerpo a temperatura T en un ambiente a T_{amb} pierde calor a un ritmo proporcional a la diferencia: $dT/dt = -k(T - T_{amb})$, con k en 1/s. Estado: T. Parámetros: k, T_{amb}. Condición inicial: $T(0) = T_0$. Solución conocida: $T(t) = T_{amb} + (T_0 - T_{amb})e^{-kt}$.

```
// enfriamiento.sce
k = 0.05;       // 1/s
Tamb = 20;      // C
T0 = 90;        // C
function Tdot = enfria(t, T)
    Tdot = -k*(T - Tamb);
endfunction
t = 0:1:120;
T = ode(T0, 0, t, enfria);
Tex = Tamb + (T0 - Tamb)*exp(-k*t);
mprintf("error maximo: %g\n", max(abs(T - Tex)));
fprintfMat("enfria.dat", [t' T' Tex'], "%g");   // para gnuplot
```

El error máximo es del orden de 10^{-6} o menor: `ode` está haciendo bien su trabajo y el programa está bien escrito. Recién después de esa comprobación tiene sentido cambiar k o T_0 y predecir. `fprintfMat` Scilab Enterprises 2026a escribe la tabla en columnas para que gnuplot la dibuje, que es el flujo del curso.

Péndulo. Con ángulo θ desde la vertical, sin aproximar, $\ddot{\theta} = -\frac{g}{L}\sin\theta$. Estado: $[\theta; \omega]$. Parámetros: g, L. No hay solución elemental para ángulos grandes, así que la comprobación es la conservación de la energía por unidad de masa, $E = \frac{1}{2}L^2\omega^2 + gL(1 - \cos\theta)$, que debe quedar constante.

```
// pendulo.sce
g = 9.81; L = 0.5;
function Ydot = pend(t, Y)
    Ydot = [Y(2); -g/L*sin(Y(1))];
endfunction
t = 0:0.01:10;
Y = ode([%pi/2; 0], 0, t, pend);
theta = Y(1, :); omega = Y(2, :);
E = 0.5*L^2*omega.^2 + g*L*(1 - cos(theta));
mprintf("variacion relativa de E: %g\n", (max(E) - min(E))/E(1));
// periodo: tiempos en que theta cruza cero hacia abajo
i = find(theta(1:$-1) > 0 & theta(2:$) <= 0);
mprintf("periodo aprox: %g s\n", mean(diff(t(i))));
mprintf("periodo lineal 2*pi*sqrt(L/g): %g s\n",
   2*%pi*sqrt(L/g));
```

La energía varía en una parte en 10^6 o menos, y el período sale mayor que el de la aproximación lineal, como debe ser para $\pi/2$. Los dos guiones tienen la misma forma, parámetros, función de estado, tiempos, `ode`, comprobación, salida, y esa forma es la que se repite en todos los prácticos de simulación. Modelos de parámetros concentrados más grandes, circuitos, sistemas térmicos con varias masas, se escriben igual, con más filas en el vector de estado Karnopp, Margolis y Rosenberg 2012; cuando la red es grande conviene una herramienta que arme las ecuaciones sola, como Xcos, Simusol o Modelica Scilab Enterprises 2026d; Saravia Alía 2015; Modelica Association 2026.

4.14.2. Errores y comprobaciones

Una simulación tiene tres fuentes de error, y conviene saber cuál se está mirando: el error del modelo (las hipótesis, lo que se dejó afuera), el error del método (la discretización, la tolerancia de `ode`) y el error de redondeo (los 16 dígitos). El primero se ve comparando con datos; los otros dos, comparando con una solución conocida, con una cantidad conservada, o repitiendo con otra tolerancia o método y viendo cuánto cambia. Un resultado que cambia al cambiar el método no es un resultado. `ode` acepta las tolerancias como argumentos opcionales, `rtol` y `atol`, que se pueden apretar para ver si el resultado se mueve.

4.15. Ver y comunicar los resultados

4.15.1. Visualización rápida con `plot`

Hasta este punto se ha mostrado como usar Scilab principalmente como una calculadora científica. Sin embargo, en muchos problemas resulta útil observar gráficamente los resultados obtenidos.

Scilab dispone de un completo sistema para la generación de gráficos científicos. Pero en la materia no usaremos las funciones gráficas de Scilab sino las de Gnuplot, aunque las explicaremos más adelante.

Por ahora utilizaremos el comando `plot()` de Scilab, que permite obtener gráficos sencillos con muy pocas instrucciones.

Por lo tanto en esta sección el objetivo no es aprender gráficos, sino utilizar la representación gráfica como una ayuda para comprender funciones, verificar resultados y detectar errores en los programas.

Primer ejemplo

```
x = 0:0.1:2*%pi;
y = sin(x);

plot(x,y)
```

Se obtiene la representación de la función seno en el intervalo $[0, 2\pi]$.

Segundo ejemplo

```
x = -5:0.1:5;

plot(x,x.^2)
```

Obsérvese el uso del operador `.^`, que eleva al cuadrado cada elemento del vector.

Tercer ejemplo

```
x = -2:0.01:2;

plot(x,exp(x))
```

Este gráfico permite apreciar el rápido crecimiento de la función exponencial.

Comparación de funciones

También es posible representar varias curvas simultáneamente.

```
x = 0:0.1:2*%pi;

plot(x,sin(x),x,cos(x))
```

El gráfico permite comparar ambas funciones sobre el mismo sistema de ejes.

Graficar datos experimentales

No siempre se representan funciones matemáticas. También pueden graficarse datos almacenados en vectores.

```
t = [0 1 2 3 4 5];
v = [0 2 4 3 5 7];

plot(t,v)
```

Este tipo de representación será muy utilizada durante las prácticas de laboratorio para visualizar resultados de simulaciones y mediciones.

Observaciones

- `plot(x,y)` representa los valores del vector `y` en función de `x`.
- Ambos vectores deben tener la misma cantidad de elementos.
- Si solamente se escribe `plot(y)`, Scilab utiliza automáticamente como eje horizontal los índices del vector.
- Más adelante se estudiarán títulos, leyendas, colores, estilos de línea, múltiples gráficos y las demás posibilidades del sistema gráfico de Scilab.

4.15.2. De Scilab a LaTeX

Scilab incluye la función `prettyprint` que exporta matrices y expresiones al formato LaTeX, MathML o HTML.

```
--> A = [1 2; 3 4];
--> prettyprint(A, "latex")
 ans  =
  \left(\begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array}\right)

--> prettyprint(%pi/2, "latex")
--> prettyprint([1/2, sqrt(3)/2; -sqrt(3)/2, 1/2], "latex")
```

La cadena devuelta se pega directamente en un documento LaTeX dentro de un entorno matemático:

```
\[
  M = \left(\begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array}\right)
\]
```

Para *gráficos*: exportar con `xs2pdf(gcf(), "figura.pdf")` y luego insertarlo con `\includegraphics{f`
Ese flujo (el script de Scilab genera tanto números como figuras, LaTeX los integra) es la manera estándar de mantener un informe científico reproducible.

4.16. Python interactivo como calculadora vectorial

4.16.1. Por qué mirar Python después de Scilab

Todo lo hecho en la consola de Scilab (variables, vectores, matrices, funciones predefinidas) se puede repetir en Python casi línea por línea. Python no nació como entorno de cálculo: es un lenguaje de propósito general Python Software Foundation 2026c al que se le agregan bibliotecas para que trabaje como uno. Tres piezas alcanzan para llegar al mismo lugar que Scilab: NumPy para los arreglos Harris, Millman, Walt, Gommers, Virtanen, Cournapeau, Wieser et al. 2020, SciPy para el cálculo numérico Virtanen, Gommers, Oliphant, Haberland, Reddy, Cournapeau et al. 2020 y Matplotlib para los gráficos Hunter 2007b, que se ve en la unidad de gráficos. Esta unidad se limita al uso interactivo: escribir un comando, ver el resultado, escribir otro. Programar viene después.

4.16.2. Tres consolas

El intérprete se abre con `python3` y muestra el indicador `>>>`. Es suficiente pero incómodo: sin colores, sin completar nombres, sin recordar bien la historia. IPython Pérez y Granger 2007a es la consola mejorada: se abre con `ipython3`, completa con tabulador, muestra la ayuda de cualquier objeto agregando `?` al nombre, numera las entradas y salidas y guarda la historia entre sesiones. Jupyter Kluyver, Ragan-Kelley, Pérez et al. 2016b es la misma idea dentro del navegador: un cuaderno con celdas de código, texto y resultados que se guarda como archivo y se puede volver a ejecutar; es el equivalente de un guion de Scilab

con los resultados incrustados. Para esta unidad IPython es lo más parecido a la consola de Scilab.

```
$ ipython3
In [1]: 2**10
Out[1]: 1024
In [2]: import numpy as np
In [3]: np.sqrt?
```

La segunda línea es la diferencia visible: en Scilab `sqrt` existe apenas se abre la consola; en Python hay que importar NumPy y llamarla `np.sqrt`. La convención `import numpy as np` es universal y conviene adoptarla desde el primer día.

4.16.3. Variables: nombre, tipo, tamaño, dirección

Lo que en Scilab se ve con `typeof`, `size` y `who`, en Python se ve con `type`, `len` o `.shape`, y `id`. La variable sigue teniendo nombre, tipo, tamaño y dirección; sólo cambia cómo se preguntan.

```
In [1]: T = 35.2
In [2]: type(T)
Out[2]: float
In [3]: id(T)
Out[3]: 140234567890128
In [4]: import sys; sys.getsizeof(T)
Out[4]: 24
In [5]: T = T + 2       # asignacion, no ecuacion
In [6]: whos             # solo en IPython: lista de variables
```

Dos diferencias de fondo respecto de Scilab. La primera: en Python el escalar no es una matriz de 1×1; `3.5` es un `float`, `3` es un `int` y son tipos distintos. La segunda: la lista de Python, `[1, 2, 3]`, no es un vector numérico; `[1,2,3]*2` repite la lista, no multiplica. Para tener el vector de Scilab hay que pasar a NumPy.

4.16.4. Escalares, vectores y matrices con NumPy

El arreglo de NumPy (`ndarray`) es el objeto que hace de Python un entorno de cálculo Harris, Millman, Walt, Gommers, Virtanen, Cournapeau, Wieser et al. 2020: un bloque de números del mismo tipo con una forma (`shape`), sobre el que los operadores actúan elemento a elemento, exactamente como la matriz de Scilab con `.*`.

```
In [1]: x = np.arange(0, 10, 0.1)      # 0:0.1:10 en Scilab (fin
    excluido)
In [2]: x = np.linspace(0, 10, 101)    # linspace(0,10,101), fin
    incluido
In [3]: y = np.sin(x)
In [4]: x.shape, x.dtype
Out[4]: ((101,), dtype('float64'))
In [5]: A = np.array([[1, 2], [3, 4]])
```

```
In [6]: A * A           # elemento a elemento (.* de Scilab)
In [7]: A @ A           # producto matricial (* de Scilab)
In [8]: A.T             # traspuesta
In [9]: A[0, 1]         # indices desde 0: fila 0, columna 1
In [10]: x[-1]          # ultimo elemento ($ de Scilab)
In [11]: x[2:5]         # elementos 2, 3 y 4 (el 5 excluido)
```

Las tres trampas para quien viene de Scilab están ahí: los índices empiezan en cero, los rangos excluyen el extremo final, y el asterisco es elemento a elemento mientras que el producto matricial es `@`. Vale la pena escribir las tres en la pizarra y volver a ellas cada vez que aparecen.

4.16.5. Operaciones relacionales y lógicas

Sobre arreglos, las comparaciones devuelven arreglos de booleanos que sirven como máscara, igual que en Scilab con `find` o con índices lógicos.

```
In [1]: y > 0
Out[1]: array([False,  True,  True, ...])
In [2]: y[y > 0]              # solo los positivos
In [3]: np.sum(y > 0)         # cuantos son
In [4]: (x > 2) & (x < 5)     # y logico sobre arreglos: &, no
   'and'
In [5]: np.where(y > 0, y, 0)
```

Aquí también hay una trampa: `and` y `or` son para escalares; sobre arreglos hay que usar `&` y `|`, con paréntesis, porque tienen mayor precedencia que las comparaciones.

4.16.6. Cadenas

Las cadenas se escriben con comillas simples o dobles, se unen con `+`, se cortan con la misma notación de índices que los arreglos y se convierten con `str`, `float`, `int`. La forma más cómoda de mezclar texto y números es la cadena con formato: `f"T = {T:.2f} K"`.

4.16.7. Funciones predefinidas

NumPy trae las funciones matemáticas usuales, vectorizadas (`np.sin`, `np.exp`, `np.log`, `np.abs`, `np.round`), las constantes `np.pi` y `np.e`, y las estadísticas básicas (`np.mean`, `np.std`, `np.max`, `np.argmax`, `np.cumsum`). Lo que en Scilab está en un solo espacio de nombres, aquí está repartido: lo básico en NumPy, lo numérico (polinomios, álgebra lineal, integración, ecuaciones diferenciales) en SciPy Virtanen, Gommers, Oliphant, Haberland, Reddy, Cournapeau et al. 2020, que se ve en la bolilla de bibliotecas científicas. Para saber qué hay, IPython ofrece `np.<Tab>` para listar y `np.mean?` para leer la ayuda; la documentación oficial NumPy Developers 2026 está organizada por tema y tiene una tabla de equivalencias con MATLAB que sirve casi igual para Scilab.

4.16.8. Ayuda y espacio de trabajo

`help(np.linspace)` o `np.linspace?` muestran la ayuda; `dir(np)` lista lo que hay dentro de un módulo; `whos` en IPython lista las variables; `del x` borra una. No hay equivalente directo de `save`/`load` del espacio de trabajo completo; lo habitual es guardar arreglos con `np.save` o `np.savetxt` y volver a correr el cuaderno.

4.16.9. Tabla de equivalencias

Scilab	Python (NumPy)
`x = 0:0.1:10`	`x = np.arange(0, 10.1, 0.1)`
`linspace(0,10,101)`	`np.linspace(0, 10, 101)`
`A * B`	`A @ B`
`A .* B`	`A * B`
`A'`	`A.T`
`A(1,2)`	`A[0, 1]`
`x($)`	`x[-1]`
`size(A)`	`A.shape`
`find(y >0)`	`np.where(y >0)[0]`
`y(y >0)`	`y[y >0]`
`%pi`	`np.pi`
`typeof(x)`	`type(x)`, `x.dtype`
`who`	`whos` (IPython)

La conclusión de la unidad es la que importa: todo lo aprendido con Scilab como calculadora se traslada casi directamente a Python; lo que cambia es la notación, no la idea.

4.17. GNU Octave interactivo como calculadora vectorial

4.17.1. Por qué mirar Octave después de Scilab

Octave es el pariente más cercano de Scilab: nació en 1988 como herramienta para un curso de ingeniería química, se publicó como software libre en 1993 y desde entonces creció como implementación libre del lenguaje de MATLAB Eaton 2001; Eaton et al. 2024. Todo lo hecho en la consola de Scilab (variables, vectores, matrices, funciones predefinidas) se repite en Octave casi sin cambiar una tecla, y lo poco que cambia es porque Octave sigue a MATLAB donde Scilab eligió otro camino. Por eso vale la pena verlo: quien maneja Scilab maneja Octave en una tarde, y quien maneja Octave puede leer casi todo el material de MATLAB, que es enorme Quarteroni, Saleri y Gervasio 2014. Esta unidad se limita al uso interactivo: escribir un comando, ver el resultado, escribir otro. Programar viene después.

4.17.2. Dos consolas

La consola de texto se abre con `octave-cli` en la terminal y muestra el indicador `>>`; es la más parecida a `scilab-cli`. La interfaz gráfica se abre con `octave` o `octave -gui` (o desde el icono del menú) y trae, como la de Scilab, la consola, el editor, el explorador de variables, el historial de comandos y la ayuda en paneles. Es la misma calculadora en dos envolturas: los comandos son idénticos, y para esta unidad la consola alcanza. Existe además Jupyter con el núcleo de Octave (`octave_kernel`), que da un cuaderno como el de Python, pero no se lo necesita aquí.

```
$ octave-cli
>> 2^10
ans = 1024
>> sqrt(2)
ans = 1.4142
>> help sqrt
```

La primera diferencia visible: en Scilab el último resultado también se llama `ans`, pero la potencia es `^` igual, y `sqrt` existe apenas se abre la consola sin importar nada, como en Scilab. Hasta aquí no hay diferencia con Scilab salvo el indicador.

4.17.3. Variables: nombre, tipo, tamaño, dirección

Lo que en Scilab se ve con `typeof`, `size` y `who`, en Octave se ve con `class`, `size` y `who`/`whos`. La variable sigue teniendo nombre, tipo, tamaño y dirección; sólo cambia cómo se preguntan.

```
>> T = 35.2
T = 35.200
>> class(T)
ans = double
>> size(T)
ans =
   1   1
>> whos T
  Attr   Name        Size                     Bytes  Class
         T           1x1                          8  double
>> T = T + 2;       % asignacion, no ecuacion; el ; suprime la
   salida
>> who
```

Dos observaciones. La primera: igual que en Scilab, el escalar es una matriz de 1×1, y todo número es `double` salvo que se pida otra cosa; `whos` muestra los bytes que ocupa, que es lo más cerca que se llega a la dirección. La segunda: el comentario es `%` o `#`, no `//`, y el punto y coma al final de la línea suprime la salida, exactamente como en Scilab.

4.17.4. Escalares, vectores y matrices

La matriz es el tipo básico, con la misma notación de corchetes que Scilab: espacio o coma separan elementos, punto y coma separa filas, dos puntos genera rangos, y `*` es el producto matricial mientras que `.*` es el elemento a elemento. La única diferencia de fondo está en el último elemento: Scilab usa `$`, Octave usa `end`.

```
>> x = 0:0.1:10;                % 101 valores, fin incluido, como
   Scilab
>> x = linspace(0, 10, 101);    % lo mismo
>> y = sin(x);
>> size(x), class(x)
>> A = [1 2; 3 4]
>> A .* A           % elemento a elemento
>> A * A            % producto matricial
>> A'               % traspuesta (conjugada)
>> A(1, 2)          % indices desde 1: fila 1, columna 2
>> x(end)           % ultimo elemento ($ de Scilab)
>> x(3:5)           % elementos 3, 4 y 5, extremos incluidos
>> A(:, 1)          % toda la primera columna
>> A(7)             % septimo elemento por columnas, como Scilab
```

Aquí no hay trampas: índices desde 1, rangos con el extremo incluido, `*` matricial y `.*` elemento a elemento, recorrido por columnas. Es Scilab con `end` en vez de `$`. Las diferencias con Scilab que sí muerden aparecen en otro lado y conviene listarlas de una vez: la cadena se escribe con comillas simples o dobles pero las dobles interpretan `\n`; el operador de potencia matricial es `^` y el elemento a elemento `.^`, igual; el "no igual" se escribe `~=` o `!=` en lugar de `<>`; el "no" lógico es `~` o `!`; las constantes no llevan `%`: `pi`, `e`, `i`, `Inf`, `NaN`, `eps`, y no están protegidas, así que `pi = 3` las pisa; y `true`/`false` en lugar de `%t`/`%f`.

4.17.5. Operaciones relacionales y lógicas

Sobre matrices, las comparaciones devuelven matrices lógicas que sirven como máscara, exactamente como en Scilab.

```
>> y > 0
>> y(y > 0)              % solo los positivos
>> sum(y > 0)            % cuantos son
>> find(y > 0)           % indices, como en Scilab
>> (x > 2) & (x < 5)     % y logico elemento a elemento
>> any(y > 0.99), all(y < 2)
```

Octave distingue, como C, entre `&` y `&&`: el simple actúa elemento a elemento sobre matrices, el doble sólo sobre escalares y con cortocircuito. Sobre vectores hay que usar el simple.

4.17.6. Cadenas

Las cadenas van entre comillas simples (literal) o dobles (con secuencias de escape); se unen con corchetes, `[s1 s2]`, o con `strcat`; se miden con `length`; se convierten con `num2str`, `str2num`, `str2double`; y `sprintf` y `printf` dan formato al estilo de C. Aquí Octave se aparta de Scilab: `+` entre cadenas no concatena, suma los códigos ASCII, y una matriz de cadenas es una matriz de caracteres donde todas las filas deben tener el mismo largo, o bien un `cell` de cadenas, `{'a', 'bb'}`.

```
>> s = ['T = ' num2str(35.2) ' K']
>> printf("T = %.1f K\n", 35.2)
>> nombres = {'Nombre', 'Apellido'};  nombres{2}
>> length(s), upper(s), strrep(s, 'K', 'C')
```

4.17.7. Funciones predefinidas

Todo está en un solo espacio de nombres, como en Scilab, y con nombres casi siempre iguales: `sin`, `exp`, `log`, `log10`, `log2`, `abs`, `round`, `floor`, `fix` (trunca hacia cero, el `int` de Scilab), `sqrt`, `real`, `imag`, `conj`, `mean`, `std`, `max`, `min`, `sort`, `cumsum`, `zeros`, `ones`, `eye`, `rand`, `randn`, `det`, `inv`, `eig`, `roots`, `polyval`, `polyfit`, `trapz`, `quad`, `fzero`, `fsolve`, `ode45`, `lsode`. Las diferencias de nombre con Scilab son pocas y se aprenden en la tabla de más abajo; las de comportamiento son dos que hay que saber: los polinomios de Octave son vectores de coeficientes de mayor a menor grado, no objetos como en Scilab, y `eig` devuelve autovalores y autovectores en el orden inverso a `spec`. Lo numérico avanzado que en Scilab está en el núcleo, aquí en parte está en paquetes de Octave Forge Octave Forge community 2026 (`pkg install -forge control`, `signal`, `statistics`) que se cargan con `pkg load`. Para saber qué hay, `lookfor palabra` busca en las descripciones, y la documentación oficial Eaton et al. 2024 está organizada por tema.

4.17.8. Ayuda y espacio de trabajo

`help nombre` muestra la ayuda de una función y `doc nombre` abre el manual en el punto correspondiente; `lookfor` busca por palabra clave, como `apropos` en Scilab; `who`/`whos` listan las variables; `clear x` borra una y `clear` todas; `save archivo.mat` y `load` guardan y recuperan el espacio de trabajo, y `diary` graba la sesión, igual que en Scilab. `pwd`, `cd`, `ls` y `mkdir` funcionan sin paréntesis, como en la terminal.

4.17.9. Tabla de equivalencias

Scilab	Octave
`->`	`»`
`//` comentario	`%` o `#`
`x($)`	`x(end)`
`%pi`, `%e`, `%i`	`pi`, `e`, `i` (no protegidas)
`%inf`, `%nan`, `%eps`	`Inf`, `NaN`, `eps`
`%t`, `%f`	`true`, `false`
`a <>b`	`a ~= b`
`~a`	`~a` o `!a`
`typeof(x)`	`class(x)`
`int(x)`	`fix(x)`
`s1 + s2`	`[s1 s2]`, `strcat(s1, s2)`
`string(x)`, `evstr(s)`	`num2str(x)`, `str2num(s)`
`msprintf`	`sprintf`
`stdev(x)`	`std(x)`
`gsort(x)`	`sort(x, 'descend')`
`[V,D] = spec(A)`	`[V,D] = eig(A)` (mismo orden)
`poly([1 2],'x','coeff')`	`[2 1]` (coeficientes de mayor a menor)
`horner(p, x)`	`polyval(p, x)`
`derivat(p)`	`polyder(p)`
`A\b`	`A\b`
`intg(a, b, f)`	`quad(f, a, b)`
`inttrap(x, y)`	`trapz(x, y)`
`fsolve(x0, f)`	`fsolve(f, x0)`, `fzero(f, x0)`
`reglin(x, y)`	`polyfit(x, y, 1)`
`ode(y0, t0, t, f)`	`lsode(f, y0, t)`, `ode45(f, [t0 tf], y0)`
`deff('y=f(x)','y=x^2')`	`f = @(x) x.^2`
`apropos palabra`	`lookfor palabra`
`struct`, `list`	`struct`, `cell`

La conclusión de la unidad es más fuerte que en los otros lenguajes: todo lo aprendido con Scilab como calculadora se traslada a Octave tecla por tecla, con una lista de diferencias que cabe en una tabla; y por el mismo precio se gana acceso a todo lo que se escribió para MATLAB.

4.18. Perl interactivo como calculadora

4.18.1. Qué se puede esperar

Perl es un lenguaje de propósito general y gran expresividad Wall, Christiansen y Orwant 2000a; The Perl Documentation Team 2026d: la misma operación admite varias escrituras según el nivel del programador, su forma de pensar y la metodología que use, funcional, objetos, procedural. No nació como entorno de cálculo, y para usarlo como calculadora escalar alcanza el propio lenguaje; como calculadora vectorial y numérica hace falta PDL, la Perl Data Language Glazebrook y Economou 1997; PDL Developers 2026a, que le da

arreglos numéricos con operaciones elemento a elemento, álgebra lineal, y, a través de sus módulos `PDL::GSL::*`, integración, raíces, derivadas e interpolación tomadas de la GNU Scientific Library Galassi et al. 2009. Lo único que PDL no trae es un integrador de ecuaciones diferenciales, que se agrega con un módulo aparte. Esta unidad muestra tres capas: qué se puede hacer con el Perl que ya viene instalado, qué agrega PDL para repetir lo hecho con Scilab, y qué módulos hacen falta para el cálculo numérico de la unidad.

4.18.2. Instalación

Perl viene con GNU/Linux y macOS. Lo demás se instala así en Debian y Ubuntu (en otras distribuciones, los nombres son parecidos; en Windows, Strawberry Perl y después `cpan`):

```
sudo apt install pdl                    # PDL con sus modulos GSL
   y la consola pdl2
sudo apt install libmath-ode-perl       # Math::ODE, integrador de
   EDO en Perl puro
sudo apt install libmath-gsl-perl       # Math::GSL, incluido
   Math::GSL::ODEIV (opcional)
sudo apt install liblatex-table-perl    # LaTeX::Table, tablas
   para el informe (opcional)
# o, para cualquier modulo de CPAN:  cpan Math::ODE
```

El paquete `pdl` de Debian está compilado con GSL, así que los módulos `PDL::GSL::INTEG`, `MROOT`, `DIFF`, `INTERP` y `MC` vienen dentro y no hay que buscarlos aparte; si se instala PDL con `cpan`, hay que tener `libgsl-dev` antes para que se compilen.

4.18.3. Consolas

Perl no trae consola interactiva. Hay tres sustitutos. El primero es la ejecución de una línea con `perl -e`, útil para una cuenta suelta. El segundo es el depurador arrancado sobre un programa vacío, `perl -de1`, que acepta expresiones y las evalúa: es la consola rudimentaria del lenguaje The Perl Documentation Team 2026b. El tercero es la consola de PDL, `pdl2`, que ya carga la biblioteca y completa nombres. Todo lo que se hace en `pdl2` se puede hacer en `perl -de1` escribiendo antes `use PDL;`, pero sin completado.

```
$ perl -e 'print 2**10, "\n"'
1024
$ perl -de1
  DB<1> $x = 3.5
  DB<2> p sqrt($x)
1.87082869338697
  DB<3> x [1, 2, 3]
  DB<4> q
$ pdl2
pdl> p sqrt(pdl(2,3,4))
[1.4142136 1.7320508 2]
pdl> quit
```

En el depurador `p` imprime una expresión y `x` muestra una estructura de datos con su forma; `V` lista las variables; `q` sale. En `pdl2` `p` imprime también, y `quit` sale. Para lo escalar y de texto alcanza el depurador; para lo vectorial la comparación con Scilab se hace desde `pdl2`, más abajo. Ninguna de las dos guarda registro de la sesión: se graba desde afuera, en la terminal del sistema, con `script archivo.txt` antes de arrancar y `exit` al terminar.

4.18.4. Variables: nombre, tipo, tamaño, dirección

En Perl el tipo de la variable lo dice el primer carácter del nombre: `$x` es un escalar, `@a` un arreglo, `%h` una tabla asociativa The Perl Documentation Team 2026a. El escalar guarda indistintamente un número o una cadena; es el operador el que decide cómo interpretarlo. El tamaño de un arreglo es `scalar(@a)`; la dirección se ve tomando una referencia, que al imprimirse muestra el tipo y la posición en memoria.

```
  DB<1> $T = 35.2
  DB<2> p ref(\$T)
SCALAR
  DB<3> p \$T
SCALAR(0x55d3a8c1e2b0)
  DB<4> @v = (1, 2, 3)
  DB<5> p scalar(@v)
3
  DB<6> $T = $T + 2        # asignacion; tambien $T += 2
  DB<7> p "35" + "7"       # los operadores numericos convierten:
    42
  DB<8> p "35" . "7"       # el punto concatena: 357
```

Las dos últimas líneas son la característica de Perl que hay que mostrar temprano: no hay tipo numérico ni tipo cadena en la variable; `+` suma y `.` concatena, y cada uno convierte lo que recibe. Los booleanos son 1 y la cadena vacía; infinito y NaN salen de `9**9**9` y de restarlo consigo mismo; no hay unidad imaginaria sin un módulo.

4.18.5. Escalares y arreglos sin PDL

Los operadores aritméticos son los usuales, con `**` para la potencia y `%` para el resto. Las funciones matemáticas del núcleo son pocas: `sqrt`, `sin`, `cos`, `exp`, `log`, `abs`, `atan2`, `int` (trunca); el resto viene con módulos del propio Perl: `POSIX` da `round`, `floor`, `ceil`, `log10`, `strftime`; `Math::Trig` da `pi`, `tan` y las hiperbólicas; `Math::Complex` da la unidad imaginaria `i`, `cplx`, `Re`, `Im`, y hace que `sqrt(-9)` devuelva `3i` en vez de un error The Perl Documentation Team 2026c. El arreglo de Perl es una lista, no un vector: no tiene operaciones elemento a elemento; hay que recorrerlo con `map` o con un ciclo, y eso ya es programar.

```
  DB<1> use POSIX; use Math::Trig; use Math::Complex;
  DB<2> p round(2.5); p log10(1000); p cos(pi)
  DB<3> $z = sqrt(-9); p $z; p Re($z*$z)          # 3i ; -9
  DB<4> @x = map { $_ / 10 } 0 .. 100;    # 0, 0.1, ..., 10
```

```
DB<5> @y = map { sin($_) } @x;
DB<6> p "@y[0..3]"
DB<7> p $x[-1]                          # ultimo elemento
DB<8> @s = sort { $a <=> $b } @y;       # orden numerico
```

Es exactamente el ciclo que Scilab evita: la operación se aplica una por una. Perl lo escribe corto, pero no lo vectoriza.

4.18.6. PDL: la calculadora vectorial

PDL agrega el tipo *piddle* (hoy también llamado *ndarray*): un arreglo de números del mismo tipo, con forma, sobre el que los operadores actúan elemento a elemento y las funciones matemáticas están vectorizadas PDL Developers 2026a. La consola `pdl2` es lo más parecido a Scilab que ofrece Perl.

```
$ pdl2
pdl> $x = sequence(101) / 10          # 0:0.1:10
pdl> $x = zeroes(101)->xlinvals(0, 10)    # linspace
pdl> $y = sin($x)
pdl> p $x->dims                       # 101
pdl> $A = pdl([[1, 2], [3, 4]])
pdl> p $A * $A                        # elemento a elemento
pdl> p $A x $A                        # producto matricial
pdl> p $A->transpose
pdl> p $A->at(1, 0)                   # columna 1, fila 0: PDL
   indexa (col,fila), desde 0
pdl> p $x->slice('2:4')               # elementos 2, 3, 4
   (incluido)
pdl> p sum($y > 0)                    # cuantos positivos
pdl> p $y->where($y > 0)              # solo los positivos
pdl> p PI, E                          # constantes cargadas
pdl> $z = sqrt(pdl(cdouble, -9)); p $z      # 3i: complejos
   nativos (PDL >= 2.056)
pdl> p czip(4,2)**2; p re($z), im($z), conj($z)
```

Aquí las diferencias con Scilab son las mismas que en Python (índices desde cero, `*` elemento a elemento) salvo tres: el producto matricial es `x`, la porción `'2:4'` incluye el extremo como en Scilab, y los índices van con la columna primero, `at(col, fila)` y `slice(çol,fila")`, porque PDL guarda columna primero; el índice lineal `flat` recorre por columnas, como Scilab. Los gráficos se hacen con `PDL::Graphics::Simple`, que elige un motor disponible (gnuplot entre ellos), o escribiendo los datos y llamando a gnuplot directamente, que es lo que se viene haciendo desde Scilab.

```
pdl> use PDL::Graphics::Simple
pdl> line($x, $y)
pdl> wcols $x, $y, 'seno.dat'         # dos columnas para gnuplot
```

4.18.7. Álgebra lineal, polinomios y ajuste

Con PDL cargado están también `PDL::MatrixOps` (`inv`, `det`, `eigens`, `eigens_sym`, `lu_decomp`, `lu_backsub`, `identity`), `PDL::Math` (`polyroots`, funciones especiales, `erf`), y `PDL::Fit::Linfit` para el ajuste lineal por mínimos cuadrados. No hay objeto polinomio: se trabaja con el vector de coeficientes, y evaluar (Horner), multiplicar y derivar se escriben en una línea con `map`. No hay `cond`: se arma con las normas.

```
pdl> use PDL::MatrixOps; use PDL::Fit::Linfit;
pdl> $A = pdl([[2,3,-1],[1,-1,2],[3,1,1]]); $b =
   pdl([[1],[8],[6]]);
pdl> ($lu,$perm,$par) = lu_decomp($A); $x =
   lu_backsub($lu,$perm,$b)    # A\b
pdl> p inv($A) x $b; p det($A)
pdl> ($V,$d) = eigens($A)                    # autovectores,
   autovalores
pdl> ($rr,$ri) = polyroots(pdl(1,-5,5,5,-6), zeroes(5))    #
   raices de x^4-5x^3+5x^2+5x-6
pdl> ($yfit,$coef) = linfit1d($y, cat($x, ones($x)))        #
   recta a*x+b: coef = (a,b)
```

4.18.8. Cálculo numérico: los módulos GSL de PDL

Lo que en Scilab son `intg`, `fsolve`, `numderivative`, `intsplin` e `int2d`, en PDL está en los módulos que envuelven a la GNU Scientific Library Galassi et al. 2009; PDL Developers 2026b: se cargan con `use` y trabajan con funciones anónimas de Perl.

```
pdl> use PDL::GSL::INTEG; use PDL::GSL::MROOT; use
   PDL::GSL::DIFF; use PDL::GSL::MC; use PDL::Func;
# cuadratura adaptativa: (valor, error, estado) = qag(f, a, b,
   epsabs, epsrel, subintervalos, regla)
pdl> ($I,$err) = gslinteg_qag(sub { exp(-$_[0]**2) }, 0, 1,
   1e-10, 1e-10, 1000, 6)
# raiz de un sistema no lineal (aqui una ecuacion): x0 primero,
   como fsolve de Scilab
pdl> $r = gslmroot_fsolver(pdl(2), sub { my $x=shift; sin($x) -
   $x/2 })
# derivada numerica de una funcion en un punto
pdl> ($d,$err) = gsldiff(sub { sin($_[0]) }, PI/4)
# integral por spline sobre puntos
pdl> $Is = PDL::Func->init(Interpolate=>"Spline", x=>$x,
   y=>$y)->integrate(0, 10)
# integral doble: qag anidada, o Monte Carlo en n dimensiones
pdl> ($I2,$e2) = gslmc_plain(sub { my $v=shift;
   $v->at(0)+$v->at(1) }, pdl(0,0), pdl(1,1), 100000)
```

Los trapecios y Simpson no vienen como función y se escriben en una línea con `sum` sobre porciones. Ninguna función devuelve un indicador `info` como Scilab: se comprueba evaluando el residuo o mirando el error estimado.

4.18.9. Ecuaciones diferenciales: un módulo aparte

PDL no incluye el integrador de EDO de GSL. Se agrega con un módulo, y funciona igual en `perl -de1` y en `pdl2`. El más simple es `Math::ODE` Wu 2026, Perl puro, Runge–Kutta de orden 4 con paso fijo, y alcanza para todos los ejercicios de simulación de la unidad; `Math::RungeKutta` agrega paso adaptativo, y `Math::GSL::ODEIV` Leto y Ju 2026 expone todos los integradores de GSL, incluidos los rígidos, con una interfaz más pesada. Con `Math::ODE` el patrón es el de `ode` de Scilab: función de estado que devuelve la lista de derivadas, condición inicial, tiempos y paso; `evolve` integra y `y` devuelve la lista de `[t, y1, y2, ...]`, que se pasa a piddles para comprobar y graficar.

```
pdl> use Math::ODE;
pdl> $k = 0.05; $Tamb = 20;
pdl> sub enf { my ($t,$T) = @_; return [ -$k*($T->[0]-$Tamb) ] }
        # dT/dt
pdl> $o = Math::ODE->new(step=>0.1, initial=>[90], ODE=>[\&enf],
   t0=>0, tf=>60);
pdl> $o->evolve;
pdl> $t = pdl(map { $_->[0] } @{$o->y}); $T = pdl(map { $_->[1]
   } @{$o->y});
pdl> p abs($T->at(-1) - ($Tamb + 70*exp(-$k*60)))          #
   error frente a la exacta
pdl> line($t, $T)
```

Como es de paso fijo, repetir con distintos pasos muestra el efecto directamente, cosa que los integradores adaptativos de Scilab y SciPy esconden.

4.18.10. Cadenas

Perl es excelente en ésto. Comillas dobles interpolan variables (`"T = $T K"`), simples no; `.` concatena, `length` mide, `substr` corta, `index` busca, `sprintf` da formato, `uc`/`lc` cambian mayúsculas, y las expresiones regulares están en la sintaxis con `=~`, que se ven en la parte de programación. Los piddles son numéricos: los vectores de cadenas son listas de Perl.

4.18.11. Salida a LaTeX

No hay `prettyprint`, pero una matriz en `pmatrix` se arma con dos `join`, y para tablas completas está `LaTeX::Table` Schulz 2026; `wcols` con separador `&` vuelca un piddle a columnas listas para pegar.

```
pdl> @A = ([1,2],[3,4]);
pdl> $s = '\begin{pmatrix}' . join(' \\\\ ', map { join(' & ',
   @$_) } @A) . '\end{pmatrix}'
pdl> use LaTeX::Table; p LaTeX::Table->new({data=>\@A,
   header=>[['a','b']]})->generate_string
```

4.18.12. Ayuda

`perldoc perlfunc` lista las funciones del núcleo; `perldoc -f sqrt` una sola; `perldoc PDL::Core` lo básico de PDL y `perldoc PDL::Ufunc` las funciones de reducción (`sum`, `sumover`, `avg`, `max`, `qsort`, `qsorti`, `any`, `all`, ...), que son las que más se usan en los ejercicios; `perldoc PDL::MatrixOps`, `PDL::GSL::INTEG`, `Math::ODE` lo de cada módulo. En `pdl2`, `?` seguido de un nombre abre la ayuda y `apropos` busca por palabra.

4.18.13. Tabla de equivalencias

Scilab	Perl (núcleo)	PDL y módulos
`x = 0:0.1:10`	`map {$_/10} 0..100`	`sequence(101)/10`
`sin(x)`	`map {sin} @x`	`sin($x)`
`A * B`	—	`$A x $B`
`A .* B`	—	`$A * $B`
`A'`	—	`$A->transpose`
`A(1,2)`	`$A[0][1]`	`$A->at(1,0)` (col, fila)
`x($)`	`$x[-1]`	`$x->at(-1)`
`size(A)`	`scalar(@A)`	`$A->dims`
`y(y >0)`	`grep {$_ >0} @y`	`$y->where($y >0)`
`%pi,%e,%i`	`pi` (Math::Trig), `exp(1)`, `i` (Math::Complex)	`PI`, `E`, `czip(0,1)`
`round`, `int`	`round` (POSIX), `int`	`rint`, `->long`
`typeof(x)`	`ref(\$x)`	`$x->type`
`A\b`	—	`lu_backsub(lu_decomp($A),`
`spec(A)`	—	`eigens($A)`
`roots(p)`	—	`polyroots($c, zeroes(n))`
`reglin(x,y)`	—	`linfit1d`
`intg(a,b,f)`	—	`gslinteg_qag(\&f,a,b,...)`
`fsolve(x0,f)`	—	`gslmroot_fsolver(pdl(x0),`
`numderivative(f,x)`	—	`gsldiff(\&f,x)`
`intsplin(x,y)`	—	`PDL::Func ... ->integrate`
`int2d`	—	`gslinteg_qag` anidada, `gslmc`
`ode(y0,t0,t,f)`	`Math::ODE` (módulo aparte)	idem
`prettyprint`	`join`, `LaTeX::Table`	`wcols`
`diary`	`script` (desde afuera)	idem

La conclusión: Perl como calculadora escalar sirve; como calculadora vectorial y numérica sirve con PDL y sus módulos GSL, que cubren todo lo de la unidad salvo las ecuaciones diferenciales, que piden un módulo más. Lo que distingue a Perl es su expresividad: la misma operación admite varias escrituras según el nivel del programador, su forma de pensar, y que paradigma de programación use: funcional, objetos, etc. .

Perl como calculadora escalar sirve; como calculadora vectorial y numérica sirve con PDL y sus módulos GSL, que cubren todo lo de la unidad salvo las ecuaciones diferenciales, que piden un módulo más.

4.19. JavaScript interactivo como calculadora

4.19.1. Dónde vive JavaScript

JavaScript es el lenguaje de los navegadores Ecma International 2025; MDN contributors 2026: toda computadora con un navegador tiene un intérprete y una consola esperando. Fuera del navegador se ejecuta con Node OpenJS Foundation 2026b. Como calculadora escalar es inmediato; como calculadora vectorial hace falta una biblioteca, y la más directa es math.js Jong y math.js contributors 2026, que da matrices, operaciones elemento a elemento y álgebra matricial con una sintaxis parecida a la de Scilab. Esta unidad muestra la consola primero, después math.js.

4.19.2. Dos consolas

En el navegador la consola se abre con F12 o Ctrl+Shift+I, pestaña *Console*: se escribe una expresión y se ve el resultado, con completado de nombres y ayuda al pasar el ratón. En la terminal, `node` sin argumentos abre un intérprete equivalente OpenJS Foundation 2026b. Los ejemplos que siguen valen en las dos, salvo los gráficos, que necesitan la página.

```
$ node
> 2**10
1024
> Math.sqrt(2)
1.4142135623730951
> .help
```

Como en Python, las funciones matemáticas están en un espacio de nombres, aquí `Math`: `Math.sin`, `Math.exp`, `Math.log`, `Math.abs`, `Math.round`, `Math.PI`, `Math.E`. Están desde el principio, no hay que importar nada.

4.19.3. Variables: nombre, tipo, tamaño, dirección

Se declara con `let` (variable) o `const` (constante); `var` es la forma vieja y conviene evitarla. El tipo lo tiene el valor, no la variable, y se pregunta con `typeof`. Hay un solo tipo numérico, `number`, de coma flotante doble: no hay entero aparte, y `0.1 + 0.2 !== 0.3` sale a la primera clase. El tamaño de un arreglo es `.length`. La dirección no se puede ver: el lenguaje no expone la memoria; lo que se puede mostrar es que dos nombres pueden apuntar al mismo arreglo.

```
> let T = 35.2
> typeof T
'number'
> T = T + 2           // asignacion; tambien T += 2
> const v = [1, 2, 3]
> v.length
3
> const w = v; w[0] = 99; v
[ 99, 2, 3 ]        // w y v son el mismo arreglo
```

```
> 0.1 + 0.2 === 0.3
false
> "35" + 7            // el + con cadena concatena
'357'
> "35" * 7            // el * convierte
245
```

Las dos últimas líneas son el pariente de Perl: la conversión la hace el operador, con la diferencia de que + en JavaScript prefiere concatenar. Comparar siempre con ===, nunca con ==, es la primera regla práctica Crockford 2008a.

4.19.4. Escalares y arreglos sin biblioteca

El arreglo de JavaScript es una lista, no un vector: `[1,2,3] * 2` no multiplica. Sí tiene métodos que aplican una función a cada elemento (`map`), filtran (`filter`) o acumulan (`reduce`), y con eso se hace lo mismo que Scilab, elemento por elemento y de forma explícita.

```
> const x = Array.from({length: 101}, (_, i) => i / 10)   // 0..10
> const y = x.map(Math.sin)
> y.filter(v => v > 0).length          // cuantos positivos
> Math.max(...y)                        // maximo
> y.reduce((s, v) => s + v, 0) / y.length   // media
> x.at(-1)                              // ultimo elemento
> x.slice(2, 5)                         // 2, 3, 4 (el 5 excluido)
> console.table(x.slice(0, 5))
```

Es la misma situación que en Perl sin PDL: escribir corto no es vectorizar. Y aparece el rasgo propio de JavaScript, la función anónima `v =>v >0` como argumento, que va a reaparecer en todos lados.

4.19.5. math.js: la calculadora vectorial

math.js Jong y math.js contributors 2026 agrega matrices, operaciones matriciales, unidades y un evaluador de expresiones. En Node se instala con `npm install mathjs`; en el navegador se carga con una etiqueta `<script>` desde un CDN. Tiene dos maneras de usarse: llamando funciones desde JavaScript, o dándole una cadena a `math.evaluate`, que entiende una sintaxis casi idéntica a la de Scilab, con `*` matricial y `.*` elemento a elemento.

```
> const math = require('mathjs')
> const x = math.range(0, 10.1, 0.1)          // 0:0.1:10
> const y = math.map(x, Math.sin)
> const A = math.matrix([[1, 2], [3, 4]])
> math.multiply(A, A)                          // producto matricial
> math.dotMultiply(A, A)                       // elemento a elemento
```

```
> math.transpose(A)
> A.subset(math.index(0, 1))                    // indices desde 0
> math.evaluate('A * A', {A})                   // matricial
> math.evaluate('A .* A', {A})                  // elemento a
   elemento
> math.evaluate('x = 0:0.1:10; sum(x .^ 2)')
> math.evaluate('5 cm + 2 inch')                // unidades
```

El evaluador es útil para mostrar la equivalencia con Scilab línea por línea; en un programa se usan las funciones. Fuera de math.js el único otro rasgo numérico que vale mencionar son los arreglos tipados (`Float64Array`), que son bloques de números contiguos como los de NumPy pero sin operaciones vectoriales.

4.19.6. Gráficos

En Node no hay ventana; el gráfico se hace en una página. Plotly.js Plotly Technologies Inc. 2026 dibuja en un elemento del documento con una llamada, y el resultado es interactivo (zoom, cursor con valores). Este es el punto donde JavaScript deja de ser calculadora y toca la unidad de interfaces web.

```
<div id="g"></div>
<script src="https://cdn.plot.ly/plotly-2.35.2.min.js"></script>
<script>
  const x = Array.from({length: 101}, (_, i) => i / 10);
  Plotly.newPlot('g', [{x, y: x.map(Math.sin), name: 'sen x'},
                       {x, y: x.map(Math.cos), name: 'cos x'}]);
</script>
```

4.19.7. Cadenas

Comillas simples o dobles son equivalentes; las comillas invertidas interpolan: `` `T = ${T.toFixed(2)} K` ``. `.length` mide, `.slice` corta, `Number("3.5")` convierte, y el formato de intercambio JSON es literalmente la sintaxis de los arreglos y objetos del lenguaje: `JSON.stringify(y)` produce texto que Python o Perl leen sin esfuerzo.

4.19.8. Ayuda

En el navegador la propia consola completa y muestra la firma de las funciones; la referencia es MDN MDN contributors 2026. En Node, `.help` lista los comandos de la consola y `Object.keys(Math)` o `Object.keys(math)` muestran qué hay en un espacio de nombres.

4.19.9. Tabla de equivalencias

Scilab	JavaScript	math.js
`x = 0:0.1:10`	`Array.from({length:101}, (_,i)=>i/10)`	`math.range(0,10.1,0.1)`
`sin(x)`	`x.map(Math.sin)`	`math.map(x, Math.sin)`
`A * B`	—	`math.multiply(A,B)`
`A .* B`	—	`math.dotMultiply(A,B)`
`A'`	—	`math.transpose(A)`
`A(1,2)`	`A[0][1]`	`A.subset(math.index(0,1))`
`x($)`	`x.at(-1)`	`x.get([x.size()[0]-1])`
`size(A)`	`A.length`	`math.size(A)`
`y(y >0)`	`y.filter(v =>v >0)`	`math.filter(y, v =>v >0)`
`%pi`	`Math.PI`	`math.pi`
`typeof(x)`	`typeof x`	`math.typeOf(x)`
`plot(x, y)`	`Plotly.newPlot(...)`	—

La conclusión: JavaScript como calculadora está en todas partes y no hay que instalar nada, pero la calculadora vectorial es una biblioteca y el gráfico exige una página. Su lugar propio no es el cálculo sino mostrar el resultado del cálculo en el navegador, y eso es la unidad de interfaces web.

4.20. Instalación de software para la materia, automatizado

Ojo, la instalacion tarda mucho, incluso compila cosas

Listing 4.1: Script de instalación del software utilizado en los prácticos

```
#!/usr/bin/env bash
set -u

# Instalador idempotente para el conjunto de practicos
   eu01--eu10 en GNU/Linux.
# Conserva lo ya incluido para eu04 (Scilab, Maxima, Octave,
   Python, Perl,
# Node/Jupyter, etc.) y agrega el software usado en los otros
   practicos.
# Soporta Debian/Ubuntu/Mint, Fedora/RHEL y Arch/Manjaro en modo
   best-effort.
#
# Los modulos de npm se instalan globalmente (sudo npm install
   -g) y se fija
# NODE_PATH en /etc/profile.d/node-global.sh para que require()
   los encuentre
# desde cualquier directorio.
#
# No desinstala ni actualiza software existente. Antes de
   instalar cada programa
```

```
# comprueba si el comando correspondiente ya esta disponible.

log() { printf '%s\n' "$*"; }
have() { command -v "$1" >/dev/null 2>&1; }

if [ ! -r /etc/os-release ]; then
  log "ERROR: no se pudo identificar la distribucion
     (/etc/os-release)."
  exit 1
fi
. /etc/os-release
ID_LIKE_ALL="${ID:-} ${ID_LIKE:-}"

case "$ID_LIKE_ALL" in
  *debian*|*ubuntu*) PM=apt ;;
  *fedora*|*rhel*)  PM=dnf ;;
  *arch*)           PM=pacman ;;
  *)
    log "ERROR: distribucion no contemplada automaticamente:
       ${PRETTY_NAME:-$ID}"
    exit 1
    ;;
esac

log "Sistema: ${PRETTY_NAME:-$ID}"
log "Gestor:  $PM"
log ""

APT_UPDATED=0
pkg_install() {
  case "$PM" in
    apt)
      if [ "$APT_UPDATED" -eq 0 ]; then
        sudo apt-get update || true
        APT_UPDATED=1
      fi
      sudo apt-get install -y "$@" || true
      ;;
    dnf)
      sudo dnf install -y "$@" || true
      ;;
    pacman)
      sudo pacman -S --needed --noconfirm "$@" || true
      ;;
  esac
}

install_if_missing() {
  # Uso: install_if_missing comando paquete_apt paquete_dnf
     paquete_arch
  cmd="$1"; pa="$2"; pd="$3"; pp="$4"
```

```
  if have "$cmd"; then
    log "[OK] $cmd ya esta instalado."
    return 0
  fi
  log "[INSTALA] $cmd"
  case "$PM" in
    apt)    pkg_install "$pa" ;;
    dnf)    pkg_install "$pd" ;;
    pacman) pkg_install "$pp" ;;
  esac
}

section() {
  log ""
  log
     "============================================================"
  log "$1"
  log
     "============================================================"
}

#
    ------------------------------------------------------------------
# PRACTICOS 04--05: calculo cientifico y lenguajes
#
    ------------------------------------------------------------------
section "Calculo cientifico y lenguajes"

# Scilab
if ! have scilab && ! have scilab-cli; then
  log "[INSTALA] Scilab"
  case "$PM" in
    apt) pkg_install scilab ;;
    dnf) pkg_install scilab ;;
    pacman)
      if have yay; then
        yay -S --needed --noconfirm scilab || true
      else
        log "[AVISO] Scilab suele instalarse desde AUR en Arch.
           Instale yay o Scilab manualmente."
      fi
      ;;
  esac
else
  log "[OK] Scilab ya esta instalado."
fi

# Complemento simbolico solicitado anteriormente
install_if_missing maxima maxima maxima maxima

# GNU Octave
```

```
install_if_missing octave octave octave octave

# Python + ecosistema cientifico/Jupyter
if ! have python3; then
  case "$PM" in
    apt) pkg_install python3 python3-pip python3-venv ;;
    dnf) pkg_install python3 python3-pip ;;
    pacman) pkg_install python python-pip ;;
  esac
else
  log "[OK] python3 ya esta instalado."
fi

case "$PM" in
  apt)
    pkg_install ipython3 python3-numpy python3-scipy
       python3-matplotlib jupyter-notebook
    ;;
  dnf)
    pkg_install python3-ipython python3-numpy python3-scipy
       python3-matplotlib python3-notebook
    ;;
  pacman)
    pkg_install ipython python-numpy python-scipy
       python-matplotlib jupyter-notebook
    ;;
esac

# Perl + PDL y modulos usados en los ejercicios
install_if_missing perl perl perl perl
case "$PM" in
  apt)
    pkg_install pdl libmath-ode-perl libmath-gsl-perl
       liblatex-table-perl cpanminus libgsl-dev
    ;;
  dnf)
    pkg_install perl-PDL perl-Math-GSL perl-App-cpanminus
       gsl-devel
    ;;
  pacman)
    pkg_install perl perl-app-cpanminus gsl
    if ! have pdl2 && have yay; then
      yay -S --needed --noconfirm perl-pdl || true
    fi
    ;;
esac

perl_module() {
  module="$1"
  if perl -M"$module" -e1 >/dev/null 2>&1; then
    log "[OK] Perl $module"
```

```
    return 0
  fi
  log "[INSTALA] Perl $module via CPAN/cpanm"
  if have cpanm; then
    sudo cpanm --notest "$module" || cpanm
       --local-lib="$HOME/perl5" --notest "$module" || true
  elif have cpan; then
    printf 'yes\n' | sudo cpan "$module" || true
  else
    log "[AVISO] no se encontro cpan/cpanm para instalar $module"
  fi
}

perl_module Math::ODE
perl_module Math::RungeKutta
perl_module Math::Symbolic
perl_module Math::GSL
perl_module LaTeX::Table

# JavaScript / Node.js
install_if_missing node nodejs nodejs nodejs
install_if_missing npm npm npm npm
# Los modulos de calculo se instalan globalmente (sudo npm
   install -g), para
# que require() funcione desde cualquier directorio. Como node
   NO busca en el
# directorio global de npm por si solo, se fija NODE_PATH para
   todo el sistema
# en /etc/profile.d/node-global.sh apuntando al prefijo global
   de root (que es
# distinto del del usuario si este configuro su propio prefijo o
   usa nvm).
NPM_MODULOS="mathjs numeric odex integrate-adaptive-simpson
   newton-raphson-method csv-parse @duckdb/node-api
   readline-sync"
if have npm; then
  NPM_GLOBAL_ROOT="$(sudo npm root -g 2>/dev/null)"
  for package in $NPM_MODULOS; do
    if [ -n "$NPM_GLOBAL_ROOT" ] && [ -d
       "$NPM_GLOBAL_ROOT/$package" ]; then
      log "[OK] npm $package (global en $NPM_GLOBAL_ROOT)"
    else
      log "[INSTALA] npm $package (global)"
      sudo npm install -g --no-fund --no-audit "$package" || true
    fi
  done
  NPM_GLOBAL_ROOT="$(sudo npm root -g 2>/dev/null)"
  if [ -n "$NPM_GLOBAL_ROOT" ]; then
    # NODE_PATH para todas las sesiones de login (bash, zsh, sh)
    printf 'export NODE_PATH="%s${NODE_PATH:+:$NODE_PATH}"\n'
       "$NPM_GLOBAL_ROOT" | sudo tee
```

```
        /etc/profile.d/node-global.sh >/dev/null
    sudo chmod 644 /etc/profile.d/node-global.sh
    export NODE_PATH="$NPM_GLOBAL_ROOT${NODE_PATH:+:$NODE_PATH}"
    log "[OK] NODE_PATH=$NPM_GLOBAL_ROOT fijado en
       /etc/profile.d/node-global.sh (rige en las proximas
       terminales; en esta ya esta exportado)."
  fi
fi

# Paquetes Octave citados
if [ "$PM" = apt ]; then
  pkg_install octave-control octave-signal octave-statistics
fi

#
   ------------------------------------------------------------------------
# PRACTICO 06: archivos de texto, CSV y consultas SQL (DuckDB)
#
   ------------------------------------------------------------------------
section "Practico 06: archivos, CSV y SQL (DuckDB)"

# DuckDB: ejecutable SQL usado por todos los lenguajes del
   practico.
install_if_missing duckdb duckdb duckdb duckdb
# libduckdb: biblioteca compartida que usa DBD::DuckDB (Perl)
   via FFI.
# Si no hay paquete en la distribucion, se instalan ambos desde
   duckdb.org.
if ! have duckdb || ! ldconfig -p 2>/dev/null | grep -q
   libduckdb; then
  log "[INSTALA] duckdb CLI y libduckdb desde duckdb.org (en
     /usr/local)"
  tmpd="$(mktemp -d)"
  if curl -fsSL
     https://github.com/duckdb/duckdb/releases/latest/download/duckdb_cli-li
     -o "$tmpd/cli.zip"; then
    sudo unzip -o -q "$tmpd/cli.zip" -d /usr/local/bin && sudo
       chmod +x /usr/local/bin/duckdb || true
  fi
  if curl -fsSL
     https://github.com/duckdb/duckdb/releases/latest/download/libduckdb-lin
     -o "$tmpd/lib.zip"; then
    sudo unzip -o -q "$tmpd/lib.zip" libduckdb.so -d
       /usr/local/lib && sudo ldconfig || true
  fi
  rm -rf "$tmpd"
fi

# JDBC jar de DuckDB: lo usa la parte a) del ejercicio Consultas
   (Scilab).
if [ ! -f /usr/local/share/java/duckdb_jdbc.jar ]; then
```

```
    log "[INSTALA] duckdb_jdbc.jar (para Scilab/JDBC)"
    sudo mkdir -p /usr/local/share/java
    sudo curl -fsSL
       https://repo1.maven.org/maven2/org/duckdb/duckdb_jdbc/1.5.5.1/duckdb_jd
       \
      -o /usr/local/share/java/duckdb_jdbc.jar || \
      log "[AVISO] no se pudo descargar duckdb_jdbc.jar"
fi

# Modulos Perl del practico 06: CSV y acceso SQL.
case "$PM" in
  apt)    pkg_install libtext-csv-perl libdbi-perl ;;
  dnf)    pkg_install perl-Text-CSV perl-DBI ;;
  pacman) pkg_install perl-text-csv ;;
esac
perl_module Text::CSV
perl_module DBI
perl_module DBD::DuckDB

# duckdb para Python.
if python3 -c 'import duckdb' >/dev/null 2>&1; then
  log "[OK] Python duckdb"
else
  case "$PM" in
    apt)    pkg_install python3-duckdb ;;
    dnf)    pkg_install python3-duckdb ;;
    pacman) pkg_install python-duckdb ;;
  esac
fi
if ! python3 -c 'import duckdb' >/dev/null 2>&1; then
  log "[INSTALA] duckdb para Python via pip --user"
  python3 -m pip install --user --break-system-packages duckdb
     2>/dev/null || \
    python3 -m pip install --user duckdb || \
    log "[AVISO] no se pudo instalar duckdb para Python"
fi

# Los import ESM de node no usan NODE_PATH; un node_modules en
   $HOME permite
# la resolucion por directorios padre para proyectos bajo $HOME.
if have npm; then
  NPM_GLOBAL_ROOT="$(sudo npm root -g 2>/dev/null || npm root -g
     2>/dev/null)"
  if [ -n "$NPM_GLOBAL_ROOT" ] && [ ! -e "$HOME/node_modules" ];
     then
    ln -s "$NPM_GLOBAL_ROOT" "$HOME/node_modules"
    log "[OK] ~/node_modules -> $NPM_GLOBAL_ROOT (para import
       ESM, p.ej. @duckdb/node-api)"
  fi
fi
```

```
#
   ------------------------------------------------------------------------
# PRACTICO 01: utilidades GNU/Linux, compresion y construccion
   de paquetes
#
   ------------------------------------------------------------------------
section "Utilidades GNU/Linux y compilacion"

install_if_missing tree tree tree tree
install_if_missing figlet figlet figlet figlet
install_if_missing cowsay cowsay cowsay cowsay
install_if_missing nano nano nano nano
install_if_missing vim vim-enhanced vim-enhanced vim

# Compresion, hashes y utilidades basicas
case "$PM" in
  apt) pkg_install gzip bzip2 xz-utils zip unzip coreutils
     xxhash file ;;
  dnf) pkg_install gzip bzip2 xz zip unzip coreutils xxhash file
     ;;
  pacman) pkg_install gzip bzip2 xz zip unzip coreutils xxhash
     file ;;
esac

# Herramientas de compilacion y empaquetado.
case "$PM" in
  apt)
    pkg_install build-essential gcc g++ make dpkg-dev devscripts
       fakeroot debhelper
    ;;
  dnf)
    pkg_install gcc gcc-c++ make rpm-build rpmdevtools
    ;;
  pacman)
    pkg_install base-devel gcc make
    ;;
esac

# Maquinas virtuales (practico 01): software libre, en los
   repositorios oficiales.
# QEMU/KVM + libvirt + virt-manager es la solucion completa;
   GNOME Boxes es la
# interfaz simple sobre lo mismo. Ambos importan discos .qcow2,
   .vmdk y .vdi.
# VirtualBox queda como alternativa opcional (su base es GPL,
   pero el paquete de
# extensiones es privativo y en Debian esta en contrib, no en
   main).
if have virt-manager || have gnome-boxes; then
  log "[OK] hay gestor de maquinas virtuales (virt-manager o
     GNOME Boxes)."
```

```
else
  log "[INSTALA] QEMU/KVM, libvirt, virt-manager y GNOME Boxes"
  case "$PM" in
    apt)    pkg_install qemu-system-x86 qemu-utils
       libvirt-daemon-system libvirt-clients virt-manager
       gnome-boxes ;;
    dnf)    pkg_install qemu-kvm libvirt virt-install
       virt-manager gnome-boxes ;;
    pacman) pkg_install qemu-full libvirt virt-manager
       gnome-boxes dnsmasq ;;
  esac
  sudo systemctl enable --now libvirtd 2>/dev/null || true
fi
# Que el usuario pueda correr maquinas sin sudo (rige al volver
   a iniciar sesion).
for g in kvm libvirt; do
  if getent group "$g" >/dev/null 2>&1 && ! id -nG "$USER" | tr
     ' ' '\n' | grep -qx "$g"; then
    sudo usermod -aG "$g" "$USER" || true
    log "[OK] usuario $USER agregado al grupo $g (efectivo al
       reiniciar la sesion)."
  fi
done
if [ "${INSTALAR_VIRTUALBOX:-0}" = 1 ] && ! have VBoxManage; then
  log "[INSTALA] VirtualBox (opcional, INSTALAR_VIRTUALBOX=1)"
  case "$PM" in
    apt) pkg_install virtualbox ;;
    dnf) pkg_install VirtualBox ;;
    pacman) pkg_install virtualbox ;;
  esac
fi

#
   ------------------------------------------------------------------
# PRACTICO 02: red y diagnostico
#
   ------------------------------------------------------------------
section "Herramientas de red"

case "$PM" in
  apt)
    pkg_install iproute2 iputils-ping openssh-client rsync wget
       curl tcpdump nmap dnsutils traceroute telnet
    ;;
  dnf)
    pkg_install iproute iputils openssh-clients rsync wget curl
       tcpdump nmap bind-utils traceroute telnet
    ;;
  pacman)
    pkg_install iproute2 iputils openssh rsync wget curl tcpdump
       nmap bind traceroute inetutils
```

```
    ;;
esac

#
    ------------------------------------------------------------------------
# PRACTICO 03: ofimatica, documentacion, LaTeX, HTML/PDF
#
    ------------------------------------------------------------------------
section "Ofimatica, LaTeX y documentacion"

if ! have libreoffice; then
  log "[INSTALA] LibreOffice (Writer, Calc e Impress)"
  case "$PM" in
    apt) pkg_install libreoffice ;;
    dnf) pkg_install libreoffice ;;
    pacman) pkg_install libreoffice-fresh ;;
  esac
else
  log "[OK] LibreOffice ya esta instalado."
fi

# TeX Live completo, como pide explicitamente eu03.
if ! have pdflatex; then
  log "[INSTALA] TeX Live completo"
  case "$PM" in
    apt) pkg_install texlive-full ;;
    dnf) pkg_install texlive-scheme-full ;;
    pacman) pkg_install texlive-meta ;;
  esac
else
  log "[OK] LaTeX/pdflatex ya esta instalado."
fi

# Editor adecuado para LaTeX.
install_if_missing texstudio texstudio texstudio texstudio

# Pandoc aparece como herramienta alternativa/complementaria en
    el material.
install_if_missing pandoc pandoc pandoc pandoc

#
    ------------------------------------------------------------------------
# PRACTICOS 07--08--10: graficos, PDF e interfaz web
#
    ------------------------------------------------------------------------
section "Graficos, PDF e interfaz web"

install_if_missing gnuplot gnuplot gnuplot gnuplot
install_if_missing display imagemagick ImageMagick imagemagick
install_if_missing evince evince evince evince
```

```
# Navegador web para HTML, consola JavaScript e interfaces web.
if have firefox || have firefox-esr || have chromium || have
   chromium-browser || have google-chrome; then
  log "[OK] hay un navegador web disponible."
else
  log "[INSTALA] Firefox"
  case "$PM" in
    apt) pkg_install firefox-esr || pkg_install firefox ;;
    dnf) pkg_install firefox ;;
    pacman) pkg_install firefox ;;
  esac
fi

# curl/wget ya se instalaron en red, pero se verifican porque
   eu08 los usa.
install_if_missing curl curl curl curl
install_if_missing wget wget wget wget

# servguiweb: eu08 lo instala desde un .deb externo. Solo
   corresponde a sistemas Debian.
# Se intenta al final y sin abortar el resto si el enlace ya no
   existe.
if [ "$PM" = apt ] && ! have servguiweb; then
  log "[OPCIONAL] Intentando instalar servguiweb desde la URL
     indicada en eu08..."
  tmpdeb="$(mktemp --suffix=.deb)"
  if wget -q -O "$tmpdeb"
     'https://ututo.org/userfiles/downloads/servguiweb_0009-1_all.deb';
     then
    sudo dpkg -i "$tmpdeb" || true
    sudo apt-get install -f -y || true
  else
    log "[AVISO] No se pudo descargar servguiweb; el enlace del
       practico puede haber quedado obsoleto."
  fi
  rm -f "$tmpdeb"
elif have servguiweb; then
  log "[OK] servguiweb ya esta instalado."
fi

#
   ------------------------------------------------------------------
# PRACTICOS 05/10: control de versiones cuando aparezca en la
   guia
#
   ------------------------------------------------------------------
section "Control de versiones"
install_if_missing git git git git

#
   ------------------------------------------------------------------
```

```
# Utilidad 'script' para registrar sesiones de terminal
#
    ------------------------------------------------------------------------
section "Registro de sesiones"
if have script; then
  log "[OK] script ya esta disponible."
else
  case "$PM" in
    apt) pkg_install bsdutils ;;
    dnf) pkg_install util-linux ;;
    pacman) pkg_install util-linux ;;
  esac
fi

#
    ------------------------------------------------------------------------
# Verificacion final
#
    ------------------------------------------------------------------------
section "VERIFICACION FINAL"

# Si cpanm instalo modulos en ~/perl5 (fallback sin permisos),
   que perl los vea.
[ -d "$HOME/perl5/lib/perl5" ] && export
   PERL5LIB="$HOME/perl5/lib/perl5${PERL5LIB:+:$PERL5LIB}"

commands=(
  scilab scilab-cli maxima octave python3 ipython3 jupyter perl
     pdl2
  node npm libreoffice pdflatex xelatex texstudio pandoc gnuplot
     display evince
  firefox git tree figlet cowsay nano vim gcc make ssh scp rsync
     wget curl
  tcpdump nmap dig traceroute ping script virt-manager
     gnome-boxes qemu-system-x86_64 servguiweb duckdb
)

for c in "${commands[@]}"; do
  if have "$c"; then
    printf '[OK] %-14s %s\n' "$c" "$(command -v "$c")"
  else
    printf '[--] %-14s no encontrado\n' "$c"
  fi
done

log ""
log "Python:"
if have python3; then
python3 - <<'PY' 2>/dev/null || true
mods = ['numpy', 'scipy', 'matplotlib', 'duckdb']
for m in mods:
```

```
    try:
        mod = __import__(m)
        print('[OK]', m, getattr(mod, '__version__', ''))
    except Exception as e:
        print('[--]', m, e)
PY
fi

log ""
log "Perl:"
if have perl; then
  for m in PDL PDL::GSL::INTEG PDL::GSL::MROOT PDL::GSL::DIFF
     Math::ODE Math::RungeKutta Math::Symbolic Math::GSL
     LaTeX::Table Text::CSV DBI DBD::DuckDB; do
    if perl -M"$m" -e1 >/dev/null 2>&1; then
      log "[OK] $m"
    else
      log "[--] $m no disponible"
    fi
  done
fi

log ""
log "Node/npm:"
if have node; then node --version || true; fi
if have node; then
  log "NODE_PATH=${NODE_PATH:-(vacio)}"
  # shellcheck disable=SC2086
  for p in $NPM_MODULOS; do
    if ( cd / && node -e "require('$p')" ) >/dev/null 2>&1; then
      log "[OK] $p: require(\"$p\") funciona desde cualquier
         directorio"
    else
      log "[--] $p: require(\"$p\") falla; abra una terminal
         nueva o ejecute: source /etc/profile.d/node-global.sh"
    fi
  done
fi

log ""
log "Notas:"
log "- Maquinas virtuales: QEMU/KVM con virt-manager o GNOME
   Boxes (software libre). VirtualBox solo si se corre con
   INSTALAR_VIRTUALBOX=1."
log "- Los grupos kvm y libvirt del usuario rigen a partir de la
   proxima sesion."
log "- Los modulos de node (mathjs, numeric, odex, ...) estan
   instalados globalmente; NODE_PATH queda fijado en
   /etc/profile.d/node-global.sh y rige a partir de la proxima
   terminal (o con: source /etc/profile.d/node-global.sh)."
log "- Overleaf es un servicio web, no requiere instalacion
```

```
    local."
log "- servguiweb se intenta solo en Debian/Ubuntu y puede
    fallar si el enlace historico ya no existe."
log ""
log "Listo. El script solo intento agregar software faltante; no
    elimina programas instalados."
```

```
input> chmod +x instalar_practicos_gnulinux.sh
input> ./instalar_practicos_gnulinux.sh

output> Sistema detectado: Debian GNU/Linux
output> [OK] Scilab ya esta instalado.
output> [OK] Maxima ya esta instalado.
```

Listing 4.2: Script de instalación del software utilizado en los prácticos

```
#!/usr/bin/env bash
set -u

# Instalador idempotente para los practicos eu01--eu10 en macOS.
# Es el equivalente del instalador GNU/Linux y conserva el
    conjunto de
# herramientas relevado en los practicos: Scilab, Maxima,
    Octave, Python,
# Perl, Node.js, LibreOffice, LaTeX, red, graficos, compilacion,
    etc.
#
# Criterios:
# - No desinstala software.
# - No reinstala deliberadamente lo que ya esta disponible.
# - Usa Homebrew para formulas y casks.
# - Usa un entorno virtual para los paquetes Python.
# - Usa cpanm en un directorio del usuario para los modulos Perl.
# - Usa npm global para los paquetes JavaScript del practico.
# - Las herramientas exclusivas de Debian/Ubuntu se informan
    como no aplicables.

log() { printf '%s\n' "$*"; }
have() { command -v "$1" >/dev/null 2>&1; }
section() {
  log ""
  log
     "============================================================"
  log "$1"
  log
     "============================================================"
}

if [ "$(uname -s)" != "Darwin" ]; then
  log "ERROR: este instalador es solamente para macOS."
  exit 1
```

```
fi

ARCH="$(uname -m)"
MACOS_VERSION="$(sw_vers -productVersion 2>/dev/null || true)"
log "macOS: ${MACOS_VERSION:-desconocido}"
log "Arquitectura: $ARCH"

#
    ------------------------------------------------------------------------
# Homebrew
#
    ------------------------------------------------------------------------
section "Homebrew"

if ! have brew; then
  log "[INSTALA] Homebrew"
  /bin/bash -c "$(curl -fsSL
     https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
     || {
    log "ERROR: no se pudo instalar Homebrew."
    exit 1
  }

  # Homebrew queda en rutas distintas en Apple Silicon e Intel.
  if [ -x /opt/homebrew/bin/brew ]; then
    eval "$(/opt/homebrew/bin/brew shellenv)"
  elif [ -x /usr/local/bin/brew ]; then
    eval "$(/usr/local/bin/brew shellenv)"
  fi
else
  log "[OK] Homebrew ya esta instalado."
fi

BREW_PREFIX="$(brew --prefix)"
export PATH="$BREW_PREFIX/bin:$BREW_PREFIX/sbin:$PATH"

brew_formula_installed() {
  brew list --formula "$1" >/dev/null 2>&1
}

brew_cask_installed() {
  brew list --cask "$1" >/dev/null 2>&1
}

brew_formula() {
  formula="$1"
  cmd="${2:-$1}"

  if have "$cmd" || brew_formula_installed "$formula"; then
    log "[OK] $formula"
    return 0
```

```
  fi

  log "[INSTALA] $formula"
  brew install "$formula" || log "[AVISO] no se pudo instalar
     $formula"
}

brew_cask() {
  cask="$1"
  app="${2:-}"

  if brew_cask_installed "$cask"; then
    log "[OK] $cask"
    return 0
  fi

  if [ -n "$app" ] && [ -d "/Applications/$app.app" ]; then
    log "[OK] $app ya esta en /Applications"
    return 0
  fi

  log "[INSTALA] $cask"
  brew install --cask "$cask" || log "[AVISO] no se pudo
     instalar $cask"
}

#
   ------------------------------------------------------------------------
# PRACTICOS 04--05: calculo cientifico y lenguajes
#
   ------------------------------------------------------------------------
section "Calculo cientifico y lenguajes"

# Scilab 2026 en Homebrew requiere Java 8.
brew_cask temurin@8 "Temurin-8"
brew_cask scilab "Scilab"

# Complemento simbolico mantenido de la version GNU/Linux.
brew_formula maxima maxima

# GNU Octave.
brew_formula octave octave

# Python. Homebrew moderno protege el entorno global de pip (PEP
   668),
# por eso los paquetes cientificos se instalan en un venv
   dedicado.
brew_formula python python3

PY_VENV="$HOME/.cc-practicos-python"
if have python3; then
```

```
  if [ ! -x "$PY_VENV/bin/python" ]; then
    log "[CREA] entorno virtual Python: $PY_VENV"
    python3 -m venv "$PY_VENV" || log "[AVISO] no se pudo crear
       el venv Python"
  else
    log "[OK] entorno virtual Python ya existe."
  fi

  if [ -x "$PY_VENV/bin/python" ]; then
    log "[INSTALA/VERIFICA] NumPy, SciPy, Matplotlib, IPython y
       Jupyter"
    "$PY_VENV/bin/python" -m pip install --upgrade pip
       >/dev/null 2>&1 || true
    "$PY_VENV/bin/python" -m pip install numpy scipy matplotlib
       ipython jupyter notebook duckdb || \
      log "[AVISO] fallo algun paquete Python"
  fi
fi

# Perl, PDL y modulos de los ejercicios.
brew_formula perl perl
brew_formula cpanminus cpanm
brew_formula gsl gsl-config

PERL_LOCAL="$HOME/perl5"
perl_module() {
  module="$1"
  if perl -M"$module" -e1 >/dev/null 2>&1; then
    log "[OK] Perl $module"
    return 0
  fi

  log "[INSTALA] Perl $module"
  if have cpanm; then
    cpanm --local-lib-contained "$PERL_LOCAL" --notest "$module"
       || \
      log "[AVISO] no se pudo instalar Perl $module"
  else
    log "[AVISO] cpanm no esta disponible para instalar $module"
  fi
}

if have perl; then
  # Si ya existe una instalacion local, permitir que las
     verificaciones la vean.
  if [ -d "$PERL_LOCAL/lib/perl5" ]; then
    export
       PERL5LIB="$PERL_LOCAL/lib/perl5${PERL5LIB:+:$PERL5LIB}"
    export PATH="$PERL_LOCAL/bin:$PATH"
  fi
```

```
    perl_module PDL
    perl_module Math::ODE
    perl_module Math::RungeKutta
    perl_module Math::Symbolic
    perl_module Math::GSL
    perl_module LaTeX::Table
    perl_module Text::CSV
    perl_module DBI
    perl_module DBD::DuckDB
fi

# JavaScript / Node.js.
brew_formula node node

npm_global() {
  package="$1"
  if npm list -g --depth=0 "$package" >/dev/null 2>&1; then
    log "[OK] npm $package"
  else
    log "[INSTALA] npm $package"
    npm install -g "$package" || log "[AVISO] no se pudo
       instalar npm $package"
  fi
}

if have npm; then
  npm_global mathjs
  npm_global numeric
  npm_global odex
  npm_global csv-parse
  npm_global @duckdb/node-api
  npm_global readline-sync

  # Los import ESM de node no usan el prefijo global; un
     node_modules en
  # $HOME permite la resolucion por directorios padre para
     proyectos bajo $HOME.
  NPM_GLOBAL_ROOT="$(npm root -g 2>/dev/null)"
  if [ -n "$NPM_GLOBAL_ROOT" ] && [ ! -e "$HOME/node_modules" ];
     then
    ln -s "$NPM_GLOBAL_ROOT" "$HOME/node_modules"
    log "[OK] ~/node_modules -> $NPM_GLOBAL_ROOT (para import
       ESM, p.ej. @duckdb/node-api)"
  fi
fi

# Paquetes de Octave mencionados. Se intenta instalarlos desde
   Octave Forge
# solamente cuando no estan ya disponibles.
octave_pkg() {
  pkg="$1"
```

```
  if ! have octave; then return 0; fi
  if octave --quiet --eval "p=pkg('list'); exit(any(cellfun(@(x)
     strcmp(x.name,'$pkg'), p)));" >/dev/null 2>&1; then
    # El codigo de salida de Octave no es fiable en todas las
       versiones para este uso.
    :
  fi
  if octave --quiet --eval "p=pkg('list'); names=cellfun(@(x)
     x.name,p,'UniformOutput',false); if
     any(strcmp(names,'$pkg')), disp('INSTALADO'); end"
     2>/dev/null | grep -q INSTALADO; then
    log "[OK] Octave $pkg"
  else
    log "[INSTALA/INTENTA] Octave $pkg"
    octave --quiet --eval "pkg install -forge $pkg" || log
       "[AVISO] no se pudo instalar Octave $pkg"
  fi
}

octave_pkg control
octave_pkg signal
octave_pkg statistics

#
   ------------------------------------------------------------------------
# PRACTICO 01: utilidades, compresion y compilacion
#
   ------------------------------------------------------------------------
section "Utilidades y compilacion"

brew_formula tree tree
brew_formula figlet figlet
brew_formula cowsay cowsay
brew_formula nano nano
brew_formula vim vim

# Herramientas GNU/comunes. Algunas ya vienen con macOS; se
   mantienen si existen.
brew_formula wget wget
brew_formula rsync rsync
brew_formula xz xz
brew_formula xxhash xxhsum
brew_formula coreutils gsha256sum

# zip, unzip, gzip, bzip2 y file suelen venir con macOS. Solo se
   informa su estado.
for c in zip unzip gzip bzip2 file; do
  if have "$c"; then
    log "[OK] $c (incluido en macOS o ya instalado)"
  else
    log "[AVISO] $c no encontrado"
```

```
  fi
done

# Herramientas de compilacion: Command Line Tools de Apple.
if xcode-select -p >/dev/null 2>&1; then
  log "[OK] Xcode Command Line Tools"
else
  log "[INSTALA] Xcode Command Line Tools"
  log "         macOS mostrara un dialogo grafico para completar
     la instalacion."
  xcode-select --install 2>/dev/null || true
fi

# Adicionalmente se instala GCC GNU si no esta registrado por
   Homebrew.
# En macOS /usr/bin/gcc normalmente es clang; ambos sirven para
   los ejercicios basicos.
brew_formula gcc gcc
brew_formula make gmake

# VirtualBox aparece como alternativa de virtualizacion.
brew_cask virtualbox "VirtualBox"

#
   ------------------------------------------------------------------
# PRACTICO 02: red y diagnostico
#
   ------------------------------------------------------------------
section "Herramientas de red"

# macOS ya incluye ssh, scp, curl, ping, traceroute, dig y
   tcpdump normalmente.
# Se conservan esas versiones si estan disponibles.
for c in ssh scp curl ping traceroute dig tcpdump; do
  if have "$c"; then
    log "[OK] $c (incluido en macOS o ya instalado)"
  else
    log "[AVISO] $c no encontrado"
  fi
done

brew_formula nmap nmap
brew_formula telnet telnet
brew_formula wget wget
brew_formula rsync rsync

#
   ------------------------------------------------------------------
# PRACTICO 03: ofimatica, documentacion, LaTeX, HTML/PDF
#
   ------------------------------------------------------------------
```

```
section "Ofimatica, LaTeX y documentacion"

brew_cask libreoffice "LibreOffice"

# MacTeX es la distribucion TeX Live completa para macOS.
if have pdflatex || brew_cask_installed mactex; then
  log "[OK] MacTeX/LaTeX"
else
  brew_cask mactex "TeXShop"
  # Homebrew indica que puede requerir recargar el PATH tras
     instalar MacTeX.
  eval "$(/usr/libexec/path_helper)" 2>/dev/null || true
fi

# TeXstudio aparece como editor en el instalador GNU/Linux.
   Homebrew lo marca
# para deprecacion en 2026, pero mientras siga disponible se
   intenta instalar.
brew_cask texstudio "texstudio"

brew_formula pandoc pandoc

#
   ------------------------------------------------------------------------
# PRACTICOS 07--08--10: graficos, PDF e interfaz web
#
   ------------------------------------------------------------------------
section "Graficos, PDF e interfaz web"

brew_formula gnuplot gnuplot

# DuckDB: ejecutable SQL del practico 06 (la formula incluye
   libduckdb,
# que usa DBD::DuckDB en Perl via FFI).
brew_formula duckdb duckdb

# JDBC jar de DuckDB: lo usa la parte a) del ejercicio Consultas
   (Scilab).
if [ ! -f /usr/local/share/java/duckdb_jdbc.jar ]; then
  log "[INSTALA] duckdb_jdbc.jar (para Scilab/JDBC)"
  sudo mkdir -p /usr/local/share/java
  sudo curl -fsSL
     https://repo1.maven.org/maven2/org/duckdb/duckdb_jdbc/1.5.5.1/duckdb_jd
     \
    -o /usr/local/share/java/duckdb_jdbc.jar || \
    log "[AVISO] no se pudo descargar duckdb_jdbc.jar"
fi
brew_formula imagemagick magick
brew_formula evince evince

brew_cask firefox "Firefox"
```

```
# wget/curl se verifican nuevamente porque algunos practicos los
    usan.
brew_formula wget wget
if have curl; then
  log "[OK] curl"
else
  brew_formula curl curl
fi

# servguiweb se distribuye en el material como paquete .deb. Ese
    paquete no es
# instalable directamente en macOS. Se conserva la advertencia
    para no ocultar
# la diferencia entre plataformas.
if have servguiweb; then
  log "[OK] servguiweb"
else
  log "[NO APLICA] servguiweb del practico esta empaquetado como
     .deb para Debian/Ubuntu."
fi

#
    ------------------------------------------------------------------------
# Control de versiones
#
    ------------------------------------------------------------------------
section "Control de versiones"

# Apple puede incluir git con Command Line Tools, pero se
    instala Homebrew Git
# solo si no hay ninguno disponible.
brew_formula git git

#
    ------------------------------------------------------------------------
# Registro de sesiones
#
    ------------------------------------------------------------------------
section "Registro de sesiones"

if have script; then
  log "[OK] script (incluido en macOS)"
else
  log "[AVISO] no se encontro la utilidad script"
fi

#
    ------------------------------------------------------------------------
# Archivo de entorno para Python y Perl
#
```

```
    ------------------------------------------------------------------------
section "Entorno de lenguajes"

ENV_FILE="$HOME/.cc-practicos-env.sh"
cat > "$ENV_FILE" <<ENVEOF
# Generado por instalar_practicos_macos.sh
export CC_PRACTICOS_PYTHON="$PY_VENV"
if [ -d "\$CC_PRACTICOS_PYTHON/bin" ]; then
  export PATH="\$CC_PRACTICOS_PYTHON/bin:\$PATH"
fi
export PERL_LOCAL="$PERL_LOCAL"
if [ -d "\$PERL_LOCAL/lib/perl5" ]; then
  export
     PERL5LIB="\$PERL_LOCAL/lib/perl5\${PERL5LIB:+:\$PERL5LIB}"
  export PATH="\$PERL_LOCAL/bin:\$PATH"
fi
ENVEOF

log "[OK] archivo de entorno: $ENV_FILE"
log "Para activar Python/Perl del practico en una terminal
   nueva:"
log "    source \"$ENV_FILE\""

#
    ------------------------------------------------------------------------
# Verificacion final
#
    ------------------------------------------------------------------------
section "VERIFICACION FINAL"

# Cargar el entorno local para verificar modulos.
# shellcheck disable=SC1090
source "$ENV_FILE" 2>/dev/null || true

commands=(
  scilab maxima octave python3 ipython jupyter perl node npm
     libreoffice
  pdflatex xelatex pandoc gnuplot magick evince firefox git tree
     figlet cowsay
  nano vim gcc make ssh scp rsync wget curl tcpdump nmap dig
     traceroute ping
  telnet script VBoxManage
)

for c in "${commands[@]}"; do
  if have "$c"; then
    printf '[OK] %-14s %s\n' "$c" "$(command -v "$c")"
  else
    printf '[--] %-14s no encontrado\n' "$c"
  fi
done
```

```
log ""
log "Python:"
if [ -x "$PY_VENV/bin/python" ]; then
  "$PY_VENV/bin/python" - <<'PY' 2>/dev/null || true
mods = ['numpy', 'scipy', 'matplotlib', 'IPython', 'jupyter']
for m in mods:
    try:
        mod = __import__(m)
        print('[OK]', m, getattr(mod, '__version__', ''))
    except Exception as e:
        print('[--]', m, e)
PY
fi

log ""
log "Perl:"
if have perl; then
  for m in PDL Math::ODE Math::RungeKutta Math::Symbolic
     Math::GSL LaTeX::Table; do
    if perl -M"$m" -e1 >/dev/null 2>&1; then
      log "[OK] $m"
    else
      log "[--] $m no disponible"
    fi
  done
fi

log ""
log "Node/npm:"
if have node; then node --version || true; fi
if have npm; then
  for p in mathjs numeric odex; do
    if npm list -g --depth=0 "$p" >/dev/null 2>&1; then
      log "[OK] $p"
    else
      log "[--] $p no disponible globalmente"
    fi
  done
fi

log ""
log "Notas de compatibilidad macOS:"
log "- Los paquetes Debian devscripts, debhelper, fakeroot y el
   .deb de servguiweb no tienen equivalencia directa necesaria
   en macOS."
log "- WSL2 es exclusivo de Windows."
log "- Overleaf es un servicio web y no necesita instalacion
   local."
log "- VMware requiere su propio instalador/licencia; VirtualBox
   se intenta mediante Homebrew."
```

```
log "- Scilab instalado por Homebrew requiere Java 8; el script
   instala temurin@8."
log "- MacTeX puede requerir abrir una terminal nueva para que
   todos sus comandos queden en PATH."
log ""
log "Listo. El script intento agregar solamente el software
   faltante."
```

```
chmod +x instalar_practicos_macos.sh

./instalar_practicos_macos.sh
```

[style=terminal]

genera

```
~/.cc-practicos-env.sh

source ~/.cc-practicos-env.sh
```

Listing 4.3: Script de instalación del software utilizado en los prácticos

```
#requires -Version 5.1
<#
  Instalador de software para los practicos - Windows
  Ejecutar preferentemente desde PowerShell.

  Objetivos:
  - Instalar solo lo que falta.
  - Usar winget cuando sea posible.
  - Mantener Scilab y Maxima.
  - Instalar herramientas equivalentes a las usadas en los
     practicos.
  - Informar lo que es propio de GNU/Linux y no corresponde
     instalar en Windows.

  Nota:
  Algunas instalaciones pueden solicitar permisos de
     administrador o abrir
  instaladores graficos.
#>

$ErrorActionPreference = "Continue"

Write-Host
   "============================================================"
Write-Host " Software para los practicos - Windows"
Write-Host
   "============================================================"
Write-Host ""

function Test-Command {
```

```
    param([Parameter(Mandatory=$true)][string]$Name)
    return [bool](Get-Command $Name -ErrorAction
       SilentlyContinue)
}

function Install-WingetPackage {
    param(
        [Parameter(Mandatory=$true)][string]$DisplayName,
        [Parameter(Mandatory=$true)][string]$Id,
        [string]$Command = ""
    )

    if ($Command -and (Test-Command $Command)) {
        Write-Host "[OK] $DisplayName"
        return
    }

    $installed = winget list --id $Id --exact
       --accept-source-agreements 2>$null |
                 Select-String -SimpleMatch $Id

    if ($installed) {
        Write-Host "[OK] $DisplayName ya figura instalado."
        return
    }

    Write-Host "[FALTA] $DisplayName"
    winget install --id $Id --exact `
        --accept-package-agreements `
        --accept-source-agreements
}

function Test-PythonModule {
    param([string]$Module)
    if (-not (Test-Command "python")) { return $false }
    python -c "import $Module" 2>$null
    return ($LASTEXITCODE -eq 0)
}

function Install-PythonModule {
    param([string]$Module, [string]$Package = "")
    if (-not $Package) { $Package = $Module }

    if (Test-PythonModule $Module) {
        Write-Host "[OK] Python: $Module"
    } else {
        Write-Host "[FALTA] Python: $Module"
        python -m pip install --user $Package
    }
}
```

```
function Test-NpmGlobal {
    param([string]$Package)
    npm list -g --depth=0 $Package 2>$null | Out-Null
    return ($LASTEXITCODE -eq 0)
}

function Install-NpmGlobal {
    param([string]$Package)
    if (-not (Test-Command "npm")) {
        Write-Host "[AVISO] npm no esta disponible todavia:
           $Package"
        return
    }

    if (Test-NpmGlobal $Package) {
        Write-Host "[OK] npm: $Package"
    } else {
        Write-Host "[FALTA] npm: $Package"
        npm install -g $Package
    }
}

# ------------------------------------------------------------
# winget
# ------------------------------------------------------------

if (-not (Test-Command "winget")) {
    Write-Host "[ERROR] No se encontro winget."
    Write-Host "Instale/actualice 'App Installer' desde
       Microsoft Store"
    Write-Host "y vuelva a ejecutar este script."
    exit 1
}

Write-Host "[OK] winget"
Write-Host ""

# ------------------------------------------------------------
# Software cientifico
# ------------------------------------------------------------

Install-WingetPackage "Scilab"       "Scilab.Scilab"
Install-WingetPackage "Maxima"       "Maxima.Maxima"
Install-WingetPackage "GNU Octave"   "GNU.Octave"        "octave"

# DuckDB: ejecutable SQL del practico 06
Install-WingetPackage "DuckDB CLI"   "DuckDB.cli"        "duckdb"

# ------------------------------------------------------------
# Python y entorno cientifico
# ------------------------------------------------------------
```

```
Install-WingetPackage "Python 3"     "Python.Python.3.13"
   "python"

# Refrescar PATH de la sesion cuando sea posible
$machinePath = [Environment]::GetEnvironmentVariable("Path",
   "Machine")
$userPath    = [Environment]::GetEnvironmentVariable("Path",
   "User")
$env:Path    = "$machinePath;$userPath"

if (Test-Command "python") {
    python -m ensurepip --upgrade 2>$null
    python -m pip install --upgrade pip

    Install-PythonModule "numpy"
    Install-PythonModule "scipy"
    Install-PythonModule "matplotlib"
    Install-PythonModule "IPython" "ipython"
    Install-PythonModule "jupyter"
    Install-PythonModule "duckdb"
} else {
    Write-Host "[AVISO] Python fue instalado pero esta sesion
       aun no lo ve."
    Write-Host "        Cierre PowerShell, vuelva a abrirlo y
       ejecute otra vez."
}

# ------------------------------------------------------------
# Perl
# ------------------------------------------------------------

Install-WingetPackage "Strawberry Perl"
   "StrawberryPerl.StrawberryPerl" "perl"

$machinePath = [Environment]::GetEnvironmentVariable("Path",
   "Machine")
$userPath    = [Environment]::GetEnvironmentVariable("Path",
   "User")
$env:Path    = "$machinePath;$userPath"

if (Test-Command "cpan") {
    $perlModules = @(
        "PDL",
        "Math::ODE",
        "Math::GSL",
        "Math::RungeKutta",
        "Math::Symbolic",
        "LaTeX::Table",
        "Text::CSV",
        "DBI",
```

```
        "DBD::DuckDB"
    )

    foreach ($m in $perlModules) {
        perl -M$m -e 1 2>$null
        if ($LASTEXITCODE -eq 0) {
            Write-Host "[OK] Perl: $m"
        } else {
            Write-Host "[FALTA] Perl: $m"
            cpan -T $m
        }
    }
} else {
    Write-Host "[AVISO] CPAN no esta disponible todavia."
}

# ------------------------------------------------------------
# JavaScript
# ------------------------------------------------------------

Install-WingetPackage "Node.js LTS" "OpenJS.NodeJS.LTS" "node"

$machinePath = [Environment]::GetEnvironmentVariable("Path",
   "Machine")
$userPath    = [Environment]::GetEnvironmentVariable("Path",
   "User")
$env:Path    = "$machinePath;$userPath"

Install-NpmGlobal "mathjs"
Install-NpmGlobal "numeric"
Install-NpmGlobal "odex"
Install-NpmGlobal "csv-parse"
Install-NpmGlobal "@duckdb/node-api"
Install-NpmGlobal "readline-sync"

# ------------------------------------------------------------
# Ofimatica, documentacion y graficos
# ------------------------------------------------------------

Install-WingetPackage "LibreOffice"
   "TheDocumentFoundation.LibreOffice"
Install-WingetPackage "MiKTeX"       "MiKTeX.MiKTeX"
Install-WingetPackage "TeXstudio"    "TeXstudio.TeXstudio"
Install-WingetPackage "Pandoc"       "JohnMacFarlane.Pandoc"
   "pandoc"
Install-WingetPackage "Gnuplot"      "gnuplot.gnuplot"
   "gnuplot"
Install-WingetPackage "ImageMagick"  "ImageMagick.ImageMagick"
   "magick"
Install-WingetPackage "Firefox"      "Mozilla.Firefox"
```

```
# ------------------------------------------------------------
# Desarrollo y edicion
# ------------------------------------------------------------

Install-WingetPackage "Git"           "Git.Git" "git"
Install-WingetPackage "Vim"           "vim.vim" "vim"
Install-WingetPackage "Nano"          "GNU.Nano" "nano"

# ------------------------------------------------------------
# Red y transferencia
# ------------------------------------------------------------

# Windows modernos incluyen OpenSSH opcionalmente. Si falta, se
   intenta
# habilitar la caracteristica del sistema.
if (Test-Command "ssh") {
    Write-Host "[OK] OpenSSH / ssh"
} else {
    Write-Host "[FALTA] OpenSSH Client"
    try {
        Add-WindowsCapability -Online -Name
           OpenSSH.Client~~~~0.0.1.0
    } catch {
        Write-Host "[AVISO] No se pudo instalar OpenSSH
           automaticamente."
    }
}

Install-WingetPackage "rsync (cwRsync)" "itefix.cwRsync"
Install-WingetPackage "wget"            "JernejSimoncic.Wget"
   "wget"
Install-WingetPackage "Nmap"            "Insecure.Nmap"
   "nmap"
Install-WingetPackage "Wireshark"
   "WiresharkFoundation.Wireshark"

# curl, ping, tracert, nslookup y tar suelen venir con Windows.
foreach ($cmd in @("curl", "ping", "tracert", "nslookup",
   "tar")) {
    if (Test-Command $cmd) {
        Write-Host "[OK] $cmd (incluido en Windows)"
    } else {
        Write-Host "[AVISO] No se encontro $cmd."
    }
}

# Telnet es una caracteristica opcional de Windows.
if (Test-Command "telnet") {
    Write-Host "[OK] telnet"
} else {
    Write-Host "[FALTA] Cliente Telnet"
```

```
    try {
        Enable-WindowsOptionalFeature -Online `
            -FeatureName TelnetClient -All -NoRestart | Out-Null
    } catch {
        Write-Host "[AVISO] No se pudo habilitar Telnet
           automaticamente."
    }
}

# ------------------------------------------------------------
# Utilidades
# ------------------------------------------------------------

Install-WingetPackage "7-Zip"       "7zip.7zip"
Install-WingetPackage "TreeSize Free" "JAMSoftware.TreeSize.Free"
Install-WingetPackage "Figlet"      "cmatsuoka.figlet" "figlet"

# ------------------------------------------------------------
# Compilacion
# ------------------------------------------------------------

Install-WingetPackage "MSYS2"       "MSYS2.MSYS2"
Install-WingetPackage "CMake"       "Kitware.CMake" "cmake"

# ------------------------------------------------------------
# Virtualizacion
# ------------------------------------------------------------

Install-WingetPackage "VirtualBox"  "Oracle.VirtualBox"

# ------------------------------------------------------------
# Componentes que no tienen equivalencia directa
# ------------------------------------------------------------

Write-Host ""
Write-Host
   "============================================================"
Write-Host " Elementos especificos de GNU/Linux"
Write-Host
   "============================================================"
Write-Host ""
Write-Host "- apt, dpkg, debuild, dpkg-buildpackage y fakeroot:"
Write-Host "  son herramientas propias de Debian/Ubuntu."
Write-Host "- servguiweb distribuido como paquete .deb:"
Write-Host "  no se instala directamente en Windows."
Write-Host "- tcpdump:"
Write-Host "  se usa Wireshark/Npcap como alternativa practica
   en Windows."
Write-Host "- dig:"
Write-Host "  Windows incluye nslookup; si un ejercicio exige
   dig literalmente,"
```

```
Write-Host "  conviene realizarlo en WSL."
Write-Host "- script:"
Write-Host "  no tiene el mismo comportamiento nativo de
   GNU/Linux."
Write-Host ""

# ------------------------------------------------------------
# Verificacion final
# ------------------------------------------------------------

Write-Host
   "============================================================"
Write-Host " Verificacion final"
Write-Host
   "============================================================"

$commands = @(
    "python",
    "perl",
    "node",
    "npm",
    "git",
    "ssh",
    "scp",
    "curl",
    "ping",
    "nmap",
    "pandoc",
    "gnuplot",
    "magick",
    "vim",
    "nano",
    "cmake",
    "duckdb"
)

foreach ($cmd in $commands) {
    if (Test-Command $cmd) {
        Write-Host ("[OK]    {0}" -f $cmd)
    } else {
        Write-Host ("[REVISAR] {0}" -f $cmd)
    }
}

Write-Host ""
Write-Host "Instalacion/verificacion terminada."
Write-Host ""
Write-Host "IMPORTANTE:"
Write-Host "Si se instalaron programas nuevos, cierre y vuelva a
   abrir PowerShell"
Write-Host "antes de realizar una segunda verificacion."
```

```
Write-Host ""
Write-Host "Para los ejercicios especificamente GNU/Linux, se
   recomienda WSL2."
```

```
Set-ExecutionPolicy -Scope Process Bypass
.\instalar_practicos_windows.ps1
```

4.21. Instalación de software para el practico 4 descripto

Hablamos de dos instancias en estas instrucciones. La del sistema operativo, sea ejecutando un comando en la terminal o dando clic a un menú en el entorno gráfico, que sirve para arrancar el programa de la calculadora;avanzada y la de la misma calculadora avanzada, con su propia consola e indicador, que es lo que debemos aprender a usar.

Estudie con detalle lo siguiente según el sistema operativo que usa y la calculadora que quiera probar.

- **GNU/Linux.** Usaremos una terminal (Konsole, GNOME Terminal, xterm, etc.) que internamente corre Bash /hay otros pero usaremos bash). Es el entorno nativo del curso: Bash y Perl ya vienen instalados, y las calculadoras se instalan con el gestor de paquetes de la distribución. En Debian, Ubuntu y derivadas:

```
sudo apt install scilab                                    #
   Scilab
sudo apt install octave                                    #
   Octave
sudo apt install pdl                                       #
   Perl: consola pdl2 y PDL (perl ya esta)
sudo apt install python3 ipython3 python3-numpy
   python3-scipy python3-matplotlib jupyter-notebook
sudo apt install nodejs npm && sudo npm install -g mathjs
   numeric odex
export NODE_PATH="$(npm root -g)"
 # JavaScript; el navegador ya esta
```

  En Fedora, `dnf`; en Arch, `pacman`, con nombres parecidos.

  Lo más crpitico es saber el nombre los paquetes a instalar en cada distribución ademas de saber usar el instalador nativo de cada aplicación. Además pueden cambiar con el tiempo.

  Solicite ayuda a sus profesores, grupos de ayuda de la distribución, o a una IA, para éstos menesteres.

- **Windows con PowerShell.** PowerShell y cmd son las terminales nativas; ninguna calculadora viene instalada. Cada una se instala con su instalador gráfico, o con `winget` desde PowerShell:

```
winget install Scilab.Scilab                    # o instalador de scilab.org
winget install GNU.Octave                       # o instalador de octave.org
winget install StrawberryPerl.StrawberryPerl    # Perl; luego cpan PDL para pdl2
winget install Python.Python.3.12               # o instalador de python.org
pip install numpy scipy matplotlib ipython jupyter
winget install OpenJS.NodeJS                    # o instalador de nodejs.org
npm install -g mathjs numeric odex
  $env:NODE_PATH = (npm root -g)
```

Alternativa que junta Python y sus bibliotecas en un solo instalador: Anaconda o Miniconda. Los iconos que quedan en el menú Inicio (Scilab, Python 3.x, IDLE, Anaconda Navigator) arrancan las mismas calculadoras que los comandos.

- **Windows con Bash.** Hay dos formas. WSL (Subsistema de Windows para Linux) instala un GNU/Linux completo dentro de Windows: `wsl -install` desde PowerShell, reiniciar, y desde ahí es el caso GNU/Linux de arriba, con `apt`; la ventana gráfica de Scilab requiere Windows 11 (WSLg), en Windows 10 se usa `scilab-cli`. Git Bash o MSYS2 dan sólo la terminal Bash: las calculadoras son las de Windows, instaladas como en el punto anterior, y se llaman desde Bash por su nombre.

- **Windows con máquina virtual GNU/Linux.** Se instala VirtualBox (o VMware Player) y dentro una imagen de Debian o Ubuntu; a partir de ahí es el caso GNU/Linux. Lo ideal es correr una máquina virtual que ya tenga preinstaladas las calculadoras que se quieran usar: la cátedra, o la IA, puede indicar una imagen con Scilab, Python, PDL y Node listos, que se importa en VirtualBox y se arranca. Cambia con el tiempo.

- **Mac.** La aplicación Terminal corre zsh, hermano de Bash; Perl y Python vienen, pero conviene instalar lo demás con Homebrew:

```
brew install --cask scilab          # o el .dmg de scilab.org
brew install octave                 # Octave
cpan PDL                            # Perl ya viene
brew install python ipython jupyterlab && pip3 install numpy
   scipy matplotlib
brew install node && npm install -g mathjs numeric odex
export NODE_PATH="$(npm root -g)"
```

Cuadro 4.1: Instalación de las calculadoras según sistema operativo y lenguaje. En todos los casos, después de instalar: Scilab se arranca con `scilab` o `scilab-cli`; Octave con `octave` o `octave-cli`; Python con `ipython3` (o `python3`, `jupyter notebook`); Perl con `perl -de1` o `pdl2`; JavaScript con `node` o con la consola del navegador (F12). Perl usa además `libmath-ode-perl` (o `cpan Math::ODE`) para las EDO. Para JavaScript, `mathjs`, `numeric` y `odex` se instalan globalmente con `npm install -g mathjs numeric odex`; en el navegador math.js no se instala: se carga con `<script src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/13.0.0/math.js"></script>`.

Sistema	Scilab	Octave	Python	Perl	JavaScript
GNU/Linux Debian, Ubuntu, Mint (`apt`)	`sudo apt install scilab`	`sudo apt install octave` (paquetes extra: `octave-control`, `octave-signal`, `octave-statistics`)	`sudo apt install python3 ipython3 python3-numpy python3-scipy python3-matplotlib jupyter-notebook`	Perl ya viene. `sudo apt install pdl libmath-ode-perl`	`sudo apt install nodejs npm` `sudo npm install -g mathjs numeric odex` `export NODE_PATH=$(npm root -g)` Navegador: ya viene.
GNU/Linux Fedora, RHEL (`dnf`)	`sudo dnf install scilab`	`sudo dnf install octave`	`sudo dnf install python3 python3-ipython python3-numpy python3-scipy python3-matplotlib python3-notebook`	Perl ya viene. `sudo dnf install perl-PDL` `cpan Math::ODE`	`sudo dnf install nodejs npm` `sudo npm install -g mathjs numeric odex` `export NODE_PATH=$(npm root -g)`
GNU/Linux Arch, Manjaro (`pacman`)	En AUR, no en los repositorios oficiales: `yay -S scilab`	`sudo pacman -S octave`	`sudo pacman -S python ipython python-numpy python-scipy python-matplotlib jupyter-notebook`	Perl ya viene. En AUR: `yay -S perl-pdl` o `cpan PDL`; `cpan Math::ODE`	`sudo pacman -S nodejs npm` `sudo npm install -g mathjs numeric odex` `export NODE_PATH=$(npm root -g)`

Sistema	Scilab	Octave	Python	Perl	JavaScript

Windows con PowerShell o cmd (nativo)	`winget install Scilab.Scilab` o instalador de scilab.org	`winget install GNU.Octave` o instalador de octave.org	`winget install Python.Python.3.12`
			`pip install numpy scipy matplotlib ipython jupyter`
			Alternativa todo en uno: `winget install Anaconda.Miniconda3` y luego `conda install numpy scipy matplotlib ipython jupyter`
				`winget install StrawberryPerl.StrawberryPerl`
				`cpan PDL; cpan Math::ODE`
					`winget install OpenJS.NodeJS` o instalador de nodejs.org
					`npm install -g mathjs numeric odex`
					`$env:NODE_PATH=(npm root -g)`
					Navegador: Edge ya viene.

Windows con Bash: WSL (GNU/Linux dentro de Windows)	En PowerShell, una sola vez: `wsl -install`, reiniciar; después, dentro de WSL, es la fila Debian/Ubuntu.
	`sudo apt install scilab`
	La ventana gráfica requiere Windows 11 (WSLg); en Windows 10 usar `scilab-cli`.
		`sudo apt install octave`
		Idem: la ventana gráfica sólo con WSLg; si no, `octave-cli`.
			`sudo apt install python3 ipython3 python3-numpy python3-scipy python3-matplotlib jupyter-notebook`
			Jupyter abre el navegador de Windows.
				Perl ya viene.
				`sudo apt install pdl libmath-ode-perl`
					`sudo apt install nodejs npm`
					`sudo npm install -g mathjs numeric odex`
					`export NODE_PATH=$(npm root -g)`

Windows con Bash: Git Bash o MSYS2 (sólo la terminal)	Se instala la terminal con `winget install Git.Git` (Git Bash) o `winget install MSYS2.MSYS2`. Las calculadoras son las de Windows: se instalan como en la fila "Windows con PowerShell" y se llaman desde Bash por su nombre (`scilab`, `octave`, `ipython`, `perl -de1`, `node`). En MSYS2 además `pacman -S mingw-w64-x86_64-python-numpy` etc. da un Python propio y `pacman -S mingw-w64-x86_64-octave` un Octave propio, pero no Scilab.				
Windows con máquina virtual GNU/Linux	`winget install Oracle.VirtualBox` (o VMware Workstation Player). Dentro, una de dos: instalar Debian o Ubuntu desde su ISO y seguir la fila Debian/Ubuntu; o importar la imagen `.ova` de la cátedra, que ya trae Scilab, Octave, Python con IPython y NumPy/SciPy, PDL con Math::ODE, y Node con math.js, numeric.js y odex, y no requiere instalar nada más. Es la opción recomendada en Windows si no se quiere pelear con WSL.				

Sistema	Scilab	Octave	Python	Perl	JavaScript
macOS (Terminal con zsh; Homebrew)	`brew install -cask scilab` o el `.dmg` de scilab.org	`brew install octave`	`brew install python ipython jupyterlab pip3 install numpy scipy matplotlib`	Perl ya viene. `cpan PDL`; `cpan Math::ODE` (o `brew install perl` y luego `cpan`)	`brew install node npm install -g mathjs numeric odex export NODE_PATH=$(npm root -g)` Navegador: Safari ya viene.

4.22. Secciones expandidas, no leer en la primera lectura

Aquí algunas secciones que tratan mas extensamente algunos temas.

4.22.1. Sistemas y predicción

Un **sistema** es un conjunto de partes o elementos relacionados entre sí y con su entorno. Su comportamiento depende no sólo de las partes consideradas aisladamente, sino también de las relaciones, interacciones y condiciones de contorno que las organizan.

En algunas tradiciones filosóficas y científicas se privilegió el análisis de **estructuras**: relaciones relativamente estables entre elementos. La teoría de sistemas conserva esa preocupación por la organización, pero agrega con más fuerza la interacción, el entorno, los flujos y, cuando corresponde, la dinámica temporal.

Esta diferencia entre estructura y sistema expresa una oposición filosófica clásica: algunas definiciones de metafísica tienden a pensar lo real como conjunto de entidades con propiedades relativamente fijas e inmutables, esencias; la dialéctica, también según algunas definiciones, en cambio, piensa lo real como relaciones, procesos, tensiones y transformaciones en el tiempo. Por eso la teoría de sistemas se acerca más a una mirada dinámica: no sólo pregunta qué partes hay, sino cómo se relacionan, qué intercambian y cómo cambian.

Sistema material

Salvo aclaración contraria, usamos la palabra **sistema** para referirnos, a un **sistema material o físico**.

Llamamos **sistema material** a la parte del universo físico que tomamos como objeto de estudio y sobre la cual podemos efectuar observaciones, mediciones, intervenciones o contrastes. Esta definición no pretende resolver disputas ontológicas sobre la naturaleza última de la materia; sólo fija el uso científico del término dentro de este texto.

Sistema material = parte del universo físico estudiada.

El término "material" se usa aquí en sentido físico y metodológico: designa aquello que se estudia mediante magnitudes medibles. No implica adoptar una doctrina filosófica materialista completa.

Personalmente me considero materialista, pero no veo la necesidad de involucrar una posición filosófica en este trabajo.

Intento evitar hablar de realidad, dado que es una palabra muy cargada filosóficamente, pero cuando lo hacemos, en este contexto, hablamos de sistemas materiales sobre los cuales pueden efectuarse mediciones, intervenciones y contrastes.

Se hace difícil dejar de hablar de "material" si bien es una palabra que puede parecer cargada. En definitiva la ciencia estudia el universo físico, más allá de que alguien pueda pensar que la materia no es "la realidad última", es al menos lo que medimos.

Con este sentido queda claro a qué nos referimos. No afirmamos que la materia sea la realidad última, sino lo que medimos. Es metodología, epistemología, no ontología.

- Un **sistema material** es una porción del universo en el que vivimos, seleccionada para su estudio.
- No definimos aquí "sistema" de manera completamente formal: lo tomamos como un preconcepto general desarrollado por la teoría de sistemas.
- Todo sistema material tiene una **frontera**, que puede ser real o ideal.
- En el exterior de la frontera se encuentra el **ambiente** o **entorno**.
- Sistema, frontera y ambiente forman, tomados en conjunto, el universo considerado.

Podemos pensar el universo como un continuo que, en cada instante, puede dividirse en sistemas materiales. Toda porción del universo pertenece a algún sistema en cada tiempo, dejando de lado efectos relativistas o cuánticos.

- Un sistema puede dividirse en **subsistemas** o **partes**, que también pueden estudiarse como sistemas.
- La **extensión geométrica y física** del sistema es la región ocupada por el sistema y las propiedades físicas asociadas a ella.
- La **frontera** puede tener también extensión geométrica y física, especialmente cuando se estudian intercambios de materia, energía o cantidad de movimiento. Aunque muchas veces se cuantifique como superficie, la frontera puede tener espesor, estructura y propiedades físicas propias.

Son sistemas materiales, partes del universo físico. Por ejemplo: el sistema solar, un circuito eléctrico, un reactor químico, una población biológica, una economía pensada en términos de bienes materiales, trabajo, etc.

Cambio, diferencia y estado

- *El* **cambio,** *o la* **diferencia** *y el* **estado** *se definen, muchas veces, experimentalmente para cada sistema material. Deben poder observarse, detectarse y medirse.*

Figura 4.3: Rosas del desierto. Frontera fractal.

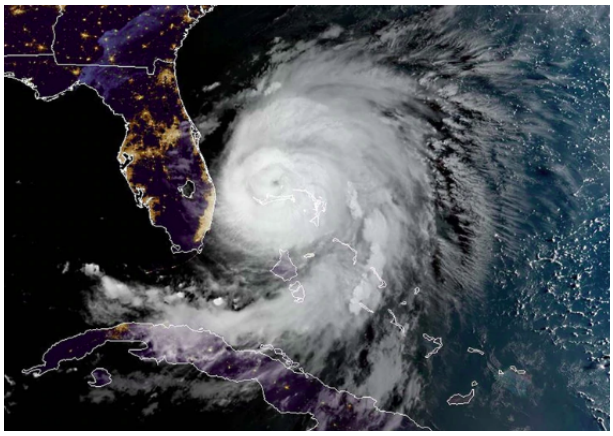


Figura 4.4: Sistema atmosférico cambiando.

El **estado** de un sistema es la información necesaria para describirlo en un momento dado, según el modelo elegido. Puede incluir posiciones, temperaturas, presiones, composiciones, velocidades u otras variables relevantes.

- Si se representa el tiempo a saltos, como es común, podremos ver que si un sistema se modifica, habrá diferencias entre los estados de dos momentos diferentes.
- O si al tiempo se lo representa como continuo en cada momento el sistema estará en un estado con movimiento. El cambio se describe mediante tasas de variación, derivadas o leyes de evolución. Estará cambiando en cada instante. Una foto sin embargo no captará dicho cambio. Se requieren al menos dos fotos separadas por un salto diferencial, pero que permita ver el cambio.
- En todo caso debemos poder definir experimentalmente el estado o situación de un sistema, lo que puede incluir movimientos y velocidades.
- Si se puede pensar el espacio de estados como geométrico, se pueden pensar los cambios como segmentos, donde los extremos del segmento son el estado inicial y el final del cambio. No necesariamente todos los puntos intermedios de un segmento son estados, por ejemplo, el espacio podría tener "huecos".

Lenguaje

Partimos de un lenguaje. Si queremos trabajar científicamente, conviene usar un lenguaje lo más preciso posible; en muchos casos, un lenguaje formal o matemático, que pueda describir sistemas, sus estados y sus cambios.

Hay muchas ideas atrás del desarrollo y uso de lenguajes, especialmente de lenguajes formales y matemáticos, para predecir sistemas. Presentaremos algunas básicas.

En este contexto un lenguaje es el mecanismo con el cual describimos el estado de un sistema y predecimos su futuro. Así, calculamos, predecimos, y podemos comunicarnos con otras personas para actuar cooperativamente. No nos limitamos a usar la intuición y otros mecanismos que nuestros cerebros desarrollaron.

En concreto, un **lenguaje** evolucionó para ser lo que describimos en el párrafo anterior hacia un sistema de signos y reglas que permite describir estados, relaciones y cambios de un sistema. Cuando ese lenguaje es formal o matemático, permite además construir modelos, deducir consecuencias y calcular predicciones.

Con esos símbolos y reglas de formación se desarrollan expresiones (fórmulas o enunciados) bien formadas.

Si el lenguaje se basa en lógica de primer orden (lo más simple) aparecen variables, constantes, predicados, funciones, conectores lógicos y cuantificadores. Con esos elementos se pueden formular enunciados y construir teorías. Con las variables por ejemplo se pueden caracterizar estados.

Símbolos: unidades elementales del lenguaje formal. Son marcas permitidas por el alfabeto del lenguaje: letras, signos lógicos, paréntesis, cuantificadores, constantes, variables, operadores.

Tokens: unidades léxicas reconocidas por el analizador del lenguaje. Un token puede estar formado por uno o varios caracteres. Por ejemplo, `forall`, `<=`, `variable`, `numero`.

Palabras: secuencias finitas de símbolos del alfabeto. En lenguajes formales, una palabra no necesariamente tiene significado: puede ser sólo una cadena de símbolos. En programación, muchas veces una palabra reconocida por el lenguaje funciona como token.

Gramática: conjunto de reglas que indica cómo se construyen expresiones válidas a partir de símbolos, palabras o tokens.

Sintaxis: forma estructural de las expresiones bien formadas. La sintaxis dice si una expresión está correctamente construida según la gramática.

Semántica: interpretación de las expresiones bien formadas. Indica qué significan, qué denotan o bajo qué condiciones son verdaderas.

Verdad: En semántica formal, una fórmula es verdadera si se satisface en un modelo.

Adecuación empírica: En ciencias físicas no hablamos solo de verdad absoluta del modelo, sino de su adecuación:

El grado en que un modelo representa correctamente ciertos aspectos de un sistema dentro de un dominio de validez. La adecuación se evalúa comparando las predicciones del modelo con observaciones, mediciones o experimentos, revisando los errores, los supuestos usados y el rango de condiciones en que el modelo funciona.

Predicción: uso de un modelo para estimar un estado, valor o comportamiento no observado todavía, a partir del estado conocido del sistema, sus parámetros, sus condiciones iniciales y sus reglas de evolución. En ciencias físicas, una predicción se contrasta luego con observaciones, mediciones o experimentos.

En definitiva, la predicción es el uso de un modelo para estimar qué ocurrirá, o qué valor tendrá una magnitud, bajo ciertas condiciones dadas.

La relación queda así:

símbolos $\longrightarrow$ palabras/tokens $\longrightarrow$ expresiones bien formadas $\longrightarrow$ interpretación semántica

Ejemplo:

$$\forall x \, (P(x) \to Q(x))$$

Ahí:

- símbolos: $\forall$, x, P, Q, (,), $\to$;
- tokens: cuantificador, variable, predicado, paréntesis, conector;
- sintaxis: la fórmula está bien formada;
- gramática: las reglas que permiten formar esa fórmula;
- semántica: la interpretación que dice cuándo esa fórmula es verdadera.

Teoría

Un **sistema axiomático** o **teoría axiomática** es un conjunto de axiomas formulados en un lenguaje formal.

Hay:

- un **lenguaje**, con símbolos permitidos;
- **axiomas**, que fijan condiciones generales;
- **reglas de inferencia**, que indican qué se puede deducir;
- **teoremas**, que son consecuencias de los axiomas según esas reglas.

Dominio: objetos, conjuntos y lógica de segundo nivel

Una teoría formal no queda completamente comprendida sólo por sus fórmulas (axiomas o teoremas).

También interesa saber cómo se interpretan esas fórmulas. No alcanza con las fórmulas solas para estudiar sistemas tal como queremos hacerlo.

Para eso introducimos un **dominio**: el objeto formal sobre el cual habla la teoría.

Según el marco elegido, ese objeto puede construirse como conjunto, categoría, tipo u otra forma matemática.

En lo que sigue usaremos dominios construidos como conjuntos, por ser el marco más habitual y suficiente para los modelos que vamos a trabajar aquí. Esto no excluye otros marcos formales posibles.

La teoría de conjuntos es una buena herramienta para construir dominios, relaciones y funciones.

La lógica de primer orden, tomada por sí sola, no define el concepto de conjunto y no siempre define el conjunto concreto con el que queremos trabajar.

Por eso es necesario incorporar el concepto de conjunto mediante una teoría de conjuntos, y especificar exactamente de qué se trata. Escalares y vectores cumplen muchas veces los mismos axiomas. Y no es obvio especificar de qué queremos hablar.

Una teoría de conjuntos, como ZFC, puede formularse en lógica de primer orden interpretando el símbolo de pertenencia $\in$ como tal y axiomas que regulan su uso.

A veces la palabra objeto, además de nombrar un conjunto u otra construcción formal, puede referirse a un elemento de un conjunto. Nótese que un elemento de un conjunto, también puede ser un conjunto.

También existen lenguajes más expresivos, como la lógica de segundo orden, que permiten cuantificar no sólo sobre objetos individuales, sino también sobre propiedades, relaciones, funciones o subconjuntos. Esa mayor expresividad puede ser útil, aunque introduce problemas técnicos propios y depende de la semántica adoptada.

Muchas veces los estados de un sistema se denominan espacio de estados y se consideran formando un conjunto que se toma como dominio.

El dominio del modelo es el conjunto de estados posibles del sistema, o más generalmente el conjunto de objetos sobre los que el modelo habla. Sobre ese dominio se interpretan las variables, relaciones y ecuaciones. Las proposiciones no son el dominio: son enunciados que pueden resultar verdaderos o falsos al evaluarse en un modelo y, si corresponde, en un estado determinado.

En lógica/modelos formales, el dominio es el conjunto de objetos sobre los que se interpretan las variables, funciones y relaciones.

En un modelo de sistema dinámico, el dominio relevante suele ser el espacio de estados: el conjunto de estados posibles del sistema.

En éste contexto las expresiones, fórmulas, axiomas, proposiciones no forman el dominio; son enunciados del lenguaje que se evalúan como verdaderos o falsos respecto de un estado,

una estructura o un modelo.

Axiomas vs. dominio

Lo relevante en lo anterior y lo que justifica introducir el concepto de dominio es que los axiomas no describen necesariamente un único objeto. Más bien fijan condiciones generales que pueden ser satisfechas por distintas estructuras. Por eso una misma teoría puede tener muchos dominios posibles.

Además, aunque pudiera definirse el dominio totalmente usando axiomas, muchas veces el concepto de dominio a partir de una teoría como la de conjuntos aporta claridad, ya que nos da directamente lo que necesitamos. Puede haber un largo camino entre axiomas y un dominio hecho y derecho.

La diferencia entre un sistema axiomático y un dominio es la diferencia entre **definir condiciones abstractas** y **dar una realización concreta**.

Un sistema axiomático describe qué debe cumplirse. Una estructura muestra dónde y cómo se cumple.

Así, los **axiomas** son enunciados generales formulados en un lenguaje formal. No contienen por sí mismos objetos concretos, sino condiciones que cualquier interpretación aceptable debe satisfacer.

Por ejemplo, los axiomas de grupo dicen, entre otras cosas:

- hay una operación binaria;
- la operación es asociativa;
- existe un elemento neutro;
- cada elemento tiene inverso.

Eso todavía no dice si hablamos de números enteros, matrices, simetrías, permutaciones o transformaciones geométricas. Sólo fija condiciones.

Estructura e interpretación

Una **estructura** está formada por:

- un **dominio**, es decir, el conjunto (objeto en general) considerado.
- una **interpretación** de los **símbolos** del lenguaje: **operaciones**, **relaciones**, **funciones** o **constantes**, definidos sobre ese dominio.

Los conectores lógicos, como $\wedge$, $\vee$, $\neg$ o $\rightarrow$, y los cuantificadores, como $\forall$ y $\exists$, no son elegidos por cada estructura: pertenecen a la lógica general. Lo que una estructura interpreta son los símbolos no lógicos del lenguaje.

Por ejemplo, en una teoría de grupos puede aparecer un símbolo de operación $*$. Ese símbolo no es todavía la suma, la multiplicación ni la composición. En la estructura $(\mathbb{Z}, +, 0)$, el símbolo $*$ puede interpretarse como la suma y el elemento neutro como 0. En otra estructura, el mismo símbolo puede interpretarse como composición de simetrías.

Modelos

Un **modelo** es una estructura considerada semánticamente, para serlo debe:

- permitir evaluar el valor de verdad de las fórmulas (expresiones, axiomas) en relación con la estructura.
- resultar positiva esa evaluación para todos los axiomas, es decir, que estos resulten verdaderos en relación con la estructura.

Con dominio e interpretación ya tenemos una estructura. Para hablar de modelo hace falta introducir la relación semántica de satisfacción. Una estructura se convierte en **modelo** de una teoría cuando satisface sus axiomas. O sea, cuando, bajo esa interpretación, los axiomas resultan verdaderos.

A los dos aspectos que forman una estructura el modelo añade:

- una condición de **satisfacción**: reglas para determinar cuándo una fórmula, expresión o axioma es verdadera en esa estructura.

Por ejemplo, si el lenguaje tiene: $P(x)$ y el modelo dice que: $P =$ conjunto de los elementos rojos

Entonces, a será verdadero si el elemento nombrado por a pertenece al conjunto de elementos rojos del modelo.

La idea central de la semántica formal es que el significado de una expresión se estudia mediante sus condiciones de verdad: saber qué significa una fórmula (expresión, axioma, proposición) exige saber bajo qué interpretación resulta verdadera o falsa.

Entonces:

Axiomas: dicen qué propiedades deben cumplirse.

Estructura: da objetos concretos y operaciones o relaciones concretas.

Modelo: estructura que cumple los axiomas.

La idea central: Con axiomas solamente tenemos una teoría abstracta. Con una estructura tenemos una interpretación concreta. Si esa estructura satisface los axiomas, entonces tenemos un modelo.

Aplicado a ciencias físicas:

Un conjunto de ecuaciones puede ocupar el lugar de axiomas o condiciones básicas del modelo: leyes, restricciones, relaciones constitutivas y reglas de evolución. Pero para predecir un sistema concreto no alcanza con esas ecuaciones generales. Hace falta una estructura interpretada: variables definidas, parámetros, condiciones iniciales, condiciones de borde, dominio de trabajo, relaciones entre magnitudes e interpretación física de cada símbolo. Esa estructura interpretada es el modelo efectivo del sistema estudiado.

Modelos matemáticos y predicción

Como vimos, un sistema material puede ser representado mediante:

- una estructura matemática: variables, parámetros, relaciones, ecuaciones, condiciones iniciales y condiciones de borde;
- cuando corresponde, reglas de evolución.

Si sólo describimos relaciones posibles entre magnitudes, tenemos una estructura formal que representa la estructura de un sistema material. Si además indicamos cómo cambia el estado del sistema, tenemos un sistema representado: una estructura formal más reglas de evolución.

Así, un **modelo matemático** de un sistema material es otro sistema, interpretado físicamente.

En el caso estático, funciona como estructura formal; en el caso dinámico, como estructura formal más reglas de evolución. Está construido para describir el sistema material y calcular magnitudes desconocidas, presentes y futuras.

Generalmente el cambio mantendrá la estructura con distintos valores en sus variables, solo en casos especiales habrá cambios de estructura, podemos asimilar metafóricamente esos cambios a los cambios de fase en las sustancias.

Es decir, un sistema puede cambiar aún con una estructura dada fija, pero con valores diferentes en diferentes estados. Por eso es compatible pensar en una estructura fija representando sistemas dinámicos. También puede cambiar de estructura de un momento a otro. En general es más complejo estudiar este tipo de sistemas. Los cambios de fase nos permiten pensar en ello. Si calentamos un cuerpo en condiciones de la vida diaria, su forma al comienzo no cambiará. Luego pasará a líquido y finalmente a gas. Cada una de esas fases puede ser representada con estructuras y fórmulas diferentes. Mientras la fase no cambia la estructura no cambia, sí cambia el valor de las variables. Algunos modelos sobre la economía usan estas metáforas. Cuando hay una revolución cambian las estructuras sociales en forma cualitativa. Sin revolución hay cambios cuantitativos evolutivos dentro del mismo sistema político.

Si el sistema no cambia o solo cambia mediante intervención externa, las estructuras dan relaciones que describen sus posibles estados internos. De hecho en este caso se puede hablar de estructura más que de sistema.

Cuando el modelo incluye una regla de cambio temporal, hablamos de un **sistema**. En ese caso no basta con describir qué objetos, variables y relaciones hay en cada momento o configuración. También debe indicarse cómo evoluciona el estado del sistema en el tiempo. A las estructuras se les suma el cambio y se estudia como sistema, con partes que se relacionan y cambian entre sí

Si el estado debido a las reglas de evolución llega a un equilibrio, como se postula en termodinámica para sistemas aislados, será un sistema en equilibrio, conducido a este por sus reglas de cambio. No es una estructura, no es inmutable. Es un sistema que en esas condiciones ya no cambia.

Podemos resumirlo así:

1. Hay dos sistemas, el material y el representado. El sistema material pertenece al universo físico; el sistema representado pertenece al plano formal del modelo.

2. En el caso inmutable, el sistema material puede pensarse como una estructura física. Si bien no es muy útil esta idea. En general lo que está bajo estudio es siempre denominado sistema. Y se lo representa siempre con una estructura formal, cuyas variables cambian con el tiempo. Eventualmente puede también cambiarse el modelo. Formalmente puede pensarse en una sola estructura que contiene las estructuras antes y después del cambio.

3. En ambos casos, inmutable o dinámico, usamos una estructura formal para representarlo.

4. La estructura formal del caso dinámico muestra, en cada momento, una configuración o "foto" del sistema material, por ello, para representarlo fielmente agregamos reglas de cambio.

5. Si sólo hay relaciones posibles entre magnitudes, hay estructura formal fija representando a una estructura material. No hay cambio en los valores de las variables, sin acción externa.

6. Si hay reglas de evolución, hay sistema en el universo material y estructura formal en el modelo. Estructura con valores que cambian, y/o cambios de estructura.

7. Un sistema en equilibrio no es una estructura inmutable: es un sistema que llegó al equilibrio por sus reglas de cambio y en esas condiciones ya no cambia. Si el equilibrio es estable y se lo perturba, cambia, pero vuelve al equilibrio.

8. Una estructura (sea representación, la formal, o física) es un caso particular de sistema, uno inmutable.

Tipos de modelos

Los modelos pueden ser:

Físicos: maquetas, túneles de viento, prototipos a escala.

Analógicos: un circuito eléctrico que reproduce el comportamiento de un sistema mecánico.

Matemáticos: representaciones formales mediante variables, parámetros, relaciones, ecuaciones, condiciones iniciales, condiciones de borde y reglas de evolución. Son los más generales y los que estudiaremos.

Un **modelo matemático** no es sólo un conjunto de ecuaciones. Es una representación formal del sistema: variables, parámetros, dominio, relaciones, ecuaciones, condiciones iniciales, condiciones de borde e interpretación física de esas magnitudes.

Estático o dinámico: un modelo estático describe el sistema que no puede cambiar; un modelo dinámico describe cómo evoluciona el sistema en el tiempo, incluye modelos que llegan al equilibrio y no cambian salvo cambios externos o perturbaciones.

Determinístico o estocástico: en un modelo determinístico, el estado futuro queda completamente determinado por el estado actual; en un modelo estocástico interviene el azar mediante variables aleatorias o incertidumbre probabilística.

Lineal o no lineal: los modelos lineales admiten el principio de superposición y son más fáciles de analizar; los no lineales son más ajustados al mundo material pero más difíciles de tratar. Permiten representar dependencias, interacciones y cambios de régimen que no pueden describirse mediante superposición simple.

Por ejemplo, en la mecánica tenemos, muchas veces, sistemas lineales, en la termodinámica no. La forma general es similar teniendo en la acción un símil de la entropía.

Continuo o discreto: el tiempo, el espacio o las variables del modelo pueden representarse como magnitudes continuas o como valores discretos.

4.22.2. Ciencia y filosofía

Definiciones desde mi perspectiva.

La ciencia: es la capacidad de predecir el futuro en forma contrastable o en todo caso la actividad que produce avances en eso. Es experimental. Debe ser refutable porque toda teoría debiera poder ser negada por al menos un experimento que pueda fallar. No sirven, a este efecto, experimentos que no puedan fallar.

Disminuye el error de las predicciones, amplía el ámbito de aplicación y unifica diferentes campos en un mismo marco. De lo que sigue se concluye que se requiere consenso para que una temática se considere en el ámbito científico, si bien no para emprender el desarrollo de teorías. Lo que impulsa el consenso es la capacidad de predecir, el acuerdo es posterior. Requiere financiamiento.

La predicción empírica contrastable es el criterio principal de cientificidad.

La ciencia es empírica. No tiene sentido la expresión ciencia empírica, porque no hay otra. Muchas personas no coinciden; personalmente prefiero poner otros nombres a otras actividades, es una discusión de diccionarios, no de fondo.

Es interesante notar que si definimos ciencia como predicción y para predecir, en casos concretos, hacemos modelos y los simulamos, la ciencia y la capacidad de simular, forman parte del mismo concepto. A medida que generalizamos sistemas nuestra ciencia se hace mas abstracta. Pero no deja de ser la capacidad de saber, o el trabajo concreto de buscar como, para cada sistema material del universo, se puede hacer un modelo y simularlo.

La filosofía: incluye aquello que la ciencia no ha cubierto, donde la humanidad no ha logrado consenso, donde no siempre hay predicciones o no son razonablemente precisas. Es otro tipo de conocimiento.

Para identificar una posición filosófica con claridad debe visualizarse un conjunto de ideas contradictorias sobre un mismo tema y la imposibilidad de resolver la disputa a favor de una u otra sobre una base racional y universalmente aceptada sobre el universo de aplicación.

No hay consenso ni formas indiscutibles de cerrar el problema. En general, estas disputas también expresan intereses diferenciados de distintos actores; cuando esos intereses pasan al terreno de la acción colectiva, entramos directamente en la política.

Idea base: instrumentalismo predictivo y operacional

Estudiamos primero el conocimiento de base que describe lo que la ciencia hace.

Podemos llamarlo, si decidimos darle un nombre como si fuera idea filosófica: "Instrumentalismo predictivo y operacional".

Para estudiar un sistema material se construye una estructura formal que funciona como modelo cuando sus símbolos se vinculan con procedimientos de medición y sus resultados pueden compararse con observaciones, mediciones o resultados experimentales.

1. Hay dos planos: el sistema material estudiado y el modelo formal usado para estudiarlo.
2. El sistema estudiado se aborda mediante magnitudes observables, medibles o calculables a partir de mediciones.
3. El modelo formal se construye con variables, parámetros, relaciones, ecuaciones, condiciones iniciales, condiciones de borde y, cuando corresponde, reglas de evolución.
4. Si sólo se describen relaciones entre magnitudes, sin reglas de evolución, el modelo es estático.
5. Si se describen cambios de estado mediante reglas de evolución, el modelo es dinámico.
6. Un modelo en equilibrio no debe confundirse con ausencia de dinámica: puede ser el resultado de reglas de evolución que llevan a un estado estable.
7. La validez científica del modelo se evalúa comparando sus resultados con observaciones, mediciones o resultados experimentales.
8. La falsabilidad nos asegura que no estamos ante una tautología o "trampa" lógica. En tal sentido sostenemos que la matemática no es ciencia, en todo caso es lo que hay entre axiomas, estructuras y teoremas. O lo que hay en el lenguaje. Matemática es lenguaje. Es un auxiliar de la razón y de la ciencia, pero no ciencia.

La distinción queda limpia:

Sistema material: parte del universo físico tomada como objeto de estudio.

Si se quiere evitar el uso de la palabra material, se puede usar "físico", pero sería superficial, porque, en última instancia, es sólo un significante, al cual le damos el significado que queremos.

Sistema formal o matemático: estructura simbólica construida con variables, parámetros, relaciones, ecuaciones y reglas.

Modelo: sistema formal interpretado para describir, calcular o predecir magnitudes de un sistema material.

Esa formulación no abandona el estudio del universo. Al contrario: lo deja claro. Sólo separa el uso científico de "materia" de la disputa filosófica materialismo – idealismo.

De todo lo anterior, el punto que requiere atención filosófica puede resumirse así:

"Una estructura formal representa un sistema material mediante medición y predicción".

La afirmación, si se la lee sin más, puede ser entendida como más que pura ciencia: podría contener ideas filosóficas sobre la ciencia, de varios tipos:

Preguntas filosóficas

A. ¿Es el mundo predecible?

B. ¿Qué lugar tiene el azar en la predicción científica?

Las primeras dos las respondemos en términos de la ciencia pasando por el costado a la filosofía.

Decimos que hemos encontrado sistemas de los cuales podemos predecir en parte su comportamiento hasta un determinado error, por ejemplo 0.00001 % en determinada magnitud.

No podemos responder universalmente para todos los sistemas y todos los dominios científicos esta cuestión. Pero vamos avanzando y disminuyendo los errores de las predicciones.

Hemos aprendido a manejar el azar en varios sistemas, ya sea porque los errores se comportan aleatoriamente o porque hay sistemas, sobre todo cuánticos con resultados de experimentos que son aleatorios. Manejamos esto con la teoría de la probabilidad y obtenemos funciones de distribución de probabilidades más que números inmutables.

Podría implicar que:

- Existen sistemas "materiales", que pueden ser medidos y predichos, y que una estructura formal puede representarlos al menos parcialmente y predecirlos.

  Sin embargo, una lectura cuidadosa no nos dice que todos los sistemas puedan ser representados y predichos.

  Si bien muchas ciencias lo afirman en su ámbito de competencia. Por ejemplo, la termodinámica afirma que los sistemas materiales, partes del universo cumplen ciertas leyes que tienen carácter predictivo.

  En todo caso, para lo que nos compete, habrá sistemas materiales que puedan ser predichos y a los cuales se les aplica lo que decimos. Y esto es mucho.

Por otra parte, existe un conjunto de pensadores, agrupados como bayesianistas que entienden la confirmación científica como actualización racional de probabilidades ante nueva evidencia. Es especialmente relevante para discutir azar, incertidumbre, grados de creencia y evaluación probabilística de hipótesis.

Ahora, para mí, el bayesianismo no agrega una cuestión filosófica central nueva. Lo que agrega pertenece al debate general sobre probabilidad, incertidumbre y confirmación.

Para mí la ciencia aumenta la precisión reduciendo el error de predicción, y en sistemas aleatorios predice distribuciones de probabilidad en vez de valores únicos.

Entonces, el bayesianismo queda como un teorema, una técnica o enfoque particular dentro de un problema más amplio de manejo de incertidumbre, evidencia y probabilidad.

C. ¿Es la materia la realidad última?

D. ¿Influye la subjetividad en el comportamiento del universo por vías no directas? Vías directas serían que el experimentador esté estresado y patee un instrumento.

No me refiero a cuestiones sociales históricas o políticas, que ya están en otras preguntas, sino a:

La observación científica no es una recepción pasiva de datos puros: depende de habilidades, expectativas, conceptos previos, juicio experto y formas aprendidas de percibir e interpretar.

E. ¿Influyen la historia y la sociedad en la ciencia? Y, si lo hacen, ¿es buena ciencia? Es evidente que sí lo hacen: asignan presupuestos, matan a científicos, etc. La pregunta es básicamente si debemos intentar aislar a los científicos de esto. Y pensamos a la ciencia como una actividad encerrada en sí misma.

F. ¿Deben imponerse objetivos sociales, políticos o económicos a la ciencia?

G. ¿Sirve una descripción en términos de mecanismos reales? ¿El modelo da eso? ¿El modelo y la ciencia sólo sirven para calcular y predecir?

H. ¿Qué significa interpretar un modelo?

- Estamos ante una interpretación materialista, al decir "sistema material" y por ello pueden incomodarse los idealistas.

  Hasta que se dice que sistema material es simplemente lo que se mide.

  Bunge afirma que lo que se mide es la realidad material. El marxismo–leninismo coincide y agrega que no sólo es cuestión de interpretar al mundo sino de transformarlo y que una interpretación implica un tipo de transformación.

  Coincido.

  Pero no es necesario hacerlo. Con ese cuidado, el materialismo, el marxismo y el idealismo agregarían lo suyo sin negar lo otro dicho y todos felices.

  Entonces, si sólo usamos la palabra material en el sentido indicado en la definición de sistema material, el problema se resuelve. No afirma supremacía de la materia, ni que ésta sea la realidad última, ni que las ideas emerjan de la materia. Sólo afirma que hay sistemas que son parte del universo y que denominamos materiales.

Luego de estudiar el mínimo práctico y las diversas filosofías, veremos qué responde cada una a las preguntas efectuadas.

Como mínimo práctico de la ciencia empírica, casi todas las corrientes aceptarían esto: una teoría se formula mediante estructuras formales, se interpreta vinculándola con mediciones, y se evalúa por su capacidad de describir, calcular, predecir o contrastar fenómenos.

Cuando decimos que la ciencia intenta predecir medidas futuras con cada vez más precisión, casi todos coincidirán, si bien agregarán otras consideraciones. Algunos tomarán como ciencia otras prácticas no predictivas, pero no son las que estudiamos acá. O sea, podemos definir otras prácticas como ciencia. No niego esa posibilidad. Sólo no la adopto.

Autores que apuntan a este núcleo: Percy Williams Bridgman, Pierre Duhem, Henri Poincaré, John Dewey, Charles S. Peirce, Karl Popper.

Tesis

- Un modelo vale por su capacidad de calcular, medir, operar, predecir y exponerse a contraste o refutación.
- Incluye el falsacionismo como exigencia de predicciones refutables, el operacionalismo como vínculo con procedimientos de medición, y el convencionalismo como atención a la elección de marcos formales útiles.
- El sentido de una teoría o modelo depende de sus consecuencias prácticas, uso, investigación y resolución de problemas, no de una correspondencia ontológica rígida.
- Trabaja la formalización, la distinción entre lenguaje, teoría, estructura y modelo, y la necesidad de conectar los términos teóricos con observaciones o mediciones.

Interpretar:

- es asignar a una estructura formal un uso calculatorio dentro de una teoría.
- no exige que la estructura copie la realidad; basta con que ordene fenómenos, permita operar con ellos y produzca resultados útiles.
- en una variable, implica saber qué mide, cómo se mide y qué predicciones permite calcular.

En resumen, lo planteado es compatible con la ciencia empírica corriente, para no caer en una posición filosófica hay que limitar el alcance del concepto de sistema material a la medición, interpretación física y predicción.

Las diversas corrientes filosóficas difieren en lo que agregan a ese mínimo.

¿Ausencia de posición es posición?

Se podría considerar una posición filosófica por ausencia de postulados filosóficos diversos.

Siempre es posible objetar que la ausencia de postulados filosóficos explícitos funciona, ante alternativas que sí los tienen, como una postura filosófica por sí misma.

Esto aparece claramente en la cuestión ontológica sobre el "materialismo de la realidad". En este texto ese debate se reconoce, pero se lo deja fuera del desarrollo de este trabajo. Esa decisión también puede leerse como una posición filosófica. En mi caso no lo es, porque

planteo esa discusión en otros ámbitos y fijo allí mi posición. Pero sí ocurre con algunas corrientes, como el positivismo, que desplazan explícitamente el debate ontológico como forma de negar o suspender esa discusión.

Resumiendo, hay tres opciones:

1. definirse (lo hago pero no desarrollo ni debato acá),
2. posponer el tema (lo que hago),
3. negar el tema (lo que hace el positivismo).

Es claro que cualquier filosofía de la ciencia, y en general toda filosofía debiera encarar el debate materia – ideas. Y no hacerlo esconde algo.

Puntos filosóficos:

En general, las filosofías de la ciencia comparten la base anterior: el mínimo operativo de la ciencia empírica consiste en medir, formalizar, calcular, predecir y contrastar con mayor o menor entusiasmo. La diferencia es filosófica: cambia la interpretación de esa práctica, no el núcleo operativo que permite hacer ciencia. Además, cada corriente plantea cuestiones propias de la filosofía, como la ontología vinculada a la realidad.

Es posible que existan corrientes con otra visión, lo que discutiría en ese caso es si hacen filosofía de la ciencia o se refieren a otro tema.

Realismo científico materialista: Agrega:

- El modelo científico representa sistemas materiales, mecanismos y relaciones objetivas.
- Enfatiza el sistema material real, mecanismos, ontología y explicación, no sólo predicción.

Autor: Mario Bunge

Tesis: un modelo científico no es sólo una herramienta predictiva ni una estructura formal vacía. Debe tener interpretación factual: sus variables, relaciones y ecuaciones deben referirse a propiedades, procesos o relaciones de sistemas materiales. La ciencia busca describir, explicar y contrastar mecanismos reales, no sólo salvar apariencias.

Interpretar: es vincular una estructura formal con un sistema material real, de modo que sus variables, parámetros, relaciones y ecuaciones refieran a propiedades, procesos o mecanismos materiales controlables empíricamente.

Positivismo, empirismo formal y empirismo constructivo: Agrega:

- Impulsa la reconstrucción lógica del lenguaje científico, las reglas de conexión directa o indirecta con observación y medición, y la adecuación empírica a fenómenos observables.
- Evita comprometerse con que el modelo describa entidades o estructuras que dice no observar directamente. En particular no se compromete con la ontología material de la realidad.

- Pero es más cauteloso con expresiones como "estructura o sistema material real", porque su interés principal no es afirmar una ontología del mundo, sino reconstruir lógicamente el lenguaje científico y sus condiciones de verificación o confirmación.

Autores: Rudolf Carnap, Carl Hempel, Hans Reichenbach, Bas van Fraassen.

Tesis:

- Importa la reconstrucción lógica del lenguaje científico, la relación entre términos teóricos y observacionales, y la confirmación.
- No necesariamente sostiene una ontología materialista fuerte como en Bunge.
- No exige creer que el modelo sea verdadero respecto de entidades no observables; basta con que sea empíricamente adecuado respecto de lo observable.

Interpretar:

- Es vincular el modelo teórico con fenómenos observables. La interpretación no compromete necesariamente con la existencia real de entidades no observables; exige que el modelo sea empíricamente adecuado respecto de lo que puede observarse y medirse.
- Una estructura formal adquiere sentido científico cuando sus símbolos pueden vincularse con observaciones, mediciones y procedimientos de verificación o confirmación.

Marxismo: Agrega:

- Sostiene la referencia al sistema material real. La predicción no agota la ciencia, importan las explicaciones de la realidad.
- La predictibilidad es derivada, es una consecuencia importante de conocer relaciones reales
- El conocimiento científico se verifica y desarrolla en una práctica material orientada a transformar el mundo. Importan las prácticas históricas, sociales, políticas y las transformaciones que la ciencia provoca y que la permiten.

Coincido con estas afirmaciones, pero las entiendo como filosofía y política, no como ciencia.

Autores: Karl Marx, Friedrich Engels, Vladimir Ilich Lenin.

Tesis:

- La ciencia no se reduce a predicción. Debe insertarse en una práctica material, técnica, social e histórica que permite conocer y transformar el mundo.
- Un modelo científico vale cuando permite comprender relaciones materiales y actuar sobre ellas en la práctica.
- La predicción importa, pero subordinada a una idea más amplia: conocer leyes, tendencias, contradicciones y condiciones materiales para transformar el mundo.

- En *Materialismo y empiriocriticismo*, Lenin defiende que el conocimiento científico refleja la realidad objetiva, aunque de modo histórico, aproximado y corregible.
- Para Lenin, la ciencia no es sólo instrumento predictivo; es conocimiento objetivo de una realidad independiente de la conciencia.
- La ciencia produce representaciones cada vez más adecuadas de la realidad objetiva. Su capacidad predictiva importa porque confirma que esas representaciones captan relaciones reales del mundo material.

Interpretar un modelo es relacionar una forma teórica con procesos materiales objetivos y con la práctica capaz de verificar, transformar y corregir ese conocimiento. La interpretación no se agota en predecir: debe expresar relaciones reales y probarse en la práctica.

Hermenéutica e interpretación histórico–constructiva: Agrega:

- Toda interpretación o reconstrucción depende no sólo de medición y predicción, sino también de lenguaje, historia, prácticas, intereses y marcos conceptuales.
- No hay una conexión puramente neutra entre estructura formal y sistema material.

Autores: Martin Heidegger, Hans-Georg Gadamer, Paul Ricoeur, Thomas Kuhn, Paul Feyerabend, Ernst von Glasersfeld.

Tesis: La interpretación científica no depende sólo de medición y predicción, sino también de lenguaje, historia, prácticas, intereses y marcos conceptuales.

Interpretar: es comprender el modelo desde un marco de lenguaje, historia, prácticas, intereses y conceptos compartidos; no hay interpretación puramente neutra entre estructura formal y sistema material.

Estructuralismo y modelos idealizados: Agrega: El foco se desplaza desde enunciados aislados hacia familias de modelos, estructuras, idealizaciones o ficciones útiles para representar y estudiar fenómenos.

Autores: Nancy Cartwright, Ronald Giere, Mary Morgan, Margaret Morrison, Joseph D. Sneed, Wolfgang Stegmüller, Patrick Suppes.

Tesis:

- Los modelos no son copias ni representaciones literales del sistema material; son construcciones parciales, idealizadas, ficcionales o mediadoras.
- Las teorías científicas se entienden como familias de modelos o estructuras, no como enunciados aislados interpretados de modo bungeano.

La interpretación:

- Usa una construcción simplificada, parcial o ficticia para intervenir sobre un fenómeno.
- El modelo no reproduce literalmente la realidad; selecciona rasgos, omite otros y permite calcular, medir o predecir dentro de ciertos límites.

- Interpretar o reconstruir una teoría es presentarla como familia de modelos o estructuras. Lo importante no es una oración aislada, sino el sistema de relaciones formales que permite representar clases de fenómenos.

Tabla compacta:

Cuadro 4.2: Respuestas filosóficas a las preguntas C–H

Posición	**C**	**D**	**E**	**F**	**G**	**H**
Realismo científico materialista.	Sí.	No como influencia directa; sí como parte del contexto de investigación.	No. No crea leyes. Influye socialmente.	No inventa verdad. Puede orientar la búsqueda.	Sí. Representa sistemas reales. No es sólo predicción.	Referir a procesos materiales.
Positivismo / empirismo.	Suspende la cuestión.	No como influencia directa; se controla mediante observación y método.	No; es sólo contexto externo.	No define cientificidad.	No es su interés, sólo como adecuación observable.	Conectar con observación.
Marxismo.	Sí.	No como influencia directa; importa más la práctica social e histórica.	Sí. Influye fuertemente.	Sí, práctica transformadora.	Sí. Busca relaciones reales.	Insertar en la práctica material.

Hermenéutica / constructivismo.	Depende. Se interpreta dentro de una tradición teórica, histórica y lingüística.	Sí, en la interpretación; no necesariamente en el comportamiento del universo.	Influye constitutivamente.	Sí, por intereses y prácticas.	No busca mecanismos reales; enfatiza construcción situada.	Comprender desde marcos históricos.
Estructuralismo / modelos idealizados.	No es el foco.	No es el foco; el foco está en estructuras.	Sí; influye en la construcción de modelos.	No necesariamente; depende del uso de los modelos.	No como centro; prioriza familias de modelos, idealizaciones y relaciones estructurales.	Ubicar en una estructura o familia de modelos.

Nótese que no hay dos corrientes que respondan Sí a las mismas preguntas. Es decir, las preguntas nos sirven para discriminar todas las ideas.

Cuadro 4.3: Respuestas afirmativas por corriente

Posición	**C**	**D**	**E**	**F**	**G**
Realismo científico materialista	Sí				Sí
Positivismo / empirismo					
Marxismo	Sí		Sí	Sí	Sí
Hermenéutica / constructivismo		Sí	Sí	Sí	
Estructuralismo / modelos idealizados			Sí		

Debe decirse que Berkeley plantea un subjetivismo fuerte, pero más que en el debate sobre la ciencia entra en el debate ontológico de fondo idealismo vs. materialismo.

Otras corrientes y preguntas posibles

Las corrientes anteriores alcanzan para discriminar las posiciones principales frente a las preguntas planteadas. Sin embargo, podrían agregarse otras filosofías de la ciencia si se quisiera ampliar el mapa.

Pluralismo epistemológico / anarquismo metodológico: Paul Feyerabend critica la idea de un método científico único y universal. Su posición no encaja bien en el positivismo, el realismo materialista ni el instrumentalismo predictivo. Puede ubicarse cerca de la interpretación histórico–constructiva, porque subraya historia, prácticas, controversias, estilos de investigación y pluralidad de métodos.

Programas de investigación: Imre Lakatos propone entender la ciencia como competencia entre programas de investigación, con un núcleo duro protegido por hipótesis auxiliares. Una teoría aislada no se abandona por una anomalía puntual; importa si el programa es progresivo o degenerativo según produzca nuevos problemas, predicciones y desarrollos.

Realismo crítico: El realismo crítico insiste en que la ciencia busca mecanismos generativos reales, incluso cuando no son directamente observables. Se aproxima al realismo materialista, pero enfatiza sistemas abiertos, niveles de realidad y mecanismos subyacentes.

Epistemología social y feminista: Estas corrientes subrayan que la objetividad científica no depende sólo del método individual, sino también de comunidades, crítica pública, instituciones, valores, intereses, género, poder y organización social de la investigación.

Otras preguntas posibles

I. ¿Puede una teoría científica referirse legítimamente a entidades no observables?

J. ¿La explicación científica exige descubrir mecanismos causales reales?

K. ¿Los valores sociales, éticos o políticos deben intervenir en la elección de problemas, métodos o aplicaciones científicas?

L. ¿La objetividad científica depende sólo del método individual o también de comunidades, crítica pública e instituciones?

M. ¿Una teoría puede ser científicamente aceptable aunque use idealizaciones o supuestos falsos?

N. ¿La ciencia progresa acumulativamente o mediante rupturas de paradigmas, marcos, modelos o programas de investigación?

O. ¿La probabilidad expresa azar objetivo, ignorancia, grado de creencia o herramienta de cálculo?

P. ¿La matemática es parte de la ciencia o una herramienta formal auxiliar?

Q. ¿Qué pesa más en ciencia: predicción, explicación, intervención o comprensión?

Estas preguntas no reemplazan las anteriores. Sirven para ampliar la comparación si se quiere distinguir con más detalle entre Popper, Kuhn, Lakatos, Feyerabend, realismo crítico y epistemologías sociales.

Una investigación seria sobre el tema debiera hacer muchas preguntas, analizar a nivel de pensadores individuales y en función de las respuestas, agrupar pensamientos. Mi apuesta sería que el resultado sería similar a la tabla presentada y fortalecería la tesis que separa ciencia de filosofía en la filosofía de la ciencia.

Leyes de la ciencia y ecuaciones

Las leyes de la ciencia son relaciones matemáticas entre magnitudes físicas que se verifican experimentalmente y tienen validez universal dentro de su dominio de aplicación. Son la materia prima de los modelos matemáticos de sistemas físicos.

Pero no constituyen por sí solas el modelo completo: deben combinarse con hipótesis, parámetros, condiciones iniciales, condiciones de borde e interpretación física de las variables.

Algunas leyes fundamentales que se expresan como ecuaciones diferenciales:

Segunda ley de Newton: $\mathbf{F} = m\mathbf{a} = m\dfrac{d^2\mathbf{x}}{dt^2}$. La fuerza neta sobre un cuerpo es igual a su masa por su aceleración.

Leyes de la Termodinámica: Conservación de la energía–masa, crecimiento de la entropía en los procesos irreversibles. Definición de temperatura y entropía cero a temperatura cero.

Ley de enfriamiento de Newton: $\dfrac{dT}{dt} = -k(T - T_{\text{amb}})$. La tasa de cambio de temperatura de un objeto es proporcional a la diferencia entre su temperatura y la del ambiente.

Crecimiento y decaimiento exponencial: $\dfrac{dN}{dt} = kN$. La tasa de cambio de una población o cantidad es proporcional a la cantidad actual. Solución: $N(t) = N_0 e^{kt}$.

Ecuación de un circuito RC: $RC\dfrac{dV}{dt} + V = V_{\text{fuente}}$.

Oscilador armónico: $\dfrac{d^2x}{dt^2} + \frac{b}{m}\dfrac{dx}{dt} + \frac{k}{m}x = \frac{F(t)}{m}$.

El proceso de modelado consiste en:

1. Identificar el sistema y definir las variables relevantes.
2. Aplicar las leyes físicas pertinentes para obtener las ecuaciones del modelo.
3. Simplificar haciendo hipótesis razonables.
4. Resolver el modelo (analíticamente o numéricamente).
5. Validar comparando las predicciones con datos experimentales.
6. Usar el modelo para predicción o diseño.

4.22.3. Simulación

Al hablar del modelo de un sistema que cambia, aparece la posibilidad de predecir su comportamiento futuro dentro de cierto dominio de validez. Y esto es simularlo.

Así, cuando el modelo se resuelve o ejecuta para estudiar su comportamiento, hablamos de **simulación**. Cuando sus resultados se usan para estimar estados no observados o futuros, hablamos de **predicción**.

Así, la **simulación** es la ejecución de un modelo matemático para obtener información sobre el comportamiento del sistema. Cuando las ecuaciones del modelo no tienen solución analítica cerrada, la simulación numérica es la única alternativa práctica.

La simulación permite:

- Estudiar sistemas peligrosos o costosos sin riesgo (accidentes nucleares, colisiones de vehículos, epidemias).
- Explorar escenarios futuros o hipotéticos.
- Optimizar el diseño de un sistema antes de construirlo.
- Comprender el comportamiento de sistemas complejos cuya intuición no alcanza.

La **predicción** es el uso del modelo para estimar el estado futuro del sistema a partir del estado actual. La calidad de la predicción depende de la fidelidad del modelo y de la precisión de las condiciones iniciales. En sistemas caóticos (como la atmósfera), pequeñas incertidumbres en las condiciones iniciales se amplifican exponencialmente, limitando el horizonte de predicción.

Diferentes medios y metodologías para simular

Existen diferentes formas de interpretar y modelar estos sistemas, entre ellas:

1. La forma más directa es que se planteen las variables, se escriban las ecuaciones y se resuelvan, en forma analítica.
2. Por saltos, numérica. A veces no se puede resolver analíticamente o es preferible resolverlas numéricamente.

   Podemos calcular cada paso o "salto". Usando rutinas numéricas que resuelven las ecuaciones cuando es posible ordenarlas de forma de tener las derivadas de las variables de estado despejadas.

   Y podemos usar software que hace esto por nosotros.

   Scilab con ecuaciones diferenciales

   Se plantean las ecuaciones diferenciales y se resuelven, sea analíticamente (hay mucha bibliografía) o en forma numérica, con programas específicos para ello.

   Por ejemplo, se las puede resolver en Scilab con la función ode:

```
function ydot=f(t, y)
    ydot=y^2-y*sin(t)+cos(t)
endfunction
y0=0;
t0=0;
t=0:0.1:%pi;
y = ode(y0,t0,t,f);
plot(t,y)
```

Resultado.

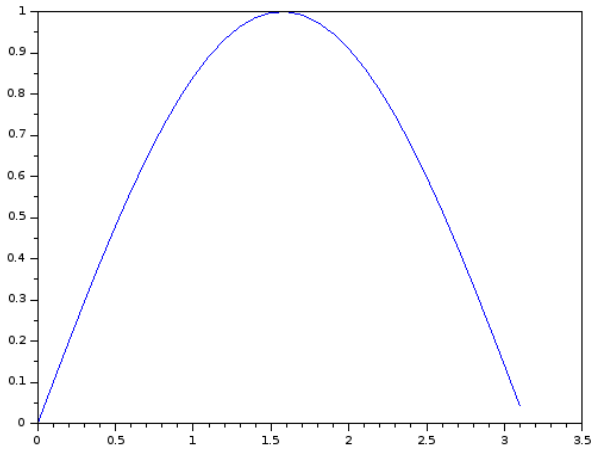


Figura 4.5: Resultado ODE. Scilab.

Sceptre puro

#opa.d

```
opa
DEFINED PARAMETERS
DPX = X1(-6.*PX +5. *PY +10.)
DPY = X2(5.*PX-7.*PY)
DPZ= X3(.2*PY - .2 *PZ-.5))
PX = 6.
PY=5
PZ= 4
OUTPUTS
PX, PY, PZ
RUN CONTROLS
STOP TIME = 20.
END
```

```
sceptre opa
gnuplot -p -e  "plot 'opa.sim' using 1:2 with line, \
   'opa.sim' using 1:3 with line,\
   'opa.sim' using 1:4 with line "
```

Sceptre–simusol es una versión del sceptre modificada para ser usada junto a Simusol.

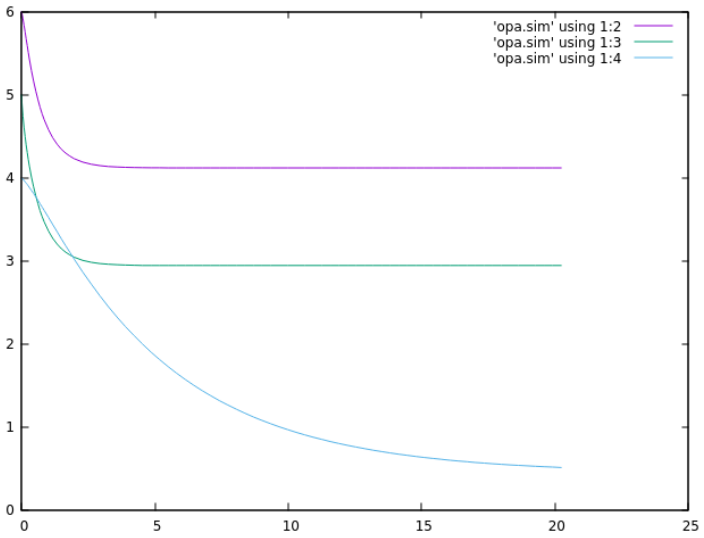


Figura 4.6: Resultado sceptre y gnuplot.

3. Los circuitos o redes representan los sistemas de componentes discretos de una forma visual casi directa.

   En estos casos no siempre se pueden despejar las variables relevantes, hay que resolver sistemas de ecuaciones.

   Existen programas como Spice o Sceptre que resuelven circuitos cuando es posible tener coeficientes que identifican las propiedades de cada elemento en el circuito. Simusol dibuja los circuitos y ejecuta Sceptre. El entorno Xcos con Modelica de Scilab, también puede resolver circuitos.

   Simusol con Sceptre y otros esquemas a parámetros concentrados, circuitos o redes

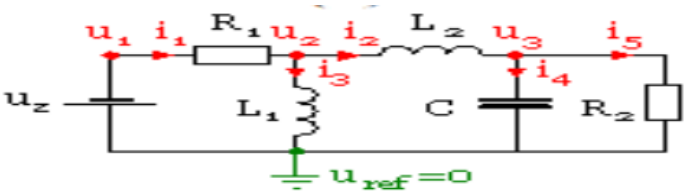


Figura 4.7: circuito

Es el clásico esquema del Simusol y los circuitos eléctricos, cuya teoría ya describimos como base física de todos los métodos de trabajo.

Lo interesante es que la teoría de circuitos puede resolver los circuitos aunque no tenga definido el flujo de información o "la causalidad". En los programas de circuitos no se debe proveer un esquema de causalidad. En su reemplazo se plantean ecuaciones, y se resuelve un sistema de ecuaciones.

No se usa una idea de causalidad (por otro lado arbitraria), sino de interdependencia (ecuaciones de Kirchhoff), tal como plantea la realidad física del problema.

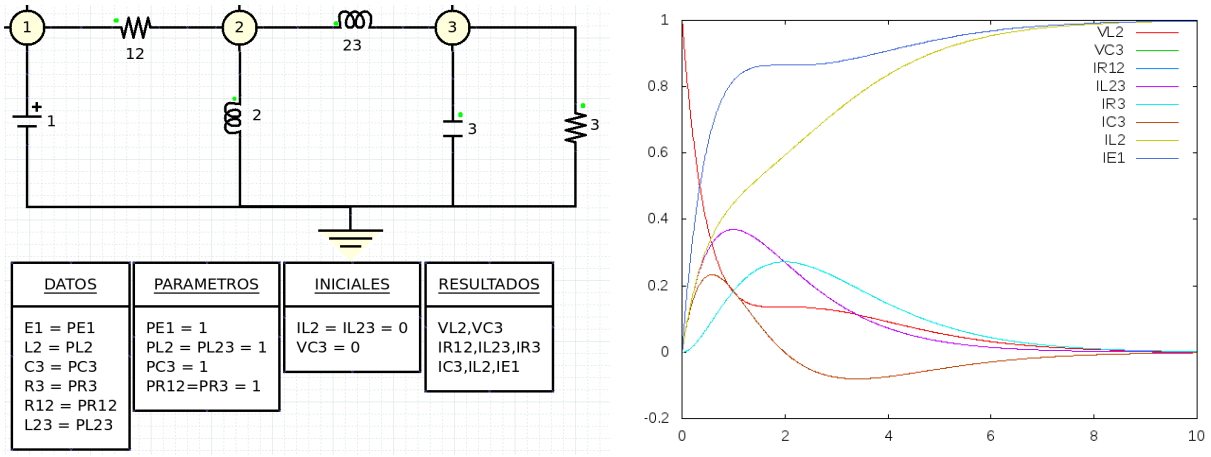


Figura 4.8: circuito en simusol

Xcos y circuitos

En el menú de Scilab en las "demostrations", "Xcos" se puede ver como se modelan circuitos con "Modelica Demos", por ejemplo "RLC"

La extensión permite hacer componentes para diagramas de bloques donde quién es entrada y quién salida se define en el momento y por lo tanto algunos circuitos se pueden representar como circuitos y no como bloques, conectando esta estructura en el marco de un diagrama de bloques tradicional.

Modelica

Es un estándar y un software que permite representar en componentes para diferentes tipos de simulaciones. Varios programas de diagramas de bloques son capaces, con extensiones, de utilizar Modelica, como Xcos.

También la "Functional Mock-up Interface" (o FMI), define una interfaz estandarizada para ser usada en simulaciones para desarrollar sistemas complejos ciberfísicos.

4. Los diagramas de bloques representan un sistema desde el punto de vista de un supuesto flujo de información. No siempre puede realizarse. Sí cuando no hay interdependencia. Implican no sólo representar al sistema sino también indicar una forma en particular de resolverlos.

   Los entornos Xcos de Scilab, Scicos o Simulink de Matlab resuelven sistemas planteados en esta forma.

Diagramas de bloques

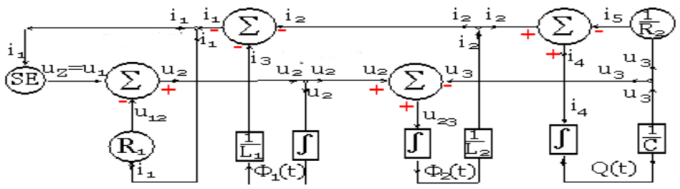


Figura 4.9: Diagrama de bloques o de señales. (Hroncová, Darina. Gmiterko, Alexander. 2013)

Scilab con XCOS. Diagramas de bloques

Un diagrama de bloques de un sistema es una representación gráfica de las funciones que lleva a cabo cada "componente" y su flujo de señales.

Los componentes no son necesariamente los mismos que en el caso de circuitos. De hecho en los circuitos se expresan relaciones que deben satisfacerse, sin indicar cómo se calculan, aquí los componentes indican explícitamente una forma de cálculo, son funciones.

Tales diagramas muestran las relaciones existentes entre los diversos "componentes".

Un diagrama de bloques tiene la ventaja de indicar de forma directa para su cálculo el flujo de las señales del sistema real.

En un diagrama de bloques todas las variables del sistema se enlazan unas con otras mediante bloques funcionales. El bloque funcional o simplemente bloque es un símbolo para representar la operación matemática que sobre la señal de entrada hace el bloque para producir la salida.

Obsérvese que la señal sólo puede pasar en la dirección de las flechas. Por tanto, un diagrama de bloques de un sistema de control muestra explícitamente una propiedad unilateral, aunque puede haber realimentación.

En muchos casos se usan con funciones de transferencia basadas en las transformadas de Laplace.

5. Los diagramas de enlace son una abstracción que representa los sistemas a partir de sus flujos de energía.

   Hay varios programas que trabajan con estos esquemas.

   Bond–Graphs o Diagramas de enlace

   Modelan sistemas físicos.

   Se construye una representación gráfica de un sistema dinámico material.

   Los enlaces representan flujos bidireccionales de energía, independientemente del "dominio".

   Los gráficos tienen enlaces y elementos que pueden tener uno, dos o más puertos.

   Los enlaces transmiten energía en función de dos variables del tipo fuerza y flujo.

   Hay dos tipos de uniones donde van a parar los enlaces, del tipo 0 (suma de flujos 0, esfuerzos iguales, paralelo) y del tipo 1 (suma de esfuerzos 0 y flujos iguales, serie).

   Fueron inventados por Henry Paynter, en el MIT.

   Nota: los diagramas de flujo de señales que se usan en la metodología BG, intercambian nodos y ramas con relación a los diagramas de bloques.

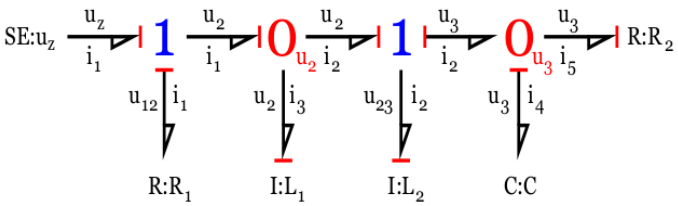


Figura 4.10: Representaciones de diagrama de enlace

Se estandariza el reemplazo de las ecuaciones diferenciales de segundo orden de cada componente y cada dominio con dos ecuaciones diferenciales de primer orden, y se adopta una nomenclatura que generaliza las variables en diferentes dominios, planteando analogías, tal como lo hicimos.

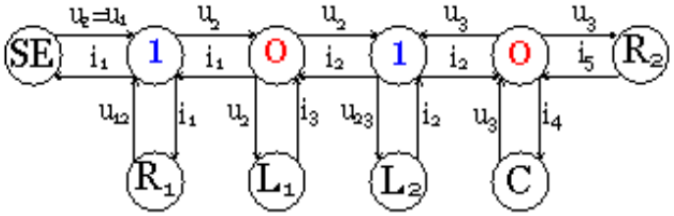


Figura 4.11: Representaciones de diagrama de enlace con flujos bilaterales

Desde el punto de vista teórico, el origen de esta metodología es la "Termodinámica de Redes" (Oster, George F. Perelson, Alan S. Katchalsky, Aharon. 1971; Oster, George F. Perelson, Alan S. 1973; Oster, George F. Perelson, Alan S. 1973; Redlich, Otto. 1968; Romero, Ignacio. 2010; Breedveld, Peter C. 1982; Birkett, S. H. Roe, Andy R. 2002; Salamon, Peter. Andersen, Bjarne. Nulton, James. Konopka, Andrzej. 2006).

Las ideas de la termodinámica de redes inspiran también una metodología alternativa para la enseñanza de la física desde los niveles iniciales (Herrmann. Job. 2006; Herrmann, F. and Schmid, G. B. 1985). Se utiliza la energía y su conservación como núcleo unificador de la disciplina; y presenta una forma concreta de plantear los diferentes dominios análogamente.

Causalidad en bond–graphs

Un punto habitual de la metodología de los diagramas Bond Graphs, del cual se han preocupado numerosos autores, es el estudio de la causalidad. Las relaciones de causalidad representan una secuencia de cálculos que permiten predecir el estado de un sistema a partir del estado actual. Cada sistema representado puede tener más de un esquema de causalidad. El esquema indica, en la simulación numérica, cómo ir calculando sucesivamente las variables necesarias. En muchos casos no existe una causalidad directa que permita llevar de un estado al siguiente y es necesario utilizar programas que numéricamente resuelvan ecuaciones implícitas.

Indicar la causalidad en un diagrama de enlace permite trabajarlo como un diagrama de bloques.

6. Funciones de transferencia:

Transformadas de Laplace. Funciones de transferencia

La función de transferencia para sistemas continuos, con salida y entrada única, se define como la relación entre la transformada de Laplace de la salida y la transformada de Laplace de la entrada, cuando todas las condiciones iniciales son cero.

La "otra" forma de trabajar los sistemas (más tradicional) es el usar funciones de transferencia, donde se usan transformadas de Laplace y se estudia la respuesta de los sistemas para cada frecuencia de las señales de entrada.

La representación de espacio de estados es también conocida como "dominio temporal" y provee una forma compacta de estudiar sistemas con p entradas y q salidas.

Usando funciones de transferencia se debieran tener *pq* transformaciones de Laplace para codificar toda la información sobre el sistema.

Inconvenientes de funciones de transferencia superados por la representación de espacio de estados:

- requiere sistemas con componentes lineales invariantes en el tiempo
- requiere condiciones iniciales cero.
- no da una idea del estado interno del sistema
- no aplica para sistemas con múltiples salidas y entradas
- es complejo de usar en computadoras

Representación de espacio de estados

Es un modelo matemático de un sistema físico como un conjunto de entradas, salidas y variables de estado, relacionadas por ecuaciones diferenciales de primer orden. Se caracteriza por una algebrización de la teoría general de sistemas.

El espacio de estados es el conformado por las variables de estado.

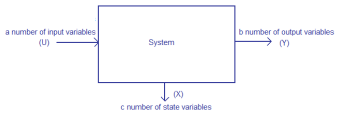


Figura 4.12: Representación de espacio de estados.

4.22.4. Modelos de parámetros concentrados, circuitos

Hay muchas formas de modelar sistemas materiales, se puede usar métodos para mecánica de medios continuos, parámetros concentrados, etc.

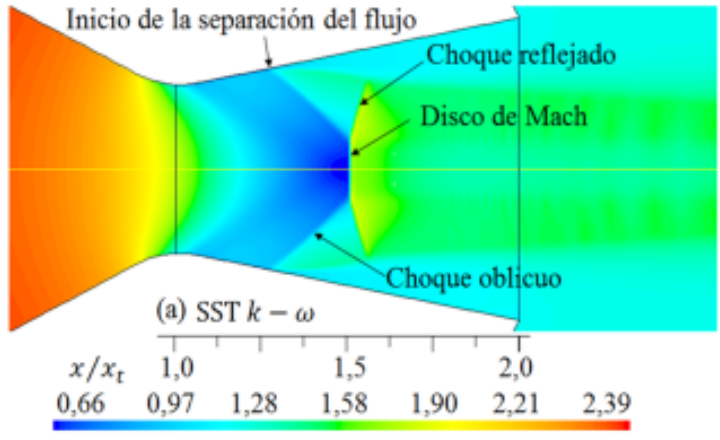


Figura 4.13: Simulación de ondas de choque

Un modelo de parámetros concentrados, simplifica el análisis de un sistema espacialmente distribuido, mediante la creación de una topología de elementos discretos que aproximan el comportamiento de los componentes distribuidos reales bajo ciertas restricciones.

Reduce las ecuaciones en derivadas parciales espaciales (PDEs) y temporales del continuo (dimensión infinita) de nuestro sistema a un conjunto de ecuaciones diferenciales ordinarias (ODEs) con un número finito de parámetros, del que podemos obtener una solución mucho más fácilmente.

Simusol trabaja con éste tipo de modelos.

Primero estudiaremos un caso específico: el de los circuitos eléctricos y luego mostraremos analogías para otros casos o "dominios".

Diccionario

Componente: Un dispositivo con dos o más terminales en el que puede fluir interiormente una carga o una corriente.

Conductor: Un hilo de resistencia cero que une los elementos para formar el circuito.

Rama: Secuencia de componentes.

Nodo: Punto de un circuito donde concurren más de dos conductores o componentes. Bajo ciertas condiciones puede tener un voltaje. Se aplica la ley de corrientes de Kirchhoff.

Malla, lazo o loop: Cualquier camino cerrado en un circuito eléctrico. Se aplica la ley de voltajes de Kirchhoff.

Cut-set, conjunto de corte: Conjunto de componentes o conectores que al sacarlos separan al circuito en dos. Se aplica la ley de corrientes de Kirchhoff.

Loops y cut-sets

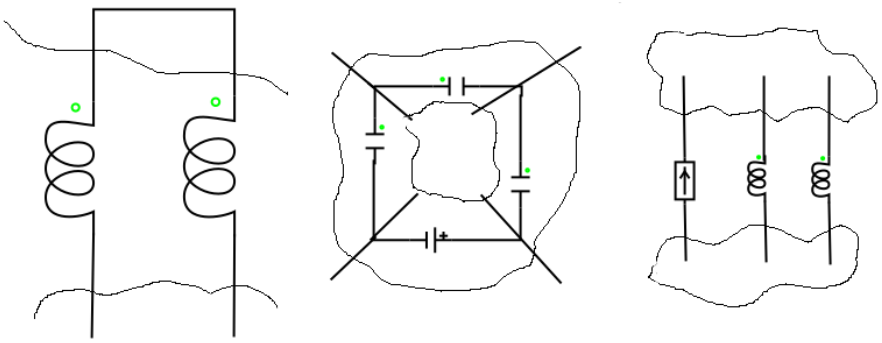


Figura 4.14: Loops y cut-sets

Componentes: variables y ecuaciones

- Se divide un sistema complejo y en algunos casos continuo en una red discreta de partes interconectadas.

- Los fenómenos físicos en esta red implican intercambio de energía entre sus componentes y cambios en la expresión de la misma en cada componente.

  Esta energía se manifiesta afuera de cada componente, en los nodos e hilos, en función de las fuerzas (voltajes) y los flujos (corrientes), variables éstas, denominadas de circuito. Adentro de cada componente en términos de las variables internas, algunas de la cuales determinan el estado. Todas las variables vinculadas por ecuaciones o de estado (reversibles: condensadores e inductores), o de transferencia (irreversibles: resistencias).

- En estos intercambios aparecen en cada componente, en general como ecuaciones diferenciales de segundo orden, que vinculan sus variables de estado y las de circuito. La ecuación diferencial representa lo que la física o disciplina que estudia la componente conoce sobre el comportamiento de la misma. Resolver esa ecuación a partir de los valores iniciales de las variables implica conocer el comportamiento futuro del componente, para lo cual debe contemplarse simultáneamente lo que pasa en todos los otros componentes.

- Se suelen obtener dos ecuaciones diferenciales de primer orden, a partir de la de segundo orden. Con lo cual se tiene un conjunto con dos ecuaciones por componente, más las ecuaciones de la red o circuito.

- Porque los sistemas físicos son de segundo orden: `https://en.m.wikipedia.org/wiki/Ostrogradsky_instability`

Circuitos eléctricos.

- El caso concreto de los sistemas eléctricos se trata con la teoría de circuitos.

- Se asume que los parámetros eléctricos del circuito (resistencia, capacitancia, inductancia) se encuentran confinados a una región pequeña del espacio, en los llamados componentes electrónicos (resistores, condensadores, inductancias), y que están conectados en un circuito mediante hilos perfectamente conductores.

- Con la teoría de los modelos con parámetros concentrados podemos usar una física "discreta", para pensar los sistemas.

- Comenzaremos el estudio de los circuitos desde su física, o sea la teoría de circuitos, más allá, de que luego usemos diversas metodologías para modelarlos:

  - de los circuitos, en éste caso la más cerca de la realidad física,
  - de los diagramas de bloques,
  - de los bond-graphs,
  - o cualquier otra.

Leyes de circuito de Kirchhoff

Todo sistema electromagnético obedece a las 4 ecuaciones de Maxwell, así cómo uno mecánico a las 3 leyes de Newton o uno térmico a las 3 leyes de la termodinámica.

Bajo ciertas condiciones las ecuaciones de Maxwell en un circuito (o modelo de parámetros concentrados) se reducen a:

corrientes: La suma de las corrientes I (flujos) que entran por un nodo debe ser igual a la suma de las corrientes que salen por ese nodo.

tensiones: La suma de las tensiones V (fuerzas) en un lazo debe ser 0.

En un circuito estas ecuaciones que tienen como incógnitas las V y las I son lineales y pueden representarse matricialmente.

Condiciones

- El tamaño del circuito debe ser mucho mayor que la longitud de onda de las señales eléctricas que circulen por el circuito.
- Las fuerzas electromotrices deben ser conservativas, para que exista en cada nodo un voltaje definido. Esto no ocurre si existe un campo magnético variable atravesando el circuito. Puede representarse estas situaciones no asignando voltajes a los nodos, y dando un valor definido a la suma de voltajes en un lazo.
- Los nodos no deben almacenar cargas, o éstas se deben poder representar por capacitores en cada nodo, conectados a tierra, o una red de capacidades.

Cuando se da alguna de estas condiciones, las leyes de Kirchhoff se cumplen. Se deben utilizar las ecuaciones de Maxwell completas para otros casos.

Tipos de componentes y sus "leyes"

E, Fuente de tensión: Transforma algún tipo de energía en energía eléctrica. y provee tensión definida. $V = E_F$. Su estado está dado por la energía acumulada, pero el estado no influye en el voltaje que da. Reversible.

I, Fuente de corriente: Transformara algún tipo de energía en energía eléctrica. y provee corriente definida. $I = I_F$. Su estado está dado por la energía acumulada, pero el estado no influye en la corriente que da. Reversible.

R, Resistor: Transfiere corriente de un nodo a otro en forma irreversible. Ley de Ohm. $V = RI$. Transforma también energía eléctrica en calor. No tiene estado.

C, Condensador: Acumula gorge. $V = Q/C, Q$: gorge, o sea: $\dot{V} = I/C$, dado que $\dot{Q} = I$. Estado: Q. Reversible cuando no hay desplazamiento de cargas instantáneos.

L, Inductor: Acumula flujo. $I = \Phi/L$, o sea: $\dot{I} = V/L$, dado que $\dot{\Phi} = V$; Estado: Φ. Reversible cuando no hay desplazamientos de flujos instantáneos.

M (mutua), Transformador: Vincula dos ramas. Son cuatro conectores. Existe una relación entre las tensiones de dos pares. Reversible, en general. No tiene estado.

Otros: .

- $$\dot{V} \equiv \frac{dV}{dt}$$
- **Leyes:** ecuaciones de estado o de transferencia.
- Los valores de E, I, R, C, L, M pueden ser constantes o variables o dependientes de diversas otras variables o incluso las relaciones entre las variables obedecer a ecuaciones más complejas. Se verá en cada caso, hasta qué punto el sistema pueda ser físicamente coherente y resoluble.
- **Gorge:** los capacitores tienen carga total 0, lo que tienen se ha propuesto que se llame "gorge".
- **Flujo magnético:** Cuidado con el nombre flujo magnético que es un momento, en términos de la nomenclatura de variables que usamos, no un flujo. En realidad el momento es la integral en el tiempo del voltaje, que para un inductor coincide con el flujo magnético.
- **Componentes variables:** Ante un cambio del componente (su forma por ejemplo), los condensadores y las inductancias conservan el gorge y el flujo respectivamente.

Relaciones entre variables

A partir del estado de cada componente, su constitución y cómo se conecta el circuito, las fuerzas y los flujos se interrelacionan y determinan los valores futuros del circuito.

- En algunos casos, en particular si se cierran o abren llaves, los condensadores y las inductancias ajustarán inicialmente su carga y flujo magnético según el circuito. Lo harán en principio, en loops de condensadores o cut-sets de inductores. Aparecerán ΔQ y $\Delta\Phi$ en cada loop o cut-set, a éste efecto, para equilibrar las variables de circuito.
- Luego del ajuste previo, en un condensador, dado su estado por su carga, el voltaje está determinado. La corriente dependerá del circuito y, una vez dada, el futuro voltaje estará determinado en el próximo "instante".
- Luego del ajuste previo, en una inductancia, la corriente está dada por su flujo magnético, el voltaje dependerá del circuito y, dado este, quedará determinado el cambio de corriente.
- En una resistencia, el voltaje y la corriente están interrelacionados directamente. No "influencian" el futuro. El circuito y las corrientes y voltajes de las fuentes, condensadores e inductores, las determinan instante a instante.

Nomenclatura temporal en cada paso

Se habla de un estado:

- previo $_0$: antes de que se interconectan componentes.

- inicial $_i$: luego de interconectar componentes, cuando se acomodaron las variables de estado, y se fijan las fuerzas y flujos.

- próximo $_p$: luego de un intervalo, donde las variables fuerza y flujo cambiaron diferencialmente el estado.

Nombres generalizados

Relaciones entre fuerzas V, flujos I, momentos Φ y desplaz. Q **generalizados**.

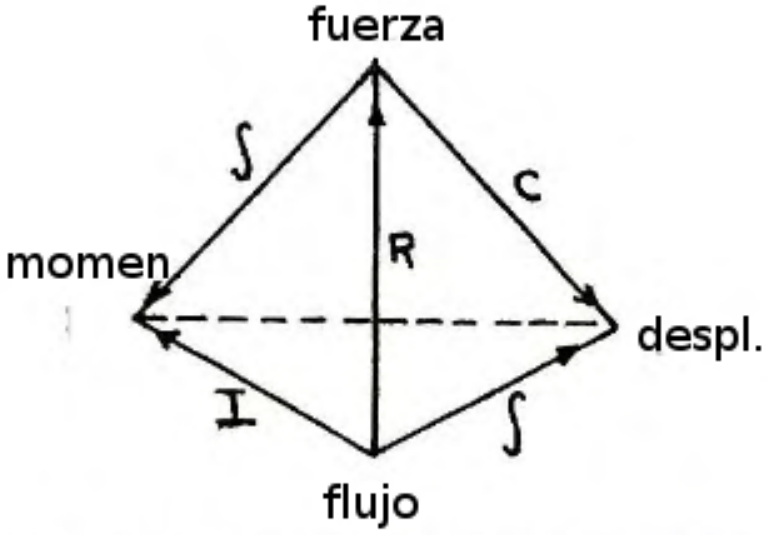


Figura 4.15: Tetraedro o pirámide de Paynter. (Zgorzelski, Maciej. Cameron, Timothy M. 1998).

Las variables mantienen relaciones:

- por un lado integro-diferenciales (de cambio) entre sí ($\Phi_p = \int V dt + \Phi_i$, $Q_p = \int I dt + I_i$);

- por otro ecuaciones de estado (al estar el estado "interno" determinado, determinan otra variable "externa");

- y por otro relaciones irreversibles del tipo "Onsager" (en las resistencias no hay estado pero V e I están interrelacionadas).

El trabajo viene dado por el producto de la fuerza por el desplazamiento, o el momento (impulso) por el flujo. Se obtiene potencia con el producto de la fuerza por el flujo.

Más allá de que éstos nombres correspondan a su uso por analogía en otros sistemas, conviene irlos presentando.

	Q	I	V	Φ	energía	trabajo	R/
	ext. estado desplaz.	ext. circuito flujo	int. circuito fuerza	int. estado momento	estado	cambio	
inductor		⋆ E: Φ/L	C: $L\dot{I} = \dot{\Phi}$	⋆ $\Phi_p = \int V dt + \Phi_i$	$\frac{1}{2}\frac{\Phi^2}{L}$	ΦdI	R
condensador	⋆ $Q_p = \int I dt + Q_i$	C: $C\dot{V} = \dot{Q}$	⋆ E: Q/C		$\frac{1}{2}\frac{Q^2}{C}$	VdQ	R
resistor		⋆ T: V/R	⋆ T: RI		0	$VIdt$	I

Cuadro 4.4: Relaciones de cuatro, E: ecuación de estado C: ecuación de cambio, T: ecuación de transporte, R: reversible, I: irreversible. ⋆ pares antes denominados fuerza-flujo.

Tabla con las relaciones

Ejemplo RLC

Veamos un circuito serie: E R L C.

- El voltaje o caída de tensión en:
  - la resistencia es RI
  - la bobina es $L\dot{I}$, ($\dot{I} = \frac{dI}{dt}$)
  - el condensador es Q/C (Q= gorge, $I = \frac{dQ}{dt}$)
  - en la fuente $-E$
- La suma de voltajes en el lazo es cero.

  $RI + L\dot{I} + Q/C - E = 0$, con $\dot{Q} = I$ (dos ecuaciones diferenciales de primer orden)

  $R\dot{I} + L\dot{\dot{I}} + I/C = 0$ (derivando, una de segundo)
- Entonces, $I = 0$ es al menos una solución.
- Las variables: $I_L = I$ y V_C son consecuencia del estado, y dadas por Φ_0 y Q_0 luego del primer ajuste: $I_L = \Phi/L$ y $V_C = Q/C$
- podemos calcular (inicialmente imponemos $V_C = E$)

$$\dot{I} = \frac{E - V_C - RI_L}{L}$$

- y con ello los próximos valores, a partir de los actuales:

  $I_p = I + \int \dot{I} dt$, con I_p, calculamos Φ_p

  y $Q_p = Q + \int I dt$

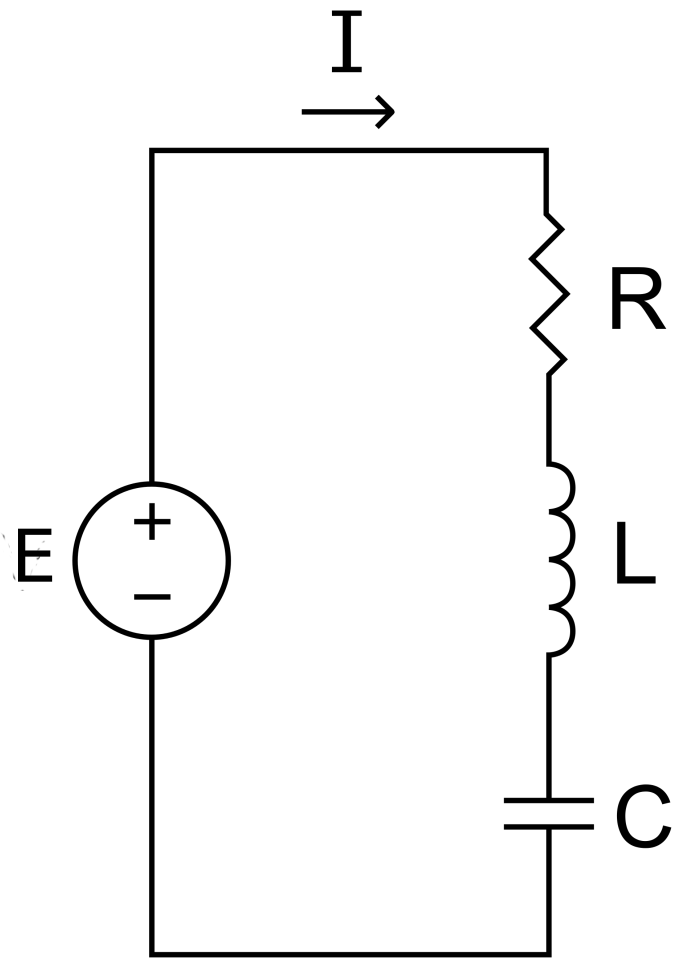


Figura 4.16: RLC

Así, podemos ir calculando paso a paso el próximo valor.

Ejercicio: resolver en planilla de cálculo para diferentes valores iniciales. Planilla RLC.ods

Podemos elegir la corriente del inductor y la tensión del condensador iniciales que queramos.

Con ello calcular las derivadas de ambas.

Y con ello la nueva corriente y la nueva tensión, y con ello el próximo estado. Luego se repite para los sucesivos tiempos.

Numéricamente podemos usando un integrador tipo "Euler" decir que las variables de estado en el próximo momento serán:

$$I_p = \dot{I}dt + I_i$$

$$Q_p = \dot{Q}dt + Q \qquad Q_p = Idt + Q_i$$

Si queremos usar el voltaje en vez de la carga como variable, queda:

	A	B	C	D	E	F	G
1	L	C	R	E	DT		T
2	1	1	1	1	0,5		
3	IL	VC	VR	E	VL	DIL	
4	1	1	=A4*C$2	=+D$2	=+D4-C4-B4	=+E4/A$2	0
5	=A4+F4*E$2	=+B4+A4/B$2*E$2	=A5*C$2	=+D$2	=+D5-C5-B5	=+E5/A$2	=+G4+E$2
6	=A5+F5*E$2	=+B5+A5/B$2*E$2	=A6*C$2	=+D$2	=+D6-C6-B6	=+E6/A$2	=+G5+E$2
7	=A6+F6*E$2	=+B6+A6/B$2*E$2	=A7*C$2	=+D$2	=+D7-C7-B7	=+E7/A$2	=+G6+E$2
8	=A7+F7*E$2	=+B7+A7/B$2*E$2	=A8*C$2	=+D$2	=+D8-C8-B8	=+E8/A$2	=+G7+E$2
9	=A8+F8*E$2	=+B8+A8/B$2*E$2	=A9*C$2	=+D$2	=+D9-C9-B9	=+E9/A$2	=+G8+E$2

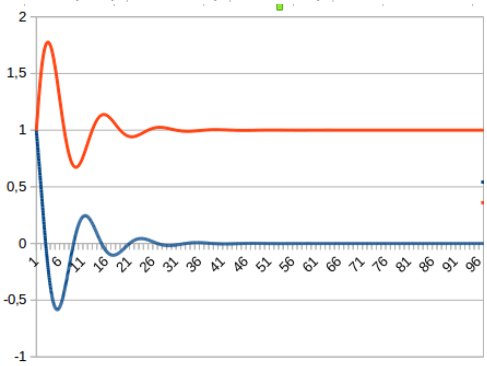


Figura 4.17: Fórmulas y Gráfico. IL es azul (baja inicialmente), VC es rojo (sube inicialmente)

$$I_p = \dot{I}dt + I_i$$

$$CV_p = Idt + CV \qquad V_p = I/Cdt + V_i$$

$$V_p = \dot{V}dt + V_i$$

Ejemplo: condiciones previas

Analizaremos un caso donde se ajustan las condiciones previas. Se supone que hay una llave que cierra y conecta el loop y a partir de allí se inician los cálculos.

Debe ser: $V_{C10} = V_{C20}$, $Q_{C10}/C_1 = Q_{C20}/C_2$

Si los Q no cumplen $Q_{C10}/C_1 = Q_{C20}/C_2$, habrá un desplazamiento de cargas instantáneo y no reversible (la energía eléctrica no se conserva), donde la carga de un condensador carga al otro (la carga o gorge si se conserva).

$Q_{C1} = Q_{C10} + \Delta Q$, y $Q_{C2} = Q_{C20} - \Delta Q$ y $Q_{C1}/C_1 = Q_{C2}/C_2$, donde tenemos tres ecuaciones y tres incógnitas: Q_{C1}, Q_{C2} y ΔQ.

Una vez igualados los voltajes podemos continuar el proceso anterior. Las ecuaciones que usamos garantizan que hasta que no se abra o cierre otra llave, las tensiones en los condensadores cumplirán la condición pedida.

Analogías, software

- Muchos sistemas físicos, en general los que son modelables con parámetros concentrados, encuentran analogías con los circuitos. Por ello, estudiarlos es útil no sólo

	A	B	C	D	E	F	G
1	L	C	R	E	DT		T
2	1	1	1	1	0,5		
3	IL	VC	VR	E	VL	DIL	
4	1	1	1	1	-1	-1	0
5	,50	1,50	,50	1,00	-1,00	-1,00	,50
6	,00	1,75	,00	1,00	-,75	-,75	1,00
7	-,38	1,75	-,38	1,00	-,38	-,38	1,50
8	-,56	1,56	-,56	1,00	,00	,00	2,00
9	-,56	1,28	-,56	1,00	,28	,28	2,50
10	-,42	1,00	-,42	1,00	,42	,42	3,00
11	-,21	,79	-,21	1,00	,42	,42	3,50
12	,00	,68	,00	1,00	,32	,32	4,00
13	,16	,68	,16	1,00	,16	,16	4,50
14	,24	,76	,24	1,00	,00	,00	5,00
15	,24	,88	,24	1,00	-,12	-,12	5,50
16	,18	1,00	,18	1,00	-,18	-,18	6,00
17	,09	1,09	,09	1,00	-,18	-,18	6,50
18	,00	1,13	,00	1,00	-,13	-,13	7,00
19	-,07	1,13	-,07	1,00	-,07	-,07	7,50
20	-,10	1,10	-,10	1,00	,00	,00	8,00
21	-,10	1,05	-,10	1,00	,05	,05	8,50
22	-,08	1,00	-,08	1,00	,08	,08	9,00
23	-,04	,96	-,04	1,00	,08	,08	9,50
24	,00	,94	,00	1,00	,06	,06	10,00
25	,03	,94	,03	1,00	,03	,03	10,50
26	,04	,96	,04	1,00	,00	,00	11,00
27	,04	,98	,04	1,00	-,02	-,02	11,50
28	,03	1,00	,03	1,00	-,03	-,03	12,00

Figura 4.18: Resultados

para resolver circuitos.

Se ha encontrado que con componentes generalizados de los mismos tipos que se encuentran en los circuitos se puede representar una amplia gama de sistemas.

- Programas como Sceptre, Spice o Siru resuelven circuitos eléctricos; y mediante analogías, otros sistemas.

Dominios

- Para cada "dominio" entre los:
  - mecánicos,
  - eléctricos,
  - térmicos,
  - etc.

  las variables adoptan nombres específicos de cada dominio, como voltaje y corriente en el eléctrico. En general se denominan fuerzas, flujos, desplazamientos y momentos.

- El nombre "dominio energético" se origina en la metodología de los diagramas de enlace.

- En Simusol cada circuito corresponde a un dominio. Puede haber muchos circuitos en un diagrama (dia), incluso de diferentes dominios. Pueden conectarse mediante "transformadores", circuitos de diferentes dominios. También se relacionan entre sí usando variables comunes.

Nombres generalizados: F F D M

Fuerzas, Flujos, Desplazamientos y Momentos.

- En el capítulo anterior designamos como fuerzas y flujos en general a pares de variables intensivas y extensivas respectivamente, cuyo producto daba el trabajo.
- Ahora veremos que en general esos pares también vienen de a pares, correspondientes cada par a una de las dos ecuaciones diferenciales de primer orden en que se separa la ecuación de segundo orden que determina casi toda rama de sistemas físicos a parámetros concentrados.
- Entonces, se reajusta la nomenclatura y denominamos a un par, el de la derivada primera, como desplazamiento y fuerza, y al de la segunda como flujo y momento (o impulso).
- Las leyes de Kirchhoff en su versión generalizada dicen que la suma de "desplazamientos" y la de "flujos" en cada nodo es cero, y la suma de "fuerzas" y la de "momentos" es cero en cada loop.

Ejemplo mecánico, analogía:

$$F_g = mg$$

$$F_r = -kx$$

Variables: v y x, podría ser: p (momento) y x.

$$v = \dot{x}$$

la suma de fuerzas es $m\dot{v}$

$$mg - kx - m\dot{v} = 0$$

entonces:

$$\dot{v} = \frac{k}{m}x - g$$
$$\dot{x} = v$$

Analogía

R y b, si hubiera amortiguador.

En un caso el circuito es "serie" en el otro "paralelo".

Verificar que las ecuaciones quedan idénticas.

Existen otras analogías posibles.

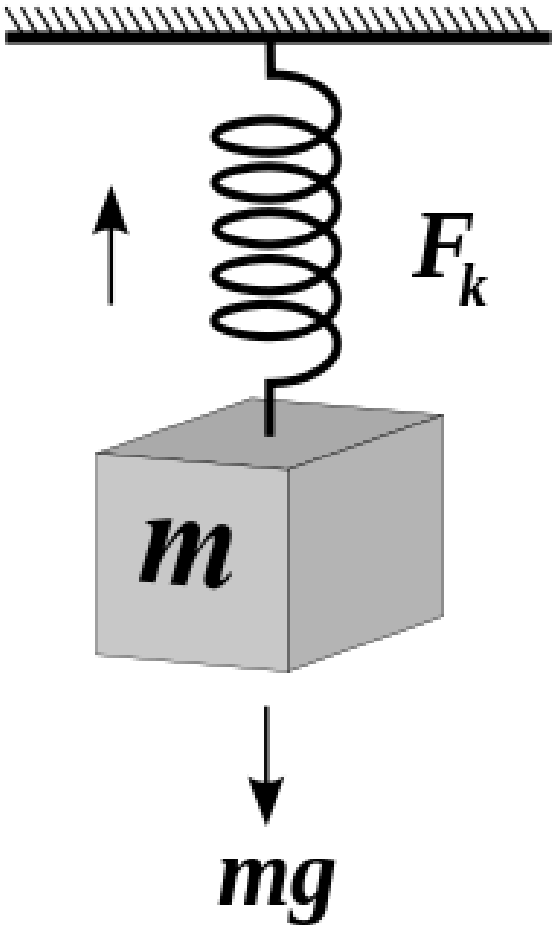


Figura 4.19: Masa y resorte

Ejemplo térmico, analogía

En el caso térmico la analogía también es clara, aunque no hay nada que haga de masa o de inductancia.

T_a: temperatura del aire, $q = \dot{Q}$: flujo de calor, Q: calor acumulado, C_p: capacidad calorífica.

$$T_a - qR - Q/C_p = 0$$

completamos:

Verificar que las ecuaciones quedan idénticas. Existen otros esquemas térmicos posibles, por un lado es posible sustituir el "Calor" por la energía interna que es una función de estado termodinámica. También es posible incorporar la entropía.

I	v
Q	x
V	F
L	m
C	$-1/k$

Cuadro 4.5: Analogía eléctrica–mecánica.

I	v	q
Q	x	Q
V	F	T
L	m	
C	$-1/k$	C_p
R	b	R

Cuadro 4.6: Analogía eléctrica–mecánica–térmica.

Metodologías de trabajo

Cada metodología tiene sus formas de representación como ecuaciones, gráficos, etc.

Las representaciones gráficas de modelos en cada metodología son un formalismo y medio de comunicación entre humanos y entre humanos y software.

4.22.5. Matemáticas

Tipo de datos

En el contexto del cálculo científico, los datos se organizan en estructuras de complejidad creciente.

Constantes y variables

Una **constante** es un valor fijo que no cambia durante la ejecución del programa: π, e, la velocidad de la luz, la constante gravitacional.

Una **variable** es un nombre que representa un valor que puede cambiar. En cálculo científico las variables tienen siempre tipo numérico real o complejo.

Escalares

Un **escalar** es un único valor numérico. Representa una magnitud que queda completamente caracterizada por un número: la temperatura de un punto, la masa de un objeto, el tiempo transcurrido.

Vectores

Un **vector** es una secuencia ordenada de n escalares llamados *componentes* o *elementos*:

$$\mathbf{v} = (v_1, v_2, \ldots, v_n)$$

Los vectores pueden ser de fila o de columna. Representan magnitudes que tienen tanto

módulo como dirección (velocidad, fuerza), o simplemente colecciones ordenadas de datos (una serie temporal, los valores de una función en puntos discretos).

Las operaciones fundamentales entre vectores son:

Suma: $\mathbf{u}+\mathbf{v} = (u_1+v_1,\, u_2+v_2,\, \ldots,\, u_n+v_n)$. Solo está definida entre vectores del mismo tamaño.

Multiplicación por escalar: $k\mathbf{v} = (kv_1,\, kv_2,\, \ldots,\, kv_n)$.

Producto escalar o punto: $\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{n} u_i v_i$. Produce un escalar. Geométricamente, $\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta$, donde θ es el ángulo entre los vectores.

Norma o módulo: $\|\mathbf{v}\| = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{\sum_{i=1}^{n} v_i^2}$.

Producto vectorial: solo definido en $\mathbb{R}^3$. Produce un vector perpendicular a ambos operandos con módulo $\|\mathbf{u}\|\|\mathbf{v}\| \sin\theta$.

Matrices

Una **matriz** de m filas y n columnas es un arreglo rectangular de $m \times n$ escalares:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

El elemento a_{ij} está en la fila i y la columna j. Un vector es un caso particular de matriz (de una fila o una columna). Un escalar es una matriz 1×1.

Las operaciones fundamentales entre matrices son:

Suma: $C = A+B$ con $c_{ij} = a_{ij}+b_{ij}$. Solo está definida entre matrices del mismo tamaño.

Multiplicación por escalar: kA con $(kA)_{ij} = k\, a_{ij}$.

Producto matricial: $C = AB$ con $c_{ij} = \sum_{k=1}^{p} a_{ik}\, b_{kj}$. Requiere que el número de columnas de A sea igual al número de filas de B. Si A es $m \times p$ y B es $p \times n$, entonces C es $m \times n$. El producto matricial **no es conmutativo**: en general $AB \neq BA$.

Transposición: A^T con $(A^T)_{ij} = a_{ji}$. Intercambia filas y columnas.

Multiplicación elemento a elemento: también llamada producto de Hadamard, produce una matriz del mismo tamaño con $c_{ij} = a_{ij}\, b_{ij}$. Se distingue del producto matricial y es muy útil en cálculo numérico.

Raíces de polinomios

Un **polinomio** de grado n es una función de la forma:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

con $a_n \neq 0$. Los coeficientes a_i pueden ser reales o complejos.

Una **raíz** o **cero** del polinomio es un valor r tal que $p(r) = 0$. El teorema fundamental del álgebra garantiza que todo polinomio de grado n con coeficientes complejos tiene exactamente n raíces complejas (contadas con multiplicidad).

Para polinomios de grado 1 y 2 existen fórmulas explícitas. Para grado 3 y 4 también existen pero son complejas. Para grado ≥ 5 no existe fórmula algebraica general (teorema de Abel-Ruffini), por lo que se recurre a métodos numéricos.

Los algoritmos numéricos para encontrar raíces de polinomios trabajan con el vector de coeficientes del polinomio ordenados de mayor a menor grado. Un polinomio puede factorizarse a partir de sus raíces:

$$p(x) = a_n(x - r_1)(x - r_2)\cdots(x - r_n)$$

Conocidas las raíces, el polinomio queda completamente determinado.

Álgebra lineal

El álgebra lineal es la rama de la matemática que estudia los espacios vectoriales y las transformaciones lineales. Es la herramienta matemática más utilizada en ciencias e ingeniería.

Sistemas de ecuaciones lineales

Un sistema de m ecuaciones lineales con n incógnitas puede escribirse en forma matricial como:

$$A\mathbf{x} = \mathbf{b}$$

donde A es la *matriz de coeficientes* $(m \times n)$, $\mathbf{x}$ es el vector de incógnitas y $\mathbf{b}$ es el vector de términos independientes.

El sistema puede tener:

- Solución única: si A es cuadrada y no singular.
- Infinitas soluciones: si el sistema es compatible e indeterminado.
- Sin solución: si el sistema es incompatible.

Para un sistema cuadrado con A no singular, la solución es $\mathbf{x} = A^{-1}\mathbf{b}$. En la práctica no se calcula la inversa explícitamente, pues es costoso y propenso a errores numéricos. En cambio se usan métodos de factorización como la **eliminación de Gauss** o la factorización **LU**.

Determinante

El **determinante** de una matriz cuadrada A, denotado $\det(A)$ o $|A|$, es un escalar que contiene información sobre la transformación lineal que representa A:

- $\det(A) \neq 0$ si y solo si A es invertible (no singular).
- $|\det(A)|$ es el factor de escala del volumen bajo la transformación.

- $\det(A) < 0$ indica que la transformación invierte la orientación.

Valores y vectores propios

Un **valor propio** λ y su correspondiente **vector propio v** de una matriz A satisfacen:

$$A\mathbf{v} = \lambda\mathbf{v}$$

Los vectores propios son las direcciones que la transformación lineal no rota, solo escala. Los valores propios son los factores de escala.

La ecuación característica $\det(A - \lambda I) = 0$ produce un polinomio de grado n cuyas raíces son los valores propios.

Los valores propios tienen aplicaciones fundamentales: estabilidad de sistemas dinámicos, modos normales de vibración, análisis de componentes principales, compresión de datos, motores de búsqueda.

Norma de una matriz y número de condición

La **norma** de una matriz generaliza el concepto de módulo de un vector. El **número de condición** $\kappa(A)$ mide cuán mal condicionado está un sistema lineal: si $\kappa(A)$ es grande, pequeños errores en **b** producen grandes errores en la solución **x**. Los sistemas con $\kappa(A) \gg 1$ se llaman *mal condicionados* y requieren cuidado especial en el tratamiento numérico.

Derivación numérica

La derivada de una función f en un punto x se define como:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Cuando f no se conoce analíticamente sino a través de valores discretos, o cuando la derivación analítica es difícil, se utilizan aproximaciones numéricas basadas en diferencias finitas.

Las tres aproximaciones fundamentales son:

Diferencia hacia adelante: $f'(x) \approx \dfrac{f(x+h) - f(x)}{h}$

Diferencia hacia atrás: $f'(x) \approx \dfrac{f(x) - f(x-h)}{h}$

Diferencia central: $f'(x) \approx \dfrac{f(x+h) - f(x-h)}{2h}$

La diferencia central tiene error de orden $O(h^2)$ mientras que las diferencias hacia adelante y hacia atrás tienen error de orden $O(h)$, lo que hace a la diferencia central más precisa para el mismo paso h.

La elección del paso h presenta un compromiso: un h muy grande introduce error de truncamiento (la aproximación no es exacta); un h muy pequeño introduce error de redondeo

(las operaciones en punto flotante pierden cifras significativas al restar números muy próximos). El valor óptimo de h es del orden de $\sqrt{\epsilon_{\text{máquina}}}$ para diferencias de primer orden y $\epsilon_{\text{máquina}}^{1/3}$ para diferencias centrales.

Jacobiano y hessiano

Cuando una función depende de varias variables, la derivada deja de ser un único número. Si una función vectorial $\mathbf{F}$ depende de un vector de variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, sus derivadas parciales se organizan en una matriz llamada **jacobiano**:

$$J_{ij} = \frac{\partial F_i}{\partial x_j}.$$

El jacobiano describe cómo cambia cada componente de la salida cuando cambia cada variable de entrada.

Si f es una función escalar de varias variables, las segundas derivadas parciales se organizan en la matriz **hessiana**:

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \, \partial x_j}.$$

El hessiano describe la curvatura local de la función. En cálculo numérico, jacobianos y hessianos aparecen en métodos de optimización, resolución de sistemas no lineales y análisis de estabilidad.

Integración numérica

La integral definida $\displaystyle\int_a^b f(x)\,dx$ representa el área con signo bajo la curva f entre a y b. Cuando f no tiene primitiva elemental o se conoce solo en puntos discretos, se usan métodos de integración numérica (*cuadratura numérica*).

Todos los métodos de cuadratura dividen el intervalo $[a, b]$ en n subintervalos de ancho $h = (b - a)/n$ y aproximan la integral por una suma ponderada de valores de f.

Al calcular numéricamente una integral se suele controlar el error de la aproximación. El **error absoluto** mide la diferencia directa entre el valor calculado y el valor exacto:

$$e_a = |I_{\text{exacto}} - I_{\text{aprox}}|.$$

El **error relativo** compara ese error con el tamaño del valor exacto:

$$e_r = \frac{|I_{\text{exacto}} - I_{\text{aprox}}|}{|I_{\text{exacto}}|}.$$

Como en general el valor exacto no se conoce, los programas numéricos trabajan con estimaciones del error y con tolerancias pedidas por el usuario.

Regla del rectángulo

Aproxima el área de cada subintervalo por un rectángulo:

$$\int_a^b f(x)\,dx \approx h \sum_{i=0}^{n-1} f(x_i)$$

Error de orden $O(h)$.

Regla del trapecio

Aproxima el área de cada subintervalo por un trapecio uniendo $f(x_i)$ con $f(x_{i+1})$:

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right]$$

Error de orden $O(h^2)$.

Regla de Simpson

Aproxima f por una parábola en cada par de subintervalos (requiere n par):

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + f(x_n)\right]$$

Error de orden $O(h^4)$, significativamente más precisa que el trapecio con el mismo número de puntos.

Cuadratura de Gauss

En lugar de nodos igualmente espaciados, usa puntos y pesos óptimamente elegidos para maximizar el orden del polinomio que se integra exactamente. Con n puntos, integra exactamente polinomios de grado hasta $2n - 1$. Es el método más eficiente cuando f es suave.

Splines

Un **spline** es una función construida por tramos, normalmente usando polinomios, que pasa por un conjunto de puntos dados o los aproxima. La idea es reemplazar una curva desconocida por varias curvas simples unidas suavemente.

Los splines se usan para interpolar datos discretos, suavizar curvas y calcular derivadas o integrales aproximadas cuando la función sólo se conoce en una cantidad finita de puntos.

Triangulación

En dos dimensiones, una región puede dividirse en triángulos. Esa división se llama **triangulación**. Permite aproximar una integral doble sobre una región complicada sumando contribuciones calculadas sobre triángulos simples.

En cálculo numérico, la triangulación aparece cuando la región de integración no es un rectángulo o cuando los datos están dados sobre puntos dispersos del plano.

Ecuaciones diferenciales ordinarias

Una **ecuación diferencial ordinaria** (EDO) es una ecuación que relaciona una función desconocida $y(t)$ con sus derivadas. La forma general de una EDO de primer orden es:

$$\frac{dy}{dt} = f(t, y)$$

con una **condición inicial** $y(t_0) = y_0$.

Este tipo de problema se denomina **problema de valor inicial** (PVI). La solución es una función $y(t)$ que satisface la ecuación y la condición inicial.

Una EDO de orden n puede reducirse a un sistema de n EDOs de primer orden introduciendo variables auxiliares. Por ejemplo, la ecuación del oscilador amortiguado:

$$m\ddot{x} + b\dot{x} + kx = F(t)$$

se convierte en el sistema:

$$\dot{x}_1 = x_2, \qquad \dot{x}_2 = \frac{1}{m}\left[F(t) - bx_2 - kx_1\right]$$

donde $x_1 = x$ y $x_2 = \dot{x}$. En general, para simular un sistema físico se formulan sus ecuaciones como un sistema de EDOs de primer orden, que es el formato estándar que aceptan los métodos numéricos.

Métodos numéricos para EDOs

Los métodos numéricos para EDOs producen aproximaciones de la solución en una secuencia de puntos $t_0, t_1, t_2, \ldots$ con paso $h = t_{i+1} - t_i$. El valor calculado en t_i se denota $y_i \approx y(t_i)$.

Método de Euler

Es el método más simple. A partir de y_i, la pendiente en ese punto es $f(t_i, y_i)$, y se avanza linealmente:

$$y_{i+1} = y_i + h\, f(t_i,\, y_i)$$

El error local (en un paso) es de orden $O(h^2)$; el error global (acumulado en todo el intervalo) es de orden $O(h)$. Es *explícito*: el valor nuevo y_{i+1} se calcula directamente sin resolver una ecuación.

El método de Euler es fácil de entender e implementar pero tiene precisión limitada y puede ser inestable para pasos grandes. Se usa principalmente como introducción pedagógica.

Métodos de Runge-Kutta

La familia de métodos de Runge-Kutta mejora la precisión evaluando f en varios puntos dentro del intervalo $[t_i, t_{i+1}]$ y tomando una combinación ponderada de esas evaluaciones.

El método de **Runge-Kutta de orden 4** (RK4) es el más utilizado en la práctica:

$$\begin{aligned}
k_1 &= h\, f(t_i,\; y_i) \\
k_2 &= h\, f\big(t_i + \tfrac{h}{2},\; y_i + \tfrac{k_1}{2}\big) \\
k_3 &= h\, f\big(t_i + \tfrac{h}{2},\; y_i + \tfrac{k_2}{2}\big) \\
k_4 &= h\, f(t_i + h,\; y_i + k_3)
\end{aligned}$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

El error global es de orden $O(h^4)$: al reducir el paso a la mitad, el error se divide por 16. RK4 es preciso, estable y relativamente simple de implementar.

Métodos adaptativos

Los métodos de paso fijo aplican el mismo h en todo el intervalo. Los **métodos adaptativos** ajustan el paso automáticamente: usan un paso pequeño donde la solución varía rápidamente y un paso grande donde varía lentamente, manteniendo el error por debajo de una tolerancia especificada.

Los métodos adaptativos más usados son Runge-Kutta-Fehlberg (RKF45), Dormand-Prince y los métodos de predictor-corrector de Adams. Estos son los que usan las funciones de solución de EDOs de los entornos de cálculo científico cuando se les especifica una tolerancia.

Métodos implícitos

Los métodos descritos hasta ahora son *explícitos*: y_{i+1} se calcula directamente. Los métodos *implícitos* expresan y_{i+1} en función de sí mismo, lo que requiere resolver una ecuación en cada paso.

Los métodos implícitos son más costosos por paso pero permiten usar pasos mucho más grandes en sistemas *rígidos* (*stiff*): aquellos con componentes que varían a velocidades muy diferentes. Un sistema mecánico con partes muy rígidas, o un sistema químico con reacciones muy rápidas y muy lentas, son ejemplos típicos de sistemas rígidos.

El ciclo modelo-EDO-método-programa

La simulación de un proceso físico sigue siempre el mismo ciclo:

1. **Modelo:** se identifican las variables relevantes y se aplican las leyes físicas para obtener las ecuaciones que gobiernan el sistema. Se establecen las hipótesis simplificadoras y el dominio de validez del modelo.

2. **EDO:** el modelo matemático se formula como un sistema de ecuaciones diferenciales ordinarias de primer orden:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}), \qquad \mathbf{y}(t_0) = \mathbf{y}_0$$

   donde $\mathbf{y}$ es el vector de variables de estado y $\mathbf{f}$ codifica las leyes físicas.

3. **Método numérico:** se elige el método apropiado según las características del problema (orden de precisión requerido, rigidez del sistema, restricciones de tiempo de cómputo). Se fijan el paso de tiempo h o la tolerancia de error, y las condiciones iniciales.

4. **Programa:** se implementa el método numérico, se integra el sistema de EDOs en el intervalo de tiempo de interés y se procesan y visualizan los resultados. Se verifica la consistencia física de la solución y, si es posible, se valida comparando con soluciones analíticas conocidas o con datos experimentales.

Este ciclo rara vez es lineal: los resultados de la simulación frecuentemente revelan la necesidad de refinar el modelo, ajustar el método o revisar los parámetros, iniciando una nueva iteración.

Validación y verificación

La **verificación** responde a la pregunta: ¿el programa resuelve correctamente las ecuaciones del modelo? Se verifica comparando con soluciones analíticas conocidas (cuando existen), comprobando la consistencia de las unidades, y estudiando la convergencia: al reducir el paso h, el error debe reducirse según el orden teórico del método.

La **validación** responde a: ¿el modelo representa correctamente el sistema físico? Se valida comparando las predicciones del modelo con datos experimentales independientes. Un modelo puede estar perfectamente verificado (el programa es correcto) pero no validado (el modelo no representa bien la realidad).

El **análisis de sensibilidad** estudia cómo cambia la solución ante variaciones en los parámetros del modelo. Permite identificar qué parámetros son críticos (pequeñas variaciones producen grandes cambios en la solución) y cuáles son irrelevantes, orientando los esfuerzos experimentales hacia la medición precisa de los parámetros críticos.

4.23. Ejercicios

4.23.1. Criterio general para la resolución.

Los ejercicios de esta guía tienen como objetivo mostrar como usar una calculadora científica con el lenguaje elegido. Las consignas se formulan en términos de problemas, sin depender de la sintaxis de un lenguaje particular. Las particularidades de cada lenguaje se indican en las notas de respuesta correspondientes.

Son 61 ejercicios con soliciones en cinco lenguajes. En cada cursada se debe identificar cuáles son obligatorios, optativos y avanzados, y aclarar si cada alumno debe resolverlos en uno o en todos los lenguajes. Y si pueden o no elegir cúal.

Los profesores deberán decidir si muestran los ejercicios con suluciones o si los muestran despues de cierto tiempo.

Para cursos donde el estudiante avanza por su cuenta puede ser conveniente la primera pasada hacerla con estudiantes viendo las soluciones. Más adelante dar problemas mas complejos sin soluciones. Probablemente una vez los estudiantes completen este y los siguientes tres pŕacticos donde tendrán una visión mas o menos completa para realizar sistemas de cálculo: funciones, estructura, entrada–salida + archivos y gráficos.

He incorporado ejercicios típicos de las materias de Algebra, Geometría, Análisis y Física básicas de los primeros años de carreras científicas y tecnológicas que los estudiantes de este curso puedan o haber hecho o deber realizar despues.

Encabezado de cuatro ejercicios

En algunos ejercicios que siguen (caracteres, fecha, tipos, valores y unidades) se trabaja con escalares y cadenas, que están en el núcleo de cada lenguaje y no en su biblioteca de vectores. Por eso los comandos son los mismos en todas las calculadoras de un mismo lenguaje, y en cada uno se da un solo listado:

- **Scilab**: igual en la ventana gráfica y en `scilab-cli`.

- **Octave**: igual en la interfaz gráfica y en `octave-cli`; para estos cuatro ejercicios no necesita paquetes adicionales.
- **Python**: igual en `python3`, IPython y Jupyter. Se supone `import numpy as np` hecho, porque algunos comandos usan NumPy para vectores de cadenas (`np.array`, `np.char`); todo lo demás es Python puro.
- **Perl**: igual en `perl -de1` y en `pdl2`. En `perl -de1` hay que cargar a mano los módulos que se nombran (`use POSIX`, `use Time::Piece`, `use Math::Trig`, `use Math::Complex`, y `use PDL` para las pocas líneas que usan `pdl`); en `pdl2` PDL ya está y los otros se cargan igual. En los listados el indicador es el de `perl -de1` (`p` imprime), y en `pdl2` funciona idéntico.
- **JavaScript**: igual en `node` y en la consola del navegador. Las líneas que usan `math.` necesitan math.js cargado (`require` en node, `<script>` en la página); el resto es JavaScript puro. La única diferencia de consola es que `new Date().toLocaleDateString()` usa el idioma del sistema en node y el del navegador en la página.

Recién en el ejercicio de vectores y matrices vuelven a diferenciarse las calculadoras de un mismo lenguaje, porque ahí hace falta la biblioteca de arreglos.

4.23.2. Ejercicios básicos

Instalación y ayuda; escalares y tipos; caracteres y fechas.

1. **Ejercicio:** Sistema operativo y calculadora

   Instale todo lo necesario para trabajar. Ejecute el script incluído en la teoría de prácticos anteriores a tal efecto, o siga las instrucciones de la tabla provista en la teoría de éste práctico.

   Arranque cada calculadora que vaya a usar, reconozca su indicador y salga.

   Según el sistema operativo en el que vaya a trabajar y el lenguaje que aprenda, serán las calculadoras (cada una es uno o varios programas a ejecutar).

   Cada lenguaje ofrece una o varias calculadoras. Scilab trae una, con dos envolturas (ventana gráfica o texto). Python trae varias: el intérprete `python3`, la consola mejorada IPython y el cuaderno Jupyter en el navegador; se recomienda IPython. Perl no trae ninguna propia y se usan dos sustitutos: el depurador `perl -de1` para escalares y texto, y la consola `pdl2` de PDL para vectores. JavaScript trae dos: `node` en la terminal y la consola del navegador (F12) dentro de una página.

   - **Scilab**: `scilab` (o el icono del menú) abre la ventana gráfica con la consola, indicador `->`; `scilab-cli` abre la misma consola en texto dentro de la terminal. Se sale con `exit`.
   - **Python**: son tres, se usara una distinta según la tarea que deba hacer.
     - `python3`, el intérprete solo, indicador `>>`, sale con `exit()`;

- `ipython3`, la consola mejorada, indicador `In [1]:`, con completado, ayuda con `?` y órdenes `%`, sale con `exit`;
- `jupyter notebook`, la misma calculadora en el navegador, con celdas. Se recomienda IPython. En las tres hay que escribir `import numpy as np` para tener la calculadora vectorial.

▪ **Perl**: no trae calculadora propia; se usan dos sustitutos.

- `perl -de1`, el depurador sobre un programa vacío, indicador `DB<1>`, para escalares y texto, donde `p` imprime y `q` sale; y
- `pdl2`, la consola de PDL, indicador `pdl>`, para vectores y matrices, sale con `quit`.

▪ **JavaScript**: son dos.

- `node` en la terminal, indicador `>`, sale con `.exit`; y
- la consola del navegador, abriendo Firefox o Chrome desde el menú y pulsando F12 (pestaña Console).

En las dos hay que cargar math.js para tener la calculadora vectorial: `require("mathjs")` en node, una etiqueta `<script>` en la página.

Node.js es el mismo intérprete de JavaScript que trae el navegador, empaquetado para correr en la terminal, sin navegador.

2. **Ejercicio:** Entorno interactivo básico

a) Ejecute la calculadora elegida y trabaje en su terminal.

b) Averigüe en qué directorio está trabajando.

c) Cree un directorio `practico4` y cámbiese a él.

d) Abra un registro de la sesión en un archivo de texto con su nombre (`guia4-NOMBREESTUDIANTE`) trabaje, y ciérrelo al terminar. Ese archivo se entrega con el práctico.

Scilab. Una sola calculadora, con dos envolturas: `scilab` (ventana gráfica, también desde el icono del menú) y `scilab-cli` (la misma consola en texto). En las dos, `diary` graba comandos y resultados.

```
pwd
mkdir("practico4"); chdir("practico4")
diary("guia4-NOMBREESTUDIANTE.txt")
// ... trabajo ...
diary(0)
exit
```

Octave. Una sola calculadora, disponible como `octave` (interfaz gráfica) y `octave-cli` (consola de texto). En ambas, `diary` registra comandos y resultados.

```
pwd
```

```
% Se usa un directorio temporal para no depender de un
   sistema de archivos editable.
try
    tmpdir = tempname;
    mkdir(tmpdir);
    cd(tmpdir);
catch err
    printf("No se pudo crear o cambiar al directorio
       temporal: %s\n", err.message);
end_try_catch
diary guia4-NOMBREESTUDIANTE.txt
% ... trabajo ...
diary off
```

Python. Tres calculadoras, segun lo que vaya a hacer, y cada una registra distinto. En IPython, `%logstart` graba los comandos (con `-o` también las salidas); `pwd`, `mkdir` y `cd` son órdenes de IPython, no de Python. El intérprete `python3` no tiene registro propio: se lo registra desde afuera con `script`, o se guarda sólo la historia con `readline`. En Jupyter el cuaderno mismo es el registro, con entradas y salidas; se guarda con Ctrl-S y se exporta a texto.

`script` es una orden de la terminal del sistema (viene en GNU/Linux y macOS, dentro de bsdutils o util-linux) que graba en un archivo todo lo que pasa en la terminal a partir de ese momento: lo que escribís, lo que los programas responden, los errores, todo, hasta que escribís `exit`. Es un grabador de la sesión de terminal, exactamente lo que `diary` hace dentro de Scilab, pero desde afuera y para cualquier programa. En Windows nativo (cmd, PowerShell) `script` no existe; el equivalente en PowerShell es `Start-Transcript guia4.txt` y `Stop-Transcript`; en WSL y Git Bash sí está `script`.

```
import os
import tempfile

# --- IPython (recomendada) ---
# pwd
# mkdir practico4
# cd practico4
# %logstart -o guia4-NOMBREESTUDIANTE.py
# ... trabajo ...
# %logstop
# exit

# --- python3 ---
print(os.getcwd())
# Se usa un directorio temporal para no depender de un
   sistema de archivos editable.
workdir = tempfile.mkdtemp(prefix="practico4_")
try:
    os.chdir(workdir)
    # ... trabajo ...
```

```
except OSError as e:
    print("No se pudo cambiar al directorio temporal:", e)
# El historial de readline se omite en ejecucion no
   interactiva.

# --- Jupyter ---
# ... trabajo en celdas; Ctrl-S guarda
   guia4-NOMBREESTUDIANTE.ipynb ...
```

Perl. Dos calculadoras, `perl -de1` y `pdl2`, y ninguna tiene registro de sesión dentro del intérprete: se registra desde afuera con `script`. ver nota en Python

```
#!/usr/bin/env perl
use strict;
use warnings;
use Cwd;

# Mostrar directorio de trabajo actual
print "directorio_actual = ", getcwd(), "\n";

# Crear y cambiar a un directorio de trabajo de ejemplo (en
   /tmp para el sandbox)
my $dir = "/tmp/practico4";
mkdir $dir unless -d $dir;
chdir $dir or warn "No se pudo cambiar a $dir: $!";
print "nuevo_directorio = ", getcwd(), "\n";

# En Perl interactivo se puede usar el modulo Term::ReadLine
   para historial.
# Aqui simulamos el guardado de comandos en un archivo de
   log.
open(my $fh, '>', "$dir/guia4-historia.txt") or warn $!;
if ($fh) {
    print $fh "# Comandos de ejemplo ejecutados en el
       entorno de trabajo\n";
    print $fh "print getcwd(), \"\\n\";\n";
    close($fh);
    print "historial guardado en $dir/guia4-historia.txt\n";
}
```

JavaScript. Dos calculadoras. `node` en la terminal: se registra desde afuera con `script`, como Perl; además `.save` guarda sólo los comandos escritos en la sesión (REPL es como Node llama a su consola interactiva; `.help`, `.save`, `.load`, `.exit` son sus órdenes propias). La consola del navegador no es una terminal, así que `script` no la alcanza; el registro se guarda desde la propia consola con el botón derecho: "Guardar todos los mensajes en archivo" en Firefox, "Save as..." en Chrome, que escribe entradas y salidas en un `.log`. En el navegador no hay directorio de trabajo que cambiar.

```
const fs = require("fs")
const os = require("os")
```

```
const path = require("path")

// --- node ---
console.log(process.cwd())
// Se usa un directorio temporal para no depender de un
   sistema de archivos editable.
const workdir = fs.mkdtempSync(path.join(os.tmpdir(),
   "practico4_"))
try {
    process.chdir(workdir)
} catch (err) {
    console.log("No se pudo cambiar al directorio
       temporal:", err.message)
}
// ... trabajo ...
// .save guia4-comandos.js              // solo los comandos
   (comando REPL)
// .exit                                // comando REPL

// --- consola del navegador ---
// abrir Firefox o Chrome desde el menu, F12, pestana Console
// ... trabajo ...
// boton derecho en la consola -> Guardar todos los mensajes
   en archivo (Firefox)
//                              -> Save as... (Chrome)
```

3. **Ejercicio:** Consulta de ayuda y documentación

   - (a) Consulte la ayuda de la tangente;
   - (b) busque funciones relacionadas con polinomios;
   - (c) busque ayuda sobre ordenar, evaluar polinomios y resolver EDO;
   - (d) anote la forma general y un ejemplo mínimo de cada una;
   - (e) explique por qué conviene consultar la ayuda ante una función nueva.

   La ayuda permite conocer la forma de llamada, los argumentos, los valores devueltos, las opciones y ejemplos de una función. Conviene consultarla porque funciones con propósitos semejantes pueden recibir argumentos en distinto orden o devolver resultados diferentes según el lenguaje.

   Scilab. `help` consulta una función y `apropos` busca por tema.

```
// ayuda
printf("help tan: muestra la documentacion de la funcion
   tangente.\n")
printf("apropos polynomial: busca funciones relacionadas con
   polinomios.\n")
printf("help gsort: muestra la documentacion de la funcion
   gsort.\n")
```

```
printf("help horner: muestra la documentacion de la funcion
   horner.\n")
printf("help ode: muestra la documentacion de la funcion
   ode.\n")

// ordenar: gsort(x)
x=[3 1 2];
gsort(x,"g","i")

// evaluar un polinomio: horner(p,x)
p=poly([-1 0 1],"x","coeff");
horner(p,2)

// resolver una EDO: ode(y0,t0,t,f)
function dy=f(t,y)
    dy=-y;
endfunction
t=0:0.1:1;
y=ode(1,0,t,f);
```

Octave. `help` consulta una función, `lookfor` busca por tema y `doc` abre el manual.

```
% ayuda
help tan
lookfor polynomial
help sort
help polyval
help ode45

% ordenar: sort(x)
x=[3 1 2];
sort(x)

% evaluar un polinomio: polyval(p,x)
p=[1 0 -1];
polyval(p,2)

% resolver una EDO: ode45(f,[t0 tf],y0)
f=@(t,y) -y;
[t,y]=ode45(f,[0 1],1);
```

Python. En IPython puede agregarse `?` al nombre; `help` funciona también en el intérprete común.

```
import numpy as np
from scipy.integrate import solve_ivp

# ayuda
```

```
help(np.tan)
print([x for x in dir(np) if 'poly' in x.lower()])  #
   equivalente a np.*poly*? en IPython
help(np.sort)
help(np.polyval)
help(solve_ivp)

# ordenar: np.sort(x)
x=np.array([3,1,2])
np.sort(x)

# evaluar un polinomio: np.polyval(p,x)
p=np.array([1,0,-1])
np.polyval(p,2)

# resolver una EDO: solve_ivp(f,[t0,tf],y0)
def f(t,y):
    return -y

sol=solve_ivp(f,[0,1],[1])
```

Perl. `perldoc` consulta la documentación de Perl y de sus módulos. En `pdl2` también pueden usarse `?` y `apropos`.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::Math;
use Math::ODE;

# polyval no esta exportado por esta version de PDL::Math;
# lo implementamos localmente (coeficientes de mayor a menor
   grado).
sub polyval_pdl {
    my ($c, $x) = @_;
    my $i = sequence($c->nelem);
    return sum($c * ($x ** $i));
}

# Ejemplos de consulta/ayuda reproducidos como script
   ejecutable.

# ordenar: qsort
my $x = pdl([3, 1, 2]);
my $x_ordenado = $x->qsort;
print "qsort($x) = $x_ordenado\n";

# evaluar un polinomio: polyval(coeficientes, x)
my $c = pdl([1, 0, -1]);    # x^2 - 1
my $valor = polyval_pdl($c, pdl(2));
```

```
print "polyval($c, 2) = $valor\n";

# resolver una EDO con Math::ODE
sub f {
    my ($t, $y) = @_;
    return [-$y->[0]];
}

my %ode_args;
$ode_args{step}    = 0.1;
$ode_args{initial} = [1];
$ode_args{ODE}     = [\&f];
$ode_args{t0}      = 0;
$ode_args{tf}      = 1;
$ode_args{verbose} = 0;
my $o = Math::ODE->new(%ode_args);
$o->evolve;
my @t_sorted = sort { $a <=> $b } keys %{$o->{values}};
my $y_final = $o->{values}{$t_sorted[-1]}[0];
print "EDO y'(t)=-y, y(0)=1 en t=1: $y_final\n";
```

JavaScript. Node tiene `.help` para su consola, pero JavaScript y math.js no poseen un sistema de ayuda interactiva comparable a Scilab u Octave; para las funciones se consulta la documentación. math.js no integra EDO; para ese caso en esta guía se usa `odex`.

```
const math = require('mathjs')
// ayuda del REPL de node

// buscar nombres relacionados con polinomios en math.js
Object.keys(math).filter(k=>/poly/i.test(k))

// ordenar: sort(comparador)
const x = [3,1,2]
x.sort((a,b)=>a-b)

// evaluar la expresion de un polinomio
math.evaluate("x^2-1",{x:2})

// resolver una EDO con odex
const odex = require("odex")
let f = (t,y)=>[-y[0]]
const sol = new odex.Solver(f,1).integrate(0,[1])
sol(1)
```

4. **Ejercicio:** Operaciones numéricas básicas

   Realice en la consola las siguientes operaciones, en este orden, y responda las preguntas de cada bloque.

   a) Aritmética encadenada:

- sume $1 + 4$;
- sume 3 al resultado anterior;
- divida el resultado por 4;
- sume 3 y eleve al cuadrado;
- guarde el resultado en `t`.

¿Cómo se llama en la calculadora que usa la variable que guarda el último resultado? ¿Qué significa asignársela a `t`?

b) Funciones elementales:

- calcule $t \cdot 3$;
- calcule $u = \sqrt{3 - \ln 4}$;
- redondee u al entero más cercano;
- trunque u.

¿Qué operación es `sqrt`? ¿De qué base es `log`? ¿Qué devuelve cada una de las dos últimas funciones?

c) Variable no definida: sume $v + t$ sin haber definido v. ¿Qué pasa?

d) Cadenas:

- defina `palabra` igual a la cadena hola;
- concatene `palabra` con chau, y con " chau" (con espacio);
- defina `nombre` y `apellido` con su nombre y apellido, y concaténelos en `completo`.

¿Por qué hola va entre comillas? ¿Por qué en `completo` no hacen falta comillas?

e) Complejos y constantes:

- calcule $\sqrt{-9}$ y eleve el resultado al cuadrado;
- calcule $1/3$;
- calcule $\ln e$.

¿Cómo se escribe la constante e y la unidad imaginaria en su calculadora?

f) Logaritmos:

- calcule $\ln 2$, $\cos \pi$, $\log_{10} 2$, $\log_2 10$;
- calcule con `log10` y con `log2` el logaritmo de su propia base.

¿Cuál es la base del logaritmo natural?

g) Liste las variables definidas hasta aquí.

h) Operaciones con complejos:

- defina el complejo $4 + 2i$;
- elévelo al cuadrado;
- multiplique el resultado por $4 + 2i$ y guárdelo en `m`;
- obtenga la parte real, la parte imaginaria y el conjugado de `m`;
- multiplique el conjugado por `m`;
- calcule $(16 - 88i)(16 + 88i)$, $|16 - 88i|$ y $\sqrt{8000}$.

¿Qué hace cada una de las tres operaciones sobre `m`? ¿Qué relación hay entre los tres últimos resultados?

Scilab. `ans` es el último resultado; `sqrt(-9)` da directamente `3.i`; las constantes llevan `%` y están protegidas.

```
1+4
ans+3
ans/4
(ans+3)^2
t=ans
t*3
u=sqrt(3-log(4))
round(u)
int(u)
// v+t
palabra="hola"
palabra+"chau"
palabra+" chau"
nombre="su_nombre"; apellido="su_apellido"
completo=nombre+apellido
complejo=sqrt(-9)
entero=complejo^2
1/3
log(%e)
log(2), cos(%pi), log10(2), log2(10)
log10(10), log2(2)
who
complejo=4+2*%i
complejo^2
m=ans*(4+2*%i)
real(m), imag(m), conj(m)
m2=ans*m
(16-88*%i)*(16+88*%i)
abs(16-88*%i)
sqrt(8000)
```

Octave. `ans` guarda el último resultado; `sqrt(-9)` devuelve directamente `3i`; `pi`, `e`, `i` y `j` no están protegidas y pueden sobrescribirse. Las cadenas se concatenan con corchetes.

```
1+4
ans+3
ans/4
(ans+3)^2
t=ans
t*3
u=sqrt(3-log(4))
round(u)
fix(u)
try
    v+t
catch err
```

```
    printf("error: %s\n", err.message);
end_try_catch
palabra="hola"
[palabra "chau"]
[palabra " chau"]
nombre="su_nombre"; apellido="su_apellido";
completo=[nombre apellido]
complejo=sqrt(-9)
entero=complejo^2
1/3
log(e)
log(2), cos(pi), log10(2), log2(10)
log10(10), log2(2)
who
complejo=4+2i
complejo^2
m=ans*(4+2i)
real(m), imag(m), conj(m)
m2=ans*m
(16-88i)*(16+88i)
abs(16-88i)
sqrt(8000)
```

Python. Tres calculadoras, mismos comandos de cálculo; cambia la variable del último resultado y cómo se listan las variables. En las tres la unidad imaginaria es `1j`, e y π son `np.e` y `np.pi` sin protección, y `np.sqrt(-9)` da `nan` con aviso: para el complejo, `np.sqrt(-9+0j)` o `np.emath.sqrt(-9)`. `np.round` y el `round` de Python difieren en `.5`.

```
# --- python3: el ultimo resultado se guarda en una variable
    ---
import numpy as np
r1 = 1+4
r2 = r1+3
r3 = r2/4
r4 = (r3+3)**2
t = r4
r5 = t*3
u = np.sqrt(3-np.log(4))
np.round(u); round(u)
int(u); np.trunc(u)
try:
    v+t                          # NameError: v no esta definida
except NameError as err:
    print("NameError:", err)
palabra = "hola"
palabra+"chau"; palabra+" chau"
nombre = "su_nombre"; apellido = "su_apellido"
completo = nombre+apellido
complejo = np.sqrt(-9)       # nan y aviso
```

```
complejo = np.sqrt(-9+0j)    # 3j
entero = complejo**2
1/3
np.log(np.e)
np.log(2), np.cos(np.pi), np.log10(2), np.log2(10)
np.log10(10), np.log2(2)
print([name for name in dir() if not name.startswith('_')])
   # variables definidas
complejo = 4+2j
r6 = complejo**2
m = r6*(4+2j)
m.real, m.imag, m.conjugate()
m2 = m.conjugate()*m
(16-88j)*(16+88j)
abs(16-88j)
np.sqrt(8000)

# --- IPython: ademas _2, _3, Out[2]; variables con %whos ---
# En el REPL de IPython, _ y Out[n] almacenan resultados
   anteriores.
# %whos lista las variables definidas.

# --- Jupyter: igual que IPython, en celdas; _ y Out[n];
   %whos ---
# cada linea en una celda, Shift-Enter ejecuta; la salida
   queda debajo
```

Perl. Dos calculadoras. En `perl -de1` no hay variable de último resultado: hay que asignar; `p` imprime; `sqrt(-9)` da error salvo con `Math::Complex`, que trae `i`, `pi`, `Re`, `Im` y `~` para conjugar; `round` y `log10` vienen con `POSIX`; `.` concatena y `+` suma convirtiendo; `V` lista las variables. En `pdl2` tampoco hay último resultado, pero todo lo demás viene cargado: los complejos son nativos (`cdouble`, `czip`, `re`, `im`, `conj`), y están `PI`, `E`, `log10`, `rint` (redondea) y `long` (trunca); las variables se listan recorriendo la tabla de símbolos.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::Constants qw(PI E);

# Operaciones aritmeticas basicas
my $r = 1 + 4;
$r = $r + 3;
$r = $r / 4;
$r = ($r + 3) ** 2;
my $t = $r;
print "t = $t\n";
print "t*3 = ", $t * 3, "\n";

my $u = sqrt(3 - log(4));
```

```
print "u = $u\n";
print "round(u) = ", int($u + 0.5), "\n";
print "int(u) = ", int($u), "\n";

# Sumar una variable no definida (v): con use strict da
    error de compilacion
eval '$v + $t';
print "v+t sin definir v da: $@" if $@;

my $palabra = "hola";
print "palabra.chau = ", $palabra . "chau", "\n";
print "palabra chau = ", $palabra . " chau", "\n";

my $nombre    = "su_nombre";
my $apellido  = "su_apellido";
my $completo  = $nombre . $apellido;
print "completo = $completo\n";

# Numeros complejos con PDL
my $complejo = sqrt(pdl(cdouble, -9));
print "sqrt(-9) = $complejo\n";
my $entero = $complejo ** 2;
print "entero = $entero\n";

print "1/3 = ", 1 / 3, "\n";
print "log(E) = ", log(E), "\n";
print "log(2) = ", log(2), "\n";
print "cos(PI) = ", cos(PI), "\n";
print "log10(2) = ", log(2) / log(10), "\n";
print "log2(10) = ", log(10) / log(2), "\n";
print "log10(10) = ", log(10) / log(10), "\n";
print "log2(2) = ", log(2) / log(2), "\n";

# Variables definidas (filtrando nombres de usuario)
my @vars = sort grep { /^[a-z]\w*$/ } keys %main::;
print "variables = @vars\n";

my $cplx = czip(4, 2);
print "czip(4,2) = $cplx\n";
my $m = $cplx ** 2 * czip(4, 2);
print "m = $m\n";
print "re(m) = ", $m->re, "\n";
print "im(m) = ", $m->im, "\n";
print "conj(m) = ", $m->conj, "\n";

my $m2 = $m * $m->conj;
print "m*conj(m) = $m2\n";

my $prod = czip(16, -88) * czip(16, 88);
print "(16-88i)*(16+88i) = $prod\n";
print "abs(16-88i) = ", abs(czip(16, -88)), "\n";
```

```
print "sqrt(8000) = ", sqrt(8000), "\n";
```

JavaScript. Dos calculadoras. En `node` el último resultado es `_`; en la consola del navegador es `$_`. En las dos, `Math.sqrt(-9)` da `NaN`: los complejos sólo con math.js (`require` en node, `<script>` en el navegador); `+` con una cadena concatena; `v+t` con `v` sin definir da `ReferenceError`; las variables globales se listan con `Object.keys(globalThis)`.

```
const math = require('mathjs')
// --- node ---
let ans = 1+4
ans = ans+3
ans = ans/4
ans = (ans+3)**2
let t = ans
ans = t*3
let u = Math.sqrt(3-Math.log(4))
Math.round(u); Math.trunc(u)
// v+t                          // ReferenceError en el REPL
let palabra = "hola"
palabra+"chau"; palabra+" chau"
let nombre = "su_nombre"; let apellido = "su_apellido"; let
   completo = nombre+apellido
Math.sqrt(-9)                   // NaN
let complejo = math.sqrt(-9)        // 3i
let entero = math.pow(complejo,2)
ans = 1/3
Math.log(Math.E)
Math.log(2); Math.cos(Math.PI); Math.log10(2); Math.log2(10)
Math.log10(10); Math.log2(2)
Object.keys(globalThis)         // variables del REPL
complejo=math.complex(4,2); ans = math.pow(complejo,2)
let m = math.multiply(ans,math.complex(4,2))
math.re(m); math.im(m); math.conj(m)
let m2 = math.multiply(math.conj(m),m)
math.multiply(math.complex(16,-88),math.complex(16,88))
math.abs(math.complex(16,-88)); Math.sqrt(8000)

// --- consola del navegador (F12): igual, salvo ---
// math.js: en el HTML de la pagina
// <script
   src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/13.0.0/math.js"></
// y en la consola  math  ya existe, sin require
```

5. **Ejercicio:** Manejo de fechas y cadenas

   Ejecute un paso por vez en la terminal de su lenguaje.

   a) Defina tres variables cuya suma produzca “Hoy es (fecha)”, con la fecha tomada del sistema, y guárdela en una variable.

b) Cuente los caracteres de cada una de las tres cadenas.

c) Transforme la frase en "Hoy no es (fecha)" empleando la sílaba "no" extraída de la palabra "mecanografía".

Scilab.

```
a="Hoy"; b=" es "; c=date();
frase=a+b+c
length(a), length(b), length(c)
no=part("mecanografia",5:6)
frase2=a+" "+no+b+c
```

Octave.

```
a="Hoy"; b=" es "; c=date();
frase=[a b c]
length(a), length(b), length(c)
no="mecanografia"(5:6)
frase2=[a " " no b c]
```

Python.

```
import datetime
a="Hoy"; b=" es "; c=str(datetime.date.today())
frase=a+b+c
len(a), len(b), len(c)
no="mecanografia"[4:6]
frase2=a+" "+no+b+c
```

Perl. `Time::Piece` obtiene la fecha del sistema; `ymd` la devuelve como año-mes-día sin usar una cadena de formato.

```
use Time::Piece;
length($a); length($b); length($c)
```

JavaScript.

```
let a = "Hoy"; let b = " es "; const c = new
   Date().toLocaleDateString()
let frase = a+b+c
a.length; b.length; c.length
let no = "mecanografia".slice(4,6)
let frase2 = a+" "+no+b+c
```

6. **Ejercicio:** Tipos de datos y dimensiones

Considere las variables: $a = 5$; $b = 5{,}0$; c la unidad imaginaria; d infinito; e "no es un número"; f verdadero; g la cadena `5`; v el rango $1, \ldots, 5$; A la matriz $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$. Antes de ejecutar:

a) Prediga el tipo, tamaño y contenido de cada variable.

b) Compruébelo con las funciones de tipo y tamaño de su calculadora.

c) Pruebe y explique: $a + b$, $a + c$, $a + g$, $d + 1$, $d - d$, $e == e$, $f + 1$.

d) Explique la diferencia entre un número, una cadena y un valor lógico.

e) Investigue qué representan el infinito, el "no es un número" y la unidad imaginaria.

f) Determine qué conversiones son automáticas y cuáles requieren una conversión explícita.

g) Escriba una función que reciba un dato y muestre su tipo y dimensiones.

Scilab. 5 y 5.0 son el mismo tipo; `a+g` da error; convertir con `string` y `evstr`/`strtod`.

```
a=5; b=5.0; c=%i; d=%inf; e=%nan; f=%t; g="5"; v=1:5; A=[1
   2;3 4];
type(a), typeof(a), size(a), length(g), size(v), size(A)
a+b, a+c, d+1, d-d, e==e, f+1
try
    a+g
catch
    disp(lasterror())
end
// "5"*3 no esta definido en Scilab; equivalente:
strcat(repmat("5",1,3))
string(a), evstr(g), strtod(g)
function info(x); disp(typeof(x)); disp(size(x)); endfunction
info(A)
```

Octave. Los números reales y complejos son `double`; las cadenas son `char` y los valores lógicos son `logical`. En una operación aritmética, una cadena se convierte en sus códigos de caracteres.

```
a=5; b=5.0; c=i; d=Inf; e=NaN; f=true; g="5"; v=1:5; A=[1
   2;3 4];
class(a), class(b), class(c), class(f), class(g)
size(a), length(g), size(v), size(A)
a+b, a+c, a+g, d+1, d-d, e==e, f+1
isnan(e), isinf(d)
num2str(a), str2double(g), double(g)
function info(x)
  disp(class(x)); disp(size(x));
endfunction
```

```
info(A)
```

Python. 5 es `int` y 5.0 `float`; `a+g` da error pero `"5"*3` repite la cadena; `np.nan == np.nan` es falso.

```
import numpy as np
a=5; b=5.0; c=1j; d=np.inf; e=np.nan; f=True; g="5";
   v=np.arange(1,6); A=np.array([[1,2],[3,4]])
type(a), type(b), np.shape(a), len(g), v.shape, A.shape,
   A.dtype
a+b, a+c, d+1, d-d, e==e, f+1
try:
    a+g                          # TypeError
except TypeError as err:
    print("TypeError:", err)
"5"*3                        # "555"
np.isnan(e), np.isinf(d)
str(a), int(g), float(g)
def info(x): print(type(x), np.shape(x))
info(A)
```

Perl. El escalar no tiene un tipo numérico fijo: es el operador el que decide cómo interpretarlo. Por ejemplo, `5+"5"` da 10, mientras que `5."5"` concatena. Los arreglos de Perl y los piddles tienen representaciones distintas, por lo que una función genérica que distinga el tipo de dato requiere decisiones que todavía no se han introducido. El último inciso se posterga hasta estudiar condicionales.

```
use PDL;
use Math::Complex ();

my $a = 5;
my $b = 5.0;
my $c = Math::Complex->make(0,1);
my $d = pdl('inf');
my $e = pdl('nan');
my $f = 1;
my $g = "5";
my @v = (1..5);
my $A = pdl([[1,2],[3,4]]);

# tipo, dimensiones y contenido
ref(\$a); $a;
ref(\$b); $b;
ref($c);  $c;
ref(\@v); scalar(@v); join " ", @v;
ref($A);  $A->dims; $A;

# operaciones pedidas
$a+$b;
$a+$c;
```

```
$a+$g;
$d+1;
$d-$d;
$e==$e;
$f+1;

# conversiones
$a."";
0+$g;

# La funcion generica pedida en (g) requiere
# distinguir entre tipos de datos: se vera
# despues de introducir condicionales.
```

JavaScript. Un solo tipo numérico: `5` y `5.0` son iguales; `5+"5"` concatena y `"5"*3` da 15; `NaN===NaN` es falso.

```
const math = require('mathjs')
let a = 5; let b = 5.0; let d = Infinity; let e = NaN; let f
   = true; let g = "5"; const v = [1,2,3,4,5]; const A =
   [[1,2],[3,4]]
let c = math.complex(0,1)
typeof a; typeof b; typeof g; typeof f; v.length;
   math.size(A)
a+b; math.add(a,c); a+g; d+1; d-d; e==e; f+1
"5"*3
Number.isNaN(e); Number.isFinite(d)
String(a); Number(g); parseFloat(g)
function info(x){ console.log(typeof x, math.size(x)) }
info(A)
```

7. **Ejercicio:** Unidades y conversiones

   Defina $g = 9{,}8\ \mathrm{m/s^2}$, $c = 3{\cdot}10^8\ \mathrm{m/s}$, y las constantes π y e tomadas de su calculadora, con la unidad en un comentario.

   Verifique cada valor escribiendo el nombre. Calcule el área de un círculo de radio $r = 5$ m y el volumen de una esfera del mismo radio, con unidades.

   Scilab.

```
g=9.8       // m/s^2
c=3e8       // m/s
pi=%pi
e=%e

g
c
pi
e
```

```
r=5          // m
area=pi*r^2           // m^2
volumen=4/3*pi*r^3    // m^3
```

Octave.

```
g=9.8        % m/s^2
c=3e8        % m/s
pi0=pi
e0=e

g
c
pi0
e0

r=5          % m
area=pi0*r^2           % m^2
volumen=4/3*pi0*r^3    % m^3
```

Python.

```
import numpy as np

g=9.8        # m/s^2
c=3e8        # m/s
pi=np.pi
e=np.e

g
c
pi
e

r=5          # m
area=pi*r**2           # m^2
volumen=4/3*pi*r**3    # m^3
```

Perl. `pi` viene con `Math::Trig`; *e* se obtiene como `exp(1)`.

```
use Math::Trig;


$g;
$c;
$pi;
$e;
```

```
$area;
$volumen;
```

JavaScript.

```
let g = 9.8         // m/s^2
let c = 3e8         // m/s
let pi = Math.PI
let e = Math.E

g
c
pi
e

let r = 5           // m
let area = pi*r**2            // m^2
let volumen = 4/3*pi*r**3     // m^3
```

4.23.3. Vectores, matrices y estructuras de datos

8. **Ejercicio:** Generación de vectores

   Genere sin escribir los valores:

   - (a) los enteros de 1 a 20;
   - (b) 50 valores equiespaciados entre 0 y 2π;
   - (c) los pares de 2 a 100;
   - (d) 30 aleatorios uniformes en $[0, 1)$;
   - (e) 30 aleatorios normales de media 5 y desvío 2. Para el último calcule media, desvío, máximo y mínimo y compare con los parámetros teóricos.

Scilab.

```
1:20
linspace(0,2*%pi,50)
2:2:100
rand(1,30)
x=rand(1,30,"normal")*2+5      // o grand(1,30,"nor",5,2)
mean(x), stdev(x), max(x), min(x)
```

Octave.

```
1:20
linspace(0,2*pi,50)
```

```
2:2:100
rand(1,30)
x=randn(1,30)*2+5
mean(x), std(x), max(x), min(x)
```

Python.

```
import numpy as np

np.arange(1,21)
np.linspace(0,2*np.pi,50)
np.arange(2,101,2)
np.random.rand(30)
x=np.random.normal(5,2,30)
np.mean(x), np.std(x), np.max(x), np.min(x)
```

Perl.

```
use PDL;

my $PI = 4*atan2(1,1);

sequence(20)+1;
zeroes(50)->xlinvals(0,2*$PI);
sequence(50)*2+2;
my $x = random(30);
avg($x); sqrt(sum(($x-avg($x))**2)/($x->nelem-1)); max($x);
   min($x);
```

JavaScript. Normal: No se puede directo en math.js; Box–Muller a mano.

```
const math = require('mathjs')
math.range(1,21)
math.range(0,2*Math.PI,2*Math.PI/49,true)
math.range(2,101,2)
math.random([30])
const x =
   Array.from({length:30},()=>5+2*Math.sqrt(-2*Math.log(Math.random()))*
math.mean(x); math.std(x); math.max(x); math.min(x)
```

9. **Ejercicio:** Producto escalar y norma

   Con $u = (2, 1)$ y $v = (1, 3)$ calcule: producto escalar, normas y ángulo; interprete el ángulo en radianes.

Scilab.

```
u=[2 1]; v=[1 3];
producto=u*v'
nu=norm(u)
```

```
nv=norm(v)
angulo=acos(producto/(nu*nv))
angulo/%pi                  // 0.25, es decir pi/4 rad
```

Octave.

```
u=[2 1]; v=[1 3];
producto=u*v'
nu=norm(u)
nv=norm(v)
angulo=acos(producto/(nu*nv))
angulo/pi                   % 0.25, es decir pi/4 rad
```

Python.

```
import numpy as np

u=np.array([2,1]); v=np.array([1,3])
producto=u@v
nu=np.linalg.norm(u)
nv=np.linalg.norm(v)
angulo=np.arccos(producto/(nu*nv))
angulo/np.pi                # 0.25, es decir pi/4 rad
```

Perl.

```
use PDL::Constants qw(PI);


$producto;
$nu;
$nv;
$angulo;
$angulo/PI;               # 0.25, es decir pi/4 rad
```

JavaScript.

```
const math = require('mathjs')
const u = [2,1]; const v = [1,3]
let producto = math.dot(u,v)
let nu = math.norm(u)
let nv = math.norm(v)
let angulo = Math.acos(producto/(nu*nv))
angulo/Math.PI              // 0.25, es decir pi/4 rad
```

10. **Ejercicio:** Suma y resta de vectores

    Con $u = (1, 3, -2, 5, 0)$ y $v = (2, -1, 4, 0, 3)$:

Calcule para ambos:

- (a) la suma;
- (b) la diferencia;
- (c) el producto elemento a elemento;
- (d) el producto escalar;
- (e) norma de u a mano, $\sqrt{\sum u_i^2}$, verificada con la función de norma;
- (f) ángulo entre u y v con $\cos\theta = \dfrac{u \cdot v}{\|u\|\|v\|}$.

Scilab.

```
u=[1 3 -2 5 0]; v=[2 -1 4 0 3];
u+v, u-v, u.*v
u*v'
sqrt(sum(u.^2)), norm(u)
acos(u*v'/(norm(u)*norm(v)))
```

Octave.

```
u=[1 3 -2 5 0]; v=[2 -1 4 0 3];
u+v, u-v, u.*v
u*v'
sqrt(sum(u.^2)), norm(u)
acos(u*v'/(norm(u)*norm(v)))
```

Python.

```
import numpy as np

u=np.array([1,3,-2,5,0]); v=np.array([2,-1,4,0,3])
u+v, u-v, u*v
u@v
np.sqrt(np.sum(u**2)), np.linalg.norm(u)
np.arccos(u@v/(np.linalg.norm(u)*np.linalg.norm(v)))
```

Perl.

```
use PDL;
use PDL::Math qw(acos);

my $u = pdl(1,3,-2,5,0);
my $v = pdl(2,-1,4,0,3);

$u+$v; $u-$v; $u*$v;
inner($u,$v);
sqrt(sum($u**2)); sqrt(inner($u,$u));
acos(inner($u,$v)/(sqrt(inner($u,$u))*sqrt(inner($v,$v))));
```

JavaScript.

```
const math = require('mathjs')
const u = [1,3,-2,5,0]; const v = [2,-1,4,0,3]
math.add(u,v); math.subtract(u,v); math.dotMultiply(u,v)
math.dot(u,v)
math.sqrt(math.sum(math.dotPow(u,2))); math.norm(u)
math.acos(math.dot(u,v)/(math.norm(u)*math.norm(v)))
```

11. **Ejercicio:** Multiplicación matricial

Teniendo en cuenta los dos tipos de multiplicaciones, la matricial y la elemento a elemento, y que en cada calculadora se escriben distinto.

a) Vector columna y transpuesta:
   - cree el vector columna $a = (121, 342, 531)^T$ de dos formas distintas;
   - obtenga $b = a^T$.

   ¿Qué operación es la transpuesta?

b) Armado de matrices a partir de vectores, con $u = (1, 2, 3)$ y $v = (4, 5, 6)$:
   - apile u sobre v en `c`;
   - ponga u y v uno al lado del otro en `d`;
   - ponga u^T y v^T como columnas en `e`.

c) Las dos multiplicaciones, con a y b:
   - $a + b$;
   - el producto matricial ab;
   - la división;
   - el producto elemento a elemento;
   - la división elemento a elemento;
   - $4a$, $a/3$ redondeado, $a - b$.

   ¿Qué distingue en la escritura un producto del otro?

d) Vector complejo $A = (4i, 2, 3, 3 - 3i)$:
   - A^2;
   - $A \cdot A$ elemento a elemento;
   - $\sqrt{A}$;
   - $\sqrt{A \cdot A}$.

e) Cadenas: cree $B = (a, b)$ y $C = (a, 4)$ y súmelos elemento a elemento.

f) Matriz compleja $\begin{pmatrix} 1+i & 2+3i \\ i & 4+i \end{pmatrix}$: obtenga la transpuesta conjugada y la transpuesta sin conjugar, y explique la diferencia.

g) Matrices especiales: ceros 4×4, identidad 5×5, unos 3×3, aleatoria 4×2.

Scilab. `*` es producto matricial y `.*` producto elemento a elemento. `'` transpone y conjuga, mientras `.'` sólo transpone. Como a es columna y b fila, la división matricial `a/b` no tiene dimensiones compatibles. A^2 tampoco está definido como potencia matricial porque A no es una matriz cuadrada.

```
a=[121;342;531]
a=[121 342 531]'
b=a'

u=[1 2 3]; v=[4 5 6];
c=[u;v]
d=[u v]
e=[u' v']

A3=a*ones(1,size(b,2))
B3=ones(size(a,1),1)*b
A3+B3
a*b
// a/b no es posible con estas dimensiones
A3.*B3
A3./B3
4*a
round(a/3)
A3-B3

A=[4*%i 2 3 3-3*%i]
// A^2 no existe como potencia matricial: A no es cuadrada
A.^2
A.*A
sqrt(A)
sqrt(A.*A)

B=["a" "b"]; C=["a" "4"];
B+C

matriz=[1+%i 2+3*%i; %i 4+%i]
conjugada=matriz'
transpuesta=matriz.'

zeros(4,4)
eye(5,5)
ones(3,3)
rand(4,2)
```

Octave. `*` es producto matricial y `.*` producto elemento a elemento. `'` es la transpuesta conjugada y `.'` la transpuesta sin conjugar. La división matricial `a/b` no tiene dimensiones compatibles. Para el vector A, `A^2` es una potencia matricial y no está definida; la potencia elemento a elemento es `A.^2`.

```
a=[121;342;531]
a=[121 342 531]'
b=a'
```

```
u=[1 2 3]; v=[4 5 6];
c=[u;v]
d=[u v]
e=[u' v']

a+b
a*b
% a/b no es posible con estas dimensiones
a.*b
a./b
4*a
round(a/3)
a-b

A=[4i 2 3 3-3i]
% A^2 no existe como potencia matricial: A no es cuadrada
A.^2
A.*A
sqrt(A)
sqrt(A.*A)

B=["a" "b"]; C=["a" "4"];
B+C

matriz=[1+i 2+3i; i 4+i]
conjugada=matriz'
transpuesta=matriz.'

zeros(4,4)
eye(5)
ones(3,3)
rand(4,2)
```

Python. Un arreglo de una dimensión no es fila ni columna, por eso las dos construcciones de `a` conservan dimensión 3×1. `@` es producto matricial y `*` producto elemento a elemento. Python no tiene un operador de división matricial; `/` es división elemento a elemento. `.T` transpone y `.conj().T` transpone y conjuga.

```
import numpy as np

a=np.array([[121],[342],[531]])
a=np.array([121,342,531]).reshape(3,1)
b=a.T

u=np.array([1,2,3]); v=np.array([4,5,6])
c=np.vstack([u,v])
d=np.hstack([u,v])
e=np.column_stack([u,v])

a+b
```

```
a@b
# no hay operador de division matricial
a*b
a/b
4*a
np.round(a/3)
a-b

A=np.array([4j,2,3,3-3j])
# A no es cuadrada: no hay potencia matricial A^2
A**2
A*A
np.sqrt(A)
np.sqrt(A*A)

B=np.array(["a","b"]); C=np.array(["a","4"])
np.char.add(B,C)

matriz=np.array([[1+1j,2+3j],[1j,4+1j]])
conjugada=matriz.conj().T
transpuesta=matriz.T

np.zeros((4,4))
np.eye(5)
np.ones((3,3))
np.random.rand(4,2)
```

Perl. En PDL un piddle unidimensional se interpreta como fila; para obtener una columna se agrega una dimensión unitaria. `x` es producto matricial y `*` producto elemento a elemento. No hay aquí una división matricial equivalente a `a/b`; `/` es elemento a elemento.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;

# Vector columna y su transpuesta
my $a = pdl([121, 342, 531])->reshape(1, 3);
my $b = $a->transpose;

print "a = $a\n";
print "b = $b\n";
print "a+b = ", $a + $b, "\n";
print "a x b = ", $a x $b, "\n";
print "a*b = ", $a * $b, "\n";
print "a/b = ", $a / $b, "\n";
print "4*a = ", 4 * $a, "\n";
print "rint(a/3) = ", rint($a / 3), "\n";
print "a-b = ", $a - $b, "\n";
```

```
# Vectores y apilados
my $u = pdl([1, 2, 3]);
my $v = pdl([4, 5, 6]);
my $c = $u->dummy(1, 1)->glue(1, $v->dummy(1, 1));       #
   vstack
my $d = $u->append($v);                                    #
   hstack
my $e = $u->dummy(0, 1)->glue(0, $v->dummy(0, 1));       #
   column_stack
print "c = $c\n";
print "d = $d\n";
print "e = $e\n";

# Numeros complejos
my $A = czip(pdl([4, 2, 3, 3])->reshape(2, 2), pdl([0, 0, 0,
   -3])->reshape(2, 2));
print "A = $A\n";
print "A**2 = ", $A ** 2, "\n";
print "A*A = ", $A * $A, "\n";
print "sqrt(A) = ", sqrt($A), "\n";
print "sqrt(A*A) = ", sqrt($A * $A), "\n";

# Concatenacion de cadenas (nativa de Perl)
my @B = ("a", "b");
my @C = ("a", "4");
my @concat = map { $B[$_] . $C[$_] } 0 .. 1;
print "concatenacion = @concat\n";

# Matriz compleja: conjugada y transpuesta
my $M = pdl([[czip(1, 1), czip(2, 3)], [czip(0, 1), czip(4,
   1)]]);
my $conjugada = $M->conj;
my $transpuesta = $M->transpose;
print "M = $M\n";
print "conjugada = $conjugada\n";
print "transpuesta = $transpuesta\n";

# Matrices especiales
my $Z = zeroes(4, 4);
my $I = identity(5);
my $O = ones(3, 3);
my $R = random(4, 2);
print "zeros = $Z\n";
print "identity = $I\n";
print "ones = $O\n";
print "random = $R\n";
```

JavaScript. En math.js el vector debe conservar dos dimensiones para distinguir columna y fila. `multiply` es producto matricial y `dotMultiply` producto elemento a elemento. La división matricial requiere un divisor invertible, por lo que no corresponde para la fila b. `ctranspose` transpone y conjuga.

```
const math = require('mathjs')
let a = math.matrix([[121],[342],[531]])
a=math.transpose(math.matrix([[121,342,531]]))
let b = math.transpose(a)

const u = [1,2,3]; const v = [4,5,6]
let c = math.concat([u],[v],0)
let d = math.concat(u,v)
let e = math.transpose(math.concat([u],[v],0))

math.add(a,b)
math.multiply(a,b)
// math.divide(a,b) no corresponde: b no es invertible
math.dotMultiply(a,b)
math.dotDivide(a,b)
math.multiply(4,a)
math.round(math.dotDivide(a,3))
math.subtract(a,b)

const A = [math.complex(0,4),2,3,math.complex(3,-3)]
// A no es cuadrada: no hay potencia matricial A^2
math.dotPow(A,2)
math.dotMultiply(A,A)
math.map(A, math.sqrt)
math.map(math.dotMultiply(A,A), math.sqrt)

const B = ["a","b"]; const C = ["a","4"]
B.map((s,i)=>s+C[i])

let matriz = math.matrix([
  [math.complex(1,1),math.complex(2,3)],
  [math.complex(0,1),math.complex(4,1)]
])
let conjugada = math.ctranspose(matriz)
let transpuesta = math.transpose(matriz)

math.zeros(4,4)
math.identity(5)
math.ones(3,3)
math.random([4,2])
```

12. **Ejercicio:** Cadenas de caracteres

    Con `s1` igual a “Me llamo ” y `s2` igual a “Luciano”: concatene en `frase`; mida las longitudes de las tres cadenas; busque la posición de Luciano en `frase`; extraiga el nombre por posiciones fijas, luego usando la posición encontrada, luego usando la posición y la longitud; reemplace Luciano por Pedro. Cree un vector de cadenas (Nombre, Apellido, s1, s2), mida cada elemento, extraiga subconjuntos, apile dos filas, y una los elementos de una fila con la letra y entre ellos. Convierta el rango $0, 0{,}1, \ldots, 1$ a cadenas y de vuelta a números; calcule con los números el producto

escalar y el producto externo. Obtenga el código numérico del guion y el carácter correspondiente al código, y una las cadenas del rango con un guion. Pase "laboratorio" a mayúsculas, y un vector de cadenas a mayúsculas y de vuelta a minúsculas. Explique cada función con un comentario.

Scilab. Posiciones desde 1; `strindex` da todas las apariciones.

```
s1="Me llamo "; s2="Luciano";
frase=s1+s2
length(s1), length(s2), length(frase)
strindex(frase,"Luciano")
nombre=part(frase,[10:16])
i=strindex(frase,"Luciano"); nombre=part(frase,[i:16])
nombre=part(frase,[i:i+length("Luciano")-1])
strsubst(frase,"Luciano","Pedro")
matriz=["Nombre" "Apellido" s1 s2]
length(matriz)
u=matriz(1:4)
v1=matriz(1,:); v2=matriz(1,:); v3=[v1;v2]
strcat(v1,"y")
m=string(0:0.1:1)
mn=evstr(m)
mn*mn'
mn'*mn
codigo_guion=ascii("-")
guion=ascii(codigo_guion)
strcat(m,guion)
u="laboratorio"; convstr(u,"u")
A=["esta" "es" "mi" "matriz"]; B=convstr(A,"u");
   convstr(B,"l")
```

Octave. Para cadenas de longitudes distintas se usa un arreglo de celdas; se accede a su contenido con llaves.

```
s1="Me llamo "; s2="Luciano";
frase=[s1 s2]
length(s1), length(s2), length(frase)
strfind(frase,"Luciano")
i=strfind(frase,"Luciano");
nombre=frase(i:i+length("Luciano")-1)
strrep(frase,"Luciano","Pedro")
matriz={"Nombre","Apellido",s1,s2}
length(matriz)
u=matriz(1:4)
v1=matriz(1,:); v2=matriz(1,:); v3=[v1;v2]
strcat(v1,"y")
m=arrayfun(@num2str,0:0.1:1,"UniformOutput",false)
mn=str2double(m)
mn*mn'
mn'*mn
```

```
codigo_guion=double("-")
guion=char(codigo_guion)
strcat(m,guion)
u="laboratorio"; upper(u)
A={"esta","es","mi","matriz"};
B=cellfun(@upper,A,"UniformOutput",false);
cellfun(@lower,B,"UniformOutput",false)
```

Python. Posiciones desde 0; `find` da la primera aparición; cortes `[i:j]`; `join` une con separador. Las conversiones del vector se hacen con operaciones de NumPy, sin escribir ciclos explícitos.

```
import numpy as np

s1="Me llamo "; s2="Luciano"
frase=s1+s2
len(s1), len(s2), len(frase)
frase.find("Luciano")
nombre=frase[9:16]
i=frase.find("Luciano"); nombre=frase[i:16]
nombre=frase[i:i+len("Luciano")]
frase.replace("Luciano","Pedro")
matriz=np.array(["Nombre","Apellido",s1,s2])
np.char.str_len(matriz)
u=matriz[0:4]
v1=matriz[0:1]; v2=matriz[1:2]; v3=np.vstack([v1,v2])
"y".join(v1)

x=np.round(np.arange(0,1.05,0.1),1)
m=x.astype(str)
mn=m.astype(float)

mn@mn
np.outer(mn,mn)
codigo_guion=ord("-")
guion=chr(codigo_guion)
guion.join(m)
u="laboratorio"; u.upper()
A=np.array(["esta","es","mi","matriz"])
B=np.char.upper(A)
np.char.lower(B)
```

Perl. Instrucciones: `.` concatena, `index`, `substr`, `s///`, `join`, `uc`/`lc`; las cadenas y los números se convierten solos.

```
length($s1); length($s2); length($frase)
index($frase,"Luciano")
($f2=$frase) =~ s/Luciano/Pedro/; $f2
@matriz=("Nombre","Apellido",$s1,$s2)
join " ", map { length } @matriz
```

```
@u=@matriz[0..3]
@v1=@matriz[0..1]; @v2=@matriz[2..3]; @v3=(@v1,@v2)
join "y", @v1
@m=map { $_/10 } 0..10
@ms=map { $_."" } @m
@mn=map { $_+0 } @ms
join $guion, @ms
uc "laboratorio"
@A=("esta","es","mi","matriz");
@B=map { uc } @A;
join " ", map { lc } @B
```

JavaScript. `indexOf`, `slice`, `replace`, `join`, `toUpperCase`; `Number` y `String` convierten.

```
const math = require('mathjs')
let s1 = "Me llamo "; let s2 = "Luciano"
let frase = s1+s2
s1.length; s2.length; frase.length
frase.indexOf("Luciano")
let nombre = frase.slice(9,16)
let i = frase.indexOf("Luciano"); nombre=frase.slice(i,16)
nombre=frase.slice(i,i+"Luciano".length)
frase.replace("Luciano","Pedro")
const matriz = ["Nombre","Apellido",s1,s2]
matriz.map(s=>s.length)
let u = matriz.slice(0,4)
let v1 = matriz.slice(0,2); let v2 = matriz.slice(2,4);
   const v3 = [v1,v2]
v1.join("y")
const m =
   Array.from({length:11},(_,i)=>String(Math.round(i)/10))
let mn = m.map(Number)
math.dot(mn,mn)
math.kron([mn],math.transpose([mn]))
let codigo = "-".charCodeAt(0)
let guion = String.fromCharCode(codigo)
m.join(guion)
"laboratorio".toUpperCase()
const A = ["esta","es","mi","matriz"]
let B = A.map(s=>s.toUpperCase())
B.map(s=>s.toLowerCase())
```

13. **Ejercicio:** Rangos y secuencias

    a) Genere con el operador de rango los vectores $u = (1, \ldots, 8)$, $v = (0, 0{,}2, 0{,}4, \ldots, 3{,}4)$ y $w = (9, 8, 7, 6, 5, 4)$.

    b) Cree la matriz A de tabla de multiplicar 5×5 ($A_{ij} = ij$) a partir de un único vector y su transpuesto.

c) Extraiga la primera fila y la última columna de A y calcule su producto escalar.

d) Calcule la transpuesta de A.

e) Cree una matriz aleatoria B de 5×5 y efectúe las operaciones habituales con A.

f) Extraiga el elemento de la tercera fila y cuarta columna de A.

g) Extraiga la diagonal de A.

h) Multiplique el sexto elemento de B por el decimoctavo de A, contando los elementos por columnas como hace Scilab.

i) Extraiga las filas 3 y 4 de A y B.

j) Extraiga las columnas 1 y 5 de A y B.

k) Con el vector de posiciones $b = (1, 6, 8, 20)$, contadas por columnas, extraiga los elementos correspondientes de A y B.

Scilab.

```
u=1:8
v=0:0.2:3.4
w=9:-1:4
x=1:5; A=x'*x
A(1,:)*A(:,$)
A'
B=rand(5,5); A+B, A*B, A.*B
A(3,4)
diag(A)
B(6)*A(18)
A(3:4,:), B(3:4,:)
A(:,[1 5]), B(:,[1 5])
b=[1 6 8 20]; A(b), B(b)
```

Octave. `end` representa el último índice.

```
u=1:8
v=0:0.2:3.4
w=9:-1:4
x=1:5; A=x'*x
A(1,:)*A(:,end)
A'
B=rand(5,5); A+B, A*B, A.*B
A(3,4)
diag(A)
B(6)*A(18)
A(3:4,:), B(3:4,:)
A(:,[1 5]), B(:,[1 5])
b=[1 6 8 20]; A(b), B(b)
```

Python. `arange` excluye el extremo final; el índice lineal por columnas exige `ravel(order='F')`.

```
import numpy as np

u=np.arange(1,9)
v=np.arange(0,3.41,0.2)           # o np.linspace(0,3.4,18)
w=np.arange(9,3,-1)
x=np.arange(1,6); A=np.outer(x,x)
A[0,:] @ A[:,-1]
A.T
B=np.random.rand(5,5); A+B, A@B, A*B
A[2,3]
np.diag(A)
B.ravel(order='F')[5] * A.ravel(order='F')[17]
A[2:4,:], B[2:4,:]
A[:,[0,4]], B[:,[0,4]]
b=np.array([1,6,8,20]); A.ravel(order='F')[b-1]
```

Perl. PDL: `sequence` genera; `slice("col,fila")` y `at(col,fila)` van con la columna primero; `flat` recorre por columnas, como Scilab.

```
use PDL;

my $x = pdl(1..5);
my $A = outer($x,$x);
my $B = random(5,5);
my $b = pdl(1,6,8,20);

inner($A->slice(":,0"),$A->slice("-1,:"));
$A->transpose;

$A+$B;
$A x $B;
$A*$B;

$A->at(3,2);
$A->diagonal(0,1);

$B->flat->at(5)*$A->flat->at(17);

$A->slice(":,2:3");
$B->slice(":,2:3");

$A->slice("0:4:4,:");
$B->slice("0:4:4,:");

$A->flat->index($b-1);
$B->flat->index($b-1);
```

JavaScript. math.js: `range` excluye el final; `flatten` recorre por filas, así que

para índices lineales contados por columnas se aplana la transpuesta.

```
const math = require('mathjs')
let u = math.range(1,9)
let v = math.range(0,3.41,0.2)
let w = math.range(9,3,-1)

let x = math.range(1,6)
let A =
   math.matrix(math.multiply(math.transpose([x.toArray()]),[x.toArray()]
math.multiply(math.row(A,0),math.column(A,4))
math.transpose(A)

let B = math.matrix(math.random([5,5]))
math.add(A,B)
math.multiply(A,B)
math.dotMultiply(A,B)

A.get([2,3])
math.diag(A)

math.flatten(math.transpose(B)).get([5]) *
math.flatten(math.transpose(A)).get([17])

math.subset(A,math.index([2,3],math.range(0,5)))
math.subset(B,math.index([2,3],math.range(0,5)))

math.subset(A,math.index(math.range(0,5),[0,4]))
math.subset(B,math.index(math.range(0,5),[0,4]))

const b = [1,6,8,20]
b.map(i=>math.flatten(math.transpose(A)).get([i-1]))
b.map(i=>math.flatten(math.transpose(B)).get([i-1]))
```

14. **Ejercicio:** Recta paramétrica

Calcule puntos de la recta por $P_0 = (1, 2)$ con dirección $d = (3, -1)$ para $t \in [-2, 2]$ y repita con otra dirección.

Scilab.

```
t=-2:0.1:2;

x=1+3*t;
y=2-t;
puntos=[x' y'];

x2=1+t;
y2=2+2*t;
puntos2=[x2' y2'];
```

Octave.

```
t=-2:0.1:2;

x=1+3*t;
y=2-t;
puntos=[x(:) y(:)];

x2=1+t;
y2=2+2*t;
puntos2=[x2(:) y2(:)];
```

Python.

```
import numpy as np

t=np.arange(-2,2.05,0.1)

x=1+3*t
y=2-t
puntos=np.column_stack((x,y))

x2=1+t
y2=2+2*t
puntos2=np.column_stack((x2,y2))
```

Perl.

```
```

JavaScript.

```
const math = require('mathjs')
let t = math.range(-2,2.05,0.1).toArray()

let x = t.map(v=>1+3*v)
let y = t.map(v=>2-v)
let puntos = math.transpose([x,y])

let x2 = t.map(v=>1+v)
let y2 = t.map(v=>2+2*v)
let puntos2 = math.transpose([x2,y2])
```

15. **Ejercicio:** Plano en el espacio

    Verifique si $P_1 = (1, 0, 2)$ y $P_2 = (2, 1, 1)$ pertenecen al plano $2x-y+z = 4$ evaluando $n \cdot P + d$ con $n = (2, -1, 1)$, $d = -4$.

Scilab.

```
P1=[1 0 2]; P2=[2 1 1]; n=[2 -1 1]; d=-4; n*P1'+d, n*P2'+d
```

Octave.

```
P1=[1 0 2]; P2=[2 1 1]; n=[2 -1 1]; d=-4;
n*P1'+d
n*P2'+d
```

Python.

```
import numpy as np

P1=np.array([1,0,2]); P2=np.array([2,1,1]);
   n=np.array([2,-1,1]); d=-4; n@P1+d, n@P2+d
```

Perl.

JavaScript.

```
const math = require('mathjs')
const P1 = [1,0,2]; const P2 = [2,1,1]; const n = [2,-1,1];
   let d = -4; math.dot(n,P1)+d; math.dot(n,P2)+d
```

16. **Ejercicio:** Elipse parametrizada

Calcule puntos de la elipse $x = 3\cos t$, $y = 2\sin t$ y repita con otros coeficientes.

Scilab.

```
t=linspace(0,2*%pi,300);

x=3*cos(t);
y=2*sin(t);
puntos=[x' y'];

x2=4*cos(t);
y2=sin(t);
puntos2=[x2' y2'];
```

Octave.

```
t=linspace(0,2*pi,300);

x=3*cos(t);
y=2*sin(t);
```

```
puntos=[x(:) y(:)];

x2=4*cos(t);
y2=sin(t);
puntos2=[x2(:) y2(:)];
```

Python.

```
import numpy as np

t=np.linspace(0,2*np.pi,300)

x=3*np.cos(t)
y=2*np.sin(t)
puntos=np.column_stack((x,y))

x2=4*np.cos(t)
y2=np.sin(t)
puntos2=np.column_stack((x2,y2))
```

Perl.

JavaScript.

```
const math = require('mathjs')
let t = math.range(0,2*Math.PI,2*Math.PI/299,true).toArray()

let x = t.map(v=>3*Math.cos(v))
let y = t.map(v=>2*Math.sin(v))
let puntos = math.transpose([x,y])

let x2 = t.map(v=>4*Math.cos(v))
let y2 = t.map(v=>Math.sin(v))
let puntos2 = math.transpose([x2,y2])
```

17. **Ejercicio:** Indexación de matrices

Construya la matriz 8×8 con los enteros 1 a 64 llenada por columnas. Sin volver a escribir elementos:

- (a) obtenga la primera fila;
- (b) obtenga la última fila usando el índice de "último";
- (c) obtenga la segunda columna;
- (d) obtenga la penúltima columna;
- (e) extraiga el bloque central 4×4;
- (f) extraiga las filas pares;
- (g) extraiga las columnas impares;

- (h) extraiga los elementos en filas pares y columnas impares;
- (i) invierta el orden de las filas;
- (j) invierta el orden de las columnas;
- (k) reemplace el borde por ceros;

Scilab.

```
A=matrix(1:64,8,8)
A(1,:)
A($,:)
A(:,2)
A(:,$-1)
A(3:6,3:6)
A(2:2:$,:)
A(:,1:2:$)
A(2:2:$,1:2:$)
A($:-1:1,:)
A(:,$:-1:1)
A([1 $],:)=0; A(:,[1 $])=0
```

Octave. `end` sustituye al símbolo `$` de Scilab.

```
A=reshape(1:64,8,8)
A(1,:)
A(end,:)
A(:,2)
A(:,end-1)
A(3:6,3:6)
A(2:2:end,:)
A(:,1:2:end)
A(2:2:end,1:2:end)
A(end:-1:1,:)
A(:,end:-1:1)
A([1 end],:)=0; A(:,[1 end])=0
```

Python. `reshape` llena por filas; `order='F'` indica que debe llenar por columnas.

```
import numpy as np
A=np.arange(1,65).reshape(8,8,order='F')
A[0]
A[-1]
A[:,1]
A[:,-2]
A[2:6,2:6]
A[1::2]
A[:,0::2]
A[1::2,0::2]
```

```
A[::-1]
A[:,::-1]
A[[0,-1],:]=0; A[:,[0,-1]]=0
```

Perl. PDL guarda primero la columna: `sequence(8,8)` queda llenada por columnas. En `slice`, el primer índice es la columna y `.=` modifica los elementos seleccionados.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;

# Matriz 8x8 llenada por columnas (equivalente a order='F'
   de NumPy)
my $A = sequence(64)->reshape(8, 8)->transpose + 1;
print "A = $A\n";

print "primera fila = ", $A->slice(':,0'), "\n";
print "ultima fila = ", $A->slice(':,-1'), "\n";
print "segunda columna = ", $A->slice('1,:'), "\n";
print "penultima columna = ", $A->slice('-2,:'), "\n";
print "submatriz 2:5,2:5 = ", $A->slice('2:5,2:5'), "\n";
print "filas pares = ", $A->slice(':,1:-1:2'), "\n";
print "columnas impares = ", $A->slice('0:-1:2,:'), "\n";
print "filas pares, columnas impares = ",
   $A->slice('0:-1:2,1:-1:2'), "\n";
print "filas invertidas = ", $A->slice(':,-1:0'), "\n";
print "columnas invertidas = ", $A->slice('-1:0,:'), "\n";

# Poner en cero primera y ultima fila, y primera y ultima
   columna
my $B = $A->copy;
$B->slice(':,0') .= 0;
$B->slice(':,-1') .= 0;
$B->slice('0,:') .= 0;
$B->slice('-1,:') .= 0;
print "B = $B\n";
```

JavaScript. math.js llena por filas; se transpone la matriz para obtener el llenado por columnas. No admite índices negativos para indicar el último elemento.

```
const math = require('mathjs')
  let A =
     math.transpose(math.reshape(math.range(1,65),[8,8]))
  let n = math.size(A)[0]
  math.row(A,0)
  math.row(A,n-1)
  math.column(A,1)
  math.column(A,n-2)
  math.subset(A,math.index(math.range(2,6),math.range(2,6)))
```

```
    math.subset(A,math.index(math.range(1,n,2),math.range(0,n)))
    math.subset(A,math.index(math.range(0,n),math.range(0,n,2)))
    math.subset(A,math.index(math.range(1,n,2),math.range(0,n,2)))
    math.subset(A,math.index(math.range(n-1,-1,-1),math.range(0,n)))
    math.subset(A,math.index(math.range(0,n),math.range(n-1,-1,-1)))
    A=math.subset(A,math.index([0,n-1],math.range(0,n)),0)
    A=math.subset(A,math.index(math.range(0,n),[0,n-1]),0)
```

18. **Ejercicio:** Álgebra matricial

Con

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix},$$

realice las siguientes operaciones:

- (a) obtenga la transpuesta de A;
- (b) calcule el determinante;
- (c) calcule la inversa si el determinante es distinto de cero;
- (d) verifique que $AA^{-1} \approx I$;
- (e) explique por qué el resultado puede no ser exactamente la matriz identidad;
- (f) indique cuándo existe la inversa de una matriz;
- (g) explique qué significa que el determinante sea cero;
- (h) describa las funciones utilizadas para calcular la transpuesta, el determinante y la inversa.

Para esta matriz el determinante es distinto de cero, por lo que la inversa existe. Al multiplicar una matriz por su inversa pueden aparecer pequeños errores respecto de la identidad debido al redondeo de la aritmética de punto flotante.

Una matriz cuadrada es invertible si y sólo si su determinante es distinto de cero. Si el determinante es cero, la matriz es singular: sus filas o columnas son linealmente dependientes y no tiene inversa.

Scilab. `'` obtiene la transpuesta conjugada; como A es real, coincide con la transpuesta. `det(A)` calcula el determinante, `inv(A)` la inversa y `eye` construye la matriz identidad.

```
A=[1 2 3;4 5 6;7 8 10]

At=A'
d=det(A)

Ai=inv(A)
P=A*Ai

P
P-eye(3,3)
```

Octave. `'` obtiene la transpuesta conjugada; como A es real, coincide con la transpuesta. `det` calcula el determinante, `inv` la inversa y `eye` la identidad.

```
A=[1 2 3;4 5 6;7 8 10]

At=A'
d=det(A)

Ai=inv(A)
P=A*Ai

P
P-eye(3)
```

Python. `A.T` obtiene la transpuesta, `np.linalg.det(A)` calcula el determinante, `np.linalg.inv(A)` la inversa y `np.eye` construye la identidad. `np.allclose` permite verificar igualdad aproximada teniendo en cuenta errores de redondeo.

```
import numpy as np

A=np.array([[1,2,3],
            [4,5,6],
            [7,8,10]])

At=A.T
d=np.linalg.det(A)

Ai=np.linalg.inv(A)
P=A@Ai

P
P-np.eye(3)
np.allclose(P,np.eye(3))
```

Perl. `transpose` obtiene la transpuesta. `PDL::MatrixOps` proporciona `det` e `inv`; `x` representa el producto matricial e `identity` construye la identidad.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::MatrixOps;

my $A = pdl([[1,2,3],[4,5,6],[7,8,10]]);

my $At = $A->transpose;
print "At = $At\n";

my $d = det($A);
print "det(A) = $d\n";
```

```
my $Ai = $A->inv;
print "inv(A) = $Ai\n";

my $P = $A x $Ai;
print "P = $P\n";
print "P - I = ", ($P - identity(3))->flat, "\n";

my $es_identidad = max(abs($P - identity(3))) < 1e-10;
print "P es aproximadamente la identidad: ", ($es_identidad
   ? "si" : "no"), "\n";
```

JavaScript. `math.transpose` obtiene la transpuesta, `math.det` calcula el determinante, `math.inv` la inversa y `math.identity` construye la identidad.

```
const math = require('mathjs')
const A = [[1,2,3],
   [4,5,6],
   [7,8,10]]

let At = math.transpose(A)
let d = math.det(A)

let Ai = math.inv(A)
let P = math.multiply(A,Ai)

P
math.subtract(P,math.identity(3))
```

19. **Ejercicio:** Independencia lineal

Con las columnas $(1,0,2)$, $(2,1,0)$, $(0,1,1)$, calcule el determinante: si es distinto de cero, forman base de $\mathbb{R}^3$.

Scilab.

```
A=[1 2 0;0 1 1;2 0 1]; det(A)
```

Octave.

```
A=[1 2 0;0 1 1;2 0 1];
det(A)
rank(A)
```

Python.

```
import numpy as np
```

```
A=np.array([[1,2,0],[0,1,1],[2,0,1]]); np.linalg.det(A);
   np.linalg.matrix_rank(A)
```

Perl.

JavaScript.

```
const math = require('mathjs')
const A = [[1,2,0],[0,1,1],[2,0,1]]; math.det(A)
```

20. **Ejercicio:** Transformaciones lineales

Aplique $A = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}$ al cuadrado unidad dado por sus vértices en columnas y calcule los vértices transformados.

Scilab.

```
P=[0 1 1 0 0;0 0 1 1 0]; A=[2 1;0 1]; Q=A*P;
```

Octave.

```
P=[0 1 1 0 0;0 0 1 1 0];
A=[2 1;0 1];
Q=A*P
```

Python.

```
import numpy as np

P=np.array([[0,1,1,0,0],[0,0,1,1,0]]);
   A=np.array([[2,1],[0,1]]); Q=A@P
```

Perl.

JavaScript.

```
const math = require('mathjs')
const P = [[0,1,1,0,0],[0,0,1,1,0]]; const A =
   [[2,1],[0,1]]; let Q = math.multiply(A,P)
```

21. **Ejercicio:** Autovalores de matrices

Para $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ calcule autovalores y autovectores y verifique $AX - XD \approx 0$. ¿Qué información aportan los autovalores?

Los autovectores indican direcciones que la transformación no cambia de dirección. El autovalor correspondiente indica el factor por el que se multiplica un vector en esa dirección. En este caso los autovalores son 3 y 1.

Scilab. `spec` devuelve los autovectores en las columnas de `X` y los autovalores en la matriz diagonal `D`.

```
A=[2 1;1 2]
[X,D]=spec(A)
A*X-X*D
```

Octave. `eig` devuelve los autovectores en las columnas de `X` y los autovalores en la matriz diagonal `D`.

```
A=[2 1;1 2]
[X,D]=eig(A)
A*X-X*D
```

Python. `np.linalg.eig` devuelve primero los autovalores y luego una matriz cuyas columnas son los autovectores.

```
import numpy as np

A=np.array([[2,1],[1,2]])
d,X=np.linalg.eig(A)
D=np.diag(d)
d
X
A@X-X@D
```

Perl. `eigens_sym` se usa porque la matriz es simétrica.

```
use PDL;
use PDL::LinearAlgebra;

my $A = pdl([[2,1],[1,2]]);
($X,$d)=eigens_sym($A);
$d;
$X;
$A x $X - $X x stretcher($d);
```

JavaScript. `math.eigs` devuelve los autovalores y los autovectores.

```
const math = require('mathjs')
const A = [[2,1],[1,2]]
let e = math.eigs(A)
let D = math.diag(e.values)
let X = math.transpose(e.eigenvectors.map(v=>v.vector))
e.values
```

```
X
math.subtract(math.multiply(A,X),math.multiply(X,D))
```

22. **Ejercicio:** Triangularización de matrices

Con $A = \begin{pmatrix} 3 & 5 & 1 & 4 \\ 8 & 2 & 4 & 6 \\ 6 & 7 & 9 & 2 \\ 1 & 3 & 5 & 8 \end{pmatrix}$:

- (a) triangular superior;
- (b) triangular inferior;
- (c) diagonal principal;
- (d) qué elementos se anulan en cada caso;
- (e) sume superior e inferior: ¿qué pasa con la diagonal?

En la triangular superior se anulan los elementos que están debajo de la diagonal principal. En la triangular inferior se anulan los que están por encima.

Al sumar ambas matrices se recuperan todos los elementos de A, pero los elementos de la diagonal aparecen dos veces. Por lo tanto,

$$U + L = A + \text{diag}(\text{diag}(A)).$$

Scilab.

```
A=[3 5 1 4;8 2 4 6;6 7 9 2;1 3 5 8]

U=triu(A)
L=tril(A)
d=diag(A)

U+L
```

Octave.

```
A=[3 5 1 4;8 2 4 6;6 7 9 2;1 3 5 8]

U=triu(A)
L=tril(A)
d=diag(A)

U+L
```

Python.

```
import numpy as np

A=np.array([[3,5,1,4],
            [8,2,4,6],
            [6,7,9,2],
            [1,3,5,8]])

U=np.triu(A)
L=np.tril(A)
d=np.diag(A)

U+L
```

Perl. PDL no tiene aquí una función equivalente a `triu`/`tril`; las máscaras se construyen comparando las coordenadas de los elementos.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::MatrixOps;

my $A = pdl([[3,5,1,4],[8,2,4,6],[6,7,9,2],[1,3,5,8]]);

my $n = $A->dim(0);
my $i = sequence($n)->dummy(0,$n);
my $j = sequence($n)->dummy(1,$n);

my $U = $A * ($i <= $j);
my $L = $A * ($i >= $j);
my $d = $A->diagonal(0,1);

print "U = $U\n";
print "L = $L\n";
print "d = $d\n";
print "U+L = ", $U + $L, "\n";
```

JavaScript. Se usan máscaras numéricas para no introducir todavía condicionales.

```
const math = require('mathjs')
const A = [[3,5,1,4],
   [8,2,4,6],
   [6,7,9,2],
   [1,3,5,8]]

const MU = [[1,1,1,1],
    [0,1,1,1],
    [0,0,1,1],
    [0,0,0,1]]
```

```
const ML = [[1,0,0,0],
    [1,1,0,0],
    [1,1,1,0],
    [1,1,1,1]]

let U = math.dotMultiply(A,MU)
let L = math.dotMultiply(A,ML)
let d = math.diag(A)

math.add(U,L)
```

23. **Ejercicio:** Ordenamiento de datos

    Con $c = (1, 6, 6, 3, 0, 5, 3, 2, 5, 4)$

    - (a) ordénelo en forma decreciente;
    - (b) obtenga también los índices de ordenamiento;
    - (c) verifique que el vector ordenado sale de aplicar los índices al original;
    - (d) explique qué indica cada índice;
    - (e) use los índices para ver de qué posiciones salieron los tres primeros valores ordenados.

    Cada elemento del vector de índices indica la posición que ocupaba en el vector original el elemento correspondiente del vector ordenado. Por lo tanto, al aplicar esos índices a c debe recuperarse exactamente el vector ordenado.

    Scilab. `gsort` ordena en forma decreciente por defecto. Los índices comienzan en 1.

```
c=[1 6 6 3 0 5 3 2 5 4];

[y,k]=gsort(c)

y
c(k)

k(1:3)
c(k(1:3))
```

    Octave. `sort` permite pedir el orden decreciente con `"descend"`. Los índices comienzan en 1.

```
c=[1 6 6 3 0 5 3 2 5 4];

[y,k]=sort(c,"descend")

y
```

```
c(k)

k(1:3)
c(k(1:3))
```

Python. `argsort` devuelve los índices que producirían el ordenamiento. Los índices comienzan en 0.

```
import numpy as np

c=np.array([1,6,6,3,0,5,3,2,5,4])

k=np.argsort(-c)
y=c[k]

y
c[k]

k[:3]
c[k[:3]]
```

Perl. En PDL, `qsorti` devuelve los índices de ordenamiento. Los índices comienzan en 0.

```
use PDL;

my $c = pdl(1,6,6,3,0,5,3,2,5,4);
my $k = qsorti(-$c);
my $y = $c->index($k);

$y;
$c->index($k);

$k->slice("0:2");
$c->index($k->slice("0:2"));
```

JavaScript. Se ordenan primero los índices según los valores de `c`. Los índices comienzan en 0.

```
const c = [1,6,6,3,0,5,3,2,5,4]

let k = c.map((v,i)=>i).sort((a,b)=>c[b]-c[a])
let y = k.map(i=>c[i])

y
k.map(i=>c[i])

k.slice(0,3)
k.slice(0,3).map(i=>c[i])
```

24. **Ejercicio:** Resumen matricial

Con $A = \begin{pmatrix} 3 & 5 & 1 \\ 8 & 2 & 4 \\ 6 & 7 & 9 \end{pmatrix}$ calcule y muestre:

- (a) suma de todos los elementos;
- (b) promedio;
- (c) máximo y mínimo;
- (d) diagonal;
- (e) transpuesta;
- (f) suma de cada columna;
- (g) suma de cada fila. Indique qué función usó y explique cada resultado.

Scilab.

```
A=[3 5 1;8 2 4;6 7 9];
sum(A), mean(A), max(A), min(A), diag(A), A'
sum(A,"r")      // por columnas
sum(A,"c")      // por filas
```

Octave. `A(:)` convierte todos los elementos de la matriz en un único vector, permitiendo calcular suma, promedio, máximo y mínimo globales. `sum(A,1)` suma cada columna y `sum(A,2)` suma cada fila.

```
A=[3 5 1;8 2 4;6 7 9];

suma=sum(A(:))          % 45
promedio=mean(A(:))     % 5
maximo=max(A(:))        % 9
minimo=min(A(:))        % 1

diagonal=diag(A)        % [3;2;9]
transpuesta=A'

columnas=sum(A,1)       % [17 14 14]
filas=sum(A,2)          % [9;14;22]
```

Python.

```
import numpy as np

A=np.array([[3,5,1],[8,2,4],[6,7,9]])
A.sum(), A.mean(), A.max(), A.min(), np.diag(A), A.T
A.sum(axis=0)   # por columnas
A.sum(axis=1)   # por filas
```

Perl. `sumover` suma a lo largo de la primera dimensión (las columnas de cada fila).

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::MatrixOps;

my $A = pdl([[3,5,1],[8,2,4],[6,7,9]]);
print "A = $A\n";

print "sum = ", sum($A), "\n";
print "avg = ", avg($A), "\n";
print "max = ", max($A), "\n";
print "min = ", min($A), "\n";
print "diag = ", $A->diagonal(0,1), "\n";
print "transpose = ", $A->transpose, "\n";

print "suma por columnas = ", $A->transpose->sumover, "\n";
print "suma por filas = ", $A->sumover, "\n";
```

JavaScript.

```
const math = require('mathjs')
const A = [[3,5,1],[8,2,4],[6,7,9]]
math.sum(A); math.mean(A); math.max(A); math.min(A);
   math.diag(A); math.transpose(A)
math.sum(A,0)    // por columnas
math.sum(A,1)    // por filas
```

25. **Ejercicio:** Estructuras de mediciones

Represente una medición física como un registro con campos `magnitud`, `valor`, `unidad` e `instrumento` (por ejemplo temperatura, 23.5, C, termómetro).

- (a) Muestre magnitud, valor y unidad accediendo a los campos;
- (b) cambie el valor;
- (c) agregue un campo `fecha`;
- (d) escriba una función `mostrar_medicion(m)` que devuelva la cadena `temperatura = 23.5 C`;
- (e) cree tres mediciones (temperatura, presión, masa);
- (f) guárdelas en una lista y muestre cada una accediendo por posición.

Scilab.

```
m=struct();
m.magnitud="temperatura";
m.valor=23.5;
```

```
m.unidad="C";
m.instrumento="termometro";

m.magnitud
m.valor
m.unidad

m.valor=24.1;
m.fecha=date();

function s=mostrar_medicion(m)
    s=m.magnitud+" = "+string(m.valor)+" "+m.unidad;
endfunction

m1=struct("magnitud","temperatura","valor",23.5,...
          "unidad","C","instrumento","termometro");

m2=struct("magnitud","presion","valor",1013,...
          "unidad","hPa","instrumento","barometro");

m3=struct("magnitud","masa","valor",2.4,...
          "unidad","kg","instrumento","balanza");

mediciones=list(m1,m2,m3);

mostrar_medicion(mediciones(1))
mostrar_medicion(mediciones(2))
mostrar_medicion(mediciones(3))
```

Octave. La estructura y sus campos se manejan de forma similar; `num2str` convierte el valor numérico a cadena antes de concatenar.

```
m=struct();
m.magnitud="temperatura";
m.valor=23.5;
m.unidad="C";
m.instrumento="termometro";

m.magnitud
m.valor
m.unidad

m.valor=24.1;
m.fecha=date();

function s=mostrar_medicion(m)
    s=[m.magnitud " = " num2str(m.valor) " " m.unidad];
endfunction

m1=struct("magnitud","temperatura","valor",23.5,...
```

```
          "unidad","C","instrumento","termometro");

m2=struct("magnitud","presion","valor",1013,...
          "unidad","hPa","instrumento","barometro");

m3=struct("magnitud","masa","valor",2.4,...
          "unidad","kg","instrumento","balanza");

mediciones=[m1,m2,m3];

mostrar_medicion(mediciones(1))
mostrar_medicion(mediciones(2))
mostrar_medicion(mediciones(3))
```

Python. El registro se representa con un diccionario.

```
import datetime

m={"magnitud":"temperatura",
   "valor":23.5,
   "unidad":"C",
   "instrumento":"termometro"}

m["magnitud"]
m["valor"]
m["unidad"]

m["valor"]=24.1
m["fecha"]=str(datetime.date.today())

def mostrar_medicion(m):
    return m["magnitud"]+" = "+str(m["valor"])+"
       "+m["unidad"]

m1={"magnitud":"temperatura",
    "valor":23.5,
    "unidad":"C",
    "instrumento":"termometro"}

m2={"magnitud":"presion",
    "valor":1013,
    "unidad":"hPa",
    "instrumento":"barometro"}

m3={"magnitud":"masa",
    "valor":2.4,
    "unidad":"kg",
    "instrumento":"balanza"}

mediciones=[m1,m2,m3]
```

```
mostrar_medicion(mediciones[0])
mostrar_medicion(mediciones[1])
mostrar_medicion(mediciones[2])
```

Perl. El registro se representa con un hash; la función recibe una referencia al hash.

```
#!/usr/bin/env perl
use strict;
use warnings;

# Hash que representa una medicion
my %m;
$m{magnitud}    = "temperatura";
$m{valor}       = 23.5;
$m{unidad}      = "C";
$m{instrumento} = "termometro";

print "magnitud = $m{magnitud}\n";
print "valor = $m{valor}\n";
print "unidad = $m{unidad}\n";

# Modificacion
$m{valor} = 24.1;
use POSIX qw(strftime);
$m{fecha} = strftime("%Y-%m-%d", localtime);

# Funcion para mostrar una medicion
sub mostrar_medicion {
    my $med = shift;
    return $med->{magnitud} . " = " . $med->{valor} . " " .
       $med->{unidad};
}

# Lista de mediciones
my %m1;
$m1{magnitud}    = "temperatura";
$m1{valor}       = 23.5;
$m1{unidad}      = "C";
$m1{instrumento} = "termometro";

my %m2;
$m2{magnitud}    = "presion";
$m2{valor}       = 1013;
$m2{unidad}      = "hPa";
$m2{instrumento} = "barometro";

my %m3;
$m3{magnitud}    = "masa";
$m3{valor}       = 2.4;
```

```
$m3{unidad}      = "kg";
$m3{instrumento} = "balanza";

my @mediciones = (\%m1, \%m2, \%m3);
print mostrar_medicion($mediciones[0]), "\n";
print mostrar_medicion($mediciones[1]), "\n";
print mostrar_medicion($mediciones[2]), "\n";
```

JavaScript. El registro se representa con un objeto literal.

```
const m = {magnitud:"temperatura",
   valor:23.5,
   unidad:"C",
   instrumento:"termometro"}

m.magnitud
m.valor
m.unidad

m.valor=24.1
m.fecha=new Date().toLocaleDateString()

function mostrar_medicion(m){
    return m.magnitud+" = "+m.valor+" "+m.unidad
}

const m1 = {magnitud:"temperatura",
    valor:23.5,
    unidad:"C",
    instrumento:"termometro"}

const m2 = {magnitud:"presion",
    valor:1013,
    unidad:"hPa",
    instrumento:"barometro"}

const m3 = {magnitud:"masa",
    valor:2.4,
    unidad:"kg",
    instrumento:"balanza"}

const mediciones = [m1,m2,m3]

mostrar_medicion(mediciones[0])
mostrar_medicion(mediciones[1])
mostrar_medicion(mediciones[2])
```

26. **Ejercicio:** Contenedores heterogéneos

    Guarde nombre, edad y notas de una persona en:

- (a) un contenedor accedido por posición;
- (b) un contenedor de celdas, si el lenguaje dispone de él;
- (c) un contenedor con campos nombrados y nombre de tipo, si existe;
- (d) un contenedor con campos nombrados al que se le puede definir comportamiento propio;
- (e) compare: ¿cuándo conviene acceso por posición y cuándo por nombre?

El acceso por posición es compacto y conveniente cuando el orden de los datos es conocido y fijo. El acceso por nombre es más legible y reduce errores cuando cada dato tiene un significado diferente.

Scilab. Scilab distingue explícitamente `list`, `cell`, `tlist` y `mlist`.

```
L=list("Ana",20,[8 9 7])
L(1), L(2), L(3)

C=cell(3,1)
C{1}="Ana"
C{2}=20
C{3}=[8 9 7]
C{1}, C{2}, C{3}

p=tlist(["persona" "nombre" "edad" "notas"],...
        "Ana",20,[8 9 7])
p.nombre
p.edad
p.notas

q=mlist(["persona" "nombre" "edad" "notas"],...
        "Ana",20,[8 9 7])
q.nombre
q.edad
q.notas
```

Octave. El arreglo de celdas sirve como contenedor heterogéneo por posición. La estructura proporciona campos con nombre. Para definir un tipo propio con comportamiento se necesita una clase, que se verá más adelante.

```
L={"Ana",20,[8 9 7]}
L{1}, L{2}, L{3}

p=struct("nombre","Ana",...
         "edad",20,...
         "notas",[8 9 7])

p.nombre
p.edad
p.notas
```

```
% Un tipo propio con comportamiento requiere classdef.
```

Python. La lista proporciona acceso por posición. Python no tiene un tipo "celda" distinto. `namedtuple` permite crear un tipo con nombre y campos nombrados. Para agregar comportamiento propio se necesita definir una clase, que se verá más adelante.

```
L=["Ana",20,[8,9,7]]
L[0]
L[1]
L[2]

from collections import namedtuple

Persona=namedtuple("Persona",["nombre","edad","notas"])
p=Persona("Ana",20,[8,9,7])

p.nombre
p.edad
p.notas

# Comportamiento propio requiere definir una clase.
```

Perl. La lista proporciona acceso por posición. Perl no tiene un contenedor de celdas distinto. Un hash permite campos nombrados y `bless` le asigna un nombre de tipo. Para darle comportamiento propio se necesita un paquete con métodos, que se verá más adelante.

```
#!/usr/bin/env perl
use strict;
use warnings;

# Lista heterogenea (array de Perl)
my @L = ("Ana", 20, [8, 9, 7]);
print "nombre = $L[0]\n";
print "edad = $L[1]\n";
print "notas = ", join(" ", @{$L[2]}), "\n";

# Registro con campos nombrados (hash de Perl)
my %persona;
$persona{nombre} = "Ana";
$persona{edad}   = 20;
$persona{notas}  = [8, 9, 7];
print "ref = ", ref(\%persona), "\n";
print "nombre = $persona{nombre}\n";
print "edad = $persona{edad}\n";
print "notas = ", join(" ", @{$persona{notas}}), "\n";

# Comportamiento propio: package con metodos
```

```perl
package Persona;
sub new {
    my ($class, %args) = @_;
    return bless \%args, $class;
}
sub nombre { return shift->{nombre}; }
sub edad   { return shift->{edad}; }
sub notas  { return shift->{notas}; }

package main;
my $pers = Persona->new(nombre => "Ana", edad => 20, notas
   => [8, 9, 7]);
print "p->nombre = ", $pers->nombre, "\n";
print "p->edad = ", $pers->edad, "\n";
print "p->notas = ", join(" ", @{$pers->notas}), "\n";
```

JavaScript. El arreglo proporciona acceso por posición y el objeto campos con nombre. JavaScript no tiene un contenedor de celdas distinto. Para crear un tipo propio con comportamiento se necesita un constructor o una clase, que se verá más adelante.

```javascript
// Acceso por posicion
const L = ["Ana", 20, [8, 9, 7]]
L[0]
L[1]
L[2]

// Campos nombrados con objeto literal
const p = {
    nombre: "Ana",
    edad: 20,
    notas: [8, 9, 7]
}

p.nombre
p.edad
p.notas

// Tipo propio con comportamiento requiere definir una clase.
```

27. **Ejercicio:** Diccionarios de datos

Cree una tabla de asociación clave–valor `datos` con `temperatura`, `presion` y `masa`; asígneles números; acceda por clave; acceda con la clave guardada en una variable; agregue en ejecución la clave `energia`; calcule una distancia con `velocidad` y `tiempo` agregadas del mismo modo; pruebe una clave con espacios.

JavaScript. El objeto o `Map` funciona como diccionario.

```javascript
const datos={};
```

```
datos["temperatura"]=23.5; datos["presion"]=1013;
   datos["masa"]=2.4;
datos["temperatura"]
let clave="temperatura"; datos[clave]
clave="energia"; datos[clave]=1500
datos["velocidad"]=12.8; datos["tiempo"]=5;
const distancia=datos["velocidad"]*datos["tiempo"]
datos["temperatura media"]=25
```

Scilab. No hay diccionario nativo; `struct` lo simula, pero la clave con espacios no se puede (ver el ejercicio siguiente).

```
datos=struct();
datos.temperatura=23.5; datos.presion=1013; datos.masa=2.4;
datos.temperatura
clave="temperatura"; datos(clave)
clave="energia"; datos(clave)=1500
datos("velocidad")=12.8; datos("tiempo")=5;
distancia=datos("velocidad")*datos("tiempo")
datos("temperatura media")=25       // No se puede: error
```

Octave. Una estructura funciona como diccionario; los campos dinámicos se escriben entre paréntesis.

```
datos=struct();
datos.temperatura=23.5; datos.presion=1013; datos.masa=2.4;
datos.temperatura
clave="temperatura"; datos.(clave)
clave="energia"; datos.(clave)=1500;
datos.("velocidad")=12.8; datos.("tiempo")=5;
distancia=datos.("velocidad")*datos.("tiempo")
datos.("temperatura media")=25
```

Python. El diccionario es un tipo del lenguaje.

```
datos={}
datos["temperatura"]=23.5; datos["presion"]=1013;
   datos["masa"]=2.4
datos["temperatura"]
clave="temperatura"; datos[clave]
clave="energia"; datos[clave]=1500
datos["velocidad"]=12.8; datos["tiempo"]=5
distancia=datos["velocidad"]*datos["tiempo"]
datos["temperatura media"]=25
```

Perl. El hash es el diccionario.

```
#!/usr/bin/env perl
```

```
use strict;
use warnings;

my %datos;
$datos{temperatura} = 23.5;
$datos{presion}     = 1013;
$datos{masa}        = 2.4;

print "temperatura = $datos{temperatura}\n";

my $clave = "temperatura";
print "$clave = $datos{$clave}\n";

$clave = "energia";
$datos{$clave} = 1500;
print "$clave = $datos{$clave}\n";

$datos{velocidad} = 12.8;
$datos{tiempo}    = 5;
my $distancia = $datos{velocidad} * $datos{tiempo};
print "distancia = $distancia\n";

$datos{"temperatura media"} = 25;
print "temperatura media = $datos{'temperatura media'}\n";
```

28. **Ejercicio:** Vectores paralelos

    Use dos vectores paralelos, uno de claves de texto y otro de valores:

    - (a) créelos con temperatura, presión y masa;
    - (b) busque la posición de `presion` y obtenga su valor;
    - (c) agregue `velocidad`;
    - (d) modifique el valor de `temperatura`;
    - (e) repita con una clave con espacios;
    - (f) compare con la estructura o el diccionario: qué gana y qué pierde cada forma.

    Los vectores paralelos permiten representar una asociación clave–valor usando estructuras muy simples. Para buscar una clave hay que encontrar primero su posición en el vector de claves y usar luego esa misma posición en el vector de valores. Por eso ambos vectores deben mantenerse siempre sincronizados.

    Una estructura o diccionario permite, en cambio, acceder directamente al valor mediante su clave, por lo que resulta más legible y reduce el riesgo de descoordinar claves y valores. Los vectores paralelos pueden ser útiles cuando se quiere trabajar explícitamente con los dos vectores o cuando el contenedor disponible impone restricciones a las claves. Por ejemplo, en Scilab permiten usar claves con espacios, que no pueden emplearse como nombres de campos de una estructura.

    Scilab. Es la forma de tener claves con espacios.

```
claves=["temperatura","presion","masa"];
   valores=[23.5,1013,2.4];
clave="presion"; i=find(claves==clave); valor=valores(i)
claves($+1)="velocidad"; valores($+1)=12.8;
i=find(claves=="temperatura"); valores(i)=24.0;
claves($+1)="temperatura media"; valores($+1)=25;
```

Octave. Las claves se guardan en un arreglo de celdas y se comparan con `strcmp`.

```
claves={"temperatura","presion","masa"};
   valores=[23.5,1013,2.4];
clave="presion"; i=find(strcmp(claves,clave));
   valor=valores(i)
claves{end+1}="velocidad"; valores(end+1)=12.8;
i=find(strcmp(claves,"temperatura")); valores(i)=24.0;
claves{end+1}="temperatura media"; valores(end+1)=25;
```

Python. Innecesario porque existe `dict`; `np.append` devuelve un arreglo nuevo.

```
import numpy as np

claves=np.array(["temperatura","presion","masa"])
valores=np.array([23.5,1013,2.4])

clave="presion"
i=np.where(claves==clave)[0][0]
valor=valores[i]

claves=np.append(claves,"velocidad")
valores=np.append(valores,12.8)

i=np.where(claves=="temperatura")[0][0]
valores[i]=24.0

claves=np.append(claves,"temperatura media")
valores=np.append(valores,25)

dict(zip(claves,valores))
```

Perl. Cadenas se comparan con `eq`, no con `==`.

```
@claves=("temperatura","presion","masa");
   @valores=(23.5,1013,2.4);
push @claves,"velocidad"; push @valores,12.8;
($i)=grep { $claves[$_] eq "temperatura" } 0..$#claves;
   $valores[$i]=24.0;
push @claves,"temperatura media"; push @valores,25;
```

```
@h{@claves}=@valores;                    # pasar a hash
```

JavaScript.

```
const claves = ["temperatura","presion","masa"]; const
   valores = [23.5,1013,2.4]
let clave = "presion"; let i = claves.indexOf(clave); let
   valor = valores[i]
claves.push("velocidad"); valores.push(12.8)
i=claves.indexOf("temperatura"); valores[i]=24.0
claves.push("temperatura media"); valores.push(25)
Object.fromEntries(claves.map((k,i)=>[k,valores[i]]))   //
   pasar a objeto
```

4.23.4. Evaluación de cadenas

29. **Ejercicio:** Expresiones dinámicas

- (a) Evalúe la cadena `3 + 4*5`;
- (b) evalúe una cadena que use una variable ya definida ($x = 10$, $x^2 + 3x + 1$);
- (c) evalúe una cadena que use funciones matemáticas ($\sin(\pi/2) + \sqrt{9}$);
- (d) ejecute una instrucción construida como texto para crear una variable cuyo nombre está en una cadena;
- (e) compare la evaluación de expresiones con la ejecución de instrucciones: cuál devuelve un valor y cuál no;
- (f) explique por qué para guardar datos con nombre es más claro un registro o diccionario que crear variables por texto, y por qué evaluar texto que viene de afuera es peligroso.

Evaluar una expresión significa calcular su valor. Por ejemplo, `3+4*5` produce el valor 23. Ejecutar una instrucción significa producir un efecto, por ejemplo crear o modificar una variable.

Algunos lenguajes separan ambas operaciones en funciones distintas: Scilab usa `evstr` para evaluar y `execstr` para ejecutar; Python usa `eval` y `exec`. Otros emplean una misma función según el texto recibido, como Octave, Perl y JavaScript.

Para guardar datos identificados por un nombre es preferible una estructura o diccionario. La clave forma parte explícita del contenedor y no obliga a construir nombres de variables mediante cadenas.

Evaluar texto recibido desde afuera es peligroso porque ese texto puede contener instrucciones distintas de las esperadas y modificar datos, archivos o el estado del programa. Por eso estas funciones deben usarse solamente con texto cuyo contenido se conoce y controla.

Scilab. Dos funciones: `evstr` evalúa, `execstr` ejecuta.

```
expr="3 + 4*5"; r=evstr(expr)
x=10; expr="x^2 + 3*x + 1"; r=evstr(expr)
expr="sin(%pi/2) + sqrt(9)"; r=evstr(expr)
nombre="temperatura"; valor=23.5; execstr(nombre+" =
   valor"); temperatura
```

Octave. `eval` evalúa expresiones o asignaciones construidas como texto.

```
expr="3 + 4*5"; r=eval(expr)
x=10; expr="x^2 + 3*x + 1"; r=eval(expr)
expr="sin(pi/2) + sqrt(9)"; r=eval(expr)
nombre="temperatura"; valor=23.5;
eval([nombre " = valor;"]); temperatura
```

Python. `eval` evalúa, `exec` ejecuta.

```
import numpy as np

expr="3 + 4*5"; r=eval(expr)
x=10; expr="x**2 + 3*x + 1"; r=eval(expr)
expr="np.sin(np.pi/2) + np.sqrt(9)"; r=eval(expr)
nombre="temperatura"; valor=23.5; exec(nombre+" = valor");
   temperatura
```

Perl. Un solo `eval` hace las dos cosas y devuelve el último valor.

```
use Math::Trig; $expr="sin(pi/2) + sqrt(9)"; $r=eval $expr
```

JavaScript. `eval` puede evaluar expresiones y ejecutar instrucciones. `math.evaluate` evalúa expresiones escritas con la sintaxis de math.js.

```
const math = require('mathjs')
let expr = "3 + 4*5"; let r = eval(expr)
let x = 10; expr="x**2 + 3*x + 1"; r=eval(expr)
expr="Math.sin(Math.PI/2) + Math.sqrt(9)"; r=eval(expr)
r=math.evaluate("sin(pi/2) + sqrt(9)")
let nombre = "temperatura"; let valor = 23.5;
   globalThis[nombre]=valor; temperatura
```

4.23.5. Lógica y máscaras

30. **Ejercicio:** Máscaras de selección

    Construya la matriz 8×8 con los enteros 1 a 64 llenada por columnas. Sin ciclos y partiendo de la matriz original en cada inciso:

- (a) reemplace por -1 los múltiplos de 3;
- (b) obtenga un vector con los elementos mayores que 40.

Scilab. Una comparación aplicada a una matriz produce una máscara lógica. Esa máscara permite seleccionar o modificar elementos.

```
A=matrix(1:64,8,8)
B=A
B(modulo(B,3)==0)=-1
v=A(A>40)
```

Octave. La comparación produce una máscara lógica. Se modifica una copia para conservar la matriz original.

```
A=reshape(1:64,8,8);
B=A;
B(mod(B,3)==0)=-1
v=A(A>40)
```

Python. Se modifica una copia para conservar la matriz original.

```
import numpy as np
A=np.arange(1,65).reshape(8,8,order='F')
B=A.copy()
B[B%3==0]=-1
v=A[A>40]
```

Perl. `where` permite seleccionar elementos para consultarlos o modificarlos en el lugar.

```
```

JavaScript. Las comparaciones producen valores lógicos que en operaciones aritméticas actúan como 1 para verdadero y 0 para falso. Así se puede modificar la matriz sin usar una estructura condicional.

```
const math = require('mathjs')
let A = math.transpose(math.reshape(math.range(1,65),[8,8]))

let B = A.map(v => v*(v%3!=0) - (v%3==0))

let v = math.filter(math.flatten(A),v => v>40)
```

31. **Ejercicio:** Umbral y filtrado

    Para una matriz aleatoria $m \times n$:

    - (a) cuente los elementos mayores que 0,72 y guárdelos en un vector;

- (b) obtenga los comprendidos entre 0,25 y 0,75.

Scilab.

```
A=rand(4,5);
k=find(A>0.72); length(k)
v=A(A>0.72)
A(A>0.25 & A<0.75)
```

Octave.

```
A=rand(4,5);
k=find(A>0.72); length(k)
v=A(A>0.72)
A(A>0.25 & A<0.75)
```

Python. Los paréntesis alrededor de cada comparación son obligatorios.

```
import numpy as np

A=np.random.rand(4,5)
np.sum(A>0.72)
v=A[A>0.72]
A[(A>0.25)&(A<0.75)]
```

Perl.

```
use PDL;

my $A = random(4,5);

which($A>0.72)->nelem;
$A->where(($A>0.25)&($A<0.75));
```

JavaScript.

```
const math = require('mathjs')
let A = math.random([4,5])
let v = math.filter(math.flatten(A), x=>x>0.72); v.length
math.filter(math.flatten(A), x=>x>0.25 && x<0.75)
```

32. **Ejercicio:** Operadores lógicos

Con x de 200 puntos en $[-3, 3]$ e $y = x^3 - 3x$:

- (a) ¿en cuántos puntos es $y > 0$? (sume la máscara);
- (b) extraiga los x donde $-1 < y < 1$ por indexación lógica;

- (c) ¿hay algún punto con $y > 2$ y $x < 0$ a la vez?;
- (d) reemplace por cero los y negativos.

Scilab.

```
x=linspace(-3,3,200); y=x.^3-3*x;
sum(y>0)
x(abs(y)<1)
or((y>2)&(x<0))
y(y<0)=0
```

Octave.

```
x=linspace(-3,3,200); y=x.^3-3*x;
sum(y>0)
x(abs(y)<1)
any((y>2)&(x<0))
y(y<0)=0;
```

Python. `&` con paréntesis, nunca `and`.

```
import numpy as np

x=np.linspace(-3,3,200); y=x**3-3*x
np.sum(y>0)
x[np.abs(y)<1]
np.any((y>2)&(x<0))
y[y<0]=0
```

Perl.

```
use PDL;

my $x = zeroes(200)->xlinvals(-3,3);
my $y = $x**3 - 3*$x;

sum($y>0);
$x->where(abs($y)<1);
any(($y>2)&($x<0));
```

JavaScript. Las comparaciones producen valores lógicos. El signo `+` delante de una comparación convierte `true` en 1 y `false` en 0, permitiendo sumar la máscara.

```
const math = require('mathjs')
let x = math.range(-3,3,6/199,true).toArray()
let y = x.map(v=>v**3-3*v)

math.sum(y.map(v=>+(v>0)))
```

```
x.filter((v,i)=>Math.abs(y[i])<1)

y.some((v,i)=>v>2 && x[i]<0)

y=y.map(v=>v*(v>=0))
```

33. **Ejercicio:** Funciones definidas por tramos

Con x de 1001 puntos en $[-5,5]$, evalúe:

$$f(x)=\begin{cases}-x^2, & x<-1,\\ 2x+1, & -1\le x\le 2,\\ \sqrt{x-2}, & x>2,\end{cases}$$

mediante máscaras lógicas, y explique qué condición incluye cada punto de frontera.

Las tres máscaras no se superponen. El punto $x=-1$ pertenece al segundo tramo porque éste usa $x\ge -1$, y el punto $x=2$ también pertenece al segundo tramo porque usa $x\le 2$. El tercer tramo comienza solamente para $x>2$.

Scilab.

```
x=linspace(-5,5,1001); f=zeros(x);
m1=x<-1; m2=(x>=-1)&(x<=2); m3=x>2;
f(m1)=-x(m1).^2; f(m2)=2*x(m2)+1; f(m3)=sqrt(x(m3)-2);
```

Octave. Cada expresión se evalúa solamente sobre los elementos de su tramo. Los puntos -1 y 2 pertenecen al segundo tramo.

```
x=linspace(-5,5,1001); f=zeros(size(x));
m1=x<-1; m2=(x>=-1)&(x<=2); m3=x>2;
f(m1)=-x(m1).^2;
f(m2)=2*x(m2)+1;
f(m3)=sqrt(x(m3)-2);
```

Python. `np.sqrt(x-2)` sobre todo el vector da `nan` y aviso: evaluar sólo en el tramo, o `np.piecewise`.

```
import numpy as np

x=np.linspace(-5,5,1001); f=np.empty_like(x)
m1=x<-1; m2=(x>=-1)&(x<=2); m3=x>2
f[m1]=-x[m1]**2; f[m2]=2*x[m2]+1; f[m3]=np.sqrt(x[m3]-2)
# o:  f=np.piecewise(x,[m1,m2,m3],[lambda v:-v**2, lambda
   v:2*v+1, lambda v:np.sqrt(v-2)])
```

Perl.

JavaScript. Las comparaciones se convierten en 1 o 0 mediante el signo +. Así cada expresión contribuye solamente en su tramo. `Math.max` evita calcular una raíz de un número negativo.

```
const math = require('mathjs')
let x = math.range(-5,5.005,0.01).toArray()

let m1 = x.map(v=>+(v<-1))
let m2 = x.map(v=>+(v>=-1 && v<=2))
let m3 = x.map(v=>+(v>2))

let f = x.map((v,i)=>
    -v*v*m1[i]
    +(2*v+1)*m2[i]
    +Math.sqrt(Math.max(v-2,0))*m3[i]
)
```

4.23.6. Ecuaciones lineales

34. **Ejercicio:** Sistemas lineales

Resuelva

$$\begin{cases} 2x + 3y - z = 1 \\ x - y + 2z = 8 \\ 3x + y + z = 6 \end{cases}$$

- (a) escríbalo como $A\mathbf{x} = \mathbf{b}$;
- (b) resuélvalo con la función de resolución directa (más estable que invertir);
- (c) resuélvalo invirtiendo la matriz y compare;
- (d) verifique $A\mathbf{x} - \mathbf{b} = 0$;
- (e) calcule el número de condición: ¿está bien condicionado?;
- (f) perturbe $\mathbf{b}$ un 1 % y resuelva de nuevo: ¿cuánto cambió?

La solución es

$$\mathbf{x} = (-1, 3, 6)^T.$$

El número de condición mide cuánto puede amplificarse una perturbación relativa de los datos al calcular la solución. Un valor próximo a 1 indica un sistema muy bien condicionado; valores grandes indican mayor sensibilidad a errores o perturbaciones.

En este caso el sistema no está fuertemente mal condicionado.

Para la perturbación propuesta se toma

$$\mathbf{b}_2 = 1{,}01\,\mathbf{b}.$$

Como solamente se ha multiplicado todo el segundo miembro por 1,01, la linealidad del sistema hace que

$$\mathbf{x}_2 = 1{,}01\,\mathbf{x}.$$

Por lo tanto, en esta perturbación particular la solución cambia exactamente un 1 %.

Scilab. El operador `A\b` resuelve directamente el sistema. `linsolve(A,-b)` usa la forma $Ax + b = 0$.

```
A=[2 3 -1;1 -1 2;3 1 1]
b=[1;8;6]

x=A\b

x2=inv(A)*b
x2-x

A*x-b

cond(A)

b2=1.01*b
x3=A\b2

x3-x
norm(x3-x)/norm(x)
```

Octave. La barra invertida resuelve directamente el sistema sin calcular la inversa.

```
A=[2 3 -1;1 -1 2;3 1 1]
b=[1;8;6]

x=A\b

x2=inv(A)*b
x2-x

A*x-b

cond(A)

b2=1.01*b
x3=A\b2

x3-x
norm(x3-x)/norm(x)
```

Python. `solve` resuelve directamente el sistema; `inv` calcula explícitamente la inversa.

```
import numpy as np

A=np.array([[2,3,-1],
            [1,-1,2],
            [3,1,1]])
b=np.array([1,8,6])

x=np.linalg.solve(A,b)

x2=np.linalg.inv(A)@b
x2-x

A@x-b

np.linalg.cond(A)

b2=1.01*b
x3=np.linalg.solve(A,b2)

x3-x
np.linalg.norm(x3-x)/np.linalg.norm(x)
```

Perl. PDL::MatrixOps permite resolver el sistema mediante descomposición LU. Como no se dispone aquí de una función directa para el número de condición, se calcula el producto de las normas de la matriz y de su inversa.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::MatrixOps;

my $A = pdl([[2,3,-1],[1,-1,2],[3,1,1]]);
my $b = pdl([1,8,6]);

my $b_col = $b->dummy(0,1);
my $x = $A->inv x $b_col;
print "x = $x\n";

my $x2 = $A->inv x $b_col;
print "x2 - x = ", ($x2 - $x)->flat, "\n";

print "A x - b = ", ($A x $x - $b_col)->flat, "\n";

my $condA = sqrt(sum($A**2)) * sqrt(sum($A->inv**2));
print "cond(A) = $condA\n";

my $b2 = 1.01 * $b;
my $b2_col = $b2->dummy(0,1);
my $x3 = $A->inv x $b2_col;
```

```
print "x3 - x = ", ($x3 - $x)->flat, "\n";
my $rel_err = sqrt(sum(($x3 - $x)**2)) / sqrt(sum($x**2));
print "||x3-x||/||x|| = $rel_err\n";
```

JavaScript. `math.lusolve` resuelve el sistema directamente. El producto de las normas de la matriz y de su inversa permite estimar su condicionamiento.

```
const math = require('mathjs')
const A = [[2,3,-1],
   [1,-1,2],
   [3,1,1]]
const b = [1,8,6]

let x = math.lusolve(A,b)

let x2 = math.multiply(math.inv(A),b)
math.subtract(x2,x)

math.subtract(math.multiply(A,x),math.transpose([b]))

math.norm(A,'fro')*math.norm(math.inv(A),'fro')

let b2 = math.multiply(b,1.01)
let x3 = math.lusolve(A,b2)

math.subtract(x3,x)
math.norm(math.subtract(x3,x),'fro')/math.norm(x,'fro')
```

35. **Ejercicio:** Combinaciones lineales

Con $v_1 = (1, 2, 0)$, $v_2 = (0, 1, 1)$, $v_3 = (2, 0, 1)$, determine si $b = (5, 5, 3)$ es combinación lineal de ellos:

- (a) arme la matriz con los vectores como columnas;
- (b) resuelva $Ac = b$;
- (c) verifique $Ac - b$.

Scilab.

```
A=[1 0 2;2 1 0;0 1 1]; b=[5;5;3];
c=A\b
A*c-b
```

Octave.

```
A=[1 0 2;2 1 0;0 1 1]; b=[5;5;3];
c=A\b
A*c-b
```

Python.

```
import numpy as np

A=np.array([[1,0,2],[2,1,0],[0,1,1]]); b=np.array([5,5,3])
c=np.linalg.solve(A,b)
A@c-b
```

Perl.

```
$A x $c - $b
```

JavaScript.

```
const math = require('mathjs')
const A = [[1,0,2],[2,1,0],[0,1,1]]; const b = [5,5,3]
let c = math.lusolve(A,b)
math.subtract(math.multiply(A,c),math.transpose([b]))
```

36. **Ejercicio:** Cuatro incógnitas

Resuelva

$$\begin{aligned} x + 2y + 3z - w &= -1, \\ 2x - 2y - 6w &= 2, \\ 2x - y - z + 3w &= 1, \\ x + y + 2z - 4w &= -3. \end{aligned}$$

- (a) escriba la forma matricial;
- (b) resuelva el sistema mediante una función de resolución directa;
- (c) resuélvalo mediante otra forma disponible;
- (d) verifique la solución calculando $A\mathbf{x} - \mathbf{b}$;
- (e) calcule el número de condición.

La solución obtenida por los distintos métodos debe coincidir, salvo pequeños errores de redondeo. La verificación $A\mathbf{x} - \mathbf{b}$ debe producir el vector nulo o valores numéricamente muy próximos a cero.

El número de condición depende de la norma empleada. Scilab, Octave y NumPy disponen de una función específica. En Perl y JavaScript se calcula aquí

$$\|A\| \, \|A^{-1}\|,$$

usando la norma de Frobenius.

Scilab. `A\b` resuelve directamente el sistema. `linsolve(A,-b)` usa la forma $Ax + b = 0$.

```
A=[1 2 3 -1;2 -2 0 -6;2 -1 -1 3;1 1 2 -4]
b=[-1;2;1;-3]

x=A\b

x2=linsolve(A,-b)
x2-x

A*x-b

cond(A)
```

Octave. Como segunda forma se muestra la multiplicación por la inversa.

```
A=[1 2 3 -1;2 -2 0 -6;2 -1 -1 3;1 1 2 -4]
b=[-1;2;1;-3]

x=A\b

x2=inv(A)*b
x2-x

A*x-b

cond(A)
```

Python. `solve` resuelve directamente el sistema. `lstsq` proporciona otra forma de obtener la solución.

```
import numpy as np

A=np.array([[1,2,3,-1],
            [2,-2,0,-6],
            [2,-1,-1,3],
            [1,1,2,-4]])
b=np.array([-1,2,1,-3])

x=np.linalg.solve(A,b)

x2=np.linalg.lstsq(A,b,rcond=None)[0]
x2-x

A@x-b

np.linalg.cond(A)
```

Perl. PDL::MatrixOps permite resolver mediante descomposición LU. Como aquí no se usa una función específica de número de condición, se calcula $\|A\|_F\|A^{-1}\|_F$.

```perl
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::MatrixOps;

my $A = pdl([[1,2,3,-1],[2,-2,0,-6],[2,-1,-1,3],[1,1,2,-4]]);
my $b = pdl([-1,2,1,-3]);

# Solucion del sistema A x = b
my $b_col = $b->dummy(0,1);
my $x = $A->inv x $b_col;
print "x = $x\n";

# Minimos cuadrados (formula normal)
my $x2 = ($A->transpose x $A)->inv x $A->transpose x $b_col;
print "x2 - x = ", ($x2 - $x)->flat, "\n";

# Residuo
print "A x - b = ", ($A x $x - $b_col)->flat, "\n";

# Numero de condicion
my $condA = sqrt(sum($A**2)) * sqrt(sum($A->inv**2));
print "cond(A) = $condA\n";
```

JavaScript. `math.lusolve` resuelve directamente el sistema. Como segunda forma se usa la inversa. Para el condicionamiento se calcula $\|A\|_F \|A^{-1}\|_F$.

```javascript
const math = require('mathjs')
const A = [[1,2,3,-1],
   [2,-2,0,-6],
   [2,-1,-1,3],
   [1,1,2,-4]]
const b = [-1,2,1,-3]

let x = math.lusolve(A,b)

let x2 = math.multiply(math.inv(A),b)
math.subtract(x2,x)

math.subtract(math.multiply(A,x),math.transpose([b]))

math.norm(A,"fro")*math.norm(math.inv(A),"fro")
```

4.23.7. Ecuaciones no lineales

37. **Ejercicio:** Ecuación no lineal

- (a) Defina $f(x) = \sin x - x/2$;
- (b) busque una raíz partiendo de un valor inicial cercano a 2;
- (c) verifique calculando f en el resultado;
- (d) repita con otro valor inicial y compare;
- (e) cuando el resolvedor lo proporcione, obtenga el indicador de convergencia; en caso contrario, compruebe la solución mediante el residuo $f(x)$.

Los valores iniciales próximos a 2 y -2 conducen a las dos raíces no nulas simétricas. Una raíz calculada numéricamente se comprueba evaluando f en ella: el resultado debe ser próximo a cero.

Scilab. `fsolve` devuelve la raíz, el valor de la función y un indicador de convergencia.

```
deff("[y]=f(x)","y=sin(x)-x/2")

[x,fx,info]=fsolve(2,f)
x
f(x)
info

[x2,fx2,info2]=fsolve(-2,f)
x2
f(x2)
info2
```

Octave. `fsolve` recibe primero la función y luego el valor inicial. Un valor positivo de `info` indica convergencia.

```
f=@(x) sin(x)-x/2;

[x,fx,info]=fsolve(f,2)
x
f(x)
info

[x2,fx2,info2]=fsolve(f,-2)
x2
f(x2)
info2
```

Python. `fsolve` devuelve normalmente la raíz. Con `full_output=True` devuelve además información sobre la convergencia; `ier=1` indica que encontró una solución.

```
import numpy as np

from scipy import optimize
```

```
def f(x):
    return np.sin(x)-x/2

x=optimize.fsolve(f,2)
x
f(x)

x2=optimize.fsolve(f,-2)
x2
f(x2)

x,infod,ier,msg=optimize.fsolve(f,2,full_output=True)
ier
```

Perl. `PDL::GSL::MROOT` proporciona `gslmroot_fsolver`. Esta interfaz devuelve la raíz pero no un indicador de convergencia separado, por lo que aquí se comprueba el resultado evaluando el residuo.

```
use PDL;
use PDL::GSL::MROOT;

sub f {
    my $x=shift;
    return sin($x)-$x/2;
}

$x;
f($x);

$x2;
f($x2);
```

JavaScript. math.js no dispone de un resolvedor numérico equivalente a `fsolve`. La resolución numérica de esta ecuación se posterga hasta introducir métodos iterativos.

```
const math = require('mathjs')
// No se puede resolver numericamente con math.js solo
let f = x=>Math.sin(x)-x/2

f(2)
f(-2)
```

4.23.8. Polinomios

Scilab. El polinomio es un objeto: `poly(v,"x","coeff")` con coeficientes en orden creciente, `poly(r,"x","roots")` desde las raíces; `roots`, `horner`, `coeff`, `derivat`; se opera con `+` y `*`.

Octave. El polinomio se representa mediante un vector de coeficientes de mayor a menor grado. Se usan `roots`, `poly`, `polyval`, `polyder` y `conv`.

Python. El polinomio es su vector de coeficientes de mayor a menor grado: `np.roots`, `np.poly`, `np.polyval`, `np.polyder`, `np.polymul`. Alternativa: `numpy.polynomial.Polynomial`, con coeficientes crecientes como Scilab, que admite `p*q`, `p.roots()`, `p.deriv()`, `p(x)`.

Perl. No hay objeto polinomio; `PDL::Math::polyroots(re,im)` da las raíces de coeficientes en orden decreciente; evaluar con Horner y multiplicar y derivar coeficientes se escriben con `map`/PDL.

JavaScript. `math.polynomialRoot` llega sólo hasta grado 3: para grados 4 y más. No se puede con math.js salvo con la matriz compañera y `math.eigs`; Horner con `reduce`; `math.derivative` y `math.simplify` trabajan el polinomio como expresión simbólica.

38. **Ejercicio:** Polinomios y coeficientes

    Dado $p(x) = x^4 - 5x^3 + 5x^2 + 5x - 6$:

    - (a) defínalo por sus coeficientes;
    - (b) halle las raíces;
    - (c) evalúe p en sus raíces y verifique que da aproximadamente cero;
    - (d) reconstruya el polinomio a partir de las raíces y compárelo con p.

    La comparación final puede dar diferencias numéricas muy pequeñas por redondeo, en lugar de ceros exactos.

    Scilab.

```
p=poly([-6 5 5 -5 1],"x","coeff")
r=roots(p)

horner(p,r)

p2=poly(r,"x","roots")
p-p2
```

    Octave.

```
p=[1 -5 5 5 -6]
r=roots(p)

polyval(p,r)

p2=poly(r)
p2-p
```

Python.

```
import numpy as np

p=np.array([1,-5,5,5,-6])
r=np.roots(p)

np.polyval(p,r)

p2=np.poly(r)
p2-p
```

Perl. `polyroots` obtiene las raíces; `polyval` evalúa el polinomio en todas ellas sin escribir un ciclo explícito, y `polyfromroots` reconstruye los coeficientes a partir de las raíces.

```
use PDL;
use PDL::Math qw(polyroots);

sub polyval { my($c,$x)=@_; my $n=$c->nelem; my
   $y=zeroes($x); for my $i (0..$n-1){ $y += $c->at($i) *
   ($x**($n-1-$i)); } return $y; }
sub polymul { my($a,$b)=@_; my
   $c=zeroes($a->nelem+$b->nelem-1); for my $i
   (0..$a->nelem-1){ my $end=$i+$b->nelem-1;
   $c->slice("$i:$end") += $a->at($i)*$b; } return $c; }
sub poly { my $r=shift; my $c=pdl(1); for my $root
   ($r->list){ $c=polymul($c,pdl(-$root,1)); } return
   $c->slice('-1:0'); }

my $c = pdl(1,-5,5,5,-6);
my ($rr,$ri) = polyroots($c, zeroes($c->nelem));
my $r = $rr;

polyval($c,$r);

my $c2 = poly($r);

$c2-$c;
```

JavaScript. Para grado 4 no hay una función equivalente a `roots` en math.js. Las raíces se obtienen como los autovalores de la matriz compañera. La evaluación se hace mediante Horner. Para reconstruir el polinomio se multiplican los factores asociados a las raíces mediante recorridos implícitos.

```
const math = require('mathjs')
const c = [1,-5,5,5,-6]

// matriz companera
const C = [[5,-5,-5,6],
   [1,0,0,0],
```

```
    [0,1,0,0],
    [0,0,1,0]]

let r = math.eigs(C).values

let horner = (c,x)=>c.reduce(
    (s,a)=>math.add(math.multiply(s,x),a),0
)

r.map(x=>horner(c,x))

let polymul = (a,b)=>{
    const r = Array(a.length+b.length-1).fill(0)
    a.forEach((x,i)=>
        b.forEach((y,j)=>
            r[i+j]=math.add(r[i+j],math.multiply(x,y))
        )
    )
    return r
}

let polyfromroots = rs=>rs.reduce(
    (c,z)=>polymul(c,[1,math.unaryMinus(z)]),[1]
)

let c2 = polyfromroots(r)

c2
math.subtract(c2,c)
```

39. **Ejercicio:** Producto polinomial

Multiplique el polinomio de orden cuatro con raíces $(-1,-1,2,3)$ por el de orden tres con raíces $(0,-2,1)$:

- (a) defina ambos desde sus raíces;
- (b) calcule el producto;
- (c) halle las raíces del producto y compare con las de los factores;
- (d) obtenga los coeficientes del producto.

Las raíces de un producto son las raíces de sus factores, conservando sus multiplicidades. Por lo tanto, las raíces esperadas del producto son

$$-1,-1,2,3,0,-2,1.$$

El orden en que las funciones numéricas devuelven estas raíces puede ser diferente.

Scilab.

```
rp=[-1 -1 2 3]
rq=[0 -2 1]

p=poly(rp,"x","roots")
q=poly(rq,"x","roots")

r=p*q

roots(r)
[rp rq]

coeff(r)
```

Octave. `poly` construye los coeficientes a partir de las raíces y `conv` multiplica los polinomios.

```
rp=[-1 -1 2 3];
rq=[0 -2 1];

p=poly(rp)
q=poly(rq)

r=conv(p,q)

roots(r)
[rp rq]

r
```

Python.

```
import numpy as np

rp=np.array([-1,-1,2,3])
rq=np.array([0,-2,1])

p=np.poly(rp)
q=np.poly(rq)

r=np.polymul(p,q)

np.roots(r)
np.concatenate((rp,rq))

r
```

Perl. `Math::Polynomial` permite construir un polinomio desde sus raíces y multiplicar polinomios directamente. `PDL::Math::polyroots` obtiene las raíces a

partir de los coeficientes.

```
use PDL;
use PDL::Math qw(polyroots);
use Math::Polynomial;

@rp=(-1,-1,2,3);
@rq=(0,-2,1);

my $poly_p = Math::Polynomial->from_roots(@rp);
my $poly_q = Math::Polynomial->from_roots(@rq);
my $poly_r = $poly_p->mul($poly_q);

@cr=reverse $poly_r->coefficients;

my ($rr,$ri) = polyroots(pdl(@cr), zeroes(scalar @cr));
pdl(@rp,@rq);

pdl(@cr);
```

JavaScript. math.js no tiene una función equivalente a `roots` para un polinomio de grado siete. Las raíces se obtienen como los autovalores de la matriz compañera.

```
const math = require('mathjs')
let polymul = (a,b)=>{
  const r=Array(a.length+b.length-1).fill(0)
  a.forEach((x,i)=>b.forEach((y,j)=>r[i+j]+=x*y))
  return r
}

let polyfromroots = rs=>
  rs.reduce((c,z)=>polymul(c,[1,-z]),[1])

const rp = [-1,-1,2,3]
const rq = [0,-2,1]

let p = polyfromroots(rp)
let q = polyfromroots(rq)

let r = polymul(p,q)

const C = [
 [-r[1],-r[2],-r[3],-r[4],-r[5],-r[6],-r[7]],
 [1,0,0,0,0,0,0],
 [0,1,0,0,0,0,0],
 [0,0,1,0,0,0,0],
 [0,0,0,1,0,0,0],
 [0,0,0,0,1,0,0],
 [0,0,0,0,0,1,0]
]

let raices = math.eigs(C).values
```

```
raices
rp.concat(rq)

r
```

40. **Ejercicio:** Raíces de polinomios

Con el polinomio de orden seis de raíces $(-1, -1, 1, 2, 3, 3)$:

- (a) defínalo desde las raíces;
- (b) derívelo;
- (c) halle las raíces de la derivada (candidatos a extremos);
- (d) evalúe el polinomio en ellas;
- (e) clasifique los puntos críticos y determine el máximo y el mínimo absolutos en el intervalo $[-1{,}5, 3{,}2]$.

Para clasificar un punto crítico puede evaluarse la segunda derivada: si $p''(x) > 0$ corresponde a un mínimo local y si $p''(x) < 0$ corresponde a un máximo local.

Los puntos críticos son aproximadamente

$$-1, \quad -0{,}04739, \quad 1{,}44259, \quad 2{,}43813, \quad 3.$$

A partir del signo de la segunda derivada se obtiene:

x	$p(x)$	clasificación
-1	0	mínimo local
$-0{,}04739$	18,07155	máximo local
1,44259	$-3{,}57012$	mínimo local
2,43813	2,35135	máximo local
3	0	mínimo local

Para obtener los extremos absolutos en un intervalo deben compararse también los valores en sus extremos. En $[-1{,}5, 3{,}2]$ se obtiene aproximadamente

$$p(-1{,}5) = 44{,}296875, \qquad p(3{,}2) = 1{,}862784.$$

Por lo tanto, el máximo absoluto es

$$44{,}296875$$

en $x = -1{,}5$, y el mínimo absoluto es aproximadamente

$$-3{,}57012$$

en $x \simeq 1{,}44259$.

Scilab.

```
p=poly([-1 -1 1 2 3 3],"x","roots")

dp=derivat(p)
criticos=real(roots(dp))
valores=horner(p,criticos)

ddp=derivat(dp)
segunda=horner(ddp,criticos)

x=[-1.5 criticos' 3.2]
vx=horner(p,x)

[vmin,imin]=min(vx)
xmin=x(imin)

[vmax,imax]=max(vx)
xmax=x(imax)
```

Octave. El signo de la segunda derivada permite clasificar los puntos críticos. Para los extremos absolutos se agregan también los extremos del intervalo.

```
p=poly([-1 -1 1 2 3 3])

dp=polyder(p)
criticos=roots(dp)
valores=polyval(p,criticos)

ddp=polyder(dp)
segunda=polyval(ddp,criticos)

x=[-1.5;criticos;3.2]
vx=polyval(p,x)

[vmin,imin]=min(vx)
xmin=x(imin)

[vmax,imax]=max(vx)
xmax=x(imax)
```

Python. El signo de la segunda derivada permite clasificar los puntos críticos. `argmin` y `argmax` permiten recuperar también la posición donde se alcanza cada extremo absoluto.

```
import numpy as np

p=np.poly([-1,-1,1,2,3,3])

dp=np.polyder(p)
```

```
criticos=np.roots(dp)
valores=np.polyval(p,criticos)

ddp=np.polyder(dp)
segunda=np.polyval(ddp,criticos)

x=np.concatenate(([-1.5],criticos,[3.2]))
vx=np.polyval(p,x)

vmin=np.min(vx)
xmin=x[np.argmin(vx)]

vmax=np.max(vx)
xmax=x[np.argmax(vx)]
```

Perl. Los valores de la segunda derivada permiten clasificar los puntos críticos. Para determinar los extremos absolutos se evalúa también en los extremos del intervalo.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::Math;

# p(x) = (x+1)^2 (x-1)(x-2)(x-3)^2
# Lo construimos por convolucion de factores.
sub polymul {
    my ($p, $q) = @_;
    my $n = $p->nelem + $q->nelem - 1;
    my $r = zeroes($n);
    for my $i (0 .. $p->nelem - 1) {
        for my $j (0 .. $q->nelem - 1) {
            $r->set($i + $j, $r->at($i + $j) + $p->at($i) *
               $q->at($j));
        }
    }
    return $r;
}

# polyval manual (coeficientes de mayor a menor grado)
sub polyval_pdl {
    my ($c, $x) = @_;
    my $result = zeroes($x);
    for my $i (0 .. $c->nelem - 1) {
        $result += $c->at($i) * ($x ** $i);
    }
    return $result;
}

# derivada
```

```
sub polyder_pdl {
    my $c = shift;
    my $n = $c->nelem;
    return pdl(map { $c->at($_) * ($n - 1 - $_) } 0 .. $n -
       2);
}

my $poli = polymul(polymul(polymul(polymul(pdl([1, 1]),
   pdl([1, 1])), pdl([1, -1])), pdl([1, -2])), pdl([1, -6,
   9]));
print "polinomio = $poli\n";

my $dp  = polyder_pdl($poli);
my $ddp = polyder_pdl($dp);

my ($rr, $ri) = polyroots($dp, zeroes($dp->nelem));
my $criticos = $rr;
print "criticos = $criticos\n";

my $valores  = polyval_pdl($poli, $criticos);
my $segunda  = polyval_pdl($ddp, $criticos);
print "valores = $valores\n";
print "segunda = $segunda\n";

my $x = pdl([-1.5])->append($criticos)->append(pdl([3.2]));
my $vx = polyval_pdl($poli, $x);
print "x = $x\n";
print "vx = $vx\n";

print "min(vx) = ", min($vx), "\n";
print "max(vx) = ", max($vx), "\n";
```

JavaScript. Se reutilizan `polyfromroots` y `horner` del ejercicio anterior. Las raíces de la derivada se obtienen como los autovalores de su matriz compañera.

```
const math = require('mathjs')

let polymul = (a,b)=>{
    const r = Array(a.length+b.length-1).fill(0)
    a.forEach((x,i)=>
        b.forEach((y,j)=>
            r[i+j]=math.add(r[i+j],math.multiply(x,y))
        )
    )
    return r
}

let polyfromroots = rs=>rs.reduce(
    (c,z)=>polymul(c,[1,math.unaryMinus(z)]),[1]
)
```

```
let horner = (c,x)=>c.reduce(
    (s,a)=>math.add(math.multiply(s,x),a),0
)

let p = polyfromroots([-1,-1,1,2,3,3])

let polyder = c=>
  c.slice(0,-1).map((a,i)=>a*(c.length-1-i))

let dp = polyder(p)
let ddp = polyder(dp)

// dp tiene grado cinco
let a = dp.map(v=>v/dp[0])

const C = [
 [-a[1],-a[2],-a[3],-a[4],-a[5]],
 [1,0,0,0,0],
 [0,1,0,0,0],
 [0,0,1,0,0],
 [0,0,0,1,0]
]

let criticos = math.eigs(C).values

let valores = criticos.map(x=>horner(p,x))
let segunda = criticos.map(x=>horner(ddp,x))

const x = [-1.5,...criticos,3.2]
let vx = x.map(z=>horner(p,z))

let vmin = Math.min(...vx)
let xmin = x[vx.indexOf(vmin)]

let vmax = Math.max(...vx)
let xmax = x[vx.indexOf(vmax)]
```

4.23.9. Cálculo numérico. Funciones de una línea matemática

41. **Ejercicio:** Mapeo de funciones

    a) Defina la función $f(x) = x^2 + 1$ de manera que pueda recibir tanto un escalar como un vector.

    b) Aplíquela al vector formado por los enteros de -3 a 3.

    c) Compare el resultado con evaluar directamente la expresión $x^2 + 1$ sobre el mismo vector.

    d) Defina $g(x) = \sin x + \cos x$ y aplíquela a 100 puntos del intervalo $[0, 2\pi]$.

e) Observe que no fue necesario escribir explícitamente un ciclo para calcular la función en todos los elementos del vector.

En los lenguajes que permiten operaciones vectorizadas, una sola expresión puede aplicarse a todos los elementos del vector. El recorrido de sus elementos queda implícito en las operaciones utilizadas.

Scilab. Para que las operaciones algebraicas se realicen elemento a elemento deben utilizarse, cuando corresponda, los operadores con punto.

```
deff("[y]=f(x)","y=x.^2+1")

f(2)

x=-3:3
y=f(x)

y2=x.^2+1
y-y2

deff("[y]=g(x)","y=sin(x)+cos(x)")

t=linspace(0,2*%pi,100)
z=g(t)
```

También puede escribirse la función en varias líneas:

```
function y=f(x)
    y=x.^2+1
endfunction
```

Octave. Los operadores con punto realizan las operaciones algebraicas elemento a elemento.

```
f=@(x) x.^2+1;

f(2)

x=-3:3;
y=f(x)

y2=x.^2+1;
y-y2

g=@(x) sin(x)+cos(x);

t=linspace(0,2*pi,100);
z=g(t);
```

Python. Con arreglos de NumPy, las operaciones aritméticas y las funciones universales se aplican elemento a elemento.

```
import numpy as np

def f(x):
    return x**2+1

f(2)

x=np.arange(-3,4)
y=f(x)

y2=x**2+1
y-y2

def g(x):
    return np.sin(x)+np.cos(x)

t=np.linspace(0,2*np.pi,100)
z=g(t)
```

Perl. Con PDL, las operaciones aritméticas y las funciones matemáticas se aplican también a todos los elementos de un piddle.

```
use PDL;
use Math::Trig qw(pi);

sub f {
    my $x=shift;
    return $x**2+1;
}

f(2);

$y;

$y-$y2;

sub g {
    my $x=shift;
    return sin($x)+cos($x);
}

$z;
```

JavaScript. math.js permite aplicar operaciones matemáticas directamente sobre arreglos, sin escribir explícitamente el recorrido de sus elementos.

```
const math = require('mathjs')
```

```
let f = x=>math.add(math.dotPow(x,2),1)

f(2)

const x = [-3,-2,-1,0,1,2,3]
let y = f(x)

let y2 = math.add(math.dotPow(x,2),1)
math.subtract(y,y2)

let g = x=>math.add(math.map(x, math.sin), math.map(x,
   math.cos))

let t = math.range(0,2*Math.PI,2*Math.PI/99,true).toArray()
let z = g(t)
```

42. **Ejercicio:** Vectorización de funciones

    a) Defina la función escalar
    $$f(x) = x^2 + 1.$$

    b) Aplique la función a cada uno de los enteros de -3 a 3, sin escribir explícitamente un ciclo.

    c) Compare el resultado con el obtenido al escribir directamente la expresión x^2+1 sobre el vector, cuando el lenguaje permita operaciones vectorizadas.

    d) Defina
    $$g(x) = \sin x + \cos x$$
    y aplíquela, elemento a elemento, a 100 puntos de $[0, 2\pi]$.

    e) Explique en qué sentido la aplicación elemento a elemento realiza un recorrido implícito de los valores del vector.

Al aplicar la función elemento a elemento no se escribe ningún ciclo: el recorrido de los valores del vector lo hace implícitamente la calculadora, que toma cada componente, le aplica la función y va formando el vector de resultados. El ciclo existe, pero queda a cargo del intérprete y no del programa.

Scilab. `feval` permite aplicar una función escalar a cada elemento de un vector. Esto es distinto de definir una función que opere directamente sobre vectores. En una expresión vectorizada, la potencia elemento a elemento se escribe `.^`.

```
deff("[y]=f(x)", "y = x^2 + 1");

x = -3:3;
y = feval(x, f);

y2 = x.^2 + 1;
```

```
// comparacion: deben coincidir
y - y2

deff("[y]=g(x)", "y = sin(x) + cos(x)");

t = linspace(0, 2*%pi, 100);
z = feval(t, g);
```

Octave. `arrayfun` aplica una función escalar a cada elemento de un arreglo. Para operar directamente sobre un vector, la potencia elemento a elemento se escribe `.^`.

```
f = @(x) x^2 + 1;

x = -3:3;
y = arrayfun(f, x);

y2 = x.^2 + 1;

% comparacion: deben coincidir
y - y2

g = @(x) sin(x) + cos(x);

t = linspace(0, 2*pi, 100);
z = arrayfun(g, t);
```

Python. `numpy.vectorize` permite presentar una función escalar como una función aplicable a un arreglo. No debe confundirse con una operación vectorizada de NumPy: internamente sigue aplicando la función a los elementos.

```
import math
import numpy as np

def f(x):
    return x**2 + 1

x = np.arange(-3, 4)
y = np.vectorize(f)(x)

y2 = x**2 + 1

# comparacion: deben coincidir
print(y - y2)

def g(x):
    return math.sin(x) + math.cos(x)

t = np.linspace(0, 2*np.pi, 100)
z = np.vectorize(g)(t)
```

Perl. `map` aplica una función a cada elemento de una lista. Perl básico no posee operaciones aritméticas vectorizadas sobre listas; para compararlas con una expresión vectorizada puede utilizarse, por ejemplo, PDL.

```
use Math::Trig qw(pi);

sub f {
    my ($x) = @_;
    return $x**2 + 1;
}

@x = -3..3;
@y = map { f($_) } @x;

sub g {
    my ($x) = @_;
    return sin($x) + cos($x);
}

@t = map { 2*pi*$_/99 } 0..99;
@z = map { g($_) } @t;
```

```
# Con PDL, la expresion puede escribirse directamente:
use PDL;

$x = pdl(-3, -2, -1, 0, 1, 2, 3);
$y2 = $x**2 + 1;

# Comparacion con la aplicacion elemento a elemento (map):
@y = map { $_**2 + 1 } -3 .. 3;

print "map: @y\n";
print "pdl: $y2\n";
```

JavaScript. `map` aplica una función a cada elemento de un arreglo. Los arreglos de JavaScript no poseen directamente operaciones aritméticas vectorizadas; bibliotecas como math.js las proporcionan.

```
const f = x => x**2 + 1;

const x = [-3,-2,-1,0,1,2,3];
const y = x.map(f);

const g = x => Math.sin(x) + Math.cos(x);

const t = Array.from(
    {length: 100},
    (_,i) => 2*Math.PI*i/99
);
```

```
const z = t.map(g);
```

```
const math = require('mathjs')

// Con math.js, la expresion puede escribirse sobre el
   vector:
const x = [-3,-2,-1,0,1,2,3];
const y2 = math.add(math.dotPow(x, 2), 1);

// comparacion con la aplicacion elemento a elemento:
const y = x.map(v => v**2 + 1);
console.log('map:', y);
console.log('vectorizado:', y2);
```

43. **Ejercicio:** Ajuste lineal de datos

Con $x = (0, 1, 2, 3, 4)$ e $y = (1{,}1, 2{,}0, 2{,}9, 4{,}2, 5{,}1)$:

- (a) ajuste la recta $y = ax + b$ por mínimos cuadrados;
- (b) calcule los residuos y su desvío

$$\sigma = \sqrt{\frac{\sum_i (y_i - \hat{y}_i)^2}{n-2}};$$

- (c) interprete a, b y el desvío.

El ajuste da aproximadamente

$$a = 1{,}02, \qquad b = 1{,}02,$$

por lo que

$$\hat{y} = 1{,}02x + 1{,}02.$$

La pendiente indica que, por cada unidad que aumenta x, el valor ajustado de y aumenta aproximadamente 1,02. La ordenada al origen es el valor ajustado de y para $x = 0$.

El desvío residual es aproximadamente

$$\sigma = 0{,}126,$$

y mide la dispersión de los datos alrededor de la recta ajustada.

Scilab.

```
x=[0 1 2 3 4]
y=[1.1 2.0 2.9 4.2 5.1]

[a,b]=reglin(x,y)
```

```
yfit=a*x+b
res=y-yfit

n=length(x)
sig=sqrt(sum(res.^2)/(n-2))
```

Octave. `polyfit` devuelve primero la pendiente y luego la ordenada al origen.

```
x=[0 1 2 3 4]
y=[1.1 2.0 2.9 4.2 5.1]

p=polyfit(x,y,1)
a=p(1)
b=p(2)

yfit=polyval(p,x)
res=y-yfit

n=length(x)
sig=sqrt(sum(res.^2)/(n-2))
```

Python.

```
import numpy as np

x=np.array([0,1,2,3,4])
y=np.array([1.1,2.0,2.9,4.2,5.1])

a,b=np.polyfit(x,y,1)

yfit=a*x+b
res=y-yfit

n=len(x)
sig=np.sqrt(np.sum(res**2)/(n-2))
```

Perl. `linfit1d` realiza el ajuste lineal. El desvío residual se calcula con la misma expresión utilizada en los demás lenguajes.

```
use PDL;
use PDL::Fit::Linfit;

my $x = pdl(0,1,2,3,4);
my $y = pdl(1.1,2.0,2.9,4.2,5.1);

($yfit,$coef)=linfit1d($y,cat($x,ones($x)));

$a = $coef->slice(0);
$b = $coef->slice(1);
$sig = sqrt(sum(($y-$yfit)**2)/($x->nelem-2));
```

JavaScript. Las ecuaciones normales permiten obtener la pendiente y la ordenada al origen mediante un sistema lineal de dos ecuaciones.

```
const math = require('mathjs')

const x = [0,1,2,3,4]
const y = [1.1,2.0,2.9,4.2,5.1]

let n = x.length
let sx = math.sum(x)
let sy = math.sum(y)
let sxx = math.dot(x,x)
let sxy = math.dot(x,y)

const [a,b] = math.lusolve(
    [[sxx,sx],[sx,n]],
    [sxy,sy]
).map(r=>r[0])

let yfit = x.map(v=>a*v+b)
let res = y.map((v,i)=>v-yfit[i])

let sig = Math.sqrt(
    math.sum(res.map(v=>v*v))/(n-2)
)
```

44. **Ejercicio:** Límites numéricos

Estime $\lim_{x\to 0} \frac{\sin x}{x}$ evaluando en $h = 10^{-1}, \ldots, 10^{-5}$.

Scilab.

```
h=[1e-1 1e-2 1e-3 1e-4 1e-5]; sin(h)./h
```

Octave.

```
h=[1e-1 1e-2 1e-3 1e-4 1e-5]; sin(h)./h
```

Python.

```
import numpy as np

h=np.array([1e-1,1e-2,1e-3,1e-4,1e-5]); np.sin(h)/h
```

Perl.

```
# Estimacion de lim_{x->0} sin(x)/x evaluando en h = 1e-1 ..
   1e-5
```

```
my @h = (1e-1, 1e-2, 1e-3, 1e-4, 1e-5);
my @lim = map { sin($_) / $_ } @h;

print "h         = @h\n";
print "sin(h)/h = @lim\n";
```

JavaScript.

```
const h = [1e-1,1e-2,1e-3,1e-4,1e-5]; h.map(v=>Math.sin(v)/v)
```

45. **Ejercicio:** Derivadas numéricas

Calcule la derivada de $\sin x$ en $x = \pi/4$ por diferencias hacia adelante, hacia atrás y central, para $h = 10^{-1}, 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}$; compare los errores con $\cos(\pi/4)$. ¿Cuál método es más preciso? ¿Qué pasa cuando h es muy pequeño?

La derivada exacta es

$$f'(\pi/4) = \cos(\pi/4) = \frac{\sqrt{2}}{2}.$$

Las diferencias hacia adelante y hacia atrás tienen un error de aproximación de orden h, mientras que la diferencia central tiene un error de orden h^2. Por eso, para valores moderadamente pequeños de h, la diferencia central resulta considerablemente más precisa.

Sin embargo, disminuir h indefinidamente no mejora siempre el resultado. Cuando h es muy pequeño se restan números casi iguales y los errores de redondeo de la aritmética de punto flotante comienzan a ser importantes. El error puede entonces dejar de disminuir e incluso aumentar.

Scilab.

```
x0=%pi/4
ex=cos(x0)
h=[1e-1 1e-2 1e-4 1e-6 1e-8]

ad=(sin(x0+h)-sin(x0))./h
at=(sin(x0)-sin(x0-h))./h
ce=(sin(x0+h)-sin(x0-h))./(2*h)

ead=abs(ad-ex)
eat=abs(at-ex)
ece=abs(ce-ex)
```

Octave.

```
x0=pi/4;
ex=cos(x0);
h=[1e-1 1e-2 1e-4 1e-6 1e-8];
```

```
ad=(sin(x0+h)-sin(x0))./h;
at=(sin(x0)-sin(x0-h))./h;
ce=(sin(x0+h)-sin(x0-h))./(2*h);

ead=abs(ad-ex)
eat=abs(at-ex)
ece=abs(ce-ex)
```

Python.

```
import numpy as np

x0=np.pi/4
ex=np.cos(x0)
h=np.array([1e-1,1e-2,1e-4,1e-6,1e-8])

ad=(np.sin(x0+h)-np.sin(x0))/h
at=(np.sin(x0)-np.sin(x0-h))/h
ce=(np.sin(x0+h)-np.sin(x0-h))/(2*h)

ead=np.abs(ad-ex)
eat=np.abs(at-ex)
ece=np.abs(ce-ex)
```

Perl.

```
# Derivada de sin(x) en x0 = pi/4 por diferencias finitas

use Math::Trig qw(pi);

my $x0 = pi / 4;
my $ex = cos($x0);
my @h  = (1e-1, 1e-2, 1e-4, 1e-6, 1e-8);

my @ad = map { (sin($x0 + $_) - sin($x0)) / $_ } @h;
my @at = map { (sin($x0) - sin($x0 - $_)) / $_ } @h;
my @ce = map { (sin($x0 + $_) - sin($x0 - $_)) / (2 * $_) }
   @h;

my @ead = map { abs($ad[$_] - $ex) } 0 .. $#h;
my @eat = map { abs($at[$_] - $ex) } 0 .. $#h;
my @ece = map { abs($ce[$_] - $ex) } 0 .. $#h;

print "h              = @h\n";
print "error adelante = @ead\n";
print "error atras    = @eat\n";
print "error central  = @ece\n";
```

JavaScript.

```
let x0 = Math.PI/4
```

```
let ex = Math.cos(x0)
const h = [1e-1,1e-2,1e-4,1e-6,1e-8]

let ad = h.map(v=>(Math.sin(x0+v)-Math.sin(x0))/v)
let at = h.map(v=>(Math.sin(x0)-Math.sin(x0-v))/v)
let ce = h.map(v=>(Math.sin(x0+v)-Math.sin(x0-v))/(2*v))

let ead = ad.map(v=>Math.abs(v-ex))
let eat = at.map(v=>Math.abs(v-ex))
let ece = ce.map(v=>Math.abs(v-ex))
```

46. **Ejercicio:** Tangente a una curva

Para $f(x) = x^2$, calcule la tangente en $x_0 = 2$ y explique la pendiente.

Como

$$f'(x) = 2x,$$

en $x_0 = 2$ la pendiente es

$$m = f'(2) = 4.$$

Además,

$$f(2) = 4,$$

por lo que la recta tangente es

$$y - 4 = 4(x - 2),$$

o equivalentemente,

$$y = 4x - 4.$$

La pendiente $m = 4$ indica que, cerca de $x = 2$, un incremento pequeño de una unidad en x produce aproximadamente un incremento de cuatro unidades en $f(x)$.

Scilab.

```
x0=2
y0=x0^2
m=2*x0

x=linspace(0,4,100)
y=x.^2
yt=y0+m*(x-x0)
```

Octave.

```
x0=2;
y0=x0^2;
m=2*x0;

x=linspace(0,4,100);
```

```
y=x.^2;
yt=y0+m*(x-x0);
```

Python.

```
import numpy as np

x0=2
y0=x0**2
m=2*x0

x=np.linspace(0,4,100)
y=x**2
yt=y0+m*(x-x0)
```

Perl.

```
# Recta tangente a f(x) = x^2 en x0 = 2

my $x0 = 2;
my $y0 = $x0**2;
my $m  = 2 * $x0;

my @x  = map { 4 * $_ / 99 } 0 .. 99;
my @y  = map { $_**2 } @x;
my @yt = map { $y0 + $m * ($_ - $x0) } @x;

print "pendiente m = $m; tangente: y = $m*x - 4\n";
```

JavaScript.

```
const math = require('mathjs')
let x0 = 2
let y0 = x0**2
let m = 2*x0

let x = math.range(0,4,4/99,true).toArray()
let y = x.map(v=>v*v)
let yt = x.map(v=>y0+m*(v-x0))
```

47. **Ejercicio:** Series de Taylor

Compare e^x con

$$T_3(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$$

en $[-1, 1]$ y observe dónde la aproximación es mejor.

El polinomio T_3 es la aproximación de Taylor de e^x alrededor de $x = 0$. Por eso el error es menor cerca de 0 y aumenta al alejarse de ese punto.

Para observarlo numéricamente se calcula

$$|e^x - T_3(x)|\,.$$

Scilab.

```
x=linspace(-1,1,100)

y=exp(x)
T3=1+x+x.^2/2+x.^3/6

error=abs(y-T3)

min(error)
max(error)
```

Octave.

```
x=linspace(-1,1,100);

y=exp(x);
T3=1+x+x.^2/2+x.^3/6;

error=abs(y-T3);

min(error)
max(error)
```

Python.

```
import numpy as np

x=np.linspace(-1,1,100)

y=np.exp(x)
T3=1+x+x**2/2+x**3/6

error=np.abs(y-T3)

np.min(error)
np.max(error)
```

Perl.

```
use PDL;

my $x = zeroes(100)->xlinvals(-1,1);
my $y = exp($x);
my $T3 = 1 + $x + $x**2/2 + $x**3/6;
```

```
my $error = abs($y - $T3);

min($error);
max($error);
```

JavaScript.

```
const math = require('mathjs')
let x = math.range(-1,1,2/99,true).toArray()

let y = x.map(Math.exp)
let T3 = x.map(v=>1+v+v*v/2+v**3/6)

let error = y.map((v,i)=>Math.abs(v-T3[i]))

Math.min(...error)
Math.max(...error)
```

48. **Ejercicio:** Integración numérica

Calcule $\int_0^1 x^2\,dx$ mediante una rutina de cuadratura numérica y compare el resultado con $1/3$.

Scilab. `intg` realiza integración numérica de una función.

```
function y = f(x)
    y = x^2;
endfunction

[I, err] = intg(0, 1, f)
I - 1/3
```

Octave. `quad` realiza cuadratura numérica y también devuelve información sobre el error.

```
f = @(x) x.^2;
[I, ier, nfun, err] = quad(f, 0, 1)
I - 1/3
```

Python. `integrate.quad`, de SciPy, realiza cuadratura numérica y devuelve el valor calculado y una estimación del error.

```
from scipy import integrate

I, err = integrate.quad(lambda x: x**2, 0, 1)
I - 1/3
```

Perl. `gslinteg_qag`, de `PDL::GSL::INTEG`, realiza cuadratura numérica y devuelve el valor y una estimación del error.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::GSL::INTEG;

my ($I, $err) = gslinteg_qag(sub { $_[0] ** 2 }, 0, 1,
   1e-10, 1e-10, 1000, 6);

print "I = $I\n";
print "err = $err\n";
print "I - 1/3 = ", $I - 1 / 3, "\n";
```

JavaScript. Aplicamos la regla del trapecio compuesta con un número fijo de subintervalos; el resultado se compara con 1/3.

```
let n = 1000
let h = 1/n
let I = 0.5
   *h*(0**2+1**2)+h*Array.from({length:n-1},(_,i)=>((i+1)*h)**2).reduce(
I
I-1/3
```

49. **Ejercicio:** Integrales dobles

Calcule

$$\int_0^1 \int_0^1 (x+y)\,dx\,dy :$$

- (a) defina $f(x,y) = x+y$;
- (b) integre sobre el cuadrado $[0,1] \times [0,1]$;
- (c) compare con el valor exacto.

El valor exacto es

$$\int_0^1 \int_0^1 (x+y)\,dx\,dy = 1.$$

Por lo tanto, la diferencia entre el resultado numérico y 1 debe ser cero o un valor muy pequeño debido al error numérico.

Scilab. `int2d` integra sobre triángulos. El cuadrado se divide en dos triángulos.

```
deff("[z]=f(x,y)","z=x+y")

X=[0 0;1 1;1 0]
Y=[0 0;0 1;1 1]
```

```
[I,err]=int2d(X,Y,f)

I
I-1
```

Octave. `integral2` integra directamente sobre una región rectangular.

```
f=@(x,y) x+y;

I=integral2(f,0,1,0,1)

I-1
```

Python. `dblquad` realiza una integral doble. La función recibe primero la variable correspondiente a la integral interna.

```
from scipy import integrate

def f(y,x):
    return x+y

I,err=integrate.dblquad(f,0,1,0,1)

I
I-1
```

Perl. `gslinteg_qag` puede aplicarse dos veces: para cada valor de y se integra primero respecto de x, y luego se integra el resultado respecto de y.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::GSL::INTEG;

sub fxy {
    my ($x, $y) = @_;
    return $x + $y;
}

sub interna {
    my $y = shift;
    my ($I) = gslinteg_qag(sub { fxy($_[0], $y) }, 0, 1,
       1e-10, 1e-10, 1000, 6);
    return $I;
}

my ($I, $err) = gslinteg_qag(\&interna, 0, 1, 1e-10, 1e-10,
   1000, 6);
```

```
print "I = $I\n";
print "I - 1 = ", $I - 1, "\n";
```

JavaScript. Como math.js no proporciona aquí una rutina de integración doble, aplicamos la regla del trapecio sobre una grilla uniforme.

```
let n = 100
let h = 1/n
const xs = Array.from({length:n+1},(_,i)=>i*h)
let f = (x,y)=>x+y
let I = 0
for(let i=0;i<=n;i++){
    for(let j=0;j<=n;j++){
        let wi = (i===0||i===n)?0.5:1
        let wj = (j===0||j===n)?0.5:1
        I += wi*wj*h*h*f(xs[i],xs[j])
    }
}
I
I-1
```

50. **Ejercicio:** Ecuaciones diferenciales ordinarias

   a) Resuelva
   $$\dot{y} = -y, \qquad y(0) = 1,$$
   cuya solución exacta es $y = e^{-t}$, hasta $t = 5$.

   b) Repita el cálculo usando pasos máximos 0,5, 0,1 y 0,01 (o esos pasos directamente cuando el integrador sea de paso fijo). Compare el error en $t = 5$ con la solución exacta.

   c) ¿Qué ocurre con el error al disminuir el paso máximo?

   d) En general, limitar el paso máximo obliga al integrador a realizar una discretización más fina y puede disminuir el error. En los integradores adaptativos el error también depende de las tolerancias internas, por lo que la disminución no tiene por qué continuar indefinidamente.

   No confundir la separación de los puntos de salida con el paso interno del integrador.

Scilab. `ode` utiliza un integrador adaptativo. El vector de tiempos indica dónde se desea obtener la solución; el paso interno máximo se fija mediante `%ODEOPTIONS`.

```
function ydot=f(t,y)
    ydot=-y
endfunction

tex=5
```

```
vex=exp(-tex)

h1=0.5
%ODEOPTIONS=[1,0,0,h1,0,2,5000,12,5,0,-1,-1]
y1=ode(1,0,tex,f)
e1=abs(y1-vex)

h2=0.1
%ODEOPTIONS=[1,0,0,h2,0,2,5000,12,5,0,-1,-1]
y2=ode(1,0,tex,f)
e2=abs(y2-vex)

h3=0.01
%ODEOPTIONS=[1,0,0,h3,0,2,5000,12,5,0,-1,-1]
y3=ode(1,0,tex,f)
e3=abs(y3-vex)

e1,e2,e3

clear %ODEOPTIONS
```

Octave. `ode45` usa paso adaptativo. `MaxStep` limita el paso interno máximo.

```
f=@(t,y) -y;

tex=5;
vex=exp(-tex);

h1=0.5;
op1=odeset("MaxStep",h1);
[t1,y1]=ode45(f,[0 tex],1,op1);
e1=abs(y1(end)-vex);

h2=0.1;
op2=odeset("MaxStep",h2);
[t2,y2]=ode45(f,[0 tex],1,op2);
e2=abs(y2(end)-vex);

h3=0.01;
op3=odeset("MaxStep",h3);
[t3,y3]=ode45(f,[0 tex],1,op3);
e3=abs(y3(end)-vex);

e1,e2,e3
```

Python. `solve_ivp` usa paso adaptativo. `max_step` limita el paso interno máximo.

```
import numpy as np
```

```
from scipy.integrate import solve_ivp

def f(t,y):
    return -y

tex=5
vex=np.exp(-tex)

h1=0.5
y1=solve_ivp(f,[0,tex],[1],
             t_eval=[tex],max_step=h1).y[0,-1]
e1=abs(y1-vex)

h2=0.1
y2=solve_ivp(f,[0,tex],[1],
             t_eval=[tex],max_step=h2).y[0,-1]
e2=abs(y2-vex)

h3=0.01
y3=solve_ivp(f,[0,tex],[1],
             t_eval=[tex],max_step=h3).y[0,-1]
e3=abs(y3-vex)

e1,e2,e3
```

Perl. `Math::ODE` utiliza aquí un método de paso fijo, por lo que los valores indicados son directamente los pasos de integración.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use Math::ODE;

# EDO: y' = -y, y(0)=1
sub f {
    my ($t, $y) = @_;
    return -$y->[0];
}

sub final_value {
    my ($h, $T) = @_;
    my %args;
    $args{step}    = $h;
    $args{initial} = [1];
    $args{ODE}     = [\&f];
    $args{t0}      = 0;
    $args{tf}      = $T;
    $args{verbose} = 0;
    my $o = Math::ODE->new(%args);
    $o->evolve;
```

```
    my @t = sort { $a <=> $b } keys %{$o->{values}};
    return $o->{values}{$t[-1]}[0];
}

my $tex = 5;
my $vex = exp(-$tex);

my $h1 = 0.5;
my $h2 = 0.1;
my $h3 = 0.01;

my $y1 = final_value($h1, $tex);
my $y2 = final_value($h2, $tex);
my $y3 = final_value($h3, $tex);

my $e1 = abs($y1 - $vex);
my $e2 = abs($y2 - $vex);
my $e3 = abs($y3 - $vex);

print "e1=$e1, e2=$e2, e3=$e3\n";
my $errores = pdl($e1, $e2, $e3);
```

JavaScript. Con `odex` el integrador es adaptativo y `maxStepSize` limita el paso máximo.

```
const odex=require("odex")

const f=(t,y)=>[-y[0]]
const tex=5
const vex=Math.exp(-tex)

const s1=new odex.Solver(f,1)
s1.maxStepSize=0.5
const sol1=s1.integrate(0,[1])
const e1=Math.abs(sol1(tex)[0]-vex)

const s2=new odex.Solver(f,1)
s2.maxStepSize=0.1
const sol2=s2.integrate(0,[1])
const e2=Math.abs(sol2(tex)[0]-vex)

const s3=new odex.Solver(f,1)
s3.maxStepSize=0.01
const sol3=s3.integrate(0,[1])
const e3=Math.abs(sol3(tex)[0]-vex)

console.log(e1,e2,e3)
```

4.23.10. LaTeX

51. **Ejercicio:** Exportación a LaTeX

    - (a) genere la representación LaTeX de la matriz

    $$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix};$$

    - (b) péguela en un entorno matemático y verifique el resultado;
    - (c) genere la representación LaTeX de la expresión $\pi/2$;
    - (d) péguela y escriba el resultado esperado;
    - (e) explique para qué puede ser útil esta conversión.

    Para la matriz se espera obtener una representación equivalente a

```
\begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
```

    que, dentro de un entorno matemático, produce

    $$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}.$$

    Para la expresión escalar se espera una representación equivalente a

```
\frac{\pi}{2}
```

    que produce

    $$\frac{\pi}{2}.$$

    La conversión a LaTeX permite trasladar resultados obtenidos en la calculadora o lenguaje de programación a informes y documentos matemáticos sin tener que volver a escribir manualmente las expresiones.

    Scilab. `prettyprint` puede producir directamente una representación en LaTeX.

```
A=[1 2;3 4]

sA=prettyprint(A,"latex")
sE=prettyprint(%pi/2,"latex")
```

Octave. Octave base no proporciona una conversión general de expresiones numéricas a LaTeX. Para este caso sencillo la representación puede construirse mediante cadenas y conversiones numéricas.

```
A=[1 2;3 4];

bs=char(92);

sA=[bs "begin{pmatrix}" ...
    num2str(A(1,1)) " & " num2str(A(1,2)) ...
    bs bs ...
    num2str(A(2,1)) " & " num2str(A(2,2)) ...
    bs "end{pmatrix}"]

sE=[bs "frac{" bs "pi}{2}"]
```

Python. SymPy dispone de `latex` para obtener la representación LaTeX de objetos y expresiones simbólicas.

```
import sympy as sp

A=sp.Matrix([[1,2],[3,4]])

sA=sp.latex(A)
sE=sp.latex(sp.pi/2)

sA
sE
```

Perl. La matriz puede escribirse como una cadena LaTeX a partir de sus filas. Para expresiones simbólicas puede utilizarse un módulo específico.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::Constants qw(PI);

# Matriz de ejemplo
my @A = ([1, 2], [3, 4]);

# Generar representacion LaTeX de la matriz
my $sA = '\\begin{pmatrix}' . join(' \\\\ ', map { join(' &
    ', @$_) } @A) . '\\end{pmatrix}';
print "sA = $sA\n";

# Representacion LaTeX de pi/2
my $sE = '\\frac{\\pi}{2}';
print "sE = $sE\n";
```

JavaScript. `toTex` convierte una expresión analizada por math.js a una representación LaTeX.

```
const math = require('mathjs')
let sA = math.parse("[[1,2],[3,4]]").toTex()
let sE = math.parse("pi/2").toTex()

sA
sE
```

4.23.11. Expresiones regulares

52. **Ejercicio:** Dominios en expresiones regulares

Escriba una expresión regular que permita determinar si una cadena tiene el formato de una dirección de correo electrónico sencilla.

Considere inicialmente válido el formato

`usuario@dominio.extension`

donde el usuario puede contener letras, números, puntos, guiones y guiones bajos; el dominio puede contener letras, números y guiones; y la extensión debe contener entre dos y seis letras.

Use como datos de prueba, uno a la vez:

```
ana.perez@ejemplo.com
juan_23@universidad.edu.ar
usuario@@servidor.com
sin-dominio
maria@equipo
```

a) Escriba la expresión regular correspondiente al formato indicado.

b) Determine cuáles de las cadenas de prueba son válidas.

c) Modifique la expresión regular para aceptar dominios con varios componentes, como `universidad.edu.ar`, y vuelva a evaluar las cadenas de prueba.

Para el formato inicial puede utilizarse

```
^[A-Za-z0-9._-]+@[A-Za-z0-9-]+\.[A-Za-z]{2,6}$
```

Los símbolos `^` y `$` indican que debe coincidir la cadena completa.

Con esta expresión resulta válida solamente

```
ana.perez@ejemplo.com
```

La dirección

```
juan_23@universidad.edu.ar
```

no coincide todavía porque el dominio contiene más de un componente.

Para admitir dominios con varios componentes puede utilizarse

```
^[A-Za-z0-9._-]+@[A-Za-z0-9-]+(?:\.[A-Za-z0-9-]+)*\.[A-Za-z]{2,6}$
```

Con esta segunda expresión son válidas

```
ana.perez@ejemplo.com
juan_23@universidad.edu.ar
```

y continúan siendo inválidas

```
usuario@@servidor.com
sin-dominio
maria@equipo
```

Scilab. `regexp` permite buscar una expresión regular. Una coincidencia produce una posición; una cadena inválida no produce ninguna.

```
patron="/^[A-Za-z0-9._-]+@[A-Za-z0-9-]+\.[A-Za-z]{2,6}$/"

regexp("ana.perez@ejemplo.com",patron)
regexp("juan_23@universidad.edu.ar",patron)
regexp("usuario@@servidor.com",patron)
regexp("sin-dominio",patron)
regexp("maria@equipo",patron)

patron2="/^[A-Za-z0-9._-]+@[A-Za-z0-9-]+(?:\.[A-Za-z0-9-]+)*\.[A-Za-z]{2

regexp("ana.perez@ejemplo.com",patron2)
regexp("juan_23@universidad.edu.ar",patron2)
regexp("usuario@@servidor.com",patron2)
regexp("sin-dominio",patron2)
regexp("maria@equipo",patron2)
```

Octave. `regexp` devuelve la posición de la coincidencia. Una salida vacía indica que la cadena no cumple el formato.

```
patron="^[A-Za-z0-9._-]+@[A-Za-z0-9-]+\.[A-Za-z]{2,6}$";

regexp("ana.perez@ejemplo.com",patron)
regexp("juan_23@universidad.edu.ar",patron)
regexp("usuario@@servidor.com",patron)
regexp("sin-dominio",patron)
```

```
regexp("maria@equipo",patron)

patron2="^[A-Za-z0-9._-]+@[A-Za-z0-9-]+(?:\.[A-Za-z0-9-]+)*\.[A-Za-z]{2,

regexp("ana.perez@ejemplo.com",patron2)
regexp("juan_23@universidad.edu.ar",patron2)
regexp("usuario@@servidor.com",patron2)
regexp("sin-dominio",patron2)
regexp("maria@equipo",patron2)
```

Python. `re.fullmatch` comprueba que la expresión regular coincida con la cadena completa. Una coincidencia devuelve un objeto; una cadena inválida devuelve `None`.

```
import re

patron=r"[A-Za-z0-9._-]+@[A-Za-z0-9-]+\.[A-Za-z]{2,6}"

re.fullmatch(patron,"ana.perez@ejemplo.com")
re.fullmatch(patron,"juan_23@universidad.edu.ar")
re.fullmatch(patron,"usuario@@servidor.com")
re.fullmatch(patron,"sin-dominio")
re.fullmatch(patron,"maria@equipo")

patron2=r"[A-Za-z0-9._-]+@[A-Za-z0-9-]+(?:\.[A-Za-z0-9-]+)*\.[A-Za-z]{2,

re.fullmatch(patron2,"ana.perez@ejemplo.com")
re.fullmatch(patron2,"juan_23@universidad.edu.ar")
re.fullmatch(patron2,"usuario@@servidor.com")
re.fullmatch(patron2,"sin-dominio")
re.fullmatch(patron2,"maria@equipo")
```

Perl. El operador `=~` aplica la expresión regular a una cadena. Los anclajes obligan a que coincida la cadena completa.

```
"ana.perez@ejemplo.com" =~ $patron;
"juan_23@universidad.edu.ar" =~ $patron;
"usuario@@servidor.com" =~ $patron;
"sin-dominio" =~ $patron;
"maria@equipo" =~ $patron;


"ana.perez@ejemplo.com" =~ $patron2;
"juan_23@universidad.edu.ar" =~ $patron2;
"usuario@@servidor.com" =~ $patron2;
"sin-dominio" =~ $patron2;
"maria@equipo" =~ $patron2;
```

JavaScript. `test` devuelve `true` cuando la cadena cumple la expresión regular y `false` en caso contrario.

```
let patron = /^[A-Za-z0-9._-]+@[A-Za-z0-9-]+\.[A-Za-z]{2,6}$/
patron.test("ana.perez@ejemplo.com")
patron.test("juan_23@universidad.edu.ar")
patron.test("usuario@@servidor.com")
patron.test("sin-dominio")
patron.test("maria@equipo")

let patron2 =
   /^[A-Za-z0-9._-]+@[A-Za-z0-9-]+(?:\.[A-Za-z0-9-]+)*\.[A-Za-z]{2,6}$/
patron2.test("ana.perez@ejemplo.com")
patron2.test("juan_23@universidad.edu.ar")
patron2.test("usuario@@servidor.com")
patron2.test("sin-dominio")
patron2.test("maria@equipo")
```

53. **Ejercicio:** Extracción de patrones

Un archivo contiene mediciones escritas de la siguiente forma:

```
temperatura = 23.5 C
presion = 1013 hPa
masa = 2.40 kg
velocidad = -3.2 m/s
dato incorrecto
```

Utilice una expresión regular para extraer de cada medición válida:

- el nombre de la magnitud;
- el valor numérico, incluyendo números negativos y decimales;
- la unidad.

a) Construya una expresión regular con tres grupos de captura: magnitud, valor y unidad.

b) Aplique la expresión al contenido y obtenga solamente las líneas que respetan el formato indicado.

c) Convierta el valor capturado a un valor numérico.

d) Guarde las mediciones válidas en una estructura o diccionario con los campos correspondientes a magnitud, valor y unidad.

e) Muestre las mediciones extraídas y compruebe que la línea `dato incorrecto` no fue incluida.

Una expresión regular apropiada es

```
^([A-Za-z]+) *= *([+-]?[0-9]+(?:[.][0-9]+)?) +([A-Za-z]+(?:/[A-Za-z]+)?)$
```

Los tres grupos capturan, respectivamente, la magnitud, el valor y la unidad. Los anclajes inicial y final hacen que deba coincidir la línea completa.

Los datos válidos extraídos son

```
temperatura   23.5     C
presion       1013     hPa
masa          2.40     kg
velocidad     -3.2     m/s
```

La línea `dato incorrecto` no coincide con la expresión y, por lo tanto, no aparece entre las mediciones extraídas.

JavaScript. `match` obtiene los grupos capturados de cada línea.

```
const lineas=[
"temperatura = 23.5 C",
"presion = 1013 hPa",
"masa = 2.40 kg",
"velocidad = -3.2 m/s",
"dato incorrecto"
]

const patron=/^([A-Za-z]+) *= *([+-]?[0-9]+(?:[.][0-9]+)?)
   +([A-Za-z]+(?:\/[A-Za-z]+)?)$/

const mediciones=[]
for(const linea of lineas){
    const m=linea.match(patron)
    if(m){
        mediciones.push({
            magnitud:m[1],
            valor:Number(m[2]),
            unidad:m[3]
        })
    }
}

console.log(mediciones)
```

Scilab. `regexp` devuelve en su cuarto resultado los textos capturados por los grupos entre paréntesis. Aquí las mediciones se guardan en una estructura cuyos campos contienen los datos extraídos.

```
lineas=[
"temperatura = 23.5 C"
"presion = 1013 hPa"
"masa = 2.40 kg"
"velocidad = -3.2 m/s"
"dato incorrecto"
```

```
];

texto=strcat(lineas,ascii(10));

patron="/^([A-Za-z]+) *= *([+-]?[0-9]+(?:[.][0-9]+)?)
   +([A-Za-z]+(?:/[A-Za-z]+)?)$/m";

[inicio,fin,coincidencias,cap]=regexp(texto,patron);

magnitud=cap(1:3:$);
valor=strtod(cap(2:3:$));
unidad=cap(3:3:$);

mediciones=struct( ..
    "magnitud",magnitud, ..
    "valor",valor, ..
    "unidad",unidad ..
);

mediciones
```

Octave. `regexp` con la opción `tokens` devuelve los grupos capturados. `cellfun` realiza el recorrido de las coincidencias de manera implícita.

```
lineas={
  "temperatura = 23.5 C"
  "presion = 1013 hPa"
  "masa = 2.40 kg"
  "velocidad = -3.2 m/s"
  "dato incorrecto"
};

texto=strjoin(lineas,"\n");

patron="^([A-Za-z]+) *= *([+-]?[0-9]+(?:[.][0-9]+)?)
   +([A-Za-z]+(?:/[A-Za-z]+)?)$";

tok=regexp(texto,patron,"tokens","lineanchors");

mediciones=cellfun(
  @(t) struct(
    "magnitud",t{1},
    "valor",str2double(t{2}),
    "unidad",t{3}
  ),
  tok,
  "UniformOutput",false
);

mediciones
```

Python. `findall` devuelve una tupla por cada coincidencia. `map` transforma implícitamente cada tupla en un diccionario.

```
import re

texto="""temperatura = 23.5 C
presion = 1013 hPa
masa = 2.40 kg
velocidad = -3.2 m/s
dato incorrecto"""

patron=r"^([A-Za-z]+) *= *([+-]?[0-9]+(?:[.][0-9]+)?)
   +([A-Za-z]+(?:/[A-Za-z]+)?)$"

datos=re.findall(patron,texto,re.MULTILINE)

mediciones=list(map(
    lambda t: {
        "magnitud":t[0],
        "valor":float(t[1]),
        "unidad":t[2]
    },
    datos
))

mediciones
```

Perl. La opción `g` obtiene todas las coincidencias y `m` permite que los anclajes correspondan al comienzo y al final de cada línea. Los grupos capturados se reciben consecutivamente. `map` construye los diccionarios sin escribir un ciclo explícito.

```
#!/usr/bin/env perl
use strict;
use warnings;
use Data::Dumper;

my $texto = "temperatura = 23.5 C\npresion = 1013 hPa\nmasa
   = 2.40 kg\nvelocidad = -3.2 m/s\ndato incorrecto";

my @cap = $texto =~ /^([A-Za-z]+) *=
   *([+-]?[0-9]+(?:\.[0-9]+)?)
   +([A-Za-z]+(?:\/[A-Za-z]+)?)$/gm;

my @mediciones;
for my $i (0 .. (@cap / 3 - 1)) {
    my %med;
    $med{magnitud} = $cap[3 * $i];
    $med{valor}    = 0 + $cap[3 * $i + 1];
    $med{unidad}   = $cap[3 * $i + 2];
    push @mediciones, \%med;
}
```

```
print Dumper(\@mediciones);
```

4.23.12. Simulación e integradores avanzados

La teoría necesaria está en la parte de modelos y simulación de la unidad. En todos los ejercicios el esquema es el mismo: parámetros, función de estado $\dot{Y} = F(t, Y)$, vector de tiempos, integrador, comprobación.

Scilab. `Y=ode(Y0,t0,t,F)`: `F(t,Y)` devuelve una columna; el resultado tiene una fila por variable de estado.

Octave. `[t,Y]=ode45(F,t,Y0)` devuelve una fila por instante y una columna por variable de estado; `odeset` fija opciones.

Python. `sol=solve_ivp(F,[t0,tf],Y0,t_eval=t)`: `Y0` lista o arreglo, `F(t,Y)` devuelve lista o arreglo, resultado en `sol.y` con una fila por variable; `max_step` fuerza el paso; `events=` detiene en una condición.

Perl. Perl y PDL solos no traen integrador de EDO, pero con un módulo aparte se hace toda la sección: `Math::ODE` (Perl puro, Runge–Kutta 4 de paso fijo, `apt install libmath-ode-perl`), que es el que usan los listados; `Math::RungeKutta` (`cpan Math::RungeKutta`), con paso adaptativo como `ode` de Scilab; o `Math::GSL::ODEIV` (`apt install libmath-gsl-perl`), con los integradores de GSL incluidos los rígidos. Funcionan igual en `perl -de1` y en `pdl2`. El esquema es el mismo de las otras calculadoras: parámetros, función de estado que devuelve la lista de derivadas, condición inicial, tiempos y paso, `evolve` para integrar, y los resultados pasados a piddles para comprobar. Como `Math::ODE` es de paso fijo, los ejercicios que piden repetir con distintos pasos muestran el efecto directamente.

JavaScript. Con math.js solo no se puede integrar EDO; con `odex` sí. Los listados cargan el módulo con `require(.ºdex")`, definen la función de estado y obtienen la solución con `Solver.integrate`.

54. **Ejercicio:** Enfriamiento de Newton

    Dado

    $$\frac{dT}{dt} = -k(T - T_{\text{amb}})$$

    con

    $$T(0) = 90\,^{\circ}\text{C}, \qquad T_{\text{amb}} = 20\,^{\circ}\text{C}, \qquad k = 0{,}05\ \text{min}^{-1}.$$

    - (a) simule, y defina los parámetros y la función de la EDO;
    - (b) resuelva en $[0, 60]$ min;
    - (c) compare con la solución analítica

    $$T(t) = T_{\text{amb}} + (T_0 - T_{\text{amb}})e^{-kt};$$

- (d) calcule el error final
$$|T(60) - T_{\text{exacta}}(60)|;$$
- (e) repita con pasos máximos 1, 0,1 y 0,01 min (o esos pasos directamente cuando el integrador sea de paso fijo) y compare los errores;
- (f) determine a qué tiempo alcanza 40 °C, analítica y numéricamente.

El tiempo analítico se obtiene de

$$40 = T_{\text{amb}} + (T_0 - T_{\text{amb}})e^{-kt},$$

por lo que

$$t_{40} = -\frac{1}{k}\log\left(\frac{40 - T_{\text{amb}}}{T_0 - T_{\text{amb}}}\right).$$

Al disminuir el paso máximo, en general se obtiene una solución numérica más precisa. En los integradores adaptativos el error también está controlado por sus tolerancias internas, por lo que reducir indefinidamente el paso máximo no necesariamente produce una mejora proporcional.

Scilab. `ode` es adaptativo. El vector de tiempos indica dónde se desea obtener la solución; el paso interno máximo se fija mediante el cuarto elemento, `hmax`, de `%ODEOPTIONS`.

```
T0=90
Tamb=20
k=0.05

function dT=f(t,T)
    dT=-k*(T-Tamb)
endfunction

t=0:0.1:60
%ODEOPTIONS=[1,0,0,0.1,0,2,50000,12,5,0,-1,-1]
T=ode(T0,0,t,f)

Texacta=Tamb+(T0-Tamb)*exp(-k*t)
error_final=abs(T($)-Texacta($))

h1=1
%ODEOPTIONS=[1,0,0,h1,0,2,50000,12,5,0,-1,-1]
T1=ode(T0,0,60,f)
e1=abs(T1-(Tamb+(T0-Tamb)*exp(-k*60)))

h2=0.1
%ODEOPTIONS=[1,0,0,h2,0,2,50000,12,5,0,-1,-1]
T2=ode(T0,0,60,f)
e2=abs(T2-(Tamb+(T0-Tamb)*exp(-k*60)))

h3=0.01
%ODEOPTIONS=[1,0,0,h3,0,2,50000,12,5,0,-1,-1]
```

```
T3=ode(T0,0,60,f)
e3=abs(T3-(Tamb+(T0-Tamb)*exp(-k*60)))

e1,e2,e3

t40=-log((40-Tamb)/(T0-Tamb))/k

%ODEOPTIONS=[1,0,0,0.1,0,2,50000,12,5,0,-1,-1]
T=ode(T0,0,t,f)
i40=find(T<=40,1)
t40_numerico=t(i40)

clear %ODEOPTIONS
```

Octave. `ode45` usa paso adaptativo. `MaxStep` limita el paso interno máximo.

```
T0=90;
Tamb=20;
k=0.05;

f=@(t,T) -k*(T-Tamb);

t=(0:0.1:60)';
op=odeset("MaxStep",0.1);
[t,T]=ode45(f,t,T0,op);

Texacta=Tamb+(T0-Tamb)*exp(-k*t);
error_final=abs(T(end)-Texacta(end))

h1=1;
op1=odeset("MaxStep",h1);
[t1,T1]=ode45(f,[0 60],T0,op1);
e1=abs(T1(end)-(Tamb+(T0-Tamb)*exp(-k*60)));

h2=0.1;
op2=odeset("MaxStep",h2);
[t2,T2]=ode45(f,[0 60],T0,op2);
e2=abs(T2(end)-(Tamb+(T0-Tamb)*exp(-k*60)));

h3=0.01;
op3=odeset("MaxStep",h3);
[t3,T3]=ode45(f,[0 60],T0,op3);
e3=abs(T3(end)-(Tamb+(T0-Tamb)*exp(-k*60)));

e1,e2,e3

t40=-log((40-Tamb)/(T0-Tamb))/k

i40=find(T<=40,1);
t40_numerico=t(i40)
```

Python. `solve_ivp` usa paso adaptativo. `max_step` limita el paso interno máximo.

```
import numpy as np
from scipy.integrate import solve_ivp

T0=90
Tamb=20
k=0.05

def f(t,T):
    return -k*(T-Tamb)

t=np.arange(0,60.05,0.1)

T=solve_ivp(
    f,[0,60],[T0],
    t_eval=t,max_step=0.1
).y[0]

Texacta=Tamb+(T0-Tamb)*np.exp(-k*t)
error_final=abs(T[-1]-Texacta[-1])

h1=1
T1=solve_ivp(
    f,[0,60],[T0],
    t_eval=[60],max_step=h1
).y[0,-1]
e1=abs(T1-(Tamb+(T0-Tamb)*np.exp(-k*60)))

h2=0.1
T2=solve_ivp(
    f,[0,60],[T0],
    t_eval=[60],max_step=h2
).y[0,-1]
e2=abs(T2-(Tamb+(T0-Tamb)*np.exp(-k*60)))

h3=0.01
T3=solve_ivp(
    f,[0,60],[T0],
    t_eval=[60],max_step=h3
).y[0,-1]
e3=abs(T3-(Tamb+(T0-Tamb)*np.exp(-k*60)))

e1,e2,e3

t40=-np.log((40-Tamb)/(T0-Tamb))/k

i40=np.where(T<=40)[0][0]
t40_numerico=t[i40]
```

Perl. `Math::ODE` utiliza aquí un método de paso fijo, por lo que los valores indicados son directamente los pasos de integración.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use Math::ODE;

my $T0   = 90;
my $Tamb = 20;
my $k    = 0.05;

# Ley de enfriamiento: T' = -k*(T - Tamb)
sub enf {
    my ($t, $T) = @_;
    return -$k * ($T->[0] - $Tamb);
}

sub final_value {
    my ($h, $T) = @_;
    my %args;
    $args{step}    = $h;
    $args{initial} = [$T0];
    $args{ODE}     = [\&enf];
    $args{t0}      = 0;
    $args{tf}      = $T;
    $args{verbose} = 0;
    my $o = Math::ODE->new(%args);
    $o->evolve;
    my @t = sort { $a <=> $b } keys %{$o->{values}};
    return $o->{values}{$t[-1]}[0];
}

# Trayectoria completa con paso 0.1
my %args_tray;
$args_tray{step}    = 0.1;
$args_tray{initial} = [$T0];
$args_tray{ODE}     = [\&enf];
$args_tray{t0}      = 0;
$args_tray{tf}      = 60;
$args_tray{verbose} = 0;
my $o = Math::ODE->new(%args_tray);
$o->evolve;

my @t_vals = sort { $a <=> $b } keys %{$o->{values}};
my $T = pdl(map { $o->{values}{$_}[0] } @t_vals);
my $t = pdl(@t_vals);

my $Texacta = $Tamb + ($T0 - $Tamb) * exp(-$k * $t);
my $error_final = abs($T->at(-1) - $Texacta->at(-1));
print "error_final = $error_final\n";
```

```
my $h1 = 1;
my $h2 = 0.1;
my $h3 = 0.01;

my $T1 = final_value($h1, 60);
my $T2 = final_value($h2, 60);
my $T3 = final_value($h3, 60);

my $Tf_exacta = $Tamb + ($T0 - $Tamb) * exp(-$k * 60);
my $e1 = abs($T1 - $Tf_exacta);
my $e2 = abs($T2 - $Tf_exacta);
my $e3 = abs($T3 - $Tf_exacta);

print "e1=$e1, e2=$e2, e3=$e3\n";
my $errores = pdl($e1, $e2, $e3);

# Tiempo en que T alcanza 40 grados
my $t40 = -log((40 - $Tamb) / ($T0 - $Tamb)) / $k;
print "t40 = $t40\n";

my $i40 = which($T <= 40)->at(0);
my $t40_numerico = $t->at($i40);
print "t40_numerico = $t40_numerico\n";
```

JavaScript. Con `odex` el integrador es adaptativo y `maxStepSize` limita el paso máximo. Para localizar numéricamente el cruce de 40 °C se evalúa la solución sobre una grilla y se utiliza `find`, sin escribir un ciclo explícito.

```
const odex=require("odex")

let T0 = 90
let Tamb = 20
let k = 0.05

let f = (t,T)=>[-k*(T[0]-Tamb)]

let resolver = h=>{
    const s=new odex.Solver(f,1)
    s.maxStepSize=h
    return s.integrate(0,[T0])
}

let sol = resolver(0.1)

let T60 = sol(60)[0]
let Tex60 = Tamb+(T0-Tamb)*Math.exp(-k*60)
let errorFinal = Math.abs(T60-Tex60)

let sol1 = resolver(1)
let e1 = Math.abs(sol1(60)[0]-Tex60)
```

```
let sol2 = resolver(0.1)
let e2 = Math.abs(sol2(60)[0]-Tex60)

let sol3 = resolver(0.01)
let e3 = Math.abs(sol3(60)[0]-Tex60)

console.log(e1,e2,e3)

let t40 = -Math.log((40-Tamb)/(T0-Tamb))/k

const tiempos = Array.from(
    {length:601},
    (_,i)=>i/10
)

let sol_t40 = resolver(0.1)
let t40_numerico = tiempos.slice(1).find(
    t=>sol_t40(t)[0]<=40
)

t40
t40_numerico
```

55. **Ejercicio:** Tiro parabólico sin resistencia

Para un proyectil lanzado desde el suelo y que vuelve a la misma altura,

$$x(t) = v_0 \cos\theta\, t, \qquad y(t) = v_0 \sin\theta\, t - \frac{1}{2} g t^2.$$

Considere $v_0 = 20$ m/s, $g = 9{,}8$ m/s^2 y los ángulos

$$10^\circ,\ 20^\circ,\ 30^\circ,\ 40^\circ,\ 45^\circ,\ 50^\circ,\ 60^\circ,\ 70^\circ,\ 80^\circ.$$

- (a) calcule para cada ángulo el alcance

$$R = \frac{v_0^2 \sin(2\theta)}{g};$$

- (b) determine cuál de los ángulos considerados produce el mayor alcance;
- (c) explique por qué, en ausencia de resistencia del aire y cuando el proyectil parte y llega a la misma altura, el alcance máximo se obtiene para $\theta = 45^\circ$.

Como v_0 y g son constantes, el alcance depende solamente de

$$\sin(2\theta).$$

Esta función alcanza su valor máximo, igual a 1, cuando

$$2\theta = 90°,$$

por lo tanto

$$\theta = 45°.$$

Además, los ángulos complementarios producen el mismo alcance porque

$$\sin(2\theta) = \sin(180° - 2\theta).$$

Por ejemplo, 30° y 60° tienen el mismo alcance, al igual que 40° y 50°.

Scilab. La expresión del alcance se evalúa directamente sobre todo el vector de ángulos.

```
v0=20
g=9.8

grados=[10 20 30 40 45 50 60 70 80]
th=grados*%pi/180

alcance=v0^2*sin(2*th)/g

[alcance_maximo,k]=max(alcance)
angulo_maximo=grados(k)
```

Octave. La expresión se evalúa directamente sobre todo el vector de ángulos.

```
v0=20;
g=9.8;

grados=[10 20 30 40 45 50 60 70 80];
th=grados*pi/180;

alcance=v0^2*sin(2*th)/g;

[alcance_maximo,k]=max(alcance)
angulo_maximo=grados(k)
```

Python. NumPy evalúa la expresión sobre todo el arreglo.

```
import numpy as np

v0=20
g=9.8

grados=np.array([10,20,30,40,45,50,60,70,80])
th=np.deg2rad(grados)
```

```
alcance=v0**2*np.sin(2*th)/g

alcance_maximo=np.max(alcance)
angulo_maximo=grados[np.argmax(alcance)]
```

Perl. PDL evalúa la expresión sobre todo el piddle.

```
use PDL;
use PDL::Constants qw(PI);




$alcance;
$alcance_maximo;
$angulo_maximo;
```

JavaScript. Con math.js la expresión puede aplicarse a todo el arreglo de ángulos.

```
const math = require('mathjs')
let v0 = 20
let g = 9.8

const grados = [10,20,30,40,45,50,60,70,80]
let th = math.dotMultiply(grados,Math.PI/180)

let alcance = math.dotMultiply(
    v0**2/g,
    math.map(math.dotMultiply(2,th), math.sin)
)

let alcance_maximo = Math.max(...alcance)
let k = alcance.indexOf(alcance_maximo)
let angulo_maximo = grados[k]
```

56. **Ejercicio:** Proyectil con resistencia del aire

Considere

$$\dot{x} = v_x, \qquad \dot{y} = v_y,$$

$$\dot{v}_x = -\frac{b}{m}v_x, \qquad \dot{v}_y = -g - \frac{b}{m}v_y,$$

con

$$m = 1 \text{ kg}, \qquad b = 0{,}1 \text{ kg/s}, \qquad g = 9{,}8 \text{ m/s}^2,$$

$$v_0 = 50 \text{ m/s}, \qquad \theta = 45^\circ.$$

- (a) escriba el problema como cuatro EDO de primer orden con estado

$$[x, y, v_x, v_y];$$

- (b) implemente el sistema y obtenga la trayectoria;
- (c) repita el cálculo con $b = 0$ y compare el alcance y el tiempo de vuelo;
- (d) con $b = 0{,}1$ kg/s repita para 30°, 45° y 60° y determine cuál de esos ángulos produce el mayor alcance;
- (e) explique por qué, cuando existe resistencia del aire, el ángulo de alcance máximo ya no tiene por qué ser 45°.

La condición inicial es

$$x(0) = 0, \qquad y(0) = 0,$$

$$v_x(0) = v_0 \cos\theta, \qquad v_y(0) = v_0 \sin\theta.$$

En los cálculos siguientes se utiliza una separación temporal de 0,01 s para estimar el instante en que el proyectil vuelve al suelo. El alcance se aproxima mediante la última posición calculada con $y \geq 0$.

Sin resistencia del aire, la simetría de la trayectoria hace que el alcance máximo corresponda a 45°. Cuando existe resistencia, la velocidad horizontal disminuye durante el movimiento y las trayectorias con mayor tiempo de vuelo sufren durante más tiempo el efecto del rozamiento. Por lo tanto, se pierde la simetría entre ángulos complementarios y el ángulo óptimo no tiene por qué ser 45°.

Scilab. Los distintos casos se escriben explícitamente. `find` permite localizar los puntos calculados antes de que el proyectil pase por debajo del suelo.

```
global m b g

m=1
g=9.8
v0=50

function ds=proy(t,s)
    global m b g
    ds=[
        s(3)
        s(4)
        -(b/m)*s(3)
        -g-(b/m)*s(4)
    ]
endfunction
```

```
t=0:0.01:12

b=0.1
th=45*%pi/180
s0=[0;0;v0*cos(th);v0*sin(th)]

S45=ode(s0,0,t,proy)
k45=find(S45(2,:)>=0)

a45=S45(1,k45($))
vuelo45=t(k45($))

b=0
th=45*%pi/180
s0=[0;0;v0*cos(th);v0*sin(th)]

S0=ode(s0,0,t,proy)
k0=find(S0(2,:)>=0)

a0=S0(1,k0($))
vuelo0=t(k0($))

comparacion=[
    a45 vuelo45
    a0 vuelo0
]

b=0.1

th=30*%pi/180
s0=[0;0;v0*cos(th);v0*sin(th)]
S30=ode(s0,0,t,proy)
k30=find(S30(2,:)>=0)
a30=S30(1,k30($))

th=45*%pi/180
s0=[0;0;v0*cos(th);v0*sin(th)]
S45=ode(s0,0,t,proy)
k45=find(S45(2,:)>=0)
a45=S45(1,k45($))

th=60*%pi/180
s0=[0;0;v0*cos(th);v0*sin(th)]
S60=ode(s0,0,t,proy)
k60=find(S60(2,:)>=0)
a60=S60(1,k60($))

alcances=[a30 a45 a60]
angulos=[30 45 60]
```

```
[alcance_maximo,j]=max(alcances)
angulo_maximo=angulos(j)
```

Octave. Una función auxiliar resuelve un caso. Las diferentes combinaciones de coeficiente de resistencia y ángulo se invocan explícitamente.

```
function [t,x,y,alcance,vuelo]=trayectoria(beta,grados)

  m=1;
  g=9.8;
  v0=50;

  th=grados*pi/180;

  f=@(t,s) [
    s(3);
    s(4);
    -(beta/m)*s(3);
    -g-(beta/m)*s(4)
  ];

  tm=(0:0.01:12)';

  [tm,S]=ode45(
    f,tm,
    [0;0;v0*cos(th);v0*sin(th)]
  );

  k=find(S(:,2)>=0);

  t=tm(k);
  x=S(k,1);
  y=S(k,2);

  alcance=x(end);
  vuelo=t(end);

endfunction

[t45,x45,y45,a45,v45]=trayectoria(0.1,45);
[t0,x0,y0,a0,v0t]=trayectoria(0,45);

comparacion=[
  a45  v45
  a0   v0t
]

[t30,x30,y30,a30,v30]=trayectoria(0.1,30);
[t45,x45,y45,a45,v45]=trayectoria(0.1,45);
[t60,x60,y60,a60,v60]=trayectoria(0.1,60);
```

```
alcances=[a30 a45 a60];
angulos=[30 45 60];

[alcance_maximo,j]=max(alcances);

angulo_maximo=angulos(j)
```

Python. Una función auxiliar resuelve cada caso. Las máscaras booleanas permiten conservar solamente la parte de la trayectoria que permanece sobre el suelo.

```
import numpy as np
from scipy.integrate import solve_ivp

m=1
g=9.8
v0=50

def trayectoria(beta,grados):

    th=np.deg2rad(grados)

    def f(t,s):
        return [
            s[2],
            s[3],
            -(beta/m)*s[2],
            -g-(beta/m)*s[3]
        ]

    t=np.linspace(0,12,1201)

    S=solve_ivp(
        f,[0,12],
        [0,0,v0*np.cos(th),v0*np.sin(th)],
        t_eval=t
    ).y

    k=S[1]>=0

    tt=t[k]
    x=S[0,k]
    y=S[1,k]

    return tt,x,y,x[-1],tt[-1]

t45,x45,y45,a45,v45=trayectoria(0.1,45)
t0,x0,y0,a0,v0t=trayectoria(0,45)
```

```
comparacion=np.array([
    [a45,v45],
    [a0,v0t]
])

t30,x30,y30,a30,v30=trayectoria(0.1,30)
t45,x45,y45,a45,v45=trayectoria(0.1,45)
t60,x60,y60,a60,v60=trayectoria(0.1,60)

alcances=np.array([a30,a45,a60])
angulos=np.array([30,45,60])

alcance_maximo=np.max(alcances)
angulo_maximo=angulos[np.argmax(alcances)]
```

Perl. Con `Math::ODE` la trayectoria completa se convierte a PDL. `which` permite localizar los puntos que permanecen sobre el suelo sin escribir un ciclo explícito.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use PDL::Constants qw(PI);
use Math::ODE;

my $m  = 1;
my $g  = 9.8;
my $v0 = 50;

sub trayectoria {
    my ($beta, $grados) = @_;
    my $th = $grados * PI / 180;
    my $vx0 = $v0 * cos($th);
    my $vy0 = $v0 * sin($th);

    my $f_x  = sub { my ($t, $s) = @_; return $s->[2]; };
    my $f_y  = sub { my ($t, $s) = @_; return $s->[3]; };
    my $f_vx = sub { my ($t, $s) = @_; return -($beta / $m)
       * $s->[2]; };
    my $f_vy = sub { my ($t, $s) = @_; return -$g - ($beta /
       $m) * $s->[3]; };

    my %args;
    $args{step}    = 0.01;
    $args{initial} = [0, 0, $vx0, $vy0];
    $args{ODE}     = [$f_x, $f_y, $f_vx, $f_vy];
    $args{t0}      = 0;
    $args{tf}      = 12;
    $args{verbose} = 0;
    my $o = Math::ODE->new(%args);
    $o->evolve;
```

```
    my @t_vals = sort { $a <=> $b } keys %{$o->{values}};
    my @x = map { $o->{values}{$_}[0] } @t_vals;
    my @y = map { $o->{values}{$_}[1] } @t_vals;

    # conservar solo y >= 0
    my @valid;
    for my $i (0 .. $#y) {
        push @valid, $i if $y[$i] >= 0;
    }
    my @tt = @t_vals[@valid];
    my @xx = @x[@valid];
    my @yy = @y[@valid];

    return (\@tt, \@xx, \@yy, $xx[-1], $tt[-1]);
}

my ($t45, $x45, $y45, $a45, $v45) = trayectoria(0.1, 45);
my ($t0, $x0, $y0, $a0, $v0t)      = trayectoria(0, 45);

my $comparacion = pdl([$a45, $v45], [$a0, $v0t]);
print "comparacion = $comparacion\n";

my ($t30, $x30, $y30, $a30, $v30) = trayectoria(0.1, 30);
($t45, $x45, $y45, $a45, $v45)     = trayectoria(0.1, 45);
my ($t60, $x60, $y60, $a60, $v60) = trayectoria(0.1, 60);

my $alcances = pdl($a30, $a45, $a60);
my $angulos  = pdl(30, 45, 60);

my $alcance_maximo = max($alcances);
my $kmax = $alcances->maximum_ind;
my $angulo_maximo = $angulos->at($kmax);

print "alcances = $alcances\n";
print "angulo_maximo = $angulo_maximo\n";
```

JavaScript. Con `odex` se obtiene una función que representa la solución. Se evalúa sobre tiempos crecientes mediante `map`; `findIndex` localiza el primer punto que queda debajo del suelo. No se escriben ciclos explícitos.

```
const odex=require("odex")

let m = 1
let g = 9.8
let v0 = 50

let trayectoria = (beta,grados)=>{

    let th = grados*Math.PI/180
```

```
    let f = (t,s)=>[
        s[2],
        s[3],
        -(beta/m)*s[2],
        -g-(beta/m)*s[3]
    ]

    const s0 = [
        0,
        0,
        v0*Math.cos(th),
        v0*Math.sin(th)
    ]

    const sol = new odex.Solver(f,4).integrate(0,s0)

    const tiempos = Array.from(
        {length:1200},
        (_,i)=>(i+1)/100
    )

    let estados = tiempos.map(t=>sol(t))

    let k = estados.findIndex(s=>s[1]<0)

    let alcance = estados[k-1][0]
    let vuelo = tiempos[k-1]

    return {
        alcance:alcance,
        vuelo:vuelo
    }
}

let conAire45 = trayectoria(0.1,45)
let sinAire45 = trayectoria(0,45)

const comparacion = [
    conAire45,
    sinAire45
]

let sol30 = trayectoria(0.1,30)
let sol45 = trayectoria(0.1,45)
let sol60 = trayectoria(0.1,60)

const alcances = [
    sol30.alcance,
    sol45.alcance,
    sol60.alcance
]
```

```
const angulos = [30,45,60]

let kmax = alcances.indexOf(
    Math.max(...alcances)
)

let angulo_maximo = angulos[kmax]
```

57. **Ejercicio:** Oscilador armónico simple

Considere el oscilador forzado y amortiguado

$$m\ddot{x} + b\dot{x} + kx = F_0 \cos(\omega t),$$

con

$$m = 1 \text{ kg}, \qquad k = 4 \text{ N/m}, \qquad F_0 = 1 \text{ N},$$

y condiciones iniciales

$$x(0) = 0, \qquad \dot{x}(0) = 0.$$

- (a) escriba la ecuación como un sistema de dos EDO de primer orden y resuélvalo en $[0, 30]$ s;
- (b) para $\omega = 2$ rad/s compare las respuestas correspondientes a

$$b = 0{,}5, \qquad b = 2\sqrt{mk}, \qquad b = 6 \text{ kg/s};$$

- (c) con $b = 0{,}5$ kg/s repita para $\omega = 1$, 2 y 3 rad/s. Estime la amplitud estacionaria como

$$A = \max_{t>20} |x(t)|$$

y determine cuál de las tres frecuencias produce la mayor amplitud.

Definiendo

$$v = \dot{x},$$

el sistema de primer orden es

$$\dot{x} = v, \qquad \dot{v} = \frac{F_0 \cos(\omega t) - bv - kx}{m}.$$

El amortiguamiento crítico es

$$b_c = 2\sqrt{mk} = 4 \text{ kg/s}.$$

Por lo tanto,

$$b = 0{,}5 < b_c$$

corresponde al régimen subamortiguado,

$$b = 4 = b_c$$

al régimen críticamente amortiguado, y

$$b = 6 > b_c$$

al régimen sobreamortiguado.

Esta clasificación describe principalmente la parte transitoria de la respuesta. Como existe una fuerza externa periódica, después de que disminuye el transitorio permanece una respuesta periódica con la frecuencia de excitación.

La frecuencia natural sin amortiguamiento es

$$\omega_0 = \sqrt{\frac{k}{m}} = 2 \text{ rad/s}.$$

Por ello, entre las frecuencias 1, 2 y 3 rad/s, se espera la mayor amplitud para una frecuencia próxima a 2 rad/s. El cálculo numérico permite comprobarlo.

Scilab. Se escriben explícitamente los distintos casos y se utilizan operaciones vectoriales para calcular las amplitudes.

```
global m k F0 b w

m=1
k=4
F0=1

function dY=osc(t,Y)
    global m k F0 b w
    dY=[
        Y(2)
        (F0*cos(w*t)-b*Y(2)-k*Y(1))/m
    ]
endfunction

t=linspace(0,30,3001)

b=0.5
w=2
Y05=ode([0;0],0,t,osc)
x05=Y05(1,:)

b=2*sqrt(m*k)
```

```
w=2
Yc=ode([0;0],0,t,osc)
xc=Yc(1,:)

b=6
w=2
Y6=ode([0;0],0,t,osc)
x6=Y6(1,:)

b=0.5

w=1
Y1=ode([0;0],0,t,osc)
x1=Y1(1,:)
a1=max(abs(x1(t>20)))

w=2
Y2=ode([0;0],0,t,osc)
x2=Y2(1,:)
a2=max(abs(x2(t>20)))

w=3
Y3=ode([0;0],0,t,osc)
x3=Y3(1,:)
a3=max(abs(x3(t>20)))

amplitudes=[a1 a2 a3]
frecuencias=[1 2 3]

[amplitud_maxima,j]=max(amplitudes)
frecuencia_maxima=frecuencias(j)
```

Octave. Una función auxiliar resuelve un caso. Las distintas combinaciones de amortiguamiento y frecuencia se evalúan mediante llamadas explícitas.

```
function [t,x,amp]=simular_oscilador(b,w)

  m=1;
  k=4;
  F0=1;

  f=@(t,Y) [
    Y(2);
    (F0*cos(w*t)-b*Y(2)-k*Y(1))/m
  ];

  t=linspace(0,30,3001)';

  [t,Y]=ode45(f,t,[0;0]);
```

```
  x=Y(:,1);

  amp=max(abs(x(t>20)));

endfunction

[t,x05,a05]=simular_oscilador(0.5,2);

[t,xc,ac]=simular_oscilador(
    2*sqrt(1*4),2
);

[t,x6,a6]=simular_oscilador(6,2);

[t,x1,a1]=simular_oscilador(0.5,1);
[t,x2,a2]=simular_oscilador(0.5,2);
[t,x3,a3]=simular_oscilador(0.5,3);

amplitudes=[a1 a2 a3];
frecuencias=[1 2 3];

[amplitud_maxima,j]=max(amplitudes);

frecuencia_maxima=frecuencias(j)
```

Python. NumPy permite seleccionar directamente la parte de la trayectoria con $t > 20$ y calcular allí la amplitud máxima.

```
import numpy as np
from scipy.integrate import solve_ivp

m=1
k=4
F0=1

def simular_oscilador(b,w):

    def osc(t,Y):
        return [
            Y[1],
            (F0*np.cos(w*t)-b*Y[1]-k*Y[0])/m
        ]

    t=np.linspace(0,30,3001)

    x=solve_ivp(
        osc,[0,30],[0,0],
        t_eval=t
    ).y[0]
```

```
    amp=np.max(np.abs(x[t>20]))

    return t,x,amp

t,x05,a05=simular_oscilador(0.5,2)
t,xc,ac=simular_oscilador(2*np.sqrt(m*k),2)
t,x6,a6=simular_oscilador(6,2)

t,x1,a1=simular_oscilador(0.5,1)
t,x2,a2=simular_oscilador(0.5,2)
t,x3,a3=simular_oscilador(0.5,3)

amplitudes=np.array([a1,a2,a3])
frecuencias=np.array([1,2,3])

amplitud_maxima=np.max(amplitudes)

frecuencia_maxima=frecuencias[
    np.argmax(amplitudes)
]
```

Perl. La solución de `Math::ODE` se convierte en un objeto PDL. Así puede seleccionarse $t > 20$ y calcularse el máximo sin escribir un ciclo explícito.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use Math::ODE;

my $m  = 1;
my $k  = 4;
my $F0 = 1;

sub simular_oscilador {
    my ($beta, $omega) = @_;

    my $f_x = sub { my ($t, $s) = @_; return $s->[1]; };
    my $f_v = sub {
        my ($t, $s) = @_;
        return ($F0 * cos($omega * $t) - $beta * $s->[1] -
           $k * $s->[0]) / $m;
    };

    my %args;
    $args{step}    = 0.01;
    $args{initial} = [0, 0];
    $args{ODE}     = [$f_x, $f_v];
    $args{t0}      = 0;
    $args{tf}      = 30;
    $args{verbose} = 0;
```

```
    my $o = Math::ODE->new(%args);
    $o->evolve;

    my @t_vals = sort { $a <=> $b } keys %{$o->{values}};
    my $t = pdl(@t_vals);
    my $x = pdl(map { $o->{values}{$_}[0] } @t_vals);

    my $mask = $t > 20;
    my $amp  = max(abs($x->where($mask)));
    return ($amp, $t, $x);
}

my ($a05, $t05, $x05) = simular_oscilador(0.5, 2);
my ($ac, $tc, $xc)    = simular_oscilador(2 * sqrt($m * $k),
   2);
my ($a6, $t6, $x6)    = simular_oscilador(6, 2);

my ($a1, $t1, $x1) = simular_oscilador(0.5, 1);
my ($a2, $t2, $x2) = simular_oscilador(0.5, 2);
my ($a3, $t3, $x3) = simular_oscilador(0.5, 3);

my $amplitudes = pdl($a1, $a2, $a3);
my $frecuencias = pdl(1, 2, 3);

my $amplitud_maxima = max($amplitudes);
my $kmax = $amplitudes->maximum_ind;
my $frecuencia_maxima = $frecuencias->at($kmax);

print "amplitudes = $amplitudes\n";
print "frecuencia_maxima = $frecuencia_maxima\n";
```

JavaScript. Con `odex` se obtiene una función que representa la solución. La trayectoria para $t > 20$ se evalúa mediante `map`, sin escribir ciclos explícitos.

```
const odex=require("odex")

let m = 1
let k = 4
let F0 = 1

let oscilador = (b,w)=>{

    let f = (t,Y)=>[
        Y[1],
        (
            F0*Math.cos(w*t)
            -b*Y[1]
            -k*Y[0]
        )/m
    ]
```

```
    return new odex.Solver(
        f,2
    ).integrate(0,[0,0])
}

let sol05 = oscilador(0.5,2)
let solCritico = oscilador(
    2*Math.sqrt(m*k),2
)
let sol6 = oscilador(6,2)

let solw1 = oscilador(0.5,1)
let solw2 = oscilador(0.5,2)
let solw3 = oscilador(0.5,3)

const tiempos = Array.from(
    {length:1001},
    (_,i)=>20+i/100
)

let a1 = Math.max(
    ...tiempos.map(
        t=>Math.abs(solw1(t)[0])
    )
)

let a2 = Math.max(
    ...tiempos.map(
        t=>Math.abs(solw2(t)[0])
    )
)

let a3 = Math.max(
    ...tiempos.map(
        t=>Math.abs(solw3(t)[0])
    )
)

const amplitudes = [a1,a2,a3]
const frecuencias = [1,2,3]

let amplitud_maxima = Math.max(...amplitudes)

let kmax = amplitudes.indexOf(
    amplitud_maxima
)

let frecuencia_maxima = frecuencias[kmax]
```

58. **Ejercicio:** Crecimiento logístico poblacional

Considere el modelo

$$\dot{N} = rN\left(1 - \frac{N}{K}\right), \qquad r = 0{,}3 \text{ ao}^{-1}, \qquad K = 1000, \qquad N(0) = 50.$$

- (a) resuelva numéricamente el problema en $[0, 40]$ años;
- (b) compare con la solución analítica

$$N(t) = \frac{K}{1 + \dfrac{K - N_0}{N_0} e^{-rt}}$$

  y calcule el error máximo en el intervalo;
- (c) determine numéricamente cuándo la población crece más rápido y verifique que en ese instante se cumple aproximadamente

$$N = \frac{K}{2}.$$

La velocidad de crecimiento es

$$v(N) = rN\left(1 - \frac{N}{K}\right) = rN - \frac{r}{K}N^2.$$

Es una parábola cóncava cuyo máximo se encuentra en

$$N = \frac{K}{2}.$$

Para los datos del problema,

$$\frac{K}{2} = 500.$$

Por lo tanto, al buscar el máximo de $\dot{N}$ sobre la solución numérica, la población correspondiente debe resultar aproximadamente igual a 500.

A partir de la solución analítica también puede obtenerse el instante exacto correspondiente:

$$t_{\text{máx}} = \frac{1}{r} \log\left(\frac{K - N_0}{N_0}\right).$$

Scilab. Las operaciones sobre toda la trayectoria se realizan vectorialmente.

```
global r K

r=0.3
K=1000
N0=50
```

```
function dN=f(t,N)
    global r K
    dN=r*N*(1-N/K)
endfunction

t=linspace(0,40,401)

N=ode(N0,0,t,f)

Nex=K./(1+(K-N0)/N0*exp(-r*t))

error=abs(N-Nex)
error_maximo=max(error)

v=r*N.*(1-N/K)

[vmax,kmax]=max(v)

tiempo_maximo=t(kmax)
poblacion_en_maximo=N(kmax)

K/2

tmax_exacto=log((K-N0)/N0)/r
```

Octave. Las operaciones sobre toda la trayectoria se realizan vectorialmente.

```
r=0.3;
K=1000;
N0=50;

f=@(t,N) r*N*(1-N/K);

t=linspace(0,40,401)';

[t,N]=ode45(f,t,N0);

Nex=K./(1+(K-N0)/N0*exp(-r*t));

error=abs(N-Nex);
error_maximo=max(error)

v=r*N.*(1-N/K);

[vmax,kmax]=max(v);

tiempo_maximo=t(kmax)
poblacion_en_maximo=N(kmax)
```

```
K/2

tmax_exacto=log((K-N0)/N0)/r
```

Python. NumPy permite realizar directamente las operaciones sobre toda la trayectoria.

```
import numpy as np
from scipy.integrate import solve_ivp

r=0.3
K=1000
N0=50

def f(t,N):
    return r*N*(1-N/K)

t=np.linspace(0,40,401)

N=solve_ivp(
    f,[0,40],[N0],
    t_eval=t
).y[0]

Nex=K/(1+(K-N0)/N0*np.exp(-r*t))

error=np.abs(N-Nex)
error_maximo=np.max(error)

v=r*N*(1-N/K)

kmax=np.argmax(v)

tiempo_maximo=t[kmax]
poblacion_en_maximo=N[kmax]

K/2

tmax_exacto=np.log((K-N0)/N0)/r
```

Perl. La salida de `Math::ODE` se convierte a PDL para trabajar vectorialmente con toda la trayectoria.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use Math::ODE;

my $r  = 0.3;
```

```
my $K  = 1000;
my $N0 = 50;

# Modelo logistico: N' = r*N*(1 - N/K)
sub logi {
    my ($t, $N) = @_;
    return $r * $N->[0] * (1 - $N->[0] / $K);
}

my %args;
$args{step}    = 0.1;
$args{initial} = [$N0];
$args{ODE}     = [\&logi];
$args{t0}      = 0;
$args{tf}      = 40;
$args{verbose} = 0;
my $o = Math::ODE->new(%args);
$o->evolve;

my @t_vals = sort { $a <=> $b } keys %{$o->{values}};
my $N = pdl(map { $o->{values}{$_}[0] } @t_vals);
my $t = pdl(@t_vals);

# Solucion exacta
my $Nex = $K / (1 + (($K - $N0) / $N0) * exp(-$r * $t));
my $error = abs($N - $Nex);
my $error_maximo = max($error);
print "error_maximo = $error_maximo\n";

# Velocidad de crecimiento
my $v = $r * $N * (1 - $N / $K);
my $kmax = $v->maximum_ind;
my $tiempo_maximo      = $t->at($kmax);
my $poblacion_en_maximo = $N->at($kmax);

print "tiempo_maximo = $tiempo_maximo\n";
print "poblacion_en_maximo = $poblacion_en_maximo\n";
print "K/2 = ", $K / 2, "\n";

my $tmax_exacto = log(($K - $N0) / $N0) / $r;
print "tmax_exacto = $tmax_exacto\n";
```

JavaScript. Con `odex` se obtiene una función que representa la solución. Se evalúa sobre una malla mediante `map`; de este modo pueden calcularse el error máximo y la velocidad máxima sin escribir ciclos explícitos.

```
const odex=require("odex")

let r = 0.3
let K = 1000
let N0 = 50
```

```
let f = (t,N)=>[
    r*N[0]*(1-N[0]/K)
]

const sol = new odex.Solver(
    f,1
).integrate(0,[N0])

const t = Array.from(
    {length:401},
    (_,i)=>i/10
)

let N = t.map(
    ti=>sol(ti)[0]
)

let Nex = t.map(
    ti=>
        K/(1+(K-N0)/N0*Math.exp(-r*ti))
)

let error = N.map(
    (Ni,i)=>Math.abs(Ni-Nex[i])
)

let error_maximo = Math.max(...error)

let v = N.map(
    Ni=>r*Ni*(1-Ni/K)
)

let vmax = Math.max(...v)

let kmax = v.indexOf(vmax)

let tiempo_maximo = t[kmax]
let poblacion_en_maximo = N[kmax]

K/2

let tmax_exacto =
    Math.log((K-N0)/N0)/r
```

59. **Ejercicio:** Modelo depredador–presa de Lotka–Volterra

    - (a) resuelva numéricamente el sistema en $[0, 30]$;
    - (b) describa el comportamiento de ambas poblaciones y explique la relación entre sus oscilaciones.

Los puntos de equilibrio positivos satisfacen

$$a - by = 0, \qquad -c + dx = 0.$$

Por lo tanto,

$$x_{\text{eq}} = \frac{c}{d} = 20, \qquad y_{\text{eq}} = \frac{a}{b} = 10.$$

Las dos poblaciones oscilan alrededor de este estado de equilibrio.

Para la población de presas,

$$\dot{x} = x(a - by),$$

de modo que las presas aumentan cuando la población de depredadores está por debajo de

$$\frac{a}{b} = 10,$$

y disminuyen cuando está por encima.

Para los depredadores,

$$\dot{y} = y(-c + dx),$$

por lo que aumentan cuando la población de presas supera

$$\frac{c}{d} = 20$$

y disminuyen cuando está por debajo.

En consecuencia, los máximos de ambas poblaciones no ocurren al mismo tiempo: primero aumenta la población de presas y, posteriormente, aumenta la de depredadores. El aumento de los depredadores provoca entonces la disminución de las presas; al disminuir las presas, también disminuyen posteriormente los depredadores. Este desfase produce las oscilaciones características del modelo.

Scilab. La solución contiene una fila para cada población. Los mínimos y máximos permiten resumir el intervalo de valores recorrido durante las oscilaciones.

```
global a b c d

a=1
b=0.1
c=1.5
d=0.075

function dz=lv(t,z)
    global a b c d
```

```
    dz=[
        a*z(1)-b*z(1)*z(2)
        -c*z(2)+d*z(1)*z(2)
    ]
endfunction

t=linspace(0,30,3001)

z=ode([10;5],0,t,lv)

x=z(1,:)
y=z(2,:)

equilibrio=[c/d a/b]

resumen=[
    min(x) max(x)
    min(y) max(y)
]
```

Octave. Las columnas de la solución corresponden a las dos poblaciones.

```
a=1;
b=0.1;
c=1.5;
d=0.075;

lv=@(t,z) [
    a*z(1)-b*z(1)*z(2);
    -c*z(2)+d*z(1)*z(2)
];

t=linspace(0,30,3001)';

[t,z]=ode45(lv,t,[10;5]);

x=z(:,1);
y=z(:,2);

equilibrio=[c/d a/b]

resumen=[
    min(x) max(x);
    min(y) max(y)
]
```

Python. Las dos filas de la solución corresponden respectivamente a presas y depredadores.

```
import numpy as np
from scipy.integrate import solve_ivp

a=1
b=0.1
c=1.5
d=0.075

def lv(t,z):
    return [
        a*z[0]-b*z[0]*z[1],
        -c*z[1]+d*z[0]*z[1]
    ]

t=np.linspace(0,30,3001)

x,y=solve_ivp(
    lv,[0,30],[10,5],
    t_eval=t
).y

equilibrio=np.array([
    c/d,
    a/b
])

resumen=np.array([
    [x.min(),x.max()],
    [y.min(),y.max()]
])
```

Perl. La solución completa se convierte a PDL para analizar las dos poblaciones de manera vectorial.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use Math::ODE;

my $alpha = 1;
my $beta  = 0.1;
my $gamma = 1.5;
my $delta = 0.075;

# Lotka-Volterra
sub lv_x { my ($t, $z) = @_; return $alpha * $z->[0] - $beta
   * $z->[0] * $z->[1]; }
sub lv_y { my ($t, $z) = @_; return -$gamma * $z->[1] +
   $delta * $z->[0] * $z->[1]; }
```

```
my %args;
$args{step}    = 0.01;
$args{initial} = [10, 5];
$args{ODE}     = [\&lv_x, \&lv_y];
$args{t0}      = 0;
$args{tf}      = 30;
$args{verbose} = 0;
my $o = Math::ODE->new(%args);
$o->evolve;

my @t_vals = sort { $a <=> $b } keys %{$o->{values}};
my @x_vals = map { $o->{values}{$_}[0] } @t_vals;
my @y_vals = map { $o->{values}{$_}[1] } @t_vals;
my $x = pdl(@x_vals);
my $y = pdl(@y_vals);

my $equilibrio = pdl($gamma / $delta, $alpha / $beta);
print "equilibrio = $equilibrio\n";

my $resumen = pdl([$x->min, $x->max], [$y->min, $y->max]);
print "resumen = $resumen\n";
```

JavaScript. Con `odex` se obtiene una función que representa la solución. Se evalúa sobre una malla mediante `map`, sin escribir un ciclo explícito.

```
const odex=require("odex")

let a = 1
let b = 0.1
let c = 1.5
let d = 0.075

let lv = (t,z)=>[
    a*z[0]-b*z[0]*z[1],
    -c*z[1]+d*z[0]*z[1]
]

const sol = new odex.Solver(
    lv,2
).integrate(0,[10,5])

const t = Array.from(
    {length:3001},
    (_,i)=>i/100
)

let z = t.map(
    ti=>sol(ti)
)
```

```
let x = z.map(
    zi=>zi[0]
)

let y = z.map(
    zi=>zi[1]
)

const equilibrio = [
    c/d,
    a/b
]

const resumen = [
    [Math.min(...x),Math.max(...x)],
    [Math.min(...y),Math.max(...y)]
]
```

60. **Ejercicio:** Descarga de capacitor

Considere la descarga de un capacitor descrita por

$$RC\dot{V} + V = 0,$$

con

$$V(0) = V_0, \qquad R = 1000\,\Omega, \qquad C = 100\,\mu\text{F}, \qquad V_0 = 5\ \text{V}.$$

- (a) calcule la constante de tiempo

$$\tau = RC;$$

- (b) resuelva numéricamente la ecuación en $[0, 5\tau]$;
- (c) compare con la solución exacta

$$V(t) = V_0 e^{-t/\tau};$$

- (d) repita el cálculo utilizando los pasos máximos

$$h_1 = \frac{5\tau}{20}, \qquad h_2 = \frac{5\tau}{100}, \qquad h_3 = \frac{5\tau}{500},$$

que corresponden a las separaciones de mallas uniformes de 21, 101 y 501 puntos en $[0, 5\tau]$. Compare el error en $t = 5\tau$;

- (e) agregue una fuente constante $V_{\text{fuente}} = 3$ V, simule la carga desde $V(0) = 0$ y verifique que la tensión se aproxima a V_{fuente}.

Para los datos del problema,

$$\tau = RC = (1000)(100 \times 10^{-6}) = 0{,}1 \text{ s}.$$

Por lo tanto,

$$5\tau = 0{,}5 \text{ s}.$$

Para la descarga, la solución exacta es

$$V(t) = V_0 e^{-t/\tau}.$$

Al agregar una fuente constante, la ecuación pasa a ser

$$RC\dot{V} + V = V_{\text{fuente}},$$

o equivalentemente,

$$\dot{V} = \frac{V_{\text{fuente}} - V}{RC}.$$

Con $V(0) = 0$, su solución exacta es

$$V(t) = V_{\text{fuente}} \left(1 - e^{-t/\tau}\right).$$

Por ello,

$$\lim_{t \to \infty} V(t) = V_{\text{fuente}}.$$

En $t = 5\tau$,

$$V(5\tau) = 3(1 - e^{-5}) \approx 2{,}98 \text{ V},$$

que ya es muy próximo a 3 V.

Scilab. `ode` usa un integrador adaptativo. Los puntos de salida no determinan por sí solos el paso interno; el paso máximo se fija mediante `%ODEOPTIONS`.

```
global R C Vf

R=1000
C=100e-6
V0=5

tau=R*C
tf=5*tau

function dV=f(t,V)
```

```
    global R C
    dV=-V/(R*C)
endfunction

t=linspace(0,tf,101)

%ODEOPTIONS=[1,0,0,tf/100,0,2,5000,12,5,0,-1,-1]
V=ode(V0,0,t,f)

Vex=V0*exp(-t/tau)

error_maximo=max(abs(V-Vex))

h1=tf/20
%ODEOPTIONS=[1,0,0,h1,0,2,5000,12,5,0,-1,-1]
V1=ode(V0,0,tf,f)
e1=abs(V1-V0*exp(-5))

h2=tf/100
%ODEOPTIONS=[1,0,0,h2,0,2,5000,12,5,0,-1,-1]
V2=ode(V0,0,tf,f)
e2=abs(V2-V0*exp(-5))

h3=tf/500
%ODEOPTIONS=[1,0,0,h3,0,2,5000,12,5,0,-1,-1]
V3=ode(V0,0,tf,f)
e3=abs(V3-V0*exp(-5))

e1,e2,e3

Vf=3

function dV=g(t,V)
    global R C Vf
    dV=(Vf-V)/(R*C)
endfunction

%ODEOPTIONS=[1,0,0,h2,0,2,5000,12,5,0,-1,-1]
Vc=ode(0,0,t,g)

Vc_exacta=Vf*(1-exp(-t/tau))

Vc($)
Vc_exacta($)
Vf-Vc($)

clear %ODEOPTIONS
```

Octave. `ode45` usa paso adaptativo. `MaxStep` permite fijar los tres pasos máximos pedidos.

```
R=1000;
C=100e-6;
V0=5;

tau=R*C;
tf=5*tau;

f=@(t,V) -V/(R*C);

t=linspace(0,tf,101)';

op=odeset("MaxStep",tf/100);
[t,V]=ode45(f,t,V0,op);

Vex=V0*exp(-t/tau);

error_maximo=max(abs(V-Vex))

h1=tf/20;
op1=odeset("MaxStep",h1);
[t1,V1]=ode45(f,[0 tf],V0,op1);
e1=abs(V1(end)-V0*exp(-5));

h2=tf/100;
op2=odeset("MaxStep",h2);
[t2,V2]=ode45(f,[0 tf],V0,op2);
e2=abs(V2(end)-V0*exp(-5));

h3=tf/500;
op3=odeset("MaxStep",h3);
[t3,V3]=ode45(f,[0 tf],V0,op3);
e3=abs(V3(end)-V0*exp(-5));

e1,e2,e3

Vf=3;

g=@(t,V) (Vf-V)/(R*C);

[t,Vc]=ode45(g,t,0,op);

Vc_exacta=Vf*(1-exp(-t/tau));

Vc(end)
Vc_exacta(end)
Vf-Vc(end)
```

Python. `solve_ivp` usa paso adaptativo. `max_step` limita el paso interno máximo; `t_eval` sólo indica dónde se desea obtener la solución.

```
import numpy as np
from scipy.integrate import solve_ivp

R=1000
C=100e-6
V0=5

tau=R*C
tf=5*tau

def f(t,V):
    return -V/(R*C)

t=np.linspace(0,tf,101)

V=solve_ivp(
    f,[0,tf],[V0],
    t_eval=t,
    max_step=tf/100
).y[0]

Vex=V0*np.exp(-t/tau)

error_maximo=np.max(
    np.abs(V-Vex)
)

h1=tf/20
V1=solve_ivp(
    f,[0,tf],[V0],
    t_eval=[tf],
    max_step=h1
).y[0,-1]
e1=abs(V1-V0*np.exp(-5))

h2=tf/100
V2=solve_ivp(
    f,[0,tf],[V0],
    t_eval=[tf],
    max_step=h2
).y[0,-1]
e2=abs(V2-V0*np.exp(-5))

h3=tf/500
V3=solve_ivp(
    f,[0,tf],[V0],
    t_eval=[tf],
    max_step=h3
).y[0,-1]
e3=abs(V3-V0*np.exp(-5))
```

```
e1,e2,e3

Vf=3

def g(t,V):
    return (Vf-V)/(R*C)

Vc=solve_ivp(
    g,[0,tf],[0],
    t_eval=t,
    max_step=h2
).y[0]

Vc_exacta=Vf*(1-np.exp(-t/tau))

Vc[-1]
Vc_exacta[-1]
Vf-Vc[-1]
```

Perl. `Math::ODE` utiliza un paso fijo, por lo que los valores h_1, h_2 y h_3 son directamente los pasos de integración.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use Math::ODE;

my $R  = 1000;
my $C  = 100e-6;
my $V0 = 5;
my $tau = $R * $C;
my $tf  = 5 * $tau;

# Carga del capacitor: V' = -V/(RC)
sub rc {
    my ($t, $V) = @_;
    return -$V->[0] / ($R * $C);
}

sub final_value {
    my ($rhs, $y0, $h, $T) = @_;
    my %args;
    $args{step}    = $h;
    $args{initial} = [$y0];
    $args{ODE}     = [$rhs];
    $args{t0}      = 0;
    $args{tf}      = $T;
    $args{verbose} = 0;
    my $o = Math::ODE->new(%args);
```

```
    $o->evolve;
    my @t = sort { $a <=> $b } keys %{$o->{values}};
    return $o->{values}{$t[-1]}[0];
}

my $h1 = $tf / 20;
my $h2 = $tf / 100;
my $h3 = $tf / 500;

my $V1 = final_value(\&rc, $V0, $h1, $tf);
my $V2 = final_value(\&rc, $V0, $h2, $tf);
my $V3 = final_value(\&rc, $V0, $h3, $tf);

my $V_exacta_final = $V0 * exp(-5);
my $e1 = abs($V1 - $V_exacta_final);
my $e2 = abs($V2 - $V_exacta_final);
my $e3 = abs($V3 - $V_exacta_final);

print "errores (h1,h2,h3): $e1, $e2, $e3\n";
my $errores = pdl($e1, $e2, $e3);

# Carga hacia Vf = 3 V
my $Vf = 3;
sub rcf {
    my ($t, $V) = @_;
    return ($Vf - $V->[0]) / ($R * $C);
}

my $Vc_final  = final_value(\&rcf, 0, $h2, $tf);
my $Vc_exacta = $Vf * (1 - exp(-$tf / $tau));

print "Vc_final=$Vc_final, Vc_exacta=$Vc_exacta,
   diferencia=", ($Vf - $Vc_final), "\n";
```

JavaScript. Con `odex` se fijan los pasos máximos mediante `maxStepSize`.

```
const odex=require("odex")

let R = 1000
let C = 100e-6
let V0 = 5

let tau = R*C
let tf = 5*tau

let f = (t,V)=>[
    -V[0]/(R*C)
]

let resolver = (fun,Vinicial,h)=>{
    const s=new odex.Solver(fun,1)
```

```
    s.maxStepSize=h
    return s.integrate(0,[Vinicial])
}

let h1 = tf/20
let sol1 = resolver(f,V0,h1)
let e1 = Math.abs(
    sol1(tf)[0]-V0*Math.exp(-5)
)

let h2 = tf/100
let sol2 = resolver(f,V0,h2)
let e2 = Math.abs(
    sol2(tf)[0]-V0*Math.exp(-5)
)

let h3 = tf/500
let sol3 = resolver(f,V0,h3)
let e3 = Math.abs(
    sol3(tf)[0]-V0*Math.exp(-5)
)

console.log(e1,e2,e3)

let Vf = 3

let g = (t,V)=>[
    (Vf-V[0])/(R*C)
]

let solCarga = resolver(g,0,h2)

let Vc_final = solCarga(tf)[0]
let Vc_exacta = Vf*(1-Math.exp(-5))

Vc_final
Vc_exacta
Vf-Vc_final
```

61. **Ejercicio:** Caída libre con rozamiento

Considere la caída vertical con resistencia cuadrática

$$m\dot{v} = mg - cv^2,$$

tomando como positiva la dirección hacia abajo, con

$$m = 70 \text{ kg}, \qquad g = 9{,}8 \text{ m/s}^2, \qquad c = 0{,}25 \text{ kg/m}.$$

- (a) calcule la velocidad terminal

$$v_T = \sqrt{\frac{mg}{c}};$$

- (b) resuelva numéricamente la ecuación en $[0, 30]$ s con $v(0) = 0$;
- (c) compare la solución numérica con

$$v(t) = v_T \tanh\left(\frac{gt}{v_T}\right)$$

y calcule el error máximo;

- (d) agregue la posición mediante

$$\dot{x} = v, \qquad x(0) = 0,$$

y resuelva el sistema para $[x, v]$;

- (e) en $t = 5$, 10, 15 y 20 s compare la distancia recorrida con la correspondiente a una caída sin resistencia,

$$x_{\text{sin aire}}(t) = \frac{1}{2}gt^2,$$

y calcule cuánto disminuye la distancia recorrida debido a la resistencia del aire.

La velocidad terminal se obtiene imponiendo

$$\dot{v} = 0.$$

Entonces

$$mg - cv_T^2 = 0,$$

y por lo tanto

$$v_T = \sqrt{\frac{mg}{c}}.$$

La diferencia

$$\Delta x = x_{\text{sin aire}} - x_{\text{aire}}$$

permite cuantificar el efecto del rozamiento. También puede calcularse la reducción porcentual

$$100\frac{\Delta x}{x_{\text{sin aire}}}.$$

A medida que aumenta el tiempo, la velocidad con resistencia se aproxima a v_T, mientras que sin resistencia la velocidad continuaría creciendo linealmente. Por ello, la diferencia entre ambas distancias aumenta con el tiempo.

Scilab. La solución de velocidad se compara vectorialmente con la expresión exacta. Como la malla tiene paso 0,1 s, los tiempos pedidos corresponden directamente a posiciones conocidas del vector.

```
global m g c

m=70
g=9.8
c=0.25

vT=sqrt(m*g/c)

function dv=f(t,v)
    global m g c
    dv=g-c/m*v^2
endfunction

t=linspace(0,30,301)

v=ode(0,0,t,f)

vex=vT*tanh(g*t/vT)

error_velocidad=max(abs(v-vex))

function dY=caida(t,Y)
    global m g c

    dY=[
        Y(2)
        g-c/m*Y(2)^2
    ]
endfunction

Y=ode([0;0],0,t,caida)

x=Y(1,:)
v=Y(2,:)

ts=[5 10 15 20]

x_aire=x([51 101 151 201])

x_sin_aire=g*ts.^2/2

reduccion=x_sin_aire-x_aire

porcentaje=100*reduccion./x_sin_aire

comparacion=[
    ts
```

```
    x_aire
    x_sin_aire
    reduccion
    porcentaje
]
```

Octave. La solución se obtiene sobre una malla de 0,1 s, por lo que los cuatro tiempos pedidos corresponden directamente a elementos del vector de posición.

```
m=70;
g=9.8;
c=0.25;

vT=sqrt(m*g/c)

f=@(t,v) g-c/m*v.^2;

t=linspace(0,30,301)';

[t,v]=ode45(f,t,0);

vex=vT*tanh(g*t/vT);

error_velocidad=max(abs(v-vex))

caida=@(t,Y) [
    Y(2);
    g-c/m*Y(2)^2
];

[t,Y]=ode45(
    caida,t,[0;0]
);

x=Y(:,1);
v=Y(:,2);

ts=[5 10 15 20];

x_aire=x([51 101 151 201])';

x_sin_aire=g*ts.^2/2;

reduccion=x_sin_aire-x_aire;

porcentaje=100*reduccion./x_sin_aire;

comparacion=[
    ts;
    x_aire;
```

```
    x_sin_aire;
    reduccion;
    porcentaje
]
```

Python. NumPy permite comparar toda la trayectoria y seleccionar directamente las posiciones correspondientes a los cuatro tiempos pedidos.

```
import numpy as np
from scipy.integrate import solve_ivp

m=70
g=9.8
c=0.25

vT=np.sqrt(m*g/c)

def f(t,v):
    return g-c/m*v**2

t=np.linspace(0,30,301)

v=solve_ivp(
    f,[0,30],[0],
    t_eval=t
).y[0]

vex=vT*np.tanh(g*t/vT)

error_velocidad=np.max(
    np.abs(v-vex)
)

def caida(t,Y):
    return [
        Y[1],
        g-c/m*Y[1]**2
    ]

x,v=solve_ivp(
    caida,[0,30],[0,0],
    t_eval=t
).y

ts=np.array([5,10,15,20])

x_aire=x[
    [50,100,150,200]
]
```

```
x_sin_aire=g*ts**2/2

reduccion=x_sin_aire-x_aire

porcentaje=100*reduccion/x_sin_aire

comparacion=np.array([
    ts,
    x_aire,
    x_sin_aire,
    reduccion,
    porcentaje
])
```

Perl. Con `Math::ODE` se utiliza un paso fijo de 0,1 s. Por ello los tiempos pedidos corresponden a índices conocidos de la trayectoria.

```
#!/usr/bin/env perl
use strict;
use warnings;
use PDL;
use Math::ODE;

my $m = 70;
my $g = 9.8;
my $c = 0.25;
my $vT = sqrt($m * $g / $c);

# EDO para la velocidad: v' = g - (c/m)*v^2
sub f_vel {
    my ($t, $v) = @_;
    return $g - ($c / $m) * ($v->[0] ** 2);
}

my %args_vel;
$args_vel{step}    = 0.1;
$args_vel{initial} = [0];
$args_vel{ODE}     = [\&f_vel];
$args_vel{t0}      = 0;
$args_vel{tf}      = 30;
$args_vel{verbose} = 0;
my $o_vel = Math::ODE->new(%args_vel);
$o_vel->evolve;

my @t_vals = sort { $a <=> $b } keys %{$o_vel->{values}};
my $v_numeric = pdl(map { $o_vel->{values}{$_}[0] } @t_vals);
my $t_numeric = pdl(@t_vals);

my $vex = $vT * tanh($g * $t_numeric / $vT);
my $error_velocidad = max(abs($v_numeric - $vex));
print "error_velocidad = $error_velocidad\n";
```

```
# EDO para posicion y velocidad: Y' = [v, g - (c/m)*v^2]
sub caida_x { my ($t, $Y) = @_; return $Y->[1]; }
sub caida_v { my ($t, $Y) = @_; return $g - ($c / $m) *
   ($Y->[1] ** 2); }

my %args_yv;
$args_yv{step}    = 0.1;
$args_yv{initial} = [0, 0];
$args_yv{ODE}     = [\&caida_x, \&caida_v];
$args_yv{t0}      = 0;
$args_yv{tf}      = 30;
$args_yv{verbose} = 0;
my $o_yv = Math::ODE->new(%args_yv);
$o_yv->evolve;

my @t2 = sort { $a <=> $b } keys %{$o_yv->{values}};
my @x_numeric = map { $o_yv->{values}{$_}[0] } @t2;

my $t2_p = pdl(@t2);
my $x_p  = pdl(@x_numeric);

my @ts = (5, 10, 15, 20);
my @x_aire;
for my $ti (@ts) {
    my $idx = (abs($t2_p - $ti))->minimum_ind;
    push @x_aire, $x_p->at($idx);
}
my $x_aire = pdl(@x_aire);
my $ts_p = pdl(@ts);
my $x_sin_aire = $g * $ts_p ** 2 / 2;
my $reduccion  = $x_sin_aire - $x_aire;
my $porcentaje = 100 * $reduccion / $x_sin_aire;

print "ts = $ts_p\n";
print "x_aire = $x_aire\n";
print "x_sin_aire = $x_sin_aire\n";
print "reduccion = $reduccion\n";
print "porcentaje = $porcentaje\n";
```

JavaScript. Con `odex` la solución se evalúa sobre una malla mediante `map`. Así se puede calcular el error máximo sin escribir ciclos explícitos.

```
const odex=require("odex")

let m = 70
let g = 9.8
let c = 0.25

let vT = Math.sqrt(m*g/c)
```

```
let caida = (t,Y)=>[
    Y[1],
    g-c/m*Y[1]**2
]

const sol = new odex.Solver(
    caida,2
).integrate(0,[0,0])

const t = Array.from(
    {length:301},
    (_,i)=>i/10
)

let Y = t.map(
    ti=>sol(ti)
)

let x = Y.map(
    Yi=>Yi[0]
)

let v = Y.map(
    Yi=>Yi[1]
)

let vex = t.map(
    ti=>
        vT*Math.tanh(g*ti/vT)
)

let error = v.map(
    (vi,i)=>Math.abs(vi-vex[i])
)

let error_velocidad = Math.max(...error)

const ts = [5,10,15,20]

const x_aire = [
    x[50],
    x[100],
    x[150],
    x[200]
]

let x_sin_aire = ts.map(
    ti=>g*ti**2/2
)

let reduccion = x_sin_aire.map(
```

```
    (xi,i)=>xi-x_aire[i]
)

let porcentaje = reduccion.map(
    (di,i)=>100*di/x_sin_aire[i]
)

const comparacion = [
    ts,
    x_aire,
    x_sin_aire,
    reduccion,
    porcentaje
]
```

Capítulo 5

Programación

5.1. Programación

Programar es una habilidad cultural, no sólo una técnica profesional Saravia Alía 2005. Quien no puede escribir un programa depende de que otro haya previsto exactamente su problema, y en ciencia eso casi nunca ocurre: el cálculo que hace falta suele ser uno que nadie automatizó antes.

Esta unidad trata los conceptos de programación que no dependen del lenguaje. Todo lo que sigue vale igual en un entorno de cálculo científico, en un lenguaje de propósito general o en el intérprete de comandos del sistema operativo. Los ejemplos están escritos con la sintaxis del entorno que usamos en el laboratorio, pero funcionan acá como pseudocódigo: lo que hay que retener es la idea, no la palabra reservada. Una introducción general a estos conceptos, con ejercicios, está en Saravia Alía 2002 y en Romero Mora 2014.

5.1.1. De la orden suelta al programa

Un **comando** es una instrucción individual que el sistema ejecuta inmediatamente. Cuando se escribe un comando en una terminal o en la consola de un entorno de cálculo, el intérprete lo procesa y devuelve el resultado al instante.

Un **programa** o **script** es una secuencia de comandos almacenada en un archivo de texto que puede ejecutarse como una unidad. La diferencia no es de naturaleza sino de escala y propósito: un programa permite automatizar tareas repetitivas, estructurar soluciones complejas y reutilizar el trabajo.

Que la diferencia no sea de naturaleza tiene una consecuencia importante: el archivo de comandos es un objeto de estudio en sí mismo. Se puede leer, comentar, versionar, mandar por correo, publicar junto a un artículo. La sesión interactiva, en cambio, se pierde cuando se cierra la ventana. Ésa es la primera razón por la que el trabajo serio termina siempre en un archivo.

El flujo de trabajo típico en programación científica es:

1. Se desarrollan y prueban ideas interactivamente en la consola, ejecutando comandos uno por uno.

2. Las secuencias que funcionan se guardan en un archivo de script.
3. El script se refina, se documenta y se organiza en funciones.
4. El resultado es un programa reproducible que cualquier persona puede ejecutar y obtener los mismos resultados.

Ese ciclo —explorar, fijar, ordenar— es el que conviene incorporar desde el principio. El error frecuente del principiante no es escribir mal el código sino no escribirlo nunca en un archivo, y tener que reconstruir de memoria, dos semanas después, la secuencia de comandos que produjo un resultado.

5.1.2. Algoritmos y programas

Un **algoritmo** es un procedimiento finito y no ambiguo que, a partir de ciertos datos de entrada, produce un resultado en una cantidad finita de pasos. Es una noción anterior e independiente de las computadoras: el método para dividir dos números con lápiz y papel, o el de Euclides para el máximo común divisor, son algoritmos.

Un **programa** es la expresión de un algoritmo en un lenguaje que una máquina puede ejecutar. La distinción importa por dos razones. La primera es que un mismo algoritmo admite muchos programas, en muchos lenguajes, y elegir bien el algoritmo pesa mucho más que optimizar el programa: ningún ajuste de código salva a un método que hace un número de operaciones desproporcionado al problema. La segunda es que la corrección de un algoritmo se puede razonar y demostrar, mientras que la de un programa además hay que probarla, porque puede haber errores en la traducción.

De un algoritmo interesan, más allá de que funcione:

La corrección: que termine, y que al terminar el resultado sea el que se buscaba. Lo primero no es automático: un bucle mal planteado puede no terminar nunca.

El costo: cuántas operaciones y cuánta memoria requiere en función del tamaño del problema. Un método cuyo costo crece con el cuadrado del tamaño de los datos es utilizable con mil datos e inutilizable con un millón.

La estabilidad numérica: en cálculo científico, si los errores de redondeo inevitables se amplifican a lo largo del proceso o se mantienen acotados.

El estudio sistemático de los algoritmos, su análisis y su costo es un campo con desarrollo propio Donald E. Knuth 2022. Para el enfoque que nos interesa acá —resolver problemas escribiendo programas, y aprender los conceptos por el camino— son útiles los textos introductorios que organizan el aprendizaje alrededor de problemas Downey 2015.

5.1.3. Interpretación de comandos y compilación

La brecha entre el texto que escribe quien programa y las instrucciones que ejecuta el procesador puede salvarse de dos maneras.

Un **compilador** traduce el programa completo antes de que se ejecute, y produce un archivo ejecutable. El proceso incluye varias etapas: análisis léxico, que reconoce las palabras del lenguaje; análisis sintáctico, que verifica la estructura gramatical; análisis semántico, que verifica el significado; optimización; y generación del código de máquina.

Un **intérprete** lee, traduce y ejecuta una instrucción por vez, en tiempo de ejecución, sin producir un ejecutable previo. Requiere estar instalado en el sistema donde el programa corre.

Las ventajas de cada uno se siguen de esa diferencia. La compilación da mayor velocidad de ejecución, porque el trabajo de traducción ya está hecho, y detecta antes cierta clase de errores, porque revisa todo el programa antes de correr nada; el precio es un ciclo de trabajo más lento y la necesidad de recompilar para cada plataforma Kernighan y Ritchie 1978. La interpretación da un ciclo interactivo —escribir, probar, corregir, sin esperas— y portabilidad, porque el mismo texto corre donde haya intérprete; el precio es velocidad y que los errores aparecen recién al llegar la ejecución a la línea que los contiene.

Conviene notar que ésta no es una propiedad del lenguaje sino de la implementación: existen intérpretes de lenguajes que se suelen compilar, y compiladores de lenguajes que se suelen interpretar. Además, casi todo lo que se usa hoy es híbrido. Es común compilar a un código intermedio, o *bytecode*, que ejecuta una máquina virtual, y que un compilador *en tiempo de ejecución* traduzca a código nativo las partes que más se repiten.

El caso más cotidiano de intérprete es el propio intérprete de comandos del sistema operativo, que es a la vez la interfaz de uso diario y un lenguaje de programación completo Ramey y Fox 2025; Cooper 2014. Ahí se ve con claridad que la frontera entre «usar el sistema» y «programarlo» es sólo una cuestión de guardar en un archivo lo que se venía escribiendo.

5.1.4. Estructuras de control

Las estructuras de control determinan el orden en que se ejecutan las instrucciones de un programa. Sin ellas, las instrucciones se ejecutarían secuencialmente de la primera a la última, lo que limitaría enormemente la expresividad de los programas.

Secuencia

La **secuencia** es la estructura más simple: las instrucciones se ejecutan una tras otra en el orden en que aparecen. Es la estructura por defecto cuando no se indica ninguna otra.

Selección

Las estructuras de **selección** o condicionales permiten ejecutar distintos bloques de código según se cumpla o no una condición.

La forma básica es la selección binaria: si la condición es verdadera se ejecuta un bloque; si es falsa, otro. La forma extendida permite encadenar múltiples condiciones excluyentes: se evalúan en orden y se ejecuta el bloque correspondiente a la primera condición verdadera. Cuando ninguna condición es verdadera puede ejecutarse un bloque por defecto.

```
if condicion then
    bloque_1
elseif otra_condicion then
    bloque_2
else
    bloque_por_defecto
end
```

El selector múltiple es una forma conveniente de selección cuando la condición es la igualdad de una variable con varios valores posibles.

```
select variable
case valor_1 then
    bloque_1
case valor_2 then
    bloque_2
else
    bloque_por_defecto
end
```

Dos advertencias que valen en cualquier lenguaje. La primera: el orden de las condiciones encadenadas importa, porque se evalúan de arriba hacia abajo y la primera verdadera gana; una condición general puesta antes que una particular hace que la particular nunca se alcance. La segunda: comparar números reales por igualdad es casi siempre un error, porque el resultado de un cálculo en punto flotante rara vez es exactamente el valor esperado. Lo correcto es preguntar si la diferencia es menor que una tolerancia.

Iteración

Las estructuras de **iteración** o bucles permiten repetir un bloque de instrucciones.

El **bucle con contador** repite el bloque un número determinado de veces, recorriendo una secuencia de valores. Es adecuado cuando se conoce de antemano cuántas iteraciones se necesitan.

```
for i = 1:n
    bloque
end
```

El **bucle con condición al inicio** evalúa la condición antes de cada iteración y repite el bloque mientras la condición sea verdadera. Si la condición es falsa desde el principio, el bloque no se ejecuta ninguna vez.

```
while condicion
    bloque
end
```

El **bucle con condición al final** ejecuta el bloque al menos una vez y luego verifica la condición. Es adecuado cuando la primera iteración debe realizarse sin condición previa.

Las instrucciones de **corte** y **continuación** modifican el flujo dentro de un bucle: la primera lo termina inmediatamente; la segunda salta al inicio de la siguiente iteración omitiendo el resto del bloque.

Todo bucle con condición necesita que algo dentro del cuerpo pueda hacerla falsa. Si no, no termina. En los métodos iterativos del cálculo numérico —los que repiten hasta que el error sea menor que una tolerancia— esa garantía no existe cuando el método no converge, y por eso se les agrega siempre un número máximo de iteraciones: la condición de salida real es «convergí *o* me cansé», y el programa debe distinguir los dos casos al terminar.

Recursión

La **recursión** es una forma de repetición en la que una función se llama a sí misma. Todo problema recursivo tiene una estructura de dos partes: el *caso base*, que termina la recursión, y el *paso recursivo*, que reduce el problema a uno más pequeño del mismo tipo.

La recursión es elegante y natural para ciertos problemas —árboles, listas, algoritmos de dividir y conquistar— pero puede ser ineficiente si se recalculan los mismos subproblemas repetidamente. La técnica de *memoización* almacena los resultados ya calculados para evitar la repetición.

Toda recursión puede reescribirse como iteración y viceversa; la elección es de claridad, no de posibilidad. Conviene saber que cada llamada pendiente ocupa memoria en la pila, de modo que una recursión muy profunda puede agotarla. En los lenguajes que provienen de la tradición funcional la recursión es la forma natural de repetir, y el mecanismo se remonta al origen mismo de esa tradición McCarthy 1960.

5.1.5. Funciones predefinidas

Una **función** en programación científica es un bloque de código que recibe cero o más argumentos de entrada, realiza un cálculo y devuelve uno o más resultados. Las funciones permiten:

- Reutilizar código: se escribe una vez y se llama cuantas veces sea necesario.
- Estructurar el programa en partes comprensibles.
- Abstraer los detalles de implementación.
- Facilitar la verificación y el diagnóstico de errores.

Los entornos de cálculo científico incluyen una gran cantidad de funciones predefinidas agrupadas por categorías. Conocer qué categorías existen importa más que memorizar nombres: lo que hay que adquirir es el reflejo de preguntarse si lo que se necesita ya está hecho, antes de escribirlo.

Matemáticas elementales: trigonométricas, exponenciales, logarítmicas, redondeo, valor absoluto, resto de la división.

Álgebra lineal: resolución de sistemas, determinante, inversa, autovalores y autovectores, descomposiciones, normas.

Estadística: media, mediana, desviación estándar, varianza, correlación, generación de números aleatorios con distintas distribuciones.

Polinomios: evaluación, raíces, ajuste, derivación e integración simbólica de polinomios.

Análisis numérico: integración, derivación, interpolación, búsqueda de raíces, optimización, ecuaciones diferenciales.

Manipulación de matrices: construcción, redimensionamiento, concatenación, extracción de submatrices, ordenamiento, búsqueda.

Cadenas de caracteres: conversión entre número y texto, concatenación, búsqueda, separación por delimitadores.

Entrada y salida: apertura y cierre de archivos, lectura y escritura con formato, diálogos con quien usa el programa.

Gráficos: representación en dos y tres dimensiones, curvas de nivel, control de propiedades, exportación.

Sistema: ejecución de programas externos, manejo de directorios, consulta del entorno.

Ese repertorio es lo que distingue un entorno especializado de un lenguaje de propósito general: en el primero, casi todo esto es parte del lenguaje y está disponible sin importar nada; en el segundo, lo mismo existe pero en bibliotecas que hay que conocer, instalar y registrar como dependencia Harris, Millman, Walt, Gommers, Virtanen, Cournapeau y Oliphant 2020; Virtanen, Gommers, Oliphant, Haberland, Reddy y Cournapeau 2020.

La habilidad que conviene desarrollar es doble: saber que la función existe, y saber verificar que hace lo que uno cree. Lo segundo importa tanto como lo primero, porque las convenciones difieren entre entornos —en qué dimensión opera una función sobre una matriz, si un rango incluye o no el extremo, qué devuelve ante un caso degenerado— y esas diferencias no producen un error sino un resultado distinto.

5.1.6. Funciones definidas por el usuario

Una **función** es un bloque de código con nombre que recibe cero o más argumentos, realiza un cálculo y devuelve cero o más resultados. Las funciones son el mecanismo fundamental de abstracción y reutilización en programación: permiten dar nombre a una idea y usarla después sin volver a pensar cómo está hecha Abelson, G. J. Sussman y J. Sussman 1996.

Los conceptos asociados son:

Signatura o interfaz: el nombre de la función, el número y tipo de sus parámetros y el tipo de su valor de retorno.

Implementación o cuerpo: el código que realiza el cálculo.

Llamada: el punto del programa donde se invoca la función con argumentos concretos.

Parámetros formales: los nombres que la función usa internamente para referirse a sus argumentos.

Argumentos: los valores concretos que se pasan en una llamada.

```
function [salida_1, salida_2] = nombre(entrada_1, entrada_2)
    // cuerpo
endfunction
```

La distinción entre interfaz e implementación es la que hace posible todo lo demás. Quien usa la función necesita conocer sólo la primera; quien la escribe puede cambiar la segunda sin romper nada, mientras la interfaz se mantenga. Eso es lo que permite mejorar un método numérico sin tocar los programas que lo llaman.

Una función bien diseñada:

- Hace una sola cosa y la hace bien.
- Tiene un nombre que describe claramente qué hace.
- Tiene parámetros con nombres descriptivos.
- Es corta: si no cabe en una pantalla, probablemente hace demasiado.
- No tiene efectos secundarios inesperados.
- Está documentada.

El paso de argumentos puede ser **por valor** —se pasa una copia del dato, y modificarlo dentro de la función no afecta a la variable original— o **por referencia**, donde se pasa la dirección del dato y las modificaciones sí afectan al original. La diferencia deja de ser académica en cuanto los datos son grandes: copiar una matriz de un millón de elementos en cada llamada tiene un costo, y por eso varios lenguajes copian sólo si la función efectivamente modifica el dato.

Conviene además distinguir el resultado de una función de sus **efectos secundarios**: escribir un archivo, modificar una variable global, dibujar en pantalla. Una función que sólo calcula y devuelve se puede probar sola, con una lista de entradas y salidas esperadas; una que además hace cosas por fuera, no. Ésa es la razón práctica por la que conviene separar el cálculo de la entrada y salida.

5.1.7. Variables: alcance y tiempo de vida

Alcance

El **alcance** de una variable define en qué partes del programa es accesible su nombre.

Las variables **locales** se definen dentro de una función o bloque y sólo son accesibles dentro de ese contexto. Dejan de existir cuando la función termina. La localidad es la norma recomendada: reduce el acoplamiento entre partes del programa y permite razonar sobre una función mirando solamente su código.

Las variables **globales** son accesibles desde cualquier parte del programa. Permiten compartir información entre funciones sin pasar parámetros, pero dificultan el razonamiento, porque cualquier función puede modificarlas en cualquier momento y para entender un error hay que revisar el programa entero. Su uso debe minimizarse.

Los **parámetros** tienen alcance local a la función: son variables que reciben su valor en la llamada y sólo existen durante su ejecución.

Tiempo de vida

El **tiempo de vida** de una variable es el intervalo durante el cual existe en memoria. No hay que confundirlo con el alcance: el alcance es una propiedad del texto del programa, el tiempo de vida es una propiedad de la ejecución.

Las variables **automáticas o de pila** se crean al entrar al bloque donde se definen y se destruyen al salir. La mayoría de las variables locales son automáticas.

Las variables **estáticas** persisten durante toda la ejecución del programa, aunque su alcance puede ser local. Una variable local estática mantiene su valor entre llamadas sucesivas a la función, lo cual sirve para llevar un contador o guardar un resultado intermedio caro de recalcular.

Las variables **dinámicas** se crean y destruyen explícitamente durante la ejecución mediante operaciones de reserva y liberación de memoria. Este modelo es más flexible pero requiere gestión: en algunos lenguajes es manual, con una operación de reserva y otra de liberación que el programador debe emparejar, y en otros es automática mediante un *recolector de basura* que libera lo que ya no es alcanzable. La gestión manual es más eficiente y es fuente de dos errores clásicos: liberar dos veces, y no liberar nunca.

Tipado estático y dinámico

Hay dos sentidos distintos de «estático» y «dinámico» que conviene no mezclar. Uno es el del tiempo de vida, recién visto. El otro es el del **tipado**.

El **tipado estático** verifica los tipos antes de ejecutar: el tipo de cada variable se conoce en tiempo de compilación y las operaciones se validan de antemano. Detecta una clase entera de errores sin correr el programa, y permite generar código más eficiente, porque el traductor sabe exactamente qué tamaño tiene cada dato.

El **tipado dinámico** verifica los tipos en tiempo de ejecución: una misma variable puede contener valores de distintos tipos en distintos momentos, y las estructuras pueden crecer o reducirse según haga falta. Es mucho más cómodo para el trabajo exploratorio, y es lo que hace que en un entorno de cálculo se pueda escribir una línea y ver el resultado sin declarar nada. El precio es que los errores de tipo aparecen recién al ejecutar la línea que los contiene, quizá después de una hora de cálculo.

El **tipado fuerte** no permite conversiones implícitas entre tipos incompatibles; el **débil**

las realiza automáticamente, lo que resulta cómodo y ocasionalmente produce resultados inesperados. Las dos clasificaciones son independientes: hay lenguajes dinámicos y fuertes, y estáticos y débiles.

¿El tipo define la operación, o la operación define la lectura del dato?

Hay una diferencia de diseño que ilumina todo lo anterior. En la mayoría de los lenguajes, el tipo del dato determina qué hace un operador: si los dos operandos son números, el signo más suma; si son cadenas, quizá concatene.

Existe la decisión contraria: que sea el *operador* el que determine cómo se interpreta el dato. En ese diseño hay un operador para sumar y otro distinto para concatenar, y cada uno lee sus operandos como corresponde, convirtiendo si hace falta. Escribir la suma sobre dos cadenas que contienen dígitos da un número; escribir la concatenación sobre dos números da una cadena Wall, Christiansen y Orwant 2000b.

Ninguna de las dos opciones es «la correcta». La primera detecta antes los errores de mezclar cosas distintas; la segunda elimina de raíz la ambigüedad de qué hace un operador sobrecargado, al precio de tener más operadores. Lo importante es advertir que aquello que uno toma por natural en el lenguaje que aprendió primero es una decisión de diseño, y que hubo otras.

5.1.8. Paradigmas de programación

Un **paradigma de programación** es un estilo o enfoque fundamental para estructurar y expresar los programas. No es una característica de un lenguaje sino una forma de pensar sobre los problemas y sus soluciones. La mayoría de los lenguajes actuales admiten varios paradigmas, y un mismo programa puede combinarlos.

Imperativa y estructurada

La **programación imperativa** describe *cómo* se realiza un cálculo, mediante una secuencia de instrucciones que modifican el estado del programa. Es el paradigma más cercano al funcionamiento de la máquina.

La **programación estructurada** propone que todo programa puede escribirse usando únicamente tres estructuras —secuencia, selección e iteración— eliminando el salto incondicional. El argumento de fondo no es estético: con saltos arbitrarios, el estado del programa en un punto dado depende de por cuál de muchos caminos se llegó ahí, y razonar sobre el código se vuelve impracticable. Con las tres estructuras, cada bloque tiene una entrada y una salida, y se puede razonar por partes.

Orientada a objetos

La **programación orientada a objetos** organiza el programa en torno a *objetos*: entidades que combinan datos (atributos) y comportamiento (métodos). Los conceptos centrales son:

Clase: plantilla que define la estructura y el comportamiento de un tipo de objeto.

Objeto o instancia: un ejemplar concreto de una clase.

Encapsulamiento: los datos internos son privados y sólo se accede a ellos por los métodos de la clase.

Herencia: una clase puede heredar atributos y métodos de otra, extendiendo o modificando su comportamiento.

Polimorfismo: distintos objetos pueden responder al mismo mensaje de formas diferentes según su tipo.

La idea de fondo es la misma que la de la función, llevada a los datos: separar la interfaz de la implementación, de modo que se pueda cambiar cómo está representado algo sin tocar el código que lo usa.

Funcional

La **programación funcional** trata los programas como evaluaciones de funciones matemáticas y evita el estado mutable y los efectos secundarios. Sus conceptos centrales son:

Funciones de primera clase: las funciones son valores que pueden guardarse en variables, pasarse como argumentos y devolverse como resultados.

Funciones de orden superior: funciones que reciben o devuelven otras funciones. Las tres canónicas aplican una función a cada elemento de una colección, seleccionan los que cumplen una condición, y combinan todos en un solo valor.

Inmutabilidad: los datos no se modifican; en lugar de cambiar un valor, se crea uno nuevo.

Transparencia referencial: una función siempre devuelve el mismo resultado para los mismos argumentos, sin efectos secundarios, de modo que la llamada se puede reemplazar por su resultado sin cambiar el significado del programa.

Esa última propiedad es la que hace que el código funcional sea fácil de probar y de paralelizar: si nada comparte estado mutable, no hay carreras. El paradigma tiene raíces en la formalización de las funciones recursivas sobre expresiones simbólicas McCarthy 1960. En cálculo científico aparece más de lo que parece: aplicar una operación a todos los elementos de un vector, sin escribir el bucle, es exactamente una función de orden superior.

Lógica

La **programación lógica** expresa el programa como un conjunto de hechos y de reglas, y el intérprete encuentra las soluciones por deducción. Quien programa describe *qué* debe ser verdad, no *cómo* encontrarlo: el mecanismo de búsqueda lo aporta el sistema Kowalski 1979. Es el paradigma más alejado de la máquina y el más cercano a la especificación del problema.

Orientada a eventos

En la **programación orientada a eventos** el flujo del programa está determinado por sucesos: acciones de quien lo usa, mensajes de la red, temporizadores, lecturas de un sensor. El programa registra *manejadores* para cada tipo de evento y espera que ocurran. Es el paradigma dominante en interfaces gráficas y en sistemas de adquisición de datos, y lo veremos en detalle en las unidades correspondientes.

Literaria

La **programación literaria** propone escribir el programa como un documento dirigido a personas, intercalando el código con su explicación en prosa, y que sea una herramienta la que extraiga de ahí el programa ejecutable D. E Knuth 1984. La idea central es invertir la prioridad: el destinatario principal del texto es el lector humano, y la máquina es el destinatario secundario. Los cuadernos de cálculo interactivos, que veremos más adelante, son una implementación moderna y parcial de esta idea.

Declarativa

La **programación declarativa** describe qué se quiere obtener sin especificar cómo obtenerlo. Los lenguajes de consulta a bases de datos y los de descripción de documentos son los ejemplos más extendidos: se describe el resultado deseado y el sistema decide el procedimiento. La programación lógica y la funcional son, en este sentido, casos particulares de lo declarativo.

Guiada por pruebas

El **desarrollo guiado por pruebas** no es exactamente un paradigma de estructuración del código sino de orden de trabajo: primero se escribe la prueba que especifica el comportamiento esperado, después el código mínimo que la hace pasar, y por último se reordena el código sin cambiar su comportamiento. La disciplina obliga a decidir la interfaz antes que la implementación, que es el orden correcto y el que no sale espontáneamente.

Programación estructurada

La **programación estructurada**, formalizada por Dijkstra en los años 60, propone que todo programa puede escribirse usando únicamente tres estructuras: secuencia, selección e iteración, eliminando el uso del `goto`. Esta disciplina hace los programas más comprensibles y mantenibles.

Programación orientada a objetos

Haskell y Erlang son lenguajes puramente funcionales. Python, Scala y JavaScript soportan programación funcional junto con otros paradigmas.

5.1.9. Buenas prácticas como integración de los paradigmas

Los paradigmas suelen presentarse como opciones excluyentes, y en la práctica no lo son. Lo que se conoce como «buenas prácticas» es en buena medida el conjunto de recomendaciones que quedan al destilar cada paradigma y quedarse con lo que resultó valioso en general:

- De la programación **estructurada**: que cada bloque tenga una entrada y una salida, y que el flujo se pueda seguir leyendo de arriba hacia abajo.
- De la **funcional**: preferir funciones sin efectos secundarios, que dependan sólo de sus argumentos; evitar el estado compartido; separar el cálculo de la entrada y salida.
- De la **orientada a objetos**: agrupar los datos con las operaciones que los manipulan, y separar la interfaz de la implementación.
- De la **declarativa**: expresar la intención en vez del mecanismo cuando se puede, porque el mecanismo puede mejorar después sin tocar la intención.
- De la **guiada por pruebas**: especificar el comportamiento esperado antes de implementarlo.
- De la **literaria**: escribir para que otro lea.

Hay además una tradición, nacida junto a los sistemas operativos de tipo Unix, que aporta un principio propio: construir programas pequeños que hagan una cosa bien y que se puedan combinar entre sí, comunicándose por texto Ritchie y Thompson 1974; Raymond 2003. Es la misma idea de modularidad, aplicada un nivel más arriba: no ya funciones dentro de un programa sino programas dentro de un sistema.

5.1.10. Ingeniería del software científico

El software científico tiene características particulares: se escribe para resolver un problema específico de investigación, a menudo por personas sin formación formal en ingeniería de software, en tiempo limitado, y con la validación como objetivo central. Sin embargo puede tener una vida mucho más larga de lo esperado: se reutiliza, se modifica, se publica y otros lo usan para construir sobre él. Por eso las buenas prácticas son especialmente importantes.

Hay además una diferencia de fondo con el software comercial. En un programa de gestión, si el resultado está mal, alguien lo nota: el saldo no cuadra. En un programa científico, un resultado incorrecto suele ser un número perfectamente plausible, y la única defensa es que el programa haya sido validado contra algo que se conoce de antemano.

Código legible

El código se lee muchas más veces de las que se escribe. Escribirlo legible no es un lujo sino una inversión: facilita la detección de errores, la colaboración y el mantenimiento. Y el primer lector beneficiado es uno mismo dentro de seis meses.

Las prácticas que mejoran la legibilidad incluyen usar nombres descriptivos para variables y funciones —no `x1`, `temp2` o `aux`, sino `velocidad_inicial`, `temperatura_ambiente`, `resultado_integral`—; mantener funciones cortas con un único propósito; usar espaciado y sangría consistentes; y evitar los números sin nombre: en lugar de multiplicar por `9.8` conviene definir una constante `g` con un comentario que indique qué es y en qué unidades.

Comentarios y documentación

Un buen comentario explica *por qué* se hace algo, no *qué* se hace: eso lo dice el código mismo si está bien escrito. Los comentarios que repiten el código son ruido, y además hay que actualizarlos cuando el código cambia, lo que no siempre ocurre; un comentario desactualizado es peor que ninguno, porque miente.

En cálculo científico hay un tipo de comentario que sí es indispensable y que el código no puede expresar: la referencia al origen. De dónde salió la fórmula, qué aproximación se está usando, en qué rango vale, qué unidades tienen las variables. Eso no está en ninguna otra parte.

La **documentación** de una función describe su propósito, sus parámetros, su valor de retorno y, cuando corresponde, las hipótesis que asume y las limitaciones que tiene. Debería estar en el código mismo, no en un documento separado que inevitablemente queda desactualizado.

Pruebas y validación

Una **prueba** es un fragmento de código que verifica que otra parte del código hace lo que debería. Las pruebas son la única forma confiable de saber que el software funciona, y la única forma de cambiarlo después sin miedo.

Las **pruebas unitarias** verifican una función aislada; las de **integración**, que los componentes funcionen bien juntos; las de **regresión**, que un cambio no rompa lo que antes andaba.

En software científico conviene distinguir dos cosas que suelen confundirse. La **verificación** pregunta si el programa resuelve correctamente las ecuaciones que dice resolver; se responde comparando con soluciones analíticas conocidas, con casos límite y con resultados publicados. La **validación** pregunta si esas ecuaciones describen el fenómeno real; se responde comparando con datos experimentales. Un programa puede estar perfectamente verificado y ser inválido, porque el modelo no corresponde al sistema físico.

Casos de prueba que casi siempre están disponibles: el valor en un punto donde la solución se conoce a mano, el comportamiento cuando un parámetro tiende a cero o a infinito, una magnitud que debe conservarse a lo largo de la simulación, y la comprobación de que al refinar el paso el resultado converge al orden esperado.

Modularidad, bibliotecas y reutilización

Un programa modular está dividido en componentes independientes con interfaces bien definidas. Cada módulo tiene una responsabilidad clara y limitada, puede desarrollarse y

probarse por separado, y puede reutilizarse en otros programas.

La **reutilización** evita duplicar código: cada pieza de conocimiento debería tener una única representación en el sistema. Cuando el mismo código aparece en dos lugares, un cambio debe hacerse en dos lugares, lo que lleva inevitablemente a que queden distintos.

Un módulo reutilizable que se distribuye por separado es una **biblioteca**. La forma en que se incorpora al programa admite dos variantes que conviene distinguir:

Enlace estático: el código de la biblioteca se copia dentro del ejecutable en el momento de construirlo. El resultado es un archivo grande pero autosuficiente, que corre en cualquier sistema compatible sin requisitos adicionales, y que queda congelado: si la biblioteca se corrige, hay que volver a construirlo.

Enlace dinámico: el ejecutable guarda sólo la referencia, y el código de la biblioteca se carga en memoria al arrancar el programa. El ejecutable es chico, una sola copia de la biblioteca sirve para todos los programas que la usan, y una corrección se propaga sola. El precio es la dependencia: si la biblioteca falta o cambia de manera incompatible, el programa deja de funcionar.

Ese compromiso reaparece en la reproducibilidad. El enlace dinámico hace que el mismo programa, con los mismos datos, pueda dar resultados distintos en dos máquinas con versiones distintas de una biblioteca de cálculo. Por eso registrar las versiones de las dependencias es parte de publicar un resultado.

Reproducibilidad

El software científico reproducible puede ser ejecutado por cualquier persona, en cualquier sistema compatible, y producir los mismos resultados. Para lograrlo es necesario:

- Publicar el código fuente.
- Documentar las dependencias y sus versiones.
- Fijar las semillas de los generadores de números aleatorios cuando se usan.
- Separar los datos de entrada del código de análisis.
- Usar formatos de datos estándar y abiertos.
- Proporcionar instrucciones claras para reproducir el análisis.

Vale detenerse en la semilla: los generadores de números «aleatorios» de una computadora son deterministas, y producen siempre la misma secuencia a partir del mismo valor inicial. Fijarlo no es hacer trampa, es lo que permite que un resultado estadístico sea verificable por otro. Lo que sí sería trampa es probar muchas semillas y publicar la que dio mejor.

Conviene notar que reproducibilidad no es lo mismo que corrección. Un programa reproducible puede estar sistemáticamente equivocado. Lo que la reproducibilidad garantiza es que el error sea encontrable.

5.1.11. Uso de inteligencia artificial para programar

Las herramientas de generación automática de texto basadas en grandes modelos de lenguaje cambiaron la forma en que se escribe código. Pueden generar código a partir de una descripción en lenguaje natural, explicar código existente, sugerir correcciones, detectar errores comunes y traducir entre lenguajes.

Conviene tener alguna idea de qué son. La aspiración de construir sistemas que resuelvan problemas que requieren inteligencia es vieja y tuvo etapas muy distintas Feigenbaum 1977; Russell y Norvig 2021. La etapa actual se apoya en el aprendizaje automático: en lugar de programar las reglas, se ajustan los parámetros de un modelo a partir de ejemplos Mitchell 1997; Bishop 2006. Las redes profundas llevaron ese enfoque a escalas que permitieron resolver tareas antes inabordables LeCun, Bengio e Hinton 2015; Goodfellow, Bengio y Courville 2016, y una arquitectura particular, basada en el mecanismo de atención Vaswani et al. 2017, hizo posible entrenar modelos de lenguaje sobre grandes cantidades de texto y usarlos después para tareas que no se les enseñaron explícitamente Radford et al. 2018.

Eso permite entender qué hacen y qué no. El modelo estima qué continuación es probable dado un texto. Que el código producido sea correcto es una consecuencia frecuente de que el código correcto sea lo más probable en los textos con que se entrenó, no una propiedad garantizada del procedimiento.

Usos productivos

La herramienta es especialmente útil para generar código repetitivo, recordar la sintaxis exacta de algo poco usado, obtener un punto de partida que después se refina, explicar código ajeno o código propio escrito hace tiempo, sugerir nombres, detectar errores tipográficos y convertir código de un lenguaje a otro.

Limitaciones fundamentales

No reemplaza entender el algoritmo, por razones que no son provisorias:

No verifica la corrección: genera código plausible, no correcto. Puede producir algo que parece razonable pero calcula otra cosa, especialmente en matemática y física.

No conoce el contexto: no conoce los datos del problema, las hipótesis del modelo, los rangos válidos de las variables ni los criterios de validación. Sin eso, el código puede ser sintácticamente correcto e inapropiado.

Los errores no son obvios: un error humano suele manifestarse como un resultado claramente disparatado. Un error sutil en código generado produce resultados plausibles pero incorrectos, que pasan desapercibidos si no se valida.

No puede diseñar el algoritmo: decidir qué método numérico usar, cómo formular las ecuaciones, qué aproximaciones valen y cómo interpretar los resultados requiere comprensión del dominio.

Inventa: puede producir funciones que no existen, documentación incorrecta o explicaciones plausibles y falsas. La verificación es indispensable.

El uso correcto es como el de una calculadora: acelera el trabajo mecánico pero no reemplaza entender qué se está calculando ni para qué. La práctica recomendada es pedir el código, leerlo y entenderlo completamente antes de usarlo, verificar con casos de prueba conocidos, y mantener siempre la capacidad de escribirlo uno mismo.

Hay un motivo adicional, propio de esta materia. Todo lo anterior —las estructuras de control, el alcance de las variables, la distinción entre verificar y validar, el control de versiones— es lo que permite juzgar si el código que uno recibe sirve. Sin eso, quien programa con asistencia automática no está programando: está apostando.

Toda la parte concreta de esta unidad —cómo se escribe efectivamente cada una de estas cosas— se desarrolla en las partes dedicadas a cada lenguaje. La documentación de referencia de cada uno es el lugar donde verificar la sintaxis exacta Python Software Foundation 2026a.

5.2. Programación, temas avanzados

La primera parte teórica cubrió lo que todo programa necesita: secuencia, control, funciones, variables, paradigmas y las prácticas que los integran. Lo que sigue son mecanismos que aparecen apenas los programas crecen o se salen del cálculo numérico puro, y que tienen una particularidad común: cada lenguaje los tiene de forma nativa, los ofrece a medias o hay que emularlos. Conviene conocerlos como ideas para reconocerlos después bajo cualquier disfraz.

5.2.1. Funciones que recuerdan

Estado que sobrevive a la llamada

Una función normal olvida todo al terminar: sus variables locales dejan de existir y la siguiente llamada empieza de cero. Hay situaciones donde eso no alcanza. Un contador que debe devolver 1, 2, 3 en llamadas sucesivas; una función que acumula estadísticas de todo lo que procesó; una que guarda resultados ya calculados para no repetir el trabajo.

Hay tres formas de conseguir que algo sobreviva entre llamadas, y conviene distinguirlas porque no son equivalentes:

Variable global. El estado vive afuera, en un espacio compartido. Funciona, y tiene el problema que ya vimos: cualquier parte del programa puede modificarlo, y dos usos independientes de la misma función se pisan entre sí.

Variable local estática. El estado vive adentro de la función y persiste entre llamadas, pero nadie más lo ve. Es mejor que la global porque queda encapsulado; sigue teniendo el problema de que hay un solo estado para todas las llamadas.

Estado explícito. La función recibe el estado como argumento y devuelve el estado modificado. Quien llama se encarga de guardarlo. Es la forma más limpia —la función sigue siendo pura, sin efectos secundarios— y la más incómoda de escribir.

Clausuras

Existe una cuarta forma, más poderosa. Si el lenguaje permite que una función devuelva otra función, y que la función devuelta siga viendo las variables locales de la que la creó, entonces se puede fabricar un contador que lleve su propio estado privado:

```
function crear_contador()
    n = 0
    function contar()
        n = n + 1
        devolver n
    end
    devolver contar
end

c1 = crear_contador()
c2 = crear_contador()
c1()   ->  1
c1()   ->  2
c2()   ->  1
```

Eso es una **clausura**: una función junto con el entorno en que fue definida. La variable `n` ya no existe en ningún lugar visible del programa, pero la función devuelta la sigue viendo, y cada llamada a `crear_contador` produce un `n` distinto. Por eso `c1` y `c2` cuentan por separado, que es justamente lo que la variable estática no permitía.

El nombre viene de que la función «cierra» sobre las variables libres de su cuerpo: las que no son ni parámetros ni locales suyas, sino del entorno donde se la definió. El mecanismo se formuló al estudiar cómo implementar el cálculo lambda con alcance léxico G. J. Sussman y Steele 1975, y es el que hace que las funciones sean verdaderamente valores de primera clase: no alcanza con poder pasarlas como argumento, hay que poder crearlas en tiempo de ejecución con información capturada Abelson, G. J. Sussman y J. Sussman 1996.

Vale insistir en la distinción: un contador con variable global tiene estado persistente, pero *no* es una clausura. Funciona parecido y hasta puede alcanzar para el problema concreto, pero la memoria está en un espacio compartido y no en el entorno privado de la función. La diferencia se nota apenas se necesitan dos contadores.

Currificación y aplicación parcial

La **currificación** transforma una función de varios argumentos en una cadena de funciones de un argumento:

```
sumar(a, b) = a + b
```

```
currificada:  sumar(a)  devuelve una función que espera b

sumar(3)(5)  ->  8
```

La **aplicación parcial** es la operación relacionada: fijar algunos argumentos y obtener una función más específica. `sumar3 = sumar(3)` deja una función de un solo argumento que suma tres a lo que reciba.

Las dos necesitan clausuras para implementarse cómodamente, porque la función que se devuelve tiene que recordar el argumento ya fijado.

En cálculo científico la aplicación parcial es más útil de lo que parece, y aparece constantemente aunque no se la nombre. Los integradores numéricos, los buscadores de raíces y los ajustadores de parámetros esperan recibir una función de una sola variable. Pero el modelo físico casi nunca tiene una sola: tiene la variable independiente y además la masa, la constante del resorte, el coeficiente de amortiguamiento. Fijar los parámetros y quedarse con una función de una variable es exactamente aplicación parcial, y cuando el lenguaje no la ofrece hay que hacerla a mano —pasando los parámetros por una lista adicional, o por una variable global, o armando una estructura que los lleve.

5.2.2. Evaluación dinámica

Casi todos los lenguajes interpretados ofrecen una operación que toma una cadena de texto y la ejecuta como si fuera código. La idea es notable: el programa puede construir programa nuevo mientras corre.

Que sea posible se sigue de algo que a esta altura ya vimos: el intérprete es un programa que recibe texto y lo ejecuta, así que ponerlo a disposición del programa que está corriendo no requiere ningún mecanismo adicional. Llevada al extremo, esta idea da lugar a lenguajes donde el programa y los datos tienen la misma representación, y donde el propio intérprete puede escribirse en el lenguaje que interpreta McCarthy 1960; Abelson, G. J. Sussman y J. Sussman 1996.

Los usos legítimos son acotados y reales: construir una fórmula que el usuario escribió en un archivo de configuración, generar código repetitivo a partir de una plantilla, armar consultas cuya forma depende de los datos.

Los inconvenientes son serios y conviene tenerlos presentes antes de recurrir a esta herramienta:

Los errores aparecen tarde. Un error de sintaxis en una cadena no se detecta al leer el programa sino recién cuando se la evalúa, quizá después de una hora de cálculo.

No se puede analizar. Ninguna herramienta puede revisar, buscar ni reorganizar código que todavía no existe.

Es un riesgo de seguridad. Si la cadena se arma con algo que vino de afuera —un archivo, un formulario, una lectura de red— quien provea ese texto puede hacer que el programa ejecute lo que quiera.

La regla práctica: si el problema se puede resolver con una tabla de funciones, con una estructura de datos o con una función de orden superior, esa solución es preferible. La evaluación dinámica es el último recurso, no el primero.

5.2.3. Procesamiento de texto

Buena parte del trabajo real con datos no es numérica sino textual: leer un archivo de un instrumento y separar sus campos, buscar todas las líneas que cumplen cierta forma, reemplazar un patrón, validar que lo que llegó tenga la estructura esperada. El texto plano es el formato de intercambio universal, y por eso saber procesarlo es una habilidad transversal.

Expresiones regulares

Una **expresión regular** es una notación para describir un conjunto de cadenas. En lugar de buscar un texto fijo, se describe una forma: «uno o más dígitos», «una letra seguida de cualquier cosa y después una be», «algo que termine en punto y coma».

Los elementos básicos, comunes a todos los dialectos:

Literales: un carácter se representa a sí mismo.

Clases: un conjunto de caracteres aceptables, escrito entre corchetes, con rangos y con negación.

Cuantificadores: cero o más, uno o más, cero o uno, o un número determinado de repeticiones del elemento anterior.

Alternativa: una forma u otra.

Anclas: principio de línea, fin de línea, límite de palabra.

Agrupamiento: paréntesis que delimitan una parte, tanto para aplicarle un cuantificador como para recuperar después lo que coincidió con ella.

La idea no es un truco de programación: tiene fundamento formal. Las expresiones regulares describen exactamente el mismo conjunto de lenguajes que reconocen los autómatas finitos, y esa equivalencia es lo que permite implementarlas eficientemente Kleene 1956; Hopcroft, Motwani y Ullman 2006.

Ese fundamento explica también su límite, que es la fuente del error más frecuente. Un autómata finito no puede contar, y por lo tanto una expresión regular no puede reconocer estructuras anidadas de profundidad arbitraria: paréntesis balanceados, etiquetas dentro de etiquetas, bloques dentro de bloques. Todo intento de analizar un formato jerárquico con expresiones regulares funciona con los casos de prueba y falla con el caso real.

Dos advertencias prácticas más. La primera es la **voracidad**: los cuantificadores toman por omisión la coincidencia más larga posible, de modo que un patrón pensado para capturar un campo puede tragarse toda la línea; casi todos los dialectos tienen versiones

no voraces. La segunda es el costo: los motores basados en retroceso pueden tardar un tiempo que crece exponencialmente con ciertos patrones y ciertas entradas Friedl 2006.

Los dialectos difieren en detalles —qué hay que escapar, cómo se delimita el patrón, qué extensiones se aceptan— y el que se tomó como referencia de hecho fue el de un lenguaje pensado desde el principio para el procesamiento de texto Wall, Christiansen y Orwant 2000b.

Cuándo no usarlas

Antes de escribir una expresión regular conviene preguntarse si el problema no es más simple. Separar una línea por un carácter delimitador, buscar una subcadena fija o cortar por posición son operaciones que existen en cualquier lenguaje, se leen mejor y no tienen sorpresas.

Y si el formato es jerárquico, lo que corresponde no es una expresión regular sino un analizador, que es el tema que sigue. Para los formatos de intercambio conocidos, además, hay bibliotecas ya escritas y probadas: analizar un formato estándar a mano es reescribir mal algo que ya está bien hecho.

5.2.4. De texto a estructura: análisis léxico y sintáctico

Convertir texto en una estructura manipulable es un problema tan frecuente que tiene teoría y herramientas propias. Aparece al leer un archivo de configuración, al interpretar una fórmula que el usuario escribió, al procesar la salida de un instrumento, y en el caso extremo, al implementar un lenguaje.

El proceso se divide en dos etapas, y esa división es la idea principal que conviene retener Aho et al. 2006:

Análisis léxico. Convierte la secuencia de caracteres en una secuencia de *unidades léxicas*: números, identificadores, operadores, paréntesis. Descarta los espacios y los comentarios. Acá las expresiones regulares son la herramienta adecuada, porque cada tipo de unidad es un lenguaje regular.

Análisis sintáctico. Toma esa secuencia y construye una estructura que refleja la jerarquía: qué es operando de qué, qué está dentro de qué. Acá las expresiones regulares ya no alcanzan, por la razón de la sección anterior.

La estructura resultante suele ser un árbol. La expresión `x = 3 + 4` se convierte en algo con la forma «asignación, cuyo destino es la variable x y cuyo origen es una suma de 3 y 4». Sobre esa forma sí se puede trabajar: evaluarla, transformarla, verificarla o traducirla.

Para decir qué combinaciones son válidas se usa una **gramática**, que es un conjunto de reglas. La de las expresiones aritméticas es el ejemplo canónico:

```
expresion = termino  { ("+" | "-") termino }
termino   = factor   { ("*" | "/") factor }
factor    = numero | "(" expresion ")"
```

Esas tres líneas contienen la precedencia de operadores: la multiplicación queda más abajo en la jerarquía, y por eso se agrupa antes. Y el paréntesis, al reintroducir `expresion` dentro de `factor`, es lo que hace la gramática recursiva y lo que la saca del alcance de las expresiones regulares.

Un **analizador descendente recursivo** es la traducción directa de esa gramática a código: una función por cada regla, que llama a las funciones de las reglas que aparecen a su derecha. Es el tipo de analizador que se escribe a mano con menos esfuerzo, y su valor pedagógico es alto porque la correspondencia entre la gramática y el programa es literal.

En orden de preferencia práctica: si existe una biblioteca para el formato, usarla; si el formato es plano, separar por delimitadores; si es jerárquico y propio, escribir el analizador descendente recursivo; y sólo en casos grandes, recurrir a un generador de analizadores. Escribir un analizador a mano para algo que ya tiene norma y biblioteca es casi siempre trabajo perdido.

5.2.5. Estructuras de datos más allá del arreglo

El cálculo numérico se apoya en un solo tipo de estructura: el arreglo homogéneo de números, con todos sus elementos del mismo tipo y accesibles por un índice entero. Para todo lo demás hacen falta otras.

Diccionario o tabla asociativa. Asocia claves a valores, con la clave típicamente textual. Es la estructura de propósito general más útil que existe: representa un registro con campos, una configuración, un conteo por categoría, un índice. La operación de buscar por clave es de costo aproximadamente constante, lo que la hace muy superior a recorrer una lista comparando.

Lista heterogénea. Una secuencia cuyos elementos pueden ser de tipos distintos, incluso otras listas. Es lo que permite representar estructuras irregulares, como el árbol que devuelve un analizador.

Registro o estructura. Un conjunto de campos con nombre. A diferencia del diccionario, los campos se conocen al escribir el programa. Sirve para dejar de pasar diez argumentos sueltos y pasar uno solo que los agrupa, y para que el conjunto tenga nombre propio.

Referencia. Un valor que designa a otro valor en lugar de contenerlo. Es lo que permite que dos partes del programa compartan la misma estructura en vez de copias, y lo que hace posibles las estructuras que se referencian entre sí, como listas enlazadas y grafos.

Los lenguajes difieren mucho en cuáles de éstas son nativas y cómo se escriben, y esa diferencia es una de las que más se nota al cambiar de lenguaje. Donde no existen se emulan: un diccionario con dos arreglos paralelos de claves y valores, un registro con una estructura o con convenciones de nombres, una referencia pasando y devolviendo el valor completo.

5.2.6. Tipos definidos por el usuario y sobrecarga

Los tipos que trae el lenguaje no siempre alcanzan. Un número complejo, una magnitud con unidad, un polinomio, una matriz dispersa: en todos esos casos hay un conjunto de valores y un conjunto de operaciones que tienen sentido sobre ellos.

Definir un tipo propio consiste en decidir dos cosas: cómo se representa —qué campos tiene, qué invariantes deben cumplirse— y qué operaciones se le pueden aplicar. Es la misma separación entre interfaz e implementación que ya vimos en las funciones, ahora aplicada a los datos: quien usa el tipo trabaja con las operaciones, no con la representación, y por eso la representación se puede cambiar después.

La **sobrecarga de operadores** es el paso siguiente: hacer que los símbolos habituales funcionen con el tipo nuevo, de modo que sumar dos magnitudes con unidad se escriba con el signo más y no con una llamada a función. La ventaja es que las fórmulas del programa se parecen a las fórmulas del papel, que en cálculo científico es exactamente lo que se quiere.

Los cuidados son dos. El primero es no traicionar la expectativa: si un símbolo hace algo que no se parece a lo que hace habitualmente, quien lee el código va a entender mal y con confianza. El segundo es no confundir sobrecarga con conversión implícita: que se pueda sumar una magnitud con unidad y un número desnudo suele ser un error, no una comodidad.

Vale notar la conexión con lo visto en la primera parte. Un tipo con sus operaciones es, esencialmente, un objeto sin las clases; y lo que hace que la sobrecarga funcione es el polimorfismo, es decir que el mismo símbolo se resuelva de distinta manera según el tipo de los operandos.

5.2.7. Cómo corre el programa: entorno de ejecución

Hasta acá el programa se pensó como algo que alguien escribe y ejecuta sentado frente a la máquina. En el trabajo real eso es la excepción.

Ejecución por lotes. El programa corre sin nadie delante, con todos sus datos definidos de antemano y con la salida a archivos. La consecuencia inmediata es que no puede haber ninguna instrucción que pida un dato interactivamente: todo entra por argumentos de línea de comandos o por un archivo de configuración. Es también la razón por la que un programa debe informar su avance y terminar con un código de salida que indique si funcionó, porque nadie va a estar mirando.

Servidores sin interfaz gráfica. El cálculo pesado se hace en máquinas que no tienen pantalla. Un programa que intenta abrir una ventana ahí falla, y por eso las bibliotecas de graficación tienen un modo sin pantalla que escribe directamente a archivo. Conviene que la parte de cálculo esté separada de la de visualización desde el principio, justamente para poder correr la primera en un servidor.

Automatización de la reconstrucción. Cuando un resultado depende de una cadena de pasos —procesar los datos, ajustar el modelo, generar las figuras, compilar el informe— rehacerlos a mano cada vez es lento y propenso a olvidos. La solución

clásica es declarar qué depende de qué y dejar que una herramienta calcule qué hay que rehacer después de un cambio S. I. Feldman 1979. Nótese que es un enfoque declarativo: se describen las relaciones, no el orden de las operaciones.

Aislamiento del entorno. El problema de la reproducibilidad que vimos —el mismo programa da distinto en dos máquinas por versiones distintas de una biblioteca— tiene una respuesta más fuerte que documentar las versiones: empaquetar el entorno completo junto con el código, de modo que se ejecute igual en cualquier máquina que pueda correr el contenedor Boettiger 2015. Tiene su costo, y no reemplaza a las buenas prácticas: un cálculo irreproducible dentro de un contenedor sigue siendo irreproducible, sólo que ahora en un paquete más grande.

Un hilo común atraviesa los cuatro puntos: el programa que se escribe para ser usado por otros —o por uno mismo dentro de seis meses, en otra máquina— tiene requisitos distintos del que se escribe para correr una vez. Separar el cálculo de la interacción, no depender de lo que haya instalado, y dejar registrado cómo se llega del dato crudo al resultado son las tres decisiones que hacen la diferencia.

5.3. Programación en Scilab

La unidad teórica presentó los conceptos sin atarlos a ninguna herramienta: qué es un algoritmo, cómo se estructura el control, qué significa el alcance de una variable, qué distingue un paradigma de otro. Esta parte muestra cómo se escriben esas mismas cosas en Scilab, que es el entorno con el que trabajamos en el laboratorio.

El orden de lectura importa: lo que hay que aprender son los conceptos, y la sintaxis es la forma que toman en un entorno particular. Quien entienda qué es un bucle con condición al inicio lo va a reconocer en cualquier lenguaje, aunque cambien las palabras.

5.3.1. Por qué Scilab

Actualmente existen numerosos lenguajes de programación utilizados en ciencia e ingeniería. Cada uno posee fortalezas particulares y ningún lenguaje es el más adecuado para todas las aplicaciones.

En esta asignatura se utiliza **Scilab** porque fue diseñado específicamente para el cálculo científico. Su sintaxis es sencilla, permite trabajar naturalmente con vectores y matrices y dispone de una gran cantidad de funciones matemáticas incorporadas. Esto permite que el estudiante centre su atención en la resolución de problemas físicos y matemáticos, sin tener que preocuparse inicialmente por aspectos propios de la programación de propósito general INRIA 2026; Scilab Group 2000.

Hay un argumento adicional y más fuerte. En Ingeniería y en Ciencias no se enseña un lenguaje: se enseñan algoritmos y métodos de resolución de problemas. El lenguaje cambia con los años —hace treinta años era Fortran, después apareció Matlab, hoy domina Python, y dentro de diez años probablemente aparezca otro. En cambio, plantear un algoritmo, discretizar un problema, resolver un sistema lineal, integrar una ecuación dife-

rencial y procesar datos experimentales son habilidades que permanecen durante toda la vida profesional y que pueden implementarse prácticamente en cualquier lenguaje.

Desde ese punto de vista, Scilab no es el objetivo de la materia sino el vehículo para enseñar pensamiento computacional aplicado a la Física y a las Energías Renovables.

Es importante destacar que esta asignatura forma parte de los planes de estudio de la **Licenciatura en Energías Renovables** y de la **Tecnicatura Universitaria en Energía Solar**. En ambas carreras resulta fundamental la utilización de herramientas de cálculo numérico y simulación para analizar datos experimentales, resolver problemas de ingeniería y modelar sistemas energéticos. Scilab permite desarrollar algoritmos, resolver sistemas de ecuaciones, realizar simulaciones, procesar datos experimentales y generar gráficos con una sintaxis muy próxima al lenguaje matemático Urroz 2001; Calvo Rolle 2003.

No obstante, la formación de un profesional moderno no debería limitarse a un único lenguaje. Se recomienda que, una vez adquiridos los fundamentos, el estudiante incorpore Python Downey 2015 y, posteriormente, al menos un lenguaje compilado como C o Fortran Kernighan y Ritchie 1978. Ningún científico trabaja hoy con un solo lenguaje: es habitual desarrollar una simulación de alto rendimiento en un lenguaje compilado, automatizar su ejecución con guiones del intérprete de comandos Ramey y Fox 2025, procesar los resultados en Python y hacer los cálculos exploratorios y los prototipos numéricos en Scilab.

Para el primer contacto con el entorno y para consultar mientras se trabaja hay material introductorio en castellano Romero Mora 2014; Mora Escobar 2001; Condorí, Finetti y Saravia 2011; Echevarría Líbano 2013 y en inglés Baudin y Kroeter 2011; Gomez 2003; Scilab 2024; la referencia completa de funciones es la ayuda del propio sistema Scilab Enterprises 2026c; Scilab Group 2026.

5.3.2. La consola y el archivo

En Scilab la distinción entre comando y programa, que la teoría presentó en abstracto, es inmediata y visible. La consola ejecuta y responde de a una instrucción por vez; el archivo guarda una secuencia para ejecutarla como unidad. Ambas cosas usan exactamente el mismo lenguaje, y ésa es la propiedad que hace cómodo el ciclo de trabajo: lo que se probó a mano se pega en un archivo sin traducir nada.

Hay dos tipos de archivo, con una diferencia de propósito:

`.sce`: un guión, es decir una secuencia de instrucciones que se ejecuta de principio a fin. Se corre con `exec(.ªrchivo.sce")` o con el botón de ejecutar del editor.

`.sci`: un archivo de definiciones de funciones. No ejecuta nada por sí mismo: al cargarlo con `exec`, las funciones que contiene quedan disponibles para ser llamadas.

Esa separación es la versión concreta de la modularidad de la que habla la teoría: el `.sci` es la biblioteca, el `.sce` es el programa que la usa. Conviene adoptarla desde el principio, porque es lo que permite después probar las funciones por separado y reutilizarlas en otro trabajo.

Un detalle que confunde al principio: el punto y coma al final de una instrucción suprime la impresión del resultado. Sin él, Scilab muestra en la consola el valor de todo lo que calcula, lo cual es útil mientras se explora e insoportable dentro de un bucle de diez mil iteraciones.

5.3.3. Entrada y salida

En un programa, además de calcular, suele ser necesario leer datos y mostrar resultados. Para pedir un dato al usuario se usa `input`. Si se espera texto, se agrega `"string"` como segundo argumento:

```
nombre = input("Ingrese su nombre: ", "string");
edad = input("Ingrese su edad: ");
```

Para mostrar un resultado simple se usa `disp`:

```
disp(nombre);
disp(edad);
```

Para mostrar texto combinado con valores y controlar el formato se usa `printf`. El símbolo `\n` produce un salto de línea, `%f` imprime un número real y `%s` imprime una cadena:

```
radio = 5;
area = %pi * radio^2;
printf("radio = %f m\n", radio);
printf("area = %f m^2\n", area);
```

Estas instrucciones corresponden al flujo de entrada y salida de un programa: leer datos, procesarlos y presentar resultados.

Conviene distinguir las tres, porque hacen cosas distintas. `disp` muestra el valor tal como Scilab lo representa, sin control sobre el formato. `printf` permite decidir cuántos decimales, cómo se alinean las columnas y qué texto acompaña a cada número, y es la que corresponde cuando la salida la va a leer una persona o la va a leer otro programa. La cantidad de decimales no es un detalle estético: imprimir ocho cifras de una magnitud medida con tres es afirmar una precisión que no se tiene.

Una advertencia sobre `input`: detiene el programa a esperar una respuesta. Eso está bien para un ejercicio, y está mal para un cálculo que va a correr sin nadie delante o dentro de un guión automatizado. En cuanto el programa crezca, conviene que los datos entren por argumentos de la función o desde un archivo, y reservar `input` para lo interactivo.

5.3.4. Estructuras de control

Sentencia condicional `if`

La sentencia condicional `if` permite ejecutar una instrucción solamente cuando se cumple una determinada condición.

Ejemplo

Se desea escribir un programa que calcule el valor absoluto de un número.

Si el número ingresado es negativo debe multiplicarse por -1; si es positivo debe dejarse sin modificar.

El algoritmo puede resumirse de la siguiente forma:

1. Introducir el número n.
2. Si $n < 0$, calcular $m = -n$.
3. En caso contrario, hacer $m = n$.

Obsérvese que el algoritmo se escribió antes que el programa, en castellano y sin sintaxis. Ése es el orden correcto y conviene sostenerlo: primero se decide qué hay que hacer, después cómo se escribe.

La sintaxis general de la sentencia es

```
if condicion then

    instrucciones

else

    instrucciones

end
```

El programa correspondiente al ejemplo es

```
n=input("Introduzca el numero");

if n<0 then
    m=-1*n;
else
    m=n;
end

disp(m,"Valor absoluto=");
```

Tres particularidades de Scilab que conviene retener. La palabra `then` es obligatoria después de la condición, y todo bloque se cierra con `end`. Los valores lógicos se escriben `%T` y `%F`, con el signo de porcentaje que Scilab usa para sus constantes predefinidas, igual que en `%pi` o `%e`. Y si la condición se evalúa sobre un vector, Scilab exige que *todos* sus elementos sean verdaderos para tomar la rama; cuando lo que se quiere es preguntar si alguno lo es, hay que usar `or` explícitamente.

Ciclo incondicional `for`

El ciclo `for` se utiliza para ejecutar una secuencia de instrucciones un número conocido de veces.

Ejemplo

Se desea mostrar los números del 1 al 10 en pantalla. Una forma poco eficiente sería escribir:

```
disp(1)
disp(2)
disp(3)
...
disp(10)
```

El mismo procedimiento puede resumirse mediante un ciclo:

```
for n=1:10
    disp(n)
end
```

La sintaxis general del comando es

```
for variable=expresion

    instrucciones

end
```

Ejemplo

Sea el vector

```
v=[2,3,5,2,5,7,9,0,8,1];
```

Para recorrer todos sus elementos puede utilizarse

```
for i=1:10

disp("El elemento "
    +string(i)
    +" es igual a "
    +string(v(i)));

end
```

Acá aparece algo propio de este tipo de entornos y que conviene decir temprano: en Scilab, muchos bucles no hacen falta. La operación elemento a elemento sobre un vector completo se escribe sin ciclo, y además de ser más corta es mucho más rápida, porque el trabajo lo hace código compilado en vez del intérprete instrucción por instrucción. Escribir `y = x.^2` en lugar de recorrer `x` con un `for` es la forma idiomática, y es un caso de lo que la teoría llamó función de orden superior: se aplica una operación a toda una colección sin describir el recorrido.

El bucle explícito sigue siendo necesario cuando cada paso depende del anterior —un método iterativo, la integración de una ecuación diferencial paso a paso— porque ahí no hay operación vectorial posible.

Sentencia condicional `select-case`

La sentencia `select-case` permite ejecutar diferentes bloques de instrucciones dependiendo del valor de una variable.

Su sintaxis general es

```
select expresion

case valor1 then

    instrucciones

case valor2 then

    instrucciones

...

else

    instrucciones

end
```

Ejemplo

Ejecute el siguiente programa

```
opcion=input(
"Introduzca un numero entero entre 1 y 4");

select opcion

case 1 then

disp("Este programa no hara nada")

case 2 then

disp("Ha ganado un millon de pesos!");

case 3 then

for i=1:5
    disp("Me gusta programar con Scilab")
end

end
```

Sentencia de iteración `while`

La sentencia `while` permite repetir un bloque de instrucciones mientras se cumpla una determinada condición. La condición se evalúa *antes* de cada iteración; si es falsa desde el comienzo, el bloque no se ejecuta ninguna vez.

Ejemplo

Se desea calcular el factorial de un número entero $n \geq 0$, definido como $n! = 1 \cdot 2 \cdot 3 \cdots n$. El algoritmo puede resumirse de la siguiente forma:

1. Introducir el número n.
2. Inicializar el resultado $m = 1$ y el contador $i = 1$.
3. Mientras $i \leq n$: calcular $m = m \cdot i$ y luego $i = i + 1$.
4. Mostrar el resultado.

El programa correspondiente es

```
n = input("Introduzca el numero entero");
m = 1;
i = 1;
while i <= n
    m = m * i;
    i = i + 1;
end
disp(m, "Factorial =");
```

Nótese que el contador se incrementa dentro del bloque. Ésa es la parte que hace que la condición pueda volverse falsa, y olvidarla produce un bucle que no termina: la falla más común de esta estructura. En los métodos iterativos del cálculo numérico, donde la condición de salida es que el error baje de una tolerancia, conviene además agregar un tope de iteraciones y distinguir al terminar si se salió por convergencia o por agotamiento.

Simulación del ciclo `do - while`

Scilab no dispone de una instrucción `do - while` nativa. Sin embargo, el mismo comportamiento —ejecutar el bloque *al menos una vez* y repetirlo mientras se cumpla una condición— puede obtenerse combinando `while %T` con una instrucción `break` condicional al final del bloque. La estructura equivalente es

```
while %T
    instrucciones
    if ~condicion then
        break
    end
end
```

Éste es un buen ejemplo de por qué conviene aprender los conceptos antes que la sintaxis: la estructura «repetir al menos una vez» existe como idea aunque el lenguaje no tenga una palabra reservada para ella, y se construye con lo que hay.

Ejemplo

Se desea acumular la suma $S = 1+2+3+\ldots$, mostrando cada término, hasta que la suma supere un límite L ingresado por el usuario. El algoritmo puede resumirse de la siguiente forma:

1. Introducir el límite L.
2. Inicializar la suma $S = 0$ y el contador $i = 1$.
3. Calcular $S = S + i$ y mostrar i y la suma parcial.
4. Si $S \geq L$, terminar; en caso contrario, hacer $i = i + 1$ y volver al paso 3.

El programa correspondiente al ejemplo es

```
L = input("Introduzca el limite");
S = 0;
i = 1;
while %T
    S = S + i;
    printf("i=%d   S=%.0f\n", i, S);
    if S >= L then
        break
    end
    i = i + 1;
end
disp(S, "Suma final =");
```

5.3.5. Funciones definidas por el usuario

La teoría presentó la función como el mecanismo fundamental de abstracción Abelson, G. J. Sussman y J. Sussman 1996. En Scilab se escribe así:

```
function [s1, s2] = nombre(e1, e2)
    // cuerpo
endfunction
```

Los corchetes de la izquierda enumeran las salidas y los paréntesis de la derecha las entradas. Con una sola salida los corchetes pueden omitirse. Lo que hace que una salida efectivamente valga es que se le asigne un valor dentro del cuerpo: no existe una instrucción de retorno que devuelva un valor, sino que se asigna a la variable declarada como salida.

Tres cosas específicas del entorno:

Las funciones viven en archivos `.sci` y se cargan con `exec`. Una función definida en la consola desaparece con `clear`; una guardada en archivo, no.

Los argumentos son opcionales por posición. La variable `argn` informa cuántos se recibieron efectivamente, lo que permite dar valores por omisión a los últimos parámetros. Es cómodo y hay que documentarlo, porque quien lee la llamada no ve qué se omitió.

Los argumentos se pasan por valor. Modificar un parámetro dentro de la función no afecta a la variable del llamador. Por eficiencia, Scilab evita la copia real mientras no se modifique el dato, pero desde el punto de vista de quien programa el efecto es el del pasaje por valor.

Una función que sólo calcula y devuelve se puede probar sola. Una que además pide datos con `input` o dibuja en pantalla, no. Por eso conviene que las funciones reciban todo lo que necesitan por argumentos y devuelvan todo lo que producen, y que la entrada y salida quede en el guión que las llama.

5.3.6. Variables

Scilab es de tipado dinámico: una variable no se declara, existe cuando se le asigna algo y puede cambiar de tipo. Eso es lo que hace cómodo el trabajo exploratorio y lo que hace que ciertos errores aparezcan tarde.

La particularidad más importante del entorno es que el tipo básico es la matriz. Un escalar es una matriz de uno por uno y un vector es una matriz con una sola fila o columna; no son categorías distintas. De ahí se sigue la distinción entre los operadores matriciales y los operadores elemento a elemento, que llevan un punto delante: `*` es producto de matrices y `.*` es producto término a término, y lo mismo vale para la división y la potencia. Es la fuente de error más frecuente al empezar, y no da un mensaje: da un resultado distinto del esperado, o un error de dimensiones que desconcierta.

Sobre el alcance y el tiempo de vida:

Las variables de una función son locales. Dejan de existir cuando la función termina.

`global` declara una variable compartida, y hay que declararla así tanto donde se define como en cada función que la usa. Vale la advertencia de la teoría: dificulta el razonamiento y conviene evitarla.

El espacio de trabajo de la consola persiste entre ejecuciones. Ésa es la causa de un error que cuesta encontrar: un guión que funciona porque una variable quedó definida de una corrida anterior, y que falla en otra máquina. La costumbre de empezar los guiones con `clear` evita justamente eso, y es un requisito de reproducibilidad, no una manía de orden.

Para inspeccionar el estado hay `whos`, que lista las variables con su tipo y tamaño, y `typeof`, que informa el tipo de una en particular.

5.3.7. Los paradigmas en Scilab

Scilab no impone un paradigma, y en la práctica se programa combinando varios:

Imperativo y estructurado es el estilo predominante, con las tres estructuras de control que vimos y sin salto incondicional.

Vectorial, que es funcional encubierto: aplicar una operación a toda una matriz sin escribir el recorrido es exactamente aplicar una función a una colección. Además, las funciones son valores: se pueden guardar en variables y pasarse como argumento a otra función, que es lo que hacen los integradores numéricos y los buscadores de raíces cuando reciben la función a integrar.

Basado en datos estructurados: las estructuras y las listas permiten agrupar valores heterogéneos bajo un nombre, que es el primer paso hacia la orientación a objetos —agrupar los datos que van juntos— sin llegar a las clases.

Por bloques: la herramienta de modelado gráfico del entorno permite construir un sistema conectando bloques en vez de escribir las ecuaciones, que es una forma declarativa de especificar un modelo Scilab Enterprises 2026d; Campbell, Chancelier y Nikoukhah 2006.

Orientado a eventos cuando se construyen interfaces gráficas, que es materia de otra unidad.

Nada de esto es exclusivo del entorno, y ése es el punto: lo que se aprende acá se traslada.

5.3.8. Buenas prácticas aplicadas

Las recomendaciones de la teoría se concretan así:

Nombres descriptivos. Scilab no limita la longitud de los identificadores; no hay ninguna razón para llamar `a1` a la velocidad inicial.

Constantes con nombre. En lugar de multiplicar por `9.8`, definir `g = 9.8` con un comentario que indique qué es y en qué unidades. Las constantes matemáticas ya vienen: `%pi`, `%e`, `%eps`.

Comentarios que expliquen por qué. Se escriben con `//`. En cálculo científico, el comentario indispensable es el que registra de dónde salió la fórmula, qué aproximación se usa y en qué rango vale.

Documentación de la función. Las líneas de comentario que siguen inmediatamente a `function` son las que muestra `help` sobre esa función. Conviene poner ahí el propósito, los parámetros con sus unidades y el valor de retorno.

Pruebas. La función `assert_checkequal` y sus variantes permiten escribir comprobaciones ejecutables. Un archivo de pruebas junto a cada `.sci`, con los casos que se conocen a mano, es la diferencia entre poder cambiar el código y no atreverse.

Validación numérica. Los casos que casi siempre están disponibles: un valor que se conoce analíticamente, el comportamiento cuando un parámetro tiende a cero, una magnitud que debe conservarse a lo largo de la simulación, y la comprobación de que al refinar el paso el resultado converge al orden esperado. Recuérdese la distinción de la teoría: eso verifica que el programa resuelve bien las ecuaciones, no que las ecuaciones describan el fenómeno.

Sobre la composición con otras herramientas: Scilab puede ejecutar programas del sistema y leer su salida, lo que permite combinarlo con todo lo que ya está instalado en vez de reescribirlo Scilab Enterprises 2026b; Raymond 2003. La escritura de matrices a archivos de texto Scilab Enterprises 2026a es la interfaz habitual con las herramientas de graficación y con otros programas.

5.3.9. Módulos y reutilización

La progresión natural es: una función que se repite se saca a un `.sci`; un conjunto de `.sci` relacionados se agrupa en un directorio; y ese directorio se convierte en una biblioteca que se carga de una vez con `genlib` y queda disponible por su nombre.

El entorno tiene además un sistema de módulos externos, con paquetes que aportan funcionalidad específica y se instalan desde el propio Scilab. Vale acá la advertencia de la teoría sobre reproducibilidad: si un cálculo depende de un módulo externo, hay que registrar cuál y en qué versión, porque el mismo programa con otra versión puede dar otro resultado.

5.3.10. Control de versiones

Todo lo dicho en la teoría se aplica sin cambios: los archivos `.sce` y `.sci` son texto plano, que es exactamente el caso en que un sistema de control de versiones funciona mejor, porque puede mostrar qué líneas cambiaron entre una versión y otra.

Lo que conviene decidir es qué no entra al repositorio. Entra el código, los datos primarios y la documentación. No entran los resultados intermedios, las figuras que el programa genera ni los archivos temporales del entorno: se pueden volver a producir ejecutando el programa, y lo único que hacen en la historia es ensuciarla. Guardar la figura y no el guión que la produjo es, además, el orden equivocado de prioridades.

5.4. Scilab: temas avanzados

La segunda parte teórica presentó una serie de mecanismos —clausuras, currificación, evaluación dinámica, expresiones regulares, análisis léxico y sintáctico, tipos propios con sobrecarga— señalando que cada lenguaje los tiene de forma nativa, a medias, o hay que emularlos. Scilab está en el segundo y el tercer caso casi siempre, porque fue pensado para cálculo numérico y no para manipulación simbólica de texto ni para metaprogramación.

Eso no los vuelve inaccesibles: los tiene todos, en formas que vale la pena conocer, entre otras cosas porque emular un mecanismo obliga a entenderlo mejor que usarlo ya hecho.

La referencia de todas las funciones que siguen es la ayuda del sistema Scilab Enterprises 2026c.

5.4.1. Estado que sobrevive a la llamada

De las cuatro formas que vimos —global, estática local, estado explícito y clausura— Scilab ofrece cómodamente la primera y la tercera.

Un contador con estado global:

```
function y = contador()
    global n
    if ~exists("n", "global") then n = 0; end
    n = n + 1;
    y = n;
endfunction
```

Funciona como contador persistente, pero conviene ser preciso con los nombres: la memoria está en una variable global, no encerrada en el entorno léxico privado de la función. Es una simulación práctica del comportamiento de una clausura, no una clausura. La diferencia se nota apenas se quieren dos contadores independientes: con esta forma comparten el mismo `n` y se pisan entre sí.

Una variante con reinicio, que muestra el patrón completo:

```
function y = contador(reset)
    global contador_valor
    if ~exists("reset", "local") then
        reset = %f;
    end
    if reset | ~exists("contador_valor", "global") then
        contador_valor = 0;
    end
    contador_valor = contador_valor + 1;
    y = contador_valor;
endfunction
```

Uso:

```
contador()
contador()
contador(%t)    // reinicia y devuelve 1
```

La limitación de fondo es que en Scilab una función no puede capturar automáticamente las variables de su entorno como ocurre en los lenguajes donde la clausura es nativa. Cuando hacen falta varios estados independientes, la alternativa correcta es la tercera forma de la teoría: pasar el estado explícitamente y devolverlo modificado.

5.4.2. Currificación y aplicación parcial

Se simulan con estructuras: una función devuelve el estado capturado y otra lo aplica.

```
function f = sumar_parcial(a)
    f = struct("a", a);
endfunction

function y = aplicar_suma(f, b)
    y = f.a + b;
endfunction

s = sumar_parcial(3);
aplicar_suma(s, 10)
```

Comparado con `sumar(3)(5)` de un lenguaje con clausuras, acá la aplicación queda partida en dos pasos explícitos. Es más verboso y tiene una ventaja pedagógica: se ve exactamente qué se está capturando y dónde vive.

Éste es además el mecanismo que aparece en la práctica cuando hay que pasarle a un integrador o a un buscador de raíces una función que depende de parámetros: se arma la estructura con los parámetros del modelo y se la pasa como argumento adicional.

5.4.3. Estructuras como objetos

Scilab no tiene clases, pero tiene estructuras, y con la convención de que la función reciba y devuelva la estructura se obtiene algo muy parecido a un método:

```
function p = cumplir_anios(p)
    p.edad = p.edad + 1;
endfunction

p = struct("nombre", "Ana", "edad", 20);
p = cumplir_anios(p);
```

Lo que hace evidente el pasaje por valor de Scilab es la reasignación: si se escribiera sólo `cumplir_anios(p)` sin asignar, la estructura original no cambiaría. Ésa es la diferencia entre este esquema y un método de verdad, donde el objeto se modifica en el lugar.

La estructura cumple además los papeles de registro y de diccionario de la teoría. Como registro, con campos conocidos al escribir el programa; como diccionario, accediendo con un nombre de campo calculado en tiempo de ejecución:

```
h = struct();
h.nombre = "Ana";
h.edad = 20;

clave = "edad";
h(clave)
```

Para colecciones heterogéneas —elementos de tipos distintos, incluso otras listas— está

`list`, que es la lista heterogénea de la teoría:

```
L = list(10, "texto", [1 2 3], %t);
L(1)
L(2)
```

5.4.4. Sobrecarga de operadores

Con `mlist` se pueden definir tipos propios y darles comportamiento especial frente a los operadores, de modo que un objeto responda distinto a las operaciones habituales. Es la sobrecarga descrita en la teoría, con sus dos cuidados: que el símbolo siga significando algo parecido a lo que significa siempre, y que no se confunda sobrecarga con conversión implícita.

Es un mecanismo avanzado y de uso poco frecuente en el laboratorio, pero conviene saber que existe: es lo que permitiría, por ejemplo, definir una magnitud con unidad y que sumarla con el signo más verifique que las unidades sean compatibles.

5.4.5. Evaluación dinámica

Scilab tiene las dos operaciones que la teoría describió: `evstr` evalúa una expresión y devuelve su valor, `execstr` ejecuta una instrucción completa.

```
cmd = "x = 2 + 3";
execstr(cmd);
x
```

Y para una expresión matemática:

```
s = "3 + 4 * 5";
r = evstr(s);   // 23
```

Valen las tres advertencias de la teoría, y una acá es concreta: esto **no** es un evaluador seguro de expresiones, porque ejecuta código Scilab. Si la cadena viene de un archivo de configuración o de algo que escribió el usuario, se está permitiendo que ese texto haga cualquier cosa que pueda hacer un programa de Scilab. Para evaluar fórmulas de origen no controlado hace falta un analizador propio, que es el tema de más abajo.

5.4.6. Expresiones regulares

Scilab tiene soporte con `regexp`, y el patrón se escribe entre barras, como en el dialecto de referencia:

```
s = "abc123def456";
r = regexp(s, "/[0-9]+/");
```

Para obtener además las posiciones de lo encontrado:

```
s = "abc123def456";
[m, ini, fin] = regexp(s, "/[0-9]+/");

// m   = ["123" "456"]
// ini = [4 10]
// fin = [6 12]
```

Patrones de uso frecuente:

```
"/[0-9]+/"          // uno o mas digitos
"/[A-Za-z]+/"       // letras
"/[a-z]+/"          // minusculas
"/[A-Z]+/"          // mayusculas
"/[a-zA-Z0-9_]+/"   // palabra simple
"/^hola/"           // empieza con hola
"/mundo$/"          // termina con mundo
"/a.b/"             // a, cualquier caracter, b
"/a*/"              // cero o mas a
"/a+/"              // una o mas a
"/a?/"              // cero o una a
```

Un ejemplo de extracción:

```
s = "contacto: ana@test.org y bob@mail.com";
emails = regexp(s,
   "/[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}/");
```

Búsqueda y reemplazo sin patrones

Para casos simples conviene no usar expresiones regulares, que es lo que ya recomendamos en la teoría. Scilab tiene funciones directas:

```
strindex("hola mundo", "mundo")          // posicion de una
   subcadena
s2 = strsubst("hola mundo", "mundo", "Scilab")   // reemplazo
   literal
```

Acá aparece una limitación real frente a los lenguajes pensados para texto: Scilab no tiene una operación de sustitución con expresión regular en una sola instrucción. Lo habitual es localizar con `regexp`, obtener las posiciones y armar el reemplazo a mano trabajando con esas posiciones.

5.4.7. De texto a estructura

Separar por delimitador

El caso más simple, y el que resuelve la mayoría de los archivos de datos:

```
s = "nombre=Ana";
partes = tokens(s, "=");
clave = partes(1);
valor = partes(2);
```

Archivos de configuración

Con eso ya se puede leer un archivo de pares clave-valor y armar una estructura:

```
function d = parse_config(lineas)
    d = struct();
    for i = 1:size(lineas, "*")
        linea = stripblanks(lineas(i));
        if linea == "" then
            continue;
        end
        partes = tokens(linea, "=");
        if size(partes, "*") == 2 then
            clave = stripblanks(partes(1));
            valor = stripblanks(partes(2));
            d(clave) = valor;
        end
    end
endfunction
```

Uso:

```
lineas = [
  "nombre = Ana"
  "edad = 20"
  "curso = Scilab"
];
d = parse_config(lineas);
d.nombre
d.edad
```

Es exactamente el mecanismo que la teoría recomendaba para sacar los parámetros del código a un archivo aparte.

Análisis léxico

Antes de un analizador sintáctico suele haber un lexer, que separa las unidades léxicas. Acá las expresiones regulares son la herramienta adecuada, porque cada tipo de unidad es un lenguaje regular:

```
s = "3 + 4 * 5";
nums = regexp(s, "/[0-9]+/");
ops  = regexp(s, "/[+*\-/]/");
```

Análisis sintáctico

Para expresiones con paréntesis, precedencia o bloques anidados, las expresiones regulares no alcanzan, por la razón formal que vimos: un autómata finito no cuenta Hopcroft, Motwani y Ullman 2006. Ahí hay que escribir un analizador, normalmente descendente recursivo, con una función por regla de la gramática Aho et al. 2006:

```
// parse_expr()
// parse_term()
// parse_factor()
```

Sobre la gramática de expresiones aritméticas:

```
expr   = term   { ("+" | "-") term }
term   = factor { ("*" | "/") factor }
factor = numero | "(" expr ")"
```

Scilab puede implementar eso, pero es más trabajo que en un lenguaje con mejores herramientas de texto.

Qué usar en cada caso

`tokens` para separar texto simple por un delimitador.

`regexp` para reconocer patrones.

`mgetl` para leer un archivo línea por línea.

`csvRead` para leer datos separados por comas ya estructurados en columnas.

`evstr` para evaluar expresiones Scilab confiables.

`execstr` para ejecutar código generado.

Analizador propio para formatos o lenguajes con estructura anidada.

La idea fuerte, que vale más allá de Scilab: las expresiones regulares sirven para piezas simples; el analizador sirve cuando hay estructura anidada, jerarquía o reglas gramaticales.

5.4.8. Qué no tiene Scilab

Conviene cerrar con el mapa de equivalencias y de ausencias, porque ordena todo lo anterior y sirve para quien venga de otro lenguaje.

Lo que se puede emular:

Idea	En Scilab
tabla asociativa	`struct`, con campo dinámico
arreglo heterogéneo	`list`
objeto simple	`struct`, `tlist`, `mlist`
estado persistente	`global`, `struct` o estado explícito
método	función que recibe y devuelve la estructura
sobrecarga	`mlist`
expresión regular	`regexp`
evaluación dinámica	`evstr`, `execstr`

Lo que no tiene de forma natural: clausuras reales con captura del entorno léxico, referencias generales, literales cómodos para tablas asociativas, autovivificación de estructuras anidadas, contexto escalar y de lista, y metaprogramación flexible.

Ninguna de esas ausencias es un defecto: Scilab está pensado para cálculo numérico, y ahí las matrices, el álgebra lineal y los métodos numéricos son nativos y cómodos, que es exactamente lo contrario de lo que ocurre en los lenguajes donde estos mecanismos de texto sí son nativos Wall, Christiansen y Orwant 2000b. Cada herramienta resuelve bien el problema para el que fue hecha.

5.5. Panorama de lenguajes

Las partes anteriores tratan cada lenguaje por separado. Ésta los pone uno al lado del otro, que es la pregunta que aparece naturalmente apenas se conoce más de uno: cuál usar, y por qué.

5.5.1. Criterios para clasificar un lenguaje

Los lenguajes se clasifican según varios criterios, y conviene tenerlos separados porque son independientes entre sí: hay lenguajes de alto nivel compilados y de bajo nivel interpretados, de tipado estático y débil, y así con todas las combinaciones.

Nivel de abstracción

Los lenguajes de **bajo nivel** están cercanos al hardware: el lenguaje ensamblador usa mnemónicos para representar instrucciones del procesador directamente. Son difíciles de programar pero permiten control total sobre el hardware.

Los lenguajes de **alto nivel** abstraen los detalles del hardware y se acercan al lenguaje humano o matemático. Facilitan la programación a costa de cierto control sobre los detalles de ejecución.

Conviene notar que «alto» y «bajo» son relativos y se corrieron con el tiempo: C se consideraba de alto nivel frente al ensamblador, y hoy se lo ubica del lado bajo frente a los lenguajes que traen recolección de basura y estructuras de datos incorporadas Kernighan y Ritchie 1978. Lo que no cambia es el compromiso: cuanto más se abstrae, menos control queda sobre lo que hace la máquina, y a la inversa.

Traducción

Compilados frente a interpretados, con las consecuencias que ya vimos en la teoría: velocidad de ejecución y detección temprana de errores de un lado, ciclo interactivo y portabilidad del otro. Y el recordatorio de que no es una propiedad del lenguaje sino de la implementación.

Tipado

Estático frente a dinámico, según cuándo se verifican los tipos; y fuerte frente a débil, según si el lenguaje convierte por su cuenta entre tipos incompatibles. Las dos clasificaciones son independientes.

Propósito

De propósito general frente a especializados. Un entorno de cálculo científico trae las matrices y las funciones matemáticas incorporadas al lenguaje, lo que lo hace inmejorable para el cálculo y limitado fuera de él. Un lenguaje de propósito general no trae nada de eso pero sirve para todo lo demás, y consigue lo primero con bibliotecas.

5.5.2. Comparación

En la actualidad ningún científico utiliza un único lenguaje de programación. Es habitual combinar diferentes herramientas según el problema a resolver. Por ejemplo, un investigador puede desarrollar una simulación de alto rendimiento en Fortran o C Kernighan y Ritchie 1978, automatizar su ejecución mediante guiones del intérprete de comandos Ramey y Fox 2025, procesar los resultados con Python Downey 2015 y realizar cálculos exploratorios o prototipos numéricos en Scilab Scilab Enterprises 2026c.

Esa división del trabajo no es un accidente ni una comodidad transitoria: es la aplicación del principio de componer herramientas que hagan bien una cosa, que ya vimos al tratar el sistema operativo Ritchie y Thompson 1974; Raymond 2003.

En esta asignatura se comienza con Scilab porque constituye un excelente entorno para aprender programación aplicada al cálculo científico. Su sintaxis es sencilla y muy cercana al lenguaje matemático, lo que permite concentrarse en los conceptos fundamentales de algoritmos, funciones, vectores, matrices y métodos numéricos. Una vez adquiridas estas habilidades, el aprendizaje de otros lenguajes, especialmente Python, resulta considerablemente más sencillo.

Por ello, se recomienda que todo estudiante de Ciencias Físicas adquiera, a lo largo de su formación, conocimientos de Scilab, Python y al menos un lenguaje compilado como C o Fortran. La combinación de estas herramientas proporciona una base sólida para el trabajo académico, la investigación y el desarrollo profesional.

Vale una advertencia sobre las modas. Hace treinta años el lenguaje dominante en cálculo científico era otro, y dentro de diez probablemente lo sea uno que hoy no existe. Lo que permanece son los algoritmos, la forma de plantear un problema y los criterios de calidad del software; el lenguaje es el vehículo Saravia Alía 2005.

5.5.3. Un recorrido por varios lenguajes

Las partes por lenguaje de esta unidad describen qué es cada uno hoy y para qué sirve. Lo que sigue es de dónde salió cada uno, que ayuda a entender por qué son como son.

Fortran

Fortran (*Formula Translation*, 1957) fue el primer lenguaje de alto nivel de uso generalizado. Diseñado por IBM para cálculo científico, introdujo conceptos que hoy damos por sentados: variables, expresiones aritméticas, subrutinas, entrada/salida formateada. Sigue siendo relevante en computación de alto rendimiento: grandes códigos de simulación climática, aerodinámica y física de partículas están escritos en Fortran moderno (Fortran 90/95/2003/2018), que añadió arreglos dinámicos, módulos, programación orientada a objetos y paralelismo.

C

C (1972, Ritchie en Bell Labs) fue diseñado para implementar el sistema operativo Unix. Es un lenguaje de nivel medio: más cercano al hardware que Python pero más abstracto que el ensamblador. Permite manipulación directa de memoria mediante punteros, control preciso del tamaño de las variables y acceso directo al hardware. Es el lenguaje más influyente de la historia: la mayoría de los sistemas operativos, intérpretes de Python, máquinas virtuales de Java y bases de datos están implementados en C.

Python

Python (1991, van Rossum) es un lenguaje de alto nivel, multiparadigma, de tipado dinámico y fuerte, con sintaxis clara y minimalista que favorece la legibilidad. Tiene un ecosistema extraordinario para ciencia de datos e IA: NumPy (arreglos numéricos), SciPy (algoritmos científicos), Matplotlib (gráficos), Pandas (datos tabulares), TensorFlow y PyTorch (aprendizaje profundo). Es el lenguaje dominante en ciencia de datos, aprendizaje automático, scripting y automatización.

Perl

Perl (1987, Larry Wall) fue diseñado para procesamiento de texto y administración de sistemas. Introdujo características poderosas de manejo de cadenas y expresiones regulares. Su lema "hay más de una forma de hacerlo" contrasta con el "solo debería haber una forma obvia" de Python. Perl introdujo el concepto de variables dinámicas modernas: escalares (`$`), arreglos (`@`) y tablas hash (`%`) que pueden contener cualquier tipo de dato y redimensionarse automáticamente.

Bash

Bash (*Bourne Again Shell*) es el intérprete de comandos estándar en GNU/Linux. Es simultáneamente un lenguaje de programación completo. Los scripts de bash permiten

automatizar tareas del sistema, encadenar comandos, procesar archivos de texto y gestionar procesos. Su poder radica en la composición de comandos mediante tuberías y en el acceso directo a todas las herramientas del sistema.

JavaScript

JavaScript (1995, Brendan Eich) fue creado en diez días para añadir interactividad a las páginas web. Hoy es el único lenguaje que ejecutan nativamente todos los navegadores y, con Node.js, también puede correr en el servidor. Es el lenguaje más ejecutado del mundo. Soporta programación orientada a objetos basada en prototipos, funcional y orientada a eventos. Su ecosistema (npm) tiene millones de paquetes.

Scilab

Scilab (1990, INRIA) es un entorno de cálculo científico de código abierto, alternativa libre a MATLAB. Proporciona un lenguaje interpretado de alto nivel con soporte nativo para vectores y matrices, funciones matemáticas y estadísticas, integración numérica, resolución de EDOs, gráficos 2D y 3D y el simulador de sistemas dinámicos Xcos. Es especialmente adecuado para enseñanza de métodos numéricos e ingeniería.

LaTeX

LaTeX puede verse como un lenguaje de programación declarativo y de marcas para composición tipográfica. Tiene variables, macros (funciones), estructuras condicionales y bucles, aunque la mayoría de los usuarios no los utiliza explícitamente. El paquete `pgfplots` permite generar gráficos directamente desde LaTeX; el paquete `tikz` permite programar dibujos vectoriales complejos.

5.6. Programación en Python

Python es hoy el lenguaje más usado en cálculo científico y en análisis de datos. No lo tratamos como reemplazo de lo anterior sino como el segundo lenguaje: quien ya entiende qué es un bucle, qué es el alcance de una variable y qué distingue verificar de validar, aprende Python leyendo la referencia Python Software Foundation 2026b. Lo que sigue señala qué cambia respecto de lo que ya vimos.

5.6.1. Qué tipo de lenguaje es

Python es interpretado, de tipado dinámico y fuerte, y de propósito general. Las tres cosas importan.

Que sea **interpretado** le da el ciclo interactivo que ya conocemos, con la misma consecuencia: los errores aparecen recién al llegar la ejecución a la línea que los contiene. En rigor compila primero a un código intermedio que ejecuta una máquina virtual, que es el esquema híbrido que describimos en la teoría.

Que sea de tipado **dinámico** significa que las variables no se declaran; que sea **fuerte** significa que no convierte tipos por su cuenta: sumar un número y una cadena es un error, no una concatenación silenciosa. Ésa es una diferencia real con otros lenguajes dinámicos y evita una clase entera de errores.

Que sea de **propósito general** es la diferencia de fondo con un entorno de cálculo. Python no viene con las matrices ni con las funciones matemáticas incorporadas al lenguaje: se importan de bibliotecas. Eso lo hace más incómodo para el cálculo rápido y mucho más capaz para todo lo demás —leer archivos de un instrumento, hablar con una base de datos, servir una página web, controlar un dispositivo.

5.6.2. Lo que se escribe distinto

La sangría es sintaxis. No hay palabra que cierre un bloque: el bloque es lo que está indentado. Es la decisión más visible del lenguaje y hace imposible que la sangría mienta sobre la estructura, que es un error frecuente en los demás.

Los índices empiezan en cero y el rango superior es excluyente: `range(0, 5)` da cinco valores, de 0 a 4. Al venir de un entorno de cálculo, donde se empieza en uno y el rango incluye ambos extremos, es la fuente garantizada de los primeros errores.

La iteración es sobre los elementos, no sobre los índices. La forma idiomática recorre directamente la colección; si además hace falta la posición, se pide explícitamente. Escribir un bucle sobre índices para después acceder por índice es la marca de quien viene de otro lenguaje.

Las estructuras básicas son otras. Lista, tupla, diccionario y conjunto están en el lenguaje mismo, no en una biblioteca. El diccionario en particular —la tabla asociativa de la teoría— es tan central que buena parte del lenguaje está construida sobre él.

Las funciones son valores de pleno derecho. Se pueden definir dentro de otras, devolver, guardar en una lista. Las clausuras funcionan como se describieron en la segunda parte teórica, sin emulación: la función interna ve las variables de la externa. Y hay aplicación parcial en la biblioteca estándar, lista para el caso de fijar los parámetros de un modelo antes de pasárselo a un integrador.

5.6.3. Módulos y entorno

Cada archivo es un módulo, y un directorio con módulos es un paquete. La instrucción de importación es explícita: nada está disponible hasta que se lo pide. Eso hace más largo el encabezado de cada programa y elimina la duda de dónde salió cada nombre.

La contrapartida es que el programa depende de qué hay instalado. La práctica establecida es crear un **entorno virtual** por proyecto: un directorio con su propia copia de las bibliotecas, aislado del sistema y de los demás proyectos. Y registrar la lista de dependencias con sus versiones en un archivo que viaja con el código.

Esto no es burocracia: es la respuesta concreta al problema de reproducibilidad que planteamos en la teoría. Un cálculo publicado sin la lista de versiones no se puede repetir, porque las bibliotecas cambian y los resultados numéricos con ellas.

5.6.4. El ecosistema científico

Lo que en un entorno de cálculo viene incorporado, en Python se arma con cuatro piezas que conviene conocer por separado:

El arreglo n-dimensional Harris, Millman, Walt, Gommers, Virtanen, Cournapeau y Oliphant 2020, que aporta las operaciones vectorizadas. Vale exactamente lo dicho para el entorno de cálculo: la operación sobre el arreglo completo es más corta y mucho más rápida que el bucle, porque el trabajo lo hace código compilado. Escribir bucles sobre arreglos grandes es el error de rendimiento más común.

Los algoritmos numéricos Virtanen, Gommers, Oliphant, Haberland, Reddy y Cournapeau 2020: integración, optimización, ajuste de curvas, álgebra lineal, ecuaciones diferenciales. Es el equivalente de las funciones que en el entorno de cálculo son parte del lenguaje.

Los gráficos Hunter 2007a, con el mismo modelo de objetos con propiedades modificables que vimos en la unidad de gráficos.

Las tablas McKinney 2022, para datos con columnas de tipos distintos, series temporales y valores faltantes, que es lo que el arreglo numérico homogéneo no cubre y aparece apenas se trabaja con datos experimentales reales.

A eso se suma el cuaderno interactivo, que ya tratamos en su unidad Pérez y Granger 2007b.

5.6.5. Buenas prácticas

Python tiene la particularidad de haber convertido las convenciones de escritura en documentos públicos y aceptados por toda la comunidad Rossum, Warsaw y Coghlan 2001, e incluso de haber puesto por escrito sus principios de diseño Peters 2004. Conviene leerlos como caso de estudio de la sección de legibilidad de la teoría: prácticamente todo lo que ahí figura como recomendación general, acá está como norma con número.

Lo demás se traslada sin cambios, con su realización concreta:

- La documentación de una función va en una cadena al principio de su cuerpo, y es lo que devuelve la ayuda interactiva.
- Las pruebas se escriben con las herramientas de la biblioteca estándar o con un marco de pruebas, y se corren automáticamente.
- Los verificadores estáticos revisan estilo y errores probables sin ejecutar el programa.

- Las anotaciones de tipo son opcionales y no las verifica el intérprete, pero permiten que una herramienta externa detecte antes los errores de tipo. Es una forma de recuperar parte de la ventaja del tipado estático sin perder la comodidad del dinámico.

Para el nivel intermedio —modelo de datos, iteradores, decoradores, clausuras tal como el lenguaje las implementa— la referencia es Ramalho 2022; para el primer contacto orientado a resolver problemas, Downey 2015.

5.7. Programación en Perl

Perl entra en la materia por una razón precisa: es el lenguaje donde los temas de la segunda parte teórica están en el centro del diseño y no agregados después. Las expresiones regulares son parte de la sintaxis, las tablas asociativas son un tipo básico, y el procesamiento de texto es el caso de uso para el que fue pensado The Perl Foundation 2026; Christiansen et al. 2012.

En cálculo científico su lugar es acotado y real: cuando hay que tomar la salida de un instrumento o de otro programa, extraer los datos, reordenarlos y dejarlos listos para el análisis, Perl resuelve en diez líneas lo que en otro lenguaje son cincuenta.

5.7.1. Qué tipo de lenguaje es

Interpretado, de tipado dinámico y débil, y con una filosofía de diseño explícitamente opuesta a la de Python: en lugar de que haya una única forma evidente de hacer cada cosa, se acepta que haya muchas. Eso lo hace expresivo y hace que dos programas correctos puedan parecerse muy poco entre sí, lo cual tiene consecuencias sobre la legibilidad que conviene tener presentes.

Que el tipado sea **débil** conecta directamente con algo que planteamos en la teoría. La cadena `"12"` y el número `12` se convierten uno en otro según haga falta, y lo que decide cómo se interpreta el dato no es su tipo sino el operador: hay un operador de suma y otro distinto de concatenación, y cada uno lee sus operandos como corresponde. Ahí está el ejemplo concreto de la decisión de diseño que discutimos en abstracto Wall, Christiansen y Orwant 2000b.

5.7.2. Contexto

Hay un concepto que no tiene equivalente en los otros lenguajes de la unidad y que hay que entender antes que nada: la misma expresión puede significar cosas distintas según dónde se la use. En **contexto escalar** un arreglo devuelve su cantidad de elementos; en **contexto de lista** devuelve sus elementos. No es una excentricidad: es lo que permite que muchas construcciones se escriban sin decir explícitamente qué se quiere.

También es una fuente de errores para quien viene de otro lenguaje, y la primera cosa a revisar cuando un programa en Perl hace algo inesperado.

5.7.3. Estructuras de datos

Los tres tipos básicos se distinguen por el signo que precede al nombre: escalar para un valor único —número, cadena o referencia—, arreglo para una lista ordenada con índice numérico, y tabla asociativa para pares de clave y valor.

Ésta última es el diccionario de la teoría, nativo y con sintaxis propia. Contar ocurrencias, agrupar por categoría o indexar registros por identificador son operaciones de una línea.

Las estructuras anidadas —listas de listas, tablas de listas, tablas de tablas— se arman con **referencias**, que es el otro concepto de la teoría que acá es nativo: un escalar puede contener una referencia a otra estructura, y así se construye cualquier forma arbitraria. La **autovivificación** completa el cuadro: al escribir en un nivel que no existe, se crean automáticamente los niveles intermedios. Es cómodo y es una trampa, porque también crea la estructura cuando el nombre estaba mal escrito.

5.7.4. Expresiones regulares

Es lo que hay que aprender de este lenguaje aunque no se lo use después. El patrón no es una cadena que se pasa a una función: es sintaxis, con operadores propios para buscar y para sustituir, con modificadores para ignorar mayúsculas, aplicar globalmente o permitir comentarios dentro del patrón. Todo lo dicho en la segunda parte teórica —clases, cuantificadores, anclas, agrupamiento, voracidad— está acá en su forma canónica, y las partes capturadas quedan disponibles en variables numeradas.

La razón para tratarlo en detalle es que este dialecto se volvió el de referencia de hecho: la mayoría de los lenguajes modernos implementan una variante compatible, de modo que lo que se aprenda acá se traslada casi sin cambios Friedl 2006. Vale también el límite que señalamos: por más extensiones que tenga el dialecto, un formato jerárquico pide un analizador, no un patrón Aho et al. 2006.

5.7.5. Lo que la teoría llamó avanzado

Varias cosas que en otros lenguajes hay que emular, acá vienen de fábrica:

Clausuras. Una función anónima definida dentro de otra captura las variables de su entorno. El contador con estado privado se escribe directamente, sin variables globales.

Evaluación dinámica. Existe la operación que ejecuta una cadena como código, con las tres advertencias que ya dimos. Tiene además una segunda forma, que ejecuta un bloque atrapando los errores, y que es el mecanismo de manejo de excepciones del lenguaje.

Funciones de orden superior. Aplicar una función a cada elemento, filtrar los que cumplen una condición y ordenar con un criterio propio son operaciones del lenguaje que reciben un bloque de código como argumento.

Tipos propios y sobrecarga. Se puede asociar una estructura a un espacio de nombres y definir qué hacen los operadores sobre ella, que es la sobrecarga descrita en la teoría.

5.7.6. Buenas prácticas

Acá hacen más falta que en otros lenguajes, precisamente por la libertad sintáctica. Dos hábitos que no son opcionales: activar desde la primera línea los modos que exigen declarar las variables y que avisan de construcciones dudosas —sin eso, un nombre mal escrito crea una variable nueva en silencio— y evitar las variables especiales implícitas en el código que otro va a leer, por más que acorten la escritura.

Lo demás se traslada: nombres descriptivos, funciones cortas, la documentación integrada en el propio archivo, y las pruebas automáticas, para las que el lenguaje tiene un formato de salida estandarizado.

Para empezar, la introducción por problemas es Schwartz, b. d. f. b. d. y Phoenix 2021; la referencia completa está en la documentación del lenguaje The Perl Foundation 2026, cuyas secciones sobre expresiones regulares y sobre estructuras de datos anidadas son las directamente pertinentes a esta unidad.

5.7.7. Cuándo conviene y cuándo no

Conviene para procesar texto, transformar formatos, extraer datos de salidas irregulares y pegar programas entre sí; es decir, para el trabajo que rodea al cálculo. No conviene para el cálculo numérico pesado, para el que no está pensado, ni para programas grandes que mucha gente vaya a mantener, donde la libertad sintáctica se paga.

Como en todo lo demás de esta unidad, el criterio no es cuál lenguaje es mejor sino cuál corresponde a la tarea Raymond 2003.

5.8. Programación en el intérprete de comandos: bash

El intérprete de comandos es el caso más claro de lo que planteamos al principio de la unidad: la diferencia entre un comando y un programa es sólo que el segundo está guardado en un archivo. Todo lo que se escribe a mano en la terminal es un programa de una línea, y todo guión es una secuencia de esas líneas Ramey y Fox 2025.

Por eso conviene aprenderlo aunque el cálculo se haga en otro lado. El guión de shell es lo que junta las piezas: prepara los datos, corre la simulación con distintos parámetros, guarda los resultados, genera las figuras y arma el informe.

5.8.1. Qué tipo de lenguaje es

Interpretado, de tipado prácticamente inexistente —todo es texto— y diseñado no para calcular sino para **coordinar programas**. Ésa es la diferencia de fondo con los otros tres lenguajes de la unidad: acá las operaciones primitivas no son sumar y multiplicar sino ejecutar un programa, conectar su salida con la entrada de otro y decidir según cómo terminó.

De esa vocación se siguen sus rasgos más característicos:

La tubería. La salida de un programa se conecta a la entrada del siguiente sin archivos intermedios. Es composición de funciones, en el sentido que le dimos en la teoría, sólo que las funciones son programas y el valor que circula es texto.

El código de salida. Todo programa devuelve un número al terminar: cero si funcionó, otra cosa si no. Ese número es lo que evalúan las estructuras condicionales, y por eso un guión puede reaccionar a que algo salió mal sin leer la salida.

El texto plano como interfaz universal. No hay tipos ni estructuras: lo que pasa de un programa a otro son líneas de texto. Es lo que hace que programas escritos por gente distinta en épocas distintas se puedan combinar Ritchie y Thompson 1974; Raymond 2003.

La expansión de nombres de archivo. Los patrones se resuelven antes de ejecutar el comando, que recibe ya la lista de nombres. Es un mecanismo del intérprete, no de cada programa, y explica por qué un archivo con espacios en el nombre rompe tantos guiones.

5.8.2. Lo mismo, escrito distinto

Las estructuras de control de la teoría están todas, con dos particularidades:

La condición es un programa. Lo que la estructura condicional evalúa no es una expresión booleana sino el código de salida de un comando. La comparación numérica y la de cadenas se hacen con un programa dedicado a eso, y por eso los espacios alrededor de los corchetes no son opcionales: son la separación entre el nombre del programa y sus argumentos.

El bucle recorre palabras. La iteración natural es sobre una lista de elementos —nombres de archivo, valores de un parámetro, líneas de un archivo— y no sobre un rango numérico, aunque también se puede.

Las funciones existen, agrupan comandos bajo un nombre y reciben argumentos por posición. Devuelven un código de salida, no un valor; cuando hace falta devolver un dato, se lo escribe en la salida estándar y quien llama lo captura.

Las variables son de texto y son globales por omisión, con una palabra reservada para hacerlas locales a una función. Vale acá lo dicho en la teoría, agravado: en un lenguaje sin declaraciones, una variable global modificada por descuido es difícil de rastrear.

5.8.3. El problema de las comillas

Hay un tema que merece sección propia porque es la causa de la mayoría de los errores, incluidos algunos con consecuencias serias.

El intérprete, antes de ejecutar un comando, expande las variables y después parte el resultado en palabras usando los espacios. Si una variable contiene un nombre de archivo con un espacio, el comando recibe dos argumentos en lugar de uno. Si contiene un patrón,

se expande contra los archivos del directorio. Poner la variable entre comillas dobles suspende esa partición y esa expansión.

La regla es simple y no admite excepciones prácticas: toda expansión de variable va entre comillas, salvo que se quiera explícitamente el efecto contrario. Un guión que funciona con archivos de nombres simples y falla con los reales casi siempre está fallando por esto.

Existe una herramienta que detecta éste y los demás errores clásicos sin ejecutar el guión Holen 2026, y correrla sobre todo lo que se escriba es la práctica más rentable de esta parte.

5.8.4. El entorno de ejecución

Ésta es la parte donde el shell realiza lo que en la segunda parte teórica quedó planteado en abstracto:

Procesos por lotes. El guión que corre sin nadie delante recibe sus datos por argumentos de línea de comandos, informa el avance por la salida de errores y termina con un código de salida que indica si funcionó. Conviene además hacer que el guión se detenga ante el primer error en lugar de seguir con datos incompletos, que es lo que hace por omisión.

Barridos de parámetros. Correr la misma simulación con veinte valores distintos de un coeficiente, guardar cada resultado en un archivo con nombre sistemático y juntar todo al final es el uso más frecuente en cálculo científico, y es un bucle de tres líneas.

Automatización de la reconstrucción. Cuando la cadena de pasos crece y no todo hay que rehacerlo siempre, el guión lineal se queda corto y corresponde declarar las dependencias S. I. Feldman 1979.

Trabajos programados y máquinas remotas. El guión es la unidad que se ejecuta a una hora determinada o en un servidor sin pantalla, que es el escenario normal del cálculo pesado.

5.8.5. Buenas prácticas y cuándo dejar el shell

Las de la teoría, con sus formas concretas: nombres descriptivos, funciones cortas, comentar el porqué, y una cabecera que declare qué hace el guión y qué espera recibir. Además, las específicas: activar los modos que abortan ante error y ante variable no definida, entrecomillar toda expansión, y usar la herramienta de análisis estático antes de confiar en el resultado Newham y Rosenblatt 2005; Cooper 2014.

Y una decisión que conviene tomar temprano: el shell es excelente para coordinar y pésimo para calcular. En cuanto aparecen aritmética con decimales, estructuras de datos, manipulación fina de texto o más de un centenar de líneas, corresponde pasar a otro lenguaje Google 2024. Reconocer ese punto es parte de saber usar la herramienta; insistir con el shell más allá de él produce guiones que nadie puede mantener, incluido quien los escribió.

Vale por último recordar que la portabilidad tiene grados. Hay un conjunto de comportamientos normalizados que funcionan en cualquier sistema The IEEE y The Open Group 2024, y hay extensiones cómodas que existen sólo en algunos intérpretes. Si el guión va a correr en una sola máquina, dan lo mismo; si va a distribuirse, conviene saber cuál se está usando.

5.9. Programación en C

Los cuatro lenguajes anteriores son interpretados y de tipado dinámico. C es lo contrario en ambas cosas, y por eso su lugar en la materia no es el de una herramienta más sino el de un cambio de nivel: es donde se ven las cosas que los otros esconden. Quien entendió qué es la dirección de una variable, el tiempo de vida o el pasaje por referencia leyendo la teoría, acá los ve funcionando.

En cálculo científico se usa cuando el tiempo de cómputo importa, y como lenguaje en el que están escritas las bibliotecas numéricas que los demás lenguajes llaman por debajo. La referencia clásica y breve es Kernighan y Ritchie 1978; para un curso completo sobre la norma vigente, King 2008; International Organization for Standardization 2018.

5.9.1. Qué tipo de lenguaje es

Compilado, de tipado estático y débil, y de nivel comparativamente bajo: cercano a la máquina, sin recolector de basura y con acceso directo a la memoria.

Que sea **compilado** implica el ciclo de trabajo que describimos en la teoría —editar, compilar, enlazar, ejecutar— y su contrapartida: una clase entera de errores se detecta antes de correr nada, y el resultado se ejecuta sin intérprete y a la velocidad del procesador.

Que el tipado sea **estático** obliga a declarar cada variable con su tipo antes de usarla. Eso, que al principio parece burocracia, es lo que permite al compilador saber cuánta memoria ocupa cada dato y generar código eficiente, y lo que hace que un error de tipos aparezca al compilar y no después de una hora de cálculo.

Que sea **débil** significa que las conversiones entre tipos numéricos ocurren solas, y ahí está una de sus trampas: dividir dos enteros da un entero, de modo que `1/2` vale cero y no un medio.

5.9.2. Lo que acá se ve y en los otros no

Memoria

La teoría distinguió variables automáticas, estáticas y dinámicas. En C esa distinción es explícita y tiene consecuencias visibles.

Las automáticas viven en la pila y desaparecen al salir del bloque. Las estáticas persisten toda la ejecución. Las dinámicas se reservan y se liberan con llamadas explícitas, y ahí aparecen los dos errores clásicos que mencionamos en abstracto y que acá son concretos

y frecuentes: no liberar lo reservado, con lo que el programa consume memoria indefinidamente, y usar un puntero después de liberarlo o fuera de los límites reservados, con lo que el programa lee o escribe donde no debe.

Ese segundo caso merece atención porque es donde C se separa radicalmente de los otros lenguajes. Escribir fuera de un arreglo no produce un mensaje de error: produce **comportamiento indefinido**, una categoría que la norma define explícitamente International Organization for Standardization 2018 y que significa que el programa puede hacer cualquier cosa. Puede fallar, puede seguir andando con datos corrompidos, o puede funcionar perfecto hasta que se lo compila con otras opciones. Los resultados plausibles pero incorrectos, que en la teoría señalamos como el peligro central del software científico, acá tienen esta causa concreta.

Punteros

Un puntero es una variable que contiene una dirección. Es el concepto que la teoría llamó referencia, sin intermediarios. Sirve para tres cosas: pasar datos grandes a una función sin copiarlos, permitir que la función modifique lo que recibió —el pasaje por referencia del que hablamos, que en C se hace pasando la dirección— y construir estructuras enlazadas.

La aritmética de punteros y la indexación de arreglos son la misma operación escrita distinto, lo cual es elegante y explica por qué el lenguaje no verifica los límites: para el compilador, salirse del arreglo es simplemente sumar de más.

Números reales

Todos los lenguajes de la unidad representan los reales de la misma manera IEEE 2019, pero es acá donde se elige explícitamente la precisión y donde las consecuencias quedan expuestas. Es un buen lugar para tratar en serio lo que en la teoría dijimos de paso: por qué comparar reales por igualdad es un error, qué es la cancelación catastrófica al restar dos números parecidos, y por qué el orden de las operaciones cambia el resultado Goldberg 1991b. Nada de esto es propio de C, pero en C no queda tapado por ninguna comodidad del lenguaje.

5.9.3. Lo que hay que emular

Varios temas de la segunda parte teórica no existen acá de forma nativa:

No hay clausuras. Se emulan con un puntero a función más una estructura que lleve el estado, que se pasa como argumento adicional. Es exactamente lo que hacen las bibliotecas numéricas cuando reciben una función a integrar junto con un puntero a los parámetros del modelo: eso es aplicación parcial hecha a mano.

No hay evaluación dinámica. El programa está compilado; no hay intérprete disponible en tiempo de ejecución.

No hay expresiones regulares en el lenguaje , aunque sí en bibliotecas normalizadas del sistema.

No hay tablas asociativas ni listas heterogéneas. Hay que implementarlas o usar una biblioteca. Es un ejercicio instructivo: escribir una tabla asociativa una vez hace entender por qué en los otros lenguajes buscar por clave es rápido.

No hay sobrecarga de operadores. Las operaciones sobre tipos propios se escriben como llamadas a función.

Lo que sí hay son **estructuras**, que son los registros de la teoría, y punteros a función, que permiten pasar comportamiento como argumento. Con esas dos piezas se construyen los demás mecanismos.

5.9.4. Compilación separada y construcción

Un programa en C se reparte en varios archivos: los que contienen el código y los encabezados que declaran las interfaces. Esa división es la modularidad de la teoría en su forma más explícita, con la interfaz literalmente en un archivo aparte de la implementación.

Cada archivo se compila por separado y después se enlazan todos, con la distinción entre enlace estático y dinámico que ya vimos. Cuando los archivos son varios, rehacer a mano lo que corresponde es lento y propenso a olvidos, y ahí entra la herramienta que declara dependencias y decide qué recompilar S. I. Feldman 1979; Stallman, McGrath y P. D. Smith 2023.

5.9.5. Buenas prácticas

Las de la teoría, más algunas que en C no son opcionales:

- Compilar con todas las advertencias activadas y tratarlas como errores. El compilador detecta gratis buena parte de los problemas.
- Verificar el valor devuelto por toda función que pueda fallar, empezando por las de reserva de memoria y las de apertura de archivos.
- Usar herramientas de análisis de memoria para detectar fugas y accesos fuera de límites, que no se manifiestan solos.
- Inicializar las variables. Una variable automática no inicializada contiene basura, no cero.

Sobre el estilo y sobre cómo se decide qué escribir en C y qué no, Kernighan y Pike 1999 es la referencia breve.

5.9.6. Cuándo conviene

Cuando el tiempo de cómputo domina y el problema ya está entendido, o cuando hay que hablar directamente con el hardware. No conviene para explorar, para prototipar ni

para procesar texto: escribir en C lo que se resuelve en diez líneas de otro lenguaje cuesta mucho más de lo que ahorra.

La estrategia habitual en cálculo científico es mixta, y conviene enunciarla porque es lo que se hace en la práctica: prototipar en un lenguaje interpretado, medir dónde está el tiempo, y reescribir en C sólo esa parte, dejando que el lenguaje de alto nivel siga coordinando. Optimizar antes de medir es la forma más común de perder tiempo.

5.10. Programación en JavaScript

JavaScript entra en la materia por una razón que ninguno de los otros lenguajes cumple: es el único que se ejecuta dentro del navegador, y por lo tanto el único con el que se puede hacer que una página muestre un resultado que reaccione a lo que hace quien la mira. Todo lo que en la unidad de interfaces gráficas se hace del lado del cliente pasa por acá.

El nombre corriente designa a una implementación de un lenguaje normalizado Ecma International 2025; la referencia práctica de uso diario es la documentación del navegador Network 2026, y el tratamiento completo está en Flanagan 2020.

5.10.1. Qué tipo de lenguaje es

Interpretado, de tipado dinámico y débil, y con un modelo de ejecución propio que es lo que más lo distingue: un solo hilo con bucle de eventos.

Que el tipado sea **débil** lo emparenta con Perl y lo separa de Python: las conversiones ocurren solas y con reglas que no siempre son las esperadas. Es el origen de la mayoría de sus rarezas y la razón por la cual conviene usar siempre la comparación estricta, que no convierte, en lugar de la que sí lo hace.

Que sea de **un solo hilo** tiene una consecuencia que ya discutimos en la unidad de interfaces: nada corre en paralelo, así que una función que tarda bloquea todo, incluida la respuesta de la página. De ahí que casi todas las operaciones que pueden demorar —leer un archivo, pedir datos por la red— sean asincrónicas por diseño.

5.10.2. Eventos y asincronía

Éste es el aporte propio del lenguaje a la unidad, porque es donde el modelo de eventos deja de ser un tema de las interfaces gráficas y pasa a ser la forma normal de escribir cualquier programa.

El esquema es el que ya vimos: el programa registra funciones manejadoras y el bucle de eventos las llama cuando corresponde. Lo que cambia acá es que también se usa para lo que en otros lenguajes es una operación común y corriente. Pedir datos a un servidor no devuelve los datos: devuelve la promesa de que llegarán, y el código que los usa se escribe como continuación.

Las tres formas históricas de escribir eso —función de retorno anidada, encadenamiento de promesas, y la notación que permite escribirlo con apariencia secuencial— conviene

conocerlas todas, porque el código existente las mezcla. La última es la que hay que usar al escribir código nuevo.

La consecuencia práctica para cálculo: un bucle numérico pesado congela la interfaz, exactamente como el callback largo de la unidad de interfaces gráficas. Si hay que calcular en serio dentro del navegador, corresponde hacerlo en un hilo aparte.

5.10.3. Clausuras

En ningún otro lenguaje de la unidad las clausuras son tan centrales. Cada función manejadora que se registra es, típicamente, una clausura: captura las variables del entorno donde se la definió y las sigue viendo cuando el evento ocurre, mucho después.

Ése es precisamente el mecanismo que permite que un manejador sepa sobre qué elemento tiene que actuar sin recibir nada por argumento. Y es también el origen de un error clásico: registrar manejadores dentro de un bucle y que todos terminen viendo el último valor de la variable del ciclo, porque capturaron la variable y no su valor. Es un buen ejemplo para discutir qué es exactamente lo que una clausura captura.

La aplicación parcial y las funciones de orden superior están en el lenguaje: aplicar una función a cada elemento de una colección, filtrar y reducir son métodos de los arreglos, y se usan constantemente.

5.10.4. Estructuras de datos

El objeto es a la vez el registro y la tabla asociativa de la teoría: un conjunto de pares de clave y valor, con sintaxis para ambos usos. Los arreglos son heterogéneos y de tamaño variable. Y hay un formato de intercambio derivado directamente de esa sintaxis Bray 2017, que se volvió el estándar de facto para transportar datos estructurados entre programas, mucho más allá del navegador.

Eso es lo que hace que este lenguaje sea relevante aun para quien no haga páginas web: el formato de intercambio con el que hoy conversan los instrumentos, los servicios y los programas salió de acá.

Los números merecen una advertencia: hay un solo tipo numérico, en doble precisión, y por lo tanto no hay enteros exactos más allá de cierto tamaño. Todo lo dicho sobre representación de reales Goldberg 1991b vale acá sin la posibilidad de elegir otro tipo.

5.10.5. Dentro y fuera del navegador

Dentro del navegador, el lenguaje tiene acceso al documento y puede modificarlo: cambiar el contenido, crear elementos, responder a eventos y dibujar. La conexión con la unidad de gráficos es directa, porque el formato vectorial que allí vimos se puede generar y animar desde acá World Wide Web Consortium 2026.

Fuera del navegador, el mismo lenguaje corre como programa del sistema con acceso a archivos, procesos y red OpenJS Foundation 2026a. Eso lo convierte en un lenguaje de propósito general y explica por qué aparece en tantos lugares donde no hay ninguna página

web involucrada.

Vale la aclaración porque es fuente de confusión: el lenguaje es el mismo, lo que cambia es qué funciones hay disponibles. Un programa escrito para el navegador no corre fuera de él, y a la inversa, no por el lenguaje sino por el entorno.

5.10.6. Buenas prácticas

El lenguaje arrastra decisiones tempranas que por compatibilidad no se pueden revertir, de modo que conviene usar deliberadamente un subconjunto y evitar el resto Crockford 2008b. Es un buen caso de estudio de algo que la teoría plantea en general: un lenguaje muy usado no puede corregir sus errores sin romper lo ya escrito.

En concreto: declarar las variables con las formas de alcance de bloque y no con la antigua; usar comparación estricta; activar el modo que convierte en error una serie de construcciones dudosas; y correr un verificador estático, que acá detecta bastante más que en otros lenguajes por lo permisivo que es el intérprete.

Lo demás se traslada sin cambios, con la advertencia habitual sobre las dependencias: el ecosistema tiende a acumular una cantidad enorme de bibliotecas pequeñas, y cada una es una dependencia más que hay que registrar si el resultado tiene que ser reproducible.

5.10.7. Cuándo conviene

Cuando el resultado tiene que verse y usarse en un navegador, sin instalar nada. Para eso no hay alternativa, y es una ventaja real: un cálculo publicado como página web lo puede ejecutar cualquiera desde cualquier máquina.

No conviene para el cálculo numérico pesado ni para el procesamiento de archivos grandes, donde los otros lenguajes de la unidad están mejor equipados. El reparto habitual es el mismo de siempre: el cálculo donde corresponde, y este lenguaje para la parte que el usuario toca.

5.11. Ejercicios

5.11.1. Criterio general para la resolución.

Los ejercicios de esta guía tienen como objetivo practicar programación en el lenguaje elegido. Las consignas se formulan en términos de problemas, algoritmos y estructuras de programación, sin depender de la sintaxis de un lenguaje particular.

Cuando exista una función que resuelva directamente una tarea, deberá emplearse, salvo que la consigna indique lo contrario.

Mas adelante realizaremos algunos ejercicios en lenguajes compilados.

5.11.2. El entorno interactivo o REPL

La sigla `REPL` proviene del inglés *Read-Eval-Print Loop*, es decir, *bucle de lectura, evaluación e impresión*. Es el modo interactivo de un intérprete, donde se escribe una instrucción, se presiona la tecla de entrada y el entorno:

1. *lee* lo que se escribió;
2. *evalúa* la instrucción;
3. *imprime* el resultado, si lo hay;
4. repite el ciclo esperando la siguiente línea.

En español se suele llamar *consola interactiva*, *intérprete interactivo* o, informalmente, *terminal interactiva*. Sin embargo, conviene distinguir:

- la *terminal del sistema operativo* es el programa general (bash, cmd, PowerShell) desde el cual se ejecutan comandos como `node programa.js` o `python3 programa.py`;
- el *REPL* es el modo interactivo específico de cada lenguaje (Node.js, Python, Octave, Scilab), que se inicia ejecutando el intérprete sin indicarle un archivo.

En esta guía, cuando se mencione el "REPL de Node.js" o el "entorno interactivo" de otro lenguaje, se hace referencia a este segundo modo: el prompt del intérprete donde se pueden probar instrucciones de una en una.

5.11.3. Programas, y estructuras de control

1. **Ejercicio:** REPL y archivos de comandos

   Abra el entorno interactivo del lenguaje elegido y realice, una por una, las siguientes operaciones, observando el resultado de cada paso, como hizo en el práctico anterior:

   - asigne el valor entero 5 a una variable;
   - asigne el valor real 3.14 a otra variable;
   - asigne la cadena "hola" a una tercera variable;
   - calcule la suma de las dos variables numéricas;
   - muestre el contenido de la cadena;
   - genere 100 valores equiespaciados entre 0 y 2π.

   Luego guarde una secuencia equivalente de instrucciones en un archivo de programa y ejecútelo desde el entorno de trabajo correspondiente.

   Responda:

   1. ¿Hay alguna diferencia entre ejecutar las instrucciones interactivamente y ejecutarlas desde un archivo?

2. Determine cómo se controla, en el lenguaje elegido, la visualización del resultado de una instrucción. Observe especialmente qué ocurre al escribir una expresión directamente en el entorno interactivo y qué ocurre con la misma expresión dentro de un archivo. No estamos estudiando todavía entrada y salida, pero usaremos un comando básico para mostrar explícitamente el valor de una variable.
3. Modifique intencionalmente el cálculo de la suma para hacer referencia a una variable o nombre inexistente. Compare la información de diagnóstico obtenida al producir el error interactivamente y al ejecutar el archivo.
4. Agregue comentarios al archivo utilizando la sintaxis correspondiente al lenguaje elegido.
5. Ejecute nuevamente el archivo desde una sesión limpia. Use el comando apropiado para limpiar el espacio de trabajo cuando exista; en caso contrario, cierre y vuelva a abrir la sesión.

Scilab. En Scilab, un archivo de comandos suele utilizar la extensión `.sce` y se ejecuta mediante `exec`. Una instrucción terminada con punto y coma no muestra automáticamente su resultado. Para mostrar explícitamente un valor puede utilizarse `disp`. Los comentarios de una línea comienzan con `//`. El comando `clear` elimina las variables de usuario de la sesión. Un error escrito en la consola se informa inmediatamente; al ejecutar un archivo, el diagnóstico permite además localizar la instrucción dentro del archivo.

Contenido de `practica.sce`:

```
// Ejemplo de archivo de comandos

a = 5;
b = 3.14;
nombre = "hola";

suma = a + b;

disp(suma)
disp(nombre)

x = linspace(0, 2*%pi, 100);
disp(x)

// Para producir intencionalmente un error,
// cambie el calculo anterior por:
// suma = a + no_existe;
```

Desde la consola de Scilab:

```
exec("practica.sce")
```

Para repetir desde una sesión limpia:

```
clear
exec("practica.sce")
```

Octave. En Octave, los archivos de comandos utilizan normalmente la extensión `.m`. Pueden ejecutarse con `run` o, si se encuentran en el directorio de trabajo o en la ruta de búsqueda, escribiendo directamente su nombre. El punto y coma suprime la visualización automática de un resultado. Para mostrar explícitamente un valor puede utilizarse `disp`. Los comentarios de una línea pueden comenzar con `%`. El comando `clear` elimina las variables del espacio de trabajo. En un archivo, los mensajes de error permiten identificar además la línea donde se produjo el problema.

Contenido de `practica.m`:

```
% Ejemplo de archivo de comandos

a = 5;
b = 3.14;
nombre = "hola";

suma = a + b;

disp(suma)
disp(nombre)

x = linspace(0, 2*pi, 100);
disp(x)

% Para producir intencionalmente un error:
% suma = a + no_existe;
```

Desde la consola de Octave:

```
run("practica.m")
```

Para repetir desde una sesión limpia:

```
clear
run("practica.m")
```

Python. En el intérprete interactivo de Python, una expresión escrita directamente muestra su valor, mientras que una asignación no lo hace. Dentro de un archivo, una expresión aislada tampoco se muestra automáticamente; para hacerlo se utiliza `print`. Los comentarios de una línea comienzan con `#`. Los archivos utilizan normalmente la extensión `.py` y pueden ejecutarse con `python practica.py`. El intérprete estándar no posee un comando destinado a reiniciar

completamente el espacio de trabajo; para esta actividad es preferible cerrar la sesión y abrir una nueva. Cuando ocurre un error dentro de un archivo, el *traceback* informa el nombre del archivo y la línea correspondiente.

Contenido de `practica.py`:

```
import numpy as np

# Ejemplo de archivo de comandos

a = 5
b = 3.14
nombre = "hola"

suma = a + b

print(suma)
print(nombre)

x = np.linspace(0, 2*np.pi, 100)
print(x)

# Para producir intencionalmente un error:
# suma = a + no_existe
```

Desde una terminal:

```
python practica.py
```

En el intérprete interactivo pueden probarse, por ejemplo:

```
a = 5
b = 3.14
a + b

nombre = "hola"
nombre
```

Para comenzar una sesión completamente limpia, cierre el intérprete:

```
exit()
```

y vuelva a iniciarlo.

Perl. Para trabajar interactivamente con Perl y PDL puede utilizarse el shell `pdl2` o `perldl`. En el shell PDL puede habilitarse la visualización automática de expresiones con `do_print 1`; una expresión terminada con punto y coma no se muestra automáticamente. Para mostrar explícitamente un valor se utiliza `print` o el atajo `p` del shell PDL. Los comentarios comienzan con `#`. Los archivos de Perl

utilizan normalmente la extensión `.pl`; desde el shell PDL pueden ejecutarse mediante `do`. Para obtener una sesión limpia se puede salir del shell y volver a iniciarlo.

En una sesión interactiva de PDL puede probarse:

```
do_print 1

$a = 5
$b = 3.14
$a + $b

$nombre = 'hola'
$nombre
```

Contenido de `practica.pl`:

```
use PDL;
use PDL::Constants qw(PI);

# Ejemplo de archivo de comandos

$a = 5;
$b = 3.14;
$nombre = 'hola';

$suma = $a + $b;

print $suma, "\n";
print $nombre, "\n";

$x = sequence(100)*2*PI/99;
print $x, "\n";

# Para producir intencionalmente un error:
# no_existe();
```

Desde el shell PDL:

```
do './practica.pl'
```

También puede ejecutarse directamente desde una terminal:

```
perl practica.pl
```

Para comenzar nuevamente desde una sesión limpia:

```
exit
```

y vuelva a iniciar `pdl2` o `perldl`.

JavaScript. Node.js proporciona un entorno interactivo REPL cuando se ejecuta `node` sin indicar un archivo. En el REPL se muestra automáticamente el valor de una expresión; en un archivo las expresiones no se muestran por sí solas y se utiliza `console.log` para visualizar valores. A diferencia de Scilab y Octave, el punto y coma no funciona como mecanismo para suprimir la salida del REPL. Los comentarios de una línea comienzan con `//`. Los archivos utilizan normalmente la extensión `.js` y se ejecutan con `node practica.js`. El comando especial `.clear` limpia el contexto del REPL.

En el REPL de Node.js puede probarse:

```
let a = 5
let b = 3.14
a + b

let nombre = "hola"
nombre
```

Contenido de `practica.js`:

```
let a = 5
let b = 3.14
let nombre = "hola"

let suma = a + b

console.log(suma)
console.log(nombre)

let x = Array.from(
    {length:100},
    (_,i) => 2*Math.PI*i/99
)

console.log(x)

// Para producir intencionalmente un error:
// suma = a + no_existe
```

Desde una terminal:

```
node practica.js
```

Para limpiar una sesión interactiva de Node.js:

```
.clear
```

2. **Ejercicio:** Variables en el REPL

   En este ejercicio se comparará la ejecución de un archivo como programa independiente con su ejecución desde una sesión interactiva del mismo lenguaje.

   a) Con un editor de texto cree un archivo de programa que asigne $x = 3$ y calcule

   $$y = x^2 + 1.$$

   b) Ejecute el archivo directamente desde una terminal del sistema operativo.

   c) Inicie el intérprete interactivo en modo texto y, desde esa sesión, ejecute el mismo archivo.

   d) ¿Se mostró automáticamente el valor de y en ambos casos? Modifique el archivo, si es necesario, agregando una instrucción que muestre explícitamente el valor de y.

   e) Elimine del archivo solamente la instrucción que asigna un valor a x, dejando el cálculo de y y su visualización.

      Desde el intérprete interactivo:

      1. asigne $x = 3$;
      2. ejecute el archivo;
      3. cambie el valor de x, por ejemplo a $x = 5$;
      4. ejecute nuevamente el mismo archivo.

      Finalmente, salga del intérprete e intente ejecutar ese mismo archivo directamente desde la terminal del sistema operativo.

      Explique por qué el archivo puede utilizar el valor de x definido previamente cuando se ejecuta dentro de una sesión interactiva, pero ese valor no está disponible al iniciar una nueva ejecución desde la terminal del sistema operativo.

   La diferencia importante es que al ejecutar un programa directamente desde la terminal se inicia un nuevo proceso con su propio espacio de variables. En cambio, cuando el archivo se ejecuta desde un intérprete que ya está abierto, sus instrucciones pueden ejecutarse en el contexto de esa sesión y utilizar variables definidas previamente, de acuerdo con las reglas de cada lenguaje.

   Scilab. Los archivos de comandos de Scilab utilizan habitualmente la extensión `.sce`. Desde una sesión interactiva se ejecutan mediante `exec`. El punto y coma suprime la visualización automática; para mostrar explícitamente un valor se puede utilizar `disp`.

   Contenido inicial de `programa.sce`:

```
x = 3;
y = x^2 + 1;

// agregar para el inciso (d):
// disp(y)
```

   Desde la terminal del sistema operativo:

```
scilab-cli -f programa.sce
```

Para iniciar Scilab en modo texto:

```
scilab-cli
```

Desde la sesión interactiva:

```
exec("programa.sce",-1)
```

Para mostrar explícitamente el resultado, el archivo queda:

```
x = 3;
y = x^2 + 1;
disp(y)
```

Para el último inciso elimine solamente la primera instrucción:

```
y = x^2 + 1;
disp(y)
```

y desde la sesión interactiva pruebe:

```
x = 3
exec("programa.sce",-1)

x = 5
exec("programa.sce",-1)
```

En ambos casos el archivo utiliza el valor de `x` existente en la sesión. Si se cierra Scilab y se ejecuta directamente

```
scilab-cli -f programa.sce
```

la variable `x` no existe en esa nueva sesión y se produce un error.

Octave. Los archivos de comandos de Octave utilizan normalmente la extensión `.m`. Desde la sesión interactiva pueden ejecutarse mediante `source`. El punto y coma suprime la visualización automática; `disp` permite mostrar explícitamente un valor.

Contenido inicial de `programa.m`:

```
x = 3;
y = x^2 + 1;
```

```
% agregar para el inciso (d):
% disp(y)
```

Desde la terminal del sistema operativo:

```
octave programa.m
```

Para abrir el intérprete:

```
octave
```

Desde Octave:

```
source("programa.m")
```

También puede ejecutarse, si se encuentra en el directorio de trabajo, escribiendo su nombre:

```
programa
```

Para mostrar explícitamente el resultado:

```
x = 3;
y = x^2 + 1;
disp(y)
```

Para el último inciso:

```
y = x^2 + 1;
disp(y)
```

y desde la sesión interactiva:

```
x = 3;
source("programa.m")

x = 5;
source("programa.m")
```

Si se intenta ejecutar este último archivo desde una nueva sesión,

```
octave programa.m
```

la variable `x` no está definida.

Python. Los archivos de Python utilizan normalmente la extensión `.py`. Una expresión escrita en el intérprete interactivo puede mostrar su valor automáticamente, pero dentro de un archivo es necesario utilizar una instrucción como `print` para mostrarlo. El contenido de un archivo puede ejecutarse desde una sesión interactiva mediante `exec`.

Contenido inicial de `programa.py`:

```
x = 3
y = x**2 + 1

# agregar para el inciso (d):
# print(y)
```

Desde la terminal del sistema operativo:

```
python3 programa.py
```

Para abrir el intérprete:

```
python3
```

Desde Python:

```
exec(open("programa.py").read())
```

Para mostrar explícitamente el resultado:

```
x = 3
y = x**2 + 1
print(y)
```

Para el último inciso, el archivo queda:

```
y = x**2 + 1
print(y)
```

Desde la sesión interactiva:

```
x = 3
exec(open("programa.py").read())

x = 5
exec(open("programa.py").read())
```

La primera ejecución muestra

```
10
```

y la segunda

```
26
```

Si se sale de Python y se ejecuta

```
python3 programa.py
```

se obtiene un error porque esa nueva ejecución no contiene la variable `x` definida anteriormente.

Perl. Perl no proporciona como parte del intérprete estándar un REPL equivalente a los de Python u Octave. En este curso puede utilizarse `pdl2`, el intérprete interactivo de PDL. Un archivo Perl utiliza normalmente la extensión `.pl`; puede ejecutarse dentro de la sesión mediante `do`. Para que el ejemplo final pueda compartir la variable con la sesión se utiliza aquí una variable del paquete, declarada mediante `our`.

Contenido inicial de `programa.pl`:

```
use strict;
use warnings;

our $x = 3;
our $y = $x**2 + 1;

# agregar para el inciso (d):
# print $y, "\n";
```

Desde la terminal del sistema operativo:

```
perl programa.pl
```

Para abrir el intérprete interactivo de PDL:

```
pdl2
```

Desde la sesión interactiva:

```
do './programa.pl';
```

Para mostrar explícitamente el resultado:

```
use strict;
```

```
use warnings;

our $x = 3;
our $y = $x**2 + 1;

print $y, "\n";
```

Para el último inciso, elimine la asignación pero conserve la declaración:

```
use strict;
use warnings;

our $x;
our $y = $x**2 + 1;

print $y, "\n";
```

Desde `pdl2`:

```
$x = 3;
do './programa.pl';

$x = 5;
do './programa.pl';
```

Los resultados son nuevamente 10 y 26. Si el archivo se ejecuta como un proceso Perl nuevo, el valor asignado previamente en `pdl2` no pertenece a ese proceso; con `warnings` se informa además que `$x` no posee un valor inicial.

JavaScript. Node.js proporciona un intérprete interactivo REPL al ejecutar `node` sin indicar un archivo. Los archivos JavaScript utilizan normalmente la extensión `.js` y se ejecutan con `node`. Dentro del REPL, `.load` carga un archivo en la sesión actual. Para mostrar explícitamente un resultado se utiliza `console.log`.

Contenido inicial de `programa.js`:

```
let x = 3;
let y = x**2 + 1;

// agregar para el inciso (d):
// console.log(y);
```

Desde la terminal del sistema operativo:

```
node programa.js
```

Para abrir el REPL:

```
```

```
node
```

Desde Node.js:

```
.load programa.js
```

Para mostrar explícitamente el resultado:

```
let x = 3;
let y = x**2 + 1;

console.log(y);
```

Para el último inciso:

```
let y = x**2 + 1;
console.log(y);
```

Desde el REPL:

```
let x = 3
.load programa.js

x = 5
.load programa.js
```

Los resultados son 10 y 26. En cambio, al salir del REPL y ejecutar

```
node programa.js
```

se inicia un proceso nuevo y se produce un error porque `x` no está definida.

3. **Ejercicio:** Tiro parabólico con ciclos

   Considere el movimiento de un proyectil lanzado desde el origen, sin resistencia del aire, con velocidad inicial v_0 y ángulo de lanzamiento θ.

   Utilice

   $$g = 9{,}8 \text{ m/s}^2.$$

   Para cada lanzamiento calcule:

   - el tiempo de vuelo,

     $$t_v = \frac{2v_0 \sin\theta}{g};$$

- el alcance,

$$R = \frac{v_0^2 \sin(2\theta)}{g};$$

- la altura máxima,

$$H = \frac{v_0^2 \sin^2 \theta}{2g};$$

- la posición horizontal y vertical durante el vuelo,

$$x(t) = v_0 \cos\theta\, t,$$

$$y(t) = v_0 \sin\theta\, t - \frac{1}{2} g t^2.$$

Mantenga constante la velocidad inicial y realice el cálculo para los cinco ángulos

$$10^\circ, \qquad 30^\circ, \qquad 45^\circ, \qquad 60^\circ, \qquad 80^\circ.$$

Resuelva el problema en dos etapas.

a) Cree un archivo de programa que suponga que las variables correspondientes a v_0 y θ ya fueron definidas previamente en la sesión interactiva.

   El programa debe calcular el tiempo de vuelo, el alcance, la altura máxima y una trayectoria formada por varios valores de t, $x(t)$ e $y(t)$.

b) Desde la terminal interactiva asigne un valor a v_0 y el primero de los ángulos. Ejecute el programa.

   Repita manualmente el procedimiento para los cinco ángulos, sin utilizar todavía un ciclo.

   Por ejemplo, el esquema de trabajo será

```
asignar v0
asignar angulo = 10
ejecutar programa
guardar resultados

asignar angulo = 30
ejecutar programa
guardar resultados

...

asignar angulo = 80
ejecutar programa
guardar resultados
```

c) Guarde, después de cada ejecución, los resultados escalares en vectores indizados por número de ejecución:

$$\text{angulo}(i), \qquad \text{alcance}(i), \qquad \text{altura}(i), \qquad \text{tiempo}(i).$$

Organice luego estos resultados en una matriz equivalente a

$$\begin{pmatrix} \theta_1 & R_1 & H_1 & t_{v1} \\ \theta_2 & R_2 & H_2 & t_{v2} \\ \theta_3 & R_3 & H_3 & t_{v3} \\ \theta_4 & R_4 & H_4 & t_{v4} \\ \theta_5 & R_5 & H_5 & t_{v5} \end{pmatrix}.$$

Las trayectorias completas pueden guardarse por separado, ya que cada ángulo tiene un tiempo de vuelo diferente y, por lo tanto, puede tener un vector de tiempos distinto.

d) Muestre la matriz de resultados.

e) Determine numéricamente cuál de los cinco ángulos considerados produce el mayor alcance.

Indique tanto el alcance máximo como el ángulo correspondiente.

f) Escriba una segunda versión del programa que realice automáticamente los cinco lanzamientos mediante un ciclo `for`.

Dentro del ciclo:

- tome el ángulo correspondiente;
- calcule el tiempo de vuelo;
- calcule el alcance;
- calcule la altura máxima;
- calcule la trayectoria;
- almacene los resultados en la posición correspondiente de los vectores o de la estructura utilizada.

g) Compare las dos formas de resolución:

- ejecución manual repetida desde la terminal interactiva;
- ejecución automática mediante un ciclo `for`.

Explique qué instrucciones se repiten en la primera versión y cómo el ciclo permite expresar esa repetición de manera más compacta.

Scilab. El primer archivo puede utilizar directamente las variables `v0` y `angulo` definidas previamente en la sesión. El ángulo debe convertirse de grados a radianes. Para la trayectoria puede utilizarse `linspace`. En la segunda versión se introduce el ciclo `for`.

Ejemplo del archivo correspondiente a un solo lanzamiento:

```
theta=angulo*%pi/180;
```

```
tiempo_vuelo=2*v0*sin(theta)/g;
alcance=v0^2*sin(2*theta)/g;
altura=v0^2*sin(theta)^2/(2*g);

t=linspace(0,tiempo_vuelo,101);

x=v0*cos(theta)*t;
y=v0*sin(theta)*t-g*t.^2/2;
```

Desde la sesión interactiva puede realizarse, por ejemplo:

```
g=9.8;
v0=20;

angulo=10;
exec("tiro.sce",-1);
angulos(1)=angulo;
alcances(1)=alcance;
alturas(1)=altura;
tiempos(1)=tiempo_vuelo;

angulo=30;
exec("tiro.sce",-1);
angulos(2)=angulo;
alcances(2)=alcance;
alturas(2)=altura;
tiempos(2)=tiempo_vuelo;
```

y continuar de la misma manera para 45°, 60° y 80°.

La matriz resumen puede construirse mediante

```
resultados=[
    angulos'
    alcances'
    alturas'
    tiempos'
]';

resultados
```

El mayor alcance se obtiene con

```
[alcance_maximo,k]=max(alcances);

angulo_maximo=angulos(k)
```

La segunda versión utiliza un ciclo:

```
```

```
g=9.8;
v0=20;

angulos=[10 30 45 60 80];

for i=1:length(angulos)

    theta=angulos(i)*%pi/180;

    tiempos(i)=2*v0*sin(theta)/g;
    alcances(i)=v0^2*sin(2*theta)/g;
    alturas(i)=v0^2*sin(theta)^2/(2*g);

    t=linspace(0,tiempos(i),101);

    trayectorias(i).t=t;
    trayectorias(i).x=v0*cos(theta)*t;
    trayectorias(i).y=v0*sin(theta)*t-g*t.^2/2;

end

resultados=[
    angulos'
    alcances'
    alturas'
    tiempos'
]';

[alcance_maximo,k]=max(alcances);
angulo_maximo=angulos(k);
```

Octave. El procedimiento es análogo. El archivo correspondiente a un lanzamiento utiliza las variables `v0`, `angulo` y `g` existentes en el espacio de trabajo. La segunda versión introduce un ciclo `for`.

```
theta=angulo*pi/180;

tiempo_vuelo=2*v0*sin(theta)/g;
alcance=v0^2*sin(2*theta)/g;
altura=v0^2*sin(theta)^2/(2*g);

t=linspace(0,tiempo_vuelo,101);

x=v0*cos(theta)*t;
y=v0*sin(theta)*t-g*t.^2/2;
```

Ejemplo de ejecuciones manuales:

```
g=9.8;
```

```
v0=20;

angulo=10;
source("tiro.m");
angulos(1)=angulo;
alcances(1)=alcance;
alturas(1)=altura;
tiempos(1)=tiempo_vuelo;

angulo=30;
source("tiro.m");
angulos(2)=angulo;
alcances(2)=alcance;
alturas(2)=altura;
tiempos(2)=tiempo_vuelo;
```

Después de completar las cinco ejecuciones:

```
resultados=[
    angulos(:) alcances(:) alturas(:) tiempos(:)
]

[alcance_maximo,k]=max(alcances);
angulo_maximo=angulos(k)
```

Versión con ciclo:

```
g=9.8;
v0=20;

angulos=[10 30 45 60 80];

for i=1:length(angulos)

    theta=angulos(i)*pi/180;

    tiempos(i)=2*v0*sin(theta)/g;
    alcances(i)=v0^2*sin(2*theta)/g;
    alturas(i)=v0^2*sin(theta)^2/(2*g);

    t=linspace(0,tiempos(i),101);

    trayectorias{i}.t=t;
    trayectorias{i}.x=v0*cos(theta)*t;
    trayectorias{i}.y=v0*sin(theta)*t-g*t.^2/2;

end

resultados=[
    angulos(:) alcances(:) alturas(:) tiempos(:)
```

```
];

[alcance_maximo,k]=max(alcances);
angulo_maximo=angulos(k);
```

Python. En Python, `exec` permite ejecutar el archivo dentro del espacio de nombres de la sesión interactiva. En la segunda versión se utiliza por primera vez un ciclo `for`.

Contenido de `tiro.py` para un solo lanzamiento:

```
theta=np.deg2rad(angulo)

tiempo_vuelo=2*v0*np.sin(theta)/g
alcance=v0**2*np.sin(2*theta)/g
altura=v0**2*np.sin(theta)**2/(2*g)

t=np.linspace(0,tiempo_vuelo,101)

x=v0*np.cos(theta)*t
y=v0*np.sin(theta)*t-g*t**2/2
```

Desde la sesión interactiva:

```
import numpy as np

g=9.8
v0=20

angulo=10
exec(open("tiro.py").read())
angulos[0]=angulo
alcances[0]=alcance
alturas[0]=altura
tiempos[0]=tiempo_vuelo

angulo=30
exec(open("tiro.py").read())
angulos[1]=angulo
alcances[1]=alcance
alturas[1]=altura
tiempos[1]=tiempo_vuelo
```

Antes de esas asignaciones pueden prepararse los vectores mediante

```
angulos=np.zeros(5)
alcances=np.zeros(5)
alturas=np.zeros(5)
tiempos=np.zeros(5)
```

Después de completar las cinco ejecuciones:

```
resultados=np.column_stack(
    (angulos,alcances,alturas,tiempos)
)

k=np.argmax(alcances)

alcance_maximo=alcances[k]
angulo_maximo=angulos[k]
```

Versión con ciclo:

```
import numpy as np

g=9.8
v0=20

angulos=np.array([10,30,45,60,80])

alcances=np.zeros(5)
alturas=np.zeros(5)
tiempos=np.zeros(5)

trayectorias=[]

for i in range(5):

    theta=np.deg2rad(angulos[i])

    tiempos[i]=2*v0*np.sin(theta)/g
    alcances[i]=v0**2*np.sin(2*theta)/g
    alturas[i]=v0**2*np.sin(theta)**2/(2*g)

    t=np.linspace(0,tiempos[i],101)

    x=v0*np.cos(theta)*t
    y=v0*np.sin(theta)*t-g*t**2/2

    trayectorias.append(
        {"t":t,"x":x,"y":y}
    )

resultados=np.column_stack(
    (angulos,alcances,alturas,tiempos)
)

k=np.argmax(alcances)

alcance_maximo=alcances[k]
```

```
angulo_maximo=angulos[k]
```

Perl. En Perl/PDL pueden mantenerse los datos numéricos en piddles. Para la segunda parte se utiliza explícitamente un ciclo `for`, que permite repetir el mismo bloque para cada ángulo.

Versión con ciclo:

```
use PDL;
use PDL::Constants qw(PI);

$g=9.8;
$v0=20;

$angulos=pdl(10,30,45,60,80);

$alcances=zeroes(5);
$alturas=zeroes(5);
$tiempos=zeroes(5);

@trayectorias=();

for ($i=0; $i<5; $i++) {

    $angulo=$angulos->at($i);
    $theta=$angulo*PI/180;

    $tv=2*$v0*sin($theta)/$g;
    $R=$v0**2*sin(2*$theta)/$g;
    $H=$v0**2*sin($theta)**2/(2*$g);

    $tiempos->set($i,$tv);
    $alcances->set($i,$R);
    $alturas->set($i,$H);

    $t=sequence(101)*$tv/100;

    $x=$v0*cos($theta)*$t;
    $y=$v0*sin($theta)*$t-$g*$t**2/2;

    $trayectorias[$i]={
        t=>$t,
        x=>$x,
        y=>$y
    };
}

$k=$alcances->maximum_ind;

$alcance_maximo=$alcances->at($k);
$angulo_maximo=$angulos->at($k);
```

```
p $angulos;
p $alcances;
p $alturas;
p $tiempos;
p $angulo_maximo;
```

JavaScript. En JavaScript los resultados escalares pueden almacenarse en arreglos y las trayectorias en un arreglo de objetos. El ciclo `for` permite automatizar las cinco ejecuciones.

```
const g=9.8
const v0=20

const angulos=[10,30,45,60,80]

const alcances=[]
const alturas=[]
const tiempos=[]
const trayectorias=[]

for(let i=0;i<angulos.length;i++){

    const theta=angulos[i]*Math.PI/180

    tiempos[i]=
        2*v0*Math.sin(theta)/g

    alcances[i]=
        v0**2*Math.sin(2*theta)/g

    alturas[i]=
        v0**2*Math.sin(theta)**2/(2*g)

    const t=Array.from(
        {length:101},
        (_,j)=>j*tiempos[i]/100
    )

    const x=t.map(
        ti=>v0*Math.cos(theta)*ti
    )

    const y=t.map(
        ti=>
            v0*Math.sin(theta)*ti
            -g*ti**2/2
    )

    trayectorias[i]={
```

```
            t:t,
            x:x,
            y:y
        }
    }

    const alcance_maximo=Math.max(...alcances)

    const k=alcances.indexOf(
        alcance_maximo
    )

    const angulo_maximo=angulos[k]

    const resultados=angulos.map(
        (angulo,i)=>[
            angulo,
            alcances[i],
            alturas[i],
            tiempos[i]
        ]
    )

    console.log(resultados)
    console.log(alcance_maximo)
    console.log(angulo_maximo)
```

4. **Ejercicio:** Calendario con meses y bisiestos

   Antes de ejecutar el programa, defina en la sesión interactiva las variables `mes` y `anio`.

   La variable `mes` representa el número de mes y `anio` el año correspondiente.

   El programa deberá:

   a) validar que `mes` sea un número entero comprendido entre 1 y 12;

   b) validar que `anio` sea un número entero positivo;

   c) mostrar un mensaje de error si alguno de los valores no es válido;

   d) utilizar una estructura condicional `if-elseif-else`, o su sintaxis equivalente, para determinar el nombre del mes;

   e) determinar la cantidad de días del mes;

   f) cuando el mes seleccionado sea febrero, determinar si el año es bisiesto. Considere que un año es bisiesto si es divisible por 400, o si es divisible por 4 pero no por 100;

   g) mostrar el nombre del mes y su cantidad de días;

   h) probar el programa modificando previamente el valor de `mes`. Utilice, entre otros,

$$2, \quad 7, \quad 12, \quad 2{,}5, \quad 0, \quad 13.$$

i) probar febrero con años bisiestos y no bisiestos, por ejemplo

$$2000, \quad 2024, \quad 2025, \quad 2100.$$

j) escriba una segunda versión que almacene los nombres de los doce meses y sus cantidades de días en colecciones y utilice un ciclo `for` para mostrar los doce meses.

Para un año no bisiesto, las cantidades de días son

$$31,\ 28,\ 31,\ 30,\ 31,\ 30,\ 31,\ 31,\ 30,\ 31,\ 30,\ 31.$$

La condición de año bisiesto es

(anio divisible por 400) o [(anio divisible por 4) y no divisible por 100].

Scilab. En esta primera versión se utiliza solamente `if`, `elseif` y `else`. El resto de la división entera puede calcularse con `modulo`. Los índices de los vectores comienzan en 1.

```
validoMes=%f;
validoAnio=%f;

if type(mes)==1 then
    if size(mes,"*")==1 then
        validoMes=(mes==floor(mes)) & (mes>=1) & (mes<=12);
    end
end

if type(anio)==1 then
    if size(anio,"*")==1 then
        validoAnio=(anio==floor(anio)) & (anio>0);
    end
end

if ~validoMes | ~validoAnio then

    disp("Mes o anio no valido")

else

    bisiesto=(modulo(anio,400)==0) | ..
             ((modulo(anio,4)==0) & (modulo(anio,100)<>0));

    if mes==1 then
        nombre="Enero";
```

```
        dias=31;

    elseif mes==2 then
        nombre="Febrero";

        if bisiesto then
            dias=29;
        else
            dias=28;
        end

    elseif mes==3 then
        nombre="Marzo";
        dias=31;

    elseif mes==4 then
        nombre="Abril";
        dias=30;

    elseif mes==5 then
        nombre="Mayo";
        dias=31;

    elseif mes==6 then
        nombre="Junio";
        dias=30;

    elseif mes==7 then
        nombre="Julio";
        dias=31;

    elseif mes==8 then
        nombre="Agosto";
        dias=31;

    elseif mes==9 then
        nombre="Septiembre";
        dias=30;

    elseif mes==10 then
        nombre="Octubre";
        dias=31;

    elseif mes==11 then
        nombre="Noviembre";
        dias=30;

    else
        nombre="Diciembre";
        dias=31;
    end
```

```
    disp(nombre)
    disp(dias)

end
```

Segunda versión:

```
nombres=["Enero","Febrero","Marzo","Abril", ..
         "Mayo","Junio","Julio","Agosto", ..
         "Septiembre","Octubre","Noviembre","Diciembre"];

dias=[31 28 31 30 31 30 31 31 30 31 30 31];

bisiesto=(modulo(anio,400)==0) | ..
          ((modulo(anio,4)==0) & (modulo(anio,100)<>0));

if bisiesto then
    dias(2)=29;
end

for i=1:12
    disp([string(i),nombres(i),string(dias(i))])
end
```

Octave. La estructura es la misma que en Scilab: `if-elseif-else`. Para el resto se utiliza `mod`. Los índices comienzan en 1.

```
validoMes=isnumeric(mes) && isscalar(mes) ...
           && isfinite(mes) && mes==fix(mes) ...
           && mes>=1 && mes<=12;

validoAnio=isnumeric(anio) && isscalar(anio) ...
            && isfinite(anio) && anio==fix(anio) ...
            && anio>0;

if !validoMes || !validoAnio

  disp("Mes o anio no valido")

else

  bisiesto=(mod(anio,400)==0) || ...
            (mod(anio,4)==0 && mod(anio,100)!=0);

  if mes==1
    nombre="Enero";
    dias=31;
```

```
elseif mes==2
  nombre="Febrero";

  if bisiesto
    dias=29;
  else
    dias=28;
  endif

elseif mes==3
  nombre="Marzo";
  dias=31;

elseif mes==4
  nombre="Abril";
  dias=30;

elseif mes==5
  nombre="Mayo";
  dias=31;

elseif mes==6
  nombre="Junio";
  dias=30;

elseif mes==7
  nombre="Julio";
  dias=31;

elseif mes==8
  nombre="Agosto";
  dias=31;

elseif mes==9
  nombre="Septiembre";
  dias=30;

elseif mes==10
  nombre="Octubre";
  dias=31;

elseif mes==11
  nombre="Noviembre";
  dias=30;

else
  nombre="Diciembre";
  dias=31;
endif

disp(nombre)
```

```
  disp(dias)

endif
```

Segunda versión:

```
nombres={"Enero","Febrero","Marzo","Abril", ...
         "Mayo","Junio","Julio","Agosto", ...
         "Septiembre","Octubre","Noviembre","Diciembre"};

dias=[31 28 31 30 31 30 31 31 30 31 30 31];

bisiesto=(mod(anio,400)==0) || ...
          (mod(anio,4)==0 && mod(anio,100)!=0);

if bisiesto
  dias(2)=29;
endif

for i=1:12
  disp([num2str(i) " " nombres{i} " " num2str(dias(i))])
endfor
```

Python. Python utiliza `if`, `elif` y `else`. Los índices de las listas comienzan en 0.

```
valido_mes=(
    type(mes) in (int,float)
    and mes==int(mes)
    and 1<=mes<=12
)

valido_anio=(
    type(anio) in (int,float)
    and anio==int(anio)
    and anio>0
)

if not valido_mes or not valido_anio:

    print("Mes o anio no valido")

else:

    mes=int(mes)
    anio=int(anio)

    bisiesto=(
        anio%400==0
        or (anio%4==0 and anio%100!=0)
    )
```

```
    if mes==1:
        nombre="Enero"
        dias=31

    elif mes==2:
        nombre="Febrero"

        if bisiesto:
            dias=29
        else:
            dias=28

    elif mes==3:
        nombre="Marzo"
        dias=31

    elif mes==4:
        nombre="Abril"
        dias=30

    elif mes==5:
        nombre="Mayo"
        dias=31

    elif mes==6:
        nombre="Junio"
        dias=30

    elif mes==7:
        nombre="Julio"
        dias=31

    elif mes==8:
        nombre="Agosto"
        dias=31

    elif mes==9:
        nombre="Septiembre"
        dias=30

    elif mes==10:
        nombre="Octubre"
        dias=31

    elif mes==11:
        nombre="Noviembre"
        dias=30

    else:
        nombre="Diciembre"
```

```
            dias=31

    print(nombre)
    print(dias)
```

Segunda versión:

```
nombres=[
    "Enero","Febrero","Marzo","Abril",
    "Mayo","Junio","Julio","Agosto",
    "Septiembre","Octubre","Noviembre","Diciembre"
]

dias=[31,28,31,30,31,30,31,31,30,31,30,31]

bisiesto=(
    anio%400==0
    or (anio%4==0 and anio%100!=0)
)

if bisiesto:
    dias[1]=29

for i in range(12):
    print(i+1,nombres[i],dias[i])
```

Perl. Perl utiliza `if`, `elsif` y `else`. Para verificar que una variable contiene un valor numérico puede utilizarse `looks_like_number`. Los índices de los arreglos comienzan en 0.

```
use Scalar::Util qw(looks_like_number);

$valido_mes=
    looks_like_number($mes)
    && int($mes)==$mes
    && $mes>=1
    && $mes<=12;

$valido_anio=
    looks_like_number($anio)
    && int($anio)==$anio
    && $anio>0;

if (!$valido_mes || !$valido_anio) {

    print "Mes o anio no valido\n";

}
else {
```

```
$bisiesto=
    ($anio%400==0)
    || (($anio%4==0) && ($anio%100!=0));

if ($mes==1) {
    $nombre='Enero';
    $dias=31;
}
elsif ($mes==2) {
    $nombre='Febrero';

    if ($bisiesto) {
        $dias=29;
    }
    else {
        $dias=28;
    }
}
elsif ($mes==3) {
    $nombre='Marzo';
    $dias=31;
}
elsif ($mes==4) {
    $nombre='Abril';
    $dias=30;
}
elsif ($mes==5) {
    $nombre='Mayo';
    $dias=31;
}
elsif ($mes==6) {
    $nombre='Junio';
    $dias=30;
}
elsif ($mes==7) {
    $nombre='Julio';
    $dias=31;
}
elsif ($mes==8) {
    $nombre='Agosto';
    $dias=31;
}
elsif ($mes==9) {
    $nombre='Septiembre';
    $dias=30;
}
elsif ($mes==10) {
    $nombre='Octubre';
    $dias=31;
}
```

```
    elsif ($mes==11) {
        $nombre='Noviembre';
        $dias=30;
    }
    else {
        $nombre='Diciembre';
        $dias=31;
    }

    print $nombre,"\n";
    print $dias,"\n";
}
```

Segunda versión:

```
@nombres=(
    'Enero','Febrero','Marzo','Abril',
    'Mayo','Junio','Julio','Agosto',
    'Septiembre','Octubre','Noviembre','Diciembre'
);

@dias=(31,28,31,30,31,30,31,31,30,31,30,31);

$bisiesto=
    ($anio%400==0)
    || (($anio%4==0) && ($anio%100!=0));

if ($bisiesto) {
    $dias[1]=29;
}

for ($i=0; $i<12; $i++) {
    print $i+1," ",$nombres[$i]," ",$dias[$i],"\n";
}
```

JavaScript. JavaScript utiliza `if`, `else if` y `else`. `Number.isInteger` permite verificar directamente que un valor sea entero. Los índices de los arreglos comienzan en 0.

```
const validoMes=
    Number.isInteger(mes)
    && mes>=1
    && mes<=12

const validoAnio=
    Number.isInteger(anio)
    && anio>0

if(!validoMes || !validoAnio){
```

```
        console.log("Mes o anio no valido")

    }
    else{

        const bisiesto=
            anio%400===0
            || (anio%4===0 && anio%100!==0)

        let nombre
        let dias

        if(mes===1){
            nombre="Enero"
            dias=31
        }
        else if(mes===2){
            nombre="Febrero"

            if(bisiesto){
                dias=29
            }
            else{
                dias=28
            }
        }
        else if(mes===3){
            nombre="Marzo"
            dias=31
        }
        else if(mes===4){
            nombre="Abril"
            dias=30
        }
        else if(mes===5){
            nombre="Mayo"
            dias=31
        }
        else if(mes===6){
            nombre="Junio"
            dias=30
        }
        else if(mes===7){
            nombre="Julio"
            dias=31
        }
        else if(mes===8){
            nombre="Agosto"
            dias=31
        }
```

```
    else if(mes===9){
        nombre="Septiembre"
        dias=30
    }
    else if(mes===10){
        nombre="Octubre"
        dias=31
    }
    else if(mes===11){
        nombre="Noviembre"
        dias=30
    }
    else{
        nombre="Diciembre"
        dias=31
    }

    console.log(nombre)
    console.log(dias)
}
```

Segunda versión:

```
const nombres=[
    "Enero","Febrero","Marzo","Abril",
    "Mayo","Junio","Julio","Agosto",
    "Septiembre","Octubre","Noviembre","Diciembre"
]

const dias=[31,28,31,30,31,30,31,31,30,31,30,31]

const bisiesto=
    anio%400===0
    || (anio%4===0 && anio%100!==0)

if(bisiesto){
    dias[1]=29
}

for(let i=0;i<12;i++){
    console.log(i+1,nombres[i],dias[i])
}
```

5. **Ejercicio:** Días laborables y clases

   Construya una colección con los siete días de la semana, ordenados desde domingo hasta sábado. Utilice la correspondencia

$$1 = \text{domingo}, \quad 2 = \text{lunes}, \quad \ldots, \quad 7 = \text{sábado}.$$

Antes de ejecutar el programa, defina una variable `dia` con un entero entre 1 y 7. El programa deberá indicar el nombre del día, si es laborable, si pertenece al fin de semana y si corresponde a una clase. Utilice una pequeña colección `dias_clase` que cada estudiante adaptará a su horario.

Luego:

a) valide que `dia` sea un entero entre 1 y 7;

b) prediga el resultado para cada valor antes de ejecutar;

c) recorra posteriormente toda la colección mediante un ciclo `for`;

d) identifique colección, recorrido, condición y salida;

e) evite escribir una condición diferente para cada día.

Scilab. Los índices comienzan en 1. En el ejemplo se supone que hay clase los martes y jueves, índices 3 y 5.

```
dias=["domingo","lunes","martes","miercoles","jueves","viernes","sabado"
dias_clase=[3 5];
valido=%f;
if type(dia)==1 then
    if size(dia,"*")==1 then
        valido=(dia==floor(dia)) & (dia>=1) & (dia<=7);
    end
end
if ~valido then
    disp("Dia no valido")
else
    disp(dias(dia))
    if (dia>=2) & (dia<=6) then disp("laborable")
    else disp("fin de semana")
    end
    if or(dia==dias_clase) then disp("dia con clase")
    else disp("dia sin clase")
    end
end
for i=1:7
    if (i>=2) & (i<=6) then estado="laborable";
    else estado="fin de semana";
    end
    disp([dias(i),estado])
end
```

Octave. La colección se guarda en una celda. `any` permite comprobar si el índice está dentro de `dias_clase`.

```
dias={"domingo","lunes","martes","miercoles","jueves","viernes","sabado"
dias_clase=[3 5];
valido=isnumeric(dia) && isscalar(dia) && isfinite(dia) ...
```

```
        && dia==fix(dia) && dia>=1 && dia<=7;
if !valido
  disp("Dia no valido")
else
  disp(dias{dia})
  if dia>=2 && dia<=6, disp("laborable")
  else, disp("fin de semana")
  endif
  if any(dia==dias_clase), disp("dia con clase")
  else, disp("dia sin clase")
  endif
endif
for i=1:7
  if i>=2 && i<=6, estado="laborable";
  else, estado="fin de semana";
  endif
  disp([dias{i} " - " estado])
endfor
```

Python. La variable `dia` usa valores 1 a 7 aunque los índices de las listas comienzan en 0.

```
dias=["domingo","lunes","martes","miercoles","jueves","viernes","sabado"
dias_clase=[3,5]
valido=type(dia) in (int,float) and int(dia)==dia and
   1<=dia<=7
if not valido:
    print("Dia no valido")
else:
    dia=int(dia)
    print(dias[dia-1])
    if 2<=dia<=6: print("laborable")
    else: print("fin de semana")
    if dia in dias_clase: print("dia con clase")
    else: print("dia sin clase")
for i in range(1,8):
    if 2<=i<=6: estado="laborable"
    else: estado="fin de semana"
    print(dias[i-1],estado)
```

Perl. Los índices de arreglos comienzan en 0, pero `dia` conserva la numeración 1 a 7.

```
use Scalar::Util qw(looks_like_number);
@dias=('domingo','lunes','martes','miercoles','jueves','viernes','sabado
@dias_clase=(3,5);
$valido=looks_like_number($dia) && int($dia)==$dia &&
   $dia>=1 && $dia<=7;
if (!$valido) { print "Dia no valido\n"; }
```

```
else {
    print $dias[$dia-1],"\n";
    if ($dia>=2 && $dia<=6) { print "laborable\n"; }
    else { print "fin de semana\n"; }
    if (grep {$_==$dia} @dias_clase) { print "dia con
       clase\n"; }
    else { print "dia sin clase\n"; }
}
for ($i=1; $i<=7; $i++) {
    if ($i>=2 && $i<=6) { $estado='laborable'; }
    else { $estado='fin de semana'; }
    print $dias[$i-1]," ",$estado,"\n";
}
```

JavaScript. El arreglo se indexa desde 0, pero la variable `dia` conserva la numeración 1 a 7.

```
const
   dias=["domingo","lunes","martes","miercoles","jueves","viernes","saba
const diasClase=[3,5]
const valido=Number.isInteger(dia) && dia>=1 && dia<=7
if(!valido){ console.log("Dia no valido") }
else{
    console.log(dias[dia-1])
    if(dia>=2 && dia<=6){ console.log("laborable") }
    else{ console.log("fin de semana") }
    if(diasClase.includes(dia)){ console.log("dia con
       clase") }
    else{ console.log("dia sin clase") }
}
for(let i=1;i<=7;i++){
    let estado

    if(i>=2 && i<=6){ estado="laborable" }
    else{ estado="fin de semana" }
    console.log(dias[i-1],estado)
}
```

6. **Ejercicio:** Ciclos `for` y `while`

   En el ejercicio anterior el número de elementos era conocido: siete días. Esa situación se adapta a una repetición contada mediante `for`. Escriba ahora una segunda versión que utilice una repetición controlada por condición. Recorra los días desde domingo y termine tan pronto como se hayan revisado tres días laborables.

   a) identifique la inicialización de las variables de control;

   b) identifique la condición de continuación;

   c) identifique la actualización de las variables;

d) produzca intencionalmente un ciclo infinito suprimiendo la actualización del índice en la versión con `while`; explique por qué no termina y corríjalo;

e) compare cuándo resulta más apropiado `for` y cuándo `while`.

Scilab. La repetición contada se escribe con `for`; la repetición controlada por condición, con `while`.

```
dias=["domingo","lunes","martes","miercoles","jueves","viernes","sabado"
for i=1:7
    if (i>=2) & (i<=6) then disp([dias(i),"laborable"])
    else disp([dias(i),"fin de semana"])
    end
end
i=1; revisados=0;
while (i<=7) & (revisados<3)
    if (i>=2) & (i<=6) then
        disp(dias(i))
        revisados=revisados+1;
    end
    i=i+1;
end
// Ciclo infinito: quite la actualizacion i=i+1 del while
   anterior.
```

Octave. En `while` se controlan el índice del día y la cantidad de laborables revisados.

```
dias={"domingo","lunes","martes","miercoles","jueves","viernes","sabado"
for i=1:7
  if i>=2 && i<=6, disp([dias{i} " - laborable"])
  else, disp([dias{i} " - fin de semana"])
  endif
endfor
i=1; revisados=0;
while i<=7 && revisados<3
  if i>=2 && i<=6
    disp(dias{i})
    revisados=revisados+1;
  endif
  i=i+1;
endwhile
% Ciclo infinito: quite i=i+1.
```

Python. La condición combina que todavía queden días y que todavía no se hayan revisado tres laborables.

```
dias=["domingo","lunes","martes","miercoles","jueves","viernes","sabado"
for i in range(1,8):
```

```
    if 2<=i<=6: print(dias[i-1],"laborable")
    else: print(dias[i-1],"fin de semana")
i=1
revisados=0
while i<=7 and revisados<3:
    if 2<=i<=6:
        print(dias[i-1])
        revisados=revisados+1
    i=i+1
# Ciclo infinito: quite i=i+1.
```

Perl. La actualización del índice es indispensable para que la condición pueda llegar a ser falsa.

```
@dias=('domingo','lunes','martes','miercoles','jueves','viernes','sabado
for ($i=1; $i<=7; $i++) {
    if ($i>=2 && $i<=6) { print $dias[$i-1]," laborable\n"; }
    else { print $dias[$i-1]," fin de semana\n"; }
}
$i=1; $revisados=0;
while ($i<=7 && $revisados<3) {
    if ($i>=2 && $i<=6) {
        print $dias[$i-1],"\n";
        $revisados=$revisados+1;
    }
    $i=$i+1;
}
# Ciclo infinito: quite $i=$i+1.
```

JavaScript. El `for` expresa una cantidad predeterminada de repeticiones y el `while` continúa mientras se mantenga una condición.

```
const
   dias=["domingo","lunes","martes","miercoles","jueves","viernes","saba
for(let i=1;i<=7;i++){
    if(i>=2 && i<=6){ console.log(dias[i-1],"laborable") }
    else{ console.log(dias[i-1],"fin de semana") }
}
let i=1
let revisados=0
while(i<=7 && revisados<3){
    if(i>=2 && i<=6){
        console.log(dias[i-1])
        revisados=revisados+1
    }
    i=i+1
}
// Ciclo infinito: quite i=i+1.
```

7. **Ejercicio:** Paridad de libreta

   Asigne a una variable denominada `libreta` su número de Libreta Universitaria. Determine si el valor es par o impar utilizando el resto de la división entera por 2. Asigne a `mensaje` la cadena "es par" o "es impar" y muéstrela.

   a) valide primero que `libreta` represente un entero positivo;

   b) muestre el mensaje solamente si la Libreta Universitaria es válida;

   c) pruebe con cero y números negativos. Explique que tienen paridad matemática, pero son rechazados por la validación adoptada para una LU;

   d) extienda la idea a un vector de enteros, admitiendo allí cero y negativos;

   e) cuente cuántos elementos pares e impares contiene el vector.

   Scilab. El resto se obtiene mediante `modulo`.

```
valido=%f;
if type(libreta)==1 then
    if size(libreta,"*")==1 then
        valido=(libreta==floor(libreta)) & (libreta>0);
    end
end
if ~valido then disp("Libreta no valida")
else
    if modulo(libreta,2)==0 then mensaje="es par";
    else mensaje="es impar";
    end
    disp(mensaje)
end
v=[4 7 0 -2 13 18]; pares=0; impares=0;
for i=1:length(v)
    if modulo(v(i),2)==0 then pares=pares+1;
    else impares=impares+1;
    end
end
disp(pares); disp(impares)
```

   Octave. La condición `libreta==fix(libreta)` verifica que no haya parte fraccionaria.

```
valido=isnumeric(libreta) && isscalar(libreta) &&
   isfinite(libreta) ...
       && libreta==fix(libreta) && libreta>0;
if !valido, disp("Libreta no valida")
else
  if mod(libreta,2)==0, mensaje="es par";
  else, mensaje="es impar";
  endif
  disp(mensaje)
```

```
endif
v=[4 7 0 -2 13 18]; pares=0; impares=0;
for i=1:length(v)
  if mod(v(i),2)==0, pares=pares+1;
  else, impares=impares+1;
  endif
endfor
pares
impares
```

Python. Se acepta un valor numérico sin parte fraccionaria y mayor que cero.

```
valido=type(libreta) in (int,float) and
   int(libreta)==libreta and libreta>0
if not valido:
    print("Libreta no valida")
else:
    if libreta%2==0: mensaje="es par"
    else: mensaje="es impar"
    print(mensaje)
v=[4,7,0,-2,13,18]
pares=0; impares=0
for n in v:
    if n%2==0: pares=pares+1
    else: impares=impares+1
print(pares)
print(impares)
```

Perl. `looks_like_number` evita aplicar el resto a una cadena no numérica.

```
use Scalar::Util qw(looks_like_number);
$valido=looks_like_number($libreta) &&
   int($libreta)==$libreta && $libreta>0;
if (!$valido) { print "Libreta no valida\n"; }
else {
    if ($libreta%2==0) { $mensaje='es par'; }
    else { $mensaje='es impar'; }
    print $mensaje,"\n";
}
@v=(4,7,0,-2,13,18); $pares=0; $impares=0;
for $n (@v) {
    if ($n%2==0) { $pares=$pares+1; }
    else { $impares=$impares+1; }
}
print $pares,"\n";
print $impares,"\n";
```

JavaScript. `Number.isInteger` verifica directamente que el valor sea un entero.

```
const valido=Number.isInteger(libreta) && libreta>0
if(!valido){ console.log("Libreta no valida") }
else{
    let mensaje

    if(libreta%2===0){ mensaje="es par" }
    else{ mensaje="es impar" }
    console.log(mensaje)
}
const v=[4,7,0,-2,13,18]
let pares=0; let impares=0
for(let n of v){
    if(n%2===0){ pares=pares+1 }
    else{ impares=impares+1 }
}
console.log(pares)
console.log(impares)
```

8. **Ejercicio:** Mediciones con validación de entrada

   Escriba un programa que solicite mediciones reales una por una.

   La carga deberá finalizar cuando el usuario introduzca la palabra `fin`. Además, la palabra `cancelar` deberá permitir interrumpir anticipadamente la operación.

   Las palabras `fin` y `cancelar` son valores de control y no deben contarse como entradas rechazadas.

   No almacene las mediciones ingresadas en un vector. Mantenga solamente las variables necesarias para calcular:

   - cantidad total de datos válidos;
   - cantidad de entradas rechazadas;
   - media acumulada;
   - valor mínimo;
   - valor máximo;
   - cantidad de valores positivos;
   - cantidad de valores negativos;
   - cantidad de valores iguales a cero.

   Para actualizar la media sin conservar las mediciones anteriores, si después de incorporar un nuevo dato hay n valores válidos, utilice

   $$\overline{x}_n = \overline{x}_{n-1} + \frac{x_n - \overline{x}_{n-1}}{n}.$$

   El programa deberá:

a) inicializar las variables acumuladoras antes de comenzar la carga;

b) utilizar una repetición controlada por condición;

c) leer cada entrada inicialmente como una cadena, de modo que pueda distinguir un número de las palabras `fin` y `cancelar`;

d) validar la entrada antes de utilizarla como número;

e) informar inmediatamente cuando una entrada sea rechazada;

f) actualizar las cantidades acumuladas cada vez que se ingrese una medición válida;

g) inicializar el mínimo y el máximo con la primera medición válida, en lugar de asignarles de antemano un valor arbitrario;

h) si se utiliza `cancelar`, indicar que el resumen mostrado es parcial;

i) mostrar al finalizar la cantidad de datos válidos, la cantidad de entradas rechazadas y las cantidades de valores positivos, negativos y nulos;

j) si existe al menos un dato válido, mostrar además la media, el mínimo y el máximo;

k) tratar correctamente el caso en el que no se haya ingresado ninguna medición válida.

Escriba después una segunda versión suponiendo conocida previamente la cantidad N de mediciones válidas que se desea ingresar.

En esta segunda versión:

- suponga que `N` contiene un entero positivo;
- utilice un ciclo `for` para procesar exactamente N mediciones válidas;
- si una entrada no es válida, informe el error y vuelva a solicitar esa misma medición;
- no utilice los centinelas `fin` ni `cancelar`;
- calcule exactamente las mismas cantidades que en la primera versión.

Compare ambas soluciones. Indique por qué una repetición controlada por condición resulta apropiada cuando no se conoce de antemano la cantidad de datos, mientras que una repetición contada resulta natural cuando dicha cantidad es conocida.

Scilab. En Scilab, la entrada puede leerse como cadena mediante `input(...,"string")`. En esta solución se utiliza `isnum` para comprobar que la cadena representa un número y `strtod` para convertirla.

Primera versión:

```
cantidad=0;
rechazadas=0;
media=0;
positivos=0;
negativos=0;
ceros=0;
```

```
minimo=0;
maximo=0;

seguir=%t;
cancelado=%f;

while seguir

    s=input("Medicion (fin/cancelar): ","string");

    if s=="fin" then

        seguir=%f;

    elseif s=="cancelar" then

        seguir=%f;
        cancelado=%t;

    elseif ~isnum(s) then

        rechazadas=rechazadas+1;
        disp("Entrada rechazada")

    else

        x=strtod(s);
        cantidad=cantidad+1;

        if cantidad==1 then

            media=x;
            minimo=x;
            maximo=x;

        else

            media=media+(x-media)/cantidad;

            if x<minimo then
                minimo=x;
            end

            if x>maximo then
                maximo=x;
            end

        end

        if x>0 then
```

```
            positivos=positivos+1;

        elseif x<0 then

            negativos=negativos+1;

        else

            ceros=ceros+1;

        end

    end

end

if cancelado then
    disp("Operacion cancelada: resumen parcial")
end

disp("Cantidad de datos validos")
disp(cantidad)

disp("Entradas rechazadas")
disp(rechazadas)

disp("Valores positivos")
disp(positivos)

disp("Valores negativos")
disp(negativos)

disp("Valores iguales a cero")
disp(ceros)

if cantidad==0 then

    disp("No se ingresaron datos validos")

else

    disp("Media")
    disp(media)

    disp("Minimo")
    disp(minimo)

    disp("Maximo")
    disp(maximo)
```

```
end
```

Segunda versión:

```
cantidad=0;
rechazadas=0;
media=0;
positivos=0;
negativos=0;
ceros=0;

minimo=0;
maximo=0;

for k=1:N

    valido=%f;

    while ~valido

        s=input("Medicion: ","string");

        if isnum(s) then

            x=strtod(s);
            valido=%t;

        else

            rechazadas=rechazadas+1;
            disp("Entrada rechazada")

        end

    end

    cantidad=cantidad+1;

    if cantidad==1 then

        media=x;
        minimo=x;
        maximo=x;

    else

        media=media+(x-media)/cantidad;

        if x<minimo then
            minimo=x;
        end
```

```
            if x>maximo then
                maximo=x;
            end

        end

        if x>0 then

            positivos=positivos+1;

        elseif x<0 then

            negativos=negativos+1;

        else

            ceros=ceros+1;

        end

    end

    disp("Cantidad de datos validos")
    disp(cantidad)

    disp("Entradas rechazadas")
    disp(rechazadas)

    disp("Valores positivos")
    disp(positivos)

    disp("Valores negativos")
    disp(negativos)

    disp("Valores iguales a cero")
    disp(ceros)

    disp("Media")
    disp(media)

    disp("Minimo")
    disp(minimo)

    disp("Maximo")
    disp(maximo)
```

Octave. En Octave, la opción `"s"` de `input` permite leer una cadena. `str2double` realiza la conversión y `isfinite` permite comprobar que el resultado sea un número válido.

Primera versión:

```
cantidad=0;
rechazadas=0;
media=0;
positivos=0;
negativos=0;
ceros=0;

minimo=0;
maximo=0;

seguir=true;
cancelado=false;

while seguir

  s=input("Medicion (fin/cancelar): ","s");

  if strcmp(s,"fin")

    seguir=false;

  elseif strcmp(s,"cancelar")

    seguir=false;
    cancelado=true;

  else

    x=str2double(s);

    if !isfinite(x)

      rechazadas=rechazadas+1;
      disp("Entrada rechazada")

    else

      cantidad=cantidad+1;

      if cantidad==1

        media=x;
        minimo=x;
        maximo=x;

      else

        media=media+(x-media)/cantidad;
```

```
          if x<minimo
            minimo=x;
          endif

          if x>maximo
            maximo=x;
          endif

        endif

        if x>0

          positivos=positivos+1;

        elseif x<0

          negativos=negativos+1;

        else

          ceros=ceros+1;

        endif

      endif

    endif

endwhile

if cancelado
  disp("Operacion cancelada: resumen parcial")
endif

disp("Cantidad de datos validos")
disp(cantidad)

disp("Entradas rechazadas")
disp(rechazadas)

disp("Valores positivos")
disp(positivos)

disp("Valores negativos")
disp(negativos)

disp("Valores iguales a cero")
disp(ceros)

if cantidad==0
```

```
  disp("No se ingresaron datos validos")

else

  disp("Media")
  disp(media)

  disp("Minimo")
  disp(minimo)

  disp("Maximo")
  disp(maximo)

endif
```

Segunda versión:

```
cantidad=0;
rechazadas=0;
media=0;
positivos=0;
negativos=0;
ceros=0;

minimo=0;
maximo=0;

for k=1:N

  valido=false;

  while !valido

    s=input("Medicion: ","s");
    x=str2double(s);

    if isfinite(x)

      valido=true;

    else

      rechazadas=rechazadas+1;
      disp("Entrada rechazada")

    endif

  endwhile

  cantidad=cantidad+1;
```

```
    if cantidad==1

      media=x;
      minimo=x;
      maximo=x;

    else

      media=media+(x-media)/cantidad;

      if x<minimo
        minimo=x;
      endif

      if x>maximo
        maximo=x;
      endif

    endif

    if x>0

      positivos=positivos+1;

    elseif x<0

      negativos=negativos+1;

    else

      ceros=ceros+1;

    endif

  endfor

  disp("Cantidad de datos validos")
  disp(cantidad)

  disp("Entradas rechazadas")
  disp(rechazadas)

  disp("Valores positivos")
  disp(positivos)

  disp("Valores negativos")
  disp(negativos)

  disp("Valores iguales a cero")
  disp(ceros)
```

```
disp("Media")
disp(media)

disp("Minimo")
disp(minimo)

disp("Maximo")
disp(maximo)
```

Python. En Python se utiliza una expresión regular para comprobar que la cadena tenga la forma de un número antes de convertirla con `float`. Esto permite mantener la validación explícita sin introducir todavía manejo de excepciones.

Primera versión:

```
import re

patron=re.compile(
    r"[+-]?(?:\d+(?:\.\d*)?|\.\d+)(?:[eE][+-]?\d+)?"
)

cantidad=0
rechazadas=0
media=0
positivos=0
negativos=0
ceros=0

minimo=0
maximo=0

seguir=True
cancelado=False

while seguir:

    s=input("Medicion (fin/cancelar): ")

    if s=="fin":

        seguir=False

    elif s=="cancelar":

        seguir=False
        cancelado=True

    elif patron.fullmatch(s) is None:

        rechazadas=rechazadas+1
        print("Entrada rechazada")
```

```
    else:

        x=float(s)
        cantidad=cantidad+1

        if cantidad==1:

            media=x
            minimo=x
            maximo=x

        else:

            media=media+(x-media)/cantidad

            if x<minimo:
                minimo=x

            if x>maximo:
                maximo=x

        if x>0:

            positivos=positivos+1

        elif x<0:

            negativos=negativos+1

        else:

            ceros=ceros+1

if cancelado:
    print("Operacion cancelada: resumen parcial")

print("Cantidad de datos validos")
print(cantidad)

print("Entradas rechazadas")
print(rechazadas)

print("Valores positivos")
print(positivos)

print("Valores negativos")
print(negativos)

print("Valores iguales a cero")
print(ceros)
```

```
if cantidad==0:

    print("No se ingresaron datos validos")

else:

    print("Media")
    print(media)

    print("Minimo")
    print(minimo)

    print("Maximo")
    print(maximo)
```

Segunda versión:

```
import re

patron=re.compile(
    r"[+-]?(?:\d+(?:\.\d*)?|\.\d+)(?:[eE][+-]?\d+)?"
)

cantidad=0
rechazadas=0
media=0
positivos=0
negativos=0
ceros=0

minimo=0
maximo=0

for k in range(N):

    valido=False

    while not valido:

        s=input("Medicion: ")

        if patron.fullmatch(s) is not None:

            x=float(s)
            valido=True

        else:

            rechazadas=rechazadas+1
            print("Entrada rechazada")
```

```
        cantidad=cantidad+1

        if cantidad==1:

            media=x
            minimo=x
            maximo=x

        else:

            media=media+(x-media)/cantidad

            if x<minimo:
                minimo=x

            if x>maximo:
                maximo=x

        if x>0:

            positivos=positivos+1

        elif x<0:

            negativos=negativos+1

        else:

            ceros=ceros+1

    print("Cantidad de datos validos")
    print(cantidad)

    print("Entradas rechazadas")
    print(rechazadas)

    print("Valores positivos")
    print(positivos)

    print("Valores negativos")
    print(negativos)

    print("Valores iguales a cero")
    print(ceros)

    print("Media")
    print(media)

    print("Minimo")
    print(minimo)
```

```
print("Maximo")
print(maximo)
```

Perl. En Perl puede utilizarse `looks_like_number`, del módulo `Scalar::Util`, para comprobar que una cadena representa un número antes de utilizarla en operaciones aritméticas.

Primera versión:

```
use Scalar::Util qw(looks_like_number);

$cantidad=0;
$rechazadas=0;
$media=0;
$positivos=0;
$negativos=0;
$ceros=0;

$minimo=0;
$maximo=0;

$seguir=1;
$cancelado=0;

while ($seguir) {

    print "Medicion (fin/cancelar): ";
    $s=<STDIN>;
    chomp $s;

    if ($s eq 'fin') {

        $seguir=0;

    }
    elsif ($s eq 'cancelar') {

        $seguir=0;
        $cancelado=1;

    }
    elsif (!looks_like_number($s)) {

        $rechazadas=$rechazadas+1;
        print "Entrada rechazada\n";

    }
    else {

        $x=0+$s;
```

```
            $cantidad=$cantidad+1;

            if ($cantidad==1) {

                $media=$x;
                $minimo=$x;
                $maximo=$x;

            }
            else {

                $media=$media+($x-$media)/$cantidad;

                if ($x<$minimo) {
                    $minimo=$x;
                }

                if ($x>$maximo) {
                    $maximo=$x;
                }

            }

            if ($x>0) {

                $positivos=$positivos+1;

            }
            elsif ($x<0) {

                $negativos=$negativos+1;

            }
            else {

                $ceros=$ceros+1;

            }

        }

    }

    if ($cancelado) {
        print "Operacion cancelada: resumen parcial\n";
    }

    print "Cantidad de datos validos\n";
    print $cantidad,"\n";

    print "Entradas rechazadas\n";
```

```
print $rechazadas,"\n";

print "Valores positivos\n";
print $positivos,"\n";

print "Valores negativos\n";
print $negativos,"\n";

print "Valores iguales a cero\n";
print $ceros,"\n";

if ($cantidad==0) {

    print "No se ingresaron datos validos\n";

}
else {

    print "Media\n";
    print $media,"\n";

    print "Minimo\n";
    print $minimo,"\n";

    print "Maximo\n";
    print $maximo,"\n";

}
```

Segunda versión:

```
use Scalar::Util qw(looks_like_number);

$cantidad=0;
$rechazadas=0;
$media=0;
$positivos=0;
$negativos=0;
$ceros=0;

$minimo=0;
$maximo=0;

for ($k=1; $k<=$N; $k++) {

    $valido=0;

    while (!$valido) {

        print "Medicion: ";
        $s=<STDIN>;
```

```
        chomp $s;

        if (looks_like_number($s)) {

            $x=0+$s;
            $valido=1;

        }
        else {

            $rechazadas=$rechazadas+1;
            print "Entrada rechazada\n";

        }

    }

    $cantidad=$cantidad+1;

    if ($cantidad==1) {

        $media=$x;
        $minimo=$x;
        $maximo=$x;

    }
    else {

        $media=$media+($x-$media)/$cantidad;

        if ($x<$minimo) {
            $minimo=$x;
        }

        if ($x>$maximo) {
            $maximo=$x;
        }

    }

    if ($x>0) {

        $positivos=$positivos+1;

    }
    elsif ($x<0) {

        $negativos=$negativos+1;

    }
    else {
```

```
        $ceros=$ceros+1;

    }

}

print "Cantidad de datos validos\n";
print $cantidad,"\n";

print "Entradas rechazadas\n";
print $rechazadas,"\n";

print "Valores positivos\n";
print $positivos,"\n";

print "Valores negativos\n";
print $negativos,"\n";

print "Valores iguales a cero\n";
print $ceros,"\n";

print "Media\n";
print $media,"\n";

print "Minimo\n";
print $minimo,"\n";

print "Maximo\n";
print $maximo,"\n";
```

JavaScript. Node.js no posee una operación de entrada sincrónica equivalente a `input` como parte de su interfaz básica. Para mantener el mismo algoritmo secuencial se utiliza aquí `readline-sync`. Puede instalarse una vez mediante `npm install readline-sync`.

Primera versión:

```
const rl=require("readline-sync")

let cantidad=0
let rechazadas=0
let media=0
let positivos=0
let negativos=0
let ceros=0

let minimo=0
let maximo=0

let seguir=true
```

```
let cancelado=false

while(seguir){

    const s=rl.question(
        "Medicion (fin/cancelar): "
    )

    if(s==="fin"){

        seguir=false

    }
    else if(s==="cancelar"){

        seguir=false
        cancelado=true

    }
    else{

        const x=Number(s)

        if(s==="" || !Number.isFinite(x)){

            rechazadas=rechazadas+1
            console.log("Entrada rechazada")

        }
        else{

            cantidad=cantidad+1

            if(cantidad===1){

                media=x
                minimo=x
                maximo=x

            }
            else{

                media=media+(x-media)/cantidad

                if(x<minimo){
                    minimo=x
                }

                if(x>maximo){
                    maximo=x
                }
```

```
            }

            if(x>0){

                positivos=positivos+1

            }
            else if(x<0){

                negativos=negativos+1

            }
            else{

                ceros=ceros+1

            }

        }

    }

}

if(cancelado){
    console.log("Operacion cancelada: resumen parcial")
}

console.log("Cantidad de datos validos")
console.log(cantidad)

console.log("Entradas rechazadas")
console.log(rechazadas)

console.log("Valores positivos")
console.log(positivos)

console.log("Valores negativos")
console.log(negativos)

console.log("Valores iguales a cero")
console.log(ceros)

if(cantidad===0){

    console.log("No se ingresaron datos validos")

}
else{
```

```
    console.log("Media")
    console.log(media)

    console.log("Minimo")
    console.log(minimo)

    console.log("Maximo")
    console.log(maximo)

}
```

Segunda versión:

```
const rl=require("readline-sync")

let cantidad=0
let rechazadas=0
let media=0
let positivos=0
let negativos=0
let ceros=0

let minimo=0
let maximo=0

let x

for(let k=1;k<=N;k++){

    let valido=false

    while(!valido){

        const s=rl.question("Medicion: ")
        x=Number(s)

        if(s!=="" && Number.isFinite(x)){

            valido=true

        }
        else{

            rechazadas=rechazadas+1
            console.log("Entrada rechazada")

        }

    }

    cantidad=cantidad+1
```

```
    if(cantidad===1){

        media=x
        minimo=x
        maximo=x

    }
    else{

        media=media+(x-media)/cantidad

        if(x<minimo){
            minimo=x
        }

        if(x>maximo){
            maximo=x
        }

    }

    if(x>0){

        positivos=positivos+1

    }
    else if(x<0){

        negativos=negativos+1

    }
    else{

        ceros=ceros+1

    }

}

console.log("Cantidad de datos validos")
console.log(cantidad)

console.log("Entradas rechazadas")
console.log(rechazadas)

console.log("Valores positivos")
console.log(positivos)

console.log("Valores negativos")
console.log(negativos)
```

```
console.log("Valores iguales a cero")
console.log(ceros)

console.log("Media")
console.log(media)

console.log("Minimo")
console.log(minimo)

console.log("Maximo")
console.log(maximo)
```

9. **Ejercicio:** Selección múltiple y ternario

   Un sistema registra el estado de funcionamiento de un equipo mediante un código entero:

   $$1 = \text{apagado}, \qquad 2 = \text{en espera}, \qquad 3 = \text{funcionando}, \qquad 4 = \text{alarma}.$$

   Antes de ejecutar el programa, defina las variables `estado` y `temperatura`.

   El programa deberá:

   a) validar que `estado` sea un entero comprendido entre 1 y 4;

   b) utilizar una estructura de selección múltiple, mediante `case` o el mecanismo equivalente del lenguaje, para asignar a una variable `descripcion` una de las cadenas:

      - “apagado”;
      - “en espera”;
      - “funcionando”;
      - “alarma”.

   c) mostrar la descripción obtenida;

   d) utilizar una expresión condicional abreviada para asignar a una variable `temperatura_estado` uno de los dos valores

      $$\text{“normal”} \qquad \text{o} \qquad \text{“alta”},$$

      considerando alta una temperatura mayor o igual que 70 °C;

   e) mostrar el resultado;

   f) probar el programa con diferentes combinaciones de estado y temperatura;

   g) escribir nuevamente la decisión sobre la temperatura mediante una estructura `if-else` y comparar ambas formas.

   En los lenguajes que disponen del operador condicional ternario, la forma general es

```
condicion ? valor_verdadero : valor_falso.
```

Otros lenguajes disponen de una expresión equivalente con una sintaxis diferente.

Scilab. En Scilab, la selección múltiple se realiza con `select` y `case`. Scilab no utiliza la sintaxis `?:`; para la segunda decisión se utiliza `if-else`, lo que permite compararlo con los lenguajes que poseen una expresión condicional abreviada.

```
select estado
case 1 then
    descripcion="apagado";
case 2 then
    descripcion="en espera";
case 3 then
    descripcion="funcionando";
case 4 then
    descripcion="alarma";
else
    descripcion="estado no valido";
end

disp(descripcion)

if temperatura>=70 then
    temperatura_estado="alta";
else
    temperatura_estado="normal";
end

disp(temperatura_estado)
```

Octave. En Octave, la selección múltiple se realiza mediante `switch` y `case`. Para una decisión entre dos valores puede utilizarse una expresión condicional mediante `ifelse`.

```
switch estado
  case 1
    descripcion="apagado";
  case 2
    descripcion="en espera";
  case 3
    descripcion="funcionando";
  case 4
    descripcion="alarma";
  otherwise
    descripcion="estado no valido";
endswitch

disp(descripcion)
```

```
temperatura_estado=ifelse(
    temperatura>=70,
    "alta",
    "normal"
);

disp(temperatura_estado)
```

Python. Python dispone de selección múltiple mediante `match-case`. Su expresión condicional abreviada escribe primero el valor correspondiente al caso verdadero.

```
match estado:
    case 1:
        descripcion="apagado"
    case 2:
        descripcion="en espera"
    case 3:
        descripcion="funcionando"
    case 4:
        descripcion="alarma"
    case _:
        descripcion="estado no valido"

print(descripcion)

temperatura_estado=(
    "alta" if temperatura>=70 else "normal"
)

print(temperatura_estado)
```

Perl. En Perl, para este ejercicio la selección de varias alternativas puede expresarse mediante una cadena de decisiones. El operador ternario utiliza la forma `condicion ? valor1 : valor2`.

```
if ($estado==1) {
    $descripcion='apagado';
}
elsif ($estado==2) {
    $descripcion='en espera';
}
elsif ($estado==3) {
    $descripcion='funcionando';
}
elsif ($estado==4) {
    $descripcion='alarma';
}
else {
```

```
    $descripcion='estado no valido';
}

print $descripcion,"\n";

$temperatura_estado=
    $temperatura>=70 ? 'alta' : 'normal';

print $temperatura_estado,"\n";
```

JavaScript. JavaScript dispone tanto de `switch-case` como del operador condicional ternario `?:`.

```
let descripcion

switch(estado){
case 1:
    descripcion="apagado"
    break

case 2:
    descripcion="en espera"
    break

case 3:
    descripcion="funcionando"
    break

case 4:
    descripcion="alarma"
    break

default:
    descripcion="estado no valido"
}

console.log(descripcion)

const temperaturaEstado=
    temperatura>=70 ? "alta" : "normal"

console.log(temperaturaEstado)
```

5.11.4. Funciones, alcance y modularización

10. **Ejercicio:** Doble con validación de tipos

    Escriba una función denominada `doble` que reciba un valor numérico `x` y devuelva

$$2x.$$

La función deberá aceptar, utilizando la misma definición:

- un número escalar;
- un vector;
- una matriz.

Luego:

a) Pruebe la función con números enteros y reales.

b) Si el lenguaje o el entorno numérico utilizado admite números complejos, pruébela también con un número complejo.

c) Pruebe la función con un vector.

d) Pruebe la función con una matriz.

e) Determine si fue necesario modificar la función para trabajar con vectores y matrices.

f) Agregue documentación que indique qué recibe la función, qué devuelve y qué entradas admite.

g) Haga que la función rechace una entrada no numérica con un mensaje de error claro.

Compare el comportamiento de los distintos lenguajes. En particular, observe cuáles permiten aplicar directamente la multiplicación por un escalar a un vector o una matriz y cuáles requieren utilizar una representación numérica específica.

Scilab. En Scilab, escalares, vectores y matrices numéricas utilizan el mismo tipo básico. La multiplicación de una matriz por un escalar se escribe directamente como `2*x`.

```
function y=doble(x)
    // doble(x)
    // Recibe un escalar, vector o matriz numerica.
    // Devuelve cada elemento multiplicado por 2.

    if type(x)<>1 then
        error("La entrada debe ser numerica")
    end

    y=2*x;
endfunction

doble(5)
doble(3.5)
doble([1 2 3])
doble([1 2;3 4])
doble(2+3*%i)
```

Octave. En Octave, la misma operación escalar puede aplicarse directamente a escalares, vectores y matrices. Los números complejos pertenecen también a los tipos numéricos.

```
function y=doble(x)
  % doble(x)
  % Recibe un escalar, vector o matriz numerica.
  % Devuelve cada elemento multiplicado por 2.

  if !isnumeric(x)
    error("La entrada debe ser numerica")
  endif

  y=2*x;
endfunction

doble(5)
doble(3.5)
doble([1 2 3])
doble([1 2;3 4])
doble(2+3i)
```

Python. En Python se utiliza NumPy para que escalares, vectores y matrices puedan tratarse con el mismo procedimiento. Una lista se convierte primero en un arreglo numérico.

```
import numpy as np

def doble(x):
    """
    Recibe un escalar, vector o matriz numerica.
    Devuelve cada elemento multiplicado por 2.
    """

    a=np.asarray(x)

    if not np.issubdtype(a.dtype,np.number):
        raise TypeError("La entrada debe ser numerica")

    return 2*a

print(doble(5))
print(doble(3.5))
print(doble([1,2,3]))
print(doble([[1,2],[3,4]]))
print(doble(2+3j))
```

Perl. En Perl se utiliza PDL para representar vectores y matrices numéricas. Los escalares numéricos se tratan directamente. Debido a la naturaleza de los escalares

de Perl, la distinción entre una cadena que representa un número y un número es menos estricta que en otros lenguajes.

```
use PDL;
use Scalar::Util qw(looks_like_number blessed);

sub doble {
    my ($x)=@_;

    # Un piddle puede ser vector o matriz.
    if (blessed($x) && $x->isa('PDL')) {
        return 2*$x;
    }

    die "La entrada debe ser numerica\n"
        unless looks_like_number($x);

    return 2*$x;
}

print doble(5),"\n";
print doble(3.5),"\n";

$v=pdl([1,2,3]);
$m=pdl([[1,2],[3,4]]);

print doble($v),"\n";
print doble($m),"\n";
```

JavaScript. JavaScript no aplica directamente una multiplicación escalar a un arreglo. Por eso se conserva el mismo significado de la función recorriendo recursivamente los arreglos; de este modo la misma función acepta escalares, vectores y matrices.

```
function doble(x){

    // Escalar numerico.
    if(typeof x==="number" && Number.isFinite(x)){
        return 2*x
    }

    // Vector o matriz representados mediante arreglos.
    if(Array.isArray(x)){
        return x.map(doble)
    }

    throw new TypeError(
        "La entrada debe ser numerica"
    )
```

```
}

console.log(doble(5))
console.log(doble(3.5))
console.log(doble([1,2,3]))
console.log(doble([[1,2],[3,4]]))
```

La diferencia de JavaScript es genuina: con arreglos nativos necesita recorrerlos; Scilab, Octave, NumPy y PDL disponen de operaciones numéricas sobre sus representaciones vectoriales/matriciales.

11. **Ejercicio:** Conversión de temperaturas

    Escriba una función denominada `celsius_a_fahrenheit` que reciba una temperatura en grados Celsius y devuelva su equivalente en grados Fahrenheit mediante

    $$F = \frac{9}{5}C + 32.$$

    Escriba también una función denominada `fahrenheit_a_celsius` que realice la conversión inversa:

    $$C = \frac{5}{9}(F - 32).$$

    Ambas funciones deberán aceptar:

    - un número real escalar;
    - un vector de números reales;
    - una matriz de números reales.

    Una entrada no numérica deberá ser rechazada con un mensaje de error.

    Luego:

    a) Verifique las conversiones conocidas

    $$0\,^\circ\mathrm{C} = 32\,^\circ\mathrm{F}, \qquad 100\,^\circ\mathrm{C} = 212\,^\circ\mathrm{F}.$$

    b) Verifique también que

    $$-40\,^\circ\mathrm{C} = -40\,^\circ\mathrm{F}.$$

    c) Convierta el vector

    $$C = [-40,\ 0,\ 20,\ 100]$$

    a grados Fahrenheit.

    d) Aplique después la conversión inversa al resultado y compruebe que se recuperan las temperaturas originales.

e) Calcule el error máximo de la composición mediante

$$e = \max_i |C_i - \text{fahrenheit_a_celsius}\,(\text{celsius_a_fahrenheit}(C_i))|\,.$$

f) Repita la prueba con una matriz de temperaturas.

g) Determine si fue necesario modificar las funciones para trabajar con vectores y matrices.

h) Agregue documentación a ambas funciones.

i) Pruebe una entrada no numérica y verifique que sea rechazada.

Scilab. En Scilab, la multiplicación por un escalar y la suma de un escalar se aplican directamente a todos los elementos de un vector o matriz.

```
function F=celsius_a_fahrenheit(C)

    if type(C)<>1 then
        error("La entrada debe ser numerica")
    end

    if ~isreal(C) then
        error("La temperatura debe ser real")
    end

    F=(9/5)*C+32;

endfunction


function C=fahrenheit_a_celsius(F)

    if type(F)<>1 then
        error("La entrada debe ser numerica")
    end

    if ~isreal(F) then
        error("La temperatura debe ser real")
    end

    C=(5/9)*(F-32);

endfunction


celsius_a_fahrenheit(0)
celsius_a_fahrenheit(100)
celsius_a_fahrenheit(-40)

C=[-40 0 20 100];
```

```
F=celsius_a_fahrenheit(C);

C_recuperado=fahrenheit_a_celsius(F);

error_maximo=max(abs(C-C_recuperado))


M=[0 10;20 100];

FM=celsius_a_fahrenheit(M);

M_recuperada=fahrenheit_a_celsius(FM);
```

Octave. En Octave, las mismas expresiones trabajan directamente con escalares, vectores y matrices.

```
function F=celsius_a_fahrenheit(C)

  if !isnumeric(C)
    error("La entrada debe ser numerica")
  endif

  if !isreal(C)
    error("La temperatura debe ser real")
  endif

  F=(9/5)*C+32;

endfunction


function C=fahrenheit_a_celsius(F)

  if !isnumeric(F)
    error("La entrada debe ser numerica")
  endif

  if !isreal(F)
    error("La temperatura debe ser real")
  endif

  C=(5/9)*(F-32);

endfunction


celsius_a_fahrenheit(0)
celsius_a_fahrenheit(100)
celsius_a_fahrenheit(-40)
```

```
C=[-40 0 20 100];

F=celsius_a_fahrenheit(C);

C_recuperado=fahrenheit_a_celsius(F);

error_maximo=max(abs(C-C_recuperado))


M=[0 10;20 100];

FM=celsius_a_fahrenheit(M);

M_recuperada=fahrenheit_a_celsius(FM);
```

Python. En Python se utiliza NumPy para que las mismas funciones puedan trabajar con escalares, vectores y matrices.

```
import numpy as np


def celsius_a_fahrenheit(C):

    a=np.asarray(C)

    if not np.issubdtype(a.dtype,np.number):
        raise TypeError("La entrada debe ser numerica")

    if not np.isrealobj(a):
        raise ValueError("La temperatura debe ser real")

    return (9/5)*a+32


def fahrenheit_a_celsius(F):

    a=np.asarray(F)

    if not np.issubdtype(a.dtype,np.number):
        raise TypeError("La entrada debe ser numerica")

    if not np.isrealobj(a):
        raise ValueError("La temperatura debe ser real")

    return (5/9)*(a-32)


print(celsius_a_fahrenheit(0))
print(celsius_a_fahrenheit(100))
print(celsius_a_fahrenheit(-40))
```

```
C=np.array([-40,0,20,100])

F=celsius_a_fahrenheit(C)

C_recuperado=fahrenheit_a_celsius(F)

error_maximo=np.max(
    np.abs(C-C_recuperado)
)

print(error_maximo)


M=np.array([
    [0,10],
    [20,100]
])

FM=celsius_a_fahrenheit(M)

M_recuperada=fahrenheit_a_celsius(FM)
```

Perl. En Perl se utiliza PDL para representar vectores y matrices numéricas. Las dos funciones aplican las mismas expresiones a escalares y *piddles*.

```
use PDL;
use Scalar::Util qw(looks_like_number blessed);


sub celsius_a_fahrenheit {

    my ($C)=@_;

    if (blessed($C) && $C->isa('PDL')) {
        return (9/5)*$C+32;
    }

    die "La entrada debe ser numerica\n"
        unless looks_like_number($C);

    return (9/5)*$C+32;
}


sub fahrenheit_a_celsius {

    my ($F)=@_;

    if (blessed($F) && $F->isa('PDL')) {
```

```
        return (5/9)*($F-32);
    }

    die "La entrada debe ser numerica\n"
        unless looks_like_number($F);

    return (5/9)*($F-32);
}


print celsius_a_fahrenheit(0),"\n";
print celsius_a_fahrenheit(100),"\n";
print celsius_a_fahrenheit(-40),"\n";


$C=pdl([-40,0,20,100]);

$F=celsius_a_fahrenheit($C);

$C_recuperado=
    fahrenheit_a_celsius($F);

$error_maximo=
    max(abs($C-$C_recuperado));

print $error_maximo,"\n";


$M=pdl([
    [0,10],
    [20,100]
]);

$FM=celsius_a_fahrenheit($M);

$M_recuperada=
    fahrenheit_a_celsius($FM);
```

JavaScript. JavaScript no aplica directamente operaciones aritméticas a los elementos de un arreglo. Por eso se utiliza una función auxiliar que aplica la conversión recursivamente a escalares, vectores y matrices.

```
function aplicar(x,conversion){

    if(typeof x==="number" && Number.isFinite(x)){
        return conversion(x)
    }

    if(Array.isArray(x)){
        return x.map(
```

```
            v=>aplicar(v,conversion)
        )
    }

    throw new TypeError(
        "La entrada debe ser numerica"
    )
}


function celsiusAFahrenheit(C){

    return aplicar(
        C,
        x=>(9/5)*x+32
    )

}


function fahrenheitACelsius(F){

    return aplicar(
        F,
        x=>(5/9)*(x-32)
    )

}


console.log(
    celsiusAFahrenheit(0)
)

console.log(
    celsiusAFahrenheit(100)
)

console.log(
    celsiusAFahrenheit(-40)
)


const C=[-40,0,20,100]

const F=celsiusAFahrenheit(C)

const Crecuperado=fahrenheitACelsius(F)


let errorMaximo=0
```

```
for(let i=0;i<C.length;i++){

    const error=Math.abs(
        C[i]-Crecuperado[i]
    )

    if(error>errorMaximo){
        errorMaximo=error
    }

}

console.log(errorMaximo)


const M=[
    [0,10],
    [20,100]
]

const FM=celsiusAFahrenheit(M)

const Mrecuperada=fahrenheitACelsius(FM)
```

12. **Ejercicio:** Módulo de paridad

    Escriba una función denominada `paridad` cuyo parámetro de entrada sea un número entero y cuya salida sea una de las cadenas:

    - "es par";
    - "es impar".

    La función deberá comprobar que la entrada sea un número entero. Si la entrada no cumple esa condición, deberá producir un mensaje de error.

    Guarde la función en un archivo fuente y utilícela posteriormente desde una sesión nueva del intérprete.

    Luego:

    a) Pruebe la función con enteros positivos, cero y enteros negativos.
    b) Pruebe una entrada no entera y compruebe que sea rechazada.
    c) Pruebe una entrada no numérica y compruebe que sea rechazada.
    d) Cierre la sesión del intérprete, abra una sesión nueva y cargue, importe o haga accesible el archivo mediante el mecanismo propio del lenguaje.
    e) Utilice nuevamente la función sin volver a escribir su definición.
    f) Escriba una segunda función denominada `paridades` que reciba un vector o colección de números enteros y devuelva una colección con el mensaje correspondiente a cada elemento.

g) En `paridades`, reutilice la función escalar `paridad`; no repita dentro de ella el cálculo de paridad.

h) Pruebe la segunda función con

$$[4,\ 7,\ 0,\ -2,\ 13,\ 18].$$

i) Compruebe qué ocurre si alguno de los elementos de la colección no es entero.

j) Explique la diferencia entre definir una función solamente en una sesión interactiva y almacenarla en un archivo que puede reutilizarse posteriormente.

El algoritmo de la función escalar será el mismo en todos los lenguajes:

```
validar n

si resto(n,2) es cero
    devolver "es par"
si no
    devolver "es impar"
```

La función `paridades` deberá recorrer la colección y aplicar `paridad` a cada elemento.

Scilab. En Scilab, las funciones pueden almacenarse en un archivo `.sci`. El archivo se carga mediante `exec`. En este caso las dos funciones pueden almacenarse en el mismo archivo.

Archivo `paridad.sci`:

```
function mensaje=paridad(n)

    valido=%f;

    if type(n)==1 then
        if size(n,"*")==1 then
            if isreal(n) then
                valido=(n==floor(n));
            end
        end
    end

    if ~valido then
        error("La entrada debe ser un numero entero")
    end

    if modulo(n,2)==0 then
        mensaje="es par";
    else
        mensaje="es impar";
    end
```

```
endfunction


function mensajes=paridades(v)

    if type(v)<>1 then
        error("La entrada debe ser un vector numerico")
    end

    if ~or(size(v)==1) then
        error("La entrada debe ser un vector")
    end

    mensajes=emptystr(size(v,1),size(v,2));

    for i=1:length(v)
        mensajes(i)=paridad(v(i));
    end

endfunction
```

Desde una sesión nueva:

```
exec("paridad.sci",-1)

paridad(8)
paridad(7)
paridad(0)
paridad(-3)

v=[4 7 0 -2 13 18];

paridades(v)
```

Octave. En Octave, una función pública se guarda normalmente en un archivo `.m` con el mismo nombre que la función. Por eso se utilizan aquí dos archivos. Si se encuentran en el directorio actual o en la ruta de búsqueda, Octave los localiza al invocarlos; no es necesario un `import` explícito. Esta es una diferencia propia del mecanismo de archivos de funciones de Octave.

Archivo `paridad.m`:

```
function mensaje=paridad(n)

  valido=isnumeric(n) ...
         && isscalar(n) ...
         && isreal(n) ...
         && isfinite(n) ...
         && n==fix(n);
```

```
  if !valido
    error("La entrada debe ser un numero entero")
  endif

  if mod(n,2)==0
    mensaje="es par";
  else
    mensaje="es impar";
  endif

endfunction
```

Archivo `paridades.m`:

```
function mensajes=paridades(v)

  if !isnumeric(v) || !isvector(v)
    error("La entrada debe ser un vector numerico")
  endif

  mensajes=cell(size(v));

  for i=1:numel(v)
    mensajes{i}=paridad(v(i));
  endfor

endfunction
```

Desde una sesión nueva, estando los archivos en el directorio actual:

```
paridad(8)
paridad(7)
paridad(0)
paridad(-3)

v=[4 7 0 -2 13 18];

paridades(v)
```

Python. En Python, ambas funciones pueden almacenarse en un mismo módulo `.py` y luego importarse desde otra sesión. La función `paridades` utiliza explícitamente un ciclo y reutiliza `paridad`.

Archivo `paridad.py`:

```
import math


def paridad(n):
```

```
    valido=(
        isinstance(n,(int,float))
        and not isinstance(n,bool)
        and math.isfinite(n)
        and n==int(n)
    )

    if not valido:
        raise ValueError(
            "La entrada debe ser un numero entero"
        )

    if n%2==0:
        return "es par"
    else:
        return "es impar"


def paridades(v):

    if not isinstance(v,(list,tuple)):
        raise TypeError(
            "La entrada debe ser una coleccion"
        )

    mensajes=[]

    for n in v:
        mensajes.append(paridad(n))

    return mensajes
```

Desde una sesión nueva:

```
from paridad import paridad,paridades

print(paridad(8))
print(paridad(7))
print(paridad(0))
print(paridad(-3))

v=[4,7,0,-2,13,18]

print(paridades(v))
```

Perl. En Perl, las funciones reutilizables pueden agruparse en un módulo `.pm`. En este ejemplo se exportan explícitamente `paridad` y `paridades`. La representación vectorial utilizada es una referencia a un arreglo.

Archivo `Paridad.pm`:

```
package Paridad;

use strict;
use warnings;
use Exporter 'import';
use Scalar::Util qw(looks_like_number);

our @EXPORT_OK=qw(paridad paridades);


sub paridad {

    my ($n)=@_;

    my $valido=
        looks_like_number($n)
        && int($n)==$n;

    die "La entrada debe ser un numero entero\n"
        unless $valido;

    if ($n%2==0) {
        return 'es par';
    }
    else {
        return 'es impar';
    }

}


sub paridades {

    my ($v)=@_;

    die "La entrada debe ser un arreglo\n"
        unless ref($v) eq 'ARRAY';

    my @mensajes;

    for my $n (@$v) {
        push @mensajes,paridad($n);
    }

    return \@mensajes;

}


1;
```

Desde una sesión nueva:

```
use strict;
use warnings;
use lib '.';
use Paridad qw(paridad paridades);

print paridad(8),"\n";
print paridad(7),"\n";
print paridad(0),"\n";
print paridad(-3),"\n";

my @v=(4,7,0,-2,13,18);

my $mensajes=paridades(\@v);

for my $mensaje (@$mensajes) {
    print $mensaje,"\n";
}
```

JavaScript. En Node.js, ambas funciones pueden almacenarse en un módulo JavaScript y exportarse mediante `module.exports`. Desde otro archivo o desde una sesión apropiada pueden recuperarse mediante `require`.

Archivo `paridad.js`:

```
function paridad(n){

    if(!Number.isInteger(n)){
        throw new TypeError(
            "La entrada debe ser un numero entero"
        )
    }

    if(n%2===0){
        return "es par"
    }
    else{
        return "es impar"
    }

}


function paridades(v){

    if(!Array.isArray(v)){
        throw new TypeError(
```

```
            "La entrada debe ser un arreglo"
        )
    }

    const mensajes=[]

    for(const n of v){
        mensajes.push(paridad(n))
    }

    return mensajes

}


module.exports={
    paridad,
    paridades
}
```

Desde otro archivo o una nueva sesión:

```
const {
    paridad,
    paridades
}=require("./paridad")

console.log(paridad(8))
console.log(paridad(7))
console.log(paridad(0))
console.log(paridad(-3))

const v=[4,7,0,-2,13,18]

console.log(paridades(v))
```

13. **Ejercicio:** Alcance de variables

    Considere el siguiente algoritmo:

```
x <- 10

definir doble(x):
    auxiliar <- x * 2
    devolver auxiliar

mostrar x

y <- doble(x)
```

```
mostrar x
mostrar y
```

Antes de ejecutarlo, prediga los tres valores que serán mostrados.

Luego:

a) Implemente el algoritmo en el lenguaje elegido y compare el resultado con su predicción.

b) Compruebe que el parámetro denominado `x` dentro de `doble` es distinto de la variable `x` existente fuera de la función.

c) Compruebe que la llamada a `doble(x)` no modifica la variable exterior `x`.

d) Cree dentro de la función una variable denominada `auxiliar`.

e) Intente utilizar `auxiliar` después de finalizar la llamada a la función y observe qué ocurre.

f) Cambie el nombre del parámetro de `x` a `valor` y verifique que el resultado no cambia.

g) Investigue el mecanismo que ofrece el lenguaje para que una función pueda acceder y modificar deliberadamente una variable definida en un ámbito exterior.

h) Realice un pequeño ejemplo en el que una función incremente una variable exterior denominada `contador`.

i) Compare ese mecanismo con el uso de parámetros y valores de retorno.

j) Explique por qué conviene evitar el uso innecesario de variables globales o de estado compartido.

En la primera parte, los tres resultados esperados son

$$10, \qquad 10, \qquad 20.$$

La variable `x` exterior no debe ser modificada por la función `doble`. La variable `auxiliar` debe existir solamente durante la ejecución de la función.

La última parte tiene por objeto observar una diferencia entre los lenguajes: algunos requieren declarar explícitamente una variable como global, mientras que otros permiten que una función acceda a variables pertenecientes a un ámbito exterior mediante otros mecanismos.

Scilab. En Scilab, los parámetros y las variables creadas dentro de una función pertenecen al ámbito de esa función. Para compartir deliberadamente una variable puede utilizarse `global`.

```
x=10;

function resultado=doble(x)

    auxiliar=x*2;
```

```
    resultado=auxiliar;

endfunction

disp(x)

y=doble(x);

disp(x)
disp(y)

// auxiliar no existe aqui.
```

Cambie ahora el nombre del parámetro:

```
function resultado=doble(valor)

    auxiliar=valor*2;
    resultado=auxiliar;

endfunction

y=doble(x);

disp(x)
disp(y)
```

Ejemplo de variable compartida:

```
global contador

contador=0;

function incrementar()
    global contador
    contador=contador+1;
endfunction

incrementar();
incrementar();

disp(contador)
```

Octave. En Octave, los parámetros y las variables creadas dentro de una función son locales. Una variable global debe declararse explícitamente con `global` tanto fuera como dentro de la función que la utiliza.

```
x=10;
```

```
function resultado=doble(x)

  auxiliar=x*2;
  resultado=auxiliar;

endfunction

disp(x)

y=doble(x);

disp(x)
disp(y)

% auxiliar no existe aqui.
```

Cambie ahora el nombre del parámetro:

```
function resultado=doble(valor)

  auxiliar=valor*2;
  resultado=auxiliar;

endfunction

y=doble(x);

disp(x)
disp(y)
```

Ejemplo de variable compartida:

```
global contador

contador=0;

function incrementar()

  global contador

  contador=contador+1;

endfunction

incrementar();
incrementar();

disp(contador)
```

Python. En Python, los parámetros y las variables asignadas dentro de una función pertenecen a su ámbito local. Para modificar desde una función una variable del ámbito global se utiliza la declaración `global`.

```
x=10

def doble(x):

    auxiliar=x*2
    return auxiliar

print(x)

y=doble(x)

print(x)
print(y)

# auxiliar no existe aqui.
```

Cambie ahora el nombre del parámetro:

```
def doble(valor):

    auxiliar=valor*2
    return auxiliar

y=doble(x)

print(x)
print(y)
```

Ejemplo de variable compartida:

```
contador=0

def incrementar():

    global contador

    contador=contador+1

incrementar()
incrementar()

print(contador)
```

Perl. En Perl, las variables declaradas mediante `my` tienen alcance léxico. Para este ejercicio se utilizan variables léxicas en las funciones. Una variable de paquete

declarada mediante `our` puede utilizarse como estado compartido.

```
use strict;
use warnings;

my $x=10;

sub doble {

    my ($x)=@_;

    my $auxiliar=$x*2;

    return $auxiliar;
}

print $x,"\n";

my $y=doble($x);

print $x,"\n";
print $y,"\n";

# $auxiliar no existe aqui.
```

Cambie ahora el nombre del parámetro:

```
sub doble {

    my ($valor)=@_;

    my $auxiliar=$valor*2;

    return $auxiliar;
}

my $y=doble($x);

print $x,"\n";
print $y,"\n";
```

Ejemplo de variable compartida:

```
our $contador=0;

sub incrementar {

    $contador=$contador+1;
```

```
}

incrementar();
incrementar();

print $contador,"\n";
```

JavaScript. En JavaScript, los parámetros y las variables declaradas dentro de una función son locales a esa función. Una función puede acceder a variables declaradas en un ámbito exterior mediante un cierre o *closure*; no se utiliza una declaración equivalente a `global` para este caso.

```
let x=10

function doble(x){

    let auxiliar=x*2

    return auxiliar

}

console.log(x)

let y=doble(x)

console.log(x)
console.log(y)

// auxiliar no existe aqui.
```

Cambie ahora el nombre del parámetro:

```
function doble(valor){

    let auxiliar=valor*2

    return auxiliar

}

let y=doble(x)

console.log(x)
console.log(y)
```

Ejemplo de variable compartida mediante el ámbito exterior:

```
let contador=0
```

```
function incrementar(){

    contador=contador+1

}

incrementar()
incrementar()

console.log(contador)
```

Acá la diferencia de JavaScript es intencional y propia del lenguaje: el mismo objetivo de compartir contador se consigue mediante el ámbito léxico/cierre, mientras Scilab, Octave y Python ofrecen una declaración global, y Perl puede utilizar una variable de paquete.

14. **Ejercicio:** Normalización con contrato

    Escriba una función denominada `normalizar` con la interfaz conceptual

    ```
    normalizar(v, metodo="minmax") -> y
    ```

    donde:

    - `v` debe ser un vector numérico real, finito y no vacío;
    - `metodo` es un argumento opcional;
    - si no se especifica `metodo`, se utilizará "`minmax`";
    - los valores permitidos de `metodo` son "`max`", "`minmax`" y "`zscore`";
    - `y` es un vector de las mismas dimensiones que `v`.

    Los métodos se definen de la siguiente manera.

    Para `"max"`:

    $$s = \max_i |v_i|,$$

    y, si $s \neq 0$,

    $$y_i = \frac{v_i}{s}.$$

    Si $s = 0$, devuelva un vector de ceros.

    Para "`minmax`":

    $$a = \min_i v_i, \qquad b = \max_i v_i,$$

    y, si $a \neq b$,

$$y_i = \frac{v_i - a}{b - a}.$$

Si todos los elementos son iguales, devuelva un vector de ceros.

Para `"zscore"`, calcule

$$\overline{v} = \frac{1}{n}\sum_{i=1}^{n} v_i$$

y el desvío estándar poblacional

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - \overline{v})^2}.$$

Si $\sigma \neq 0$,

$$y_i = \frac{v_i - \overline{v}}{\sigma}.$$

Si $\sigma = 0$, devuelva un vector de ceros.

Luego:

a) Implemente la función utilizando `"minmax"` como valor predeterminado del argumento `metodo`.

b) Valide que `v` sea un vector numérico real, finito y no vacío.

c) Rechace cualquier método diferente de los tres indicados.

d) Pruebe la función sin especificar el argumento opcional.

e) Pruebe explícitamente los tres métodos.

f) Pruebe un vector constante y explique por qué debe tratarse como un caso particular.

g) Pruebe un vector que contenga valores positivos y negativos.

h) Pruebe una matriz y verifique que sea rechazada.

i) Pruebe un vector vacío y verifique que sea rechazado.

j) Pruebe un nombre de método inexistente y verifique que sea rechazado.

k) Identifique el mecanismo que utiliza el lenguaje para determinar que un argumento opcional no fue suministrado.

Utilice, entre otros, los siguientes casos de prueba:

$$[2, 4, 6, 8], \qquad [-4, 2, 8], \qquad [5, 5, 5, 5].$$

Scilab. En Scilab, `argn(2)` permite conocer la cantidad de argumentos recibidos por una función.

```
function y=normalizar(v,metodo)

    if argn(2)<2 then
        metodo="minmax";
    end

    valido=(type(v)==1) & ..
           (size(v,"*")>0) & ..
           or(size(v)==1) & ..
           isreal(v);

    if valido then
        valido=~or(isnan(v)) & ..
               ~or(abs(v)==%inf);
    end

    if ~valido then
        error("v debe ser un vector numerico real y finito")
    end

    y=zeros(size(v,1),size(v,2));

    if metodo=="max" then

        s=max(abs(v));

        if s<>0 then
            y=v/s;
        end

    elseif metodo=="minmax" then

        a=min(v);
        b=max(v);

        if a<>b then
            y=(v-a)/(b-a);
        end

    elseif metodo=="zscore" then

        n=length(v);
        media=sum(v)/n;

        desvio=sqrt(
            sum((v-media).^2)/n
        );

        if desvio<>0 then
            y=(v-media)/desvio;
```

```
        end

    else

        error("Metodo no valido")

    end

endfunction


v=[2 4 6 8];

normalizar(v)
normalizar(v,"max")
normalizar(v,"minmax")
normalizar(v,"zscore")

normalizar([-4 2 8],"max")
normalizar([5 5 5 5],"zscore")
```

Octave. En Octave, `nargin` indica cuántos argumentos fueron recibidos por la función.

```
function y=normalizar(v,metodo)

  if nargin<2
    metodo="minmax";
  endif

  valido=isnumeric(v) ...
         && !isempty(v) ...
         && isvector(v) ...
         && isreal(v) ...
         && all(isfinite(v(:)));

  if !valido
    error("v debe ser un vector numerico real y finito")
  endif

  y=zeros(size(v));

  if strcmp(metodo,"max")

    s=max(abs(v));

    if s!=0
      y=v/s;
    endif
```

```
  elseif strcmp(metodo,"minmax")

    a=min(v);
    b=max(v);

    if a!=b
      y=(v-a)/(b-a);
    endif

  elseif strcmp(metodo,"zscore")

    n=numel(v);
    media=sum(v)/n;

    desvio=sqrt(
      sum((v-media).^2)/n
    );

    if desvio!=0
      y=(v-media)/desvio;
    endif

  else

    error("Metodo no valido")

  endif

endfunction


v=[2 4 6 8];

normalizar(v)
normalizar(v,"max")
normalizar(v,"minmax")
normalizar(v,"zscore")

normalizar([-4 2 8],"max")
normalizar([5 5 5 5],"zscore")
```

Python. En Python, el valor predeterminado puede declararse directamente en la definición de la función. Se utiliza NumPy para representar el vector numérico.

```
import numpy as np


def normalizar(v,metodo="minmax"):

    a=np.asarray(v)
```

```
    valido=(
        a.ndim==1
        and a.size>0
        and np.issubdtype(a.dtype,np.number)
        and np.isrealobj(a)
    )

    if valido:
        valido=np.all(np.isfinite(a))

    if not valido:
        raise ValueError(
            "v debe ser un vector numerico real y finito"
        )

    a=a.astype(float)

    y=np.zeros_like(a)

    if metodo=="max":

        s=np.max(np.abs(a))

        if s!=0:
            y=a/s

    elif metodo=="minmax":

        minimo=np.min(a)
        maximo=np.max(a)

        if minimo!=maximo:
            y=(a-minimo)/(maximo-minimo)

    elif metodo=="zscore":

        n=a.size
        media=np.sum(a)/n

        desvio=np.sqrt(
            np.sum((a-media)**2)/n
        )

        if desvio!=0:
            y=(a-media)/desvio

    else:

        raise ValueError("Metodo no valido")
```

```
    return y


v=np.array([2,4,6,8])

print(normalizar(v))
print(normalizar(v,"max"))
print(normalizar(v,"minmax"))
print(normalizar(v,"zscore"))

print(normalizar([-4,2,8],"max"))
print(normalizar([5,5,5,5],"zscore"))
```

Perl. En Perl, la cantidad de elementos de `@_` permite saber si el argumento opcional fue proporcionado. El vector se representa mediante una referencia a un arreglo.

```
use Scalar::Util qw(looks_like_number);
use POSIX qw(isfinite);


sub normalizar {

    my ($v,$metodo)=@_;

    $metodo='minmax'
        unless defined $metodo;

    die "v debe ser un vector\n"
        unless ref($v) eq 'ARRAY'
        && @$v>0;

    for my $x (@$v) {

        die "v debe contener numeros reales y finitos\n"
            unless looks_like_number($x)
            && isfinite($x);

    }

    my @y=(0) x scalar(@$v);

    if ($metodo eq 'max') {

        my $s=0;

        for my $x (@$v) {

            if (abs($x)>$s) {
                $s=abs($x);
```

```
            }

        }

        if ($s!=0) {

            for my $i (0..$#$v) {
                $y[$i]=$v->[$i]/$s;
            }

        }

    }
    elsif ($metodo eq 'minmax') {

        my $minimo=$v->[0];
        my $maximo=$v->[0];

        for my $x (@$v) {

            if ($x<$minimo) {
                $minimo=$x;
            }

            if ($x>$maximo) {
                $maximo=$x;
            }

        }

        if ($minimo!=$maximo) {

            for my $i (0..$#$v) {

                $y[$i]=
                    ($v->[$i]-$minimo)/
                    ($maximo-$minimo);

            }

        }

    }
    elsif ($metodo eq 'zscore') {

        my $n=scalar(@$v);
        my $suma=0;

        for my $x (@$v) {
            $suma=$suma+$x;
        }
```

```
        my $media=$suma/$n;
        my $suma2=0;

        for my $x (@$v) {
            $suma2=$suma2+($x-$media)**2;
        }

        my $desvio=sqrt($suma2/$n);

        if ($desvio!=0) {

            for my $i (0..$#$v) {

                $y[$i]=
                    ($v->[$i]-$media)/
                    $desvio;

            }

        }

    }
    else {

        die "Metodo no valido\n";

    }

    return \@y;
}


my @v=(2,4,6,8);

print join(' ',@{normalizar(\@v)}),"\n";

print join(
    ' ',
    @{normalizar(\@v,'max')}
),"\n";

print join(
    ' ',
    @{normalizar(\@v,'minmax')}
),"\n";

print join(
    ' ',
    @{normalizar(\@v,'zscore')}
),"\n";
```

JavaScript. En JavaScript, un parámetro puede recibir directamente un valor predeterminado en la declaración de la función. El vector se representa mediante un arreglo.

```
function normalizar(v,metodo="minmax"){

    let valido=
        Array.isArray(v)
        && v.length>0

    if(valido){

        for(const x of v){

            if(
                typeof x!=="number"
                || !Number.isFinite(x)
            ){
                valido=false
            }

        }

    }

    if(!valido){

        throw new TypeError(
            "v debe ser un vector numerico real y finito"
        )

    }

    let y=Array(v.length).fill(0)


    if(metodo==="max"){

        let s=0

        for(const x of v){

            if(Math.abs(x)>s){
                s=Math.abs(x)
            }

        }

        if(s!==0){

            for(let i=0;i<v.length;i++){
```

```
                y[i]=v[i]/s
            }

        }

    }
    else if(metodo==="minmax"){

        let minimo=v[0]
        let maximo=v[0]

        for(const x of v){

            if(x<minimo){
                minimo=x
            }

            if(x>maximo){
                maximo=x
            }

        }

        if(minimo!==maximo){

            for(let i=0;i<v.length;i++){

                y[i]=
                    (v[i]-minimo)/
                    (maximo-minimo)

            }

        }

    }
    else if(metodo==="zscore"){

        let n=v.length
        let suma=0

        for(const x of v){
            suma=suma+x
        }

        let media=suma/n
        let suma2=0

        for(const x of v){
            suma2=suma2+(x-media)**2
        }
```

```
        let desvio=Math.sqrt(suma2/n)

        if(desvio!==0){

            for(let i=0;i<v.length;i++){

                y[i]=
                    (v[i]-media)/
                    desvio

            }

        }

    }
    else{

        throw new Error(
            "Metodo no valido"
        )

    }

    return y

}


const v=[2,4,6,8]

console.log(normalizar(v))

console.log(
    normalizar(v,"max")
)

console.log(
    normalizar(v,"minmax")
)

console.log(
    normalizar(v,"zscore")
)

console.log(
    normalizar([-4,2,8],"max")
)

console.log(
    normalizar([5,5,5,5],"zscore")
```

```
)
```

5.11.5. Estructuras de datos

15. **Ejercicio:** Procesamiento de líneas de texto

    Cada línea contiene los datos de un estudiante con la forma

    ```
    Apellido, Nombre; LU=12345; nota=8.50
    ```

    El programa deberá procesar una colección de líneas con ese formato.

    Para este ejercicio se adoptarán las siguientes reglas:

    - la línea debe contener exactamente tres partes separadas por punto y coma;
    - la primera parte debe contener un apellido y un nombre separados por una coma;
    - la segunda parte debe comenzar con `LU=` y contener un número entero positivo;
    - la tercera parte debe comenzar con `nota=` y contener un número real comprendido entre 0 y 10;
    - los espacios al comienzo y al final de cada campo deben eliminarse;
    - el apellido y el nombre se almacenarán en minúsculas, para adoptar una regla de normalización común;
    - una línea que no cumpla alguna de estas condiciones será rechazada.

    Para cada línea válida construya un registro o estructura con los campos

    ```
    apellido
    nombre
    lu
    nota
    ```

    donde `lu` y `nota` deberán quedar almacenados como valores numéricos y no como cadenas.

    Escriba una función conceptual

    ```
    procesar_linea(linea) -> valido, registro, error
    ```

    que devuelva:

    - un valor lógico que indique si la línea es válida;
    - el registro construido cuando la línea es válida;
    - una cadena que identifique el campo incorrecto cuando la línea es rechazada.

Luego:

a) elimine los espacios innecesarios de cada campo;
b) separe apellido, nombre, LU y nota mediante operaciones básicas sobre cadenas;
c) no utilice una expresión regular para analizar la línea completa;
d) convierta la LU a un valor entero;
e) convierta la nota a un valor numérico real;
f) convierta apellido y nombre a minúsculas;
g) rechace líneas con cantidad incorrecta de campos;
h) rechace apellido o nombre vacíos;
i) rechace una LU que no sea un entero positivo;
j) rechace una nota que no sea numérica o que se encuentre fuera del intervalo de 0 a 10;
k) informe cuál campo produjo el rechazo;
l) almacene únicamente los registros válidos;
m) procese las diez líneas de prueba indicadas a continuación;
n) cuente cuántos registros fueron aceptados y cuántos rechazados.

Utilice las mismas líneas de prueba en todos los lenguajes:

```
Perez, Ana; LU=12345; nota=8.50
Gomez, Luis; LU=23456; nota=6
LOPEZ, Maria; LU=34567; nota=10
Suarez, Juan; LU=45678; nota=0
Diaz, Sofia; LU=56789; nota=7.25
Ruiz, Pedro; LU=abc; nota=8
Torres, Lucia; LU=67890; nota=12
Molina; LU=78901; nota=5
Castro, Elena; nota=9; LU=89012
Nunez, Tomas; LU=90123; nota=x
```

Antes de ejecutar el programa, determine cuáles líneas deberían ser aceptadas y cuáles rechazadas.

Scilab. En Scilab se utilizan `strsplit` para separar los campos, `stripblanks` para eliminar espacios exteriores, `convstr(...,"l")` para convertir a minúsculas y `strtod` para realizar las conversiones numéricas. Cada estudiante válido se representa mediante un `struct`.

```
function [valido,registro,error_campo]=procesar_linea(linea)
    valido=%f;
    registro=struct( ..
```

```
        "apellido","", ..
        "nombre","", ..
        "lu",0, ..
        "nota",0 ..
    );

    error_campo="";

    campos=strsplit(
        stripblanks(linea),
        ";"
    );

    if size(campos,"*")<>3 then
        error_campo="formato";
        return
    end


    persona=strsplit(
        stripblanks(campos(1)),
        ","
    );

    if size(persona,"*")<>2 then
        error_campo="apellido/nombre";
        return
    end

    apellido=convstr(
        stripblanks(persona(1)),
        "l"
    );

    nombre=convstr(
        stripblanks(persona(2)),
        "l"
    );

    if apellido=="" | nombre=="" then
        error_campo="apellido/nombre";
        return
    end


    campo_lu=strsplit(
        stripblanks(campos(2)),
        "="
    );

    if size(campo_lu,"*")<>2 then
```

```
        error_campo="LU";
        return
    end

    etiqueta=convstr(
        stripblanks(campo_lu(1)),
        "l"
    );

    if etiqueta<>"lu" then
        error_campo="LU";
        return
    end

    [lu,resto]=strtod(
        stripblanks(campo_lu(2))
    );

    if resto<>"" | isnan(lu) then
        error_campo="LU";
        return
    end

    if lu<>floor(lu) | lu<=0 then
        error_campo="LU";
        return
    end


    campo_nota=strsplit(
        stripblanks(campos(3)),
        "="
    );

    if size(campo_nota,"*")<>2 then
        error_campo="nota";
        return
    end

    etiqueta=convstr(
        stripblanks(campo_nota(1)),
        "l"
    );

    if etiqueta<>"nota" then
        error_campo="nota";
        return
    end

    [nota,resto]=strtod(
        stripblanks(campo_nota(2))
```

```
    );

    if resto<>"" | isnan(nota) then
        error_campo="nota";
        return
    end

    if nota<0 | nota>10 then
        error_campo="nota";
        return
    end


    registro=struct( ..
        "apellido",apellido, ..
        "nombre",nombre, ..
        "lu",lu, ..
        "nota",nota ..
    );

    valido=%t;

endfunction


lineas=[
"Perez, Ana; LU=12345; nota=8.50"
"Gomez, Luis; LU=23456; nota=6"
"LOPEZ, Maria; LU=34567; nota=10"
"Suarez, Juan; LU=45678; nota=0"
"Diaz, Sofia; LU=56789; nota=7.25"
"Ruiz, Pedro; LU=abc; nota=8"
"Torres, Lucia; LU=67890; nota=12"
"Molina; LU=78901; nota=5"
"Castro, Elena; nota=9; LU=89012"
"Nunez, Tomas; LU=90123; nota=x"
];

registros=list();

aceptados=0;
rechazados=0;

for i=1:length(lineas)

    [valido,r,error_campo]= ..
        procesar_linea(lineas(i));

    if valido then

        aceptados=aceptados+1;
```

```
        registros(aceptados)=r;

    else

        rechazados=rechazados+1;

        disp(
            "Linea "+string(i)+ ..
            " rechazada: "+error_campo
        )

    end

end

disp("Aceptados")
disp(aceptados)

disp("Rechazados")
disp(rechazados)
```

Octave. En Octave se utilizan `strsplit`, `strtrim` y `lower`. Los registros válidos se representan mediante estructuras.

```
function [valido,registro,error_campo]=procesar_linea(linea)

  valido=false;

  registro=struct(
    "apellido","",
    "nombre","",
    "lu",0,
    "nota",0
  );

  error_campo="";


  campos=strsplit(
    strtrim(linea),
    ";"
  );

  if numel(campos)!=3
    error_campo="formato";
    return
  endif


  persona=strsplit(
```

```
    strtrim(campos{1}),
    ","
  );

  if numel(persona)!=2
    error_campo="apellido/nombre";
    return
  endif

  apellido=lower(
    strtrim(persona{1})
  );

  nombre=lower(
    strtrim(persona{2})
  );

  if isempty(apellido) || isempty(nombre)
    error_campo="apellido/nombre";
    return
  endif


  campo_lu=strsplit(
    strtrim(campos{2}),
    "="
  );

  if numel(campo_lu)!=2
    error_campo="LU";
    return
  endif

  etiqueta=lower(
    strtrim(campo_lu{1})
  );

  if !strcmp(etiqueta,"lu")
    error_campo="LU";
    return
  endif

  lu=str2double(
    strtrim(campo_lu{2})
  );

  if !isfinite(lu) || lu!=fix(lu) || lu<=0
    error_campo="LU";
    return
  endif
```

```
  campo_nota=strsplit(
    strtrim(campos{3}),
    "="
  );

  if numel(campo_nota)!=2
    error_campo="nota";
    return
  endif

  etiqueta=lower(
    strtrim(campo_nota{1})
  );

  if !strcmp(etiqueta,"nota")
    error_campo="nota";
    return
  endif

  nota=str2double(
    strtrim(campo_nota{2})
  );

  if !isfinite(nota) || nota<0 || nota>10
    error_campo="nota";
    return
  endif


  registro=struct(
    "apellido",apellido,
    "nombre",nombre,
    "lu",lu,
    "nota",nota
  );

  valido=true;

endfunction


lineas={
  "Perez, Ana; LU=12345; nota=8.50",
  "Gomez, Luis; LU=23456; nota=6",
  "LOPEZ, Maria; LU=34567; nota=10",
  "Suarez, Juan; LU=45678; nota=0",
  "Diaz, Sofia; LU=56789; nota=7.25",
  "Ruiz, Pedro; LU=abc; nota=8",
  "Torres, Lucia; LU=67890; nota=12",
  "Molina; LU=78901; nota=5",
```

```
  "Castro, Elena; nota=9; LU=89012",
  "Nunez, Tomas; LU=90123; nota=x"
};

registros=struct(
  "apellido",{},
  "nombre",{},
  "lu",{},
  "nota",{}
);

aceptados=0;
rechazados=0;

for i=1:numel(lineas)

  [valido,r,error_campo]= ...
    procesar_linea(lineas{i});

  if valido

    aceptados=aceptados+1;
    registros(aceptados)=r;

  else

    rechazados=rechazados+1;

    disp(
      ["Linea " num2str(i) ...
       " rechazada: " error_campo]
    )

  endif

endfor

disp("Aceptados")
disp(aceptados)

disp("Rechazados")
disp(rechazados)
```

Python. En Python se utilizan las operaciones básicas de cadenas `strip`, `split` y `lower`. Cada registro válido se representa mediante un diccionario.

```
def procesar_linea(linea):

    registro={
        "apellido":"",
```

```
        "nombre":"",
        "lu":0,
        "nota":0
    }


    campos=linea.strip().split(";")

    if len(campos)!=3:
        return False,registro,"formato"


    persona=campos[0].strip().split(",")

    if len(persona)!=2:
        return False,registro,"apellido/nombre"

    apellido=persona[0].strip().lower()
    nombre=persona[1].strip().lower()

    if apellido=="" or nombre=="":
        return False,registro,"apellido/nombre"


    campo_lu=campos[1].strip().split("=")

    if len(campo_lu)!=2:
        return False,registro,"LU"

    if campo_lu[0].strip().lower()!="lu":
        return False,registro,"LU"

    texto_lu=campo_lu[1].strip()

    try:
        lu=float(texto_lu)
    except ValueError:
        return False,registro,"LU"

    if not lu.is_integer() or lu<=0:
        return False,registro,"LU"

    lu=int(lu)


    campo_nota=campos[2].strip().split("=")

    if len(campo_nota)!=2:
        return False,registro,"nota"

    if campo_nota[0].strip().lower()!="nota":
```

```
            return False,registro,"nota"

        texto_nota=campo_nota[1].strip()

        try:
            nota=float(texto_nota)
        except ValueError:
            return False,registro,"nota"

        if nota<0 or nota>10:
            return False,registro,"nota"


        registro={
            "apellido":apellido,
            "nombre":nombre,
            "lu":lu,
            "nota":nota
        }

        return True,registro,""


    lineas=[
        "Perez, Ana; LU=12345; nota=8.50",
        "Gomez, Luis; LU=23456; nota=6",
        "LOPEZ, Maria; LU=34567; nota=10",
        "Suarez, Juan; LU=45678; nota=0",
        "Diaz, Sofia; LU=56789; nota=7.25",
        "Ruiz, Pedro; LU=abc; nota=8",
        "Torres, Lucia; LU=67890; nota=12",
        "Molina; LU=78901; nota=5",
        "Castro, Elena; nota=9; LU=89012",
        "Nunez, Tomas; LU=90123; nota=x"
    ]

    registros=[]

    aceptados=0
    rechazados=0

    for i in range(len(lineas)):

        valido,r,error_campo=(
            procesar_linea(lineas[i])
        )

        if valido:

            aceptados=aceptados+1
            registros.append(r)
```

```
    else:

        rechazados=rechazados+1

        print(
            "Linea",
            i+1,
            "rechazada:",
            error_campo
        )

print("Aceptados")
print(aceptados)

print("Rechazados")
print(rechazados)
```

Perl. En Perl cada registro se representa mediante una referencia a un *hash*. Se utilizan operaciones de separación y eliminación de espacios antes de realizar las conversiones numéricas.

```
use Scalar::Util qw(looks_like_number);


sub trim {

    my ($s)=@_;

    while (
        length($s)>0
        && substr($s,0,1) eq ' '
    ) {
        $s=substr($s,1);
    }

    while (
        length($s)>0
        && substr($s,-1,1) eq ' '
    ) {
        chop $s;
    }

    return $s;
}


sub procesar_linea {

    my ($linea)=@_;
```

```
    my $registro={
        apellido=>'',
        nombre=>'',
        lu=>0,
        nota=>0
    };


    my @campos=split(';',trim($linea));

    if (@campos!=3) {
        return (0,$registro,'formato');
    }


    my @persona=split(',',trim($campos[0]));

    if (@persona!=2) {
        return (0,$registro,'apellido/nombre');
    }

    my $apellido=lc(trim($persona[0]));
    my $nombre=lc(trim($persona[1]));

    if ($apellido eq '' || $nombre eq '') {
        return (0,$registro,'apellido/nombre');
    }


    my @campo_lu=split('=',trim($campos[1]));

    if (@campo_lu!=2) {
        return (0,$registro,'LU');
    }

    if (lc(trim($campo_lu[0])) ne 'lu') {
        return (0,$registro,'LU');
    }

    my $texto_lu=trim($campo_lu[1]);

    if (!looks_like_number($texto_lu)) {
        return (0,$registro,'LU');
    }

    my $lu=0+$texto_lu;

    if (int($lu)!=$lu || $lu<=0) {
        return (0,$registro,'LU');
    }
```

```
    my @campo_nota=
        split('=',trim($campos[2]));

    if (@campo_nota!=2) {
        return (0,$registro,'nota');
    }

    if (lc(trim($campo_nota[0])) ne 'nota') {
        return (0,$registro,'nota');
    }

    my $texto_nota=trim($campo_nota[1]);

    if (!looks_like_number($texto_nota)) {
        return (0,$registro,'nota');
    }

    my $nota=0+$texto_nota;

    if ($nota<0 || $nota>10) {
        return (0,$registro,'nota');
    }


    $registro={
        apellido=>$apellido,
        nombre=>$nombre,
        lu=>$lu,
        nota=>$nota
    };

    return (1,$registro,'');
}


my @lineas=(
    'Perez, Ana; LU=12345; nota=8.50',
    'Gomez, Luis; LU=23456; nota=6',
    'LOPEZ, Maria; LU=34567; nota=10',
    'Suarez, Juan; LU=45678; nota=0',
    'Diaz, Sofia; LU=56789; nota=7.25',
    'Ruiz, Pedro; LU=abc; nota=8',
    'Torres, Lucia; LU=67890; nota=12',
    'Molina; LU=78901; nota=5',
    'Castro, Elena; nota=9; LU=89012',
    'Nunez, Tomas; LU=90123; nota=x'
);

my @registros;
```

```
my $aceptados=0;
my $rechazados=0;

for (
    my $i=0;
    $i<@lineas;
    $i++
) {

    my (
        $valido,
        $registro,
        $error_campo
    )=procesar_linea($lineas[$i]);

    if ($valido) {

        $aceptados=$aceptados+1;
        push @registros,$registro;

    }
    else {

        $rechazados=$rechazados+1;

        print(
            "Linea ",
            $i+1,
            " rechazada: ",
            $error_campo,
            "\n"
        );

    }

}

print "Aceptados\n";
print $aceptados,"\n";

print "Rechazados\n";
print $rechazados,"\n";
```

JavaScript. En JavaScript cada registro se representa mediante un objeto. Se utilizan `trim`, `split` y `toLowerCase` para procesar las cadenas.

```
function procesarLinea(linea){

    let registro={
```

```
        apellido:"",
        nombre:"",
        lu:0,
        nota:0
    }


    let campos=linea.trim().split(";")

    if(campos.length!==3){
        return [false,registro,"formato"]
    }


    let persona=
        campos[0].trim().split(",")

    if(persona.length!==2){
        return [
            false,
            registro,
            "apellido/nombre"
        ]
    }

    let apellido=
        persona[0].trim().toLowerCase()

    let nombre=
        persona[1].trim().toLowerCase()

    if(apellido===""  || nombre===""){
        return [
            false,
            registro,
            "apellido/nombre"
        ]
    }


    let campoLU=
        campos[1].trim().split("=")

    if(campoLU.length!==2){
        return [false,registro,"LU"]
    }

    if(
        campoLU[0]
        .trim()
        .toLowerCase()!=="lu"
```

```
    ){
        return [false,registro,"LU"]
    }

    let textoLU=campoLU[1].trim()
    let lu=Number(textoLU)

    if(
        textoLU===""
        || !Number.isInteger(lu)
        || lu<=0
    ){
        return [false,registro,"LU"]
    }


    let campoNota=
        campos[2].trim().split("=")

    if(campoNota.length!==2){
        return [false,registro,"nota"]
    }

    if(
        campoNota[0]
        .trim()
        .toLowerCase()!=="nota"
    ){
        return [false,registro,"nota"]
    }

    let textoNota=campoNota[1].trim()
    let nota=Number(textoNota)

    if(
        textoNota===""
        || !Number.isFinite(nota)
        || nota<0
        || nota>10
    ){
        return [false,registro,"nota"]
    }


    registro={
        apellido:apellido,
        nombre:nombre,
        lu:lu,
        nota:nota
    }
```

```
    return [true,registro,""]
}


let lineas=[
    "Perez, Ana; LU=12345; nota=8.50",
    "Gomez, Luis; LU=23456; nota=6",
    "LOPEZ, Maria; LU=34567; nota=10",
    "Suarez, Juan; LU=45678; nota=0",
    "Diaz, Sofia; LU=56789; nota=7.25",
    "Ruiz, Pedro; LU=abc; nota=8",
    "Torres, Lucia; LU=67890; nota=12",
    "Molina; LU=78901; nota=5",
    "Castro, Elena; nota=9; LU=89012",
    "Nunez, Tomas; LU=90123; nota=x"
]

let registros=[]

let aceptados=0
let rechazados=0

for(let i=0;i<lineas.length;i++){

    let [
        valido,
        registro,
        errorCampo
    ]=procesarLinea(lineas[i])

    if(valido){

        aceptados=aceptados+1
        registros.push(registro)

    }
    else{

        rechazados=rechazados+1

        console.log(
            "Linea",
            i+1,
            "rechazada:",
            errorCampo
        )

    }

}
```

```
console.log("Aceptados")
console.log(aceptados)

console.log("Rechazados")
console.log(rechazados)
```

16. **Ejercicio:** Representaciones de datos

    Se dispone de información sobre varios experimentos. Para cada experimento se desea almacenar:

    - nombre;
    - fecha;
    - operador;
    - vector de mediciones;
    - estado, que puede ser "`aprobado`" o "`rechazado`".

    Utilice inicialmente los siguientes datos:

Nombre	Fecha	Operador	Mediciones	Estado
Solar1	2026-08-01	Ana	10.2, 10.4, 10.1	aprobado
Solar2	2026-08-02	Luis	9.8, 10.0, 9.9	rechazado
Solar3	2026-08-03	Ana	11.0, 10.8, 11.2	aprobado
Solar4	2026-08-04	Sofia	10.5, 10.7, 10.6	aprobado

    Represente exactamente los mismos datos de dos maneras.

    a) En la primera representación utilice colecciones paralelas: una colección para los nombres, otra para las fechas, otra para los operadores, otra para los vectores de mediciones y otra para los estados.

    b) En la segunda representación utilice una colección de registros o estructuras. Cada elemento deberá contener los cinco campos de un experimento.

    c) En ambas representaciones muestre el nombre y las mediciones del segundo experimento.

    d) Agregue en ambas representaciones el experimento

    ```
    Solar5
    2026-08-05
    Pedro
    [9.5, 9.7, 9.6]
    aprobado
    ```

    e) Recorra todos los experimentos y muestre solamente los que tengan estado `.ªprobado"`.

    f) Calcule la media de las mediciones de cada experimento.

    g) Determine qué experimento posee la mayor media.

h) Elimine el segundo experimento de ambas representaciones.

i) Compruebe que los datos restantes continúan correspondiéndose correctamente.

j) Compare ambas representaciones. En particular, explique por qué mantener varias colecciones paralelas exige conservar cuidadosamente las mismas posiciones, mientras que un registro mantiene agrupados los datos correspondientes a un mismo experimento.

En todos los lenguajes deberá realizarse conceptualmente la misma secuencia:

```
construir representacion
agregar experimento
recorrer experimentos
seleccionar aprobados
calcular medias
buscar mayor media
eliminar experimento
```

Scilab. En Scilab, las colecciones paralelas pueden combinar vectores de cadenas con una `list` para las mediciones. Para la segunda representación se utiliza una `list` cuyos elementos son `struct`.

Primera representación:

```
nombres=["Solar1","Solar2","Solar3","Solar4"];

fechas=[
    "2026-08-01"
    "2026-08-02"
    "2026-08-03"
    "2026-08-04"
];

operadores=["Ana","Luis","Ana","Sofia"];

mediciones=list( ..
    [10.2 10.4 10.1], ..
    [9.8 10.0 9.9], ..
    [11.0 10.8 11.2], ..
    [10.5 10.7 10.6] ..
);

estados=[
    "aprobado"
    "rechazado"
    "aprobado"
    "aprobado"
];
```

```
// Segundo experimento.
disp(nombres(2))
disp(mediciones(2))

// Agregar Solar5.
nombres(5)="Solar5";
fechas(5)="2026-08-05";
operadores(5)="Pedro";
mediciones(5)=[9.5 9.7 9.6];
estados(5)="aprobado";

// Mostrar aprobados.
for i=1:length(nombres)
    if estados(i)=="aprobado" then
        disp(nombres(i))
    end
end

// Calcular medias.
medias=zeros(1,length(nombres));

for i=1:length(nombres)

    v=mediciones(i);
    suma=0;

    for j=1:length(v)
        suma=suma+v(j);
    end

    medias(i)=suma/length(v);

end

// Buscar la mayor media.
mayor=medias(1);
pos=1;

for i=2:length(medias)

    if medias(i)>mayor then
        mayor=medias(i);
        pos=i;
    end

end

disp(nombres(pos))
disp(mayor)

// Eliminar segundo experimento.
```

```
nombres(2)=[];
fechas(2)=[];
operadores(2)=[];
mediciones(2)=null();
estados(2)=[];
```

Segunda representación:

```
experimentos=list();

experimentos(1)=struct( ..
    "nombre","Solar1", ..
    "fecha","2026-08-01", ..
    "operador","Ana", ..
    "mediciones",[10.2 10.4 10.1], ..
    "estado","aprobado" ..
);

experimentos(2)=struct( ..
    "nombre","Solar2", ..
    "fecha","2026-08-02", ..
    "operador","Luis", ..
    "mediciones",[9.8 10.0 9.9], ..
    "estado","rechazado" ..
);

experimentos(3)=struct( ..
    "nombre","Solar3", ..
    "fecha","2026-08-03", ..
    "operador","Ana", ..
    "mediciones",[11.0 10.8 11.2], ..
    "estado","aprobado" ..
);

experimentos(4)=struct( ..
    "nombre","Solar4", ..
    "fecha","2026-08-04", ..
    "operador","Sofia", ..
    "mediciones",[10.5 10.7 10.6], ..
    "estado","aprobado" ..
);

disp(experimentos(2).nombre)
disp(experimentos(2).mediciones)

experimentos(5)=struct( ..
    "nombre","Solar5", ..
    "fecha","2026-08-05", ..
    "operador","Pedro", ..
    "mediciones",[9.5 9.7 9.6], ..
    "estado","aprobado" ..
```

```
);

medias=zeros(1,length(experimentos));

for i=1:length(experimentos)

    e=experimentos(i);

    if e.estado=="aprobado" then
        disp(e.nombre)
    end

    suma=0;

    for j=1:length(e.mediciones)
        suma=suma+e.mediciones(j);
    end

    medias(i)=suma/length(e.mediciones);

end

mayor=medias(1);
pos=1;

for i=2:length(medias)

    if medias(i)>mayor then
        mayor=medias(i);
        pos=i;
    end

end

disp(experimentos(pos).nombre)
disp(mayor)

experimentos(2)=null();
```

Octave. En Octave, las colecciones paralelas pueden representarse mediante celdas. Para la segunda representación se utiliza un arreglo de estructuras.

Primera representación:

```
nombres={
  "Solar1","Solar2","Solar3","Solar4"
};

fechas={
  "2026-08-01","2026-08-02",
  "2026-08-03","2026-08-04"
```

```
};

operadores={"Ana","Luis","Ana","Sofia"};

mediciones={
  [10.2 10.4 10.1],
  [9.8 10.0 9.9],
  [11.0 10.8 11.2],
  [10.5 10.7 10.6]
};

estados={
  "aprobado","rechazado",
  "aprobado","aprobado"
};

disp(nombres{2})
disp(mediciones{2})

nombres{5}="Solar5";
fechas{5}="2026-08-05";
operadores{5}="Pedro";
mediciones{5}=[9.5 9.7 9.6];
estados{5}="aprobado";

for i=1:length(nombres)

  if strcmp(estados{i},"aprobado")
    disp(nombres{i})
  endif

endfor

medias=zeros(1,length(nombres));

for i=1:length(nombres)

  v=mediciones{i};
  suma=0;

  for j=1:length(v)
    suma=suma+v(j);
  endfor

  medias(i)=suma/length(v);

endfor

mayor=medias(1);
pos=1;
```

```
for i=2:length(medias)

  if medias(i)>mayor
    mayor=medias(i);
    pos=i;
  endif

endfor

disp(nombres{pos})
disp(mayor)

nombres(2)=[];
fechas(2)=[];
operadores(2)=[];
mediciones(2)=[];
estados(2)=[];
```

Segunda representación:

```
experimentos(1)=struct(
  "nombre","Solar1",
  "fecha","2026-08-01",
  "operador","Ana",
  "mediciones",[10.2 10.4 10.1],
  "estado","aprobado"
);

experimentos(2)=struct(
  "nombre","Solar2",
  "fecha","2026-08-02",
  "operador","Luis",
  "mediciones",[9.8 10.0 9.9],
  "estado","rechazado"
);

experimentos(3)=struct(
  "nombre","Solar3",
  "fecha","2026-08-03",
  "operador","Ana",
  "mediciones",[11.0 10.8 11.2],
  "estado","aprobado"
);

experimentos(4)=struct(
  "nombre","Solar4",
  "fecha","2026-08-04",
  "operador","Sofia",
  "mediciones",[10.5 10.7 10.6],
  "estado","aprobado"
);
```

```
disp(experimentos(2).nombre)
disp(experimentos(2).mediciones)

experimentos(5)=struct(
  "nombre","Solar5",
  "fecha","2026-08-05",
  "operador","Pedro",
  "mediciones",[9.5 9.7 9.6],
  "estado","aprobado"
);

medias=zeros(1,length(experimentos));

for i=1:length(experimentos)

  if strcmp(experimentos(i).estado,"aprobado")
    disp(experimentos(i).nombre)
  endif

  v=experimentos(i).mediciones;
  suma=0;

  for j=1:length(v)
    suma=suma+v(j);
  endfor

  medias(i)=suma/length(v);

endfor

mayor=medias(1);
pos=1;

for i=2:length(medias)

  if medias(i)>mayor
    mayor=medias(i);
    pos=i;
  endif

endfor

disp(experimentos(pos).nombre)
disp(mayor)

experimentos(2)=[];
```

Python. En Python, la primera representación utiliza listas paralelas. La segunda utiliza una lista de diccionarios.

Primera representación:

```
nombres=[
    "Solar1","Solar2","Solar3","Solar4"
]

fechas=[
    "2026-08-01","2026-08-02",
    "2026-08-03","2026-08-04"
]

operadores=[
    "Ana","Luis","Ana","Sofia"
]

mediciones=[
    [10.2,10.4,10.1],
    [9.8,10.0,9.9],
    [11.0,10.8,11.2],
    [10.5,10.7,10.6]
]

estados=[
    "aprobado","rechazado",
    "aprobado","aprobado"
]

print(nombres[1])
print(mediciones[1])

nombres.append("Solar5")
fechas.append("2026-08-05")
operadores.append("Pedro")
mediciones.append([9.5,9.7,9.6])
estados.append("aprobado")

for i in range(len(nombres)):

    if estados[i]=="aprobado":
        print(nombres[i])

medias=[0]*len(nombres)

for i in range(len(nombres)):

    suma=0

    for x in mediciones[i]:
        suma=suma+x

    medias[i]=suma/len(mediciones[i])
```

```
mayor=medias[0]
pos=0

for i in range(1,len(medias)):

    if medias[i]>mayor:
        mayor=medias[i]
        pos=i

print(nombres[pos])
print(mayor)

del nombres[1]
del fechas[1]
del operadores[1]
del mediciones[1]
del estados[1]
```

Segunda representación:

```
experimentos=[
    {
        "nombre":"Solar1",
        "fecha":"2026-08-01",
        "operador":"Ana",
        "mediciones":[10.2,10.4,10.1],
        "estado":"aprobado"
    },
    {
        "nombre":"Solar2",
        "fecha":"2026-08-02",
        "operador":"Luis",
        "mediciones":[9.8,10.0,9.9],
        "estado":"rechazado"
    },
    {
        "nombre":"Solar3",
        "fecha":"2026-08-03",
        "operador":"Ana",
        "mediciones":[11.0,10.8,11.2],
        "estado":"aprobado"
    },
    {
        "nombre":"Solar4",
        "fecha":"2026-08-04",
        "operador":"Sofia",
        "mediciones":[10.5,10.7,10.6],
        "estado":"aprobado"
    }
]
```

```
print(experimentos[1]["nombre"])
print(experimentos[1]["mediciones"])

experimentos.append({
    "nombre":"Solar5",
    "fecha":"2026-08-05",
    "operador":"Pedro",
    "mediciones":[9.5,9.7,9.6],
    "estado":"aprobado"
})

medias=[0]*len(experimentos)

for i in range(len(experimentos)):

    e=experimentos[i]

    if e["estado"]=="aprobado":
        print(e["nombre"])

    suma=0

    for x in e["mediciones"]:
        suma=suma+x

    medias[i]=suma/len(e["mediciones"])

mayor=medias[0]
pos=0

for i in range(1,len(medias)):

    if medias[i]>mayor:
        mayor=medias[i]
        pos=i

print(experimentos[pos]["nombre"])
print(mayor)

del experimentos[1]
```

Perl. En Perl, la primera representación utiliza arreglos paralelos y referencias a arreglos para las mediciones. La segunda utiliza un arreglo de referencias a *hashes*.

Primera representación:

```
@nombres=(
    'Solar1','Solar2','Solar3','Solar4'
);
```

```
@fechas=(
    '2026-08-01','2026-08-02',
    '2026-08-03','2026-08-04'
);

@operadores=(
    'Ana','Luis','Ana','Sofia'
);

@mediciones=(
    [10.2,10.4,10.1],
    [9.8,10.0,9.9],
    [11.0,10.8,11.2],
    [10.5,10.7,10.6]
);

@estados=(
    'aprobado','rechazado',
    'aprobado','aprobado'
);

print $nombres[1],"\n";
print join(' ',@{$mediciones[1]}),"\n";

push @nombres,'Solar5';
push @fechas,'2026-08-05';
push @operadores,'Pedro';
push @mediciones,[9.5,9.7,9.6];
push @estados,'aprobado';

for ($i=0;$i<@nombres;$i++) {

    if ($estados[$i] eq 'aprobado') {
        print $nombres[$i],"\n";
    }

}

@medias=(0) x scalar(@nombres);

for ($i=0;$i<@nombres;$i++) {

    $suma=0;

    for $x (@{$mediciones[$i]}) {
        $suma=$suma+$x;
    }

    $medias[$i]=
        $suma/scalar(@{$mediciones[$i]});
```

```
}

$mayor=$medias[0];
$pos=0;

for ($i=1;$i<@medias;$i++) {

    if ($medias[$i]>$mayor) {
        $mayor=$medias[$i];
        $pos=$i;
    }

}

print $nombres[$pos],"\n";
print $mayor,"\n";

splice @nombres,1,1;
splice @fechas,1,1;
splice @operadores,1,1;
splice @mediciones,1,1;
splice @estados,1,1;
```

Segunda representación:

```
@experimentos=(
    {
        nombre=>'Solar1',
        fecha=>'2026-08-01',
        operador=>'Ana',
        mediciones=>[10.2,10.4,10.1],
        estado=>'aprobado'
    },
    {
        nombre=>'Solar2',
        fecha=>'2026-08-02',
        operador=>'Luis',
        mediciones=>[9.8,10.0,9.9],
        estado=>'rechazado'
    },
    {
        nombre=>'Solar3',
        fecha=>'2026-08-03',
        operador=>'Ana',
        mediciones=>[11.0,10.8,11.2],
        estado=>'aprobado'
    },
    {
        nombre=>'Solar4',
        fecha=>'2026-08-04',
        operador=>'Sofia',
```

```
        mediciones=>[10.5,10.7,10.6],
        estado=>'aprobado'
    }
);

print $experimentos[1]{nombre},"\n";

print join(
    ' ',
    @{$experimentos[1]{mediciones}}
),"\n";

push @experimentos,{
    nombre=>'Solar5',
    fecha=>'2026-08-05',
    operador=>'Pedro',
    mediciones=>[9.5,9.7,9.6],
    estado=>'aprobado'
};

@medias=(0) x scalar(@experimentos);

for ($i=0;$i<@experimentos;$i++) {

    $e=$experimentos[$i];

    if ($e->{estado} eq 'aprobado') {
        print $e->{nombre},"\n";
    }

    $suma=0;

    for $x (@{$e->{mediciones}}) {
        $suma=$suma+$x;
    }

    $medias[$i]=
        $suma/scalar(@{$e->{mediciones}});

}

$mayor=$medias[0];
$pos=0;

for ($i=1;$i<@medias;$i++) {

    if ($medias[$i]>$mayor) {
        $mayor=$medias[$i];
        $pos=$i;
    }
```

```
}

print $experimentos[$pos]{nombre},"\n";
print $mayor,"\n";

splice @experimentos,1,1;
```

JavaScript. En JavaScript, la primera representación utiliza arreglos paralelos. La segunda utiliza un arreglo de objetos.

Primera representación:

```
const nombres=[
    "Solar1","Solar2","Solar3","Solar4"
]

const fechas=[
    "2026-08-01","2026-08-02",
    "2026-08-03","2026-08-04"
]

const operadores=[
    "Ana","Luis","Ana","Sofia"
]

const mediciones=[
    [10.2,10.4,10.1],
    [9.8,10.0,9.9],
    [11.0,10.8,11.2],
    [10.5,10.7,10.6]
]

const estados=[
    "aprobado","rechazado",
    "aprobado","aprobado"
]

console.log(nombres[1])
console.log(mediciones[1])

nombres.push("Solar5")
fechas.push("2026-08-05")
operadores.push("Pedro")
mediciones.push([9.5,9.7,9.6])
estados.push("aprobado")

for(let i=0;i<nombres.length;i++){

    if(estados[i]==="aprobado"){
        console.log(nombres[i])
    }
```

```
}

const medias=Array(nombres.length).fill(0)

for(let i=0;i<nombres.length;i++){

    let suma=0

    for(const x of mediciones[i]){
        suma=suma+x
    }

    medias[i]=suma/mediciones[i].length

}

let mayor=medias[0]
let pos=0

for(let i=1;i<medias.length;i++){

    if(medias[i]>mayor){
        mayor=medias[i]
        pos=i
    }

}

console.log(nombres[pos])
console.log(mayor)

nombres.splice(1,1)
fechas.splice(1,1)
operadores.splice(1,1)
mediciones.splice(1,1)
estados.splice(1,1)
```

Segunda representación:

```
const experimentos=[
    {
        nombre:"Solar1",
        fecha:"2026-08-01",
        operador:"Ana",
        mediciones:[10.2,10.4,10.1],
        estado:"aprobado"
    },
    {
        nombre:"Solar2",
        fecha:"2026-08-02",
```

```
        operador:"Luis",
        mediciones:[9.8,10.0,9.9],
        estado:"rechazado"
    },
    {
        nombre:"Solar3",
        fecha:"2026-08-03",
        operador:"Ana",
        mediciones:[11.0,10.8,11.2],
        estado:"aprobado"
    },
    {
        nombre:"Solar4",
        fecha:"2026-08-04",
        operador:"Sofia",
        mediciones:[10.5,10.7,10.6],
        estado:"aprobado"
    }
]

console.log(experimentos[1].nombre)
console.log(experimentos[1].mediciones)

experimentos.push({
    nombre:"Solar5",
    fecha:"2026-08-05",
    operador:"Pedro",
    mediciones:[9.5,9.7,9.6],
    estado:"aprobado"
})

const medias=Array(experimentos.length).fill(0)

for(let i=0;i<experimentos.length;i++){

    const e=experimentos[i]

    if(e.estado==="aprobado"){
        console.log(e.nombre)
    }

    let suma=0

    for(const x of e.mediciones){
        suma=suma+x
    }

    medias[i]=suma/e.mediciones.length

}
```

```
let mayor=medias[0]
let pos=0

for(let i=1;i<medias.length;i++){

    if(medias[i]>mayor){
        mayor=medias[i]
        pos=i
    }

}

console.log(experimentos[pos].nombre)
console.log(mayor)

experimentos.splice(1,1)
```

17. **Ejercicio:** Colecciones heterogéneas

    Construya una colección capaz de contener simultáneamente elementos de diferentes tipos.

    Utilice exactamente los siguientes elementos:

    - el entero `25`;
    - el número real `3.14`;
    - las cadenas `"temperatura"` y `"solar"`;
    - el vector numérico

      $$[1, 2, 3];$$

    - un registro que represente una medición con los campos

      ```
      valor = 23.5
      unidad = "C"
      ```

    Para este ejercicio se utilizarán las categorías conceptuales

    ```
    entero
    real
    cadena
    vector
    registro
    ```

    independientemente del nombre que el lenguaje utilice internamente para sus tipos de datos.

    Luego:

    a) Recorra la colección e indique la categoría de cada elemento.

b) Sume únicamente los elementos numéricos escalares.

c) Concatene, separadas por un espacio, únicamente las cadenas.

d) Calcule la suma de los elementos de los vectores contenidos dentro de la colección.

e) Para cada registro, muestre por separado sus campos `valor` y `unidad`.

f) Cuente cuántos elementos pertenecen a cada una de las cinco categorías.

g) Construya una nueva colección denominada `filtrada` que contenga solamente los elementos de la categoría `"vector"`.

h) Cambie la categoría seleccionada y compruebe que el mismo procedimiento permite obtener, por ejemplo, solamente las cadenas.

i) Explique por qué una matriz numérica común no puede representar naturalmente todos los elementos de esta colección.

j) Compare cómo representa cada lenguaje una colección heterogénea y un registro con campos nombrados.

Con los datos indicados, antes de ejecutar el programa prediga:

- la suma de los escalares numéricos;
- el texto obtenido al concatenar las cadenas;
- la suma de los elementos del vector;
- la cantidad de elementos de cada categoría.

Scilab. En Scilab, una `list` puede contener elementos de tipos diferentes y `struct` permite construir registros con campos nombrados. Los números enteros y reales utilizados aquí pertenecen al mismo tipo numérico básico, por lo que se distinguen conceptualmente examinando si el valor posee parte fraccionaria.

```
function categoria=clasificar(x)

    t=typeof(x);

    if t=="constant" then

        if size(x,"*")==1 then

            if x==floor(x) then
                categoria="entero";
            else
                categoria="real";
            end

        else

            categoria="vector";
```

```
        end

    elseif t=="string" then

        categoria="cadena";

    elseif t=="st" then

        categoria="registro";

    else

        categoria="otro";

    end

endfunction


medicion=struct( ..
    "valor",23.5, ..
    "unidad","C" ..
);

coleccion=list( ..
    25, ..
    3.14, ..
    "temperatura", ..
    "solar", ..
    [1 2 3], ..
    medicion ..
);


suma_escalares=0;
texto="";
suma_vectores=0;

enteros=0;
reales=0;
cadenas=0;
vectores=0;
registros=0;

filtrada=list();
cantidad_filtrada=0;

categoria_buscada="vector";


for i=1:length(coleccion)
```

```
x=coleccion(i);

categoria=clasificar(x);

disp(categoria)


if categoria=="entero" then

    enteros=enteros+1;
    suma_escalares=suma_escalares+x;

elseif categoria=="real" then

    reales=reales+1;
    suma_escalares=suma_escalares+x;

elseif categoria=="cadena" then

    cadenas=cadenas+1;

    if texto=="" then
        texto=x;
    else
        texto=texto+" "+x;
    end

elseif categoria=="vector" then

    vectores=vectores+1;

    for j=1:length(x)
        suma_vectores=suma_vectores+x(j);
    end

elseif categoria=="registro" then

    registros=registros+1;

    disp(x.valor)
    disp(x.unidad)

end


if categoria==categoria_buscada then

    cantidad_filtrada=cantidad_filtrada+1;
    filtrada(cantidad_filtrada)=x;
```

```
    end

end


disp("Suma de escalares")
disp(suma_escalares)

disp("Cadenas")
disp(texto)

disp("Suma de vectores")
disp(suma_vectores)

disp("Enteros")
disp(enteros)

disp("Reales")
disp(reales)

disp("Cadenas")
disp(cadenas)

disp("Vectores")
disp(vectores)

disp("Registros")
disp(registros)

disp(filtrada)
```

Octave. En Octave, una celda puede contener elementos de diferentes tipos y una estructura permite representar un registro. Los valores enteros y reales se distinguen aquí por su valor y no por su tipo interno.

```
function categoria=clasificar(x)

  if isnumeric(x)

    if isscalar(x)

      if x==fix(x)
        categoria="entero";
      else
        categoria="real";
      endif

    elseif isvector(x)

      categoria="vector";
```

```
    else
      categoria="otro";
    endif
  elseif ischar(x) || isstring(x)
    categoria="cadena";
  elseif isstruct(x)
    categoria="registro";
  else
    categoria="otro";
  endif
endfunction

medicion=struct(
  "valor",23.5,
  "unidad","C"
);

coleccion={
  25,
  3.14,
  "temperatura",
  "solar",
  [1 2 3],
  medicion
};

suma_escalares=0;
texto="";
suma_vectores=0;

enteros=0;
reales=0;
cadenas=0;
vectores=0;
registros=0;

filtrada={};
```

```
categoria_buscada="vector";


for i=1:numel(coleccion)

  x=coleccion{i};

  categoria=clasificar(x);

  disp(categoria)


  if strcmp(categoria,"entero")

    enteros=enteros+1;
    suma_escalares=suma_escalares+x;

  elseif strcmp(categoria,"real")

    reales=reales+1;
    suma_escalares=suma_escalares+x;

  elseif strcmp(categoria,"cadena")

    cadenas=cadenas+1;

    if isempty(texto)
      texto=x;
    else
      texto=[texto " " x];
    endif

  elseif strcmp(categoria,"vector")

    vectores=vectores+1;

    for j=1:length(x)
      suma_vectores=suma_vectores+x(j);
    endfor

  elseif strcmp(categoria,"registro")

    registros=registros+1;

    disp(x.valor)
    disp(x.unidad)

  endif


  if strcmp(categoria,categoria_buscada)
```

```
    filtrada{end+1}=x;

  endif

endfor


disp("Suma de escalares")
disp(suma_escalares)

disp("Cadenas")
disp(texto)

disp("Suma de vectores")
disp(suma_vectores)

disp("Enteros")
disp(enteros)

disp("Reales")
disp(reales)

disp("Cadenas")
disp(cadenas)

disp("Vectores")
disp(vectores)

disp("Registros")
disp(registros)

disp(filtrada)
```

Python. En Python, una lista puede contener objetos de tipos diferentes y un diccionario puede utilizarse como registro con campos nombrados.

```
def clasificar(x):

    if isinstance(x,int) and not isinstance(x,bool):
        return "entero"

    elif isinstance(x,float):
        return "real"

    elif isinstance(x,str):
        return "cadena"

    elif isinstance(x,list):
        return "vector"
```

```
    elif isinstance(x,dict):
        return "registro"

    else:
        return "otro"


medicion={
    "valor":23.5,
    "unidad":"C"
}

coleccion=[
    25,
    3.14,
    "temperatura",
    "solar",
    [1,2,3],
    medicion
]


suma_escalares=0
texto=""
suma_vectores=0

enteros=0
reales=0
cadenas=0
vectores=0
registros=0

filtrada=[]

categoria_buscada="vector"


for x in coleccion:

    categoria=clasificar(x)

    print(categoria)


    if categoria=="entero":

        enteros=enteros+1
        suma_escalares=suma_escalares+x

    elif categoria=="real":
```

```
            reales=reales+1
            suma_escalares=suma_escalares+x

        elif categoria=="cadena":

            cadenas=cadenas+1

            if texto=="":
                texto=x
            else:
                texto=texto+" "+x

        elif categoria=="vector":

            vectores=vectores+1

            for valor in x:
                suma_vectores=suma_vectores+valor

        elif categoria=="registro":

            registros=registros+1

            print(x["valor"])
            print(x["unidad"])


        if categoria==categoria_buscada:

            filtrada.append(x)


print("Suma de escalares")
print(suma_escalares)

print("Cadenas")
print(texto)

print("Suma de vectores")
print(suma_vectores)

print("Enteros")
print(enteros)

print("Reales")
print(reales)

print("Cadenas")
print(cadenas)
```

```
print("Vectores")
print(vectores)

print("Registros")
print(registros)

print(filtrada)
```

Perl. En Perl, una lista puede contener escalares y referencias de diferentes clases. Los vectores se representan mediante referencias a arreglos y los registros mediante referencias a *hashes*. Como los escalares de Perl no distinguen internamente de la misma manera que otros lenguajes entre entero, real y cadena, la clasificación numérica se realiza mediante `looks_like_number`.

```
use Scalar::Util qw(looks_like_number);


sub clasificar {

    my ($x)=@_;

    if (ref($x) eq 'ARRAY') {

        return 'vector';

    }
    elsif (ref($x) eq 'HASH') {

        return 'registro';

    }
    elsif (looks_like_number($x)) {

        if (int($x)==$x) {
            return 'entero';
        }
        else {
            return 'real';
        }

    }
    else {

        return 'cadena';

    }

}
```

```
$medicion={
    valor=>23.5,
    unidad=>'C'
};

@coleccion=(
    25,
    3.14,
    'temperatura',
    'solar',
    [1,2,3],
    $medicion
);


$suma_escalares=0;
$texto='';
$suma_vectores=0;

$enteros=0;
$reales=0;
$cadenas=0;
$vectores=0;
$registros=0;

@filtrada=();

$categoria_buscada='vector';


for $x (@coleccion) {

    $categoria=clasificar($x);

    print $categoria,"\n";


    if ($categoria eq 'entero') {

        $enteros=$enteros+1;

        $suma_escalares=
            $suma_escalares+$x;

    }
    elsif ($categoria eq 'real') {

        $reales=$reales+1;

        $suma_escalares=
            $suma_escalares+$x;
```

```
    }
    elsif ($categoria eq 'cadena') {

        $cadenas=$cadenas+1;

        if ($texto eq '') {
            $texto=$x;
        }
        else {
            $texto=$texto.' '.$x;
        }

    }
    elsif ($categoria eq 'vector') {

        $vectores=$vectores+1;

        for $valor (@$x) {

            $suma_vectores=
                $suma_vectores+$valor;

        }

    }
    elsif ($categoria eq 'registro') {

        $registros=$registros+1;

        print $x->{valor},"\n";
        print $x->{unidad},"\n";

    }


    if ($categoria eq $categoria_buscada) {

        push @filtrada,$x;

    }

}


print "Suma de escalares\n";
print $suma_escalares,"\n";

print "Cadenas\n";
print $texto,"\n";
```

```
print "Suma de vectores\n";
print $suma_vectores,"\n";

print "Enteros\n";
print $enteros,"\n";

print "Reales\n";
print $reales,"\n";

print "Cadenas\n";
print $cadenas,"\n";

print "Vectores\n";
print $vectores,"\n";

print "Registros\n";
print $registros,"\n";
```

JavaScript. En JavaScript, un arreglo puede contener elementos de diferentes tipos. Un objeto se utiliza aquí como registro. JavaScript posee un único tipo numérico básico, por lo que se utiliza `Number.isInteger` para distinguir conceptualmente enteros y reales.

```
function clasificar(x){

    if(typeof x==="number"){

        if(Number.isInteger(x)){
            return "entero"
        }
        else{
            return "real"
        }

    }
    else if(typeof x==="string"){

        return "cadena"

    }
    else if(Array.isArray(x)){

        return "vector"

    }
    else if(
        typeof x==="object"
        && x!==null
    ){
```

```
            return "registro"

        }
        else{

            return "otro"

        }

}


const medicion={
    valor:23.5,
    unidad:"C"
}

const coleccion=[
    25,
    3.14,
    "temperatura",
    "solar",
    [1,2,3],
    medicion
]


let sumaEscalares=0
let texto=""
let sumaVectores=0

let enteros=0
let reales=0
let cadenas=0
let vectores=0
let registros=0

const filtrada=[]

const categoriaBuscada="vector"


for(const x of coleccion){

    const categoria=clasificar(x)

    console.log(categoria)


    if(categoria==="entero"){
```

```
            enteros=enteros+1
            sumaEscalares=sumaEscalares+x

        }
        else if(categoria==="real"){

            reales=reales+1
            sumaEscalares=sumaEscalares+x

        }
        else if(categoria==="cadena"){

            cadenas=cadenas+1

            if(texto===""){
                texto=x
            }
            else{
                texto=texto+" "+x
            }

        }
        else if(categoria==="vector"){

            vectores=vectores+1

            for(const valor of x){
                sumaVectores=sumaVectores+valor
            }

        }
        else if(categoria==="registro"){

            registros=registros+1

            console.log(x.valor)
            console.log(x.unidad)

        }


        if(categoria===categoriaBuscada){

            filtrada.push(x)

        }

    }


    console.log("Suma de escalares")
```

```
console.log(sumaEscalares)

console.log("Cadenas")
console.log(texto)

console.log("Suma de vectores")
console.log(sumaVectores)

console.log("Enteros")
console.log(enteros)

console.log("Reales")
console.log(reales)

console.log("Cadenas")
console.log(cadenas)

console.log("Vectores")
console.log(vectores)

console.log("Registros")
console.log(registros)

console.log(filtrada)
```

18. **Ejercicio:** Estudiantes con estructuras

    Represente un estudiante mediante un registro o estructura con los campos:

    - `nombre`: cadena no vacía;
    - `documento`: número entero positivo;
    - `carrera`: cadena no vacía;
    - `notas`: vector o colección de números reales comprendidos entre 0 y 10;
    - `regular`: valor lógico.

    Para este ejercicio se adoptarán las siguientes reglas:

    - una evaluación está aprobada cuando su nota es mayor o igual que 4;
    - un estudiante aprobó todas las evaluaciones si posee al menos una nota y todas sus notas son mayores o iguales que 4;
    - un estudiante es regular si posee al menos tres notas, aprobó todas las evaluaciones y su promedio es mayor o igual que 6;
    - el promedio solamente está definido cuando existe al menos una nota.

    Escriba las siguientes funciones:

```
crear_estudiante(nombre, documento, carrera)
agregar_nota(estudiante, nota)
eliminar_nota(estudiante, posicion)
promedio(estudiante)
aprobo_todas(estudiante)
actualizar_regularidad(estudiante)
```

Las funciones `agregar_nota` y `eliminar_nota` deberán devolver el estudiante actualizado.

La posición utilizada por `eliminar_nota` se expresará siempre desde el punto de vista del usuario comenzando en 1, aunque el lenguaje utilice internamente índices que comiencen en 0.

Luego:

a) Cree los siguientes estudiantes:

```
Ana Perez    documento 301    carrera Solar
Luis Gomez   documento 302    carrera Solar
Sofia Ruiz   documento 303    carrera Computacion
```

b) Agregue las siguientes notas:

Ana Perez	8	7	6
Luis Gomez	7	3	8
Sofia Ruiz	9	10	8

c) Muestre el promedio de cada estudiante.

d) Determine para cada estudiante si aprobó todas las evaluaciones.

e) Actualice y muestre su condición de regularidad.

f) Elimine la tercera nota de Ana Perez y vuelva a calcular su condición de regularidad.

g) Agregue nuevamente una nota igual a 9 a Ana Perez y vuelva a actualizar su condición.

h) Almacene los tres estudiantes en una colección.

i) Recorra la colección y muestre el nombre, documento, carrera, promedio y condición de regularidad de cada estudiante.

j) Busque dentro de la colección al estudiante cuyo documento sea `302`.

k) Recorra la colección y muestre solamente los estudiantes cuya carrera sea `"Solar"`.

l) Pruebe qué ocurre al intentar agregar una nota menor que 0 o mayor que 10.

m) Pruebe qué ocurre al intentar eliminar una posición inexistente.

Antes de ejecutar el programa, prediga cuáles de los tres estudiantes serán regulares después de cargar las tres notas iniciales.

Scilab. En Scilab se utiliza `struct` para representar cada estudiante y una `list` para almacenar la colección.

```
function e=crear_estudiante(nombre,documento,carrera)

    if typeof(nombre)<>"string" | nombre=="" then
        error("Nombre no valido")
    end

    if typeof(carrera)<>"string" | carrera=="" then
        error("Carrera no valida")
    end

    valido=(type(documento)==1);

    if valido then
        valido=(size(documento,"*")==1);
    end

    if valido then
        valido=isreal(documento) & ..
               documento==floor(documento) & ..
               documento>0;
    end

    if ~valido then
        error("Documento no valido")
    end

    e=struct( ..
        "nombre",nombre, ..
        "documento",documento, ..
        "carrera",carrera, ..
        "notas",[], ..
        "regular",%f ..
    );

endfunction


function r=aprobo_todas(e)

    r=%f;

    if length(e.notas)>0 then
        r=and(e.notas>=4);
    end

endfunction


function p=promedio(e)
```

```
    if length(e.notas)==0 then
        error("No hay notas")
    end

    p=sum(e.notas)/length(e.notas);

endfunction


function e=actualizar_regularidad(e)

    e.regular=%f;

    if length(e.notas)>=3 then

        if aprobo_todas(e) then

            if promedio(e)>=6 then
                e.regular=%t;
            end

        end

    end

endfunction


function e=agregar_nota(e,nota)

    valido=(type(nota)==1);

    if valido then
        valido=(size(nota,"*")==1) & ..
               isreal(nota) & ..
               ~isnan(nota) & ..
               abs(nota)<>%inf;
    end

    if valido then
        valido=(nota>=0) & (nota<=10);
    end

    if ~valido then
        error("Nota no valida")
    end

    e.notas($+1)=nota;

    e=actualizar_regularidad(e);
```

```
endfunction


function e=eliminar_nota(e,posicion)

    valido=(type(posicion)==1);

    if valido then
        valido=(size(posicion,"*")==1) & ..
               posicion==floor(posicion) & ..
               posicion>=1 & ..
               posicion<=length(e.notas);
    end

    if ~valido then
        error("Posicion no valida")
    end

    e.notas(posicion)=[];

    e=actualizar_regularidad(e);

endfunction


ana=crear_estudiante(
    "Ana Perez",
    301,
    "Solar"
);

luis=crear_estudiante(
    "Luis Gomez",
    302,
    "Solar"
);

sofia=crear_estudiante(
    "Sofia Ruiz",
    303,
    "Computacion"
);


ana=agregar_nota(ana,8);
ana=agregar_nota(ana,7);
ana=agregar_nota(ana,6);

luis=agregar_nota(luis,7);
luis=agregar_nota(luis,3);
luis=agregar_nota(luis,8);
```

```
sofia=agregar_nota(sofia,9);
sofia=agregar_nota(sofia,10);
sofia=agregar_nota(sofia,8);


disp(promedio(ana))
disp(promedio(luis))
disp(promedio(sofia))

disp(aprobo_todas(ana))
disp(aprobo_todas(luis))
disp(aprobo_todas(sofia))

disp(ana.regular)
disp(luis.regular)
disp(sofia.regular)


ana=eliminar_nota(ana,3);
disp(ana.regular)

ana=agregar_nota(ana,9);
disp(ana.regular)


estudiantes=list(ana,luis,sofia);


for i=1:length(estudiantes)

    e=estudiantes(i);

    disp(e.nombre)
    disp(e.documento)
    disp(e.carrera)
    disp(promedio(e))
    disp(e.regular)

end


documento_buscado=302;

for i=1:length(estudiantes)

    e=estudiantes(i);

    if e.documento==documento_buscado then
        disp(e.nombre)
    end
```

```
end


for i=1:length(estudiantes)

    e=estudiantes(i);

    if e.carrera=="Solar" then
        disp(e.nombre)
    end

end
```

Octave. En Octave se utiliza una estructura para cada estudiante y un arreglo de estructuras para la colección.

```
function e=crear_estudiante(nombre,documento,carrera)

  if !ischar(nombre) || isempty(strtrim(nombre))
    error("Nombre no valido")
  endif

  if !ischar(carrera) || isempty(strtrim(carrera))
    error("Carrera no valida")
  endif

  valido=isnumeric(documento) ...
         && isscalar(documento) ...
         && isreal(documento) ...
         && isfinite(documento) ...
         && documento==fix(documento) ...
         && documento>0;

  if !valido
    error("Documento no valido")
  endif

  e=struct(
    "nombre",nombre,
    "documento",documento,
    "carrera",carrera,
    "notas",[],
    "regular",false
  );

endfunction


function r=aprobo_todas(e)
```

```
  r=false;

  if length(e.notas)>0
    r=all(e.notas>=4);
  endif

endfunction


function p=promedio(e)

  if isempty(e.notas)
    error("No hay notas")
  endif

  p=sum(e.notas)/length(e.notas);

endfunction


function e=actualizar_regularidad(e)

  e.regular=false;

  if length(e.notas)>=3

    if aprobo_todas(e)

      if promedio(e)>=6
        e.regular=true;
      endif

    endif

  endif

endfunction


function e=agregar_nota(e,nota)

  valido=isnumeric(nota) ...
         && isscalar(nota) ...
         && isreal(nota) ...
         && isfinite(nota) ...
         && nota>=0 ...
         && nota<=10;

  if !valido
    error("Nota no valida")
```

```
  endif

  e.notas(end+1)=nota;

  e=actualizar_regularidad(e);

endfunction


function e=eliminar_nota(e,posicion)

  valido=isnumeric(posicion) ...
         && isscalar(posicion) ...
         && posicion==fix(posicion) ...
         && posicion>=1 ...
         && posicion<=length(e.notas);

  if !valido
    error("Posicion no valida")
  endif

  e.notas(posicion)=[];

  e=actualizar_regularidad(e);

endfunction


ana=crear_estudiante(
  "Ana Perez",
  301,
  "Solar"
);

luis=crear_estudiante(
  "Luis Gomez",
  302,
  "Solar"
);

sofia=crear_estudiante(
  "Sofia Ruiz",
  303,
  "Computacion"
);


ana=agregar_nota(ana,8);
ana=agregar_nota(ana,7);
ana=agregar_nota(ana,6);
```

```
luis=agregar_nota(luis,7);
luis=agregar_nota(luis,3);
luis=agregar_nota(luis,8);

sofia=agregar_nota(sofia,9);
sofia=agregar_nota(sofia,10);
sofia=agregar_nota(sofia,8);


disp(promedio(ana))
disp(promedio(luis))
disp(promedio(sofia))

disp(aprobo_todas(ana))
disp(aprobo_todas(luis))
disp(aprobo_todas(sofia))

disp(ana.regular)
disp(luis.regular)
disp(sofia.regular)


ana=eliminar_nota(ana,3);
disp(ana.regular)

ana=agregar_nota(ana,9);
disp(ana.regular)


estudiantes=[ana,luis,sofia];


for i=1:length(estudiantes)

  e=estudiantes(i);

  disp(e.nombre)
  disp(e.documento)
  disp(e.carrera)
  disp(promedio(e))
  disp(e.regular)

endfor


documento_buscado=302;

for i=1:length(estudiantes)

  e=estudiantes(i);
```

```
  if e.documento==documento_buscado
    disp(e.nombre)
  endif

endfor


for i=1:length(estudiantes)

  e=estudiantes(i);

  if strcmp(e.carrera,"Solar")
    disp(e.nombre)
  endif

endfor
```

Python. En Python se utiliza un diccionario para representar cada estudiante y una lista para almacenar la colección.

```
import math


def crear_estudiante(nombre,documento,carrera):

    if not isinstance(nombre,str) or nombre.strip()=="":
        raise ValueError("Nombre no valido")

    if not isinstance(carrera,str) or carrera.strip()=="":
        raise ValueError("Carrera no valida")

    valido=(
        isinstance(documento,int)
        and not isinstance(documento,bool)
        and documento>0
    )

    if not valido:
        raise ValueError("Documento no valido")

    return {
        "nombre":nombre,
        "documento":documento,
        "carrera":carrera,
        "notas":[],
        "regular":False
    }


def aprobo_todas(e):
```

```
    if len(e["notas"])==0:
        return False

    for nota in e["notas"]:

        if nota<4:
            return False

    return True


def promedio(e):

    if len(e["notas"])==0:
        raise ValueError("No hay notas")

    suma=0

    for nota in e["notas"]:
        suma=suma+nota

    return suma/len(e["notas"])


def actualizar_regularidad(e):

    e["regular"]=False

    if len(e["notas"])>=3:

        if aprobo_todas(e):

            if promedio(e)>=6:
                e["regular"]=True

    return e


def agregar_nota(e,nota):

    valido=(
        isinstance(nota,(int,float))
        and not isinstance(nota,bool)
        and math.isfinite(nota)
        and 0<=nota<=10
    )

    if not valido:
        raise ValueError("Nota no valida")
```

```
    e["notas"].append(nota)

    return actualizar_regularidad(e)


def eliminar_nota(e,posicion):

    valido=(
        isinstance(posicion,int)
        and not isinstance(posicion,bool)
        and 1<=posicion<=len(e["notas"])
    )

    if not valido:
        raise ValueError("Posicion no valida")

    del e["notas"][posicion-1]

    return actualizar_regularidad(e)


ana=crear_estudiante(
    "Ana Perez",
    301,
    "Solar"
)

luis=crear_estudiante(
    "Luis Gomez",
    302,
    "Solar"
)

sofia=crear_estudiante(
    "Sofia Ruiz",
    303,
    "Computacion"
)


ana=agregar_nota(ana,8)
ana=agregar_nota(ana,7)
ana=agregar_nota(ana,6)

luis=agregar_nota(luis,7)
luis=agregar_nota(luis,3)
luis=agregar_nota(luis,8)

sofia=agregar_nota(sofia,9)
sofia=agregar_nota(sofia,10)
sofia=agregar_nota(sofia,8)
```

```
print(promedio(ana))
print(promedio(luis))
print(promedio(sofia))

print(aprobo_todas(ana))
print(aprobo_todas(luis))
print(aprobo_todas(sofia))

print(ana["regular"])
print(luis["regular"])
print(sofia["regular"])


ana=eliminar_nota(ana,3)
print(ana["regular"])

ana=agregar_nota(ana,9)
print(ana["regular"])


estudiantes=[ana,luis,sofia]


for e in estudiantes:

    print(e["nombre"])
    print(e["documento"])
    print(e["carrera"])
    print(promedio(e))
    print(e["regular"])


documento_buscado=302

for e in estudiantes:

    if e["documento"]==documento_buscado:
        print(e["nombre"])


for e in estudiantes:

    if e["carrera"]=="Solar":
        print(e["nombre"])
```

Perl. En Perl, cada estudiante se representa mediante una referencia a un *hash* y la colección mediante un arreglo de esas referencias.

```
```

```
use Scalar::Util qw(looks_like_number);


sub crear_estudiante {

    my ($nombre,$documento,$carrera)=@_;

    die "Nombre no valido\n"
        if !defined($nombre) || $nombre eq '';

    die "Carrera no valida\n"
        if !defined($carrera) || $carrera eq '';

    my $valido=
        looks_like_number($documento)
        && int($documento)==$documento
        && $documento>0;

    die "Documento no valido\n"
        unless $valido;

    return {
        nombre=>$nombre,
        documento=>$documento,
        carrera=>$carrera,
        notas=>[],
        regular=>0
    };

}


sub aprobo_todas {

    my ($e)=@_;

    return 0
        if @{$e->{notas}}==0;

    for my $nota (@{$e->{notas}}) {

        if ($nota<4) {
            return 0;
        }

    }

    return 1;

}
```

```
sub promedio {

    my ($e)=@_;

    die "No hay notas\n"
        if @{$e->{notas}}==0;

    my $suma=0;

    for my $nota (@{$e->{notas}}) {
        $suma=$suma+$nota;
    }

    return $suma/@{$e->{notas}};

}


sub actualizar_regularidad {

    my ($e)=@_;

    $e->{regular}=0;

    if (@{$e->{notas}}>=3) {

        if (aprobo_todas($e)) {

            if (promedio($e)>=6) {
                $e->{regular}=1;
            }

        }

    }

    return $e;

}


sub agregar_nota {

    my ($e,$nota)=@_;

    my $valido=
        looks_like_number($nota)
        && $nota>=0
        && $nota<=10;
```

```
    die "Nota no valida\n"
        unless $valido;

    push @{$e->{notas}},$nota;

    return actualizar_regularidad($e);

}


sub eliminar_nota {

    my ($e,$posicion)=@_;

    my $valido=
        looks_like_number($posicion)
        && int($posicion)==$posicion
        && $posicion>=1
        && $posicion<=@{$e->{notas}};

    die "Posicion no valida\n"
        unless $valido;

    splice(
        @{$e->{notas}},
        $posicion-1,
        1
    );

    return actualizar_regularidad($e);

}


$ana=crear_estudiante(
    'Ana Perez',
    301,
    'Solar'
);

$luis=crear_estudiante(
    'Luis Gomez',
    302,
    'Solar'
);

$sofia=crear_estudiante(
    'Sofia Ruiz',
    303,
    'Computacion'
);
```

```
$ana=agregar_nota($ana,8);
$ana=agregar_nota($ana,7);
$ana=agregar_nota($ana,6);

$luis=agregar_nota($luis,7);
$luis=agregar_nota($luis,3);
$luis=agregar_nota($luis,8);

$sofia=agregar_nota($sofia,9);
$sofia=agregar_nota($sofia,10);
$sofia=agregar_nota($sofia,8);


print promedio($ana),"\n";
print promedio($luis),"\n";
print promedio($sofia),"\n";

print aprobo_todas($ana),"\n";
print aprobo_todas($luis),"\n";
print aprobo_todas($sofia),"\n";

print $ana->{regular},"\n";
print $luis->{regular},"\n";
print $sofia->{regular},"\n";


$ana=eliminar_nota($ana,3);
print $ana->{regular},"\n";

$ana=agregar_nota($ana,9);
print $ana->{regular},"\n";


@estudiantes=($ana,$luis,$sofia);


for $e (@estudiantes) {

    print $e->{nombre},"\n";
    print $e->{documento},"\n";
    print $e->{carrera},"\n";
    print promedio($e),"\n";
    print $e->{regular},"\n";

}


$documento_buscado=302;
```

```
for $e (@estudiantes) {

    if ($e->{documento}==$documento_buscado) {
        print $e->{nombre},"\n";
    }

}


for $e (@estudiantes) {

    if ($e->{carrera} eq 'Solar') {
        print $e->{nombre},"\n";
    }

}
```

JavaScript. En JavaScript se utiliza un objeto para representar cada estudiante y un arreglo para almacenar la colección.

```
function crearEstudiante(nombre,documento,carrera){

    if(
        typeof nombre!=="string"
        || nombre.trim()===""
    ){
        throw new Error("Nombre no valido")
    }

    if(
        typeof carrera!=="string"
        || carrera.trim()===""
    ){
        throw new Error("Carrera no valida")
    }

    if(
        !Number.isInteger(documento)
        || documento<=0
    ){
        throw new Error("Documento no valido")
    }

    return {
        nombre:nombre,
        documento:documento,
        carrera:carrera,
        notas:[],
        regular:false
    }
```

```
}


function aproboTodas(e){

    if(e.notas.length===0){
        return false
    }

    for(const nota of e.notas){

        if(nota<4){
            return false
        }

    }

    return true

}


function promedio(e){

    if(e.notas.length===0){
        throw new Error("No hay notas")
    }

    let suma=0

    for(const nota of e.notas){
        suma=suma+nota
    }

    return suma/e.notas.length

}


function actualizarRegularidad(e){

    e.regular=false

    if(e.notas.length>=3){

        if(aproboTodas(e)){

            if(promedio(e)>=6){
                e.regular=true
            }
```

```
        }

    }

    return e

}


function agregarNota(e,nota){

    if(
        typeof nota!=="number"
        || !Number.isFinite(nota)
        || nota<0
        || nota>10
    ){
        throw new Error("Nota no valida")
    }

    e.notas.push(nota)

    return actualizarRegularidad(e)

}


function eliminarNota(e,posicion){

    if(
        !Number.isInteger(posicion)
        || posicion<1
        || posicion>e.notas.length
    ){
        throw new Error("Posicion no valida")
    }

    e.notas.splice(posicion-1,1)

    return actualizarRegularidad(e)

}


let ana=crearEstudiante(
    "Ana Perez",
    301,
    "Solar"
)
```

```
let luis=crearEstudiante(
    "Luis Gomez",
    302,
    "Solar"
)

let sofia=crearEstudiante(
    "Sofia Ruiz",
    303,
    "Computacion"
)


ana=agregarNota(ana,8)
ana=agregarNota(ana,7)
ana=agregarNota(ana,6)

luis=agregarNota(luis,7)
luis=agregarNota(luis,3)
luis=agregarNota(luis,8)

sofia=agregarNota(sofia,9)
sofia=agregarNota(sofia,10)
sofia=agregarNota(sofia,8)


console.log(promedio(ana))
console.log(promedio(luis))
console.log(promedio(sofia))

console.log(aproboTodas(ana))
console.log(aproboTodas(luis))
console.log(aproboTodas(sofia))

console.log(ana.regular)
console.log(luis.regular)
console.log(sofia.regular)


ana=eliminarNota(ana,3)
console.log(ana.regular)

ana=agregarNota(ana,9)
console.log(ana.regular)


const estudiantes=[ana,luis,sofia]


for(const e of estudiantes){
```

```
        console.log(e.nombre)
        console.log(e.documento)
        console.log(e.carrera)
        console.log(promedio(e))
        console.log(e.regular)

}


const documentoBuscado=302

for(const e of estudiantes){

    if(e.documento===documentoBuscado){
        console.log(e.nombre)
    }

}


for(const e of estudiantes){

    if(e.carrera==="Solar"){
        console.log(e.nombre)
    }

}
```

Con las tres notas iniciales, Ana y Sofia deben quedar regulares; Luis no, porque posee una nota menor que 4.

19. **Ejercicio:** Alias y copias

    Considere primero el siguiente experimento conceptual:

```
a <- vector [1, 2, 3]
b <- a

modificar el primer elemento de b
asignándole el valor 100

mostrar a
mostrar b
```

    Antes de ejecutarlo, prediga si la modificación realizada mediante `b` modificará también el contenido observado mediante `a`.

    El objetivo del ejercicio es distinguir tres situaciones conceptuales:

    - dos variables que pueden compartir el mismo objeto;

- una copia superficial, en la que se crea un contenedor nuevo pero algunos elementos internos pueden continuar compartidos;
- una copia profunda, en la que también se copian los objetos contenidos.

No se espera que los cinco lenguajes produzcan el mismo resultado ante una asignación. Precisamente deberá observarse y explicar esa diferencia.

Realice los siguientes experimentos en este orden:

a) Cree el vector

$$a = [1, 2, 3]$$

y realice una asignación directa de `a` a una segunda variable `b`.

b) Modifique el primer elemento de `b` asignándole 100 y muestre después `a` y `b`.

c) Indique si la asignación produjo, desde el punto de vista observable del programa, dos valores independientes o dos accesos a los mismos datos modificables.

d) Construya ahora un registro denominado `persona1` con los datos

```
nombre = "Ana"
notas = [8, 9, 7]
```

e) Obtenga una segunda variable `persona2` mediante la asignación o copia ordinaria correspondiente al lenguaje.

f) Modifique solamente la primera nota de `persona2`, asignándole el valor 10.

g) Muestre las notas de `persona1` y de `persona2` y explique el resultado.

h) Cuando el lenguaje permita distinguir explícitamente una copia superficial de una copia profunda, realice ambas y modifique un elemento del vector interno de cada copia.

i) Explique qué se copia en cada caso: el contenedor exterior, los objetos interiores o ambos.

j) Escriba una función denominada `modificar` que reciba un vector y asigne 100 a su primer elemento.

k) Ejecute

```
a <- [1, 2, 3]
modificar(a)
mostrar a
```

y determine si la modificación realizada dentro de la función resulta visible fuera de ella.

l) Explique los resultados obtenidos sin utilizar de manera intercambiable las expresiones “paso por referencia”, “alias”, “copia” y “copia profunda”.

En todos los lenguajes deberán utilizarse los mismos datos y realizarse las mismas modificaciones. Los resultados pueden ser diferentes debido al modelo de datos de cada lenguaje.

Scilab. En Scilab se realizarán los experimentos con matrices y estructuras. Observe experimentalmente el efecto de la asignación antes de generalizar el resultado a otros tipos de objetos del lenguaje.

Primer experimento:

```
a=[1 2 3];

b=a;

b(1)=100;

disp("a")
disp(a)

disp("b")
disp(b)
```

Experimento con una estructura:

```
persona1=struct( ..
    "nombre","Ana", ..
    "notas",[8 9 7] ..
);

persona2=persona1;

persona2.notas(1)=10;

disp("persona1")
disp(persona1.notas)

disp("persona2")
disp(persona2.notas)
```

Experimento con una función:

```
function v=modificar(v)

    v(1)=100;

endfunction


a=[1 2 3];

modificar(a);

disp("a despues de modificar(a)")
disp(a)
```

```
b=modificar(a);

disp("resultado devuelto")
disp(b)
```

Octave. Octave utiliza copia diferida para matrices, celdas y estructuras. Aunque una implementación puede compartir inicialmente el almacenamiento, una modificación produce el comportamiento observable de valores independientes.

Primer experimento:

```
a=[1 2 3];

b=a;

b(1)=100;

disp("a")
disp(a)

disp("b")
disp(b)
```

Experimento con una estructura:

```
persona1=struct(
  "nombre","Ana",
  "notas",[8 9 7]
);

persona2=persona1;

persona2.notas(1)=10;

disp("persona1")
disp(persona1.notas)

disp("persona2")
disp(persona2.notas)
```

Experimento con una función:

```
function v=modificar(v)

  v(1)=100;

endfunction
```

```
a=[1 2 3];

modificar(a);

disp("a despues de modificar(a)")
disp(a)


b=modificar(a);

disp("resultado devuelto")
disp(b)
```

Python. En Python, una asignación no copia una lista o un diccionario: el nuevo nombre puede quedar asociado al mismo objeto. El módulo `copy` permite realizar copias superficiales y profundas.

Primer experimento:

```
a=[1,2,3]

b=a

b[0]=100

print("a")
print(a)

print("b")
print(b)
```

Experimento con asignación directa:

```
persona1={
    "nombre":"Ana",
    "notas":[8,9,7]
}

persona2=persona1

persona2["notas"][0]=10

print("persona1")
print(persona1["notas"])

print("persona2")
print(persona2["notas"])
```

Copia superficial:

```
import copy

persona1={
    "nombre":"Ana",
    "notas":[8,9,7]
}

persona2=copy.copy(persona1)

persona2["notas"][0]=10

print("persona1")
print(persona1["notas"])

print("persona2")
print(persona2["notas"])
```

Copia profunda:

```
import copy

persona1={
    "nombre":"Ana",
    "notas":[8,9,7]
}

persona2=copy.deepcopy(persona1)

persona2["notas"][0]=10

print("persona1")
print(persona1["notas"])

print("persona2")
print(persona2["notas"])
```

Experimento con una función:

```
def modificar(v):

    v[0]=100


a=[1,2,3]

modificar(a)
```

```
print(a)
```

Perl. En Perl se representan aquí los vectores mediante referencias a arreglos y los registros mediante referencias a *hashes*. Copiar una referencia produce otra referencia al mismo objeto. Una copia superficial puede construirse creando un nuevo arreglo o *hash*; para la copia profunda se utiliza `dclone` de `Storable`.

Primer experimento:

```
$a=[1,2,3];

$b=$a;

$b->[0]=100;

print "a\n";
print join(" ",@$a),"\n";

print "b\n";
print join(" ",@$b),"\n";
```

Experimento con asignación directa:

```
$persona1={
    nombre=>'Ana',
    notas=>[8,9,7]
};

$persona2=$persona1;

$persona2->{notas}[0]=10;

print "persona1\n";
print join(
    " ",
    @{$persona1->{notas}}
),"\n";

print "persona2\n";
print join(
    " ",
    @{$persona2->{notas}}
),"\n";
```

Copia superficial:

```
$persona1={
    nombre=>'Ana',
    notas=>[8,9,7]
```

```
};

$persona2={%$persona1};

$persona2->{notas}[0]=10;

print "persona1\n";
print join(
    " ",
    @{$persona1->{notas}}
),"\n";

print "persona2\n";
print join(
    " ",
    @{$persona2->{notas}}
),"\n";
```

Copia profunda:

```
use Storable qw(dclone);

$persona1={
    nombre=>'Ana',
    notas=>[8,9,7]
};

$persona2=dclone($persona1);

$persona2->{notas}[0]=10;

print "persona1\n";
print join(
    " ",
    @{$persona1->{notas}}
),"\n";

print "persona2\n";
print join(
    " ",
    @{$persona2->{notas}}
),"\n";
```

Experimento con una función:

```
sub modificar {

    my ($v)=@_;

    $v->[0]=100;
```

```
}


$a=[1,2,3];

modificar($a);

print join(" ",@$a),"\n";
```

JavaScript. En JavaScript, los arreglos y objetos son valores de referencia. Una asignación directa conserva la referencia al mismo objeto. El operador de expansión permite construir una copia superficial y `structuredClone` permite realizar una copia profunda de los datos utilizados en este ejercicio.

Primer experimento:

```
const a=[1,2,3]

const b=a

b[0]=100

console.log("a")
console.log(a)

console.log("b")
console.log(b)
```

Experimento con asignación directa:

```
const persona1={
    nombre:"Ana",
    notas:[8,9,7]
}

const persona2=persona1

persona2.notas[0]=10

console.log("persona1")
console.log(persona1.notas)

console.log("persona2")
console.log(persona2.notas)
```

Copia superficial:

```
const persona1={
```

```
    nombre:"Ana",
    notas:[8,9,7]
}

const persona2={...persona1}

persona2.notas[0]=10

console.log("persona1")
console.log(persona1.notas)

console.log("persona2")
console.log(persona2.notas)
```

Copia profunda:

```
const persona1={
    nombre:"Ana",
    notas:[8,9,7]
}

const persona2=structuredClone(persona1)

persona2.notas[0]=10

console.log("persona1")
console.log(persona1.notas)

console.log("persona2")
console.log(persona2.notas)
```

Experimento con una función:

```
function modificar(v){

    v[0]=100

}


const a=[1,2,3]

modificar(a)

console.log(a)
```

5.11.6. Calidad, pruebas y rendimiento

20. **Ejercicio:** Depuración de normalización

    Se desea escribir una función que reciba un vector numérico real no vacío `v` y produzca:

    - la media de sus elementos;
    - el valor mínimo;
    - el valor máximo;
    - un vector normalizado en el intervalo $[0, 1]$ mediante

    $$y_i = \frac{v_i - \text{mín}(v)}{\text{máx}(v) - \text{mín}(v)}.$$

    Si todos los elementos de `v` son iguales, el vector normalizado deberá ser un vector de ceros.

    El algoritmo correcto debe recorrer todos los elementos del vector, calcular la suma, el mínimo y el máximo, y solamente después realizar la normalización.

    Se proporciona a continuación una versión deliberadamente defectuosa del programa.

    Todos los programas contienen conceptualmente los mismos errores:

    a) la suma no se inicializa correctamente;
    b) el máximo se inicializa con un valor que no necesariamente pertenece al vector;
    c) el recorrido no procesa todos los elementos;
    d) no se trata correctamente el caso en que todos los valores son iguales.

    Para cada error:

    I) prediga qué consecuencia puede producir;
    II) ejecute el programa y observe el resultado;
    III) identifique la instrucción responsable;
    IV) construya un caso de prueba pequeño que permita detectar el error;
    V) explique por qué el algoritmo es incorrecto;
    VI) corrija el programa;
    VII) vuelva a ejecutar todas las pruebas después de cada corrección.

    Utilice como mínimo los siguientes casos:

    $$[2, 4, 6, 8],$$

    $$[-8, -5, -2],$$

$$[5, 5, 5],$$

y

$$[7].$$

Para cada caso determine previamente la media, el mínimo y el máximo esperados.

Al finalizar, compruebe que la versión corregida:

- recorre todos los elementos;
- calcula correctamente la media;
- funciona con vectores cuyos elementos son todos negativos;
- funciona con un vector de un solo elemento;
- evita una división por cero para un vector constante.

Scilab. En Scilab, los índices comienzan en 1. La versión siguiente contiene intencionalmente los cuatro errores indicados en la consigna.

Versión defectuosa:

```
function [y,media,minimo,maximo]=analizar(v)

    n=length(v);

    suma=v(1);

    minimo=v(1);
    maximo=0;

    for i=1:n-1

        suma=suma+v(i);

        if v(i)<minimo then
            minimo=v(i);
        end

        if v(i)>maximo then
            maximo=v(i);
        end

    end

    media=suma/n;

    y=(v-minimo)/(maximo-minimo);

endfunction
```

Versión corregida:

```
function [y,media,minimo,maximo]=analizar(v)

    n=length(v);

    if n==0 then
        error("El vector no puede estar vacio")
    end

    suma=0;

    minimo=v(1);
    maximo=v(1);

    for i=1:n

        suma=suma+v(i);

        if v(i)<minimo then
            minimo=v(i);
        end

        if v(i)>maximo then
            maximo=v(i);
        end

    end

    media=suma/n;

    if maximo==minimo then

        y=zeros(size(v,1),size(v,2));

    else

        y=(v-minimo)/(maximo-minimo);

    end

endfunction


[y,media,minimo,maximo]=analizar([2 4 6 8])
[y,media,minimo,maximo]=analizar([-8 -5 -2])
[y,media,minimo,maximo]=analizar([5 5 5])
[y,media,minimo,maximo]=analizar([7])
```

Octave. En Octave, los índices comienzan en 1. La versión defectuosa reproduce

los mismos cuatro errores lógicos.

Versión defectuosa:

```
function [y,media,minimo,maximo]=analizar(v)

  n=length(v);

  suma=v(1);

  minimo=v(1);
  maximo=0;

  for i=1:n-1

    suma=suma+v(i);

    if v(i)<minimo
      minimo=v(i);
    endif

    if v(i)>maximo
      maximo=v(i);
    endif

  endfor

  media=suma/n;

  y=(v-minimo)/(maximo-minimo);

endfunction
```

Versión corregida:

```
function [y,media,minimo,maximo]=analizar(v)

  n=length(v);

  if n==0
    error("El vector no puede estar vacio")
  endif

  suma=0;

  minimo=v(1);
  maximo=v(1);

  for i=1:n

    suma=suma+v(i);
```

```
    if v(i)<minimo
      minimo=v(i);
    endif

    if v(i)>maximo
      maximo=v(i);
    endif

  endfor

  media=suma/n;

  if maximo==minimo

    y=zeros(size(v));

  else

    y=(v-minimo)/(maximo-minimo);

  endif

endfunction


[y,media,minimo,maximo]=analizar([2 4 6 8])
[y,media,minimo,maximo]=analizar([-8 -5 -2])
[y,media,minimo,maximo]=analizar([5 5 5])
[y,media,minimo,maximo]=analizar([7])
```

Python. En Python, los índices comienzan en 0. Por eso el mismo error de recorrido se expresa mediante `range(n-1)`.

Versión defectuosa:

```
def analizar(v):

    n=len(v)

    suma=v[0]

    minimo=v[0]
    maximo=0

    for i in range(n-1):

        suma=suma+v[i]

        if v[i]<minimo:
            minimo=v[i]
```

```
        if v[i]>maximo:
            maximo=v[i]

    media=suma/n

    y=[]

    for x in v:
        y.append(
            (x-minimo)/(maximo-minimo)
        )

    return y,media,minimo,maximo
```

Versión corregida:

```
def analizar(v):

    n=len(v)

    if n==0:
        raise ValueError(
            "El vector no puede estar vacio"
        )

    suma=0

    minimo=v[0]
    maximo=v[0]

    for i in range(n):

        suma=suma+v[i]

        if v[i]<minimo:
            minimo=v[i]

        if v[i]>maximo:
            maximo=v[i]

    media=suma/n

    y=[]

    if maximo==minimo:

        for x in v:
            y.append(0)

    else:
```

```
        for x in v:

            y.append(
                (x-minimo)/
                (maximo-minimo)
            )

    return y,media,minimo,maximo


print(analizar([2,4,6,8]))
print(analizar([-8,-5,-2]))
print(analizar([5,5,5]))
print(analizar([7]))
```

Perl. En Perl se utiliza una referencia a un arreglo para representar el vector. El índice inicial es 0. La versión defectuosa contiene los mismos cuatro errores.

Versión defectuosa:

```
sub analizar {

    my ($v)=@_;

    my $n=@$v;

    my $suma=$v->[0];

    my $minimo=$v->[0];
    my $maximo=0;

    for (
        my $i=0;
        $i<$n-1;
        $i++
    ) {

        $suma=$suma+$v->[$i];

        if ($v->[$i]<$minimo) {
            $minimo=$v->[$i];
        }

        if ($v->[$i]>$maximo) {
            $maximo=$v->[$i];
        }

    }

    my $media=$suma/$n;
```

```
    my @y;

    for my $x (@$v) {

        push @y,
            ($x-$minimo)/
            ($maximo-$minimo);

    }

    return (
        \@y,
        $media,
        $minimo,
        $maximo
    );

}
```

Versión corregida:

```
sub analizar {

    my ($v)=@_;

    my $n=@$v;

    die "El vector no puede estar vacio\n"
        if $n==0;

    my $suma=0;

    my $minimo=$v->[0];
    my $maximo=$v->[0];

    for (
        my $i=0;
        $i<$n;
        $i++
    ) {

        $suma=$suma+$v->[$i];

        if ($v->[$i]<$minimo) {
            $minimo=$v->[$i];
        }

        if ($v->[$i]>$maximo) {
            $maximo=$v->[$i];
        }
```

```
    }

    my $media=$suma/$n;

    my @y;

    if ($maximo==$minimo) {

        for my $x (@$v) {
            push @y,0;
        }

    }
    else {

        for my $x (@$v) {

            push @y,
                ($x-$minimo)/
                ($maximo-$minimo);

        }

    }

    return (
        \@y,
        $media,
        $minimo,
        $maximo
    );

}


for my $v (
    [2,4,6,8],
    [-8,-5,-2],
    [5,5,5],
    [7]
) {

    my (
        $y,
        $media,
        $minimo,
        $maximo
    )=analizar($v);

    print "media = $media\n";
```

```
    print "minimo = $minimo\n";
    print "maximo = $maximo\n";
    print join(" ",@$y),"\n";

}
```

JavaScript. En JavaScript, el vector se representa mediante un arreglo y los índices comienzan en 0. La versión defectuosa contiene los mismos cuatro errores lógicos.

Versión defectuosa:

```
function analizar(v){

    let n=v.length

    let suma=v[0]

    let minimo=v[0]
    let maximo=0

    for(let i=0;i<n-1;i++){

        suma=suma+v[i]

        if(v[i]<minimo){
            minimo=v[i]
        }

        if(v[i]>maximo){
            maximo=v[i]
        }

    }

    let media=suma/n

    let y=[]

    for(const x of v){

        y.push(
            (x-minimo)/(maximo-minimo)
        )

    }

    return {
        y:y,
        media:media,
        minimo:minimo,
```

```
            maximo:maximo
        }

    }
```

Versión corregida:

```
    function analizar(v){

        let n=v.length

        if(n===0){
            throw new Error(
                "El vector no puede estar vacio"
            )
        }

        let suma=0

        let minimo=v[0]
        let maximo=v[0]

        for(let i=0;i<n;i++){

            suma=suma+v[i]

            if(v[i]<minimo){
                minimo=v[i]
            }

            if(v[i]>maximo){
                maximo=v[i]
            }

        }

        let media=suma/n

        let y=[]

        if(maximo===minimo){

            for(const x of v){
                y.push(0)
            }

        }
        else{

            for(const x of v){
```

```
                y.push(
                    (x-minimo)/
                    (maximo-minimo)
                )

            }

        }

        return {
            y:y,
            media:media,
            minimo:minimo,
            maximo:maximo
        }

    }


    console.log(
        analizar([2,4,6,8])
    )

    console.log(
        analizar([-8,-5,-2])
    )

    console.log(
        analizar([5,5,5])
    )

    console.log(
        analizar([7])
    )
```

21. **Ejercicio:** Reescritura de código

    Considere el siguiente algoritmo, escrito deliberadamente de manera poco clara:

```
definir f(x, y, z):
    a <- x^2 + y^2
    b <- raiz_cuadrada(a)
    c <- b / z
    si c > 1 entonces r <- 1; en otro caso r <- c
    devolver r
```

    La intención de la función es calcular

$$d = \sqrt{x^2 + y^2},$$

formar posteriormente la razón

$$q = \frac{d}{z},$$

y devolver

$$r = \begin{cases} 1, & q > 1, \\ q, & q \leq 1. \end{cases}$$

Para evitar ambigüedades, adopte para este ejercicio el siguiente contrato:

- `x`, `y` y `z` deben ser números reales escalares y finitos;
- `z` debe ser estrictamente positivo;
- la función devuelve un único número real comprendido entre 0 y 1;
- esta versión de la función trabaja solamente con escalares.

Antes de reescribir el programa, identifique en la versión original problemas relacionados con:

- nombres poco descriptivos;
- ausencia de documentación;
- ausencia de validación;
- posibilidad de división por cero;
- comportamiento no definido para un denominador negativo;
- instrucciones excesivamente compactas;
- falta de claridad acerca de los tipos de argumentos admitidos;
- falta de casos de prueba.

Luego escriba una nueva función denominada

```
razon_limitada(x, y, z)
```

siguiendo estas reglas:

a) utilice nombres descriptivos para las variables internas;
b) documente el propósito de la función;
c) valide los tres argumentos antes de realizar los cálculos;
d) rechace valores no numéricos, complejos, no finitos o no escalares;
e) rechace `z` cuando sea menor o igual que cero;
f) utilice una instrucción principal por línea;
g) mantenga una sangría consistente;
h) produzca mensajes de error claros;

i) calcule primero la distancia;

j) calcule después la razón;

k) limite finalmente el resultado a un máximo de 1 mediante una estructura condicional;

l) escriba y ejecute los mismos casos de prueba en todos los lenguajes.

Utilice como mínimo los siguientes casos:

(x, y, z)	resultado esperado
$(3, 4, 10)$	0,5
$(3, 4, 5)$	1
$(3, 4, 2)$	1
$(0, 0, 2)$	0

Pruebe además:

- `z=0`;
- `z=-2`;
- una entrada no numérica.

Explique por qué una versión más extensa puede ser preferible a una versión muy compacta cuando mejora la legibilidad, permite validar argumentos y facilita las pruebas y la depuración.

Scilab. En Scilab se comprueba explícitamente que cada argumento sea un escalar numérico real y finito.

```
function resultado=razon_limitada(x,y,z)

    // Calcula sqrt(x^2+y^2)/z
    // y limita el resultado a un maximo de 1.

    valido_x=(type(x)==1) & ..
             (size(x,"*")==1) & ..
             isreal(x);

    if valido_x then
        valido_x=~isnan(x) & abs(x)<>%inf;
    end

    valido_y=(type(y)==1) & ..
             (size(y,"*")==1) & ..
             isreal(y);

    if valido_y then
        valido_y=~isnan(y) & abs(y)<>%inf;
    end
```

```
    valido_z=(type(z)==1) & ..
             (size(z,"*")==1) & ..
             isreal(z);

    if valido_z then
        valido_z=~isnan(z) & abs(z)<>%inf;
    end

    if ~valido_x | ~valido_y | ~valido_z then
        error("Los argumentos deben ser escalares reales y
           finitos")
    end

    if z<=0 then
        error("z debe ser estrictamente positivo")
    end

    suma_cuadrados=x^2+y^2;

    distancia=sqrt(suma_cuadrados);

    razon=distancia/z;

    if razon>1 then
        resultado=1;
    else
        resultado=razon;
    end

endfunction


disp(razon_limitada(3,4,10))
disp(razon_limitada(3,4,5))
disp(razon_limitada(3,4,2))
disp(razon_limitada(0,0,2))
```

Octave. En Octave se utiliza la misma secuencia de validación y cálculo.

```
function resultado=razon_limitada(x,y,z)

  % Calcula sqrt(x^2+y^2)/z
  % y limita el resultado a un maximo de 1.

  valido_x=isnumeric(x) ...
           && isscalar(x) ...
           && isreal(x) ...
           && isfinite(x);

  valido_y=isnumeric(y) ...
```

```
            && isscalar(y) ...
            && isreal(y) ...
            && isfinite(y);

  valido_z=isnumeric(z) ...
            && isscalar(z) ...
            && isreal(z) ...
            && isfinite(z);

  if !valido_x || !valido_y || !valido_z
    error("Los argumentos deben ser escalares reales y
       finitos")
  endif

  if z<=0
    error("z debe ser estrictamente positivo")
  endif

  suma_cuadrados=x^2+y^2;

  distancia=sqrt(suma_cuadrados);

  razon=distancia/z;

  if razon>1
    resultado=1;
  else
    resultado=razon;
  endif

endfunction


disp(razon_limitada(3,4,10))
disp(razon_limitada(3,4,5))
disp(razon_limitada(3,4,2))
disp(razon_limitada(0,0,2))
```

Python. En Python se utiliza `math.isfinite` para rechazar valores infinitos o NaN.

```
import math


def razon_limitada(x,y,z):
    """
    Calcula sqrt(x^2+y^2)/z
    y limita el resultado a un maximo de 1.
    """
```

```
    valido_x=(
        isinstance(x,(int,float))
        and not isinstance(x,bool)
        and math.isfinite(x)
    )

    valido_y=(
        isinstance(y,(int,float))
        and not isinstance(y,bool)
        and math.isfinite(y)
    )

    valido_z=(
        isinstance(z,(int,float))
        and not isinstance(z,bool)
        and math.isfinite(z)
    )

    if not valido_x or not valido_y or not valido_z:

        raise TypeError(
            "Los argumentos deben ser escalares reales y
               finitos"
        )

    if z<=0:

        raise ValueError(
            "z debe ser estrictamente positivo"
        )

    suma_cuadrados=x**2+y**2

    distancia=math.sqrt(suma_cuadrados)

    razon=distancia/z

    if razon>1:
        resultado=1
    else:
        resultado=razon

    return resultado


print(razon_limitada(3,4,10))
print(razon_limitada(3,4,5))
print(razon_limitada(3,4,2))
print(razon_limitada(0,0,2))
```

Perl. En Perl se utiliza `looks_like_number` para validar la representación

numérica e `isfinite` para comprobar que el valor sea finito.

```
use Scalar::Util qw(looks_like_number);
use POSIX qw(isfinite);


sub razon_limitada {

    my ($x,$y,$z)=@_;

    # Calcula sqrt(x^2+y^2)/z
    # y limita el resultado a un maximo de 1.

    my $valido_x=
        looks_like_number($x)
        && isfinite($x);

    my $valido_y=
        looks_like_number($y)
        && isfinite($y);

    my $valido_z=
        looks_like_number($z)
        && isfinite($z);

    die "Los argumentos deben ser escalares reales y
       finitos\n"
        unless $valido_x
            && $valido_y
            && $valido_z;

    die "z debe ser estrictamente positivo\n"
        if $z<=0;

    my $suma_cuadrados=
        $x**2+$y**2;

    my $distancia=
        sqrt($suma_cuadrados);

    my $razon=
        $distancia/$z;

    my $resultado;

    if ($razon>1) {

        $resultado=1;

    }
    else {
```

```
        $resultado=$razon;

    }

    return $resultado;

}


print razon_limitada(3,4,10),"\n";
print razon_limitada(3,4,5),"\n";
print razon_limitada(3,4,2),"\n";
print razon_limitada(0,0,2),"\n";
```

JavaScript. En JavaScript, `Number.isFinite` permite comprobar que cada argumento sea un número finito.

```
function razonLimitada(x,y,z){

    // Calcula sqrt(x^2+y^2)/z
    // y limita el resultado a un maximo de 1.

    let validoX=
        typeof x==="number"
        && Number.isFinite(x)

    let validoY=
        typeof y==="number"
        && Number.isFinite(y)

    let validoZ=
        typeof z==="number"
        && Number.isFinite(z)

    if(!validoX || !validoY || !validoZ){

        throw new TypeError(
            "Los argumentos deben ser escalares reales y
               finitos"
        )

    }

    if(z<=0){

        throw new RangeError(
            "z debe ser estrictamente positivo"
        )
```

```
    }

    let sumaCuadrados=
        x**2+y**2

    let distancia=
        Math.sqrt(sumaCuadrados)

    let razon=
        distancia/z

    let resultado

    if(razon>1){

        resultado=1

    }
    else{

        resultado=razon

    }

    return resultado

}


console.log(razonLimitada(3,4,10))
console.log(razonLimitada(3,4,5))
console.log(razonLimitada(3,4,2))
console.log(razonLimitada(0,0,2))
```

22. **Ejercicio:** Documentación con JSDoc

    Utilice la función `razon_limitada` desarrollada en el ejercicio anterior y escriba documentación suficiente para que otra persona pueda utilizarla sin necesidad de leer su implementación.

    La documentación deberá describir, como mínimo:

    - nombre de la función;
    - propósito;
    - argumentos, indicando tipo y significado;
    - valor de retorno;
    - intervalo de valores posibles del resultado;
    - precondiciones;
    - condiciones que producen error;

- fórmula matemática utilizada;
- al menos un ejemplo de uso;
- autor;
- fecha;
- versión.

La documentación deberá especificar explícitamente que:

- `x`, `y` y `z` son números reales escalares y finitos;
- `z` debe ser estrictamente positivo;
- la función calcula

$$q = \frac{\sqrt{x^2 + y^2}}{z};$$

- el resultado es

$$\text{mín}(q, 1);$$

- el valor devuelto pertenece al intervalo $[0, 1]$;
- la función no acepta vectores ni matrices.

Luego escriba pruebas que verifiquen, como mínimo:

a) dos casos normales;
b) un caso en el que la razón sea exactamente 1;
c) un caso en el que la razón sea mayor que 1 y deba limitarse;
d) un caso en el que el resultado sea 0;
e) un valor de `z` igual a 0 que deba producir error;
f) un valor de `z` negativo que deba producir error;
g) una entrada no numérica que deba producir error.

Utilice los mismos datos de prueba en todos los lenguajes:

(x, y, z)	resultado esperado
$(3, 4, 10)$	0,5
$(6, 8, 20)$	0,5
$(3, 4, 5)$	1
$(3, 4, 2)$	1
$(0, 0, 2)$	0

Las pruebas de error utilizarán:

```
(3,4,0)
(3,4,-2)
("3",4,10)
```

Compare finalmente los mecanismos de documentación disponibles en los cinco lenguajes. Distinga entre la información destinada a quien utiliza una función y los comentarios destinados a explicar detalles internos de su implementación.

Scilab. En Scilab puede colocarse un bloque de comentarios al comienzo de la función. La documentación debe describir la interfaz y no simplemente repetir las instrucciones del programa.

```
function resultado=razon_limitada(x,y,z)

    // FUNCION:
    //   razon_limitada
    //
    // PROPOSITO:
    //   Calcula la razon entre la distancia
    //   sqrt(x^2+y^2) y el valor z.
    //   El resultado se limita a un maximo de 1.
    //
    // ARGUMENTOS:
    //   x: escalar real y finito.
    //   y: escalar real y finito.
    //   z: escalar real, finito y mayor que cero.
    //
    // RETORNO:
    //   resultado: numero real entre 0 y 1.
    //
    // PRECONDICIONES:
    //   x, y y z deben ser escalares reales y finitos.
    //   z debe ser estrictamente positivo.
    //
    // ERRORES:
    //   Se produce un error si algun argumento no
    //   cumple las precondiciones.
    //
    // FORMULA:
    //   q=sqrt(x^2+y^2)/z
    //   resultado=min(q,1)
    //
    // EJEMPLO:
    //   razon_limitada(3,4,10)
    //   produce 0.5
    //
    // AUTOR: nombre del autor
    // FECHA: fecha
    // VERSION: 1.0

    valido_x=(type(x)==1) & ..
             (size(x,"*")==1) & ..
             isreal(x);

    if valido_x then
```

```
        valido_x=~isnan(x) & abs(x)<>%inf;
    end

    valido_y=(type(y)==1) & ..
             (size(y,"*")==1) & ..
             isreal(y);

    if valido_y then
        valido_y=~isnan(y) & abs(y)<>%inf;
    end

    valido_z=(type(z)==1) & ..
             (size(z,"*")==1) & ..
             isreal(z);

    if valido_z then
        valido_z=~isnan(z) & abs(z)<>%inf;
    end

    if ~valido_x | ~valido_y | ~valido_z then
        error("Los argumentos deben ser escalares reales y
           finitos")
    end

    if z<=0 then
        error("z debe ser estrictamente positivo")
    end

    distancia=sqrt(x^2+y^2);
    razon=distancia/z;

    if razon>1 then
        resultado=1;
    else
        resultado=razon;
    end

endfunction


disp(razon_limitada(3,4,10))
disp(razon_limitada(6,8,20))
disp(razon_limitada(3,4,5))
disp(razon_limitada(3,4,2))
disp(razon_limitada(0,0,2))
```

Octave. En Octave puede utilizarse un bloque de comentarios inmediatamente después de la declaración de la función. Ese texto puede actuar además como texto de ayuda asociado a la función.

```
```

```
function resultado=razon_limitada(x,y,z)

  % RAZON_LIMITADA Calcula una razon limitada a 1.
  %
  % Uso:
  %   resultado=razon_limitada(x,y,z)
  %
  % Argumentos:
  %   x - escalar real y finito.
  %   y - escalar real y finito.
  %   z - escalar real, finito y positivo.
  %
  % Retorno:
  %   Numero real perteneciente a [0,1].
  %
  % Formula:
  %   q=sqrt(x^2+y^2)/z
  %   resultado=min(q,1)
  %
  % Precondiciones:
  %   x, y y z deben ser escalares reales
  %   y finitos. z debe ser mayor que cero.
  %
  % Errores:
  %   Argumento no valido o z<=0.
  %
  % Ejemplo:
  %   razon_limitada(3,4,10)
  %   produce 0.5
  %
  % Autor: nombre del autor
  % Fecha: fecha
  % Version: 1.0

  valido_x=isnumeric(x) ...
           && isscalar(x) ...
           && isreal(x) ...
           && isfinite(x);

  valido_y=isnumeric(y) ...
           && isscalar(y) ...
           && isreal(y) ...
           && isfinite(y);

  valido_z=isnumeric(z) ...
           && isscalar(z) ...
           && isreal(z) ...
           && isfinite(z);

  if !valido_x || !valido_y || !valido_z
    error("Los argumentos deben ser escalares reales y
```

```
        finitos")
  endif

  if z<=0
    error("z debe ser estrictamente positivo")
  endif

  distancia=sqrt(x^2+y^2);
  razon=distancia/z;

  if razon>1
    resultado=1;
  else
    resultado=razon;
  endif

endfunction


disp(razon_limitada(3,4,10))
disp(razon_limitada(6,8,20))
disp(razon_limitada(3,4,5))
disp(razon_limitada(3,4,2))
disp(razon_limitada(0,0,2))
```

Python. En Python se utiliza habitualmente una cadena de documentación o *docstring* como primera instrucción de la función.

```
import math


def razon_limitada(x,y,z):
    """
    Calcula sqrt(x^2+y^2)/z y limita
    el resultado a un maximo de 1.

    Argumentos
    ----------
    x : int o float
        Escalar real y finito.
    y : int o float
        Escalar real y finito.
    z : int o float
        Escalar real, finito y mayor que cero.

    Retorno
    -------
    float
        Numero perteneciente al intervalo [0,1].
```

```
    Precondiciones
    --------------
    x, y y z deben ser escalares reales y finitos.
    z debe ser estrictamente positivo.

    Errores
    -------
    TypeError
        Si algun argumento no es numerico valido.
    ValueError
        Si z es menor o igual que cero.

    Formula
    -------
    q=sqrt(x**2+y**2)/z
    resultado=min(q,1)

    Ejemplo
    -------
    razon_limitada(3,4,10)
    devuelve 0.5

    Autor
    -----
    nombre del autor

    Fecha
    -----
    fecha

    Version
    -------
    1.0
    """

    valido_x=(
        isinstance(x,(int,float))
        and not isinstance(x,bool)
        and math.isfinite(x)
    )

    valido_y=(
        isinstance(y,(int,float))
        and not isinstance(y,bool)
        and math.isfinite(y)
    )

    valido_z=(
        isinstance(z,(int,float))
        and not isinstance(z,bool)
        and math.isfinite(z)
```

```
    )

    if not valido_x or not valido_y or not valido_z:
        raise TypeError(
            "Los argumentos deben ser escalares reales y
               finitos"
        )

    if z<=0:
        raise ValueError(
            "z debe ser estrictamente positivo"
        )

    distancia=math.sqrt(x**2+y**2)
    razon=distancia/z

    if razon>1:
        resultado=1
    else:
        resultado=razon

    return resultado


print(razon_limitada(3,4,10))
print(razon_limitada(6,8,20))
print(razon_limitada(3,4,5))
print(razon_limitada(3,4,2))
print(razon_limitada(0,0,2))
```

Perl. En Perl puede utilizarse POD para documentación destinada al usuario del programa o módulo. Los comentarios con # continúan siendo apropiados para explicar detalles internos de implementación.

```
use Scalar::Util qw(looks_like_number);
use POSIX qw(isfinite);


=head2 razon_limitada(x,y,z)

Calcula

    q=sqrt(x^2+y^2)/z

y devuelve q cuando q<=1, o 1 cuando q>1.

Argumentos:

    x   Escalar real y finito.
    y   Escalar real y finito.
```

```
    z   Escalar real, finito y positivo.

Retorno:

    Numero real perteneciente al intervalo [0,1].

Precondiciones:

    x, y y z deben ser escalares reales y finitos.
    z debe ser estrictamente positivo.

Errores:

    Se produce un error si los argumentos no cumplen
    las precondiciones.

Ejemplo:

    razon_limitada(3,4,10)

produce 0.5.

Autor: nombre del autor
Fecha: fecha
Version: 1.0

=cut


sub razon_limitada {

    my ($x,$y,$z)=@_;

    my $valido_x=
        looks_like_number($x)
        && isfinite($x);

    my $valido_y=
        looks_like_number($y)
        && isfinite($y);

    my $valido_z=
        looks_like_number($z)
        && isfinite($z);

    die "Los argumentos deben ser escalares reales y
       finitos\n"
        unless $valido_x
            && $valido_y
            && $valido_z;
```

```
    die "z debe ser estrictamente positivo\n"
        if $z<=0;

    my $distancia=sqrt($x**2+$y**2);
    my $razon=$distancia/$z;

    my $resultado;

    if ($razon>1) {
        $resultado=1;
    }
    else {
        $resultado=$razon;
    }

    return $resultado;

}


print razon_limitada(3,4,10),"\n";
print razon_limitada(6,8,20),"\n";
print razon_limitada(3,4,5),"\n";
print razon_limitada(3,4,2),"\n";
print razon_limitada(0,0,2),"\n";
```

JavaScript. En JavaScript puede utilizarse una convención como JSDoc para documentar la interfaz de la función.

```
/**
 * Calcula sqrt(x^2+y^2)/z y limita
 * el resultado a un maximo de 1.
 *
 * @param {number} x Escalar real y finito.
 * @param {number} y Escalar real y finito.
 * @param {number} z Escalar real, finito y positivo.
 *
 * @returns {number}
 * Numero perteneciente al intervalo [0,1].
 *
 * Precondiciones:
 * x, y y z deben ser numeros finitos.
 * z debe ser estrictamente positivo.
 *
 * Errores:
 * Se produce un error si algun argumento no
 * cumple las precondiciones.
 *
 * Formula:
 * q=sqrt(x^2+y^2)/z
```

```
 * resultado=min(q,1)
 *
 * Ejemplo:
 * razonLimitada(3,4,10) devuelve 0.5.
 *
 * Autor: nombre del autor
 * Fecha: fecha
 * Version: 1.0
 */
function razonLimitada(x,y,z){

    let validoX=
        typeof x==="number"
        && Number.isFinite(x)

    let validoY=
        typeof y==="number"
        && Number.isFinite(y)

    let validoZ=
        typeof z==="number"
        && Number.isFinite(z)

    if(!validoX || !validoY || !validoZ){

        throw new TypeError(
            "Los argumentos deben ser escalares reales y
               finitos"
        )

    }

    if(z<=0){

        throw new RangeError(
            "z debe ser estrictamente positivo"
        )

    }

    let distancia=
        Math.sqrt(x**2+y**2)

    let razon=distancia/z

    let resultado

    if(razon>1){
        resultado=1
    }
    else{
```

```
            resultado=razon
        }

        return resultado

    }


    console.log(razonLimitada(3,4,10))
    console.log(razonLimitada(6,8,20))
    console.log(razonLimitada(3,4,5))
    console.log(razonLimitada(3,4,2))
    console.log(razonLimitada(0,0,2))
```

23. **Ejercicio:** Pruebas unitarias

    Utilice la función `normalizar` desarrollada anteriormente y construya una batería de pruebas que pueda ejecutarse automáticamente.

    Se supone que la función posee la interfaz conceptual

    ```
    normalizar(v, metodo="minmax") -> y
    ```

    y admite los métodos

    ```
    "max"
    "minmax"
    "zscore"
    ```

    Recuerde que:

    - `v` debe ser un vector numérico real, finito y no vacío;
    - una matriz debe ser rechazada;
    - un vector que contenga NaN debe ser rechazado;
    - un método inexistente debe producir un error;
    - un vector constante debe producir un vector de ceros para los tres métodos;
    - para el método `"max"` se utiliza

      $$s = \max_i |v_i|, \qquad y_i = \frac{v_i}{s};$$

    - para el método `"minmax"`, cuando el vector no es constante,

      $$y_i = \frac{v_i - \min(v)}{\max(v) - \min(v)};$$

    - para `"zscore"` se utiliza el desvío estándar poblacional.

    La batería deberá contener exactamente las siguientes diez pruebas:

a) **Resultado exacto.**
Compruebe que

$$\text{normalizar}([0,5,10],''\,\text{minmax}'') = [0,0{,}5,1].$$

b) **Comparación aproximada.**
Para

$$v = [1,2,3]$$

compruebe que el método `"zscore"` produce aproximadamente

$$\left[-\sqrt{\frac{3}{2}}, 0, \sqrt{\frac{3}{2}}\right].$$

Utilice una tolerancia de 10^{-12}.

c) **Caso límite: vector constante.**
Compruebe que

$$\text{normalizar}([5,5,5],''\,\text{minmax}'') = [0,0,0].$$

d) **Entrada vacía.**
Compruebe automáticamente que un vector vacío produce un error.

e) **Dimensiones incorrectas.**
Compruebe que la matriz

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

es rechazada.

f) **NaN.**
Compruebe que el vector

$$[1, \text{NaN}, 3]$$

es rechazado.

g) **Método inválido.**
Compruebe que

```
normalizar([1,2,3], "desconocido")
```

produce un error.

h) **Propiedad matemática.**
Para un vector no constante normalizado mediante `"minmax"`, compruebe que el mínimo del resultado sea 0 y el máximo sea 1.

i) **Prueba vectorial.**
Compruebe aproximadamente que

$$\text{normalizar}([2,4,6,8],''\text{minmax}'') = \left[0, \frac{1}{3}, \frac{2}{3}, 1\right].$$

j) **Prueba de regresión.**
Compruebe que el método `"max"` utiliza el máximo del valor absoluto y no simplemente el máximo del vector:

$$\text{normalizar}([-8,2,4],''\text{max}'') = [-1, 0{,}25, 0{,}5].$$

Las pruebas deberán:

- ejecutarse sin intervención del usuario;
- detenerse o informar claramente cuando una prueba falle;
- poder repetirse desde una sesión limpia;
- no depender de variables creadas manualmente en una sesión interactiva;
- utilizar las mismas entradas y resultados esperados en los cinco lenguajes.

La función `normalizar` deberá estar definida o cargada antes de ejecutar el archivo de pruebas.

Scilab. En Scilab puede construirse una pequeña infraestructura de pruebas utilizando funciones que comprueben condiciones lógicas. Una prueba que falla produce un error y detiene la ejecución.

```
function comprobar(condicion,nombre)

    if condicion then
        mprintf("OK: %s\n",nombre);
    else
        error("FALLO: "+nombre)
    end

endfunction


function r=aprox_igual(a,b,tolerancia)

    r=and(
        abs(a-b)<=tolerancia
    );

endfunction


tolerancia=1d-12;
```

```
// 1. Resultado exacto.

y=normalizar(
    [0 5 10],
    "minmax"
);

comprobar(
    and(y==[0 0.5 1]),
    "resultado exacto"
);


// 2. Comparacion aproximada.

esperado=[
    -sqrt(3/2)
    0
    sqrt(3/2)
];

esperado=esperado';

y=normalizar(
    [1 2 3],
    "zscore"
);

comprobar(
    aprox_igual(
        y,
        esperado,
        tolerancia
    ),
    "zscore aproximado"
);


// 3. Vector constante.

y=normalizar(
    [5 5 5],
    "minmax"
);

comprobar(
    and(y==[0 0 0]),
    "vector constante"
);
```

```
// 4. Vector vacio.

hubo_error=%f;

try
    normalizar([]);
catch
    hubo_error=%t;
end

comprobar(
    hubo_error,
    "vector vacio"
);


// 5. Matriz.

hubo_error=%f;

try
    normalizar(
        [1 2;3 4]
    );
catch
    hubo_error=%t;
end

comprobar(
    hubo_error,
    "matriz rechazada"
);


// 6. NaN.

hubo_error=%f;

try
    normalizar(
        [1 %nan 3]
    );
catch
    hubo_error=%t;
end

comprobar(
    hubo_error,
    "NaN rechazado"
);
```

```
// 7. Metodo invalido.

hubo_error=%f;

try
    normalizar(
        [1 2 3],
        "desconocido"
    );
catch
    hubo_error=%t;
end

comprobar(
    hubo_error,
    "metodo invalido"
);


// 8. Propiedad minmax.

y=normalizar(
    [-4 2 8],
    "minmax"
);

comprobar(
    abs(min(y))<=tolerancia & ..
    abs(max(y)-1)<=tolerancia,
    "propiedad minmax"
);


// 9. Prueba vectorial.

y=normalizar(
    [2 4 6 8],
    "minmax"
);

esperado=[
    0
    1/3
    2/3
    1
];

esperado=esperado';
```

```
comprobar(
    aprox_igual(
        y,
        esperado,
        tolerancia
    ),
    "prueba vectorial"
);


// 10. Regresion.

y=normalizar(
    [-8 2 4],
    "max"
);

comprobar(
    aprox_igual(
        y,
        [-1 0.25 0.5],
        tolerancia
    ),
    "regresion metodo max"
);

disp("Todas las pruebas pasaron")
```

Octave. En Octave puede utilizarse el mismo esquema: cada comprobación produce un error si la condición esperada no se cumple.

```
function comprobar(condicion,nombre)

  if all(condicion(:))
    printf("OK: %s\n",nombre);
  else
    error(["FALLO: " nombre])
  endif

endfunction


function r=aprox_igual(a,b,tolerancia)

  r=all(
    abs(a(:)-b(:))<=tolerancia
  );

endfunction
```

```
tolerancia=1e-12;


% 1. Resultado exacto.

y=normalizar(
  [0 5 10],
  "minmax"
);

comprobar(
  y==[0 0.5 1],
  "resultado exacto"
);


% 2. Comparacion aproximada.

esperado=[
  -sqrt(3/2)
  0
  sqrt(3/2)
];

y=normalizar(
  [1 2 3],
  "zscore"
);

comprobar(
  aprox_igual(
    y,
    esperado,
    tolerancia
  ),
  "zscore aproximado"
);


% 3. Vector constante.

y=normalizar(
  [5 5 5],
  "minmax"
);

comprobar(
  y==[0 0 0],
  "vector constante"
);
```

```
% 4. Vector vacio.

hubo_error=false;

try

  normalizar([]);

catch

  hubo_error=true;

end_try_catch

comprobar(
  hubo_error,
  "vector vacio"
);


% 5. Matriz.

hubo_error=false;

try

  normalizar(
    [1 2;3 4]
  );

catch

  hubo_error=true;

end_try_catch

comprobar(
  hubo_error,
  "matriz rechazada"
);


% 6. NaN.

hubo_error=false;

try

  normalizar(
```

```
    [1 NaN 3]
  );

catch

  hubo_error=true;

end_try_catch

comprobar(
  hubo_error,
  "NaN rechazado"
);


% 7. Metodo invalido.

hubo_error=false;

try

  normalizar(
    [1 2 3],
    "desconocido"
  );

catch

  hubo_error=true;

end_try_catch

comprobar(
  hubo_error,
  "metodo invalido"
);


% 8. Propiedad minmax.

y=normalizar(
  [-4 2 8],
  "minmax"
);

comprobar(
  abs(min(y))<=tolerancia ...
  && abs(max(y)-1)<=tolerancia,
  "propiedad minmax"
);
```

```
% 9. Prueba vectorial.

y=normalizar(
  [2 4 6 8],
  "minmax"
);

esperado=[
  0
  1/3
  2/3
  1
];

comprobar(
  aprox_igual(
    y,
    esperado,
    tolerancia
  ),
  "prueba vectorial"
);


% 10. Regresion.

y=normalizar(
  [-8 2 4],
  "max"
);

comprobar(
  aprox_igual(
    y,
    [-1 0.25 0.5],
    tolerancia
  ),
  "regresion metodo max"
);

disp("Todas las pruebas pasaron")
```

Python. En Python puede utilizarse el módulo `unittest`. Como la función `normalizar` utiliza NumPy, las comparaciones vectoriales aproximadas pueden realizarse con `numpy.allclose`.

```
import unittest
import numpy as np
```

```
from normalizar import normalizar


class PruebasNormalizar(unittest.TestCase):

    def test_01_resultado_exacto(self):

        y=normalizar(
            [0,5,10],
            "minmax"
        )

        self.assertTrue(
            np.array_equal(
                y,
                np.array([0,0.5,1])
            )
        )


    def test_02_zscore_aproximado(self):

        y=normalizar(
            [1,2,3],
            "zscore"
        )

        esperado=np.array([
            -np.sqrt(3/2),
            0,
            np.sqrt(3/2)
        ])

        self.assertTrue(
            np.allclose(
                y,
                esperado,
                atol=1e-12,
                rtol=0
            )
        )


    def test_03_vector_constante(self):

        y=normalizar(
            [5,5,5],
            "minmax"
        )

        self.assertTrue(
```

```
            np.array_equal(
                y,
                np.array([0,0,0])
            )
        )


    def test_04_vector_vacio(self):

        with self.assertRaises(
            (ValueError,TypeError)
        ):

            normalizar([])


    def test_05_matriz(self):

        with self.assertRaises(
            (ValueError,TypeError)
        ):

            normalizar([
                [1,2],
                [3,4]
            ])


    def test_06_nan(self):

        with self.assertRaises(
            (ValueError,TypeError)
        ):

            normalizar([
                1,
                np.nan,
                3
            ])


    def test_07_metodo_invalido(self):

        with self.assertRaises(
            (ValueError,TypeError)
        ):

            normalizar(
                [1,2,3],
                "desconocido"
            )
```

```
    def test_08_propiedad_minmax(self):

        y=normalizar(
            [-4,2,8],
            "minmax"
        )

        self.assertAlmostEqual(
            np.min(y),
            0,
            delta=1e-12
        )

        self.assertAlmostEqual(
            np.max(y),
            1,
            delta=1e-12
        )


    def test_09_vector(self):

        y=normalizar(
            [2,4,6,8],
            "minmax"
        )

        esperado=np.array([
            0,
            1/3,
            2/3,
            1
        ])

        self.assertTrue(
            np.allclose(
                y,
                esperado,
                atol=1e-12,
                rtol=0
            )
        )


    def test_10_regresion_max(self):

        y=normalizar(
            [-8,2,4],
            "max"
```

```
        )

        esperado=np.array([
            -1,
            0.25,
            0.5
        ])

        self.assertTrue(
            np.allclose(
                y,
                esperado,
                atol=1e-12,
                rtol=0
            )
        )


if __name__=="__main__":
    unittest.main()
```

Perl. En Perl puede utilizarse `Test::More`. La función `normalizar` deberá estar definida o cargada antes de ejecutar las pruebas.

```
use strict;
use warnings;

use Test::More;


sub aprox_igual {

    my (
        $a,
        $b,
        $tolerancia
    )=@_;

    return 0
        unless @$a==@$b;

    for (
        my $i=0;
        $i<@$a;
        $i++
    ) {

        return 0
            if abs(
                $a->[$i]-$b->[$i]
```

```
            )>$tolerancia;

    }

    return 1;

}


my $tolerancia=1e-12;


// normalizar debe estar cargada
// antes de estas pruebas.


// 1. Resultado exacto.

is_deeply(
    normalizar(
        [0,5,10],
        'minmax'
    ),
    [0,0.5,1],
    'resultado exacto'
);


// 2. Comparacion aproximada.

ok(
    aprox_igual(
        normalizar(
            [1,2,3],
            'zscore'
        ),
        [
            -sqrt(3/2),
            0,
            sqrt(3/2)
        ],
        $tolerancia
    ),
    'zscore aproximado'
);


// 3. Vector constante.

is_deeply(
    normalizar(
```

```
            [5,5,5],
            'minmax'
        ),
        [0,0,0],
        'vector constante'
);


// 4. Vector vacio.

my $hubo_error=0;

eval {

    normalizar([]);

};

$hubo_error=1
    if $@ ne '';

ok(
    $hubo_error,
    'vector vacio'
);


// 5. Matriz.

$hubo_error=0;

eval {

    normalizar([
        [1,2],
        [3,4]
    ]);

};

$hubo_error=1
    if $@ ne '';

ok(
    $hubo_error,
    'matriz rechazada'
);


// 6. NaN.
```

```
$hubo_error=0;

eval {

    normalizar([
        1,
        'NaN',
        3
    ]);

};

$hubo_error=1
    if $@ ne '';

ok(
    $hubo_error,
    'NaN rechazado'
);


// 7. Metodo invalido.

$hubo_error=0;

eval {

    normalizar(
        [1,2,3],
        'desconocido'
    );

};

$hubo_error=1
    if $@ ne '';

ok(
    $hubo_error,
    'metodo invalido'
);


// 8. Propiedad minmax.

my $y=normalizar(
    [-4,2,8],
    'minmax'
);

my $minimo=$y->[0];
```

```
my $maximo=$y->[0];

for my $x (@$y) {

    if ($x<$minimo) {
        $minimo=$x;
    }

    if ($x>$maximo) {
        $maximo=$x;
    }

}

ok(
    abs($minimo)<=
        $tolerancia
    &&
    abs($maximo-1)<=
        $tolerancia,
    'propiedad minmax'
);


// 9. Prueba vectorial.

ok(
    aprox_igual(
        normalizar(
            [2,4,6,8],
            'minmax'
        ),
        [
            0,
            1/3,
            2/3,
            1
        ],
        $tolerancia
    ),
    'prueba vectorial'
);


// 10. Regresion.

ok(
    aprox_igual(
        normalizar(
            [-8,2,4],
            'max'
```

```
        ),
        [
            -1,
            0.25,
            0.5
        ],
        $tolerancia
    ),
    'regresion metodo max'
);


done_testing();
```

JavaScript. En JavaScript puede utilizarse el módulo de aserciones incluido en Node.js. Si una aserción falla, el programa finaliza indicando la prueba responsable.

```
const assert=require("assert").strict

const {
    normalizar
}=require("./normalizar")


function aproxIgual(a,b,tolerancia){

    if(a.length!==b.length){
        return false
    }

    for(let i=0;i<a.length;i++){

        if(
            Math.abs(a[i]-b[i])
            >tolerancia
        ){
            return false
        }

    }

    return true

}


const tolerancia=1e-12


// 1. Resultado exacto.
```

```
assert.deepStrictEqual(
    normalizar(
        [0,5,10],
        "minmax"
    ),
    [0,0.5,1],
    "resultado exacto"
)


// 2. Comparacion aproximada.

assert.ok(
    aproxIgual(
        normalizar(
            [1,2,3],
            "zscore"
        ),
        [
            -Math.sqrt(3/2),
            0,
            Math.sqrt(3/2)
        ],
        tolerancia
    ),
    "zscore aproximado"
)


// 3. Vector constante.

assert.deepStrictEqual(
    normalizar(
        [5,5,5],
        "minmax"
    ),
    [0,0,0],
    "vector constante"
)


// 4. Vector vacio.

assert.throws(
    ()=>{
        normalizar([])
    },
    undefined,
    "vector vacio"
)
```

```
// 5. Matriz.

assert.throws(
    ()=>{
        normalizar([
            [1,2],
            [3,4]
        ])
    },
    undefined,
    "matriz rechazada"
)


// 6. NaN.

assert.throws(
    ()=>{
        normalizar([
            1,
            NaN,
            3
        ])
    },
    undefined,
    "NaN rechazado"
)


// 7. Metodo invalido.

assert.throws(
    ()=>{
        normalizar(
            [1,2,3],
            "desconocido"
        )
    },
    undefined,
    "metodo invalido"
)


// 8. Propiedad minmax.

let y=normalizar(
    [-4,2,8],
    "minmax"
)
```

```
let minimo=y[0]
let maximo=y[0]

for(const x of y){

    if(x<minimo){
        minimo=x
    }

    if(x>maximo){
        maximo=x
    }

}

assert.ok(
    Math.abs(minimo)<=tolerancia
    &&
    Math.abs(maximo-1)<=tolerancia,
    "propiedad minmax"
)


// 9. Prueba vectorial.

assert.ok(
    aproxIgual(
        normalizar(
            [2,4,6,8],
            "minmax"
        ),
        [
            0,
            1/3,
            2/3,
            1
        ],
        tolerancia
    ),
    "prueba vectorial"
)


// 10. Regresion.

assert.ok(
    aproxIgual(
        normalizar(
            [-8,2,4],
            "max"
```

```
        ),
        [
            -1,
            0.25,
            0.5
        ],
        tolerancia
    ),
    "regresion metodo max"
)


console.log(
    "Todas las pruebas pasaron"
)
```

24. **Ejercicio:** Rendimiento de vectorización

Considere la función

$$f(x) = \sin(x)e^{-x^2}.$$

Para un entero $n > 1$, construya n valores igualmente espaciados en el intervalo

$$[-3, 3]$$

y calcule

$$y_i = f(x_i).$$

Implemente exactamente tres versiones del cálculo:

a) **Crecimiento dinámico.**
Comience con una colección vacía y agregue cada resultado durante el recorrido.

b) **Preasignación.**
Reserve previamente espacio para los n resultados y asigne cada elemento mediante un ciclo.

c) **Operación masiva.**
Utilice una operación sobre la colección completa sin escribir explícitamente el ciclo cuando el lenguaje o la biblioteca numérica utilizada lo permita.

En Scilab, Octave y Python con NumPy, la tercera versión utilizará operaciones numéricas vectorizadas.

En Perl y JavaScript estándar, utilice una transformación masiva mediante `map`. En esos dos casos no denomine a esta operación "vectorización numérica": el objetivo es comparar también las diferencias entre los mecanismos que ofrecen los lenguajes.

Utilice los tamaños

$$n = 1000, \quad 5000, \quad 10000, \quad 20000.$$

Para cada tamaño:

i) genere una única vez el vector de entrada;

ii) ejecute las tres versiones con exactamente los mismos valores de entrada;

iii) compruebe que los resultados coincidan con una tolerancia de

$$10^{-12};$$

iv) ejecute cada versión cinco veces;

v) mida solamente el cálculo de `y`, sin incluir la generación de los valores de entrada;

vi) calcule el promedio de los cinco tiempos;

vii) registre los resultados en una tabla con las columnas

```
n
crecimiento
preasignacion
masiva
```

Luego:

a) Grafique el tiempo medio en función de n para las tres versiones.

b) Determine si la preasignación mejora el tiempo respecto del crecimiento dinámico.

c) Determine si la operación masiva es la más rápida en su lenguaje y entorno de ejecución.

d) No suponga de antemano que una versión debe ser necesariamente más rápida que otra: utilice las mediciones.

e) Explique por qué los resultados pueden depender del lenguaje, del tipo de colección, de la biblioteca numérica, del intérprete o compilador y del tamaño del problema.

f) Compare la claridad de las tres implementaciones.

g) Explique la diferencia entre "no escribir un ciclo explícito" y disponer realmente de operaciones numéricas vectorizadas.

h) No compare directamente los tiempos absolutos obtenidos en lenguajes diferentes como si constituyeran una competencia entre lenguajes. Compare principalmente las tres estrategias dentro del mismo entorno.

Scilab. Scilab permite operar directamente sobre vectores. En la tercera versión se utilizan operaciones elemento a elemento. Para medir tiempo pueden utilizarse `tic` y `toc`.

```
function y=version_crecimiento(x)

    y=[];

    for i=1:length(x)

        y(i)=sin(x(i))* ..
             exp(-(x(i)^2));

    end

endfunction


function y=version_preasignada(x)

    y=zeros(size(x,1),size(x,2));

    for i=1:length(x)

        y(i)=sin(x(i))* ..
             exp(-(x(i)^2));

    end

endfunction


function y=version_masiva(x)

    y=sin(x).*exp(-(x.^2));

endfunction


tamanos=[
    1000
    5000
    10000
    20000
];

repeticiones=5;
tolerancia=1d-12;

resultados=zeros(
    length(tamanos),
    4
);
```

```
for k=1:length(tamanos)

    n=tamanos(k);

    x=linspace(-3,3,n);


    // Comprobar resultados.

    y1=version_crecimiento(x);
    y2=version_preasignada(x);
    y3=version_masiva(x);

    error12=max(abs(y1-y2));
    error13=max(abs(y1-y3));

    if error12>tolerancia | ..
       error13>tolerancia then

        error("Las versiones no coinciden")

    end


    tiempo1=0;
    tiempo2=0;
    tiempo3=0;


    for r=1:repeticiones

        tic();
        y1=version_crecimiento(x);
        tiempo1=tiempo1+toc();

        tic();
        y2=version_preasignada(x);
        tiempo2=tiempo2+toc();

        tic();
        y3=version_masiva(x);
        tiempo3=tiempo3+toc();

    end


    tiempo1=tiempo1/repeticiones;
    tiempo2=tiempo2/repeticiones;
    tiempo3=tiempo3/repeticiones;
```

```
    resultados(k,:)=[
        n
        tiempo1
        tiempo2
        tiempo3
    ]';

end


disp(
    "n crecimiento preasignacion masiva"
)

disp(resultados)
```

Octave. Octave también permite aplicar las funciones matemáticas directamente a vectores. En la tercera versión se utilizan los operadores elemento a elemento correspondientes.

```
function y=version_crecimiento(x)

  y=[];

  for i=1:length(x)

    y(i)=sin(x(i))* ...
         exp(-(x(i)^2));

  endfor

endfunction


function y=version_preasignada(x)

  y=zeros(size(x));

  for i=1:length(x)

    y(i)=sin(x(i))* ...
         exp(-(x(i)^2));

  endfor

endfunction


function y=version_masiva(x)
```

```
  y=sin(x).*exp(-(x.^2));

endfunction


tamanos=[
  1000
  5000
  10000
  20000
];

repeticiones=5;
tolerancia=1e-12;

resultados=zeros(
  length(tamanos),
  4
);


for k=1:length(tamanos)

  n=tamanos(k);

  x=linspace(-3,3,n);


  y1=version_crecimiento(x);
  y2=version_preasignada(x);
  y3=version_masiva(x);

  error12=max(abs(y1-y2));
  error13=max(abs(y1-y3));

  if error12>tolerancia || ...
     error13>tolerancia

    error("Las versiones no coinciden")

  endif


  tiempo1=0;
  tiempo2=0;
  tiempo3=0;


  for r=1:repeticiones
```

```
    tic();
    y1=version_crecimiento(x);
    tiempo1=tiempo1+toc();

    tic();
    y2=version_preasignada(x);
    tiempo2=tiempo2+toc();

    tic();
    y3=version_masiva(x);
    tiempo3=tiempo3+toc();

  endfor


  tiempo1=tiempo1/repeticiones;
  tiempo2=tiempo2/repeticiones;
  tiempo3=tiempo3/repeticiones;


  resultados(k,:)=[
    n
    tiempo1
    tiempo2
    tiempo3
  ]';

endfor


disp(
  "n crecimiento preasignacion masiva"
)

disp(resultados)
```

Python. En Python se utiliza NumPy para la tercera versión. Las dos primeras versiones recorren el mismo arreglo de entrada mediante Python.

```
import math
import time

import numpy as np


def version_crecimiento(x):

    y=[]

    for valor in x:
```

```
            resultado=(
                math.sin(valor)
                *math.exp(-(valor*valor))
            )

            y.append(resultado)

        return y


    def version_preasignada(x):

        y=[0.0]*len(x)

        for i in range(len(x)):

            valor=x[i]

            y[i]=(
                math.sin(valor)
                *math.exp(-(valor*valor))
            )

        return y


    def version_masiva(x):

        return (
            np.sin(x)
            *np.exp(-(x**2))
        )


    tamanos=[
        1000,
        5000,
        10000,
        20000
    ]

    repeticiones=5
    tolerancia=1e-12

    resultados=[]


    for n in tamanos:

        x=np.linspace(-3,3,n)
```

```
    y1=version_crecimiento(x)
    y2=version_preasignada(x)
    y3=version_masiva(x)

    error12=np.max(
        np.abs(
            np.asarray(y1)
            -np.asarray(y2)
        )
    )

    error13=np.max(
        np.abs(
            np.asarray(y1)
            -np.asarray(y3)
        )
    )

    if (
        error12>tolerancia
        or error13>tolerancia
    ):

        raise RuntimeError(
            "Las versiones no coinciden"
        )


    tiempo1=0
    tiempo2=0
    tiempo3=0


    for r in range(repeticiones):

        inicio=time.perf_counter()

        y1=version_crecimiento(x)

        tiempo1=(
            tiempo1
            +time.perf_counter()
            -inicio
        )


        inicio=time.perf_counter()

        y2=version_preasignada(x)
```

```
        tiempo2=(
            tiempo2
            +time.perf_counter()
            -inicio
        )


        inicio=time.perf_counter()

        y3=version_masiva(x)

        tiempo3=(
            tiempo3
            +time.perf_counter()
            -inicio
        )


    tiempo1=tiempo1/repeticiones
    tiempo2=tiempo2/repeticiones
    tiempo3=tiempo3/repeticiones


    resultados.append([
        n,
        tiempo1,
        tiempo2,
        tiempo3
    ])


print(
    "n",
    "crecimiento",
    "preasignacion",
    "masiva"
)

for fila in resultados:
    print(*fila)
```

Perl. Perl no utiliza aquí una biblioteca de arreglos numéricos. La tercera versión se expresa mediante `map`, por lo que representa una transformación masiva de la colección y no una operación numérica vectorizada.

```
use strict;
use warnings;

use Time::HiRes qw(time);
```

```
sub version_crecimiento {

    my ($x)=@_;

    my @y;

    for my $valor (@$x) {

        my $resultado=
            sin($valor)
            *exp(-($valor*$valor));

        push @y,$resultado;

    }

    return \@y;

}


sub version_preasignada {

    my ($x)=@_;

    my $n=scalar(@$x);

    my @y=(0) x $n;

    for (
        my $i=0;
        $i<$n;
        $i++
    ) {

        my $valor=$x->[$i];

        $y[$i]=
            sin($valor)
            *exp(-($valor*$valor));

    }

    return \@y;

}


sub version_masiva {
```

```
    my ($x)=@_;

    my @y=map {

        sin($_)*exp(-($_*$_))

    } @$x;

    return \@y;

}


sub max_error {

    my ($a,$b)=@_;

    my $error=0;

    for (
        my $i=0;
        $i<@$a;
        $i++
    ) {

        my $d=abs(
            $a->[$i]-$b->[$i]
        );

        if ($d>$error) {
            $error=$d;
        }

    }

    return $error;

}


my @tamanos=(
    1000,
    5000,
    10000,
    20000
);

my $repeticiones=5;
my $tolerancia=1e-12;
```

```
print(
    "n crecimiento ",
    "preasignacion masiva\n"
);


for my $n (@tamanos) {

    my @x=(0) x $n;

    for (
        my $i=0;
        $i<$n;
        $i++
    ) {

        $x[$i]=
            -3+6*$i/($n-1);

    }


    my $y1=
        version_crecimiento(\@x);

    my $y2=
        version_preasignada(\@x);

    my $y3=
        version_masiva(\@x);


    if (
        max_error($y1,$y2)>$tolerancia
        ||
        max_error($y1,$y3)>$tolerancia
    ) {

        die "Las versiones no coinciden\n";

    }


    my $tiempo1=0;
    my $tiempo2=0;
    my $tiempo3=0;


    for (
        my $r=0;
```

```
        $r<$repeticiones;
        $r++
    ) {

        my $inicio=time();

        $y1=version_crecimiento(\@x);

        $tiempo1=
            $tiempo1+time()-$inicio;


        $inicio=time();

        $y2=version_preasignada(\@x);

        $tiempo2=
            $tiempo2+time()-$inicio;


        $inicio=time();

        $y3=version_masiva(\@x);

        $tiempo3=
            $tiempo3+time()-$inicio;

    }


    $tiempo1=
        $tiempo1/$repeticiones;

    $tiempo2=
        $tiempo2/$repeticiones;

    $tiempo3=
        $tiempo3/$repeticiones;


    print(
        "$n ",
        "$tiempo1 ",
        "$tiempo2 ",
        "$tiempo3\n"
    );

}
```

JavaScript. En JavaScript estándar se utiliza `map` para la tercera versión. Se trata de una transformación masiva de un arreglo; el ejercicio no la considera

vectorización numérica. El ejemplo de temporización está pensado para Node.js.

```
const {
    performance
}=require("node:perf_hooks")


function versionCrecimiento(x){

    let y=[]

    for(const valor of x){

        let resultado=
            Math.sin(valor)
            *Math.exp(-(valor*valor))

        y.push(resultado)

    }

    return y

}


function versionPreasignada(x){

    let y=new Array(x.length)

    for(let i=0;i<x.length;i++){

        let valor=x[i]

        y[i]=
            Math.sin(valor)
            *Math.exp(-(valor*valor))

    }

    return y

}


function versionMasiva(x){

    return x.map(
        valor=>
            Math.sin(valor)
            *Math.exp(-(valor*valor))
```

```
    )

}


function maxError(a,b){

    let error=0

    for(let i=0;i<a.length;i++){

        let d=Math.abs(
            a[i]-b[i]
        )

        if(d>error){
            error=d
        }

    }

    return error

}


function crearX(n){

    let x=new Array(n)

    for(let i=0;i<n;i++){

        x[i]=
            -3+6*i/(n-1)

    }

    return x

}


let tamanos=[
    1000,
    5000,
    10000,
    20000
]

let repeticiones=5
let tolerancia=1e-12
```

```
console.log(
    "n crecimiento preasignacion masiva"
)


for(const n of tamanos){

    let x=crearX(n)


    let y1=versionCrecimiento(x)
    let y2=versionPreasignada(x)
    let y3=versionMasiva(x)


    if(
        maxError(y1,y2)>tolerancia
        ||
        maxError(y1,y3)>tolerancia
    ){

        throw new Error(
            "Las versiones no coinciden"
        )

    }


    let tiempo1=0
    let tiempo2=0
    let tiempo3=0


    for(let r=0;r<repeticiones;r++){

        let inicio=performance.now()

        y1=versionCrecimiento(x)

        tiempo1=
            tiempo1
            +performance.now()
            -inicio


        inicio=performance.now()

        y2=versionPreasignada(x)
```

```
        tiempo2=
            tiempo2
            +performance.now()
            -inicio


        inicio=performance.now()

        y3=versionMasiva(x)

        tiempo3=
            tiempo3
            +performance.now()
            -inicio

    }


    tiempo1=tiempo1/repeticiones
    tiempo2=tiempo2/repeticiones
    tiempo3=tiempo3/repeticiones


    console.log(
        n,
        tiempo1,
        tiempo2,
        tiempo3
    )

}
```

25. **Ejercicio:** Revisión de código de IA

    Utilice una herramienta de inteligencia artificial para generar una solución al problema de normalización desarrollado anteriormente.

    El objetivo de este ejercicio no es aceptar automáticamente el código generado, sino practicar su lectura, verificación, prueba y corrección.

    Todos los estudiantes deberán utilizar esencialmente la misma consulta, cambiando únicamente el nombre del lenguaje.

    Utilice la siguiente consulta:

    ```
    Escriba en [LENGUAJE] una funcion denominada normalizar
    con la interfaz conceptual

    normalizar(v, metodo="minmax") -> y

    v debe ser un vector numerico real, finito y no vacio.
    ```

```
Los metodos permitidos son:

"max"
"minmax"
"zscore"

Para "max":

s = max(abs(v))

Si s es distinto de cero:

y = v/s

Si s es cero, devolver un vector de ceros.

Para "minmax":

a = min(v)
b = max(v)

Si a es distinto de b:

y = (v-a)/(b-a)

Si a es igual a b, devolver un vector de ceros.

Para "zscore", utilizar la media y el desvio estandar
poblacional:

media = sum(v)/n

desvio =
sqrt(sum((v-media)^2)/n)

Si el desvio es distinto de cero:

y = (v-media)/desvio

Si el desvio es cero, devolver un vector de ceros.

La funcion debe:

- aceptar solamente vectores;
- rechazar matrices;
- rechazar vectores vacios;
- rechazar valores no numericos;
- rechazar numeros complejos;
```

```
- rechazar NaN e infinito;
- rechazar nombres de metodo no validos;
- conservar la forma del vector de entrada;
- incluir documentacion;
- producir mensajes de error claros.

No utilices codigo de otros lenguajes.
```

Reemplace `[LENGUAJE]` por:

```
Scilab
Octave
Python
Perl
JavaScript
```

según corresponda.

No solicite inicialmente explicaciones adicionales ni correcciones. Conserve exactamente la primera respuesta obtenida para poder analizarla.

Registre:

- la consulta exacta realizada;
- la respuesta completa producida por la herramienta;
- el nombre de la herramienta utilizada;
- el modelo o versión, si esa información está disponible;
- la fecha de realización del ejercicio.

Luego realice las siguientes actividades sin pedir primero a la herramienta que revise su propia respuesta:

a) Lea el código completo antes de ejecutarlo.

b) Explique qué hace cada parte del programa.

c) Identifique las estructuras de datos utilizadas.

d) Identifique las estructuras de control utilizadas.

e) Compruebe que la función respete exactamente la interfaz solicitada.

f) Compruebe que el método predeterminado sea `"minmax"`.

g) Compruebe que las fórmulas de los tres métodos coincidan con las de la consigna.

h) Compruebe específicamente que el método `"max"` utilice

$$\max_i |v_i|$$

y no simplemente $\max_i v_i$.

i) Compruebe que `"zscore"` utilice el desvío estándar poblacional y no el muestral.

j) Compruebe que el vector constante sea tratado sin realizar una división por cero.

k) Compruebe la sintaxis del código utilizando el intérprete o entorno correspondiente.

l) Ejecute la misma batería de pruebas utilizada en el ejercicio de pruebas automáticas.

m) Clasifique cada prueba como:

```
pasa
falla
no puede ejecutarse
```

n) Para cada prueba que falle, determine si la causa se encuentra en:

- el algoritmo;
- la validación;
- la sintaxis;
- una función o biblioteca inexistente;
- una suposición incorrecta sobre el lenguaje;
- otra causa.

ñ) Si todas las pruebas pasan, no invente un error. En ese caso, busque posibles limitaciones, decisiones discutibles, problemas de claridad o casos adicionales que el contrato no cubra.

o) Consulte la documentación del lenguaje cuando tenga dudas acerca de una función, operador o biblioteca utilizada por el código generado.

p) Corrija personalmente los problemas encontrados.

q) Ejecute nuevamente toda la batería de pruebas sobre la versión corregida.

r) Compare la versión generada inicialmente con la versión corregida.

s) Compare ambas con la solución desarrollada anteriormente por usted.

t) Identifique cualquier parte del código generado que resulte más complicada de lo necesario.

u) Identifique cualquier parte de la respuesta de la herramienta que parezca correcta por su explicación pero no coincida con el código realmente producido.

Construya una tabla resumen con la forma:

Prueba	Esperado	Obtenido	Resultado
Resultado exacto	pasa		
Zscore aproximado	pasa		
Vector constante	pasa		
Vector vacío	error		
Matriz	error		
NaN	error		
Método inválido	error		
Propiedad minmax	pasa		
Prueba vectorial	pasa		
Regresión max	pasa		

Después de corregir el código, complete una segunda tabla con los resultados de la versión corregida.

Finalmente responda:

I) ¿La primera respuesta de la herramienta podía utilizarse sin modificaciones?

II) ¿Las explicaciones proporcionadas coincidían con el comportamiento real del programa?

III) ¿Qué errores o limitaciones pudieron detectarse solamente mediante la ejecución de pruebas?

IV) ¿Qué problemas pudieron detectarse solamente mediante la lectura del código?

V) ¿Qué afirmaciones fue necesario comprobar en la documentación del lenguaje?

VI) ¿La herramienta utilizó mecanismos propios del lenguaje o intentó trasladar construcciones pertenecientes a otro lenguaje?

VII) ¿El código generado era más complicado de lo necesario?

VIII) ¿Habría sido posible verificar correctamente la respuesta sin comprender previamente el algoritmo de normalización?

IX) ¿Qué riesgos existirían al incorporar directamente código generado por una herramienta sin leerlo ni probarlo?

Scilab. Al revisar código Scilab generado automáticamente, compruebe en particular:

- la sintaxis `function ... endfunction`;
- que los índices comiencen en 1;
- la diferencia entre operaciones matriciales y operaciones elemento a elemento;
- el uso correcto de `.*`, `./` y `.^` cuando corresponda;
- los valores `%t`, `%f`, `%nan` y `%inf`;
- que las funciones utilizadas existan realmente en Scilab;
- que no se hayan introducido accidentalmente construcciones de MATLAB u Octave.

Octave. Al revisar código Octave generado automáticamente, compruebe:

- que los índices comiencen en 1;
- las diferencias entre operaciones matriciales y elemento a elemento;
- el uso apropiado de `.*`, `./` y `.^`;
- la distinción entre vector y matriz;
- que las funciones utilizadas pertenezcan realmente a Octave o a paquetes disponibles en el entorno;
- que no se hayan utilizado funciones exclusivas de MATLAB sin comprobar su disponibilidad.

Python. Al revisar código Python generado automáticamente, compruebe:

- qué bibliotecas importa;
- si utiliza listas o arreglos NumPy;
- si las operaciones propuestas son realmente elemento a elemento;
- si distingue correctamente vectores de matrices;
- si las comprobaciones sobre NaN e infinito funcionan para toda la colección;
- si utiliza correctamente los tipos y dimensiones de NumPy;
- si una condición intenta utilizar directamente un arreglo lógico donde se requiere un único valor lógico.

Perl. Al revisar código Perl generado automáticamente, compruebe:

- si los vectores se representan mediante arreglos o referencias a arreglos;
- la correcta desreferenciación mediante `@{}`, `->` u otros mecanismos apropiados;
- que no se suponga que una operación escalar se aplica automáticamente elemento por elemento;
- que la validación numérica sea adecuada;
- que los módulos utilizados estén realmente disponibles;
- que no se hayan trasladado directamente construcciones propias de Python, JavaScript, MATLAB u otros lenguajes.

JavaScript. Al revisar código JavaScript generado automáticamente, compruebe:

- que los vectores se representen mediante arreglos;
- que no se intente multiplicar, dividir o restar arreglos como si fueran vectores numéricos;
- el uso apropiado de `Array.isArray`;
- el uso de `Number.isFinite` para validar números finitos;
- que las transformaciones elemento por elemento se realicen explícitamente, por ejemplo mediante ciclos o `map`;
- que no se suponga la existencia de bibliotecas numéricas que no fueron importadas;
- que no se hayan trasladado directamente construcciones de Python, NumPy, MATLAB u otros lenguajes.

5.11.7. Temas avanzados de programación

26. **Ejercicio:** Contador con clausura

    Construya una abstracción que represente contadores independientes.

    Cada contador deberá conservar tres valores:

    - `inicial`: valor al que vuelve el contador cuando se reinicia;
    - `incremento`: cantidad que se suma en cada avance;
    - `actual`: estado actual del contador.

    Para que el comportamiento sea comparable en todos los lenguajes, utilice la siguiente interfaz conceptual:

```
crear_contador(inicial=0, incremento=1) -> contador

avanzar(contador) -> contador_actualizado, valor

consultar(contador) -> valor

reiniciar(contador) -> contador_actualizado
```

    La operación `avanzar` deberá primero modificar el estado y luego devolver el nuevo valor.

    Por ejemplo:

```
c1 <- crear_contador()

c1, valor <- avanzar(c1)     valor = 1
c1, valor <- avanzar(c1)     valor = 2
c1, valor <- avanzar(c1)     valor = 3

consultar(c1)                -> 3

c2 <- crear_contador(10, 2)

c2, valor <- avanzar(c2)     valor = 12
c2, valor <- avanzar(c2)     valor = 14

consultar(c1)                -> 3
consultar(c2)                -> 14
```

    Por lo tanto, modificar un contador no deberá modificar el estado de ningún otro contador.

    El constructor deberá validar que:

    - `inicial` sea un número real escalar;

- `incremento` sea un número real escalar;
- `incremento` sea distinto de cero.

Luego:

a) Cree un contador `c1` utilizando los valores predeterminados.

b) Avance `c1` tres veces y compruebe que los valores obtenidos sean 1, 2 y 3.

c) Consulte `c1` sin modificarlo y compruebe que continúe valiendo 3.

d) Cree un segundo contador mediante

```
c2 <- crear_contador(10, 2)
```

e) Avance `c2` dos veces y compruebe que produzca 12 y 14.

f) Compruebe que `c1` continúe valiendo 3.

g) Reinicie `c1` y compruebe que vuelva a 0.

h) Avance nuevamente `c1` y compruebe que produzca 1.

i) Reinicie `c2` y compruebe que vuelva a 10.

j) Cree un contador descendente mediante

```
c3 <- crear_contador(100, -10)
```

y compruebe que sus primeros tres avances produzcan

$$90, \quad 80, \quad 70.$$

k) Intente crear un contador con incremento igual a cero y compruebe que se produzca un error.

l) Explique dónde se almacena el estado de cada contador y por qué no es necesaria una variable global compartida.

En este ejercicio, "estado interno" significa estado asociado a cada contador y conservado entre operaciones. La representación concreta puede ser una estructura, registro, diccionario, *hash* u objeto, según el lenguaje.

Como extensión, investigue si el lenguaje permite ocultar completamente ese estado mediante cierres, objetos u otro mecanismo. Compare esa solución con la representación explícita utilizada en este ejercicio, pero conserve el mismo comportamiento observable.

Scilab. En Scilab se utiliza una `struct` para representar el estado. Como las funciones trabajan naturalmente con valores que se devuelven al llamador, las operaciones que modifican el contador devuelven la estructura actualizada.

```
function c=crear_contador(inicial,incremento)

    rhs=argn(2);

    if rhs<1 then
```

```
        inicial=0;
    end

    if rhs<2 then
        incremento=1;
    end

    valido_inicial= ..
        type(inicial)==1 & ..
        size(inicial,"*")==1 & ..
        isreal(inicial);

    valido_incremento= ..
        type(incremento)==1 & ..
        size(incremento,"*")==1 & ..
        isreal(incremento);

    if ~valido_inicial | ..
       ~valido_incremento then

        error( ..
            "Inicial e incremento deben ser escalares
               reales" ..
        )

    end

    if incremento==0 then
        error("El incremento no puede ser cero")
    end

    c=struct( ..
        "inicial",inicial, ..
        "incremento",incremento, ..
        "actual",inicial ..
    );

endfunction


function [c,valor]=avanzar(c)

    c.actual= ..
        c.actual+c.incremento;

    valor=c.actual;

endfunction


function valor=consultar(c)
```

```
    valor=c.actual;

endfunction


function c=reiniciar(c)

    c.actual=c.inicial;

endfunction


c1=crear_contador();

[c1,v]=avanzar(c1);
disp(v)

[c1,v]=avanzar(c1);
disp(v)

[c1,v]=avanzar(c1);
disp(v)

disp(consultar(c1))


c2=crear_contador(10,2);

[c2,v]=avanzar(c2);
disp(v)

[c2,v]=avanzar(c2);
disp(v)


// c1 debe continuar en 3.

disp(consultar(c1))
disp(consultar(c2))


c1=reiniciar(c1);

disp(consultar(c1))

[c1,v]=avanzar(c1);
disp(v)


c2=reiniciar(c2);
```

```
disp(consultar(c2))


c3=crear_contador(100,-10);

[c3,v]=avanzar(c3);
disp(v)

[c3,v]=avanzar(c3);
disp(v)

[c3,v]=avanzar(c3);
disp(v)
```

Octave. En Octave se utiliza igualmente una estructura. Cada operación que modifica el contador devuelve explícitamente el estado actualizado.

```
function c=crear_contador(inicial=0,incremento=1)

  valido_inicial= ...
    isnumeric(inicial) ...
    && isscalar(inicial) ...
    && isreal(inicial);

  valido_incremento= ...
    isnumeric(incremento) ...
    && isscalar(incremento) ...
    && isreal(incremento);

  if !valido_inicial || !valido_incremento

    error(
      "Inicial e incremento deben ser escalares reales"
    )

  endif

  if incremento==0
    error("El incremento no puede ser cero")
  endif

  c=struct(
    "inicial",inicial,
    "incremento",incremento,
    "actual",inicial
  );

endfunction
```

```
function [c,valor]=avanzar(c)

  c.actual= ...
    c.actual+c.incremento;

  valor=c.actual;

endfunction


function valor=consultar(c)

  valor=c.actual;

endfunction


function c=reiniciar(c)

  c.actual=c.inicial;

endfunction


c1=crear_contador();

[c1,v]=avanzar(c1);
disp(v)

[c1,v]=avanzar(c1);
disp(v)

[c1,v]=avanzar(c1);
disp(v)

disp(consultar(c1))


c2=crear_contador(10,2);

[c2,v]=avanzar(c2);
disp(v)

[c2,v]=avanzar(c2);
disp(v)

disp(consultar(c1))
disp(consultar(c2))
```

```
c1=reiniciar(c1);

disp(consultar(c1))

[c1,v]=avanzar(c1);
disp(v)


c2=reiniciar(c2);

disp(consultar(c2))


c3=crear_contador(100,-10);

[c3,v]=avanzar(c3);
disp(v)

[c3,v]=avanzar(c3);
disp(v)

[c3,v]=avanzar(c3);
disp(v)
```

Python. En Python se utiliza un diccionario para representar el estado. Aunque el diccionario es mutable, las funciones devuelven igualmente el contador para conservar la misma interfaz conceptual que en los demás lenguajes.

```
def crear_contador(
    inicial=0,
    incremento=1
):

    valido_inicial=(
        isinstance(inicial,(int,float))
        and not isinstance(inicial,bool)
    )

    valido_incremento=(
        isinstance(incremento,(int,float))
        and not isinstance(incremento,bool)
    )

    if (
        not valido_inicial
        or not valido_incremento
    ):
        raise TypeError(
            "Inicial e incremento deben ser escalares reales"
        )
```

```
    if incremento==0:
        raise ValueError(
            "El incremento no puede ser cero"
        )

    return {
        "inicial":inicial,
        "incremento":incremento,
        "actual":inicial
    }


def avanzar(c):

    c["actual"]=(
        c["actual"]
        +c["incremento"]
    )

    valor=c["actual"]

    return c,valor


def consultar(c):

    return c["actual"]


def reiniciar(c):

    c["actual"]=c["inicial"]

    return c


c1=crear_contador()

c1,v=avanzar(c1)
print(v)

c1,v=avanzar(c1)
print(v)

c1,v=avanzar(c1)
print(v)

print(consultar(c1))
```

```
c2=crear_contador(10,2)

c2,v=avanzar(c2)
print(v)

c2,v=avanzar(c2)
print(v)

print(consultar(c1))
print(consultar(c2))


c1=reiniciar(c1)

print(consultar(c1))

c1,v=avanzar(c1)
print(v)


c2=reiniciar(c2)

print(consultar(c2))


c3=crear_contador(100,-10)

c3,v=avanzar(c3)
print(v)

c3,v=avanzar(c3)
print(v)

c3,v=avanzar(c3)
print(v)
```

Perl. En Perl se utiliza una referencia a un *hash* para representar el estado. Cada contador posee su propio *hash*; por lo tanto no es necesaria una variable global compartida.

```
use strict;
use warnings;

use Scalar::Util qw(looks_like_number);


sub crear_contador {

    my (
        $inicial,
```

```
        $incremento
    )=@_;

    $inicial=0
        unless defined($inicial);

    $incremento=1
        unless defined($incremento);

    die(
        "Inicial e incremento ",
        "deben ser escalares reales\n"
    )
        unless
            looks_like_number($inicial)
            &&
            looks_like_number($incremento);

    die "El incremento no puede ser cero\n"
        if $incremento==0;

    return {
        inicial=>$inicial,
        incremento=>$incremento,
        actual=>$inicial
    };

}


sub avanzar {

    my ($c)=@_;

    $c->{actual}=
        $c->{actual}
        +$c->{incremento};

    my $valor=
        $c->{actual};

    return ($c,$valor);

}


sub consultar {

    my ($c)=@_;

    return $c->{actual};
```

```
}


sub reiniciar {

    my ($c)=@_;

    $c->{actual}=
        $c->{inicial};

    return $c;

}


my $c1=crear_contador();

my $v;

($c1,$v)=avanzar($c1);
print $v,"\n";

($c1,$v)=avanzar($c1);
print $v,"\n";

($c1,$v)=avanzar($c1);
print $v,"\n";

print consultar($c1),"\n";


my $c2=crear_contador(10,2);

($c2,$v)=avanzar($c2);
print $v,"\n";

($c2,$v)=avanzar($c2);
print $v,"\n";

print consultar($c1),"\n";
print consultar($c2),"\n";


$c1=reiniciar($c1);

print consultar($c1),"\n";

($c1,$v)=avanzar($c1);
print $v,"\n";
```

```
$c2=reiniciar($c2);

print consultar($c2),"\n";


my $c3=
    crear_contador(100,-10);

($c3,$v)=avanzar($c3);
print $v,"\n";

($c3,$v)=avanzar($c3);
print $v,"\n";

($c3,$v)=avanzar($c3);
print $v,"\n";
```

JavaScript. En JavaScript se utiliza un objeto para representar el estado. Las funciones modifican ese objeto y lo devuelven para conservar la misma interfaz conceptual utilizada en los otros lenguajes.

```
function crearContador(
    inicial=0,
    incremento=1
){

    let validoInicial=
        typeof inicial==="number"
        && Number.isFinite(inicial)

    let validoIncremento=
        typeof incremento==="number"
        && Number.isFinite(incremento)

    if(
        !validoInicial
        || !validoIncremento
    ){

        throw new TypeError(
            "Inicial e incremento deben ser escalares reales"
        )

    }

    if(incremento===0){

        throw new RangeError(
            "El incremento no puede ser cero"
```

```
        )

    }

    return {
        inicial:inicial,
        incremento:incremento,
        actual:inicial
    }

}


function avanzar(c){

    c.actual=
        c.actual
        +c.incremento

    let valor=c.actual

    return [c,valor]

}


function consultar(c){

    return c.actual

}


function reiniciar(c){

    c.actual=c.inicial

    return c

}


let c1=crearContador()

let resultado

resultado=avanzar(c1)
c1=resultado[0]
console.log(resultado[1])

resultado=avanzar(c1)
```

```
c1=resultado[0]
console.log(resultado[1])

resultado=avanzar(c1)
c1=resultado[0]
console.log(resultado[1])

console.log(consultar(c1))


let c2=crearContador(10,2)

resultado=avanzar(c2)
c2=resultado[0]
console.log(resultado[1])

resultado=avanzar(c2)
c2=resultado[0]
console.log(resultado[1])

console.log(consultar(c1))
console.log(consultar(c2))


c1=reiniciar(c1)

console.log(consultar(c1))

resultado=avanzar(c1)
c1=resultado[0]
console.log(resultado[1])


c2=reiniciar(c2)

console.log(consultar(c2))


let c3=
    crearContador(100,-10)

resultado=avanzar(c3)
c3=resultado[0]
console.log(resultado[1])

resultado=avanzar(c3)
c3=resultado[0]
console.log(resultado[1])

resultado=avanzar(c3)
c3=resultado[0]
```

```
console.log(resultado[1])
```

27. **Ejercicio:** Currificación de funciones
    Considere la función

$$P_f = P(1 + d)(1 + i),$$

donde:

- P es el precio base;
- d es un recargo o descuento expresado como fracción;
- i es un impuesto expresado como fracción.

Por ejemplo, un descuento del 10 % se representa mediante

$$d = -0{,}10,$$

mientras que un impuesto del 21 % se representa mediante

$$i = 0{,}21.$$

Implemente primero una función ordinaria con la interfaz conceptual

```
precio(P, d, i) -> precio_final
```

Luego construya una versión especializada por etapas con el comportamiento

```
con_impuesto <- precio_especializado(i)

con_descuento <- con_impuesto(d)

precio_final <- con_descuento(P)
```

Por ejemplo:

```
con_iva21 <- precio_especializado(0.21)

con_descuento10 <- con_iva21(-0.10)

con_descuento10(1000) -> 1089
con_descuento10(2500) -> 2722.5
```

La primera llamada fija el impuesto. La segunda fija el descuento o recargo. La función obtenida finalmente sólo necesita recibir el precio base.

Para este ejercicio se adoptará el siguiente contrato:

- P debe ser un número real, finito y mayor o igual que cero;
- d debe ser un número real y finito;
- i debe ser un número real y finito;
- la función deberá rechazar argumentos que no cumplan estas condiciones.

Realice las siguientes actividades:

a) Implemente la función ordinaria

```
precio(P, d, i)
```

y compruebe que

$$\text{precio}(1000, -0{,}10, 0{,}21) = 1089.$$

b) Implemente

```
precio_especializado(i)
```

de modo que devuelva una función que reciba únicamente d.

c) La función obtenida en el punto anterior deberá devolver, a su vez, una función que reciba únicamente P.

d) Construya

```
con_iva21 <- precio_especializado(0.21)
```

y explique qué valor queda conservado en esa función.

e) Construya

```
con_descuento10 <- con_iva21(-0.10)
```

y explique qué valores quedan conservados ahora.

f) Compruebe que

```
con_descuento10(1000)
con_descuento10(2500)
```

producen respectivamente

$$1089$$

y

$$2722{,}5.$$

g) Utilizando la misma función `con_iva21`, construya también

```
con_recargo5 <- con_iva21(0.05)
```

y compruebe que

$$\text{con_recargo5}(1000) = 1270{,}5.$$

h) Construya otra especialización independiente:

```
con_iva10 <- precio_especializado(0.10)

con_iva10_descuento20 <- con_iva10(-0.20)
```

y compruebe que

$$\text{con_iva10_descuento20}(1000) = 880.$$

i) Compruebe que la creación de `con_iva10` no modifica `con_iva21`.

j) Aplique `con_descuento10` a los precios

$$[1000, 2500, 5000]$$

y obtenga

$$[1089, 2722{,}5, 5445].$$

k) Calcule los mismos resultados mediante la función ordinaria `precio` y compruebe que coincidan.

l) Explique qué ventaja puede tener una función especializada cuando los mismos valores de i y d deben utilizarse para muchos precios.

m) Explique qué información queda almacenada en cada una de las siguientes variables:

```
con_iva21
con_descuento10
con_recargo5
con_iva10
```

n) Distinga entre:

- llamar repetidamente a una función de tres argumentos;
- fijar algunos argumentos y obtener una nueva función;
- transformar una operación de varios argumentos en una secuencia de funciones de un solo argumento.

El comportamiento conceptual deberá ser el mismo en los cinco lenguajes:

```
precio_especializado(i)
        |
        v
funcion que espera d
        |
        v
```

```
funcion que espera P
        |
        v
precio final
```

Scilab. Las versiones recientes de Scilab permiten construir funciones lambda que conservan los valores existentes en el momento de su creación. Así pueden implementarse directamente las distintas etapas de especialización.

```
function comprobar_real(x,nombre)

    valido=type(x)==1 & ..
           size(x,"*")==1 & ..
           isreal(x);

    if valido then
        valido=~isnan(x) & ..
               abs(x)<>%inf;
    end

    if ~valido then
        error(nombre+" debe ser un escalar real y finito")
    end

endfunction


function precio_final=precio(P,d,i)

    comprobar_real(P,"P");
    comprobar_real(d,"d");
    comprobar_real(i,"i");

    if P<0 then
        error("P debe ser mayor o igual que cero")
    end

    precio_final=P*(1+d)*(1+i);

endfunction


function f=especializar_descuento(i)

    comprobar_real(i,"i");

    f=#(d) -> crear_funcion_precio(i,d);

endfunction
```

```
function f=crear_funcion_precio(i,d)

    comprobar_real(d,"d");

    f=#(P) -> precio(P,d,i);

endfunction


function f=precio_especializado(i)

    f=especializar_descuento(i);

endfunction


disp(precio(1000,-0.10,0.21))


con_iva21=precio_especializado(0.21);

con_descuento10=con_iva21(-0.10);

disp(con_descuento10(1000))
disp(con_descuento10(2500))


con_recargo5=con_iva21(0.05);

disp(con_recargo5(1000))


con_iva10=precio_especializado(0.10);

con_iva10_descuento20= ..
    con_iva10(-0.20);

disp(
    con_iva10_descuento20(1000)
)


// Comprobar que con_iva21 no cambio.

disp(con_descuento10(1000))


precios=[
    1000
    2500
```

```
    5000
];

resultado=zeros(size(precios));

for k=1:length(precios)

    resultado(k)= ..
        con_descuento10(precios(k));

end

disp(resultado)


// Mismos valores con la funcion ordinaria.

resultado_ordinario= ..
    zeros(size(precios));

for k=1:length(precios)

    resultado_ordinario(k)= ..
        precio(
            precios(k),
            -0.10,
            0.21
        );

end

disp(resultado_ordinario)
```

Octave. Octave permite almacenar funciones en variables mediante manejadores de funciones. Las funciones anónimas utilizadas aquí conservan los parámetros que se fijan en cada etapa.

```
function comprobar_real(x,nombre)

  valido=isnumeric(x) ...
         && isscalar(x) ...
         && isreal(x) ...
         && isfinite(x);

  if !valido
    error(
      [nombre " debe ser un escalar real y finito"]
    )
  endif
```

```
endfunction


function precio_final=precio(P,d,i)

  comprobar_real(P,"P");
  comprobar_real(d,"d");
  comprobar_real(i,"i");

  if P<0
    error("P debe ser mayor o igual que cero")
  endif

  precio_final=P*(1+d)*(1+i);

endfunction


function f=crear_funcion_precio(i,d)

  comprobar_real(d,"d");

  f=@(P) precio(P,d,i);

endfunction


function f=precio_especializado(i)

  comprobar_real(i,"i");

  f=@(d) crear_funcion_precio(i,d);

endfunction


disp(precio(1000,-0.10,0.21))


con_iva21=precio_especializado(0.21);

con_descuento10=con_iva21(-0.10);

disp(con_descuento10(1000))
disp(con_descuento10(2500))


con_recargo5=con_iva21(0.05);

disp(con_recargo5(1000))
```

```
con_iva10=precio_especializado(0.10);

con_iva10_descuento20= ...
  con_iva10(-0.20);

disp(
  con_iva10_descuento20(1000)
)


disp(con_descuento10(1000))


precios=[
  1000
  2500
  5000
];

resultado=zeros(size(precios));

for k=1:length(precios)

  resultado(k)= ...
    con_descuento10(precios(k));

endfor

disp(resultado)


resultado_ordinario= ...
  zeros(size(precios));

for k=1:length(precios)

  resultado_ordinario(k)= ...
    precio(
      precios(k),
      -0.10,
      0.21
    );

endfor

disp(resultado_ordinario)
```

Python. En Python, una función puede devolver otra función. Las funciones internas conservan los valores de las variables del entorno en el que fueron creadas.

```
import math


def comprobar_real(x,nombre):

    valido=(
        isinstance(x,(int,float))
        and not isinstance(x,bool)
        and math.isfinite(x)
    )

    if not valido:
        raise TypeError(
            nombre+
            " debe ser un escalar real y finito"
        )


def precio(P,d,i):

    comprobar_real(P,"P")
    comprobar_real(d,"d")
    comprobar_real(i,"i")

    if P<0:
        raise ValueError(
            "P debe ser mayor o igual que cero"
        )

    return P*(1+d)*(1+i)


def precio_especializado(i):

    comprobar_real(i,"i")

    def con_impuesto(d):

        comprobar_real(d,"d")

        def aplicar(P):

            return precio(P,d,i)

        return aplicar

    return con_impuesto


print(
```

```
    precio(
        1000,
        -0.10,
        0.21
    )
)


con_iva21=precio_especializado(0.21)

con_descuento10=con_iva21(-0.10)

print(con_descuento10(1000))
print(con_descuento10(2500))


con_recargo5=con_iva21(0.05)

print(con_recargo5(1000))


con_iva10=precio_especializado(0.10)

con_iva10_descuento20=(
    con_iva10(-0.20)
)

print(
    con_iva10_descuento20(1000)
)


print(con_descuento10(1000))


precios=[
    1000,
    2500,
    5000
]

resultado=[]

for P in precios:

    resultado.append(
        con_descuento10(P)
    )

print(resultado)
```

```
resultado_ordinario=[]

for P in precios:

    resultado_ordinario.append(
        precio(
            P,
            -0.10,
            0.21
        )
    )

print(resultado_ordinario)
```

Perl. En Perl, las funciones anónimas se representan mediante referencias a subrutinas. Una subrutina anónima puede conservar variables pertenecientes al entorno en el que fue creada.

```
use strict;
use warnings;

use Scalar::Util qw(looks_like_number);


sub comprobar_real {

    my ($x,$nombre)=@_;

    my $valido=
        defined($x)
        && looks_like_number($x);

    die(
        $nombre,
        " debe ser un escalar real y finito\n"
    )
        unless $valido;

}


sub precio {

    my ($P,$d,$i)=@_;

    comprobar_real($P,'P');
    comprobar_real($d,'d');
    comprobar_real($i,'i');
```

```
    die "P debe ser mayor o igual que cero\n"
        if $P<0;

    return $P*(1+$d)*(1+$i);

}


sub precio_especializado {

    my ($i)=@_;

    comprobar_real($i,'i');

    return sub {

        my ($d)=@_;

        comprobar_real($d,'d');

        return sub {

            my ($P)=@_;

            return precio(
                $P,
                $d,
                $i
            );

        };

    };

}


print(
    precio(
        1000,
        -0.10,
        0.21
    ),
    "\n"
);


my $con_iva21=
    precio_especializado(0.21);

my $con_descuento10=
```

```
    $con_iva21->(-0.10);

print $con_descuento10->(1000),"\n";
print $con_descuento10->(2500),"\n";


my $con_recargo5=
    $con_iva21->(0.05);

print $con_recargo5->(1000),"\n";


my $con_iva10=
    precio_especializado(0.10);

my $con_iva10_descuento20=
    $con_iva10->(-0.20);

print(
    $con_iva10_descuento20->(1000),
    "\n"
);


print $con_descuento10->(1000),"\n";


my @precios=(
    1000,
    2500,
    5000
);

my @resultado;

for my $P (@precios) {

    push(
        @resultado,
        $con_descuento10->($P)
    );

}

print join(" ",@resultado),"\n";


my @resultado_ordinario;

for my $P (@precios) {
```

```
    push(
        @resultado_ordinario,
        precio(
            $P,
            -0.10,
            0.21
        )
    );

}

print(
    join(
        " ",
        @resultado_ordinario
    ),
    "\n"
);
```

JavaScript. En JavaScript, una función puede devolver otra función y las funciones internas conservan las variables necesarias del entorno en el que fueron creadas.

```
function comprobarReal(x,nombre){

    let valido=
        typeof x==="number"
        && Number.isFinite(x)

    if(!valido){

        throw new TypeError(
            nombre+
            " debe ser un escalar real y finito"
        )

    }

}


function precio(P,d,i){

    comprobarReal(P,"P")
    comprobarReal(d,"d")
    comprobarReal(i,"i")

    if(P<0){

        throw new RangeError(
```

```
            "P debe ser mayor o igual que cero"
        )

    }

    return P*(1+d)*(1+i)

}


function precioEspecializado(i){

    comprobarReal(i,"i")

    return function(d){

        comprobarReal(d,"d")

        return function(P){

            return precio(P,d,i)

        }

    }

}


console.log(
    precio(
        1000,
        -0.10,
        0.21
    )
)


let conIva21=
    precioEspecializado(0.21)

let conDescuento10=
    conIva21(-0.10)

console.log(
    conDescuento10(1000)
)

console.log(
    conDescuento10(2500)
)
```

```
let conRecargo5=
    conIva21(0.05)

console.log(
    conRecargo5(1000)
)


let conIva10=
    precioEspecializado(0.10)

let conIva10Descuento20=
    conIva10(-0.20)

console.log(
    conIva10Descuento20(1000)
)


console.log(
    conDescuento10(1000)
)


let precios=[
    1000,
    2500,
    5000
]

let resultado=[]

for(const P of precios){

    resultado.push(
        conDescuento10(P)
    )

}

console.log(resultado)


let resultadoOrdinario=[]

for(const P of precios){

    resultadoOrdinario.push(
        precio(
```

```
                P,
                -0.10,
                0.21
            )
        )

}

console.log(
    resultadoOrdinario
)
```

28. **Ejercicio:** Medición con operadores

    Defina un tipo de dato denominado `Medicion` para representar una medición de longitud mediante los campos

    ```
    valor
    unidad
    incertidumbre
    ```

    En este ejercicio se admitirán solamente las unidades

    ```
    "m"
    "cm"
    ```

    y se utilizará la relación

    $$1\,\mathrm{m} = 100\,\mathrm{cm}.$$

    La incertidumbre deberá expresarse siempre en la misma unidad que el valor.

    Por ejemplo,

    ```
    Medicion(2.05, "m", 0.02)
    ```

    representa

    $$(2{,}05 \pm 0{,}02)\,\mathrm{m}.$$

    El constructor deberá comprobar que:

    - el valor sea un número real, escalar y finito;
    - la unidad sea `"m"` o `"cm"`;
    - la incertidumbre sea un número real, escalar, finito y mayor o igual que cero.

    Implemente las siguientes operaciones conceptuales:

```
convertir_medicion(m, unidad)
sumar_mediciones(a, b)
restar_mediciones(a, b)
multiplicar_escalar(m, k)
dividir_escalar(m, k)
comparar_mediciones(a, b)
texto_medicion(m)
```

Se utilizarán exactamente las siguientes reglas.

Conversión de unidades

Al convertir de metros a centímetros,

$$v_{\mathrm{cm}} = 100 v_{\mathrm{m}}, \qquad u_{\mathrm{cm}} = 100 u_{\mathrm{m}}.$$

Al convertir de centímetros a metros,

$$v_{\mathrm{m}} = \frac{v_{\mathrm{cm}}}{100}, \qquad u_{\mathrm{m}} = \frac{u_{\mathrm{cm}}}{100}.$$

Suma y resta

Antes de operar, convierta la segunda medición a la unidad de la primera.

El resultado conservará la unidad de la primera medición.

Para este ejercicio se supondrá que las incertidumbres son independientes y se propagará la incertidumbre mediante

$$u_r = \sqrt{u_a^2 + u_b^2}.$$

Multiplicación por un escalar

Para un escalar k,

$$v_r = kv, \qquad u_r = |k| u.$$

División por un escalar

Para $k \neq 0$,

$$v_r = \frac{v}{k}, \qquad u_r = \frac{u}{|k|}.$$

La división por cero deberá producir un error.

Comparación

La función

```
comparar_mediciones(a, b)
```

comparará solamente los valores nominales, después de expresar ambas mediciones en metros.

Devolverá:

```
-1    si a < b
 0    si a = b
 1    si a > b
```

La incertidumbre no intervendrá en esta comparación. Esta es una decisión simplificada adoptada específicamente para este ejercicio.

Representación textual

La representación deberá mostrar valor, unidad e incertidumbre, por ejemplo:

```
2.05 m +/- 0.02 m
```

Utilice inicialmente:

```
m1 = Medicion(2.05, "m", 0.02)
m2 = Medicion(30, "cm", 1)
```

y compruebe los siguientes resultados:

$$m_1 + m_2 = (2{,}35 \pm \sqrt{0{,}02^2 + 0{,}01^2})\,\mathrm{m},$$

$$m_1 - m_2 = (1{,}75 \pm \sqrt{0{,}02^2 + 0{,}01^2})\,\mathrm{m},$$

$$2m_1 = (4{,}10 \pm 0{,}04)\,\mathrm{m},$$

$$\frac{m_1}{2} = (1{,}025 \pm 0{,}01)\,\mathrm{m},$$

y

$$m_1 = (205 \pm 2)\,\mathrm{cm}.$$

Además,

```
comparar_mediciones(m1, m2)
```

deberá producir

```
1
```

porque $2{,}05\,\mathrm{m} > 30\,\mathrm{cm}$.

Realice las siguientes actividades:

a) Implemente el constructor y las siete operaciones indicadas.

b) Compruebe los resultados anteriores.

c) Compruebe que sumar

```
m2 + m1
```

produce el resultado expresado en centímetros, ya que la unidad de la primera medición debe conservarse.

d) Compruebe que la conversión no modifica la medición original.

e) Intente construir mediciones con:

```
unidad = "kg"
incertidumbre = -0.1
valor = NaN
```

y compruebe que sean rechazadas.

f) Intente dividir una medición por cero.

g) Explique por qué la incertidumbre también debe cambiar cuando se cambia la unidad.

h) Explique por qué no sería correcto sumar directamente `2.05` y `30` sin tener en cuenta las unidades.

i) Implemente la suma y la resta mediante sobrecarga de operadores cuando el mecanismo del lenguaje lo permita razonablemente.

j) Compare

```
sumar_mediciones(m1, m2)
```

con una expresión sobrecargada como

```
m1 + m2
```

cuando ambas formas estén disponibles.

k) Explique qué cambia con la sobrecarga: la forma de escribir la operación o su significado matemático.

La semántica de todas las operaciones deberá ser idéntica en los cinco lenguajes. La sintaxis de sobrecarga puede ser diferente porque forma parte del diseño propio de cada lenguaje.

Scilab. En Scilab puede utilizarse una `tlist` para definir el tipo `medicion`. Las operaciones comunes se implementan mediante funciones independientes. Como extensión se sobrecargan aquí los operadores `+` y `-`.

```
function comprobar_real(x,nombre)

    valido=type(x)==1 & ..
           size(x,"*")==1 & ..
           isreal(x);
```

```
    if valido then
        valido=~isnan(x) & ..
               abs(x)<>%inf;
    end

    if ~valido then
        error(nombre+" debe ser un escalar real y finito")
    end

endfunction


function m=Medicion(valor,unidad,incertidumbre)

    comprobar_real(valor,"valor");
    comprobar_real(incertidumbre,"incertidumbre");

    if type(unidad)<>10 | ..
       size(unidad,"*")<>1 then

        error("Unidad no valida")

    end

    if unidad<>"m" & unidad<>"cm" then
        error("Unidad no valida")
    end

    if incertidumbre<0 then
        error("La incertidumbre no puede ser negativa")
    end

    m=tlist( ..
        ["medicion","valor","unidad","incertidumbre"], ..
        valor, ..
        unidad, ..
        incertidumbre ..
    );

endfunction


function r=convertir_medicion(m,unidad)

    if unidad<>"m" & unidad<>"cm" then
        error("Unidad no valida")
    end

    if m.unidad==unidad then
```

```
        r=Medicion(
            m.valor,
            m.unidad,
            m.incertidumbre
        );

    elseif m.unidad=="m" & unidad=="cm" then

        r=Medicion(
            100*m.valor,
            "cm",
            100*m.incertidumbre
        );

    else

        r=Medicion(
            m.valor/100,
            "m",
            m.incertidumbre/100
        );

    end

endfunction


function r=sumar_mediciones(a,b)

    b2=convertir_medicion(
        b,
        a.unidad
    );

    incertidumbre=sqrt( ..
        a.incertidumbre^2 + ..
        b2.incertidumbre^2 ..
    );

    r=Medicion(
        a.valor+b2.valor,
        a.unidad,
        incertidumbre
    );

endfunction


function r=restar_mediciones(a,b)

    b2=convertir_medicion(
```

```
        b,
        a.unidad
    );

    incertidumbre=sqrt( ..
        a.incertidumbre^2 + ..
        b2.incertidumbre^2 ..
    );

    r=Medicion(
        a.valor-b2.valor,
        a.unidad,
        incertidumbre
    );

endfunction


function r=multiplicar_escalar(m,k)

    comprobar_real(k,"escalar");

    r=Medicion(
        m.valor*k,
        m.unidad,
        m.incertidumbre*abs(k)
    );

endfunction


function r=dividir_escalar(m,k)

    comprobar_real(k,"escalar");

    if k==0 then
        error("Division por cero")
    end

    r=Medicion(
        m.valor/k,
        m.unidad,
        m.incertidumbre/abs(k)
    );

endfunction


function resultado=comparar_mediciones(a,b)

    a2=convertir_medicion(a,"m");
```

```
    b2=convertir_medicion(b,"m");

    if a2.valor<b2.valor then

        resultado=-1;

    elseif a2.valor>b2.valor then

        resultado=1;

    else

        resultado=0;

    end

endfunction


function s=texto_medicion(m)

    s=msprintf(
        "%.12g %s +/- %.12g %s",
        m.valor,
        m.unidad,
        m.incertidumbre,
        m.unidad
    );

endfunction


// Sobrecarga de +.

function r=%medicion_a_medicion(a,b)

    r=sumar_mediciones(a,b);

endfunction


// Sobrecarga de -.

function r=%medicion_s_medicion(a,b)

    r=restar_mediciones(a,b);

endfunction


// Presentacion del nuevo tipo.
```

```
function %medicion_p(m)

    disp(texto_medicion(m));

endfunction


m1=Medicion(
    2.05,
    "m",
    0.02
);

m2=Medicion(
    30,
    "cm",
    1
);


suma=m1+m2;
resta=m1-m2;

doble=multiplicar_escalar(
    m1,
    2
);

mitad=dividir_escalar(
    m1,
    2
);

m1_cm=convertir_medicion(
    m1,
    "cm"
);


disp(suma)
disp(resta)
disp(doble)
disp(mitad)
disp(m1_cm)

disp(
    comparar_mediciones(
        m1,
        m2
    )
```

```
)
```

Octave. En Octave puede definirse una clase y asociar los operadores a métodos. Los métodos `plus`, `minus`, `mtimes` y `mrdivide` implementan respectivamente las operaciones utilizadas en este ejercicio. Guarde la clase en el archivo `Medicion.m`.

```
classdef Medicion

  properties

    valor
    unidad
    incertidumbre

  endproperties


  methods

    function obj=Medicion(valor,unidad,incertidumbre)

      valido_valor= ...
        isnumeric(valor) ...
        && isscalar(valor) ...
        && isreal(valor) ...
        && isfinite(valor);

      valido_incertidumbre= ...
        isnumeric(incertidumbre) ...
        && isscalar(incertidumbre) ...
        && isreal(incertidumbre) ...
        && isfinite(incertidumbre);

      if !valido_valor
        error("Valor no valido")
      endif

      if !valido_incertidumbre
        error("Incertidumbre no valida")
      endif

      if !ischar(unidad) ...
         || !(strcmp(unidad,"m") ...
              || strcmp(unidad,"cm"))

        error("Unidad no valida")

      endif

      if incertidumbre<0
```

```
      error(
        "La incertidumbre no puede ser negativa"
      )
    endif

    obj.valor=valor;
    obj.unidad=unidad;
    obj.incertidumbre=incertidumbre;

  endfunction


  function r=convertir(obj,unidad)

    if !(strcmp(unidad,"m") ...
         || strcmp(unidad,"cm"))

      error("Unidad no valida")

    endif

    if strcmp(obj.unidad,unidad)

      r=Medicion(
        obj.valor,
        obj.unidad,
        obj.incertidumbre
      );

    elseif strcmp(obj.unidad,"m") ...
           && strcmp(unidad,"cm")

      r=Medicion(
        100*obj.valor,
        "cm",
        100*obj.incertidumbre
      );

    else

      r=Medicion(
        obj.valor/100,
        "m",
        obj.incertidumbre/100
      );

    endif

  endfunction
```

```
function r=plus(a,b)

  if !isa(b,"Medicion")
    error("Se esperaba una Medicion")
  endif

  b2=b.convertir(a.unidad);

  u=sqrt(
    a.incertidumbre^2 ...
    +b2.incertidumbre^2
  );

  r=Medicion(
    a.valor+b2.valor,
    a.unidad,
    u
  );

endfunction


function r=minus(a,b)

  if !isa(b,"Medicion")
    error("Se esperaba una Medicion")
  endif

  b2=b.convertir(a.unidad);

  u=sqrt(
    a.incertidumbre^2 ...
    +b2.incertidumbre^2
  );

  r=Medicion(
    a.valor-b2.valor,
    a.unidad,
    u
  );

endfunction


function r=mtimes(a,k)

  valido= ...
    isnumeric(k) ...
    && isscalar(k) ...
    && isreal(k) ...
    && isfinite(k);
```

```
    if !valido
      error("Escalar no valido")
    endif

    r=Medicion(
      a.valor*k,
      a.unidad,
      a.incertidumbre*abs(k)
    );

  endfunction


  function r=mrdivide(a,k)

    valido= ...
      isnumeric(k) ...
      && isscalar(k) ...
      && isreal(k) ...
      && isfinite(k);

    if !valido
      error("Escalar no valido")
    endif

    if k==0
      error("Division por cero")
    endif

    r=Medicion(
      a.valor/k,
      a.unidad,
      a.incertidumbre/abs(k)
    );

  endfunction


  function resultado=comparar(a,b)

    a2=a.convertir("m");
    b2=b.convertir("m");

    if a2.valor<b2.valor

      resultado=-1;

    elseif a2.valor>b2.valor

      resultado=1;
```

```
      else

        resultado=0;

      endif

    endfunction


    function s=texto(obj)

      s=sprintf(
        "%.12g %s +/- %.12g %s",
        obj.valor,
        obj.unidad,
        obj.incertidumbre,
        obj.unidad
      );

    endfunction


    function disp(obj)

      printf("%s\n",obj.texto());

    endfunction

  endmethods

endclassdef
```

Archivo de prueba:

```
m1=Medicion(
  2.05,
  "m",
  0.02
);

m2=Medicion(
  30,
  "cm",
  1
);


suma=m1+m2;
resta=m1-m2;
```

```
doble=m1*2;
mitad=m1/2;

m1_cm=m1.convertir("cm");


disp(suma)
disp(resta)
disp(doble)
disp(mitad)
disp(m1_cm)

disp(
  m1.comparar(m2)
)
```

Python. En Python se utiliza una clase. Los métodos especiales `__add__`, `__sub__`, `__mul__` y `__truediv__` permiten escribir las operaciones aritméticas con los operadores habituales.

```
import math


class Medicion:

    def __init__(
        self,
        valor,
        unidad,
        incertidumbre
    ):

        valido_valor=(
            isinstance(valor,(int,float))
            and not isinstance(valor,bool)
            and math.isfinite(valor)
        )

        valido_incertidumbre=(
            isinstance(
                incertidumbre,
                (int,float)
            )
            and not isinstance(
                incertidumbre,
                bool
            )
            and math.isfinite(
                incertidumbre
            )
```

```
        )

        if not valido_valor:
            raise TypeError(
                "Valor no valido"
            )

        if not valido_incertidumbre:
            raise TypeError(
                "Incertidumbre no valida"
            )

        if unidad not in ("m","cm"):
            raise ValueError(
                "Unidad no valida"
            )

        if incertidumbre<0:
            raise ValueError(
                "La incertidumbre no puede ser negativa"
            )

        self.valor=valor
        self.unidad=unidad
        self.incertidumbre=incertidumbre


    def convertir(self,unidad):

        if unidad not in ("m","cm"):
            raise ValueError(
                "Unidad no valida"
            )

        if self.unidad==unidad:

            return Medicion(
                self.valor,
                self.unidad,
                self.incertidumbre
            )

        if (
            self.unidad=="m"
            and unidad=="cm"
        ):

            return Medicion(
                100*self.valor,
                "cm",
                100*self.incertidumbre
```

```
            )

        return Medicion(
            self.valor/100,
            "m",
            self.incertidumbre/100
        )


    def __add__(self,other):

        if not isinstance(
            other,
            Medicion
        ):
            return NotImplemented

        other2=other.convertir(
            self.unidad
        )

        u=math.sqrt(
            self.incertidumbre**2
            +other2.incertidumbre**2
        )

        return Medicion(
            self.valor+other2.valor,
            self.unidad,
            u
        )


    def __sub__(self,other):

        if not isinstance(
            other,
            Medicion
        ):
            return NotImplemented

        other2=other.convertir(
            self.unidad
        )

        u=math.sqrt(
            self.incertidumbre**2
            +other2.incertidumbre**2
        )

        return Medicion(
```

```
            self.valor-other2.valor,
            self.unidad,
            u
        )


    def __mul__(self,k):

        valido=(
            isinstance(k,(int,float))
            and not isinstance(k,bool)
            and math.isfinite(k)
        )

        if not valido:
            return NotImplemented

        return Medicion(
            self.valor*k,
            self.unidad,
            self.incertidumbre*abs(k)
        )


    def __rmul__(self,k):

        return self*k


    def __truediv__(self,k):

        valido=(
            isinstance(k,(int,float))
            and not isinstance(k,bool)
            and math.isfinite(k)
        )

        if not valido:
            return NotImplemented

        if k==0:
            raise ZeroDivisionError(
                "Division por cero"
            )

        return Medicion(
            self.valor/k,
            self.unidad,
            self.incertidumbre/abs(k)
        )
```

```
    def comparar(self,other):

        a=self.convertir("m")
        b=other.convertir("m")

        if a.valor<b.valor:
            return -1

        if a.valor>b.valor:
            return 1

        return 0


    def __str__(self):

        return (
            str(self.valor)
            +" "
            +self.unidad
            +" +/- "
            +str(self.incertidumbre)
            +" "
            +self.unidad
        )


def convertir_medicion(m,unidad):

    return m.convertir(unidad)


def sumar_mediciones(a,b):

    return a+b


def restar_mediciones(a,b):

    return a-b


def multiplicar_escalar(m,k):

    return m*k


def dividir_escalar(m,k):

    return m/k
```

```
def comparar_mediciones(a,b):

    return a.comparar(b)


def texto_medicion(m):

    return str(m)


m1=Medicion(
    2.05,
    "m",
    0.02
)

m2=Medicion(
    30,
    "cm",
    1
)


suma=m1+m2
resta=m1-m2

doble=m1*2
mitad=m1/2

m1_cm=m1.convertir("cm")


print(suma)
print(resta)
print(doble)
print(mitad)
print(m1_cm)

print(
    comparar_mediciones(
        m1,
        m2
    )
)
```

Perl. En Perl se utiliza una clase representada mediante una referencia a un *hash*. El módulo `overload` permite asociar operadores con métodos de la clase.

```
use strict;
use warnings;

package Medicion;

use Scalar::Util qw(looks_like_number);
use POSIX qw(isfinite);

use overload
    '+'  => 'sumar',
    '-'  => 'restar',
    '*'  => 'multiplicar',
    '/'  => 'dividir',
    '""' => 'texto',
    fallback => 0;


sub comprobar_real {

    my ($x,$nombre)=@_;

    my $valido=
        defined($x)
        && looks_like_number($x)
        && isfinite($x);

    die "$nombre no valido\n"
        unless $valido;

}


sub new {

    my (
        $class,
        $valor,
        $unidad,
        $incertidumbre
    )=@_;

    comprobar_real(
        $valor,
        'Valor'
    );

    comprobar_real(
        $incertidumbre,
        'Incertidumbre'
    );
```

```
    die "Unidad no valida\n"
        unless
            $unidad eq 'm'
            || $unidad eq 'cm';

    die(
        "La incertidumbre ",
        "no puede ser negativa\n"
    )
        if $incertidumbre<0;

    my $self={
        valor=>$valor,
        unidad=>$unidad,
        incertidumbre=>$incertidumbre
    };

    return bless $self,$class;

}


sub convertir {

    my ($self,$unidad)=@_;

    die "Unidad no valida\n"
        unless
            $unidad eq 'm'
            || $unidad eq 'cm';

    if ($self->{unidad} eq $unidad) {

        return Medicion->new(
            $self->{valor},
            $self->{unidad},
            $self->{incertidumbre}
        );

    }

    if (
        $self->{unidad} eq 'm'
        &&
        $unidad eq 'cm'
    ) {

        return Medicion->new(
            100*$self->{valor},
            'cm',
            100*$self->{incertidumbre}
```

```
        );

    }

    return Medicion->new(
        $self->{valor}/100,
        'm',
        $self->{incertidumbre}/100
    );

}


sub sumar {

    my ($a,$b,$invertido)=@_;

    die "Se esperaba una Medicion\n"
        unless ref($b) eq 'Medicion';

    my $b2=
        $b->convertir(
            $a->{unidad}
        );

    my $u=sqrt(
        $a->{incertidumbre}**2
        +$b2->{incertidumbre}**2
    );

    return Medicion->new(
        $a->{valor}+$b2->{valor},
        $a->{unidad},
        $u
    );

}


sub restar {

    my ($a,$b,$invertido)=@_;

    die "Se esperaba una Medicion\n"
        unless ref($b) eq 'Medicion';

    die "Orden de operandos no soportado\n"
        if $invertido;

    my $b2=
        $b->convertir(
```

```
            $a->{unidad}
        );

    my $u=sqrt(
        $a->{incertidumbre}**2
        +$b2->{incertidumbre}**2
    );

    return Medicion->new(
        $a->{valor}-$b2->{valor},
        $a->{unidad},
        $u
    );

}


sub multiplicar {

    my ($self,$k,$invertido)=@_;

    comprobar_real(
        $k,
        'Escalar'
    );

    return Medicion->new(
        $self->{valor}*$k,
        $self->{unidad},
        $self->{incertidumbre}*abs($k)
    );

}


sub dividir {

    my ($self,$k,$invertido)=@_;

    die "Orden de operandos no soportado\n"
        if $invertido;

    comprobar_real(
        $k,
        'Escalar'
    );

    die "Division por cero\n"
        if $k==0;

    return Medicion->new(
```

```
        $self->{valor}/$k,
        $self->{unidad},
        $self->{incertidumbre}/abs($k)
    );

}


sub comparar {

    my ($a,$b)=@_;

    my $a2=$a->convertir('m');
    my $b2=$b->convertir('m');

    return -1
        if $a2->{valor}<$b2->{valor};

    return 1
        if $a2->{valor}>$b2->{valor};

    return 0;

}


sub texto {

    my ($self)=@_;

    return sprintf(
        "%.12g %s +/- %.12g %s",
        $self->{valor},
        $self->{unidad},
        $self->{incertidumbre},
        $self->{unidad}
    );

}


package main;


my $m1=Medicion->new(
    2.05,
    'm',
    0.02
);

my $m2=Medicion->new(
```

```
    30,
    'cm',
    1
);


my $suma=$m1+$m2;
my $resta=$m1-$m2;

my $doble=$m1*2;
my $mitad=$m1/2;

my $m1_cm=
    $m1->convertir('cm');


print $suma,"\n";
print $resta,"\n";
print $doble,"\n";
print $mitad,"\n";
print $m1_cm,"\n";

print(
    $m1->comparar($m2),
    "\n"
);
```

JavaScript. En la implementación JavaScript se conserva exactamente la misma semántica mediante una clase y métodos nombrados. De este modo no se cambia el problema matemático para intentar imitar una sintaxis perteneciente a otro lenguaje.

```
class Medicion{

    constructor(
        valor,
        unidad,
        incertidumbre
    ){

        let validoValor=
            typeof valor==="number"
            && Number.isFinite(valor)

        let validoIncertidumbre=
            typeof incertidumbre==="number"
            && Number.isFinite(
                incertidumbre
            )
```

```
        if(!validoValor){

            throw new TypeError(
                "Valor no valido"
            )

        }

        if(!validoIncertidumbre){

            throw new TypeError(
                "Incertidumbre no valida"
            )

        }

        if(
            unidad!=="m"
            && unidad!=="cm"
        ){

            throw new RangeError(
                "Unidad no valida"
            )

        }

        if(incertidumbre<0){

            throw new RangeError(
                "La incertidumbre no puede ser negativa"
            )

        }

        this.valor=valor
        this.unidad=unidad
        this.incertidumbre=incertidumbre

    }


    convertir(unidad){

        if(
            unidad!=="m"
            && unidad!=="cm"
        ){

            throw new RangeError(
                "Unidad no valida"
```

```
            )

        }

        if(this.unidad===unidad){

            return new Medicion(
                this.valor,
                this.unidad,
                this.incertidumbre
            )

        }

        if(
            this.unidad==="m"
            && unidad==="cm"
        ){

            return new Medicion(
                100*this.valor,
                "cm",
                100*this.incertidumbre
            )

        }

        return new Medicion(
            this.valor/100,
            "m",
            this.incertidumbre/100
        )

    }


    sumar(otra){

        if(!(otra instanceof Medicion)){

            throw new TypeError(
                "Se esperaba una Medicion"
            )

        }

        let otra2=
            otra.convertir(this.unidad)

        let u=Math.sqrt(
            this.incertidumbre**2
```

```
            +otra2.incertidumbre**2
        )

        return new Medicion(
            this.valor+otra2.valor,
            this.unidad,
            u
        )

    }


    restar(otra){

        if(!(otra instanceof Medicion)){

            throw new TypeError(
                "Se esperaba una Medicion"
            )

        }

        let otra2=
            otra.convertir(this.unidad)

        let u=Math.sqrt(
            this.incertidumbre**2
            +otra2.incertidumbre**2
        )

        return new Medicion(
            this.valor-otra2.valor,
            this.unidad,
            u
        )

    }


    multiplicar(k){

        if(
            typeof k!=="number"
            || !Number.isFinite(k)
        ){

            throw new TypeError(
                "Escalar no valido"
            )

        }
```

```
        return new Medicion(
            this.valor*k,
            this.unidad,
            this.incertidumbre*Math.abs(k)
        )

    }


    dividir(k){

        if(
            typeof k!=="number"
            || !Number.isFinite(k)
        ){

            throw new TypeError(
                "Escalar no valido"
            )

        }

        if(k===0){

            throw new RangeError(
                "Division por cero"
            )

        }

        return new Medicion(
            this.valor/k,
            this.unidad,
            this.incertidumbre/Math.abs(k)
        )

    }


    comparar(otra){

        let a=this.convertir("m")
        let b=otra.convertir("m")

        if(a.valor<b.valor){
            return -1
        }

        if(a.valor>b.valor){
            return 1
```

```
            }

            return 0

        }


        toString(){

            return (
                this.valor
                +" "
                +this.unidad
                +" +/- "
                +this.incertidumbre
                +" "
                +this.unidad
            )

        }

    }


    function convertirMedicion(m,unidad){

        return m.convertir(unidad)

    }


    function sumarMediciones(a,b){

        return a.sumar(b)

    }


    function restarMediciones(a,b){

        return a.restar(b)

    }


    function multiplicarEscalar(m,k){

        return m.multiplicar(k)

    }
```

```
function dividirEscalar(m,k){

    return m.dividir(k)

}


function compararMediciones(a,b){

    return a.comparar(b)

}


function textoMedicion(m){

    return m.toString()

}


let m1=new Medicion(
    2.05,
    "m",
    0.02
)

let m2=new Medicion(
    30,
    "cm",
    1
)


let suma=
    sumarMediciones(m1,m2)

let resta=
    restarMediciones(m1,m2)

let doble=
    multiplicarEscalar(m1,2)

let mitad=
    dividirEscalar(m1,2)

let m1Cm=
    convertirMedicion(
        m1,
        "cm"
```

```
    )


console.log(
    textoMedicion(suma)
)

console.log(
    textoMedicion(resta)
)

console.log(
    textoMedicion(doble)
)

console.log(
    textoMedicion(mitad)
)

console.log(
    textoMedicion(m1Cm)
)

console.log(
    compararMediciones(
        m1,
        m2
    )
)
```

29. **Ejercicio:** Analizador de expresiones

    Construya un pequeño intérprete de expresiones aritméticas compuesto por dos etapas claramente separadas:

    a) un analizador léxico o *lexer*;
    b) un analizador sintáctico o *parser* que, además de comprobar la sintaxis, calcule el valor de la expresión.

    El intérprete deberá aceptar exclusivamente:

    $$+, \quad -, \quad *, \quad /, \quad (, \quad),$$

    números enteros no negativos y números decimales no negativos.

    Los signos `+` y `-` serán únicamente operadores binarios. Por lo tanto, en este ejercicio no se aceptarán expresiones como

    ```
    -3
    +4
    3 * -2
    ```

El signo de un resultado podrá ser negativo como consecuencia de una operación, por ejemplo

```
3 - 5
```

pero los literales numéricos del lenguaje no incluirán signo.

Formato de los números

Se admitirán:

```
0
15
3.14
100.25
```

pero no:

```
.5
3.
1e3
```

Por lo tanto, un número tendrá conceptualmente la forma

```
DIGITOS
```

o

```
DIGITOS "." DIGITOS
```

donde cada grupo `DIGITOS` contiene al menos un dígito.

Tokens

El analizador léxico deberá producir exclusivamente los siguientes tipos de tokens:

```
NUM(valor)
MAS
MENOS
POR
DIV
PI
PD
FIN
```

donde:

- `PI` representa `(`;
- `PD` representa `)`;

- `FIN` marca el final de la entrada.

Los espacios, tabuladores y saltos de línea entre tokens deberán ignorarse.
Por ejemplo,

```
3 + 4 * 2
```

deberá producir conceptualmente

```
NUM(3), MAS, NUM(4), POR, NUM(2), FIN
```

Gramática

Todos los lenguajes deberán implementar exactamente la siguiente gramática:

```
expresion  -> termino { ("+" | "-") termino }

termino    -> factor { ("*" | "/") factor }

factor     -> NUM
           | "(" expresion ")"
```

Esta gramática establece que:

- la multiplicación y la división tienen mayor prioridad que la suma y la resta;
- operadores con la misma prioridad se evalúan de izquierda a derecha;
- los paréntesis modifican el orden de evaluación.

Por ejemplo,

$$3 + 4 \cdot 2 = 11,$$

mientras que

$$(3 + 4) \cdot 2 = 14.$$

Además,

$$8 - 3 - 2 = (8 - 3) - 2 = 3$$

y

$$18/3/2 = (18/3)/2 = 3.$$

Implemente las siguientes operaciones conceptuales:

```
tokenizar(texto) -> tokens

interpretar(texto) -> valor
```

El parser deberá construirse a partir de tres funciones que correspondan directamente con la gramática:

```
analizar_expresion
analizar_termino
analizar_factor
```

No utilice una función de evaluación automática de cadenas para implementar el parser.

Casos válidos

Compruebe exactamente las siguientes expresiones:

expresión	resultado
`3 + 4 * 2`	11
`(3 + 4) * 2`	14
`10 / (2 + 3)`	2
`8 - 3 - 2`	3
`18 / 3 / 2`	3
`3.5 * 2`	7
`2 * (3 + 4 * 5)`	46
`3 - 5`	-2

Utilice una tolerancia de 10^{-12} para comparar resultados numéricos.

Casos de error

El intérprete deberá rechazar, como mínimo:

Entrada	Motivo
`3 + @`	carácter desconocido
`(3 + 4`	falta paréntesis derecho
`3 + * 4`	dos operadores consecutivos
`3 / (2 - 2)`	división por cero
`3 4`	falta un operador
`3 + 4)`	paréntesis derecho inesperado
`3.`	número decimal mal formado
cadena vacía	expresión incompleta

El mensaje concreto puede variar entre lenguajes, pero todas esas entradas deberán ser rechazadas.

Finalmente:

a) Muestre los tokens producidos para

```
3 + 4 * 2
```

b) Explique por qué el lexer no necesita conocer la precedencia de los operadores.

c) Explique por qué el parser sí necesita distinguir `expresion`, `termino` y `factor`.

d) Explique cómo la estructura de la gramática determina la precedencia.

e) Explique cómo los ciclos utilizados en `analizar_expresion` y `analizar_termino` producen asociatividad de izquierda a derecha.

f) Explique por qué

```
3 + * 4
```

puede contener tokens individualmente válidos y, sin embargo, ser una expresión sintácticamente inválida.

g) Explique la diferencia entre un error léxico, un error sintáctico y un error producido durante la evaluación, como una división por cero.

Scilab. En Scilab se representa la secuencia de tokens mediante una `list` de estructuras. La posición actual del parser se pasa explícitamente entre las funciones.

```
function r=es_digito(c)

    if c=="" then
        r=%f;
        return
    end

    codigo=ascii(c);

    r=(codigo>=48) & (codigo<=57);

endfunction


function r=es_blanco(c)

    codigo=ascii(c);

    r=(codigo==32) | ..
      (codigo==9)  | ..
      (codigo==10) | ..
      (codigo==13);

endfunction


function t=crear_token(tipo,valor)

    t=struct(
        "tipo",tipo,
        "valor",valor
```

```
    );

endfunction


function tokens=tokenizar(texto)

    tokens=list();

    i=1;
    n=length(texto);

    while i<=n

        c=part(texto,i);

        if es_blanco(c) then

            i=i+1;
            continue

        end


        if es_digito(c) then

            numero="";

            while i<=n

                c=part(texto,i);

                if ~es_digito(c) then
                    break
                end

                numero=numero+c;
                i=i+1;

            end


            if i<=n then

                c=part(texto,i);

                if c=="." then

                    numero=numero+".";
                    i=i+1;
```

```
                    if i>n then
                        error(
                            "Numero decimal mal formado"
                        )
                    end

                    c=part(texto,i);

                    if ~es_digito(c) then
                        error(
                            "Numero decimal mal formado"
                        )
                    end

                    while i<=n

                        c=part(texto,i);

                        if ~es_digito(c) then
                            break
                        end

                        numero=numero+c;
                        i=i+1;

                    end

                end

            end


            tokens($+1)=crear_token(
                "NUM",
                strtod(numero)
            );

            continue

        end


        if c=="+" then

            tokens($+1)=crear_token(
                "MAS",
                0
            );

        elseif c=="-" then
```

```
            tokens($+1)=crear_token(
                "MENOS",
                0
            );

        elseif c=="*" then

            tokens($+1)=crear_token(
                "POR",
                0
            );

        elseif c=="/" then

            tokens($+1)=crear_token(
                "DIV",
                0
            );

        elseif c=="(" then

            tokens($+1)=crear_token(
                "PI",
                0
            );

        elseif c==")" then

            tokens($+1)=crear_token(
                "PD",
                0
            );

        elseif c=="." then

            error(
                "Numero decimal mal formado"
            );

        else

            error(
                "Caracter desconocido: "+c
            );

        end

        i=i+1;

    end
```

```
    tokens($+1)=crear_token(
        "FIN",
        0
    );

endfunction


function [valor,pos]=analizar_factor(tokens,pos)

    token=tokens(pos);

    if token.tipo=="NUM" then

        valor=token.valor;
        pos=pos+1;
        return

    end


    if token.tipo=="PI" then

        pos=pos+1;

        [valor,pos]= ..
            analizar_expresion(
                tokens,
                pos
            );

        if tokens(pos).tipo<>"PD" then
            error(
                "Falta parentesis derecho"
            )
        end

        pos=pos+1;
        return

    end


    error(
        "Se esperaba un numero o parentesis izquierdo"
    );

endfunction
```

```
function [valor,pos]=analizar_termino(tokens,pos)

    [valor,pos]= ..
        analizar_factor(
            tokens,
            pos
        );


    while %t

        tipo=tokens(pos).tipo;

        if tipo<>"POR" & ..
           tipo<>"DIV" then

            break

        end


        operador=tipo;
        pos=pos+1;

        [derecha,pos]= ..
            analizar_factor(
                tokens,
                pos
            );


        if operador=="POR" then

            valor=valor*derecha;

        else

            if derecha==0 then
                error(
                    "Division por cero"
                )
            end

            valor=valor/derecha;

        end

    end

endfunction
```

```
function [valor,pos]=analizar_expresion(tokens,pos)

    [valor,pos]= ..
        analizar_termino(
            tokens,
            pos
        );


    while %t

        tipo=tokens(pos).tipo;

        if tipo<>"MAS" & ..
           tipo<>"MENOS" then

            break

        end


        operador=tipo;
        pos=pos+1;

        [derecha,pos]= ..
            analizar_termino(
                tokens,
                pos
            );


        if operador=="MAS" then

            valor=valor+derecha;

        else

            valor=valor-derecha;

        end

    end

endfunction


function valor=interpretar(texto)

    tokens=tokenizar(texto);
```

```
    [valor,pos]= ..
        analizar_expresion(
            tokens,
            1
        );

    if tokens(pos).tipo<>"FIN" then
        error(
            "Tokens inesperados despues de la expresion"
        )
    end

endfunction


function comprobar(texto,esperado)

    obtenido=interpretar(texto);

    if abs(obtenido-esperado)>1d-12 then

        error(
            "Resultado incorrecto para: "+texto
        )

    end

    mprintf(
        "OK: %s = %.12g\n",
        texto,
        obtenido
    );

endfunction


function comprobar_error(texto)

    hubo_error=%f;

    try

        interpretar(texto);

    catch

        hubo_error=%t;

    end
```

```
    if ~hubo_error then

        error(
            "Se esperaba error para: "+texto
        )

    end

    mprintf(
        "OK error: %s\n",
        texto
    );

endfunction


comprobar(
    "3 + 4 * 2",
    11
);

comprobar(
    "(3 + 4) * 2",
    14
);

comprobar(
    "10 / (2 + 3)",
    2
);

comprobar(
    "8 - 3 - 2",
    3
);

comprobar(
    "18 / 3 / 2",
    3
);

comprobar(
    "3.5 * 2",
    7
);

comprobar(
    "2 * (3 + 4 * 5)",
    46
);
```

```
comprobar(
    "3 - 5",
    -2
);


comprobar_error(
    "3 + @"
);

comprobar_error(
    "(3 + 4"
);

comprobar_error(
    "3 + * 4"
);

comprobar_error(
    "3 / (2 - 2)"
);

comprobar_error(
    "3 4"
);

comprobar_error(
    "3 + 4)"
);

comprobar_error(
    "3."
);

comprobar_error(
    ""
);
```

Octave. En Octave se utiliza un arreglo de celdas cuyos elementos son estructuras que representan tokens. La posición del parser también se pasa explícitamente entre las funciones.

```
function r=es_digito(c)

  r=(c>="0" && c<="9");

endfunction


function t=crear_token(tipo,valor)
```

```
  t=struct(
    "tipo",tipo,
    "valor",valor
  );

endfunction


function tokens=tokenizar(texto)

  tokens={};

  i=1;
  n=length(texto);

  while i<=n

    c=texto(i);

    if isspace(c)

      i=i+1;
      continue

    endif


    if es_digito(c)

      numero="";

      while i<=n

        c=texto(i);

        if !es_digito(c)
          break
        endif

        numero=[numero c];
        i=i+1;

      endwhile


      if i<=n && texto(i)=="."

        numero=[numero "."];

        i=i+1;
```

```
        if i>n || !es_digito(texto(i))

          error(
            "Numero decimal mal formado"
          )

        endif


        while i<=n

          c=texto(i);

          if !es_digito(c)
            break
          endif

          numero=[numero c];
          i=i+1;

        endwhile

      endif


      tokens{end+1}=crear_token(
        "NUM",
        str2double(numero)
      );

      continue

    endif


    switch c

      case "+"

        tokens{end+1}= ...
          crear_token("MAS",0);

      case "-"

        tokens{end+1}= ...
          crear_token("MENOS",0);

      case "*"

        tokens{end+1}= ...
```

```
            crear_token("POR",0);

        case "/"

          tokens{end+1}= ...
            crear_token("DIV",0);

        case "("

          tokens{end+1}= ...
            crear_token("PI",0);

        case ")"

          tokens{end+1}= ...
            crear_token("PD",0);

        case "."

          error(
            "Numero decimal mal formado"
          )

        otherwise

          error(
            ["Caracter desconocido: " c]
          )

      endswitch

      i=i+1;

    endwhile


    tokens{end+1}= ...
      crear_token("FIN",0);

endfunction


function [valor,pos]=analizar_factor(tokens,pos)

    token=tokens{pos};

    if strcmp(token.tipo,"NUM")

      valor=token.valor;
      pos=pos+1;
      return
```

```
  endif


  if strcmp(token.tipo,"PI")

    pos=pos+1;

    [valor,pos]= ...
      analizar_expresion(
        tokens,
        pos
      );

    if !strcmp(
        tokens{pos}.tipo,
        "PD"
    )

      error(
        "Falta parentesis derecho"
      )

    endif

    pos=pos+1;
    return

  endif


  error(
    "Se esperaba un numero o parentesis izquierdo"
  );

endfunction


function [valor,pos]=analizar_termino(tokens,pos)

  [valor,pos]= ...
    analizar_factor(
      tokens,
      pos
    );


  while true

    tipo=tokens{pos}.tipo;
```

```
    if !strcmp(tipo,"POR") ...
       && !strcmp(tipo,"DIV")

      break

    endif


    operador=tipo;
    pos=pos+1;

    [derecha,pos]= ...
      analizar_factor(
        tokens,
        pos
      );


    if strcmp(operador,"POR")

      valor=valor*derecha;

    else

      if derecha==0
        error("Division por cero")
      endif

      valor=valor/derecha;

    endif

  endwhile

endfunction


function [valor,pos]=analizar_expresion(tokens,pos)

  [valor,pos]= ...
    analizar_termino(
      tokens,
      pos
    );


  while true

    tipo=tokens{pos}.tipo;

    if !strcmp(tipo,"MAS") ...
```

```
        && !strcmp(tipo,"MENOS")

      break

    endif


    operador=tipo;
    pos=pos+1;

    [derecha,pos]= ...
      analizar_termino(
        tokens,
        pos
      );


    if strcmp(operador,"MAS")

      valor=valor+derecha;

    else

      valor=valor-derecha;

    endif

  endwhile

endfunction


function valor=interpretar(texto)

  tokens=tokenizar(texto);

  [valor,pos]= ...
    analizar_expresion(
      tokens,
      1
    );

  if !strcmp(
      tokens{pos}.tipo,
      "FIN"
  )

    error(
      "Tokens inesperados despues de la expresion"
    )
```

```
  endif

endfunction


function comprobar(texto,esperado)

  obtenido=interpretar(texto);

  if abs(obtenido-esperado)>1e-12

    error(
      ["Resultado incorrecto para: " texto]
    )

  endif

  printf(
    "OK: %s = %.12g\n",
    texto,
    obtenido
  );

endfunction


function comprobar_error(texto)

  hubo_error=false;

  try

    interpretar(texto);

  catch

    hubo_error=true;

  end_try_catch


  if !hubo_error

    error(
      ["Se esperaba error para: " texto]
    )

  endif

  printf(
    "OK error: %s\n",
```

```
    texto
  );

endfunction


comprobar(
  "3 + 4 * 2",
  11
);

comprobar(
  "(3 + 4) * 2",
  14
);

comprobar(
  "10 / (2 + 3)",
  2
);

comprobar(
  "8 - 3 - 2",
  3
);

comprobar(
  "18 / 3 / 2",
  3
);

comprobar(
  "3.5 * 2",
  7
);

comprobar(
  "2 * (3 + 4 * 5)",
  46
);

comprobar(
  "3 - 5",
  -2
);


comprobar_error("3 + @");
comprobar_error("(3 + 4");
comprobar_error("3 + * 4");
comprobar_error("3 / (2 - 2)");
```

```
comprobar_error("3 4");
comprobar_error("3 + 4)");
comprobar_error("3.");
comprobar_error("");
```

Python. En Python los tokens se representan mediante diccionarios. Las tres funciones del parser implementan directamente las tres reglas de la gramática.

```
def crear_token(tipo,valor=None):

    return {
        "tipo":tipo,
        "valor":valor
    }


def tokenizar(texto):

    tokens=[]

    i=0
    n=len(texto)

    while i<n:

        c=texto[i]

        if c.isspace():

            i+=1
            continue


        if c.isdigit():

            numero=""

            while (
                i<n
                and texto[i].isdigit()
            ):

                numero+=texto[i]
                i+=1


            if (
                i<n
                and texto[i]=="."
            ):
```

```
                numero+="."
                i+=1

                if (
                    i>=n
                    or not texto[i].isdigit()
                ):

                    raise ValueError(
                        "Numero decimal mal formado"
                    )


                while (
                    i<n
                    and texto[i].isdigit()
                ):

                    numero+=texto[i]
                    i+=1


            tokens.append(
                crear_token(
                    "NUM",
                    float(numero)
                )
            )

            continue


        tipos={
            "+":"MAS",
            "-":"MENOS",
            "*":"POR",
            "/":"DIV",
            "(":"PI",
            ")":"PD"
        }


        if c in tipos:

            tokens.append(
                crear_token(
                    tipos[c]
                )
            )
```

```
            i+=1
            continue


        if c==".":

            raise ValueError(
                "Numero decimal mal formado"
            )


        raise ValueError(
            "Caracter desconocido: "+c
        )


    tokens.append(
        crear_token("FIN")
    )

    return tokens


def analizar_factor(tokens,pos):

    token=tokens[pos]


    if token["tipo"]=="NUM":

        return token["valor"],pos+1


    if token["tipo"]=="PI":

        valor,pos=analizar_expresion(
            tokens,
            pos+1
        )

        if tokens[pos]["tipo"]!="PD":

            raise SyntaxError(
                "Falta parentesis derecho"
            )

        return valor,pos+1


    raise SyntaxError(
        "Se esperaba un numero o parentesis izquierdo"
```

```
    )


def analizar_termino(tokens,pos):

    valor,pos=analizar_factor(
        tokens,
        pos
    )


    while tokens[pos]["tipo"] in (
        "POR",
        "DIV"
    ):

        operador=tokens[pos]["tipo"]

        derecha,pos=analizar_factor(
            tokens,
            pos+1
        )


        if operador=="POR":

            valor=valor*derecha

        else:

            if derecha==0:

                raise ZeroDivisionError(
                    "Division por cero"
                )

            valor=valor/derecha


    return valor,pos


def analizar_expresion(tokens,pos):

    valor,pos=analizar_termino(
        tokens,
        pos
    )


    while tokens[pos]["tipo"] in (
```

```
            "MAS",
            "MENOS"
        ):

            operador=tokens[pos]["tipo"]

            derecha,pos=analizar_termino(
                tokens,
                pos+1
            )


            if operador=="MAS":

                valor=valor+derecha

            else:

                valor=valor-derecha


        return valor,pos


    def interpretar(texto):

        tokens=tokenizar(texto)

        valor,pos=analizar_expresion(
            tokens,
            0
        )


        if tokens[pos]["tipo"]!="FIN":

            raise SyntaxError(
                "Tokens inesperados despues de la expresion"
            )


        return valor


    def comprobar(texto,esperado):

        obtenido=interpretar(texto)

        if abs(obtenido-esperado)>1e-12:

            raise AssertionError(
```

```
            "Resultado incorrecto para: "
            +texto
        )

    print(
        "OK:",
        texto,
        "=",
        obtenido
    )


def comprobar_error(texto):

    try:

        interpretar(texto)

    except (
        ValueError,
        SyntaxError,
        ZeroDivisionError
    ):

        print(
            "OK error:",
            repr(texto)
        )

        return


    raise AssertionError(
        "Se esperaba error para: "
        +repr(texto)
    )


comprobar(
    "3 + 4 * 2",
    11
)

comprobar(
    "(3 + 4) * 2",
    14
)

comprobar(
    "10 / (2 + 3)",
    2
```

```
)

comprobar(
    "8 - 3 - 2",
    3
)

comprobar(
    "18 / 3 / 2",
    3
)

comprobar(
    "3.5 * 2",
    7
)

comprobar(
    "2 * (3 + 4 * 5)",
    46
)

comprobar(
    "3 - 5",
    -2
)


for texto in [
    "3 + @",
    "(3 + 4",
    "3 + * 4",
    "3 / (2 - 2)",
    "3 4",
    "3 + 4)",
    "3.",
    ""
]:

    comprobar_error(texto)
```

Perl. En Perl los tokens se representan mediante referencias a *hashes*. El uso de `eval` en la función de prueba se emplea únicamente para capturar excepciones; no se utiliza para evaluar la expresión aritmética.

```
use strict;
use warnings;


sub crear_token {
```

```
    my ($tipo,$valor)=@_;

    return {
        tipo=>$tipo,
        valor=>$valor
    };

}


sub tokenizar {

    my ($texto)=@_;

    my @tokens;

    my $i=0;
    my $n=length($texto);


    while ($i<$n) {

        my $c=substr(
            $texto,
            $i,
            1
        );


        if ($c=~/\s/) {

            $i++;
            next;

        }


        if ($c=~/[0-9]/) {

            my $numero='';


            while ($i<$n) {

                $c=substr(
                    $texto,
                    $i,
                    1
                );
```

```
                    last
                        unless $c=~/[0-9]/;

                    $numero.=$c;
                    $i++;

                }


                if (
                    $i<$n
                    &&
                    substr(
                        $texto,
                        $i,
                        1
                    ) eq '.'
                ) {

                    $numero.='.';
                    $i++;


                    if (
                        $i>=$n
                        ||
                        substr(
                            $texto,
                            $i,
                            1
                        )!~/[0-9]/
                    ) {

                        die(
                            "Numero decimal ",
                            "mal formado\n"
                        );

                    }


                    while ($i<$n) {

                        $c=substr(
                            $texto,
                            $i,
                            1
                        );

                        last
                            unless $c=~/[0-9]/;
```

```
                $numero.=$c;
                $i++;

            }

        }


        push(
            @tokens,
            crear_token(
                'NUM',
                0+$numero
            )
        );

        next;

    }


    my %tipos=(
        '+'=>'MAS',
        '-'=>'MENOS',
        '*'=>'POR',
        '/'=>'DIV',
        '('=>'PI',
        ')'=>'PD'
    );


    if (exists $tipos{$c}) {

        push(
            @tokens,
            crear_token(
                $tipos{$c},
                undef
            )
        );

        $i++;
        next;

    }


    if ($c eq '.') {

        die(
```

```
                    "Numero decimal mal formado\n"
                );

            }


            die(
                "Caracter desconocido: ",
                $c,
                "\n"
            );

        }


        push(
            @tokens,
            crear_token(
                'FIN',
                undef
            )
        );

        return \@tokens;

    }


    sub analizar_factor {

        my ($tokens,$pos)=@_;

        my $token=$tokens->[$pos];


        if ($token->{tipo} eq 'NUM') {

            return (
                $token->{valor},
                $pos+1
            );

        }


        if ($token->{tipo} eq 'PI') {

            my ($valor,$nuevo_pos)=
                analizar_expresion(
                    $tokens,
                    $pos+1
```

```
            );


        die(
            "Falta parentesis derecho\n"
        )
            unless
                $tokens->[$nuevo_pos]{tipo}
                eq 'PD';


        return (
            $valor,
            $nuevo_pos+1
        );

    }


    die(
        "Se esperaba un numero ",
        "o parentesis izquierdo\n"
    );

}


sub analizar_termino {

    my ($tokens,$pos)=@_;

    my ($valor,$nuevo_pos)=
        analizar_factor(
            $tokens,
            $pos
        );

    $pos=$nuevo_pos;


    while (
        $tokens->[$pos]{tipo} eq 'POR'
        ||
        $tokens->[$pos]{tipo} eq 'DIV'
    ) {

        my $operador=
            $tokens->[$pos]{tipo};


        my ($derecha,$siguiente)=
```

```
            analizar_factor(
                $tokens,
                $pos+1
            );

        $pos=$siguiente;


        if ($operador eq 'POR') {

            $valor=
                $valor*$derecha;

        }
        else {

            die "Division por cero\n"
                if $derecha==0;

            $valor=
                $valor/$derecha;

        }

    }


    return ($valor,$pos);

}


sub analizar_expresion {

    my ($tokens,$pos)=@_;

    my ($valor,$nuevo_pos)=
        analizar_termino(
            $tokens,
            $pos
        );

    $pos=$nuevo_pos;


    while (
        $tokens->[$pos]{tipo} eq 'MAS'
        ||
        $tokens->[$pos]{tipo} eq 'MENOS'
    ) {
```

```
        my $operador=
            $tokens->[$pos]{tipo};


        my ($derecha,$siguiente)=
            analizar_termino(
                $tokens,
                $pos+1
            );

        $pos=$siguiente;


        if ($operador eq 'MAS') {

            $valor=
                $valor+$derecha;

        }
        else {

            $valor=
                $valor-$derecha;

        }

    }


    return ($valor,$pos);

}


sub interpretar {

    my ($texto)=@_;

    my $tokens=
        tokenizar($texto);


    my ($valor,$pos)=
        analizar_expresion(
            $tokens,
            0
        );


    die(
        "Tokens inesperados ",
```

```
            "despues de la expresion\n"
        )
            unless
                $tokens->[$pos]{tipo}
                eq 'FIN';


        return $valor;

    }


    sub comprobar {

        my ($texto,$esperado)=@_;

        my $obtenido=
            interpretar($texto);


        die(
            "Resultado incorrecto para: ",
            $texto,
            "\n"
        )
            if abs(
                $obtenido-$esperado
            )>1e-12;


        print(
            "OK: ",
            $texto,
            " = ",
            $obtenido,
            "\n"
        );

    }


    sub comprobar_error {

        my ($texto)=@_;

        my $hubo_error=0;


        eval {

            interpretar($texto);
```

```
    };


    $hubo_error=1
        if $@ ne '';


    die(
        "Se esperaba error para: ",
        $texto,
        "\n"
    )
        unless $hubo_error;


    print(
        "OK error: ",
        $texto,
        "\n"
    );

}


comprobar(
    '3 + 4 * 2',
    11
);

comprobar(
    '(3 + 4) * 2',
    14
);

comprobar(
    '10 / (2 + 3)',
    2
);

comprobar(
    '8 - 3 - 2',
    3
);

comprobar(
    '18 / 3 / 2',
    3
);

comprobar(
```

```
    '3.5 * 2',
    7
);

comprobar(
    '2 * (3 + 4 * 5)',
    46
);

comprobar(
    '3 - 5',
    -2
);


for my $texto (
    '3 + @',
    '(3 + 4',
    '3 + * 4',
    '3 / (2 - 2)',
    '3 4',
    '3 + 4)',
    '3.',
    ''
) {

    comprobar_error($texto);

}
```

JavaScript. En JavaScript los tokens se representan mediante objetos contenidos en un arreglo. Las funciones del parser reciben la posición actual y devuelven también la nueva posición.

```
function crearToken(tipo,valor=null){

    return {
        tipo:tipo,
        valor:valor
    }

}


function esDigito(c){

    return (
        c>="0"
        && c<="9"
    )
```

```
}


function tokenizar(texto){

    let tokens=[]

    let i=0


    while(i<texto.length){

        let c=texto[i]


        if(/\s/.test(c)){

            i++
            continue

        }


        if(esDigito(c)){

            let numero=""


            while(
                i<texto.length
                && esDigito(texto[i])
            ){

                numero+=texto[i]
                i++

            }


            if(
                i<texto.length
                && texto[i]==="."
            ){

                numero+="."
                i++


                if(
                    i>=texto.length
```

```
                    || !esDigito(texto[i])
                ){

                    throw new SyntaxError(
                        "Numero decimal mal formado"
                    )

                }


                while(
                    i<texto.length
                    && esDigito(texto[i])
                ){

                    numero+=texto[i]
                    i++

                }

            }


            tokens.push(
                crearToken(
                    "NUM",
                    Number(numero)
                )
            )

            continue

        }


        let tipos={
            "+":"MAS",
            "-":"MENOS",
            "*":"POR",
            "/":"DIV",
            "(":"PI",
            ")":"PD"
        }


        if(
            Object.prototype.hasOwnProperty.call(
                tipos,
                c
            )
        ){
```

```
            tokens.push(
                crearToken(
                    tipos[c]
                )
            )

            i++
            continue

        }


        if(c==="."){

            throw new SyntaxError(
                "Numero decimal mal formado"
            )

        }


        throw new SyntaxError(
            "Caracter desconocido: "+c
        )

    }


    tokens.push(
        crearToken("FIN")
    )

    return tokens

}


function analizarFactor(tokens,pos){

    let token=tokens[pos]


    if(token.tipo==="NUM"){

        return [
            token.valor,
            pos+1
        ]

    }
```

```
    if(token.tipo==="PI"){

        let resultado=
            analizarExpresion(
                tokens,
                pos+1
            )

        let valor=resultado[0]

        pos=resultado[1]


        if(tokens[pos].tipo!=="PD"){

            throw new SyntaxError(
                "Falta parentesis derecho"
            )

        }


        return [
            valor,
            pos+1
        ]

    }


    throw new SyntaxError(
        "Se esperaba un numero o parentesis izquierdo"
    )

}


function analizarTermino(tokens,pos){

    let resultado=
        analizarFactor(
            tokens,
            pos
        )

    let valor=resultado[0]

    pos=resultado[1]
```

```
    while(
        tokens[pos].tipo==="POR"
        || tokens[pos].tipo==="DIV"
    ){

        let operador=
            tokens[pos].tipo


        resultado=
            analizarFactor(
                tokens,
                pos+1
            )

        let derecha=
            resultado[0]

        pos=resultado[1]


        if(operador==="POR"){

            valor=
                valor*derecha

        }
        else{

            if(derecha===0){

                throw new RangeError(
                    "Division por cero"
                )

            }

            valor=
                valor/derecha

        }

    }


    return [
        valor,
        pos
    ]
```

```
}


function analizarExpresion(tokens,pos){

    let resultado=
        analizarTermino(
            tokens,
            pos
        )

    let valor=resultado[0]

    pos=resultado[1]


    while(
        tokens[pos].tipo==="MAS"
        || tokens[pos].tipo==="MENOS"
    ){

        let operador=
            tokens[pos].tipo


        resultado=
            analizarTermino(
                tokens,
                pos+1
            )

        let derecha=
            resultado[0]

        pos=resultado[1]


        if(operador==="MAS"){

            valor=
                valor+derecha

        }
        else{

            valor=
                valor-derecha

        }

    }
```

```
        return [
            valor,
            pos
        ]

    }


    function interpretar(texto){

        let tokens=
            tokenizar(texto)


        let resultado=
            analizarExpresion(
                tokens,
                0
            )

        let valor=
            resultado[0]

        let pos=
            resultado[1]


        if(tokens[pos].tipo!=="FIN"){

            throw new SyntaxError(
                "Tokens inesperados despues de la expresion"
            )

        }


        return valor

    }


    function comprobar(texto,esperado){

        let obtenido=
            interpretar(texto)


        if(
            Math.abs(
```

```
            obtenido-esperado
        )>1e-12
    ){

        throw new Error(
            "Resultado incorrecto para: "
            +texto
        )

    }


    console.log(
        "OK:",
        texto,
        "=",
        obtenido
    )

}


function comprobarError(texto){

    let huboError=false


    try{

        interpretar(texto)

    }
    catch(error){

        huboError=true

    }


    if(!huboError){

        throw new Error(
            "Se esperaba error para: "
            +texto
        )

    }


    console.log(
        "OK error:",
```

```
            JSON.stringify(texto)
        )

    }


    comprobar(
        "3 + 4 * 2",
        11
    )

    comprobar(
        "(3 + 4) * 2",
        14
    )

    comprobar(
        "10 / (2 + 3)",
        2
    )

    comprobar(
        "8 - 3 - 2",
        3
    )

    comprobar(
        "18 / 3 / 2",
        3
    )

    comprobar(
        "3.5 * 2",
        7
    )

    comprobar(
        "2 * (3 + 4 * 5)",
        46
    )

    comprobar(
        "3 - 5",
        -2
    )


    for(const texto of [
        "3 + @",
        "(3 + 4",
        "3 + * 4",
```

```
      "3 / (2 - 2)",
      "3 4",
      "3 + 4)",
      "3.",
      ""
  ]){

      comprobarError(texto)

  }
```

5.11.8. Módulos, bibliotecas y reutilización

30. **Ejercicio:** Biblioteca en módulos

    Organice varias funciones desarrolladas en los ejercicios anteriores como una pequeña biblioteca reutilizable.

    La biblioteca deberá ofrecer exactamente las siguientes operaciones conceptuales:

    ```
    paridad(n)
    celsius_a_fahrenheit(t)
    fahrenheit_a_celsius(t)
    razon_limitada(x, y, z)
    ```

    Los contratos serán los mismos en los cinco lenguajes.

    Paridad

    ```
    paridad(n)
    ```

    recibe un entero y devuelve:

    ```
    "es par"
    "es impar"
    ```

    según corresponda.

    Una entrada no entera deberá producir un error.

    Conversión de temperatura

    Las funciones

    ```
    celsius_a_fahrenheit(t)
    fahrenheit_a_celsius(t)
    ```

reciben un número real escalar y finito y utilizan

$$F = \frac{9}{5}C + 32$$

y

$$C = (F - 32)\frac{5}{9}.$$

Razón limitada

La función

```
razon_limitada(x, y, z)
```

recibe números reales escalares y finitos, con $z > 0$, calcula

$$q = \frac{\sqrt{x^2 + y^2}}{z}$$

y devuelve

$$\text{mín}(q, 1).$$

La biblioteca deberá organizarse conceptualmente mediante:

```
mi_biblioteca/
    codigo de la biblioteca
    pruebas/
    ejemplos/
    mecanismo de carga o importacion
    README.md
```

La organización concreta deberá respetar los mecanismos habituales del lenguaje utilizado.

El trabajo deberá cumplir los siguientes requisitos:

a) Las funciones deberán estar fuera del programa principal de ejemplo.

b) El programa de ejemplo deberá utilizar las funciones mediante el mecanismo de carga, importación o inclusión correspondiente al lenguaje.

c) Las pruebas deberán poder ejecutarse desde una sesión limpia.

d) Ninguna función deberá depender de variables creadas previamente en una sesión interactiva.

e) La implementación de una función deberá existir en un único lugar. Los programas de prueba y los ejemplos no deberán duplicar su código.

f) El archivo `README.md` deberá indicar:

- qué funciones contiene la biblioteca;

- qué argumentos recibe cada función;
- qué devuelve;
- qué errores puede producir;
- cómo cargar o importar la biblioteca;
- cómo ejecutar las pruebas;
- cómo ejecutar el ejemplo.

g) Ejecute exactamente las siguientes comprobaciones:

$$\text{paridad}(4) = ''\text{ es par}'',$$

$$\text{paridad}(7) = ''\text{ es impar}'',$$

$$\text{celsius_a_fahrenheit}(0) = 32,$$

$$\text{celsius_a_fahrenheit}(100) = 212,$$

$$\text{fahrenheit_a_celsius}(32) = 0,$$

$$\text{fahrenheit_a_celsius}(-40) = -40,$$

$$\text{razon_limitada}(3, 4, 10) = 0{,}5,$$

$$\text{razon_limitada}(3, 4, 2) = 1.$$

h) Compruebe además que se produzca un error con:

```
paridad(2.5)
razon_limitada(3, 4, 0)
```

i) Utilice una tolerancia de 10^{-12} cuando compare resultados reales.

j) Explique qué parte de la organización cumple en cada lenguaje el papel de:

```
biblioteca
carga o importacion
pruebas
ejemplo de uso
documentacion
```

Scilab. En Scilab puede utilizarse un archivo `.sci` para cada grupo de funciones y un archivo `loader.sce` que cargue las definiciones. Los programas y pruebas se guardan en archivos `.sce`.

Una organización posible es:

```
mi_biblioteca/
    funciones/
        paridad.sci
        temperatura.sci
        razon_limitada.sci
    pruebas/
        test_biblioteca.sce
    ejemplos/
        ejemplo_uso.sce
    loader.sce
    README.md
```

Archivo `funciones/paridad.sci`:

```
function resultado=paridad(n)

    valido=type(n)==1 & ..
           size(n,"*")==1 & ..
           isreal(n);

    if valido then
        valido=~isnan(n) & ..
               abs(n)<>%inf & ..
               n==floor(n);
    end

    if ~valido then
        error("n debe ser un entero")
    end

    if modulo(n,2)==0 then
        resultado="es par";
    else
        resultado="es impar";
    end

endfunction
```

Archivo `funciones/temperatura.sci`:

```
function comprobar_real(t)

    valido=type(t)==1 & ..
           size(t,"*")==1 & ..
           isreal(t);

    if valido then
        valido=~isnan(t) & ..
               abs(t)<>%inf;
```

```
    end

    if ~valido then
        error("Se esperaba un escalar real y finito")
    end

endfunction


function f=celsius_a_fahrenheit(c)

    comprobar_real(c);

    f=c*9/5+32;

endfunction


function c=fahrenheit_a_celsius(f)

    comprobar_real(f);

    c=(f-32)*5/9;

endfunction
```

Archivo funciones/razon_limitada.sci:

```
function comprobar_argumento(x,nombre)

    valido=type(x)==1 & ..
           size(x,"*")==1 & ..
           isreal(x);

    if valido then
        valido=~isnan(x) & ..
               abs(x)<>%inf;
    end

    if ~valido then
        error(nombre+" debe ser un escalar real y finito")
    end

endfunction


function resultado=razon_limitada(x,y,z)

    comprobar_argumento(x,"x");
    comprobar_argumento(y,"y");
    comprobar_argumento(z,"z");
```

```
    if z<=0 then
        error("z debe ser estrictamente positivo")
    end

    razon=sqrt(x^2+y^2)/z;

    if razon>1 then
        resultado=1;
    else
        resultado=razon;
    end

endfunction
```

Archivo `loader.sce`:

```
exec(
    "funciones/paridad.sci",
    -1
);

exec(
    "funciones/temperatura.sci",
    -1
);

exec(
    "funciones/razon_limitada.sci",
    -1
);
```

Archivo `pruebas/test_biblioteca.sce`:

```
exec("../loader.sce",-1);

tolerancia=1d-12;


function comprobar(condicion,nombre)

    if ~condicion then
        error("FALLO: "+nombre)
    end

    mprintf(
        "OK: %s\n",
        nombre
    );
```

```
endfunction


comprobar(
    paridad(4)=="es par",
    "paridad 4"
);

comprobar(
    paridad(7)=="es impar",
    "paridad 7"
);

comprobar(
    abs(
        celsius_a_fahrenheit(0)-32
    )<=tolerancia,
    "0 C"
);

comprobar(
    abs(
        celsius_a_fahrenheit(100)-212
    )<=tolerancia,
    "100 C"
);

comprobar(
    abs(
        fahrenheit_a_celsius(32)
    )<=tolerancia,
    "32 F"
);

comprobar(
    abs(
        fahrenheit_a_celsius(-40)+40
    )<=tolerancia,
    "-40 F"
);

comprobar(
    abs(
        razon_limitada(3,4,10)-0.5
    )<=tolerancia,
    "razon 0.5"
);

comprobar(
    razon_limitada(3,4,2)==1,
    "razon limitada"
```

```
);


hubo_error=%f;

try
    paridad(2.5);
catch
    hubo_error=%t;
end

comprobar(
    hubo_error,
    "paridad invalida"
);


hubo_error=%f;

try
    razon_limitada(3,4,0);
catch
    hubo_error=%t;
end

comprobar(
    hubo_error,
    "z igual a cero"
);
```

Archivo `ejemplos/ejemplo_uso.sce`:

```
exec("../loader.sce",-1);

disp(paridad(17))

disp(
    celsius_a_fahrenheit(25)
)

disp(
    fahrenheit_a_celsius(77)
)

disp(
    razon_limitada(3,4,10)
)
```

Octave. En Octave cada función pública puede almacenarse en su propio archivo `.m`. Un archivo de carga puede incorporar el directorio de funciones al camino de

búsqueda.

Una organización posible es:

```
mi_biblioteca/
    funciones/
        paridad.m
        celsius_a_fahrenheit.m
        fahrenheit_a_celsius.m
        razon_limitada.m
    pruebas/
        test_biblioteca.m
    ejemplos/
        ejemplo_uso.m
    loader.m
    README.md
```

Archivo `funciones/paridad.m`:

```
function resultado=paridad(n)

  valido=isnumeric(n) ...
         && isscalar(n) ...
         && isreal(n) ...
         && isfinite(n) ...
         && n==fix(n);

  if !valido
    error("n debe ser un entero")
  endif

  if mod(n,2)==0
    resultado="es par";
  else
    resultado="es impar";
  endif

endfunction
```

Archivo `funciones/celsius_a_fahrenheit.m`:

```
function f=celsius_a_fahrenheit(c)

  valido=isnumeric(c) ...
         && isscalar(c) ...
         && isreal(c) ...
         && isfinite(c);

  if !valido
```

```
    error(
      "Se esperaba un escalar real y finito"
    )
  endif

  f=c*9/5+32;

endfunction
```

Archivo funciones/fahrenheit_a_celsius.m:

```
function c=fahrenheit_a_celsius(f)

  valido=isnumeric(f) ...
         && isscalar(f) ...
         && isreal(f) ...
         && isfinite(f);

  if !valido
    error(
      "Se esperaba un escalar real y finito"
    )
  endif

  c=(f-32)*5/9;

endfunction
```

Archivo funciones/razon_limitada.m:

```
function resultado=razon_limitada(x,y,z)

  valores={x,y,z};

  for k=1:3

    v=valores{k};

    valido=isnumeric(v) ...
           && isscalar(v) ...
           && isreal(v) ...
           && isfinite(v);

    if !valido
      error(
        "Los argumentos deben ser escalares reales y finitos"
      )
    endif

  endfor
```

```
  if z<=0
    error("z debe ser estrictamente positivo")
  endif

  razon=sqrt(x^2+y^2)/z;

  if razon>1
    resultado=1;
  else
    resultado=razon;
  endif

endfunction
```

Archivo `loader.m`:

```
addpath("funciones");
```

Archivo `pruebas/test_biblioteca.m`:

```
run("../loader.m");

tolerancia=1e-12;

assert(
  strcmp(
    paridad(4),
    "es par"
  )
);

assert(
  strcmp(
    paridad(7),
    "es impar"
  )
);

assert(
  abs(
    celsius_a_fahrenheit(0)-32
  )<=tolerancia
);

assert(
  abs(
    celsius_a_fahrenheit(100)-212
  )<=tolerancia
);
```

```
assert(
  abs(
    fahrenheit_a_celsius(32)
  )<=tolerancia
);

assert(
  abs(
    fahrenheit_a_celsius(-40)+40
  )<=tolerancia
);

assert(
  abs(
    razon_limitada(3,4,10)-0.5
  )<=tolerancia
);

assert(
  razon_limitada(3,4,2)==1
);


hubo_error=false;

try
  paridad(2.5);
catch
  hubo_error=true;
end_try_catch

assert(hubo_error);


hubo_error=false;

try
  razon_limitada(3,4,0);
catch
  hubo_error=true;
end_try_catch

assert(hubo_error);


disp("Todas las pruebas pasaron")
```

Archivo `ejemplos/ejemplo_uso.m`:

```
run("../loader.m");
```

```
disp(paridad(17))

disp(
  celsius_a_fahrenheit(25)
)

disp(
  fahrenheit_a_celsius(77)
)

disp(
  razon_limitada(3,4,10)
)
```

Python. En Python se utiliza un paquete. El archivo `__init__.py` define qué nombres públicos ofrece la biblioteca.

Una organización posible es:

```
mi_biblioteca/
    mi_biblioteca/
        __init__.py
        funciones.py
    pruebas/
        test_biblioteca.py
    ejemplos/
        ejemplo_uso.py
    README.md
```

Archivo `mi_biblioteca/funciones.py`:

```
import math


def comprobar_real(x):

    valido=(
        isinstance(x,(int,float))
        and not isinstance(x,bool)
        and math.isfinite(x)
    )

    if not valido:
        raise TypeError(
            "Se esperaba un escalar real y finito"
        )


def paridad(n):
```

```
    if (
        not isinstance(n,int)
        or isinstance(n,bool)
    ):
        raise TypeError(
            "n debe ser un entero"
        )

    if n%2==0:
        return "es par"

    return "es impar"


def celsius_a_fahrenheit(c):

    comprobar_real(c)

    return c*9/5+32


def fahrenheit_a_celsius(f):

    comprobar_real(f)

    return (f-32)*5/9


def razon_limitada(x,y,z):

    comprobar_real(x)
    comprobar_real(y)
    comprobar_real(z)

    if z<=0:
        raise ValueError(
            "z debe ser estrictamente positivo"
        )

    razon=math.sqrt(
        x**2+y**2
    )/z

    if razon>1:
        return 1

    return razon
```

Archivo `mi_biblioteca/__init__.py`:

```
```

```
from .funciones import paridad
from .funciones import celsius_a_fahrenheit
from .funciones import fahrenheit_a_celsius
from .funciones import razon_limitada
```

Archivo pruebas/test_biblioteca.py:

```
import unittest

from mi_biblioteca import (
    paridad,
    celsius_a_fahrenheit,
    fahrenheit_a_celsius,
    razon_limitada
)


class PruebasBiblioteca(unittest.TestCase):

    def test_paridad(self):

        self.assertEqual(
            paridad(4),
            "es par"
        )

        self.assertEqual(
            paridad(7),
            "es impar"
        )


    def test_temperatura(self):

        self.assertAlmostEqual(
            celsius_a_fahrenheit(0),
            32,
            delta=1e-12
        )

        self.assertAlmostEqual(
            celsius_a_fahrenheit(100),
            212,
            delta=1e-12
        )

        self.assertAlmostEqual(
            fahrenheit_a_celsius(32),
            0,
            delta=1e-12
        )
```

```
        self.assertAlmostEqual(
            fahrenheit_a_celsius(-40),
            -40,
            delta=1e-12
        )


    def test_razon(self):

        self.assertAlmostEqual(
            razon_limitada(3,4,10),
            0.5,
            delta=1e-12
        )

        self.assertEqual(
            razon_limitada(3,4,2),
            1
        )


    def test_errores(self):

        with self.assertRaises(
            (TypeError,ValueError)
        ):
            paridad(2.5)

        with self.assertRaises(
            (TypeError,ValueError)
        ):
            razon_limitada(3,4,0)


if __name__=="__main__":
    unittest.main()
```

Archivo `ejemplos/ejemplo_uso.py`:

```
from mi_biblioteca import (
    paridad,
    celsius_a_fahrenheit,
    fahrenheit_a_celsius,
    razon_limitada
)


print(paridad(17))

print(
```

```
    celsius_a_fahrenheit(25)
)

print(
    fahrenheit_a_celsius(77)
)

print(
    razon_limitada(3,4,10)
)
```

Perl. En Perl puede agruparse la interfaz pública en un módulo `.pm`. El módulo `Exporter` permite seleccionar las funciones que podrán importarse en los programas clientes.

Una organización posible es:

```
mi_biblioteca/
    lib/
        MiBiblioteca.pm
    t/
        biblioteca.t
    ejemplos/
        ejemplo_uso.pl
    README.md
```

Archivo `lib/MiBiblioteca.pm`:

```
package MiBiblioteca;

use strict;
use warnings;

use Exporter qw(import);
use Scalar::Util qw(looks_like_number);
use POSIX qw(isfinite);


our @EXPORT_OK=qw(
    paridad
    celsius_a_fahrenheit
    fahrenheit_a_celsius
    razon_limitada
);


sub comprobar_real {

    my ($x)=@_;
```

```
    my $valido=
        defined($x)
        && looks_like_number($x)
        && isfinite($x);

    die(
        "Se esperaba un escalar real y finito\n"
    )
        unless $valido;

}


sub paridad {

    my ($n)=@_;

    comprobar_real($n);

    die "n debe ser un entero\n"
        unless int($n)==$n;

    return 'es par'
        if $n%2==0;

    return 'es impar';

}


sub celsius_a_fahrenheit {

    my ($c)=@_;

    comprobar_real($c);

    return $c*9/5+32;

}


sub fahrenheit_a_celsius {

    my ($f)=@_;

    comprobar_real($f);

    return ($f-32)*5/9;

}
```

```
sub razon_limitada {

    my ($x,$y,$z)=@_;

    comprobar_real($x);
    comprobar_real($y);
    comprobar_real($z);

    die(
        "z debe ser estrictamente positivo\n"
    )
        if $z<=0;

    my $razon=
        sqrt($x**2+$y**2)/$z;

    return 1
        if $razon>1;

    return $razon;

}


1;
```

Archivo `t/biblioteca.t`:

```
use strict;
use warnings;

use FindBin;
use lib "$FindBin::Bin/../lib";

use Test::More;

use MiBiblioteca qw(
    paridad
    celsius_a_fahrenheit
    fahrenheit_a_celsius
    razon_limitada
);


is(
    paridad(4),
    'es par',
    'paridad 4'
);
```

```
is(
    paridad(7),
    'es impar',
    'paridad 7'
);

cmp_ok(
    abs(
        celsius_a_fahrenheit(0)-32
    ),
    '<=',
    1e-12,
    '0 C'
);

cmp_ok(
    abs(
        celsius_a_fahrenheit(100)-212
    ),
    '<=',
    1e-12,
    '100 C'
);

cmp_ok(
    abs(
        fahrenheit_a_celsius(32)
    ),
    '<=',
    1e-12,
    '32 F'
);

cmp_ok(
    abs(
        fahrenheit_a_celsius(-40)+40
    ),
    '<=',
    1e-12,
    '-40 F'
);

cmp_ok(
    abs(
        razon_limitada(3,4,10)-0.5
    ),
    '<=',
    1e-12,
    'razon 0.5'
);
```

```
is(
    razon_limitada(3,4,2),
    1,
    'razon limitada'
);


my $hubo_error=0;

eval {
    paridad(2.5);
};

$hubo_error=1
    if $@ ne '';

ok(
    $hubo_error,
    'paridad invalida'
);


$hubo_error=0;

eval {
    razon_limitada(3,4,0);
};

$hubo_error=1
    if $@ ne '';

ok(
    $hubo_error,
    'z igual a cero'
);


done_testing();
```

Archivo ejemplos/ejemplo_uso.pl:

```
use strict;
use warnings;

use FindBin;
use lib "$FindBin::Bin/../lib";

use MiBiblioteca qw(
    paridad
    celsius_a_fahrenheit
    fahrenheit_a_celsius
```

```
    razon_limitada
);


print paridad(17),"\n";

print(
    celsius_a_fahrenheit(25),
    "\n"
);

print(
    fahrenheit_a_celsius(77),
    "\n"
);

print(
    razon_limitada(3,4,10),
    "\n"
);
```

JavaScript. En JavaScript para Node.js puede utilizarse un módulo CommonJS. La biblioteca exporta únicamente las funciones que constituyen su interfaz pública.

Una organización posible es:

```
mi_biblioteca/
    src/
        mi_biblioteca.js
    pruebas/
        test_biblioteca.js
    ejemplos/
        ejemplo_uso.js
    README.md
```

Archivo `src/mi_biblioteca.js`:

```
function comprobarReal(x){

    if(
        typeof x!=="number"
        || !Number.isFinite(x)
    ){

        throw new TypeError(
            "Se esperaba un escalar real y finito"
        )

    }
```

```
}


function paridad(n){

    if(!Number.isInteger(n)){

        throw new TypeError(
            "n debe ser un entero"
        )

    }

    if(n%2===0){
        return "es par"
    }

    return "es impar"

}


function celsiusAFahrenheit(c){

    comprobarReal(c)

    return c*9/5+32

}


function fahrenheitACelsius(f){

    comprobarReal(f)

    return (f-32)*5/9

}


function razonLimitada(x,y,z){

    comprobarReal(x)
    comprobarReal(y)
    comprobarReal(z)

    if(z<=0){

        throw new RangeError(
            "z debe ser estrictamente positivo"
        )
```

```
    }

    let razon=
        Math.sqrt(x**2+y**2)/z

    if(razon>1){
        return 1
    }

    return razon

}


module.exports={
    paridad,
    celsiusAFahrenheit,
    fahrenheitACelsius,
    razonLimitada
}
```

Archivo pruebas/test_biblioteca.js:

```
const assert=
    require("assert").strict

const {
    paridad,
    celsiusAFahrenheit,
    fahrenheitACelsius,
    razonLimitada
}=require("../src/mi_biblioteca")


const tolerancia=1e-12


assert.equal(
    paridad(4),
    "es par"
)

assert.equal(
    paridad(7),
    "es impar"
)

assert.ok(
    Math.abs(
        celsiusAFahrenheit(0)-32
```

```
    )<=tolerancia
)

assert.ok(
    Math.abs(
        celsiusAFahrenheit(100)-212
    )<=tolerancia
)

assert.ok(
    Math.abs(
        fahrenheitACelsius(32)
    )<=tolerancia
)

assert.ok(
    Math.abs(
        fahrenheitACelsius(-40)+40
    )<=tolerancia
)

assert.ok(
    Math.abs(
        razonLimitada(3,4,10)-0.5
    )<=tolerancia
)

assert.equal(
    razonLimitada(3,4,2),
    1
)

assert.throws(
    ()=>{
        paridad(2.5)
    }
)

assert.throws(
    ()=>{
        razonLimitada(3,4,0)
    }
)


console.log(
    "Todas las pruebas pasaron"
)
```

Archivo `ejemplos/ejemplo_uso.js`:

```
```

```
const {
    paridad,
    celsiusAFahrenheit,
    fahrenheitACelsius,
    razonLimitada
}=require("../src/mi_biblioteca")


console.log(
    paridad(17)
)

console.log(
    celsiusAFahrenheit(25)
)

console.log(
    fahrenheitACelsius(77)
)

console.log(
    razonLimitada(3,4,10)
)
```

5.11.9. Programas compilados

No es tema de la materia el programar en éstos lenguajes, pero conviene tener una idea general de que se trata.

31. **Ejercicio:** Hola mundo compilado

    Escriba en c y en fortran, el famoso programa que imprime "Hola mundo". Compile. Ejecute.

    Luego baje el paquete GNU con su versión en fuentes del programa "Hola mundo". Desempaquételo, Constrúyalo, compile, instale y ejecute.

    Verifique que no es necesario ningún programa instalado, al menos en los casos sin librerías dinámicas, para correr su programa, a diferencia de los programas para lenguajes como scilab que vimos. Teniendo el ejecutable se puede usar. Encuentre una computadora en la cual ese mismo ejecutable no funciona, por ejemplo una computadora con Windows si lo compilo en GNU/Linux, a diferencia de los scripts en los que vino trabajando.

Lenguaje	**Principales ventajas**	**Limitaciones**
Scilab	Especializado en cálculo numérico, álgebra lineal, gráficos y simulación. Permite resolver problemas científicos con pocas líneas de código y una sintaxis muy cercana al lenguaje matemático.	Su utilización fuera del ámbito científico es limitada y dispone de un ecosistema más pequeño que otros lenguajes.
Python	Lenguaje multipropósito ampliamente utilizado en investigación, inteligencia artificial, ciencia de datos, automatización y computación científica. Posee excelentes bibliotecas como NumPy, SciPy, Matplotlib y Pandas.	Fue creado como programa educativo para obligar a los estudiantes a identar el código. Esto hace que no sea un lenguaje que pueda expresarse en una dimensión. En principio intenta que haya una sola forma de hacer las cosas, lo cual uniformiza, y sirve en grandes equipos de trabajo. Requiere aprender conceptos adicionales (instalación de paquetes, entornos virtuales, módulos, etc.) antes de aprovechar todo su potencial.
C	Lenguaje compilado de muy alta eficiencia. Permite desarrollar programas rápidos y comprender el funcionamiento interno de la computadora, el manejo de memoria y las estructuras de datos.	Su sintaxis es más compleja y exige mayor cuidado en la administración de memoria, por lo que resulta menos adecuado para un primer contacto con la programación científica.
Fortran	Lenguaje diseñado para el cálculo científico. Continúa siendo ampliamente utilizado en simulaciones numéricas de gran escala, dinámica de fluidos, meteorología, física computacional e ingeniería. Los compiladores modernos generan código extremadamente eficiente.	Su ecosistema es más reducido que el de Python y posee menos herramientas para aplicaciones de propósito general.
Shell (sh/-bash)	Ideal para automatizar tareas del sistema operativo, ejecutar programas, administrar archivos y construir flujos de trabajo científicos. Es una herramienta indispensable en servidores y sistemas Linux.	No está pensado para desarrollar programas científicos complejos ni para realizar cálculo numérico intensivo.
Perl	Gran expresividad para representar algoritmos. Excelente para procesamiento de texto, archivos y automatiza-	Por su gran expresividad diferentes personas programan en forma diferente, lo que complica formar equipos con programado-

Capítulo 6

Entradas y salidas

6.1. Entrada y salida de datos

6.1.1. Introducción

Un programa resulta útil cuando puede recibir información del exterior, procesarla y comunicar o almacenar sus resultados. Se denomina *entrada/salida* (E/S, o I/O por *input/output*) al conjunto de mecanismos que permiten ese intercambio.

La entrada y la salida no se limitan al teclado y la pantalla. Un mismo programa puede leer o escribir:

- la entrada, salida y error estándar;
- archivos de texto;
- archivos tabulares, como CSV;
- archivos binarios;
- formatos nativos o serializados de cada entorno;
- tuberías que conectan programas entre sí;
- dispositivos y archivos virtuales del sistema operativo;
- la entrada o salida de otros programas ejecutados como procesos externos.

En todos los lenguajes utilizados en el curso aparecen las mismas ideas fundamentales, aunque cambien los nombres de las funciones. Conviene por lo tanto distinguir el *concepto* de su implementación concreta.

Por ejemplo, la operación conceptual

$$\text{abrir} \longrightarrow \text{escribir} \longrightarrow \text{cerrar}$$

se expresa con `mopen`/`mfprintf`/`mclose` en Scilab, con `fopen`/`fprintf`/`fclose` en Octave, con `open` y un administrador de contexto en Python, con `open` y un manejador de archivo en Perl, y con las funciones del módulo `fs` en Node.js.Scilab Enterprises 2026c; Eaton et al. 2024; Python Software Foundation 2026a; The Perl Foundation 2026; OpenJS Foundation 2026a

El objetivo de esta unidad no es memorizar comandos aislados sino aprender a construir un flujo completo:

obtener datos
↓
validarlos
↓
procesarlos
↓
analizarlos
↓
mostrar, almacenar o transmitir los resultados

Este flujo es el que se empleará posteriormente para datos experimentales, simulaciones, gráficos e informes.

6.1.2. Panorama comparativo

La tabla siguiente resume las operaciones principales que se utilizarán. No debe interpretarse como una lista exhaustiva: cada lenguaje dispone de muchas otras funciones y bibliotecas.

Conviene además ordenar estos mecanismos por su *nivel de abstracción*. No todos resuelven el mismo tipo de problema ni conviene emplear siempre la interfaz de menor nivel. En términos generales, el recorrido pedagógico de esta unidad será:

entrada y salida de consola ⟶ texto línea por línea ⟶ datos formateados ⟶ matrices completas y tablas CSV ⟶ persistencia nativa y binaria.

Las primeras interfaces permiten controlar cada operación de lectura o escritura; las de mayor nivel expresan de una sola vez operaciones sobre una estructura completa. Aprender ambas permite elegir el nivel adecuado en lugar de convertir todo problema de E/S en una sucesión de operaciones elementales.

Tarea	**Scilab**	**Octave**	**Python**	**Perl**	**JavaScript/No**
Entrada interactiva	`input`	`input`	`input`	`<STDIN>`	`readline`, `process.stdin`
Salida simple	`disp`	`disp`	`print`	`print`	`console.log`
Salida formateada	`mprintf`, `mfprintf`	`printf`, `fprintf`	`format`, f-strings	`printf`, `sprintf`	plantillas, `toFixed`

Tarea	Scilab	Octave	Python	Perl	JavaScript/No
Abrir/cerrar archivo	`mopen`, `mclose`	`fopen`, `fclose`	`open`, `with`	`open`, `close`	`fs.open*`, `fs.close*`
Texto línea a línea	`mgetl`, `mputl`	`fgetl`, `fgets`, `fputs`	iteración sobre archivo, `readline`, `write`	`<$fh>`, `print $fh`	`fs`, `readline`
Datos formateados	`mfscanf`, `mfprintf`	`fscanf`, `fprintf`, `textscan`	conversión explícita, módulos de texto/CSV	`scanf`-like mediante módulos, `printf`	conversión explícita, plantillas
Matrices de texto	`fscanfMat`, `fprintfMat`	`dlmread`, `dlmwrite`	`numpy.loadtxt`, `numpy.savetxt`	recorrido explícito o biblioteca numérica	recorrido explícito o biblioteca numérica
CSV	`csvRead`, `csvWrite`	`csvread`/`csvwrite`, `dlmread`/`dlmwrite`, paquetes	`csv`, NumPy, pandas	`Text::CSV`, `Text::CSV_XS`	módulos CSV, `fs`
Persistencia nativa	`save`, `load`	`save`, `load`	`.npy`/`.npz`, `pickle`	`Storable`	`v8.serialize`/de
Binario crudo	`mput`, `mget`	`fwrite`, `fread`	`struct`, `array`, NumPy	`pack`, `unpack`	`Buffer`
Proceso externo	`host`	`system`, `popen`	`subprocess`	`system`, `IPC::Open3`	`child_process`

En JavaScript, las referencias de esta unidad corresponden a **Node.js**. El JavaScript de un navegador está deliberadamente restringido para acceder al sistema de archivos local y no ofrece la misma interfaz.OpenJS Foundation 2026a

6.1.3. Entrada y salida interactiva

Leer primero como texto

La entrada realizada por una persona debe considerarse inicialmente texto. Un usuario puede escribir un número correcto, una cadena vacía, un valor fuera de rango o simplemente caracteres que no puedan convertirse a número.

Para programas robustos conviene seguir la secuencia:

1. leer una cadena;
2. eliminar espacios o el fin de línea cuando corresponda;
3. verificar la sintaxis esperada;
4. convertir al tipo requerido;

5. verificar restricciones adicionales, por ejemplo que el número sea finito, entero, positivo o esté dentro de cierto intervalo;

6. repetir la solicitud si la entrada no es válida.

Este método evita que una expresión introducida accidentalmente sea evaluada como código y permite producir mensajes de error más precisos.

Por ejemplo, para leer una dimensión de una matriz no alcanza con comprobar que se obtuvo un número: además debe verificarse que sea entero y positivo. Para un elemento real debe comprobarse, cuando el lenguaje lo permita, que el resultado sea finito y no `NaN` o infinito.

Colecciones de tamaño fijo y dinámico

Algunas entradas permiten conocer de antemano la cantidad de datos. Si se sabe que se leerá una matriz de $m \times n$, los entornos matriciales como Scilab y Octave, y bibliotecas como NumPy, permiten reservar previamente el espacio. La preasignación evita realocaciones repetidas y hace explícitas las dimensiones esperadas.

En otros problemas la cantidad de datos se conoce recién al finalizar la entrada. Un ejemplo es una lista de calificaciones terminada por un valor centinela. En ese caso resulta natural utilizar una colección dinámica: listas en Python, arreglos dinámicos en Perl y JavaScript, o ampliar un vector cuando el tamaño del problema es pequeño.

La estructura de almacenamiento debe elegirse de acuerdo con la forma en que llegan los datos, no solamente por comodidad sintáctica.

Salida simple y salida formateada

La salida puede tener dos objetivos diferentes:

- **inspección**: mostrar rápidamente una matriz, un vector o una estructura completa;
- **presentación**: producir un texto cuyo ancho, cantidad de decimales, alineación y unidades estén controlados.

Para inspección suelen utilizarse `disp`, `print` o `console.log`. Para informes se emplean especificaciones de formato, por ejemplo:

```
valor =  12.3457 m
error =   0.0021 m
```

Las familias `printf`/`fprintf` de Scilab, Octave y Perl usan convenciones inspiradas en C. En Python se usan habitualmente f-strings o `format`; en JavaScript se combinan plantillas con métodos como `toFixed`.

Una especificación como `%.4f` no modifica el valor numérico almacenado: solamente controla su representación textual.

6.1.4. Archivos de texto: lectura, escritura y actualización

Abrir, usar y cerrar

Un archivo abierto queda asociado a un descriptor, manejador u objeto de archivo. El programa debe conservar esa referencia durante las operaciones de lectura o escritura y cerrarla cuando termina.

Los modos más habituales son:

Modo	Significado
`r`	lectura; el archivo debe existir
`w`	escritura; crea el archivo o reemplaza su contenido anterior
`a`	agregado; las nuevas escrituras se incorporan al final
`r+`, `w+`, `a+`	variantes de lectura/escritura cuando el lenguaje las admite
`b`	modificador para tratar el archivo como binario en interfaces que distinguen texto de binario

No debe suponerse que la apertura siempre tiene éxito. El archivo puede no existir, la ruta puede ser incorrecta o el proceso puede no tener permisos. La comprobación del error de apertura forma parte de la lógica del programa.The IEEE y The Open Group 2024; The Linux man-pages project 2026

También es importante cerrar los archivos. Además de liberar recursos, el cierre fuerza la entrega de datos que todavía puedan permanecer en memorias intermedias de la biblioteca de E/S.

Acceso secuencial, posición y actualización

Hasta ahora hemos considerado principalmente archivos que se recorren desde el comienzo hasta el final. Éste es el *acceso secuencial*.

Un archivo ordinario mantiene además una posición corriente. Cada lectura o escritura avanza esa posición. En muchas interfaces es posible consultar o modificarla mediante operaciones equivalentes a `tell` y `seek`.

Esto permite el *acceso aleatorio*: desplazarse directamente a una posición determinada sin recorrer previamente todo el archivo.

Debe distinguirse entre:

- **reemplazar**: abrir con un modo como `w`, eliminando el contenido anterior;
- **agregar**: abrir con `a`, escribiendo siempre al final;
- **actualizar**: abrir para lectura y escritura y modificar partes de un archivo existente.

La actualización directa resulta sencilla cuando los registros tienen tamaño fijo, especialmente en archivos binarios:

$$\text{posición del registro } i = \text{cabecera} + i\,\text{tamaño del registro}.$$

En archivos de texto o CSV los registros suelen tener longitudes diferentes. Modificar un campo puede cambiar la longitud de una línea, desplazando todo lo que sigue. Por esa razón normalmente resulta más seguro:

leer $\longrightarrow$ modificar en memoria $\longrightarrow$ escribir un archivo nuevo $\longrightarrow$ reemplazar el anterior.

Las interfaces concretas proporcionan operaciones de posicionamiento equivalentes a:

Entorno	**Posicionamiento**
Scilab	`mseek`, `mtell`
Octave	`fseek`, `ftell`, `frewind`
Python	`seek`, `tell`
Perl	`seek`, `tell`
Node.js	posición explícita en operaciones de `fs`

El acceso aleatorio debe usarse conociendo la representación del archivo. Una posición corresponde a bytes, no necesariamente a filas, caracteres o registros lógicos.

Lectura línea por línea

El procesamiento línea por línea es apropiado para:

- archivos de configuración;
- listas de nombres;
- informes;
- bitácoras;
- datos que deben validarse registro por registro;
- archivos demasiado grandes para cargarlos completamente en memoria.

La operación general es:

$$\text{abrir} \to \text{leer línea} \to \text{validar/procesar} \to \text{repetir} \to \text{cerrar}.$$

En un archivo de datos es conveniente distinguir entre errores de E/S y registros inválidos. El primer caso significa que no pudo leerse correctamente el flujo; el segundo indica que se leyó una línea pero su contenido no cumple el formato esperado.

Entrada y salida formateada

Los archivos tabulares numéricos sencillos pueden escribirse mediante un formato conocido. Por ejemplo:

```
0.000   20.1500
0.500   20.3200
1.000   20.8100
```

Si el formato es estable pueden utilizarse rutinas equivalentes a `scanf`/`printf`. Cuando existe la posibilidad de campos opcionales, textos con separadores o datos incompletos, resulta más robusto leer la línea completa y analizarla explícitamente.

Archivos de ancho fijo

Un informe puede reservar un ancho determinado a cada campo:

```
DNI        Apellido            Nombre          Nota
30123456   Perez               Ana             8.50
28765432   Gomez               Luis            5.00
```

Este formato es útil para lectura humana, impresión y comparación visual. El ancho forma parte de la *presentación*; no debe confundirse con la estructura lógica de los datos. Para intercambio entre programas suele ser preferible CSV u otro formato estructurado.

Reemplazar o agregar: bitácoras

Una bitácora (*log*) debe conservar ejecuciones anteriores. Por ello se abre normalmente en modo de agregado. Cada línea debería ser autosuficiente e incluir, al menos, fecha/hora, operación y resultado o diagnóstico.

Una buena regla es no mezclar salida normal y errores sin identificarlos. Por ejemplo:

```
2026-08-21 18:30:12  10 / 2   RESULTADO 5
2026-08-21 18:30:18  10 / 0   ERROR division por cero
```

El archivo puede luego procesarse con el propio lenguaje o con herramientas de terminal como `grep`, `wc`, `sort` y `awk`.

Escritura incremental y visibilidad en tiempo real

Cuando un proceso genera mediciones continuamente, no siempre conviene esperar hasta el final para escribir todo. Otros procesos pueden necesitar observar el archivo mientras crece, por ejemplo mediante:

```
tail -f sensor.txt
```

Hay que distinguir dos conceptos:

- **vaciar los buffers de usuario** (*flush*): hacer que los datos pendientes lleguen al sistema operativo;
- **forzar persistencia física**: solicitar que el sistema operativo sincronice los datos con el dispositivo de almacenamiento.

Para que otro proceso vea nuevas líneas normalmente alcanza con vaciar o cerrar el flujo; no es necesario forzar una sincronización física en cada medición. Hacerlo continuamente puede ser muy costoso.

Octave dispone de `fflush`. Python ofrece el método `flush` de los objetos de archivo. Perl puede activar `autoflush`. En Node.js los métodos síncronos de agregado simplifican ejercicios pequeños, mientras que los flujos permiten soluciones asincrónicas. En Scilab, para un ejercicio simple y portable, puede cerrarse y reabrirse el archivo en modo de agregado entre bloques de mediciones.

La elección entre esta E/S incremental y las operaciones de alto nivel depende del modo en que existen los datos. Si un instrumento, una simulación prolongada o un sistema de control produce muestras una por una, resulta natural escribir o procesar cada registro a medida que llega. Si, en cambio, la matriz completa ya está disponible en memoria, suele ser más claro y menos propenso a errores leerla o escribirla mediante una operación matricial de alto nivel. En forma resumida:

$$\text{flujo continuo} \longrightarrow \text{E/S incremental}, \qquad \text{matriz completa} \longrightarrow \text{E/S matricial de alto nivel}.$$

No es una regla absoluta: el tamaño de los datos, la memoria disponible y la necesidad de observar resultados parciales también intervienen en la elección.

6.1.5. Datos tabulares, persistencia e intercambio

Matrices numéricas en texto

Cuando todos los elementos son numéricos y cada fila posee la misma cantidad de columnas, una matriz puede representarse con un número por campo y un salto de línea por fila. Este formato es simple y muy útil para intercambiar datos con herramientas científicas como Gnuplot o SimusolSaravia Alía 2015.

Es importante elegir un formato numérico suficientemente ancho. Por ejemplo, escribir varios números sin espacio suficiente puede producir un archivo en el que dos campos parezcan unidos.

En Scilab, `fprintfMat`/`fscanfMat` realizan esta tarea de una sola operación. Octave ofrece `dlmwrite`/`dlmread`; NumPy, `savetxt`/`loadtxt`. En Perl y Node.js el mismo formato puede generarse mediante recorridos explícitos o bibliotecas numéricas.

En Física y en otras áreas de cálculo científico este formato matricial simple es especialmente útil: muchos resultados experimentales o de simulación son, en esencia, columnas

numéricas con una variable independiente y una o más magnitudes calculadas o medidas. El archivo puede inspeccionarse directamente, ser leído por otro lenguaje y servir de entrada a programas científicos como Gnuplot o SimusolSaravia Alía 2015. Por eso conviene distinguirlo de un formato nativo: el primero privilegia transparencia e interoperabilidad; el segundo conserva mejor la estructura interna de un entorno concreto.

Por ejemplo, en Scilab pueden calcularse dos funciones sobre la misma grilla y guardarse las tres columnas en un archivo de texto preparado para ser leído por Gnuplot u otro programa:

```
function y = f(x)
    y = (x.^2 - 1) ./ (x.^3 + 2)
endfunction

function y = g(x)
    y = (x.^2 + 1) ./ (x.^3 + 1)
endfunction

x  = linspace(0, 10, 200)
yf = f(x)
yg = g(x)

// columnas: x, f(x), g(x)
fprintfMat("fg.txt", [x(:) yf(:) yg(:)], "%15.8f")
```

El ancho del formato es importante: si los campos se escriben sin separación suficiente, dos números consecutivos pueden parecer unidos y el archivo deja de ser una tabla legible y robusta para el intercambio.

CSV

El formato CSV se utiliza ampliamente para el intercambio de datos tabulares y posee una descripción de referencia en RFC 4180.Shafranovich 2005

CSV representa una tabla mediante campos separados por un delimitador. Es una opción particularmente práctica cuando los datos deben intercambiarse con programas de planilla de cálculo como Excel o LibreOffice Calc, o cuando una tabla combina números y texto. Su aparente simplicidad puede ser engañosa: un campo puede contener el propio delimitador, comillas o saltos de línea y, en esos casos, debe aplicarse el mecanismo de entrecomillado correspondiente.

Por ejemplo:

```
apellido,nombre,dni,calle,numero
Perez,Ana,30123456,"Av. Norte, km 3",125
```

Por esa razón, cuando los campos pueden contener texto arbitrario, es preferible utilizar una rutina CSV real y no simplemente dividir cada línea en las comas.

Otros aspectos que deben definirse son:

- presencia o ausencia de una cabecera;

- delimitador utilizado;
- símbolo decimal;
- codificación de caracteres;
- representación de valores ausentes;
- tipos esperados de cada columna.

Scilab proporciona `csvRead`/`csvWrite`. Python posee el módulo estándar `csv`; Perl dispone de `Text::CSV` y `Text::CSV_XS`; en Node.js existen módulos específicos para CSV. Octave puede utilizar sus funciones delimitadas y paquetes adicionales cuando se necesitan campos heterogéneos.

Validación de registros y cabeceras

Antes de procesar un archivo tabular conviene verificar:

1. que la cabecera sea la esperada;
2. que cada registro tenga la cantidad correcta de campos;
3. que los campos obligatorios no estén vacíos;
4. que los campos numéricos puedan convertirse y estén en rango;
5. que los identificadores respeten las reglas elegidas.

Esta validación es especialmente importante cuando se ordenan registros, generan informes o construyen documentos a partir de los datos. Un error de entrada debe detectarse antes de producir decenas de salidas incorrectas.

Ordenamiento y transformación de registros

Una vez leída una tabla, puede ser necesario ordenar sus filas por una columna, por ejemplo un DNI. Para que el orden sea correcto debe decidirse primero si el campo se interpreta como número o como cadena. Un identificador con ceros iniciales puede necesitar conservarse como texto aun cuando esté formado sólo por dígitos.

El patrón general es:

$$\text{CSV} \rightarrow \text{validar} \rightarrow \text{convertir a estructura interna} \rightarrow \text{ordenar} \rightarrow \text{generar una nueva representación.}$$

Normalización de datos tabulares

Una tabla puede ser sintácticamente correcta y, sin embargo, contener mucha redundancia. Supóngase un archivo de mediciones:

```
sensor_id,nombre,ubicacion,unidad,tiempo_h,valor
S01,Termometro norte,Laboratorio A,C,0,20.1
S01,Termometro norte,Laboratorio A,C,1,20.4
S02,Termometro sur,Laboratorio B,C,0,19.7
```

Los atributos `nombre`, `ubicacion` y `unidad` dependen del sensor, mientras que `tiempo_h` y `valor` describen una medición. Repetir los datos del sensor en todas las mediciones produce anomalías:

- una actualización debe repetirse en muchas filas;
- puede quedar información contradictoria si sólo se modifica una parte;
- no es natural registrar un sensor que todavía no tenga mediciones;
- eliminar todas las mediciones puede eliminar accidentalmente también la información del sensor.

Una organización más adecuada separa:

```
sensores.csv
mediciones.csv
```

En `sensores.csv`, `sensor_id` identifica unívocamente un sensor. En `mediciones.csv`, el par

$$(\text{sensor_id}, \text{tiempo_h})$$

puede utilizarse como clave de una medición. El `sensor_id` de cada medición debe además aparecer entre los sensores conocidos. Aun usando sólo archivos CSV, un programa puede verificar esa condición de integridad.

Esta separación introduce la idea de *normalización*: almacenar cada hecho en el lugar donde depende de su clave y evitar duplicaciones innecesarias. En una introducción resultan útiles las primeras formas normales:

- **primera forma normal**: cada campo contiene un valor atómico y no una lista de valores;
- **segunda forma normal**: si una clave es compuesta, los atributos que no pertenecen a la clave no deben depender solamente de una parte de ella;
- **tercera forma normal**: los atributos que no pertenecen a la clave no deben depender unos de otros de manera transitiva.

En el ejemplo original, nombre, ubicación y unidad dependen solamente de `sensor_id`, no del par completo que identifica una medición. Por eso se trasladan a la relación de sensores. El objetivo práctico es reconocer la redundancia, identificar las claves y comprobar desde un programa que las dos tablas mantienen referencias coherentes.

La consulta mediante SQL se estudia por separado y no forma parte de esta sección.

Formatos nativos y serialización

Los formatos de texto son excelentes para intercambio, pero pierden parte de la información que existe en memoria: tipos, dimensiones, estructuras, objetos y otros metadatos. Los entornos proporcionan mecanismos de persistencia que conservan más información.

Entorno	Ejemplo de persistencia
Scilab	`save`/`load`, habitualmente archivos `.sod`
Octave	`save`/`load`, formatos de Octave o MAT
Python	NumPy `.npy`/`.npz`; `pickle` para objetos Python
Perl	`Storable`
Node.js	`v8.serialize`/`v8.deserialize` u otros formatos serializados

Estos formatos son apropiados para resultados intermedios dentro del mismo entorno, pero no deben suponerse portables entre lenguajes o incluso entre todas las versiones de un mismo entorno. Para intercambio se prefieren formatos documentados como CSV, JSON, texto tabular, formatos de imagen o, cuando corresponda, formatos científicos estandarizados.Shafranovich 2005; Bray 2017

Interoperabilidad

Un buen ejercicio de interoperabilidad consiste en escribir un CSV desde un lenguaje y leerlo desde otro. Para que funcione deben acordarse:

- delimitador;
- codificación;
- símbolo decimal;
- nombres y orden de las columnas;
- representación de valores ausentes;
- precisión numérica.

La extensión del archivo no garantiza que estos acuerdos se cumplan; el formato debe estar documentado.

Generación de informes y documentos

Los datos tabulares pueden transformarse en documentos. Un programa puede leer un CSV, validar cada registro y producir:

- un informe de texto;
- un archivo de ancho fijo;
- una carta por registro;
- un archivo fuente para otro sistema, por ejemplo LaTeX;
- un PDF generado posteriormente por una herramienta externa.

Cuando el programa inserta datos en un lenguaje de marcado debe escapar los caracteres especiales de ese lenguaje. En LaTeX, por ejemplo, caracteres como `&`, `%`, `$`, `#`, `_`, `{` y `}` tienen significado propio. Copiarlos sin tratamiento puede producir un documento inválido o con un significado diferente del esperado.Lamport 1994

El flujo típico es:

`personas.csv` → programa → `carta.tex` → `pdflatex` → `carta.pdf`.

Esto combina procesamiento de archivos con ejecución de procesos externos.

6.1.6. Representación binaria y estándares de archivos

Texto frente a binario

El número real 3,14159 puede almacenarse como los caracteres

```
3.1415900000
```

o como un patrón de bytes que represente directamente un número de punto flotante. En el segundo caso es necesario conocer exactamente:

- el tipo numérico;
- la cantidad de bytes;
- el orden de los bytes;
- cualquier cabecera o metadato que acompañe a los datos.

Los números reales de doble precisión se representan habitualmente mediante IEEE 754 de 64 bits, es decir, ocho bytes por valor.IEEE 2019

Un archivo de texto suele ocupar más espacio y requiere conversión entre texto y números, pero puede inspeccionarse con un editor y es muy portable. Un archivo binario suele ser más compacto y rápido, pero solamente puede interpretarse correctamente si se conoce su representación.

Orden de bytes: endianness

Un valor formado por varios bytes puede almacenarse comenzando por el byte menos significativo (*little endian*) o por el más significativo (*big endian*). Si un archivo binario debe intercambiarse entre programas, el orden debe fijarse explícitamente.

En Scilab las funciones `mput`/`mget` aceptan especificaciones como `dl` y `db` para dobles little endian y big endian, respectivamente. Octave permite indicar arquitecturas `ieee-le` e `ieee-be` en la E/S binaria. Python dispone del módulo `struct`; Perl de `pack`/`unpack`; Node.js ofrece operaciones específicas en `Buffer`, por ejemplo para dobles LE o BE.

Si el formato no se documenta, una secuencia de bytes carece de significado unívoco.Bryant y O'Hallaron 2016; Scilab Enterprises 2026c; Eaton et al. 2024; Python Software Foundation 2026a; The Perl Foundation 2026; OpenJS Foundation 2026a

Firmas mágicas y extensiones

Muchos formatos comienzan con una secuencia característica de bytes, llamada *firma mágica* o *magic bytes*. Herramientas como `file` pueden utilizar esas firmas y otras reglas para identificar el contenido sin confiar solamente en la extensión.The IEEE y The Open Group 2024; International Organization for Standardization 2004

Ejemplos conocidos incluyen PDF, PNG, JPEG, ZIP y ELF. En cambio, un archivo de números binarios crudos puede no tener ninguna firma: el programa que lo lee debe conocer el tipo y la estructura por otros medios.

La extensión es una convención útil para las personas y las aplicaciones, pero no prueba qué contiene realmente el archivo.

Codificación de texto y saltos de línea

Un archivo de texto también tiene una representación binaria. Es necesario conocer su codificación, normalmente UTF-8 en los ejemplos modernos. Además, los sistemas operativos han usado históricamente distintas convenciones para el fin de línea.Yergeau 2003; The IEEE y The Open Group 2024

Las bibliotecas actuales suelen normalizar estas diferencias cuando un archivo se abre en modo texto. En modo binario, en cambio, el programa recibe los bytes sin esa interpretación.

6.1.7. Entrada, salida y error estándar

Cuando comienza un proceso en un sistema tipo Unix ya existen tres flujos:The IEEE y The Open Group 2024

Descriptor	Nombre	Uso
0	stdin	entrada estándar
1	stdout	salida estándar normal
2	stderr	mensajes de diagnóstico y error

La terminal suele conectar el teclado con stdin y la pantalla con stdout y stderr, pero el programa no necesita saber qué hay en el otro extremo. El shell puede sustituir esos extremos por archivos o tuberías.

Ejemplos:

```
programa < entrada.txt
programa > salida.txt
programa >> salida.txt
programa 2> errores.txt
programa > salida.txt 2> errores.txt
```

Una buena práctica es escribir resultados normales en stdout y diagnósticos en stderr. Así se puede redirigir el resultado sin mezclarlo con mensajes de error.

Flujos estándar en los lenguajes del curso

Entorno	stdin	stdout	stderr
Scilab	`%io(1)`	`%io(2)`	descriptor `0`
Octave	`stdin()`	`stdout()`	`stderr()`
Python	`sys.stdin`	`sys.stdout`	`sys.stderr`
Perl	`STDIN`	`STDOUT`	`STDERR`
Node.js	`process.stdin`	`process.stdout`	`process.stderr`

Las funciones de alto nivel, como `input`, `print` o `console.log`, suelen utilizar estos flujos implícitamente.

6.1.8. Redirección y tuberías

Una tubería conecta stdout de un proceso con stdin de otro.The IEEE y The Open Group 2024; Ramey y Fox 2025

```
cat /etc/passwd | cut -d: -f1 | sort | head -10
```

Cada programa realiza una operación pequeña y puede combinarse con otros. La ventaja principal es que no es necesario crear un archivo intermedio para cada etapa.

La misma idea se aplica a programas escritos por los estudiantes. Un programa que lea de stdin y escriba sólo los resultados en stdout puede utilizarse de forma interactiva, con archivos o dentro de una tubería sin modificar el código.

6.1.9. Ejecución de programas externos

Un lenguaje de programación puede delegar una tarea a otro programa. Algunos ejemplos de esta unidad son:

- `gnuplot` para producir gráficos;

- `pdflatex` para construir un PDF;
- `curl` o `wget` para obtener datos;
- `file` o `xxd` para inspeccionar archivos;
- herramientas de terminal para ordenar, filtrar o contar líneas.

La ejecución de un proceso externo debe considerarse una operación de E/S. El programa puede proporcionar stdin al hijo y capturar stdout y stderr.

No alcanza con ejecutar el comando: debe comprobarse su *código de terminación*. Por convención, en sistemas tipo Unix el código cero significa éxito y otros valores indican diferentes clases de error.The IEEE y The Open Group 2024

Entorno	**Mecanismos habituales**
Scilab	`host`, que puede devolver estado, stdout y stderr
Octave	`system`; `popen` cuando se necesita un flujo
Python	módulo `subprocess`
Perl	`system`; `IPC::Open3` para stdin/stdout/stderr separados
Node.js	módulo `child_process`: `spawn`, `spawnSync`, `execFile`, etc.

Cuando el nombre del programa y sus argumentos se conocen por separado es preferible utilizar una interfaz que reciba un arreglo de argumentos en lugar de construir una única cadena para el shell. Esto reduce errores de quoting y evita que datos no confiables sean interpretados como parte de un comando.Scilab Enterprises 2026b; Eaton et al. 2024; Python Software Foundation 2026a; The Perl Foundation 2026; OpenJS Foundation 2026a

6.1.10. El sistema de archivos como interfaz del sistema operativo

En GNU/Linux no todos los nombres del sistema de archivos representan datos guardados en un disco. Directorios como `/proc` y `/dev` exponen información o dispositivos mediante una interfaz semejante a la de un archivo.Linux Foundation 2015; The Linux Kernel Organization 2026

Propietarios, grupos y permisos

Un archivo no está formado solamente por su nombre y contenido. El sistema de archivos mantiene también metadatos tales como su propietario, grupo, permisos, tamaño y fechas de acceso y modificación.

En GNU/Linux, el modelo tradicional de permisos distingue tres clases de usuarios:

- **propietario** (*user*): usuario al que pertenece el archivo;
- **grupo** (*group*): grupo asociado al archivo;

- **otros** (*others*): todos los demás usuarios.

Para cada una pueden asignarse tres permisos básicos:

Letra	Permiso	Significado en un archivo ordinario
`r`	lectura	permite leer su contenido
`w`	escritura	permite modificar su contenido
`x`	ejecución	permite solicitar su ejecución como programa

Visualización de permisos El comando

```
ls -l datos.txt
```

puede producir, por ejemplo:

```
-rw-r----- 1 ana laboratorio 2840 ago 21 18:30 datos.txt
```

Los primeros caracteres pueden separarse conceptualmente como:

```
-  rw-  r--  ---
|   |    |    |
|   |    |    +-- otros
|   |    +------- grupo
|   +------------ propietario
+---------------- tipo de archivo
```

El primer carácter indica el tipo de objeto. Algunos valores habituales son:

- `-`: archivo ordinario;
- `d`: directorio;
- `l`: enlace simbólico;
- `c`: dispositivo de caracteres;
- `b`: dispositivo de bloques.

Los nueve caracteres siguientes expresan los permisos del propietario, grupo y otros.The IEEE y The Open Group 2024; The Linux man-pages project 2026

En el ejemplo,

```
rw-r-----
```

significa que el propietario puede leer y escribir, los miembros del grupo pueden leer y los restantes usuarios no poseen ninguno de esos permisos.

Representación numérica Los permisos pueden representarse también mediante números. Se asignan los valores:

$$r = 4, \qquad w = 2, \qquad x = 1.$$

La suma proporciona el permiso de cada clase. Por ejemplo:

$$6 = 4 + 2 = rw-, \qquad 5 = 4 + 1 = r - x, \qquad 7 = 4 + 2 + 1 = rwx.$$

Por lo tanto:

`-rw-r--r--`

corresponde a:

`644`

y:

`-rwxr-xr-x`

corresponde a:

`755`

Cambio de permisos: chmod El comando `chmod` modifica los permisos.

Puede utilizarse la forma numérica:

```
chmod 644 datos.txt
chmod 755 programa.sh
```

o una forma simbólica:

```
chmod u+x programa.sh
chmod g+w datos.txt
chmod o-r datos.txt
chmod a+r informe.txt
```

Las letras utilizadas son:

`u`	propietario (*user*)
`g`	grupo
`o`	otros
`a`	todos (*all*)

Los operadores `+`, `-` y `=` agregan, eliminan o establecen exactamente determinados permisos.

Propietario y grupo El propietario y el grupo pueden observarse también mediante `ls -l`. Para modificarlos existen:

```
chown usuario archivo
chgrp grupo archivo
```

También pueden establecerse propietario y grupo conjuntamente:

```
chown usuario:grupo archivo
```

La posibilidad de cambiar el propietario está restringida por el sistema. En las tareas administrativas suele ser necesario utilizar privilegios adicionales, por ejemplo mediante `sudo`.

Los usuarios pueden pertenecer a varios grupos. Los comandos:

```
id
groups
```

permiten consultar esta información.

Los permisos de los directorios Los mismos bits `r`, `w` y `x` se utilizan para directorios, pero su significado cambia:

Permiso	Significado en un directorio
`r`	permite leer la lista de nombres contenidos en el directorio
`w`	permite crear, eliminar o cambiar nombres de entradas del directorio, si los restantes permisos necesarios lo permiten
`x`	permite atravesar o buscar dentro del directorio y acceder a sus elementos por nombre

Esta diferencia es importante. El permiso de escritura de un archivo controla la modificación de su *contenido*; eliminar el archivo modifica en realidad la entrada correspondiente del *directorio* que lo contiene.

Por ejemplo, es posible que un usuario no pueda modificar el contenido de un archivo pero sí pueda eliminarlo si posee los permisos apropiados sobre el directorio.

También es posible conocer el nombre de los elementos de un directorio mediante el permiso `r` pero no poder acceder a ellos si falta el permiso `x`.

Permisos iniciales y umask Cuando un programa crea un archivo, los permisos finales no dependen solamente del programa. El proceso posee además una *máscara de creación* denominada `umask`.

Puede consultarse mediante:

```
umask
```

Por ejemplo, una máscara:

```
0022
```

elimina normalmente los permisos de escritura para grupo y otros de los permisos solicitados durante la creación.

Conceptualmente:

$$\text{permisos finales} = \text{permisos solicitados} - \text{permisos inhibidos por umask.}$$

Por ello un programa que crea un archivo no debería suponer que éste tendrá exactamente los mismos permisos en todas las cuentas y sistemas.

Permisos y errores de entrada/salida Los permisos forman parte directa de la programación de E/S. Una llamada como:

$$\text{programa} \longrightarrow \text{abrir archivo} \longrightarrow \text{sistema operativo}$$

puede fracasar aunque el archivo exista.

Por ejemplo:

```
cat datos.txt
```

puede producir un error de permiso si el usuario no posee derecho de lectura.

De igual manera, un programa que intente:

- leer un archivo sin permiso `r`;
- modificarlo sin permiso `w`;
- crear un archivo dentro de un directorio no escribible;
- atravesar un directorio sin permiso `x`;

recibirá un error del sistema operativo.

Por ello los programas deben comprobar siempre el resultado de las operaciones de apertura, lectura, escritura y creación, en lugar de suponer que tendrán éxito.

Bits especiales GNU/Linux y otros sistemas de tradición Unix poseen además los bits *set-user-ID*, *set-group-ID* y *sticky*. No son necesarios para comprender la E/S básica, pero aparecen al examinar permisos reales.

El bit *sticky* es especialmente frecuente en directorios compartidos, como `/tmp`. Permite que varios usuarios creen archivos en el mismo directorio evitando, con las reglas correspondientes, que un usuario elimine libremente los archivos pertenecientes a otros.

Los bits *setuid* y *setgid* modifican determinados aspectos de la identidad efectiva con la que se ejecuta un programa o del grupo asociado a archivos creados en ciertos directorios. Debido a sus implicancias de seguridad, deben utilizarse con conocimiento preciso de su comportamiento.

Listas de control de acceso El modelo propietario–grupo–otros es deliberadamente simple. Los sistemas GNU/Linux modernos pueden disponer además de listas de control de acceso (*ACL*), que permiten asignar permisos diferentes a usuarios o grupos individuales.

Cuando están disponibles pueden inspeccionarse y modificarse, por ejemplo, mediante:

```
getfacl archivo
setfacl -m u:ana:rw archivo
```

Las ACL complementan los bits tradicionales; no deben confundirse con una segunda copia independiente de los mismos permisos.The Linux man-pages project 2026

macOS macOS pertenece también a la tradición Unix y conserva el modelo POSIX de propietario, grupo y permisos `rwx`. Por ello comandos como:

```
ls -l
chmod
chown
chgrp
umask
```

tienen esencialmente el mismo significado básico que en GNU/Linux.

macOS utiliza además listas de control de acceso, por lo que un archivo puede tener reglas adicionales que no se expresen completamente mediante los nueve bits tradicionales.Apple Inc. 2002

Por esta razón el modelo explicado para GNU/Linux resulta directamente útil en macOS, aunque las extensiones y algunos detalles administrativos puedan ser diferentes.

Windows Windows protege también archivos y directorios, pero su modelo principal no es la terna Unix:

```
propietario / grupo / otros
rwx rwx rwx
```

Los sistemas de archivos como NTFS utilizan principalmente *listas de control de acceso* (ACL). Una lista contiene reglas que conceden o deniegan derechos a usuarios y grupos determinados.

Entre esos derechos pueden aparecer, por ejemplo:

- lectura;
- escritura;

- modificación;
- lectura y ejecución;
- eliminación;
- control total.

Además, las reglas pueden heredarse desde los directorios superiores.

Desde la línea de comandos de Windows pueden inspeccionarse y modificarse ACL mediante `icacls`.Microsoft 2025 Por ejemplo:

```
icacls datos.txt
```

Por tanto, los tres sistemas comparten la idea general:

el sistema operativo decide qué identidad puede realizar qué operación sobre cada objeto

pero utilizan modelos diferentes:

Sistema	Modelo habitual
GNU/Linux	permisos POSIX de propietario/grupo/otros, complementados cuando corresponde por ACL
macOS	permisos POSIX más ACL
Windows/NTFS	ACL con reglas para usuarios y grupos, herencia y un conjunto más detallado de derechos

Para la programación portable, la conclusión más importante no es memorizar todos los mecanismos administrativos sino comprender que la existencia de un archivo no garantiza que un proceso pueda leerlo, modificarlo, ejecutarlo o eliminarlo. Cada operación de E/S está sometida a las reglas de acceso que aplica el sistema operativo.

El directorio /proc

`/proc` es un sistema de archivos virtual. Muchos de sus archivos se generan al momento de leerlos y reflejan el estado actual del núcleo y de los procesos.The Linux Kernel Organization 2026; The Linux man-pages project 2026

Ejemplos:

```
cat /proc/cpuinfo
cat /proc/meminfo
cat /proc/net/dev
```

Un programa en cualquiera de los cinco lenguajes puede leer estos recursos con las mismas rutinas de texto que usaría para un archivo ordinario. Sin embargo, no debe suponerse que poseen todas las propiedades de un archivo almacenado: pueden cambiar entre dos lecturas, no tener un tamaño convencional o no admitir búsqueda arbitraria.

El directorio /dev

`/dev` contiene nodos que representan dispositivos o interfaces especiales. Por ejemplo `/dev/urandom` proporciona un flujo de bytes pseudoaleatorios producido por el sistema.The Linux Kernel Organization 2026; The Linux man-pages project 2026

Leer un dispositivo mediante una llamada de archivo ilustra el principio Unix "todo es un archivo": diferentes recursos presentan una interfaz de E/S uniforme aunque su implementación sea muy distinta.

Archivos ordinarios, virtuales y dispositivos

Conviene distinguir:

- **archivo ordinario**: sus bytes se almacenan normalmente en un sistema de archivos persistente;
- **archivo virtual**: su contenido representa información generada por el sistema;
- **dispositivo**: las lecturas y escrituras se traducen en operaciones sobre un recurso controlado por el núcleo.

Esta abstracción permite reutilizar las mismas ideas de apertura, lectura, escritura y cierre.

6.1.11. Gráficos, imágenes e informes como salida

La salida de un programa no tiene que ser texto. Un gráfico guardado como PNG o PDF también es un archivo generado por el programa.

Un principio importante del práctico es que, cuando se quiere verificar un flujo completo de persistencia, el gráfico debe construirse a partir de los datos *leídos nuevamente del archivo*, no de las variables que quedaron en memoria después de escribirlo. Esto comprueba realmente que el proceso de escritura y recuperación fue correcto.

Scilab y Octave poseen sistemas gráficos propios. Python utiliza habitualmente Matplotlib. En Perl y Node.js puede usarse una biblioteca gráfica o escribirse un archivo de datos y delegar la representación a Gnuplot. En todos los casos el concepto es el mismo:

$$\text{datos} \rightarrow \text{archivo} \rightarrow \text{lectura} \rightarrow \text{gráfico} \rightarrow \text{imagen}.$$

Williams, Kelley y many others 2026; The Matplotlib development team 2026; International Organization for Standardization 2004; Scilab Enterprises 2026c; Eaton et al. 2024

6.1.12. Flujo integrado de procesamiento

Los ejercicios finales combinan las técnicas anteriores. Un programa de procesamiento de datos experimentales debería separar responsabilidades en funciones o módulos:

1. localizar y abrir la entrada;
2. verificar cabecera, cantidad de columnas y tipos;
3. descartar o informar registros inválidos;
4. almacenar sólo los datos aceptados;
5. calcular estadísticas;
6. producir gráficos;
7. escribir un informe textual;
8. registrar la fuente y la cantidad de datos procesados;
9. comprobar los errores de las salidas generadas.

Cuando los datos se descargan de Internet es conveniente conservar una copia local con nombre estable. De esa manera el análisis puede repetirse aunque la fuente externa cambie o deje de estar disponible. También debe registrarse la procedencia de los datos en el informe.

La simulación de un tanque solar ilustra el mismo patrón: se leen datos de radiación, se integran las ecuaciones del modelo, se escriben resultados y se genera un resumen. La física del problema y la E/S son responsabilidades separadas; la misma rutina numérica podría alimentarse con datos provenientes de otro archivo o de un sensor.

6.1.13. Métodos de entrada y salida en Scilab

Las funciones de esta sección siguen la documentación oficial de Scilab.Scilab Enterprises 2026c; Scilab Enterprises 2026a; Scilab Enterprises 2026b

Scilab dispone de mecanismos de alto nivel para matrices y CSV, pero también de funciones de flujo semejantes a las de C. Conviene utilizar las funciones modernas y reservar las interfaces históricas para leer programas antiguos.

Entrada y salida de consola

Las funciones principales son:

- `input`: solicita entrada interactiva; para validación robusta conviene leer una cadena con `input(...,"string")`;
- `disp`: presenta variables completas con un formato automático;
- `mprintf` o `printf`: producen salida formateada;
- `msprintf`: produce la misma representación formateada pero en una cadena.

Ejemplo de lectura segura de un real:

```
valido = %f;
while ~valido
    s = stripblanks(input("Valor: ", "string"));
    if isnum(s) then
        x = strtod(s);
        valido = isfinite(x);
    end
    if ~valido then
        disp("Se esperaba un numero finito")
    end
end
mprintf("x = %.4f\n", x);
```

Archivos de texto

Las funciones de uso habitual son:

- `mopen`: abre un archivo y devuelve un descriptor;
- `mclose`: cierra un descriptor y devuelve un indicador de error;
- `mgetl`: lee una o varias líneas completas;
- `mputl`: escribe cadenas como líneas de texto;
- `mfscanf`: realiza lectura formateada;
- `mfprintf`: realiza escritura formateada;
- `meof`: comprueba fin de archivo;
- `merror`: consulta el indicador de error de un flujo;
- `mclearerr`: restablece indicadores de error cuando corresponde.

```
fd = mopen("datos.txt", "wt");
if fd < 0 then
    error("No se pudo abrir datos.txt")
end

for k = 1:10
    mfprintf(fd, "%4d %12.6f\n", k, sqrt(k));
end

err = mclose(fd);
if err <> 0 then
    error("Error al cerrar datos.txt")
end
```

Los modos “`rt`”, “`wt`” y “`at`” hacen explícita la intención de lectura, reemplazo o agregado de texto.

Matrices completas de texto

Para matrices numéricas regulares resultan especialmente cómodas:

- `fprintfMat`: escribe una matriz con un formato controlado;
- `fscanfMat`: lee una matriz desde un archivo de texto.

```
A = rand(5,3);
fprintfMat("A.txt", A, "%18.10e");
B = fscanfMat("A.txt");
```

Son apropiadas para intercambiar tablas numéricas simples con herramientas científicas externas.

CSV

`csvRead` y `csvWrite` permiten trabajar con tablas delimitadas. Deben especificarse correctamente separador, símbolo decimal, cabecera y tipo de resultado cuando el archivo mezcla texto y números.

```
A = rand(3,3);
csvWrite(A, "A.csv", ",", ".", "%.8f");
B = csvRead("A.csv", ",", ".", "double");
```

Cuando el CSV posee campos de texto complejos es importante comprobar qué reglas de quoting ofrece la función utilizada y no reemplazar un parser CSV por un simple `tokens(linea,",")` si pueden aparecer comas dentro de un campo.

Formato nativo

`save` y `load` conservan variables en el formato propio de Scilab. Es conveniente para resultados intermedios que serán reutilizados por Scilab, pero no es la primera elección para intercambio con otros lenguajes.

```
A = rand(4,4);
texto = "resultado intermedio";
save("estado.sod", "A", "texto");
clear A texto
load("estado.sod");
```

Binarios crudos

`mput` y `mget` escriben y leen números con una representación binaria especificada. El sufijo `l` o `b` puede fijar little o big endian. Por ejemplo, `dl` representa un `double` IEEE de 64 bits little endian.

```
v = [1.0 2.5 3.14159 -7.2 0.001];
fd = mopen("datos.bin", "wb");
mput(v, "dl", fd);
mclose(fd);

fd = mopen("datos.bin", "rb");
w = mget(length(v), "dl", fd);
mclose(fd);
```

Flujos estándar

Scilab mantiene descriptores para los flujos estándar:

- `%io(1)`: entrada estándar;
- `%io(2)`: salida estándar;
- descriptor `0`: error estándar.

Por ejemplo:

```
mfprintf(%io(2), "resultado normal\n");
mfprintf(0, "mensaje de diagnostico\n");
```

Un programa que use estos flujos puede participar de redirecciones y tuberías creadas por el shell.

Procesos externos

La función moderna para ejecutar comandos del sistema es `host`. Puede devolver el código de terminación, stdout y stderr:

```
[estado, salida, errores] = host("file datos.bin");
if estado <> 0 then
    mfprintf(0, "%s\n", errores);
else
    disp(salida);
end
```

La familia histórica `unix*` debe evitarse en código nuevo; `host` es portable entre Linux, macOS y Windows dentro de las posibilidades del comando ejecutado.

Funciones históricas

En programas antiguos pueden aparecer `file`, `read` y `write`. Estas interfaces históricas pertenecen a la tradición de E/S del cálculo científico que Scilab heredó, en particular, de

Fortran. Continúan siendo relevantes para comprender código heredado, pero para programas nuevos se prefieren, según el caso, `mopen`/`mclose`, `mgetl`/`mputl`, `mfscanf`/`mfprintf`, `fscanfMat`/`fprintfMat`, `csvRead`/`csvWrite` y `save`/`load`.

6.1.14. Métodos de entrada y salida en Octave

Las interfaces de esta sección corresponden al manual oficial de GNU Octave.Eaton et al. 2024

Octave posee una interfaz de E/S muy cercana a la biblioteca estándar de C y, al mismo tiempo, funciones de alto nivel orientadas a matrices.

Consola y validación

`input` puede leer expresiones o cadenas. Para validar datos de usuario conviene leer con la opción de cadena y convertir explícitamente mediante `str2double`.

```
valido = false;
while !valido
  s = strtrim(input("Valor: ", "s"));
  x = str2double(s);
  valido = isfinite(x);
  if !valido
    fprintf(stderr(), "Se esperaba un numero finito\n");
  endif
endwhile
printf("x = %.4f\n", x);
```

`disp` es apropiado para inspección de matrices y `printf` para salida controlada.

Archivos de texto

La interfaz principal incluye `fopen`, `fclose`, `fgetl`, `fgets`, `fputs`, `fprintf`, `fscanf` y `textscan`. `fopen` devuelve `-1` si no puede abrir el archivo y puede proporcionar un mensaje de error.

```
[fid, msg] = fopen("datos.txt", "w");
if fid < 0
  error(msg);
endif

for k = 1:10
  fprintf(fid, "%4d %12.6f\n", k, sqrt(k));
endfor

if fclose(fid) != 0
  error("Error al cerrar datos.txt");
endif
```

`fflush(fid)` permite vaciar la salida pendiente sin cerrar el archivo, lo que resulta útil en adquisición o monitoreo en tiempo real.

Matrices, CSV y persistencia

Para tablas numéricas simples pueden utilizarse `dlmread` y `dlmwrite`; también existen `csvread` y `csvwrite` y paquetes que amplían el soporte de datos tabulares heterogéneos.

`save` y `load` conservan variables en formatos propios de Octave o compatibles con MAT. Debe distinguirse ese uso de un archivo pensado para intercambio universal.

```
A = rand(4,4);
dlmwrite("A.csv", A, "delimiter", ",", "precision", "%.17g");
save("A.mat", "A", "-binary");

B = dlmread("A.csv", ",");
clear A
load("A.mat", "A");
```

Binarios

`fwrite` y `fread` permiten especificar precisión y arquitectura. Los nombres `ieee-le` e `ieee-be` fijan el orden de bytes.

```
v = [1.0 2.5 3.14159 -7.2 0.001];
fid = fopen("datos.bin", "wb", "ieee-le");
fwrite(fid, v, "double");
fclose(fid);

fid = fopen("datos.bin", "rb", "ieee-le");
w = fread(fid, length(v), "double")';
fclose(fid);
```

Flujos estándar y procesos

Octave proporciona `stdin()`, `stdout()` y `stderr()`. Por ejemplo:

```
fprintf(stdout(), "resultado\n");
fprintf(stderr(), "diagnostico\n");
```

Para procesos externos, `system` devuelve estado y salida. `popen` permite una interacción orientada a flujo cuando se necesita leer o escribir mientras el proceso está activo.

6.1.15. Métodos de entrada y salida en Python

Las interfaces estándar de esta sección se describen en la documentación oficial de Python; para las operaciones de arreglos se usa además NumPy.Python Software Foundation 2026a;

NumPy Developers 2026

Python ofrece una interfaz uniforme basada en objetos de archivo. Muchas bibliotecas aceptan cualquier objeto que se comporte como un flujo, no sólo un archivo almacenado en disco.

Consola y conversión

`input` devuelve siempre una cadena. Esta propiedad favorece la validación explícita:

```
import math

while True:
    texto = input("Valor: ").strip()
    try:
        x = float(texto)
    except ValueError:
        print("Entrada no numerica")
        continue
    if math.isfinite(x):
        break
    print("Se esperaba un numero finito")

print(f"x = {x:.4f}")
```

Archivos y administradores de contexto

La construcción `with` garantiza que el archivo se cierre al abandonar el bloque, incluso si se produce una excepción:

```
with open("datos.txt", "w", encoding="utf-8") as f:
    for k in range(1, 11):
        f.write(f"{k:4d} {k**0.5:12.6f}\n")
```

Para leer línea por línea puede iterarse directamente sobre el objeto de archivo. Los modos “`r`”, “`w`”, “`a`”, “`rb`” y “`wb`” distinguen las operaciones fundamentales.

El método `flush()` vacía los buffers de Python. Si además se requiere forzar persistencia física puede utilizarse `os.fsync`, aunque no debe hacerse innecesariamente en cada escritura.

CSV

El módulo estándar `csv` implementa las reglas de delimitación y quoting. Para archivos con cabecera son cómodos `DictReader` y `DictWriter`:

```
import csv

with open("personas.csv", newline="", encoding="utf-8") as f:
```

```
    for fila in csv.DictReader(f):
        print(fila["apellido"], fila["dni"])
```

Cuando se trabaja con matrices numéricas, NumPy ofrece `savetxt`/`loadtxt`; para tablas más complejas pueden emplearse bibliotecas como pandas.

Persistencia

NumPy almacena arreglos mediante `save`/`load` en archivos `.npy`. El módulo `pickle` serializa muchos objetos Python, pero no debe cargarse un archivo `pickle` proveniente de una fuente no confiable: su deserialización puede ejecutar código.

Para intercambio con otros lenguajes conviene preferir formatos documentados.

Binarios

El módulo `struct` permite empaquetar y desempaquetar valores con una representación explícita:

```
import struct

v = [1.0, 2.5, 3.14159, -7.2, 0.001]
with open("datos.bin", "wb") as f:
    f.write(struct.pack("<5d", *v))

with open("datos.bin", "rb") as f:
    w = struct.unpack("<5d", f.read(5*8))
```

El carácter `<` especifica little endian y `d` un doble IEEE de 64 bits.

Flujos estándar y procesos

`sys.stdin`, `sys.stdout` y `sys.stderr` son objetos de archivo de texto. `input()` lee de stdin y `print()` escribe, por defecto, en stdout.

```
import sys
print("resultado", file=sys.stdout)
print("diagnostico", file=sys.stderr)
```

El módulo `subprocess` permite ejecutar programas, enviar datos a stdin, capturar stdout/stderr y comprobar el código de terminación. Para argumentos conocidos por separado se prefiere una lista:

```
import subprocess

r = subprocess.run(
    ["file", "datos.bin"],
    text=True,
    capture_output=True
```

```
)
if r.returncode != 0:
    raise RuntimeError(r.stderr)
print(r.stdout)
```

6.1.16. Métodos de entrada y salida en Perl

Las construcciones básicas de esta sección siguen la documentación oficial de Perl.The Perl Foundation 2026

Perl fue diseñado desde sus primeras versiones para procesar texto y archivos. Sus manejadores de archivo y expresiones regulares hacen natural el trabajo con registros, bitácoras y transformación de datos.

Entrada estándar y validación

La forma clásica de leer una línea es `<STDIN>`. `chomp` elimina el separador de línea final.

```
use strict;
use warnings;

my $x;
while (1) {
    print "Valor: ";
    my $s = <STDIN>;
    die "Fin de entrada\n" unless defined $s;
    chomp $s;

    if ($s =~ /^[+-]?(?:\d+(?:\.\d*)?|\.\d+)(?:[eE][+-]?\d+)?$/)
       {
        $x = 0 + $s;
        last;
    }
    warn "Entrada no numerica\n";
}
printf "x = %.4f\n", $x;
```

`use strict` y `use warnings` ayudan a detectar numerosos errores de programación y deben usarse en los programas del curso.

Archivos

Se recomienda la forma de tres argumentos de `open` y manejadores léxicos:

```
open my $fh, ">:encoding(UTF-8)", "datos.txt"
    or die "No se pudo abrir datos.txt: $!";

for my $k (1..10) {
```

```
    printf {$fh} "%4d %12.6f\n", $k, sqrt($k);
}

close $fh or die "Error al cerrar: $!";
```

Para leer:

```
open my $fh, "<:encoding(UTF-8)", "datos.txt" or die $!;
while (my $linea = <$fh>) {
    chomp $linea;
    # validar y procesar
}
close $fh;
```

El modo `»` abre para agregado. Para monitoreo continuo puede utilizarse `IO::Handle` y activar `autoflush`.

CSV

Para CSV real se prefiere `Text::CSV` o `Text::CSV_XS`. Estos módulos comprenden comillas y delimitadores, a diferencia de un simple `split /,/`.Text::CSV maintainers 2025; Shafranovich 2005

```
use Text::CSV;

my $csv = Text::CSV->new({ binary => 1, auto_diag => 1 });
open my $fh, "<:encoding(UTF-8)", "personas.csv" or die $!;

my $cabecera = $csv->getline($fh);
while (my $fila = $csv->getline($fh)) {
    # $fila es una referencia a un arreglo de campos
}
close $fh;
```

Persistencia y binarios

`Storable` permite guardar y recuperar estructuras Perl. Para binarios crudos, `pack` y `unpack` convierten entre valores y secuencias de bytes.

```
my @v = (1.0, 2.5, 3.14159, -7.2, 0.001);
open my $fh, ">:raw", "datos.bin" or die $!;
print {$fh} pack("d<*", @v);
close $fh;

open $fh, "<:raw", "datos.bin" or die $!;
local $/;
my $bytes = <$fh>;
close $fh;
my @w = unpack("d<*", $bytes);
```

Flujos estándar y procesos

Perl dispone de los manejadores preabiertos `STDIN`, `STDOUT` y `STDERR`:

```
print STDOUT "resultado\n";
print STDERR "diagnostico\n";
```

`system` es suficiente cuando sólo interesa el código de terminación. Cuando se necesitan stdin, stdout y stderr separados, `IPC::Open3` permite establecer los tres flujos del proceso hijo.

6.1.17. Métodos de entrada y salida en JavaScript con Node.js

Las APIs de archivos, streams, Buffer, V8 y procesos externos se documentan en la referencia oficial de Node.js.OpenJS Foundation 2026a

Node.js organiza gran parte de su E/S alrededor de *streams* y ofrece también variantes síncronas convenientes para programas didácticos pequeños.

Entrada y salida estándar

`process.stdin`, `process.stdout` y `process.stderr` representan los tres flujos estándar. Para entrada interactiva suele usarse el módulo `readline`.

```
"use strict";
const readline = require("node:readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.question("Nombre: ", nombre => {
    if (nombre.trim() === "") {
        console.error("Nombre vacio");
        process.exitCode = 1;
    } else {
        console.log(`Nombre: ${nombre.trim()}`);
    }
    rl.close();
});
```

Para programas que reciben toda la entrada mediante una redirección también es posible leer el descriptor 0 como archivo:

```
const fs = require("node:fs");
const entrada = fs.readFileSync(0, "utf8");
```

Archivos de texto

El módulo `node:fs` proporciona operaciones síncronas y asincrónicas. Las versiones síncronas simplifican los primeros ejercicios:

```
"use strict";
const fs = require("node:fs");

let salida = "";
for (let k = 1; k <= 10; k++) {
    salida += `${String(k).padStart(4)}
       ${Math.sqrt(k).toFixed(6)}\n`;
}
fs.writeFileSync("datos.txt", salida, "utf8");

const texto = fs.readFileSync("datos.txt", "utf8");
console.log(texto);
```

Para bitácoras se utiliza agregado, por ejemplo `appendFileSync`. Para volúmenes grandes o procesos de larga duración son preferibles los streams de lectura y escritura, que evitan mantener todo el archivo en memoria.

CSV

Un CSV puramente numérico y controlado puede generarse con recorridos y `join`. Para datos generales deben utilizarse módulos CSV que implementen correctamente quoting y escapes.

```
const fs = require("node:fs");
const filas = [
    ["sensor_id", "tiempo_h", "valor"],
    ["S01", 0, 20.1],
    ["S01", 1, 20.4]
];
fs.writeFileSync(
    "mediciones.csv",
    filas.map(f => f.join(",")).join("\n") + "\n",
    "utf8"
);
```

Este ejemplo es válido porque se conocen los datos y ninguno contiene comas o saltos de línea. No debe generalizarse a texto arbitrario.

Persistencia y binarios

El módulo `node:v8` ofrece `serialize` y `deserialize` para conservar muchas estructuras JavaScript en un formato binario propio del entorno.

Para binarios documentados se utiliza `Buffer`. Por ejemplo:

```
const fs = require("node:fs");
const v = [1.0, 2.5, 3.14159, -7.2, 0.001];
const b = Buffer.alloc(v.length * 8);

v.forEach((x, i) => b.writeDoubleLE(x, i * 8));
fs.writeFileSync("datos.bin", b);

const r = fs.readFileSync("datos.bin");
const w = v.map((_, i) => r.readDoubleLE(i * 8));
console.log(w);
```

`writeDoubleBE`/`readDoubleBE` realizan la misma tarea en big endian.

Procesos externos

El módulo `node:child_process` incluye `spawn`, `spawnSync`, `execFile` y variantes. Para programas pequeños, `spawnSync` permite obtener claramente estado, stdout y stderr:

```
const { spawnSync } = require("node:child_process");

const r = spawnSync(
    "file",
    ["datos.bin"],
    { encoding: "utf8" }
);

if (r.status !== 0) {
    console.error(r.stderr);
    process.exitCode = 1;
} else {
    process.stdout.write(r.stdout);
}
```

Separar el programa de sus argumentos evita depender innecesariamente de la interpretación de una cadena por el shell.

6.1.18. Criterios para elegir un formato

No existe un formato universalmente mejor. La elección depende del objetivo.

Formato	Ventajas	Limitaciones
Texto libre	legible, fácil de generar	requiere definir cómo interpretar cada línea
Texto tabular	simple, adecuado para datos numéricos y Gnuplot	pocos metadatos; tipos implícitos
CSV	intercambio tabular ampliamente soportado	dialectos, quoting, tipos y valores ausentes requieren acuerdos

Formato	Ventajas	Limitaciones
Binario crudo	compacto, representación exacta conocida	sin metadatos; exige documentar tipo, orden y tamaño
Formato nativo	conserva estructuras y tipos del entorno	menor interoperabilidad
Imagen/PDF	adecuado para presentar resultados	no es una representación conveniente para reprocesar los datos originales

Como regla general:

- use texto o CSV cuando la prioridad sea intercambio y transparencia;
- use un formato nativo cuando la prioridad sea recuperar rápidamente el estado dentro del mismo entorno;
- use binario crudo sólo cuando el formato esté perfectamente definido;
- conserve los datos originales además de los informes y gráficos derivados.

6.1.19. Resumen del flujo de trabajo

Los mecanismos estudiados pueden verse como capas sucesivas:

entrada estándar y pantalla
↓
archivos de texto
↓
archivos tabulares y CSV
↓
persistencia nativa y binarios
↓
redirecciones, tuberías y procesos externos
↓
archivos virtuales y dispositivos
↓
integración de lectura, análisis, gráficos e informes

Los cinco lenguajes implementan estas capas con interfaces diferentes, pero el modelo conceptual es el mismo. Dominar ese modelo permite transferir lo aprendido de un lenguaje a otro y elegir el mecanismo de E/S adecuado para cada problema.

6.2. Ejercicios

6.2.1. Entrada y salida interactiva

1. **Ejercicio:** Carga matricial elemento por elemento

   Escriba un programa que permita ingresar desde la entrada estándar una matriz A de m filas y n columnas, elemento por elemento.

   a) solicite y valide m y n como enteros positivos;

   b) reserve la matriz completa antes de la carga cuando el entorno utilice una representación matricial que lo permita;

   c) recorra filas y columnas mediante dos ciclos anidados;

   d) informe qué elemento $A(i, j)$ se está ingresando;

   e) repita la solicitud cuando la entrada no represente un número finito;

   f) muestre la matriz al finalizar;

   g) calcule la suma de cada fila y cada columna;

   h) determine el valor y la posición del primer máximo encontrado;

   i) escriba una segunda versión que permita ingresar los mn elementos en una única línea, separados por espacios, validando la cantidad de datos;

   j) compare ambos métodos de entrada.

   Scilab. Puede preasignar con `zeros(m,n)`. Lea inicialmente como cadena con `input(...,"string")` y convierta sólo después de validar.

```
function x = leer_real(msg)
    while %t
        s = stripblanks(input(msg, "string"));
        if s <> "" & isnum(s) then
            x = strtod(s);
            if ~isnan(x) & ~isinf(x) then
                return
            end
        end
        disp("Entrada invalida: ingrese un numero finito.")
    end
endfunction

function n = leer_entero_positivo(msg)
    while %t
        x = leer_real(msg);
        if x > 0 & x == floor(x) then
            n = x;
            return
        end
        disp("Debe ser un entero positivo.")
    end
```

```
endfunction

m = leer_entero_positivo("Numero de filas m: ");
n = leer_entero_positivo("Numero de columnas n: ");

A = zeros(m,n);
for i = 1:m
    for j = 1:n
        A(i,j) = leer_real(msprintf("A(%d,%d) = ",i,j));
    end
end

disp("Matriz A:");
disp(A);
disp("Suma de cada fila:");
disp(sum(A,2));
disp("Suma de cada columna:");
disp(sum(A,1));

[mayor,pos] = max(A);
mprintf("Primer maximo: A(%d,%d) = %g\n", pos(1), pos(2),
   mayor);

// Segunda forma: todos los datos en una linea, por filas.
while %t
    s = stripblanks(input(msprintf( ..
        "Ingrese %d valores separados por espacios:
           ",m*n),"string"));
    partes = tokens(s," ");
    if size(partes,"*") <> m*n then
        disp("Cantidad incorrecta de elementos.");
        continue
    end

    valores = zeros(1,m*n);
    ok = %t;
    for k = 1:m*n
        if ~isnum(partes(k)) then
            ok = %f;
            break
        end
        valores(k) = strtod(partes(k));
        if isnan(valores(k)) | isinf(valores(k)) then
            ok = %f;
            break
        end
    end
    if ok then break, end
    disp("Todos los elementos deben ser numeros finitos.")
end
```

```
// matrix llena por columnas; el traspuesto conserva el
   orden por filas.
A2 = matrix(valores,n,m)';
disp("Matriz ingresada en una sola linea:");
disp(A2);
```

Octave. Puede preasignar con `zeros(m,n)`. Use entrada como texto y `str2double` para realizar una conversión controlada.

```
while true
  s=input("Numero de filas m: ","s"); m=str2double(s);
  if isfinite(m) && m==fix(m) && m>0, break; endif
  disp("m debe ser un entero positivo")
endwhile
while true
  s=input("Numero de columnas n: ","s"); n=str2double(s);
  if isfinite(n) && n==fix(n) && n>0, break; endif
  disp("n debe ser un entero positivo")
endwhile
A=zeros(m,n);
for i=1:m
  for j=1:n
    while true
      s=input(sprintf("A(%d,%d) = ",i,j),"s");
         x=str2double(s);
      if isfinite(x), A(i,j)=x; break; endif
      disp("Entrada invalida")
    endwhile
  endfor
endfor
disp(A)
disp("Suma por filas:"); disp(sum(A,2))
disp("Suma por columnas:"); disp(sum(A,1))
[mayor,k]=max(A(:)); [fi,co]=ind2sub(size(A),k);
printf("Primer maximo: A(%d,%d) = %g\n",fi,co,mayor)
while true
  s=input(sprintf("Ingrese %d valores: ",m*n),"s");
  v=sscanf(s,"%f")';
  if numel(v)==m*n && all(isfinite(v)), break; endif
  disp("Cantidad o valores invalidos")
endwhile
A2=reshape(v,n,m)'; disp(A2);
```

Python. Puede usar una lista de listas o una matriz de NumPy. Con NumPy, `zeros`, `sum`, `argmax` y `reshape` simplifican la implementación.

```
import math
import numpy as np

def leer_entero_positivo(msg):
    while True:
```

```
        try:
            n = int(input(msg).strip())
            if n > 0:
                return n
        except ValueError:
            pass
        print("Debe ser un entero positivo")

def leer_real(msg):
    while True:
        try:
            x = float(input(msg).strip())
            if math.isfinite(x):
                return x
        except ValueError:
            pass
        print("Debe ser un numero finito")

m = leer_entero_positivo("Numero de filas m: ")
n = leer_entero_positivo("Numero de columnas n: ")
A = np.zeros((m, n))
for i in range(m):
    for j in range(n):
        A[i, j] = leer_real(f"A({i+1},{j+1}) = ")

print(A)
print("Suma por filas:", A.sum(axis=1))
print("Suma por columnas:", A.sum(axis=0))
k = int(np.argmax(A))
i, j = np.unravel_index(k, A.shape)
print(f"Primer maximo: A({i+1},{j+1}) = {A[i,j]}")

while True:
    try:
        v = np.array([float(x) for x in input(f"Ingrese
           {m*n} valores: ").split()])
        if v.size == m*n and np.all(np.isfinite(v)):
            break
    except ValueError:
        pass
    print("Cantidad o valores invalidos")
A2 = v.reshape(m, n)
print(A2)
```

Perl. Represente la matriz mediante un arreglo de referencias a arreglos. Para validar números puede emplear `Scalar::Util::looks_like_number`.

```
use strict; use warnings;
use Scalar::Util qw(looks_like_number);
use POSIX qw(isfinite);
sub posint { my($m)=@_; while(1){ print $m; chomp(my
```

```
    $s=<STDIN>); return int($s) if
    looks_like_number($s)&&isfinite($s)&&$s>0&&$s==int($s);
    print "Debe ser entero positivo\n" } }
sub real { my($m)=@_; while(1){ print $m; chomp(my
    $s=<STDIN>); return 0+$s if
    looks_like_number($s)&&isfinite($s); print "Debe ser
    numero finito\n" } }
my $m=posint("Numero de filas m: "); my $n=posint("Numero de
    columnas n: ");
my @A; for my $i (0..$m-1){ for my $j (0..$n-1){
    $A[$i][$j]=real(sprintf("A(%d,%d) = ",$i+1,$j+1)) } }
for my $r (@A){ print join(" ",@$r),"\n" }
for my $i (0..$m-1){ my $s=0; $s+=$_ for @{$A[$i]}; print
    "Suma fila ",$i+1,": $s\n" }
for my $j (0..$n-1){ my $s=0; $s+=$A[$_][$j] for 0..$m-1;
    print "Suma columna ",$j+1,": $s\n" }
my($max,$fi,$co)=($A[0][0],0,0); for my $i (0..$m-1){ for my
    $j (0..$n-1){
    if($A[$i][$j]>$max){($max,$fi,$co)=($A[$i][$j],$i,$j)} } }
printf "Primer maximo: A(%d,%d) = %g\n",$fi+1,$co+1,$max;
while(1){ print "Ingrese ",$m*$n," valores: "; chomp(my
    $s=<STDIN>); my @v=split /\s+/,$s; next unless @v==$m*$n
    && !grep {!looks_like_number($_)||!isfinite($_)} @v; my
    @B; for my $i(0..$m-1){ $B[$i]=[map {0+$_}
    @v[$i*$n..$i*$n+$n-1]] } print join(" ",@$_),"\n" for @B;
    last }
```

JavaScript. En Node.js represente la matriz como un arreglo de arreglos. Para la entrada puede emplear `readline` o el mecanismo síncrono configurado en el curso; valide con `Number.isFinite`.

```
"use strict";
const readline=require("node:readline/promises"); const
    {stdin,stdout}=require("node:process"); const
    rl=readline.createInterface({input:stdin,output:stdout});
async function posint(msg){for(;;){const n=Number((await
    rl.question(msg)).trim());if(Number.isInteger(n)&&n>0)return
    n;console.log("Debe ser entero positivo")}}
async function real(msg){for(;;){const x=Number((await
    rl.question(msg)).trim());if(Number.isFinite(x))return
    x;console.log("Debe ser numero finito")}}
async function main(){const m=await posint("Numero de filas
    m: "),n=await posint("Numero de columnas n: ");const
    A=Array.from({length:m},()=>Array(n).fill(0));for(let
    i=0;i<m;i++)for(let j=0;j<n;j++)A[i][j]=await
    real(`A(${i+1},${j+1}) =
    `);console.table(A);console.log("Suma
    filas:",A.map(r=>r.reduce((a,b)=>a+b,0)));console.log("Suma
    columnas:",Array.from({length:n},(_,j)=>A.reduce((s,r)=>s+r[j],0)));l
    max=A[0][0],fi=0,co=0;for(let i=0;i<m;i++)for(let
    j=0;j<n;j++)if(A[i][j]>max){max=A[i][j];fi=i;co=j}console.log(`Primer
```

```
    maximo: A(${fi+1},${co+1}) = ${max}`);for(;;){const
    v=(await rl.question(`Ingrese ${m*n} valores:
    `)).trim().split(/\s+/).map(Number);if(v.length===m*n&&v.every(Number
    B=Array.from({length:m},(_,i)=>v.slice(i*n,(i+1)*n));console.table(B)
    o valores invalidos")}rl.close()}
main().catch(e=>{console.error(e);process.exitCode=1});
```

2. **Ejercicio:** Calificaciones con ingreso dinámico

   Lea desde la entrada estándar una serie de notas. Cada nota válida debe ser un número real comprendido entre 0 y 10. La entrada termina cuando el usuario introduce un valor negativo.

   Almacene las notas aceptadas en una colección que pueda crecer durante la lectura y, al finalizar, muestre:

   - cantidad de notas ingresadas;
   - promedio;
   - nota máxima y mínima;
   - cantidad de notas mayores o iguales que 4.

   Organice el programa en funciones separadas para entrada, procesamiento y presentación. Si no se ingresó ninguna nota válida, no intente calcular el promedio.

   Scilab. Puede leer cada valor como texto y convertirlo después de validarlo; los vectores de Scilab pueden ampliarse durante la carga.

```
function notas = leer_notas()
    notas = [];
    while %t
        s = stripblanks(input( ..
            "Nota [0..10], negativa para terminar:
               ","string"));
        if s == "" | ~isnum(s) then
            disp("Entrada no numerica.");
            continue
        end
        x = strtod(s);
        if isnan(x) | isinf(x) then
            disp("La nota debe ser finita.");
        elseif x < 0 then
            return
        elseif x > 10 then
            disp("La nota debe estar entre 0 y 10.");
        else
            notas = [notas x];
        end
    end
endfunction
```

```
function [n,promedio,minimo,maximo,aprobadas] =
   procesar(notas)
    n = size(notas,"*");
    if n == 0 then
        promedio = %nan;
        minimo = %nan;
        maximo = %nan;
        aprobadas = 0;
        return
    end
    promedio = mean(notas);
    minimo = min(notas);
    maximo = max(notas);
    aprobadas = sum(notas >= 4);
endfunction

function presentar(n,promedio,minimo,maximo,aprobadas)
    mprintf("Cantidad de notas: %d\n",n);
    if n == 0 then
        disp("No se ingresaron notas validas.");
        return
    end
    mprintf("Promedio: %.2f\n",promedio);
    mprintf("Minima: %.2f\n",minimo);
    mprintf("Maxima: %.2f\n",maximo);
    mprintf("Notas >= 4: %d\n",aprobadas);
endfunction

notas = leer_notas();
[n,promedio,minimo,maximo,aprobadas] = procesar(notas);
presentar(n,promedio,minimo,maximo,aprobadas);
```

Octave. Use entrada textual y conversión controlada; un vector puede ampliarse agregando nuevos elementos.

```
notas=leer_notas();
[n,prom,minimo,maximo,aprobadas]=procesar(notas);
presentar(n,prom,minimo,maximo,aprobadas);

function notas=leer_notas()
  notas=[];
  while true
    s=input("Nota [0..10], negativa termina: ","s");
       x=str2double(s);
    if !isfinite(x), disp("Entrada invalida"); continue;
       endif
    if x<0, return; endif
    if x>10, disp("Fuera de rango"); continue; endif
    notas(end+1)=x;
  endwhile
endfunction
```

```
function [n,prom,minimo,maximo,aprobadas]=procesar(notas)
  n=numel(notas); aprobadas=sum(notas>=4);
  if n==0, prom=NaN; minimo=NaN; maximo=NaN; return; endif
  prom=mean(notas); minimo=min(notas); maximo=max(notas);
endfunction

function presentar(n,prom,minimo,maximo,aprobadas)
  printf("Cantidad: %d\n",n);
  if n==0, disp("No hay notas validas"); return; endif
  printf("Promedio: %.2f\nMinima: %.2f\nMaxima:
     %.2f\nAprobadas: %d\n", ...
         prom,minimo,maximo,aprobadas);
endfunction
```

Python. Una lista es apropiada para la colección dinámica; convierta la entrada con `float` dentro de un manejo explícito de errores.

```
import math

def leer_notas():
    notas = []
    while True:
        try:
            x = float(input("Nota [0..10], negativa termina:
               "))
        except ValueError:
            print("Entrada invalida"); continue
        if not math.isfinite(x):
            print("La nota debe ser finita"); continue
        if x < 0:
            return notas
        if x > 10:
            print("Fuera de rango"); continue
        notas.append(x)

def procesar(notas):
    if not notas:
        return {"n": 0, "aprobadas": 0}
    return {"n": len(notas), "promedio":
       sum(notas)/len(notas),
            "minimo": min(notas), "maximo": max(notas),
            "aprobadas": sum(x >= 4 for x in notas)}

def presentar(s):
    print("Cantidad:", s["n"])
    if s["n"] == 0:
        print("No hay notas validas"); return
    print(f"Promedio: {s['promedio']:.2f}")
    print(f"Minima: {s['minimo']:.2f}")
    print(f"Maxima: {s['maximo']:.2f}")
```

```
    print("Aprobadas:", s["aprobadas"])

presentar(procesar(leer_notas()))
```

Perl. Use un arreglo y `push`; lea desde `STDIN` y valide antes de convertir.

```
use strict; use warnings;
use Scalar::Util qw(looks_like_number); use POSIX
   qw(isfinite);
sub leer_notas { my @n; while(1){ print "Nota [0..10],
   negativa termina: "; chomp(my $s=<STDIN>);
   if(!looks_like_number($s)||!isfinite($s)){warn "Entrada
   invalida\n";next} my $x=0+$s; last if $x<0;
   if($x>10){warn "Fuera de rango\n";next} push @n,$x }
   return \@n }
sub procesar { my($a)=@_; my $n=@$a; return
   {n=>0,aprobadas=>0} unless $n;
   my($s,$min,$max,$ap)=(0,$a->[0],$a->[0],0);
   for(@$a){$s+=$_;$min=$_ if $_<$min;$max=$_ if
   $_>$max;$ap++ if $_>=4} return
   {n=>$n,promedio=>$s/$n,minimo=>$min,maximo=>$max,aprobadas=>$ap}
   }
sub presentar { my($r)=@_; print "Cantidad: $r->{n}\n";
   return print "No hay notas validas\n" unless $r->{n};
   printf "Promedio: %.2f\nMinima: %.2f\nMaxima:
   %.2f\nAprobadas: %d\n",@$r{qw(promedio minimo maximo
   aprobadas)} }
presentar(procesar(leer_notas()));
```

JavaScript. Use un arreglo y `push`; convierta con `Number` y valide con `Number.isFinite`.

```
"use strict";
const
   readline=require("node:readline/promises");const{stdin,stdout}=requir
   rl=readline.createInterface({input:stdin,output:stdout});
async function leerNotas(){const a=[];for(;;){const
   x=Number((await rl.question("Nota [0..10], negativa
   termina:
   ")).trim());if(!Number.isFinite(x)){console.log("Entrada
   invalida");continue}if(x<0)return
   a;if(x>10){console.log("Fuera de
   rango");continue}a.push(x)}}
function
   procesar(a){if(!a.length)return{n:0,aprobadas:0};return{n:a.length,pr
(async()=>{const s=procesar(await
   leerNotas());console.log("Cantidad:",s.n);if(s.n)console.log(`Promedi
   ${s.promedio.toFixed(2)}\nMinima:
   ${s.minimo.toFixed(2)}\nMaxima:
   ${s.maximo.toFixed(2)}\nAprobadas: ${s.aprobadas}`);else
   console.log("No hay notas
```

```
    validas");rl.close()})().catch(e=>{console.error(e);process.exitCode=
```

3. **Ejercicio:** Separación de pares e impares

   Ingrese diez números enteros desde la entrada estándar. Valide cada dato antes de almacenarlo.

   Calcule:

   - el promedio de los elementos pares;
   - la suma de los elementos impares.

   Si no existen elementos pares, indíquelo en lugar de efectuar una división por cero. Muestre ambos resultados por la salida estándar.

   Scilab. Puede usar `mod(x,2)` para determinar la paridad.

```
function x = leer_entero(msg)
    while %t
        s = stripblanks(input(msg,"string"));
        if s <> "" & isnum(s) then
            y = strtod(s);
            if ~isnan(y) & ~isinf(y) & y == floor(y) then
                x = y;
                return
            end
        end
        disp("Debe ingresar un numero entero.")
    end
endfunction

v = zeros(1,10);
for i = 1:10
    v(i) = leer_entero(msprintf("Entero %d: ",i));
end

es_par = pmodulo(v,2) == 0;
pares = v(find(es_par));
impares = v(find(~es_par));

if size(pares,"*") == 0 then
    disp("No hay elementos pares.");
else
    mprintf("Promedio de pares: %.6f\n",mean(pares));
end
mprintf("Suma de impares: %g\n",sum(impares));
```

   Octave. Puede usar `mod(x,2)` para determinar la paridad.

```
v=zeros(1,10);
for k=1:10
```

```
  while true
    s=input(sprintf("Entero %d: ",k),"s"); x=str2double(s);
    if isfinite(x) && x==fix(x), v(k)=x; break; endif
    disp("Debe ser entero")
  endwhile
endfor
pares=v(mod(v,2)==0); impares=v(mod(v,2)!=0);
if isempty(pares), disp("No hay pares"); else
   printf("Promedio pares: %g\n",mean(pares)); endif
printf("Suma impares: %g\n",sum(impares));
```

Python. Puede usar el operador % y una lista para almacenar los datos.

```
v = []
while len(v) < 10:
    try:
        x = int(input(f"Entero {len(v)+1}: ").strip())
        v.append(x)
    except ValueError:
        print("Debe ser entero")
pares = [x for x in v if x % 2 == 0]
impares = [x for x in v if x % 2 != 0]
if pares:
    print("Promedio pares:", sum(pares)/len(pares))
else:
    print("No hay pares")
print("Suma impares:", sum(impares))
```

Perl. Puede usar el operador % sobre cada entero validado.

```
use strict; use warnings;
my @v; while(@v<10){ print "Entero ",@v+1,": "; chomp(my
   $s=<STDIN>); if($s =~ /^[-+]?\d+$/){push
   @v,0+$s}else{warn "Debe ser entero\n"} }
my @p=grep {$_%2==0} @v; my @i=grep {$_%2!=0} @v;
if(@p){my $s=0;$s+=$_ for @p; print "Promedio pares:
   ",$s/@p,"\n"}else{print "No hay pares\n"}
my $si=0;$si+=$_ for @i; print "Suma impares: $si\n";
```

JavaScript. Puede usar el operador %; compruebe además que cada dato sea `Number.isInteger`.

```
"use strict";
const
   readline=require("node:readline/promises");const{stdin,stdout}=requir
   rl=readline.createInterface({input:stdin,output:stdout});
(async()=>{const v=[];while(v.length<10){const s=(await
   rl.question(`Entero ${v.length+1}: `)).trim();const
   x=Number(s);if(Number.isInteger(x))v.push(x);else
   console.log("Debe ser entero")}const
   p=v.filter(x=>x%2===0),i=v.filter(x=>x%2!==0);console.log(p.length?`P
```

```
    pares: ${p.reduce((a,b)=>a+b,0)/p.length}`:"No hay
    pares");console.log("Suma
    impares:",i.reduce((a,b)=>a+b,0));rl.close()})().catch(console.error)
```

4. **Ejercicio:** Diagonal de una matriz cuadrada

   Ingrese m, n y los elementos de una matriz de m filas y n columnas. Valide las dimensiones y los elementos.

   Si $m = n$, calcule la suma de los elementos de la diagonal principal y muestre el resultado. En caso contrario, presente un mensaje indicando que la operación requiere una matriz cuadrada.

   Scilab. Puede almacenar la matriz con `zeros(m,n)` y acceder a `A(i,i)`.

```
function n = leer_dimension(msg)
    while %t
        s = stripblanks(input(msg,"string"));
        if s <> "" & isnum(s) then
            x = strtod(s);
            if x > 0 & x == floor(x) then n = x; return, end
        end
        disp("La dimension debe ser un entero positivo.")
    end
endfunction

function x = leer_real(msg)
    while %t
        s = stripblanks(input(msg,"string"));
        if s <> "" & isnum(s) then
            x = strtod(s);
            if ~isnan(x) & ~isinf(x) then return, end
        end
        disp("Valor numerico invalido.")
    end
endfunction

m = leer_dimension("m: ");
n = leer_dimension("n: ");
A = zeros(m,n);
for i = 1:m
    for j = 1:n
        A(i,j) = leer_real(msprintf("A(%d,%d): ",i,j));
    end
end

if m <> n then
    disp("La suma diagonal requiere una matriz cuadrada.");
else
    suma = 0;
    for i = 1:n
        suma = suma + A(i,i);
```

```
    end
    mprintf("Suma de la diagonal principal: %g\n",suma);
end
```

Octave. Puede almacenar la matriz con `zeros(m,n)` y usar `diag` o un recorrido explícito.

```
m=round(str2double(input("m: ","s")));
n=round(str2double(input("n: ","s")));
if m<=0 || n<=0, error("Dimensiones invalidas"); endif
A=zeros(m,n);
for i=1:m
  for j=1:n
    x=str2double(input(sprintf("A(%d,%d): ",i,j),"s"));
    if !isfinite(x), error("Elemento no finito"); endif
    A(i,j)=x;
  endfor
endfor
if m==n
  printf("Suma diagonal: %g\n",sum(diag(A)));
else
  disp("La matriz debe ser cuadrada");
endif
```

Python. Con NumPy puede usar `trace`; con listas, recorra explícitamente `A[i][i]`.

```
import math
m = int(input("m: ")); n = int(input("n: "))
if m <= 0 or n <= 0:
    raise ValueError("Dimensiones invalidas")
A = []
for i in range(m):
    fila = []
    for j in range(n):
        x = float(input(f"A({i+1},{j+1}): "))
        if not math.isfinite(x):
            raise ValueError("Elemento no finito")
        fila.append(x)
    A.append(fila)
if m == n:
    print("Suma diagonal:", sum(A[i][i] for i in range(n)))
else:
    print("La matriz debe ser cuadrada")
```

Perl. Use un arreglo de arreglos y recorra manualmente la diagonal.

```
use strict; use warnings; use Scalar::Util
   qw(looks_like_number);
print "m: "; chomp(my $m=<STDIN>); print "n: "; chomp(my
   $n=<STDIN>); die "Dimensiones invalidas\n" unless
```

```
    $m=~/^\d+$/&&$n=~/^\d+$/&&$m>0&&$n>0;
my @A; for my $i(0..$m-1){for my $j(0..$n-1){print
    "A(",$i+1,",",$j+1,"): "; chomp(my $x=<STDIN>); die "Dato
    invalido\n" unless looks_like_number($x);
    $A[$i][$j]=0+$x}}
if($m==$n){my $s=0;$s+=$A[$_][$_] for 0..$n-1;print "Suma
    diagonal: $s\n"}else{print "La matriz debe ser
    cuadrada\n"}
```

JavaScript. Use un arreglo de arreglos y recorra manualmente la diagonal.

```
"use strict";
const
    readline=require("node:readline/promises");const{stdin,stdout}=requir
    rl=readline.createInterface({input:stdin,output:stdout});
(async()=>{const m=Number(await rl.question("m:
    ")),n=Number(await rl.question("n:
    "));if(!Number.isInteger(m)||!Number.isInteger(n)||m<=0||n<=0)throw
    new Error("Dimensiones invalidas");const
    A=Array.from({length:m},()=>Array(n));for(let
    i=0;i<m;i++)for(let j=0;j<n;j++){const x=Number(await
    rl.question(`A(${i+1},${j+1}):
    `));if(!Number.isFinite(x))throw new Error("Elemento no
    finito");A[i][j]=x}console.log(m===n?`Suma diagonal:
    ${A.reduce((s,r,i)=>s+r[i],0)}`:"La matriz debe ser
    cuadrada");rl.close()})().catch(e=>{console.error(e.message);process.
```

5. **Ejercicio:** Norma y extremos de un vector

   Ingrese un entero positivo n y luego un vector real de dimensión n. Valide que se reciban exactamente n valores finitos.

   a) Calcule la norma euclídea

   $$\|v\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}.$$

   b) Obtenga el valor mínimo y el máximo del vector.

   c) Muestre los tres resultados con mensajes descriptivos.

   Cuando el entorno disponga de operaciones numéricas equivalentes, compare el resultado obtenido mediante recorrido explícito con el obtenido mediante las funciones de biblioteca.

   Scilab. Puede comparar el cálculo explícito con `norm`, `min` y `max`.

```
function n = leer_n()
    while %t
        s = stripblanks(input("Dimension n: ","string"));
        if s <> "" & isnum(s) then
            x = strtod(s);
```

```
            if x > 0 & x == floor(x) then n = x; return, end
        end
        disp("n debe ser entero positivo.")
    end
endfunction

n = leer_n();
while %t
    s = stripblanks(input(msprintf( ..
        "Ingrese %d valores separados por espacios:
           ",n),"string"));
    p = tokens(s," ");
    if size(p,"*") <> n then
        disp("Cantidad incorrecta de valores.");
        continue
    end
    v = zeros(1,n);
    ok = %t;
    for i = 1:n
        if ~isnum(p(i)) then ok=%f; break, end
        v(i) = strtod(p(i));
        if isnan(v(i)) | isinf(v(i)) then ok=%f; break, end
    end
    if ok then break, end
    disp("Todos los elementos deben ser numeros finitos.")
end

norma_explicita = sqrt(sum(v.^2));
minimo = min(v);
maximo = max(v);

mprintf("Norma explicita: %.12g\n",norma_explicita);
mprintf("Norma con norm(): %.12g\n",norm(v,2));
mprintf("Minimo: %.12g\n",minimo);
mprintf("Maximo: %.12g\n",maximo);
```

Octave. Puede comparar el cálculo explícito con `norm`, `min` y `max`.

```
n=str2double(input("Dimension n: ","s"));
if !isfinite(n) || n<=0 || n!=fix(n), error("n invalido");
   endif
while true
  s=input(sprintf("Ingrese %d valores: ",n),"s");
     v=sscanf(s,"%f")';
  if numel(v)==n && all(isfinite(v)), break; endif
  disp("Entrada invalida")
endwhile
norma=sqrt(sum(v.^2));
printf("Norma: %.8g\nMinimo: %.8g\nMaximo:
   %.8g\n",norma,min(v),max(v));
printf("norm(v) = %.8g\n",norm(v));
```

Python. Con NumPy puede usar `numpy.linalg.norm`, `min` y `max`.

```
import math
import numpy as np
n = int(input("Dimension n: "))
if n <= 0:
    raise ValueError("n invalido")
while True:
    try:
        v = np.array([float(x) for x in input(f"Ingrese {n}
            valores: ").split()])
        if v.size == n and np.all(np.isfinite(v)):
            break
    except ValueError:
        pass
    print("Entrada invalida")
norma = math.sqrt(sum(float(x*x) for x in v))
print("Norma:", norma)
print("Minimo:", float(v.min()))
print("Maximo:", float(v.max()))
print("numpy.linalg.norm:", float(np.linalg.norm(v)))
```

Perl. Con arreglos básicos realice el cálculo explícito; si se dispone de una biblioteca numérica puede comparar ambos métodos.

```
use strict; use warnings; use Scalar::Util
   qw(looks_like_number); use List::Util qw(min max sum);
   use POSIX qw(isfinite);
print "Dimension n: "; chomp(my $n=<STDIN>); die "n
   invalido\n" unless $n=~/^\d+$/&&$n>0;
my @v; while(1){print "Ingrese $n valores: ";chomp(my
   $s=<STDIN>);@v=split /\s+/,$s;last if
   @v==$n&&!grep{!looks_like_number($_)||!isfinite($_)}@v;warn
   "Entrada invalida\n"}
@v=map{0+$_}@v; my $norm=sqrt(sum(map{$_*$_}@v)); print
   "Norma: $norm\nMinimo: ",min(@v),"\nMaximo:
   ",max(@v),"\n";
```

JavaScript. Con arreglos básicos use `reduce`, `Math.min` y `Math.max`, teniendo en cuenta el desempaquetado de los elementos.

```
"use strict";
const
   readline=require("node:readline/promises");const{stdin,stdout}=requir
   rl=readline.createInterface({input:stdin,output:stdout});
(async()=>{const n=Number(await rl.question("Dimension n:
   "));if(!Number.isInteger(n)||n<=0)throw new Error("n
   invalido");let v;for(;;){v=(await rl.question(`Ingrese
   ${n} valores:
   `)).trim().split(/\s+/).map(Number);if(v.length===n&&v.every(Number.i
   invalida")}const
   norma=Math.sqrt(v.reduce((s,x)=>s+x*x,0));console.log("Norma:",norma,
```

6. **Ejercicio:** Ensamble matricial y propiedades

   Ingrese dos matrices reales B y C con igual cantidad de filas y construya la matriz A concatenando sus columnas:

   $$A = [B\ C].$$

   Valide que las dimensiones sean compatibles. Si A resulta cuadrada, calcule, utilizando las operaciones numéricas disponibles en el entorno:

   a) la inversa, cuando exista;
   b) los valores y vectores propios;
   c) la traza;
   d) el determinante.

   Si alguna operación no puede realizarse, informe la causa en lugar de producir un resultado silenciosamente inválido.

   Scilab. Dispone de concatenación matricial y funciones como `inv`, `spec`, `trace` y `det`.

```
function A = leer_matriz(nombre,m,n)
    A = zeros(m,n);
    for i = 1:m
        for j = 1:n
            while %t
                s = stripblanks(input(msprintf( ..
                    "%s(%d,%d): ",nombre,i,j),"string"));
                if s <> "" & isnum(s) then
                    x = strtod(s);
                    if ~isnan(x) & ~isinf(x) then
                        A(i,j) = x;
                        break
                    end
                end
                disp("Valor invalido.")
            end
        end
    end
endfunction

m = input("Cantidad de filas comunes: ");
nb = input("Columnas de B: ");
nc = input("Columnas de C: ");
if m<=0 | nb<=0 | nc<=0 | m<>floor(m) | nb<>floor(nb) |
   nc<>floor(nc) then
    error("Las dimensiones deben ser enteros positivos.")
end
```

```
B = leer_matriz("B",m,nb);
C = leer_matriz("C",m,nc);
A = [B C];
disp("A = [B C]:");
disp(A);

if size(A,1) <> size(A,2) then
    disp("A no es cuadrada: no se calculan inversa,
       espectro, traza ni det.");
else
    d = det(A);
    mprintf("det(A) = %.12g\n",d);
    mprintf("trace(A) = %.12g\n",trace(A));
    [V,D] = spec(A);
    disp("Valores propios (diagonal de D):");
    disp(diag(D));
    disp("Vectores propios (columnas de V):");
    disp(V);
    if abs(d) <= 100*%eps*max(1,norm(A,1)) then
        disp("A es singular o numericamente casi singular;
           no se invierte.");
    else
        disp("inv(A):");
        disp(inv(A));
    end
end
```

Octave. Dispone de concatenación matricial y funciones como `inv`, `eig`, `trace` y `det`.

```
B=input("B = "); C=input("C = ");
if !isnumeric(B) || !isnumeric(C) || rows(B)!=rows(C)
  error("B y C deben tener igual cantidad de filas");
endif
A=[B C]; disp("A ="); disp(A)
if rows(A)!=columns(A), disp("A no es cuadrada"); return;
   endif
tr=trace(A); d=det(A); [V,D]=eig(A);
printf("Traza = %g\nDeterminante = %g\n",tr,d);
disp("Valores propios:"); disp(diag(D));
disp("Vectores propios:"); disp(V)
if abs(d) > 100*eps*norm(A,1), disp("Inversa:");
   disp(inv(A));
else, disp("A es singular o casi singular"); endif
```

Python. Use NumPy; `concatenate` o `hstack` y las rutinas de `numpy.linalg`.

```
import ast
import numpy as np
B = np.array(ast.literal_eval(input("B, por ejemplo
   [[1,2],[3,4]]: ")), dtype=float)
```

```
C = np.array(ast.literal_eval(input("C: ")), dtype=float)
if B.ndim != 2 or C.ndim != 2 or B.shape[0] != C.shape[0]:
    raise ValueError("Dimensiones incompatibles")
A = np.concatenate((B, C), axis=1)
print("A =\n", A)
if A.shape[0] != A.shape[1]:
    print("A no es cuadrada")
else:
    d = float(np.linalg.det(A))
    valores, vectores = np.linalg.eig(A)
    print("Traza:", float(np.trace(A)))
    print("Determinante:", d)
    print("Valores propios:\n", valores)
    print("Vectores propios:\n", vectores)
    if abs(d) > 100*np.finfo(float).eps*np.linalg.norm(A, 1):
        print("Inversa:\n", np.linalg.inv(A))
    else:
        print("A es singular o casi singular")
```

Perl. Los arreglos básicos no incluyen álgebra lineal. Utilice la biblioteca numérica configurada para el curso, si está disponible, y documente qué operaciones proporciona.

```
use strict; use warnings; use PDL; use PDL::MatrixOps
   qw(eigens);
sub leer { my($nom)=@_; print "Filas y columnas de $nom: ";
   chomp(my $s=<STDIN>); my($m,$n)=split /\s+/,$s; die
   "Dimensiones invalidas\n" unless $m>0&&$n>0; my @M; for
   my $i(0..$m-1){print "$nom fila ",$i+1,": "; chomp(my
   $l=<STDIN>); my @r=split /\s+/,$l; die "Fila invalida\n"
   unless @r==$n; $M[$i]=[map{0+$_}@r]} return \@M }
my $B=leer("B"); my $C=leer("C"); die "Filas
   incompatibles\n" unless @$B==@$C;
my @Ar; for my
   $i(0..$#$B){$Ar[$i]=[@{$B->[$i]},@{$C->[$i]}]}; my
   $A=pdl(\@Ar); print "A=\n$A\n";
my($fil,$col)=@{$A->dims}[1,0]; if($fil!=$col){print "A no
   es cuadrada\n";exit}
my $tr=0;$tr+=$Ar[$_][$_] for 0..$fil-1; my $det=$A->det;
   print "Traza=$tr\nDeterminante=$det\n";
my($evec,$eval)=eigens($A); print "Valores
   propios=\n$eval\nVectores propios=\n$evec\n";
my $inv=eval{$A->inv}; print $inv ? "Inversa=\n$inv\n" : "A
   es singular\n";
```

JavaScript. Los arreglos básicos no incluyen álgebra lineal. Utilice la biblioteca numérica configurada para el curso, si está disponible, y documente qué operaciones proporciona.

```
"use strict";
const math=require("mathjs");const
```

```
    readline=require("node:readline/promises");const{stdin,stdout}=requir
    rl=readline.createInterface({input:stdin,output:stdout});
(async()=>{const B=JSON.parse(await rl.question("B como
    [[...],[...]]: "));const C=JSON.parse(await
    rl.question("C:
    "));if(!Array.isArray(B)||!Array.isArray(C)||B.length!==C.length)thro
    new Error("Filas incompatibles");const
    A=B.map((r,i)=>[...r,...C[i]]);console.log("A=",A);if(A.length!==A[0]
    no es cuadrada");rl.close();return}const
    d=math.det(A);const
    e=math.eigs(A);console.log("Traza:",math.trace(A));console.log("Deter
    propios:",e.values);console.log("Vectores
    propios:",e.eigenvectors);if(Math.abs(d)>1e-12)console.log("Inversa:"
    console.log("A es
    singular");rl.close()})().catch(e=>{console.error(e);process.exitCode
```

7. **Ejercicio:** Polinomio con raíces y gráfico

   Ingrese un entero no negativo n y los $n+1$ coeficientes reales de un polinomio

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

   con $a_n \neq 0$.

   a) valide el orden y los coeficientes;

   b) calcule las raíces utilizando la rutina numérica disponible en el entorno;

   c) evalúe el polinomio en un intervalo que permita apreciar su comportamiento;

   d) genere un gráfico de $p(x)$;

   e) muestre las raíces encontradas.

   Scilab. Puede usar las representaciones polinomiales y `roots`; para el gráfico use las herramientas gráficas de Scilab.

```
function n = leer_grado()
    while %t
        s = stripblanks(input("Grado n: ","string"));
        if s<>"" & isnum(s) then
            x=strtod(s);
            if x>=0 & x==floor(x) then n=x; return, end
        end
        disp("El grado debe ser un entero no negativo.")
    end
endfunction

n = leer_grado();
coef_desc = zeros(1,n+1);
for k = 1:n+1
    potencia = n-k+1;
    while %t
```

```
        s = stripblanks(input(msprintf("Coeficiente de x^%d:
           ",potencia), ..
                                "string"));
        if s<>"" & isnum(s) then
            x=strtod(s);
            if ~isnan(x) & ~isinf(x) then
                coef_desc(k)=x;
                break
            end
        end
        disp("Coeficiente invalido.")
    end
end
if coef_desc(1)==0 then error("El coeficiente principal no
   puede ser cero."), end

// poly(...,"coeff") recibe coeficientes en orden creciente.
coef_asc = coef_desc($:-1:1);
p = poly(coef_asc,"x","coeff");
r = roots(p);
disp("Raices:");
disp(r);

x = linspace(-10,10,800);
y = horner(p,x);
scf();
plot(x,y);
xgrid();
xtitle("p(x)","x","p(x)");
```

Octave. Puede usar `roots`, `polyval` y las herramientas gráficas de Octave.

```
n=str2double(input("Grado n: ","s"));
if !isfinite(n) || n<0 || n!=fix(n), error("Grado
   invalido"); endif
while true
  c=sscanf(input(sprintf("Ingrese %d coeficientes:
     ",n+1),"s"),"%f")';
  if numel(c)==n+1 && all(isfinite(c)) && c(1)!=0, break;
     endif
  disp("Coeficientes invalidos")
endwhile
r=roots(c); disp("Raices:"); disp(r)
x=linspace(-10,10,1000); y=polyval(c,x);
plot(x,y,"b-",[x(1) x(end)],[0 0],"k--"); grid on
xlabel("x"); ylabel("p(x)"); title("Polinomio");
print("polinomio.png","-dpng");
```

Python. Con NumPy puede usar `numpy.roots` y `polyval`; para el gráfico use la biblioteca gráfica configurada en el curso.

```
import numpy as np
import matplotlib.pyplot as plt
n = int(input("Grado n: "))
if n < 0:
    raise ValueError("Grado invalido")
while True:
    try:
        c = np.array([float(x) for x in input(f"Ingrese
           {n+1} coeficientes: ").split()])
        if c.size == n+1 and np.all(np.isfinite(c)) and c[0]
           != 0:
            break
    except ValueError:
        pass
    print("Coeficientes invalidos")
raices = np.roots(c)
print("Raices:", raices)
x = np.linspace(-10, 10, 1000)
y = np.polyval(c, x)
plt.plot(x, y); plt.axhline(0, color="black", linewidth=0.8)
plt.grid(); plt.xlabel("x"); plt.ylabel("p(x)")
plt.savefig("polinomio.png"); plt.close()
```

Perl. Use la biblioteca numérica y gráfica configurada para el curso; los arreglos básicos de Perl no proporcionan cálculo de raíces polinomiales.

```
use strict; use warnings; use PDL; use PDL::Math
   qw(polyroots); use POSIX qw(isfinite);
print "Grado n: "; chomp(my $n=<STDIN>); die "Grado
   invalido\n" unless $n=~/^\d+$/;
print "Ingrese ",$n+1," coeficientes: "; chomp(my
   $s=<STDIN>); my @c=split /\s+/,$s; die "Coeficientes
   invalidos\n" unless @c==$n+1&&$c[0]!=0;
my $roots=polyroots(pdl(\@c)); print "Raices:\n$roots\n";
sub horner{my($x,$c)=@_;my $y=0;$y=$y*$x+$_ for @$c;return
   $y}
open my $d,">","polinomio.dat" or die $!; for(my
   $x=-10;$x<=10;$x+=0.02){print $d "$x
   ",horner($x,\@c),"\n"} close $d;
open my $g,">","polinomio.gp" or die $!; print $g "set
   terminal pngcairo\nset output 'polinomio.png'\nset
   grid\nplot 'polinomio.dat' using 1:2 with lines title
   'p(x)', 0 notitle\n"; close $g;
   system("gnuplot","polinomio.gp")==0 or warn "No se pudo
   ejecutar gnuplot\n";
```

JavaScript. Use la biblioteca numérica y gráfica configurada para el curso; el núcleo de JavaScript no proporciona cálculo de raíces polinomiales.

```
"use strict";
const
```

```
    math=require("mathjs"),fs=require("node:fs"),{spawnSync}=require("nod
    rl=readline.createInterface({input:stdin,output:stdout});
function peval(c,z){let y=math.complex(0,0);for(const a of
    c)y=math.add(math.multiply(y,z),a);return y}
function roots(c){const
    n=c.length-1;if(n===1)return[math.complex(-c[1]/c[0],0)];c=c.map(x=>x
    r=Array.from({length:n},(_,k)=>math.complex({r:1.2,phi:2*Math.PI*k/n+
    it=0;it<500;it++){let md=0;const
    old=r;r=old.map((z,i)=>{let den=math.complex(1,0);for(let
    j=0;j<n;j++)if(i!==j)den=math.multiply(den,math.subtract(z,old[j]));c
    dz=math.divide(peval(c,z),den);md=Math.max(md,math.abs(dz));return
    math.subtract(z,dz)});if(md<1e-12)break}return r}
(async()=>{const n=Number(await rl.question("Grado n:
    "));const c=(await rl.question(`Ingrese ${n+1}
    coeficientes:
    `)).trim().split(/\s+/).map(Number);if(!Number.isInteger(n)||n<0||c.l
    new Error("Coeficientes
    invalidos");console.log("Raices:",roots(c).map(String));let
    data="";for(let x=-10;x<=10;x+=0.02)data+=`${x}
    ${Number(peval(c,math.complex(x,0)).re)}\n`;fs.writeFileSync("polinom
    terminal pngcairo\nset output 'polinomio.png'\nset
    grid\nplot 'polinomio.dat' w l, 0
    notitle\n");spawnSync("gnuplot",["polinomio.gp"],{stdio:"inherit"});r
```

8. **Ejercicio:** Saludo interactivo con nombre y apellido

   Solicite nombre y apellido mediante la entrada estándar y muestre por la salida estándar exactamente una línea de la forma:

   ```
   Me llamo nombre apellido
   ```

   Elimine los saltos de línea introducidos por la lectura y rechace entradas vacías.

   Scilab. Puede leer cadenas con `input(...,"string")` y mostrarlas con `disp` o `printf`.

```
function s = leer_no_vacio(msg)
    while %t
        s = stripblanks(input(msg,"string"));
        if s <> "" then return, end
        disp("La entrada no puede estar vacia.")
    end
endfunction

nombre = leer_no_vacio("Nombre: ");
apellido = leer_no_vacio("Apellido: ");
mprintf("Me llamo %s %s\n",nombre,apellido);
```

   Octave. Puede usar `input(...,"s")` y `printf`.

```
while true
  nombre=strtrim(input("Nombre: ","s"));
  if !isempty(nombre), break; endif
endwhile
while true
  apellido=strtrim(input("Apellido: ","s"));
  if !isempty(apellido), break; endif
endwhile
printf("Me llamo %s %s\n",nombre,apellido);
```

Python. Use `input` y `print`.

```
while True:
    nombre = input("Nombre: ").strip()
    if nombre:
        break
while True:
    apellido = input("Apellido: ").strip()
    if apellido:
        break
print(f"Me llamo {nombre} {apellido}")
```

Perl. Lea desde `STDIN`, elimine el salto de línea con `chomp` y escriba con `print`.

```
use strict; use warnings;
sub no_vacio{my($m)=@_;while(1){print $m;chomp(my
   $s=<STDIN>);$s=~s/^\s+|\s+$//g;return $s if length$s}}
my $n=no_vacio("Nombre: "); my $a=no_vacio("Apellido: ");
   print "Me llamo $n $a\n";
```

JavaScript. Use el mecanismo de lectura de consola configurado para Node.js y `console.log`.

```
"use strict";
const
   readline=require("node:readline/promises");const{stdin,stdout}=requir
   rl=readline.createInterface({input:stdin,output:stdout});
async function nv(m){for(;;){const s=(await
   rl.question(m)).trim();if(s)return s}}
(async()=>{const n=await nv("Nombre: "),a=await
   nv("Apellido: ");console.log(`Me llamo ${n}
   ${a}`);rl.close()})();
```

9. **Ejercicio:** Formato de salida numérica

   Para un radio real positivo r, calcule el área de un círculo y el volumen de una esfera:

$$A = \pi r^2, \qquad V = \frac{4}{3}\pi r^3.$$

a) muestre primero los valores utilizando la representación numérica predeterminada del lenguaje;

b) muestre después los mismos valores con exactamente cuatro cifras decimales y las unidades correspondientes;

c) compare ambas salidas y explique qué controla una especificación de formato.

Scilab. Compare `disp` con `printf`.

```
while %t
    s = stripblanks(input("Radio positivo: ","string"));
    if s<>"" & isnum(s) then
        r = strtod(s);
        if r>0 & ~isnan(r) & ~isinf(r) then break, end
    end
    disp("El radio debe ser un real positivo.")
end

area = %pi*r^2;
volumen = (4/3)*%pi*r^3;

disp("Salida predeterminada:");
disp(area);
disp(volumen);

mprintf("Area = %.4f m^2\n",area);
mprintf("Volumen = %.4f m^3\n",volumen);
```

Octave. Compare la visualización predeterminada con `printf`.

```
r=str2double(input("Radio: ","s"));
if !isfinite(r) || r<=0, error("Radio invalido"); endif
area=pi*r^2; volumen=(4/3)*pi*r^3;
disp(area); disp(volumen);
printf("Area = %.4f m^2\n",area);
printf("Volumen = %.4f m^3\n",volumen);
```

Python. Compare `print` con una cadena de formato, por ejemplo una *f-string*.

```
import math
r = float(input("Radio: "))
if not math.isfinite(r) or r <= 0:
    raise ValueError("Radio invalido")
area = math.pi*r*r
volumen = (4.0/3.0)*math.pi*r**3
print(area)
print(volumen)
print(f"Area = {area:.4f} m^2")
print(f"Volumen = {volumen:.4f} m^3")
```

Perl. Compare `print` con `printf`.

```
use strict; use warnings; use Math::Trig qw(pi);
print "Radio: "; chomp(my $r=<STDIN>); die "Radio
   invalido\n" unless $r>0; my $a=pi*$r*$r; my
   $v=4*pi*$r**3/3;
print "$a\n$v\n"; printf "Area = %.4f m^2\nVolumen = %.4f
   m^3\n",$a,$v;
```

JavaScript. Compare `console.log` con `toFixed` o una construcción equivalente de formato.

```
"use strict";
const
   readline=require("node:readline/promises");const{stdin,stdout}=requir
   rl=readline.createInterface({input:stdin,output:stdout});
(async()=>{const r=Number(await rl.question("Radio:
   "));if(!Number.isFinite(r)||r<=0)throw new Error("Radio
   invalido");const
   a=Math.PI*r*r,v=4*Math.PI*r**3/3;console.log(a);console.log(v);consol
   = ${a.toFixed(4)} m^2`);console.log(`Volumen =
   ${v.toFixed(4)} m^3`);rl.close()})().catch(console.error);
```

10. **Ejercicio:** Presentación de matrices y vectores

    Genere una matriz real pseudoaleatoria A de 5×5 y el vector

    $$b = (1{,}1, 1{,}2, 1{,}4, 1{,}6, 1{,}8, 2{,}0).$$

    Muestre por pantalla, precedidos por mensajes descriptivos:

    - A;
    - b;
    - $100A$;
    - nuevamente b.

    Utilice tanto una salida adecuada para inspeccionar estructuras completas como una salida formateada adecuada para construir mensajes controlados. Compare ambas formas de presentación.

    Scilab. Puede usar `rand`, `disp` y `printf`.

```
A = rand(5,5);
b = 1:0.2:2;

disp("A =");
disp(A);
disp("b =");
disp(b);
disp("100*A =");
disp(100*A);
```

```
disp("b nuevamente =");
disp(b);

mprintf("\nPresentacion formateada de 100*A:\n");
for i = 1:size(A,1)
    for j = 1:size(A,2)
        mprintf("%10.4f ",100*A(i,j));
    end
    mprintf("\n");
end
mprintf("b contiene %d elementos, desde %.1f hasta %.1f\n",
   ..
        size(b,"*"),b(1),b($));
```

Octave. Puede usar `rand`, visualización directa o `disp`, y `printf`.

```
A=rand(5,5); b=1:0.2:2;
disp("A:"); disp(A)
disp("b:"); disp(b)
disp("100*A:"); disp(100*A)
disp("b nuevamente:"); disp(b)
printf("A(1,1)=%.4f, b(1)=%.1f\n",A(1,1),b(1));
```

Python. Con NumPy puede usar `numpy.random` y `print`; para formato fino use cadenas de formato.

```
import numpy as np
A = np.random.random((5, 5))
b = np.arange(1.0, 2.01, 0.2)
print("A:"); print(A)
print("b:"); print(b)
print("100*A:"); print(100*A)
print("b nuevamente:"); print(b)
print(f"A(1,1)={A[0,0]:.4f}, b(1)={b[0]:.1f}")
```

Perl. Construya la matriz con arreglos; use `print` y `printf`.

```
use strict; use warnings;
my @A=map{[map{rand()}1..5]}1..5; my @b=map{1+0.2*$_}0..5;
print "A:\n"; print join(" ",map{sprintf "%.6f",$_}
   @{$_}),"\n" for @A;
print "b:\n",join(" ",@b),"\n100*A:\n"; print join("
   ",map{sprintf "%.6f",100*$_} @$_),"\n" for @A;
print "b nuevamente:\n",join(" ",@b),"\n";
```

JavaScript. Construya la matriz con arreglos; use `console.log` para la estructura y formato explícito para los mensajes numéricos.

```
"use strict";
const
   A=Array.from({length:5},()=>Array.from({length:5},Math.random));const
```

```
    b=Array.from({length:6},(_,i)=>1+0.2*i);console.log("A:");console.tab
    nuevamente:",b);console.log(`A(1,1)=${A[0][0].toFixed(4)},
    b(1)=${b[0].toFixed(1)}`);
```

6.2.2. Archivos de texto: lectura, escritura y actualización

11. **Ejercicio:** Escritura de resultados en archivo de texto

    Repita la generación de la matriz A y del vector b del ejercicio anterior, pero guarde en `salida1.txt`, con mensajes descriptivos, los valores de A, b y $100A$.

    El programa deberá:

    a) abrir explícitamente el archivo para escritura;

    b) comprobar que la apertura haya tenido éxito;

    c) escribir los números con un formato definido;

    d) cerrar el archivo y comprobar, cuando el entorno lo permita, el resultado del cierre;

    e) volver a abrir o inspeccionar el archivo y verificar su contenido.

    Scilab. Use `mopen`, `mfprintf` y `mclose`.

```
A = rand(5,5);
b = 1:0.2:2;

fd = mopen("salida1.txt","wt");
if fd < 0 then error("No se pudo abrir salida1.txt"), end

mfprintf(fd,"A =\n");
for i = 1:size(A,1)
    for j = 1:size(A,2)
        mfprintf(fd,"%12.6f ",A(i,j));
    end
    mfprintf(fd,"\n");
end

mfprintf(fd,"\nb =\n");
for j = 1:size(b,"*")
    mfprintf(fd,"%8.3f ",b(j));
end
mfprintf(fd,"\n\n100*A =\n");
for i = 1:size(A,1)
    for j = 1:size(A,2)
        mfprintf(fd,"%12.6f ",100*A(i,j));
    end
    mfprintf(fd,"\n");
end
```

```
estado_cierre = mclose(fd);
if estado_cierre <> 0 then error("Error al cerrar
   salida1.txt"), end

lineas = mgetl("salida1.txt");
disp("Contenido recuperado de salida1.txt:");
disp(lineas);
```

Octave. Use `fopen`, `fprintf` y `fclose`.

```
A=rand(5,5); b=1:0.2:2;
[fid,msg]=fopen("salida1.txt","w"); if fid<0, error(msg);
   endif
fprintf(fid,"Matriz A:\n");
for i=1:rows(A), fprintf(fid,"%10.6f ",A(i,:));
   fprintf(fid,"\n"); endfor
fprintf(fid,"Vector b:\n"); fprintf(fid,"%6.2f ",b);
   fprintf(fid,"\n100*A:\n");
for i=1:rows(A), fprintf(fid,"%10.6f ",100*A(i,:));
   fprintf(fid,"\n"); endfor
if fclose(fid)!=0, error("Error al cerrar salida1.txt");
   endif
fid=fopen("salida1.txt","r"); disp(fread(fid,Inf,"*char")');
   fclose(fid);
```

Python. Use `open` y preferentemente un administrador de contexto `with`.

```
from pathlib import Path
import numpy as np
A = np.random.random((5, 5)); b = np.arange(1.0, 2.01, 0.2)
try:
    with open("salida1.txt", "w", encoding="utf-8") as f:
        f.write("Matriz A:\n")
        np.savetxt(f, A, fmt="%10.6f")
        f.write("Vector b:\n")
        f.write(" ".join(f"{x:6.2f}" for x in b) + "\n")
        f.write("100*A:\n")
        np.savetxt(f, 100*A, fmt="%10.6f")
except OSError as e:
    raise SystemExit(f"Error de E/S: {e}")
print(Path("salida1.txt").read_text(encoding="utf-8"))
```

Perl. Use un manejador léxico y la forma de tres argumentos de `open`; compruebe los errores de E/S.

```
use strict; use warnings;
my @A=map{[map{rand()}1..5]}1..5; my @b=map{1+0.2*$_}0..5;
open my $f,">","salida1.txt" or die "salida1.txt: $!"; print
   $f "Matriz A:\n"; print $f join("
   ",map{sprintf"%10.6f",$_}@$_),"\n" for @A; print $f
   "Vector b:\n",join("
```

```
    ",map{sprintf"%6.2f",$_}@b),"\n100*A:\n"; print $f join("
    ",map{sprintf"%10.6f",100*$_}@$_),"\n" for @A; close $f
    or die $!;
open $f,"<","salida1.txt" or die $!; print while <$f>; close
    $f;
```

JavaScript. Use el módulo `fs` de Node.js; documente si emplea la interfaz síncrona o asíncrona.

```
"use strict";
const fs=require("node:fs");const
    A=Array.from({length:5},()=>Array.from({length:5},Math.random));const
    b=Array.from({length:6},(_,i)=>1+0.2*i);let s="Matriz
    A:\n"+A.map(r=>r.map(x=>x.toFixed(6).padStart(10)).join("
    ")).join("\n")+"\nVector
    b:\n"+b.map(x=>x.toFixed(2).padStart(6)).join("
    ")+"\n100*A:\n"+A.map(r=>r.map(x=>(100*x).toFixed(6).padStart(10)).jo
    ")).join("\n")+"\n";try{fs.writeFileSync("salida1.txt",s,{encoding:"u
    de E/S:",e.message);process.exitCode=1}
```

12. **Ejercicio:** Interacción entre entrada, cálculo y archivos

    Realice las siguientes dos tareas:

    a) solicite dos números reales mediante la entrada estándar, calcule su suma y guarde en `suma.txt` los dos operandos y el resultado con un formato numérico explícito;

    b) lea desde `A.txt` y `B.txt` dos matrices reales de 3×3, calcule el producto matricial AB y guarde el resultado en `producto.txt`.

    En ambos casos compruebe los errores de lectura y escritura y valide que los datos tengan la forma esperada.

    Scilab. Para archivos de texto puede usar `mopen`/`mclose` y las rutinas de lectura/escritura formateada o matricial.

```
function x = leer_real(msg)
    while %t
        s=stripblanks(input(msg,"string"));
        if s<>"" & isnum(s) then
            x=strtod(s);
            if ~isnan(x) & ~isinf(x) then return, end
        end
        disp("Entrada numerica invalida.")
    end
endfunction

// a) Entrada interactiva y salida a suma.txt
x = leer_real("Primer real: ");
y = leer_real("Segundo real: ");
```

```
z = x+y;
fd = mopen("suma.txt","wt");
if fd<0 then error("No se pudo abrir suma.txt"), end
mfprintf(fd,"x = %.8f\ny = %.8f\nsuma = %.8f\n",x,y,z);
if mclose(fd)<>0 then error("Error al cerrar suma.txt"), end

// b) Dos matrices 3x3 desde texto tabular
if ~isfile("A.txt") | ~isfile("B.txt") then
    error("Faltan A.txt o B.txt")
end
A = fscanfMat("A.txt");
B = fscanfMat("B.txt");
if or(size(A)<>[3 3]) | or(size(B)<>[3 3]) then
    error("A.txt y B.txt deben contener matrices 3x3")
end
P = A*B;
fprintfMat("producto.txt",P,"%.10f");
disp("Producto AB:");
disp(P);
```

Octave. Puede usar `fopen`/`fclose` y las rutinas de lectura/escritura numérica disponibles.

```
a=str2double(input("a: ","s")); b=str2double(input("b:
   ","s"));
if !isfinite(a) || !isfinite(b), error("Operandos
   invalidos"); endif
fid=fopen("suma.txt","w"); if fid<0, error("No se pudo abrir
   suma.txt"); endif
fprintf(fid,"%.8f + %.8f = %.8f\n",a,b,a+b); fclose(fid);
A=dlmread("A.txt"); B=dlmread("B.txt");
if !isequal(size(A),[3 3]) || !isequal(size(B),[3 3]),
   error("Se esperaban matrices 3x3"); endif
P=A*B; fid=fopen("producto.txt","w");
for i=1:3, fprintf(fid,"%.8f %.8f %.8f\n",P(i,:)); endfor
fclose(fid);
```

Python. Use `open`; para matrices puede usar NumPy y sus rutinas de carga/guardado de texto.

```
import numpy as np
a = float(input("a: ")); b = float(input("b: "))
try:
    with open("suma.txt", "w", encoding="utf-8") as f:
        f.write(f"{a:.8f} + {b:.8f} = {a+b:.8f}\n")
    A = np.loadtxt("A.txt"); B = np.loadtxt("B.txt")
    if A.shape != (3,3) or B.shape != (3,3):
        raise ValueError("Se esperaban matrices 3x3")
    np.savetxt("producto.txt", A @ B, fmt="%.8f")
except (OSError, ValueError) as e:
    raise SystemExit(f"Error: {e}")
```

Perl. Use manejadores léxicos; represente las matrices con arreglos de arreglos y valide la cantidad de elementos por fila.

```
use strict; use warnings;
sub matriz{my($nom)=@_;open my $f,"<",$nom or die "$nom:
   $!";my @M;while(<$f>){next if /^\s*$/;my @r=split;die
   "Fila invalida en $nom\n" unless @r==3;push
   @M,[map{0+$_}@r]}close$f;die "Se esperaba 3x3\n" unless
   @M==3;return \@M}
print "a: ";chomp(my $a=<STDIN>);print "b: ";chomp(my
   $b=<STDIN>);open my $s,">","suma.txt" or die$!;printf $s
   "%.8f + %.8f = %.8f\n",$a,$b,$a+$b;close$s;
my $A=matriz("A.txt");my $B=matriz("B.txt");my
   @P=map{[(0)x3]}1..3;for my $i(0..2){for my $j(0..2){for
   my $k(0..2){$P[$i][$j]+=$A->[$i][$k]*$B->[$k][$j]}}}open
   my $f,">","producto.txt" or die$!;print $f join("
   ",map{sprintf"%.8f",$_}@$_),"\n" for @P;close$f;
```

JavaScript. Use `fs`; represente las matrices con arreglos de arreglos y convierta cada campo después de validarlo.

```
"use strict";
const
   fs=require("node:fs"),readline=require("node:readline/promises");cons
   rl=readline.createInterface({input:stdin,output:stdout});function
   matriz(n){const
   M=fs.readFileSync(n,"utf8").trim().split(/\r?\n/).map(l=>l.trim().spl
   new Error(`${n} no es 3x3`);return M}function
   mult(A,B){return
   A.map((r,i)=>B[0].map((_,j)=>r.reduce((s,x,k)=>s+x*B[k][j],0)))}
(async()=>{const a=Number(await rl.question("a:
   ")),b=Number(await rl.question("b:
   "));if(!Number.isFinite(a)||!Number.isFinite(b))throw new
   Error("Operandos
   invalidos");fs.writeFileSync("suma.txt",`${a.toFixed(8)}
   + ${b.toFixed(8)} = ${(a+b).toFixed(8)}\n`);const
   P=mult(matriz("A.txt"),matriz("B.txt"));fs.writeFileSync("producto.tx
   ")).join("\n")+"\n");rl.close()})().catch(e=>{console.error(e);proces
```

13. **Ejercicio:** Enumeración de distancias en archivo

Para un entero positivo n, defina

$$d_j = 0{,}5 + 0{,}25(j-1), \qquad j = 1, \ldots, n.$$

Genere un archivo `distancias.txt` con exactamente n líneas de la forma

```
La distancia numero j es d
```

reemplazando j y d por sus valores. Escriba las distancias con dos cifras decimales y compruebe que el archivo contenga exactamente n líneas.

Scilab. Construya el vector mediante operaciones vectoriales o un ciclo y escriba con `mfprintf`.

```
while %t
    s=stripblanks(input("n entero positivo: ","string"));
    if s<>"" & isnum(s) then
        n=strtod(s);
        if n>0 & n==floor(n) then break, end
    end
    disp("n debe ser entero positivo.")
end

fd=mopen("distancias.txt","wt");
if fd<0 then error("No se pudo crear distancias.txt"), end
for j=1:n
    d=0.5+0.25*(j-1);
    mfprintf(fd,"La distancia numero %d es %.2f\n",j,d);
end
if mclose(fd)<>0 then error("Error al cerrar
   distancias.txt"), end

lineas=mgetl("distancias.txt");
if size(lineas,"*")<>n then error("Cantidad de lineas
   incorrecta"), end
mprintf("Archivo verificado: %d lineas.\n",size(lineas,"*"));
```

Octave. Construya el vector mediante operaciones vectoriales o un ciclo y escriba con `fprintf`.

```
n=str2double(input("n: ","s"));
if !isfinite(n) || n<=0 || n!=fix(n), error("n invalido");
   endif
fid=fopen("distancias.txt","w"); if fid<0, error("No se pudo
   crear archivo"); endif
for j=1:n
  d=0.5+0.25*(j-1); fprintf(fid,"La distancia numero %d es
     %.2f\n",j,d);
endfor
fclose(fid);
fid=fopen("distancias.txt","r"); c=0; while
   ischar(fgetl(fid)), c++; endwhile; fclose(fid);
printf("Lineas: %d\n",c);
```

Python. Puede generar los valores con un ciclo o una comprensión y escribir con `print(..., file=...)` o `write`.

```
n = int(input("n: "))
if n <= 0:
    raise ValueError("n invalido")
with open("distancias.txt", "w", encoding="utf-8") as f:
    for j in range(1, n+1):
```

```
        d = 0.5 + 0.25*(j-1)
        f.write(f"La distancia numero {j} es {d:.2f}\n")
with open("distancias.txt", encoding="utf-8") as f:
    print("Lineas:", sum(1 for _ in f))
```

Perl. Use un ciclo y `printf` sobre un manejador de archivo.

```
use strict; use warnings; print "n: "; chomp(my $n=<STDIN>);
   die "n invalido\n" unless $n=~/^\d+$/&&$n>0; open my
   $f,">","distancias.txt" or die$!; for my $j(1..$n){printf
   $f "La distancia numero %d es %.2f\n",$j,0.5+0.25*($j-1)}
   close$f; open$f,"<","distancias.txt" or die$!;my
   $c=0;$c++ while <$f>;close$f;print "Lineas: $c\n";
```

JavaScript. Use un ciclo y el módulo `fs`; construya cada línea mediante una cadena formateada.

```
"use strict";
const
   fs=require("node:fs"),readline=require("node:readline/promises");cons
   rl=readline.createInterface({input:stdin,output:stdout});
(async()=>{const n=Number(await rl.question("n:
   "));if(!Number.isInteger(n)||n<=0)throw new Error("n
   invalido");let s="";for(let j=1;j<=n;j++)s+=`La distancia
   numero ${j} es
   ${(0.5+0.25*(j-1)).toFixed(2)}\n`;fs.writeFileSync("distancias.txt",s
```

14. **Ejercicio:** Graficación de datos almacenados

Genere 100 valores equiespaciados

$$x \in [0, 2\pi]$$

y, para cada uno, calcule

$$y = \sin(x) + \varepsilon,$$

donde ε es un pequeño ruido pseudoaleatorio de amplitud máxima 0,1.

a) guarde los pares (x, y) en un archivo de texto tabular con una línea de cabecera;

b) cierre el archivo;

c) léalo nuevamente desde un programa distinto o desde una nueva fase del mismo programa;

d) grafique los datos recuperados, no los que quedaron en memoria;

e) inspeccione el archivo desde la terminal con una herramienta de texto;

f) agregue a la cabecera la fecha y hora de generación.

Scilab. Puede usar `mopen`, `mfprintf`, `mclose`, una rutina de lectura tabular y las funciones gráficas de Scilab.

```
N=100;
x=linspace(0,2*%pi,N);
ruido=0.2*(rand(1,N)-0.5); // -0.1 <= ruido < 0.1
y=sin(x)+ruido;

fecha=getdate();
encabezado=msprintf("# generado %04d-%02d-%02d
   %02d:%02d:%02d", ..
    fecha(1),fecha(2),fecha(6),fecha(7),fecha(8),fecha(9));

fd=mopen("datos_grafico.txt","wt");
if fd<0 then error("No se pudo crear el archivo"), end
mfprintf(fd,"%s\n",encabezado);
mfprintf(fd,"# x,y\n");
for i=1:N
    mfprintf(fd,"%.12g,%.12g\n",x(i),y(i));
end
mclose(fd);

// Recuperar desde el archivo, no desde las variables
   originales.
lineas=mgetl("datos_grafico.txt");
M=zeros(size(lineas,"*")-2,2);
for i=3:size(lineas,"*")
    p=tokens(lineas(i),",");
    if size(p,"*")<>2 | ~isnum(p(1)) | ~isnum(p(2)) then
        error(msprintf("Fila invalida: %d",i))
    end
    M(i-2,:)=[strtod(p(1)) strtod(p(2))];
end

scf();
plot(M(:,1),M(:,2));
xgrid();
xtitle("Datos recuperados del archivo","x","y");

[estado,salida,error_sistema]=host("head -8
   datos_grafico.txt");
if estado==0 then disp(salida), else disp(error_sistema), end
```

Octave. Use las funciones de archivo y las herramientas gráficas de Octave; puede generar el eje con `linspace`.

```
x=linspace(0,2*pi,100)'; y=sin(x)+0.2*(rand(size(x))-0.5);
fid=fopen("datos_xy.csv","w"); fprintf(fid,"# generado
   %s\n",datestr(now)); fprintf(fid,"x,y\n");
for i=1:numel(x), fprintf(fid,"%.8f,%.8f\n",x(i),y(i));
   endfor
fclose(fid);
fid=fopen("datos_xy.csv","r"); fgetl(fid); fgetl(fid);
   C=textscan(fid,"%f%f","Delimiter",","); fclose(fid);
```

```
x2=C{1}; y2=C{2}; plot(x2,y2,"o-"); grid on;
   print("datos_xy.png","-dpng");
system("head datos_xy.csv");
```

Python. Puede usar NumPy para los datos, `csv` o `savetxt`/ `loadtxt` para el archivo y la biblioteca gráfica configurada.

```
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x) + np.random.uniform(-0.1, 0.1, size=x.size)
with open("datos_xy.csv", "w", encoding="utf-8") as f:
    f.write(f"# generado {datetime.now().isoformat()}\n")
    f.write("x,y\n")
    for a,b in zip(x,y):
        f.write(f"{a:.8f},{b:.8f}\n")
rec = np.loadtxt("datos_xy.csv", delimiter=",", skiprows=2)
plt.plot(rec[:,0], rec[:,1], "o-"); plt.grid();
   plt.savefig("datos_xy.png"); plt.close()
with open("datos_xy.csv", encoding="utf-8") as f:
    print("".join(f.readlines()[:5]))
```

Perl. Use archivos de texto y la biblioteca gráfica configurada para el curso; mantenga el formato tabular común.

```
use strict; use warnings; use Math::Trig qw(pi); use POSIX
   qw(strftime);
open my $f,">","datos_xy.csv" or die$!;print $f "# generado
   ",strftime("%Y-%m-%d %H:%M:%S",localtime),"\nx,y\n";for
   my
   $i(0..99){my$x=2*pi*$i/99;my$y=sin($x)+(rand()*0.2-0.1);printf$f"%.8f
open$f,"<","datos_xy.csv" or die$!;<$f>;<$f>;open
   my$d,">","datos_xy.dat" or die$!;while(<$f>){s/,/
   /;print$d$_}close$f;close$d;open my$g,">","datos_xy.gp"
   or die$!;print$g"set terminal pngcairo\nset output
   'datos_xy.png'\nset grid\nplot 'datos_xy.dat' with
   linespoints\n";close$g;system("gnuplot","datos_xy.gp");
```

JavaScript. Use `fs` para el archivo y la biblioteca gráfica configurada para el curso; mantenga el formato tabular común.

```
"use strict";
const
   fs=require("node:fs"),{spawnSync}=require("node:child_process");let
   s=`# generado ${new Date().toISOString()}\nx,y\n`;for(let
   i=0;i<100;i++){const
   x=2*Math.PI*i/99,y=Math.sin(x)+(Math.random()*0.2-0.1);s+=`${x.toFixe
   lines=fs.readFileSync("datos_xy.csv","utf8").trim().split(/\r?\n/).sl
   ")).join("\n")+"\n");fs.writeFileSync("datos_xy.gp","set
   terminal pngcairo\nset output 'datos_xy.png'\nset
```

```
    grid\nplot 'datos_xy.dat' w
    lp\n");spawnSync("gnuplot",["datos_xy.gp"],{stdio:"inherit"});console
```

15. **Ejercicio:** Bitácora de operaciones

    Implemente un sistema de registro (*log*) que conserve en `operaciones.log` una línea por cada operación realizada por una calculadora elemental.

    Cada línea deberá contener como mínimo:

    - fecha y hora;
    - operandos;
    - operación solicitada;
    - resultado o la palabra `ERROR` y una descripción.

    a) realice diez operaciones, incluyendo una división por cero;

    b) cierre y vuelva a abrir el archivo y muestre su contenido;

    c) ejecute nuevamente el programa y compruebe que las nuevas líneas se agregan sin destruir las anteriores;

    d) desde la terminal, cuente las líneas que contienen `ERROR`;

    e) explique la diferencia entre abrir para reemplazar y abrir para agregar.

    Scilab. Use modo de apertura de agregado con `mopen` y escritura formateada con `mfprintf`.

```
function marca = ahora()
    d=getdate();
    marca=msprintf("%04d-%02d-%02d %02d:%02d:%02d", ..
        d(1),d(2),d(6),d(7),d(8),d(9));
endfunction

function log_escribir(linea)
    fd=mopen("operaciones.log","at");
    if fd<0 then error("No se pudo abrir operaciones.log"),
       end
    mfprintf(fd,"%s\n",linea);
    mclose(fd);
endfunction

function calcular_y_registrar(a,b,op)
    ok=%t;
    resultado=%nan;
    select op
    case "+" then resultado=a+b;
    case "-" then resultado=a-b;
    case "*" then resultado=a*b;
    case "/" then
        if b==0 then ok=%f; else resultado=a/b; end
```

```
    else
        ok=%f;
    end

    if ok then
        linea=msprintf("[%s] %g %s %g =
           %.12g",ahora(),a,op,b,resultado);
    else
        linea=msprintf("[%s] %g %s %g = ERROR operacion
           invalida", ..
                       ahora(),a,op,b);
    end
    log_escribir(linea);
endfunction

calcular_y_registrar(2,3,"+");
calcular_y_registrar(9,4,"-");
calcular_y_registrar(5,7,"*");
calcular_y_registrar(8,2,"/");
calcular_y_registrar(8,0,"/");
calcular_y_registrar(1,1,"+");
calcular_y_registrar(10,3,"-");
calcular_y_registrar(6,6,"*");
calcular_y_registrar(15,5,"/");
calcular_y_registrar(2,0,"/");

// El archivo se cierra en cada escritura y se vuelve a leer
   aqui.
disp(mgetl("operaciones.log"));
[estado,salida,error_sistema]=host("grep -c ERROR
   operaciones.log");
if estado==0 then
    mprintf("Lineas con ERROR: %s\n",stripblanks(salida));
else
    disp(error_sistema)
end
```

Octave. Use `fopen` en modo de agregado y `fprintf`.

```
ops={10,2,"+"; 8,3,"-"; 4,5,"*"; 9,3,"/"; 1,0,"/"; ...
     7,2,"+"; 6,4,"-"; 3,3,"*"; 5,2,"/"; 12,7,"+"};
fid=fopen("operaciones.log","a"); if fid<0, error("No se
   pudo abrir log"); endif
for k=1:rows(ops)
  a=ops{k,1}; b=ops{k,2}; op=ops{k,3};
     marca=datestr(now,"yyyy-mm-dd HH:MM:SS");
  if strcmp(op,"/") && b==0
    fprintf(fid,"%s | %g %s %g | ERROR division por
       cero\n",marca,a,op,b);
  else
    switch op, case "+", r=a+b; case "-", r=a-b; case "*",
```

```
        r=a*b; case "/", r=a/b; endswitch
    fprintf(fid,"%s | %g %s %g = %g\n",marca,a,op,b,r);
  endif
endfor
fclose(fid); type("operaciones.log"); system("grep -c ERROR
   operaciones.log");
```

Python. Use `open(...,"a")` y el módulo de fecha/hora estándar.

```
from datetime import datetime
from pathlib import Path
ops = [(10,2,"+"),(8,3,"-"),(4,5,"*"),(9,3,"/"),(1,0,"/"),
       (7,2,"+"),(6,4,"-"),(3,3,"*"),(5,2,"/"),(12,7,"+")]
def calcular(a,b,op):
    if op == "+": return a+b
    if op == "-": return a-b
    if op == "*": return a*b
    if op == "/" and b != 0: return a/b
    raise ZeroDivisionError("division por cero")
with open("operaciones.log", "a", encoding="utf-8") as f:
    for a,b,op in ops:
        marca = datetime.now().isoformat(timespec="seconds")
        try:
            r = calcular(a,b,op); f.write(f"{marca} | {a}
               {op} {b} = {r}\n")
        except Exception as e:
            f.write(f"{marca} | {a} {op} {b} | ERROR {e}\n")
contenido =
   Path("operaciones.log").read_text(encoding="utf-8")
print(contenido)
print("Errores:", sum("ERROR" in l for l in
   contenido.splitlines()))
```

Perl. Use la forma de tres argumentos de `open` con modo de agregado y obtenga la fecha/hora con una rutina estándar.

```
use strict; use warnings; use POSIX qw(strftime);
my
   @ops=([10,2,"+"],[8,3,"-"],[4,5,"*"],[9,3,"/"],[1,0,"/"],[7,2,"+"],[6
open my$f,">>","operaciones.log" or
   die$!;for(@ops){my($a,$b,$op)=@$_;my$ts=strftime("%Y-%m-%d
   %H:%M:%S",localtime);if($op eq"/"&&$b==0){print$f"$ts |
   $a $op $b | ERROR division por cero\n";next}my$r=$op
   eq"+"?$a+$b:$op eq"-"?$a-$b:$op
   eq"*"?$a*$b:$a/$b;print$f"$ts | $a $op $b =
   $r\n"}close$f;open$f,"<","operaciones.log" or
   die$!;my$e=0;while(<$f>){print;$e++
   if/ERROR/}close$f;print"Errores: $e\n";
```

JavaScript. Use `fs` en modo de agregado y una representación de fecha/hora estándar de JavaScript.

```
"use strict";
const fs=require("node:fs");const
   ops=[[10,2,"+"],[8,3,"-"],[4,5,"*"],[9,3,"/"],[1,0,"/"],[7,2,"+"],[6,
   calc(a,b,op){if(op==="+")return a+b;if(op==="-")return
   a-b;if(op==="*")return a*b;if(op==="/"&&b!==0)return
   a/b;throw new Error("division por cero")}let
   out="";for(const[a,b,op]of ops){const ts=new
   Date().toISOString();try{out+=`${ts} | ${a} ${op} ${b} =
   ${calc(a,b,op)}\n`}catch(e){out+=`${ts} | ${a} ${op} ${b}
   | ERROR
   ${e.message}\n`}}fs.appendFileSync("operaciones.log",out);const
   c=fs.readFileSync("operaciones.log","utf8");console.log(c);console.lo
```

16. **Ejercicio:** Monitoreo de un sensor en tiempo real

    Implemente un programa que simule un sensor de temperatura. Cada intervalo de tiempo deberá generar una medición

$$T(t) = 20 + 5\sin\left(\frac{2\pi t}{86400}\right) + \varepsilon,$$

    donde ε es un pequeño ruido pseudoaleatorio.

    El programa deberá:

    a) crear `sensor.txt` y escribir una cabecera;

    b) agregar periódicamente una línea con tiempo y temperatura;

    c) hacer visible cada nueva línea a otros procesos sin esperar a que termine toda la simulación;

    d) permitir que, desde otra terminal, se observe el crecimiento del archivo con `tail -f sensor.txt`;

    e) al terminar, leer nuevamente el archivo, calcular media, máximo y mínimo y generar un gráfico;

    f) marcar en el gráfico las mediciones que se aparten más de dos desvíos estándar de la media y guardar el gráfico como PNG.

    Scilab. Use escritura incremental y cierre o vaciado apropiado del flujo para que `tail -f` vea las líneas; use las funciones gráficas de Scilab.

```
archivo="sensor.txt";
n_mediciones=30;
intervalo=1; // segundos, para la demostracion

fd=mopen(archivo,"wt");
if fd<0 then error("No se pudo crear sensor.txt"), end
mfprintf(fd,"# tiempo_s,temperatura_C\n");
mclose(fd);
```

```
for i=1:n_mediciones
    t=(i-1)*intervalo;
    eps_ruido=0.5*(rand()-0.5);
    T=20+5*sin(2*%pi*t/86400)+eps_ruido;

    // Abrir, agregar y cerrar hace visible inmediatamente
       cada linea.
    fd=mopen(archivo,"at");
    if fd<0 then error("No se pudo abrir sensor.txt para
       agregar"), end
    mfprintf(fd,"%.3f,%.6f\n",t,T);
    mclose(fd);
    sleep(intervalo*1000);
end

lineas=mgetl(archivo);
n=size(lineas,"*")-1;
t=zeros(n,1); temp=zeros(n,1);
for i=1:n
    p=tokens(lineas(i+1),",");
    t(i)=strtod(p(1));
    temp(i)=strtod(p(2));
end

media=mean(temp);
desvio=stdev(temp);
maximo=max(temp);
minimo=min(temp);
out=find(abs(temp-media)>2*desvio);

mprintf("Media: %.4f  Min: %.4f  Max:
   %.4f\n",media,minimo,maximo);
scf();
plot(t,temp);
xgrid();
xtitle("Sensor simulado","tiempo [s]","temperatura [C]");
if size(out,"*")>0 then
    ax=gca(); ax.auto_clear="off";
    plot(t(out),temp(out),"ro");
    ax.auto_clear="on";
end
xs2png(gcf(),"sensor.png");
```

Octave. Use escritura incremental y `fflush` cuando corresponda; use las herramientas gráficas de Octave.

```
N=60; intervalo=0.2; fid=fopen("sensor.txt","w");
fprintf(fid,"tiempo_s,temperatura_C\n");
for k=1:N
  t=(k-1)*intervalo;
     T=20+5*sin(2*pi*t/86400)+0.2*(rand()-0.5);
```

```
  fprintf(fid,"%.3f,%.6f\n",t,T); fflush(fid);
     pause(intervalo);
endfor
fclose(fid);
fid=fopen("sensor.txt","r"); fgetl(fid);
   C=textscan(fid,"%f%f","Delimiter",","); fclose(fid);
t=C{1}; T=C{2}; mu=mean(T); s=std(T); out=abs(T-mu)>2*s;
printf("Media=%g, min=%g, max=%g\n",mu,min(T),max(T));
plot(t,T,"b.-"); hold on; plot(t(out),T(out),"ro");
   yline(mu,"k--"); hold off; grid on;
print("sensor.png","-dpng");
```

Python. Use un archivo con vaciado explícito (`flush`) y una pausa entre mediciones; use la biblioteca gráfica configurada.

```
import time
import numpy as np
import matplotlib.pyplot as plt
N=60; intervalo=0.2
with open("sensor.txt", "w", buffering=1, encoding="utf-8")
   as f:
    f.write("tiempo_s,temperatura_C\n")
    for k in range(N):
        t=k*intervalo
        T=20+5*np.sin(2*np.pi*t/86400)+np.random.uniform(-0.1,0.1)
        f.write(f"{t:.3f},{T:.6f}\n"); f.flush();
           time.sleep(intervalo)
D=np.loadtxt("sensor.txt",delimiter=",",skiprows=1);
   t=D[:,0]; T=D[:,1]
mu=T.mean(); s=T.std(ddof=1); out=np.abs(T-mu)>2*s
print("Media, min, max:",mu,T.min(),T.max())
plt.plot(t,T,".-"); plt.scatter(t[out],T[out]);
   plt.axhline(mu,linestyle="--")
plt.grid(); plt.savefig("sensor.png"); plt.close()
```

Perl. Active el vaciado automático o realice un vaciado explícito del manejador; use la biblioteca gráfica configurada.

```
use strict; use warnings; use Math::Trig qw(pi); use
   Time::HiRes qw(sleep); use IO::Handle;
my($N,$dt)=(60,0.2);open my$f,">","sensor.txt" or
   die$!;$f->autoflush(1);print$f"tiempo_s,temperatura_C\n";for
   my$k(0..$N-1){my$t=$k*$dt;my$T=20+5*sin(2*pi*$t/86400)+(rand()*0.2-0.
open$f,"<","sensor.txt" or
   die$!;<$f>;my(@t,@T);while(<$f>){chomp;my($a,$b)=split/,/;push@t,$a;p
   for@T;my$mu=$sum/@T;my$ss=0;$ss+=($_-$mu)**2
   for@T;my$sd=sqrt($ss/(@T-1));my($min,$max)=($T[0],$T[0]);for(@T){$min
   if$_<$min;$max=$_ if$_>$max}print"Media=$mu min=$min
   max=$max\n";
open my$d,">","sensor.dat" or die$!;for
   my$i(0..$#t){print$d"$t[$i] $T[$i]
```

```
",abs($T[$i]-$mu)>2*$sd?1:0,"\n"}close$d;open
my$g,">","sensor.gp" or die$!;print$g"set terminal
pngcairo\nset output 'sensor.png'\nplot 'sensor.dat' u
1:2 w l, '' u 1:(\$3?\$2:1/0) w
p\n";close$g;system("gnuplot","sensor.gp");
```

JavaScript. Si usa un flujo de escritura, controle cuándo se envían los datos al sistema; para el intervalo puede emplear temporizadores de Node.js.

```
"use strict";
const
   fs=require("node:fs"),{spawnSync}=require("node:child_process");const
   sleep=ms=>new Promise(r=>setTimeout(r,ms));
(async()=>{const
   N=60,dt=0.2;fs.writeFileSync("sensor.txt","tiempo_s,temperatura_C\n")
   k=0;k<N;k++){const
   t=k*dt,T=20+5*Math.sin(2*Math.PI*t/86400)+(Math.random()*0.2-0.1);fs.
   sleep(dt*1000)}const
   D=fs.readFileSync("sensor.txt","utf8").trim().split(/\r?\n/).slice(1)
   mu=T.reduce((a,b)=>a+b,0)/T.length,sd=Math.sqrt(T.reduce((s,x)=>s+(x-
   min,
   max:",mu,Math.min(...T),Math.max(...T));fs.writeFileSync("sensor.dat"
   ${x}
   ${Math.abs(x-mu)>2*sd?1:0}`).join("\n")+"\n");fs.writeFileSync("senso
   terminal pngcairo\nset output 'sensor.png'\nplot
   'sensor.dat' u 1:2 w l, '' u 1:($3?$2:1/0) w
   p\n");spawnSync("gnuplot",["sensor.gp"],{stdio:"inherit"})})().catch(
```

6.2.3. Datos tabulares, persistencia e intercambio

17. **Ejercicio:** Persistencia matricial en texto, CSV y formato nativo

    Genere una matriz real pseudoaleatoria de 4×4 y guárdela de tres maneras:

    a) como tabla de texto numérico sin cabecera;

    b) como archivo CSV con una fila por cada fila de la matriz;

    c) en un formato binario o nativo del entorno capaz de conservar la matriz como objeto numérico.

    Cierre los archivos, elimine o reemplace la matriz en memoria y recupérela nuevamente desde cada representación. Verifique dimensiones y valores.

    Scilab. Para texto tabular puede usar `fprintfMat`/`fscanfMat`, para CSV `csvWrite`/`csvRead` y para formato nativo `save`/`load`.

```
A=rand(4,4);

// 1) Texto tabular
fprintfMat("matriz.txt",A,"%.17g");
```

```
// 2) CSV
csvWrite(A,"matriz.csv",",",".","%.17g");

// 3) Formato nativo de Scilab
save("matriz.sod","A");

A_original=A;
clear A;

A_txt=fscanfMat("matriz.txt");
A_csv=csvRead("matriz.csv",",",".","double");
load("matriz.sod","A");
A_sod=A;
clear A;

mprintf("Error maximo texto :
   %.3e\n",max(abs(A_txt-A_original)));
mprintf("Error maximo CSV   :
   %.3e\n",max(abs(A_csv-A_original)));
mprintf("Error maximo nativo:
   %.3e\n",max(abs(A_sod-A_original)));

if or(size(A_txt)<>size(A_original)) |
   or(size(A_csv)<>size(A_original)) then
    error("Alguna representacion no conservo las
       dimensiones")
end
```

Octave. Use las rutinas de texto/CSV disponibles y `save`/ `load` para el formato nativo.

```
A=rand(4,4);
dlmwrite("matriz.txt",A,"delimiter"," ","precision","%.17g");
dlmwrite("matriz.csv",A,"delimiter",",","precision","%.17g");
save("matriz.mat","A","-binary"); clear A
T=dlmread("matriz.txt"); C=dlmread("matriz.csv",",");
   load("matriz.mat","A");
printf("Errores: texto=%g csv=%g
   nativo=%g\n",max(abs(T(:)-A(:))),max(abs(C(:)-A(:))),0);
```

Python. Con NumPy puede usar `savetxt`/`loadtxt` para texto y CSV, y `save`/`load` para el formato `.npy`.

```
import numpy as np
A=np.random.random((4,4))
np.savetxt("matriz.txt",A,fmt="%.17g")
np.savetxt("matriz.csv",A,fmt="%.17g",delimiter=",")
np.save("matriz.npy",A)
del A
T=np.loadtxt("matriz.txt");
```

```
    C=np.loadtxt("matriz.csv",delimiter=",");
    N=np.load("matriz.npy")
print("Dimensiones:",T.shape,C.shape,N.shape)
print("Errores:",np.max(abs(T-N)),np.max(abs(C-N)))
```

Perl. Escriba texto y CSV explícitamente; para una representación nativa puede usar el mecanismo de serialización configurado para el curso.

```
use strict; use warnings; use Storable qw(store retrieve);
my@A=map{[map{rand()}1..4]}1..4;open my$t,">","matriz.txt"
   or die$!;print$t join(" ",map{sprintf"%.17g",$_}@$_),"\n"
   for@A;close$t;open my$c,">","matriz.csv" or die$!;print$c
   join(",",map{sprintf"%.17g",$_}@$_),"\n"
   for@A;close$c;store(\@A,"matriz.sto");undef@A;
sub leer{my($f,$sep)=@_;open my$h,"<",$f or
   die$!;my@M;while(<$h>){chomp;push@M,[map{0+$_}split/$sep/]}close$h;re
   ",scalar(@$T),"x",scalar(@{$T->[0]}),"\n";
```

JavaScript. Use `fs` para texto y CSV; para la tercera representación use el mecanismo binario/serializado configurado para el curso.

```
"use strict";
const fs=require("node:fs"),v8=require("node:v8");let
   A=Array.from({length:4},()=>Array.from({length:4},Math.random));fs.wr
   ")).join("\n")+"\n");fs.writeFileSync("matriz.csv",A.map(r=>r.map(x=>
   read=(f,sep)=>fs.readFileSync(f,"utf8").trim().split(/\r?\n/).map(l=>
   T=read("matriz.txt",/\s+/),C=read("matriz.csv",","),N=v8.deserialize(
```

18. **Ejercicio:** Verificación de matrices guardadas

Genere una matriz real pseudoaleatoria de 6×6 y redondee sus elementos a dos cifras decimales. Guárdela y recupérela utilizando:

- texto tabular;
- CSV;
- el formato nativo o binario utilizado en el ejercicio anterior.

Para cada representación compruebe:

a) que las dimensiones se conservaron;

b) que los valores recuperados coinciden con los escritos dentro de la precisión del formato;

c) si el archivo puede inspeccionarse con un editor de texto;

d) qué información adicional, además de los números, conserva el formato.

Scilab. Compare matrices con una norma o con la diferencia máxima y use las mismas rutinas de persistencia del ejercicio anterior.

```
A=round(100*rand(6,6))/100;

fprintfMat("verifica.txt",A,"%.2f");
csvWrite(A,"verifica.csv",",",".","%.2f");
save("verifica.sod","A");

A0=A;
clear A;
T=fscanfMat("verifica.txt");
C=csvRead("verifica.csv",",",".","double");
load("verifica.sod","A");
N=A;
clear A;

mprintf("Dimensiones originales: %d x
   %d\n",size(A0,1),size(A0,2));
mprintf("Texto  - diferencia maxima: %.3e\n",max(abs(T-A0)));
mprintf("CSV    - diferencia maxima: %.3e\n",max(abs(C-A0)));
mprintf("Nativo - diferencia maxima: %.3e\n",max(abs(N-A0)));

disp("Primeras lineas del archivo de texto:");
disp(mgetl("verifica.txt",3));
disp("Primeras lineas del CSV:");
disp(mgetl("verifica.csv",3));

[estado,salida,error_sistema]=host("file verifica.txt
   verifica.csv verifica.sod");
if estado==0 then disp(salida), else disp(error_sistema), end
```

Octave. Compare matrices con una norma o diferencia máxima y use las mismas rutinas de persistencia del ejercicio anterior.

```
A=round(100*rand(6,6))/100;
dlmwrite("m6.txt",A,"delimiter"," ","precision","%.2f");
dlmwrite("m6.csv",A,"delimiter",",","precision","%.2f");
   save("m6.mat","A","-binary");
T=dlmread("m6.txt"); C=dlmread("m6.csv",","); clear B;
   load("m6.mat","A"); B=A;
printf("Dimensiones: %d
   %d\n",isequal(size(T),size(B)),isequal(size(C),size(B)));
printf("Diferencias maximas: %g %g
   %g\n",max(abs(T(:)-B(:))),max(abs(C(:)-B(:))),0);
system("file m6.txt m6.csv m6.mat");
```

Python. Con NumPy puede comparar con `allclose` o una norma.

```
from pathlib import Path
import numpy as np
A=np.round(np.random.random((6,6)),2)
np.savetxt("m6.txt",A,fmt="%.2f")
```

```
np.savetxt("m6.csv",A,fmt="%.2f",delimiter=",")
np.save("m6.npy",A)
T=np.loadtxt("m6.txt");
   C=np.loadtxt("m6.csv",delimiter=","); N=np.load("m6.npy")
for nombre,M in [("texto",T),("csv",C),("nativo",N)]:
    print(nombre,M.shape,np.max(np.abs(M-A)))
print("Texto legible:",Path("m6.txt").read_text()[:80])
print("Nativo comienza:",Path("m6.npy").read_bytes()[:16])
```

Perl. Recorra las matrices recuperadas y calcule la diferencia máxima, o use la biblioteca numérica disponible.

```
use strict; use warnings; use Storable qw(store retrieve);
my@A=map{[map{int(rand()*101)/100}1..6]}1..6;sub
   guardar{my($f,$sep,$M)=@_;open my$h,">",$f or
   die$!;print$h join($sep,map{sprintf"%.2f",$_}@$_),"\n"
   for@$M;close$h}guardar("m6.txt","
   ",\@A);guardar("m6.csv",",",\@A);store(\@A,"m6.sto");sub
   leer{my($f,$re)=@_;open my$h,"<",$f or
   die$!;my@M;while(<$h>){chomp;push@M,[map{0+$_}split/$re/]}close$h;\@M
   err{my($X,$Y)=@_;my$e=0;for my$i(0..$#$X){for
   my$j(0..$#{$X->[$i]}){my$d=abs($X->[$i][$j]-$Y->[$i][$j]);$e=$d
   if$d>$e}}$e}print"Errores texto=",err(\@A,$T),"
   csv=",err(\@A,$C)," nativo=",err(\@A,$N),"\n";
```

JavaScript. Recorra las matrices recuperadas y calcule la diferencia máxima, o use la biblioteca numérica disponible.

```
"use strict";
const fs=require("node:fs"),v8=require("node:v8");const
   A=Array.from({length:6},()=>Array.from({length:6},()=>Math.round(Math
   save=(f,sep)=>fs.writeFileSync(f,A.map(r=>r.map(x=>x.toFixed(2)).join
   ");save("m6.csv",",");fs.writeFileSync("m6.v8",v8.serialize(A));const
   read=(f,sep)=>fs.readFileSync(f,"utf8").trim().split(/\r?\n/).map(l=>
   err(X,Y){let e=0;for(let i=0;i<X.length;i++)for(let
   j=0;j<X[i].length;j++)e=Math.max(e,Math.abs(X[i][j]-Y[i][j]));return
   e}console.log("Errores:",err(A,T),err(A,C),err(A,N));console.log("Tex
   legible:",fs.readFileSync("m6.txt","utf8").slice(0,80));console.log("
```

19. **Ejercicio:** Ordenamiento de registros por DNI

Prepare un archivo `alumnos.csv` con los campos:

```
apellido,nombre,libreta,dni
```

y al menos ocho registros.

Escriba un programa que:

a) lea y valide la cabecera;

b) interprete cada fila como un registro;

c) valide que el DNI sea un entero positivo;

d) ordene los registros de mayor a menor DNI;

e) escriba `alumnos_ordenados.csv` conservando la cabecera;

f) vuelva a leer el archivo generado y compruebe el orden.

Scilab. Puede usar `csvRead`/`csvWrite` cuando la mezcla de texto y números resulte adecuada, o procesar las líneas explícitamente.

```
lineas=mgetl("alumnos.csv");
if size(lineas,"*")<2 then error("alumnos.csv esta vacio"),
   end
if stripblanks(lineas(1))<>"apellido,nombre,libreta,dni" then
    error("Cabecera inesperada en alumnos.csv")
end

apellido=emptystr(1,0); nombre=emptystr(1,0);
   libreta=emptystr(1,0); dni=[];
for i=2:size(lineas,"*")
    c=tokens(lineas(i),",");
    if size(c,"*")<>4 then
        error(msprintf("Fila %d: se esperaban cuatro
           campos",i))
    end
    if ~isnum(c(4)) then error(msprintf("Fila %d: DNI
       invalido",i)), end
    d=strtod(c(4));
    if d<=0 | d<>floor(d) then error(msprintf("Fila %d: DNI
       invalido",i)), end
    apellido=[apellido c(1)];
    nombre=[nombre c(2)];
    libreta=[libreta c(3)];
    dni=[dni d];
end

[dni_ord,idx]=gsort(dni,"g","d");
fd=mopen("alumnos_ordenados.csv","wt");
if fd<0 then error("No se pudo crear el archivo de salida"),
   end
mfprintf(fd,"apellido,nombre,libreta,dni\n");
for k=1:size(idx,"*")
    i=idx(k);
    mfprintf(fd,"%s,%s,%s,%d\n",apellido(i),nombre(i),libreta(i),dni(i))
end
mclose(fd);

// Verificacion del orden leyendo nuevamente los DNI.
r=mgetl("alumnos_ordenados.csv");
anterior=%inf;
for i=2:size(r,"*")
    c=tokens(r(i),",");
```

```
    actual=strtod(c(4));
    if actual>anterior then error("El archivo no quedo
       ordenado"), end
    anterior=actual;
end
mprintf("Orden descendente verificado en %d
   registros.\n",size(r,"*")-1);
```

Octave. Puede procesar el CSV línea por línea o usar la rutina CSV disponible en la instalación.

```
fid=fopen("alumnos.csv","r"); if fid<0, error("No existe
   alumnos.csv"); endif
cab=strtrim(fgetl(fid)); if
   !strcmp(cab,"apellido,nombre,libreta,dni"),
   error("Cabecera invalida"); endif
C=textscan(fid,"%s%s%s%d","Delimiter",","); fclose(fid);
if any(C{4}<=0), error("DNI invalido"); endif
[~,idx]=sort(C{4},"descend");
fid=fopen("alumnos_ordenados.csv","w");
   fprintf(fid,"%s\n",cab);
for k=idx',
   fprintf(fid,"%s,%s,%s,%d\n",C{1}{k},C{2}{k},C{3}{k},C{4}(k));
   endfor
fclose(fid);
fid=fopen("alumnos_ordenados.csv","r"); fgetl(fid);
   D=textscan(fid,"%s%s%s%d","Delimiter",","); fclose(fid);
assert(all(diff(D{4})<=0));
```

Python. El módulo estándar `csv` permite trabajar con filas o diccionarios y mantener los campos como texto hasta validarlos.

```
import csv
with open("alumnos.csv",newline="",encoding="utf-8") as f:
    r=csv.DictReader(f)
    if r.fieldnames != ["apellido","nombre","libreta","dni"]:
        raise ValueError("Cabecera invalida")
    filas=[]
    for row in r:
        dni=int(row["dni"])
        if dni<=0: raise ValueError("DNI invalido")
        row["dni"]=dni; filas.append(row)
filas.sort(key=lambda r:r["dni"], reverse=True)
with
   open("alumnos_ordenados.csv","w",newline="",encoding="utf-8")
   as f:
    w=csv.DictWriter(f,fieldnames=["apellido","nombre","libreta","dni"])
    w.writeheader(); w.writerows(filas)
with
   open("alumnos_ordenados.csv",newline="",encoding="utf-8")
   as f:
```

```
    d=[int(r["dni"]) for r in csv.DictReader(f)]
assert d==sorted(d,reverse=True)
```

Perl. Para CSV robusto use `Text::CSV` si está disponible; de lo contrario, limite explícitamente el formato aceptado y documente esa restricción.

```
use strict; use warnings; use Text::CSV;
   my$csv=Text::CSV->new({binary=>1});open
   my$f,"<:encoding(UTF-8)","alumnos.csv" or
   die$!;my$h=$csv->getline($f);die"Cabecera invalida\n"
   unless join(",",@$h)
   eq"apellido,nombre,libreta,dni";my@r;while(my$row=$csv->getline($f)){
   invalido\n"
   unless$row->[3]=~/^\d+$/&&$row->[3]>0;push@r,$row}close$f;@r=sort{$b-
   or
   die$!;$csv->say($f,$h);$csv->say($f,$_)for@r;close$f;for(my$i=1;$i<@r
   incorrecto\n" if$r[$i-1][3]<$r[$i][3]}
```

JavaScript. Puede procesar líneas con `fs`; para CSV general use la biblioteca configurada en el curso.

```
"use strict";
const fs=require("node:fs");const
   lines=fs.readFileSync("alumnos.csv","utf8").trim().split(/\r?\n/),cab
   new Error("Cabecera invalida");const
   r=lines.map(l=>{const[a,n,lib,d]=l.split(",");const
   dni=Number(d);if(!Number.isInteger(dni)||dni<=0)throw new
   Error("DNI
   invalido");return{a,n,lib,dni}}).sort((x,y)=>y.dni-x.dni);fs.writeFil
   d=r.map(x=>x.dni);if(d.some((x,i)=>i&&d[i-1]<x))throw new
   Error("Orden incorrecto");
```

20. **Ejercicio:** Tabulación de registros con ancho fijo

    Lea `alumnos.csv` y genere `alumnos_formateado.txt` con el encabezado

```
Apellido   Nombre                Libreta
```

    y una línea por alumno.

    Use campos de ancho fijo:

    - apellido: 10 caracteres;
    - nombre: 20 caracteres;
    - libreta: 7 caracteres.

    Defina qué debe ocurrir cuando un campo exceda su ancho y aplique la misma regla a todos los registros. Compare el archivo de ancho fijo con el CSV.

    Scilab. Use salida formateada con `mfprintf`.

```
function s=recortar(s,n)
    if length(s)>n then s=part(s,1:n); end
endfunction

lineas=mgetl("alumnos.csv");
if stripblanks(lineas(1))<>"apellido,nombre,libreta,dni" then
    error("Cabecera inesperada")
end

fd=mopen("alumnos_formateado.txt","wt");
if fd<0 then error("No se pudo crear
   alumnos_formateado.txt"), end
mfprintf(fd,"%-10s %-20s
   %-7s\n","Apellido","Nombre","Libreta");
mfprintf(fd,"%-10s %-20s
   %-7s\n","----------","--------------------","-------");

for i=2:size(lineas,"*")
    c=tokens(lineas(i),",");
    if size(c,"*")<>4 then error(msprintf("Fila invalida:
       %d",i)), end
    a=recortar(c(1),10);
    n=recortar(c(2),20);
    l=recortar(c(3),7);
    mfprintf(fd,"%-10s %-20s %-7s\n",a,n,l);
end
mclose(fd);

disp(mgetl("alumnos_formateado.txt"));
```

Octave. Use salida formateada con `fprintf`.

```
fid=fopen("alumnos.csv","r"); fgetl(fid);
   C=textscan(fid,"%s%s%s%d","Delimiter",","); fclose(fid);
fid=fopen("alumnos_formateado.txt","w"); fprintf(fid,"%-10s
   %-20s %-7s\n","Apellido","Nombre","Libreta");
for k=1:numel(C{1})
  ap=C{1}{k}(1:min(10,end)); no=C{2}{k}(1:min(20,end));
     li=C{3}{k}(1:min(7,end));
  fprintf(fid,"%-10s %-20s %-7s\n",ap,no,li);
endfor
fclose(fid);
```

Python. Use especificadores de ancho en cadenas de formato.

```
import csv
ANCHO=(10,20,7)
def cortar(s,n): return str(s)[:n]
with open("alumnos.csv",newline="",encoding="utf-8") as f,
   open("alumnos_formateado.txt","w",encoding="utf-8") as g:
    r=csv.DictReader(f)
```

```
    g.write(f"{'Apellido':<10} {'Nombre':<20}
       {'Libreta':<7}\n")
    for x in r:
        g.write(f"{cortar(x['apellido'],10):<10}
           {cortar(x['nombre'],20):<20}
           {cortar(x['libreta'],7):<7}\n")
```

Perl. Use `printf` con especificadores de ancho.

```
use strict; use warnings; use
   Text::CSV;my$c=Text::CSV->new({binary=>1});open
   my$f,"<:encoding(UTF-8)","alumnos.csv" or
   die$!;my$h=$c->getline($f);open
   my$g,">:encoding(UTF-8)","alumnos_formateado.txt" or
   die$!;printf$g"%-10s %-20s
   %-7s\n","Apellido","Nombre","Libreta";while(my$r=$c->getline($f)){my(
   %-20s %-7s\n",$a,$n,$l}close$f;close$g;
```

JavaScript. Use operaciones de cadena como relleno y recorte para construir campos de ancho fijo.

```
"use strict";
const fs=require("node:fs");const
   l=fs.readFileSync("alumnos.csv","utf8").trim().split(/\r?\n/).slice(1
   cut=(s,n)=>s.slice(0,n);let out=`${"Apellido".padEnd(10)}
   ${"Nombre".padEnd(20)}
   ${"Libreta".padEnd(7)}\n`;for(const x of
   l){const[a,n,lib]=x.split(",");out+=`${cut(a,10).padEnd(10)}
   ${cut(n,20).padEnd(20)}
   ${cut(lib,7).padEnd(7)}\n`}fs.writeFileSync("alumnos_formateado.txt",
```

21. **Ejercicio:** Correspondencia generada desde CSV

    Prepare `personas.csv` con los campos:

```
apellido,nombre,dni,calle,numero
```

    Lea el archivo y genere:

    (a) `cartas.txt`.

    (b) `carta.pdf` usando LaTeX.

    Para cada registro escriba un bloque de la forma:

```
Estimado/a Sr./Sra.:

Por favor, confirme si estos datos son correctos:

Nombre: ...
Apellido: ...
```

```
Calle: ...
Numero: ...
Terminacion DNI: ...

Atentamente,
```

La terminación del DNI deberá contener sus tres últimos dígitos, incluyendo ceros iniciales si correspondiera. Valide los registros antes de generar la carta.

Scilab. Lea el CSV como una matriz de cadenas con `csvRead`. Genere simultáneamente `cartas.txt` y `carta.tex`, escapando los caracteres que tienen significado especial en LaTeX. Finalmente ejecute `pdflatex` mediante `host`.

```
function r = latex_escape(s)
    r = s;
    r = strsubst(r, "\", "@@BS@@");
    r = strsubst(r, "&", "\&");
    r = strsubst(r, "%", "\%");
    r = strsubst(r, "$", "\$");
    r = strsubst(r, "#", "\#");
    r = strsubst(r, "_", "\_");
    r = strsubst(r, "{", "\{");
    r = strsubst(r, "}", "\}");
    r = strsubst(r, "~", "\textasciitilde{}");
    r = strsubst(r, "^", "\textasciicircum{}");
    r = strsubst(r, "@@BS@@", "\textbackslash{}");
endfunction

function ok = solo_digitos(s)
    ok = %f;
    if length(s) == 0 then
        return
    end

    c = ascii(s);
    ok = and(c >= ascii("0") & c <= ascii("9"));
endfunction

cabecera = stripblanks(mgetl("personas.csv", 1));

if cabecera <> "apellido,nombre,dni,calle,numero" then
    error("Cabecera inesperada en personas.csv");
end

datos = csvRead( ...
    "personas.csv", ",", ".", "string", ...
    [], [], [], 1);

if size(datos, 2) <> 5 then
    error("El archivo debe contener exactamente cinco
       columnas");
end
```

```
// Validar primero todos los registros.
for i = 1:size(datos, 1)

    for j = 1:5
        datos(i,j) = stripblanks(datos(i,j));
        if datos(i,j) == "" then
            error(msprintf("Campo vacio en registro %d", i));
        end
    end

    dni = datos(i,3);

    if ~solo_digitos(dni) then
        error(msprintf("DNI invalido en registro %d", i));
    end
end

ftxt = mopen("cartas.txt", "wt");
if ftxt < 0 then
    error("No se pudo crear cartas.txt");
end

ftex = mopen("carta.tex", "wt");
if ftex < 0 then
    mclose(ftxt);
    error("No se pudo crear carta.tex");
end

mfprintf(ftex, "\documentclass[11pt]{article}\n");
mfprintf(ftex, "\usepackage[T1]{fontenc}\n");
mfprintf(ftex, "\usepackage[utf8]{inputenc}\n");
mfprintf(ftex, "\usepackage[spanish]{babel}\n");
mfprintf(ftex, "\usepackage[margin=2.5cm]{geometry}\n");
mfprintf(ftex, "\setlength{\parindent}{0pt}\n");
mfprintf(ftex, "\begin{document}\n\n");

for i = 1:size(datos, 1)

    apellido = datos(i,1);
    nombre   = datos(i,2);
    dni      = datos(i,3);
    calle    = datos(i,4);
    numero   = datos(i,5);

    // Se trabaja como cadena para conservar ceros.
    if length(dni) >= 3 then
        ultimos = part(dni, length(dni)-2:length(dni));
    else
        ultimos = part("000", 1:3-length(dni)) + dni;
    end
```

```
    // Archivo de texto.
    mfprintf(ftxt, "Estimado/a Sr./Sra.:\n\n");
    mfprintf(ftxt, ...
        "Por favor, confirme si estos datos son
           correctos:\n\n");

    mfprintf(ftxt, "Nombre: %s\n", nombre);
    mfprintf(ftxt, "Apellido: %s\n", apellido);
    mfprintf(ftxt, "Calle: %s\n", calle);
    mfprintf(ftxt, "Numero: %s\n", numero);
    mfprintf(ftxt, "Terminacion DNI: %s\n\n", ultimos);
    mfprintf(ftxt, "Atentamente,\n\n");
    mfprintf(ftxt,
       "---------------------------------------\n\n");

    // Version LaTeX.
    nombre_l   = latex_escape(nombre);
    apellido_l = latex_escape(apellido);
    calle_l    = latex_escape(calle);
    numero_l   = latex_escape(numero);

    mfprintf(ftex, "Estimado/a Sr./Sra.:\par\medskip\n\n");

    mfprintf(ftex, ...
        "Por favor, confirme si estos datos son
           correctos:\par\medskip\n\n");

    mfprintf(ftex, "\begin{tabular}{@{}ll}\n");
    mfprintf(ftex, "Nombre: & %s \\\\\n", nombre_l);
    mfprintf(ftex, "Apellido: & %s \\\\\n", apellido_l);
    mfprintf(ftex, "Calle: & %s \\\\\n", calle_l);
    mfprintf(ftex, "Numero: & %s \\\\\n", numero_l);
    mfprintf(ftex, "Terminacion DNI: & %s \\\\\n", ultimos);
    mfprintf(ftex, "\end{tabular}\n\n");

    mfprintf(ftex, "\bigskip\n");
    mfprintf(ftex, "Atentamente,\n");
    mfprintf(ftex, "\newpage\n\n");
end

mfprintf(ftex, "\end{document}\n");

mclose(ftxt);
mclose(ftex);

[estado, salida, errores] = host( ...
    "pdflatex -interaction=nonstopmode -halt-on-error
       carta.tex");

if estado <> 0 then
```

```
    error("No se pudo generar carta.pdf");
end
```

Octave. Lea y valide los cinco campos. Genere los archivos `cartas.txt` y `carta.tex`; utilice `system` para ejecutar `pdflatex`.

```
function r = latex_escape(s)
  r = s;
  r = strrep(r, '\', '@@BS@@');
  r = strrep(r, '&', '\&');
  r = strrep(r, '%', '\%');
  r = strrep(r, '$', '\$');
  r = strrep(r, '#', '\#');
  r = strrep(r, '_', '\_');
  r = strrep(r, '{', '\{');
  r = strrep(r, '}', '\}');
  r = strrep(r, '~', '\textasciitilde{}');
  r = strrep(r, '^', '\textasciicircum{}');
  r = strrep(r, '@@BS@@', '\textbackslash{}');
endfunction

fid = fopen("personas.csv", "r");
if fid < 0
  error("No se pudo abrir personas.csv");
endif

cabecera = strtrim(fgetl(fid));

if !strcmp(cabecera, "apellido,nombre,dni,calle,numero")
  fclose(fid);
  error("Cabecera inesperada");
endif

C = textscan(fid, "%q%q%q%q%q", ...
             "Delimiter", ",", ...
             "EndOfLine", "\n");

fclose(fid);

n = numel(C{1});

for i = 1:n
  apellido = strtrim(C{1}{i});
  nombre   = strtrim(C{2}{i});
  dni      = strtrim(C{3}{i});
  calle    = strtrim(C{4}{i});
  numero   = strtrim(C{5}{i});

  if isempty(apellido) || isempty(nombre) || ...
     isempty(dni) || isempty(calle) || isempty(numero)
    error("Registro %d: hay campos vacios", i);
```

```
  endif

  if isempty(regexp(dni, "^[0-9]+$", "once"))
    error("Registro %d: DNI invalido", i);
  endif
endfor

ftxt = fopen("cartas.txt", "w");
if ftxt < 0
  error("No se pudo crear cartas.txt");
endif

ftex = fopen("carta.tex", "w");
if ftex < 0
  fclose(ftxt);
  error("No se pudo crear carta.tex");
endif

fprintf(ftex, "\\documentclass[11pt]{article}\n");
fprintf(ftex, "\\usepackage[T1]{fontenc}\n");
fprintf(ftex, "\\usepackage[utf8]{inputenc}\n");
fprintf(ftex, "\\usepackage[spanish]{babel}\n");
fprintf(ftex, "\\usepackage[margin=2.5cm]{geometry}\n");
fprintf(ftex, "\\setlength{\\parindent}{0pt}\n");
fprintf(ftex, "\\begin{document}\n\n");

for i = 1:n

  apellido = strtrim(C{1}{i});
  nombre   = strtrim(C{2}{i});
  dni      = strtrim(C{3}{i});
  calle    = strtrim(C{4}{i});
  numero   = strtrim(C{5}{i});

  ultimos = ["000" dni];
  ultimos = ultimos(end-2:end);

  fprintf(ftxt, "Estimado/a Sr./Sra.:\n\n");
  fprintf(ftxt, ...
      "Por favor, confirme si estos datos son
         correctos:\n\n");

  fprintf(ftxt, "Nombre: %s\n", nombre);
  fprintf(ftxt, "Apellido: %s\n", apellido);
  fprintf(ftxt, "Calle: %s\n", calle);
  fprintf(ftxt, "Numero: %s\n", numero);
  fprintf(ftxt, "Terminacion DNI: %s\n\n", ultimos);
  fprintf(ftxt, "Atentamente,\n\n");
  fprintf(ftxt,
     "----------------------------------------\n\n");
```

```
  fprintf(ftex, "Estimado/a Sr./Sra.:\\par\\medskip\n\n");

  fprintf(ftex, ...
      "Por favor, confirme si estos datos son correctos:" ...
      "\\par\\medskip\n\n");

  fprintf(ftex, "\\begin{tabular}{@{}ll}\n");
  fprintf(ftex, "Nombre: & %s \\\\\n", ...
          latex_escape(nombre));
  fprintf(ftex, "Apellido: & %s \\\\\n", ...
          latex_escape(apellido));
  fprintf(ftex, "Calle: & %s \\\\\n", ...
          latex_escape(calle));
  fprintf(ftex, "Numero: & %s \\\\\n", ...
          latex_escape(numero));
  fprintf(ftex, "Terminacion DNI: & %s \\\\\n", ultimos);
  fprintf(ftex, "\\end{tabular}\n\n");

  fprintf(ftex, "\\bigskip\n");
  fprintf(ftex, "Atentamente,\n");
  fprintf(ftex, "\\newpage\n\n");
endfor

fprintf(ftex, "\\end{document}\n");

fclose(ftxt);
fclose(ftex);

[estado, salida] = system( ...
  "pdflatex -interaction=nonstopmode -halt-on-error
     carta.tex");

if estado != 0
  error("No se pudo generar carta.pdf:\n%s", salida);
endif
```

Python. Use `csv.DictReader` para interpretar correctamente el CSV. Genere `cartas.txt` y `carta.tex` y ejecute `pdflatex` mediante `subprocess`.

```
import csv
import re
import subprocess

CAMPOS = ["apellido", "nombre", "dni", "calle", "numero"]

def latex_escape(s):
    s = s.replace("\\", "@@BS@@")
    s = s.replace("&", r"\&")
    s = s.replace("%", r"\%")
    s = s.replace("$", r"\$")
    s = s.replace("#", r"\#")
```

```
    s = s.replace("_", r"\_")
    s = s.replace("{", r"\{")
    s = s.replace("}", r"\}")
    s = s.replace("~", r"\textasciitilde{}")
    s = s.replace("^", r"\textasciicircum{}")
    s = s.replace("@@BS@@", r"\textbackslash{}")
    return s

registros = []

with open("personas.csv",
          newline="",
          encoding="utf-8") as f:

    lector = csv.DictReader(f)

    if lector.fieldnames != CAMPOS:
        raise ValueError("Cabecera inesperada en
           personas.csv")

    for numero_fila, r in enumerate(lector, start=2):

        if None in r:
            raise ValueError(
                f"Fila {numero_fila}: numero incorrecto de
                   campos"
            )

        for campo in CAMPOS:
            if r[campo] is None:
                raise ValueError(
                    f"Fila {numero_fila}: falta {campo}"
                )

            r[campo] = r[campo].strip()

            if not r[campo]:
                raise ValueError(
                    f"Fila {numero_fila}: {campo} esta vacio"
                )

        if re.fullmatch(r"[0-9]+", r["dni"]) is None:
            raise ValueError(
                f"Fila {numero_fila}: DNI invalido"
            )

        registros.append(r)

with open("cartas.txt", "w", encoding="utf-8") as txt, \
     open("carta.tex", "w", encoding="utf-8") as tex:
```

```
        tex.write(r"\documentclass[11pt]{article}" "\n")
        tex.write(r"\usepackage[T1]{fontenc}" "\n")
        tex.write(r"\usepackage[utf8]{inputenc}" "\n")
        tex.write(r"\usepackage[spanish]{babel}" "\n")
        tex.write(r"\usepackage[margin=2.5cm]{geometry}" "\n")
        tex.write(r"\setlength{\parindent}{0pt}" "\n")
        tex.write(r"\begin{document}" "\n\n")

        for r in registros:

            ultimos = r["dni"][-3:].zfill(3)

            txt.write("Estimado/a Sr./Sra.:\n\n")
            txt.write(
                "Por favor, confirme si estos datos son
                   correctos:\n\n"
            )
            txt.write(f"Nombre: {r['nombre']}\n")
            txt.write(f"Apellido: {r['apellido']}\n")
            txt.write(f"Calle: {r['calle']}\n")
            txt.write(f"Numero: {r['numero']}\n")
            txt.write(f"Terminacion DNI: {ultimos}\n\n")
            txt.write("Atentamente,\n\n")
            txt.write("----------------------------------------\n\n")

            nombre = latex_escape(r["nombre"])
            apellido = latex_escape(r["apellido"])
            calle = latex_escape(r["calle"])
            numero = latex_escape(r["numero"])

            tex.write(r"Estimado/a Sr./Sra.:\par\medskip" "\n\n")

            tex.write(
                r"Por favor, confirme si estos datos son
                   correctos:"
                r"\par\medskip" "\n\n"
            )

            tex.write(r"\begin{tabular}{@{}ll}" "\n")
            tex.write(f"Nombre: & {nombre} \\\\\n")
            tex.write(f"Apellido: & {apellido} \\\\\n")
            tex.write(f"Calle: & {calle} \\\\\n")
            tex.write(f"Numero: & {numero} \\\\\n")
            tex.write(f"Terminacion DNI: & {ultimos} \\\\\n")
            tex.write(r"\end{tabular}" "\n\n")

            tex.write(r"\bigskip" "\n")
            tex.write("Atentamente,\n")
            tex.write(r"\newpage" "\n\n")

        tex.write(r"\end{document}" "\n")
```

```
subprocess.run(
    [
        "pdflatex",
        "-interaction=nonstopmode",
        "-halt-on-error",
        "carta.tex",
    ],
    check=True,
)
```

Perl. Use `Text::CSV` para leer el archivo de manera estructurada. Valide primero todos los registros, genere las dos representaciones y ejecute `pdflatex` al finalizar.

```
use strict;
use warnings;
use utf8;
use Text::CSV;

sub latex_escape {
    my ($s) = @_;

    $s =~ s/\\/\@\@BS\@\@/g;
    $s =~ s/&/\\&/g;
    $s =~ s/%/\\%/g;
    $s =~ s/\$/\\\$/g;
    $s =~ s/#/\\#/g;
    $s =~ s/_/\\_/g;
    $s =~ s/\{/\\{/g;
    $s =~ s/\}/\\}/g;
    $s =~ s/~/\\textasciitilde{}/g;
    $s =~ s/\^/\\textasciicircum{}/g;
    $s =~ s/\@\@BS\@\@/\\textbackslash{}/g;

    return $s;
}

my @campos =
    qw(apellido nombre dni calle numero);

my $csv = Text::CSV->new({
    binary    => 1,
    auto_diag => 1,
});

open my $fh, "<:encoding(UTF-8)", "personas.csv"
    or die "No se pudo abrir personas.csv: $!";

my $cabecera = $csv->getline($fh)
    or die "Archivo CSV vacio\n";
```

```
die "Cabecera inesperada\n"
    unless @$cabecera == @campos;

for my $i (0 .. $#campos) {
    die "Cabecera inesperada\n"
        unless $cabecera->[$i] eq $campos[$i];
}

my @registros;
my $fila = 1;

while (my $r = $csv->getline($fh)) {

    ++$fila;

    die "Fila $fila: numero incorrecto de campos\n"
        unless @$r == 5;

    for (@$r) {
        s/^\s+//;
        s/\s+$//;

        die "Fila $fila: campo vacio\n"
            if $_ eq "";
    }

    die "Fila $fila: DNI invalido\n"
        unless $r->[2] =~ /^[0-9]+$/;

    push @registros, $r;
}

close $fh;

open my $txt, ">:encoding(UTF-8)", "cartas.txt"
    or die "No se pudo crear cartas.txt: $!";

open my $tex, ">:encoding(UTF-8)", "carta.tex"
    or die "No se pudo crear carta.tex: $!";

print $tex "\\documentclass[11pt]{article}\n";
print $tex "\\usepackage[T1]{fontenc}\n";
print $tex "\\usepackage[utf8]{inputenc}\n";
print $tex "\\usepackage[spanish]{babel}\n";
print $tex "\\usepackage[margin=2.5cm]{geometry}\n";
print $tex "\\setlength{\\parindent}{0pt}\n";
print $tex "\\begin{document}\n\n";

for my $r (@registros) {

    my ($apellido, $nombre, $dni, $calle, $numero) = @$r;
```

```
    my $ultimos = substr("000" . $dni, -3);

    print $txt "Estimado/a Sr./Sra.:\n\n";
    print $txt
        "Por favor, confirme si estos datos son
           correctos:\n\n";

    print $txt "Nombre: $nombre\n";
    print $txt "Apellido: $apellido\n";
    print $txt "Calle: $calle\n";
    print $txt "Numero: $numero\n";
    print $txt "Terminacion DNI: $ultimos\n\n";
    print $txt "Atentamente,\n\n";
    print $txt
       "---------------------------------------\n\n";

    $nombre   = latex_escape($nombre);
    $apellido = latex_escape($apellido);
    $calle    = latex_escape($calle);
    $numero   = latex_escape($numero);

    print $tex "Estimado/a Sr./Sra.:\\par\\medskip\n\n";

    print $tex
        "Por favor, confirme si estos datos son correctos:" .
        "\\par\\medskip\n\n";

    print $tex "\\begin{tabular}{\@{}ll}\n";
    print $tex "Nombre: & $nombre \\\\\n";
    print $tex "Apellido: & $apellido \\\\\n";
    print $tex "Calle: & $calle \\\\\n";
    print $tex "Numero: & $numero \\\\\n";
    print $tex "Terminacion DNI: & $ultimos \\\\\n";
    print $tex "\\end{tabular}\n\n";

    print $tex "\\bigskip\n";
    print $tex "Atentamente,\n";
    print $tex "\\newpage\n\n";
}

print $tex "\\end{document}\n";

close $txt;
close $tex;

system(
    "pdflatex",
    "-interaction=nonstopmode",
    "-halt-on-error",
    "carta.tex"
```

```
) == 0
    or die "No se pudo generar carta.pdf\n";
```

JavaScript. Use un analizador CSV, por ejemplo `csv-parse`, en lugar de dividir las líneas manualmente. Genere `cartas.txt` y `carta.tex`, y ejecute `pdflatex` mediante `child_process`.

```
"use strict";

const fs = require("node:fs");
const { execFileSync } = require("node:child_process");
const { parse } = require("csv-parse/sync");

const CAMPOS =
    ["apellido", "nombre", "dni", "calle", "numero"];

function latexEscape(s) {
    return s
        .replace(/\\/g, "@@BS@@")
        .replace(/&/g, "\\&")
        .replace(/%/g, "\\%")
        .replace(/\$/g, "\\$")
        .replace(/#/g, "\\#")
        .replace(/_/g, "\\_")
        .replace(/\{/g, "\\{")
        .replace(/\}/g, "\\}")
        .replace(/~/g, "\\textasciitilde{}")
        .replace(/\^/g, "\\textasciicircum{}")
        .replace(/@@BS@@/g, "\\textbackslash{}");
}

const contenido =
    fs.readFileSync("personas.csv", "utf8");

const registros = parse(contenido, {
    columns: cabecera => {
        if (
            cabecera.length !== CAMPOS.length ||
            !cabecera.every((x, i) => x === CAMPOS[i])
        ) {
            throw new Error(
                "Cabecera inesperada en personas.csv"
            );
        }

        return cabecera;
    },

    skip_empty_lines: true,
    trim: true,
    bom: true
```

```
});

registros.forEach((r, i) => {

    const fila = i + 2;

    for (const campo of CAMPOS) {
        if (
            typeof r[campo] !== "string" ||
            r[campo].trim() === ""
        ) {
            throw new Error(
                `Fila ${fila}: campo ${campo} invalido`
            );
        }

        r[campo] = r[campo].trim();
    }

    if (!/^[0-9]+$/.test(r.dni)) {
        throw new Error(
            `Fila ${fila}: DNI invalido`
        );
    }
});

let texto = "";

let tex =
`\\documentclass[11pt]{article}
\\usepackage[T1]{fontenc}
\\usepackage[utf8]{inputenc}
\\usepackage[spanish]{babel}
\\usepackage[margin=2.5cm]{geometry}
\\setlength{\\parindent}{0pt}
\\begin{document}

`;

for (const r of registros) {

    const ultimos =
        r.dni.slice(-3).padStart(3, "0");

    texto +=
`Estimado/a Sr./Sra.:

Por favor, confirme si estos datos son correctos:

Nombre: ${r.nombre}
Apellido: ${r.apellido}
```

```
Calle: ${r.calle}
Numero: ${r.numero}
Terminacion DNI: ${ultimos}

Atentamente,

----------------------------------------

`;

    tex +=
`Estimado/a Sr./Sra.:\\par\\medskip

Por favor, confirme si estos datos son
   correctos:\\par\\medskip

\\begin{tabular}{@{}ll}
Nombre: & ${latexEscape(r.nombre)} \\\\
Apellido: & ${latexEscape(r.apellido)} \\\\
Calle: & ${latexEscape(r.calle)} \\\\
Numero: & ${latexEscape(r.numero)} \\\\
Terminacion DNI: & ${ultimos} \\\\
\\end{tabular}

\\bigskip
Atentamente,

\\newpage

`;
}

tex += "\\end{document}\n";

fs.writeFileSync(
    "cartas.txt",
    texto,
    "utf8"
);

fs.writeFileSync(
    "carta.tex",
    tex,
    "utf8"
);

execFileSync(
    "pdflatex",
    [
        "-interaction=nonstopmode",
        "-halt-on-error",
```

```
        "carta.tex"
    ],
    {
        stdio: "inherit"
    }
);
```

22. **Ejercicio:** Normalización de datos tabulares en archivos CSV

    Considere el archivo `mediciones_sensores.csv`:

```
sensor_id,nombre,ubicacion,unidad,tiempo_h,valor
S01,Termometro norte,Laboratorio A,C,0,20.1
S01,Termometro norte,Laboratorio A,C,1,20.4
S01,Termometro norte,Laboratorio A,C,2,20.8
S02,Termometro sur,Laboratorio B,C,0,19.7
S02,Termometro sur,Laboratorio B,C,1,20.0
S02,Termometro sur,Laboratorio B,C,2,20.3
```

    Suponga que cada sensor puede realizar como máximo una medición en un mismo instante.

    a) determine una clave que identifique cada medición;

    b) identifique la dependencia

$$\text{sensor_id} \longrightarrow \{\text{nombre}, \text{ubicacion}, \text{unidad}\};$$

    c) explique las anomalías de actualización, inserción y eliminación producidas por la repetición de los datos del sensor;

    d) normalice la información separándola en `sensores.csv` y `mediciones.csv`;

    e) use `sensor_id` como clave de sensores y (sensor_id, tiempo_h) como clave de mediciones;

    f) agregue a `sensores.csv` un sensor `S03` todavía sin mediciones y explique por qué ahora puede representarse sin inventar una medición;

    g) compruebe mediante un programa que todo `sensor_id` presente en `mediciones.csv` exista también en `sensores.csv`.

    Explique por qué la normalización elimina redundancia pero no información.

    Scilab. Puede leer los CSV completos o línea por línea; para la comprobación referencial compare los identificadores de ambos conjuntos.

```
lineas=mgetl("mediciones_sensores.csv");
esperada="sensor_id,nombre,ubicacion,unidad,tiempo_h,valor";
if stripblanks(lineas(1))<>esperada then error("Cabecera
   inesperada"), end

s_id=emptystr(1,0); s_nombre=emptystr(1,0);
   s_ubic=emptystr(1,0); s_unidad=emptystr(1,0);
```

```
m_id=emptystr(1,0); m_t=[]; m_valor=[];

for i=2:size(lineas,"*")
    c=tokens(lineas(i),",");
    if size(c,"*")<>6 then error(msprintf("Fila %d
       invalida",i)), end
    if ~isnum(c(5)) | ~isnum(c(6)) then error("Tiempo o
       valor invalido"), end
    id=c(1);
    idx=find(s_id==id);
    if size(idx,"*")==0 then
        s_id=[s_id id];
        s_nombre=[s_nombre c(2)];
        s_ubic=[s_ubic c(3)];
        s_unidad=[s_unidad c(4)];
    else
        k=idx(1);
        if s_nombre(k)<>c(2) | s_ubic(k)<>c(3) |
           s_unidad(k)<>c(4) then
            error("Anomalia: un mismo sensor tiene datos
               descriptivos distintos")
        end
    end
    tiempo=strtod(c(5));
    // La clave de mediciones es (sensor_id, tiempo_h).
    repetida=find((m_id==id) & (m_t==tiempo));
    if size(repetida,"*")>0 then
        error("Clave duplicada en mediciones: "+id+" /
           "+c(5))
    end
    m_id=[m_id id];
    m_t=[m_t tiempo];
    m_valor=[m_valor strtod(c(6))];
end

// S03 existe aunque aun no tenga mediciones.
s_id=[s_id "S03"];
s_nombre=[s_nombre "Termometro exterior"];
s_ubic=[s_ubic "Patio"];
s_unidad=[s_unidad "C"];

fd=mopen("sensores.csv","wt");
mfprintf(fd,"sensor_id,nombre,ubicacion,unidad\n");
for i=1:size(s_id,"*")
    mfprintf(fd,"%s,%s,%s,%s\n",s_id(i),s_nombre(i),s_ubic(i),s_unidad(i
end
mclose(fd);

fd=mopen("mediciones.csv","wt");
mfprintf(fd,"sensor_id,tiempo_h,valor\n");
for i=1:size(m_id,"*")
```

```
    mfprintf(fd,"%s,%.12g,%.12g\n",m_id(i),m_t(i),m_valor(i));
end
mclose(fd);

// Comprobacion referencial.
for i=1:size(m_id,"*")
    if size(find(s_id==m_id(i)),"*")==0 then
        error("Medicion con sensor inexistente: "+m_id(i))
    end
end
disp("Normalizacion e integridad referencial verificadas.");
```

Octave. Puede leer los CSV con las herramientas disponibles o procesarlos como texto; mantenga los identificadores como cadenas.

```
fid=fopen("mediciones_sensores.csv","r"); fgetl(fid);
C=textscan(fid,"%s%s%s%s%f%f","Delimiter",","); fclose(fid);
ids=C{1}; nombres=C{2}; ubic=C{3}; unidades=C{4};
   tiempos=C{5}; valores=C{6};
[uids,ia]=unique(ids,"stable");
fid=fopen("sensores.csv","w");
   fprintf(fid,"sensor_id,nombre,ubicacion,unidad\n");
for k=1:numel(uids), i=ia(k);
   fprintf(fid,"%s,%s,%s,%s\n",ids{i},nombres{i},ubic{i},unidades{i});
   endfor
fprintf(fid,"S03,Termometro exterior,Patio,C\n");
   fclose(fid);
fid=fopen("mediciones.csv","w");
   fprintf(fid,"sensor_id,tiempo_h,valor\n");
for i=1:numel(ids),
   fprintf(fid,"%s,%g,%g\n",ids{i},tiempos(i),valores(i));
   endfor
fclose(fid);
sensores=[uids; {"S03"}];
for i=1:numel(ids), assert(any(strcmp(sensores,ids{i})));
   endfor
disp("Integridad referencial verificada");
```

Python. El módulo `csv` permite mantener separadas las dos relaciones; un `set` de identificadores facilita la comprobación referencial.

```
import csv
sensores={}; mediciones=[]
with
   open("mediciones_sensores.csv",newline="",encoding="utf-8")
   as f:
    r=csv.DictReader(f)
    for row in r:
        sid=row["sensor_id"]
        desc=(row["nombre"],row["ubicacion"],row["unidad"])
        if sid in sensores and sensores[sid]!=desc:
```

```
            raise ValueError(f"Datos contradictorios para
                {sid}")
        sensores[sid]=desc
        mediciones.append((sid,float(row["tiempo_h"]),float(row["valor"]
sensores["S03"]=("Termometro exterior","Patio","C")
with open("sensores.csv","w",newline="",encoding="utf-8") as
    f:
     w=csv.writer(f);
        w.writerow(["sensor_id","nombre","ubicacion","unidad"])
     for sid,(n,u,un) in sensores.items():
        w.writerow([sid,n,u,un])
with open("mediciones.csv","w",newline="",encoding="utf-8")
    as f:
     w=csv.writer(f);
        w.writerow(["sensor_id","tiempo_h","valor"]);
        w.writerows(mediciones)
for sid,_,_ in mediciones:
     assert sid in sensores
print("Integridad referencial verificada")
```

Perl. Use `Text::CSV` si está disponible y un hash para indexar los sensores por identificador.

```
use strict; use warnings; use
    Text::CSV;my$c=Text::CSV->new({binary=>1});open
    my$f,"<:encoding(UTF-8)","mediciones_sensores.csv" or
    die$!;my$h=$c->getline($f);my%ix;@ix{@$h}=(0..$#$h);my(%s,@m);while(m
    ubicacion unidad)}]);die"Datos contradictorios para
    $id\n" if
    exists$s{$id}&&$s{$id}ne$d;$s{$id}=$d;push@m,[$id,$r->[$ix{tiempo_h}]
    exterior","Patio","C");open$f,">:encoding(UTF-8)","sensores.csv"
    or die$!;$c->say($f,[qw(sensor_id nombre ubicacion
    unidad)]);for my$id(sort
    keys%s){$c->say($f,[$id,split(/\x1f/,$s{$id})])}close$f;open$f,">:enc
    or die$!;$c->say($f,[qw(sensor_id tiempo_h
    valor)]);$c->say($f,$_)for@m;close$f;die"Integridad
    rota\n" if grep{!exists$s{$_->[0]}}@m;print"Integridad
    referencial verificada\n";
```

JavaScript. Use la biblioteca CSV configurada y un `Set` o `Map` para indexar los identificadores.

```
"use strict";
const fs=require("node:fs");const
    L=fs.readFileSync("mediciones_sensores.csv","utf8").trim().split(/\r?
    Map(),med=[];for(const l of L){const
    r=l.split(","),id=r[ix.sensor_id],d=[r[ix.nombre],r[ix.ubicacion],r[i
    new Error(`Datos contradictorios
    ${id}`);sens.set(id,d);med.push([id,Number(r[ix.tiempo_h]),Number(r[i
    exterior","Patio","C"]);fs.writeFileSync("sensores.csv","sensor_id,no
    med)if(!sens.has(id))throw new Error("Integridad
```

```
    rota");console.log("Integridad referencial verificada");
```

23. **Ejercicio:** Consultas relacionales sobre dos archivos CSV

    Utilice `sensores.csv` y `mediciones.csv` del ejercicio anterior.

    Realice el ejercicio de dos maneras:

    a) Utilice desde cada lenguaje una biblioteca o módulo que permita ejecutar consultas SQL, cuando exista una alternativa adecuada.

    b) Utilice DuckDB, u otro programa equivalente, como ejecutable independiente. En este caso el gestor SQL actuará como una herramienta externa común a todos los lenguajes de trabajo.

    Realice consultas SQL que:

    a) muestren cada archivo como una relación;

    b) unan ambas relaciones mediante `sensor_id`;

    c) seleccionen nombre, ubicación, tiempo y valor;

    d) filtren las mediciones de una ubicación determinada;

    e) ordenen las mediciones de mayor a menor valor;

    f) calculen para cada sensor cantidad, promedio, mínimo y máximo;

    g) mediante `LEFT JOIN`, incluyan también sensores sin mediciones;

    h) detecten identificadores presentes en mediciones pero ausentes en sensores;

    i) generen mediante `COPY` un nuevo archivo `resumen_sensores.csv`.

    Consultas SQL

    Las consultas siguientes son independientes del lenguaje utilizado. Guárdelas en el archivo `consulta.sql`.

```sql
SELECT *
FROM 'sensores.csv';

SELECT *
FROM 'mediciones.csv';

SELECT
    s.nombre,
    s.ubicacion,
    m.tiempo_h,
    m.valor
FROM 'mediciones.csv' AS m
JOIN 'sensores.csv' AS s
    ON m.sensor_id = s.sensor_id;

SELECT
    s.nombre,
```

```
    s.ubicacion,
    m.tiempo_h,
    m.valor
FROM 'mediciones.csv' AS m
JOIN 'sensores.csv' AS s
    ON m.sensor_id = s.sensor_id
WHERE s.ubicacion = 'Laboratorio A';

SELECT
    s.nombre,
    s.ubicacion,
    m.tiempo_h,
    m.valor
FROM 'mediciones.csv' AS m
JOIN 'sensores.csv' AS s
    ON m.sensor_id = s.sensor_id
ORDER BY m.valor DESC;

SELECT
    s.sensor_id,
    s.nombre,
    COUNT(m.valor) AS cantidad,
    AVG(m.valor) AS promedio,
    MIN(m.valor) AS minimo,
    MAX(m.valor) AS maximo
FROM 'sensores.csv' AS s
JOIN 'mediciones.csv' AS m
    ON s.sensor_id = m.sensor_id
GROUP BY
    s.sensor_id,
    s.nombre
ORDER BY
    s.sensor_id;

SELECT
    s.sensor_id,
    s.nombre,
    COUNT(m.valor) AS cantidad,
    AVG(m.valor) AS promedio,
    MIN(m.valor) AS minimo,
    MAX(m.valor) AS maximo
FROM 'sensores.csv' AS s
LEFT JOIN 'mediciones.csv' AS m
    ON s.sensor_id = m.sensor_id
GROUP BY
    s.sensor_id,
    s.nombre
ORDER BY
    s.sensor_id;

SELECT DISTINCT
```

```
    m.sensor_id
FROM 'mediciones.csv' AS m
LEFT JOIN 'sensores.csv' AS s
    ON m.sensor_id = s.sensor_id
WHERE s.sensor_id IS NULL;

COPY (
    SELECT
        s.sensor_id,
        s.nombre,
        s.ubicacion,
        COUNT(m.valor) AS cantidad,
        AVG(m.valor) AS promedio,
        MIN(m.valor) AS minimo,
        MAX(m.valor) AS maximo
    FROM 'sensores.csv' AS s
    LEFT JOIN 'mediciones.csv' AS m
        ON s.sensor_id = m.sensor_id
    GROUP BY
        s.sensor_id,
        s.nombre,
        s.ubicacion
    ORDER BY
        s.sensor_id
)
TO 'resumen_sensores.csv'
(FORMAT CSV, HEADER);
```

Compruebe que las mismas sentencias SQL pueden ser ejecutadas tanto por una biblioteca incorporada al lenguaje como por un gestor SQL ejecutado como programa independiente.

Scilab. Para ejecutar DuckDB dentro del proceso de Scilab puede utilizar el controlador JDBC de DuckDB mediante la interfaz Java. El mismo `consulta.sql` puede ejecutarse también con el programa externo `duckdb` mediante `host`.

```
// Leer el mismo archivo SQL que usan todos los lenguajes.
lineas = mgetl("consulta.sql");
sql = strcat(lineas, ascii(10));

// En este archivo las sentencias estan separadas por punto
   y coma.
consultas = tokens(sql, ";");

//
   ------------------------------------------------------------
// a) DuckDB mediante JDBC.
//
   ------------------------------------------------------------

javaclasspath("duckdb_jdbc.jar");
jimport java.sql.DriverManager;
```

```
conexion = jinvoke(DriverManager, ..
                   "getConnection", ..
                   "jdbc:duckdb:");

sentencia = jinvoke(conexion, "createStatement");

for k = 1:size(consultas, "*")

    q = stripblanks(consultas(k));

    if q <> "" then

        mprintf("\nConsulta %d\n", k);

        hay_resultado = jinvoke(sentencia, "execute", q);

        if hay_resultado then

            rs = jinvoke(sentencia, "getResultSet");
            md = jinvoke(rs, "getMetaData");
            nc = jinvoke(md, "getColumnCount");

            for j = 1:nc
                mprintf("%s", jinvoke(md, "getColumnLabel",
                   j));

                if j < nc then
                    mprintf(" | ");
                end
            end

            mprintf("\n");

            while jinvoke(rs, "next")

                for j = 1:nc

                    valor = jinvoke(rs, "getString", j);

                    if jinvoke(rs, "wasNull") then
                        valor = "NULL";
                    end

                    mprintf("%s", valor);

                    if j < nc then
                        mprintf(" | ");
                    end
                end
```

```
                mprintf("\n");
            end

            jinvoke(rs, "close");
        end
    end
end

jinvoke(sentencia, "close");
jinvoke(conexion, "close");


//
   ---------------------------------------------------------------
// b) DuckDB como programa externo.
//
   ---------------------------------------------------------------

[estado, salida, error_sistema] = ..
    host("duckdb -csv :memory: < consulta.sql");

if estado <> 0 then
    mfprintf(0, "DuckDB termino con codigo %d\n", estado);
    mfprintf(0, "%s\n", error_sistema);
    error("Fallo la ejecucion de DuckDB");
end

disp(salida);

disp("Archivo generado:");
disp(mgetl("resumen_sensores.csv"));
```

Octave. Octave dispone del paquete `odbc`. Con el controlador ODBC de DuckDB configurado, el mismo `consulta.sql` puede ejecutarse desde Octave. Para la segunda variante utilice `system`.

```
pkg load odbc

sql = fileread("consulta.sql");
consultas = strsplit(sql, ";");

%
   ---------------------------------------------------------------
% a) Paquete ODBC.
% Se supone configurado un DSN llamado DuckDB.
%
   ---------------------------------------------------------------

db = odbc("DuckDB", "", "");

for k = 1:numel(consultas)
```

```
  q = strtrim(consultas{k});

  if isempty(q)
    continue;
  endif

  printf("\nConsulta %d\n", k);

  if strncmpi(q, "SELECT", 6)
    datos = fetch(db, q);
    disp(datos);
  else
    execute(db, q);
  endif
endfor

close(db);


%
   ------------------------------------------------------------
% b) DuckDB como ejecutable externo.
%
   ------------------------------------------------------------

[estado, salida] = ...
    system("duckdb -csv :memory: < consulta.sql");

if estado != 0
  error("DuckDB termino con codigo %d", estado);
endif

disp(salida);

printf("\nArchivo generado:\n");
type resumen_sensores.csv
```

Python. Python dispone del módulo oficial `duckdb`. Lea `consulta.sql` y ejecute sus sentencias mediante una conexión DuckDB. Para la segunda variante utilice `subprocess`.

```
from pathlib import Path
import subprocess
import duckdb

sql = Path("consulta.sql").read_text(encoding="utf-8")

consultas = [
    q.strip()
    for q in sql.split(";")
```

```
    if q.strip()
]

#
    ------------------------------------------------------------
# a) Modulo duckdb.
#
    ------------------------------------------------------------

con = duckdb.connect()

for numero, consulta in enumerate(consultas, start=1):

    print(f"\nConsulta {numero}")

    if consulta.lstrip().upper().startswith("SELECT"):

        resultado = con.execute(consulta)

        nombres = [
            columna[0]
            for columna in resultado.description
        ]

        print(" | ".join(nombres))

        for fila in resultado.fetchall():
            print(
                " | ".join(
                    "NULL" if x is None else str(x)
                    for x in fila
                )
            )

    else:
        con.execute(consulta)

con.close()


#
    ------------------------------------------------------------
# b) DuckDB como programa externo.
#
    ------------------------------------------------------------

proceso = subprocess.run(
    ["duckdb", "-csv", ":memory:"],
    input=sql,
    text=True,
    capture_output=True
```

```
)

if proceso.returncode != 0:
    raise RuntimeError(
        f"DuckDB termino con codigo "
        f"{proceso.returncode}\n"
        f"{proceso.stderr}"
    )

print(proceso.stdout)

print(
    Path("resumen_sensores.csv")
        .read_text(encoding="utf-8")
)
```

Perl. Perl puede utilizar la interfaz estándar `DBI` con `DBD::DuckDB`. Para la ejecución de DuckDB como proceso independiente puede emplearse `IPC::Open3`.

```
use strict;
use warnings;
use DBI;
use IPC::Open3;
use Symbol qw(gensym);

open my $fh, "<", "consulta.sql"
    or die "No se pudo abrir consulta.sql: $!";

local $/;
my $sql = <$fh>;
close $fh;

my @consultas =
    grep { length $_ }
    map {
        s/^\s+|\s+$//gr
    }
    split /;/, $sql;


#
    ------------------------------------------------------------
# a) DBI + DBD::DuckDB.
#
    ------------------------------------------------------------

my $dbh = DBI->connect(
    "dbi:DuckDB:dbname=:memory:",
    "",
    "",
    {
```

```
        RaiseError => 1,
        PrintError => 0,
    }
);

for my $i (0 .. $#consultas) {

    my $consulta = $consultas[$i];

    print "\nConsulta ", $i + 1, "\n";

    if ($consulta =~ /^\s*SELECT\b/i) {

        my $sth = $dbh->prepare($consulta);
        $sth->execute();

        print join(" | ", @{$sth->{NAME}}), "\n";

        while (my @fila = $sth->fetchrow_array()) {

            @fila = map {
                defined $_ ? $_ : "NULL"
            } @fila;

            print join(" | ", @fila), "\n";
        }

        $sth->finish();

    } else {

        $dbh->do($consulta);
    }
}

$dbh->disconnect();


#
    ------------------------------------------------------------
# b) DuckDB como programa externo.
#
    ------------------------------------------------------------

my $error = gensym;

my $pid = open3(
    my $entrada,
    my $salida,
    $error,
    "duckdb",
```

```
    "-csv",
    ":memory:"
);

print {$entrada} $sql;
close $entrada;

my $stdout = do {
    local $/;
    <$salida>
};

my $stderr = do {
    local $/;
    <$error>
};

waitpid($pid, 0);

my $estado = $? >> 8;

if ($estado != 0) {
    die "DuckDB termino con codigo $estado\n$stderr";
}

print $stdout;
```

JavaScript. Node.js dispone del cliente oficial `@duckdb/node-api`. El programa puede guardarse como `consulta.mjs`. Para ejecutar DuckDB como proceso independiente utilice `child_process`.

```
import { DuckDBConnection } from "@duckdb/node-api";

import {
    readFileSync
} from "node:fs";

import {
    spawnSync
} from "node:child_process";

const sql =
    readFileSync("consulta.sql", "utf8");

const consultas = sql
    .split(";")
    .map(q => q.trim())
    .filter(q => q.length !== 0);


//
```

```
    ------------------------------------------------------------
// a) Cliente DuckDB para Node.js.
//
    ------------------------------------------------------------

const conexion =
    await DuckDBConnection.create();

for (let i = 0; i < consultas.length; i++) {

    const consulta = consultas[i];

    console.log(
        `\nConsulta ${i + 1}`
    );

    if (
        consulta
            .trimStart()
            .toUpperCase()
            .startsWith("SELECT")
    ) {

        const resultado =
            await conexion.runAndReadAll(
                consulta
            );

        console.table(
            resultado.getRowObjectsJson()
        );

    } else {

        await conexion.run(consulta);
    }
}

conexion.closeSync();


//
    ------------------------------------------------------------
// b) DuckDB como ejecutable independiente.
//
    ------------------------------------------------------------

const proceso = spawnSync(
    "duckdb",
    ["-csv", ":memory:"],
    {
```

```
        input: sql,
        encoding: "utf8"
    }
);

if (proceso.error) {
    throw proceso.error;
}

if (proceso.status !== 0) {

    throw new Error(
        `DuckDB termino con codigo ${proceso.status}\n`
        + proceso.stderr
    );
}

console.log(proceso.stdout);

console.log(
    readFileSync(
        "resumen_sensores.csv",
        "utf8"
    )
);
```

24. **Ejercicio:** Interoperabilidad entre formatos de matrices

Considere

$$A = \begin{pmatrix} 1{,}1 & 2{,}2 & 3{,}3 \\ 4{,}4 & 5{,}5 & 6{,}6 \\ 7{,}7 & 8{,}8 & 9{,}9 \end{pmatrix}.$$

Guarde la misma matriz en:

a) un archivo de texto tabular con cuatro cifras decimales;

b) un archivo CSV con cabecera `col1,col2,col3`;

c) un formato nativo o binario del entorno.

Luego recupere las tres representaciones y:

a) compare los tamaños de los archivos;

b) verifique que los valores se recuperan correctamente;

c) abra los formatos textuales con un editor y compare su legibilidad;

d) determine cuál elegiría para intercambiar datos con otro lenguaje y cuál para conservar resultados intermedios dentro del mismo entorno;

e) intercambie el CSV con, al menos, una implementación realizada en otro de los lenguajes del curso y compruebe que ambas leen los mismos valores.

Scilab. Use las rutinas de texto matricial, CSV y `save`/ `load`.

```
A=[1.1 2.2 3.3; 4.4 5.5 6.6; 7.7 8.8 9.9];

fprintfMat("matriz.txt",A,"%.4f");
csvWrite(A,"matriz.csv",",",".","%.4f","col1,col2,col3");
save("matriz.sod","A");

T=fscanfMat("matriz.txt");
C=csvRead("matriz.csv",",",".","double",[],[],[],1);
A0=A;
clear A;
load("matriz.sod","A");
N=A;

mprintf("Error texto : %.3e\n",max(abs(T-A0)));
mprintf("Error CSV   : %.3e\n",max(abs(C-A0)));
mprintf("Error nativo: %.3e\n",max(abs(N-A0)));

[estado,salida,error_sistema]=host("wc -c matriz.txt
   matriz.csv matriz.sod");
if estado==0 then disp(salida), else disp(error_sistema), end

// Comprobacion de intercambio con otro lenguaje: Python lee
   el mismo CSV.
py=[ ..
"import csv"; ..
"with open('matriz.csv') as f:"; ..
"    r=list(csv.reader(f))"; ..
"print(r[0])"; ..
"print([[float(x) for x in fila] for fila in r[1:]])" ..
];
mputl(py,"verifica_csv.py");
[estado,salida,error_sistema]=host("python3
   verifica_csv.py");
if estado==0 then disp(salida), else disp(error_sistema), end
```

Octave. Use las rutinas de texto/CSV disponibles y `save`/ `load`.

```
A=[1.1 2.2 3.3;4.4 5.5 6.6;7.7 8.8 9.9];
dlmwrite("matriz.txt",A,"delimiter"," ","precision","%.4f");
fid=fopen("matriz.csv","w"); fprintf(fid,"col1,col2,col3\n");
for i=1:3, fprintf(fid,"%.4f,%.4f,%.4f\n",A(i,:)); endfor;
   fclose(fid);
save("matriz.mat","A","-binary");
T=dlmread("matriz.txt"); fid=fopen("matriz.csv","r");
   fgetl(fid);
   C=cell2mat(textscan(fid,"%f%f%f","Delimiter",","));
   fclose(fid);
load("matriz.mat","A");
printf("Errores texto=%g
   csv=%g\n",max(abs(T(:)-A(:))),max(abs(C(:)-A(:))));
```

```
d=dir({"matriz.txt","matriz.csv","matriz.mat"}); for
   k=1:numel(d), printf("%s: %d
   bytes\n",d(k).name,d(k).bytes); endfor
```

Python. Con NumPy puede usar `savetxt`/`loadtxt` y `save`/`load`.

```
from pathlib import Path
import numpy as np
A=np.array([[1.1,2.2,3.3],[4.4,5.5,6.6],[7.7,8.8,9.9]])
np.savetxt("matriz.txt",A,fmt="%.4f")
np.savetxt("matriz.csv",A,fmt="%.4f",delimiter=",",header="col1,col2,col
np.save("matriz.npy",A)
T=np.loadtxt("matriz.txt");
   C=np.loadtxt("matriz.csv",delimiter=",",skiprows=1);
   N=np.load("matriz.npy")
for p in ["matriz.txt","matriz.csv","matriz.npy"]:
    print(p,Path(p).stat().st_size,"bytes")
print("Errores:",np.max(abs(T-A)),np.max(abs(C-A)),np.max(abs(N-A)))
```

Perl. Escriba el CSV de acuerdo con el formato común; para el formato nativo use el mecanismo de serialización configurado.

```
use strict; use warnings; use Storable qw(store
   retrieve);use
   File::stat;my$A=[[1.1,2.2,3.3],[4.4,5.5,6.6],[7.7,8.8,9.9]];open
   my$t,">","matriz.txt" or die$!;print$t join("
   ",map{sprintf"%.4f",$_}@$_),"\n" for@$A;close$t;open
   my$c,">","matriz.csv" or
   die$!;print$c"col1,col2,col3\n";print$c
   join(",",map{sprintf"%.4f",$_}@$_),"\n"
   for@$A;close$c;store($A,"matriz.sto");print"matriz.txt:
   ",stat("matriz.txt")->size," bytes\nmatriz.csv:
   ",stat("matriz.csv")->size," bytes\nmatriz.sto:
   ",stat("matriz.sto")->size,"
   bytes\n";my$N=retrieve("matriz.sto");print"Nativo
   recuperado: ",join(" ",@{$N->[0]})," ...\n";
```

JavaScript. Escriba el CSV de acuerdo con el formato común; para el formato nativo use el mecanismo binario/serializado configurado.

```
"use strict";
const fs=require("node:fs"),v8=require("node:v8");const
   A=[[1.1,2.2,3.3],[4.4,5.5,6.6],[7.7,8.8,9.9]];fs.writeFileSync("matri
   ")).join("\n")+"\n");fs.writeFileSync("matriz.csv","col1,col2,col3\n"
   f
   of["matriz.txt","matriz.csv","matriz.v8"])console.log(f,fs.statSync(f
   N=v8.deserialize(fs.readFileSync("matriz.v8"));console.log("Primera
   fila recuperada:",N[0]);
```

6.2.4. Representación y estándares de archivos

25. **Ejercicio:** Binarios frente a archivos de texto

Considere el vector

$$v = (1{,}0,\ 2{,}5,\ 3{,}14159,\ -7{,}2,\ 0{,}001).$$

Guárdelo de dos maneras:

a) en `datos.bin`, codificando cada número como un valor IEEE 754 de 64 bits en un orden de bytes previamente elegido y documentado;

b) en `datos.txt`, una línea por número con diez cifras decimales.

Lea nuevamente ambos archivos y compare los resultados.

a) compare el tamaño de ambos archivos;

b) determine cuántos bytes ocupa cada número en la representación binaria;

c) calcule la diferencia máxima entre los valores originales y recuperados;

d) examine `datos.bin` con `xxd` u otro visor hexadecimal;

e) explique cuándo resulta preferible una representación textual y cuándo una binaria.

Scilab. Use apertura binaria y las rutinas de lectura/escritura que permiten especificar el tipo numérico; documente el orden de bytes empleado.

```
v=[1.0 2.5 3.14159 -7.2 0.001];

// IEEE 754 double, little endian: 8 bytes por numero.
fd=mopen("datos.bin","wb");
if fd<0 then error("No se pudo abrir datos.bin"), end
mput(v,"dl",fd);
mclose(fd);

fd=mopen("datos.txt","wt");
if fd<0 then error("No se pudo abrir datos.txt"), end
for i=1:size(v,"*")
    mfprintf(fd,"%.10f\n",v(i));
end
mclose(fd);

fd=mopen("datos.bin","rb");
vb=mget(size(v,"*"),"dl",fd);
mclose(fd);

vt=zeros(v);
lineas=mgetl("datos.txt");
for i=1:size(v,"*")
    vt(i)=strtod(lineas(i));
```

```
end

mprintf("Bytes teoricos binarios: %d\n",8*size(v,"*"));
mprintf("Diferencia maxima binaria: %.3e\n",max(abs(vb-v)));
mprintf("Diferencia maxima texto   : %.3e\n",max(abs(vt-v)));

[estado,salida,error_sistema]=host("wc -c datos.bin
   datos.txt; xxd datos.bin");
if estado==0 then disp(salida), else disp(error_sistema), end
```

Octave. Use `fwrite`/`fread` con tipo `double` y controle la arquitectura/orden de bytes cuando corresponda.

```
v=[1.0 2.5 3.14159 -7.2 0.001];
fid=fopen("datos.bin","wb","ieee-le");
   fwrite(fid,v,"double"); fclose(fid);
fid=fopen("datos.txt","w"); fprintf(fid,"%.10f\n",v);
   fclose(fid);
fid=fopen("datos.bin","rb","ieee-le");
   vb=fread(fid,Inf,"double")'; fclose(fid);
vt=dlmread("datos.txt")';
printf("Max dif bin=%g
   texto=%g\n",max(abs(v-vb)),max(abs(v-vt)));
db=dir("datos.bin"); dt=dir("datos.txt"); printf("bin=%d
   bytes, txt=%d bytes, %.0f bytes/numero
   binario\n",db.bytes,dt.bytes,db.bytes/numel(v));
system("xxd datos.bin | head");
```

Python. Puede usar el módulo `struct` para fijar explícitamente tipo y endianidad, o NumPy con un `dtype` explícito.

```
from pathlib import Path
import struct
v=[1.0,2.5,3.14159,-7.2,0.001]
Path("datos.bin").write_bytes(struct.pack("<5d",*v))
with open("datos.txt","w",encoding="ascii") as f:
    for x in v: f.write(f"{x:.10f}\n")
vb=list(struct.unpack("<5d",Path("datos.bin").read_bytes()))
vt=[float(x) for x in Path("datos.txt").read_text().split()]
print("Max dif bin:",max(abs(a-b) for a,b in zip(v,vb)))
print("Max dif txt:",max(abs(a-b) for a,b in zip(v,vt)))
print("Tamanos:",Path("datos.bin").stat().st_size,Path("datos.txt").stat
print("Bytes por numero
   binario:",Path("datos.bin").stat().st_size/len(v))
```

Perl. Use `pack`/`unpack` con una plantilla que fije la representación acordada.

```
use strict; use warnings;use
   File::stat;my@v=(1.0,2.5,3.14159,-7.2,0.001);open
   my$b,">:raw","datos.bin" or die$!;print$b
   pack("d<*",@v);close$b;open my$t,">","datos.txt" or
```

```
    die$!;printf$t"%.10f\n",$_
    for@v;close$t;open$b,"<:raw","datos.bin" or
    die$!;local$/;my$raw=<$b>;close$b;my@vb=unpack("d<*",$raw);open$t,"<"
    or die$!;my@vt=map{0+$_}<$t>;close$t;my$e1=0;my$e2=0;for
    my$i(0..$#v){$e1=abs($v[$i]-$vb[$i])if
    abs($v[$i]-$vb[$i])>$e1;$e2=abs($v[$i]-$vt[$i])if
    abs($v[$i]-$vt[$i])>$e2}print"Max dif bin=$e1
    texto=$e2\nTamanos: ",stat("datos.bin")->size," y
    ",stat("datos.txt")->size," bytes\n";
```

JavaScript. Use `Buffer` y los métodos de lectura/escritura de `double` que especifican endianidad.

```
"use strict";
const fs=require("node:fs");const
   v=[1.0,2.5,3.14159,-7.2,0.001],b=Buffer.alloc(v.length*8);v.forEach((
   rb=fs.readFileSync("datos.bin"),vb=v.map((_,i)=>rb.readDoubleLE(i*8))
   err=a=>Math.max(...a.map((x,i)=>Math.abs(x-v[i])));console.log("Max
   dif
   bin:",err(vb),"texto:",err(vt));console.log("Tamanos:",fs.statSync("d
   bytes/numero:",fs.statSync("datos.bin").size/v.length);
```

26. **Ejercicio:** Acceso aleatorio y actualización de un archivo

Construya un archivo binario `registros.bin` formado por diez registros de tamaño fijo.

Cada registro deberá contener, en este orden:

Campo	Representación	Tamaño
`codigo`	entero con signo de 32 bits	4 bytes
`valor`	número real IEEE 754 de 64 bits	8 bytes
`estado`	entero con signo de 32 bits	4 bytes

Utilice para todos los campos multibyte el mismo orden de bytes, previamente elegido y documentado.

Por lo tanto, cada registro ocupará:

$$4 + 8 + 4 = 16 \text{ bytes.}$$

Genere inicialmente diez registros con códigos consecutivos y valores diferentes.

Realice luego las siguientes operaciones:

a) calcule el tamaño que debería tener `registros.bin` y compárelo con el tamaño real del archivo;

b) solicite un número de registro n comprendido entre 1 y 10;

c) calcule el desplazamiento en bytes del comienzo de ese registro mediante

$$p = (n - 1)\,16;$$

d) desplace la posición corriente del archivo directamente hasta p, sin leer previamente los registros anteriores;

e) lea únicamente el registro seleccionado y muestre sus tres campos;

f) solicite un nuevo valor para el campo `valor`;

g) desplace la posición del archivo hasta el comienzo del campo `valor` de ese mismo registro y modifique solamente sus ocho bytes;

h) vuelva a leer directamente el registro modificado y compruebe que `codigo` y `estado` permanecieron inalterados;

i) lea finalmente los diez registros y compruebe que ningún otro registro fue modificado;

j) muestre, cuando el lenguaje lo permita, la posición corriente antes y después de algunas de las operaciones anteriores.

No se permite resolver la actualización leyendo el archivo completo en memoria y escribiéndolo nuevamente.

Explique la diferencia entre acceso secuencial y acceso aleatorio y por qué los registros de tamaño fijo permiten calcular directamente la posición de un registro mediante su número.

Indique también por qué esta técnica sería más difícil de aplicar directamente a un archivo CSV cuyas líneas poseen longitudes diferentes.

27. **Ejercicio:** Firmas mágicas y tipos de archivo

Explore cómo se identifican los formatos de archivo por su contenido y no solamente por su extensión.

En una terminal GNU/Linux:

```
file /bin/ls
file /etc/passwd
xxd /bin/ls | head -3
head -c 20 documento.pdf | xxd
```

a) ejecute `file` sobre al menos diez archivos de tipos distintos;

b) investigue las firmas iniciales de PDF, PNG, JPEG, ZIP y ELF;

c) examine los primeros bytes de `datos.bin` del ejercicio anterior y explique por qué un archivo de datos crudos puede no poseer una firma;

d) cree un archivo en el formato nativo o serializado de su entorno de trabajo y examine sus primeros bytes;

e) cambie deliberadamente la extensión de una copia de un archivo y compruebe si `file` continúa identificándolo por su contenido;

f) escriba un script de terminal que clasifique por tipo los archivos de un directorio y cuente cuántos hay de cada clase.

Scilab. Para el inciso sobre formato nativo puede usar un archivo creado con `save`.

```
A=rand(3,3);
save("datos_nativos.sod","A");

mprintf("Firmas habituales:\n");
mprintf("PDF : 25 50 44 46 2D        (%%PDF-)\n");
mprintf("PNG : 89 50 4E 47 0D 0A 1A 0A\n");
mprintf("JPEG: FF D8 FF\n");
mprintf("ZIP : 50 4B 03 04\n");
mprintf("ELF : 7F 45 4C 46\n");

comando=[ ..
"file /bin/ls /etc/passwd datos.bin datos.txt
   datos_nativos.sod"; ..
"xxd -l 32 datos_nativos.sod"; ..
"cp /bin/ls ls_disfrazado.txt"; ..
"file ls_disfrazado.txt"; ..
"find /usr/bin -maxdepth 1 -type f | head -10 | xargs file"
   ..
];
for i=1:size(comando,"*")
    [estado,salida,error_sistema]=host(comando(i));
    if estado==0 then disp(salida), else
       disp(error_sistema), end
end

// Clasificacion aproximada de los archivos regulares del
   directorio actual.
script=[ ..
"#!/bin/sh"; ..
"find . -maxdepth 1 -type f -print0 |"; ..
"  xargs -0 file -b | sort | uniq -c | sort -nr" ..
];
mputl(script,"clasificar.sh");
host("chmod +x clasificar.sh");
[estado,salida,error_sistema]=host("./clasificar.sh");
if estado==0 then disp(salida), else disp(error_sistema), end
```

Octave. Para el inciso sobre formato nativo puede usar una de las variantes de `save` y observar el formato elegido.

```
A=rand(3,3); save("nativo_octave.mat","A","-binary");
fid=fopen("nativo_octave.mat","rb");
   bytes=fread(fid,16,"uint8")'; fclose(fid);
printf("Primeros bytes del formato nativo:\n"); printf("%02x
   ",bytes); printf("\n");
system("file nativo_octave.mat");
system("cp nativo_octave.mat falso.txt; file falso.txt");
```

Python. Puede examinar, por ejemplo, un archivo NumPy `.npy` creado por el programa.

```
from pathlib import Path
import numpy as np
A=np.random.random((3,3)); np.save("nativo_python.npy",A)
b=Path("nativo_python.npy").read_bytes()[:16]
print("Primeros bytes:",b.hex(" "))
print("Firma NPY:",b[:6])
Path("falso.txt").write_bytes(Path("nativo_python.npy").read_bytes())
```

Perl. Examine el formato producido por el mecanismo de serialización configurado para el curso.

```
use strict; use warnings; use Storable
   qw(store);my$A=[[1,2],[3,4]];store($A,"nativo_perl.sto");open
   my$f,"<:raw","nativo_perl.sto" or
   die$!;read($f,my$b,16);close$f;print"Primeros bytes:
   ",join("
   ",map{sprintf"%02x",ord$_}split//,$b),"\n";system("file","nativo_perl
```

JavaScript. Examine el formato binario o serializado configurado para el curso.

```
"use strict";
const
   fs=require("node:fs"),v8=require("node:v8"),{spawnSync}=require("node
   A=[[1,2],[3,4]];fs.writeFileSync("nativo_js.v8",v8.serialize(A));cons
   b=fs.readFileSync("nativo_js.v8").subarray(0,16);console.log("Primero
   bytes:",b.toString("hex").match(/../g).join("
   "));spawnSync("file",["nativo_js.v8"],{stdio:"inherit"});fs.copyFileS
```

28. **Ejercicio:** Codificación de texto y finales de línea

Cree un archivo de texto `texto_utf8.txt` codificado en UTF-8 que contenga exactamente las siguientes líneas:

```
Información
Temperatura: 23 °C
Año: 2026
José Pérez
```

Realice las siguientes tareas:

a) abra el archivo como texto y muestre su contenido;

b) determine la cantidad de caracteres de cada línea;

c) abra posteriormente el mismo archivo en modo binario y determine su cantidad total de bytes;

d) examine sus primeros bytes mediante una herramienta hexadecimal como `xxd` y localice la representación UTF-8 de, al menos, los caracteres `ó`, `°` y `ñ`;

e) explique por qué la cantidad de caracteres y la cantidad de bytes no tienen que coincidir;

f) genere una segunda versión, `texto_lf.txt`, utilizando como terminación de línea un único byte LF;

g) genere una tercera versión, `texto_crlf.txt`, utilizando la secuencia CR–LF como terminación de cada línea;

h) compare el tamaño en bytes de ambos archivos y examine sus terminaciones de línea con una herramienta hexadecimal;

i) lea ambos archivos en modo texto desde el lenguaje de trabajo y observe si la biblioteca normaliza las diferencias entre LF y CR–LF;

j) lea finalmente ambos archivos en modo binario y compruebe que en ese caso las diferencias permanecen visibles.

En el informe indique claramente la diferencia entre:

- carácter;
- byte;
- codificación de caracteres;
- fin de línea;
- modo de apertura de texto;
- modo de apertura binario.

El programa no deberá depender de la cantidad de bytes que ocupa un carácter UTF-8 determinado: deberá determinarla examinando realmente los datos almacenados.

6.2.5. Entrada y salida estándar, redirección y tuberías

29. **Ejercicio:** Redirección de entrada y salida estándar

Trabaje en una terminal GNU/Linux con entrada estándar, salida estándar y error estándar.

Ejecute y analice:

```
ls /usr/bin > lista_programas.txt
date >> lista_programas.txt
ls /directorio_que_no_existe 2> errores.txt
ls /usr/bin 1> salida.txt 2> errores.txt
sort < lista_programas.txt > lista_ordenada.txt
cat /etc/passwd | cut -d: -f1 | sort | head -10
```

a) explique la diferencia entre `>` y `>>`;

b) explique qué representan los descriptores 0, 1 y 2;

c) explique por qué una tubería permite componer programas sin crear un archivo intermedio;

d) construya una tubería que liste archivos de una extensión elegida, extraiga sus nombres, los ordene y guarde los primeros veinte;

e) escriba, en el lenguaje de trabajo, un programa mínimo que lea líneas de la entrada estándar, escriba las líneas válidas por la salida estándar y los mensajes de diagnóstico por el error estándar;

f) pruebe ese programa con redirecciones de entrada, salida y error desde la terminal.

Scilab. Use las operaciones disponibles para leer la entrada estándar y escribir en las salidas correspondientes; las redirecciones se realizan desde la terminal antes de iniciar Scilab en modo apropiado.

```
// En la shell: 0=stdin, 1=stdout, 2=stderr.
// > reemplaza el archivo; >> agrega al final.
// En Scilab: %io(1)=stdin, %io(2)=stdout y el descriptor 0
   es stderr.
// Parte de terminal, ejecutada desde Scilab para conservar
   los resultados.
comandos=[ ..
"ls /usr/bin > lista_programas.txt"; ..
"date >> lista_programas.txt"; ..
"ls /directorio_que_no_existe 2> errores.txt"; ..
"ls /usr/bin 1> salida.txt 2> errores2.txt"; ..
"sort < lista_programas.txt > lista_ordenada.txt"; ..
"cat /etc/passwd | cut -d: -f1 | sort | head -10"; ..
"find . -maxdepth 1 -type f -name '*.tex' -printf '%f\n' |
   sort | head -20 > primeros20.txt" ..
];
for i=1:size(comandos,"*")
    [estado,salida,error_sistema]=host(comandos(i));
    if salida<>"" then disp(salida), end
end

// Programa minimo para stdin, stdout y stderr.
// Guardar este bloque como filtro.sce para probar
   redirecciones.
while %t
    linea=mgetl(%io(1),1);
    if size(linea,"*")==0 then break, end
    linea=stripblanks(linea);
    if linea=="" then
        mfprintf(0,"Linea vacia descartada\n");
    else
        mfprintf(%io(2),"%s\n",linea);
    end
end

// Prueba desde la terminal:
// scilab-cli -nb -f filtro.sce < entrada.txt >
   salida_filtro.txt 2> diag.txt
```

Octave. Use las operaciones de consola disponibles; las redirecciones se aplican al proceso desde la terminal.

```
while true
  linea=fgetl(0);
  if !ischar(linea), break; endif
  if isempty(strtrim(linea))
    fprintf(2,"Linea vacia descartada\n");
  else
    fprintf(1,"%s\n",linea);
  endif
endwhile
```

Python. `sys.stdin`, `sys.stdout` y `sys.stderr` representan los tres flujos estándar.

```
import sys
for linea in sys.stdin:
    linea=linea.rstrip("\n")
    if linea.strip():
        print(linea, file=sys.stdout)
    else:
        print("Linea vacia descartada", file=sys.stderr)
```

Perl. Use `STDIN`, `STDOUT` y `STDERR`.

```
use strict; use
   warnings;while(my$l=<STDIN>){chomp$l;if($l=~/\S/){print
   STDOUT "$l\n"}else{print STDERR "Linea vacia
   descartada\n"}}
```

JavaScript. En Node.js use `process.stdin`, `process.stdout` y `process.stderr`.

```
"use strict";
const readline=require("node:readline");const
   rl=readline.createInterface({input:process.stdin,crlfDelay:Infinity})
   process.stderr.write("Linea vacia descartada\n")});
```

6.2.6. El sistema de archivos como interfaz del sistema operativo

30. **Ejercicio:** Archivos especiales de GNU/Linux

    Ejecute en la terminal:

```
cat /etc/hostname
cat /proc/cpuinfo
cat /proc/meminfo
cat /dev/urandom | head -c 16 | xxd
ls -la /dev/
```

    Explique qué hace cada comando y responda:

a) ¿qué directorio contiene los dispositivos representados como archivos?;

b) ¿qué directorio expone información del sistema en ejecución?;

c) ¿en qué sentido esos recursos pueden leerse mediante las mismas operaciones de E/S utilizadas para archivos ordinarios?;

d) ¿qué diferencias observa entre un archivo almacenado en disco, un archivo de `/proc` y un dispositivo de `/dev`?;

e) relacione las observaciones con el principio "todo es un archivo".

Scilab. El núcleo del ejercicio se realiza en la terminal; opcionalmente abra desde Scilab un recurso textual de `/proc` y lea algunas líneas.

```
comandos=[ ..
"cat /etc/hostname"; ..
"head -12 /proc/cpuinfo"; ..
"head -12 /proc/meminfo"; ..
"head -c 16 /dev/urandom | xxd"; ..
"ls -la /dev/ | head -20" ..
];
for i=1:size(comandos,"*")
    [estado,salida,error_sistema]=host(comandos(i));
    if estado==0 then disp(salida), else
       disp(error_sistema), end
end

// /proc se puede leer con la misma E/S textual que un
   archivo ordinario.
lineas=mgetl("/proc/meminfo",5);
disp("Primeras cinco lineas de /proc/meminfo:");
disp(lineas);

mprintf("/dev contiene dispositivos representados mediante
   nodos de archivo.\n");
mprintf("/proc expone informacion virtual generada por el
   nucleo.\n");
mprintf("Un archivo de disco almacena bytes persistentes;
   /proc y /dev\n");
mprintf("son interfaces del nucleo y de dispositivos, no
   archivos ordinarios.\n");
```

Octave. El núcleo del ejercicio se realiza en la terminal; opcionalmente lea desde Octave un recurso textual de `/proc`.

```
fid=fopen("/etc/hostname","r");
   hostname=strtrim(fgetl(fid)); fclose(fid);
printf("Hostname: %s\n",hostname);
fid=fopen("/proc/meminfo","r");
for k=1:5, linea=fgetl(fid); if !ischar(linea), break;
   endif; disp(linea); endfor
fclose(fid);
```

```
fid=fopen("/dev/urandom","rb"); b=fread(fid,16,"uint8");
   fclose(fid);
printf("16 bytes de /dev/urandom: "); printf("%02x ",b);
   printf("\n");
```

Python. El núcleo del ejercicio se realiza en la terminal; opcionalmente use `open` para leer un recurso textual de `/proc`.

```
from pathlib import Path
print("Hostname:",Path("/etc/hostname").read_text().strip())
print("Primeras lineas de /proc/meminfo:")
print("\n".join(Path("/proc/meminfo").read_text().splitlines()[:5]))
with open("/dev/urandom","rb") as f:
    print("16 bytes aleatorios:",f.read(16).hex(" "))
```

Perl. El núcleo del ejercicio se realiza en la terminal; opcionalmente use `open` para leer un recurso textual de `/proc`.

```
use strict; use warnings;open my$h,"<","/etc/hostname" or
   die$!;chomp(my$host=<$h>);close$h;print"Hostname:
   $host\n";open my$m,"<","/proc/meminfo" or
   die$!;for(1..5){my$l=<$m>;last unless
   defined$l;print$l}close$m;open
   my$r,"<:raw","/dev/urandom" or
   die$!;read($r,my$b,16);close$r;print"16 bytes:
   ",unpack("H*",$b),"\n";
```

JavaScript. El núcleo del ejercicio se realiza en la terminal; opcionalmente use `fs` para leer un recurso textual de `/proc`.

```
"use strict";
const
   fs=require("node:fs");console.log("Hostname:",fs.readFileSync("/etc/h
   fd=fs.openSync("/dev/urandom","r"),b=Buffer.alloc(16);fs.readSync(fd,
   bytes:",b.toString("hex").match(/../g).join(" "));
```

31. **Ejercicio:** Hardware representado por el sistema de archivos

    Explore información del hardware y del sistema mediante archivos virtuales y herramientas de GNU/Linux.

```
cat /proc/cpuinfo | grep "model name" | head -1
cat /proc/cpuinfo | grep "cpu MHz" | head -1
cat /proc/meminfo | grep -E "MemTotal|MemFree|MemAvailable"
lsblk
df -h
cat /proc/diskstats | head -5
ip link show
cat /proc/net/dev
```

a) complete una tabla con modelo de CPU, frecuencia, cantidad de núcleos, caché, RAM total y almacenamiento disponible;

b) observe la memoria disponible mientras abre y cierra otras aplicaciones;

c) identifique qué datos provienen de archivos virtuales y cuáles de programas que procesan esos archivos u otras interfaces del núcleo;

d) genere desde la terminal un informe de hardware en `informe_hw.txt`;

e) escriba luego, en el lenguaje de trabajo, un programa que lea al menos dos recursos textuales de `/proc` y genere una parte equivalente del informe.

Scilab. Puede leer archivos textuales de `/proc` con las funciones modernas de lectura de líneas.

```
cpu=mgetl("/proc/cpuinfo");
mem=mgetl("/proc/meminfo");

imodel=grep(cpu,"model name");
imhz=grep(cpu,"cpu MHz");
iproc=grep(cpu,"processor");
icache=grep(cpu,"cache size");
imtotal=grep(mem,"MemTotal");
imfree=grep(mem,"MemFree");
imavail=grep(mem,"MemAvailable");

modelo=cpu(imodel(1));
frecuencia=cpu(imhz(1));
nucleos=size(iproc,"*");
cache=cpu(icache(1));
ram_total=mem(imtotal(1));
ram_libre=mem(imfree(1));
ram_disp=mem(imavail(1));

[estado,disco,error_sistema]=host("df -h / | tail -1");
if estado<>0 then disco="No disponible"; end
[estado,cores_fisicos,error_sistema]=host( ..
    "lscpu -p=CORE 2>/dev/null | grep -v '^#' | sort -u | wc
       -l");
if estado<>0 then cores_fisicos="No disponible"; end

informe=[ ..
"INFORME DE HARDWARE"; ..
"====================="; ..
modelo; ..
frecuencia; ..
msprintf("procesadores logicos : %d",nucleos); ..
"nucleos fisicos      : "+stripblanks(cores_fisicos); ..
cache; ..
ram_total; ..
ram_libre; ..
ram_disp; ..
"almacenamiento raiz : "+stripblanks(disco) ..
```

```
];
mputl(informe,"informe_hw.txt");
disp(informe);
```

Octave. Puede abrir y leer los archivos de `/proc` como archivos de texto ordinarios.

```
cpu=""; fid=fopen("/proc/cpuinfo","r");
while ischar(l=fgetl(fid)), if strncmp(l,"model name",10),
   cpu=strtrim(strsplit(l,":"){2}); break; endif; endwhile;
   fclose(fid);
mem=""; fid=fopen("/proc/meminfo","r");
while ischar(l=fgetl(fid)), if strncmp(l,"MemTotal:",9),
   mem=strtrim(l); break; endif; endwhile; fclose(fid);
fid=fopen("informe_hw.txt","w"); fprintf(fid,"CPU:
   %s\n%s\n",cpu,mem); fclose(fid);
printf("CPU: %s\n%s\n",cpu,mem);
```

Python. Puede usar `pathlib` u `open` y procesar las líneas como texto.

```
from pathlib import Path
cpuinfo=Path("/proc/cpuinfo").read_text().splitlines()
meminfo=Path("/proc/meminfo").read_text().splitlines()
def primero(pref,lineas):
    return next((x.split(":",1)[1].strip() for x in lineas
       if x.startswith(pref)),"N/D")
cpu=primero("model name",cpuinfo); mhz=primero("cpu
   MHz",cpuinfo); mem=primero("MemTotal",meminfo)
informe=f"CPU: {cpu}\nFrecuencia MHz: {mhz}\nMemoria total:
   {mem}\n"
Path("informe_hw.txt").write_text(informe,encoding="utf-8")
print(informe)
```

Perl. Use manejadores de archivo ordinarios; `/proc` se lee como texto.

```
use strict; use warnings;sub primero{my($f,$re)=@_;open
   my$h,"<",$f or
   die$!;while(<$h>){if(/$re/){close$h;return$1}}close$h;return"N/D"}my$
   name\s*:\s*(.+)$/);my$mhz=primero("/proc/cpuinfo",qr/^cpu
   MHz\s*:\s*(.+)$/);my$mem=primero("/proc/meminfo",qr/^MemTotal:\s*(.+)
   $cpu\nFrecuencia MHz: $mhz\nMemoria total: $mem\n";open
   my$f,">","informe_hw.txt" or
   die$!;print$f$rep;close$f;print$rep;
```

JavaScript. Use `fs`; los recursos textuales de `/proc` pueden leerse como archivos.

```
"use strict";
const fs=require("node:fs");const
   cpu=fs.readFileSync("/proc/cpuinfo","utf8").split(/\r?\n/),mem=fs.rea
   get=(a,p)=>(a.find(x=>x.startsWith(p))||"N/D").split(":").slice(1).jo
   rep=`CPU: ${get(cpu,"model name")}\nFrecuencia MHz:
```

```
    ${get(cpu,"cpu MHz")}\nMemoria total:
    ${get(mem,"MemTotal")}\n`;fs.writeFileSync("informe_hw.txt",rep);cons
```

32. **Ejercicio:** Propietarios, grupos y permisos de archivos

    Trabaje inicialmente en una terminal GNU/Linux dentro de un directorio creado especialmente para este ejercicio.

    Determine su identidad y los grupos a los que pertenece mediante:

```
id
groups
```

    Cree luego un archivo y un directorio de prueba:

```
touch datos.txt
mkdir prueba
ls -ld datos.txt prueba
```

    Realice las siguientes tareas:

    a) identifique en la salida de `ls -l` el tipo de objeto, los permisos del propietario, del grupo y de otros, el propietario y el grupo;

    b) interprete simbólica y numéricamente los permisos

    `rw-r-----`

    y

    `rwxr-xr-x`

    c) establezca sobre `datos.txt` los permisos

    640

    mediante `chmod` y compruebe el resultado con `ls -l`;

    d) realice modificaciones equivalentes mediante la notación simbólica de `chmod`, utilizando, entre otras, expresiones de la forma:

```
chmod u+...
chmod g-...
chmod o=...
```

    e) consulte la máscara de creación actual mediante:

```
umask
```

    cree posteriormente un archivo nuevo y explique la relación entre la máscara y los permisos que recibe el archivo;

    f) determine qué usuario y qué grupo poseen los archivos creados;

g) consulte la documentación de `chown` y `chgrp` y explique qué modifican. Si dispone de los permisos necesarios, realice una prueba sobre un archivo creado exclusivamente para este ejercicio;

h) cambie temporalmente los permisos de un archivo para impedir su lectura e intente abrirlo desde el lenguaje de trabajo. El programa deberá detectar e informar correctamente el error de apertura;

i) restaure inmediatamente los permisos normales del archivo;

j) experimente con un directorio de prueba modificando separadamente sus permisos de lectura, escritura y ejecución. Determine qué operaciones deja de poder realizar en cada caso;

k) explique por qué el permiso de escritura sobre un archivo y el permiso de escritura sobre el directorio que lo contiene controlan operaciones diferentes;

l) inspeccione los permisos de `/tmp` mediante:

```
ls -ld /tmp
```

e investigue el significado de la letra `t` cuando aparece en los permisos;

m) si el sistema dispone de soporte para ACL y de las herramientas correspondientes, examine opcionalmente un archivo mediante:

```
getfacl datos.txt
```

y explique por qué una ACL permite expresar reglas que no caben únicamente en las tres categorías propietario, grupo y otros.

No utilice `sudo` para provocar errores ni para realizar pruebas que puedan modificar archivos del sistema. Todas las modificaciones deberán realizarse sobre archivos y directorios creados específicamente para este ejercicio.

Compare brevemente el modelo observado con:

- macOS, que conserva el modelo de permisos de tradición Unix y agrega mecanismos como ACL;
- Windows, donde sistemas como NTFS utilizan principalmente listas de control de acceso para expresar los derechos de usuarios y grupos.

6.2.7. Integración de entrada, procesamiento y salida

33. **Ejercicio:** Simulación térmica de un tanque solar

Considere un tanque de agua calentado mediante radiación solar. El archivo `radiacion.dat` contiene dos columnas: tiempo t en horas y radiación $G(t)$ en $\mathrm{W/m^2}$.

Use los siguientes datos:

```
0 0
1 0
2 0
3 0
```

```
 4 0
 5 0
 6 0
 7 210
 8 420
 9 610
 10 760
 11 850
 12 900
 13 870
 14 780
 15 620
 16 400
 17 180
 18 0
 19 0
 20 0
 21 0
 22 0
 23 0
 24 0
```

La evolución se aproxima mediante

$$C\frac{dT}{dt} = A\eta G(t) - K(T - T_a),$$

con

$$T_a = 20\,^\circ\mathrm{C}, \quad A = 2\,\mathrm{m}^2, \quad \eta = 0{,}70, \quad K = 12\,\mathrm{W/K}, \quad C = 8 \times 10^5\,\mathrm{J/K}.$$

Escriba un programa que:

a) lea y valide las dos columnas de `radiacion.dat`;

b) calcule la temperatura mediante Euler,

$$T_{i+1} = T_i + \Delta t\,\frac{A\eta G_i - K(T_i - T_a)}{C},$$

con Δt expresado en segundos y $T(0) = 20\,^\circ\mathrm{C}$;

c) guarde en `temperatura_tanque.txt` tiempo, radiación y temperatura;

d) determine la temperatura máxima y la hora a la que ocurre;

e) genere `resumen_tanque.txt` con los parámetros utilizados y los principales resultados;

f) vuelva a leer el archivo de resultados y genere un gráfico de radiación y temperatura en función del tiempo.

Analice si la temperatura máxima coincide con la radiación máxima y estudie qué ocurre al duplicar A o reducir K a la mitad.

Scilab. Use las rutinas modernas de lectura/escritura tabular y las funciones gráficas de Scilab.

```
function T=simular(t,G,A,eta,K,C,Ta,T0)
    n=size(t,"*");
    T=zeros(n,1);
    T(1)=T0;
    for i=1:n-1
        dt=(t(i+1)-t(i))*3600; // horas -> segundos
        if dt<=0 then error("El tiempo debe ser
           estrictamente creciente"), end
        T(i+1)=T(i)+dt*(A*eta*G(i)-K*(T(i)-Ta))/C;
    end
endfunction

D=fscanfMat("radiacion.dat");
if size(D,2)<>2 then error("radiacion.dat debe tener dos
   columnas"), end
t=D(:,1); G=D(:,2);
if or(isnan(t)) | or(isnan(G)) | or(isinf(t)) | or(isinf(G))
   then
    error("Hay datos no finitos en radiacion.dat")
end

Ta=20; A=2; eta=0.70; K=12; C=8e5; T0=20;
T=simular(t,G,A,eta,K,C,Ta,T0);

R=[t G T];
fprintfMat("temperatura_tanque.txt",R,"%.10g");
[Tmax,imax]=max(T);
[Gmax,igmax]=max(G);

fd=mopen("resumen_tanque.txt","wt");
if fd<0 then error("No se pudo crear el resumen"), end
mfprintf(fd,"Ta = %.3f C\nA = %.3f m2\neta =
   %.3f\n",Ta,A,eta);
mfprintf(fd,"K = %.3f W/K\nC = %.6g J/K\n",K,C);
mfprintf(fd,"Tmax = %.6f C a t = %.3f h\n",Tmax,t(imax));
mfprintf(fd,"Gmax = %.6f W/m2 a t = %.3f h\n",Gmax,t(igmax));
mclose(fd);

// Volver a leer el resultado antes de graficar.
Q=fscanfMat("temperatura_tanque.txt");
scf();
subplot(2,1,1); plot(Q(:,1),Q(:,2)); xgrid();
xtitle("Radiacion","t [h]","G [W/m2]");
subplot(2,1,2); plot(Q(:,1),Q(:,3)); xgrid();
xtitle("Temperatura del tanque","t [h]","T [C]");
xs2png(gcf(),"tanque.png");

T_A2=simular(t,G,2*A,eta,K,C,Ta,T0);
T_K2=simular(t,G,A,eta,K/2,C,Ta,T0);
mprintf("Caso base : Tmax = %.3f C\n",max(T));
mprintf("A doble   : Tmax = %.3f C\n",max(T_A2));
```

```
mprintf("K/2       : Tmax = %.3f C\n",max(T_K2));
mprintf("Hora Gmax = %.2f h; hora Tmax = %.2f
   h\n",t(igmax),t(imax));
```

Octave. Use las rutinas de archivo numérico y las funciones gráficas de Octave.

```
D=dlmread("radiacion.dat"); if columns(D)!=2 ||
   any(!isfinite(D(:))), error("Archivo invalido"); endif
t=D(:,1); G=D(:,2); if any(diff(t)<=0), error("Tiempo no
   creciente"); endif
Ta=20; area=2; eta=0.70; K=12; C=8e5; T=zeros(size(t));
   T(1)=20;
for i=1:numel(t)-1
  dt=(t(i+1)-t(i))*3600;
     T(i+1)=T(i)+dt*(area*eta*G(i)-K*(T(i)-Ta))/C;
endfor
[tmx,imx]=max(T); printf("Tmax=%.3f C a t=%.2f
   h\n",tmx,t(imx));
fid=fopen("temperatura_tanque.txt","w");
   fprintf(fid,"tiempo_h radiacion temperatura\n");
for i=1:numel(t), fprintf(fid,"%.4f %.4f
   %.6f\n",t(i),G(i),T(i)); endfor; fclose(fid);
fid=fopen("resumen_tanque.txt","w"); fprintf(fid,"Ta=%g A=%g
   eta=%g K=%g C=%g\nTmax=%g
   t=%g\n",Ta,area,eta,K,C,tmx,t(imx)); fclose(fid);
R=dlmread("temperatura_tanque.txt","",1,0);
   [ax,h1,h2]=plotyy(R(:,1),R(:,2),R(:,1),R(:,3)); grid on;
   print("tanque.png","-dpng");
```

Python. Con NumPy puede cargar los datos y realizar el cálculo vectorial o iterativo; use la biblioteca gráfica configurada.

```
import numpy as np
import matplotlib.pyplot as plt
D=np.loadtxt("radiacion.dat")
if D.ndim!=2 or D.shape[1]!=2 or not np.all(np.isfinite(D)):
    raise ValueError("Archivo invalido")
t,G=D[:,0],D[:,1]
if np.any(np.diff(t)<=0): raise ValueError("Tiempo no
   creciente")
Ta=20.; area=2.; eta=.70; K=12.; C=8e5
T=np.zeros_like(t); T[0]=20
for i in range(len(t)-1):
    dt=(t[i+1]-t[i])*3600
    T[i+1]=T[i]+dt*(area*eta*G[i]-K*(T[i]-Ta))/C
imx=int(np.argmax(T)); print(f"Tmax={T[imx]:.3f} C a
   t={t[imx]:.2f} h")
np.savetxt("temperatura_tanque.txt",np.c_[t,G,T],header="tiempo_h
   radiacion temperatura",comments="")
with open("resumen_tanque.txt","w") as f:
    f.write(f"Ta={Ta} A={area} eta={eta} K={K}
```

```
        C={C}\nTmax={T[imx]} t={t[imx]}\n")
R=np.loadtxt("temperatura_tanque.txt",skiprows=1)
fig,ax=plt.subplots(); ax.plot(R[:,0],R[:,2],label="T");
   ax2=ax.twinx(); ax2.plot(R[:,0],R[:,1],label="G")
fig.savefig("tanque.png"); plt.close(fig)
```

Perl. Lea el archivo línea por línea y realice el esquema de Euler con arreglos; use la biblioteca gráfica configurada.

```
use strict; use warnings;open my$f,"<","radiacion.dat" or
   die$!;my(@t,@G);while(<$f>){next
   if/^\s*$/;my($a,$b)=split;die"Fila invalida\n" unless
   defined$b;push@t,0+$a;push@G,0+$b}close$f;my($Ta,$A,$eta,$K,$C)=(20,2
   my$i(0..$#t-1){die"Tiempo no creciente\n"
   unless$t[$i+1]>$t[$i];my$dt=($t[$i+1]-$t[$i])*3600;$T[$i+1]=$T[$i]+$d
   my$i(1..$#T){$im=$i if$T[$i]>$T[$im]}printf"Tmax=%.3f C a
   t=%.2f
   h\n",$T[$im],$t[$im];open$f,">","temperatura_tanque.txt"
   or die$!;print$f"tiempo_h radiacion temperatura\n";for
   my$i(0..$#t){printf$f"%.4f %.4f
   %.6f\n",$t[$i],$G[$i],$T[$i]}close$f;open$f,">","resumen_tanque.txt"
   or die$!;print$f"Ta=$Ta A=$A eta=$eta K=$K
   C=$C\nTmax=$T[$im] t=$t[$im]\n";close$f;open
   my$g,">","tanque.gp" or die$!;print$g"set terminal
   pngcairo\nset output 'tanque.png'\nset y2tics\nplot
   'temperatura_tanque.txt' every ::1 u 1:3 w l title 'T',
   '' every ::1 u 1:2 axes x1y2 w l title
   'G'\n";close$g;system("gnuplot","tanque.gp");
```

JavaScript. Lea el archivo con `fs`, almacene los datos en arreglos y use la biblioteca gráfica configurada.

```
"use strict";
const
   fs=require("node:fs"),{spawnSync}=require("node:child_process");const
   D=fs.readFileSync("radiacion.dat","utf8").trim().split(/\r?\n/).map(l
   new Error("Archivo invalido");const
   t=D.map(r=>r[0]),G=D.map(r=>r[1]),Ta=20,A=2,eta=.70,K=12,C=8e5,T=[20]
   i=0;i<t.length-1;i++){if(t[i+1]<=t[i])throw new
   Error("Tiempo no creciente");const
   dt=(t[i+1]-t[i])*3600;T[i+1]=T[i]+dt*(A*eta*G[i]-K*(T[i]-Ta))/C}let
   im=0;for(let
   i=1;i<T.length;i++)if(T[i]>T[im])im=i;console.log(`Tmax=${T[im].toFix
   C a t=${t[im].toFixed(2)}
   h`);fs.writeFileSync("temperatura_tanque.txt","tiempo_h
   radiacion temperatura\n"+t.map((x,i)=>`${x.toFixed(4)}
   ${G[i].toFixed(4)}
   ${T[i].toFixed(6)}`).join("\n")+"\n");fs.writeFileSync("resumen_tanqu
   A=${A} eta=${eta} K=${K} C=${C}\nTmax=${T[im]}
   t=${t[im]}\n`);fs.writeFileSync("tanque.gp","set terminal
   pngcairo\nset output 'tanque.png'\nset y2tics\nplot
```

```
    'temperatura_tanque.txt' every ::1 u 1:3 w l, '' every
    ::1 u 1:2 axes x1y2 w
    l\n");spawnSync("gnuplot",["tanque.gp"],{stdio:"inherit"});
```

34. **Ejercicio:** Descarga y análisis de datos climáticos

    Descargue desde una fuente indicada por el docente un archivo CSV de datos climáticos reales y conserve una copia local con un nombre estable. La descarga puede realizarse desde la terminal mediante una herramienta como `curl` o `wget`.

    Antes de procesar el archivo, inspeccione su cabecera y determine:

    - qué representa cada columna;
    - qué separador utiliza;
    - cómo representa valores ausentes;
    - qué campo identifica el tiempo o período de cada observación.

    Luego escriba un programa que:

    a) lea el CSV y descarte o trate explícitamente los registros inválidos;
    b) grafique la variable climática elegida en función del tiempo;
    c) calcule media, máximo y mínimo;
    d) identifique el período correspondiente al máximo y al mínimo;
    e) genere `resumen.txt` con la fuente, cantidad de registros válidos y estadísticas calculadas.

    Scilab. Use las rutinas CSV o de lectura de líneas disponibles y convierta explícitamente las columnas numéricas.

```
// Reemplace la URL por la indicada por el docente.
url="PONGA_AQUI_LA_URL";
archivo="clima.csv";
if url<>"PONGA_AQUI_LA_URL" then
    comando="curl -L --fail -o "+archivo+" '"+url+"'";
    [estado,salida,error_sistema]=host(comando);
    if estado<>0 then error("Fallo la descarga:
       "+error_sistema), end
end

if ~isfile(archivo) then error("No existe clima.csv"), end
S=csvRead(archivo,",",".","string");
if size(S,1)<2 then error("CSV sin registros"), end

disp("Cabecera:");
disp(S(1,:));
col_t=input("Numero de columna temporal: ");
col_y=input("Numero de columna de la variable numerica: ");
if col_t<1 | col_t>size(S,2) | col_y<1 | col_y>size(S,2) then
```

```
    error("Indice de columna fuera de rango")
end

// En esta solucion se supone que el campo temporal tambien
   es numerico.
t=[]; y=[];
for i=2:size(S,1)
    st=stripblanks(S(i,col_t));
    sy=stripblanks(S(i,col_y));
    if st<>"" & sy<>"" & isnum(st) & isnum(sy) then
        ti=strtod(st); yi=strtod(sy);
        if ~isnan(ti) & ~isinf(ti) & ~isnan(yi) & ~isinf(yi)
           then
            t=[t ti]; y=[y yi];
        end
    end
end
if size(y,"*")==0 then error("No quedaron registros
   numericos validos"), end

media=mean(y); [ymax,imax]=max(y); [ymin,imin]=min(y);
scf(); plot(t,y); xgrid();
xtitle("Serie climatica","tiempo","variable");
xs2png(gcf(),"clima.png");

fd=mopen("resumen.txt","wt");
if fd<0 then error("No se pudo crear resumen.txt"), end
mfprintf(fd,"Fuente: %s\n",url);
mfprintf(fd,"Registros validos: %d\n",size(y,"*"));
mfprintf(fd,"Media: %.10g\n",media);
mfprintf(fd,"Maximo: %.10g en t=%.10g\n",ymax,t(imax));
mfprintf(fd,"Minimo: %.10g en t=%.10g\n",ymin,t(imin));
mclose(fd);
```

Octave. Use las herramientas CSV disponibles en la instalación y conserve la columna temporal necesaria para identificar extremos.

```
archivo="clima.csv"; col_tiempo=1; col_valor=2;  % ajustar
   despues de inspeccionar la cabecera
fid=fopen(archivo,"r");
   cab=strsplit(strtrim(fgetl(fid)),",");
printf("Cabecera: %s\n",strjoin(cab," | "));
C=textscan(fid,repmat("%s",1,numel(cab)),"Delimiter",",");
   fclose(fid);
tiempo=C{col_tiempo}; raw=C{col_valor}; valores=[];
   periodos={};
for i=1:numel(raw), x=str2double(raw{i}); if isfinite(x),
   valores(end+1)=x; periodos{end+1}=tiempo{i}; endif; endfor
if isempty(valores), error("Sin registros validos"); endif
[ma,ia]=max(valores); [mi,ii]=min(valores); mu=mean(valores);
printf("n=%d media=%g max=%g (%s) min=%g
```

```
    (%s)\n",numel(valores),mu,ma,periodos{ia},mi,periodos{ii});
plot(1:numel(valores),valores,"b-"); grid on;
    print("clima.png","-dpng");
fid=fopen("resumen.txt","w"); fprintf(fid,"Fuente:
    %s\nRegistros: %d\nMedia: %g\nMax: %g %s\nMin: %g
    %s\n",archivo,numel(valores),mu,ma,periodos{ia},mi,periodos{ii});
    fclose(fid);
```

Python. Use el módulo `csv` o la herramienta tabular configurada en el curso; valide los valores antes de convertirlos.

```
import csv, math
import matplotlib.pyplot as plt
ARCHIVO="clima.csv"; COL_TIEMPO="periodo"; COL_VALOR="valor"
    # ajustar a la fuente
periodos=[]; valores=[]
with open(ARCHIVO,newline="",encoding="utf-8") as f:
    r=csv.DictReader(f); print("Cabecera:",r.fieldnames)
    for row in r:
        try: x=float(row[COL_VALOR])
        except (ValueError,TypeError,KeyError): continue
        if math.isfinite(x):
            periodos.append(row[COL_TIEMPO]);
            valores.append(x)
if not valores: raise ValueError("Sin registros validos")
media=sum(valores)/len(valores);
    imax=max(range(len(valores)),key=valores.__getitem__);
    imin=min(range(len(valores)),key=valores.__getitem__)
plt.plot(range(len(valores)),valores); plt.grid();
    plt.savefig("clima.png"); plt.close()
with open("resumen.txt","w",encoding="utf-8") as f:
    f.write(f"Fuente: {ARCHIVO}\nRegistros:
       {len(valores)}\nMedia: {media}\n")
    f.write(f"Max: {valores[imax]} {periodos[imax]}\nMin:
       {valores[imin]} {periodos[imin]}\n")
```

Perl. Use `Text::CSV` si está disponible y valide de forma explícita los campos numéricos.

```
use strict; use warnings; use
    Text::CSV;my($archivo,$ct,$cv)=("clima.csv","periodo","valor");my$c=T
    my$f,"<:encoding(UTF-8)",$archivo or
    die$!;my$h=$c->getline($f);print"Cabecera: ",join(" |
    ",@$h),"\n";my%ix;@ix{@$h}=(0..$#$h);die"Ajuste los
    nombres de columnas\n" unless
    exists$ix{$ct}&&exists$ix{$cv};my(@t,@v);while(my$r=$c->getline($f)){
    unless
    defined$x&&$x=~/^[-+]?\d+(?:\.\d+)?(?:[eE][-+]?\d+)?$/;push@t,$r->[$i
    datos\n" unless@v;my$sum=0;$sum+=$_
    for@v;my($imax,$imin)=(0,0);for my$i(1..$#v){$imax=$i
    if$v[$i]>$v[$imax];$imin=$i if$v[$i]<$v[$imin]}open
```

```
    my$r,">","resumen.txt" or die$!;print$r"Fuente:
    $archivo\nRegistros: ",scalar@v,"\nMedia:
    ",$sum/@v,"\nMax: $v[$imax] $t[$imax]\nMin: $v[$imin]
    $t[$imin]\n";close$r;open my$d,">","clima.dat" or
    die$!;print$d"$_ $v[$_]\n" for 0..$#v;close$d;open
    my$g,">","clima.gp" or die$!;print$g"set terminal
    pngcairo\nset output 'clima.png'\nplot 'clima.dat' w
    l\n";close$g;system("gnuplot","clima.gp");
```

JavaScript. Use la biblioteca CSV configurada y `fs`; no suponga que todos los campos son numéricos.

```
"use strict";
const
   fs=require("node:fs"),{spawnSync}=require("node:child_process");const
   ARCHIVO="clima.csv",COL_TIEMPO="periodo",COL_VALOR="valor",L=fs.readF
   new Error("Ajuste nombres de columnas");const
   t=[],v=[];for(const l of L){const
   r=l.split(","),x=Number(r[ix[COL_VALOR]]);if(Number.isFinite(x)){t.pu
   new Error("Sin datos validos");const
   mu=v.reduce((a,b)=>a+b,0)/v.length,imax=v.indexOf(Math.max(...v)),imi
   ${ARCHIVO}\nRegistros: ${v.length}\nMedia: ${mu}\nMax:
   ${v[imax]} ${t[imax]}\nMin: ${v[imin]}
   ${t[imin]}\n`);fs.writeFileSync("clima.dat",v.map((x,i)=>`${i}
   ${x}`).join("\n")+"\n");fs.writeFileSync("clima.gp","set
   terminal pngcairo\nset output 'clima.png'\nplot
   'clima.dat' w
   l\n");spawnSync("gnuplot",["clima.gp"],{stdio:"inherit"});
```

35. **Ejercicio:** Integración de lectura, análisis e informe

    Construya un programa completo que procese `mediciones.dat`, archivo de dos columnas numéricas: tiempo y valor medido.

    Puede utilizar el siguiente conjunto de prueba:

```
0.00 64.577199
0.50 59.461914
1.00 56.514274
1.50 51.465418
2.00 49.347343
2.50 46.088111
3.00 42.833860
3.50 41.692327
4.00 38.605167
4.50 37.826848
5.00 35.427845
5.50 34.131295
6.00 33.744052
6.50 33.660912
7.00 31.048082
```

```
7.50 30.469972
8.00 30.719251
8.50 30.851820
9.00 29.401016
9.50 28.472613
```

El programa deberá estar dividido en funciones y realizar el flujo completo:

a) comprobar que el archivo de entrada exista y pueda leerse;

b) leer y validar todas las filas;

c) calcular cantidad, media, mediana, desvío estándar, mínimo, máximo y rango;

d) generar `informe.txt` con un encabezado y todas las estadísticas;

e) generar un gráfico de valor en función del tiempo e incluir una línea horizontal correspondiente a la media;

f) guardar el gráfico en `grafico.png`;

g) informar por la salida estándar los nombres de los archivos generados;

h) cerrar todos los recursos y terminar con un estado de error distinto de cero si falla una operación esencial de E/S.

Después de ejecutarlo, abra `informe.txt` y `grafico.png` con herramientas externas al lenguaje de trabajo y compruebe que los resultados son utilizables fuera del programa que los creó.

Scilab. Use las rutinas modernas de archivos de texto/matrices, funciones estadísticas y el mecanismo gráfico configurado en el curso.

```
function [t,y]=leer_mediciones(archivo)
    if ~isfile(archivo) then error("No existe "+archivo), end
    D=fscanfMat(archivo);
    if size(D,2)<>2 | size(D,1)==0 then
        error("El archivo debe contener exactamente dos
           columnas numericas")
    end
    if or(isnan(D)) | or(isinf(D)) then error("Hay datos no
       finitos"), end
    t=D(:,1); y=D(:,2);
endfunction

function s=estadisticas(y)
    s=struct();
    s.n=size(y,"*");
    s.media=mean(y);
    s.mediana=median(y);
    s.desvio=stdev(y);
    s.minimo=min(y);
    s.maximo=max(y);
    s.rango=s.maximo-s.minimo;
endfunction
```

```
function escribir_informe(archivo,s)
    fd=mopen(archivo,"wt");
    if fd<0 then error("No se pudo abrir "+archivo), end
    mfprintf(fd,"INFORME DE ANALISIS\n");
    mfprintf(fd,"==================\n\n");
    mfprintf(fd,"Cantidad : %d\n",s.n);
    mfprintf(fd,"Media    : %.10g\n",s.media);
    mfprintf(fd,"Mediana  : %.10g\n",s.mediana);
    mfprintf(fd,"Desvio   : %.10g\n",s.desvio);
    mfprintf(fd,"Minimo   : %.10g\n",s.minimo);
    mfprintf(fd,"Maximo   : %.10g\n",s.maximo);
    mfprintf(fd,"Rango    : %.10g\n",s.rango);
    if mclose(fd)<>0 then error("Error al cerrar "+archivo),
       end
endfunction

function generar_grafico(t,y,media,archivo)
    scf();
    plot(t,y);
    xgrid();
    ax=gca(); ax.auto_clear="off";
    plot([min(t) max(t)],[media media],"--");
    ax.auto_clear="on";
    xtitle("Mediciones","tiempo","valor");
    xs2png(gcf(),archivo);
endfunction

archivo_entrada="mediciones.dat";
[t,y]=leer_mediciones(archivo_entrada);
s=estadisticas(y);
escribir_informe("informe.txt",s);
generar_grafico(t,y,s.media,"grafico.png");

mprintf("Generado: informe.txt\n");
mprintf("Generado: grafico.png\n");
// Las llamadas error(...) anteriores hacen fallar el script
   si una E/S
// esencial no puede completarse.
```

Octave. Use las rutinas de lectura numérica, funciones estadísticas y las herramientas gráficas de Octave.

```
function D=leer_mediciones(nombre)
  if exist(nombre,"file")!=2, error("No existe el archivo");
     endif
  D=dlmread(nombre); if columns(D)!=2 ||
     any(!isfinite(D(:))), error("Datos invalidos"); endif
endfunction
function s=estadisticas(v)
  s.n=numel(v); s.media=mean(v); s.mediana=median(v);
     s.desvio=std(v); s.min=min(v); s.max=max(v);
```

```
     s.rango=s.max-s.min;
endfunction
D=leer_mediciones("mediciones.dat"); t=D(:,1); y=D(:,2);
   s=estadisticas(y);
fid=fopen("informe.txt","w"); if fid<0, error("No se pudo
   crear informe"); endif
fprintf(fid,"INFORME\n=======\nCantidad: %d\nMedia:
   %.6f\nMediana: %.6f\nDesvio: %.6f\nMin: %.6f\nMax:
   %.6f\nRango:
   %.6f\n",s.n,s.media,s.mediana,s.desvio,s.min,s.max,s.rango);
   fclose(fid);
plot(t,y,"bo-"); hold on; plot([min(t) max(t)],[s.media
   s.media],"r--"); hold off; grid on;
   print("grafico.png","-dpng");
printf("Generados: informe.txt y grafico.png\n");
```

Python. Con NumPy puede realizar los cálculos estadísticos y con la biblioteca gráfica configurada generar el PNG; use manejo explícito de errores de E/S.

```
from pathlib import Path
import sys
import numpy as np
import matplotlib.pyplot as plt

def leer(nombre):
    p=Path(nombre)
    if not p.is_file(): raise OSError("No existe el archivo")
    D=np.loadtxt(p)
    if D.ndim!=2 or D.shape[1]!=2 or not
       np.all(np.isfinite(D)):
        raise ValueError("Datos invalidos")
    return D

def estadisticas(v):
    return
       dict(n=v.size,media=float(np.mean(v)),mediana=float(np.median(v)),
            desvio=float(np.std(v,ddof=1)),minimo=float(np.min(v)),
            maximo=float(np.max(v)),rango=float(np.ptp(v)))
try:
    D=leer("mediciones.dat"); t,y=D[:,0],D[:,1];
       s=estadisticas(y)
    with open("informe.txt","w",encoding="utf-8") as f:
        f.write("INFORME\n=======\n")
        for k,v in s.items(): f.write(f"{k}: {v}\n")
    plt.plot(t,y,"o-");
       plt.axhline(s["media"],linestyle="--"); plt.grid();
       plt.savefig("grafico.png"); plt.close()
except (OSError,ValueError) as e:
    print(f"ERROR: {e}",file=sys.stderr); raise SystemExit(1)
print("Generados: informe.txt y grafico.png")
```

Perl. Separe lectura, estadísticas, informe y gráfico en funciones; use la biblioteca estadística/gráfica configurada cuando corresponda.

```
use strict; use warnings;sub leer{my($f)=@_;open my$h,"<",$f
   or die"$f: $!";my(@t,@y);while(<$h>){next
   if/^\s*$/;my($a,$b)=split;die"Fila invalida\n" unless
   defined$b;push@t,0+$a;push@y,0+$b}close$h;return(\@t,\@y)}sub
   stats{my($v)=@_;my$n=@$v;my$sum=0;$sum+=$_
   for@$v;my$mu=$sum/$n;my@s=sort{$a<=>$b}@$v;my$med=$n%2?$s[int$n/2]:($
   for@$v;my($mi,$ma)=($s[0],$s[-1]);return{n=>$n,media=>$mu,mediana=>$m
   my$f,">","informe.txt" or
   die$!;print$f"INFORME\n=======\n";print$f"$_: $s->{$_}\n"
   for qw(n media mediana desvio minimo maximo
   rango);close$f;open my$d,">","grafico.dat" or die$!;for
   my$i(0..$#$t){print$d"$t->[$i] $y->[$i]\n"}close$d;open
   my$g,">","grafico.gp" or die$!;print$g"set terminal
   pngcairo\nset output 'grafico.png'\nplot 'grafico.dat' w
   lp title 'datos', $s->{media} w l title
   'media'\n";close$g;system("gnuplot","grafico.gp")==0 or
   die"gnuplot fallo\n";print"Generados: informe.txt y
   grafico.png\n";
```

JavaScript. Separe lectura, estadísticas, informe y gráfico en funciones; use `fs` y las bibliotecas numéricas/gráficas configuradas cuando corresponda.

```
"use strict";
const
   fs=require("node:fs"),{spawnSync}=require("node:child_process");funct
   leer(f){if(!fs.existsSync(f))throw new Error("No existe
   el archivo");const
   D=fs.readFileSync(f,"utf8").trim().split(/\r?\n/).map(l=>l.trim().spl
   new Error("Datos invalidos");return D}function
   stats(v){const
   n=v.length,media=v.reduce((a,b)=>a+b,0)/n,s=[...v].sort((a,b)=>a-b),m
   D=leer("mediciones.dat"),t=D.map(r=>r[0]),y=D.map(r=>r[1]),s=stats(y)
   ${v}`).join("\n")+"\n");fs.writeFileSync("grafico.dat",t.map((x,i)=>`
   ${y[i]}`).join("\n")+"\n");fs.writeFileSync("grafico.gp",`set
   terminal pngcairo\nset output 'grafico.png'\nplot
   'grafico.dat' w lp title 'datos', ${s.media} w l title
   'media'\n`);const
   r=spawnSync("gnuplot",["grafico.gp"],{stdio:"inherit"});if(r.status!=
   new Error("gnuplot fallo");console.log("Generados:
   informe.txt y
   grafico.png")}catch(e){console.error("ERROR:",e.message);process.exit
```

Capítulo 7

Gráficos – Gnuplot

7.1. Programa

UNIDAD 7: Gráficos, visualización e imágenes

Representación gráfica de funciones y datos. Gráficos bidimensionales. Discretización y muestreo. Curvas múltiples con dominios y resoluciones diferentes. Estilos de líneas y marcadores. Títulos, ejes, leyendas, cuadrículas, anotaciones, límites y relación de aspecto. Tratamiento gráfico de discontinuidades, singularidades y valores no representables.

Escalas y sistemas de coordenadas. Escalas lineales, semilogarítmicas y logarítmicas. Transformaciones de coordenadas y curvas polares. Representación de datos experimentales. Barras de error. Histogramas y distribuciones. Ajuste de datos, regresión lineal y análisis de residuos.

Herramientas de generación de gráficos. Uso de Gnuplot como programa gráfico externo: instalación y verificación, ejecución interactiva y no interactiva, scripts, archivos de datos, generación automática de órdenes e invocación desde programas. Uso de las capacidades gráficas nativas de Scilab y de los demás lenguajes del curso. Creación y manipulación de objetos gráficos. Propiedades y parámetros de los gráficos. Composición de figuras y paneles.

Representación tridimensional. Grillas de datos. Superficies, mallas, mapas de color y perspectivas. Curvas de nivel, mapas con relleno, contornos sobre superficies y proyecciones sobre planos. Resolución de grillas. Campos vectoriales y gradientes.

Visualización de resultados de modelos numéricos. Series temporales, trayectorias y retratos de fase. Representaciones múltiples de un mismo fenómeno. Generación de secuencias de imágenes y animaciones.

Exportación, persistencia e interoperabilidad gráfica. Formatos raster y vectoriales. PNG, PDF, SVG y formatos equivalentes. Reproducibilidad de los gráficos a partir de datos y scripts. Separación entre cálculo, datos, especificación gráfica y archivo de salida.

Resolución de los problemas mediante dos enfoques complementarios: uso de Gnuplot como motor gráfico externo y uso de las capacidades gráficas propias de Scilab, GNU Octave, Python, Perl y JavaScript/Node.js.

7.2. Gráficos, visualización e imágenes

7.2.1. Visualización como parte del cálculo científico

Como referencias generales para visualización científica, percepción gráfica, diseño de figuras, uso del color y animación se emplean Wilke 2019; Tufte 2001; Cleveland 1994; Cleveland y McGill 1984; Harrower y Brewer 2003; Okabe e Ito 2008; Borland y R. M. Taylor 2007; Crameri, Shephard y Heron 2020; Foley et al. 1990; Tversky, Morrison y Bétrancourt 2002.

La representación gráfica no es una etapa decorativa que se agrega cuando el cálculo ya terminó. En ciencia, ingeniería y análisis de datos forma parte del proceso de razonamiento: permite comprobar si los resultados tienen la forma esperada, detectar errores, comparar modelos, reconocer discontinuidades, apreciar escalas diferentes y comunicar una conclusión.

Un conjunto de números puede ocultar estructuras que aparecen inmediatamente al representarlo. El ejemplo clásico es el cuarteto de Anscombe: varios conjuntos de datos con prácticamente los mismos estadísticos elementales pueden producir gráficos cualitativamente muy distintos Wilke 2019. Por eso un programa que calcula correctamente pero representa de manera engañosa o incompleta todavía no ha comunicado correctamente el resultado.

En esta unidad se utilizarán dos caminos complementarios para producir una figura:

a) el lenguaje calcula y organiza los datos, y **Gnuplot** actúa como motor gráfico externo;

b) el lenguaje calcula los datos y utiliza sus **capacidades gráficas directas**, ya sean propias del entorno o proporcionadas por una biblioteca del lenguaje.

Los cinco lenguajes de trabajo son Scilab, GNU Octave, Python, Perl y JavaScript/Node.js. La intención no es aprender diez sintaxis desconectadas, sino reconocer una misma estructura conceptual detrás de todas ellas.

Complemento: Programa

UNIDAD 7: Gráficos e imágenes. Creación de Gráficos. Representación en 2D. Parámetros del gráfico. Representación en 3D. Curvas de nivel. Exportación, formatos de imágenes: PNG, SVG. Etc.

Complemento: Algunas buenas prácticas de visualización científica

Las reglas más importantes para los gráficos científicos son:

- los ejes siempre deben tener etiquetas con unidades;
- la leyenda debe identificar inequívocamente cada curva;

- las barras de error deben mostrarse cuando los datos tienen incertidumbre; el rango de los ejes debe mostrar los datos sin mucho espacio vacío pero sin recortarlos;
- la resolución debe ser suficiente para el uso previsto (300 DPI para impresión, 72-150 DPI para pantalla);
- los colores deben ser distinguibles también en impresión en blanco y negro y para personas con daltonismo.

Recomendaciones para la elaboración de gráficos científicos:

- títulos;
- etiquetas de ejes;
- leyendas;

El flujo común

La mayoría de los problemas gráficos de esta unidad pueden entenderse mediante el siguiente flujo:

$$\text{modelo o datos} \longrightarrow \text{cálculo} \longrightarrow \text{representación numérica} \longrightarrow \text{especificación gráfica} \longrightarrow \text{figura} \longrightarrow \text{archivo de salida.}$$

Conviene separar esas etapas. La función que integra una ecuación diferencial no debería depender de si después el resultado se dibuja con Gnuplot, Matplotlib o las funciones gráficas de Scilab. De la misma manera, cambiar el color de una curva no debería obligar a repetir una simulación costosa.

Esta separación permite además guardar los resultados intermedios. Un archivo tabular con las columnas (t, x, v) puede reutilizarse para generar una figura nueva sin volver a integrar el sistema. Esa idea conecta esta unidad con la unidad anterior sobre archivos y persistencia.

7.2.2. Dos arquitecturas para producir gráficos

Lenguaje más Gnuplot

En el primer camino el programa principal calcula los datos y delega la representación a otro proceso:

$$\texttt{programa} \longrightarrow \texttt{datos.txt} + \texttt{grafico.gnu} \longrightarrow \texttt{gnuplot} \longrightarrow \texttt{figura.png/pdf/svg}.$$

El archivo de datos contiene los valores numéricos y el archivo de órdenes la descripción de la figura: terminal, archivo de salida, ejes, título, estilos y las órdenes `plot` o `splot`. El programa puede generar ambos y luego ejecutar Gnuplot, comprobando el código de salida del proceso.

Este camino tiene ventajas importantes:

- la especificación gráfica queda en un archivo de texto pequeño y reproducible;
- el mismo script de Gnuplot puede consumir datos generados por cualquiera de los cinco lenguajes;
- la generación de figuras puede hacerse sin interfaz gráfica, por ejemplo en un servidor;
- cálculo y presentación quedan claramente desacoplados.

Su costo es que aparecen más componentes: archivos intermedios, nombres de ruta, creación de un proceso y tratamiento de errores externos.

Gráficos directos desde el lenguaje

En el segundo camino el programa trabaja con una interfaz gráfica del propio entorno o con una biblioteca cargada dentro de él:

`programa` ⟶ datos en memoria ⟶ objetos gráficos ⟶ figura ⟶ archivo.

Aquí suele existir una jerarquía de objetos: una *figura* contiene uno o más sistemas de ejes, y cada sistema contiene curvas, puntos, textos, superficies u otros objetos. El programa conserva identificadores o *handles* de esos objetos y puede modificar sus propiedades. Esta arquitectura resulta natural para exploración interactiva y animaciones.

No todos los lenguajes traen el mismo nivel de gráficos en su núcleo. Scilab y Octave poseen sistemas gráficos integrados. Python utiliza normalmente Matplotlib. Perl se apoya en módulos de PDL y bibliotecas gráficas; en esta unidad se tomarán PLplot para figuras científicas estáticas y Prima cuando se requiera interacción. En JavaScript se utilizará Plotly.js; un programa Node.js puede calcular los datos y generar una página HTML que el navegador representa sin recurrir a Gnuplot.

7.2.3. Principios comunes de representación bidimensional

Antes de estudiar una biblioteca particular conviene fijar qué significa representar correctamente una serie de datos.

Muestreo y discretización

Una función matemática es continua en su dominio, pero el computador normalmente almacena una cantidad finita de valores. Si se desea dibujar

$$y = f(x), \qquad a \le x \le b,$$

se eligen puntos

$$x_0, x_1, \ldots, x_{N-1}$$

y se calculan los valores correspondientes $y_i = f(x_i)$. El gráfico de línea une esos puntos mediante segmentos. Por lo tanto, lo que se ve depende de la discretización.

Una cantidad de puntos demasiado pequeña puede producir una poligonal visible, perder máximos estrechos, disimular oscilaciones o incluso sugerir una forma incorrecta. Aumentar N mejora la resolución, pero no reemplaza el análisis: si la función contiene una singularidad, aumentar puntos a ciegas puede hacer más evidente una línea espuria que atraviesa la asíntota.

También es perfectamente válido representar en una misma figura series con abscisas distintas. Una curva puede contener cincuenta valores sobre $[0, 10]$ y otra cien valores sobre $[0, 5]$. Cada serie es un conjunto independiente de pares (x_i, y_i).

Líneas, puntos y marcadores

Una línea comunica continuidad entre los valores mostrados. Un conjunto de puntos comunica observaciones discretas. Por eso, en una comparación entre modelo y experimento es habitual usar una línea para el modelo y marcadores para las mediciones. Unir mediciones mediante segmentos puede insinuar que se conoce lo que ocurrió entre ellas cuando en realidad no fue medido.

Color, tipo de línea y marcador son canales visuales. Conviene no depender de uno solo: dos curvas importantes deberían seguir siendo distinguibles al imprimirse en escala de grises o para una persona con visión deficiente del color Harrower y Brewer 2003; Okabe e Ito 2008.

Ejes, límites y relación de aspecto

El rango de los ejes forma parte del mensaje. Un rango excesivamente amplio aplasta las variaciones; uno recortado de manera inadecuada puede exagerarlas. En gráficos de barras el eje que codifica la longitud de las barras requiere especial cuidado porque truncarlo cambia fuertemente la percepción Tufte 2001.

Cuando los dos ejes representan longitudes comparables, como en una trayectoria plana, una curva polar o un retrato de fase con las mismas unidades, suele ser necesario mantener igual escala geométrica en ambas direcciones. De otro modo un círculo puede verse como una elipse y una trayectoria puede parecer más o menos inclinada de lo que realmente es.

Títulos, etiquetas, unidades, leyendas y anotaciones

Un gráfico científico debe poder entenderse sin depender de que el lector adivine qué contiene. Habitualmente necesita:

- un título que identifique el fenómeno o la comparación;
- nombres de ejes;
- unidades cuando las magnitudes sean dimensionales;
- leyenda cuando haya varias series y no estén rotuladas directamente;

- cuadrícula cuando facilite una lectura cuantitativa;
- anotaciones cuando exista un punto notable que merezca ser destacado.

No se trata de llenar el gráfico de elementos. Cada adición debe aportar información. Rotular dos raíces puede ser útil; rotular cada punto de una serie larga suele producir ruido visual Wilke 2019. Las anotaciones se implementan con mecanismos distintos pero equivalentes: por ejemplo, `set label` en Gnuplot, `xstring` en Scilab, `text` en GNU Octave, `Axes.text`/`Axes.annotate` en Matplotlib, `text` en PDL/PLplot y `layout.annotations` en Plotly.js.

Discontinuidades y valores no representables

Una función racional o trigonométrica puede no estar definida en ciertos puntos. Si se calculan valores a ambos lados de una asíntota y el sistema gráfico los une, aparece un segmento que no pertenece a la función.

Las estrategias generales son:

- separar explícitamente las ramas;
- eliminar el punto no definido;
- guardar un valor ausente como NaN y usar una biblioteca que corte la curva en ese punto;
- introducir una separación de bloques en el archivo de datos cuando el motor externo interprete esa separación como interrupción de la serie.

Es importante distinguir una discontinuidad matemática de un desbordamiento numérico. Por ejemplo, e^{5t} está definido para todo t real, pero para un t suficientemente grande puede exceder el máximo representable por el tipo numérico usado. El gráfico no debe ocultar esa diferencia.

7.2.4. Escalas y sistemas de coordenadas

Escalas lineales, semilogarítmicas y logarítmicas

Una escala logarítmica es útil cuando una magnitud abarca varios órdenes de magnitud o cuando se quiere reconocer la forma funcional de una relación.

Si

$$y = Ae^{kx},$$

entonces

$$\ln y = \ln A + kx,$$

de modo que una gráfica semilogarítmica con eje vertical logarítmico convierte la exponencial en una recta. Análogamente, si

$$y = Ax^n,$$

entonces

$$\log y = \log A + n \log x,$$

y una representación log–log es lineal con pendiente n.

Una escala logarítmica ordinaria sólo admite coordenadas positivas. Los ceros, valores negativos, infinitos y NaN necesitan tratamiento explícito. En un ejercicio con $\ln x$ y eje vertical logarítmico, además de exigir $x > 0$, se debe escoger una región en la que $\ln x > 0$ si se pretende representar ese valor sobre un eje y logarítmico.

Coordenadas polares

Una curva polar viene dada por un radio r en función del ángulo θ. Se puede representar de dos maneras conceptualmente equivalentes:

a) usar una interfaz gráfica que entienda coordenadas polares;

b) transformar a coordenadas cartesianas mediante

$$x = r\cos\theta, \qquad y = r\sin\theta$$

y dibujar los pares (x, y).

La segunda forma es universal y por eso resulta importante aun cuando la biblioteca disponga de un modo polar directo. Permite comprobar qué hace el sistema gráfico y usar la misma construcción en cualquier lenguaje.

7.2.5. Datos experimentales y estadísticos

Para incertidumbres, ajuste de datos, análisis numérico e histogramas se toman como referencias J. R. Taylor 1997; Bevington y D. K. Robinson 2003; Press et al. 2007; Freedman y Diaconis 1981.

Barras de error

Una medición no es sólo un punto central. Si se conoce una incertidumbre Δy_i, una representación simple muestra el intervalo

$$y_i - \Delta y_i \le y \le y_i + \Delta y_i.$$

Las barras de error permiten comparar visualmente una medición con un modelo. Que una predicción quede fuera de una barra individual no constituye por sí solo una refutación del modelo; debe considerarse qué representa exactamente la incertidumbre y el conjunto completo de observaciones.

Histogramas

Un histograma resume una muestra dividiendo el eje de valores en intervalos y contando cuántas observaciones caen en cada uno. Su forma depende del ancho de clase. Clases muy estrechas resaltan fluctuaciones de la muestra; clases muy anchas pueden borrar estructura. Por eso conviene comparar más de una elección. Una regla conocida para elegir el ancho utiliza el rango intercuartil y el número de datos Freedman y Diaconis 1981.

Al superponer una densidad teórica hay que compatibilizar las normalizaciones. Si el eje vertical contiene conteos, la densidad debe escalarse por el tamaño de la muestra y por el ancho de clase. Si el histograma está normalizado como densidad, puede compararse directamente con una función de densidad cuya integral sea uno.

Distribuciones y densidades

Una distribución describe cómo se reparten los posibles valores de una variable aleatoria. Para una variable continua puede representarse mediante una función de densidad $p(x)$, no negativa y normalizada de modo que

$$\int_{-\infty}^{\infty} p(x)\,dx = 1.$$

La probabilidad de encontrar un valor en un intervalo se obtiene integrando la densidad sobre ese intervalo. En los datos experimentales, el histograma es una aproximación empírica a esa distribución: al aumentar la cantidad de datos y elegir adecuadamente las clases, su forma puede compararse con una densidad teórica. La distribución normal utilizada en los ejemplos de esta unidad es un caso particular de esta relación entre muestra, histograma y modelo probabilístico Bevington y D. K. Robinson 2003; Press et al. 2007.

Regresión y residuos

En un ajuste lineal

$$y = ax + b$$

los parámetros pueden calcularse por mínimos cuadrados. La figura debe mostrar los puntos y la recta, pero esa comparación no es suficiente. Los residuos

$$r_i = y_i - (ax_i + b)$$

deben inspeccionarse por separado. Una estructura sistemática en los residuos puede revelar que el modelo lineal es inadecuado aunque la recta parezca visualmente próxima a los puntos.

7.2.6. Gnuplot como motor gráfico externo

La sintaxis y las capacidades de Gnuplot utilizadas en esta unidad se apoyan en la documentación y los ejemplos oficiales gnuplot development team 2025; gnuplot Team 2026; gnuplot development team 2026.

Gnuplot es un programa especializado en representación de datos y funciones. Puede utilizarse de forma interactiva, recibir una orden desde la terminal, leer órdenes por la entrada estándar o ejecutar un archivo de comandos. Para un curso de programación interesa especialmente esta última posibilidad, porque convierte la figura en un resultado reproducible a partir de texto y datos.

Instalación y verificación

Gnuplot debe estar instalado como programa independiente y accesible desde la ruta de ejecución del sistema. El procedimiento de instalación depende del sistema operativo y de la distribución utilizada; una vez instalado, la primera comprobación debe hacerse desde una terminal:

```
gnuplot --version
```

La orden debe devolver la versión instalada. Una verificación mínima del motor puede realizarse sin abrir una ventana gráfica:

```
gnuplot -e "set terminal dumb; plot sin(x)"
```

Para conocer los terminales disponibles en una instalación concreta puede iniciarse Gnuplot y ejecutar `set terminal`. Antes de automatizar la salida conviene comprobar que estén disponibles los terminales que se usarán en el curso, por ejemplo `pngcairo`, `pdfcairo` o `svg`. Esta verificación evita confundir un error del programa que genera los datos con la ausencia del ejecutable o de un terminal gráfico determinado.

Herramientas auxiliares utilizadas en el práctico Para las utilidades externas usadas en los ejercicios se consultan la documentación de GNU nano, xdg-utils, GNOME Document Viewer e ImageMagick GNU nano Project 2026; freedesktop.org 2026; GNOME Project 2026; ImageMagick Studio LLC 2026a; ImageMagick Studio LLC 2026b; para la línea de comandos se usan además las referencias de Bash y de las páginas de manual de GNU/Linux Ramey y Fox 2025; The Linux man-pages project 2026.

Además de Gnuplot, algunos ejercicios utilizan programas externos para editar archivos, abrir las figuras producidas y construir animaciones. En sistemas Debian/Ubuntu puede prepararse el entorno con:

```
sudo apt update
sudo apt install gnuplot imagemagick evince nano xdg-utils
```

Los programas cumplen papeles diferentes:

- `nano` es un editor de texto puro y puede utilizarse para crear scripts `.gnu`; sus atajos más frecuentes son `Ctrl+O` para guardar, `Ctrl+X` para salir, `Ctrl+W` para buscar, `Ctrl+K` para cortar y `Ctrl+U` para pegar;
- `display`, incluido en ImageMagick, es útil principalmente para abrir imágenes raster como PNG;
- `evince` abre documentos PDF;
- `xdg-open archivo` solicita al escritorio GNU/Linux que abra el archivo con la aplicación asociada;
- ImageMagick permite combinar una secuencia de imágenes en un GIF. Según la versión instalada, la orden principal puede ser `convert` o `magick`.

La disponibilidad puede comprobarse sin suponer que todos los equipos tienen la misma instalación:

```
command -v gnuplot
command -v display
command -v convert
command -v magick
command -v evince
command -v nano
```

En Windows debe instalarse Gnuplot como aplicación independiente y asegurarse de que `gnuplot.exe` sea accesible desde `PATH`. La comprobación es la misma desde PowerShell o `cmd`:

```
gnuplot --version
```

Para abrir PNG o PDF en Windows puede utilizarse el visor asociado del sistema; ImageMagick sólo es necesario en los ejercicios que procesan o combinan imágenes.

Sesión interactiva y ejecución por lotes

En una terminal puede iniciarse una sesión:

```
gnuplot
```

Dentro de ella se pueden probar órdenes:

```
plot sin(x)
set samples 200
replot
clear
reset
```

```
quit
```

El modo interactivo es útil para experimentar. Para automatizar conviene un archivo, por ejemplo `grafico.gnu`:

```
set terminal pngcairo size 900,600
set output 'seno.png'
set title 'Funcion seno'
set xlabel 'x [rad]'
set ylabel 'sin(x)'
set grid
set xrange [0:2*pi]
plot sin(x) with lines linewidth 2 title 'sin(x)'
```

que se ejecuta con

```
gnuplot grafico.gnu
```

También puede enviarse una orden breve con `-e` o por una tubería. La forma por lotes es la que se utilizará desde los programas porque permite controlar exactamente qué órdenes fueron ejecutadas.

Persistencia de ventanas y pausas Cuando se usa un terminal gráfico interactivo, la opción `-p` o `-persist` evita que la ventana desaparezca al terminar la entrada de órdenes. Por ejemplo:

```
echo "plot sin(x)" | gnuplot -p
gnuplot -p grafico.gnu
```

Dentro de un script pueden utilizarse pausas explícitas:

```
pause -1
pause mouse close
```

`pause -1` espera hasta que el usuario presione Enter. `pause mouse close` espera el evento de cierre de la ventana cuando el terminal gráfico admite interacción con el ratón. Por eso esta última forma no debe suponerse disponible en un terminal destinado sólo a archivos, como `pngcairo` o `pdfcairo`.

Los terminales interactivos como Qt o WXT pueden mantener varias ventanas numeradas. Si se usa Qt, una ventana concreta puede cerrarse desde Gnuplot con:

```
set terminal qt 0 close
```

El número identifica la ventana. En GNU/Linux, si un proceso gráfico externo quedara bloqueado, puede localizarse antes de terminarlo:

```
ps ax | grep gnuplot
killall gnuplot_qt
killall gnuplot_wxt
```

`killall` termina todos los procesos del usuario que tengan ese nombre, por lo que debe reservarse para procesos realmente atascados y no usarse como forma normal de cerrar una figura.

Terminal y archivo de salida

En Gnuplot la palabra *terminal* no significa necesariamente la consola de texto. Designa el dispositivo o formato gráfico de salida. Por ejemplo, `pngcairo`, `pdfcairo` y `svg` generan archivos de familias diferentes.

La secuencia conceptual es:

```
set terminal pdfcairo
set output 'figura.pdf'
plot sin(x)
```

El terminal debe fijarse antes del archivo de salida. Un script que sólo depende de formatos disponibles en la instalación puede ejecutarse sin abrir una ventana gráfica.

Funciones y archivos de datos

Gnuplot puede evaluar directamente funciones:

```
set xrange [-1:4]
f(x) = x**2 - 3*x + 2
plot f(x) with lines title 'f(x)'
```

pero cuando el cálculo lo realizó otro lenguaje es preferible que lea los datos. Un archivo

```
# x    y1    y2
0.0    0.0   0.0
0.5    0.4   0.8
1.0    0.8   1.6
```

puede representarse con

```
plot 'datos.txt' using 1:2 with lines title 'y1', \
     'datos.txt' using 1:3 with points title 'y2'
```

La expresión `using 1:3` significa que se toma la primera columna como abscisa y la tercera como ordenada. Esto permite separar claramente el formato de datos de la decisión gráfica.

Bloques, huecos y datos ausentes

Una línea en blanco dentro de un archivo 2D separa bloques y puede cortar una curva. Es una forma sencilla de evitar que se unan dos ramas de una función. También puede configurarse un marcador textual para datos ausentes, o utilizar NaN cuando el formato y la operación lo admitan. Si un archivo generado por el programa contiene la cadena `NaN`, puede declararse explícitamente:

```
set datafile missing 'NaN'
plot 'datos.txt' using 1:2 with lines
```

El valor ausente corta la curva en lugar de obligar a Gnuplot a unir puntos a ambos lados de una singularidad.

En datos de superficie la separación de bloques tiene otra función. Un formato textual habitual para una grilla contiene bloques con un valor de x fijo y varios valores de y, separados por una línea vacía:

```
x1 y1 z11
x1 y2 z12
x1 y3 z13

x2 y1 z21
x2 y2 z22
x2 y3 z23
```

La línea vacía declara la estructura de la malla. No debe considerarse un adorno: permite al lector del archivo distinguir una fila de la grilla de la siguiente sin deducir esa estructura a partir de los valores numéricos.

Estilos y elementos descriptivos

Los modificadores más frecuentes son conceptualmente equivalentes a las propiedades de las bibliotecas directas:

```
plot 'd.txt' using 1:2 with lines linewidth 2 dashtype 2 title
   'modelo', \
     'd.txt' using 1:3 with points pointtype 7 pointsize 1.2
        title 'datos'
```

Se agregan títulos, ejes y cuadrícula con `set title`, `set xlabel`, `set ylabel` y `set grid`. Los rangos se controlan con `set xrange` y `set yrange`. Para curvas planas en las que la geometría debe conservarse puede usarse una relación de aspecto unitaria.

La leyenda se controla con `set key`. Puede desplazarse o colocarse fuera del área de ejes, por ejemplo:

```
set key top right
set key outside right
```

Para destacar puntos particulares pueden superponerse puntos, etiquetas o flechas. Por ejemplo, una raíz en $(1, 0)$ puede marcarse con:

```
set label 'raiz' at 1,0 offset 1,1
set arrow from 1,0.8 to 1,0 head filled
plot f(x) with lines
```

`set arrow` crea una anotación geométrica; no modifica los datos de la curva. Para com-

posiciones donde no se desea el marco cartesiano puede emplearse `unset border`:

```
unset border
```

Debe hacerse sólo cuando la pérdida del marco no dificulte la interpretación.

Escalas logarítmicas

Los ejes se cambian con órdenes como:

```
set logscale y
plot 'datos.txt' using 1:2 with lines
unset logscale y
set logscale xy
replot
```

No se debe usar una escala logarítmica como forma de ocultar ceros, negativos o desbordamientos. El programa que genera los datos debe saber cuáles valores son válidos para la escala elegida.

Coordenadas polares

Gnuplot puede representar directamente una función polar:

```
set polar
set trange [-2*pi:2*pi]
set size ratio -1
set grid polar
plot sin(t)**2 with lines title 'r=sin^2(t)'
```

Cuando los datos ya fueron transformados a (x, y) por el lenguaje principal, no es necesario activar el modo polar: se grafica el archivo cartesiano como cualquier otra serie.

Curvas paramétricas

Una trayectoria plana también puede estar dada por dos funciones de un mismo parámetro,

$$x = x(t), \qquad y = y(t).$$

En Gnuplot se activa el modo paramétrico y se fija el intervalo del parámetro:

```
set parametric
set trange [0:7.2]
set size ratio -1
v0 = 50.0
g  = 9.8
th = pi/4
plot v0*cos(th)*t, v0*sin(th)*t - 0.5*g*t**2 with lines
```

```
unset parametric
```

En modo paramétrico, `plot` recibe el par $x(t), y(t)$. Conviene volver a `unset parametric` antes de continuar con gráficos ordinarios dentro del mismo script.

Barras de error

Si un archivo contiene

```
# x     y      error
0.0     0.03   0.05
0.5     0.46   0.06
1.0     0.85   0.05
```

pueden utilizarse tres columnas para dibujar barras verticales:

```
plot 'mediciones.txt' using 1:2:3 with yerrorbars title
   'mediciones', \
     sin(x) with lines title 'modelo'
```

El mismo principio se extiende a errores asimétricos y a errores en ambas coordenadas.

Histogramas, generación aleatoria y `set print`

La generación de normales utilizada en el ejemplo se basa en la transformación de Box–Muller Box y Muller 1958, mientras que `rand`, `set print` y `smooth frequency` siguen la sintaxis de Gnuplot gnuplot development team 2025.

Gnuplot puede agrupar datos en intervalos mediante expresiones sobre las columnas y el suavizado por frecuencia. Un histograma de conteos puede construirse definiendo un ancho de clase y sumando un valor uno dentro de cada clase:

```
binw = 0.25
bin(x) = binw * floor(x/binw) + binw/2.0
set boxwidth binw
set style fill solid 0.7
plot 'muestra.txt' using (bin($1)):(1.0) smooth frequency with
   boxes
```

`smooth frequency` reemplaza los puntos que caen en la misma abscisa de clase por un punto cuyo valor vertical es la suma de sus contribuciones. Con `(1.0)` esa suma es precisamente el número de observaciones.

Gnuplot también puede producir datos numéricos desde el propio script. La orden `set print` redirige las órdenes `print` a un archivo y `unset print` restablece la salida normal. `rand(0)` produce números pseudoaleatorios uniformes. Dos uniformes independientes permiten obtener una normal estándar mediante la transformación de Box–Muller:

```
set print 'normales.txt'
do for [i=1:1000] {
```

```
    u1 = rand(0)
    u2 = rand(0)
    z = sqrt(-2*log(u1)) * cos(2*pi*u2)
    print z
}
unset print
```

El archivo resultante puede graficarse con el histograma anterior. Si se superpone la densidad normal

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2},$$

debe respetarse la normalización: para un histograma de conteos con N datos y ancho de clase h, la curva comparable es $Nh\,\phi(x)$.

Cuando los datos ya provienen de un lenguaje de programación, resulta igualmente válido calcular allí los límites y conteos y usar Gnuplot únicamente para dibujar las barras. Esta segunda estrategia hace explícita la parte estadística del algoritmo.

Semillas y reproducibilidad pseudoaleatoria Un generador pseudoaleatorio produce una secuencia determinista a partir de un estado interno. La *semilla* fija ese estado inicial. Por eso, si se desea que un experimento numérico pueda repetirse, no alcanza con guardar los datos finales: conviene registrar también la semilla y el generador utilizado. Una misma semilla, usada con el mismo generador y la misma secuencia de llamadas, debe reproducir la misma sucesión de números.

En Gnuplot, `rand(0)` obtiene el siguiente valor de la secuencia actual, mientras que un entero positivo pasado a `rand` fija las semillas internas gnuplot development team 2025. Un patrón reproducible es:

```
semilla = 12345
aux = rand(semilla)     # fija el estado inicial

u1 = rand(0)
u2 = rand(0)
# ... continuar siempre desde esta misma secuencia ...
```

En Scilab también debe fijarse la semilla del generador que realmente se vaya a usar Scilab Enterprises 2026c. Con `rand` puede hacerse, por ejemplo:

```
semilla = 12345
rand("seed", semilla)
datos = rand(1, 1000, "normal")
```

Si se utiliza `grand`, su estado es independiente y debe inicializarse sobre ese mismo generador:

```
semilla = 12345
grand("setsd", semilla)
datos = grand(1, 1000, "nor", 0, 1)
```

Cambiar la semilla permite obtener otra realización de la misma experiencia; repetirla permite verificar resultados, localizar errores y comparar dos implementaciones bajo exactamente la misma entrada pseudoaleatoria.

Ajuste de datos

Gnuplot incluye la orden `fit`, que puede estimar parámetros de un modelo. Para una recta:

```
f(x) = a*x + b
a = 1
b = 0
fit f(x) 'regresion.txt' using 1:2 via a,b
plot 'regresion.txt' using 1:2 with points title 'datos', \
     f(x) with lines title 'ajuste'
```

En las soluciones multilenguaje conviene distinguir dos situaciones. En la modalidad Gnuplot puede usarse `fit`; en la modalidad directa se pueden calcular a y b en el lenguaje y pasar la recta ya determinada a su sistema gráfico. El resultado matemático debe coincidir dentro de la precisión numérica.

Paneles múltiples

Una figura puede dividirse en paneles mediante `multiplot`:

```
set multiplot layout 2,2
set title 'Trayectoria'
plot 'p.txt' using 2:3 with lines
set title 'Velocidades'
plot 'p.txt' using 1:4 with lines, 'p.txt' using 1:5 with lines
# ... dos paneles adicionales ...
unset multiplot
```

Los paneles que comparan una misma variable independiente deberían conservar rangos compatibles. Un autoescalado diferente en cada panel puede dificultar la comparación.

Superficies y mallas

Para datos de tres columnas organizados como grilla se utiliza `splot`. En tres dimensiones se agrega `set zlabel`; los límites de los tres ejes se controlan de manera análoga:

```
set xrange [-8:8]
set yrange [-8:8]
set zrange [-1:1]
set xlabel 'x'
set ylabel 'y'
set zlabel 'z'
```

Una malla y una superficie coloreada pueden obtenerse, por ejemplo, con:

```
set hidden3d
set view 55,35
splot 'superficie.txt' using 1:2:3 with lines title 'malla'
```

```
set pm3d depthorder
set view 55,35
splot 'superficie.txt' using 1:2:3 with pm3d notitle
```

La cantidad de puntos de la grilla afecta tanto al cálculo como a la figura. Una grilla gruesa puede perder estructura; una extremadamente fina puede producir archivos innecesariamente grandes.

Cuando Gnuplot evalúa directamente una función de dos variables, la densidad de isolíneas de muestreo se controla con `set isosamples`:

```
set isosamples 20,20
splot sin(sqrt(x**2+y**2))/sqrt(x**2+y**2)

set isosamples 80,80
replot
```

Aumentar `isosamples` refina una superficie o un mapa de contornos de una función, a costa de más evaluaciones. Esta orden no refina mágicamente un archivo de datos ya discretizado: en ese caso la resolución debe aumentarse al generar la grilla.

Curvas de nivel y mapas de color

Las curvas de nivel son conjuntos de puntos en los que

$$f(x, y) = c.$$

Su separación informa sobre la rapidez de variación de la función, y el gradiente es perpendicular a ellas. En Gnuplot puede obtenerse una vista de planta con mapa de color y contornos mediante las opciones de `pm3d`, `set view map` y `set contour`.

```
set view map
set pm3d map
set contour base
set cntrparam levels 15
set colorbox
splot 'superficie.txt' using 1:2:3 with pm3d notitle
```

Los contornos también pueden mostrarse sobre la propia superficie 3D o proyectarse sobre un plano horizontal. Estas representaciones muestran la misma función pero responden a preguntas visuales distintas. Si se desean únicamente las líneas de contorno, sin dibujar la superficie, puede emplearse `unset surface` junto con `set contour base`:

```
set view map
set contour base
```

```
unset surface
splot f(x,y) with lines
```

Al terminar conviene restaurar el estado si el script continúa con otras figuras.

Paletas de color

El color debe tratarse como una codificación de datos. Para magnitudes ordenadas conviene una paleta secuencial; para magnitudes con un punto central relevante, como cero, una paleta divergente. Las paletas arcoíris pueden crear fronteras perceptuales inexistentes y ocultar otras variaciones Borland y R. M. Taylor 2007; Crameri, Shephard y Heron 2020. Un mapa de color cuantitativo necesita una barra que relacione color y valor.

Gnuplot permite definir la correspondencia color–valor de varias maneras. Por ejemplo:

```
set palette defined (-1 'blue', 0 'white', 1 'red')
set colorbox
```

crea una paleta divergente centrada en cero. Para una paleta predefinida puede usarse, por ejemplo,

```
set palette rgbformulae 33,13,10
```

y para escala de grises:

```
set palette grey
```

La paleta debe elegirse según el significado de los datos, no sólo por su apariencia.

Animación con Gnuplot

Gnuplot puede participar en una animación generando una secuencia de imágenes. El programa principal calcula los estados, produce para cada instante los datos o parámetros necesarios y ejecuta un script que guarda cuadros numerados:

```
pend_0001.png
pend_0002.png
pend_0003.png
...
```

Luego esos cuadros pueden combinarse en GIF, vídeo u otro formato. En GNU/Linux con ImageMagick puede utilizarse, según la versión instalada:

```
convert -delay 4 -loop 0 pend_*.png pendulo.gif
# o, en instalaciones que usan el ejecutable principal de
   ImageMagick 7:
magick -delay 4 -loop 0 pend_*.png pendulo.gif
```

El patrón `pend_*.png` debe expandirse en el orden correcto de los cuadros; por eso conviene numerarlos con ceros a la izquierda (`pend_0001.png`, `pend_0002.png`, etc.). En esta

arquitectura no se modifica un objeto gráfico vivo: cada cuadro es una figura nueva. Esta diferencia será importante al comparar Gnuplot con los sistemas de objetos gráficos de los lenguajes.

Complemento: Introducción a Gnuplot

Gnuplot es un programa de graficación de código abierto disponible en todos los sistemas operativos. A diferencia de los entornos de cálculo integrados, Gnuplot es una herramienta especializada exclusivamente en la generación de gráficas: recibe datos y un script con instrucciones y produce una imagen o la muestra en pantalla.

Su modelo de uso es el de los lenguajes WYGIWYW: el usuario escribe un script de texto que describe exactamente como debe verse la gráfica, y Gnuplot lo interpreta. Esto tiene ventajas importantes: los scripts son reproducibles, versionables con Git, fáciles de automatizar y pueden generarse programáticamente desde cualquier lenguaje gnuplot development team 2025; gnuplot Team 2026.

Gnuplot puede usarse de varias formas:

Interactivamente: se escribe `gnuplot` en la terminal y se ingresan comandos uno por uno, viendo el resultado inmediatamente.

Por script: se guarda el script en un archivo `.gnu` y se ejecuta con `gnuplot script.gnu`.

Desde otro programa: Scilab, Python u otro programa genera el script y llama a Gnuplot mediante una llamada al sistema.

Complemento: Primeros pasos

Verificar la instalación:

```
$gnuplot --version
```

Iniciar la consola interactiva:

```
$gnuplot
```

```
gnuplot> plot sin(x)
gnuplot> quit
```

Ejecutar un script:

```
$gnuplot mi_script.gnu
```

Para que la ventana gráfica no se cierre inmediatamente, Gnuplot ofrece varias formas de mantenerla abierta. Desde la terminal del sistema puede usarse la opción `-p` o `-persist`:

```
$gnuplot -p mi_script.gnu
$gnuplot --persist mi_script.gnu
```

También puede agregarse una pausa dentro del propio script. La instrucción `pause -1` mantiene la ventana abierta hasta que el usuario presione Enter:

```
plot sin(x)
pause -1
```

Otra posibilidad es usar `pause mouse close`, que mantiene la ventana abierta hasta que se cierre con el mouse:

```
plot sin(x)
pause mouse close
```

La diferencia práctica es que `-p` se usa al llamar a Gnuplot desde la terminal, mientras que `pause` queda escrito dentro del script. Si el gráfico aparece y desaparece rápidamente, normalmente falta una de estas dos cosas.

Complemento: Estructura de un script Gnuplot

Un script Gnuplot tiene la siguiente estructura general:

Listing 7.1: estructura general de un script

```
#Script
# Comentarios comienzan con #

# 1. Terminal de salida
set terminal png size 800,500
set output 'grafica.png'

# 2. Elementos de la grafica
set title  'Titulo de la grafica'
set xlabel 'Eje x [unidad]'
set ylabel 'Eje y [unidad]'
set grid
set key top right     # posicion de la leyenda

# 3. Rango de los ejes (opcional, Gnuplot lo calcula solo)
set xrange [-3.14 : 3.14]
set yrange [-1.5  : 1.5]

# 4. El comando plot
plot sin(x) with lines lw 2 lc 'blue' title 'seno'

# Para PNG en pantalla en lugar de archivo:
# quitar set terminal y set output
```

Complemento: Gráfica de funciones matemáticas

Gnuplot puede graficar funciones matemáticas directamente, sin necesidad de datos:

Listing 7.2: funciones matemáticas

```
gnuplot> set terminal png size 800,400
gnuplot> set output 'funciones.png'
gnuplot> set grid
gnuplot> set xlabel 'x'
gnuplot> set ylabel 'f(x)'
gnuplot> set title  'Funciones trigonometricas'
gnuplot> set key top right
gnuplot> plot sin(x)       with lines lw 2 lc 'blue'  title
   'sin(x)', \
     cos(x)       with lines lw 2 lc 'red'   title 'cos(x)', \
     sin(2*x)/2   with lines lw 1 lc 'green' title 'sin(2x)/2'
```

Las funciones matemáticas disponibles en Gnuplot incluyen: `sin`, `cos`, `tan`, `exp`, `log`, `log10`, `sqrt`, `abs`, `floor`, `ceil`.

Las constantes predefinidas son `pi` y `e`.

Complemento: Gráficas paramétricas

Gnuplot también permite graficar curvas paramétricas. En una gráfica paramétrica no se escribe directamente $y = f(x)$, sino un par de funciones

$$x = x(t), \qquad y = y(t)$$

donde t es el parámetro. Para activar este modo se usa `set parametric`, y para fijar el rango del parámetro se usa `set trange`.

```
set terminal png size 800,500
set output 'parametrica.png'
set title  'Curva parametrica'
set xlabel 'x'
set ylabel 'y'
set grid
set size ratio -1

set parametric
set trange [0:2*pi]

plot cos(t), sin(t) with lines lw 2 lc 'blue' title
   'circunferencia'

unset parametric
```

En modo paramétrico, el comando `plot` recibe pares de expresiones. En el ejemplo anterior, `cos(t)` define la coordenada x y `sin(t)` define la coordenada y.

Un caso físico típico es la trayectoria de un proyectil sin rozamiento:

$$x(t) = v_0 \cos(\theta) t, \qquad y(t) = v_0 \sin(\theta) t - \frac{1}{2} g t^2.$$

Como Gnuplot trabaja con radianes, un ángulo en grados debe convertirse multiplicando por $\pi/180$.

```
set terminal png size 800,500
set output 'proyectil.png'
set title  'Trayectoria de un proyectil'
set xlabel 'x [m]'
set ylabel 'y [m]'
set grid
set size ratio -1

v0 = 50
g  = 9.8
theta = 45*pi/180
tf = 2*v0*sin(theta)/g

set parametric
set trange [0:tf]

plot v0*cos(theta)*t, v0*sin(theta)*t - 0.5*g*t**2 \
     with lines lw 2 lc 'blue' title 'trayectoria'

unset parametric
```

Cuando se termina una gráfica paramétrica conviene usar `unset parametric`, para volver al modo usual $y = f(x)$.

Complemento: Coordenadas polares

En coordenadas polares, un punto del plano no se describe directamente por x e y, sino por una distancia al origen r y un ángulo θ. Para graficar una curva polar se parte de una función

$$r = r(\theta)$$

y se convierte a coordenadas cartesianas mediante

$$x = r(\theta)\cos(\theta), \qquad y = r(\theta)\sin(\theta).$$

Por eso una curva polar puede pensarse como un caso particular de curva paramétrica: el parámetro es θ, y las dos coordenadas x e y se calculan a partir de $r(\theta)$.

Por ejemplo, para

$$r = \sin^2(\theta)$$

se puede graficar:

Listing 7.3: Curva polar como curva parametrica

```
set terminal png size 800,500
set output 'polar.png'
set title  'Curva polar r = sin^2(theta)'
set xlabel 'x'
set ylabel 'y'
set grid
set size ratio -1

set parametric
set trange [-2*pi:2*pi]

r(t) = sin(t)**2

plot r(t)*cos(t), r(t)*sin(t) \
     with lines lw 2 lc 'blue' title 'r=sin^2(theta)'

unset parametric
```

El comando `set size ratio -1` fuerza la misma escala en ambos ejes. Esto es importante en curvas polares, porque si las escalas de x e y son distintas, la forma geométrica aparece deformada.

Gnuplot también permite graficar curvas polares directamente con `set polar`. En ese caso no hace falta escribir explícitamente $x = r(\theta)\cos(\theta)$ e $y = r(\theta)\sin(\theta)$: Gnuplot interpreta que la función graficada representa el radio r, y que el parámetro angular es `t`.

Listing 7.4: Curva polar usando set polar

```
set terminal png size 800,500
set output 'polar_directa.png'
set title  'Curva polar con set polar'
set grid polar
set size ratio -1

set polar
set trange [-2*pi:2*pi]

plot sin(t)**2 with lines lw 2 lc 'blue' title 'r=sin^2(theta)'

unset polar
```

La forma con `set polar` es más breve. La forma paramétrica es más explícita, porque muestra la conversión geométrica entre coordenadas polares y cartesianas. Ambas producen la misma curva si se usa la misma función $r(\theta)$.

Complemento: Estilos de línea y color

Listing 7.5: estilos disponibles

```
#Script
set terminal png size 800,500
set output 'estilos.png'
set yrange [-1.5:1.5]
set grid

# Estilos de linea: lines, linespoints, points, dots
# lw = line width (grosor)
# lc = line color  (color)
# pt = point type  (tipo de marcador)
# ps = point size  (tamaño del marcador)
# lt = line type   (tipo de linea: 1=solida, 2=guiones, etc.)

plot sin(x) with lines      lw 2   lc 'blue'   title 'linea', \
     cos(x) with linespoints lw 1  lc 'red'  pt 7 ps 0.5 \
                                              title
                                                 'linea+puntos',
                                                 \
     sin(x+1) with points   pt 5   ps 0.8    lc 'green' title
        'puntos'
```

Colores disponibles por nombre: `red`, `blue`, `green`, `black`, `orange`, `purple`, `cyan`, `magenta`, `yellow`, `gray`. También se aceptan colores hexadecimales: `#2980b9`.

Complemento: Marcas, flechas y puntos destacados

Además de graficar curvas, Gnuplot permite agregar marcas auxiliares sobre el gráfico. Esto es útil para señalar raíces, máximos, mínimos, puntos experimentales especiales o valores de referencia.

Una forma simple de marcar puntos importantes es graficarlos como una segunda curva de puntos. Por ejemplo, para la función

$$f(x) = x^2 - 3x + 2$$

las raíces son $x = 1$ y $x = 2$. Se pueden mostrar sobre el eje x agregando puntos rojos:

```
set terminal png size 800,500
set output 'raices.png'
set title  'Raices de f(x)=x^2-3x+2'
set xlabel 'x'
set ylabel 'f(x)'
set xrange [-1:4]
set yrange [-1:5]
set grid
f(x) = x**2 - 3*x + 2

plot f(x) with lines lw 2 lc 'blue' title 'f(x)', \
```

```
        '-' using 1:2 with points pt 7 ps 1.5 lc 'red' title
           'raices'
1 0
2 0
e
```

También se pueden agregar líneas o flechas con `set arrow`. Una flecha sin punta funciona como línea auxiliar:

```
set arrow 1 from 1, graph 0 to 1, graph 1 nohead dt 2 lc 'red'
set arrow 2 from 2, graph 0 to 2, graph 1 nohead dt 2 lc 'red'
```

Aquí `graph 0` y `graph 1` indican la parte inferior y superior del área gráfica. Por eso las líneas verticales atraviesan todo el gráfico, independientemente del rango del eje y.

Para agregar texto explicativo se usa `set label`:

```
set label 1 'x=1' at 1,0.3 center
set label 2 'x=2' at 2,0.3 center
```

Complemento: Gráfica de puntos con datos experimentales

En ciencia, los datos experimentales se representan con puntos y la curva teórica con una línea. Los datos se leen de un archivo de texto con columnas separadas por espacios:

Listing 7.6: datos.txt

```
# x     y_experimento    error
0.0     0.05             0.03
0.5     0.49             0.04
1.0     0.84             0.03
1.5     1.00             0.04
2.0     0.91             0.03
2.5     0.60             0.04
3.0     0.14             0.03
```

Listing 7.7: gráfica de datos con barras de error

```
#Script
set terminal png size 800,400
set output 'experimento.png'
set grid
set xlabel 'x'
set ylabel 'y'
set title  'Comparacion teoria vs experimento'
set key top right

# using N:M indica que columnas usar del archivo
# ($1 $2 etc. referencian las columnas en expresiones)
plot 'datos.txt' using 1:2:3 with yerrorbars \
                 pt 7 ps 1 lc 'red'  title 'Experimento', \
```

```
     sin(x)      with lines lw 2 lc 'blue' title 'sin(x) teorico'
```

Complemento: Regresión de datos experimentales con `fit`

Además de graficar datos experimentales, Gnuplot puede ajustar una función a esos datos mediante el comando `fit`. Esto permite estimar parámetros de un modelo a partir de mediciones.

Por ejemplo, si se tienen datos aproximadamente lineales, se puede proponer el modelo

$$f(x) = ax + b$$

donde a es la pendiente y b es la ordenada al origen.

Listing 7.8: Ajuste lineal con Gnuplot

```
# Archivo de datos: regresion.txt
# x     y
# 0     1.1
# 1     2.9
# 2     5.2
# 3     6.8
# 4     9.1

set terminal png size 800,500
set output 'regresion.png'

set title  'Regresion lineal'
set xlabel 'x'
set ylabel 'y'
set grid
set key top left

# Modelo a ajustar
f(x) = a*x + b

# Valores iniciales de los parametros
a = 1
b = 0

# Ajuste de parametros
fit f(x) 'regresion.txt' using 1:2 via a,b

# Grafica de datos y modelo ajustado
plot 'regresion.txt' using 1:2 with points pt 7 ps 1.4 lc
   'red' title 'datos', \
     f(x) with lines lw 2 lc 'blue' title 'ajuste lineal'
```

El comando `fit` modifica los valores de a y b para que la función propuesta se aproxime lo mejor posible a los datos. La parte `using 1:2` indica que se usan la primera columna

como x y la segunda como y. La parte `via a,b` indica qué parámetros deben ajustarse.

El ajuste no "descubre" automáticamente la forma de la función: el usuario propone el modelo, y Gnuplot calcula los parámetros de ese modelo.

Complemento: Escala logarítmica

Listing 7.9: escalas logaritmicas

```
#Script
set terminal png size 900,350
set output 'logaritmico.png'
set multiplot layout 1,3

# log-log
set title  'log-log'
set xlabel 'x'; set ylabel 'x^2'
set logscale xy
set grid
plot [0.01:100] x**2 with lines lw 2 lc 'blue' notitle
unset logscale

# semilog-x
set title  'semilog-x'
set xlabel 'x'; set ylabel 'log10(x)'
set logscale x
set grid
plot [0.01:1000] log10(x) with lines lw 2 lc 'red' notitle
unset logscale

# semilog-y
set title  'semilog-y'
set xlabel 'x'; set ylabel 'exp(x)'
set logscale y
set grid
plot [0:5] exp(x) with lines lw 2 lc 'green' notitle
unset logscale

unset multiplot
```

Complemento: Histograma

Listing 7.10: histograma de datos

```
# Script.
#datos_hist.txt contiene un valor por linea
set terminal png size 700,400
set output 'histograma.png'
set title  'Histograma'
set xlabel 'Valor'
```

```
set ylabel 'Frecuencia'
set grid ytics
set boxwidth 0.3
set style fill solid 0.7

# numero de clases y ancho del intervalo
n      = 10
minval = -3.0
maxval =  3.0
ancho  = (maxval - minval) / n
bin(x) = ancho * floor((x - minval) / ancho) + minval + ancho/2.0

plot 'datos_hist.txt' using (bin($1)):(1.0) \
     smooth frequency with boxes \
     lc 'steelblue' notitle
```

Para generar datos aleatorios directamente desde Gnuplot puede usarse `rand(0)`. Esa función genera números uniformemente distribuidos entre 0 y 1. Para obtener números con distribución normal puede aplicarse la transformación de Box–Muller.

Listing 7.11: histograma de datos normales generados en Gnuplot

```
# Script.
set terminal png size 800,450
set output 'histograma_normal.png'

# Cantidad de datos
N = 1000

# Generar datos normales con Box-Muller
set print '/tmp/normales.txt'
do for [i=1:N] {
    u1 = rand(0)
    u2 = rand(0)
    if (u1 <= 0) u1 = 1e-12
    z = sqrt(-2*log(u1)) * cos(2*pi*u2)
    print z
}
unset print

# Histograma
binw = 0.25
bin(x) = binw * floor(x/binw) + binw/2.0

set title  'Histograma de una distribucion normal'
set xlabel 'Valor'
set ylabel 'Frecuencia'
set grid ytics
set boxwidth binw
set style fill solid 0.65
set key top right
```

```
# Curva normal teorica escalada al histograma
mu = 0.0
sigma = 1.0
normal(x) = N*binw/(sigma*sqrt(2*pi)) *
   exp(-0.5*((x-mu)/sigma)**2)

plot '/tmp/normales.txt' using (bin($1)):(1.0) \
     smooth frequency with boxes lc 'steelblue' title 'datos',
        \
     normal(x) with lines lw 2 lc 'red' title 'normal teorica'
```

En este ejemplo, `set print` redirige la salida del comando `print` hacia un archivo de datos. El ciclo `do for` genera los valores, y luego el histograma se construye agrupando cada dato en un intervalo mediante la función `bin(x)`.

La curva normal teórica debe multiplicarse por $N \cdot \Delta x$, donde N es la cantidad de datos y Δx es el ancho de cada intervalo, para que tenga la misma escala vertical que el histograma.

Complemento: Subfiguras con multiplot

Listing 7.12: subfiguras

```
#Script
set terminal png size 900,600
set output 'subfiguras.png'
set multiplot layout 2,2 title 'Cuatro graficas'

# grafica 1 (fila 1, columna 1)
set title 'Seno'
set xlabel 'x'; set ylabel 'sin(x)'
set grid
plot sin(x) with lines lw 2 lc 'blue' notitle

# grafica 2 (fila 1, columna 2)
set title 'Coseno'
set xlabel 'x'; set ylabel 'cos(x)'
plot cos(x) with lines lw 2 lc 'red' notitle

# grafica 3 (fila 2, columna 1)
set title 'Exponencial'
set xlabel 'x'; set ylabel 'exp(x)'
unset grid
set grid
plot [0:3] exp(x) with lines lw 2 lc 'green' notitle

# grafica 4 (fila 2, columna 2)
set title 'Parabolica'
set xlabel 'x'; set ylabel 'x^2'
plot x**2 with lines lw 2 lc 'orange' notitle

unset multiplot
```

Complemento: Superficies 3D

Para graficar $z = f(x, y)$:

Listing 7.13: superficie 3D

```
#Script
set terminal png size 800,600
set output 'superficie.png'
set title  'Funcion sinc 2D: sin(r)/r'
set xlabel 'x'; set ylabel 'y'; set zlabel 'z'
set hidden3d         # ocultar lineas tapadas
set ticslevel 0      # plano xy en z=0

# pm3d: colorea la superficie
set pm3d depthorder
set palette rgbformulae 33,13,10

# isosample controla la resolucion de la malla
set isosamples 50,50
set xrange [-6:6]
set yrange [-6:6]

r(x,y) = sqrt(x**2 + y**2) + 0.001
splot sin(r(x,y)) / r(x,y) with pm3d title ''
```

Listing 7.14: malla de alambre 3D

```
  #Script
set terminal png size 800,600
set output 'malla.png'
set title  'Malla de alambre'
set hidden3d
set isosamples 30,30
set xrange [-3:3]; set yrange [-3:3]

splot x*exp(-(x**2 + y**2)) with lines lc 'blue' title ''
```

Complemento: Curvas de nivel

Listing 7.15: curvas de nivel (contorno)

```
#Script
set terminal png size 800,700
set output 'contorno.png'
set title  'Curvas de nivel: x^2/4 + y^2'
set xlabel 'x'; set ylabel 'y'
set xrange [-4:4]; set yrange [-3:3]
set size ratio -1    # ejes con igual escala

# solo curvas de nivel (vista desde arriba)
```

```
set view map
set contour base
set cntrparam levels 10     # 10 niveles automaticos
set key off
set colorbox

splot x**2/4 + y**2 with pm3d notitle
```

Listing 7.16: curvas de nivel con relleno de color

```
#Script
set terminal png size 800,600
set output 'contorno_relleno.png'
set title  'Curvas de nivel con relleno'
set xlabel 'x'; set ylabel 'y'
set xrange [-3:3]; set yrange [-3:3]
set view map
set pm3d map
set palette defined (0 'blue', 0.5 'green', 1 'red')
set isosamples 60,60

splot x**2 + y**2 with pm3d notitle
```

También puede elegirse explícitamente qué niveles se desean mostrar. Para eso se usa `set cntrparam levels discrete`. Por ejemplo, si se quiere destacar sólo la curva de nivel $f(x, y) = 0$, se puede pedir el nivel cero:

Listing 7.17: curva de nivel particular

```
# Script.
set terminal png size 800,600
set output 'nivel_cero.png'
set title  'Curva de nivel f(x,y)=0'
set xlabel 'x'
set ylabel 'y'
set xrange [-4:4]
set yrange [-4:4]
set size ratio -1
set grid

f(x,y) = cos(x)*exp(-y**2/4)

set view map
set contour base
set cntrparam levels discrete 0
unset surface

splot f(x,y) with lines lw 3 lc 'black' notitle
```

Si se desea dibujar primero un mapa de color y luego superponer una curva de nivel destacada, conviene generar la curva de nivel en un archivo auxiliar con `set table` y después graficarla encima del mapa.

Listing 7.18: mapa de color con curva f

```
# Script.
set terminal png size 800,600
set output 'nivel_color_cero.png'

f(x,y) = cos(x)*exp(-y**2/4)

set xrange [-4:4]
set yrange [-4:4]
set isosamples 80,80

# Guardar la curva de nivel f=0 en un archivo
set contour base
set cntrparam levels discrete 0
unset surface
set table '/tmp/nivel_cero.dat'
splot f(x,y)
unset table

# Dibujar mapa de color y superponer la curva f=0
set title  'Mapa de color con curva f=0'
set xlabel 'x'
set ylabel 'y'
set size ratio -1
set view map
set pm3d map
set palette defined (-1 'blue', 0 'white', 1 'red')
set colorbox
unset contour

splot f(x,y) with pm3d notitle, \
      '/tmp/nivel_cero.dat' using 1:2:(0) with lines lw 3 lc
         'black' title 'f=0'
```

Complemento: Campo vectorial

Listing 7.19: campo vectorial 2D

```
# Script, datos_campo.txt: x y dx dy  (una flecha por linea)
set terminal png size 700,700
set output 'campo.png'
set title  'Campo vectorial del gradiente'
set xlabel 'x'; set ylabel 'y'
set xrange [-2.5:2.5]; set yrange [-2.5:2.5]
set size ratio -1
set grid

# using x:y:dx:dy
# as escala las flechas
plot 'datos_campo.txt' using 1:2:3:4 \
```

```
        with vectors head filled lw 1 lc 'blue' notitle
```

Complemento: Exportación a distintos formatos

Listing 7.20: terminales disponibles

```
#Script
# PNG (para web e informes)
set terminal png size 800,500 enhanced font 'Arial,11'
set output 'grafica.png'

# SVG (vectorial, para web y LaTeX con paquete svg)
set terminal svg size 800,500 enhanced font 'Arial,11'
set output 'grafica.svg'

# PDF (vectorial, ideal para LaTeX)
set terminal pdfcairo size 14cm,9cm enhanced font 'Arial,10'
set output 'grafica.pdf'

# EPS (PostScript encapsulado, para LaTeX clasico)
set terminal epscairo size 14cm,9cm enhanced font 'Arial,10'
set output 'grafica.eps'

# Para incluir el PDF en LaTeX:
# \includegraphics{grafica.pdf}
```

7.2.7. Invocación de Gnuplot desde un programa

El patrón general es el mismo en todos los lenguajes:

1. calcular y validar los datos;
2. escribir el archivo de datos si es necesario;
3. construir o copiar un archivo `.gnu`;
4. ejecutar el programa `gnuplot` como proceso hijo;
5. esperar su finalización cuando la figura sea necesaria para continuar;
6. comprobar el código de salida;
7. conservar o mostrar la salida de diagnóstico si hubo error;
8. verificar que el archivo gráfico esperado exista y sea utilizable.

La construcción del archivo de órdenes puede formar parte del propio programa. Por ejemplo, en Python puede generarse un script completo antes de invocar Gnuplot:

```
from pathlib import Path

script = """set terminal pngcairo size 900,600
set output 'figura.png'
set xlabel 'x'
set ylabel 'y'
set grid
plot 'datos.txt' using 1:2 with lines title 'modelo'
"""
Path("grafico.gnu").write_text(script, encoding="utf-8")
```

Scilab, Octave, Perl y Node.js pueden producir exactamente el mismo archivo con sus operaciones ordinarias de escritura. Lo importante es que tanto los datos como la especificación gráfica sean resultados reproducibles del programa, no órdenes editadas manualmente después del cálculo.

Conviene pasar nombres de archivo como argumentos separados siempre que la API del lenguaje lo permita. Construir una cadena que será interpretada por un *shell* obliga a tratar espacios, comillas y caracteres especiales. Si los nombres provienen de una entrada externa, una concatenación descuidada puede convertirse además en un problema de seguridad.

Scilab: Gnuplot mediante `host`

La invocación de procesos y la escritura matricial se documentan en Scilab Enterprises 2026b; Scilab Enterprises 2026a; Scilab Enterprises 2026c.

En versiones actuales de Scilab la función general para ejecutar órdenes del sistema operativo es `host`. Las antiguas variantes `unix_*` son obsoletas; por portabilidad del material nuevo se utilizará `host`.

```
[estado, salida, error_sistema] = host("gnuplot grafico.gnu")

if estado <> 0 then
    mfprintf(0, "Gnuplot termino con codigo %d\n", estado)
    mfprintf(0, "%s\n", error_sistema)
    error("No se pudo generar la figura")
end

if ~isfile("figura.png") then
    error("Gnuplot termino pero figura.png no existe")
end
```

El mismo programa puede escribir previamente el archivo de datos con `fprintfMat`, `mfprintf` o las operaciones de archivos vistas en la unidad anterior. Para una grilla 3D con líneas vacías entre bloques suele ser necesario un control explícito de la escritura.

GNU Octave: Gnuplot mediante `system`

Octave puede crear un proceso síncrono con `system`. Si se piden valores de retorno, el primero es el estado de terminación y el segundo contiene la salida estándar capturada.

```
[estado, salida] = system("gnuplot grafico.gnu");

if estado != 0
  error("Gnuplot termino con codigo %d\n%s", estado, salida);
endif

if exist("figura.png", "file") != 2
  error("No se genero figura.png");
endif
```

Cuando interese una comunicación más continua con un proceso externo Octave también dispone de mecanismos de tuberías, pero para generar una figura desde un script suele ser más simple ejecutar un proceso por lotes y esperar su terminación.

Python: Gnuplot mediante `subprocess`

Python ofrece el módulo `subprocess`. Para una ejecución corta y síncrona, `subprocess.run` permite pasar el programa y sus argumentos como una lista, sin construir una orden de shell.

```
from pathlib import Path
import subprocess

p = subprocess.run(
    ["gnuplot", "grafico.gnu"],
    text=True,
    capture_output=True
)

if p.returncode != 0:
    raise RuntimeError(
        f"Gnuplot termino con codigo {p.returncode}\n{p.stderr}"
    )

if not Path("figura.png").is_file():
    raise RuntimeError("No se genero figura.png")
```

El uso de una lista de argumentos evita que el shell tenga que reinterpretar la orden. Si realmente se requieren tuberías o redirecciones de sintaxis de shell se debe hacer esa elección de manera explícita.

Perl: Gnuplot como proceso externo

Perl puede utilizar `system` cuando sólo interesa ejecutar el proceso y obtener su código de salida:

```
use strict;
use warnings;

my $rc = system("gnuplot", "grafico.gnu");

die "No se pudo ejecutar gnuplot: $!\n"
    if $rc == -1;
die "Gnuplot termino por una senal\n"
    if $rc & 127;

my $estado = $rc >> 8;
die "Gnuplot termino con codigo $estado\n"
    if $estado != 0;

die "No se genero figura.png\n"
    unless -f "figura.png";
```

Cuando se necesita capturar por separado entrada estándar, salida estándar y error estándar puede emplearse `IPC::Open3`, como en los ejercicios de procesos externos de la unidad anterior.

JavaScript/Node.js: Gnuplot mediante `child_process`

Node.js dispone del módulo `node:child_process`. Para los prácticos, en los que el programa debe esperar a que se genere el archivo antes de continuar, una llamada síncrona resulta simple:

```
"use strict";

const fs = require("node:fs");
const { spawnSync } = require("node:child_process");

const p = spawnSync("gnuplot", ["grafico.gnu"], {
    encoding: "utf8"
});

if (p.error) {
    throw p.error;
}

if (p.status !== 0) {
    throw new Error(
        `Gnuplot termino con codigo ${p.status}\n${p.stderr}`
    );
}
```

```
if (!fs.existsSync("figura.png")) {
    throw new Error("No se genero figura.png");
}
```

Para programas interactivos o servidores es preferible el modelo asíncrono con `spawn` o `execFile`, porque no bloquea el bucle de eventos. El concepto importante es el mismo: Gnuplot es otro proceso, y el programa debe tratar su terminación y sus canales de salida como cualquier otro proceso externo.

Gnuplot por una tubería: proceso persistente y datos en tiempo de ejecución

La ejecución mediante un archivo `.gnu` no es la única forma de controlar Gnuplot desde otro programa. Gnuplot lee órdenes por su entrada estándar; por lo tanto, un lenguaje que pueda iniciar un proceso hijo y conservar abierta una tubería hacia su entrada estándar puede enviarle órdenes y datos mientras el programa continúa calculando. Conceptualmente se tienen dos arquitecturas:

Por lotes	**Proceso persistente**
programa → archivo `.gnu` → Gnuplot	programa → tubería/`stdin` → Gnuplot abierto
Gnuplot termina después del guion	Gnuplot recibe sucesivas órdenes durante el cálculo

La segunda forma es especialmente útil para mostrar la evolución de un cálculo largo, enviar datos que existen sólo en memoria o generar sucesivas representaciones sin iniciar un proceso Gnuplot nuevo en cada paso.

C no dibuja La biblioteca estándar de C no proporciona funciones de graficación: ofrece archivos, texto, memoria y las demás operaciones del lenguaje, pero no una API de ventanas Kernighan y Ritchie 1988; ISO/IEC JTC 1/SC 22 2018. Existen bibliotecas gráficas para C; sin embargo, para mostrar con claridad la comunicación entre procesos resulta muy directo dejar que Gnuplot Williams, Kelley y many others 2026 dibuje. Hay dos maneras de hacerlo, y la segunda permite observar sin capas intermedias el funcionamiento de una tubería.

Primera manera en C: archivo y `system` El programa calcula, escribe los datos en un archivo de texto y llama a Gnuplot con un guion ya escrito mediante `system` de `stdlib.h`. Es el mismo patrón general que se usa desde los lenguajes del curso cuando se genera un archivo de datos, un archivo `.gnu` y luego se ejecuta Gnuplot como proceso externo.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void)
{
```

```
    FILE *f = fopen("seno.dat", "w");
    if (f == NULL) { perror("seno.dat"); return 1; }
    for (double x = 0; x <= 10; x += 0.1)
        fprintf(f, "%g %g\n", x, sin(x));
    fclose(f);
    return system("gnuplot seno.gnu");
}
```

Con `seno.gnu` conteniendo, por ejemplo, `set terminal pdfcairo; set output 'seno.pdf'; plot 'seno.dat' with lines`, el programa se compila con `gcc -Wall -o seno seno.c -lm`. Esta forma es adecuada para una salida final reproducible. Su límite es que el gráfico se produce como una etapa separada: no permite enviar nuevos datos al mismo proceso Gnuplot mientras el cálculo avanza.

Segunda manera en C: `popen` y datos en tiempo de ejecución `popen`, disponible en sistemas POSIX IEEE and The Open Group 2018, lanza un programa y devuelve un `FILE *` conectado a su entrada o salida estándar. Si se abre para escritura, cada `fprintf` sobre ese flujo escribe directamente en la entrada estándar de Gnuplot. Es la idea de la tubería `|` del shell, pero creada desde dentro del programa y conservada por éste.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void)
{
    FILE *gp = popen("gnuplot -persist", "w");
    if (gp == NULL) { perror("gnuplot"); return 1; }

    fprintf(gp, "set title 'seno'\n");
    fprintf(gp, "plot '-' with lines title 'sen x'\n");
    for (double x = 0; x <= 10; x += 0.1)
        fprintf(gp, "%g %g\n", x, sin(x));
    fprintf(gp, "e\n");
    fflush(gp);

    return pclose(gp);
}
```

Aquí `plot '-'` indica que los datos aparecen a continuación por la misma tubería. Una línea que contiene solamente `e` termina ese bloque de datos: no hace falta crear un archivo intermedio. `fflush` es esencial porque `FILE *` utiliza un búfer; sin vaciarlo, Gnuplot puede no recibir las órdenes inmediatamente. La opción `-persist` permite conservar la ventana gráfica después de cerrar la tubería.

Gráfico que se actualiza mientras se calcula Con la tubería abierta el programa puede enviar un nuevo `plot` cada cierto número de pasos. Esto permite seguir una simulación, producir una animación sencilla o inspeccionar un cálculo largo sin esperar su finalización. Cada cuadro puede consistir en un `plot '-'` completo, sus datos, la marca

`e` y un vaciado del búfer. Mantener fijos `xrange` y `yrange` evita que los ejes cambien de escala entre cuadros.

```c
#include <stdio.h>
#include <math.h>
#include <unistd.h>          /* usleep */

int main(void)
{
    FILE *gp = popen("gnuplot", "w");
    fprintf(gp, "set xrange [0:10]\nset yrange [-1.2:1.2]\n");
    for (double t = 0; t < 20; t += 0.1) {
        fprintf(gp, "plot '-' with lines title 't = %.1f'\n", t);
        for (double x = 0; x <= 10; x += 0.1)
            fprintf(gp, "%g %g\n", x, sin(x - t));
        fprintf(gp, "e\n");
        fflush(gp);
        usleep(50000);       /* 50 ms */
    }
    fprintf(gp, "pause mouse close\n");
    return pclose(gp);
}
```

La misma estructura puede rodear un paso de un integrador de ecuaciones diferenciales: Gnuplot no necesita conocer el origen de los datos, sólo recibe órdenes y números por su entrada estándar.

Varias curvas y archivo de salida desde la misma tubería Con `plot '-' ..., '-' ...` pueden enviarse varias series; cada una termina con su propia línea `e`. Si además se quieren conservar los datos para un informe, el programa puede escribirlos simultáneamente en un archivo. También pueden enviarse por la tubería las órdenes `set terminal` y `set output` para producir el archivo gráfico final.

```c
fprintf(gp, "plot '-' with lines title 'sen', '-' with lines
   title 'cos'\n");
for (double x = 0; x <= 10; x += 0.1) fprintf(gp, "%g %g\n", x,
   sin(x));
fprintf(gp, "e\n");
for (double x = 0; x <= 10; x += 0.1) fprintf(gp, "%g %g\n", x,
   cos(x));
fprintf(gp, "e\n");
fflush(gp);
```

Errores y portabilidad en C `popen` devuelve `NULL` si no puede lanzar el proceso; `pclose` devuelve el estado de salida, que conviene comprobar. Un comando de Gnuplot mal formado se informa por su salida de error y, en una sesión que permanece activa, normalmente pueden seguir enviándose órdenes posteriores. Si el programa termina sin `pclose`, la tubería se cierra; Gnuplot termina salvo que se haya pedido persistencia. `popen` pertenece a POSIX y no al núcleo de ISO C; en Windows existe la variante `_popen`. Una

tubería de este tipo es unidireccional: para leer además la salida del proceso hacen falta dos canales o una API que proporcione comunicación bidireccional.

Equivalentes en los cinco lenguajes del curso La posibilidad no depende de C sino de que el entorno permita conservar un proceso hijo y escribir en su `stdin`. GNU Octave ofrece `popen` y `popen2`; Python ofrece `subprocess.Popen` con `stdin=PIPE`; Perl permite abrir una tubería de escritura hacia un proceso y dispone además de los módulos `IPC::Open2`/`Open3`; y Node.js crea el proceso con `spawn` y expone `child.stdin` como flujo escribible Eaton et al. 2024; Python Software Foundation 2026a; The Perl Documentation Team 2026c; OpenJS Foundation 2026b.

Scilab constituye una excepción dentro de las APIs estudiadas aquí. La función actual `host` ejecuta un proceso externo y es adecuada para el patrón por lotes. La documentación actual de `mopen` describe apertura de archivos y no documenta la sintaxis histórica `mopen('|gnuplot','w')` como mecanismo para crear una tubería hacia un proceso; por eso esa forma no se utilizará como solución del curso Scilab Enterprises 2026b; Scilab Enterprises 2026c.

Entorno	Archivo + guion	Tubería persistente hacia Gnuplot
Scilab	`host("gnuplot f.gnu")`	no documentada mediante la API estándar usada en el curso
GNU Octave	`system("gnuplot f.gnu")`	`popen("gnuplot -persist","w")`
Python	`subprocess.run(...)`	`subprocess.Popen(..., stdin=PIPE)`
Perl	`system(...)`	`open(...,"|-","gnuplot",-persist")`
JavaScript/Node.js	`spawnSync(...)`	`spawn(...)` y `child.stdin.write(...)`
C (referencia)	`system("gnuplot f.gnu")`	`popen("gnuplot -persist","w")`

En los cuatro lenguajes del curso que ofrecen directamente esta comunicación, el patrón mínimo es el mismo: iniciar Gnuplot una sola vez, escribir órdenes y datos, forzar o respetar el vaciado del flujo cuando corresponda, cerrar el canal al terminar y esperar la finalización del proceso.

GNU Octave

```
gp = popen("gnuplot -persist", "w");
fputs(gp, "set title 'seno desde Octave'\n");
fputs(gp, "plot '-' with lines\n");
for x = 0:0.1:10
  fprintf(gp, "%g %g\n", x, sin(x));
endfor
fputs(gp, "e\n");
fflush(gp);
pclose(gp);
```

Pythor

```
import subprocess

p = subprocess.Popen(
    ["gnuplot", "-persist"],
    stdin=subprocess.PIPE,
    text=True
)
p.stdin.write("set title 'seno desde Python'\n")
p.stdin.write("plot '-' with lines\n")
for i in range(101):
    x = i / 10
    p.stdin.write(f"{x} {__import__('math').sin(x)}\n")
p.stdin.write("e\n")
p.stdin.flush()
p.stdin.close()
p.wait()
```

Perl

```
use strict;
use warnings;
use IO::Handle;

open(my $gp, "|-", "gnuplot", "-persist") or die $!;
$gp->autoflush(1);
print $gp "set title 'seno desde Perl'\n";
print $gp "plot '-' with lines\n";
for (my $i = 0; $i <= 100; $i++) {
    my $x = $i / 10;
    print $gp "$x ", sin($x), "\n";
}
print $gp "e\n";
close($gp) or die "gnuplot termino con error\n";
```

JavaScript/Node.js

```
"use strict";
const { spawn } = require("node:child_process");

const gp = spawn("gnuplot", ["-persist"]);
gp.stdin.write("set title 'seno desde Node.js'\n");
gp.stdin.write("plot '-' with lines\n");
for (let i = 0; i <= 100; ++i) {
    const x = i / 10;
    gp.stdin.write(`${x} ${Math.sin(x)}\n`);
}
gp.stdin.write("e\n");
gp.stdin.end();
```

La diferencia esencial respecto de generar un archivo `.gnu` no es la sintaxis de Gnuplot sino la duración del canal: el archivo constituye una especificación cerrada que se ejecuta

de una vez; la tubería permanece abierta y permite que el estado del cálculo determine qué órdenes y qué datos se envían a continuación.

Complemento: Datos desde Scilab y ejecución desde Scilab

Un flujo razonable en cálculo científico es:

1. Scilab calcula los datos.
2. Scilab escribe los datos en un archivo de texto.
3. Scilab genera el script de Gnuplot, o se hace a mano.
4. Scilab o la terminal llama a Gnuplot con `unix()`.
5. si no se visualizó simultáneamente en el paso anterior se visualizan los archivos gráficos con display u otro software.

Operaciones vectoriales en Scilab Cuando se trabaja con muchos valores de una variable, Scilab suele usar vectores. En ese caso hay que distinguir entre operaciones matriciales y operaciones elemento a elemento.

Si x es un vector, entonces `x^2` intenta hacer una operación matricial. En cambio, `x.^2` eleva al cuadrado cada elemento del vector. Lo mismo ocurre con la división: `/` es división matricial, mientras que `./` divide elemento a elemento.

Listing 7.21: Operaciones elemento a elemento

```
x = linspace(-2, 2, 200)

y = (x.^2 - 1) ./ (x.^3 + 1)
```

En esa expresión:

- `x.^2` calcula el cuadrado de cada valor de `x`.
- `x.^3` calcula el cubo de cada valor de `x`.
- `./` divide numerador y denominador elemento a elemento.
- El `- 1` y el `+ 1` se aplican automáticamente a cada elemento del vector.

Para escribir datos en columnas, además conviene asegurar que los vectores sean columnas. La notación `x(:)` transforma `x` en vector columna, sin importar si originalmente era fila o columna.

Listing 7.22: Vectores columna para escribir datos

```
x = linspace(0, 10, 50)
y = sin(x)

datos = [x(:) y(:)]
fprintfMat('/tmp/datos.txt', datos, '%15.8f')
```

Si se usara simplemente `[x y]` y ambos vectores fueran fila, Scilab construiría una fila larga. En cambio, `[x(:) y(:)]` construye una matriz con dos columnas: la primera para x, la segunda para y. Esa es la forma que Gnuplot espera para leer datos con `using 1:2` Scilab Enterprises 2026a; Scilab Enterprises 2026c.

Listing 7.23: Scilab escribe datos para Gnuplot

Escribir datos desde Scilab

```
// calcular datos
t = linspace(0, 4*%pi, 200)
y1 = sin(t)
y2 = exp(-0.2*t) .* sin(t)

// escribir columnas separadas por espacios
// poner encabezado
fprintfMat('/tmp/datos.txt', [t(:) y1(:) y2(:)], '%12.6f', '# t
     sin(t)   amortiguado')
```

Discontinuidades y datos faltantes Cuando una función tiene una división por cero o un punto donde no está definida, Scilab puede producir valores especiales como `%nan` o `%inf`. Si esos valores se escriben en un archivo de datos y luego se leen desde Gnuplot, el gráfico puede cortarse, deformarse o fallar. Por eso conviene decidir explícitamente qué hacer con esos puntos: eliminarlos del vector de datos, reemplazarlos por `%nan`, o indicarle a Gnuplot que ciertos valores del archivo deben tratarse como datos faltantes.

Por ejemplo, para la función

$$y = \frac{x^2 - 1}{x^3 + 1}$$

hay un problema en $x = -1$, porque allí el denominador vale cero.

Listing 7.24: Manejo de una discontinuidad antes de escribir datos

```
x = linspace(-2, 2, 200)
y = (x.^2 - 1) ./ (x.^3 + 1)

// opcion 1: eliminar el punto problematico
x1 = x(x <> -1)
y1 = (x1.^2 - 1) ./ (x1.^3 + 1)
fprintfMat('/tmp/datos_filtrados.txt', [x1(:) y1(:)], '%15.8f')

// opcion 2: marcar el punto como dato faltante
y2 = y
y2(abs(x + 1) < 1e-10) = %nan
fprintfMat('/tmp/datos_nan.txt', [x(:) y2(:)], '%15.8f')
```

En Gnuplot puede indicarse que el valor `Nan` debe ser tratado como dato faltante:

Listing 7.25: Datos faltantes en Gnuplot

```
set datafile missing 'Nan'
```

```
plot '/tmp/datos_nan.txt' using 1:2 with lines title 'funcion'
```

La primera estrategia elimina el punto donde la función no está definida. La segunda conserva el dominio original, pero marca el valor problemático como faltante. En ambos casos se evita que una división por cero genere una línea falsa, un salto artificial o un error de lectura en el gráfico.

Escritura de columnas con `fprintfMat` La función `fprintfMat` permite escribir una matriz en un archivo de texto. Es especialmente útil para pasar datos desde Scilab hacia Gnuplot, porque Gnuplot lee naturalmente archivos organizados en columnas.

La forma general es:

```
fprintfMat(nombre_archivo, matriz, formato)
```

Por ejemplo:

```
x = linspace(0, 10, 50)
y = sin(x)

fprintfMat('/tmp/datos.txt', [x(:) y(:)], '%15.8f')
```

Aquí `[x(:) y(:)]` arma una matriz de dos columnas. La primera columna contiene los valores de x, y la segunda columna contiene los valores de y. Luego, en Gnuplot, se puede graficar con:

```
plot '/tmp/datos.txt' using 1:2 with lines
```

El formato `'%15.8f'` indica cómo se escribirá cada número: ancho total de 15 caracteres y 8 cifras decimales.

No hace falta agregar manualmente `\n`; `fprintfMat` escribe una fila por línea.

También puede agregarse un encabezado al archivo:

```
fprintfMat('/tmp/datos.txt', [x(:) y(:)], '%15.8f', '# x    y')
```

Ese encabezado queda comentado con `#`, por lo que Gnuplot lo ignora al leer los datos. Esto ayuda a recordar qué representa cada columna.

Generar el script Gnuplot desde Scilab La escritura de archivos y la ejecución de órdenes del sistema desde Scilab se apoyan en las funciones documentadas para E/S y procesos externos Scilab Enterprises 2026a; Scilab Enterprises 2026b; Scilab Enterprises 2026c.

La regla fundamental: las comillas dobles que Gnuplot necesita se construyen con `ascii(34)` para evitar el error de comillas heterogéneas de Scilab:

Listing 7.26: generar script Gnuplot desde Scilab

```
NL = ascii(10)    // salto de linea
Q  = ascii(34)    // comilla doble
```

```
gp = 'set terminal png size 800,400' + NL
gp = gp + 'set output ' + Q + '/tmp/grafica.png' + Q + NL
gp = gp + 'set grid' + NL
gp = gp + 'set xlabel ' + Q + 't [s]'      + Q + NL
gp = gp + 'set ylabel ' + Q + 'Amplitud'   + Q + NL
gp = gp + 'set title '  + Q + 'Oscilacion' + Q + NL
gp = gp + 'set key top right' + NL

DAT = Q + '/tmp/datos.txt' + Q

gp = gp + 'plot ' + DAT + ' using 1:2 with lines lw 2 lc ' + ..
          Q + 'blue' + Q + ' title ' + Q + 'sin(t)' + Q + ', \'
             + NL
gp = gp + '     ' + DAT + ' using 1:3 with lines lw 2 lc ' + ..
          Q + 'red'  + Q + ' title ' + Q + 'amortiguado' + Q + NL

// guardar el script
fd = mopen('/tmp/script.gnu', 'w')
mfprintf(fd, '%s', gp)
mclose(fd)


lineas = [
      'set terminal png size 800,400'
      'set output "/tmp/grafica.png"'
      'set grid'
      'set xlabel "t [s]"'
      'set ylabel "Amplitud"'
      'set title "Oscilacion"'
      'set key top right'
      'plot "/tmp/datos.txt" using 1:2 with lines lw 2 lc "blue"
         title "sin(t)", \'
      '     "/tmp/datos.txt" using 1:3 with lines lw 2 lc "red"
         title "amortiguado"'
    ]
    mputl(lineas, '/tmp/script.gnu')


// llamar a Gnuplot
ret = unix('gnuplot /tmp/script.gnu 2>/tmp/gnu_err.txt')
if ret ~= 0 then
    printf('Error gnuplot: ver /tmp/gnu_err.txt\n')
end
```

Cuando se generan muchos scripts parecidos, puede ser más cómodo partir de una plantilla y reemplazar sólo las partes que cambian. Para eso puede usarse `strsubst`, que reemplaza todas las apariciones de una cadena por otra.

Listing 7.27: Reemplazo de texto en una plantilla

```
   plantilla = [
```

```
        'set terminal png size 800,400'
        'set output "__SALIDA__"'
        'set grid'
        'set xlabel "x"'
        'set ylabel "y"'
        'set title "__TITULO__"'
        'plot "__DATOS__" using 1:2 with lines lw 2 title
          "__LEYENDA__"'
    ]

    script = plantilla
    script = strsubst(script, '__SALIDA__',  '/tmp/grafica.png')
    script = strsubst(script, '__TITULO__',  'Grafica generada
       desde Scilab')
    script = strsubst(script, '__DATOS__',   '/tmp/datos.txt')
    script = strsubst(script, '__LEYENDA__', 'datos')

    mputl(script, '/tmp/script.gnu')
```

La ventaja de este método es que separa la estructura del script de los valores particulares. La plantilla conserva la forma general del archivo Gnuplot, y `strsubst` permite cambiar nombres de archivos, títulos, etiquetas o funciones sin reescribir todo el texto.

Listing 7.28: simulación y gráfica completa

Ejemplo completo: trayectoria del proyectil

```
// ----- simulacion -----
g   = 9.8;  v0 = 60;  ang = 40 * %pi/180
b   = 0.08; m  = 1.0
y0  = [0; 0; v0*cos(ang); v0*sin(ang)]

function dydt = fode(t, y)
    vmod = sqrt(y(3)^2 + y(4)^2)
    dydt = [y(3); y(4); -(b/m)*vmod*y(3); -g-(b/m)*vmod*y(4)]
endfunction

t   = linspace(0, 12, 500)
sol = ode(y0, 0, t, fode)

// recortar cuando y < 0
idx = find(sol(2,:) < 0)
n2  = size(t,'*'); if ~isempty(idx) then n2 = idx(1); end
x   = sol(1,1:n2);  y_t = sol(2,1:n2)
vx  = sol(3,1:n2);  vy  = sol(4,1:n2)
vel = sqrt(vx.^2 + vy.^2)

// ----- escribir datos -----
fd = mopen('/tmp/proyectil.txt', 'w')
mfprintf(fd, '# t    x     y     v\n')
for i = 1:n2
    mfprintf(fd, '%.4f  %.4f  %.4f  %.4f\n', ..
             t(i), x(i), y_t(i), vel(i))
```

```
end
mclose(fd)

// ----- script gnuplot con dos graficas -----
NL = ascii(10); Q = ascii(34)
DAT = Q + '/tmp/proyectil.txt' + Q

gp = 'set terminal png size 900,380' + NL
gp = gp + 'set output ' + Q + '/tmp/proyectil.png' + Q + NL
gp = gp + 'set multiplot layout 1,2' + NL

// trayectoria
gp = gp + 'set title '  + Q + 'Trayectoria'    + Q + NL
gp = gp + 'set xlabel ' + Q + 'x [m]'          + Q + NL
gp = gp + 'set ylabel ' + Q + 'y [m]'          + Q + NL
gp = gp + 'set grid'    + NL
gp = gp + 'plot ' + DAT + ' using 2:3 with lines lw 2 lc ' + ..
          Q + 'blue' + Q + ' notitle' + NL

// velocidad
gp = gp + 'set title '  + Q + 'Velocidad vs tiempo' + Q + NL
gp = gp + 'set xlabel ' + Q + 't [s]'               + Q + NL
gp = gp + 'set ylabel ' + Q + 'v [m/s]'             + Q + NL
gp = gp + 'plot ' + DAT + ' using 1:4 with lines lw 2 lc ' + ..
          Q + 'red' + Q + ' notitle' + NL

gp = gp + 'unset multiplot' + NL

fd = mopen('/tmp/proyectil.gnu', 'w')
mfprintf(fd, '%s', gp)
mclose(fd)

unix('gnuplot /tmp/proyectil.gnu')
printf('Grafica guardada en /tmp/proyectil.png\n')
printf('Alcance: %.1f m\n', x(n2))
printf('Altura max: %.1f m\n', max(y_t))
```

Superficie 3D desde Scilab Para gráficas 3D, Gnuplot necesita los datos en formato de grilla: cada bloque de filas con el mismo x, separados por una línea en blanco:

Listing 7.29: datos de superficie para Gnuplot

```
// grilla de puntos
x = linspace(-3, 3, 40)
y = linspace(-3, 3, 40)

fd = mopen('/tmp/superficie.txt', 'w')
for i = 1:length(x)
    for j = 1:length(y)
        z = x(i)^2/4 + y(j)^2
        mfprintf(fd, '%.4f  %.4f  %.4f\n', x(i), y(j), z)
```

```
    end
    mfprintf(fd, '\n')   // linea en blanco entre bloques
end
mclose(fd)

// script gnuplot
NL = ascii(10); Q = ascii(34)
gp = 'set terminal png size 800,600' + NL
gp = gp + 'set output ' + Q + '/tmp/superficie.png' + Q + NL
gp = gp + 'set title '  + Q + 'x^2/4 + y^2'         + Q + NL
gp = gp + 'set xlabel ' + Q + 'x' + Q + NL
gp = gp + 'set ylabel ' + Q + 'y' + Q + NL
gp = gp + 'set zlabel ' + Q + 'z' + Q + NL
gp = gp + 'set hidden3d' + NL
gp = gp + 'set pm3d depthorder' + NL
gp = gp + 'set isosamples 40,40' + NL
DAT = Q + '/tmp/superficie.txt' + Q
gp = gp + 'splot ' + DAT + ' with pm3d notitle' + NL

fd = mopen('/tmp/superficie.gnu', 'w')
mfprintf(fd, '%s', gp)
mclose(fd)
unix('gnuplot /tmp/superficie.gnu')
```

Listing 7.30: curvas de nivel desde Scilab

Curvas de nivel desde Scilab

```
x = linspace(-3, 3, 60)
y = linspace(-3, 3, 60)

fd = mopen('/tmp/contorno.txt', 'w')
for i = 1:length(x)
    for j = 1:length(y)
        z = x(i)^2 + 2*y(j)^2
        mfprintf(fd, '%.4f  %.4f  %.4f\n', x(i), y(j), z)
    end
    mfprintf(fd, '\n')
end
mclose(fd)

NL = ascii(10); Q = ascii(34)
DAT = Q + '/tmp/contorno.txt' + Q

gp = 'set terminal png size 700,600' + NL
gp = gp + 'set output ' + Q + '/tmp/contorno.png' + Q + NL
gp = gp + 'set title '  + Q + 'x^2 + 2y^2 = c'    + Q + NL
gp = gp + 'set xlabel ' + Q + 'x' + Q + NL
gp = gp + 'set ylabel ' + Q + 'y' + Q + NL
gp = gp + 'set size ratio -1' + NL
gp = gp + 'set view map' + NL
gp = gp + 'set pm3d map' + NL
```

```
gp = gp + 'set contour base' + NL
gp = gp + 'set cntrparam levels 12' + NL
gp = gp + 'set key off' + NL
gp = gp + 'splot ' + DAT + ' with pm3d notitle' + NL

fd = mopen('/tmp/contorno.gnu', 'w')
mfprintf(fd, '%s', gp)
mclose(fd)
unix('gnuplot /tmp/contorno.gnu')
```

Animación: secuencia de PNG Gnuplot puede generar una secuencia de imágenes que luego se convierten en un GIF animado o video:

Listing 7.31: secuencia de PNG para animación

```
// simular oscilador
t  = linspace(0, 10, 200)
x  = exp(-0.3*t) .* cos(2*t)

NL = ascii(10); Q = ascii(34)

// generar un PNG por cada cuadro
for i = 1:5:length(t)
    nombre = sprintf('/tmp/frame_%04d.png', i)
    gp = 'set terminal png size 600,300' + NL
    gp = gp + 'set output ' + Q + nombre + Q + NL
    gp = gp + 'set xrange [0:10]' + NL
    gp = gp + 'set yrange [-1.2:1.2]' + NL
    gp = gp + 'set grid' + NL
    gp = gp + 'set title ' + Q + sprintf('t = %.2f s', t(i)) + Q
       + NL

    // datos hasta el cuadro actual
    fd = mopen('/tmp/osc_parcial.txt', 'w')
    for j = 1:i
        mfprintf(fd, '%.4f  %.4f\n', t(j), x(j))
    end
    mclose(fd)

    // punto actual
    fd = mopen('/tmp/osc_punto.txt', 'w')
    mfprintf(fd, '%.4f  %.4f\n', t(i), x(i))
    mclose(fd)

    DAT = Q + '/tmp/osc_parcial.txt' + Q
    PTO = Q + '/tmp/osc_punto.txt'   + Q
    gp = gp + 'plot ' + DAT + ' with lines lw 2 lc ' + ..
              Q + 'blue' + Q + ' notitle, \' + NL
    gp = gp + '     ' + PTO + ' with points pt 7 ps 2 lc ' + ..
              Q + 'red' + Q + ' notitle' + NL
```

```
    fd = mopen('/tmp/frame.gnu', 'w')
    mfprintf(fd, '%s', gp)
    mclose(fd)
    unix('gnuplot /tmp/frame.gnu')
end

// convertir secuencia a GIF animado con ImageMagick
unix('convert -delay 5 /tmp/frame_*.png /tmp/animacion.gif')
printf('Animacion guardada en /tmp/animacion.gif\n')
```

Animación con varios objetos por cuadro En una animación física suele ser necesario dibujar varios objetos en cada cuadro: una trayectoria acumulada, una línea que representa una cuerda o barra, y un punto que representa una partícula. La idea es generar, para cada instante, uno o más archivos de datos y un script de Gnuplot que combine esos archivos en una misma gráfica.

```
// simular pendulo simple amortiguado
function dydt = fpend(t, y)
    dydt = [y(2); -9.8*sin(y(1)) - 0.1*y(2)]
endfunction

t0 = 0
theta0 = 60*%pi/180
t = linspace(0, 8, 160)

sol = ode([theta0; 0], t0, t, fpend)
theta = sol(1,:)

// posicion de la masa, longitud L=1
px = sin(theta)
py = -cos(theta)

NL = ascii(10)
Q = ascii(34)

for i = 1:length(t)

    // trayectoria acumulada hasta el cuadro actual
    fd = mopen('/tmp/pend_traza.txt', 'w')
    for j = 1:i
        mfprintf(fd, '%.6f %.6f\n', px(j), py(j))
    end
    mclose(fd)

    // cuerda: desde el origen hasta la masa actual
    fd = mopen('/tmp/pend_cuerda.txt', 'w')
    mfprintf(fd, '%.6f %.6f\n', 0, 0)
    mfprintf(fd, '%.6f %.6f\n', px(i), py(i))
    mclose(fd)
```

```
        // masa actual
        fd = mopen('/tmp/pend_masa.txt', 'w')
        mfprintf(fd, '%.6f %.6f\n', px(i), py(i))
        mclose(fd)

        nombre = sprintf('/tmp/pend_%04d.png', i)

        gp = 'set terminal png size 600,600' + NL
        gp = gp + 'set output ' + Q + nombre + Q + NL
        gp = gp + 'set title ' + Q + sprintf('Pendulo: t = %.2f
           s', t(i)) + Q + NL
        gp = gp + 'set xrange [-1.2:1.2]' + NL
        gp = gp + 'set yrange [-1.2:0.2]' + NL
        gp = gp + 'set size ratio -1' + NL
        gp = gp + 'set grid' + NL
        gp = gp + 'unset key' + NL
        gp = gp + 'plot ' + Q + '/tmp/pend_traza.txt' + Q + ' with
           lines lw 1 lc ' + Q + 'gray' + Q + ', \' + NL
        gp = gp + '     ' + Q + '/tmp/pend_cuerda.txt' + Q + '
           with lines lw 2 lc ' + Q + 'black' + Q + ', \' + NL
        gp = gp + '     ' + Q + '/tmp/pend_masa.txt' + Q + ' with
           points pt 7 ps 2 lc ' + Q + 'red' + Q + NL

        fd = mopen('/tmp/pend.gnu', 'w')
        mfprintf(fd, '%s', gp)
        mclose(fd)

        unix('gnuplot /tmp/pend.gnu')
    end

    unix('convert -delay 4 /tmp/pend_*.png /tmp/pendulo.gif')
    printf('Animacion: /tmp/pendulo.gif\n')
```

El punto central es que Gnuplot no necesita saber que se trata de un péndulo. Sólo recibe archivos de datos: uno para la traza, uno para la cuerda y otro para la masa. La interpretación física la aporta el programa que calcula esos datos.

7.2.8. Grillas, superficies, contornos y campos vectoriales

Representación de una función de dos variables

Para representar

$$z = f(x, y)$$

se construyen dos conjuntos de coordenadas

$$x_0, \ldots, x_{N_x-1}, \qquad y_0, \ldots, y_{N_y-1}$$

y una matriz

$$Z_{ij} = f(x_i, y_j).$$

Ese objeto numérico es independiente de la herramienta gráfica. Scilab, Octave, NumPy/-Matplotlib, PDL/PLplot y Plotly.js reciben de una u otra manera la misma información: dos ejes y una matriz o grilla de valores.

Una superficie 3D es útil para apreciar la forma global, pero introduce perspectiva y oclusión. Para leer valores o comparar regiones suele ser más preciso un mapa de color o un mapa de contornos. La figura 3D y el mapa 2D son representaciones complementarias, no competidoras.

Resolución de la grilla

Cambiar N_x y N_y cambia la discretización. Una grilla gruesa produce contornos angulosos y puede omitir extremos locales. Una grilla fina exige más cálculo y memoria pero aproxima mejor la superficie. La apariencia de una isolínea puede variar con la resolución aun cuando la función matemática sea la misma.

Por eso en el práctico se comparan deliberadamente varias grillas para una misma función. El objetivo no es encontrar un número mágico de puntos, sino aprender que la resolución de visualización también es un parámetro numérico.

Curvas de nivel

Una curva de nivel de valor c satisface

$$f(x, y) = c.$$

Si las curvas están muy próximas, una pequeña distancia espacial corresponde a un cambio grande de f. Si están separadas, la función cambia más lentamente. El nivel cero tiene además una interpretación especial: es el lugar geométrico de las soluciones de $f(x, y) = 0$.

Las curvas de nivel no deberían cruzarse para una función ordinaria bien definida: un mismo punto no puede tener simultáneamente dos valores diferentes de f.

Gradiente y campo de vectores

El gradiente

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

apunta en la dirección de mayor crecimiento local y es perpendicular a las curvas de nivel. Superponer un campo de flechas a un mapa de contornos permite ver esa relación.

No conviene dibujar una flecha en cada punto de una grilla muy fina. Un campo vectorial suele representarse sobre una subgrilla para conservar legibilidad. La longitud gráfica de las flechas puede escalarse; lo importante es no alterar su dirección ni ocultar la estructura de los contornos.

7.2.9. Series temporales, paneles, trayectorias y retratos de fase

Una serie temporal representa una magnitud en función del tiempo. Si un cálculo produce columnas (t, x, v), los mismos datos permiten dibujar $x(t)$ y $v(t)$ como series temporales y, sin repetir el cálculo, construir el retrato de fase $v(x)$. Si además existen coordenadas espaciales $(x(t), y(t))$, puede obtenerse la trayectoria $y(x)$. Estas vistas contienen la misma evolución numérica, pero responden a preguntas diferentes.

Un mismo fenómeno puede requerir varias representaciones. En el movimiento de un proyectil, por ejemplo, interesan la trayectoria $y(x)$, las componentes de la velocidad, las energías y una relación de fase. Superponer todas esas magnitudes en un solo sistema de ejes sería difícil de leer; una figura formada por paneles permite conservar la relación conceptual sin mezclar escalas incompatibles.

Un retrato de fase representa una variable de estado contra otra, no contra el tiempo. En un oscilador puede graficarse v frente a x. Una espiral hacia el origen muestra de manera geométrica la pérdida de energía por amortiguamiento. El tiempo sigue presente como parámetro que recorre la curva, aunque no aparezca como eje.

Cuando varios paneles comparten una variable temporal es conveniente usar el mismo intervalo y, si corresponde, las mismas marcas. De lo contrario la comparación visual puede ser engañosa Cleveland y McGill 1984.

7.2.10. Animación

Una animación numérica representa sucesivamente los distintos estados de un sistema. Si el estado de un fenómeno físico es $\mathbf{x}(t)$, una simulación produce una sucesión

$$\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$$

correspondiente a los tiempos

$$t_0, t_1, \ldots, t_n.$$

La animación transforma cada estado numérico en un estado gráfico. No conviene borrar la figura y construirla nuevamente en cada instante: se crean los objetos gráficos una sola vez y luego se modifican sus propiedades. Las funciones gráficas y de temporización utilizadas en estos ejemplos están documentadas en la ayuda oficial de Scilab Scilab Enterprises 2026c.

El esquema general es

$$\boxed{\text{estado físico}} \longrightarrow \boxed{\text{actualizar objetos gráficos}} \longrightarrow \boxed{\text{mostrar}} \longrightarrow \boxed{\text{siguiente estado}}.$$

Ejemplo: animación de un péndulo

Para pequeñas oscilaciones, un péndulo ideal de longitud L cumple

$$\theta(t) = \theta_0 \cos\left(\sqrt{\frac{g}{L}}\, t\right),$$

y la posición de la masa es

$$x = L\sin\theta, \qquad y = -L\cos\theta.$$

El siguiente programa calcula primero los estados y luego anima el movimiento modificando los mismos objetos gráficos.

```
// Parametros fisicos
g = 9.81
L = 1
theta0 = 40 * %pi / 180

// Estados de la simulacion
dt = 0.02
t = 0:dt:10
theta = theta0 * cos(sqrt(g/L) * t)

// Preparar la figura
clf()
a = gca()
a.data_bounds = [-1.2*L -1.2*L ; 1.2*L 0.2*L]
a.isoview = "on"

// Estado inicial
x = L * sin(theta(1))
y = -L * cos(theta(1))

// Crear la barra una sola vez
xpoly([0 x], [0 y], "lines")
barra = gce()
barra.thickness = 2

// Crear la masa una sola vez
xpoly(x, y, "marks")
masa = gce()
masa.mark_style = 9
masa.mark_size = 4

// Asociar un intervalo real a cada paso
realtimeinit(dt)
realtime(0)

for k = 1:length(t)
```

```
    x = L * sin(theta(k))
    y = -L * cos(theta(k))

    // Actualizar los objetos existentes
    barra.data = [0 0 ; x y]
    masa.data = [x y]

    realtime(k)
end
```

Aquí el cálculo físico y la representación están separados: las variables `theta`, `x` e `y` describen el estado del sistema, mientras que `barra` y `masa` son solamente su representación gráfica.

7.2.11. Objetos gráficos y animación

Los sistemas gráficos directos suelen mantener una escena compuesta por objetos: figura, ejes, curvas, textos y otros elementos. Esta arquitectura se conoce como modo retenido: el gráfico conserva estado y un programa puede modificar un objeto ya existente Foley et al. 1990.

En una animación resulta más eficiente mover o actualizar el objeto que borrar y recrear toda la escena. Hay tres reglas importantes:

1. fijar los límites de los ejes antes de comenzar, para evitar cambios de escala entre cuadros;
2. separar el paso de integración numérica de la frecuencia de dibujo;
3. forzar la actualización de pantalla cuando la biblioteca la difiera.

Una simulación puede usar miles de pasos internos y mostrar sólo cien o doscientos cuadros. La frecuencia de representación se elige según lo que el ojo necesita, no según la estabilidad del integrador.

La animación no siempre explica mejor que una figura estática. Es especialmente útil cuando la evolución temporal es parte del fenómeno que se quiere entender Tversky, Morrison y Bétrancourt 2002. El práctico del péndulo utiliza justamente este caso.

Interacción con el ratón y eventos gráficos

Una figura interactiva no sólo contiene objetos: también puede recibir eventos producidos por el ratón o el teclado. Conviene separar dos mecanismos. Una operación de espera devuelve cuando ocurre un clic; una operación de consulta puede utilizarse para seguir el movimiento del puntero o detectar otros eventos mientras el programa continúa actualizando la escena.

En Scilab, `xclick` espera un evento y devuelve, entre otros datos, el botón y la posición del puntero Scilab Enterprises 2026c:

```
[boton, x, y] = xclick()
```

Los valores `x` e `y` están expresados en la escala del sistema de ejes gráfico actual, no en píxeles de la pantalla. Ésta es la propiedad que permite evaluar directamente una función matemática en el punto seleccionado, por ejemplo `z=f(x,y)`. Si la figura contiene varios sistemas de ejes, se debe comprobar cuál de ellos está activo y si la selección pertenece a sus límites.

Los códigos de evento permiten distinguir los botones. En particular, `0`, `1` y `2` corresponden a la presión de los botones izquierdo, central y derecho; los códigos `3`, `4` y `5` corresponden a los clics completos de esos mismos botones. Para seguir el movimiento del puntero sin esperar exclusivamente un clic puede utilizarse `xgetmouse`. Así, un programa puede asociar acciones diferentes a cada evento.

La interacción suele necesitar además *estado propio del programa*. Si se seleccionan sucesivamente puntos de una curva de nivel, puede conservarse un contador y una tabla con las selecciones:

```
contador = contador + 1
valor = f(x, y)
selecciones(contador, :) = [x y valor]
xstring(x, y, string(contador))
```

La etiqueta creada pasa a formar parte de la escena retenida. De manera análoga, otro botón puede crear un marcador y conservarlo sin borrar los anteriores. Esta separación entre *evento*, *coordenadas de datos*, *estado del programa* y *objeto gráfico* es el patrón básico para construir selección interactiva, anotaciones y pequeñas interfaces gráficas.

Complemento: Creación y manipulación de objetos gráficos

En Scilab y otros entornos de cálculo científico, los elementos visuales de una figura son **objetos** que pueden crearse, modificarse y eliminarse durante la ejecución del programa. Esta capacidad es la base de las animaciones y las interfaces interactivas.

La distinción general se conoce en gráficos por computadora como modo retenido frente a modo inmediato Foley et al. 1990. En el modo inmediato el programa dibuja y lo dibujado deja de existir como entidad: queda una imagen. En el modo retenido el sistema mantiene la escena como un árbol de objetos con propiedades, y el programa modifica esas propiedades. Scilab trabaja en modo retenido, y por eso `plot` devuelve un *handle* con el que después se puede mover la curva sin volver a graficar.

Ésa es exactamente la diferencia con el camino que usamos en la unidad de gráficos: en gnuplot cada figura es un archivo terminado y para cambiar un punto hay que regenerar el archivo entero gnuplot Team 2026. Ninguno de los dos es mejor; sirven para cosas distintas, y por eso la animación pertenece a este capítulo y no a aquél.

El árbol de objetos tiene una estructura fija que conviene tener presente: la figura contiene ejes, los ejes contienen entidades compuestas, y éstas contienen las entidades elementales que efectivamente se dibujan. Se recorre con `gcf()`, `gca()` y el campo `children` de cada nodo, y cada objeto tiene un conjunto de propiedades que se leen y escriben con la notación

de punto o con `get` y `set` Scilab Enterprises 2026c. Lo importante es que ese árbol es el estado del dibujo: modificar una propiedad cambia lo que se ve sin recalcular nada más.

```
// Jerarquia de objetos graficos en Scilab:
// Figure TO Axes TO (Plot, Text, Rectangle, Arc, ...)

// Crear una figura y obtener su handle (identificador)
f = figure('Figure_name', 'Manipulacion de objetos')
a = gca()    // axes actual

// Crear un objeto de tipo curva
h_curva = plot(0, 0, 'bo', 'MarkerSize', 15, ...
               'MarkerFaceColor', 'blue')
a.data_bounds = [-2, -2; 2, 2]
a.isoview = 'on'
xgrid()

// Modificar propiedades del objeto en tiempo real
// (sin redibujar todo el grafico)
set(h_curva, 'xdata', 1.0)    // mover en x
set(h_curva, 'ydata', 0.5)    // mover en y

// Animacion: mover el punto en un circulo
t = linspace(0, 2*%pi, 100)
for i = 1:length(t)
    set(h_curva, 'xdata', cos(t(i)), ...
                 'ydata', sin(t(i)))
    drawnow()
    sleep(20)
end

// Cambiar otras propiedades graficas
set(h_curva, 'mark_foreground', color('red'))
set(h_curva, 'mark_size', 20)
```

```
// Objetos graficos disponibles en Scilab y sus handles

// Rectangulo
h_rect = xrect(0.5, 0.8, 0.3, 0.2)  // x,y,ancho,alto
h_rect.fill_mode = 'on'
h_rect.background = color('green')

// Elipse / arco
h_arco = xarc(0.1, 0.9, 0.8, 0.8, 0, 360*64)
h_arco.fill_mode = 'on'
h_arco.background = color('cyan')

// Texto en el grafico
h_texto = xstring(0.3, 0.5, 'Hola mundo')
h_texto.font_size = 4
h_texto.font_foreground = color('red')
```

```
// Flecha
xarrow([0.1, 0.9], [0.1, 0.9], 'arrowsize', 0.03)

// Poligono
xpolygon([0.2, 0.4, 0.3], [0.2, 0.2, 0.4])

// Modificar y eliminar objetos
set(h_texto, 'text', 'Texto modificado')
delete(h_rect)   // eliminar el rectangulo
```

Animación para simulación de fenómenos físicos La técnica fundamental para animaciones en Scilab es modificar las propiedades de objetos gráficos existentes en lugar de redibujar toda la figura en cada cuadro. Esto produce animaciones fluidas y eficientes.

Además de eficiente, es lo que hace que la animación se entienda. Si el objeto conserva su identidad entre cuadros, el ojo sigue el movimiento en lugar de tener que reconstruirlo; cuando la figura se borra y se redibuja entera, esa continuidad se pierde Heer y Robertson 2007.

Conviene la advertencia contraria. La revisión de Tversky, Morrison y Bétrancourt sobre experimentos de aprendizaje encontró que en muchos casos la animación no mejora la comprensión frente a una secuencia bien elegida de imágenes fijas Tversky, Morrison y Bétrancourt 2002. Animar tiene sentido cuando lo que se quiere mostrar *es* la evolución temporal, que es nuestro caso, y no como adorno de un resultado que se entendería igual en una gráfica estática.

Hay tres detalles técnicos que separan una animación que funciona de una que parpadea. El primero es fijar el rango de los ejes antes de empezar: si cada cuadro se autoescala, la animación tiembla y el movimiento aparente no corresponde al del sistema. El segundo es `drawnow`, que fuerza el volcado a pantalla; sin él Scilab puede acumular los cambios y mostrarlos todos juntos al final. El tercero es desacoplar el paso de integración del paso de dibujo: el método numérico puede necesitar un paso mucho más chico que el razonable para el ojo, y dibujar un cuadro por cada paso de cálculo desperdicia tiempo sin mejorar nada.

```
// Patron general de una animacion en Scilab

function animar_sistema(datos_t, datos_x, datos_y)
    // datos_t: vector de tiempos
    // datos_x, datos_y: posiciones en cada instante

    // 1. INICIALIZACION: crear todos los objetos graficos
    f = figure('Figure_name', 'Animacion')
    a = gca()
    a.data_bounds = [min(datos_x)-0.1, min(datos_y)-0.1; ...
                     max(datos_x)+0.1, max(datos_y)+0.1]
    a.isoview = 'on'

    // Crear los objetos que se moveran
```

```
    h_objeto = plot(datos_x(1), datos_y(1), 'ro', ...
                    'MarkerSize', 15, ...
                    'MarkerFaceColor', 'red')
    h_trazo  = plot(datos_x(1), datos_y(1), 'b-', ...
                    'LineWidth', 1)
    h_tiempo = xstring(a.data_bounds(1,1), ...
                       a.data_bounds(2,2)*0.95, 't = 0.00 s')

    // 2. BUCLE DE ANIMACION: modificar propiedades
    for i = 2:length(datos_t)
        // Mover el objeto (rapido: solo cambia datos)
        set(h_objeto, 'xdata', datos_x(i), ...
                      'ydata', datos_y(i))

        // Extender el trazo
        set(h_trazo, 'xdata', datos_x(1:i), ...
                     'ydata', datos_y(1:i))

        // Actualizar el tiempo
        set(h_tiempo, 'text', ...
                      sprintf('t = %.2f s', datos_t(i)))

        // Forzar el redibujo
        drawnow()

        // Control de velocidad
        sleep(20)  // 20 ms = ~50 cuadros por segundo
    end
endfunction
```

El ejemplo que sigue integra todo lo anterior: un sistema masa-resorte con amortiguamiento y forzado externo, resuelto numéricamente y animado junto a su gráfica de posición contra tiempo.

La física del oscilador amortiguado y forzado —régimen transitorio, régimen estacionario, resonancia— está desarrollada en cualquier texto de mecánica clásica J. R. Taylor 2005. El camino que va del sistema físico al modelo, del modelo a la ecuación diferencial, de la ecuación al método numérico y del método al programa es el que organiza toda la materia Cellier y Kofman 2006, y lo que `ode()` resuelve por debajo es un método de paso adaptativo Hairer, Norsett y Wanner 1993. El uso de la figura que se actualiza como parte del método de simulación, y no como decoración, está tratado en detalle en Gould, Tobochnik y Christian 2007.

```
// Simulacion y animacion del sistema masa-resorte
// m*x'' + b*x' + k*x = F0*cos(omega*t)

function simulacion_masa_resorte()
    // Parametros
    m = 1.0; k = 4.0; b = 0.3
    F0 = 0.5; omega = 1.8
    x0 = 0; v0 = 0
```

```
// Resolver la EDO
function dydt = f(t, y)
    dydt = [y(2); (F0*cos(omega*t) - b*y(2) - k*y(1))/m]
endfunction

t = linspace(0, 20, 1000)
sol = ode([x0; v0], 0, t, f)
x = sol(1,:)

// Configurar la figura con dos paneles
f = figure('Figure_name', 'Masa-resorte animado', ...
           'position', [100, 100, 900, 500])

// Panel izquierdo: animacion mecanica
subplot(1, 2, 1)
title('Animacion mecanica')
ax_anim = gca()
ax_anim.data_bounds = [-0.5, -1.5; 3.5, 1.5]
ax_anim.isoview = 'on'

// Panel derecho: grafico temporal
subplot(1, 2, 2)
plot(t, x, 'b-', 'LineWidth', 1)
xlabel('Tiempo [s]')
ylabel('Posicion x [m]')
title('Posicion vs tiempo')
xgrid()
ax_graf = gca()

// Crear objetos de la animacion mecanica
sca(ax_anim)

// Pared (fija)
h_pared = xrect(0, 0.5, 0.1, 1.0)
h_pared.fill_mode = 'on'
h_pared.background = color('darkgray')

// Resorte (se dibujara como polilinea en zig-zag)
x_res = linspace(0.1, 1.0+x(1), 20)
y_res = 0.08*sin(linspace(0, 6*%pi, 20))
h_resorte = plot(x_res, y_res, 'k-', 'LineWidth', 2)

// Masa (bloque)
pos_masa = 1.0 + x(1)
h_masa = xrect(pos_masa, 0.4, 0.4, 0.8)
h_masa.fill_mode = 'on'
h_masa.background = color('blue')

// Marcador en el grafico temporal
sca(ax_graf)
```

```
    h_punto = plot(t(1), x(1), 'ro', 'MarkerSize', 8)

    // Bucle de animacion
    for i = 1:5:length(t)
        // Actualizar resorte
        x_res = linspace(0.1, 1.0+x(i), 20)
        y_res = 0.08*sin(linspace(0, 6*%pi, 20))
        set(h_resorte, 'xdata', x_res, 'ydata', y_res)

        // Actualizar masa
        pos = 1.0 + x(i)
        h_masa.rect = [pos, 0.4, 0.4, 0.8]

        // Actualizar marcador temporal
        set(h_punto, 'xdata', t(i), 'ydata', x(i))

        drawnow()
        sleep(15)
    end
endfunction

simulacion_masa_resorte()
```

La simulación interactiva de fenómenos físicos cierra el ciclo entre la física, las matemáticas y la computación: las leyes físicas se expresan como ecuaciones diferenciales, los métodos numéricos las resuelven y la interfaz gráfica hace visible el comportamiento del sistema de una forma que las tablas de números nunca podrían lograr.

Scilab ofrece una segunda interfaz gráfica de naturaleza distinta. En Xcos el usuario no escribe el modelo sino que lo arma conectando bloques, de manera que la interfaz no sirve para mostrar el resultado sino para construir el sistema Scilab Enterprises 2026d. Es el mismo salto conceptual de la manipulación directa, aplicado esta vez al modelo y no a los datos Campbell, Chancelier y Nikoukhah 2006.

7.2.12. Gráficos directos en Scilab

Para las funciones y objetos gráficos de Scilab se utilizan la ayuda oficial y materiales docentes de referencia Scilab Enterprises 2026c; Mora Escobar 2001; Peer 2 Peer University 2014; Nakamura 1997.

Scilab dispone de un sistema gráfico integrado. Las funciones de alto nivel crean objetos gráficos y devuelven o permiten obtener sus identificadores. La figura actual se obtiene con `gcf()` y los ejes actuales con `gca()`. Esta jerarquía permite modificar propiedades después de haber creado el gráfico Scilab Enterprises 2026c.

Curvas 2D y objetos

El patrón básico es:

```
x = linspace(0, 2*%pi, 200)
```

```
y = sin(x)

clf()
h = plot(x, y, "b-")
h.thickness = 2
xlabel("x [rad]")
ylabel("sin(x)")
title("Funcion seno")
xgrid()
```

Para varias curvas pueden suministrarse varios pares de datos o realizar varias llamadas sobre los mismos ejes. Las propiedades globales y la especificación de línea controlan color, grosor, patrón y marcador.

Una ventaja del sistema retenido es que el objeto devuelto puede modificarse:

```
h = plot(x, y)
h.foreground = color("red")
h.thickness = 3
```

Las propiedades exactas dependen del tipo de entidad. La estrategia general es obtener el handle, inspeccionar sus propiedades y modificar sólo las necesarias.

Escalas y polares

Scilab proporciona funciones específicas para ejes logarítmicos y gráficos polares. Por ejemplo:

```
x = logspace(-2, 2, 200)

clf()
loglog(x, x.^2)
xlabel("x")
ylabel("x^2")
xgrid()

clf()
theta = linspace(-2*%pi, 2*%pi, 500)
r = sin(theta).^2
polarplot(theta, r)
```

También es posible convertir manualmente (r, θ) a (x, y) y utilizar `plot`; esa vía resulta útil para mantener exactamente el mismo cálculo en los cinco lenguajes.

Datos experimentales e histogramas

Las barras de error verticales pueden añadirse con `errbar`. Si el error es simétrico se utilizan los mismos valores hacia abajo y hacia arriba:

```
x = [0 0.5 1.0 1.5]
```

```
y = [0.03 0.46 0.85 0.99]
e = [0.05 0.06 0.05 0.04]

clf()
plot(x, y, "ro")
errbar(x, y, e, e)
```

Para histogramas se dispone de `histplot`. De forma predeterminada `histplot` normaliza el histograma; cuando se quieren mostrar conteos se indica `normalization=%f`. La media, desviación y otras magnitudes se calculan independientemente del dibujo.

```
datos = grand(1, 1000, "nor", 0, 1)
clf()
histplot(20, datos, normalization=%f)
xlabel("valor")
ylabel("conteo")
```

Superficies, contornos y campos

Una grilla puede construirse y evaluarse en memoria:

```
x = linspace(-2, 2, 60)
y = linspace(-2, 2, 60)
[X, Y] = meshgrid(x, y)
Z = X .* exp(-(X.^2 + Y.^2))

clf()
surf(X, Y, Z)
xlabel("x")
ylabel("y")
zlabel("z")
colorbar()
```

Para una vista 2D pueden utilizarse mapas de color y contornos. Los campos de vectores se representan con funciones como `champ`. Debe mantenerse consistente la orientación de las matrices X, Y y Z: `surf` acepta directamente las matrices producidas por `meshgrid`, mientras que las funciones que reciben los vectores de grilla x e y, como `contourf` y `champ`, requieren trasponer los datos calculados sobre esa grilla.

```
clf()
contourf(x, y, Z', 15)
colorbar(min(Z), max(Z))

// Ejemplo de gradiente en la misma grilla (Scilab >= 2026.0)
dx = x(2) - x(1)
dy = y(2) - y(1)
[dZdx, dZdy] = gradient(Z, dx, dy)

// champ espera fx(i,j) en (x(i),y(j)); con meshgrid hay que
   trasponer
```

```
champ(x, y, dZdx', dZdy')
```

La función `gradient` está disponible desde Scilab 2026.0. En versiones anteriores el gradiente puede calcularse mediante diferencias finitas o a partir de las derivadas conocidas de la función. Cuando la interpretación de la grilla o del gradiente sea crítica, conviene comprobar dimensiones y orientación; en particular, los campos construidos a partir de `meshgrid` deben trasponerse al pasarlos a `champ`.

Paneles y exportación

`subplot` divide una figura en paneles. La exportación puede realizarse en formatos raster y vectoriales:

```
clf()
subplot(2,1,1)
plot(x, sin(x))
title("seno")

subplot(2,1,2)
plot(x, cos(x))
title("coseno")

xs2png(gcf(), "paneles.png")
xs2pdf(gcf(), "paneles.pdf")
```

Animación directa

Una animación puede conservar el objeto y modificar sus datos:

```
t = linspace(0, 2*%pi, 150)

clf()
h = plot(cos(t(1)), sin(t(1)), "ro")
a = gca()
a.data_bounds = [-1.2 -1.2; 1.2 1.2]
a.isoview = "on"

for k = 2:length(t)
    h.data = [cos(t(k)) sin(t(k))]
    drawnow()
end
```

La propiedad concreta que se actualiza depende del objeto creado. El principio es conservar el handle y cambiar su estado, en lugar de reconstruir toda la figura en cada iteración.

Complemento: Creación de gráficos

La visualización gráfica es una herramienta fundamental en ciencia e ingeniería. Un gráfico bien construido comunica en segundos lo que una tabla de números tardaría minutos en transmitir, revela patrones que los estadísticos no capturan y permite detectar errores en los datos o en los modelos.

El ejemplo clásico de eso último es el cuarteto de Anscombe: cuatro conjuntos de datos con la misma media, la misma varianza, la misma correlación y la misma recta de regresión, que graficados resultan completamente distintos —uno lineal, uno curvo, uno con un punto atípico que arrastra la recta y uno donde toda la pendiente la determina un solo dato. Ningún resumen numérico los distingue; el gráfico los distingue de inmediato Wilke 2019.

Los entornos de cálculo científico como Scilab proporcionan funciones de alto nivel para crear gráficos: el programador especifica los datos y los parámetros visuales, y el sistema se encarga de los detalles de renderizado.

Conviene entender desde el principio qué hay detrás de esa comodidad. Scilab organiza lo dibujado en una jerarquía: la *figura* es la ventana, dentro de ella hay uno o más *ejes* con su rango, etiquetas y escala, y dentro de los ejes están las entidades dibujadas —curvas, superficies, textos—, cada una con sus propiedades. Las funciones `gcf()` y `gca()` devuelven la figura y los ejes actuales, y son la puerta de entrada a todo lo que no se puede pedir con un argumento de `plot`. Toda la referencia de funciones y propiedades está en la ayuda del sistema Scilab Enterprises 2026c, y hay introducciones al lenguaje que desarrollan la parte gráfica con ejemplos Mora Escobar 2001; Peer 2 Peer University 2014; Nakamura 1997.

El flujo básico para crear un gráfico es:

```
// 1. Preparar los datos
x = linspace(0, 2*%pi, 200)
y = sin(x)

// 2. Crear la figura
clf()           // limpiar figura anterior

// 3. Trazar los datos
plot(x, y)

// 4. Agregar elementos descriptivos
xlabel('Angulo [rad]')
ylabel('Amplitud')
title('Funcion seno')
xgrid()
```

La función `clf()` limpia la figura actual antes de trazar. Si no se llama, los nuevos trazados se superponen a los anteriores, lo que a veces es deseable (`hold on` en otros entornos) y a veces no.

Conviene distinguirla de `scf(n)`, que selecciona o crea la figura número `n`. Con `clf()` se reutiliza la misma ventana borrando lo anterior; con `scf()` se trabaja en varias ventanas simultáneas. Un guión que genera diez figuras y las exporta debe decidir explícitamente

cuál de las dos usa: si no, o todo se apila en un solo gráfico ilegible, o quedan diez ventanas abiertas.

Complemento: Representación en 2D

Gráfico de líneas Es el tipo más común: conecta los puntos de datos con segmentos de recta. Adecuado para representar funciones continuas, series temporales y resultados de simulaciones.

```
x = linspace(0, 4*%pi, 500)

// Una sola curva
plot(x, sin(x))

// Varias curvas en el mismo grafico
clf()
plot(x, sin(x))
plot(x, cos(x))
plot(x, sin(x).*cos(x))
legend('sen(x)', 'cos(x)', 'sen(x)cos(x)')
xlabel('x [rad]')
ylabel('f(x)')
title('Funciones trigonometricas')
xgrid()
```

Que la línea sea una sucesión de segmentos y no una curva verdadera tiene una consecuencia práctica: la suavidad depende de cuántos puntos se hayan calculado. Con `linspace(a,b,20)` una sinusoide se ve como una poligonal, y peor aún, un máximo puede caer entre dos puntos y no aparecer nunca. Si la curva se ve quebrada donde debería ser suave, hay que aumentar la cantidad de puntos del vector, no cambiar la función.

Gráfico de puntos Se usan cuando los datos son discretos y no tiene sentido conectarlos con una línea; por ejemplo, datos experimentales individuales.

```
// Datos experimentales simulados
x_exp = [0.5, 1.2, 2.1, 3.0, 3.8, 4.5, 5.2, 6.0]
y_exp = sin(x_exp) + 0.1*(rand(1,8) - 0.5)

// Curva teorica
x_teo = linspace(0, 2*%pi, 200)
y_teo = sin(x_teo)

clf()
plot(x_teo, y_teo, 'b-')   // linea azul: teoria
plot(x_exp, y_exp, 'ro')   // puntos rojos: experimento
legend('Teoria', 'Experimento')
xlabel('x')
ylabel('y')
title('Comparacion teoria vs experimento')
```

La elección entre puntos y línea no es estética: comunica el estatus de lo representado. Los puntos son mediciones, con existencia individual; la línea es un modelo, continuo. Unir con segmentos los puntos experimentales afirma que entre dos mediciones el sistema hizo exactamente eso, que es más de lo que se midió. Por eso la convención en física es puntos para el dato y línea para la teoría, y las dos cosas en el mismo par de ejes cuando se las quiere comparar.

Gráfico de barras e histograma

```
// Grafico de barras
categorias = ['Lunes', 'Martes', 'Miercoles', ...
              'Jueves', 'Viernes']
valores = [42, 38, 55, 61, 47]

clf()
bar(valores)
set(gca(), 'x_ticks', struct('locations', 1:5, ...
    'labels', categorias))
ylabel('Cantidad')
title('Ventas por dia')

// Histograma de una distribucion normal
clf()
datos = grand(1, 1000, 'nor', 0, 1)
histplot(20, datos)
xlabel('Valor')
ylabel('Frecuencia relativa')
title('Histograma - distribucion normal (n=1000)')
```

Un histograma no muestra los datos sino un resumen de los datos, y ese resumen depende de una decisión de quien grafica: el ancho de clase. Con clases muy angostas domina la fluctuación estadística y la distribución se ve como un peine irregular; con clases muy anchas se pierde la estructura y dos picos pueden aparecer como uno solo.

No hay un valor universalmente correcto, pero sí criterios. La regla de Freedman-Diaconis fija el ancho a partir del rango intercuartil y del número de datos, $\Delta x = 2\,\mathrm{IQR}\,N^{-1/3}$, y es robusta frente a valores atípicos porque no usa la desviación estándar Freedman y Diaconis 1981. La conclusión práctica es que el número de clases se elige y se justifica, y que conviene mirar los mismos datos con dos o tres anchos antes de afirmar nada sobre su forma.

Hay además una diferencia importante entre el histograma y el gráfico de barras de categorías: en el primero el eje horizontal es continuo y el ancho de la barra significa algo; en el segundo las barras son etiquetas sin orden ni distancia. En el gráfico de barras, además, la longitud codifica el valor, y por eso el eje vertical debe arrancar en cero: recortarlo multiplica visualmente las diferencias Tufte 2001.

Gráfico logarítmico Indispensable cuando los datos abarcan varios órdenes de magnitud o cuando se quiere visualizar leyes de potencia.

```
x = logspace(-2, 3, 200)  // de 0.01 a 1000, escala log
```

```
clf()

// Escala log en ambos ejes
subplot(1,3,1)
loglog(x, x.^2)
title('log-log: y = x^2')
xgrid()

// Escala log solo en x
subplot(1,3,2)
semilogx(x, log10(x))
title('semilog-x')
xgrid()

// Escala log solo en y
subplot(1,3,3)
semilogy(x, exp(x/100))
title('semilog-y')
xgrid()
```

Además de resolver el problema de rango, la escala logarítmica convierte ciertas relaciones en rectas, y eso permite reconocerlas de un vistazo:

Semilogarítmica en y: una exponencial $y = Ae^{kx}$ se ve como recta, porque $\ln y = \ln A + kx$. La pendiente da la constante de decaimiento o crecimiento.

Logarítmica en ambos ejes: una ley de potencias $y = Ax^n$ se ve como recta, porque $\log y = \log A + n \log x$. La pendiente da directamente el exponente.

Por eso en física la gráfica log-log suele ser el primer paso del análisis y no la forma de presentar el resultado: sirve para averiguar qué relación hay. Dos advertencias: el logaritmo no está definido para valores nulos o negativos, y las barras de error se vuelven asimétricas al transformarlas.

Gráfico de dispersión con colores

```
// 500 puntos aleatorios con color segun valor
n = 500
x = randn(1, n)
y = randn(1, n)
z = x.^2 + y.^2   // distancia al origen

clf()
scatter(x, y, 30, z)  // 30 = tamaño del marcador
colorbar()
xlabel('x')
ylabel('y')
title('Distancia al origen')
```

Codificar una tercera variable con el color de los puntos es cómodo y tiene un costo que conviene conocer. Cuando se midió con qué error las personas leen distintas codificaciones

gráficas de una misma cantidad, resultó un orden bastante estable: la posición sobre una escala común se lee mejor que la longitud, la longitud mejor que el ángulo o la pendiente, y todas ellas mucho mejor que el área, el volumen o la saturación de color Cleveland y McGill 1984.

De ahí la regla: el color y el tamaño sirven para mostrar la existencia de un patrón, no para leer valores. Si el lector tiene que comparar cuantitativamente la tercera variable, conviene un segundo panel con esa variable en el eje vertical.

Complemento: Parámetros del gráfico

Un gráfico científico debe ser completo y autónomo: alguien que lo vea sin leer el texto que lo rodea debe poder entender qué representa.

Colores y estilos de línea

```
x = linspace(0, 2*%pi, 100)

clf()
// Colores: 'r'=rojo, 'g'=verde, 'b'=azul, 'k'=negro
// 'm'=magenta, 'c'=cian, 'y'=amarillo
// Estilos: '-'=solida, '--'=guiones, ':'=puntos,
    '-.'=guion-punto
// Marcadores: 'o'=circulo, '+'=cruz, 'x'=equis,
//              's'=cuadrado, 'd'=diamante, '^'=triangulo

plot(x, sin(x),       'b-',  'LineWidth', 2)
plot(x, cos(x),       'r--', 'LineWidth', 2)
plot(x, sin(2*x)/2,   'g:',  'LineWidth', 1.5)
plot(x, cos(2*x)/2,   'k-.', 'LineWidth', 1.5)

legend('sin(x)', 'cos(x)', 'sin(2x)/2', 'cos(2x)/2', ...
       'location', 'northeast')
```

El color es donde más fácil se falla, porque en la pantalla de quien hace el gráfico todo se distingue. Hay tres exigencias simultáneas: que los colores se diferencien entre sí, que sigan diferenciándose impresos en blanco y negro, y que se diferencien para quien tiene visión deficiente del color, que es alrededor del ocho por ciento de los varones. La combinación que falla siempre es rojo y verde, justamente la más usada por omisión.

Hay paletas construidas y verificadas para los tres requisitos a la vez, clasificadas según el tipo de dato: secuenciales para magnitudes ordenadas, divergentes para desviaciones respecto de un valor central y cualitativas para categorías sin orden Harrower y Brewer 2003. Para curvas superpuestas, la paleta de ocho colores de Okabe e Ito es una elección segura y se traduce directamente a los tripletes RGB que acepta Scilab Okabe e Ito 2008.

La regla práctica que resume todo: el color nunca debe ser el único canal que distingue dos curvas. Que difieran además en el tipo de línea o en el marcador, y el gráfico sobrevive a la fotocopia.

Ejes y escalas

```
clf()
```

```
x = linspace(-5, 5, 200)
plot(x, x.^3 - 3*x)

// Controlar el rango de los ejes
a = gca()            // get current axes
a.x_location = 'origin'   // eje x en y=0
a.y_location = 'origin'   // eje y en x=0
a.data_bounds = [-5, -15; 5, 15]  // [xmin,ymin; xmax,ymax]

xlabel('x', 'fontsize', 12)
ylabel('f(x)', 'fontsize', 12)
title('f(x) = x^3 - 3x', 'fontsize', 14)
xgrid()
```

El rango de los ejes es una decisión con contenido, no un detalle de presentación. Recortar el eje vertical es legítimo en un gráfico de líneas, donde lo que codifica el valor es la posición, y engañoso en uno de barras, donde lo codifica la longitud. Y dejar demasiado espacio vacío desperdicia la resolución del gráfico: si los datos varían entre 20 y 22 grados, forzar el eje de 0 a 100 aplasta toda la información en una franja.

La relación de aspecto tiene el mismo carácter. Cuando los dos ejes comparten unidades —una trayectoria en el plano, un diagrama de fases— alterarla deforma la geometría y hay que fijarla con `isoview`. Cuando las unidades son distintas, en cambio, es libre y conviene elegirla de modo que las variaciones de interés queden a unos cuarenta y cinco grados, que es donde mejor se aprecian los cambios de pendiente Cleveland 1994.

Leyendas y anotaciones

```
x = linspace(0, 2*%pi, 200)

clf()
plot(x, sin(x), 'b-', 'LineWidth', 2)

// Leyenda
legend('$\sin(x)$', 'fontsize', 12)

// Anotacion en un punto especifico
xm = %pi/2
ym = 1
xarrow([xm-0.5, xm], [ym+0.3, ym], 'arrowsize', 0.3)
xstring(xm-0.9, ym+0.35, 'Maximo')

// Texto en el grafico
title('Funcion seno con anotacion', 'fontsize', 14)
xlabel('x [rad]', 'fontsize', 12)
ylabel('Amplitud', 'fontsize', 12)
```

Cada elemento agregado al gráfico debe estar mostrando algo que el lector no vería sin él. Fondos con degradé, marcos dobles, grillas gruesas, efectos tridimensionales sobre datos bidimensionales y leyendas recuadradas compiten con los datos por la atención sin aportar información Tufte 2001. Marcar las dos raíces de una parábola es informativo; rotular los

veinte puntos de una serie convierte el gráfico en una tabla mal impresa.

Cuando hay pocas curvas, una alternativa mejor que la leyenda es rotular cada curva directamente al lado, con `xstring`: evita el ida y vuelta entre la curva y el recuadro de referencia Wilke 2019.

Subfiguras

```
x = linspace(0, 4*%pi, 300)

clf()

subplot(2, 2, 1)     // 2 filas, 2 columnas, posicion 1
plot(x, sin(x))
title('sin(x)')
xgrid()

subplot(2, 2, 2)     // posicion 2
plot(x, cos(x))
title('cos(x)')
xgrid()

subplot(2, 2, 3)     // posicion 3
plot(x, sin(x).^2)
title('sin^2(x)')
xgrid()

subplot(2, 2, 4)     // posicion 4
plot(x, sin(x).*cos(x))
title('sin(x)cos(x)')
xgrid()
```

Poner varias magnitudes en paneles separados que comparten el eje horizontal suele comunicar mejor que superponerlas: cada panel conserva la lectura por posición sobre una escala común, que es la codificación más precisa, y se evitan curvas de magnitudes incomparables forzadas a un mismo rango Cleveland y McGill 1984.

La regla al usar paneles es que los ejes compartidos tengan el mismo rango. Si cada panel se autoescala, la comparación visual entre paneles engaña aunque cada uno por separado sea correcto.

Complemento: Representación en 3D

La visualización en tres dimensiones permite explorar funciones de dos variables, superficies y trayectorias en el espacio.

Superficie 3D Para trazar $z = f(x, y)$ se necesita una grilla de puntos (x, y) y los valores z correspondientes.

```
// Crear la grilla
x = linspace(-3, 3, 60)
```

```
y = linspace(-3, 3, 60)
[X, Y] = meshgrid(x, y)

// Calcular z en cada punto de la grilla
Z = sin(sqrt(X.^2 + Y.^2)) ./ (sqrt(X.^2 + Y.^2) + 0.01)

clf()
surf(X, Y, Z)
xlabel('x')
ylabel('y')
zlabel('z')
title('Funcion sinc 2D: sin(r)/r')
colorbar()
```

```
// Superficie con mapa de colores personalizado
x = linspace(-2, 2, 50)
y = linspace(-2, 2, 50)
[X, Y] = meshgrid(x, y)
Z = X.*exp(-(X.^2 + Y.^2))

clf()
surf(X, Y, Z)
title('f(x,y) = x*exp(-(x^2+y^2))')
xlabel('x')
ylabel('y')
zlabel('f(x,y)')

// Controlar el punto de vista: elevacion y azimut
a3d = gca()
a3d.rotation_angles = [65, 45]  // [elevacion, azimut]
```

La superficie en perspectiva es atractiva y suele ser la peor forma de leer valores. La profundidad introduce oclusión —una parte de la superficie tapa a la otra—, los puntos lejanos se dibujan más chicos, y la posición, que era la codificación más precisa, deja de ser legible Cleveland y McGill 1984. Sirve para mostrar la forma cualitativa de una función; si hay que leer valores o comparar regiones, conviene el mapa de color con curvas de nivel de la sección siguiente.

Lo que Scilab dibuja no es la superficie sino una malla de cuadriláteros construida sobre la grilla de evaluación, coloreada según el valor. El paso de la grilla es entonces un parámetro con consecuencias: demasiado grueso pierde detalle y puede saltearse un máximo, demasiado fino produce una malla tan densa que el dibujo se vuelve negro de tanto borde.

Malla de alambre

```
// Malla sin relleno de color
clf()
mesh(X, Y, Z)
title('Malla de alambre')
xlabel('x')
ylabel('y')
zlabel('f(x,y)')
```

Sin ocultamiento de líneas, la malla de alambre muestra a la vez la cara de adelante y la de atrás, y la figura se vuelve ambigua: el mismo dibujo admite dos interpretaciones de qué está más cerca. El ocultamiento de la parte no visible es lo que resuelve esa ambigüedad y es una de las operaciones básicas del dibujo de escenas tridimensionales Foley et al. 1990.

Curva en 3D

```
// Helice en el espacio
t = linspace(0, 6*%pi, 500)
x = cos(t)
y = sin(t)
z = t / (2*%pi)

clf()
param3d(x, y, z)
xlabel('x')
ylabel('y')
zlabel('z = altura')
title('Helice circular')
a3d = gca()
a3d.rotation_angles = [60, 45]
```

Complemento: Curvas de nivel

Las curvas de nivel (isolíneas o contornos) representan una función $z = f(x, y)$ proyectando sobre el plano xy las curvas donde f toma valores constantes. Son especialmente útiles en geografía (curvas de nivel de un mapa topográfico), meteorología (isobaras) y optimización (curvas de costo constante).

La lectura tiene reglas propias, y conviene enunciarlas porque no son obvias. Donde las curvas se juntan la función varía rápido: el espaciado codifica el módulo del gradiente. Las curvas de nivel nunca se cruzan, porque un punto no puede tener dos valores. Las curvas cerradas encierran un máximo o un mínimo local, que se distinguen por los valores de las curvas vecinas. Y una curva de nivel es siempre perpendicular al gradiente, que es la dirección de máxima variación: por eso el descenso por gradiente en optimización avanza cortando las curvas de nivel en ángulo recto.

El caso particular del nivel cero es más útil de lo que parece: la curva $f(x, y) = 0$ es el lugar geométrico de las soluciones de esa ecuación, y dibujar el nivel cero de dos funciones sobre los mismos ejes muestra las soluciones del sistema en los puntos donde las curvas se cortan.

```
x = linspace(-3, 3, 100)
y = linspace(-3, 3, 100)
[X, Y] = meshgrid(x, y)
Z = X.^2/4 + Y.^2    // elipse

clf()

// Curvas de nivel automaticas
contour(X, Y, Z, 10)   // 10 niveles automaticos
```

```
xlabel('x')
ylabel('y')
title('Curvas de nivel: x^2/4 + y^2 = c')
xgrid()
colorbar()
```

```
// Niveles especificados manualmente
clf()
niveles = [0.25, 0.5, 1, 2, 3, 4, 6]
contour(X, Y, Z, niveles)
clabel()    // etiquetas con los valores de cada curva
xlabel('x')
ylabel('y')
title('Curvas de nivel con etiquetas')
```

```
// Superficie y curvas de nivel combinadas
clf()

subplot(1, 2, 1)
surf(X, Y, Z)
title('Vista 3D')
xlabel('x')
ylabel('y')
zlabel('z')
colorbar()

subplot(1, 2, 2)
contourf(X, Y, Z, 15)   // relleno de color entre curvas
colorbar()
contour(X, Y, Z, 15, 'k')  // lineas negras encima
title('Vista de planta con relleno')
xlabel('x')
ylabel('y')
```

```
// Campo vectorial superpuesto a curvas de nivel
// (gradiente de la funcion)
x = linspace(-2, 2, 20)
y = linspace(-2, 2, 20)
[X, Y] = meshgrid(x, y)

// Funcion y su gradiente
F = X.^2 + 2*Y.^2
dFdx = 2*X      // derivada parcial en x
dFdy = 4*Y      // derivada parcial en y

// Version densa para las curvas de nivel
xd = linspace(-2, 2, 100)
yd = linspace(-2, 2, 100)
[Xd, Yd] = meshgrid(xd, yd)
Fd = Xd.^2 + 2*Yd.^2
```

```
clf()
contour(Xd, Yd, Fd, 8)
// El gradiente es perpendicular a las curvas de nivel
champ(X(1,:), Y(:,1)', dFdx, dFdy)
title('Curvas de nivel y gradiente de x^2 + 2y^2')
xlabel('x')
ylabel('y')
```

La elección del mapa de color La combinación que mejor funciona en la práctica es el mapa de color con curvas de nivel superpuestas: el color da la impresión global de dónde están las regiones altas y bajas, y las curvas devuelven la lectura cuantitativa. Pero la paleta elegida no es neutral, y ésta es una de las decisiones donde el uso corriente está mal.

La paleta arcoíris —`jet` en otros entornos, variantes de `hotcolormap` acá— es la más difundida y tiene dos defectos serios. El primero es que no es uniforme en percepción: en algunas zonas del recorrido un cambio pequeño del valor produce un cambio grande de color, y en otras al revés, de modo que la paleta *inventa* fronteras que no están en los datos y *oculta* variaciones que sí están. El segundo es que el orden de los colores del arcoíris no es un orden perceptual: nadie sabe sin mirar la escala si el verde es mayor o menor que el naranja Borland y R. M. Taylor 2007.

La alternativa depende del dato. Para una magnitud que sólo crece, corresponde una paleta secuencial, que varía monótonamente en luminosidad y por eso se lee incluso en escala de grises. Para una magnitud con un punto medio significativo —una diferencia respecto de un valor de referencia, un potencial que cambia de signo— corresponde una divergente, construida con dos ramas simétricas que se encuentran en un color neutro Moreland 2009. Hay familias de paletas ya construidas, uniformes en percepción y verificadas para daltonismo y escala de grises, que se pueden cargar en Scilab con `colormap()` pasándole la matriz de tripletes RGB Crameri, Shephard y Heron 2020. El criterio de clasificación es el mismo que ya vimos para el color de las líneas Harrower y Brewer 2003.

Y una regla que vale siempre: un mapa de color sin barra de escala no significa nada. La barra es la que convierte el color en magnitud.

Complemento: Exportación de gráficos

Un gráfico generado en Scilab puede exportarse en distintos formatos según el uso previsto: para incluir en un documento LaTeX, para publicar en una página web, para imprimir en alta resolución.

```
// Primero crear el grafico
clf()
x = linspace(0, 2*%pi, 200)
plot(x, sin(x), 'b-', 'LineWidth', 2)
xlabel('x [rad]')
ylabel('sin(x)')
title('Funcion seno')
```

```
xgrid()

// Exportar en distintos formatos
xs2png(gcf(), 'seno.png')      // PNG: para web e informes
xs2svg(gcf(), 'seno.svg')      // SVG: vectorial para web
xs2pdf(gcf(), 'seno.pdf')      // PDF: para LaTeX
xs2eps(gcf(), 'seno.eps')      // EPS: PostScript encapsulado
```

```
// Controlar el tamaño y la resolucion al exportar
// Crear figura de tamaño especifico
f = figure('Figure_name', 'Mi grafico', ...
           'BackgroundColor', [1 1 1])  // fondo blanco

// Ajustar tamaño en pixeles antes de exportar
f.axes_size = [800, 600]    // ancho x alto en pixeles

// El grafico a esta escala tendrá buena resolución
// para un documento de pagina A4
xs2png(f, 'figura_publicacion.png')
```

Elección del formato según el uso

```
// En Scilab, el formato se elige segun el destino:

// Para incluir en LaTeX (maximo calidad):
xs2pdf(gcf(), 'figura.pdf')
// En el .tex: \includegraphics{figura.pdf}

// Para incluir en una pagina web:
xs2svg(gcf(), 'figura.svg')    // vectorial, preferido
xs2png(gcf(), 'figura.png')    // rasterizado, alternativa

// Para un informe en Word o presentacion:
xs2png(gcf(), 'figura.png')    // PNG de alta resolucion

// Para enviar por correo o redes sociales:
xs2png(gcf(), 'figura.png')
// luego convertir a JPEG si el tamaño importa:
// convert -quality 90 figura.png figura.jpg
```

La diferencia de fondo entre las dos familias es qué se guarda. Un formato rasterizado guarda el resultado de dibujar: una cuadrícula de píxeles. Un formato vectorial guarda las instrucciones para dibujar: líneas, curvas y texto como objetos. De ahí se derivan todas las demás diferencias —la dependencia de la resolución, el comportamiento al ampliar, el tamaño del archivo, y si el texto del gráfico sigue siendo texto buscable o se convirtió en manchas de píxeles.

Para un gráfico de Scilab el formato vectorial es casi siempre la elección correcta, porque el gráfico ya es un conjunto de líneas y texto, que es exactamente lo que un formato vectorial representa bien. El caso en que conviene el raster es el opuesto: cuando la figura contiene una superficie con mapa de color de muchísimos puntos, y guardar cada cuadrilátero como

objeto produce un archivo enorme y lento de abrir.

Dentro del raster, la distinción decisiva es entre compresión sin pérdida, que reconstruye exactamente los píxeles originales International Organization for Standardization 2004, y con pérdida, que descarta información que se supone poco perceptible y no se recupera nunca. Esta última viene de cuantizar los coeficientes de la transformada coseno discreta en bloques de ocho por ocho píxeles, que es también el origen de los artefactos cuadrados visibles al comprimir mucho International Telecommunication Union 1992. Para un gráfico con líneas finas y texto es una mala elección: produce halos alrededor de los trazos y ensucia las letras.

Del lado vectorial, SVG es XML y se puede abrir con un editor de texto, estilar con CSS y animar desde el navegador World Wide Web Consortium 2026; PDF es el formato que `\includegraphics` incorpora sin pérdida de calidad tipográfica International Organization for Standardization 2020. Ambos son normas públicas, lo cual tiene una consecuencia que excede lo técnico y conecta con la primera unidad: una figura guardada en un formato especificado públicamente se podrá abrir dentro de veinte años con cualquier programa.

La cuenta que conviene saber hacer es la que relaciona píxeles, pulgadas y DPI: una figura de 12 cm de ancho impresa a 300 DPI necesita $12/2{,}54 \times 300 \approx 1417$ píxeles de ancho. Por eso el `axes_size` del ejemplo anterior importa: exportar chico y después ampliar produce letras borrosas, y exportar grande y después achicar con `\includegraphics` las deja ilegibles. Conviene exportar ya en el tamaño final.

Para todo lo que viene después de generar la figura —convertir entre formatos, cambiar resolución, componer varias imágenes en una grilla, armar un GIF animado a partir de una secuencia de cuadros, o averiguar con `identify` qué tiene realmente un archivo— la herramienta es ImageMagick ImageMagick Studio LLC 2026a. Al convertir conviene recordar la dirección de la pérdida: de vectorial a raster se puede siempre, eligiendo la resolución; de raster a vectorial no se recupera lo que no está.

Formato	Tipo	Compresion	LaTeX	Web
PDF	vectorial	sin perdida	excelente	no directo
SVG	vectorial	sin perdida	con paquete	excelente
EPS	vectorial	sin perdida	clásico	no
PNG	raster	sin perdida	bueno	bueno
JPEG	raster	con perdida	aceptable	bueno
TIFF	raster	sin perdida	posible	no

Complemento: Buenas prácticas de visualización en gráficos nativos de Scilab

Como ya dijimos, un gráfico científico debe cumplir ciertos requisitos para ser válido como comunicación técnica:

```
// Ejemplo de grafico con todas las buenas practicas

// Datos
t = linspace(0, 10, 200)           // tiempo [s]
y_teo = exp(-0.3*t).*cos(2*t)      // teoria
t_exp = 0:1:10                      // tiempos de medicion
y_exp = exp(-0.3*t_exp).*cos(2*t_exp) + ...
```

```
          0.05*(rand(1,11) - 0.5)    // mediciones con ruido

clf()

// Curva teorica
plot(t, y_teo, 'b-', 'LineWidth', 2)

// Datos experimentales con barras de error
// (error estimado del instrumento: 0.03)
error = 0.03 * ones(1, length(t_exp))
// ploterr no existe en Scilab pero se puede simular:
for i = 1:length(t_exp)
    plot([t_exp(i), t_exp(i)], ...
         [y_exp(i)-error(i), y_exp(i)+error(i)], 'r-')
end
plot(t_exp, y_exp, 'ro', 'MarkerSize', 6)

// Elementos obligatorios en todo grafico cientifico:
// 1. Etiquetas de ejes CON UNIDADES
xlabel('Tiempo t [s]', 'fontsize', 12)
ylabel('Amplitud x(t) [m]', 'fontsize', 12)

// 2. Titulo descriptivo
title('Oscilador amortiguado: teoria vs experimento', ...
      'fontsize', 14)

// 3. Leyenda
legend('Teoria: e^{-0.3t}cos(2t)', ...
       'Mediciones (+/-0.03 m)', ...
       'location', 'northeast', 'fontsize', 10)

// 4. Grilla para facilitar la lectura
xgrid()

// 5. Rango de ejes apropiado (no recortado ni con aire excesivo)
a = gca()
a.data_bounds = [0, -1.1; 10, 1.1]

// 6. Exportar en formato adecuado
xs2pdf(gcf(), 'oscilador.pdf')     // para LaTeX
xs2png(gcf(), 'oscilador.png')     // para otros usos
```

Vale recapitular el fundamento, porque estas reglas no son cuestión de gusto ni de convención académica, y conocerlo permite saber cuándo se pueden romper.

Etiquetas con unidades en los ejes. Un número sin unidad no es una magnitud física. Es la regla que no admite excepción.

La tinta debe ser mayormente datos. De todo lo que hay en el gráfico, la parte que representa datos debería ser la mayor posible; lo demás compite por la atención sin aportar Tufte 2001.

Codificar por posición cuando importa el valor. Área, volumen y saturación de color se leen con mucho más error Cleveland y McGill 1984.

Barras de error cuando hay incertidumbre. Un punto sin barra afirma implícitamente que el valor se conoce con exactitud, lo cual nunca es cierto en una medición.

Color que no sea el único canal distintivo , y paletas elegidas según el tipo de dato Harrower y Brewer 2003; Okabe e Ito 2008.

Resolución adecuada al uso , que es la cuenta de DPI de la sección anterior.

Guías con ejemplos comentados de gráficos buenos y malos, y el tratamiento sistemático de las decisiones de escala, aspecto y paneles, están en Wilke 2019 y Cleveland 1994.

Complemento: Comparaciones de las gráficas: Scilab (nativas) – Gnuplot

Aspecto	**Scilab nativo**	**Gnuplot**
Modo sin pantalla	No disponible en CLI	Sí, siempre
Calidad tipográfica	Buena	Excelente
Formatos de salida	PNG, PDF, SVG, EPS	PNG, PDF, SVG, EPS, GIF
Curvas de nivel	Sí (`contour`)	Sí (`pm3d map`)
Superficies 3D	Sí (`surf`)	Sí (`splot pm3d`)
Animaciones	Sí (`drawnow`)	Vía secuencia de imágenes
Integración	Directa	Vía archivos y unix()
Reproducibilidad	Media (interactivo)	Alta (script de texto)

La regla práctica: usar Scilab nativo para exploración interactiva durante el desarrollo, y Gnuplot para producción cuando se necesita automatización, modo sin pantalla (servidores, scripts) o gráficas de calidad para publicación.

Por un cuestión de tiempos solo veremos visualización con gnuplot. Pero existe un práctico alternativo con Scilab.

La tabla de equivalencias entre `GlobalProperty` de Scilab y los parámetros de Gnuplot:

Scilab GlobalProperty	**Gnuplot**
Color de línea	`lc 'color'`
Color de relleno marcador	`fc 'color'`
Color de borde marcador	`lc 'color'`
Grosor de línea	`lw N`
Tipo de línea continua	`dt 1`
Tipo de línea punteada	`dt 3`
Tipo de línea guiones	`dt 2`
Marcador circulo relleno	`pt 7`
Marcador cuadrado relleno	`pt 5`
Tamaño marcador	`ps N`
Linea + marcadores	`with linespoints`

La tabla de equivalencias de estilos entre Scilab y Gnuplot:

Scilab linespec	**Gnuplot equivalente**
`'r-'`	`with lines lc 'red'`
`'b-'`	`with lines lc 'blue'`
`'k-'`	`with lines lc 'black' dt 2`
`'r-o'`	`with linespoints lc 'red' pt 7`
`'b-s'`	`with linespoints lc 'blue' pt 5`
`'b-^'`	`with linespoints lc 'blue' pt 8`
`'r.'`	`with points lc 'red' pt 7 ps 0.5`

La tabla de equivalencias entre las opciones de `plot2d` y los comandos de Gnuplot:

Scilab plot2d	**Gnuplot equivalente**
`logflag="nn"`	(defecto, ejes lineales)
`logflag="nl"`	`set logscale y`
`logflag="ln"`	`set logscale x`
`logflag="ll"`	`set logscale xy`
`rect=[x1,y1,x2,y2]`	`set xrange [x1:x2]; set yrange [y1:y2]`
`leg=ç1@c2"`	`title 'c1'` en cada curva
`style=N` (color)	`lc 'color'`
`xset("line style",2)`	`dt 2` (guiones)
`xset("line style",3)`	`dt 3` (punteada)
`xset("line style",4)`	`dt 4` (guion-punto)

La tabla de equivalencias de marcadores (`pt`) y estilos de línea (`dt`) entre Scilab y Gnuplot:

Scilab	**Gnuplot pt**	**Descripción**
`'o'`	`pt 7`	circulo relleno
`'s'`	`pt 5`	cuadrado relleno
`'^'`	`pt 9`	triangulo arriba relleno
`'v'`	`pt 11`	triangulo abajo relleno
`'d'`	`pt 13`	diamante relleno
`'+'`	`pt 1`	cruz
`'x'`	`pt 2`	equis
Scilab	**Gnuplot dt**	**Descripción**
`'-'`	`dt 1`	línea continua
`'-'`	`dt 2`	guiones
`':'`	`dt 3`	punteada
`'-.'`	`dt 4`	guion-punto

La diferencia principal con Scilab es que en Gnuplot el color de relleno del marcador (`fc`) y el color del borde (`lc`) se especifican por separado, lo que permite exactamente

la combinación pedida: círculo rojo con borde negro (`lc 'black' fc 'red'`) y cuadrado amarillo con borde azul (`lc 'blue' fc 'yellow'`).

La tabla de equivalencias entre `plot3d`/`plot3d1` de Scilab y los comandos de Gnuplot:

Scilab	**Gnuplot**
`plot3d(x,y,Z)`	`splot 'datos' with lines`
`plot3d1(x,y,Z)`	`splot 'datos' with pm3d`
`colormap(hotcolormap(64))`	`set palette rgbformulae 33,13,10`
`set_view(elev,rot)`	`set view elev,rot`
`contour2d`	`set view map; set pm3d map`
curvas de nivel	`set contour base`

La tabla de equivalencias entre `GlobalProperty` de `surf` y los comandos de Gnuplot:

Scilab GlobalProperty	**Gnuplot**
Paso de grilla	paso en `x=a:paso:b`
`colormap(hotcolormap)`	`set palette rgbformulae 33,13,10`
`colormap(graycolormap)`	`set palette grey`
`colormap(bluered)`	`set palette defined (-1 'blue', 1 'red')`
Malla visible	`splot 'f' with pm3d, 'f' with lines`
Solo malla	`splot 'f' with lines`
Vista superior	`set view map; set pm3d map`
Ángulo de visión	`set view elev,azimut`
Lineas ocultas	`set hidden3d`

La tabla de equivalencias entre `plot3d`/`plot3d1` de Scilab y los comandos de Gnuplot:

Scilab	**Gnuplot**
`plot3d(x,y,Z)`	`splot 'datos' with lines`
`plot3d1(x,y,Z)`	`splot 'datos' with pm3d`
`colormap(hotcolormap)`	`set palette rgbformulae 33,13,10`
`colormap(graycolormap)`	`set palette grey`
`set_view(elev,rot)`	`set view elev,rot`
Lineas ocultas	`set hidden3d`
Curvas de nivel	`set contour base`
Vista superior	`set view map; set pm3d map`

7.2.13. Gráficos directos en GNU Octave

La sintaxis gráfica de GNU Octave empleada aquí sigue su documentación oficial Eaton et al. 2024.

GNU Octave proporciona una API gráfica de alto nivel compatible en gran parte con la tradición MATLAB. El mismo conjunto de funciones puede trabajar con diferentes *toolkits*

gráficos. Para la modalidad “Octave solamente” de esta unidad se entiende que se usa un backend propio, por ejemplo Qt/OpenGL si está disponible. Seleccionar el toolkit `gnuplot` sería utilizar Gnuplot indirectamente y por lo tanto pertenecería conceptualmente al otro camino.

Se puede consultar la disponibilidad mediante las funciones del sistema gráfico y seleccionar explícitamente el backend cuando sea necesario.

Curvas, estilos y handles

```
graphics_toolkit("qt");   % si esta disponible

x = linspace(0, 2*pi, 200);
y = sin(x);

clf;
h = plot(x, y, "b-", "linewidth", 2);
xlabel("x [rad]");
ylabel("sin(x)");
title("Funcion seno");
grid on;
```

Para agregar series se utiliza `hold on` o una llamada con varios pares. Los objetos gráficos forman una jerarquía figura–ejes–entidades y sus propiedades se consultan o modifican mediante `get` y `set`.

```
hold on;
plot(x, cos(x), "r--", "linewidth", 2);
legend("sin(x)", "cos(x)");
hold off;
```

Escalas, polares y relación de aspecto

```
x = logspace(-2, 2, 200);
loglog(x, x.^2);
grid on;

figure;
theta = linspace(0, 2*pi, 500);
r = sin(theta).^2;
polar(theta, r);
```

Para una trayectoria cartesiana que deba conservar geometría puede utilizarse `axis equal`. La transformación `pol2cart` permite trabajar explícitamente con coordenadas cartesianas cuando se quiere compartir el mismo algoritmo con otros lenguajes.

Barras de error e histogramas

Octave dispone de `errorbar` y variantes para escalas logarítmicas:

```
x = [0 0.5 1.0 1.5];
y = [0.03 0.46 0.85 0.99];
e = [0.05 0.06 0.05 0.04];

errorbar(x, y, e, "o");
grid on;
```

La función `hist` dibuja un histograma cuando se llama sin variables de salida, y devuelve conteos y centros cuando se solicitan resultados. Esa doble forma permite separar cálculo y representación.

```
datos = randn(1, 1000);
hist(datos, 20);
xlabel("valor");
ylabel("conteo");
```

Superficies, contornos y vectores

```
x = linspace(-2, 2, 60);
y = linspace(-2, 2, 60);
[X, Y] = meshgrid(x, y);
Z = X .* exp(-(X.^2 + Y.^2));

figure;
surf(X, Y, Z);
xlabel("x"); ylabel("y"); zlabel("z");
colorbar;

figure;
contourf(X, Y, Z, 15);
axis equal;
colorbar;
```

Los campos vectoriales se representan con `quiver` y las variantes 3D con `quiver3`. Las funciones `mesh`, `surf`, `contour`, `contourf` y otras devuelven handles que pueden modificarse posteriormente.

Paneles, exportación y animación

```
figure;
subplot(2,1,1);
plot(x, sin(x));
title("seno");

subplot(2,1,2);
```

```
plot(x, cos(x));
title("coseno");

print("paneles.png", "-dpng");
print("paneles.pdf", "-dpdf");
```

En una animación se guarda el handle y se modifican propiedades como `xdata` e `ydata`, llamando a `drawnow` para actualizar la figura.

```
t = linspace(0, 2*pi, 150);
h = plot(cos(t(1)), sin(t(1)), "ro");
axis([-1.2 1.2 -1.2 1.2], "equal");

for k = 2:numel(t)
  set(h, "xdata", cos(t(k)), "ydata", sin(t(k)));
  drawnow();
endfor
```

7.2.14. Gráficos directos en Python

Para cálculo vectorial y visualización en Python se utilizan NumPy y Matplotlib, con sus referencias y documentación Harris, Millman, Walt, Gommers, Virtanen, Cournapeau, Wieser et al. 2020; Hunter 2007b; The Matplotlib development team 2026; Python Software Foundation 2026a.

En Python se utilizarán NumPy para los arreglos numéricos y Matplotlib para la representación. Matplotlib posee una interfaz de estado mediante `matplotlib.pyplot` y una interfaz orientada a objetos basada en `Figure` y `Axes`. Para programas de alguna complejidad conviene la segunda Hunter 2007b; The Matplotlib development team 2026.

Figura y ejes

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y, label="sin(x)")
ax.set_xlabel("x [rad]")
ax.set_ylabel("sin(x)")
ax.set_title("Funcion seno")
ax.grid(True)
ax.legend()
fig.savefig("seno.pdf")
plt.show()
```

El objeto `fig` representa la figura completa y `ax` un sistema de ejes dentro de ella. Las curvas creadas son objetos que pueden conservarse y actualizarse.

Escalas, polares y discontinuidades

```
x = np.logspace(-2, 2, 200)
y = x**2

fig, ax = plt.subplots()
ax.loglog(x, y)
ax.grid(True)
```

Para coordenadas polares puede crearse un eje con proyección polar:

```
theta = np.linspace(-2*np.pi, 2*np.pi, 500)
r = np.sin(theta)**2

fig, ax = plt.subplots(subplot_kw={"projection": "polar"})
ax.plot(theta, r)
```

NumPy utiliza `np.nan` para valores ausentes. Matplotlib interrumpe normalmente una línea en esos puntos, lo que resulta útil para representar ramas separadas de una función.

Barras de error, histogramas y regresión

```
x = np.array([0.0, 0.5, 1.0, 1.5])
y = np.array([0.03, 0.46, 0.85, 0.99])
e = np.array([0.05, 0.06, 0.05, 0.04])

fig, ax = plt.subplots()
ax.errorbar(x, y, yerr=e, fmt="o", capsize=4,
            label="mediciones")
xx = np.linspace(0, 1.5, 200)
ax.plot(xx, np.sin(xx), label="modelo")
ax.legend()
```

Un histograma se genera con `hist`; la opción de densidad permite compararlo directamente con una densidad teórica.

```
rng = np.random.default_rng(12345)
datos = rng.normal(size=1000)

fig, ax = plt.subplots()
ax.hist(datos, bins=24, density=True)
xx = np.linspace(-4, 4, 400)
phi = np.exp(-xx**2/2) / np.sqrt(2*np.pi)
ax.plot(xx, phi)
```

La regresión puede calcularse con operaciones de NumPy o con bibliotecas estadísticas.

La gráfica debe recibir los parámetros ya estimados y, cuando el problema lo pida, los residuos calculados.

Superficies, contornos y campos

Matplotlib incorpora gráficos 3D mediante el módulo `mplot3d`, que se activa al crear ejes con proyección `3d`.

```
x = np.linspace(-2, 2, 60)
y = np.linspace(-2, 2, 60)
X, Y = np.meshgrid(x, y)
Z = X * np.exp(-(X**2 + Y**2))

fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
s = ax.plot_surface(X, Y, Z)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
```

Para mapas bidimensionales:

```
fig, ax = plt.subplots()
c = ax.contourf(X, Y, Z, levels=20)
ax.contour(X, Y, Z, levels=20)
ax.quiver(X[::6, ::6], Y[::6, ::6],
          (1-2*X**2)[::6, ::6]*np.exp(-(X**2+Y**2))[::6, ::6],
          (-2*X*Y)[::6, ::6]*np.exp(-(X**2+Y**2))[::6, ::6])
fig.colorbar(c, ax=ax)
ax.set_aspect("equal")
```

En un programa real conviene calcular las componentes del gradiente en variables separadas antes de llamar a `quiver`; el ejemplo compacto muestra solamente la relación entre la grilla y el campo.

Paneles y animación

```
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(t, x)
axs[0, 1].plot(t, v)
axs[1, 0].plot(x, v)
# cuarto panel: otra magnitud
fig.tight_layout()
```

Para animación Matplotlib ofrece `FuncAnimation` y también permite actualizar manualmente objetos. El patrón básico conserva el objeto:

```
from matplotlib.animation import FuncAnimation

fig, ax = plt.subplots()
```

```
ax.set_xlim(-1.2, 1.2)
ax.set_ylim(-1.2, 1.2)
ax.set_aspect("equal")
punto, = ax.plot([], [], "ro")

def actualizar(k):
    punto.set_data([px[k]], [py[k]])
    return punto,

anim = FuncAnimation(fig, actualizar,
                     frames=len(px), interval=40)
```

La animación puede mostrarse interactivamente o guardarse mediante un escritor apropiado. Igual que en los otros entornos, los límites deben fijarse antes de iniciar la secuencia.

Complemento: La misma llamada que en Scilab

Matplotlib Hunter 2007b reproduce la interfaz de `plot` de Scilab: se importa `matplotlib.pyplot as plt` y las llamadas son las mismas. Se supone `x` e `y` creados como en la unidad interactiva (`x = np.linspace(0, 10, 101)`, `y = np.sin(x)`).

```
In [1]: import matplotlib.pyplot as plt
In [2]: plt.plot(x, y, label='sen x')
In [3]: plt.plot(x, np.cos(x), '--', label='cos x')
In [4]: plt.xlabel('x'); plt.ylabel('y'); plt.legend();
   plt.grid()
In [5]: plt.savefig('seno.png')
In [6]: plt.show()
```

En IPython conviene activar el modo interactivo con `%matplotlib` para que la ventana se actualice sin bloquear la consola; en Jupyter el gráfico aparece debajo de la celda.

Complemento: Anatomía de una figura

Detrás de `plt.plot` hay dos objetos: la figura (`Figure`, la hoja) y los ejes (`Axes`, un gráfico dentro de la hoja). La interfaz de `plt` opera sobre la figura y los ejes actuales, como `scf` y `gca` en Scilab; la forma explícita, recomendada apenas hay más de un gráfico, es pedirlos y usar sus métodos The Matplotlib development team 2026.

```
In [1]: fig, ax = plt.subplots()
In [2]: ax.plot(x, y, label='sen x')
In [3]: ax.set_xlabel('x'); ax.set_ylabel('y')
In [4]: ax.set_title('Seno'); ax.legend(); ax.grid()
In [5]: fig.savefig('seno.pdf')
In [6]: fig, axs = plt.subplots(2, 1, sharex=True)    #
   subplot(2,1,k)
In [7]: axs[0].plot(x, y); axs[1].plot(x, np.cos(x))
```

Complemento: Tipos de gráfico

Los que se usan en el laboratorio: `plot` para curvas y datos con marcadores (`'o'`, `'.-'`), `scatter` para puntos con tamaño o color por dato, `errorbar` para medidas con incerteza, `bar` e `hist` para conteos, `semilogy` y `loglog` para escalas logarítmicas, `contour` y `contourf` para curvas de nivel, `imshow` para matrices como imagen. Los formatos de línea y color siguen la notación corta de MATLAB/Scilab: `'r-'`, `'k.'`.

```
In [1]: t = np.linspace(0, 5, 20); h = 4.9 * t**2 +
   np.random.normal(0, 3, 20)
In [2]: plt.errorbar(t, h, yerr=3, fmt='o', label='medido')
In [3]: plt.plot(t, 4.9 * t**2, label='teoria')
In [4]: plt.semilogy(x, np.exp(x))
In [5]: X, Y = np.meshgrid(x, x); plt.contourf(X, Y, np.sin(X) *
   np.cos(Y))
```

Complemento: Salida: pantalla, archivo, LaTeX

`plt.show()` abre la ventana; en IPython `%matplotlib` la deja viva sin bloquear la consola; en Jupyter el gráfico aparece debajo de la celda. `savefig` escribe el archivo y el formato sale de la extensión: PNG para pantalla e informes rápidos, PDF o SVG para incluir en LaTeX sin pérdida, como se venía haciendo con gnuplot. `plt.savefig('f.pdf', bbox_inches='tight')` recorta los márgenes. Para que la tipografía coincida con el documento se puede pedir que Matplotlib use LaTeX para el texto (`plt.rcParams['text.usetex'] = True`) o exportar con el motor PGF, que produce un archivo que LaTeX compila directamente.

Complemento: Gráficos 3D con Matplotlib

Lo que en Scilab son `surf`, `plot3d` y `param3d` está en Matplotlib pidiendo unos ejes con proyección 3D The Matplotlib development team 2026. La superficie necesita las mismas grillas `X`, `Y` de `meshgrid` que las curvas de nivel; la curva paramétrica recibe los tres vectores. Es la misma idea con un paso más: crear los ejes.

```
In [1]: X, Y = np.meshgrid(x, x); Z = np.sin(X) * np.cos(Y)
In [2]: ax = plt.figure().add_subplot(projection='3d')
In [3]: ax.plot_surface(X, Y, Z, cmap='viridis')       # surf(x,
   y, Z')
In [4]: ax.plot(np.cos(x), np.sin(x), x)               # param3d
In [5]: ax.set_xlabel('x'); ax.view_init(30, 45)       #
   elevacion, azimut
```

Dos avisos. El orden de los índices: en Scilab `surf(x, y, Z)` espera `Z` con tantas filas como `y`, y en Matplotlib `plot_surface` espera `Z` con la forma de `X` e `Y` de `meshgrid`, que también es filas por `y`; conviene mostrar `Z.shape` antes de dibujar. Y el rendimiento: la ventana 3D de Matplotlib se dibuja por software y se vuelve lenta con grillas grandes; para inspeccionar interactivamente una superficie grande hay herramientas mejores, pero para la figura del informe alcanza.

Complemento: Gnuplot desde Python

Lo que se viene haciendo desde Scilab, escribir un archivo de datos y un guion y llamar a gnuplot con `unix`, se hace en Python con `np.savetxt` y `subprocess` Python Software Foundation 2026c. Tiene sentido cuando el gráfico ya está resuelto en gnuplot para LaTeX y sólo cambian los datos: no hay que reescribir nada, y el mismo guion sirve desde Scilab, Python o el shell.

```
import numpy as np, subprocess
np.savetxt('seno.dat', np.column_stack([x, y]))
guion = """set terminal pdfcairo
set output 'seno.pdf'
plot 'seno.dat' with lines title 'sen x'
"""
open('seno.gnu', 'w').write(guion)
subprocess.run(['gnuplot', 'seno.gnu'], check=True)
```

La cadena con triple comilla reemplaza a la construcción con `NL` y `Q` de Scilab: los saltos de línea y las comillas van directamente. Si el guion es largo conviene tenerlo en un archivo aparte y pasarle sólo el nombre de datos, con `gnuplot -e "datos='seno.dat'"guion.gnu`.

Complemento: Otras formas de graficar

Matplotlib y gnuplot cubren lo que el laboratorio necesita. Existen además, y basta nombrarlas: bibliotecas que dibujan sobre Matplotlib con menos código para datos tabulados (seaborn, y el método `.plot` de pandas); bibliotecas que producen gráficos interactivos en el navegador (Plotly, Bokeh), que generan HTML y JavaScript y se ven en la unidad de interfaces web, donde el mismo gráfico se hace directamente con Plotly.js; herramientas de visualización 3D dedicadas (Mayavi, PyVista); animaciones con `matplotlib.animation`; y la escritura directa de SVG como texto, que es un ejercicio de la unidad de gráficos e imágenes.

Complemento: Tabla de equivalencias

Scilab	Python (Matplotlib)
`plot(x, y)`	`plt.plot(x, y)`
`plot(x, y, 'r-')`	`plt.plot(x, y, 'r-')`
`plot2d(x, y, logflag='nl')`	`plt.semilogy(x, y)`
`xlabel`, `ylabel`, `title`	`plt.xlabel`, `plt.ylabel`, `plt.title`
`legend('a', 'b')`	`plt.legend()` con `label=` en cada `plot`
`xgrid()`	`plt.grid()`
`subplot(2,1,1)`	`plt.subplot(2,1,1)` o `plt.subplots(2,1)`
`scf(); clf()`	`plt.figure(); plt.clf()`
`contour(x, y, z, n)`	`plt.contour(X, Y, Z, n)` (con `meshgrid`)
`Matplot(A)`	`plt.imshow(A)`
`xs2png(0, 'f.png')`	`plt.savefig('f.png')`
`xs2pdf(0, 'f.pdf')`	`plt.savefig('f.pdf')`
`surf(x, y, Z)`	`ax.plot_surface(X, Y, Z)` (ejes 3D)
`param3d(x, y, z)`	`ax.plot(x, y, z)` (ejes 3D)
`unix('gnuplot f.gnu')`	`subprocess.run(['gnuplot', 'f.gnu'])`

Como en la unidad interactiva, la conclusión es que cambia la notación y no la idea: la misma secuencia datos, curva, etiquetas, archivo, con nombres casi iguales.

7.2.15. Gráficos directos en Perl

El tratamiento numérico y gráfico en Perl se apoya en PDL y en sus interfaces gráficas PLplot y Prima Glazebrook y Economou 1997; PDL Developers 2026a; D. Hunt, Laboissiere y Mertens 2026; Mertens 2026.

Perl no posee en su núcleo una única biblioteca científica gráfica comparable a Scilab u Octave. En el ecosistema PDL existen varios motores. Esto no es una desventaja conceptual: obliga a distinguir el lenguaje de la biblioteca que implementa la representación.

Para las figuras científicas estáticas de esta unidad se utilizará como referencia `PDL::Graphics::PLplot`, que proporciona gráficos 2D, histogramas, mapas, contornos y acceso a funciones 3D de PLplot. Para interacción y animación puede utilizarse `PDL::Graphics::Prima`, cuya interfaz está basada en objetos y widgets. En ambos casos el camino es "Perl directo" en el sentido de que el programa usa una biblioteca cargada en el proceso y no genera un script para un ejecutable Gnuplot.

Curvas y salida con PLplot

Un ejemplo mínimo es:

```
use strict;
use warnings;
use PDL;
use PDL::Graphics::PLplot;

my $x = sequence(200) * (2 * 4 * atan2(1,1) / 199);
```

```
my $y = sin($x);

my $pl = PDL::Graphics::PLplot->new(
    DEV  => "png",
    FILE => "seno.png"
);

$pl->xyplot(
    $x, $y,
    TITLE => "Funcion seno",
    XLAB  => "x [rad]",
    YLAB  => "sin(x)",
    PLOTTYPE => "LINE"
);

$pl->close;
```

El objeto `PLplot` concentra la configuración de la salida y los métodos de representación. Las series son objetos PDL, equivalentes conceptualmente a los vectores de Scilab, Octave o NumPy.

Puntos, error e histogramas

`xyplot` admite tipos de trazado de puntos o líneas y opciones de error vertical u horizontal. Así, los datos centrales y sus incertidumbres siguen siendo arreglos separados, como en todos los demás lenguajes.

PLplot proporciona también métodos de histograma. Otra estrategia válida, especialmente útil con fines didácticos, es calcular los intervalos y conteos con PDL y representar después ese resultado. De esa forma la decisión sobre el ancho de clase queda visible en el algoritmo.

Contornos y 3D

Para matrices 2D, `shadeplot` permite producir mapas de color o contornos. Las funciones de bajo nivel de PLplot incluyen mallas y superficies 3D. Un patrón para contornos es:

```
# $z es un PDL bidimensional calculado previamente
my $niveles = 20;
$pl->shadeplot(
    $z, $niveles,
    PLOTTYPE => "CONTOUR",
    BOX => [-2, 2, -2, 2]
);
```

Para una superficie 3D pueden usarse las funciones de PLplot que reciben los vectores x, y y la matriz Z. La orientación de la matriz debe comprobarse igual que en los otros entornos.

Interacción y animación con Prima

`PDL::Graphics::Prima` implementa un widget de trazado que forma parte de una aplicación Prima. El gráfico conserva conjuntos de datos y objetos de ejes; las aplicaciones pueden responder a eventos y modificar esos datos. Este modelo resulta más natural para una animación que crear repetidamente archivos estáticos.

Un programa del curso puede usar PLplot para las figuras de informe y Prima para experimentar con una escena interactiva. Lo importante es no confundir esta decisión de biblioteca con la alternativa Gnuplot externo: ambos caminos parten de los mismos arreglos PDL pero producen la figura mediante arquitecturas distintas.

7.2.16. Gráficos directos en JavaScript/Node.js

Para esta vía se toman como referencia la documentación de Plotly.js y Node.js Plotly Technologies Inc. 2026; OpenJS Foundation 2026b.

En un navegador, JavaScript puede dibujar directamente mediante SVG y el `canvas` 2D, que son primitivas útiles cuando se necesita controlar la geometría o la interacción a bajo nivel. Para cubrir con una sola interfaz los gráficos científicos de esta unidad se tomará Plotly.js, que ofrece una capa de más alto nivel para curvas, barras de error, histogramas, ejes logarítmicos, coordenadas polares, contornos, superficies 3D, paneles y animaciones.

Plotly.js representa normalmente en un documento HTML dentro de un navegador. Node.js, por sí solo, no proporciona el DOM ni el entorno gráfico del navegador; por eso, en este curso se separan dos tareas:

a) Node.js calcula los datos y genera el documento HTML o los datos que éste consume;

b) el navegador ejecuta Plotly.js y construye la figura interactiva.

Este camino sigue siendo distinto de Gnuplot: no se crea ni ejecuta un script `.gnu`. El motor gráfico es la biblioteca JavaScript cargada por la página.

Estructura declarativa de Plotly.js

Una figura se describe mediante trazas y un objeto de disposición. El ejemplo siguiente corresponde al código que aparecería dentro de la página HTML:

```
const x = Array.from({length: 200},
    (_, i) => 2*Math.PI*i/199);
const y = x.map(v => Math.sin(v));

const data = [{
    x,
    y,
    type: "scatter",
    mode: "lines",
    name: "sin(x)"
}];
```

```
const layout = {
    title: {text: "Funcion seno"},
    xaxis: {title: {text: "x [rad]"}},
    yaxis: {title: {text: "sin(x)"}}
};

Plotly.newPlot("grafico", data, layout);
```

La misma idea funciona para múltiples curvas: se agrega una traza por serie. Cada traza puede poseer su propio arreglo de abscisas, por lo que los dominios y las cantidades de puntos no tienen que coincidir.

Barras de error e histogramas

Una traza de dispersión puede incluir error vertical:

```
const mediciones = {
    x: [0, 0.5, 1.0, 1.5],
    y: [0.03, 0.46, 0.85, 0.99],
    type: "scatter",
    mode: "markers",
    error_y: {
        type: "data",
        array: [0.05, 0.06, 0.05, 0.04],
        visible: true
    }
};
```

Para un histograma basta una traza cuyo tipo sea `histogram`. Las opciones de normalización permiten comparar conteos, probabilidades o densidad, pero la interpretación estadística debe decidirse antes de elegir la opción gráfica.

Escalas y polares

Los ejes se configuran en `layout`. Un eje logarítmico se declara como parte de la disposición. Para polares existen trazas específicas como `scatterpolar`; alternativamente se puede realizar la transformación a cartesianas y utilizar una traza ordinaria.

Contornos y superficies

Un mapa de contornos se representa mediante una matriz 2D:

```
const contorno = {
    x: xs,
    y: ys,
    z: Z,
    type: "contour",
    colorscale: "Viridis",
```

```
    contours: {coloring: "heatmap"}
};

Plotly.newPlot("mapa", [contorno]);
```

Una superficie tridimensional usa el tipo `surface`:

```
const superficie = {
    x: xs,
    y: ys,
    z: Z,
    type: "surface",
    colorscale: "Viridis"
};

Plotly.newPlot("superficie", [superficie]);
```

Las superficies permiten además configurar contornos proyectados. Esto resulta útil para el ejercicio que compara una superficie, los contornos sobre ella y los contornos proyectados sobre un plano.

Node.js como generador del documento

Node.js puede calcular arreglos, serializarlos como JSON e insertar esos datos en una plantilla HTML. En un ejemplo mínimo:

```
"use strict";
const fs = require("node:fs");

const x = Array.from({length: 200}, (_, i) => 10*i/199);
const y = x.map(v => (v*v - 1)/(v*v*v + 1));

const html = `<!doctype html>
<meta charset="utf-8">
<div id="grafico"></div>
<script src="plotly.min.js"></script>
<script>
const x = ${JSON.stringify(x)};
const y = ${JSON.stringify(y)};
Plotly.newPlot("grafico", [{x, y, type:"scatter",
   mode:"lines"}]);
</script>`;

fs.writeFileSync("grafico.html", html, "utf8");
```

Para un trabajo reproducible conviene conservar localmente la versión de Plotly.js usada por el curso, o documentar claramente cómo se obtiene. De ese modo el archivo no depende de que una red externa esté disponible al abrirlo.

Interacción y animación

Plotly mantiene una figura en el DOM del navegador y permite actualizarla. Puede definirse una secuencia de *frames* y animarla, o modificar trazas y disposición. Esta arquitectura es retenida: la figura existente cambia sin crear un archivo PNG nuevo en cada paso.

Si el resultado final debe ser una secuencia de imágenes o un vídeo se añade una etapa de exportación. Para los objetivos conceptuales del curso conviene separar esa etapa de la animación interactiva en el navegador.

7.2.17. Matrices representadas como imágenes

Una matriz de valores puede mostrarse como una imagen en la que cada celda se convierte en un color. Esta representación aparece naturalmente en mapas de calor, imágenes numéricas y datos sobre grillas. Es diferente de una curva de nivel: en la imagen se colorea cada celda; en el contorno se reconstruyen líneas que aproximan los lugares donde la función toma valores constantes.

Las interfaces habituales son conceptualmente semejantes:

Entorno	**Operación típica**
Scilab	`Matplot`, `Sgrayplot`
GNU Octave	`imagesc`, `pcolor`
Python/Matplotlib	`imshow`, `pcolormesh`
Perl/PDL	mapas mediante PLplot/`shadeplot`
JavaScript/Plotly.js	traza `heatmap`
Gnuplot	`pm3d map` o datos matriciales

En todos los casos debe conocerse la relación entre índices de matriz y coordenadas físicas. Mostrar una matriz 100×100 sin indicar qué valores de x e y representa puede ser suficiente para inspección interna, pero no para una figura científica.

7.2.18. Exportación, formatos e interoperabilidad

El resultado gráfico puede permanecer en una ventana interactiva o persistirse en un archivo. El formato debe elegirse según el tipo de figura y su destino.

Interoperabilidad gráfica

Por interoperabilidad gráfica se entiende la posibilidad de trasladar datos y especificaciones de una representación entre herramientas distintas sin tener que repetir el cálculo original. En esta unidad aparece en varios niveles: un archivo tabular puede ser generado por cualquiera de los lenguajes y leído por Gnuplot; un mismo script `.gnu` puede reutilizarse con datos producidos por programas diferentes; y formatos públicos como PNG, PDF y SVG permiten transportar el resultado entre aplicaciones y sistemas gnuplot Team

2026; International Organization for Standardization 2004; International Organization for Standardization 2020; World Wide Web Consortium 2026.

La interoperabilidad no implica que todos los formatos conserven las mismas propiedades. Un archivo raster conserva la imagen final pero no la estructura de los objetos gráficos; un formato vectorial puede conservar geometría y texto; y una salida interactiva, como un documento HTML con Plotly.js, puede conservar además comportamiento. Por eso conviene mantener separados los datos, la especificación gráfica y el archivo de salida, y conservar como origen reproducible el programa o script que generó la figura.

Persistencia de la escena gráfica y exportación

Guardar una figura no siempre significa guardar lo mismo. Conviene distinguir entre *persistir la escena gráfica* y *exportar su representación final*. En una escena persistente se conservan los objetos gráficos y sus propiedades de forma que puedan cargarse nuevamente y seguir modificándose. En una exportación raster se conservan solamente los píxeles; en una exportación vectorial se conserva geometría dibujable, pero no debe suponerse que se recupere la jerarquía original de objetos, sus handles o sus callbacks.

Scilab permite guardar y volver a cargar una escena con `xsave` y `xload` Scilab Enterprises 2026c. Por ejemplo:

```
xsave("figura.scg", gcf())
clf()
xload("figura.scg")
```

Después de cargarla, la figura vuelve a ser una escena gráfica sobre la que se puede continuar trabajando. Reabrir `figura.png` o `figura.pdf` con un visor no es equivalente: el visor muestra una representación terminada, no los objetos originales del entorno de cálculo.

Una vez recuperada la escena, puede exportarse al formato requerido. Por ejemplo, para obtener PostScript encapsulado en Scilab:

```
xs2eps(gcf(), "figura.eps")
```

Por eso, cuando un ejercicio pide guardar, cerrar, volver a abrir y luego exportar una figura, debe identificarse qué archivo actúa como persistencia de la escena y cuál es sólo el archivo final de intercambio o publicación.

Raster y vectorial

Los formatos tratados se corresponden con las especificaciones de PNG, JPEG, SVG y PDF International Organization for Standardization 2004; International Telecommunication Union 1992; World Wide Web Consortium 2026; International Organization for Standardization 2020.

Un formato raster almacena una cuadrícula de píxeles. Al ampliar demasiado se hacen visibles los píxeles y las líneas dejan de ser nítidas. PNG es el formato raster preferido para gráficos científicos con texto y líneas porque utiliza compresión sin pérdida International

Organization for Standardization 2004.

Un formato vectorial conserva objetos geométricos: líneas, curvas, texto y rellenos. Puede ampliarse sin perder nitidez. PDF y SVG son opciones adecuadas para figuras de líneas, esquemas y gráficos destinados a documentos o web International Organization for Standardization 2020; World Wide Web Consortium 2026.

JPEG utiliza compresión con pérdida y está pensado principalmente para fotografías. La cuantización puede introducir halos y artefactos alrededor de líneas y texto, por lo que normalmente no es la primera elección para un gráfico científico International Telecommunication Union 1992.

Resolución y tamaño físico

En una imagen raster interesa distinguir píxeles de tamaño impreso. Si una figura de ancho L centímetros se imprimirá a D puntos por pulgada, un ancho orientativo en píxeles es

$$N = \frac{L}{2{,}54}\, D.$$

Por ejemplo, 12 cm a 300 dpi requieren aproximadamente 1417 píxeles. Exportar una figura demasiado pequeña y ampliarla después no crea detalle nuevo.

En un archivo vectorial la resolución no limita las líneas y el texto, aunque pueden existir elementos raster incrustados, por ejemplo un mapa de color muy denso.

Salida según el entorno

Entorno	Pantalla / interacción	Archivo
Gnuplot	terminal interactiva	`set terminal`, `set output`
Scilab	figura gráfica	`xs2png`, `xs2pdf`, `xs2svg`
Octave	figura gráfica	`print`
Python	`plt.show()`	`savefig`
Perl/PLplot	dispositivo gráfico	`DEV/FILE`
JavaScript / Plotly	navegador	exportación de Plotly / HTML

Para Plotly.js el HTML interactivo es en sí mismo una forma de persistencia: conserva datos, configuración y comportamiento. La exportación estática directa de Plotly.js permite obtener formatos como PNG, JPEG o SVG. Si el destino exige PDF puede añadirse después una conversión o una herramienta de exportación compatible, conservando el HTML como origen reproducible.

Complemento: Formatos de imagen

Para los formatos tratados aquí se toman como referencia las especificaciones de PNG, JPEG, SVG y PDF International Organization for Standardization 2004; International

Telecommunication Union 1992; World Wide Web Consortium 2026; International Organization for Standardization 2020.

Existen dos grandes familias de formatos de imagen con características fundamentalmente distintas.

Imágenes rasterizadas (mapa de bits) Una imagen rasterizada es una cuadrícula de píxeles, cada uno con un valor de color. Sus propiedades dependen de la resolución (cantidad de píxeles por pulgada, DPI). Al ampliar una imagen rasterizada más allá de su resolución original se ve pixelada.

Las operaciones de inspección y conversión realizadas con `identify` y `convert` corresponden a ImageMagick ImageMagick Studio LLC 2026a; ImageMagick Studio LLC 2026b.

```
$sudo apt install file
$#identify es de ImageMagick
$sudo apt install libimage-exiftool-perl
$sudo apt install xxd
$sudo apt install optipng

$# Identificar formato y propiedades de una imagen
$file imagen.png
$identify imagen.png        # requiere ImageMagick
$exiftool imagen.png        # metadatos completos
$
$# Ver los primeros bytes (firma magica)
$xxd imagen.png | head -3
$# PNG: 89 50 4e 47 0d 0a 1a 0a  (8950 4e47 0d0a 1a0a)
$xxd imagen.jpg | head -1
$# JPEG: ff d8 ff e0 ...

$# Convertir JPEG a PNG (sin perdida a partir de JPEG)
$convert foto.jpg foto.png

$# Comprimir un PNG sin perder calidad
$optipng -o5 figura.png
$pngcrush figura.png figura_comprimida.png
```

Los formatos rasterizados más importantes son:

PNG (*Portable Network Graphics*): compresión sin pérdida, soporta transparencia (canal alfa), ideal para gráficos científicos, capturas de pantalla y cualquier imagen donde la fidelidad exacta importa. No adecuado para fotografías (archivos grandes) International Organization for Standardization 2004.

JPEG (*Joint Photographic Experts Group*): compresión con pérdida ajustable, ideal para fotografías. Cada vez que se guarda un JPEG se pierde calidad. No adecuado para gráficos con texto o bordes nítidos International Telecommunication Union 1992.

TIFF (*Tagged Image File Format*): formato de alta calidad usado en publicación profesional e imágenes científicas. Sin pérdida. Archivos grandes.

WebP: formato moderno de Google, mejor compresión que PNG y JPEG con calidad comparable. Adoptado ampliamente en la web.

Imágenes vectoriales Una imagen vectorial almacena objetos geométricos (puntos, líneas, curvas, polígonos) y sus propiedades (color, grosor, relleno). Se puede escalar a cualquier tamaño sin pérdida de calidad porque los objetos se recalculan en cada renderizado.

```
# Examinar el contenido de un SVG (es XML legible)
$cat figura.svg | head -30

# Optimizar un SVG eliminando metadatos innecesarios
svgo figura.svg -o figura_opt.svg

# Convertir un PDF de una pagina a SVG
pdf2svg figura.pdf figura.svg

# Ver informacion de un PDF
pdfinfo documento.pdf
```

Los formatos vectoriales más importantes son:

SVG (*Scalable Vector Graphics*): estándar W3C basado en XML. Los navegadores web lo renderizan nativamente. Ideal para gráficos en páginas web, se puede estilizar con CSS y animar con JavaScript. Se puede editar con un editor de texto World Wide Web Consortium 2026.

PDF (*Portable Document Format*): puede contener tanto objetos vectoriales como imágenes rasterizadas. Es el formato estándar para gráficos incluidos en documentos LaTeX. La inclusión de un PDF en LaTeX con `\includegraphics` produce resultados de calidad tipográfica perfecta International Organization for Standardization 2020.

EPS (*Encapsulated PostScript*): predecesor del PDF, usado históricamente en publicación científica. Hoy reemplazado en la mayoría de los casos por PDF.

conversión de mágenes Convert es parte de ImageMagick ImageMagick Studio LLC 2026a; ImageMagick Studio LLC 2026b.

```
$sudo apt install imagemagick

$# Desde la terminal, convertir entre formatos con ImageMagick
$# PNG de alta resolucion a JPEG
$convert -quality 95 seno.png seno.jpg

$# Aumentar la resolucion (DPI) para impresion
$convert -density 300 seno.pdf seno_hires.png

$# Recortar espacios en blanco automaticamente
$convert -trim seno.png seno_recortado.png
```

```
$# Redimensionar
convert -resize 50% seno.png seno_pequenio.png
```

```
$sudo apt install inkscape
$# Desde la terminal, trabajar con SVG
$# Convertir SVG a PNG con Inkscape
$inkscape --export-png=figura.png --export-dpi=150 figura.svg
$
$# Convertir SVG a PDF
$inkscape --export-pdf=figura.pdf figura.svg
```

7.2.19. Comparación entre Gnuplot y la vía directa

No existe una respuesta universal a la pregunta de cuál camino es mejor. La elección depende de qué parte del trabajo se quiere simplificar.

Aspecto	Gnuplot externo	Gráficos directos
Separación cálculo–figura	Muy explícita por archivos	Puede hacerse, pero no es obligatoria
Automatización sin pantalla	Natural	Depende de la biblioteca y backend
Interacción durante el cálculo	Menor, salvo proceso persistente	Normalmente más natural
Animación por objetos	No es el camino habitual	Habitual en sistemas retenidos
Reproducibilidad	Script `.gnu` + datos	Programa + versiones de bibliotecas
Portabilidad entre lenguajes	El mismo script puede reutilizarse	Cada lenguaje tiene su API
Acoplamiento con datos en memoria	Requiere serializar o comunicar	Directo
Diagnóstico de procesos	Debe controlarse explícitamente	No hay proceso Gnuplot adicional

Se practican ambos caminos porque ejercitan aspectos distintos de programación: Gnuplot enfatiza composición de programas, archivos, procesos y especificaciones textuales; las bibliotecas directas, objetos, APIs, estado gráfico e interacción.

7.2.20. Equivalencias conceptuales entre interfaces

Las funciones no tienen por qué llamarse igual para realizar la misma tarea. La tabla siguiente debe leerse como un mapa conceptual, no como una traducción mecánica de sintaxis.

Operación	Gnuplot	Scilab
Curva 2D	`plot ... with lines`	`plot`
Puntos	`with points`	`plot/scatter`
Error vertical	`with yerrorbars`	`errbar`
Histograma	agrupación/frecuencia	`histplot`
Semilog y	`set logscale y`	`semilogy`
Log–log	`set logscale xy`	`loglog`
Polar	`set polar`	`polarplot`
Paneles	`set multiplot`	`subplot`
Superficie	`splot/pm3d`	`surf/plot3d`
Contornos	`set contour`	`contour2d/contourf`
Campo vectorial	vectores/flechas	`champ`
Archivo	terminal + output	`xs2*`

Operación	GNU Octave	Python/Matplotlib
Curva 2D	`plot`	`Axes.plot`
Puntos	`plot/scatter`	`scatter/plot`
Error vertical	`errorbar`	`errorbar`
Histograma	`hist`	`hist`
Semilog y	`semilogy`	`semilogy`
Log–log	`loglog`	`loglog`
Polar	`polar`	proyección `polar`
Paneles	`subplot`	`subplots`
Superficie	`surf/mesh`	`plot_surface`
Contornos	`contour/contourf`	`contour/contourf`
Campo vectorial	`quiver`	`quiver`
Archivo	`print`	`savefig`

Operación	Perl/PDL	JavaScript/Plotly.js
Curva 2D	`xyplot`	`scatter`, modo `lines`
Puntos	`xyplot` con puntos	`scatter`, modo `markers`
Error vertical	`YERRORBAR`	`error_y`
Histograma	`histogram` / cálculo PDL	traza `histogram`
Escala log	opciones de ejes	`type="log"` en xaxis/yaxis
Polar	transformación / biblioteca	`scatterpolar`
Paneles	`SUBPAGES`	subplots/domains
Superficie	funciones PLplot 3D	traza `surface`
Contornos	`shadeplot`	traza `contour`
Campo vectorial	primitivas PLplot	trazas de vectores/conos o segmentos
Archivo	dispositivo `DEV/FILE`	HTML / exportación Plotly

7.2.21. Un mismo problema, dos soluciones

La comparación entre las dos modalidades no debería reducirse a comprobar que ambas producen una imagen parecida. Conviene identificar qué parte cambia y qué parte permanece.

Supongamos que se quiere representar

$$f(x) = \frac{x^2 - 1}{x^3 + 1}$$

en un intervalo sin singularidades. En cualquier lenguaje el cálculo esencial es:

1. construir el vector de abscisas;
2. evaluar la expresión elemento a elemento;
3. decidir qué puntos son válidos;
4. asignar un título y nombres de ejes;
5. producir una curva;
6. guardar el resultado.

Con Gnuplot aparecen además la serialización de datos, el script y el proceso externo. Con la biblioteca directa desaparecen esos archivos intermedios, pero aparecen objetos, funciones de la API y un backend gráfico. La estructura matemática del problema no cambia.

Esta observación es el criterio que se utilizará al escribir las soluciones del práctico: las diez soluciones de un problema deben realizar el mismo trabajo conceptual. Las diferencias deben provenir de las capacidades reales de cada lenguaje y biblioteca, no de que una versión resuelva un problema más simple.

7.2.22. Criterios para elegir el camino

Puede preferirse **Gnuplot externo** cuando:

- se quiere producir muchas figuras de forma automática y sin pantalla;
- los datos ya están en archivos de texto;
- se desea reutilizar exactamente la misma especificación gráfica desde varios lenguajes;
- interesa separar con claridad el cálculo de la presentación;
- la figura debe poder regenerarse manualmente sin iniciar el programa que creó los datos.

Puede preferirse la **biblioteca directa** cuando:

- se está explorando el resultado durante el cálculo;
- los datos son grandes y ya están en memoria;
- se necesita modificar objetos de la figura;
- se requiere interacción con ratón o teclado;
- se está construyendo una animación o interfaz;
- la biblioteca dispone de una operación específica que simplifica el problema.

En un proyecto real ambos caminos pueden coexistir. Una simulación puede usar una figura directa para inspección durante el desarrollo y, al finalizar, guardar datos que un script Gnuplot convierte automáticamente en las figuras del informe.

7.2.23. Buenas prácticas para figuras científicas

Las reglas siguientes son independientes de la herramienta:

1. **Mostrar los datos antes que la decoración.** La mayor parte de la información visual debe corresponder al fenómeno que se quiere comunicar Tufte 2001.
2. **Rotular ejes y unidades.** Un número físico sin unidad es ambiguo.
3. **Usar puntos y líneas según su significado.** Las mediciones son discretas; un modelo continuo puede representarse mediante una línea.
4. **No usar el color como único identificador.** Combinar color con marcador o patrón de línea.
5. **Mantener la geometría cuando importa.** En mapas, curvas polares y trayectorias planas debe cuidarse la relación de aspecto.
6. **No ocultar discontinuidades ni fallos numéricos.** Una asíntota, un NaN y un overflow tienen causas diferentes y deben tratarse como tales.
7. **Usar barras de error cuando la incertidumbre forma parte del dato.**
8. **Elegir conscientemente la resolución.** Esto vale para el número de puntos de una curva, la grilla de una superficie, las clases de un histograma y los píxeles de una imagen exportada.
9. **Conservar el origen reproducible.** Una figura importante debe poder regenerarse a partir del programa, datos y parámetros que la produjeron. Una captura de pantalla no reemplaza ese conjunto.

7.2.24. Flujo de trabajo recomendado

Para los ejercicios de esta unidad resulta útil seguir siempre la misma secuencia:

1. escribir claramente la función, modelo o estructura de los datos;
2. determinar el dominio y localizar singularidades o restricciones;
3. elegir la discretización o grilla;
4. calcular y validar los datos sin depender todavía del gráfico;
5. decidir qué representación responde mejor a la pregunta: línea, puntos, barras de error, histograma, superficie, contorno, campo vectorial, paneles o animación;
6. construir primero una figura mínima;
7. agregar título, ejes, unidades, leyenda, escalas y relación de aspecto;
8. guardar una salida adecuada;
9. repetir la representación por el segundo camino;
10. comparar que ambas soluciones expresen la misma información.

El objetivo final no es memorizar una colección de órdenes. Es poder pasar de un resultado numérico a una representación correcta y reproducible, y reconocer que Gnuplot, Scilab, Octave, Matplotlib, PDL y Plotly.js son interfaces diferentes para resolver problemas gráficos que comparten los mismos principios.

7.3. Ejercicios

7.3.1. Problemas de representación gráfica

Instalación y verificación de Gnuplot

Los lenguajes de trabajo del curso se consideran ya instalados. Para una de las dos modalidades de resolución se utilizará además Gnuplot como programa gráfico externo, por lo que debe estar instalado y accesible desde la línea de comandos.

En GNU/Linux conviene instalarlo mediante el administrador de paquetes de la distribución. Algunos ejemplos habituales son:

```
# Debian, Ubuntu y derivados
sudo apt update
sudo apt install gnuplot

# Fedora
sudo dnf install gnuplot

# Arch Linux y derivados
```

```
sudo pacman -S gnuplot

# openSUSE
sudo zypper install gnuplot
```

En otras distribuciones utilice el administrador de paquetes correspondiente y busque el paquete denominado `gnuplot`. No es necesario compilar Gnuplot desde el código fuente para realizar este práctico.

En Windows puede instalarse mediante `winget` desde PowerShell, Símbolo del sistema o Windows Terminal:

```
winget install --id gnuplot.gnuplot -e
```

También puede utilizarse el instalador ofrecido desde la página oficial de descarga de Gnuplot. Durante la instalación debe verificarse que el ejecutable quede accesible desde la variable `PATH`, de modo que pueda invocarse como `gnuplot` desde una terminal y desde los programas del curso.

En macOS, si se utiliza Homebrew, puede instalarse mediante:

```
brew install gnuplot
```

Después de la instalación compruebe que el programa pueda ejecutarse desde una terminal:

```
gnuplot --version
```

La versión exacta no es importante para los ejercicios de esta unidad. Como prueba mínima de funcionamiento puede generarse un archivo PNG sin entrar al modo interactivo:

```
gnuplot -e "set terminal pngcairo; set output
   'prueba_gnuplot.png'; plot sin(x)"
```

Compruebe que se haya creado `prueba_gnuplot.png`. Esta prueba verifica no sólo que el ejecutable se encuentre instalado, sino también que puede generar una salida gráfica desde un comando no interactivo, que es la forma que se utilizará cuando Gnuplot sea invocado desde Scilab, Octave, Python, Perl o JavaScript/Node.js.

Familiarización con Gnuplot

Los siguientes ejercicios son específicos de Gnuplot y tienen como objetivo familiarizar al estudiante con el programa antes de utilizarlo como motor gráfico externo desde los lenguajes del curso. Por esa razón constituyen una preparación común y no se duplican en la modalidad “lenguaje solamente”.

7.4. Ejercicios

7.4.1. Familiarización con Gnuplot

1. **Ejercicio:** Instalación de la herramienta gráfica externa

   Instale y verifique la herramienta gráfica externa asignada en GNU/Linux y Windows. Compruebe que pueda invocarse desde una terminal y que sea capaz de generar una figura sencilla.

   Nota Gnuplot: El paquete y el ejecutable se denominan `gnuplot`. En un sistema basado en Debian puede verificarse con:

   ```
   sudo apt install gnuplot
   gnuplot --version
   echo "plot sin(x)" | gnuplot -p
   ```

   Respuesta (Gnuplot).

   En GNU/Linux la instalación depende de la distribución. Por ejemplo, en Debian, Ubuntu y derivados:

   ```
   sudo apt update
   sudo apt install gnuplot
   gnuplot --version
   ```

   La última orden debe mostrar la versión instalada. Para comprobar además que Gnuplot puede producir una figura sin depender de una sesión interactiva:

   ```
   gnuplot -e "set terminal pngcairo; set output
      'prueba_gnuplot.png'; plot sin(x)"
   test -s prueba_gnuplot.png && echo "archivo generado
      correctamente"
   ```

   En Windows puede instalarse, por ejemplo, desde PowerShell o Windows Terminal mediante `winget`:

   ```
   winget install --id gnuplot.gnuplot -e
   gnuplot --version
   ```

   Si la terminal estaba abierta durante la instalación puede ser necesario cerrarla y abrirla nuevamente para que se actualice `PATH`. La misma prueba no interactiva puede realizarse desde PowerShell:

   ```
   gnuplot -e "set terminal pngcairo; set output
      'prueba_gnuplot.png'; plot sin(x)"
   Test-Path .\prueba_gnuplot.png
   ```

   El ejercicio queda verificado cuando `gnuplot -version` termina sin error y se obtiene un archivo `prueba_gnuplot.png` no vacío que contiene la curva de $\sin x$.

2. **Ejercicio:** Familiarización con Gnuplot

Realice estas tareas para familiarizarse con gnuplot. Este ejercicio no tiene respuesta, lo importante es hacer lo que se indica.

7.4.2. Sección 1: Herramientas de generación de gráficos y automatización

1. **Ejercicio:** Interacción con Gnuplot: limpieza y muestreo

   Inicie Gnuplot en modo interactivo desde una terminal:

   ```
   gnuplot
   ```

   Dentro de la consola de Gnuplot realice las siguientes tareas:

   a) represente $\sin^2(x)$ y observe la sintaxis utilizada para potencias;

   b) limpie la ventana gráfica sin abandonar el programa y posteriormente restablezca la configuración a sus valores iniciales;

   c) represente $\sin(x)$ utilizando sucesivamente 30, 100 y 500 puntos de muestreo y compare la suavidad aparente de las curvas;

   d) pruebe una pausa que espere una acción del teclado y otra que mantenga la ventana hasta que sea cerrada por el usuario;

   e) genere una salida gráfica en un archivo y compruebe que la figura se haya escrito correctamente;

   f) cuando la terminal gráfica utilizada lo permita, cierre una ventana concreta y luego todas las ventanas gráficas abiertas;

   g) identifique desde el sistema operativo los procesos asociados a Gnuplot y a sus terminales gráficas y pruebe una terminación controlada de esos procesos;

   h) abandone Gnuplot mediante el comando correspondiente y explique la diferencia entre limpiar una figura, restablecer la configuración, cerrar una ventana y terminar el proceso.

   Nota Gnuplot: En modo interactivo puede usarse:

   ```
   plot sin(x)**2
   clear
   reset
   ```

   Nota Gnuplot: En Gnuplot `**` representa la potenciación. Si la terminal se cambia a una salida de archivo, la figura deja de mostrarse necesariamente en pantalla y se escribe en el archivo seleccionado.

   Scilab. La operación `clear` de Gnuplot cumple un papel comparable a `clf()` para limpiar una figura en Scilab.

   Nota Gnuplot: `set samples 50` fija en 50 la cantidad de muestras; `pause -1` espera una acción del teclado y `pause mouse close` mantiene la ventana hasta que el usuario la cierra.

```
set samples 30
plot sin(x)
q
```

Nota Gnuplot: Con la terminal Qt puede cerrarse una ventana concreta indicando su número. Si se conocen las ventanas abiertas, se cierran explícitamente una por una; Gnuplot no necesita un bucle indefinido para hacerlo. Por ejemplo, si están abiertas las ventanas 0, 1 y 2:

```
set terminal qt 1 close
set terminal qt 0 close
set terminal qt 2 close
```

```
ps ax | grep gnuplot
killall gnuplot_qt
killall gnuplot_wxt
```

Scilab. La idea de fijar la cantidad de muestras es análoga a fijar en `linspace` la cantidad de puntos de una discretización.

Respuesta (Gnuplot).

a) En Gnuplot la potenciación se escribe con `**`. Por lo tanto:

```
plot sin(x)**2
```

La expresión representa $(\sin x)^2$.

b) Para borrar solamente el dibujo actual sin abandonar Gnuplot:

```
clear
```

Para restablecer después la configuración general de la sesión:

```
reset
```

`clear` elimina el contenido gráfico de la ventana, mientras que `reset` restablece la mayoría de las opciones configurables a sus valores iniciales. Ninguna de las dos órdenes termina el proceso Gnuplot.

c) La cantidad de muestras utilizadas para evaluar una función puede cambiarse con `set samples`:

```
set samples 30
plot sin(x)
pause -1 "30 muestras: pulse Enter para continuar"

set samples 100
plot sin(x)
pause -1 "100 muestras: pulse Enter para continuar"

set samples 500
plot sin(x)
```

Con 30 puntos la poligonal que aproxima a la función puede apreciarse con mayor facilidad. Al pasar a 100 y luego a 500 puntos la representación se hace más suave. La función matemática no cambia: cambia solamente la discretización usada para dibujarla.

d) Una pausa que espera una acción del teclado puede escribirse como:

```
pause -1 "Pulse Enter para continuar"
```

En un terminal que admita interacción con el ratón puede esperarse hasta que el usuario cierre la ventana gráfica:

```
pause mouse close "Cierre la ventana para continuar"
```

e) Para generar un archivo y después recuperar el terminal interactivo sin suponer cuál era el terminal original, puede guardarse y restaurarse su estado:

```
set term push
set terminal pngcairo size 800,500
set output 'interaccion.png'
set samples 500
plot sin(x) with lines title 'sin(x)'
unset output
set term pop
```

El archivo `interaccion.png` debe existir y tener tamaño distinto de cero. En GNU/Linux puede comprobarse desde otra terminal con:

```
test -s interaccion.png && echo "archivo generado
   correctamente"
```

f) Si se dispone del terminal Qt, pueden crearse varias ventanas numeradas y cerrarlas selectivamente. Por ejemplo:

```
set terminal qt 0
plot sin(x) title 'ventana 0'

set terminal qt 1
plot cos(x) title 'ventana 1'

set terminal qt 2
plot sin(x)*cos(x) title 'ventana 2'
```

Para cerrar únicamente la ventana 1:

```
set terminal qt 1 close
```

Como en este ejemplo se conocen los números de todas las ventanas abiertas, las restantes pueden cerrarse explícitamente:

```
set terminal qt 0 close
set terminal qt 2 close
```

g) Mientras Gnuplot permanece abierto, desde otra terminal de GNU/Linux pue-

den identificarse el proceso principal y, cuando exista, el proceso correspondiente al terminal gráfico:

```
pgrep -a gnuplot
pgrep -a gnuplot_qt
pgrep -a gnuplot_x11
```

Para terminar controladamente un proceso concreto se envía primero `SIGTERM`, utilizando el PID obtenido en la consulta anterior:

```
kill -TERM 12345
```

Debe reemplazarse `12345` por el PID real. Esto permite observar la diferencia entre terminar solamente un proceso de terminal gráfico y terminar el proceso principal de Gnuplot.

h) Para abandonar normalmente Gnuplot:

```
quit
```

Las operaciones anteriores tienen alcances diferentes:

- `clear`: borra el dibujo de la ventana actual, pero conserva la sesión y su configuración;
- `reset`: restablece la configuración de Gnuplot, pero el proceso continúa ejecutándose;
- cerrar una ventana gráfica: destruye esa ventana o terminal, pero no necesariamente termina el proceso principal;
- `quit`: termina la sesión y el proceso Gnuplot de forma normal.

2. **Ejercicio:** Ejecución de Gnuplot y exportación

Genere la misma gráfica sencilla mediante distintas formas de utilización de Gnuplot.

a) desde una sesión interactiva, genere un archivo PNG que contenga la curva $\sin(x)$;

b) genere también una salida vectorial en PDF;

c) ejecute desde la terminal una orden completa mediante la opción `-e`, por ejemplo para representar

$$\frac{x^2 - 1}{x^3 + 1}$$

en un intervalo elegido;

d) escriba con un editor de texto un archivo `comandos.gnu` que fije terminal, archivo de salida, rango, título, nombres de ejes y cuadrícula, y represente $x^2 - 1$;

e) ejecute el archivo `comandos.gnu` desde la terminal;

f) envíe una orden sencilla a Gnuplot mediante la entrada estándar, por ejemplo con una tubería desde `echo`;

g) abra alguno de los archivos obtenidos con un visor apropiado;

h) pruebe una modalidad persistente que mantenga abierta la ventana gráfica cuando haya terminado el proceso que la generó;

i) compare las distintas formas de ejecución e indique cuáles resultan más adecuadas para trabajo interactivo y cuáles para automatización reproducible.

Nota Gnuplot: Ejemplo de salida PNG desde la consola:

```
set terminal png
set output "datos.png"
plot sin(x)
quit
```

Nota Gnuplot: El archivo puede abrirse, por ejemplo, con `display` de ImageMagick o con el visor predeterminado:

```
sudo apt install imagemagick
display datos.png
xdg-open datos.png
```

Nota Gnuplot: Una orden completa puede pasarse con `-e`:

```
gnuplot -e "set xrange [0:10]; plot (x**2-1)/(x**3+1) with
   lines lw 2 lc 'red' notitle; pause mouse close"
```

Nota Gnuplot: También puede guardarse, por ejemplo, `comandos.gnu`:

```
set terminal png size 700,400
set output "comandos.png"
set xrange [0:10]
set xlabel "x"
set ylabel "y"
set title "(x^2-1)"
set grid
plot x*x-1 with lines lw 2 notitle
```

```
gnuplot comandos.gnu
display comandos.png
echo "plot sin(x)" | gnuplot -p
```

Nota Gnuplot: `-p` o `-persist` evita que la ventana gráfica se cierre al terminar el comando.

Respuesta (Gnuplot).

a) Desde una sesión interactiva puede conservarse el terminal actual, cambiar provisoriamente a PNG y volver luego al terminal de pantalla:

```
set term push
set terminal pngcairo size 800,500
set output 'seno.png'
set title 'sin(x)'
set xlabel 'x'
```

```
set ylabel 'sin(x)'
set grid
plot sin(x) with lines lw 2 notitle
unset output
set term pop
```

El archivo `seno.png` debe quedar creado y no vacío.

b) Para obtener una salida vectorial puede repetirse el procedimiento con `pdfcairo`:

```
set term push
set terminal pdfcairo size 12cm,8cm
set output 'seno.pdf'
set title 'sin(x)'
set xlabel 'x'
set ylabel 'sin(x)'
set grid
plot sin(x) with lines lw 2 notitle
unset output
set term pop
```

c) El intervalo $0 \leq x \leq 10$ no contiene el cero del denominador $x^3 + 1$, que se encuentra en $x = -1$. Desde la terminal del sistema puede usarse:

```
gnuplot -p -e "set xrange [0:10]; set grid; set xlabel
   'x'; set ylabel 'f(x)'; plot (x**2-1)/(x**3+1) with
   lines lw 2 title 'f(x)'"
```

d) Un archivo `comandos.gnu` reproducible puede contener:

```
set terminal pngcairo size 800,500
set output 'cuadratica.png'
set xrange [0:10]
set xlabel 'x'
set ylabel 'x^2-1'
set title 'Funcion x^2-1'
set grid
plot x**2-1 with lines lw 2 title 'x^2-1'
unset output
```

e) Se ejecuta directamente desde la terminal:

```
gnuplot comandos.gnu
```

Después debe comprobarse que exista `cuadratica.png`.

f) La entrada estándar de Gnuplot puede alimentarse desde una tubería del shell:

```
echo "set grid; plot sin(x)" | gnuplot -p
```

Aquí no se crea un archivo `.gnu`: la orden viaja por la entrada estándar del proceso.

g) En GNU/Linux, por ejemplo, pueden utilizarse:

```
xdg-open seno.png
display seno.png
```

El primer comando normalmente delega la apertura al visor predeterminado; el segundo requiere ImageMagick.

h) La opción `-p` o `-persist` mantiene la ventana gráfica cuando termina la entrada que alimentaba a Gnuplot. Por ejemplo:

```
gnuplot -p -e "plot sin(x)"
```

i) Las modalidades cumplen objetivos diferentes:

- la sesión interactiva es apropiada para explorar y modificar órdenes a mano;
- `-e` es útil para órdenes cortas desde el shell;
- una tubería es natural cuando otro proceso genera las órdenes;
- un archivo `.gnu` es la alternativa más clara para una figura compleja y reproducible, porque conserva explícitamente toda la especificación gráfica;
- `-p` afecta la persistencia de la ventana, no la reproducibilidad de la especificación.

3. **Ejercicio:** Importación de archivos de datos

Cree un archivo de texto `datos.txt` con los pares

```
1 3
3 4
4 5
```

Luego:

a) represente la segunda columna en función de la primera utilizando líneas y verifique explícitamente que la herramienta interpreta correctamente qué columna corresponde a cada eje;

b) repita la representación utilizando solamente puntos;

c) combine líneas y marcadores;

d) modifique uno de los valores del archivo y vuelva a generar la figura sin cambiar el programa o script de graficación;

e) explique la ventaja de separar los datos del archivo que contiene las órdenes gráficas;

f) si realiza la prueba en una sesión interactiva, ciérrela de manera controlada una vez comprobada la lectura del archivo.

Nota Gnuplot: Una forma de crear y representar el archivo es:

```
cat > /tmp/datos.txt << 'EOF'
1 3
3 4
4 5
```

```
EOF
gnuplot -p
```

```
plot '/tmp/datos.txt' with lines
quit
```

Nota Gnuplot: Con `-p` la ventana puede permanecer abierta después de terminar la sesión que la creó.

Respuesta (Gnuplot).

Guarde primero los datos exactamente como se indica en `datos.txt`. La selección explícita de columnas se hace con `using 1:2`: la primera columna proporciona las abscisas y la segunda las ordenadas.

```
set terminal pngcairo size 700,450
set xlabel 'x'
set ylabel 'y'
set grid

set output 'datos_lineas.png'
plot 'datos.txt' using 1:2 with lines lw 2 title 'lineas'

set output 'datos_puntos.png'
plot 'datos.txt' using 1:2 with points pt 7 ps 1.4 title
   'puntos'

set output 'datos_lineas_puntos.png'
plot 'datos.txt' using 1:2 with linespoints lw 2 pt 7 ps 1.2
   \
     title 'lineas y puntos'
unset output
```

Al ejecutar este mismo script se obtienen tres representaciones diferentes sin modificar los datos:

```
gnuplot datos.gnu
```

La interpretación de las columnas puede comprobarse directamente: los puntos representados deben ser $(1, 3)$, $(3, 4)$ y $(4, 5)$. Si se usara `using 2:1`, los ejes quedarían intercambiados.

Modifique ahora, por ejemplo, la última línea de `datos.txt`:

```
4 8
```

y vuelva a ejecutar exactamente el mismo archivo `datos.gnu`. El último punto pasará de $(4, 5)$ a $(4, 8)$ sin cambiar ninguna orden gráfica.

La separación entre datos y especificación tiene dos ventajas principales: el mismo script puede aplicarse a datos nuevos y los mismos datos pueden representarse con especificaciones diferentes. Además, un cambio numérico no obliga a modificar el programa que define títulos, estilos, ejes o formato de salida.

Si se realiza la prueba de forma interactiva:

```
plot 'datos.txt' using 1:2 with linespoints
quit
```

`quit` termina de manera controlada la sesión después de comprobar la lectura del archivo.

4. **Ejercicio:** Invocación de Gnuplot desde lenguajes

   Prepare previamente un archivo `grafico.gnu` que genere una figura PNG. Desde cada uno de los lenguajes de trabajo del curso, ejecute Gnuplot como programa externo y realice las siguientes tareas:

   a) invoque Gnuplot pasándole `grafico.gnu`;

   b) obtenga y compruebe el estado de terminación del proceso;

   c) capture o redirija, cuando corresponda, la salida estándar y el error estándar;

   d) verifique desde el programa que el archivo gráfico esperado haya sido creado;

   e) abra opcionalmente la figura con un visor externo y compare una llamada síncrona, que espere al cierre del visor, con una llamada que permita al programa continuar;

   f) compruebe que el programa pueda continuar su ejecución después de generar la figura cuando no sea necesario esperar al visor;

   g) provoque deliberadamente un error sencillo en una copia del script y compruebe que el programa detecte el fallo de Gnuplot.

   Nota Gnuplot: Para esta modalidad conviene invocar Gnuplot con un script, por ejemplo `/tmp/ej.gnu`, en lugar de intentar controlar una sesión interactiva desde el programa.

   Scilab. En Scilab, la ejecución de programas externos puede hacerse con `host`:

```
host('gnuplot -p /tmp/ordenes.gnu')
host('gnuplot -p /tmp/ej.gnu')
host('display /tmp/ej.png')
```

   Scilab. Una llamada síncrona a `display` detiene la ejecución hasta cerrar la ventana. Si se desea continuar, puede lanzarse el visor en segundo plano:

```
host('gnuplot -p /tmp/ej.gnu')
host('display /tmp/ej.png&')
```

   Respuesta (LG: cinco lenguajes).

   Para las cinco variantes se usa el mismo archivo `grafico.gnu`:

```
set terminal pngcairo size 700,450
set output 'figura_invocacion.png'
set title 'Gnuplot invocado desde un lenguaje'
set xlabel 'x'
```

```
set ylabel 'sin(x)'
set grid
plot sin(x) with lines lw 2 title 'sin(x)'
unset output
```

Para comprobar el tratamiento de errores puede prepararse una copia `grafico_error.gnu` que contenga deliberadamente una orden inexistente:

```
set terminal pngcairo
comando_que_no_existe
```

Los ejemplos siguientes suponen GNU/Linux para la comparación opcional con el visor `display`. La generación de la figura no depende de ese visor.

Scilab + Gnuplot

```
archivo = 'figura_invocacion.png'

[estado, salida, error_sistema] = host('gnuplot grafico.gnu')
if estado <> 0 then
    mfprintf(0, 'Gnuplot termino con codigo %d\n', estado)
    mfprintf(0, '%s\n', error_sistema)
    error('Fallo Gnuplot')
end

if ~isfile(archivo) then
    error('Gnuplot termino pero no creo la figura')
end

printf('Figura creada; el programa continua.\n')

// Visor sincrono: esta llamada espera al cierre de display.
// host('display figura_invocacion.png')

// Visor asincrono: el programa puede continuar.
// host('display figura_invocacion.png >/dev/null 2>&1 &')

[estado2, salida2, error2] = host('gnuplot
   grafico_error.gnu')
if estado2 == 0 then
    warning('Se esperaba un error y no fue informado por el
       estado')
else
    printf('Error detectado correctamente: codigo %d\n',
       estado2)
end
```

GNU Octave + Gnuplot

```
archivo = "figura_invocacion.png";

[estado, salida] = system("gnuplot grafico.gnu
   2>gnuplot.err");
```

```
error_sistema = fileread("gnuplot.err");

if estado != 0
  fprintf(2, "Gnuplot termino con codigo %d\n%s\n", estado,
     error_sistema);
  error("Fallo Gnuplot");
endif

if exist(archivo, "file") != 2
  error("Gnuplot termino pero no creo la figura");
endif

printf("Figura creada; el programa continua.\n");

% Visor sincrono:
% system("display figura_invocacion.png");
% Visor asincrono:
% system("display figura_invocacion.png >/dev/null 2>&1 &");

[estado2, salida2] = system("gnuplot grafico_error.gnu
   2>gnuplot_error.err");
if estado2 == 0
  warning("Se esperaba un error de Gnuplot");
else
  printf("Error detectado correctamente: codigo %d\n",
     estado2);
endif
```

Python + Gnuplot

```
from pathlib import Path
import subprocess

archivo = Path("figura_invocacion.png")

r = subprocess.run(
    ["gnuplot", "grafico.gnu"],
    capture_output=True,
    text=True,
)
if r.returncode != 0:
    raise RuntimeError(
        f"Gnuplot termino con codigo
           {r.returncode}\n{r.stderr}"
    )
if not archivo.is_file():
    raise RuntimeError("Gnuplot termino pero no creo la
       figura")

print("Figura creada; el programa continua.")

# Visor sincrono:
```

```
# subprocess.run(["display", str(archivo)], check=False)
# Visor asincrono:
# subprocess.Popen(
#     ["display", str(archivo)],
#     stdout=subprocess.DEVNULL,
#     stderr=subprocess.DEVNULL,
# )

malo = subprocess.run(
    ["gnuplot", "grafico_error.gnu"],
    capture_output=True,
    text=True,
)
if malo.returncode == 0:
    raise RuntimeError("Se esperaba un error de Gnuplot")
print("Error detectado correctamente:", malo.returncode)
```

Perl + Gnuplot

```
use strict;
use warnings;

my $archivo = 'figura_invocacion.png';

system('gnuplot grafico.gnu >gnuplot.out 2>gnuplot.err');
my $estado = $? >> 8;

if ($estado != 0) {
    open my $ef, '<', 'gnuplot.err' or die $!;
    local $/;
    my $err = <$ef>;
    die "Gnuplot termino con codigo $estado\n$err\n";
}
-d $archivo and die "$archivo es un directorio\n";
(-s $archivo) or die "Gnuplot no creo una figura no vacia\n";

print "Figura creada; el programa continua.\n";

# Visor sincrono:
# system('display', $archivo);
# Visor asincrono:
# system("display '$archivo' >/dev/null 2>&1 &");

system('gnuplot grafico_error.gnu >gnuplot_error.out
   2>gnuplot_error.err');
my $estado2 = $? >> 8;
die "Se esperaba un error de Gnuplot\n" if $estado2 == 0;
print "Error detectado correctamente: codigo $estado2\n";
```

JavaScript/Node.js + Gnuplot

```
"use strict";
```

```
const fs = require("node:fs");
const { spawn, spawnSync } = require("node:child_process");

const archivo = "figura_invocacion.png";

const r = spawnSync("gnuplot", ["grafico.gnu"], {
  encoding: "utf8"
});

if (r.error) throw r.error;
if (r.status !== 0) {
  throw new Error(`Gnuplot termino con codigo
     ${r.status}\n${r.stderr}`);
}
if (!fs.existsSync(archivo) || fs.statSync(archivo).size ===
   0) {
  throw new Error("Gnuplot no creo una figura no vacia");
}

console.log("Figura creada; el programa continua.");

// Visor sincrono:
// spawnSync("display", [archivo], { stdio: "ignore" });
// Visor asincrono:
// const visor = spawn("display", [archivo], {
//   detached: true, stdio: "ignore"
// });
// visor.unref();

const malo = spawnSync("gnuplot", ["grafico_error.gnu"], {
  encoding: "utf8"
});
if (malo.status === 0) {
  throw new Error("Se esperaba un error de Gnuplot");
}
console.log("Error detectado correctamente:", malo.status);
```

En las cinco variantes la llamada síncrona a Gnuplot es conveniente porque el programa necesita saber si la figura terminó de generarse antes de comprobar el archivo. La asincronía sólo se introduce, opcionalmente, al abrir el visor.

5. **Ejercicio:** Construcción automática de órdenes para Gnuplot

   Escriba, en el lenguaje de trabajo, un programa que construya automáticamente un archivo de órdenes de Gnuplot y lo ejecute.

   El programa deberá:

   a) calcular una tabla de valores de

$$f(x) = \frac{x^2 - 1}{x^3 + 1}$$

en un intervalo que no contenga puntos no definidos, y guardar los pares $(x, f(x))$ en un archivo de texto;

b) construir como texto las órdenes necesarias para generar un PNG con título, nombres de ejes, cuadrícula y una curva continua;

c) escribir dichas órdenes en un archivo con extensión `.gnu`;

d) ejecutar Gnuplot desde el propio programa;

e) comprobar el estado de terminación y la existencia del PNG generado;

f) modificar desde el programa al menos una propiedad gráfica, por ejemplo el título, el rango, el estilo o el nombre del archivo de salida, regenerar el script y producir una segunda figura;

g) comparar la generación completa del archivo de órdenes con el uso de una plantilla en la que se reemplazan cadenas, e indicar cuándo resulta más conveniente cada estrategia.

Nota Gnuplot: Los archivos de órdenes de Gnuplot son archivos de texto; una extensión habitual es `.gnu`.

Scilab. En Scilab puede construirse el texto con cadenas y escribirlo con `mopen`, `mfprintf` y `mclose`:

```
NL = ascii(10); Q = ascii(34)
gp = 'set terminal png size 700,400' + NL
gp = gp + 'set output ' + Q + '/tmp/ejb.png' + Q + NL
gp = gp + 'set xrange [0:10]' + NL
gp = gp + 'set xlabel ' + Q + 'x' + Q + NL
gp = gp + 'set ylabel ' + Q + 'y' + Q + NL
gp = gp + 'set title ' + Q + '(x^2-1)/(x^3+1)' + Q + NL
gp = gp + 'set grid' + NL
gp = gp + 'plot ' + Q + '/tmp/datos.txt' + Q + ..
          ' with lines lw 2 notitle' + NL
fd = mopen('/tmp/ejb.gnu','w')
mfprintf(fd,'%s',gp)
mclose(fd)
host('gnuplot /tmp/ejb.gnu')
```

Scilab. Para modificar una plantilla puede utilizarse:

```
strsubst(cadena, viejo, nuevo)
```

Scilab. `strsubst` reemplaza todas las apariciones de `viejo` por `nuevo` dentro de `cadena`.

Respuesta (LG: cinco lenguajes).

Se elige el intervalo $0 \le x \le 10$, que no contiene la singularidad $x = -1$. En todas las variantes se generan 201 pares $(x, f(x))$, se escribe `datos_auto.txt`, se construye `auto.gnu`, se ejecuta Gnuplot y se repite la operación cambiando título, estilo y archivo de salida.

Scilab + Gnuplot

```
x = linspace(0, 10, 201)
y = (x.^2 - 1) ./ (x.^3 + 1)
fprintfMat('datos_auto.txt', [x(:) y(:)], '%15.10g')

function crear_script(titulo, salida_png, estilo)
    fd = mopen('auto.gnu', 'w')
    mfprintf(fd, "set terminal pngcairo size 800,500\n")
    mfprintf(fd, "set output '%s'\n", salida_png)
    mfprintf(fd, "set xrange [0:10]\n")
    mfprintf(fd, "set xlabel 'x'\nset ylabel 'f(x)'\n")
    mfprintf(fd, "set title '%s'\nset grid\n", titulo)
    mfprintf(fd, "plot 'datos_auto.txt' using 1:2 %s title
       'f(x)'\n", estilo)
    mfprintf(fd, "unset output\n")
    mclose(fd)
endfunction

crear_script('Curva original', 'auto_1.png', "with lines lw
   2 lc 'blue'")
[e1, s1, er1] = host('gnuplot auto.gnu')
if e1 <> 0 | ~isfile('auto_1.png') then error(er1), end

crear_script('Curva modificada', 'auto_2.png', ..
             "with linespoints lw 2 pt 7 lc 'red'")
[e2, s2, er2] = host('gnuplot auto.gnu')
if e2 <> 0 | ~isfile('auto_2.png') then error(er2), end
```

GNU Octave + Gnuplot

```
x = linspace(0, 10, 201)';
y = (x.^2 - 1) ./ (x.^3 + 1);
datos = [x y];
save -ascii datos_auto.txt datos

 titulos = {"Curva original", "Curva modificada"};
 salidas = {"auto_1.png", "auto_2.png"};
 estilos = {"with lines lw 2 lc 'blue'", ...
            "with linespoints lw 2 pt 7 lc 'red'"};

for k = 1:2
  fid = fopen("auto.gnu", "w");
  fprintf(fid, "set terminal pngcairo size 800,500\n");
  fprintf(fid, "set output '%s'\n", salidas{k});
  fprintf(fid, "set xrange [0:10]\nset xlabel 'x'\nset
     ylabel 'f(x)'\n");
  fprintf(fid, "set title '%s'\nset grid\n", titulos{k});
  fprintf(fid, "plot 'datos_auto.txt' using 1:2 %s title
     'f(x)'\n", estilos{k});
  fprintf(fid, "unset output\n");
  fclose(fid);
```

```
  [estado, salida] = system("gnuplot auto.gnu
     2>gnuplot.err");
  if estado != 0 || exist(salidas{k}, "file") != 2
    error("Fallo al generar %s", salidas{k});
  endif
endfor
```

Python + Gnuplot

```
from pathlib import Path
import math
import subprocess

xs = [10.0*i/200 for i in range(201)]
with open("datos_auto.txt", "w", encoding="utf-8") as f:
    for x in xs:
        y = (x*x - 1.0)/(x*x*x + 1.0)
        f.write(f"{x:.12g} {y:.12g}\n")

casos = [
    ("Curva original", "auto_1.png", "with lines lw 2 lc
       'blue'"),
    ("Curva modificada", "auto_2.png",
     "with linespoints lw 2 pt 7 lc 'red'"),
]

for titulo, salida, estilo in casos:
    script = f"""set terminal pngcairo size 800,500
set output '{salida}'
set xrange [0:10]
set xlabel 'x'
set ylabel 'f(x)'
set title '{titulo}'
set grid
plot 'datos_auto.txt' using 1:2 {estilo} title 'f(x)'
unset output
"""
    Path("auto.gnu").write_text(script, encoding="utf-8")
    r = subprocess.run(["gnuplot", "auto.gnu"],
                       capture_output=True, text=True)
    if r.returncode != 0:
        raise RuntimeError(r.stderr)
    if not Path(salida).is_file():
        raise RuntimeError(f"No se genero {salida}")
```

Perl + Gnuplot

```
use strict;
use warnings;

open my $df, '>', 'datos_auto.txt' or die $!;
for my $i (0..200) {
```

```
    my $x = 10.0*$i/200.0;
    my $y = ($x*$x - 1.0)/($x*$x*$x + 1.0);
    printf $df "%.12g %.12g\n", $x, $y;
}
close $df;

my @casos = (
  ['Curva original',   'auto_1.png', "with lines lw 2 lc
     'blue'"],
  ['Curva modificada', 'auto_2.png', "with linespoints lw 2
     pt 7 lc 'red'"]
);

for my $c (@casos) {
    my ($titulo, $salida, $estilo) = @$c;
    open my $gp, '>', 'auto.gnu' or die $!;
    print $gp "set terminal pngcairo size 800,500\n";
    print $gp "set output '$salida'\n";
    print $gp "set xrange [0:10]\nset xlabel 'x'\nset ylabel
       'f(x)'\n";
    print $gp "set title '$titulo'\nset grid\n";
    print $gp "plot 'datos_auto.txt' using 1:2 $estilo title
       'f(x)'\n";
    print $gp "unset output\n";
    close $gp;

    system('gnuplot', 'auto.gnu');
    my $estado = $? >> 8;
    die "Fallo Gnuplot: $estado\n" if $estado != 0;
    (-s $salida) or die "No se genero $salida\n";
}
```

JavaScript/Node.js + Gnuplot

```
"use strict";

const fs = require("node:fs");
const { spawnSync } = require("node:child_process");

let datos = "";
for (let i = 0; i <= 200; ++i) {
  const x = 10*i/200;
  const y = (x*x - 1)/(x*x*x + 1);
  datos += `${x} ${y}\n`;
}
fs.writeFileSync("datos_auto.txt", datos);

const casos = [
  ["Curva original", "auto_1.png", "with lines lw 2 lc
     'blue'"],
  ["Curva modificada", "auto_2.png",
   "with linespoints lw 2 pt 7 lc 'red'"]
```

```
];

for (const [titulo, salida, estilo] of casos) {
  const script = `set terminal pngcairo size 800,500
set output '${salida}'
set xrange [0:10]
set xlabel 'x'
set ylabel 'f(x)'
set title '${titulo}'
set grid
plot 'datos_auto.txt' using 1:2 ${estilo} title 'f(x)'
unset output
`;
  fs.writeFileSync("auto.gnu", script);
  const r = spawnSync("gnuplot", ["auto.gnu"], { encoding:
     "utf8" });
  if (r.error) throw r.error;
  if (r.status !== 0) throw new Error(r.stderr);
  if (!fs.existsSync(salida) || fs.statSync(salida).size ===
     0) {
    throw new Error(`No se genero ${salida}`);
  }
}
```

Generación completa frente a plantilla. En los ejemplos anteriores el programa genera todas las órdenes. Este método es conveniente cuando la estructura del gráfico cambia mucho o depende de la lógica del cálculo. Una plantilla resulta preferible cuando la mayor parte del script es estable y sólo cambian unos pocos campos, por ejemplo `@TITULO@`, `@SALIDA@` y `@ESTILO@`: el programa lee la plantilla, sustituye esos marcadores y escribe el `.gnu` final. La plantilla suele ser más legible y puede editarse o probarse directamente en Gnuplot; la generación completa ofrece mayor libertad pero mezcla más sintaxis gráfica dentro del programa principal.

6. **Ejercicio:** Comunicación persistente con Gnuplot mediante una tubería

   Este ejercicio estudia una modalidad diferente de ejecutar un archivo `.gnu`: el programa debe iniciar **una sola instancia** de Gnuplot y mantener abierta una tubería hacia su entrada estándar para enviarle sucesivas órdenes y datos durante la ejecución.

   Realícelo con un lenguaje cuyo entorno proporcione de forma directa una tubería de escritura hacia un proceso hijo. Entre los lenguajes del curso esta modalidad está disponible en GNU Octave, Python, Perl y JavaScript/Node.js.

   a) Inicie un proceso Gnuplot desde el programa y conserve el objeto, descriptor o flujo que permita escribir en su entrada estándar. No cree para esta parte un archivo `.gnu` intermedio.

   b) Envíe por la tubería las órdenes necesarias para fijar título, nombres de ejes, cuadrícula y límites adecuados.

c) Envíe la orden

```
plot '-' with lines
```

y, por la misma tubería, al menos 100 pares $(x, \sin x)$. Termine el bloque de datos con una línea que contenga solamente `e`.

d) Fuerce el envío de los datos cuando la API o el sistema de búferes del lenguaje lo requiera y explique qué problema aparece si las órdenes permanecen en el búfer del programa.

e) Sin cerrar Gnuplot, envíe una segunda orden `plot` con datos nuevos, por ejemplo $(x, \cos x)$, o modifique una propiedad visible antes de volver a graficar. Compruebe que se reutilizó el mismo proceso Gnuplot.

f) Utilizando todavía la misma tubería, seleccione un terminal PNG, un archivo de salida y genere una figura final. Verifique desde el programa que el archivo haya sido creado.

g) Cierre ordenadamente la entrada estándar de Gnuplot, espere la terminación del proceso cuando corresponda y compruebe su estado de salida.

h) Compare esta solución con el ejercicio anterior, en el que se construye un archivo `.gnu`. Indique ventajas y desventajas de ambas modalidades para:

   I) reproducibilidad;
   II) actualización durante un cálculo largo;
   III) envío de datos que sólo existen en memoria;
   IV) diagnóstico de errores;
   V) simplicidad del programa.

Octave. Puede abrir Gnuplot para escritura con `popen("gnuplot -persist","w")`, escribir con `fputs`/`fprintf`, vaciar el flujo con `fflush` y cerrarlo con `pclose`.

Python. Use `subprocess.Popen` con `stdin=subprocess.PIPE` y modo texto. Escriba sobre `p.stdin`, use `flush()` cuando necesite que Gnuplot reciba inmediatamente lo acumulado y cierre el flujo antes de `wait()`.

Perl. Puede abrir una tubería de escritura con `open(my $gp,"|-","gnuplot",persist")` y activar vaciado automático del handle, o utilizar una interfaz IPC equivalente.

JavaScript. Use `child_process.spawn`; la propiedad `stdin` del proceso hijo es un flujo escribible. Envíe las órdenes con `write()` y cierre el canal con `end()`.

Scilab. La API estándar utilizada en este curso no documenta una operación equivalente a `popen(...,"w")` para mantener una tubería de escritura hacia un proceso hijo. En Scilab no se exige implementar esta versión del ejercicio: reproduzca la primera figura mediante el método por lotes con `host("gnuplot archivo.gnu")` y explique la diferencia conceptual. No use `mopen('|gnuplot','w')` como si fuera una operación documentada de la versión actual.

Respuesta (lenguaje + Gnuplot mediante tubería).

En las cuatro implementaciones del curso que ofrecen una tubería de escritura y en C se mantiene **el mismo proceso Gnuplot** durante las tres representaciones. Se utilizan 120 puntos en $0 \leq x \leq 2\pi$. El tercer dibujo se envía al terminal PNG y queda guardado en `pipe_final.png`.

GNU Octave + Gnuplot

```
gp = popen("gnuplot -persist", "w");
if gp < 0
  error("No se pudo iniciar Gnuplot");
endif

x = linspace(0, 2*pi, 120);
fputs(gp, "set xrange [0:2*pi]\nset yrange [-1.2:1.2]\n");
fputs(gp, "set xlabel 'x'\nset ylabel 'y'\nset grid\n");
fputs(gp, "set title 'Seno por tuberia'\n");
fputs(gp, "plot '-' with lines lw 2 title 'sin(x)'\n");
for k = 1:numel(x)
  fprintf(gp, "%.12g %.12g\n", x(k), sin(x(k)));
endfor
fputs(gp, "e\n");
fflush(gp);

fputs(gp, "set title 'Coseno, mismo proceso'\n");
fputs(gp, "plot '-' with lines lw 2 title 'cos(x)'\n");
for k = 1:numel(x)
  fprintf(gp, "%.12g %.12g\n", x(k), cos(x(k)));
endfor
fputs(gp, "e\n");
fflush(gp);

fputs(gp, "set terminal pngcairo size 800,500\n");
fputs(gp, "set output 'pipe_final.png'\n");
fputs(gp, "set title 'Salida final por la misma tuberia'\n");
fputs(gp, "plot '-' with lines lw 2 title 'sin(x)'\n");
for k = 1:numel(x)
  fprintf(gp, "%.12g %.12g\n", x(k), sin(x(k)));
endfor
fputs(gp, "e\nunset output\n");
fflush(gp);

estado = pclose(gp);
if estado != 0 || exist("pipe_final.png", "file") != 2
  error("Fallo la ejecucion persistente de Gnuplot");
endif
```

Python + Gnuplot

```
from pathlib import Path
import math
import subprocess
```

```
p = subprocess.Popen(
    ["gnuplot", "-persist"],
    stdin=subprocess.PIPE,
    text=True,
)
assert p.stdin is not None
x = [2*math.pi*i/119 for i in range(120)]

p.stdin.write("set xrange [0:2*pi]\nset yrange [-1.2:1.2]\n")
p.stdin.write("set xlabel 'x'\nset ylabel 'y'\nset grid\n")
p.stdin.write("set title 'Seno por tuberia'\n")
p.stdin.write("plot '-' with lines lw 2 title 'sin(x)'\n")
for xi in x:
    p.stdin.write(f"{xi:.12g} {math.sin(xi):.12g}\n")
p.stdin.write("e\n")
p.stdin.flush()

p.stdin.write("set title 'Coseno, mismo proceso'\n")
p.stdin.write("plot '-' with lines lw 2 title 'cos(x)'\n")
for xi in x:
    p.stdin.write(f"{xi:.12g} {math.cos(xi):.12g}\n")
p.stdin.write("e\n")
p.stdin.flush()

p.stdin.write("set terminal pngcairo size 800,500\n")
p.stdin.write("set output 'pipe_final.png'\n")
p.stdin.write("set title 'Salida final por la misma
   tuberia'\n")
p.stdin.write("plot '-' with lines lw 2 title 'sin(x)'\n")
for xi in x:
    p.stdin.write(f"{xi:.12g} {math.sin(xi):.12g}\n")
p.stdin.write("e\nunset output\n")
p.stdin.flush()

p.stdin.close()
estado = p.wait()
if estado != 0:
    raise RuntimeError(f"Gnuplot termino con codigo
       {estado}")
if not Path("pipe_final.png").is_file():
    raise RuntimeError("No se genero pipe_final.png")
```

Perl + Gnuplot

```
use strict;
use warnings;
use IO::Handle;

open(my $gp, '|-', 'gnuplot', '-persist')
    or die "No se pudo iniciar Gnuplot: $!\n";
$gp->autoflush(1);
```

```
my $pi = 4*atan2(1,1);
my @x = map { 2*$pi*$_/119 } 0..119;

print $gp "set xrange [0:2*pi]\nset yrange [-1.2:1.2]\n";
print $gp "set xlabel 'x'\nset ylabel 'y'\nset grid\n";
print $gp "set title 'Seno por tuberia'\n";
print $gp "plot '-' with lines lw 2 title 'sin(x)'\n";
for my $x (@x) { printf $gp "%.12g %.12g\n", $x, sin($x); }
print $gp "e\n";

print $gp "set title 'Coseno, mismo proceso'\n";
print $gp "plot '-' with lines lw 2 title 'cos(x)'\n";
for my $x (@x) { printf $gp "%.12g %.12g\n", $x, cos($x); }
print $gp "e\n";

print $gp "set terminal pngcairo size 800,500\n";
print $gp "set output 'pipe_final.png'\n";
print $gp "set title 'Salida final por la misma tuberia'\n";
print $gp "plot '-' with lines lw 2 title 'sin(x)'\n";
for my $x (@x) { printf $gp "%.12g %.12g\n", $x, sin($x); }
print $gp "e\nunset output\n";

close($gp);
my $estado = $? >> 8;
die "Gnuplot termino con codigo $estado\n" if $estado != 0;
(-s 'pipe_final.png') or die "No se genero pipe_final.png\n";
```

JavaScript/Node.js + Gnuplot

```
"use strict";

const fs = require("node:fs");
const { spawn } = require("node:child_process");

const gp = spawn("gnuplot", ["-persist"], {
  stdio: ["pipe", "inherit", "inherit"]
});

const x = Array.from({length: 120}, (_, i) =>
   2*Math.PI*i/119);
function serie(nombre, funcion) {
  gp.stdin.write(`set title '${nombre}'\n`);
  gp.stdin.write(`plot '-' with lines lw 2 title
     '${nombre}'\n`);
  for (const xi of x) gp.stdin.write(`${xi}
     ${funcion(xi)}\n`);
  gp.stdin.write("e\n");
}

gp.stdin.write("set xrange [0:2*pi]\nset yrange
   [-1.2:1.2]\n");
```

```
gp.stdin.write("set xlabel 'x'\nset ylabel 'y'\nset grid\n");
serie("sin(x)", Math.sin);
serie("cos(x)", Math.cos);

gp.stdin.write("set terminal pngcairo size 800,500\n");
gp.stdin.write("set output 'pipe_final.png'\n");
serie("sin(x) final", Math.sin);
gp.stdin.write("unset output\n");
gp.stdin.end();

gp.on("close", (codigo) => {
  if (codigo !== 0) throw new Error(`Gnuplot termino con
     codigo ${codigo}`);
  if (!fs.existsSync("pipe_final.png") ||
      fs.statSync("pipe_final.png").size === 0) {
    throw new Error("No se genero pipe_final.png");
  }
  console.log("Proceso Gnuplot cerrado correctamente");
});
```

En Node.js `write()` deposita datos en el flujo. El propio sistema de streams regula el envío; para volúmenes grandes puede controlarse el valor de retorno de `write()` y esperar el evento `drain` antes de seguir produciendo datos.

C + Gnuplot. Este es el caso adicional en el que resulta natural incluir C, porque `popen` muestra directamente la relación entre un `FILE *` y la entrada estándar del proceso hijo.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/stat.h>

int main(void)
{
    FILE *gp = popen("gnuplot -persist", "w");
    if (gp == NULL) { perror("popen"); return 1; }

    const double pi = acos(-1.0);
    fprintf(gp, "set xrange [0:2*pi]\nset yrange
       [-1.2:1.2]\n");
    fprintf(gp, "set xlabel 'x'\nset ylabel 'y'\nset
       grid\n");

    fprintf(gp, "set title 'Seno por tuberia'\n");
    fprintf(gp, "plot '-' with lines lw 2 title 'sin(x)'\n");
    for (int i=0; i<120; ++i) {
        double x = 2*pi*i/119.0;
        fprintf(gp, "%.12g %.12g\n", x, sin(x));
    }
    fprintf(gp, "e\n");
```

```
    fflush(gp);

    fprintf(gp, "set title 'Coseno, mismo proceso'\n");
    fprintf(gp, "plot '-' with lines lw 2 title 'cos(x)'\n");
    for (int i=0; i<120; ++i) {
        double x = 2*pi*i/119.0;
        fprintf(gp, "%.12g %.12g\n", x, cos(x));
    }
    fprintf(gp, "e\n");
    fflush(gp);

    fprintf(gp, "set terminal pngcairo size 800,500\n");
    fprintf(gp, "set output 'pipe_final.png'\n");
    fprintf(gp, "plot '-' with lines lw 2 title 'sin(x)'\n");
    for (int i=0; i<120; ++i) {
        double x = 2*pi*i/119.0;
        fprintf(gp, "%.12g %.12g\n", x, sin(x));
    }
    fprintf(gp, "e\nunset output\n");
    fflush(gp);

    int estado = pclose(gp);
    if (estado != 0) {
        fprintf(stderr, "Gnuplot termino con estado %d\n",
           estado);
        return 2;
    }

    struct stat st;
    if (stat("pipe_final.png", &st) != 0 || st.st_size == 0)
       {
        fprintf(stderr, "No se genero pipe_final.png\n");
        return 3;
    }
    return 0;
}
```

En sistemas POSIX puede compilarse, por ejemplo, con:

```
gcc -Wall -Wextra -o pipe_gnuplot pipe_gnuplot.c -lm
./pipe_gnuplot
```

Scilab: comparación por lotes. En la API utilizada en el curso no se presenta una tubería persistente de escritura equivalente. La comparación se realiza escribiendo datos y un script, y ejecutando un proceso completo con `host`:

```
x = linspace(0, 2*%pi, 120)
fprintfMat('scilab_pipe_comparacion.txt', [x(:) sin(x(:))],
   '%15.10g')

fd = mopen('scilab_pipe_comparacion.gnu', 'w')
```

```
mfprintf(fd, "set terminal pngcairo size 800,500\n")
mfprintf(fd, "set output 'scilab_batch.png'\n")
mfprintf(fd, "set grid\nset xlabel 'x'\nset ylabel 'y'\n")
mfprintf(fd, "plot 'scilab_pipe_comparacion.txt' using 1:2
   with lines\n")
mfprintf(fd, "unset output\n")
mclose(fd)

[estado, salida, error_sistema] = ..
    host('gnuplot scilab_pipe_comparacion.gnu')
if estado <> 0 | ~isfile('scilab_batch.png') then
    error(error_sistema)
end
```

Aquí cada cambio que requiera volver a ejecutar Gnuplot inicia un proceso nuevo; en las variantes anteriores el proceso permanece abierto y recibe sucesivos comandos y datos.

Comparación de las dos arquitecturas.

Aspecto	**Archivo `.gnu` + ejecución**	**Tubería persistente**
Reproducibilidad	Muy alta: el script queda como artefacto independiente	Requiere conservar el programa que produjo la secuencia de órdenes
Actualización durante el cálculo	Requiere ejecutar nuevamente Gnuplot o regenerar archivos	Natural: se envían nuevos `plot`/`replot` al mismo proceso
Datos sólo en memoria	Normalmente se serializan antes a un archivo	Pueden enviarse directamente con `plot '-'`
Diagnóstico	Simple al ejecutar el script manualmente; fácil conservar stderr	Debe manejarse el proceso, su stderr, cierres prematuros y posibles pipes rotos
Simplicidad	Generalmente menor complejidad de programación	Más estado y sincronización, pero mayor interactividad

El vaciado del búfer es esencial cuando el lenguaje mantiene datos pendientes: si no se realiza `fflush`, `flush` o el mecanismo equivalente, Gnuplot puede permanecer esperando órdenes que el programa ya produjo pero que todavía no llegaron a su entrada estándar.

Modalidad prevista de resolución

Los ejercicios de esta unidad están formulados independientemente del lenguaje utilizado y de la herramienta gráfica elegida. En una etapa posterior cada problema se resolverá con los cinco lenguajes de trabajo del curso: Scilab, Octave, Python, Perl y JavaScript/Node.js.

Para cada lenguaje se prepararán dos soluciones:

a) **Lenguaje + Gnuplot:** el lenguaje realizará los cálculos principales, construirá los datos necesarios y utilizará Gnuplot como motor gráfico externo. Cuando corresponda, el programa deberá generar los archivos de datos y de órdenes y ejecutar Gnuplot controlando su estado de terminación.

b) **Lenguaje solamente:** los cálculos y la representación gráfica se realizarán desde el propio lenguaje y sus bibliotecas, sin utilizar Gnuplot como proceso externo.

Las consignas siguientes no prescriben comandos concretos. Salvo indicación contraria, cada gráfico deberá incluir título, identificación de los ejes, unidades cuando correspondan, leyenda cuando haya más de una serie y una cuadrícula cuando facilite la lectura. Las escalas, límites y estilos deberán elegirse de modo que la información relevante pueda observarse con claridad.

Los resultados gráficos deberán guardarse en archivos reproducibles, evitando usar capturas de pantalla como resultado principal. Cuando resulte razonable, se generará al menos una salida raster, por ejemplo PNG, y una salida vectorial, por ejemplo PDF.

7.4.3. Sección 2: Representación gráfica de funciones y datos

7. **Ejercicio:** Muestreo y estilos de una curva racional

   Considere

$$f(x) = \frac{x^2 - 1}{x^3 + 1}, \qquad 0 \leq x \leq 10.$$

   a) Genere inicialmente 50 valores de x igualmente espaciados, compruebe la cantidad y dimensión de los datos generados y represente $f(x)$ mediante una línea continua.

   b) Represente nuevamente la misma curva empleando, sucesivamente, diferentes combinaciones de color, tipo de línea y marcadores. Compare el resultado de superponer representaciones sucesivas con el de limpiar la figura antes de generar una nueva.

   c) Calcule explícitamente el vector de ordenadas y represéntelo en función del vector de abscisas. Represente luego en una misma figura $f(x)$ y $2f(x)$, utilizando estilos que permitan distinguirlas claramente, y pruebe más de una organización de los datos para producir la representación conjunta.

   d) Construya un segundo vector de 50 abscisas, pero definido en $0 \leq X \leq 5$. Represente simultáneamente $f(x)$ sobre $0 \leq x \leq 10$ y $2f(X)$ sobre $0 \leq X \leq 5$. Explique cómo se representan en una misma figura dos series cuyos dominios son diferentes. Compare además el uso de dominios propios para cada serie con la reasignación de valores de abscisa a datos ya calculados, manteniendo igual cantidad de muestras; repita con otra secuencia de abscisas de la misma longitud.

   e) Construya además una segunda discretización de $0 \leq x \leq 10$ con el doble de puntos y represente simultáneamente una curva calculada con 50 puntos y otra con 100 puntos. Explique por qué dos series de una misma figura no necesitan poseer la misma discretización del eje horizontal.

f) Compare visualmente el efecto de aumentar la cantidad de puntos sobre la suavidad aparente de la curva.

g) Para evidenciar con mayor claridad la pérdida de información por muestreo, considere además

$$q(x) = \sin(8x) + 0{,}25\sin(40x), \qquad 0 \le x \le 2\pi.$$

Represéntela usando 15, 50 y 500 puntos, manteniendo iguales los límites de los ejes, y explique qué rasgos se pierden con una discretización demasiado gruesa.

h) En una misma figura represente $q(x)$ en $[0, 2\pi]$ con 500 puntos y

$$p(x) = e^{-x/3}\cos(5x), \qquad 0 \le x \le 4\pi,$$

con 120 puntos. Use estilos y marcadores diferentes, identifique ambas curvas en la leyenda y haga explícita la cantidad de muestras usada para cada serie.

Scilab. Para datos calculados previamente puede usarse `linspace` con distinta cantidad de puntos para cada dominio.

Octave. Puede usar `linspace` con diferente número de muestras para cada curva.

Python. Puede usar `numpy.linspace` con distinto número de muestras para cada dominio.

Perl. Con PDL genere secuencias independientes con distinta cantidad de puntos para cada curva.

JavaScript. Genere arreglos independientes para cada dominio; no es necesario que las dos series compartan el mismo número de puntos.

Nota Gnuplot: En Gnuplot cada curva puede usar su propio rango y su propio archivo o conjunto de columnas; cuando los dominios no coinciden resulta natural representarlas como series independientes.

Scilab. En Scilab pueden construirse vectores de abscisas independientes con distinta cantidad de puntos, por ejemplo mediante `linspace`, y representar luego las curvas correspondientes.

Scilab. Implementación o cálculo auxiliar en Scilab:

```
// calcular y guardar dos curvas
->x = linspace(0, 10, 50)
->y = (x.^2 - 1) ./ (x.^3 + 1)

// guardar doscurvas.txt para Gnuplot, con la rutina ya
   vista.

// guardar doscurvas.txt para Gnuplot

fprintfMat('/tmp/doscurvas.txt', [x(:) y(:) 2*y(:)],
   '%15.8f')
```

```
set terminal png size 700,400
set output "/tmp/ejc1.png"
set xrange [0:10]
set grid
plot "/tmp/doscurvas.txt" using 1:2 with lines lw 2 title
   "y", \
     "/tmp/doscurvas.txt" using 1:3 with lines lw 2 title
        "2*y"
```

```
->host('gnuplot /tmp/ejc1.gnu')
->host('display /tmp/ejc1.png')
```

```
set terminal png size 700,400
set output "/tmp/ejc2.png"
set xrange [0:10]
set grid
plot "/tmp/doscurvas.txt" using 1:2 with linespoints lw 2 lc
   "red" \
     pt 7 ps 0.8 title "y", \
     "/tmp/doscurvas.txt" using 1:3 with linespoints lw 2 lc
        "blue" \
     pt 5 ps 0.8 title "2*y"
```

```
// cambiar archivo a a plot(x,y,'r-o', x,2*y,'b-s')
->host('gnuplot /tmp/ejc2.gnu')
```

```
x = linspace(0, 10, 50)
X = linspace(0,  5, 50)    // dominio reducido
y  = (x.^2 - 1) ./ (x.^3 + 1)
y2 = (X.^2 - 1) ./ (X.^3 + 1)

// dos archivos separados (uno por dominio)
   "/tmp/curva1.txt" "/tmp/curva1.txt"
fprintfMat('/tmp/curva1.txt', [x(:) y(:)],    '%12.6f')
fprintfMat('/tmp/curva2.txt', [X(:) 2*y2(:)], '%12.6f')
```

```
set terminal png size 700,400
set output "/tmp/ejd.png"
set grid
plot "/tmp/curva1.txt" using 1:2 with lines lw 2 lc "blue"
   title "y  [0,10]", \
     "/tmp/curva2.txt" using 1:2 with lines lw 2 lc "red"
        title "2*y [0,5]"
```

```
host('gnuplot /tmp/ejd.gnu')
```

Scilab. La propiedad `xdata`, que permite reutilizar un vector x diferente para una misma llamada a `plot`, no tiene un equivalente directo en Gnuplot. En Gnuplot

cada curva se representa como un par de columnas independientes; cuando los dominios no coinciden resulta natural usar archivos separados.

Scilab. Para comprobar la dimensión del muestreo puede usar `size`.

```
size(x)
```

Scilab. Para comparar superposición y limpieza de la figura, y para ensayar estilos nativos de `plot`, el archivo `eu07b` propone:

```
clf()
```

```
plot(x,(x.^2-1)./(x.^3+1))
```

(parámetros por defecto)

```
clf()
plot(x,(x.^2-1)./(x.^3+1),'r-')
```

(línea roja continua)

```
clf()
plot(x,(x.^2-1)./(x.^3+1),'b-^')
```

(línea azul con marcadores triangulares)

Scilab. Compruebe también qué sucede si no utiliza `clf()` entre un gráfico y el siguiente. Una vez calculado el vector `y`, puede graficarlo directamente contra `x`.

```
plot(x,y)
```

Scilab. Para representar simultáneamente y y $2y$, pruebe las siguientes variantes y observe las diferencias.

```
plot(x,y,x,2*y)
```

```
clf()
plot([x;x],[y;2*y])
```

```
clf()
plot([x;x],[y;2*y])

plot(x,y,'r-o', ...
     x,2*y,'b-s')
```

Scilab. Para reasignar abscisas de igual longitud puede utilizar `xdata` como propiedad global.

```
X=linspace(0,5,50)
```

```
plot(x,y,x,2*y,'xdata',X)
```

```
clf()

plot(x,y,
     x,2*y,
     'xdata',[0.5:0.5:25])
```

Respuesta (L: diez modalidades).

En este ejercicio el cálculo y la construcción explícita de las discretizaciones forman parte del objetivo, por lo que se presentan cinco soluciones con gráficos nativos y cinco soluciones en las que el lenguaje calcula los datos y Gnuplot realiza la representación.

Con 50 puntos en $[0, 10]$ se obtiene un vector de dimensión 1×50 o 50×1, según la convención del lenguaje. Las distintas curvas de una misma figura no necesitan compartir ni dominio ni cantidad de muestras: cada serie es un conjunto independiente de pares (x_i, y_i). En particular, si se reemplaza el vector de abscisas de datos ya calculados sin volver a evaluar la función, se está cambiando el significado matemático de los datos. Para representar $f(X)$ en un nuevo dominio se debe volver a calcular la función sobre X.

El aumento de 50 a 100 puntos sólo cambia la discretización, no la función. Para $q(x) = \sin(8x) + 0{,}25\sin(40x)$, 15 puntos son insuficientes para describir la componente de frecuencia alta y pueden producir una forma aparente engañosa; con 500 puntos ambas oscilaciones se distinguen claramente.

Scilab, gráficos nativos

```
function y = f(x)
    y = (x.^2 - 1) ./ (x.^3 + 1)
endfunction
function y = q(x)
    y = sin(8*x) + 0.25*sin(40*x)
endfunction
function y = p(x)
    y = exp(-x/3).*cos(5*x)
endfunction

x50 = linspace(0,10,50);    y50 = f(x50)
x100 = linspace(0,10,100); y100 = f(x100)
X = linspace(0,5,50);       yX = f(X)
mprintf('size(x50) = %d x %d, length = %d\n', ..
        size(x50,1), size(x50,2), length(x50))

scf(70); clf()
subplot(2,2,1)
plot(x50,y50,'r-o', x50,2*y50,'b-s')
legend('f(x), 50 puntos','2f(x), 50 puntos')
xgrid(); xtitle('Curva racional y estilos','x','y')
```

```
subplot(2,2,2)
plot(x50,y50,'r-', X,2*yX,'b--', x100,y100,'k:')
legend('f(x), [0,10], 50','2f(X), [0,5], 50','f(x), [0,10],
   100')
xgrid(); xtitle('Dominios y discretizaciones
   independientes','x','y')

x15=linspace(0,2*%pi,15); xq50=linspace(0,2*%pi,50);
   x500=linspace(0,2*%pi,500)
subplot(2,2,3)
plot(x15,q(x15),'r-o', xq50,q(xq50),'g--', x500,q(x500),'b-')
legend('15','50','500')
xgrid(); xtitle('Perdida de informacion por
   muestreo','x','q(x)')

xp=linspace(0,4*%pi,120)
subplot(2,2,4)
plot(x500,q(x500),'b-', xp,p(xp),'r-o')
legend('q: 500 puntos','p: 120 puntos')
xgrid(); xtitle('Series con dominios y tamanos
   distintos','x','y')
xs2png(gcf(),'ej7_scilab.png')

// Superponer llamadas sucesivas: desactivar el borrado
   automatico.
scf(71); clf(); a=gca(); a.auto_clear='off'
plot(x50,y50,'r-o'); plot(x50,2*y50,'b-s')
a.auto_clear='on'
// Con clf() antes de cada plot, la figura anterior se
   elimina.
```

GNU Octave, gráficos nativos

```
f = @(x) (x.^2-1)./(x.^3+1);
q = @(x) sin(8*x) + 0.25*sin(40*x);
p = @(x) exp(-x/3).*cos(5*x);

x50=linspace(0,10,50); y50=f(x50);
x100=linspace(0,10,100); y100=f(x100);
X=linspace(0,5,50); yX=f(X);
printf("size(x50) = %d x %d, numel = %d\n", size(x50,1),
   size(x50,2), numel(x50));

figure(70); clf;
subplot(2,2,1); plot(x50,y50,"r-o",x50,2*y50,"b-s"); grid on;
legend("f, 50","2f, 50"); title("Curva racional y estilos");

subplot(2,2,2);
   plot(x50,y50,"r-",X,2*yX,"b--",x100,y100,"k:"); grid on;
legend("f [0,10], 50","2f [0,5], 50","f [0,10], 100");
```

```
x15=linspace(0,2*pi,15); xq50=linspace(0,2*pi,50);
   x500=linspace(0,2*pi,500);
subplot(2,2,3);
   plot(x15,q(x15),"r-o",xq50,q(xq50),"g--",x500,q(x500),"b-");
grid on; legend("15","50","500"); title("Muestreo de q");

xp=linspace(0,4*pi,120);
subplot(2,2,4); plot(x500,q(x500),"b-",xp,p(xp),"r-o"); grid
   on;
legend("q: 500","p: 120"); title("Dominios independientes");
print("ej7_octave.png","-dpng","-r140");

figure(71); clf; plot(x50,y50,"r-o"); hold on;
   plot(x50,2*y50,"b-s"); hold off;
% clf antes del segundo plot elimina la primera
   representacion.
```

Python, gráficos nativos

```
import numpy as np
import matplotlib.pyplot as plt

def f(x): return (x**2-1)/(x**3+1)
def q(x): return np.sin(8*x)+0.25*np.sin(40*x)
def p(x): return np.exp(-x/3)*np.cos(5*x)

x50=np.linspace(0,10,50); y50=f(x50)
x100=np.linspace(0,10,100); y100=f(x100)
X=np.linspace(0,5,50); yX=f(X)
print(x50.shape, x50.size)

fig,ax=plt.subplots(2,2,figsize=(11,8))
ax[0,0].plot(x50,y50,'r-o',label='f, 50')
ax[0,0].plot(x50,2*y50,'b-s',label='2f, 50')
ax[0,0].grid(); ax[0,0].legend(); ax[0,0].set_title('Curva
   racional y estilos')

ax[0,1].plot(x50,y50,'r-',label='f [0,10], 50')
ax[0,1].plot(X,2*yX,'b--',label='2f [0,5], 50')
ax[0,1].plot(x100,y100,'k:',label='f [0,10], 100')
ax[0,1].grid(); ax[0,1].legend()

x15=np.linspace(0,2*np.pi,15);
   xq50=np.linspace(0,2*np.pi,50);
   x500=np.linspace(0,2*np.pi,500)
ax[1,0].plot(x15,q(x15),'r-o',label='15')
ax[1,0].plot(xq50,q(xq50),'g--',label='50')
ax[1,0].plot(x500,q(x500),'b-',label='500')
ax[1,0].grid(); ax[1,0].legend();
   ax[1,0].set_title('Muestreo de q')

xp=np.linspace(0,4*np.pi,120)
```

```
ax[1,1].plot(x500,q(x500),'b-',label='q: 500')
ax[1,1].plot(xp,p(xp),'r-o',markevery=8,label='p: 120')
ax[1,1].grid(); ax[1,1].legend();
   ax[1,1].set_title('Dominios independientes')
fig.tight_layout(); fig.savefig('ej7_python.png',dpi=140)

# Matplotlib superpone mientras se use el mismo Axes;
   ax.clear() equivale a limpiar.
```

Perl, gráficos nativos con PDL::Graphics::Simple

```
use strict; use warnings;
use PDL; use PDL::Graphics::Simple;
$ENV{PDL_SIMPLE_ENGINE} = 'prima';
sub linspace { my($a,$b,$n)=@_; return
   sequence($n)*($b-$a)/($n-1)+$a; }
sub f { my($x)=@_; return ($x**2-1)/($x**3+1); }
sub q { my($x)=@_; return sin(8*$x)+0.25*sin(40*$x); }
sub p { my($x)=@_; return exp(-$x/3)*cos(5*$x); }

my $x50=linspace(0,10,50); my $y50=f($x50);
my $x100=linspace(0,10,100); my $X=linspace(0,5,50);
print "nelem(x50) = ", $x50->nelem, "\n";

my $w=pgswin(engine=>'prima', multi=>[2,2],
   out=>'ej7_perl.ps');
$w->plot(with=>'line',$x50,$y50, with=>'points',$x50,$y50,
         with=>'line',$x50,2*$y50,
         {title=>'Curva racional y
            estilos',xlabel=>'x',ylabel=>'y'});
$w->plot(with=>'line',$x50,$y50, with=>'line',$X,2*f($X),
         with=>'line',$x100,f($x100), {title=>'Dominios
            independientes'});
my $x15=linspace(0,2*4*atan2(1,1),15);
my $xq50=linspace(0,2*4*atan2(1,1),50);
my $x500=linspace(0,2*4*atan2(1,1),500);
$w->plot(with=>'line',$x15,q($x15),
   with=>'line',$xq50,q($xq50),
         with=>'line',$x500,q($x500), {title=>'Muestreo de
            q'});
my $xp=linspace(0,4*4*atan2(1,1),120);
$w->plot(with=>'line',$x500,q($x500),
   with=>'points',$xp,p($xp),
         {title=>'q:500 y p:120'});
```

JavaScript/Node.js, gráficos nativos con Plotly.js. El programa genera un HTML autocontenido a partir de la copia local instalada con `npm install plotly.js-dist-min`; el navegador sólo actúa como backend de representación del programa JavaScript.

```
const fs=require('node:fs');
```

```
const
   plotly=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');
const
   lin=(a,b,n)=>Array.from({length:n},(_,i)=>a+(b-a)*i/(n-1));
const f=x=>(x*x-1)/(x*x*x+1);
const q=x=>Math.sin(8*x)+0.25*Math.sin(40*x);
const p=x=>Math.exp(-x/3)*Math.cos(5*x);
const
   tr=(x,fun,name,mode='lines')=>({x,y:x.map(fun),name,mode,type:'scatte
let x50=lin(0,10,50), X=lin(0,5,50), x100=lin(0,10,100);
let x15=lin(0,2*Math.PI,15), xq50=lin(0,2*Math.PI,50),
   x500=lin(0,2*Math.PI,500);
let xp=lin(0,4*Math.PI,120);
console.log('length(x50)=',x50.length);
let data=[
 [tr(x50,f,'f,50','lines+markers'),{x:x50,y:x50.map(x=>2*f(x)),name:'2f,
 [tr(x50,f,'f [0,10],50'),{x:X,y:X.map(x=>2*f(x)),name:'2f
    [0,5],50',type:'scatter'},tr(x100,f,'f,100')],
 [tr(x15,q,'15','lines+markers'),tr(xq50,q,'50'),tr(x500,q,'500')],
 [tr(x500,q,'q:500'),tr(xp,p,'p:120','lines+markers')]
];
let body=data.map((_,i)=>`<div id="p${i}"
   style="width:48%;height:420px;display:inline-block"></div>`).join('')
let
   calls=data.map((d,i)=>`Plotly.newPlot('p${i}',${JSON.stringify(d)},{t
   7 - panel
   ${i+1}',xaxis:{title:'x'},yaxis:{title:'y'}});`).join('\n');
fs.writeFileSync('ej7_node.html',`<!doctype html><meta
   charset="utf-8"><script>${plotly}</script>${body}<script>${calls}</sc
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot. Cada lenguaje genera los mismos archivos tabulares; el siguiente script utiliza series con longitudes y dominios independientes.

```
set terminal pngcairo size 1100,800
set output 'ej7_gnuplot.png'
set multiplot layout 2,2
set grid
set title 'f y 2f, 50 puntos'
plot 'r50.dat' u 1:2 w lp t 'f,50', 'r50.dat' u 1:3 w lp t
   '2f,50'
set title 'Dominios y discretizaciones independientes'
plot 'r50.dat' u 1:2 w l t 'f [0,10],50', 'r5.dat' u 1:2 w l
   t '2f [0,5],50', 'r100.dat' u 1:2 w l t 'f [0,10],100'
set title 'q: 15, 50 y 500 puntos'
plot 'q15.dat' w lp t '15', 'q50.dat' w l t '50', 'q500.dat'
   w l t '500'
set title 'q(500) y p(120)'
plot 'q500.dat' w l t 'q:500', 'p120.dat' w lp t 'p:120'
unset multiplot
```

```
unset output
```

Scilab + Gnuplot

```
function y=f(x), y=(x.^2-1)./(x.^3+1), endfunction
function y=q(x), y=sin(8*x)+0.25*sin(40*x), endfunction
function y=p(x), y=exp(-x/3).*cos(5*x), endfunction
x=linspace(0,10,50); fprintfMat('r50.dat',[x(:) f(x(:))
   2*f(x(:))],'%15.10g')
x=linspace(0,10,100); fprintfMat('r100.dat',[x(:)
   f(x(:))],'%15.10g')
x=linspace(0,5,50); fprintfMat('r5.dat',[x(:)
   2*f(x(:))],'%15.10g')
x=linspace(0,2*%pi,15); fprintfMat('q15.dat',[x(:)
   q(x(:))],'%15.10g')
x=linspace(0,2*%pi,50); fprintfMat('q50.dat',[x(:)
   q(x(:))],'%15.10g')
x=linspace(0,2*%pi,500); fprintfMat('q500.dat',[x(:)
   q(x(:))],'%15.10g')
x=linspace(0,4*%pi,120); fprintfMat('p120.dat',[x(:)
   p(x(:))],'%15.10g')
[e,o,er]=host('gnuplot ej7.gnu'); if e<>0 then error(er), end
```

GNU Octave + Gnuplot

```
f=@(x)(x.^2-1)./(x.^3+1); q=@(x)sin(8*x)+.25*sin(40*x);
   p=@(x)exp(-x/3).*cos(5*x);
x=linspace(0,10,50)'; dlmwrite('r50.dat',[x f(x)
   2*f(x)],'delimiter',' ');
x=linspace(0,10,100)'; dlmwrite('r100.dat',[x
   f(x)],'delimiter',' ');
x=linspace(0,5,50)'; dlmwrite('r5.dat',[x
   2*f(x)],'delimiter',' ');
x=linspace(0,2*pi,15)'; dlmwrite('q15.dat',[x
   q(x)],'delimiter',' ');
x=linspace(0,2*pi,50)'; dlmwrite('q50.dat',[x
   q(x)],'delimiter',' ');
x=linspace(0,2*pi,500)'; dlmwrite('q500.dat',[x
   q(x)],'delimiter',' ');
x=linspace(0,4*pi,120)'; dlmwrite('p120.dat',[x
   p(x)],'delimiter',' ');
[e,o]=system('gnuplot ej7.gnu'); if e!=0, error(o); endif
```

Python + Gnuplot

```
import numpy as np, subprocess
f=lambda x:(x**2-1)/(x**3+1); q=lambda
   x:np.sin(8*x)+.25*np.sin(40*x); p=lambda
   x:np.exp(-x/3)*np.cos(5*x)
def save(name,x,*ys):
   np.savetxt(name,np.column_stack((x,*ys)))
x=np.linspace(0,10,50); save('r50.dat',x,f(x),2*f(x))
```

```
x=np.linspace(0,10,100); save('r100.dat',x,f(x))
x=np.linspace(0,5,50); save('r5.dat',x,2*f(x))
for n in (15,50,500):
    x=np.linspace(0,2*np.pi,n); save(f'q{n}.dat',x,q(x))
x=np.linspace(0,4*np.pi,120); save('p120.dat',x,p(x))
subprocess.run(['gnuplot','ej7.gnu'],check=True)
```

Perl + Gnuplot

```
use strict; use warnings; use POSIX qw(sin cos exp);
my $pi=4*atan2(1,1);
sub f { my($x)=@_; ($x*$x-1)/($x*$x*$x+1) }
sub q { my($x)=@_; sin(8*$x)+.25*sin(40*$x) }
sub p { my($x)=@_; exp(-$x/3)*cos(5*$x) }
sub tab { my($name,$a,$b,$n,$fun,$factor)=@_; open my
   $h,'>',$name or die $!;
  for my $i(0..$n-1){my $x=$a+($b-$a)*$i/($n-1); printf $h
     "%.12g %.12g\n",$x,$factor*$fun->($x)} close $h; }
open my $h,'>','r50.dat' or die $!; for my $i(0..49){my
   $x=10*$i/49; my $y=f($x); print $h "$x $y ",2*$y,"\n"}
   close $h;
tab('r100.dat',0,10,100,\&f,1); tab('r5.dat',0,5,50,\&f,2);
tab('q15.dat',0,2*$pi,15,\&q,1);
   tab('q50.dat',0,2*$pi,50,\&q,1);
   tab('q500.dat',0,2*$pi,500,\&q,1);
   tab('p120.dat',0,4*$pi,120,\&p,1);
system('gnuplot','ej7.gnu')==0 or die 'Fallo Gnuplot';
```

JavaScript/Node.js + Gnuplot

```
const fs=require('node:fs'),
   {spawnSync}=require('node:child_process');
const f=x=>(x*x-1)/(x*x*x+1),
   q=x=>Math.sin(8*x)+.25*Math.sin(40*x),
   p=x=>Math.exp(-x/3)*Math.cos(5*x);
function tab(name,a,b,n,fun,factor=1){let s='';for(let
   i=0;i<n;i++){let x=a+(b-a)*i/(n-1);s+=`${x}
   ${factor*fun(x)}\n`;}fs.writeFileSync(name,s);}
let s='';for(let i=0;i<50;i++){let x=10*i/49,y=f(x);s+=`${x}
   ${y} ${2*y}\n`;}fs.writeFileSync('r50.dat',s);
tab('r100.dat',0,10,100,f); tab('r5.dat',0,5,50,f,2);
tab('q15.dat',0,2*Math.PI,15,q);
   tab('q50.dat',0,2*Math.PI,50,q);
   tab('q500.dat',0,2*Math.PI,500,q);
   tab('p120.dat',0,4*Math.PI,120,p);
let r=spawnSync('gnuplot',['ej7.gnu'],{encoding:'utf8'});
   if(r.status!==0) throw new Error(r.stderr);
```

8. **Ejercicio:** Comparación de dos funciones racionales

Sean

$$f(x) = \frac{x^2 - 1}{x^3 + 2}, \qquad g(x) = \frac{x^2 + 1}{x^3 + 1}, \qquad 0 \le x \le 10.$$

a) Represente cada función por separado y luego ambas en una misma figura.

b) En la figura conjunta, represente f mediante círculos rojos con borde oscuro unidos por línea continua y g mediante cuadrados amarillos con borde azul unidos por línea discontinua o punteada.

c) Incluya una leyenda que identifique las funciones y seleccione límites de ejes que permitan compararlas.

d) Guarde las tres figuras obtenidas; use, en el conjunto de salidas, al menos un formato raster y un formato vectorial. Cierre las figuras, vuelva a abrir las representaciones almacenadas y exporte al menos una de ellas en formato PostScript.

e) Explique qué ventajas presenta la figura conjunta respecto de las figuras separadas y qué información puede resultar más clara en las representaciones individuales.

Scilab. Para evaluar las funciones sobre vectores en Scilab se usan operaciones elemento a elemento, por ejemplo `(x.^2-1)./(x.^3+2)` y `(x.^2+1)./(x.^3+1)`.

Nota Gnuplot: En Gnuplot los estilos pueden controlarse con opciones de línea y marcador, por ejemplo `with linespoints`, `lc`, `dt`, `pt`, `ps` y, cuando el terminal lo admite, color de relleno.

Scilab. Implementación o cálculo auxiliar en Scilab:

```
function y = f(x); y = (x.^2 - 1) ./ (x.^3 + 2); endfunction
function y = g(x); y = (x.^2 + 1) ./ (x.^3 + 1); endfunction

x  = linspace(0, 10, 200)
yf = f(x)
yg = g(x)


// guardar datos en un archivo con dos curvas   fg.txt
fprintfMat('/tmp/fg.txt', [x(:) yf(:) yg(:)], '%15.8f')
// para evitar superposicion de numeros se usa un formato
   para numeros grande.
```

```
set terminal png size 700,400
set output "/tmp/graficaf.png"
set title  "f(x) = (x^2-1)/(x^3+2)"
set xlabel "x"
set ylabel "f(x)"
set xrange [0:10]
set grid
plot "/tmp/fg.txt" using 1:2 with lines lw 2 lc "blue"
   notitle
```

```
set terminal png size 700,400
set output "/tmp/graficag.png"
set title  "g(x) = (x^2+1)/(x^3+1)"
set xlabel "x"
set ylabel "g(x)"
set xrange [0:10]
set grid
plot "/tmp/fg.txt" using 1:3 with lines lw 2 lc "red" notitle
```

```
host('gnuplot /tmp/gf.gnu')

host('gnuplot /tmp/gg.gnu')
```

```
set terminal png size 700,400
set output "/tmp/graficafg.png"
set title  "f(x) y g(x)"
set xlabel "x"
set ylabel "y"
set xrange [0:10]
set grid
set key top right
plot "/tmp/fg.txt" using 1:2 with lines lw 2 lc "blue" title
   "f(x)=(x^2-1)/(x^3+2)", \
     "/tmp/fg.txt" using 1:3 with lines lw 2 lc "red"  title
        "g(x)=(x^2+1)/(x^3+1)"
```

```
host('gnuplot /tmp/gfg.gnu')

printf('Graficas guardadas:\n')
printf('  /tmp/graficaf.png\n')
printf('  /tmp/graficag.png\n')
printf('  /tmp/graficafg.png\n')
```

```
%# abrir las graficas guardadas (equivale a load + set
   window)
%xdg-open /tmp/graficafg.png
```

```
%
%printf('Exportado: /tmp/grafica_fg.eps\n')
%printf('Exportado: /tmp/grafica_fg.pdf\n')
```

```
// usar 30 puntos para que los marcadores sean visibles
x  = linspace(0, 10, 30)
yf = f(x)
yg = g(x)

// guardar datos en un archivo con dos curvas
//    mfprintf(fd, '%.8f %.8f %.8f\n', x(i), yf(i), yg(i))
```

```
# f: circulos rojos con borde negro, linea negra continua
# lc = color del borde y la linea
# fc = color de relleno del marcador
# pt 7 = circulo relleno
# g: cuadrados amarillos con borde azul, linea punteada azul
# dt 3 = punteada
# pt 5 = cuadrado relleno
set terminal png size 750,480
set output "/tmp/grafica_estilos.png"
set title  "f(x) y g(x) con estilos"
set xlabel "x"
set ylabel "y"
set xrange [0:10]
set grid
set key top right
plot "/tmp/fg2.txt" using 1:2 with linespoints lw 1.5 lc
   "black" \
     pt 7 ps 1.2 fc "red" title "f(x): circulo rojo / linea
        negra", \
     "/tmp/fg2.txt" using 1:3 with linespoints lw 1.5 lc
        "blue" dt 3 \
     pt 5 ps 1.2 fc "yellow" title "g(x): cuadrado amarillo
        / linea azul punteada"
```

```
host('gnuplot /tmp/g_estilos.gnu')
printf('Grafica: /tmp/grafica_estilos.png\n')
```

Scilab. Para guardar y recuperar figuras nativas puede utilizar `save` y `load`. Para crear una nueva ventana puede utilizar `set window` del menú `graphic window`; para la exportación puede utilizar `export`.

Scilab. Como alternativa nativa, las funciones pueden definirse mediante `deff` y luego representarse con `plot`.

```
deff('[y]=f(x)','y=(x.^2-1)./(x.^3+2)')
deff('[y]=g(x)','y=(x.^2+1)./(x.^3+1)')
```

```
plot(x,f,x,g)
```

Scilab. Para configurar marcadores, bordes, colores y estilos de línea en la vía nativa puede emplear las opciones de `GlobalProperty`.

Respuesta (P: persistencia de la escena; soluciones completas y comparaciones).

Las tres representaciones se obtienen con

$$f(x) = \frac{x^2 - 1}{x^3 + 2}, \qquad g(x) = \frac{x^2 + 1}{x^3 + 1}.$$

En $[0, 10]$ ambas son regulares. La figura conjunta permite comparar de manera inmediata separación, cruces y tendencia; las figuras individuales permiten utilizar límites verticales propios y apreciar variaciones pequeñas sin que la otra curva condicione la escala.

Este ejercicio exige distinguir *persistencia de escena* de simple exportación. Scilab y Octave poseen mecanismos documentados para guardar y recuperar objetos gráficos. Matplotlib permite serialización de figuras mediante `pickle` para almacenamiento de corto plazo, con importantes limitaciones de versión y seguridad. En Perl y JavaScript se conserva aquí la especificación y los datos y se reconstruye la figura; eso es reproducibilidad declarativa, no persistencia binaria del widget. Gnuplot tampoco conserva una escena editable: se conserva el script y se vuelve a ejecutar.

Scilab nativo: persistencia real de la escena

```
function y=f(x), y=(x.^2-1)./(x.^3+2), endfunction
function y=g(x), y=(x.^2+1)./(x.^3+1), endfunction
x=linspace(0,10,80); yf=f(x); yg=g(x)

scf(80); clf(); plot(x,yf,'r-o'); xgrid();
   xtitle('f(x)','x','f')
xs2png(gcf(),'ej8_f.png')

scf(81); clf(); plot(x,yg,'b--s'); xgrid();
   xtitle('g(x)','x','g')
xs2pdf(gcf(),'ej8_g.pdf')

scf(82); clf(); plot(x,yf,'r-o',x,yg,'b--s'); xgrid()
legend('f','g'); xtitle('f y g','x','y')
xsave('ej8_fg.scg',gcf())          // guarda objetos graficos
close(gcf())
scf(83); xload('ej8_fg.scg')       // recupera la escena
xs2eps(gcf(),'ej8_fg_reabierta.eps')
```

La reapertura de `ej8_fg.scg` recupera objetos gráficos; no equivale a abrir `ej8_f.png` como una imagen.

GNU Octave nativo: persistencia real mediante hgsave/hgload

```
f=@(x)(x.^2-1)./(x.^3+2); g=@(x)(x.^2+1)./(x.^3+1);
x=linspace(0,10,80); yf=f(x); yg=g(x);
figure(80); plot(x,yf,'r-o'); grid on; title('f(x)');
   print('ej8_f.png','-dpng');
figure(81); plot(x,yg,'b--s'); grid on; title('g(x)');
   print('ej8_g.pdf','-dpdf');
figure(82); plot(x,yf,'-o','color',[1 0
   0],'markerfacecolor',[1 0 0],...
                  x,yg,'--s','color',[0 0
                     1],'markerfacecolor',[1 1 0]);
grid on; legend('f','g'); title('f y g');
hgsave(gcf(),'ej8_fg.ofig'); close(gcf());
h=hgload('ej8_fg.ofig');
print(h,'ej8_fg_reabierta.eps','-depsc');
```

Python/Matplotlib nativo: persistencia de corto plazo

```
import numpy as np, matplotlib.pyplot as plt, pickle
x=np.linspace(0,10,80)
f=(x*x-1)/(x**3+2); g=(x*x+1)/(x**3+1)

fig,ax=plt.subplots(); ax.plot(x,f,'r-o'); ax.grid();
   ax.set_title('f(x)')
fig.savefig('ej8_f.png',dpi=140); plt.close(fig)
fig,ax=plt.subplots(); ax.plot(x,g,'b--s'); ax.grid();
   ax.set_title('g(x)')
fig.savefig('ej8_g.pdf'); plt.close(fig)

fig,ax=plt.subplots()
ax.plot(x,f,'-',marker='o',color='red',markeredgecolor='black',label='f'
ax.plot(x,g,'--',marker='s',color='blue',markerfacecolor='yellow',
        markeredgecolor='blue',label='g')
ax.grid(); ax.legend(); ax.set_title('f y g')
with open('ej8_fg.pickle','wb') as h: pickle.dump(fig,h)
plt.close(fig)

# Cargar solamente archivos pickle propios y con la misma
   version de Matplotlib.
with open('ej8_fg.pickle','rb') as h: fig2=pickle.load(h)
fig2.savefig('ej8_fg_reabierta.eps',format='eps')
plt.close(fig2)
```

El `pickle` no es un formato de archivo gráfico para archivo de largo plazo: depende de versiones y no debe cargarse si proviene de una fuente no confiable.

Perl nativo: reconstrucción declarativa, no serialización del widget

```
use strict; use warnings; use PDL; use
   PDL::Graphics::Prima::Simple; use Storable qw(store
   retrieve);
auto_twiddle(0);
my $x=sequence(80)*10/79;
my $f=($x**2-1)/($x**3+2); my $g=($x**2+1)/($x**3+1);
store({x=>$x->unpdl,f=>$f->unpdl,g=>$g->unpdl},'ej8_fg.stor');

my ($window,$plot)=line_plot($x,$f);
$plot->title('f y g'); $plot->x->label('x');
   $plot->y->label('y');
$plot->dataSets->{'g'}=ds::Pair($x,$g,plotType=>ppair::Squares(filled=>1
$plot->save_to_file('ej8_fg.png');
$plot->save_to_postscript('ej8_fg.eps');
$window->close;

# Reapertura conceptual: recuperar datos/especificacion y
   reconstruir el widget.
my $d=retrieve('ej8_fg.stor');
my $xr=pdl($d->{x}); my $fr=pdl($d->{f}); my
   $gr=pdl($d->{g});
```

```
my ($w2,$p2)=line_plot($xr,$fr);
$p2->dataSets->{'g'}=ds::Pair($xr,$gr,plotType=>ppair::Squares(filled=>1
$p2->save_to_postscript('ej8_fg_reconstruida.eps');
```

Aquí `Storable` conserva datos y especificación suficiente para recrear la figura; no conserva el estado interno completo del widget gráfico.

JavaScript/Node.js + Plotly.js: persistencia declarativa en JSON

```
const fs=require('node:fs');
const
   lin=(a,b,n)=>Array.from({length:n},(_,i)=>a+(b-a)*i/(n-1));
let x=lin(0,10,80), f=x.map(x=>(x*x-1)/(x*x*x+2)),
   g=x.map(x=>(x*x+1)/(x*x*x+1));
let fig={data:[
 {x,y:f,type:'scatter',mode:'lines+markers',name:'f',line:{color:'red'},
 {x,y:g,type:'scatter',mode:'lines+markers',name:'g',line:{color:'blue',
],layout:{title:'f y
   g',xaxis:{title:'x'},yaxis:{title:'y'}}};
fs.writeFileSync('ej8_fg.json',JSON.stringify(fig,null,2));
// Al reabrir, JSON.parse reconstruye data/layout y
   Plotly.newPlot vuelve a crear la figura.
let
   reabierta=JSON.parse(fs.readFileSync('ej8_fg.json','utf8'));
console.log(reabierta.layout.title);
```

Plotly.js representa la figura mediante una estructura JSON declarativa, por lo que esa estructura puede guardarse y volver a renderizarse. La exportación directa habitual es PNG/SVG; no se presenta aquí una exportación PostScript nativa porque no forma parte de la API estándar de Plotly.js. Por ello esta variante satisface la persistencia declarativa, pero no todos los formatos solicitados sin añadir un conversor externo.

Cinco variantes lenguaje + Gnuplot: regeneración reproducible. En estas variantes se conserva `fg.dat` y el siguiente script. No existe una escena Gnuplot serializada que pueda cargarse para continuar editando sus objetos: volver a "abrir" significa volver a ejecutar el script.

```
set grid; set xrange [0:10]; set xlabel 'x'; set ylabel 'y'
set terminal pngcairo size 800,500; set output 'ej8_f_gp.png'
plot 'fg.dat' u 1:2 w lp t 'f'
unset output
set terminal pdfcairo; set output 'ej8_g_gp.pdf'
plot 'fg.dat' u 1:3 w lp t 'g'
unset output
set terminal postscript eps enhanced color; set output
   'ej8_fg_gp.eps'
plot 'fg.dat' u 1:2 w lp pt 7 lc rgb 'red' t 'f', \
     'fg.dat' u 1:3 w lp pt 5 dt 2 lc rgb 'blue' t 'g'
unset output
```

```
x=linspace(0,10,80); f=(x.^2-1)./(x.^3+2);
   g=(x.^2+1)./(x.^3+1)
fprintfMat('fg.dat',[x(:) f(:) g(:)],'%15.10g');
   [e,o,er]=host('gnuplot ej8.gnu'); if e<>0 then error(er),
   end
```

```
x=linspace(0,10,80)'; f=(x.^2-1)./(x.^3+2);
   g=(x.^2+1)./(x.^3+1); dlmwrite('fg.dat',[x f
   g],'delimiter',' '); assert(system('gnuplot ej8.gnu')==0);
```

```
import numpy as np, subprocess
x=np.linspace(0,10,80);
   np.savetxt('fg.dat',np.c_[x,(x*x-1)/(x**3+2),(x*x+1)/(x**3+1)]);
   subprocess.run(['gnuplot','ej8.gnu'],check=True)
```

```
open my $h,'>','fg.dat' or die $!; for(0..79){my
   $x=10*$_/79; print $h "$x ",($x*$x-1)/($x**3+2),'
   ',($x*$x+1)/($x**3+1),"\n"} close $h;
   system('gnuplot','ej8.gnu')==0 or die;
```

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');
   let s=''; for(let i=0;i<80;i++){let x=10*i/79;s+=`${x}
   ${(x*x-1)/(x**3+2)} ${(x*x+1)/(x**3+1)}\n`;}
   fs.writeFileSync('fg.dat',s); let
   r=spawnSync('gnuplot',['ej8.gnu']); if(r.status!==0)
   throw Error('gnuplot');
```

9. **Ejercicio:** Anotación de puntos notables y funciones especiales

   Realice las siguientes representaciones, prestando especial atención a la información adicional que debe incorporarse al gráfico. Para cada figura use título, identificación de ejes, cuadrícula y leyenda cuando corresponda.

   a) Para
   $$f(x) = x^2 - 3x + 2, \qquad -1 \le x \le 4,$$
   represente la función y marque explícitamente sus dos raíces mediante puntos rojos o anotaciones equivalentes. Compruebe que las posiciones marcadas coincidan con las raíces calculadas analíticamente.

   b) Para
   $$g(x) = e^{-x}\cos(3x), \qquad 0 \le x \le 4\pi,$$
   represente la función junto con las envolventes e^{-x} y $-e^{-x}$. Explique cómo se relacionan la frecuencia de las oscilaciones y su amplitud decreciente con los distintos factores de la expresión.

   c) Represente en una misma figura
   $$\sinh x, \qquad \cosh x, \qquad \tanh x,$$

en un intervalo simétrico apropiado. Utilice una leyenda y estilos diferentes, y describa las simetrías y el comportamiento asintótico que puedan observarse.

Nota Gnuplot: Para la primera función, las raíces pueden marcarse mediante `set arrow` o agregando una segunda serie de puntos. Si se trabaja con scripts, use un archivo separado para cada figura cuando resulte conveniente.

Respuesta (T: diez modalidades).

Las raíces del polinomio son $x = 1$ y $x = 2$. En $g(x) = e^{-x}\cos(3x)$ la frecuencia angular del factor oscilatorio es 3, mientras que la amplitud queda acotada por $\pm e^{-x}$ y decrece exponencialmente. Además, $\sinh x$ es impar, $\cosh x$ es par y $\tanh x$ es impar; $\tanh x \to \pm 1$ cuando $x \to \pm\infty$.

Scilab nativo

```
x=linspace(-1,4,300); y=x.^2-3*x+2
scf(90); clf(); subplot(3,1,1); plot(x,y,'b-', [1 2],[0
   0],'ro'); xgrid(); xtitle('Raices: 1 y 2','x','f')
x=linspace(0,4*%pi,600); g=exp(-x).*cos(3*x)
subplot(3,1,2);
   plot(x,g,'b-',x,exp(-x),'r--',x,-exp(-x),'r--');
   legend('g','e^{-x}','-e^{-x}'); xgrid()
x=linspace(-4,4,400); subplot(3,1,3);
   plot(x,sinh(x),'r-',x,cosh(x),'b--',x,tanh(x),'k-.');
   legend('sinh','cosh','tanh'); xgrid()
xs2png(gcf(),'ej9_scilab.png')
```

GNU Octave nativo

```
figure(90); clf;
x=linspace(-1,4,300); subplot(3,1,1);
   plot(x,x.^2-3*x+2,'b-',1,0,'ro',2,0,'ro'); grid on;
x=linspace(0,4*pi,600); subplot(3,1,2);
   plot(x,exp(-x).*cos(3*x),'b-',x,exp(-x),'r--',x,-exp(-x),'r--');
   grid on; legend('g','e^{-x}','-e^{-x}');
x=linspace(-4,4,400); subplot(3,1,3);
   plot(x,sinh(x),'r-',x,cosh(x),'b--',x,tanh(x),'k-.');
   grid on; legend('sinh','cosh','tanh');
print('ej9_octave.png','-dpng','-r140');
```

Python nativo

```
import numpy as np, matplotlib.pyplot as plt
fig,ax=plt.subplots(3,1,figsize=(9,10))
x=np.linspace(-1,4,300); ax[0].plot(x,x*x-3*x+2);
   ax[0].scatter([1,2],[0,0],c='red'); ax[0].grid();
   ax[0].annotate('x=1',(1,0)); ax[0].annotate('x=2',(2,0))
x=np.linspace(0,4*np.pi,600);
   ax[1].plot(x,np.exp(-x)*np.cos(3*x),label='g');
   ax[1].plot(x,np.exp(-x),'--',label='e^-x');
   ax[1].plot(x,-np.exp(-x),'--',label='-e^-x');
   ax[1].grid(); ax[1].legend()
```

```
x=np.linspace(-4,4,400);
   ax[2].plot(x,np.sinh(x),label='sinh');
   ax[2].plot(x,np.cosh(x),label='cosh');
   ax[2].plot(x,np.tanh(x),label='tanh'); ax[2].grid();
   ax[2].legend()
fig.tight_layout(); fig.savefig('ej9_python.png',dpi=140)
```

Perl nativo

```
use PDL; use PDL::Graphics::Simple;
   $ENV{PDL_SIMPLE_ENGINE}='prima';
sub lin {my($a,$b,$n)=@_; sequence($n)*($b-$a)/($n-1)+$a}
my
   $w=pgswin(engine=>'prima',multi=>[3,1],out=>'ej9_perl.ps');
my $x=lin(-1,4,300);
   $w->plot(with=>'line',$x,$x**2-3*$x+2,with=>'points',pdl(1,2),pdl(0,0
   1 y 2'});
$x=lin(0,4*4*atan2(1,1),600);
   $w->plot(with=>'line',$x,exp(-$x)*cos(3*$x),with=>'line',$x,exp(-$x),
   amortiguada'});
$x=lin(-4,4,400);
   $w->plot(with=>'line',$x,sinh($x),with=>'line',$x,cosh($x),with=>'lin
   hiperbolicas'});
```

JavaScript/Node.js nativo con Plotly.js

```
const fs=require('node:fs'); const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');
const
   lin=(a,b,n)=>Array.from({length:n},(_,i)=>a+(b-a)*i/(n-1));
let
   x1=lin(-1,4,300),x2=lin(0,4*Math.PI,600),x3=lin(-4,4,400);
let figs=[
 [{x:x1,y:x1.map(x=>x*x-3*x+2),type:'scatter',mode:'lines',name:'f'},{x:
 [{x:x2,y:x2.map(x=>Math.exp(-x)*Math.cos(3*x)),name:'g',type:'scatter'}
 [{x:x3,y:x3.map(Math.sinh),name:'sinh',type:'scatter'},{x:x3,y:x3.map(M
];
let divs=figs.map((_,i)=>`<div id=p${i}
   style="height:420px"></div>`).join(''); let
   calls=figs.map((d,i)=>`Plotly.newPlot('p${i}',${JSON.stringify(d)},{t
   9.${i+1}'});`).join('');
fs.writeFileSync('ej9_node.html',`<script>${P}</script>${divs}<script>${
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot

```
set terminal pngcairo size 900,1000; set output
   'ej9_gnuplot.png'; set multiplot layout 3,1; set grid
set title 'Polinomio y raices'; plot 'quad.dat' w l t 'f',
   '-' w p pt 7 lc rgb 'red' t 'raices'
1 0
2 0
e
```

```
set title 'Oscilacion y envolventes'; plot 'damped.dat' u
   1:2 w l t 'g', '' u 1:3 w l dt 2 t 'e^-x', '' u 1:4 w l
   dt 2 t '-e^-x'
set title 'Funciones hiperbolicas'; plot 'hyp.dat' u 1:2 w l
   t 'sinh', '' u 1:3 w l t 'cosh', '' u 1:4 w l t 'tanh'
unset multiplot; unset output
```

Los cinco programas siguientes producen los tres archivos y ejecutan el mismo script.

```
x=linspace(-1,4,300); y=x.^2-3*x+2;
   fprintfMat('quad.dat',[x(:) y(:)],'%15.10g')
x=linspace(0,4*%pi,600); y=exp(-x).*cos(3*x); e=exp(-x);
   fprintfMat('damped.dat',[x(:) y(:) e(:) -e(:)],'%15.10g')
x=linspace(-4,4,400); fprintfMat('hyp.dat',[x(:) sinh(x(:))
   cosh(x(:)) tanh(x(:))],'%15.10g'); host('gnuplot ej9.gnu')
```

```
x=linspace(-1,4,300)'; dlmwrite('quad.dat',[x
   x.^2-3*x+2],'delimiter',' '); x=linspace(0,4*pi,600)';
   dlmwrite('damped.dat',[x exp(-x).*cos(3*x) exp(-x)
   -exp(-x)],'delimiter',' '); x=linspace(-4,4,400)';
   dlmwrite('hyp.dat',[x sinh(x) cosh(x)
   tanh(x)],'delimiter',' '); system('gnuplot ej9.gnu');
```

```
import numpy as np, subprocess
x=np.linspace(-1,4,300);
   np.savetxt('quad.dat',np.c_[x,x*x-3*x+2]);
   x=np.linspace(0,4*np.pi,600);
   np.savetxt('damped.dat',np.c_[x,np.exp(-x)*np.cos(3*x),np.exp(-x),-np
   x=np.linspace(-4,4,400);
   np.savetxt('hyp.dat',np.c_[x,np.sinh(x),np.cosh(x),np.tanh(x)]);
   subprocess.run(['gnuplot','ej9.gnu'],check=True)
```

```
use POSIX qw(exp sinh cosh tanh cos); sub
   tab{my($n,$a,$b,$name,$fun)=@_;open my$h,'>',$name or
   die$!;for(0..$n-1){my$x=$a+($b-$a)*$_/($n-1);print$h
   $fun->($x)}close$h} my$pi=4*atan2(1,1);
   tab(300,-1,4,'quad.dat',sub{my$x=shift;"$x
   ".($x*$x-3*$x+2)."\n"});
   tab(600,0,4*$pi,'damped.dat',sub{my$x=shift;my$e=exp(-$x);"$x
   ".($e*cos(3*$x))." $e ".(-$e)."\n"});
   tab(400,-4,4,'hyp.dat',sub{my$x=shift;"$x ".sinh($x).'
   '.cosh($x).' '.tanh($x)."\n"});
   system('gnuplot','ej9.gnu');
```

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');
   function tab(name,a,b,n,fun){let s='';for(let
   i=0;i<n;i++){let
   x=a+(b-a)*i/(n-1);s+=fun(x)}fs.writeFileSync(name,s)}
   tab('quad.dat',-1,4,300,x=>`${x} ${x*x-3*x+2}\n`);
```

```
    tab('damped.dat',0,4*Math.PI,600,x=>{let
    e=Math.exp(-x);return `${x} ${e*Math.cos(3*x)} ${e}
    ${-e}\n`}); tab('hyp.dat',-4,4,400,x=>`${x}
    ${Math.sinh(x)} ${Math.cosh(x)} ${Math.tanh(x)}\n`); let
    r=spawnSync('gnuplot',['ej9.gnu']);if(r.status!==0)throw
    Error('gnuplot');
```

10. **Ejercicio:** Trazado de una función racional simple

Considere

$$f(x) = \frac{x+1}{x+2}, \qquad 0 \le x \le 10.$$

a) Genere una discretización adecuada del intervalo y represente la función.

b) Incluya título, nombres de ejes, cuadrícula y límites que permitan observar con claridad la variación de la función.

c) Repita la representación utilizando al menos dos estilos diferentes de línea o marcadores.

d) Describa el comportamiento de la curva en el intervalo indicado y compárelo con el valor límite de la función cuando x crece.

Respuesta (T: diez modalidades).

La función puede escribirse como

$$\frac{x+1}{x+2} = 1 - \frac{1}{x+2}.$$

En $[0, 10]$ comienza en $f(0) = 1/2$, es estrictamente creciente porque $f'(x) = 1/(x+2)^2 > 0$ y se aproxima a 1 desde abajo cuando x crece.

Scilab nativo

```
x=linspace(0,10,200); y=(x+1)./(x+2); scf(100); clf();
    plot(x,y,'b-',x,y,'ro'); xgrid();
    xtitle('(x+1)/(x+2)','x','f(x)');
    xs2png(gcf(),'ej10_scilab.png')
```

GNU Octave nativo

```
x=linspace(0,10,200); y=(x+1)./(x+2);
    plot(x,y,'b-',x,y,'r.'); grid on; xlabel('x');
    ylabel('f(x)'); title('(x+1)/(x+2)');
    print('ej10_octave.png','-dpng');
```

Python nativo

```
import numpy as np, matplotlib.pyplot as plt
x=np.linspace(0,10,200); y=(x+1)/(x+2);
    plt.plot(x,y,'b-',label='linea');
    plt.plot(x[::10],y[::10],'ro',label='marcadores');
    plt.grid(); plt.xlabel('x'); plt.ylabel('f(x)');
    plt.legend(); plt.savefig('ej10_python.png',dpi=140)
```

Perl nativo

```
use PDL; use PDL::Graphics::Simple;
   $ENV{PDL_SIMPLE_ENGINE}='prima';
   my$x=sequence(200)*10/199; my$y=($x+1)/($x+2);
   my$w=pgswin(engine=>'prima',out=>'ej10_perl.ps');
   $w->plot(with=>'line',$x,$y,with=>'points',$x,$y,{title=>'(x+1)/(x+2)
```

JavaScript/Node.js nativo

```
const fs=require('node:fs'); const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');
   let
   x=Array.from({length:200},(_,i)=>10*i/199),y=x.map(x=>(x+1)/(x+2));
   fs.writeFileSync('ej10_node.html',`<script>${P}</script><div
   id=p
   style="height:500px"></div><script>Plotly.newPlot('p',${JSON.stringif
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot

```
set terminal pngcairo size 800,500; set output
   'ej10_gnuplot.png'; set xrange [0:10]; set yrange
   [0.45:1.05]; set grid; set xlabel 'x'; set ylabel 'f(x)';
   set title '(x+1)/(x+2)'; plot 'simple.dat' w l lw 2 t
   'linea', '' every 10 w p pt 7 t 'marcadores'; unset output
```

```
x=linspace(0,10,200); y=(x+1)./(x+2);
   fprintfMat('simple.dat',[x(:) y(:)],'%15.10g');
   host('gnuplot ej10.gnu')
```

```
x=linspace(0,10,200)'; y=(x+1)./(x+2);
   dlmwrite('simple.dat',[x y],'delimiter',' ');
   system('gnuplot ej10.gnu');
```

```
import numpy as np,subprocess; x=np.linspace(0,10,200);
   np.savetxt('simple.dat',np.c_[x,(x+1)/(x+2)]);
   subprocess.run(['gnuplot','ej10.gnu'],check=True)
```

```
open my$h,'>','simple.dat' or
   die$!;for(0..199){my$x=10*$_/199;print$h "$x
   ",($x+1)/($x+2),"\n"}close$h;system('gnuplot','ej10.gnu');
```

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');let
   s='';for(let i=0;i<200;i++){let x=10*i/199;s+=`${x}
   ${(x+1)/(x+2)}\n`;}fs.writeFileSync('simple.dat',s);let
   r=spawnSync('gnuplot',['ej10.gnu']);if(r.status!==0)throw
   Error('gnuplot');
```

11. **Ejercicio:** Discontinuidades y separación de ramas

Considere las funciones

$$f(x) = \frac{x^2 - 1}{x^3 + 1}, \qquad -2 \le x \le 2,$$

y

$$g(t) = t + \frac{t}{1 + \sin t}, \qquad -2\pi \le t \le 2\pi.$$

a) Determine analíticamente los puntos del intervalo en los que cada función no está definida.

b) Genere una discretización suficientemente fina y represente las funciones sin unir artificialmente puntos situados a ambos lados de una singularidad.

c) Compare al menos dos estrategias para tratar los puntos no definidos, por ejemplo eliminarlos o representarlos mediante valores ausentes.

d) Muestre las diferentes ramas con claridad y explique por qué una simple línea que atraviese una asíntota produce una representación engañosa.

e) Analice qué ocurre si uno de los puntos singulares no pertenece exactamente a la discretización elegida pero existen puntos muy próximos a él.

f) Para la primera función, implemente cuando el lenguaje lo permita una versión escalar mediante un ciclo y otra vectorizada; identifique las operaciones que deben realizarse elemento a elemento.

g) Escriba los pares $(x, f(x))$ en un archivo tabular de dos columnas y regenere la figura a partir de esos datos.

h) Genere además una figura sencilla desde una especificación gráfica guardada en un archivo, verifique la salida y modifique la especificación para agregar una segunda curva.

Nota Gnuplot: Los valores no representables pueden declararse como datos faltantes, por ejemplo:

```
set datafile missing 'Nan'
plot 'datos.txt' using 1:2 with lines
```

Nota Gnuplot: Un archivo de órdenes mínimo puede tener la forma:

```
set terminal png size 700,400
set output "/tmp/ejb.png"
set xrange [0:10]
set xlabel "x"
set ylabel "y"
set title "(x^2-1)/(x^3+1)"
set grid
plot "/tmp/datos.txt" with lines lw 2 notitle
```

Nota Gnuplot: Otro ejemplo sencillo, pensado para verificar la ejecución y luego agregar una segunda curva, es:

```
cat > /tmp/primero.gnu << 'EOF'
set terminal png size 700,400
set output '/tmp/primero.png'
set title 'Mi primera grafica'
set xlabel 'x'
set ylabel 'f(x)'
set grid
plot sin(x) with lines lw 2 lc 'blue' title 'sin(x)'
EOF
gnuplot /tmp/primero.gnu
ls -lh /tmp/primero.png
display /tmp/primero.png
```

Nota Gnuplot: Abra el PNG, verifique la salida y modifique el script para agregar `cos(x)` en rojo.

Scilab. Una implementación escalar es:

```
x = linspace(-2, 2, 20)
y = zeros(x)
x = x(x <> -1)
for i = 1:length(x)
    y(i) = (x(i)^2 - 1) / (x(i)^3 + 1)
end

// alternativa conservando un valor no representable
x = linspace(-2, 2, 20)
y = zeros(x)
for i = 1:length(x)
    if x(i) == -1 then
        y(i) = %nan
    else
        y(i) = (x(i)^2 - 1) / (x(i)^3 + 1)
    end
end
```

Scilab. Conviene preasignar `y`. También puede eliminarse el punto $x = -1$ con `x=x(x<>-1)`. Como `x(i)` es escalar, las operaciones se realizan directamente.

Scilab. Una implementación vectorial es:

```
x = linspace(-2, 2, 20)
x = x(x <> -1)
y = (x.^2 - 1) ./ (x.^3 + 1)
```

Scilab. Los operadores precedidos por punto realizan operaciones elemento a elemento; sin ese punto se solicitarían operaciones matriciales.

Scilab. Para escribir las columnas puede usarse `fprintfMat`:

```
fprintfMat('/tmp/datos.txt', [x(:) y(:)], '%15.8f')
```

Scilab. `x(:)` e `y(:)` fuerzan vectores columna. `fprintfMat` gestiona internamente la apertura y cierre del archivo.

Respuesta (T: diez modalidades).

Para la primera función,

$$x^3+1=(x+1)(x^2-x+1), \qquad x^2-1=(x-1)(x+1).$$

Por lo tanto $x=-1$ no pertenece al dominio de la expresión original, pero la discontinuidad es **removible**; para $x\neq -1$,

$$f(x)=\frac{x-1}{x^2-x+1},$$

y el valor límite en el hueco es $-2/3$. No corresponde dibujar allí una asíntota vertical.

Para

$$g(t)=t+\frac{t}{1+\sin t},$$

el denominador se anula cuando $\sin t=-1$. En $[-2\pi,2\pi]$ ocurre en $t=-\pi/2$ y $t=3\pi/2$; allí sí aparecen singularidades verticales. Si la malla no contiene exactamente un punto singular, comparar sólo por igualdad es insuficiente: conviene detectar $|1+\sin t|<\varepsilon$, insertar un valor ausente o separar explícitamente los intervalos.

Scilab nativo

```
xf=linspace(-2,2,801); yf=zeros(xf)
for k=1:length(xf)
    if abs(xf(k)+1)<1d-12 then yf(k)=%nan
    else yf(k)=(xf(k)^2-1)/(xf(k)^3+1), end
end
yv=(xf.^2-1)./(xf.^3+1); yv(abs(xf+1)<1d-12)=%nan
fprintfMat('disc_f.dat',[xf(:) yf(:)],'%15.10g')

t=linspace(-2*%pi,2*%pi,1601); den=1+sin(t); yg=t+t./den;
   yg(abs(den)<1d-8)=%nan
fprintfMat('disc_g.dat',[t(:) yg(:)],'%15.10g')
scf(110); clf(); subplot(2,1,1); plot(xf,yf,'b-'); a=gca();
   a.auto_clear='off'; plot(-1,-2/3,'ro');
   a.auto_clear='on'; xgrid(); xtitle('Hueco removible en
   x=-1','x','f')
subplot(2,1,2); plot(t,yg,'r-'); xgrid(); xtitle('Ramas de
   g','t','g'); xs2png(gcf(),'ej11_scilab.png')
```

GNU Octave nativo

```
xf=linspace(-2,2,801); ys=zeros(size(xf)); for
   k=1:numel(xf), if abs(xf(k)+1)<1e-12, ys(k)=NaN; else
   ys(k)=(xf(k)^2-1)/(xf(k)^3+1); endif; endfor
yv=(xf.^2-1)./(xf.^3+1); yv(abs(xf+1)<1e-12)=NaN;
   dlmwrite('disc_f.dat',[xf' ys'],'delimiter',' ');
t=linspace(-2*pi,2*pi,1601); d=1+sin(t); yg=t+t./d;
   yg(abs(d)<1e-8)=NaN; dlmwrite('disc_g.dat',[t'
   yg'],'delimiter',' ');
```

```
figure(110); subplot(2,1,1); plot(xf,ys,'b-',-1,-2/3,'ro');
   grid on; title('Discontinuidad removible');
   subplot(2,1,2); plot(t,yg,'r-'); grid on; ylim([-30 30]);
   title('Ramas de g'); print('ej11_octave.png','-dpng');
```

Python nativo

```
import numpy as np, matplotlib.pyplot as plt
xf=np.linspace(-2,2,801)
ys=np.empty_like(xf)
for i,x in enumerate(xf): ys[i]=np.nan if abs(x+1)<1e-12
   else (x*x-1)/(x**3+1)
yv=(xf*xf-1)/(xf**3+1); yv[np.abs(xf+1)<1e-12]=np.nan
np.savetxt('disc_f.dat',np.c_[xf,ys])
t=np.linspace(-2*np.pi,2*np.pi,1601); d=1+np.sin(t);
   yg=t+t/d; yg[np.abs(d)<1e-8]=np.nan;
   np.savetxt('disc_g.dat',np.c_[t,yg])
fig,ax=plt.subplots(2,1,figsize=(9,8)); ax[0].plot(xf,ys);
   ax[0].plot(-1,-2/3,'ro',mfc='white'); ax[0].grid();
   ax[0].set_title('Hueco removible'); ax[1].plot(t,yg);
   ax[1].set_ylim(-30,30); ax[1].grid();
   ax[1].set_title('Singularidades verticales');
   fig.tight_layout(); fig.savefig('ej11_python.png',dpi=140)
```

Perl nativo

```
use strict; use warnings; use PDL; use
   PDL::Graphics::Simple; $ENV{PDL_SIMPLE_ENGINE}='prima';
   my$pi=4*atan2(1,1);
my(@xf,@yf);
   for(0..800){my$x=-2+4*$_/800;push@xf,$x;push@yf,abs($x+1)<1e-12
   ? 'BAD' : ($x*$x-1)/($x**3+1)}
# Para el dibujo se separan las partes a cada lado del hueco
   en dos DataSets.
my$x1=sequence(200)*1/199-2; my$y1=($x1**2-1)/($x1**3+1);
   my$x2=sequence(600)*3/599-0.999;
   my$y2=($x2**2-1)/($x2**3+1);
my$t1=sequence(300)*(1.5*$pi)/299-2*$pi;
   my$t2=sequence(500)*(2*$pi-0.002)/499-$pi/2+0.001;
   my$t3=sequence(300)*($pi/2-0.001)/299+1.5*$pi+0.001;
sub g{my$t=shift; $t+$t/(1+sin($t))}
my$w=pgswin(engine=>'prima',multi=>[2,1],out=>'ej11_perl.ps');
   $w->plot(with=>'line',$x1,$y1,with=>'line',$x2,$y2,{title=>'Hueco
   removible en -1'});
   $w->plot(with=>'line',$t1,g($t1),with=>'line',$t2,g($t2),with=>'line'
   de g'});
```

JavaScript/Node.js nativo

```
const fs=require('node:fs'); const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');
```

```
const
   lin=(a,b,n)=>Array.from({length:n},(_,i)=>a+(b-a)*i/(n-1));
   let
   xf=lin(-2,2,801),yf=xf.map(x=>Math.abs(x+1)<1e-12?null:(x*x-1)/(x**3+
   let
   t=lin(-2*Math.PI,2*Math.PI,1601),yg=t.map(x=>Math.abs(1+Math.sin(x))<
let
   figs=[[{x:xf,y:yf,type:'scatter',mode:'lines',name:'f'},{x:[-1],y:[-2
   let body='<div id=a style="height:450px"></div><div id=b
   style="height:450px"></div>';
   fs.writeFileSync('ej11_node.html',`<script>${P}</script>${body}<scrip
   removible'});Plotly.newPlot('b',${JSON.stringify(figs[1])},{title:'Ra
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot

```
set datafile missing 'NaN'; set terminal pngcairo size
   900,750; set output 'ej11_gnuplot.png'; set multiplot
   layout 2,1; set grid
set title 'f: discontinuidad removible'; set xrange [-2:2];
   plot 'disc_f.dat' u 1:2 w l t 'f', '-' w p pt 6 lc rgb
   'red' t 'hueco'
-1 -0.6666666666667
e
set title 'g: singularidades en -pi/2 y 3pi/2'; set xrange
   [-2*pi:2*pi]; set yrange [-30:30]; plot 'disc_g.dat' u
   1:2 w l t 'g'
unset multiplot; unset output
```

Para las cinco variantes siguientes se eligen 801 puntos para f y 1601 para g, de modo que las singularidades caigan exactamente en la malla. Aun así se usa un criterio de tolerancia, que es el que debe conservarse si se cambia la malla.

```
x=linspace(-2,2,801); d=x.^3+1; y=(x.^2-1)./d;
   y(abs(d)<1d-10)=%nan; fprintfMat('disc_f.dat',[x(:)
   y(:)],'%15.10g'); t=linspace(-2*%pi,2*%pi,1601);
   d=1+sin(t); z=t+t./d; z(abs(d)<1d-8)=%nan;
   fprintfMat('disc_g.dat',[t(:) z(:)],'%15.10g');
   host('gnuplot ej11.gnu')
```

```
x=linspace(-2,2,801)'; d=x.^3+1; y=(x.^2-1)./d;
   y(abs(d)<1e-10)=NaN; dlmwrite('disc_f.dat',[x
   y],'delimiter',' '); t=linspace(-2*pi,2*pi,1601)';
   d=1+sin(t); z=t+t./d; z(abs(d)<1e-8)=NaN;
   dlmwrite('disc_g.dat',[t z],'delimiter',' ');
   system('gnuplot ej11.gnu');
```

```
import numpy as np,subprocess; x=np.linspace(-2,2,801);
   d=x**3+1; y=(x*x-1)/d; y[np.abs(d)<1e-10]=np.nan;
   np.savetxt('disc_f.dat',np.c_[x,y]);
   t=np.linspace(-2*np.pi,2*np.pi,1601); d=1+np.sin(t);
   z=t+t/d; z[np.abs(d)<1e-8]=np.nan;
```

```
    np.savetxt('disc_g.dat',np.c_[t,z]);
    subprocess.run(['gnuplot','ej11.gnu'],check=True)
```

```
use POSIX qw(sin); my$pi=4*atan2(1,1); open
    my$f,'>','disc_f.dat' or
    die$!;for(0..800){my$x=-2+4*$_/800;my$d=$x**3+1;print$f
    abs($d)<1e-10?"$x NaN\n":"$x
    ".(($x*$x-1)/$d)."\n"}close$f;open my$g,'>','disc_g.dat'
    or
    die$!;for(0..1600){my$t=-2*$pi+4*$pi*$_/1600;my$d=1+sin($t);print$g
    abs($d)<1e-8?"$t NaN\n":"$t
    ".($t+$t/$d)."\n"}close$g;system('gnuplot','ej11.gnu');
```

```
const
    fs=require('node:fs'),{spawnSync}=require('node:child_process');let
    a='',b='';for(let i=0;i<=800;i++){let
    x=-2+4*i/800,d=x**3+1;a+=Math.abs(d)<1e-10?`${x}
    NaN\n`:`${x} ${(x*x-1)/d}\n`;}for(let
    i=0;i<=1600;i++){let
    t=-2*Math.PI+4*Math.PI*i/1600,d=1+Math.sin(t);b+=Math.abs(d)<1e-8?`${
    NaN\n`:`${t}
    ${t+t/d}\n`;}fs.writeFileSync('disc_f.dat',a);fs.writeFileSync('disc_
    r=spawnSync('gnuplot',['ej11.gnu']);if(r.status!==0)throw
    Error('gnuplot');
```

El archivo tabular satisface el inciso de interoperabilidad. El archivo `ej11.gnu` constituye la especificación gráfica reproducible: para añadir una segunda curva basta agregar otra serie al comando `plot` y volver a ejecutarlo. Eso no debe confundirse con la persistencia de una escena gráfica editable estudiada en el ejercicio anterior.

12. **Ejercicio:** Definición de funciones y curvas múltiples

Considere

$$f(x) = x^2 + 4,$$

$$g(x) = x^3 - 300\sin x,$$

$$h(x) = x^3 + 300\cos x.$$

a) Represente f en un intervalo que permita observar claramente su mínimo.

b) Represente g en $[-10, 10]$ utilizando una discretización suficientemente fina para observar la componente oscilatoria.

c) Represente g y h conjuntamente en $[-10, 10]$, con colores y estilos diferentes.

d) Defina las funciones en unidades reutilizables del lenguaje de trabajo y separe conceptualmente la evaluación numérica de la representación gráfica.

e) Explique qué información se obtiene al representar g y h conjuntamente que no resulta tan evidente en gráficos separados.

Scilab. El papel comparable a `fplot2d` consiste en evaluar una función sobre un rango; cuando la función ya puede expresarse en sintaxis de Gnuplot, no hace falta calcular previamente los vectores.

```
# equivalente a fplot2d(x,f) en Gnuplot puro
gnuplot -e "
set terminal png size 700,400
set output '/tmp/fplot_a.png'
set title  'f(x) = x^2 + 4'
set xlabel 'x'
set ylabel 'y'
set grid
f(x) = x**2 + 4
plot f(x) with lines lw 2 lc 'blue' notitle
"
```

Scilab. Si se desea calcular previamente los datos en Scilab, puede definirse la función, escribir las columnas y llamar luego a Gnuplot:

```
// definir la funcion (equivale a deff en Scilab)
function y = f(x)
    y = x.^2 + 4
endfunction


// evaluar sobre el rango
x = linspace(-5, 5, 200)

// guardar datos /tmp/fplot_a.txt
//y = zeros(x)
//for i = 1:length(x)
//    y(i) = f(x(i))
//end
fprintfMat(nombre, [x(:) f(y(:))], '%15.8f')
//porque la funcion esta vectorizada
```

```
set terminal png size 700,400
set output "/tmp/fplot_a.png"
set title  "f(x) = x^2 + 4"
set xlabel "x"
set ylabel "f(x)"
set grid
plot "/tmp/fplot_a.txt" with lines lw 2 lc "blue" notitle
```

Scilab. Implementación o cálculo auxiliar en Scilab:

```
host('gnuplot /tmp/fplot_a.gnu')
printf('Grafica: /tmp/fplot_a.png\n')
```

```
function y = g(x)
    y = x.^3 - 300*sin(x)
```

```
endfunction

x = linspace(-10, 10, 500)


// guardar datos /tmp/fplot_b.txt
fprintfMat(nombre, [x(:) g(x(:))], '%15.8f')
```

```
set terminal png size 700,400
set output "/tmp/fplot_b.png"
set title  "g(x) = x^3 - 300*sin(x)"
set xlabel "x"
set ylabel "g(x)"
set grid
plot "/tmp/fplot_b.txt" with lines lw 2 lc "red" notitle
```

```
host('gnuplot /tmp/fplot_b.gnu')
printf('Grafica: /tmp/fplot_b.png\n')
```

```
function y = g(x)
    y = x.^3 - 300*sin(x)
endfunction

function y = h(x)
    y = x.^3 + 300*cos(x)
endfunction

x = linspace(-10, 10, 500)

// guardar las dos curvas en un solo archivo /tmp/fplot_c.txt
//    mfprintf(fd, '%.8f %.8f %.8f\n', x(i), g(x(i)),
   h(x(i)))
fprintfMat('/tmp/fplot_c.txt', [x(:) g(x(:)) h(x(:))],
   '%15.8f')
```

```
# /tmp/fplot_c.txt
set terminal png size 700,450
set output "/tmp/fplot_c.png"
set title  "g(x) y h(x)"
set xlabel "x"
set ylabel "y"
set grid
set key top left
plot "/tmp/fplot_c.txt" using 1:2 with lines lw 2 lc "red" \
     title "g(x)=x^3-300*sin(x)", \
     "/tmp/fplot_c.txt" using 1:3 with lines lw 2 lc "blue" \
     title "h(x)=x^3+300*cos(x)"
```

```
host('gnuplot /tmp/fplot_c.gnu')
printf('Grafica: /tmp/fplot_c.png\n')
```

Scilab. Este segundo procedimiento es el apropiado cuando la función se evalúa en Scilab, por ejemplo porque involucra operaciones matriciales o condiciones.

Scilab. Una forma nativa de representar directamente una función definida por el usuario es `fplot2d`. Para el primer inciso, `eu07b` indicaba explícitamente:

```
fplot2d(x,f)
```

Respuesta (L: cinco lenguajes; gráficos nativos o Gnuplot).

Se emplea $[-5,5]$ para f y $[-10,10]$ para g y h. Para estas dos últimas funciones se usan 801 muestras, suficientes para que el término oscilatorio sea visible sin confundirlo con una discretización demasiado gruesa. Al superponer g y h se observa que ambas comparten la tendencia cúbica x^3, mientras que los términos $-300\sin x$ y $300\cos x$ introducen oscilaciones desfasadas.

Scilab, gráficos nativos

```
function y=f(x), y=x.^2+4, endfunction
function y=g(x), y=x.^3-300*sin(x), endfunction
function y=h(x), y=x.^3+300*cos(x), endfunction

xf=linspace(-5,5,401)
x=linspace(-10,10,801)
scf(120); clf()
subplot(1,2,1); plot(xf,f(xf),'b-'); xgrid();
   xtitle('f(x)=x^2+4','x','f')
subplot(1,2,2); plot(x,g(x),'r-',x,h(x),'b--'); xgrid()
legend('g(x)','h(x)'); xtitle('g y h','x','y')
xs2png(gcf(),'ej12_scilab.png')
```

GNU Octave, gráficos nativos

```
f=@(x)x.^2+4; g=@(x)x.^3-300*sin(x); h=@(x)x.^3+300*cos(x);
xf=linspace(-5,5,401); x=linspace(-10,10,801);
figure(120); clf;
subplot(1,2,1); plot(xf,f(xf),'b-','linewidth',1.5); grid on;
title('f(x)=x^2+4'); xlabel('x'); ylabel('f(x)');
subplot(1,2,2);
   plot(x,g(x),'r-',x,h(x),'b--','linewidth',1.3); grid on;
legend('g(x)','h(x)'); title('g y h'); xlabel('x');
   ylabel('y');
print('ej12_octave.png','-dpng','-r140');
```

Python, gráficos nativos

```
import numpy as np
import matplotlib.pyplot as plt

def f(x): return x**2+4
def g(x): return x**3-300*np.sin(x)
```

```
def h(x): return x**3+300*np.cos(x)

xf=np.linspace(-5,5,401); x=np.linspace(-10,10,801)
fig,ax=plt.subplots(1,2,figsize=(11,4.5))
ax[0].plot(xf,f(xf)); ax[0].grid();
   ax[0].set_title('f(x)=x^2+4')
ax[0].set_xlabel('x'); ax[0].set_ylabel('f(x)')
ax[1].plot(x,g(x),label='g(x)');
   ax[1].plot(x,h(x),'--',label='h(x)')
ax[1].grid(); ax[1].legend(); ax[1].set_title('g y h');
   ax[1].set_xlabel('x')
fig.tight_layout(); fig.savefig('ej12_python.png',dpi=140)
```

Perl, gráficos nativos mediante SVG generado por el programa. Esta variante no inicia ningún programa gráfico externo: Perl calcula los puntos y escribe directamente un archivo vectorial SVG.

```
use strict; use warnings;
sub f { my($x)=@_; $x*$x+4 }
sub g { my($x)=@_; $x**3-300*sin($x) }
sub h { my($x)=@_; $x**3+300*cos($x) }
sub poly {
  my($a,$b,$n,$fun,$X0,$Y0,$W,$H,$ymin,$ymax)=@_; my @p;
  for my $i(0..$n-1){ my $x=$a+($b-$a)*$i/($n-1); my
     $y=$fun->($x);
    my $X=$X0+$W*($x-$a)/($b-$a); my
       $Y=$Y0+$H*(1-($y-$ymin)/($ymax-$ymin));
    push @p,"$X,$Y"; } return join(' ',@p);
}
open my $s,'>','ej12_perl.svg' or die $!;
print $s qq{<svg xmlns="http://www.w3.org/2000/svg"
   width="1000" height="450">};
print $s qq{<rect width="100%" height="100%" fill="white"/>};
print $s qq{<polyline fill="none" stroke="black" points="},
  poly(-5,5,401,\&f,40,40,420,350,4,30),qq{"/>};
print $s qq{<polyline fill="none" stroke="black" points="},
  poly(-10,10,801,\&g,540,40,420,350,-1300,1300),qq{"/>};
print $s qq{<polyline fill="none" stroke="gray"
   stroke-dasharray="7 5" points="},
  poly(-10,10,801,\&h,540,40,420,350,-1300,1300),qq{"/>};
print $s qq{<text x="150" y="420">f(x)=x^2+4</text>};
print $s qq{<text x="650" y="420">g(x), h(x)</text></svg>};
   close $s;
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');
const
   plotly=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');
const
   lin=(a,b,n)=>Array.from({length:n},(_,i)=>a+(b-a)*i/(n-1));
```

```
const f=x=>x*x+4, g=x=>x*x*x-300*Math.sin(x),
   h=x=>x*x*x+300*Math.cos(x);
const xf=lin(-5,5,401), x=lin(-10,10,801);
const divs='<div id="f"
   style="width:48%;height:430px;display:inline-block"></div>'+
          '<div id="gh"
             style="width:48%;height:430px;display:inline-block"></div>
const
   js=`Plotly.newPlot('f',[{x:${JSON.stringify(xf)},y:${JSON.stringify(x
 {title:'f(x)=x^2+4',xaxis:{title:'x'},yaxis:{title:'f(x)'}});
Plotly.newPlot('gh',[{x:${JSON.stringify(x)},y:${JSON.stringify(x.map(g)
{x:${JSON.stringify(x)},y:${JSON.stringify(x.map(h))},name:'h',line:{das
 {title:'g y h',xaxis:{title:'x'},yaxis:{title:'y'}});`;
fs.writeFileSync('ej12_node.html',`<!doctype
   html><script>${plotly}</script>${divs}<script>${js}</script>`);
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot. En esta modalidad las funciones se definen y evalúan en el lenguaje; Gnuplot recibe únicamente las tablas numéricas.

```
set terminal pngcairo size 1000,450
set output 'ej12_gnuplot.png'
set multiplot layout 1,2
set grid
set title 'f(x)=x^2+4'
plot 'ej12_f.dat' u 1:2 w l lw 2 t 'f'
set title 'g(x) y h(x)'
plot 'ej12_gh.dat' u 1:2 w l lw 2 t 'g', \
     'ej12_gh.dat' u 1:3 w l lw 2 dt 2 t 'h'
unset multiplot
unset output
```

Scilab + Gnuplot

```
function y=f(x), y=x.^2+4, endfunction
function y=g(x), y=x.^3-300*sin(x), endfunction
function y=h(x), y=x.^3+300*cos(x), endfunction
x=linspace(-5,5,401); fprintfMat('ej12_f.dat',[x(:)
   f(x(:))],'%15.10g')
x=linspace(-10,10,801); fprintfMat('ej12_gh.dat',[x(:)
   g(x(:)) h(x(:))],'%15.10g')
[e,o,er]=host('gnuplot ej12.gnu'); if e<>0 then error(er),
   end
```

GNU Octave + Gnuplot

```
f=@(x)x.^2+4; g=@(x)x.^3-300*sin(x); h=@(x)x.^3+300*cos(x);
x=linspace(-5,5,401)'; dlmwrite('ej12_f.dat',[x
   f(x)],'delimiter',' ');
x=linspace(-10,10,801)'; dlmwrite('ej12_gh.dat',[x g(x)
   h(x)],'delimiter',' ');
```

```
[e,o]=system('gnuplot ej12.gnu'); if e!=0, error(o); endif
```

Python + Gnuplot

```
import numpy as np, subprocess
f=lambda x:x**2+4; g=lambda x:x**3-300*np.sin(x); h=lambda
   x:x**3+300*np.cos(x)
x=np.linspace(-5,5,401);
   np.savetxt('ej12_f.dat',np.c_[x,f(x)])
x=np.linspace(-10,10,801);
   np.savetxt('ej12_gh.dat',np.c_[x,g(x),h(x)])
subprocess.run(['gnuplot','ej12.gnu'],check=True)
```

Perl + Gnuplot

```
use strict; use warnings;
sub f{$_[0]**2+4} sub g{$_[0]**3-300*sin($_[0])} sub
   h{$_[0]**3+300*cos($_[0])}
open my $a,'>','ej12_f.dat' or die $!;
   for(0..400){my$x=-5+10*$_/400; print $a "$x ",f($x),"\n"}
   close $a;
open my $b,'>','ej12_gh.dat' or die $!;
   for(0..800){my$x=-10+20*$_/800; print $b "$x ",g($x),'
   ',h($x),"\n"} close $b;
system('gnuplot','ej12.gnu')==0 or die "gnuplot fallo\n";
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');
const
   f=x=>x*x+4,g=x=>x*x*x-300*Math.sin(x),h=x=>x*x*x+300*Math.cos(x);
let a='';for(let i=0;i<=400;i++){let x=-5+10*i/400;a+=`${x}
   ${f(x)}\n`;}fs.writeFileSync('ej12_f.dat',a);
let b='';for(let i=0;i<=800;i++){let x=-10+20*i/800;b+=`${x}
   ${g(x)} ${h(x)}\n`;}fs.writeFileSync('ej12_gh.dat',b);
let
   r=spawnSync('gnuplot',['ej12.gnu'],{encoding:'utf8'});if(r.status!==0
   new Error(r.stderr);
```

7.4.4. Sección 3: Escalas, coordenadas y datos experimentales

13. **Ejercicio:** Escalas lineales y logarítmicas

Estudie cómo cambia la interpretación de una función cuando se modifican las escalas de los ejes.

a) Represente

$$f(t) = e^{5t}, \qquad 0 \le t \le 4,$$

primero con ejes lineales y luego con eje vertical logarítmico. Explique la forma que adopta la curva en cada representación.

b) Represente
$$f(t) = e^{5t}, \qquad 1 \le t \le 20,$$
utilizando una escala semilogarítmica con eje vertical logarítmico. Compare el resultado con la representación del inciso anterior.

c) Represente
$$f(t) = e^{5t}, \qquad 1 \le t \le 1000,$$
utilizando ambos ejes logarítmicos, tal como se plantea en el ejercicio original. Antes de graficar, determine si todos los valores pueden representarse numéricamente en el entorno utilizado. Si aparece desbordamiento, identifique a partir de qué valores ocurre y documente una estrategia que permita estudiar el comportamiento solicitado sin ocultar el problema numérico.

d) Represente
$$g(t) = t^2, \qquad 10^{-2} \le t \le 10^2,$$
con ejes lineales y luego con ambos ejes logarítmicos. Explique la relación entre la pendiente observada en la representación log–log y el exponente de la ley de potencia.

e) Represente
$$h(x) = \ln x$$
con escala logarítmica en el eje vertical. Elija un rango en el que los valores que deban mostrarse sean compatibles con dicha escala y explique la elección.

f) Represente conjuntamente, con escalas lineales,
$$h_1(t) = \sin(t^2), \qquad h_2(t) = t \sin t, \qquad 1 \le t \le 10.$$

g) Indique qué restricciones deben satisfacer los datos cuando se utiliza una escala logarítmica y cómo debe tratarse un valor nulo, negativo, infinito o no representable.

Scilab. En `plot2d` las combinaciones de escala se expresan con `logflag`: `"nl"` para eje vertical logarítmico, `"ll"` para ambos ejes logarítmicos y `"nn"` para escala lineal. Los estilos, leyendas y límites pueden controlarse con las opciones correspondientes de `plot2d`.

Nota Gnuplot: En Gnuplot las escalas logarítmicas se activan con `set logscale y` o `set logscale xy`; las escalas lineales se obtienen sin esas opciones o mediante `unset logscale`.

Scilab. Implementación o cálculo auxiliar en Scilab:

```
t  = linspace(1, 20, 200)
ft = exp(5*t)

// guardar los datos
fprintfMat(/tmp/exp_a.txt, [t(:) ft(:)], '%15.8f')
```

```
set terminal png size 700,400
set output "/tmp/plot2d_a.png"
set title  "f(t) = exp(5t)  [semilog y]"
set xlabel "t"
set ylabel "f(t)"
set logscale y
set grid
plot "/tmp/exp_a.txt" with lines lw 2 lc "blue" notitle
```

```
host('gnuplot /tmp/plot2d_a.gnu')
printf('Grafica: /tmp/plot2d_a.png\n')
```

```
t  = linspace(1, 1000, 500)
ft = exp(5*t)

// guardar los datos '/tmp/exp_b.txt'
fprintfMat('/tmp/exp_b.txt', [t(:) ft(:)], '%15.8f')
```

```
set terminal png size 700,400
set output "/tmp/plot2d_b.png"
set title  "f(t) = exp(5t)  [log-log]"
set xlabel "t"
set ylabel "f(t)"
set logscale xy
set grid
plot "/tmp/exp_b.txt" with lines lw 2 lc "red" notitle
```

```
host('gnuplot /tmp/plot2d_b.gnu')
printf('Grafica: /tmp/plot2d_b.png\n')
```

```
t  = linspace(1, 10, 500)
ft = sin(t.^2)

// guardar las dos curvas
fprintfMat('/tmp/sin_c.txt', [t(:) ft(:)], '%15.8f')
```

```
set terminal png size 700,400
set output "/tmp/plot2d_c.png"
set title  "f(t) = sin(t^2)"
set xlabel "t"
set ylabel "f(t)"
set xrange [1:10]
set yrange [-1.2:1.2]
set grid
plot "/tmp/sin_c.txt" with lines lw 2 lc "blue" notitle
```

```
host('gnuplot /tmp/plot2d_c.gnu')
printf('Grafica: /tmp/plot2d_c.png\n')
```

```
t  = linspace(1, 10, 500)
fc = sin(t.^2)
fd_ = t .* sin(t)

// guardar las dos curvas
fprintfMat('/tmp/cd.txt', [t(:) fc(:) fd_(:)], '%15.8f')
```

```
# --- version 1: dos curvas basicas con leyenda ---
# equivale a plot2d(t,[fc;fd],leg="sin(t^2)@t*sin(t)")

set terminal png size 700,450
set output "/tmp/plot2d_d1.png"
set title  "sin(t^2) y t*sin(t)"
set xlabel "t"
set ylabel "f(t)"
set xrange [1:10]
set grid
set key top left
plot "/tmp/cd.txt" using 1:2 with lines lw 2 lc "blue" title
   "sin(t^2)", \
     "/tmp/cd.txt" using 1:3 with lines lw 2 lc "red"  title
        "t*sin(t)"
```

```
gnuplot /tmp/plot2d_d1.gnu
```

```
# --- version 2: estilos de linea distintos ---
# equivale a xset("line style",2) y xset("line style",4)
# dt 1=solida  dt 2=guiones  dt 3=punteada  dt 4=guion-punto
set terminal png size 700,450
set output "/tmp/plot2d_d2.png"
set title  "Estilos de linea distintos"
set xlabel "t"
set ylabel "f(t)"
set xrange [1:10]
set grid
set key top left
plot "/tmp/cd.txt" using 1:2 with lines lw 2 lc "blue" dt 2 \
     title "sin(t^2) [guiones]", \
     "/tmp/cd.txt" using 1:3 with lines lw 2 lc "red" dt 4 \
     title "t*sin(t) [guion-punto]"
```

```
gnuplot /tmp/plot2d_d2.gnu
```

```
# --- version 3: con marcadores y rect ---
# equivale a plot2d con rect=[xmin,ymin,xmax,ymax]
# y style con marcadores

  set terminal png size 700,450
set output "/tmp/plot2d_d3.png"
```

```
set title  "Con marcadores y rect"
set xlabel "t"
set ylabel "f(t)"
set xrange [1:10]
set yrange [-10:10]
set grid
set key top left
set samples 30
plot "/tmp/cd.txt" using 1:2 with linespoints lw 1.5 lc
   "blue" \
     pt 7 ps 0.8 title "sin(t^2)", \
     "/tmp/cd.txt" using 1:3 with linespoints lw 1.5 lc
        "red" \
     pt 5 ps 0.8 title "t*sin(t)"
```

```
host('gnuplot /tmp/plot2d_d3.gnu')

printf('Graficas generadas:\n')
printf('  /tmp/plot2d_d1.png  (leyenda basica)\n')
printf('  /tmp/plot2d_d2.png  (estilos de linea)\n')
printf('  /tmp/plot2d_d3.png  (marcadores y rect)\n')
```

Scilab. Como referencia adicional para la vía nativa, la sintaxis general de `plot2d` es:

```
plot2d(x,y,<opciones>)
```

Scilab. En esa sintaxis, `x` puede ser un vector o matriz; si se omite, Scilab supone un vector $1:n$, siendo n el número de puntos de `y`. El argumento `y` contiene los datos a representar y las opciones se expresan como pares `clave = valor`. Además de `logflag`, ya descripto arriba, son de interés: `style` para el estilo de las curvas; `rect` para fijar $[x_{\text{mín}}, y_{\text{mín}}, x_{\text{máx}}, y_{\text{máx}}]$; `frameflag` para el cálculo del rango; `axesflag` para la forma de los ejes; `nax` para etiquetas y subdivisiones; y `leg` para leyendas. La función `legends()` ofrece mayor flexibilidad que `leg`.

```
"curva1@curva2@..."
```

Scilab. Ejemplos conservados de `eu07b`:

```
t=linspace(-2*%pi,2*%pi,100);

plot2d(t,[cos(t') cos(2*t')],
       [3,5],
       leg="cos(t)@cos(2t)")
```

```
clf();

plot2d(t,
      [cos(t'),
       cos(2*t'),
```

```
        cos(3*t')],
      [-1,2,3]);

legends(['cos(t)';
         'cos(2*t)';
         'cos(3*t)'],
         [-1,2,3],
         opt="lr");
```

```
xset("line style",2);
plot2d(t,cos(t),style=5);

xset("line style",4);
plot2d(t,sin(t),style=3);
```

Respuesta (T: diez modalidades).

En aritmética IEEE de doble precisión, el mayor argumento aproximadamente representable para la exponencial es 709,78. Por lo tanto

$$e^{5t} \text{ desborda cuando } \quad t \gtrsim \frac{709{,}78}{5} \approx 141{,}96.$$

No se reemplazan esos valores por un número arbitrario. Para el inciso con $1 \le t \le 1000$ se muestran dos cosas: el gráfico log–log directo sólo en el tramo representable y, para el dominio completo, las coordenadas exactas que vería ese gráfico,

$$(\ln t, \ln f(t)) = (\ln t, 5t),$$

sin formar numéricamente e^{5t}.

Para $h(x) = \ln x$ con eje vertical logarítmico se toma $x > 1$; por ejemplo $1{,}01 \le x \le 100$, porque una escala logarítmica en y exige $\ln x > 0$.

Scilab, gráficos nativos

```
t=linspace(0,4,300); y=exp(5*t)
scf(130); clf(); subplot(2,3,1); plot(t,y); xgrid();
   xtitle('exp(5t), lineal','t','y')
subplot(2,3,2); plot2d(t,y,logflag='nl'); xgrid();
   xtitle('exp(5t), semilog-y','t','y')
t=linspace(1,20,300); subplot(2,3,3);
   plot2d(t,exp(5*t),logflag='nl'); xgrid()

t=exp(linspace(log(1),log(140),300)); subplot(2,3,4);
   plot2d(t,exp(5*t),logflag='ll'); xgrid()
tfull=linspace(1,1000,400); subplot(2,3,5);
   plot(log(tfull),5*tfull); xgrid()
xtitle('Dominio completo: ln(f)=5t','ln(t)','ln(f)')

t=exp(linspace(log(1d-2),log(1d2),400)); scf(131); clf();
subplot(1,3,1); plot2d(t,t.^2,logflag='ll'); xgrid();
   xtitle('t^2, log-log','t','t^2')
```

```
x=linspace(1.01,100,400); subplot(1,3,2);
   plot2d(x,log(x),logflag='nl'); xgrid()
t=linspace(1,10,500); subplot(1,3,3);
   plot(t,sin(t.^2),t,t.*sin(t)); xgrid();
   legend('sin(t^2)','t sin(t)')
xs2png(gcf(),'ej13_scilab.png')
```

GNU Octave, gráficos nativos

```
figure(130); clf;
t=linspace(0,4,300); y=exp(5*t);
subplot(2,3,1); plot(t,y); grid on; title('lineal');
subplot(2,3,2); semilogy(t,y); grid on; title('semilog-y');
t=linspace(1,20,300); subplot(2,3,3); semilogy(t,exp(5*t));
   grid on;
t=logspace(0,log10(140),300); subplot(2,3,4);
   loglog(t,exp(5*t)); grid on; title('log-log seguro');
t=linspace(1,1000,400); subplot(2,3,5); plot(log(t),5*t);
   grid on; xlabel('ln t'); ylabel('ln f');
t=logspace(-2,2,400); subplot(2,3,6); loglog(t,t.^2); grid
   on; title('t^2: pendiente 2');
figure(131); x=linspace(1.01,100,400); semilogy(x,log(x));
   grid on; title('ln(x), eje y log');
figure(132); t=linspace(1,10,500);
   plot(t,sin(t.^2),t,t.*sin(t)); grid on;
   legend('sin(t^2)','t sin t');
```

Python, gráficos nativos

```
import numpy as np, matplotlib.pyplot as plt
fig,ax=plt.subplots(2,3,figsize=(12,7))
t=np.linspace(0,4,300); y=np.exp(5*t)
ax[0,0].plot(t,y); ax[0,0].set_title('lineal');
   ax[0,0].grid()
ax[0,1].semilogy(t,y); ax[0,1].set_title('semilog-y');
   ax[0,1].grid()
t=np.linspace(1,20,300); ax[0,2].semilogy(t,np.exp(5*t));
   ax[0,2].grid()
t=np.logspace(0,np.log10(140),300);
   ax[1,0].loglog(t,np.exp(5*t)); ax[1,0].grid();
   ax[1,0].set_title('log-log seguro')
t=np.linspace(1,1000,400); ax[1,1].plot(np.log(t),5*t);
   ax[1,1].grid(); ax[1,1].set_xlabel('ln t');
   ax[1,1].set_ylabel('ln f')
t=np.logspace(-2,2,400); ax[1,2].loglog(t,t*t);
   ax[1,2].grid(); ax[1,2].set_title('t^2, pendiente 2')
fig.tight_layout(); fig.savefig('ej13_python_a.png',dpi=140)
fig,ax=plt.subplots(1,2,figsize=(10,4))
x=np.linspace(1.01,100,400); ax[0].semilogy(x,np.log(x));
   ax[0].grid(); ax[0].set_title('ln x con y log')
t=np.linspace(1,10,500);
   ax[1].plot(t,np.sin(t*t),label='sin(t^2)');
```

```
    ax[1].plot(t,t*np.sin(t),label='t sin t'); ax[1].grid();
    ax[1].legend()
fig.tight_layout(); fig.savefig('ej13_python_b.png',dpi=140)
```

Perl, gráficos nativos mediante SVG. Para no depender de una biblioteca gráfica externa, las escalas logarítmicas se implementan transformando explícitamente las coordenadas antes de escribir el SVG; esto reproduce exactamente la geometría de un eje logarítmico.

```
use strict; use warnings;
sub line_svg { my($name,$pts)=@_; open my $s,'>',$name or
   die $!;
 print $s '<svg xmlns="http://www.w3.org/2000/svg"
    width="800" height="500"><rect width="100%"
    height="100%" fill="white"/>';
 print $s '<polyline fill="none" stroke="black"
    points="',join(' ',@$pts),'"/></svg>'; close $s; }
my @p; for(0..399){my$t=1+999*$_/399;
   my$X=50+700*log($t)/log(1000);
   my$Y=450-400*(5*$t-5)/(5000-5); push @p,"$X,$Y"}
line_svg('ej13_perl_loglog_transformado.svg',\@p);
# Aqui Y representa ln(f)=5t y X representa ln(t), evitando
   calcular exp(5t).
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');
const
   lin=(a,b,n)=>Array.from({length:n},(_,i)=>a+(b-a)*i/(n-1));
let t=lin(0,4,300), y=t.map(v=>Math.exp(5*v));
let safe=lin(1,140,300), full=lin(1,1000,400),
   pow=Array.from({length:400},(_,i)=>10**(-2+4*i/399));
let data=[
 {x:t,y,name:'exp(5t) lineal'},
 {x:t,y,name:'exp(5t) semilog',yaxis:'y2'},
 {x:safe,y:safe.map(v=>Math.exp(5*v)),name:'log-log seguro'},
 {x:full.map(Math.log),y:full.map(v=>5*v),name:'(ln t, ln
    f)'},
 {x:pow,y:pow.map(v=>v*v),name:'t^2'}];
let html=`<!doctype html><script>${P}</script><div id="p"
   style="width:1100px;height:700px"></div><script>
Plotly.newPlot('p',${JSON.stringify(data)},{title:'Ejercicio
   13',yaxis2:{type:'log',overlaying:'y',side:'right'}});
</script>`;fs.writeFileSync('ej13_node.html',html);
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot

```
set terminal pngcairo size 1100,700
set output 'ej13_gnuplot_a.png'
set multiplot layout 2,3
```

```
set grid
unset logscale
set title 'exp(5t), lineal'; plot 'ej13_a.dat' u 1:2 w l
   notitle
set logscale y; set title 'exp(5t), semilog-y'; plot
   'ej13_a.dat' u 1:2 w l notitle
set title 'exp(5t), 1<=t<=20'; plot 'ej13_b.dat' u 1:2 w l
   notitle
set logscale xy; set title 'log-log, tramo seguro'; plot
   'ej13_safe.dat' u 1:2 w l notitle
unset logscale; set title 'dominio completo: (ln t, ln f)';
   set xlabel 'ln t'; set ylabel 'ln f'
plot 'ej13_full.dat' u 1:2 w l notitle
set logscale xy; set title 't^2, pendiente 2'; set xlabel
   't'; set ylabel 't^2'
plot 'ej13_pow.dat' u 1:2 w l notitle
unset multiplot; unset output; unset logscale

set terminal pngcairo size 900,400
set output 'ej13_gnuplot_b.png'
set multiplot layout 1,2
set logscale y; set title 'ln x con eje y log'; plot
   'ej13_lnx.dat' u 1:2 w l notitle
unset logscale; set title 'sin(t^2) y t sin(t)'; plot
   'ej13_mix.dat' u 1:2 w l t 'sin(t^2)', '' u 1:3 w l t 't
   sin(t)'
unset multiplot; unset output
```

Generación de los datos en los cinco lenguajes. Las cinco variantes producen los mismos siete archivos. En particular, `ej13_full.dat` contiene $\ln t$ y $5t$, no e^{5t}.

Scilab + Gnuplot

```
t=linspace(0,4,300); fprintfMat('ej13_a.dat',[t(:)
   exp(5*t(:))],'%15.10g')
t=linspace(1,20,300); fprintfMat('ej13_b.dat',[t(:)
   exp(5*t(:))],'%15.10g')
t=exp(linspace(log(1),log(140),300));
   fprintfMat('ej13_safe.dat',[t(:) exp(5*t(:))],'%15.10g')
t=linspace(1,1000,400);
   fprintfMat('ej13_full.dat',[log(t(:)) 5*t(:)],'%15.10g')
t=exp(linspace(log(1d-2),log(1d2),400));
   fprintfMat('ej13_pow.dat',[t(:) t(:).^2],'%15.10g')
x=linspace(1.01,100,400); fprintfMat('ej13_lnx.dat',[x(:)
   log(x(:))],'%15.10g')
t=linspace(1,10,500); fprintfMat('ej13_mix.dat',[t(:)
   sin(t(:).^2) t(:).*sin(t(:))],'%15.10g')
[e,o,er]=host('gnuplot ej13.gnu'); if e<>0 then error(er),
   end
```

GNU Octave + Gnuplot

```
save2=@(n,a) dlmwrite(n,a,'delimiter',' ');
t=linspace(0,4,300)'; save2('ej13_a.dat',[t exp(5*t)]);
t=linspace(1,20,300)'; save2('ej13_b.dat',[t exp(5*t)]);
t=logspace(0,log10(140),300)'; save2('ej13_safe.dat',[t
   exp(5*t)]);
t=linspace(1,1000,400)'; save2('ej13_full.dat',[log(t) 5*t]);
t=logspace(-2,2,400)'; save2('ej13_pow.dat',[t t.^2]);
x=linspace(1.01,100,400)'; save2('ej13_lnx.dat',[x log(x)]);
t=linspace(1,10,500)'; save2('ej13_mix.dat',[t sin(t.^2)
   t.*sin(t)]);
[e,o]=system('gnuplot ej13.gnu'); if e!=0,error(o);endif
```

Python + Gnuplot

```
import numpy as np, subprocess
S=lambda n,*c:np.savetxt(n,np.column_stack(c))
t=np.linspace(0,4,300);S('ej13_a.dat',t,np.exp(5*t))
t=np.linspace(1,20,300);S('ej13_b.dat',t,np.exp(5*t))
t=np.logspace(0,np.log10(140),300);S('ej13_safe.dat',t,np.exp(5*t))
t=np.linspace(1,1000,400);S('ej13_full.dat',np.log(t),5*t)
t=np.logspace(-2,2,400);S('ej13_pow.dat',t,t*t)
x=np.linspace(1.01,100,400);S('ej13_lnx.dat',x,np.log(x))
t=np.linspace(1,10,500);S('ej13_mix.dat',t,np.sin(t*t),t*np.sin(t))
subprocess.run(['gnuplot','ej13.gnu'],check=True)
```

Perl + Gnuplot

```
use strict;use warnings;
sub wr{my($n,$rows)=@_;open my$h,'>',$n or die$!;print$h
   join(' ',@$_),"\n" for@$rows;close$h}
my
   @r;for(0..299){my$t=4*$_/299;push@r,[$t,exp(5*$t)]}wr('ej13_a.dat',\@
@r=();for(0..299){my$t=1+19*$_/299;push@r,[$t,exp(5*$t)]}wr('ej13_b.dat'
@r=();for(0..299){my$t=exp(log(140)*$_/299);push@r,[$t,exp(5*$t)]}wr('ej
@r=();for(0..399){my$t=1+999*$_/399;push@r,[log($t),5*$t]}wr('ej13_full.
@r=();for(0..399){my$t=10**(-2+4*$_/399);push@r,[$t,$t*$t]}wr('ej13_pow.
@r=();for(0..399){my$x=1.01+(100-1.01)*$_/399;push@r,[$x,log($x)]}wr('ej
@r=();for(0..499){my$t=1+9*$_/499;push@r,[$t,sin($t*$t),$t*sin($t)]}wr('
system('gnuplot','ej13.gnu')==0 or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');
function wr(n,rows){fs.writeFileSync(n,rows.map(r=>r.join('
   ')).join('\n')+'\n')}
let R=[];for(let i=0;i<300;i++){let
   t=4*i/299;R.push([t,Math.exp(5*t)])}wr('ej13_a.dat',R);
R=[];for(let i=0;i<300;i++){let
   t=1+19*i/299;R.push([t,Math.exp(5*t)])}wr('ej13_b.dat',R);
```

```
R=[];for(let i=0;i<300;i++){let
   t=Math.exp(Math.log(140)*i/299);R.push([t,Math.exp(5*t)])}wr('ej13_sa
R=[];for(let i=0;i<400;i++){let
   t=1+999*i/399;R.push([Math.log(t),5*t])}wr('ej13_full.dat',R);
R=[];for(let i=0;i<400;i++){let
   t=10**(-2+4*i/399);R.push([t,t*t])}wr('ej13_pow.dat',R);
R=[];for(let i=0;i<400;i++){let
   x=1.01+(100-1.01)*i/399;R.push([x,Math.log(x)])}wr('ej13_lnx.dat',R);
R=[];for(let i=0;i<500;i++){let
   t=1+9*i/499;R.push([t,Math.sin(t*t),t*Math.sin(t)])}wr('ej13_mix.dat'
let
   p=spawnSync('gnuplot',['ej13.gnu'],{encoding:'utf8'});if(p.status!==0
   new Error(p.stderr);
```

En un gráfico log–log de t^2, $\log(t^2) = 2\log t$: por eso la pendiente es 2. En cualquier escala logarítmica sólo pueden representarse valores finitos y estrictamente positivos en el eje correspondiente; ceros, negativos, infinitos y NaN deben excluirse, marcarse como ausentes o tratarse mediante una transformación matemáticamente justificada.

14. **Ejercicio:** Transformación y curvas polares

    Para una curva expresada en coordenadas polares (r, θ) se puede utilizar la transformación

    $$x = r\cos\theta, \qquad y = r\sin\theta.$$

    a) Implemente una operación reutilizable que convierta uno o varios pares (r, θ) a coordenadas cartesianas (x, y).

    b) Represente

    $$r = \sin^2\theta, \qquad -2\pi \le \theta \le 2\pi.$$

    c) Represente en una misma figura

    $$r_1 = \cos^2\theta$$

    y

    $$r_2 = \frac{\cos^2\theta}{1+\sin^2\theta}.$$

    d) Represente una familia de curvas

    $$r = k - \sin^2\theta - \cos^2\theta, \qquad k > 1,$$

    para al menos seis valores de k. Utilice colores diferentes y mantenga igual escala en ambos ejes cartesianos.

    e) Simplifique analíticamente la última expresión y explique por qué la familia obtenida está formada por círculos concéntricos.

    Nota Gnuplot: Puede emplearse `set polar`, `set trange`, `set size ratio -1` y `set grid polar`. El parámetro angular de Gnuplot se escribe como `t`.

    Nota Gnuplot: Gnuplot puede representar directamente curvas polares mediante `set polar`; el parámetro angular se expresa como `t`.

Scilab. En Scilab puede hacerse la conversión $x = r\cos\theta$, $y = r\sin\theta$ mediante una función auxiliar, por ejemplo `polares`, y trabajar luego con las coordenadas cartesianas.

```
# r = sin^2(theta) directamente en Gnuplot
gnuplot -e "
set terminal png size 600,600
set output '/tmp/polar_gnuplot.png'
set polar
set size ratio -1
set trange [-2*pi:2*pi]
set title 'r = sin^2(theta)'
set grid polar
unset border
plot sin(t)**2 with lines lw 2 lc 'blue' notitle
"
```

Scilab. Como alternativa, la curva polar puede convertirse a coordenadas cartesianas y calcularse en Scilab:

```
// polares.sce  -  conversion de polares a cartesianas
// equivale a polares.sci de Scilab

function [x, y] = polares(r, theta)
    x = r .* cos(theta)
    y = r .* sin(theta)
endfunction

// ejemplo: r = sin^2(theta)
theta = linspace(-2*%pi, 2*%pi, 500)
r     = sin(theta) .^ 2
[x, y] = polares(r, theta)
```

```
set terminal png size 600,600
set output "/tmp/polar_a.png"
set title  "r = sin^2(theta)"
set xlabel "x"
set ylabel "y"
set size ratio -1
set grid
plot "/tmp/polar_a.txt" with lines lw 2 lc "blue" notitle
```

Scilab. Implementación o cálculo auxiliar en Scilab:

```
// guardar para Gnuplot
//    mfprintf(fd, '%.8f %.8f\n', x(i), y(i))
host('gnuplot /tmp/polar_a.gnu')
printf('Grafica: /tmp/polar_a.png\n')
```

```
function [x, y] = polares(r, theta)
    x = r .* cos(theta)
```

```
    y = r .* sin(theta)
endfunction

theta = linspace(-2*%pi, 2*%pi, 500)

// curva 1: r = cos^2(theta)
r1      = cos(theta) .^ 2
[x1,y1] = polares(r1, theta)

// curva 2: r = cos^2(theta) / (1 + sin^2(theta))
r2      = cos(theta).^2 ./ (1 + sin(theta).^2)
[x2,y2] = polares(r2, theta)

// guardar ambas curvas '/tmp/polar_b1.txt'
//    mfprintf(fd, '%.8f %.8f\n', x1(i), y1(i))
fprintfMat('/tmp/polar_b1.txt', [x1(:) y1(:)], '%15.8f')
// guardar ambas curvas '/tmp/polar_b2.txt'
// mfprintf(fd, '%.8f %.8f\n', x2(i), y2(i))
fprintfMat('/tmp/polar_b2.txt', [x2(:) y2(:)], '%15.8f')
```

```
set terminal png size 700,600
set output "/tmp/polar_b.png"
set title  "Curvas en coord. cartesianas"
set xlabel "x"
set ylabel "y"
set size ratio -1
set grid
set key top right
plot "/tmp/polar_b1.txt" with lines lw 2 lc "red"  title
   "r=cos^2(t)", \
     "/tmp/polar_b2.txt" with lines lw 2 lc "blue" title
        "r=cos^2(t)/(1+sin^2(t))"
```

```
host('gnuplot /tmp/polar_b.gnu')
printf('Grafica: /tmp/polar_b.png\n')
```

Scilab. Para automatizar una familia de curvas puede resultar cómodo generar los archivos y el script de Gnuplot desde Scilab.

```
function [x, y] = polares(r, theta)
    x = r .* cos(theta)
    y = r .* sin(theta)
endfunction

theta = linspace(0, 2*%pi, 500)

// valores de k: k > 1 para que r > 0
k_vec = 1.5 : 0.5 : 4.0    // k = 1.5, 2.0, 2.5, 3.0, 3.5,
   4.0
nk    = length(k_vec)
```

```
// colores RGB variables: variar el tono de azul a rojo
// cada color es una cadena '#rrggbb'
function s = rgb2hex(r, g, b)
    // r,g,b en [0,1]
    ri = int(r*255); gi = int(g*255); bi = int(b*255)
    s = sprintf('#%02x%02x%02x', ri, gi, bi)
endfunction

NL = ascii(10); Q = ascii(34)

// generar un archivo por circulo y acumular el script
   gnuplot
gp = 'set terminal png size 700,700' + NL
gp = gp + 'set output ' + Q + '/tmp/polar_c.png' + Q + NL
gp = gp + 'set title  ' + Q + ..
          'r = k - sin^2(t) - cos^2(t)' + Q + NL
gp = gp + 'set xlabel ' + Q + 'x' + Q + NL
gp = gp + 'set ylabel ' + Q + 'y' + Q + NL
gp = gp + 'set size ratio -1' + NL
gp = gp + 'set grid' + NL
gp = gp + 'set key outside right' + NL

for j = 1:nk
    k    = k_vec(j)
    // r = k - sin^2 - cos^2 = k - 1  (identidad
       trigonometrica)
    r    = (k - 1) * ones(theta)
    [x, y] = polares(r, theta)

    // color: interpolacion de azul (j=1) a rojo (j=nk)
    t_color = (j-1) / max(nk-1, 1)
    color   = rgb2hex(t_color, 0.2, 1 - t_color)

    // guardar datos del circulo j
    nombre = sprintf('/tmp/circ_%02d.txt', j)
    fprintfMat(nombre, [x(:) y(:)], '%15.8f')

    etiq = sprintf('k=%.1f  r=%.1f', k, k-1)
    DAT  = Q + nombre + Q

    if j == 1 then
        gp = gp + 'plot ' + DAT + ' with lines lw 2 lc ' + ..
                  Q+color+Q+' title '+Q+etiq+Q
    else
        gp = gp + ', \' + NL
        gp = gp + '     ' + DAT + ' with lines lw 2 lc ' + ..
                  Q+color+Q+' title '+Q+etiq+Q
    end
end
gp = gp + NL
```

```
fd = mopen('/tmp/polar_c.gnu','w'); mfprintf(fd,'%s',gp);
   mclose(fd)
host('gnuplot /tmp/polar_c.gnu')
printf('Grafica: /tmp/polar_c.png\n')

// nota: sin^2+cos^2=1, entonces r=k-1
// los circulos tienen radio: ')
for j = 1:nk
    printf('  k=%.1f  ->  r=%.1f\n', k_vec(j), k_vec(j)-1)
end
```

Scilab. En Scilab, una forma equivalente de especificar el color es usar un vector `[R G B]` en el mapa de colores.

Scilab. Si se desea conservar la transformación en un archivo separado, el archivo original propone guardarla como `polares.sci` y cargarla desde el directorio de trabajo con `exec`. Por ejemplo:

```
exec('home\alumno\polares.sci');
```

Scilab. Para la familia de círculos, `eu07b` propone además variar explícitamente un vector `[RGB]` para asignar colores distintos.

Respuesta (L: diez modalidades).

La operación reutilizable es

$$(r, \theta) \longmapsto (r\cos\theta, r\sin\theta).$$

Para la familia del último inciso,

$$r = k - \sin^2\theta - \cos^2\theta = k - 1,$$

de modo que cada valor de k produce un círculo centrado en el origen y de radio $k-1$. Se usan $k = 1{,}5, 2, 2{,}5, 3, 3{,}5, 4$.

Scilab, gráficos nativos

```
function [x,y]=pol(r,t), x=r.*cos(t); y=r.*sin(t),
   endfunction
t=linspace(-2*%pi,2*%pi,800)
scf(140);clf();subplot(1,3,1);r=sin(t).^2;[x,y]=pol(r,t);plot(x,y);gca()
subplot(1,3,2);r=cos(t).^2;[x1,y1]=pol(r,t);r=cos(t).^2./(1+sin(t).^2);[
subplot(1,3,3);a=gca();a.auto_clear='off';for
   k=1.5:0.5:4,r=(k-1)*ones(t);[x,y]=pol(r,t);plot(x,y);end;a.auto_clear
xs2png(gcf(),'ej14_scilab.png')
```

GNU Octave, gráficos nativos

```
pol=@(r,t) deal(r.*cos(t),r.*sin(t));
   t=linspace(-2*pi,2*pi,800);
figure(140);clf;
subplot(1,3,1);[x,y]=pol(sin(t).^2,t);plot(x,y);axis
   equal;grid on;
```

```
subplot(1,3,2);[x1,y1]=pol(cos(t).^2,t);[x2,y2]=pol(cos(t).^2./(1+sin(t)
   equal;grid on;legend('r1','r2');
subplot(1,3,3);hold on;for
   k=1.5:.5:4,[x,y]=pol((k-1)*ones(size(t)),t);plot(x,y);end;hold
   off;axis equal;grid on;
print('ej14_octave.png','-dpng','-r140');
```

Python, gráficos nativos

```
import numpy as np, matplotlib.pyplot as plt
def pol(r,t): return r*np.cos(t),r*np.sin(t)
t=np.linspace(-2*np.pi,2*np.pi,800);fig,ax=plt.subplots(1,3,figsize=(12,
x,y=pol(np.sin(t)**2,t);ax[0].plot(x,y)
x1,y1=pol(np.cos(t)**2,t);x2,y2=pol(np.cos(t)**2/(1+np.sin(t)**2),t);ax[
for k in np.arange(1.5,4.01,.5):
   x,y=pol(np.full_like(t,k-1),t);ax[2].plot(x,y,label=f'k={k:g}')
for a in ax:a.set_aspect('equal');a.grid()
ax[2].legend(fontsize=7);fig.tight_layout();fig.savefig('ej14_python.png
```

Perl, gráficos nativos mediante SVG

```
use strict;use warnings;my$pi=4*atan2(1,1);
sub pol{my($r,$t)=@_;($r*cos($t),$r*sin($t))}
open my$s,'>','ej14_perl.svg' or die$!;print$s '<svg
   xmlns="http://www.w3.org/2000/svg" width="700"
   height="700"><rect width="100%" height="100%"
   fill="white"/>';
for
   my$k(1.5,2,2.5,3,3.5,4){my@p;for(0..600){my$t=2*$pi*$_/600;my($x,$y)=
   '<polyline fill="none" stroke="black" points="',join('
   ',@p),'"/>';}
print$s '</svg>';close$s;
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');
const
   t=Array.from({length:800},(_,i)=>-2*Math.PI+4*Math.PI*i/799);
const
   pol=r=>({x:t.map((q,i)=>r[i]*Math.cos(q)),y:t.map((q,i)=>r[i]*Math.si
let data=[];for(let k=1.5;k<=4.001;k+=.5){let
   z=pol(t.map(()=>k-1));z.name=`k=${k}`;data.push(z)}
fs.writeFileSync('ej14_node.html',`<!doctype
   html><script>${P}</script><div
   id="p"></div><script>Plotly.newPlot('p',${JSON.stringify(data)},{xaxi
   r=k-1'});</script>`);
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot

```
set terminal pngcairo size 1100,400
set output 'ej14_gnuplot.png'
```

```
set multiplot layout 1,3
set size ratio -1; set grid
set title 'r=sin^2(t)'; plot 'ej14_a.dat' u 1:2 w l notitle
set title 'dos curvas'; plot 'ej14_b1.dat' u 1:2 w l t 'r1',
   'ej14_b2.dat' u 1:2 w l t 'r2'
set title 'r=k-1'; plot for [i=0:5] 'ej14_c.dat' index i u
   1:2 w l t sprintf('k=%.1f',1.5+.5*i)
unset multiplot;unset output
```

Scilab + Gnuplot

```
function [x,y]=pol(r,t),x=r.*cos(t);y=r.*sin(t),endfunction
t=linspace(-2*%pi,2*%pi,800);r=sin(t).^2;[x,y]=pol(r,t);fprintfMat('ej14
r=cos(t).^2;[x,y]=pol(r,t);fprintfMat('ej14_b1.dat',[x(:)y(:)],'%15.10g'
fd=mopen('ej14_c.dat','w');for
   k=1.5:0.5:4,r=(k-1)*ones(t);[x,y]=pol(r,t);for
   i=1:length(t),mfprintf(fd,'%.12g
   %.12g\n',x(i),y(i));end;mfprintf(fd,'\n\n');end;mclose(fd)
[e,o,er]=host('gnuplot ej14.gnu');if e<>0 then error(er),end
```

GNU Octave + Gnuplot

```
t=linspace(-2*pi,2*pi,800)';pol=@(r)[r.*cos(t),r.*sin(t)];dlmwrite('ej14
   ');dlmwrite('ej14_b1.dat',pol(cos(t).^2),'delimiter','
   ');dlmwrite('ej14_b2.dat',pol(cos(t).^2./(1+sin(t).^2)),'delimiter','
   ');
fid=fopen('ej14_c.dat','w');for
   k=1.5:.5:4,A=pol((k-1)*ones(size(t)));fprintf(fid,'%.12g
   %.12g\n',A');fprintf(fid,'\n\n');end;fclose(fid);system('gnuplot
   ej14.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
t=np.linspace(-2*np.pi,2*np.pi,800)
def pol(r):return np.c_[r*np.cos(t),r*np.sin(t)]
np.savetxt('ej14_a.dat',pol(np.sin(t)**2));np.savetxt('ej14_b1.dat',pol(
with open('ej14_c.dat','w') as f:
 for k in np.arange(1.5,4.01,.5):
    np.savetxt(f,pol(np.full_like(t,k-1)));f.write('\n\n')
subprocess.run(['gnuplot','ej14.gnu'],check=True)
```

Perl + Gnuplot

```
use strict;use warnings;my$pi=4*atan2(1,1);sub
   xy{my($r,$t)=@_;($r*cos($t),$r*sin($t))}
sub curve{my($n,$rf)=@_;open my$h,'>',$n or
   die$!;for(0..799){my$t=-2*$pi+4*$pi*$_/799;my($x,$y)=xy($rf->($t),$t)
   $y\n"}close$h}
curve('ej14_a.dat',sub{sin($_[0])**2});curve('ej14_b1.dat',sub{cos($_[0]
open my$h,'>','ej14_c.dat' or die$!;for
   my$k(1.5,2,2.5,3,3.5,4){for(0..799){my$t=-2*$pi+4*$pi*$_/799;my($x,$y
```

```
    $y\n"}print$h"\n\n"}close$h;system('gnuplot','ej14.gnu')==0
    or die;
```

JavaScript/Node.js + Gnuplot

```
const
    fs=require('node:fs'),{spawnSync}=require('node:child_process');const
    N=800,t=Array.from({length:N},(_,i)=>-2*Math.PI+4*Math.PI*i/(N-1));co
    xy=(r,q)=>[r*Math.cos(q),r*Math.sin(q)];
function
    wr(n,rf){fs.writeFileSync(n,t.map(q=>xy(rf(q),q).join('
    ')).join('\n')+'\n')}wr('ej14_a.dat',q=>Math.sin(q)**2);wr('ej14_b1.d
let s='';for(let
    k=1.5;k<=4.001;k+=.5)s+=t.map(q=>xy(k-1,q).join('
    ')).join('\n')+'\n\n\n';fs.writeFileSync('ej14_c.dat',s);let
    r=spawnSync('gnuplot',['ej14.gnu'],{encoding:'utf8'});if(r.status!==0
    new Error(r.stderr);
```

15. **Ejercicio:** Incertidumbre experimental y barras de error

Considere las siguientes mediciones de una magnitud que, según el modelo teórico, debería seguir la función $y = \sin x$:

```
x       y_medido    error
0.0      0.03       0.05
0.5      0.46       0.06
1.0      0.85       0.05
1.5      0.99       0.04
2.0      0.93       0.05
2.5      0.60       0.06
3.0      0.11       0.05
3.5     -0.36       0.06
4.0     -0.76       0.05
4.5     -0.97       0.04
```

a) Represente las mediciones mediante puntos con barras de error verticales.

b) Superponga la curva teórica $y = \sin x$ en el mismo sistema de ejes.

c) Determine para cada medición si el valor teórico cae dentro del intervalo $y_{\text{medido}} \pm \text{error}$.

d) Diferencie claramente en la leyenda los datos experimentales y el modelo.

e) Interprete qué significa que una barra de error incluya o no incluya el valor predicho por el modelo. No interprete una observación aislada como una prueba concluyente de validez o invalidez del modelo.

```
# x     y_medido    error
0.0     0.03        0.05
0.5     0.46        0.06
1.0     0.85        0.05
```

```
1.5     0.99        0.04
2.0     0.93        0.05
2.5     0.60        0.06
3.0     0.11        0.05
3.5    -0.36        0.06
4.0    -0.76        0.05
4.5    -0.97        0.04
```

Nota Gnuplot: Si los datos están en `experimento.txt`, las columnas pueden seleccionarse con `using 1:2:3` y las barras verticales con `yerrorbars`:

```
plot 'experimento.txt' using 1:2:3 with yerrorbars \
     pt 7 ps 1 lc 'red' title 'Mediciones', \
     sin(x) with lines lw 2 lc 'blue' title 'sin(x)'
```

Respuesta (T: diez modalidades).

Para cada punto se calcula $y_{\text{terico}} = \sin x$ y se comprueba

$$y_{\text{medido}} - e \le \sin x \le y_{\text{medido}} + e.$$

Con los datos propuestos, **las diez predicciones teóricas quedan dentro de sus respectivas barras de error**. Esto es compatible con el modelo en esta prueba puntual, pero no constituye por sí solo una demostración de que el modelo sea correcto.

x	y_m	e	$\sin x$	dentro
0.0	0.03	0.05	0.0000	sí
0.5	0.46	0.06	0.4794	sí
1.0	0.85	0.05	0.8415	sí
1.5	0.99	0.04	0.9975	sí
2.0	0.93	0.05	0.9093	sí
2.5	0.60	0.06	0.5985	sí
3.0	0.11	0.05	0.1411	sí
3.5	-0.36	0.06	-0.3508	sí
4.0	-0.76	0.05	-0.7568	sí
4.5	-0.97	0.04	-0.9775	sí

Scilab, gráficos nativos

```
x=[0 .5 1 1.5 2 2.5 3 3.5 4 4.5]; y=[.03 .46 .85 .99 .93 .60
   .11 -.36 -.76 -.97]; e=[.05 .06 .05 .04 .05 .06 .05 .06
   .05 .04]; yt=sin(x)
inside=(yt>=y-e)&(yt<=y+e);disp([x' y' e' yt' inside'])
scf(150);clf();errbar(x,y,e,e);a=gca();a.auto_clear='off';xx=linspace(0,
```

GNU Octave, gráficos nativos

```
x=0:.5:4.5;y=[.03 .46 .85 .99 .93 .60 .11 -.36 -.76
   -.97];e=[.05 .06 .05 .04 .05 .06 .05 .06 .05
   .04];yt=sin(x);inside=(yt>=y-e)&(yt<=y+e);disp([x' y' e'
   yt' inside']);
```

```
errorbar(x,y,e,'o');hold
   on;xx=linspace(0,4.5,500);plot(xx,sin(xx),'linewidth',1.5);hold
   off;grid
   on;legend('mediciones','sin(x)');print('ej15_octave.png','-dpng');
```

Python, gráficos nativos

```
import numpy as np,matplotlib.pyplot as plt
x=np.arange(0,5,.5);y=np.array([.03,.46,.85,.99,.93,.60,.11,-.36,-.76,-.
xx=np.linspace(0,4.5,500);plt.errorbar(x,y,yerr=e,fmt='o',capsize=3,labe
```

Perl, gráficos nativos mediante SVG

```
use strict;use
   warnings;my@x=(0,.5,1,1.5,2,2.5,3,3.5,4,4.5);my@y=(.03,.46,.85,.99,.9
open my$s,'>','ej15_perl.svg' or die$!;print$s '<svg
   xmlns="http://www.w3.org/2000/svg" width="800"
   height="500"><rect width="100%" height="100%"
   fill="white"/>';sub X{50+700*$_[0]/4.5}sub
   Y{250-190*$_[0]}
my@p;for(0..500){my$x=4.5*$_/500;push@p,X($x).','.Y(sin($x))}print$s
   '<polyline fill="none" stroke="black" points="',join('
   ',@p),'"/>';
for
   my$i(0..$#x){my$ok=(sin($x[$i])>=$y[$i]-$e[$i]&&sin($x[$i])<=$y[$i]+$
   "$x[$i]
   ",($ok?'dentro':'fuera'),"\n";my$X=X($x[$i]);print$s
   '<line stroke="gray" x1="'.$X.'" x2="'.$X.'"
   y1="'.Y($y[$i]-$e[$i]).'"
   y2="'.Y($y[$i]+$e[$i]).'"/><circle cx="'.$X.'"
   cy="'.Y($y[$i]).'" r="3"/>';}
print$s '</svg>';close$s;
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');const
   x=[0,.5,1,1.5,2,2.5,3,3.5,4,4.5],y=[.03,.46,.85,.99,.93,.60,.11,-.36,
   xx=Array.from({length:500},(_,i)=>4.5*i/499);const
   data=[{x,y,mode:'markers',name:'mediciones',error_y:{type:'data',arra
   html><script>${P}</script><div
   id="p"></div><script>Plotly.newPlot('p',${JSON.stringify(data)},{titl
   y modelo'});</script>`);
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot

```
set terminal pngcairo size 800,500
set output 'ej15_gnuplot.png'
set grid;set xlabel 'x';set ylabel 'y';set title 'Mediciones
   y modelo'
plot 'ej15.dat' u 1:2:3 w yerrorbars pt 7 ps 1.2 t
   'mediciones', \
```

```
      'ej15.dat' u 1:4 w l lw 2 t 'sin(x)'
unset output
```

Scilab + Gnuplot

```
x=[0 .5 1 1.5 2 2.5 3 3.5 4 4.5]';y=[.03 .46 .85 .99 .93 .60
   .11 -.36 -.76 -.97]';e=[.05 .06 .05 .04 .05 .06 .05 .06
   .05
   .04]';yt=sin(x);inside=(yt>=y-e)&(yt<=y+e);fprintfMat('ej15.dat',[x
   y e yt inside],'%15.10g');[st,o,er]=host('gnuplot
   ej15.gnu');if st<>0 then error(er),end
```

GNU Octave + Gnuplot

```
x=(0:.5:4.5)';y=[.03 .46 .85 .99 .93 .60 .11 -.36 -.76
   -.97]';e=[.05 .06 .05 .04 .05 .06 .05 .06 .05
   .04]';yt=sin(x);inside=(yt>=y-e)&(yt<=y+e);dlmwrite('ej15.dat',[x
   y e yt inside],'delimiter',' ');system('gnuplot
   ej15.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
x=np.arange(0,5,.5);y=np.array([.03,.46,.85,.99,.93,.60,.11,-.36,-.76,-.
```

Perl + Gnuplot

```
use strict;use
   warnings;my@x=(0,.5,1,1.5,2,2.5,3,3.5,4,4.5);my@y=(.03,.46,.85,.99,.9
   my$h,'>','ej15.dat' or
   die$!;for(0..$#x){my$t=sin($x[$_]);my$ok=($t>=$y[$_]-$e[$_]&&$t<=$y[$
   $y[$_] $e[$_] $t
   $ok\n"}close$h;system('gnuplot','ej15.gnu')==0 or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');const
   x=[0,.5,1,1.5,2,2.5,3,3.5,4,4.5],y=[.03,.46,.85,.99,.93,.60,.11,-.36,
   t=Math.sin(v),ok=t>=y[i]-e[i]&&t<=y[i]+e[i]?1:0;return
   `${v} ${y[i]} ${e[i]} ${t} ${ok}`}).join('\n')+'\n');let
   r=spawnSync('gnuplot',['ej15.gnu'],{encoding:'utf8'});if(r.status!==0
   new Error(r.stderr);
```

16. **Ejercicio:** Regresión lineal de datos experimentales

    Considere los datos:

```
x       y
0.0     1.10
0.5     1.95
1.0     2.85
```

```
1.5     3.82
2.0     4.65
2.5     5.55
3.0     6.48
3.5     7.30
4.0     8.25
4.5     9.05
```

Ajuste los datos mediante el modelo

$$y = ax + b.$$

a) Determine los parámetros a y b mediante mínimos cuadrados.

b) Represente los puntos experimentales y la recta ajustada en una misma figura y exporte al menos esa figura a PNG.

c) Muestre la ecuación obtenida en la leyenda, título o una anotación dentro del gráfico.

d) Calcule y represente, en una segunda figura, los residuos

$$r_i = y_i - (ax_i + b),$$

incluyendo como referencia la recta horizontal $r = 0$.

e) Explique si el ajuste lineal parece razonable a partir de la figura de los datos y de la distribución de los residuos; compare explícitamente la conclusión obtenida al mirar sólo la recta ajustada con la que surge del gráfico de residuos.

```
# x     y
0.0     1.10
0.5     1.95
1.0     2.85
1.5     3.82
2.0     4.65
2.5     5.55
3.0     6.48
3.5     7.30
4.0     8.25
4.5     9.05
```

Nota Gnuplot: Una forma mínima de realizar el ajuste es:

```
f(x) = a*x + b
fit f(x) 'regresion.txt' using 1:2 via a,b
plot 'regresion.txt' using 1:2 with points title 'datos', \
     f(x) with lines title 'ajuste lineal'
```

Respuesta (T: diez modalidades).

Para los diez datos, el ajuste por mínimos cuadrados da

$$\boxed{a = 1{,}779151515\ldots}, \qquad \boxed{b = 1{,}096909091\ldots}.$$

El coeficiente de determinación es aproximadamente $R^2 = 0{,}99981875$. Los residuos, en el orden de los datos, son

$$0{,}00309,\ -0{,}03648,\ -0{,}02606,\ 0{,}05436,\ -0{,}00521,\ 0{,}00521,\ 0{,}04564,\ -0{,}02394,\ 0{,}03648,\ -0{,}05309.$$

Son pequeños y alternan alrededor de cero sin una tendencia evidente; por eso el modelo lineal resulta razonable para estos datos. El gráfico de la recta por sí solo puede ocultar pequeñas estructuras; el gráfico de residuos es el control más sensible.

Scilab, gráficos nativos

```
x=(0:.5:4.5)';y=[1.10 1.95 2.85 3.82 4.65 5.55 6.48 7.30
   8.25 9.05]';A=[x
   ones(x)];p=A\y;a=p(1);b=p(2);yf=a*x+b;r=y-yf;printf('a=%.10f
   b=%.10f\n',a,b)
scf(160);clf();subplot(1,2,1);plot(x,y,'ro',x,yf,'b-');xgrid();legend('d
   max(x)],[0
   0],'r--');a0.auto_clear='on';xgrid();xs2png(gcf(),'ej16_scilab.png')
```

GNU Octave, gráficos nativos

```
x=(0:.5:4.5)';y=[1.10 1.95 2.85 3.82 4.65 5.55 6.48 7.30
   8.25 9.05]';p=[x
   ones(size(x))]\y;a=p(1);b=p(2);yf=a*x+b;r=y-yf;printf('a=%.10f
   b=%.10f\n',a,b);
figure(160);subplot(1,2,1);plot(x,y,'ro',x,yf,'b-');grid
   on;legend('datos',sprintf('y=%.4fx%+.4f',a,b));subplot(1,2,2);plot(x,
   on;plot([min(x) max(x)],[0 0],'r--');hold off;grid
   on;print('ej16_octave.png','-dpng');
```

Python, gráficos nativos

```
import numpy as np,matplotlib.pyplot as plt
x=np.arange(0,5,.5);y=np.array([1.10,1.95,2.85,3.82,4.65,5.55,6.48,7.30,
fig,ax=plt.subplots(1,2,figsize=(10,4));ax[0].plot(x,y,'o',label='datos'
```

Perl, gráficos nativos mediante SVG

```
use strict;use
   warnings;my@x=(0,.5,1,1.5,2,2.5,3,3.5,4,4.5);my@y=(1.10,1.95,2.85,3.8
   b=$b\n";
open my$s,'>','ej16_perl.svg' or die$!;print$s '<svg
   xmlns="http://www.w3.org/2000/svg" width="900"
   height="450"><rect width="100%" height="100%"
   fill="white"/>';sub X{50+350*$_[0]/4.5}sub
   Y{400-35*$_[0]}print$s '<line x1="'.X(0).'"
   y1="'.Y($b).'" x2="'.X(4.5).'" y2="'.Y($a*4.5+$b).'"
   stroke="black"/>';for(0..$#x){print$s '<circle
   cx="'.X($x[$_]).'" cy="'.Y($y[$_]).'" r="3"/>'}print$s
   '</svg>';close$s;
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');const
   x=[0,.5,1,1.5,2,2.5,3,3.5,4,4.5],y=[1.10,1.95,2.85,3.82,4.65,5.55,6.4
   data=[{x,y,mode:'markers',name:'datos'},{x,y:fit,mode:'lines',name:`y
   html><script>${P}</script><div id="p"></div><div
   id="r"></div><script>Plotly.newPlot('p',${JSON.stringify(data)},{titl
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot

```
set terminal pngcairo size 900,400
set output 'ej16_gnuplot.png'
set multiplot layout 1,2
set grid
set title 'datos y ajuste'
plot 'ej16.dat' u 1:2 w p pt 7 t 'datos', '' u 1:3 w l lw 2
   t sprintf('y=%.4fx%+.4f',a,b)
set title 'residuos'
plot 'ej16.dat' u 1:4 w lp pt 7 t 'r_i', 0 w l dt 2 t 'r=0'
unset multiplot;unset output
```

Cada lenguaje calcula a y b, escribe en `ej16.dat` las columnas $x, y, ax+b, r$, y construye un pequeño archivo `ej16_pars.gnu` con las asignaciones numéricas de a y b antes de ejecutar el script anterior.

Scilab + Gnuplot

```
x=(0:.5:4.5)';y=[1.10 1.95 2.85 3.82 4.65 5.55 6.48 7.30
   8.25 9.05]';p=[x
   ones(x)]\y;a=p(1);b=p(2);yf=a*x+b;r=y-yf;fprintfMat('ej16.dat',[x
   y yf
   r],'%15.10g');fd=mopen('ej16_pars.gnu','w');mfprintf(fd,'a=%.17g\nb=%
   "ej16_plot.gnu"\n',a,b);mclose(fd);[e,o,er]=host('gnuplot
   ej16_pars.gnu');if e<>0 then error(er),end
```

GNU Octave + Gnuplot

```
x=(0:.5:4.5)';y=[1.10 1.95 2.85 3.82 4.65 5.55 6.48 7.30
   8.25 9.05]';p=[x
   ones(size(x))]\y;a=p(1);b=p(2);yf=a*x+b;r=y-yf;dlmwrite('ej16.dat',[x
   y yf r],'delimiter','
   ');fid=fopen('ej16_pars.gnu','w');fprintf(fid,'a=%.17g\nb=%.17g\nload
   "ej16_plot.gnu"\n',a,b);fclose(fid);system('gnuplot
   ej16_pars.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
x=np.arange(0,5,.5);y=np.array([1.10,1.95,2.85,3.82,4.65,5.55,6.48,7.30,
   "ej16_plot.gnu"\n');subprocess.run(['gnuplot','ej16_pars.gnu'],check=
```

Perl + Gnuplot

```
use strict;use
   warnings;my@x=(0,.5,1,1.5,2,2.5,3,3.5,4,4.5);my@y=(1.10,1.95,2.85,3.8
   my$d,'>','ej16.dat' or
   die$!;for(0..$#x){my$f=$a*$x[$_]+$b;print$d"$x[$_] $y[$_]
   $f ",($y[$_]-$f),"\n"}close$d;open
   my$p,'>','ej16_pars.gnu' or
   die$!;print$p"a=$a\nb=$b\nload
   'ej16_plot.gnu'\n";close$p;system('gnuplot','ej16_pars.gnu')==0
   or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');const
   x=[0,.5,1,1.5,2,2.5,3,3.5,4,4.5],y=[1.10,1.95,2.85,3.82,4.65,5.55,6.4
   f=a*v+b;return `${v} ${y[i]} ${f}
   ${y[i]-f}`}).join('\n')+'\n');fs.writeFileSync('ej16_pars.gnu',`a=${a
   'ej16_plot.gnu'\n`);let
   q=spawnSync('gnuplot',['ej16_pars.gnu'],{encoding:'utf8'});if(q.statu
   new Error(q.stderr);
```

17. **Ejercicio:** Distribución normal e histograma

Genere 1000 realizaciones de una variable aleatoria normal estándar $N(0,1)$. Registre la semilla utilizada para que el experimento pueda reproducirse.

a) Construya un histograma con ancho de clase aproximadamente igual a 0,25.

b) Superponga la densidad normal teórica

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2},$$

escalándola de manera coherente con la normalización elegida para el histograma.

c) Repita la figura empleando otra cantidad de clases y compare el resultado.

d) Calcule la media y la desviación estándar de la muestra y muéstrelas en el informe o en una anotación del gráfico.

e) Explique por qué un histograma obtenido con una muestra finita no coincide exactamente con la curva teórica.

Nota Gnuplot: Puede generarse una normal a partir de `rand(0)` mediante la transformación de Box–Muller y acumular frecuencias con `smooth frequency`:

```
set terminal png size 700,400
set output '/tmp/histograma_normal.png'

set print '/tmp/normales.txt'
do for [i=1:1000] {
    u1 = rand(0); u2 = rand(0)
```

```
    z  = sqrt(-2*log(u1)) * cos(2*pi*u2)
    print z
}
unset print

set title 'Distribucion normal (n=1000)'
set xlabel 'Valor'
set ylabel 'Frecuencia'
set boxwidth 0.25
set style fill solid 0.7
binw = 0.25
bin(x) = binw * floor(x/binw) + binw/2.0
plot '/tmp/normales.txt' using (bin($1)):(1.0) \
     smooth frequency with boxes \
     lc 'steelblue' notitle
```

Respuesta (T: diez modalidades).

Se fija la semilla 20260822. La semilla hace reproducible la muestra dentro de cada entorno; no implica que bibliotecas distintas produzcan la misma secuencia. Para comparar con $\varphi(x)$ se normaliza el histograma como *densidad*, de modo que su área total sea aproximadamente uno. Con ancho $\Delta = 0{,}25$, una frecuencia de n_j observaciones se representa con altura $n_j/(1000\Delta)$.

Scilab, gráficos nativos

```
grand('setsd',20260822)
z=grand(1000,1,'nor',0,1)
mu=mean(z); sd=stdev(z)
scf(170);clf();
histplot(-4:0.25:4,z,normalization=%t)
a=gca();a.auto_clear='off'; xx=linspace(-4,4,600)
phi=exp(-xx.^2/2)/sqrt(2*%pi);plot(xx,phi,'r-')
a.auto_clear='on';xgrid();xtitle(msprintf('N(0,1):
   media=%.3f sd=%.3f',mu,sd),'x','densidad')
xs2png(gcf(),'ej17_scilab_025.png')
scf(171);clf();histplot(-4:0.40:4,z,normalization=%t);xgrid();xs2png(gcf
```

GNU Octave, gráficos nativos

```
rng(20260822,'twister'); z=randn(1000,1); mu=mean(z);
   sd=std(z);
c=-4:.25:4; [n,c]=hist(z,c); bar(c,n/(sum(n)*.25),1); hold
   on;
xx=linspace(-4,4,600);
   plot(xx,exp(-xx.^2/2)/sqrt(2*pi),'r-','linewidth',2);
hold off;grid
   on;xlabel('x');ylabel('densidad');title(sprintf('media=%.3f
   sd=%.3f',mu,sd));
print('ej17_octave_025.png','-dpng');
figure;
   c=-4:.4:4;[n,c]=hist(z,c);bar(c,n/(sum(n)*.4),1);grid
   on;print('ej17_octave_040.png','-dpng');
```

Python, gráficos nativos

```
import numpy as np, matplotlib.pyplot as plt
rng=np.random.default_rng(20260822); z=rng.normal(size=1000)
mu=z.mean(); sd=z.std(ddof=1); xx=np.linspace(-4,4,600)
for w,name in [(0.25,'025'),(0.40,'040')]:
    edges=np.arange(-4,4+w,w)
    plt.hist(z,bins=edges,density=True,alpha=.65,label=f'histograma,
       ancho={w}')
    plt.plot(xx,np.exp(-xx**2/2)/np.sqrt(2*np.pi),label='densidad
       N(0,1)')
    plt.grid();plt.legend();plt.xlabel('x');plt.ylabel('densidad')
    plt.title(f'media={mu:.3f},
       sd={sd:.3f}');plt.savefig(f'ej17_python_{name}.png',dpi=140);plt.
```

Perl, gráficos nativos con PDL/PLplot

```
use strict;use warnings;use PDL;use PDL::Graphics::PLplot;
srand(20260822);my@z;while(@z<1000){my$u1=rand();my$u2=rand();next
   if$u1==0;push@z,sqrt(-2*log($u1))*cos(8*atan2(1,1)*$u2)}
my$z=pdl(@z);my$mu=$z->avg;my$sd=sqrt((($z-$mu)**2)->sum/999);print
   "media=$mu sd=$sd\n";
my$xx=zeroes(500)->xlinvals(-4,4);my$phi=exp(-$xx*$xx/2)/sqrt(8*atan2(1,
my$pl=PDL::Graphics::PLplot->new(DEV=>'pngcairo',FILE=>'ej17_perl.png',S
$pl->histogram($z,32,SUBPAGE=>1,BOX=>[-4,4,0,140],TITLE=>'ancho
   aprox. 0.25');
$pl->xyplot($xx,$phi*1000*.25,SUBPAGE=>1,PLOTTYPE=>'LINE',COLOR=>'RED',P
$pl->histogram($z,20,SUBPAGE=>2,BOX=>[-4,4,0,220],TITLE=>'menos
   clases');
$pl->xyplot($xx,$phi*1000*.4,SUBPAGE=>2,PLOTTYPE=>'LINE',COLOR=>'RED',BO
```

Aquí PLplot dibuja frecuencias; por eso la densidad teórica se multiplica por $N\Delta$ en cada panel.

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');
function mulberry32(a){return()=>{let
   t=a+=0x6D2B79F5;t=Math.imul(t^t>>>15,t|1);t^=t+Math.imul(t^t>>>7,t|61
const u=mulberry32(20260822);let
   z=[];while(z.length<1000){let
   a=u(),b=u();if(a>0)z.push(Math.sqrt(-2*Math.log(a))*Math.cos(2*Math.P
const
   mu=z.reduce((a,b)=>a+b,0)/z.length,sd=Math.sqrt(z.reduce((a,v)=>a+(v-
const
   xx=Array.from({length:600},(_,i)=>-4+8*i/599),phi=xx.map(x=>Math.exp(
const data=[{x:z,type:'histogram',histnorm:'probability
   density',xbins:{start:-4,end:4,size:.25},name:'histograma'},{x:xx,y:p
fs.writeFileSync('ej17_node.html',`<!doctype
   html><script>${P}</script><div
   id="p"></div><script>Plotly.newPlot('p',${JSON.stringify(data)},{titl
   sd=${sd.toFixed(3)}'});</script>`);
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot. Cada lenguaje escribe sus 1000 valores en `ej17.dat`, una observación por línea. El script es el mismo:

```
set terminal pngcairo size 800,500
set output 'ej17_gnuplot_025.png'
N=1000.0; binw=.25; bin(x)=binw*floor(x/binw)+binw/2
phi(x)=exp(-x*x/2)/sqrt(2*pi)
set boxwidth binw;set style fill solid .65;set grid
set xlabel 'x';set ylabel 'densidad';set xrange [-4:4]
plot 'ej17.dat' u (bin($1)):(1.0/(N*binw)) smooth frequency
   w boxes t 'histograma', \
     phi(x) w l lw 2 t 'N(0,1)'
set output 'ej17_gnuplot_040.png'
binw=.40;set boxwidth binw
replot
unset output
```

Scilab + Gnuplot

```
grand('setsd',20260822);z=grand(1000,1,'nor',0,1);fprintfMat('ej17.dat',
   sd=%g\n',mean(z),stdev(z));[e,o,er]=host('gnuplot
   ej17.gnu');if e<>0 then error(er),end
```

GNU Octave + Gnuplot

```
rng(20260822,'twister');z=randn(1000,1);save('-ascii','ej17.dat','z');pr
   sd=%g\n',mean(z),std(z));system('gnuplot ej17.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
rng=np.random.default_rng(20260822);z=rng.normal(size=1000);np.savetxt('
```

Perl + Gnuplot

```
use strict;use warnings;srand(20260822);open
   my$f,'>','ej17.dat' or
   die$!;my@z;while(@z<1000){my$a=rand();my$b=rand();next
   if$a==0;my$v=sqrt(-2*log($a))*cos(8*atan2(1,1)*$b);push@z,$v;print$f"
   or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');funct
   R(a){return()=>{let
   t=a+=0x6D2B79F5;t=Math.imul(t^t>>>15,t|1);t^=t+Math.imul(t^t>>>7,t|61
   u=R(20260822),z=[];while(z.length<1000){let
   a=u(),b=u();if(a>0)z.push(Math.sqrt(-2*Math.log(a))*Math.cos(2*Math.P
   q=spawnSync('gnuplot',['ej17.gnu'],{encoding:'utf8'});if(q.status!==0
   new Error(q.stderr);
```

Al aumentar el ancho de clase disminuye el detalle y la variabilidad visual; al disminuirlo aumenta el detalle pero también el ruido. Con una muestra finita las frecuencias fluctúan aleatoriamente alrededor de las probabilidades teóricas, por lo que ningún histograma finito coincide exactamente con la densidad.

18. **Ejercicio:** Comparación entre gráfico de barras e histograma

Considere los siguientes conteos asociados a categorías:

categoría	Lunes	Martes	Miércoles	Jueves	Viernes
valor	42	38	55	61	47

y una muestra numérica continua de 1000 valores; puede reutilizar la muestra normal generada en el ejercicio anterior.

a) Represente los datos categóricos mediante un gráfico de barras y haga que el eje vertical comience en cero.

b) Represente la muestra continua mediante un histograma con aproximadamente 20 clases.

c) Cambie el orden de las categorías del gráfico de barras y explique qué significado tiene esa operación. Cambie luego el número de clases del histograma y explique por qué esa modificación no es conceptualmente equivalente.

d) Indique en cuál de las dos representaciones el ancho de cada barra tiene significado numérico y en cuál funciona sólo como separación visual entre categorías.

e) Explique por qué un gráfico de barras y un histograma pueden parecerse visualmente pero representan estructuras de datos diferentes.

Scilab. Para las categorías puede usarse `bar`; para la muestra continua, `histplot`.

Octave. Compare `bar` con `hist` o una función equivalente de histograma disponible en la instalación.

Python. Compare `bar` con `hist` de Matplotlib.

Perl. Use las operaciones de barras e histogramas disponibles en el backend gráfico elegido y conserve la distinción entre categorías y variable continua.

JavaScript. En Plotly.js compare una traza `bar` con una traza `histogram`.

Nota Gnuplot: Construya las barras categóricas con cajas y el histograma mediante agrupación por intervalos y frecuencias.

Respuesta (T: diez modalidades).

El gráfico de barras representa cinco *categorías*; el histograma divide un eje numérico continuo en intervalos. Reordenar Lunes–Viernes cambia sólo el orden de presentación de las categorías. Cambiar las clases de un histograma cambia la partición numérica usada para resumir la misma muestra. El ancho de las barras categóricas es principalmente gráfico; el ancho de una clase del histograma es una magnitud sobre el eje de la variable.

En las siguientes soluciones se reutiliza la muestra `z` del ejercicio 17, o se la vuelve a generar con la misma semilla.

Scilab, gráficos nativos

```
grand('setsd',20260822);z=grand(1000,1,'nor',0,1);v=[42 38
   55 61 47]
scf(180);clf();subplot(1,2,1);bar(v);gca().data_bounds=[0
   0;6 70];xgrid();xtitle('categorias')
subplot(1,2,2);histplot(20,z,normalization=%f);xgrid();xtitle('muestra
   continua')
xs2png(gcf(),'ej18_scilab.png')
// Reordenar categorias: por ejemplo v([5 1 2 3 4]); no
   altera los conteos.
```

GNU Octave, gráficos nativos

```
rng(20260822);z=randn(1000,1);v=[42 38 55 61 47];
subplot(1,2,1);bar(v);ylim([0
   70]);set(gca,'xtick',1:5,'xticklabel',{'Lu','Ma','Mi','Ju','Vi'});gri
   on;
subplot(1,2,2);hist(z,20);grid
   on;print('ej18_octave.png','-dpng');
```

Python, gráficos nativos

```
import numpy as np,matplotlib.pyplot as plt
rng=np.random.default_rng(20260822);z=rng.normal(size=1000);cats=['Lunes
fig,ax=plt.subplots(1,2,figsize=(10,4));ax[0].bar(cats,v);ax[0].set_ylim
```

Perl, gráficos nativos con PDL/PLplot

```
use strict;use warnings;use PDL;use PDL::Graphics::PLplot;
my$pl=PDL::Graphics::PLplot->new(DEV=>'pngcairo',FILE=>'ej18_perl.png',S
$pl->bargraph([qw(Lu Ma Mi Ju
   Vi)],pdl(42,38,55,61,47),SUBPAGE=>1,BOX=>[0,6,0,70]);
srand(20260822);my@z;while(@z<1000){my$a=rand();my$b=rand();next
   if$a==0;push@z,sqrt(-2*log($a))*cos(8*atan2(1,1)*$b)}
$pl->histogram(pdl(@z),20,SUBPAGE=>2);$pl->close;
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');
const
   cats=['Lunes','Martes','Miercoles','Jueves','Viernes'],v=[42,38,55,61
// z es la muestra reproducible construida en el ejercicio
   17.
const
   z=fs.readFileSync('ej17.dat','utf8').trim().split(/\s+/).map(Number);
const
   data=[{x:cats,y:v,type:'bar',xaxis:'x',yaxis:'y',name:'categorias'},{
const
   layout={grid:{rows:1,columns:2,pattern:'independent'},yaxis:{rangemod
   vs histograma'};
```

```
fs.writeFileSync('ej18_node.html',`<!doctype
   html><script>${P}</script><div
   id="p"></div><script>Plotly.newPlot('p',${JSON.stringify(data)},${JSO
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot. Los lenguajes escriben `ej18_categorias.dat` con etiqueta y valor, y `ej18_muestra.dat` con la muestra continua.

```
set terminal pngcairo size 1000,450
set output 'ej18_gnuplot.png';set multiplot layout 1,2
set title 'categorias';set yrange [0:*];set style fill solid
   .7;set boxwidth .7
plot 'ej18_categorias.dat' u 2:xtic(1) w boxes notitle
set title 'histograma: 20 clases';unset
   yrange;binw=.4;bin(x)=binw*floor(x/binw)+binw/2;set
   boxwidth binw
plot 'ej18_muestra.dat' u (bin($1)):(1) smooth frequency w
   boxes notitle
unset multiplot;unset output
```

Scilab + Gnuplot

```
grand('setsd',20260822);z=grand(1000,1,'nor',0,1);fprintfMat('ej18_muest
   38 55 61 47];for i=1:5,mfprintf(fd,'%s
   %d\n',dias(i),v(i));end;mclose(fd);[e,o,er]=host('gnuplot
   ej18.gnu');if e<>0 then error(er),end
```

GNU Octave + Gnuplot

```
rng(20260822);z=randn(1000,1);save('-ascii','ej18_muestra.dat','z');fid=
   38 55 61 47];for i=1:5,fprintf(fid,'%s
   %d\n',d{i},v(i));end;fclose(fid);system('gnuplot
   ej18.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
z=np.random.default_rng(20260822).normal(size=1000);np.savetxt('ej18_mue
   {v}' for d,v in
   zip(['Lunes','Martes','Miercoles','Jueves','Viernes'],[42,38,55,61,47
```

Perl + Gnuplot

```
use strict;use warnings;srand(20260822);open
   my$m,'>','ej18_muestra.dat' or
   die$!;for(1..500){my$a=rand();my$b=rand();$a=1e-12
   if$a==0;my$r=sqrt(-2*log($a));print$m
   $r*cos(8*atan2(1,1)*$b),"\n",$r*sin(8*atan2(1,1)*$b),"\n"}close$m;ope
   my$c,'>','ej18_categorias.dat' or die$!;print$c"Lunes
   42\nMartes 38\nMiercoles 55\nJueves 61\nViernes
   47\n";close$c;system('gnuplot','ej18.gnu')==0 or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');let
   z=fs.readFileSync('ej17.dat','utf8').trim().split(/\s+/);fs.writeFile
   42\nMartes 38\nMiercoles 55\nJueves 61\nViernes
   47\n');let
   q=spawnSync('gnuplot',['ej18.gnu'],{encoding:'utf8'});if(q.status!==0
   new Error(q.stderr);
```

19. **Ejercicio:** Dispersión con una variable codificada por color

Genere de manera reproducible 500 pares (x_i, y_i) distribuidos en la región

$$-2 \le x \le 2, \qquad -2 \le y \le 2,$$

y calcule para cada punto

$$z_i = x_i^2 + y_i^2.$$

a) Represente los pares (x_i, y_i) mediante un gráfico de dispersión y codifique el valor z_i mediante el color de cada marcador.

b) Incluya una barra de color que permita relacionar la escala cromática con los valores de z y utilice una paleta apropiada para una magnitud ordenada.

c) Mantenga constante el tamaño de los marcadores para que el color sea la única codificación adicional de z en esta figura.

d) Identifique los puntos o regiones donde z es menor y mayor y compruebe esas conclusiones utilizando los valores numéricos, no sólo el color.

e) Explique qué estructura espacial se reconoce rápidamente gracias al color y por qué esta codificación es menos adecuada para comparar valores numéricos muy próximos.

Scilab. Puede usarse `scatter(x,y,tam,z)` junto con una barra de color.

Octave. Puede usarse `scatter` indicando el valor que controla el color de cada punto.

Python. Use `scatter` con el argumento `c=z` y agregue una `colorbar`.

Perl. Use una representación de puntos cuyo backend permita asociar una escala de color a una tercera magnitud.

JavaScript. En una traza `scatter`, asigne los valores de z a `marker.color` y muestre la escala correspondiente.

Nota Gnuplot: Guarde x, y y z en tres columnas y represente los puntos con una paleta controlada por la tercera columna.

Respuesta (T: diez modalidades).

Se fija nuevamente la semilla `20260822`. Como $z = x^2 + y^2$, los valores menores se concentran alrededor del origen y los mayores cerca de las esquinas del cuadrado. La comprobación numérica consiste en localizar `min(z)` y `max(z)` y mostrar los puntos asociados.

Scilab, gráficos nativos

```
grand('setsd',20260822);x=grand(500,1,'unf',-2,2);y=grand(500,1,'unf',-2
[zmin,imin]=min(z);[zmax,imax]=max(z);disp([x(imin) y(imin)
   zmin]);disp([x(imax) y(imax) zmax])
scf(190);clf();scatter(x,y,30,z,'fill');gca().isoview='on';colorbar(min(
```

GNU Octave, gráficos nativos

```
rng(20260822);x=-2+4*rand(500,1);y=-2+4*rand(500,1);z=x.^2+y.^2;[zmin,i]
   equal;colorbar;grid
   on;title('z=x^2+y^2');print('ej19_octave.png','-dpng');
```

Python, gráficos nativos

```
import numpy as np,matplotlib.pyplot as plt
r=np.random.default_rng(20260822);x=r.uniform(-2,2,500);y=r.uniform(-2,2
```

Perl, gráficos nativos con PDL/PLplot

```
use strict;use warnings;use PDL;use
   PDL::Graphics::PLplot;srand(20260822);my(@x,@y,@z);for(1..500){my$x=-
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');funct
   R(a){return()=>{let
   t=a+=0x6D2B79F5;t=Math.imul(t^t>>>15,t|1);t^=t+Math.imul(t^t>>>7,t|61
   r=R(20260822),x=[],y=[],z=[];for(let i=0;i<500;i++){let
   a=-2+4*r(),b=-2+4*r();x.push(a);y.push(b);z.push(a*a+b*b)}let
   imin=z.indexOf(Math.min(...z)),imax=z.indexOf(Math.max(...z));console
   data=[{x,y,mode:'markers',type:'scatter',marker:{size:7,color:z,color
   html><script>${P}</script><div
   id="p"></div><script>Plotly.newPlot('p',${JSON.stringify(data)},{xaxi
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot

```
set terminal pngcairo size 650,600
set output 'ej19_gnuplot.png';set xrange [-2:2];set yrange
   [-2:2];set size ratio -1
set palette rgbformulae 33,13,10;set colorbox;set cblabel
   'z=x^2+y^2';set grid
plot 'ej19.dat' u 1:2:3 w points pt 7 ps .75 palette notitle
unset output
```

Scilab + Gnuplot

```
grand('setsd',20260822);x=grand(500,1,'unf',-2,2);y=grand(500,1,'unf',-2
   y z],'%15.12g');[e,o,er]=host('gnuplot ej19.gnu');if e<>0
   then error(er),end
```

GNU Octave + Gnuplot

```
rng(20260822);x=-2+4*rand(500,1);y=-2+4*rand(500,1);z=x.^2+y.^2;dlmwrite
   y z],'delimiter',' ');system('gnuplot ej19.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
r=np.random.default_rng(20260822);x=r.uniform(-2,2,500);y=r.uniform(-2,2
```

Perl + Gnuplot

```
use strict;use warnings;srand(20260822);open
   my$f,'>','ej19.dat' or
   die$!;for(1..500){my$x=-2+4*rand();my$y=-2+4*rand();print$f"$x
   $y
   ",($x*$x+$y*$y),"\n"}close$f;system('gnuplot','ej19.gnu')==0
   or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');funct
   R(a){return()=>{let
   t=a+=0x6D2B79F5;t=Math.imul(t^t>>>15,t|1);t^=t+Math.imul(t^t>>>7,t|61
   r=R(20260822),s='';for(let i=0;i<500;i++){let
   x=-2+4*r(),y=-2+4*r();s+=`${x} ${y}
   ${x*x+y*y}\n`}fs.writeFileSync('ej19.dat',s);let
   q=spawnSync('gnuplot',['ej19.gnu'],{encoding:'utf8'});if(q.status!==0
   new Error(q.stderr);
```

El tamaño del marcador queda fijo en las diez modalidades: sólo el color codifica z.

La estructura radial se reconoce de inmediato, pero una paleta continua no es una regla de medición precisa: dos colores próximos son difíciles de comparar y la percepción del color no es lineal.

7.4.5. Sección 4: Composición de figuras y representaciones múltiples

20. **Ejercicio:** Composición de gráficos de un proyectil

Considere un proyectil sin resistencia del aire, con

$$v_0 = 50 \text{ m/s}, \qquad \theta = 45°, \qquad g = 9{,}81 \text{ m/s}^2, \qquad m = 1 \text{ kg}.$$

Mientras el proyectil se encuentre por encima del nivel de lanzamiento,

$$x(t) = v_0 \cos\theta\, t,$$

$$y(t) = v_0 \sin\theta\, t - \frac{1}{2} g t^2,$$

$$v_x(t) = v_0 \cos\theta, \qquad v_y(t) = v_0 \sin\theta - g t.$$

a) Determine el instante de regreso al nivel $y = 0$ y utilícelo como límite superior del intervalo temporal.

b) Construya una figura compuesta de 2×2 paneles que muestre:

   I) la trayectoria y frente a x;

   II) $v_x(t)$ y $v_y(t)$;

   III) las energías cinética K y potencial U en función del tiempo;

   IV) el retrato de fase v_y frente a y.

c) Utilice la misma discretización temporal en los cuatro paneles.

d) Compruebe numéricamente que $K+U$ permanece aproximadamente constante y explique las pequeñas diferencias que puedan aparecer por redondeo.

Nota Gnuplot: Puede utilizarse `set multiplot layout 2,2`. Para la trayectoria resulta apropiado el modo paramétrico mediante `set parametric` y `set trange`.

Respuesta (T: diez modalidades).

El tiempo de vuelo se obtiene de $y(t_f) = 0$ con $t_f > 0$:

$$t_f = \frac{2v_0 \sin\theta}{g} \simeq 7{,}2080 \text{ s}.$$

El alcance es aproximadamente 254,84 m, la altura máxima 63,71 m y la energía mecánica ideal $E = \tfrac{1}{2}mv_0^2 = 1250$ J.

Scilab, gráficos nativos

```
v0=50;th=%pi/4;g=9.81;m=1;tf=2*v0*sin(th)/g;t=linspace(0,tf,300);vx=v0*c
   max|E-E0|=%g\n',tf,max(abs(E-E(1))))
scf(200);clf();subplot(2,2,1);plot(x,y);xgrid();xtitle('trayectoria','x'
```

GNU Octave, gráficos nativos

```
v0=50;th=pi/4;g=9.81;m=1;tf=2*v0*sin(th)/g;t=linspace(0,tf,300);vx=v0*co
   dE=%g\n',tf,max(abs(E-E(1))));subplot(2,2,1);plot(x,y);grid
   on;subplot(2,2,2);plot(t,vx,t,vy);grid
   on;legend('vx','vy');subplot(2,2,3);plot(t,K,t,U,t,E);grid
   on;legend('K','U','E');subplot(2,2,4);plot(y,vy);grid
   on;print('ej20_octave.png','-dpng');
```

Python, gráficos nativos

```
import numpy as np,matplotlib.pyplot as plt
v0=50.;th=np.pi/4;g=9.81;m=1.;tf=2*v0*np.sin(th)/g;t=np.linspace(0,tf,30
   for a in
   ax.flat];ax[0,1].legend();ax[1,0].legend();fig.tight_layout();fig.sav
```

Perl, gráficos nativos mediante SVG

```
use strict;use
   warnings;my$pi=4*atan2(1,1);my($v0,$th,$g,$m)=(50,$pi/4,9.81,1);my$tf
   abs($_-$E[0])>$d}print"tf=$tf dE=$d\n";
```

```
sub
   panel{my($xx,$series,$ox,$oy,$w,$h)=@_;my($xmin,$xmax)=($xx->[0],$xx-
   if$_<$xmin;$xmax=$_ if$_>$xmax}for
   my$a(@$series){for(@$a){$ymin=$_ if$_<$ymin;$ymax=$_
   if$_>$ymax}}my@col=qw(black red blue green);my$out='';for
   my$k(0..$#$series){my@p;for
   my$i(0..$#$xx){push@p,($ox+$w*($xx->[$i]-$xmin)/(($xmax-$xmin)||1)).'
   fill="none" stroke="'.$col[$k].'" points="'.join('
   ',@p).'"/>'}return$out}
open my$s,'>','ej20_perl.svg' or die$!;print$s'<svg
   xmlns="http://www.w3.org/2000/svg" width="900"
   height="700">',panel(\@x,[\@y],40,30,380,280),panel(\@t,[\@vx,\@vy],4
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');const
   v0=50,th=Math.PI/4,g=9.81,m=1,tf=2*v0*Math.sin(th)/g,t=Array.from({le
   data=[{x,y,xaxis:'x',yaxis:'y',mode:'lines',name:'trayectoria'},{x:t,
   html><script>${P}</script><div
   id="p"></div><script>Plotly.newPlot('p',${JSON.stringify(data)},{grid
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot. Los programas escriben columnas $t, x, y, v_x, v_y, K, U, E$ en `ej20.dat`.

```
set terminal pngcairo size 1000,750;set output
   'ej20_gnuplot.png';set multiplot layout 2,2;set grid
set title 'trayectoria';plot 'ej20.dat' u 2:3 w l notitle
set title 'velocidades';plot '' u 1:4 w l t 'vx','' u 1:5 w
   l t 'vy'
set title 'energias';plot '' u 1:6 w l t 'K','' u 1:7 w l t
   'U','' u 1:8 w l t 'K+U'
set title 'retrato de fase';plot '' u 3:5 w l t 'vy(y)'
unset multiplot;unset output
```

Scilab + Gnuplot

```
v0=50;th=%pi/4;g=9.81;m=1;tf=2*v0*sin(th)/g;t=linspace(0,tf,300)';vx=v0*
   x y vx vy K U
   E],'%15.12g');printf('dE=%g\n',max(abs(E-E(1))));[q,o,er]=host('gnupl
   ej20.gnu');if q<>0 then error(er),end
```

GNU Octave + Gnuplot

```
v0=50;th=pi/4;g=9.81;m=1;tf=2*v0*sin(th)/g;t=linspace(0,tf,300)';vx=v0*c
   x y vx vy K U E],'delimiter','
   ');printf('dE=%g\n',max(abs(E-E(1))));system('gnuplot
   ej20.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
v0=50.;th=np.pi/4;g=9.81;m=1.;tf=2*v0*np.sin(th)/g;t=np.linspace(0,tf,30
```

Perl + Gnuplot

```
use strict;use
   warnings;my$pi=4*atan2(1,1);my($v0,$th,$g,$m)=(50,$pi/4,9.81,1);my$tf
   my$f,'>','ej20.dat' or
   die$!;for(0..299){my$t=$tf*$_/299;my$vx=$v0*cos($th),$vy=$v0*sin($th)
   $x $y $vx $vy $K $U
   ",($K+$U),"\n"}close$f;system('gnuplot','ej20.gnu')==0 or
   die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');const
   v0=50,th=Math.PI/4,g=9.81,m=1,tf=2*v0*Math.sin(th)/g;let
   s='';for(let i=0;i<300;i++){let
   t=tf*i/299,vx=v0*Math.cos(th),vy=v0*Math.sin(th)-g*t,x=vx*t,y=v0*Math
   ${x} ${y} ${vx} ${vy} ${K} ${U}
   ${K+U}\n`}fs.writeFileSync('ej20.dat',s);let
   q=spawnSync('gnuplot',['ej20.gnu'],{encoding:'utf8'});if(q.status!==0
   new Error(q.stderr);
```

En las cinco variantes la comprobación de $K+U$ debe dar diferencias del orden del redondeo, porque las expresiones analíticas conservan exactamente la energía en el modelo ideal.

21. **Ejercicio:** Objetos gráficos, propiedades y paneles

Use las capacidades gráficas nativas del lenguaje asignado y trabaje con los objetos gráficos creados:

a) genere 300 puntos de x en $[0, 2\pi]$ y represente simultáneamente $\sin(x)$ y $\cos(x)$;

b) conserve referencias a la figura, los ejes y, cuando el entorno lo permita, las curvas;

c) después de dibujarlas cambie al menos dos propiedades gráficas sin volver a calcular los datos;

d) agregue una anotación en el máximo de $\sin(x)$ y una leyenda;

e) construya una segunda figura con dos paneles: arriba las dos funciones y abajo $\sin(x) - \cos(x)$, manteniendo el mismo dominio horizontal;

f) exporte el resultado a un formato raster y a uno vectorial.

Scilab. Puede trabajar con `gcf`, `gca`, handles de entidades, propiedades, `xstring`, `subplot` y las funciones `xs2*` de exportación.

Octave. Puede usar `figure`, `axes`, `plot`, `get`/`set`, `text`, `subplot` y `print`.

Python. Use Matplotlib mediante objetos `Figure` y `Axes`; conserve las líneas devueltas por `plot`, modifique sus propiedades y exporte con `savefig`.

Perl. Con PDL puede usarse PLplot para la figura científica; si necesita conservar y modificar objetos interactivos, puede usarse PDL::Graphics::Prima.

JavaScript. Con Plotly.js puede crear la figura y modificar trazas o layout con `Plotly.restyle`/`Plotly.relayout`, agregar `annotations` y usar paneles mediante `layout.grid` o ejes independientes.

Respuesta (N: cinco modalidades nativas).

Aquí no se agrega una variante Gnuplot por lotes: el objetivo central es conservar objetos o referencias y modificar sus propiedades después de creados, sin recalcular los datos.

Scilab

```
x=linspace(0,2*%pi,300);ys=sin(x);yc=cos(x)
f=scf(210);clf();plot(x,ys,x,yc);a=gca();c=gce();hs=c.children(2);hc=c.c
hs.thickness=3;hc.line_style=2;hc.thickness=2;xstring(%pi/2,1,'max
   sin=(pi/2,1)');legend('sin','cos');xgrid()
f2=scf(211);clf();subplot(2,1,1);plot(x,ys,x,yc);legend('sin','cos');xgr
xs2png(f2,'ej21_scilab.png');xs2pdf(f2,'ej21_scilab.pdf')
```

Las referencias `f`, `a`, `hs` y `hc` permanecen válidas mientras exista la escena; el cambio de grosor y estilo se hace sobre los objetos ya dibujados.

GNU Octave

```
x=linspace(0,2*pi,300);ys=sin(x);yc=cos(x);f=figure(210);[hs,hc]=plot(x,
   max sin');legend('sin','cos');grid on;
f2=figure(211);subplot(2,1,1);plot(x,ys,x,yc);legend('sin','cos');grid
   on;subplot(2,1,2);plot(x,ys-yc);grid
   on;print(f2,'ej21_octave.png','-dpng');print(f2,'ej21_octave.pdf','-d
```

Python/Matplotlib

```
import numpy as np,matplotlib.pyplot as plt
x=np.linspace(0,2*np.pi,300);ys=np.sin(x);yc=np.cos(x);fig,ax=plt.subplo
   sin',(np.pi/2,1),xytext=(2,1.15),arrowprops={'arrowstyle':'->'});ax.l
fig2,(a1,a2)=plt.subplots(2,1,sharex=True);a1.plot(x,ys,label='sin');a1.
```

Perl con PDL::Graphics::Prima

```
use strict;use warnings;use PDL;use Prima
   qw(Application);use PDL::Graphics::Prima;
my$x=sequence(300)*(2*atan2(0,-1)/299);my$ys=sin($x);my$yc=cos($x);
my$w=Prima::MainWindow->create(text=>'Objetos
   graficos',size=>[800,650]);
my$p1=$w->insert('Plot',-sin=>ds::Pair($x,$ys,color=>cl::Blue),-cos=>ds:
my$sin_obj=$p1->dataSets->{sin};my$cos_obj=$p1->dataSets->{cos};
$p1->title('sin y cos; max
   sin=(pi/2,1)');$p1->x->label('x');$p1->y->label('y');    #
   propiedades cambiadas despues
```

```
my$p2=$w->insert('Plot',-dif=>ds::Pair($x,$ys-$yc,color=>cl::Black),plac
$p1->save_to_file('ej21_perl.png');$p1->save_to_postscript('ej21_perl.ps
run Prima;
```

La colección `dataSets` proporciona referencias a los objetos de datos; el título y los objetos de eje se modifican después de crear el gráfico. La segunda ventana lógica contiene dos widgets de trazado; PostScript es la salida vectorial.

JavaScript/Plotly.js

```
const
   x=Array.from({length:300},(_,i)=>2*Math.PI*i/299),ys=x.map(Math.sin),
   d1=document.getElementById('fig1');Plotly.newPlot(d1,[{x,y:ys,name:'s
   sin',showarrow:true}]});Plotly.restyle(d1,{'line.width':3},[0]);Plotl
   d2=document.getElementById('fig2');Plotly.newPlot(d2,[{x,y:ys,name:'s
   listo',u.length));Plotly.toImage(d2,{format:'svg'}).then(u=>console.l
   listo',u.length));
```

En esta variante `d1` y `d2` son referencias a las figuras y `Plotly.restyle` cambia propiedades de trazas ya existentes. La misma escena puede serializarse como PNG o SVG sin recalcular $\sin x$ ni $\cos x$.

7.4.6. Sección 5: Representación tridimensional

22. **Ejercicio:** Superficies tridimensionales elementales

    Represente las funciones

$$f(x, y) = x^2 + y^2, \qquad -10 \le x, y \le 10,$$

    y

$$g(x, y) = x \sin y + y \sin x, \qquad -\pi \le x, y \le \pi.$$

    a) Genere para cada función una grilla rectangular de al menos 50×50 puntos y evalúe la función en toda la grilla.

    b) Conserve, cuando use una herramienta gráfica externa, los datos de cada grilla en un archivo estructurado de modo que pueda reconstruirse su organización bidimensional sin recalcular la función.

    c) Represente cada superficie como malla de alambre.

    d) Represente cada superficie como superficie coloreada según el valor de z, incluyendo una escala o barra de color cuando la herramienta lo permita.

    e) Genere una representación que combine superficie y malla.

    f) Para al menos una de las funciones, superponga curvas de nivel directamente sobre la superficie tridimensional.

    g) Para esa misma función, proyecte un conjunto de curvas de nivel sobre un plano horizontal situado en la base de la representación, es decir sobre un valor constante de z elegido por debajo o en el límite inferior de la superficie visible.

h) Observe cada función desde al menos dos puntos de vista diferentes y explique qué características se perciben mejor en cada uno.

i) Compare la información que brindan la superficie sola, las curvas de nivel sobre la superficie y las curvas proyectadas sobre un plano.

Nota Gnuplot: Para `splot`, una grilla tabular puede escribirse como bloques con un mismo valor de x, recorriendo todos los valores de y, y separando bloques consecutivos mediante una línea en blanco. Esa separación conserva en el archivo de texto la estructura bidimensional de la malla.

Scilab. El siguiente programa reúne las variantes que estaban distribuidas en varios fragmentos: define las dos funciones, construye grillas de al menos 50×50 puntos, conserva las matrices para una representación nativa y escribe ambos archivos de datos en el formato por bloques requerido por Gnuplot. Se usa una sola rutina para escribir las dos grillas.

```
// ===== Programa unico de calculo y generacion de datos =====

function z = f(x, y)
    z = x.^2 + y.^2
endfunction

function z = g(x, y)
    z = x.*sin(y) + y.*sin(x)
endfunction

// Escribe una matriz Z como bloques de x fijo.
// La linea en blanco entre bloques es significativa para splot.
function escribir_grilla(nombre, x, y, Z)
    fd = mopen(nombre, 'w')
    for i = 1:length(x)
        M = [x(i)*ones(length(y),1), y(:), Z(i,:)']
        mfprintf(fd, '%.6f %.6f %.6f\n', M')
        mfprintf(fd, '\n')
    end
    mclose(fd)
endfunction

// --- f(x,y) = x^2 + y^2 ---
// Variante original conservada:
// x = -10:0.5:10; y = -10:0.5:10;
// Esa variante produce 41 puntos por eje; aqui se usan 50 para
// cumplir el minimo 50x50 pedido en el enunciado.
xf = linspace(-10, 10, 50)
yf = linspace(-10, 10, 50)
Zf = zeros(length(xf), length(yf))
for i = 1:length(xf)
    Zf(i,:) = f(xf(i), yf)
```

```
end
escribir_grilla('/tmp/f3d_a.txt', xf, yf, Zf)

// --- g(x,y) = x*sin(y) + y*sin(x) ---
xg = linspace(-%pi, %pi, 50)
yg = linspace(-%pi, %pi, 50)
Zg = zeros(length(xg), length(yg))
for i = 1:length(xg)
    Zg(i,:) = g(xg(i), yg)
end
escribir_grilla('/tmp/f3d_b.txt', xg, yg, Zg)

// Variantes nativas de Scilab presentes en los fragmentos
   originales:
// scf(1); clf(); plot3d (xf, yf, Zf)
// scf(2); clf(); plot3d1(xf, yf, Zf)
// scf(3); clf(); surf   (xf, yf, Zf)
// scf(4); clf(); plot3d (xg, yg, Zg)
// scf(5); clf(); plot3d1(xg, yg, Zg)
// Tambien aparecia la forma equivalente:
// deff('[z]=f(x,y)','z=x.^2+y.^2')

// Variante de escritura original, equivalente pero con dos
   bucles:
// fd = mopen(nombre,'w')
// for i = 1:length(x)
//     for j = 1:length(y)
//         mfprintf(fd,'%.4f %.4f %.4f\n',x(i),y(j),Z(i,j))
//     end
//     mfprintf(fd,'\n')
// end
// mclose(fd)
// Los originales usaban %.4f y %.6f; aqui se unifica en
   %.6f.

printf('Datos generados:\n')
printf('  /tmp/f3d_a.txt  (x^2 + y^2)\n')
printf('  /tmp/f3d_b.txt  (x*sin(y) + y*sin(x))\n')

// Una vez guardado el script Gnuplot de abajo como
   /tmp/superficies.gnu:
// [estado, salida, error_sistema] = host('gnuplot
   /tmp/superficies.gnu')
```

Nota Gnuplot: Guarde el siguiente bloque como `/tmp/superficies.gnu`. Es un único script que reemplaza los numerosos scripts parciales anteriores y genera, para ambas funciones, superficie coloreada, malla, superficie con malla, contornos en planta, contornos sobre la superficie, contornos proyectados a la base y tres perspectivas. Los nombres de salida usados en las variantes anteriores se conservan en comentarios al final del propio script.

```
# ===== Script unico para las dos superficies =====
F = '/tmp/f3d_a.txt'
G = '/tmp/f3d_b.txt'

#
   ------------------------------------------------------------------
# f(x,y) = x^2 + y^2
#
   ------------------------------------------------------------------
set xrange [-10:10]
set yrange [-10:10]
set hidden3d
set palette rgbformulae 33,13,10

# 1. Superficie con color
set terminal png size 800,600
set output '/tmp/surf_1_color.png'
unset contour
set surface
set pm3d depthorder
set view 55,35
set colorbox
set title 'f(x,y) = x^2 + y^2'
set xlabel 'x'; set ylabel 'y'; set zlabel 'z'
splot F with pm3d notitle

# 2. Malla de alambre
set output '/tmp/surf_1_malla.png'
unset pm3d
unset colorbox
unset contour
set surface
set view 55,35
set title 'f(x,y) = x^2 + y^2  [malla]'
splot F with lines lc 'blue' notitle

# 3. Superficie + malla
set output '/tmp/surf_1_pm3d_malla.png'
set pm3d depthorder
set colorbox
set view 55,35
set title 'f(x,y) = x^2 + y^2  [pm3d + malla]'
splot F with pm3d notitle, \
      F with lines lw 0.4 lc 'black' notitle

# 4. Curvas de nivel en planta con relleno
set terminal png size 700,600
set output '/tmp/surf_1_contorno.png'
set size ratio -1
set view map
set pm3d map
```

```
set contour base
set cntrparam levels 15
set colorbox
set title 'Curvas de nivel: x^2 + y^2'
splot F with pm3d notitle

# 5. Contornos sobre la superficie tridimensional
set size noratio
set terminal png size 800,600
set output '/tmp/surf_1_contornos_superficie.png'
set view 55,35
set pm3d depthorder
set contour surface
set cntrparam levels 15
set title 'Contornos sobre la superficie: x^2 + y^2'
splot F with pm3d notitle, \
      F with lines lw 0.25 lc 'black' notitle

# 6. Contornos proyectados sobre el plano de base
set output '/tmp/surf_1_contornos_base.png'
set view 55,35
set contour base
set cntrparam levels 15
set zrange [-5:200]
set xyplane at -5
set title 'Contornos proyectados en la base: x^2 + y^2'
splot F with pm3d notitle
set autoscale z
set xyplane at 0

# 7. Tres puntos de vista de los mismos datos
set terminal png size 1000,400
set output '/tmp/surf_1_vistas.png'
unset contour
set surface
set pm3d depthorder
set colorbox
set multiplot layout 1,3
set title 'Vista frontal';     set view 90,0
splot F with pm3d notitle
set title 'Vista lateral';     set view 90,90
splot F with pm3d notitle
set title 'Vista perspectiva'; set view 55,35
splot F with pm3d notitle
unset multiplot

#
   ------------------------------------------------------------------
# g(x,y) = x*sin(y) + y*sin(x)
#
   ------------------------------------------------------------------
```

```
set xrange [-pi:pi]
set yrange [-pi:pi]
set hidden3d
set palette defined (-6 'blue', 0 'white', 6 'red')

# 1. Superficie con color
set terminal png size 800,600
set output '/tmp/surf2_1_color.png'
unset contour
set surface
set pm3d depthorder
set view 50,40
set colorbox
set title 'g(x,y) = x*sin(y) + y*sin(x)'
set xlabel 'x'; set ylabel 'y'; set zlabel 'z'
splot G with pm3d notitle

# 2. Malla de alambre
set output '/tmp/surf2_1_malla.png'
unset pm3d
unset colorbox
unset contour
set surface
set view 50,40
set title 'g(x,y) = x*sin(y) + y*sin(x)  [malla]'
splot G with lines lc 'blue' notitle

# 3. Superficie + malla
set output '/tmp/surf2_1_pm3d_malla.png'
set pm3d depthorder
set colorbox
set view 50,40
set title 'g(x,y) = x*sin(y) + y*sin(x)  [pm3d + malla]'
splot G with pm3d notitle, \
      G with lines lw 0.3 lc 'black' notitle

# 4. Curvas de nivel en planta con relleno
set terminal png size 700,600
set output '/tmp/surf2_1_contorno.png'
set size ratio -1
set view map
set pm3d map
set contour base
set cntrparam levels 16
set colorbox
set title 'Curvas de nivel: x*sin(y) + y*sin(x)'
splot G with pm3d notitle

# 5. Contornos sobre la superficie tridimensional
set size noratio
set terminal png size 800,600
```

```
set output '/tmp/surf2_1_contornos_superficie.png'
set view 50,40
set pm3d depthorder
set contour surface
set cntrparam levels 16
set title 'Contornos sobre la superficie: x*sin(y) +
   y*sin(x)'
splot G with pm3d notitle, \
      G with lines lw 0.25 lc 'black' notitle

# 6. Contornos proyectados sobre la base
set output '/tmp/surf2_1_contornos_base.png'
set view 50,40
set contour base
set cntrparam levels 16
set zrange [-7:7]
set xyplane at -7
set title 'Contornos proyectados en la base: x*sin(y) +
   y*sin(x)'
splot G with pm3d notitle
set autoscale z
set xyplane at 0

# 7. Tres puntos de vista
set terminal png size 1000,400
set output '/tmp/surf2_1_vistas.png'
unset contour
set surface
set pm3d depthorder
set colorbox
set multiplot layout 1,3
set title 'Vista frontal';     set view 90,0
splot G with pm3d notitle
set title 'Vista lateral';     set view 90,90
splot G with pm3d notitle
set title 'Vista perspectiva'; set view 55,35
splot G with pm3d notitle
unset multiplot

# Variantes originales conservadas:
# - Para g tambien se uso set view 55,35; aqui se conserva
   50,40
#   en las vistas oblicuas y 55,35 queda documentado como
   alternativa.
# - Los scripts que evaluaban directamente las formulas
   usaban, segun
#   la figura, set isosamples 30,30; 40,40; 50,50 u 80,80.
#   Ejemplos equivalentes sin archivo de datos:
#     set isosamples 50,50
#     splot x**2 + y**2 with pm3d notitle
#     splot x*sin(y) + y*sin(x) with pm3d notitle
```

```
# - Nombres anteriores para f:
#     plot3d_a1.png -> surf_1_color.png
#     plot3d_a2.png -> surf_1_malla.png
#     plot3d_a3.png -> surf_1_contorno.png
# - Nombres anteriores para g:
#     plot3d_b1.png -> surf2_1_color.png
#     plot3d_b2.png -> surf2_1_malla.png
#     plot3d_b3.png -> surf2_1_pm3d_malla.png
#     plot3d_b4.png -> surf2_1_contorno.png
```

Scilab. El script anterior puede ejecutarse desde Scilab con `host('gnuplot /tmp/superficies.gnu')`. Las funciones nativas `plot3d`, `plot3d1` y `surf` siguen siendo una alternativa para representar directamente las matrices `Zf` y `Zg` sin crear un proceso Gnuplot.

Respuesta (T: diez modalidades).

Se usan grillas de 60×60. Para una herramienta externa los archivos se escriben por bloques de x constante separados por una línea en blanco, tal como se explicó arriba. Para $f = x^2 + y^2$ se aprovecha además que las curvas de nivel $f = c$ son círculos $x^2 + y^2 = c$; esto permite comprobar visual y numéricamente los contornos.

Scilab, gráficos nativos

```
xf=linspace(-10,10,60);yf=linspace(-10,10,60);Zf=zeros(60,60);for
   i=1:60,Zf(i,:)=xf(i)^2+yf.^2;end
xg=linspace(-%pi,%pi,60);yg=xg;Zg=zeros(60,60);for
   i=1:60,Zg(i,:)=xg(i)*sin(yg)+yg*sin(xg(i));end
scf(220);clf();subplot(2,2,1);plot3d(xf,yf,Zf);title('malla
   f');subplot(2,2,2);surf(xf,yf,Zf);colorbar();title('superficie
   f')
subplot(2,2,3);plot3d1(xf,yf,Zf);contour(xf,yf,Zf,15);title('contornos
   sobre
   f');subplot(2,2,4);plot3d1(xf,yf,Zf);contour(xf,yf,Zf,15,zval=-5);tit
   en base')
scf(221);clf();subplot(1,2,1);plot3d1(xg,yg,Zg,55,35);subplot(1,2,2);plo
```

La malla muestra mejor la discretización; el color permite leer magnitudes; los contornos hacen explícitos los conjuntos $z = c$ y su proyección elimina la ocultación propia de la perspectiva.

GNU Octave, gráficos nativos

```
[x,y]=meshgrid(linspace(-10,10,60));z=x.^2+y.^2;figure(220);subplot(2,2,
   on;contour3(x,y,z,15,'k');hold
   off;subplot(2,2,4);surfc(x,y,z,15);print('ej22_octave_f.png','-dpng')
[x,y]=meshgrid(linspace(-pi,pi,60));z=x.*sin(y)+y.*sin(x);figure(221);su
```

`contour3` coloca contornos a sus alturas $z = c$; `surfc` combina superficie y contornos proyectados en la base.

Python/Matplotlib

```
import numpy as np,matplotlib.pyplot as plt
```

```
x=np.linspace(-10,10,60);y=x;X,Y=np.meshgrid(x,y);Z=X*X+Y*Y
fig=plt.figure(figsize=(12,8));a=fig.add_subplot(221,projection='3d');a.
x=np.linspace(-np.pi,np.pi,60);X,Y=np.meshgrid(x,x);Z=X*np.sin(Y)+Y*np.s
```

Perl, gráficos nativos con PDL/PLplot

```
use strict;use warnings;use PDL;use PDL::Graphics::PLplot
   qw(pladv plvpor plwind plw3d plbox3 plmesh plsurf3d);
sub
   makegrid{my($a,$b,$F)=@_;my$x=zeroes(60)->xlinvals($a,$b);my$y=$x->co
sub
   draw{my($file,$x,$y,$Z,$a,$b,$z0,$z1,$alt,$az,$opt)=@_;my$lev=zeroes(
my($xf,$yf,$Zf)=makegrid(-10,10,sub{$_[0]**2+$_[1]**2});
draw('ej22_perl_f_mesh.png',$xf,$yf,$Zf,-10,10,0,200,55,35,3);
                # malla
draw('ej22_perl_f_color.png',$xf,$yf,$Zf,-10,10,0,200,55,35,4);
              # MAG_COLOR
draw('ej22_perl_f_combo.png',$xf,$yf,$Zf,-10,10,0,200,55,35,4+128);
        # + FACETED
draw('ej22_perl_f_surfacecont.png',$xf,$yf,$Zf,-10,10,0,200,55,35,4+32);
   + SURF_CONT
draw('ej22_perl_f_basecont.png',$xf,$yf,$Zf,-10,10,-5,200,55,35,4+8);
      # + BASE_CONT
my($xg,$yg,$Zg)=makegrid(-4*atan2(1,1),4*atan2(1,1),sub{$_[0]*sin($_[1])
draw('ej22_perl_g_view1.png',$xg,$yg,$Zg,-4*atan2(1,1),4*atan2(1,1),-7,7
draw('ej22_perl_g_view2.png',$xg,$yg,$Zg,-4*atan2(1,1),4*atan2(1,1),-7,7
```

Los bits usados por PLplot son `MAG_COLOR=4`, `BASE_CONT=8`, `SURF_CONT=32` y `FACETED=128`; al sumarlos se combinan las representaciones sin recalcular Z.

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');funct
   seq(a,b,n){return
   Array.from({length:n},(_,i)=>a+(b-a)*i/(n-1))}function
   grid(x,y,F){return y.map(Y=>x.map(X=>F(X,Y)))}function
   surface(x,y,z,name){return{type:'surface',x,y,z,name,colorscale:'Viri
   wire(x,y,z){let q=[];for(let
   j=0;j<y.length;j+=5)q.push({type:'scatter3d',mode:'lines',x,y:x.map((
   i=0;i<x.length;i+=5)q.push({type:'scatter3d',mode:'lines',x:y.map(()=
   q}let
   x=seq(-10,10,60),z=grid(x,x,(X,Y)=>X*X+Y*Y),sf=surface(x,x,z,'f'),wf=
   html><script>${P}</script><div id="fm"></div><div
   id="fc"></div><div id="g1"></div><div
   id="g2"></div><script>Plotly.newPlot('fm',${JSON.stringify(wf)},{titl
   malla'});Plotly.newPlot('fc',[${JSON.stringify(sf)},...${JSON.stringi
   superficie+malla+contornos'});let
   sg=${JSON.stringify(sg)},wg=${JSON.stringify(wg)};Plotly.newPlot('g1'
   vista
   1'});Plotly.newPlot('g2',[sg,...wg],{scene:{camera:{eye:{x:2.2,y:.3,z
   vista 2'});</script>`);
```

En Plotly la rotación de cámara reutiliza exactamente la misma traza y permite comparar puntos de vista sin recalcular los datos.

Gnuplot común a las cinco variantes lenguaje + Gnuplot. Se utiliza directamente el script `/tmp/superficies.gnu` ya incluido en las notas del ejercicio. Las cinco variantes sólo deben producir `f3d_a.txt` y `f3d_b.txt` en bloques de x constante y ejecutar el script.

Scilab + Gnuplot. Es la rutina `escribir_grilla` ya desarrollada arriba, seguida por `host('gnuplot /tmp/superficies.gnu')`. Es una solución completa y no se duplica aquí su listado extenso.

GNU Octave + Gnuplot

```
function gridfile(name,x,y,F),fid=fopen(name,'w');for
   i=1:numel(x),for j=1:numel(y),fprintf(fid,'%.10g %.10g
   %.10g\n',x(i),y(j),F(x(i),y(j)));end;fprintf(fid,'\n');end;fclose(fid
xf=linspace(-10,10,60);gridfile('/tmp/f3d_a.txt',xf,xf,@(x,y)x.^2+y.^2);
   /tmp/superficies.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
def gridfile(name,x,y,F):
 with open(name,'w') as f:
  for X in x:
   for Y in y:f.write(f'{X:.12g} {Y:.12g} {F(X,Y):.12g}\n')
   f.write('\n')
xf=np.linspace(-10,10,60);gridfile('/tmp/f3d_a.txt',xf,xf,lambda
   x,y:x*x+y*y);xg=np.linspace(-np.pi,np.pi,60);gridfile('/tmp/f3d_b.txt
   x,y:x*np.sin(y)+y*np.sin(x));subprocess.run(['gnuplot','/tmp/superfic
```

Perl + Gnuplot

```
use strict;use warnings;my$pi=4*atan2(1,1);sub
   grid{my($n,$a,$b,$F)=@_;open my$h,'>',$n or die$!;for
   my$i(0..59){my$x=$a+($b-$a)*$i/59;for
   my$j(0..59){my$y=$a+($b-$a)*$j/59;print$h"$x $y ",
   $F->($x,$y),"\n"}print$h"\n"}close$h}grid('/tmp/f3d_a.txt',-10,10,sub
   or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');funct
   grid(n,a,b,F){let s='';for(let i=0;i<60;i++){let
   x=a+(b-a)*i/59;for(let j=0;j<60;j++){let
   y=a+(b-a)*j/59;s+=`${x} ${y}
   ${F(x,y)}\n`}s+='\n'}fs.writeFileSync(n,s)}grid('/tmp/f3d_a.txt',-10,
   q=spawnSync('gnuplot',['/tmp/superficies.gnu'],{encoding:'utf8'});if(
   new Error(q.stderr);
```

23. **Ejercicio:** Curva paramétrica tridimensional

Considere la curva

$$x(t) = \cos t, \qquad y(t) = \sin t, \qquad z(t) = \frac{t}{2\pi}, \qquad 0 \le t \le 6\pi.$$

a) Discretice el intervalo con al menos 500 valores de t y represente la curva como una polilínea tridimensional.

b) Marque de manera diferenciada los puntos inicial y final y rotule los tres ejes.

c) Mantenga escalas compatibles en los ejes x e y para no deformar la sección circular de la hélice.

d) Genere al menos tres perspectivas de la misma curva, conservando los mismos datos y límites de los ejes.

e) Obtenga además las proyecciones sobre los planos xy, xz e yz y explique qué información sobre la curva espacial se conserva y cuál se pierde en cada una.

f) Explique por qué una única perspectiva puede ocultar parte de la geometría de una curva tridimensional.

Scilab. Para la curva espacial puede usarse `param3d`.

Octave. Puede usarse `plot3`.

Python. Cree ejes tridimensionales y use `Axes.plot(x,y,z)`.

Perl. Use la operación de polilínea tridimensional disponible en PLplot o en el backend gráfico elegido.

JavaScript. En Plotly.js use una traza `scatter3d` con `mode='lines'`.

Nota Gnuplot: Puede representar columnas x, y, z con `splot ... using 1:2:3 with lines`; alternativamente puede emplear el modo paramétrico.

Respuesta (T: diez modalidades).

La curva da tres vueltas alrededor del eje z y asciende desde $z = 0$ hasta $z = 3$. Sus proyecciones son: sobre xy, un círculo recorrido tres veces; sobre xz, una sinusoide en función de z; sobre yz, una cosenoide en función de z. Una perspectiva única puede superponer segmentos distintos y ocultar el número de vueltas o la dirección de ascenso.

Scilab, gráficos nativos

```
t=linspace(0,6*%pi,600);x=cos(t);y=sin(t);z=t/(2*%pi)
scf(230);clf();param3d(x,y,z,45,35);xtitle('helice');a=gca();a.data_boun
   -1  0;1  1  3]
scf(231);clf();subplot(1,3,1);param3d(x,y,z,0,0);subplot(1,3,2);param3d(
scf(232);clf();subplot(1,3,1);plot(x,y);gca().isoview='on';subplot(1,3,2
```

GNU Octave, gráficos nativos

```
t=linspace(0,6*pi,600);x=cos(t);y=sin(t);z=t/(2*pi);figure;plot3(x,y,z,'
   on;plot3(x(1),y(1),z(1),'go',x(end),y(end),z(end),'ro');hold
   off;axis([-1 1 -1 1 0 3]);daspect([1 1 1]);grid on;xlabel
   x;ylabel y;zlabel z;
figure;for k=1:3,subplot(1,3,k);plot3(x,y,z);axis([-1 1 -1 1
   0 3]);grid on;if k==1,view(0,90);elseif
   k==2,view(90,0);else,view(45,30);end;end;print('ej23_octave_vistas.pn
   equal;subplot(1,3,2);plot(x,z);subplot(1,3,3);plot(y,z);print('ej23_o
```

Python/Matplotlib

```
import numpy as np,matplotlib.pyplot as plt
t=np.linspace(0,6*np.pi,600);x=np.cos(t);y=np.sin(t);z=t/(2*np.pi);fig=p
for k,(el,az) in enumerate(views,1):
 a=fig.add_subplot(1,3,k,projection='3d');a.plot(x,y,z);a.scatter([x[0],
fig.savefig('ej23_python_vistas.png',dpi=140);fig,ax=plt.subplots(1,3,fi
```

Perl, gráficos nativos mediante proyección SVG

```
use strict;use
   warnings;my$pi=4*atan2(1,1);my@P=map{my$t=6*$pi*$_/599;[cos($t),sin($
   project{my($p,$az,$el,$ox,$oy)=@_;my$A=$az*$pi/180,$E=$el*$pi/180;my@
   ',@q)}open my$s,'>','ej23_perl.svg' or die$!;print$s'<svg
   xmlns="http://www.w3.org/2000/svg" width="900"
   height="650"><rect width="100%" height="100%"
   fill="white"/>';for
   my$v([0,90,150,180],[90,0,450,180],[45,35,750,180]){print$s'<polyline
   fill="none" stroke="black"
   points="'.project(\@P,@$v).'"/>'}my@xy=map{[$_->[0],$_->[1],0]}@P;my@
   fill="none" stroke="blue"
   points="'.project(\@xy,0,90,150,500).'"/><polyline
   fill="none" stroke="red"
   points="'.project(\@xz,0,0,450,500).'"/><polyline
   fill="none" stroke="green"
   points="'.project(\@yz,90,0,750,500).'"/></svg>';close$s;
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');const
   t=Array.from({length:600},(_,i)=>6*Math.PI*i/599),x=t.map(Math.cos),y
   h={x,y,z,type:'scatter3d',mode:'lines',name:'helice'};const
   proj=[{x,y,xaxis:'x',yaxis:'y',mode:'lines',name:'xy'},{x,y:z,xaxis:'
   html><script>${P}</script><div id="h"></div><div
   id="p"></div><script>Plotly.newPlot('h',[${JSON.stringify(h)}],{scene
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot

```
set terminal pngcairo size 1100,700;set output
   'ej23_gnuplot.png';set multiplot layout 2,3
set xrange [-1:1];set yrange [-1:1];set zrange [0:3];set
   view equal xy
```

```
set title 'superior';set view 90,0;splot 'ej23.dat' u 2:3:4
   w l notitle
set title 'lateral';set view 0,0;splot '' u 2:3:4 w l notitle
set title 'oblicua';set view 45,35;splot '' u 2:3:4 w l
   notitle
set title 'xy';plot '' u 2:3 w l notitle
set title 'xz';plot '' u 2:4 w l notitle
set title 'yz';plot '' u 3:4 w l notitle
unset multiplot;unset output
```

Scilab + Gnuplot

```
t=linspace(0,6*%pi,600)';x=cos(t);y=sin(t);z=t/(2*%pi);fprintfMat('ej23.
   x y z],'%15.12g');fprintfMat('ej23_extremos.dat',[[t(1)
   x(1) y(1) z(1)];[t($) x($) y($)
   z($)]],'%15.12g');[e,o,er]=host('gnuplot ej23.gnu');if
   e<>0 then error(er),end
```

GNU Octave + Gnuplot

```
t=linspace(0,6*pi,600)';x=cos(t);y=sin(t);z=t/(2*pi);dlmwrite('ej23.dat'
   x y z],'delimiter','
   ');dlmwrite('ej23_extremos.dat',[t([1,end]) x([1,end])
   y([1,end]) z([1,end])],'delimiter',' ');system('gnuplot
   ej23.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
t=np.linspace(0,6*np.pi,600);A=np.c_[t,np.cos(t),np.sin(t),t/(2*np.pi)];
```

Perl + Gnuplot

```
use strict;use warnings;my$pi=4*atan2(1,1);open
   my$f,'>','ej23.dat' or
   die$!;my@a;for(0..599){my$t=6*$pi*$_/599;my$r=[$t,cos($t),sin($t),$t/
   join(' ',@$r),"\n"}close$f;open
   my$e,'>','ej23_extremos.dat' or die$!;print$e join('
   ',@{$a[0]}),"\n",join('
   ',@{$a[-1]}),"\n";close$e;system('gnuplot','ej23.gnu')==0
   or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');let
   A=Array.from({length:600},(_,i)=>{let
   t=6*Math.PI*i/599;return[t,Math.cos(t),Math.sin(t),t/(2*Math.PI)]});f
   ')).join('\n')+'\n');fs.writeFileSync('ej23_extremos.dat',[A[0],A[A.l
   ')).join('\n')+'\n');let
   q=spawnSync('gnuplot',['ej23.gnu'],{encoding:'utf8'});if(q.status!==0
   new Error(q.stderr);
```

Los extremos se agregan al `splot` con `with points`; las cinco versiones reutilizan exactamente los mismos datos para las tres perspectivas y las tres proyecciones.

24. **Ejercicio:** Perspectivas de una superficie sinc

Considere

$$r = \sqrt{x^2 + y^2}$$

y

$$z(x, y) = \begin{cases} \dfrac{\sin r}{r}, & r \neq 0, \\ 1, & r = 0, \end{cases}$$

en

$$-8 \leq x \leq 8, \qquad -8 \leq y \leq 8.$$

a) Evalúe la función sobre una grilla de al menos 60×60 puntos, tratando correctamente el origen.

b) Genere una superficie tridimensional con mapa de colores.

c) Genere tres vistas: superior, lateral y oblicua.

d) Mantenga los mismos datos y la misma escala de color en las tres vistas.

e) Explique qué información sobre la simetría y las oscilaciones de la función resulta más clara en cada perspectiva.

Nota Gnuplot: Una especificación posible es:

```
set terminal png size 800,600
set output '/tmp/sinc2d.png'
set isosamples 60,60
set xrange [-8:8]
set yrange [-8:8]
set hidden3d
set pm3d depthorder
set palette defined (0 'blue', 0.5 'white', 1 'red')
set title 'sinc 2D'
set xlabel 'x'
set ylabel 'y'
set zlabel 'z'
set view 55,35
r(x,y) = sqrt(x**2 + y**2) + 1e-6
splot sin(r(x,y))/r(x,y) with pm3d notitle
```

Nota Gnuplot: Compare, por ejemplo, `set view 90,0`, `set view 0,0` y `set view 55,35`.

Respuesta (T: diez modalidades).

Se usa una grilla impar de 81×81, de manera que $(0, 0)$ pertenece a la grilla. En ese punto se asigna explícitamente $z = 1$; no se sustituye r por $r + \varepsilon$, porque eso cambia la función que se pretende representar. En las tres vistas se fijan los mismos límites y la misma escala cromática.

Scilab, gráficos nativos

```
x=linspace(-8,8,81);y=x;Z=zeros(81,81);for i=1:81,for
   j=1:81,r=sqrt(x(i)^2+y(j)^2);if r==0 then Z(i,j)=1;else
   Z(i,j)=sin(r)/r;end,end,end
scf(240);clf();subplot(1,3,1);surf(x,y,Z);gca().rotation_angles=[0
   270];subplot(1,3,2);surf(x,y,Z);gca().rotation_angles=[0
   0];subplot(1,3,3);surf(x,y,Z);gca().rotation_angles=[55
   35];colorbar(min(Z),max(Z));xs2png(gcf(),'ej24_scilab.png')
```

GNU Octave, gráficos nativos

```
x=linspace(-8,8,81);[X,Y]=meshgrid(x);R=hypot(X,Y);Z=ones(size(R));q=R!=
   90;90 0;45 35];for
   k=1:3,subplot(1,3,k);surf(X,Y,Z);shading
   interp;caxis([min(Z(:))
   max(Z(:))]);view(V(k,1),V(k,2));axis([-8 8 -8 8 min(Z(:))
   1]);end;colorbar;print('ej24_octave.png','-dpng');
```

Python/Matplotlib

```
import numpy as np,matplotlib.pyplot as plt
x=np.linspace(-8,8,81);X,Y=np.meshgrid(x,x);R=np.hypot(X,Y);Z=np.ones_li
for k,(el,az) in enumerate(views,1):
 a=fig.add_subplot(1,3,k,projection='3d');a.plot_surface(X,Y,Z,cmap='coo
fig.savefig('ej24_python.png',dpi=140)
```

Perl, gráficos nativos con PDL/PLplot

```
use strict;use warnings;use PDL;use PDL::Graphics::PLplot
   qw(pladv plvpor plwind plw3d plbox3
   plsurf3d);my$x=zeroes(81)->xlinvals(-8,8);my$y=$x->copy;my$X=$x->dumm
   draw{my($file,$alt,$az)=@_;my$pl=PDL::Graphics::PLplot->new(DEV=>'png
```

El valor 4 activa la coloración por magnitud en la opción de superficie de PLplot; los tres archivos reutilizan los mismos piddles x, y, Z.

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');const
   x=Array.from({length:81},(_,i)=>-8+16*i/80),z=x.map(y=>x.map(X=>{let
   r=Math.hypot(X,y);return r===0?1:Math.sin(r)/r}));const
   tr={type:'surface',x,y:x,z,colorscale:'RdBu',cmin:Math.min(...z.flat(
   cams=[{eye:{x:0,y:0,z:2.5}},{eye:{x:2.5,y:0,z:0}},{eye:{x:1.4,y:1.4,z
   html><script>${P}</script>${cams.map((_,i)=>'<div
   id="p'+i+'"></div>').join('')}<script>let
   t=${JSON.stringify(tr)},c=${JSON.stringify(cams)};c.forEach((v,i)=>Pl
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot. Los lenguajes calculan `ej24.dat` como grilla por bloques; Gnuplot no vuelve a evaluar la función.

```
set terminal pngcairo size 1100,400;set output
   'ej24_gnuplot.png';set multiplot layout 1,3
set xrange [-8:8];set yrange [-8:8];set zrange [-.25:1];set
   cbrange [-.25:1];set pm3d depthorder;set palette defined
   (-.25 'blue',0 'white',1 'red')
set title 'superior';set view 90,0;splot 'ej24.dat' u 1:2:3
   w pm3d notitle
set title 'lateral';set view 0,0;splot '' u 1:2:3 w pm3d
   notitle
set title 'oblicua';set view 55,35;splot '' u 1:2:3 w pm3d
   notitle
unset multiplot;unset output
```

Scilab + Gnuplot

```
x=linspace(-8,8,81);fd=mopen('ej24.dat','w');for
   i=1:81,y=x;r=sqrt(x(i)^2+y.^2);z=ones(y);q=find(r<>0);z(q)=sin(r(q)).
   y(:) z(:)];mfprintf(fd,'%.12g %.12g
   %.12g\n',M');mfprintf(fd,'\n');end;mclose(fd);[e,o,er]=host('gnuplot
   ej24.gnu');if e<>0 then error(er),end
```

GNU Octave + Gnuplot

```
x=linspace(-8,8,81);fid=fopen('ej24.dat','w');for
   X=x,r=hypot(X,x);z=ones(size(r));q=r!=0;z(q)=sin(r(q))./r(q);fprintf(
   %.12g
   %.12g\n',[X*ones(size(x));x;z]);fprintf(fid,'\n');end;fclose(fid);sys
   ej24.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
x=np.linspace(-8,8,81);f=open('ej24.dat','w');
for X in x:
 r=np.hypot(X,x);z=np.ones_like(r);q=r!=0;z[q]=np.sin(r[q])/r[q]
 for Y,Z in zip(x,z):f.write(f'{X:.12g} {Y:.12g} {Z:.12g}\n')
 f.write('\n')
f.close();subprocess.run(['gnuplot','ej24.gnu'],check=True)
```

Perl + Gnuplot

```
use strict;use warnings;open my$f,'>','ej24.dat' or
   die$!;for my$i(0..80){my$x=-8+16*$i/80;for
   my$j(0..80){my$y=-8+16*$j/80;my$r=sqrt($x*$x+$y*$y);my$z=$r==0?1:sin(
   $y
   $z\n"}print$f"\n"}close$f;system('gnuplot','ej24.gnu')==0
   or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');let
```

```
s='';for(let i=0;i<81;i++){let x=-8+16*i/80;for(let
j=0;j<81;j++){let
y=-8+16*j/80,r=Math.hypot(x,y),z=r===0?1:Math.sin(r)/r;s+=`${x}
${y} ${z}\n`}s+='\n'}fs.writeFileSync('ej24.dat',s);let
q=spawnSync('gnuplot',['ej24.gnu'],{encoding:'utf8'});if(q.status!==0
new Error(q.stderr);
```

La vista superior hace evidente la simetría radial y los anillos; la lateral muestra amplitudes y cambios de signo; la oblicua permite percibir simultáneamente altura, forma radial y amortiguamiento de las oscilaciones.

25. **Ejercicio:** Mapas de nivel con relleno

Considere

$$f(x, y) = \cos x \, e^{-y^2/4}, \qquad -4 \le x, y \le 4.$$

a) Evalúe la función sobre una grilla suficientemente fina.

b) Genere un mapa bidimensional con relleno de color según el valor de la función, usando una escala que distinga claramente valores negativos, cercanos a cero y positivos.

c) Superponga al menos diez curvas de nivel.

d) Resalte la curva de nivel $f(x, y) = 0$ mediante una línea negra de mayor grosor.

e) Incluya una barra de color y conserve igual escala en ambos ejes.

f) Explique qué relación existe entre las zonas de distinto color y las curvas de nivel.

Nota Gnuplot: Una base para el mapa es:

```
set terminal png size 800,600
set output '/tmp/nivel.png'
set title 'cos(x)*exp(-y^2/4)'
set xlabel 'x'
set ylabel 'y'
set xrange [-4:4]
set yrange [-4:4]
set view map
set pm3d map interpolate 3,3
set palette defined (-1 'blue', 0 'white', 1 'red')
set isosamples 80,80
set contour base
set cntrparam levels 10
set key off
set colorbox
splot cos(x)*exp(-y**2/4) with pm3d notitle
```

Respuesta (T: diez modalidades).

Como $e^{-y^2/4} > 0$ para todo y, el signo de f depende únicamente de $\cos x$. Por lo tanto, dentro de $[-4, 4]$ el nivel cero está formado por las dos rectas verticales $x = -\pi/2$ y $x = \pi/2$. Una paleta divergente centrada en cero hace visible esta separación.

Scilab, gráficos nativos

```
x=linspace(-4,4,121);y=x;Z=zeros(121,121);for
   i=1:121,Z(i,:)=cos(x(i))*exp(-y.^2/4);end
scf(250);clf();grayplot(x,y,Z);colorbar(-1,1);a=gca();a.auto_clear='off'
   -.6 -.4 -.2 0 .2 .4 .6 .8]);plot([-%pi/2 -%pi/2],[-4
   4],'k-',[%pi/2 %pi/2],[-4
   4],'k-');a.auto_clear='on';a.isoview='on';xgrid();xs2png(gcf(),'ej25_
```

GNU Octave, gráficos nativos

```
x=linspace(-4,4,121);[X,Y]=meshgrid(x);Z=cos(X).*exp(-Y.^2/4);pcolor(X,Y
   interp;caxis([-1 1]);colorbar;axis equal tight;hold
   on;contour(X,Y,Z,[-.8 -.6 -.4 -.2 .2 .4 .6
   .8],'k');contour(X,Y,Z,[0 0],'k','linewidth',3);hold
   off;print('ej25_octave.png','-dpng');
```

Python/Matplotlib

```
import numpy as np,matplotlib.pyplot as plt
x=np.linspace(-4,4,121);X,Y=np.meshgrid(x,x);Z=np.cos(X)*np.exp(-Y**2/4)
```

Perl, gráficos nativos con PDL/PLplot

```
use strict;use warnings;use PDL;use
   PDL::Graphics::PLplot;my$x=zeroes(121)->xlinvals(-4,4);my$X=$x->dummy
```

El nivel cero puede resaltarse además dibujando explícitamente las dos rectas $x = \pm\pi/2$, cuya posición se obtuvo analíticamente.

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');const
   x=Array.from({length:121},(_,i)=>-4+8*i/120),z=x.map(y=>x.map(X=>Math
   heat={type:'heatmap',x,y:x,z,zmin:-1,zmax:1,colorscale:'RdBu',colorba
   layout={xaxis:{scaleanchor:'y'},shapes:[{type:'line',x0:-Math.PI/2,x1
   html><script>${P}</script><div
   id="p"></div><script>Plotly.newPlot('p',${JSON.stringify([heat,cont])
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot

```
set terminal pngcairo size 750,650;set output
   'ej25_gnuplot.png';set view map;set size ratio -1;set
   xrange [-4:4];set yrange [-4:4];set cbrange [-1:1];set
   palette defined (-1 'blue',0 'white',1 'red');set pm3d
   map;set contour base;set cntrparam levels incremental
   -0.8,0.2,0.8;set colorbox
set arrow 1 from -pi/2,-4 to -pi/2,4 nohead front lw 3 lc
   'black';set arrow 2 from pi/2,-4 to pi/2,4 nohead front
   lw 3 lc 'black'
splot 'ej25.dat' u 1:2:3 w pm3d notitle
unset output
```

Scilab + Gnuplot

```
x=linspace(-4,4,121);fd=mopen('ej25.dat','w');for
    i=1:121,z=cos(x(i))*exp(-x.^2/4);M=[x(i)*ones(121,1) x(:)
    z(:)];mfprintf(fd,'%.12g %.12g
    %.12g\n',M');mfprintf(fd,'\n');end;mclose(fd);[e,o,er]=host('gnuplot
    ej25.gnu');if e<>0 then error(er),end
```

GNU Octave + Gnuplot

```
x=linspace(-4,4,121);fid=fopen('ej25.dat','w');for
    X=x,z=cos(X)*exp(-x.^2/4);fprintf(fid,'%.12g %.12g
    %.12g\n',[X*ones(size(x));x;z]);fprintf(fid,'\n');end;fclose(fid);sys
    ej25.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
x=np.linspace(-4,4,121);f=open('ej25.dat','w');
for X in x:
 for Y in x:f.write(f'{X:.12g} {Y:.12g}
    {np.cos(X)*np.exp(-Y*Y/4):.12g}\n')
 f.write('\n')
f.close();subprocess.run(['gnuplot','ej25.gnu'],check=True)
```

Perl + Gnuplot

```
use strict;use warnings;open my$f,'>','ej25.dat' or
    die$!;for my$i(0..120){my$x=-4+8*$i/120;for
    my$j(0..120){my$y=-4+8*$j/120;print$f"$x $y
    ",cos($x)*exp(-$y*$y/4),"\n"}print$f"\n"}close$f;system('gnuplot','ej
    or die;
```

JavaScript/Node.js + Gnuplot

```
const
    fs=require('node:fs'),{spawnSync}=require('node:child_process');let
    s='';for(let i=0;i<121;i++){let x=-4+8*i/120;for(let
    j=0;j<121;j++){let y=-4+8*j/120;s+=`${x} ${y}
    ${Math.cos(x)*Math.exp(-y*y/4)}\n`}s+='\n'}fs.writeFileSync('ej25.dat
    q=spawnSync('gnuplot',['ej25.gnu'],{encoding:'utf8'});if(q.status!==0
    new Error(q.stderr);
```

Las zonas de color representan intervalos de valores; las curvas de nivel son sus fronteras cuantitativas $f = c$. El relleno favorece una lectura global y los contornos permiten localizar con mayor precisión valores concretos.

26. **Ejercicio:** Refinamiento de grilla en contornos

Genere mapas de curvas de nivel para las funciones:

$$f_1(x, y) = x^2 - y^2, \qquad -10 \le x, y \le 10,$$

$$f_2(x, y) = \sin^2 x \cos y, \qquad -\pi \le x, y \le \pi,$$

$$f_3(x,y) = x^2 + y^2 - 2x^2y + 25, \qquad -2 \le x, y \le 2,$$
$$f_4(x,y) = x\sin y + y\sin x, \qquad -2 \le x, y \le 2.$$

a) Para f_1, f_3 y f_4, genere un mapa de al menos quince curvas de nivel y explique cualitativamente la geometría observada.

b) Para f_2, repita el gráfico con tres grillas, por ejemplo 11×11, 31×31 y 101×101.

c) Compare la posición y suavidad de las curvas al aumentar la resolución.

d) Explique qué representa matemáticamente cada curva y por qué la resolución de la grilla puede modificar su apariencia aunque no cambie la función.

Scilab. Si los datos se generan en Scilab, esto equivale a variar el paso o la cantidad de puntos de los vectores x e y.

Nota Gnuplot: En Gnuplot los contornos pueden obtenerse con `splot`, `set view map` y `set contour`; si se desea relleno de color puede combinarse con `pm3d map`. La resolución de evaluación puede controlarse con `set isosamples`.

```
cat > /tmp/contorno_a.gnu << 'EOF'
set terminal png size 700,600
set output '/tmp/contorno_a.png'

set title  'Curvas de nivel: f(x,y) = x^2 - y^2'
set xlabel 'x'
set ylabel 'y'
set xrange [-10:10]
set yrange [-10:10]
set size ratio -1
set grid

set view map
set pm3d map
set palette defined (-100 'blue', 0 'white', 100 'red')
set colorbox
set contour base
set cntrparam levels 20
set isosamples 80,80

splot x**2 - y**2 with pm3d notitle
EOF

gnuplot /tmp/contorno_a.gnu
```

```
cat > /tmp/contorno_a_lineas.gnu << 'EOF'
set terminal png size 700,600
set output '/tmp/contorno_a_lineas.png'

set title  'Curvas de nivel: x^2 - y^2'
set xlabel 'x'
set ylabel 'y'
set xrange [-10:10]
```

```
set yrange [-10:10]
set size ratio -1
set grid

set view map
set contour base
set cntrparam levels incremental -100,10,100
set isosamples 80,80
unset surface

set key outside right
splot x**2 - y**2 with lines lw 1.5
EOF

gnuplot /tmp/contorno_a_lineas.gnu
```

```
for ISO in 10 30 80; do

cat > /tmp/contorno_b_${ISO}.gnu << EOF
set terminal png size 700,600
set output '/tmp/contorno_b_${ISO}.png'

set title  'sin^2(x)*cos(y)  [isosamples=${ISO}]'
set xlabel 'x'
set ylabel 'y'
set xrange [-6.28:6.28]
set yrange [-6.28:6.28]
set size ratio -1

set view map
set pm3d map
set palette defined (-1 'blue', 0 'white', 1 'red')
set colorbox
set contour base
set cntrparam levels 16
set isosamples ${ISO},${ISO}

splot sin(x)**2 * cos(y) with pm3d notitle
EOF

gnuplot /tmp/contorno_b_${ISO}.gnu
echo "generado grilla ${ISO}x${ISO}:
   /tmp/contorno_b_${ISO}.png"

done
```

```
cat > /tmp/contorno_b_final.gnu << 'EOF'
set terminal png size 750,650
set output '/tmp/contorno_b_final.png'

set title  'sin^2(x)*cos(y)  [contornos etiquetados]'
```

```
set xlabel 'x'
set ylabel 'y'
set xrange [-6.28:6.28]
set yrange [-6.28:6.28]
set size ratio -1

set view map
set pm3d map
set palette defined (-1 'navy', -0.5 'blue', 0 'white', \
                     0.5 'red', 1 'darkred')
set colorbox
set isosamples 80,80
set contour base
set cntrparam levels incremental -1,0.2,1

splot sin(x)**2 * cos(y) with pm3d notitle, \
      sin(x)**2 * cos(y) with lines lw 0.8 lc 'black' notitle
EOF

gnuplot /tmp/contorno_b_final.gnu
```

```
gnuplot /tmp/contorno_a.gnu
gnuplot /tmp/contorno_a_lineas.gnu
for ISO in 10 30 80; do gnuplot /tmp/contorno_b_${ISO}.gnu;
   done
gnuplot /tmp/contorno_b_final.gnu
```

Scilab. Si se desea realizar los cálculos auxiliares en Scilab, puede usarse el siguiente procedimiento.

Scilab. Implementación o cálculo auxiliar en Scilab:

```
function z = f(x, y)
    z = x.^2 - y.^2
endfunction

x = -10 : 0.5 : 10
y = -10 : 0.5 : 10

fd = mopen('/tmp/surf_a.txt', 'w')
for i = 1:length(x)
    for j = 1:length(y)
        mfprintf(fd, '%.4f %.4f %.4f\n', x(i), y(j), f(x(i),
           y(j)))
    end
    mfprintf(fd, '\n')
end
mclose(fd)

NL  = ascii(10); Q = ascii(34)
DAT = Q + '/tmp/surf_a.txt' + Q
```

```
// --- superficie con color (equivale a surf) ---
gp = 'set terminal png size 800,600' + NL
gp = gp + 'set output ' + Q + '/tmp/surf_a1.png' + Q + NL
gp = gp + 'set title  ' + Q + 'f(x,y) = x^2 - y^2' + Q + NL
gp = gp + 'set xlabel ' + Q + 'x' + Q + NL
gp = gp + 'set ylabel ' + Q + 'y' + Q + NL
gp = gp + 'set zlabel ' + Q + 'z' + Q + NL
gp = gp + 'set hidden3d' + NL
gp = gp + 'set pm3d depthorder' + NL
gp = gp + 'set palette rgbformulae 33,13,10' + NL
gp = gp + 'set view 50,35' + NL
gp = gp + 'set colorbox' + NL
gp = gp + 'splot ' + DAT + ' with pm3d notitle' + NL
fd = mopen('/tmp/surf_a1.gnu','w'); mfprintf(fd,'%s',gp);
   mclose(fd)
host('gnuplot /tmp/surf_a1.gnu')

// --- superficie + malla superpuesta ---
gp = 'set terminal png size 800,600' + NL
gp = gp + 'set output ' + Q + '/tmp/surf_a2.png' + Q + NL
gp = gp + 'set title  ' + Q + 'f(x,y) = x^2 - y^2  [pm3d +
   malla]' + Q + NL
gp = gp + 'set xlabel ' + Q + 'x' + Q + NL
gp = gp + 'set ylabel ' + Q + 'y' + Q + NL
gp = gp + 'set zlabel ' + Q + 'z' + Q + NL
gp = gp + 'set hidden3d; set pm3d depthorder' + NL
gp = gp + 'set palette rgbformulae 33,13,10' + NL
gp = gp + 'set view 50,35' + NL
gp = gp + 'splot ' + DAT + ' with pm3d notitle, \' + NL
gp = gp + '      ' + DAT + ' with lines lw 0.4 lc ' +
   Q+'black'+Q+' notitle' + NL
fd = mopen('/tmp/surf_a2.gnu','w'); mfprintf(fd,'%s',gp);
   mclose(fd)
host('gnuplot /tmp/surf_a2.gnu')

// --- curvas de nivel ---
gp = 'set terminal png size 700,600' + NL
gp = gp + 'set output ' + Q + '/tmp/surf_a3.png' + Q + NL
gp = gp + 'set title  ' + Q + 'Curvas de nivel: x^2 - y^2' +
   Q + NL
gp = gp + 'set xlabel ' + Q + 'x' + Q + NL
gp = gp + 'set ylabel ' + Q + 'y' + Q + NL
gp = gp + 'set size ratio -1' + NL
gp = gp + 'set view map; set pm3d map' + NL
gp = gp + 'set contour base; set cntrparam levels 16' + NL
gp = gp + 'set palette defined (-100 '+Q+'blue'+Q + ..
          ', 0 '+Q+'white'+Q+', 100 '+Q+'red'+Q+')' + NL
gp = gp + 'set colorbox' + NL
gp = gp + 'splot ' + DAT + ' with pm3d notitle' + NL
fd = mopen('/tmp/surf_a3.gnu','w'); mfprintf(fd,'%s',gp);
```

```
    mclose(fd)
host('gnuplot /tmp/surf_a3.gnu')

printf('Graficas:\n')
printf('  /tmp/surf_a1.png  (superficie con color)\n')
printf('  /tmp/surf_a2.png  (superficie + malla)\n')
printf('  /tmp/surf_a3.png  (curvas de nivel)\n')
```

```
function z = g(x, y)
    z = sin(x).^2 .* cos(y)
endfunction

NL = ascii(10); Q = ascii(34)

// tres resoluciones de grilla
// equivale a cambiar el paso en surf(meshgrid(x,y))
pasos   = [1.0,   0.3,   0.1]
sufijos = ['gruesa', 'media', 'fina']

for k = 1:3
    paso = pasos(k)
    suf  = sufijos(k)

    x = -%pi : paso : %pi
    y = -%pi : paso : %pi

    nombre = '/tmp/surf_b_' + suf + '.txt'
    fd = mopen(nombre, 'w')
    for i = 1:length(x)
        for j = 1:length(y)
            mfprintf(fd,'%.6f %.6f %.6f\n', x(i), y(j),
               g(x(i),y(j)))
        end
        mfprintf(fd,'\n')
    end
    mclose(fd)

    DAT    = Q + nombre + Q
    salida = '/tmp/surf_b_' + suf + '.png'

    gp = 'set terminal png size 800,600' + NL
    gp = gp + 'set output ' + Q + salida + Q + NL
    gp = gp + 'set title  ' + Q + 'sin^2(x)*cos(y)  [grilla
       ' + suf + ']' + Q + NL
    gp = gp + 'set xlabel ' + Q + 'x' + Q + NL
    gp = gp + 'set ylabel ' + Q + 'y' + Q + NL
    gp = gp + 'set zlabel ' + Q + 'z' + Q + NL
    gp = gp + 'set hidden3d; set pm3d depthorder' + NL
    gp = gp + 'set palette rgbformulae 33,13,10' + NL
    gp = gp + 'set view 55,40; set colorbox' + NL
    gp = gp + 'splot ' + DAT + ' with pm3d notitle' + NL
```

```
    gnu = '/tmp/surf_b_' + suf + '.gnu'
    fd = mopen(gnu,'w'); mfprintf(fd,'%s',gp); mclose(fd)
    host('gnuplot ' + gnu)
    printf('Grilla %s (paso=%.1f): %s\n', suf, paso, salida)
end

// opciones de GlobalProperty con grilla media
DAT = Q + '/tmp/surf_b_media.txt' + Q

// opcion 1: paleta fria
gp = 'set terminal png size 800,600' + NL
gp = gp + 'set output ' + Q + '/tmp/surf_b_fria.png' + Q + NL
gp = gp + 'set title  ' + Q + 'Paleta fria' + Q + NL
gp = gp + 'set hidden3d; set pm3d depthorder; set view
   55,40' + NL
gp = gp + 'set palette defined (0 '+Q+'navy'+Q + ..
          ', 0.5 '+Q+'cyan'+Q+', 1 '+Q+'green'+Q+')' + NL
gp = gp + 'splot ' + DAT + ' with pm3d notitle' + NL
fd = mopen('/tmp/surf_fria.gnu','w'); mfprintf(fd,'%s',gp);
   mclose(fd)
host('gnuplot /tmp/surf_fria.gnu')

// opcion 2: escala de grises
gp = 'set terminal png size 800,600' + NL
gp = gp + 'set output ' + Q + '/tmp/surf_b_grises.png' + Q +
   NL
gp = gp + 'set title  ' + Q + 'Escala de grises' + Q + NL
gp = gp + 'set hidden3d; set pm3d depthorder; set view
   55,40' + NL
gp = gp + 'set palette grey' + NL
gp = gp + 'splot ' + DAT + ' with pm3d notitle' + NL
fd = mopen('/tmp/surf_grey.gnu','w'); mfprintf(fd,'%s',gp);
   mclose(fd)
host('gnuplot /tmp/surf_grey.gnu')

// opción 3: solo malla de alambre sin color
gp = 'set terminal png size 800,600' + NL
gp = gp + 'set output ' + Q + '/tmp/surf_b_malla.png' + Q +
   NL
gp = gp + 'set title  ' + Q + 'Malla de alambre' + Q + NL
gp = gp + 'set xlabel ' + Q + 'x' + Q + NL
gp = gp + 'set ylabel ' + Q + 'y' + Q + NL
gp = gp + 'set hidden3d; set view 55,40' + NL
gp = gp + 'splot ' + DAT + ' with lines lw 1 lc
   '+Q+'blue'+Q+' notitle' + NL
fd = mopen('/tmp/surf_malla.gnu','w'); mfprintf(fd,'%s',gp);
   mclose(fd)
host('gnuplot /tmp/surf_malla.gnu')

// opcion 4: vista cenital (equivale a view(0,90) en Scilab)
```

```
gp = 'set terminal png size 700,600' + NL
gp = gp + 'set output ' + Q + '/tmp/surf_b_cenital.png' + Q
   + NL
gp = gp + 'set title  ' + Q + 'Vista cenital' + Q + NL
gp = gp + 'set xlabel ' + Q + 'x' + Q + NL
gp = gp + 'set ylabel ' + Q + 'y' + Q + NL
gp = gp + 'set size ratio -1' + NL
gp = gp + 'set view map; set pm3d map' + NL
gp = gp + 'set palette rgbformulae 33,13,10; set colorbox' +
   NL
gp = gp + 'splot ' + DAT + ' with pm3d notitle' + NL
fd = mopen('/tmp/surf_cenital.gnu','w');
   mfprintf(fd,'%s',gp); mclose(fd)
host('gnuplot /tmp/surf_cenital.gnu')

printf('Opciones de GlobalProperty:\n')
printf('  /tmp/surf_b_fria.png     (paleta fria)\n')
printf('  /tmp/surf_b_grises.png   (escala de grises)\n')
printf('  /tmp/surf_b_malla.png    (malla de alambre)\n')
printf('  /tmp/surf_b_cenital.png  (vista cenital)\n')
```

Scilab. A continuación se incorpora la respuesta y el material nativo de Scilab de `eu07b`. La nota equivalente sobre variar el tamaño de la grilla no se repite porque ya figura arriba.

Respuesta.

Evaluar si usar contour o contour2d.

- contour dibuja las curvas sobre una superficie 3D (se usa junto con plot3d)
- contour2d dibuja las curvas en un plano 2D, que es lo correcto para mapas de contorno

La diferencia entre `contour` y `contour2d`:

Función	Contexto	Uso
`contour2d`	Gráfico 2D plano	Curvas de nivel en un plano
`contour`	Gráfico 3D	Curvas sobre la superficie 3D
`contour(...,zval=v)`	Gráfico 3D	Curvas proyectadas a altura v

Usaremos `contour2d` y `Sgrayplot`. El tamaño de la grilla se controla con el numero de puntos en los vectores x e y. `contour2d` requiere `frameflag=0` para no borrar el relleno de `Sgrayplot`. La grilla con `xgrid()` debe dibujarse después del relleno.

1. **Item:** $f(x,y) = x^2 - y^2$, $-10 \leq x, y \leq 10$:

```
function z = f(x, y)
    z = x.^2 - y.^2
endfunction
```

```
x = linspace(-10, 10, 80)
y = linspace(-10, 10, 80)

Z = zeros(length(x), length(y))
for i = 1:length(x)
    for j = 1:length(y)
        Z(i,j) = f(x(i), y(j))
    end
end

// --- relleno de color + curvas de nivel ---
clf()
Sgrayplot(x, y, Z)
// frameflag=0: no redibujar ejes ni borrar el Sgrayplot
contour2d(x, y, Z, 20, frameflag=0)
xgrid()   // la grilla va despues del relleno
title('Curvas de nivel: f(x,y) = x^2 - y^2')
xlabel('x')
ylabel('y')
colorbar(-200, 200)
xs2png(gcf(), '/tmp/contorno_a_color.png')

// --- solo líneas de contorno sin relleno ---
clf()
// sin Sgrayplot previo, frameflag puede ser 2 (calcula
   rango automatico)
contour2d(x, y, Z, 15, frameflag=2)
xgrid()
title('Curvas de nivel (solo líneas): x^2 - y^2')
xlabel('x')
ylabel('y')
xs2png(gcf(), '/tmp/contorno_a_lineas.png')

// --- niveles especificos con vector ---
clf()
niveles = -180:30:180
Sgrayplot(x, y, Z)
contour2d(x, y, Z, niveles, frameflag=0)
xgrid()
title('Curvas de nivel con niveles especificos')
xlabel('x')
ylabel('y')
colorbar(-200, 200)
xs2png(gcf(), '/tmp/contorno_a_niveles.png')
```

2. **Item:** $f(x, y) = \sin^2(x)\cos(y)$ con tres tamaños de grilla:

```
function z = g(x, y)
    z = sin(x).^2 .* cos(y)
endfunction

// tres resoluciones de grilla
```

```
npts    = [11,       21,       63    ]
nombres = ['gruesa', 'media', 'fina']

for k = 1:3
    n = npts(k)
    x = linspace(-%pi, %pi, n)
    y = linspace(-%pi, %pi, n)

    Z = zeros(n, n)
    for i = 1:n
        for j = 1:n
            Z(i,j) = g(x(i), y(j))
        end
    end

    clf()
    Sgrayplot(x, y, Z)
    contour2d(x, y, Z, 12, frameflag=0)
    xgrid()
    title('sin^2(x)*cos(y)  [grilla ' + nombres(k) + ..
          '  ' + string(n) + 'x' + string(n) + ']')
    xlabel('x')
    ylabel('y')
    colorbar(-1, 1)

    salida = '/tmp/contorno_b_' + nombres(k) + '.png'
    xs2png(gcf(), salida)
    printf('grilla %s (%dx%d): %s\n', nombres(k), n, n,
       salida)
end
```

Versión de alta resolución con curvas en negro superpuestas:

```
n = 100
x = linspace(-%pi, %pi, n)
y = linspace(-%pi, %pi, n)

Z = zeros(n, n)
for i = 1:n
    for j = 1:n
        Z(i,j) = g(x(i), y(j))
    end
end

clf()
gcf().figure_size = [700, 650]

Sgrayplot(x, y, Z)
contour2d(x, y, Z, 16, frameflag=0)

// poner las curvas de contorno en negro
a      = gca()
```

```
curvas = a.children
for i = 1:length(curvas)
    if curvas(i).type == 'Polyline' then
        curvas(i).foreground = color('black')
        curvas(i).thickness  = 1
    end
end

xgrid()
title('sin^2(x)*cos(y)  [relleno + contornos en negro]')
xlabel('x')
ylabel('y')
colorbar(-1, 1)
xs2png(gcf(), '/tmp/contorno_b_final.png')
```

La tabla de funciones de contorno en Scilab:

Función	**Descripción**
`contour2d(x,y,Z,n)`	n curvas de nivel en 2D
`contour2d(x,y,Z,v)`	curvas en los valores del vector v
`contour2d(...,frameflag=0)`	no redibujar ejes (superponer)
`contour2d(...,frameflag=2)`	calcular rango automático
`grayplot(x,y,Z)`	mapa de color sin suavizado
`Sgrayplot(x,y,Z)`	mapa de color suavizado
`colorbar(zmin,zmax)`	barra de color lateral
`xs2png(gcf(),'f.png')`	exportar a PNG

La diferencia entre los tamaños de grilla:

npts	**Puntos**	**Descripción**	**Archivo**
11	121	Gruesa, contornos angulosos	`contorno_b_gruesa.png`
21	441	Media, contornos aceptables	`contorno_b_media.png`
63	3969	Fina, contornos suaves	`contorno_b_fina.png`
100	10000	Alta resolución	`contorno_b_final.png`

El orden correcto de llamadas para superponer relleno y contornos:

```
clf()
Sgrayplot(x, y, Z)                      // 1. relleno de
   color
contour2d(x, y, Z, n, frameflag=0)      // 2. curvas
   (sin borrar ejes)
xgrid()                                 // 3. grilla
   (encima del relleno)
title('...'); xlabel('x'); ylabel('y')  // 4. etiquetas
colorbar(zmin, zmax)                    // 5. barra de
   color
xs2png(gcf(), 'salida.png')             // 6. exportar
```

Mostraremos también como se usa `contour`, en 3D.

```
function z = f(x, y)
    z = sin(x).^2 .* cos(y)
endfunction

x = linspace(-%pi, %pi, 60)
y = linspace(-%pi, %pi, 60)

Z = zeros(length(x), length(y))
for i = 1:length(x)
    for j = 1:length(y)
        Z(i,j) = f(x(i), y(j))
    end
end

// --- superficie 3D con curvas de nivel sobre ella ---
clf()
plot3d1(x, y, Z)        // superficie con color
contour(x, y, Z, 12)    // curvas de nivel SOBRE la
   superficie 3D
title('sin^2(x)*cos(y): superficie + contornos 3D')
xlabel('x'); ylabel('y')
xs2png(gcf(), '/tmp/contorno3d_a.png')

// --- rotar el punto de vista ---
clf()
plot3d1(x, y, Z, 60, 45)     // elevacion=60, rotacion=45
contour(x, y, Z, 12)
title('Vista rotada: elevacion 60, rotacion 45')
xlabel('x'); ylabel('y')
xs2png(gcf(), '/tmp/contorno3d_b.png')

// --- contornos proyectados en la base (zval fijo) ---
// usando contour con el parametro zval
clf()
plot3d1(x, y, Z)
// zval=-1 proyecta las curvas en el piso del grafico
contour(x, y, Z, 10, zval=-1)
title('Contornos proyectados en la base')
xlabel('x'); ylabel('y')
xs2png(gcf(), '/tmp/contorno3d_c.png')

// --- superficie + contornos en la base + vista frontal ---
clf()
gcf().figure_size = [700, 600]
plot3d1(x, y, Z, 0, 270)     // vista cenital
contour(x, y, Z, 16, zval=min(Z)-0.1)
title('Vista superior con contornos en la base')
xlabel('x'); ylabel('y')
xs2png(gcf(), '/tmp/contorno3d_d.png')
```

Respuesta (T: diez modalidades).

Cada curva de nivel es el conjunto

$$C_c = \{(x, y) : f(x, y) = c\}.$$

La función no cambia al refinar la grilla: cambia la cantidad de información numérica disponible para interpolar la posición de C_c. Una grilla gruesa puede desplazar vértices, crear segmentos angulosos o incluso omitir pequeñas estructuras.

Scilab, gráficos nativos

```
function Z=gridfun(x,y,k),Z=zeros(length(x),length(y));for
   i=1:length(x),select k,case 1 then
   Z(i,:)=x(i)^2-y.^2;case 2 then
   Z(i,:)=sin(x(i))^2*cos(y);case 3 then
   Z(i,:)=x(i)^2+y.^2-2*x(i)^2*y+25;case 4 then
   Z(i,:)=x(i)*sin(y)+y*sin(x(i));end;end,endfunction
scf(260);clf();for k=[1 3 4],subplot(2,2,find([1 3
   4]==k));a=10;if k>1 then
   a=2;end;x=linspace(-a,a,101);Z=gridfun(x,x,k);contour2d(x,x,Z,15);gca
scf(261);clf();N=[11 31 101];for
   q=1:3,x=linspace(-%pi,%pi,N(q));Z=gridfun(x,x,2);subplot(1,3,q);conto
```

GNU Octave, gráficos nativos

```
F={@(X,Y)X.^2-Y.^2,@(X,Y)sin(X).^2.*cos(Y),@(X,Y)X.^2+Y.^2-2*X.^2.*Y+25,
   10],[-pi pi],[-2 2],[-2 2]};figure;for k=[1 3
   4],r=R{k};x=linspace(r(1),r(2),101);[X,Y]=meshgrid(x);subplot(1,3,fin
   3 4]==k));contour(X,Y,F{k}(X,Y),15);axis equal;grid
   on;end;figure;N=[11 31 101];for
   q=1:3,x=linspace(-pi,pi,N(q));[X,Y]=meshgrid(x);subplot(1,3,q);contou
   equal;title(sprintf('%dx%d',N(q),N(q)));end;print('ej26_octave_refina
```

Python/Matplotlib

```
import numpy as np,matplotlib.pyplot as plt
F=[lambda X,Y:X*X-Y*Y,lambda
   X,Y:np.sin(X)**2*np.cos(Y),lambda
   X,Y:X*X+Y*Y-2*X*X*Y+25,lambda
   X,Y:X*np.sin(Y)+Y*np.sin(X)];R=[(-10,10),(-np.pi,np.pi),(-2,2),(-2,2)
for a,k in
   zip(ax,[0,2,3]):x=np.linspace(*R[k],101);X,Y=np.meshgrid(x,x);a.conto
fig.savefig('ej26_python_funciones.png',dpi=140);fig,ax=plt.subplots(1,3
for a,n in
   zip(ax,[11,31,101]):x=np.linspace(-np.pi,np.pi,n);X,Y=np.meshgrid(x,x
fig.savefig('ej26_python_refina.png',dpi=140)
```

Perl, gráficos nativos con PDL/PLplot

```
use strict;use warnings;use PDL;use
   PDL::Graphics::PLplot;my$pi=4*atan2(1,1);sub
   draw{my($file,$n,$a,$F)=@_;my$x=zeroes($n)->xlinvals(-$a,$a);my$X=$x-
   my$n(11,31,101){draw("ej26_f2_$n.png",$n,$pi,sub{sin($_[0])**2*cos($_
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const fs=require('node:fs');const
   P=fs.readFileSync(require.resolve('plotly.js-dist-min'),'utf8');funct
   seq(a,b,n){return
   Array.from({length:n},(_,i)=>a+(b-a)*i/(n-1))}function
   C(a,n,F){let
   x=seq(-a,a,n);return{type:'contour',x,y:x,z:x.map(y=>x.map(x=>F(x,y))
   f1=(x,y)=>x*x-y*y,f2=(x,y)=>Math.sin(x)**2*Math.cos(y),f3=(x,y)=>x*x+
   plots=[C(10,101,f1),C(2,101,f3),C(2,101,f4),C(Math.PI,11,f2),C(Math.P
   html><script>${P}</script>${plots.map((_,i)=>'<div
   id="p'+i+'"></div>').join('')}<script>let
   q=${JSON.stringify(plots)};q.forEach((v,i)=>Plotly.newPlot('p'+i,[v],
```

Gnuplot común a las cinco variantes lenguaje + Gnuplot. Cada archivo contiene una grilla por bloques. Para f_1, f_3, f_4 se usan 101×101 puntos; para f_2 se generan tres archivos con 11, 31 y 101 puntos por eje.

```
set terminal pngcairo size 1100,700;set output
   'ej26_gnuplot.png';set view map;unset surface;set contour
   base;set cntrparam levels 15;set size ratio -1;set
   multiplot layout 2,3
set title 'f1 101x101';splot 'ej26_f1.dat' u 1:2:3 w l
   notitle
set title 'f3 101x101';splot 'ej26_f3.dat' u 1:2:3 w l
   notitle
set title 'f4 101x101';splot 'ej26_f4.dat' u 1:2:3 w l
   notitle
set title 'f2 11x11';splot 'ej26_f2_11.dat' u 1:2:3 w l
   notitle
set title 'f2 31x31';splot 'ej26_f2_31.dat' u 1:2:3 w l
   notitle
set title 'f2 101x101';splot 'ej26_f2_101.dat' u 1:2:3 w l
   notitle
unset multiplot;unset output
```

Scilab + Gnuplot

```
function
   wr(nom,a,n,k),x=linspace(-a,a,n);fd=mopen(nom,'w');for
   i=1:n,select k,case 1 then z=x(i)^2-x.^2;case 2 then
   z=sin(x(i))^2*cos(x);case 3 then
   z=x(i)^2+x.^2-2*x(i)^2*x+25;case 4 then
   z=x(i)*sin(x)+x*sin(x(i));end;M=[x(i)*ones(n,1) x(:)
   z(:)];mfprintf(fd,'%.10g %.10g
   %.10g\n',M');mfprintf(fd,'\n');end;mclose(fd),endfunction
wr('ej26_f1.dat',10,101,1);wr('ej26_f3.dat',2,101,3);wr('ej26_f4.dat',2,
   ej26.gnu');if e<>0 then error(er),end
```

GNU Octave + Gnuplot

```
function
   wr(nom,a,n,F),x=linspace(-a,a,n);fid=fopen(nom,'w');for
   X=x,z=F(X,x);fprintf(fid,'%.10g %.10g
   %.10g\n',[X*ones(size(x));x;z]);fprintf(fid,'\n');end;fclose(fid);end
wr('ej26_f1.dat',10,101,@(x,y)x.^2-y.^2);wr('ej26_f3.dat',2,101,@(x,y)x.
   n=[11 31
   101],wr(sprintf('ej26_f2_%d.dat',n),pi,n,@(x,y)sin(x).^2.*cos(y));end
   ej26.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
def wr(name,a,n,F):
 x=np.linspace(-a,a,n);f=open(name,'w')
 for X in x:
  for Y in x:f.write(f'{X:.10g} {Y:.10g} {F(X,Y):.10g}\n')
  f.write('\n')
 f.close()
wr('ej26_f1.dat',10,101,lambda
   x,y:x*x-y*y);wr('ej26_f3.dat',2,101,lambda
   x,y:x*x+y*y-2*x*x*y+25);wr('ej26_f4.dat',2,101,lambda
   x,y:x*np.sin(y)+y*np.sin(x));
for n in (11,31,101):wr(f'ej26_f2_{n}.dat',np.pi,n,lambda
   x,y:np.sin(x)**2*np.cos(y))
subprocess.run(['gnuplot','ej26.gnu'],check=True)
```

Perl + Gnuplot

```
use strict;use warnings;my$pi=4*atan2(1,1);sub
   wr{my($nom,$a,$n,$F)=@_;open my$f,'>',$nom or
   die$!;for(0..$n-1){my$x=-$a+2*$a*$_/($n-1);for
   my$j(0..$n-1){my$y=-$a+2*$a*$j/($n-1);print$f"$x $y
   ",$F->($x,$y),"\n"}print$f"\n"}close$f}wr('ej26_f1.dat',10,101,sub{$_
   my$n(11,31,101){wr("ej26_f2_$n.dat",$pi,$n,sub{sin($_[0])**2*cos($_[1
   or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');funct
   wr(name,a,n,F){let s='';for(let i=0;i<n;i++){let
   x=-a+2*a*i/(n-1);for(let j=0;j<n;j++){let
   y=-a+2*a*j/(n-1);s+=`${x} ${y}
   ${F(x,y)}\n`}s+='\n'}fs.writeFileSync(name,s)}wr('ej26_f1.dat',10,101
   n
   of[11,31,101])wr(`ej26_f2_${n}.dat`,Math.PI,n,(x,y)=>Math.sin(x)**2*M
   q=spawnSync('gnuplot',['ej26.gnu'],{encoding:'utf8'});if(q.status!==0
   new Error(q.stderr);
```

Para f_1 se obtienen hipérbolas (y el nivel cero son las rectas $y = \pm x$). En f_2 aparecen lóbulos periódicos cuya forma es especialmente sensible a una grilla 11×11. f_3 presenta una geometría deformada por el término $-2x^2y$, y f_4 conserva la

simetría asociada al intercambio de x e y. El refinamiento hace converger la poligonal interpolada hacia la curva de nivel real, sin modificar el conjunto matemático que se intenta aproximar.

27. **Ejercicio:** Interacción con curvas de nivel mediante el ratón

Considere una función de dos variables

$$z = f(x, y)$$

definida sobre una región rectangular del plano.

a) Genere una grilla suficientemente fina y represente la función mediante curvas de nivel.

b) Permita recorrer gráficamente la región representada con el puntero del ratón.

c) Cuando se presione uno de los botones del ratón sobre un punto (x, y) del gráfico, calcule el valor

$$f(x, y)$$

y muestre en la propia figura las coordenadas del punto y el valor de la función.

d) Cada vez que se seleccione un punto de esta forma, deje junto a él una etiqueta permanente con un número correlativo:

$$1,\ 2,\ 3,\ \ldots$$

e) Cuando se presione otro botón del ratón, dibuje un marcador en la posición seleccionada.

f) Permita realizar sucesivamente varias selecciones, conservando en la figura tanto las etiquetas numeradas como los marcadores ya creados.

g) Verifique que las coordenadas obtenidas del evento del ratón correspondan a las coordenadas de los ejes y no a coordenadas de pantalla.

Respuesta (I: cinco variantes interactivas nativas).

Para concretar la consigna se toma

$$f(x, y) = x^2 + y^2, \qquad -2 \le x, y \le 2,$$

cuyas curvas de nivel son circunferencias. En todos los casos el clic se convierte a *coordenadas de los ejes*; no se usan coordenadas de pantalla para evaluar la función. El botón principal calcula $f(x, y)$, muestra las coordenadas y deja una etiqueta numerada; el botón secundario agrega un marcador. Las selecciones anteriores permanecen visibles.

Scilab

```
x=linspace(-2,2,101); y=x; Z=zeros(101,101)
for i=1:101, Z(i,:)=x(i)^2+y.^2; end
scf(270); clf(); contour2d(x,y,Z,12); a=gca(); a.isoview='on'
a.auto_clear='off'; n=0
while n<10
```

```
    [b,xc,yc]=xclick()
    if b==0 then
        n=n+1; z=xc^2+yc^2
        plot(xc,yc,'ro')
        xstring(xc,yc,msprintf('%d: (%.2f,%.2f)
           f=%.3f',n,xc,yc,z))
    elseif b==2 then
        plot(xc,yc,'kx')
    end
end
a.auto_clear='on'
```

GNU Octave

```
x=linspace(-2,2,101); [X,Y]=meshgrid(x); Z=X.^2+Y.^2;
contour(X,Y,Z,12); axis equal; grid on; hold on; n=0;
while n<10
  [xc,yc,b]=ginput(1); if isempty(b), break; endif
  if b==1
    n++; z=xc^2+yc^2; plot(xc,yc,'ro');
    text(xc,yc,sprintf('%d: (%.2f,%.2f) f=%.3f',n,xc,yc,z));
  elseif b==3
    plot(xc,yc,'kx','markersize',10,'linewidth',2);
  endif
endwhile
hold off;
```

Python/Matplotlib

```
import numpy as np, matplotlib.pyplot as plt
x=np.linspace(-2,2,101); X,Y=np.meshgrid(x,x); Z=X*X+Y*Y
fig,ax=plt.subplots(); ax.contour(X,Y,Z,12);
   ax.set_aspect('equal')
info=ax.text(.02,.98,'',transform=ax.transAxes,va='top');
   count=0
def click(ev):
    global count
    if ev.inaxes is not ax or ev.xdata is None: return
    x,y=ev.xdata,ev.ydata; z=x*x+y*y
    info.set_text(f'(x,y)=({x:.3f},{y:.3f})  f={z:.4f}')
    if ev.button==1:
        count+=1; ax.plot(x,y,'ro')
        ax.text(x,y,f'{count}',ha='left',va='bottom')
    elif ev.button==3:
        ax.plot(x,y,'kx',ms=9,mew=2)
    fig.canvas.draw_idle()
fig.canvas.mpl_connect('button_press_event',click)
plt.show()
```

Perl/Tk. En este caso las curvas de nivel se dibujan exactamente como circunferencias, por lo que no hace falta implementar un algoritmo de contornos.

```
use strict; use warnings; use Tk;
my $mw=MainWindow->new; my
   $c=$mw->Canvas(-width=>560,-height=>560)->pack;
my($o,$s)=(280,110);                    # origen y pixeles por
   unidad
for my $lev (0.5,1,2,3,4,5,6,7) {
    my $r=sqrt($lev)*$s;
    $c->createOval($o-$r,$o-$r,$o+$r,$o+$r,-outline=>'gray');
}
my $n=0;
sub datos { my($X,$Y)=@_; (($X-$o)/$s,($o-$Y)/$s) }
$c->bind('<Button-1>'=>sub {
    my $e=$c->XEvent; my($X,$Y)=($e->x,$e->y);
       my($x,$y)=datos($X,$Y);
    my $z=$x*$x+$y*$y; ++$n;
    $c->createOval($X-3,$Y-3,$X+3,$Y+3,-fill=>'red');
    $c->createText($X+7,$Y-8,-anchor=>'w',
      -text=>sprintf('%d (%.2f,%.2f) f=%.3f',$n,$x,$y,$z));
});
$c->bind('<Button-3>'=>sub {
    my $e=$c->XEvent; my($X,$Y)=($e->x,$e->y);
    $c->createLine($X-6,$Y,$X+6,$Y,-width=>2);
    $c->createLine($X,$Y-6,$X,$Y+6,-width=>2);
});
MainLoop;
```

JavaScript/Node.js. Node genera un HTML autocontenido con un `canvas`; el evento del navegador se transforma explícitamente a coordenadas matemáticas.

```
const fs=require('node:fs');
const html=`<canvas id="c" width="560"
   height="560"></canvas><div id="info"></div>
<script>
const
   c=document.getElementById('c'),g=c.getContext('2d'),o=280,s=110;let
   n=0;
function
   base(){g.clearRect(0,0,560,560);g.strokeStyle='#aaa';
 for(const L of
    [.5,1,2,3,4,5,6,7]){g.beginPath();g.arc(o,o,Math.sqrt(L)*s,0,2*Math.
function data(e){return[(e.offsetX-o)/s,(o-e.offsetY)/s]}
base(); c.oncontextmenu=e=>e.preventDefault();
c.addEventListener('mousedown',e=>{let[x,y]=data(e),z=x*x+y*y;
 document.getElementById('info').textContent='x='+x.toFixed(3)+'
    y='+y.toFixed(3)+' f='+z.toFixed(4);
 if(e.button===0){n++;g.fillStyle='red';g.beginPath();g.arc(e.offsetX,e.
 if(e.button===2){g.beginPath();g.moveTo(e.offsetX-6,e.offsetY);g.lineTo
});</script>`;
fs.writeFileSync('ej27_node.html',html);
```

Una ejecución de Gnuplot puramente por lotes no satisface el objetivo central de este ejercicio, porque no mantiene un bucle de eventos del usuario ni objetos interactivos asociados a cada clic. Por ese motivo no se agregan artificialmente cinco variantes lenguaje+Gnuplot.

28. **Ejercicio:** Gradiente sobre superficie y contornos

Considere

$$z(x,y) = x\,e^{-(x^2+y^2)}, \qquad -2 \le x, y \le 2.$$

Su gradiente es

$$\nabla z = \left((1-2x^2)e^{-(x^2+y^2)}, -2xy\,e^{-(x^2+y^2)}\right).$$

a) Evalúe z sobre una grilla de 50×50 puntos, conserve esos datos en un formato de grilla reutilizable y genere una superficie tridimensional.

b) Genere un mapa bidimensional de curvas de nivel de la misma función.

c) Sobre una subgrilla de aproximadamente 10×10 puntos, calcule el gradiente y superponga sus vectores al mapa de nivel.

d) Ajuste la escala gráfica de los vectores para que indiquen dirección y sentido sin ocultar las curvas de nivel.

e) Verifique visualmente que el gradiente sea aproximadamente perpendicular a las curvas de nivel y que apunte en la dirección de mayor crecimiento.

f) Compruebe que los archivos gráficos de salida esperados hayan sido creados correctamente.

Nota Gnuplot: Para datos de grilla pueden usarse bloques separados por líneas en blanco; el programa puede generar un script para producir la superficie y los contornos y luego invocar Gnuplot verificando los PNG.

Scilab. Una implementación del gradiente analítico sobre una subgrilla es:

```
x_v = linspace(-2, 2, 10)
y_v = linspace(-2, 2, 10)
fd  = mopen('/tmp/gradiente.txt', 'w')
for i = 1:length(x_v)
    for j = 1:length(y_v)
        xi = x_v(i); yi = y_v(j)
        dzdx = (1 - 2*xi^2) * exp(-(xi^2+yi^2))
        dzdy = -2*xi*yi     * exp(-(xi^2+yi^2))
        mfprintf(fd, '%.4f  %.4f  %.4f  %.4f\n', ..
                 xi, yi, dzdx*0.15, dzdy*0.15)
    end
end
mclose(fd)
```

Respuesta (L: diez modalidades).

Se utiliza la grilla 50×50 pedida y una subgrilla 10×10 para el campo vectorial. Para evitar que las flechas oculten los contornos se multiplican sus componentes por un factor gráfico; ese factor no modifica la dirección ni el sentido del gradiente.

Scilab, gráficos nativos

```
x=linspace(-2,2,50); y=x; Z=zeros(50,50)
for i=1:50, Z(i,:)=x(i)*exp(-(x(i)^2+y.^2)); end
q=linspace(-2,2,10); GX=zeros(10,10); GY=GX
for i=1:10
 GX(i,:)=(1-2*q(i)^2)*exp(-(q(i)^2+q.^2))
 GY(i,:)=-2*q(i)*q.*exp(-(q(i)^2+q.^2))
end
scf(280);clf();subplot(1,2,1);plot3d1(x,y,Z);title('superficie')
subplot(1,2,2);Sgrayplot(x,y,Z);contour2d(x,y,Z,15,frameflag=0)
a=gca();a.auto_clear='off';champ(q,q,.28*GX,.28*GY);a.auto_clear='on';ti
   + gradiente')
xs2png(gcf(),'ej28_scilab.png')
```

GNU Octave, gráficos nativos

```
x=linspace(-2,2,50);[X,Y]=meshgrid(x);Z=X.*exp(-(X.^2+Y.^2));
q=linspace(-2,2,10);[QX,QY]=meshgrid(q);GX=(1-2*QX.^2).*exp(-(QX.^2+QY.^
figure(280);subplot(1,2,1);surf(X,Y,Z);shading interp;grid
   on;
subplot(1,2,2);contourf(X,Y,Z,15);hold
   on;quiver(QX,QY,GX,GY,.55,'k');hold off;axis equal
   tight;colorbar;
print('ej28_octave.png','-dpng','-r140');
```

Python/Matplotlib

```
import numpy as np, matplotlib.pyplot as plt
x=np.linspace(-2,2,50);X,Y=np.meshgrid(x,x);Z=X*np.exp(-(X*X+Y*Y))
q=np.linspace(-2,2,10);QX,QY=np.meshgrid(q,q);E=np.exp(-(QX*QX+QY*QY));G
fig=plt.figure(figsize=(10,4));a=fig.add_subplot(121,projection='3d');a.
b=fig.add_subplot(122);c=b.contour(X,Y,Z,15);b.quiver(QX,QY,GX,GY,angles
```

Perl/PDL, gráficos nativos. Se usa TriD para la superficie y PGPLOT para contornos y vectores.

```
use strict;use warnings;use PDL;use PDL::Graphics::TriD;use
   PDL::IO::Pic qw(wpic);use PDL::Graphics::PGPLOT;
my$x=sequence(50)*4/49-2;my$X=$x->dummy(1,50);my$Y=$x->dummy(0,50);my$Z=
imag3d
   [$Z],{Lines=>1};wpic(grabpic3d(),'ej28_perl_superficie.png');
my$q=sequence(10)*4/9-2;my$QX=$q->dummy(1,10);my$QY=$q->dummy(0,10);my$E
dev('ej28_perl_contorno.png/png');imag $Z;hold;cont $Z;vect
   .28*$GX,.28*$GY;release;
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const
   fs=require('node:fs'),P=fs.readFileSync(require.resolve('plotly.js-di
const
   seq=(a,b,n)=>Array.from({length:n},(_,i)=>a+(b-a)*i/(n-1)),x=seq(-2,2
```

```
const q=seq(-2,2,10),vec=[];for(const y of q)for(const x of
   q){let
   e=Math.exp(-(x*x+y*y)),u=(1-2*x*x)*e,v=-2*x*y*e;vec.push({type:'scatt
const html=`<!doctype html><script>${P}</script><div
   id="s"></div><div
   id="c"></div><script>Plotly.newPlot('s',[${JSON.stringify({type:'surf
   +
   gradiente',xaxis:{scaleanchor:'y'}});</script>`;fs.writeFileSync('ej2
```

Gnuplot común a las cinco variantes lenguaje+Gnuplot

```
set terminal pngcairo size 800,600
set output 'ej28_superficie.png';set hidden3d;set pm3d
   depthorder;set view 55,35
splot 'ej28_grid.dat' u 1:2:3 w pm3d notitle
unset output
set output 'ej28_contorno_gradiente.png';set view map;set
   size ratio -1;set pm3d map;set contour base;set cntrparam
   levels 15
splot 'ej28_grid.dat' u 1:2:3 w pm3d notitle, \
      'ej28_grad.dat' u 1:2:(0):($3*.28):($4*.28):(0) w
         vectors head filled lc 'black' notitle
unset output
```

Los cinco programas siguientes producen exactamente los dos archivos de datos anteriores y luego ejecutan `ej28.gnu`.

Scilab + Gnuplot

```
x=linspace(-2,2,50);fd=mopen('ej28_grid.dat','w');for
   i=1:50,z=x(i)*exp(-(x(i)^2+x.^2));M=[x(i)*ones(50,1) x(:)
   z(:)];mfprintf(fd,'%.10g %.10g
   %.10g\n',M');mfprintf(fd,'\n');end;mclose(fd)
q=linspace(-2,2,10);fd=mopen('ej28_grad.dat','w');for
   i=1:10,for
   j=1:10,X=q(i);Y=q(j);e=exp(-(X^2+Y^2));mfprintf(fd,'%.10g
   %.10g %.10g
   %.10g\n',X,Y,(1-2*X^2)*e,-2*X*Y*e);end;end;mclose(fd);[st,o,er]=host(
   ej28.gnu');if st<>0 then error(er),end
```

GNU Octave + Gnuplot

```
x=linspace(-2,2,50);fid=fopen('ej28_grid.dat','w');for
   X=x,z=X*exp(-(X^2+x.^2));fprintf(fid,'%.10g %.10g
   %.10g\n',[X*ones(size(x));x;z]);fprintf(fid,'\n');end;fclose(fid);q=l
   X=q,for Y=q,e=exp(-(X^2+Y^2));fprintf(fid,'%.10g %.10g
   %.10g
   %.10g\n',X,Y,(1-2*X^2)*e,-2*X*Y*e);end;end;fclose(fid);system('gnuplo
   ej28.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
```

```
x=np.linspace(-2,2,50)
with open('ej28_grid.dat','w') as f:
 for X in x:
  for Y in x:f.write(f'{X} {Y} {X*np.exp(-(X*X+Y*Y))}\n')
  f.write('\n')
q=np.linspace(-2,2,10)
with open('ej28_grad.dat','w') as f:
 for X in q:
  for Y in q:
   e=np.exp(-(X*X+Y*Y));f.write(f'{X} {Y} {(1-2*X*X)*e}
      {-2*X*Y*e}\n')
subprocess.run(['gnuplot','ej28.gnu'],check=True)
```

Perl + Gnuplot

```
use strict;use warnings;open my$g,'>','ej28_grid.dat' or
   die$!;for my$i(0..49){my$x=-2+4*$i/49;for
   my$j(0..49){my$y=-2+4*$j/49;print$g"$x $y ",
   $x*exp(-($x*$x+$y*$y)),"\n"}print$g"\n"}close$g;open
   my$v,'>','ej28_grad.dat' or die$!;for
   my$i(0..9){my$x=-2+4*$i/9;for
   my$j(0..9){my$y=-2+4*$j/9;my$e=exp(-($x*$x+$y*$y));print$v"$x
   $y ",(1-2*$x*$x)*$e,"
   ",-2*$x*$y*$e,"\n"}}close$v;system('gnuplot','ej28.gnu')==0
   or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');let
   g='',v='';for(let i=0;i<50;i++){let x=-2+4*i/49;for(let
   j=0;j<50;j++){let y=-2+4*j/49;g+=`${x} ${y}
   ${x*Math.exp(-(x*x+y*y))}\n`}g+='\n'}for(let
   i=0;i<10;i++){let x=-2+4*i/9;for(let j=0;j<10;j++){let
   y=-2+4*j/9,e=Math.exp(-(x*x+y*y));v+=`${x} ${y}
   ${(1-2*x*x)*e}
   ${-2*x*y*e}\n`}}fs.writeFileSync('ej28_grid.dat',g);fs.writeFileSync(
   r=spawnSync('gnuplot',['ej28.gnu'],{encoding:'utf8'});if(r.status!==0
   new Error(r.stderr);
```

En el mapa de nivel, el gradiente corta aproximadamente en ángulo recto las curvas $z = c$ y apunta hacia valores crecientes de z. Los dos PNG deben existir y tener tamaño no nulo al terminar.

29. **Ejercicio:** Comparación de una grilla en tres representaciones

    Construya una grilla regular para

$$z = \sin(x)\cos(y), \qquad -\pi \le x, y \le \pi,$$

    con al menos 80×80 puntos. A partir de la misma grilla numérica:

a) genere una superficie tridimensional;

b) genere un mapa de curvas de nivel;

c) genere un mapa de color o imagen bidimensional de la matriz Z;

d) use la misma escala de colores en las representaciones bidimensionales cuando corresponda;

e) compare qué información se aprecia mejor en cada representación;

f) reduzca la grilla a 20×20 puntos y explique el efecto.

Nota Gnuplot: Realice una de las implementaciones con Gnuplot.

Scilab. Para la representación directa pueden usarse `Matplot` o `Sgrayplot`; mantenga consistente la orientación de la matriz respecto de los ejes.

Octave. Para el mapa matricial puede usar `imagesc` o `pcolor`.

Python. Para el mapa matricial use `imshow` o `pcolormesh` y defina explícitamente la extensión espacial si usa `imshow`.

Perl. Con PDL/PLplot puede utilizar una representación sombreada de la matriz, por ejemplo mediante `shadeplot`.

JavaScript. En Plotly.js represente la matriz con una traza `heatmap`.

Respuesta (T: diez modalidades).

Se calcula una única matriz $Z = \sin X \cos Y$ y se reutiliza sin recalcularla para la superficie, los contornos y el mapa de color. En las dos representaciones bidimensionales se fija la misma escala $[-1, 1]$. Luego se repite con 20×20 para evidenciar la pérdida de resolución.

Scilab, gráficos nativos

```
function Z=mk(n),x=linspace(-%pi,%pi,n);Z=zeros(n,n);for
   i=1:n,Z(i,:)=sin(x(i))*cos(x);end,endfunction
x=linspace(-%pi,%pi,80);Z=mk(80);scf(290);clf();subplot(1,3,1);plot3d1(x
x=linspace(-%pi,%pi,20);Z20=mk(20);scf(291);Sgrayplot(x,x,Z20);colorbar(
```

GNU Octave, gráficos nativos

```
x=linspace(-pi,pi,80);[X,Y]=meshgrid(x);Z=sin(X).*cos(Y);figure;subplot(
   interp;caxis([-1
   1]);subplot(1,3,2);contour(X,Y,Z,15);axis equal
   tight;subplot(1,3,3);imagesc(x,x,Z);axis xy equal
   tight;caxis([-1
   1]);colorbar;print('ej29_octave_80.png','-dpng');x=linspace(-pi,pi,20
   xy equal tight;caxis([-1
   1]);print('ej29_octave_20.png','-dpng');
```

Python/Matplotlib

```
import numpy as np,matplotlib.pyplot as plt
x=np.linspace(-np.pi,np.pi,80);X,Y=np.meshgrid(x,x);Z=np.sin(X)*np.cos(Y
x=np.linspace(-np.pi,np.pi,20);X,Y=np.meshgrid(x,x);Z=np.sin(X)*np.cos(Y
```

Perl/PDL, gráficos nativos

```
use strict;use warnings;use PDL;use PDL::Graphics::TriD;use
   PDL::IO::Pic qw(wpic);use PDL::Graphics::Simple
   qw(pgswin);
sub
   grid{my$n=shift;my$x=sequence($n)*(2*3.141592653589793)/($n-1)-3.1415
my$Z=grid(80);imag3d
   [$Z],{Lines=>1};wpic(grabpic3d(),'ej29_perl_superficie.png');my$w=pgs
```

JavaScript/Node.js, gráficos nativos con Plotly.js

```
const
   fs=require('node:fs'),P=fs.readFileSync(require.resolve('plotly.js-di
   x=seq(n);return{x,z:x.map(y=>x.map(X=>Math.sin(X)*Math.cos(y)))};let
   A=mk(80),B=mk(20),tr=[{type:'surface',x:A.x,y:A.x,z:A.z,cmin:-1,cmax:
   html><script>${P}</script>${tr.map((_,i)=>'<div
   id="p'+i+'"></div>').join('')}<script>let
   T=${JSON.stringify(tr)};T.forEach((v,i)=>Plotly.newPlot('p'+i,[v],{xa
```

Gnuplot común a las cinco variantes lenguaje+Gnuplot

```
set terminal pngcairo size 800,600;set cbrange [-1:1]
set output 'ej29_superficie.png';set view 55,35;set pm3d
   depthorder;splot 'ej29_80.dat' u 1:2:3 w pm3d
   notitle;unset output
set output 'ej29_contorno.png';set view map;unset pm3d;unset
   surface;set contour base;set cntrparam levels 15;set size
   ratio -1;splot 'ej29_80.dat' u 1:2:3 w l notitle;unset
   output
set output 'ej29_mapa.png';set surface;unset contour;set
   view map;set pm3d map;set size ratio -1;splot
   'ej29_80.dat' u 1:2:3 w pm3d notitle;unset output
set output 'ej29_20.png';splot 'ej29_20.dat' u 1:2:3 w pm3d
   notitle;unset output
```

Scilab + Gnuplot

```
function
   wr(nom,n),x=linspace(-%pi,%pi,n);fd=mopen(nom,'w');for
   i=1:n,z=sin(x(i))*cos(x);M=[x(i)*ones(n,1) x(:)
   z(:)];mfprintf(fd,'%.10g %.10g
   %.10g\n',M');mfprintf(fd,'\n');end;mclose(fd),endfunction;wr('ej29_80
   ej29.gnu');if e<>0 then error(er),end
```

GNU Octave + Gnuplot

```
function
   wr(nom,n),x=linspace(-pi,pi,n);fid=fopen(nom,'w');for
   X=x,z=sin(X)*cos(x);fprintf(fid,'%.10g %.10g
   %.10g\n',[X*ones(size(x));x;z]);fprintf(fid,'\n');end;fclose(fid);end
   ej29.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
def wr(name,n):
 x=np.linspace(-np.pi,np.pi,n)
 with open(name,'w') as f:
  for X in x:
   for Y in x:f.write(f'{X} {Y} {np.sin(X)*np.cos(Y)}\n')
   f.write('\n')
wr('ej29_80.dat',80);wr('ej29_20.dat',20);subprocess.run(['gnuplot','ej2
```

Perl + Gnuplot

```
use strict;use warnings;my$pi=4*atan2(1,1);sub
   wr{my($n,$N)=@_;open my$f,'>',$n or
   die$!;for(0..$N-1){my$x=-$pi+2*$pi*$_/($N-1);for
   my$j(0..$N-1){my$y=-$pi+2*$pi*$j/($N-1);print$f"$x $y
   ",sin($x)*cos($y),"\n"}print$f"\n"}close$f}wr('ej29_80.dat',80);wr('e
   or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');funct
   wr(name,n){let s='';for(let i=0;i<n;i++){let
   x=-Math.PI+2*Math.PI*i/(n-1);for(let j=0;j<n;j++){let
   y=-Math.PI+2*Math.PI*j/(n-1);s+=`${x} ${y}
   ${Math.sin(x)*Math.cos(y)}\n`}s+='\n'}fs.writeFileSync(name,s)}wr('ej
   r=spawnSync('gnuplot',['ej29.gnu'],{encoding:'utf8'});if(r.status!==0
   new Error(r.stderr);
```

La superficie hace más evidente la altura y la forma tridimensional; los contornos permiten leer conjuntos $z = c$; el mapa de color facilita reconocer rápidamente regiones positivas y negativas. Con 20×20 la interpolación se vuelve visible y las fronteras aparecen más angulosas.

7.4.7. Sección 6: Visualización de modelos numéricos y animaciones

30. **Ejercicio:** Oscilador amortiguado y retrato de fase

Considere el oscilador amortiguado

$$\ddot{x} + 0{,}4\dot{x} + 4x = 0$$

con condiciones iniciales

$$x(0) = 1, \qquad \dot{x}(0) = 0.$$

Resuelva numéricamente el sistema en

$$0 \le t \le 15$$

utilizando al menos 300 instantes de salida. Denote

$$v(t) = \dot{x}(t).$$

a) Represente $x(t)$ y $v(t)$ en dos paneles verticales con el mismo eje temporal.

b) Genere en una figura separada el retrato de fase v frente a x.

c) Use igual escala en los ejes del retrato de fase cuando resulte apropiado y marque el origen.

d) Explique por qué la trayectoria del retrato de fase se aproxima al origen y relacione este comportamiento con la disipación de energía.

e) Guarde los valores (t, x, v) en un archivo tabular que pueda reutilizarse para generar nuevamente las figuras sin repetir la integración.

Nota Gnuplot: Si la figura se produce con Gnuplot, genere el script a partir del archivo tabular y produzca las dos series temporales y el retrato de fase.

Scilab. Una implementación del cálculo auxiliar es:

```
function dydt = f(t, y)
    dydt = [y(2); -0.4*y(2) - 4*y(1)]
endfunction

t   = linspace(0, 15, 300)
sol = ode([1; 0], 0, t, f)
x   = sol(1,:)
v   = sol(2,:)

fprintfMat('/tmp/oscilador.txt', [t(:) x(:) v(:)], '%12.4f')
```

Respuesta (L: diez modalidades).

Se transforma la ecuación en

$$\dot{x} = v, \qquad \dot{v} = -0{,}4v - 4x.$$

La trayectoria de fase forma una espiral que converge al origen porque el término $0{,}4\dot{x}$ disipa energía. En todas las variantes se guardan al menos 301 filas (t, x, v) para poder regenerar las figuras sin reintegrar.

Scilab, gráficos nativos

```
function dy=f(t,y),dy=[y(2);-.4*y(2)-4*y(1)],endfunction
t=linspace(0,15,301);S=ode([1;0],0,t,f);x=S(1,:);v=S(2,:);fprintfMat('ej
   x(:) v(:)],'%15.10g')
scf(300);clf();subplot(2,1,1);plot(t,x);xgrid();ylabel('x');subplot(2,1,
scf(301);plot(x,v);a=gca();a.isoview='on';a.auto_clear='off';xgrid();plo
```

GNU Octave, gráficos nativos

```
t=linspace(0,15,301);[t,S]=ode45(@(t,y)[y(2);-.4*y(2)-4*y(1)],t,[1;0]);x
   x v],'delimiter','
   ');figure;subplot(2,1,1);plot(t,x);grid
   on;ylabel('x');subplot(2,1,2);plot(t,v);grid
   on;xlabel('t');ylabel('v');print('ej30_octave_tiempo.png','-dpng');fi
   equal;grid on;hold on;plot(0,0,'ro');hold
   off;print('ej30_octave_fase.png','-dpng');
```

Python, gráficos nativos

```
import numpy as np,matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
t=np.linspace(0,15,301);s=solve_ivp(lambda
   t,y:(y[1],-.4*y[1]-4*y[0]),(0,15),(1,0),t_eval=t);x,v=s.y;np.savetxt(
   for a in
   ax];fig.savefig('ej30_python_tiempo.png',dpi=140);plt.close(fig);plt.
```

Perl, integración RK4 y SVG nativo

```
use strict;use warnings;sub
   F{my($x,$v)=@_;($v,-.4*$v-4*$x)}sub
   step{my($x,$v,$h)=@_;my($a,$b)=F($x,$v);my($c,$d)=F($x+$h*$a/2,$v+$h*
   my$d,'>','ej30_perl.dat' or die$!;for
   my$i(0..300){my$t=$i*$h;push@t,$t;push@x,$X;push@v,$V;print$d"$t
   $X $V\n";($X,$V)=step($X,$V,$h) if $i<300}close$d;open
   my$s,'>','ej30_perl_tiempo.svg' or die$!;print$s'<svg
   xmlns="http://www.w3.org/2000/svg" width="850"
   height="520"><rect width="100%" height="100%"
   fill="white"/><text x="20" y="20">x(t) y
   v(t)</text>';my(@p1,@p2);for(0..300){my$px=70+720*$t[$_]/15;push@p1,"
   fill="none" stroke="black" points="',join('
   ',@p1),'"/><polyline fill="none" stroke="blue"
   points="',join(' ',@p2),'"/></svg>';close$s;open
   $s,'>','ej30_perl_fase.svg' or
   die$!;my@p;for(0..300){push@p,(400+150*$x[$_]).','.(260-70*$v[$_])}pr
   xmlns="http://www.w3.org/2000/svg" width="800"
   height="520"><rect width="100%" height="100%"
   fill="white"/><polyline fill="none" stroke="black"
   points="',join(' ',@p),'"/><circle cx="400" cy="260"
   r="4" fill="red"/></svg>';close$s;
```

JavaScript/Node.js, RK4 y Plotly.js

```
const
   fs=require('node:fs'),P=fs.readFileSync(require.resolve('plotly.js-di
   F(x,v){return[v,-.4*v-4*x]}function
   st(x,v,h){let[a,b]=F(x,v),[c,d]=F(x+h*a/2,v+h*b/2),[e,f]=F(x+h*c/2,v+
   t=[],x=[],v=[],X=1,V=0,h=15/300,s='';for(let
   i=0;i<=300;i++){t.push(i*h);x.push(X);v.push(V);s+=`${i*h}
   ${X}
   ${V}\n`;if(i<300)[X,V]=st(X,V,h)}fs.writeFileSync('ej30_node.dat',s);
```

```
    A=[{x:t,y:x,mode:'lines',name:'x(t)',xaxis:'x',yaxis:'y'},{x:t,y:v,mo
    html><script>${P}</script><div id="a"></div><div
    id="b"></div><script>Plotly.newPlot('a',${JSON.stringify(A)},{grid:{r
```

Gnuplot común a las cinco variantes lenguaje+Gnuplot

```
set terminal pngcairo size 800,700;set output
   'ej30_tiempo.png';set multiplot layout 2,1;set grid
set ylabel 'x';plot 'ej30.dat' u 1:2 w l notitle
set xlabel 't';set ylabel 'v';plot 'ej30.dat' u 1:3 w l
   notitle
unset multiplot;unset output
set terminal pngcairo size 600,600;set output
   'ej30_fase.png';set size ratio -1;set xlabel 'x';set
   ylabel 'v';set grid
plot 'ej30.dat' u 2:3 w l t 'trayectoria', '+' u (0):(0) w p
   pt 7 t 'origen';unset output
```

Scilab + Gnuplot

```
function
   dy=f(t,y),dy=[y(2);-.4*y(2)-4*y(1)],endfunction;t=linspace(0,15,301);
   S(1,:)' S(2,:)'],'%15.10g');[e,o,er]=host('gnuplot
   ej30.gnu');if e<>0 then error(er),end
```

GNU Octave + Gnuplot

```
t=linspace(0,15,301);[t,S]=ode45(@(t,y)[y(2);-.4*y(2)-4*y(1)],t,[1;0]);d
   S],'delimiter',' ');system('gnuplot ej30.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
from scipy.integrate import solve_ivp
t=np.linspace(0,15,301);s=solve_ivp(lambda
   t,y:(y[1],-.4*y[1]-4*y[0]),(0,15),(1,0),t_eval=t);np.savetxt('ej30.da
```

Perl + Gnuplot

```
use strict;use warnings;sub
   F{my($x,$v)=@_;($v,-.4*$v-4*$x)}sub
   S{my($x,$v,$h)=@_;my($a,$b)=F($x,$v);my($c,$d)=F($x+$h*$a/2,$v+$h*$b/
   my$f,'>','ej30.dat' or
   die$!;my($x,$v)=(1,0);my$h=15/300;for(0..300){print$f
   $_*$h," $x $v\n";($x,$v)=S($x,$v,$h) if
   $_<300}close$f;system('gnuplot','ej30.gnu')==0 or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');funct
   F(x,v){return[v,-.4*v-4*x]}function
   S(x,v,h){let[a,b]=F(x,v),[c,d]=F(x+h*a/2,v+h*b/2),[e,f]=F(x+h*c/2,v+h
```

```
    x=1,v=0,h=15/300,s='';for(let i=0;i<=300;i++){s+=`${i*h}
    ${x}
    ${v}\n`;if(i<300)[x,v]=S(x,v,h)}fs.writeFileSync('ej30.dat',s);let
    r=spawnSync('gnuplot',['ej30.gnu'],{encoding:'utf8'});if(r.status!==0
    new Error(r.stderr);
```

31. **Ejercicio:** Animación de un péndulo simple

Considere un péndulo de longitud $L = 1$ m con amortiguamiento lineal, descripto por

$$\ddot{\theta} + 0{,}1\dot{\theta} + \frac{9{,}8}{L}\sin\theta = 0,$$

con

$$\theta(0) = 60^\circ, \qquad \dot{\theta}(0) = 0.$$

Integre el movimiento para

$$0 \le t \le 8 \text{ s}$$

y obtenga una secuencia de aproximadamente 160 estados. La posición de la masa puede calcularse mediante

$$x = L\sin\theta, \qquad y = -L\cos\theta.$$

a) Genere para cada instante una imagen que muestre el punto de suspensión, la cuerda y la masa.

b) Mantenga constantes los límites y la relación de aspecto de los ejes en todos los cuadros para evitar saltos visuales de escala.

c) Dibuje la masa actual con un marcador destacado y represente en un tono más tenue la trayectoria recorrida hasta ese instante.

d) Numere los archivos de los cuadros de manera que su orden temporal pueda reconstruirse automáticamente.

e) Combine la secuencia en una animación, por ejemplo GIF o un formato animado equivalente disponible en el entorno utilizado.

f) Explique qué diferencias aparecen entre visualizar una trayectoria completa en una figura estática y observarla como una animación temporal.

Nota Gnuplot: Puede generarse un script por cuadro y producir una secuencia de PNG.

Scilab. Una base para el cálculo y la generación de la secuencia es:

```
function dydt = fpend(t, y)
    dydt = [y(2); -9.8/1.0 * sin(y(1)) - 0.1*y(2)]
endfunction

t0 = 0
theta0 = 60*%pi/180
t = linspace(0, 8, 160)
sol = ode([theta0; 0], t0, t, fpend)
```

```
theta = sol(1,:)

px = sin(theta)
py = -cos(theta)

for i = 1:length(t)
    // escribir datos del trazo y de la masa actual
    // generar la especificacion grafica del cuadro
    // producir el archivo de imagen
end
```

Scilab. Para combinar los PNG, una instalación con ImageMagick puede invocarse desde Scilab, por ejemplo:

```
host('convert -delay 4 -loop 0 /tmp/pend_*.png
   /tmp/pendulo.gif')
// En ImageMagick 7 puede usarse magick en lugar de convert.
```

Respuesta (L: diez modalidades).

La integración produce 160 estados igualmente espaciados entre 0 y 8 s. En cada cuadro se mantienen los límites $[-1{,}2,1{,}2]\times[-1{,}2,0{,}2]$ y aspecto igual. La trayectoria completa informa la geometría recorrida, mientras que la animación conserva el orden temporal, la velocidad aparente y la amortiguación.

Scilab, gráficos nativos

```
function
   dy=f(t,y),dy=[y(2);-.1*y(2)-9.8*sin(y(1))],endfunction
t=linspace(0,8,160);S=ode([%pi/3;0],0,t,f);px=sin(S(1,:));py=-cos(S(1,:)
for k=1:160
 scf(310);clf();a=gca();a.data_bounds=[-1.2 -1.2;1.2
    .2];a.isoview='on';a.auto_clear='off'
 plot(px(1:k),py(1:k),'c-');plot([0 px(k)],[0
    py(k)],'k-');plot(px(k),py(k),'ro');a.auto_clear='on'
 xs2png(gcf(),msprintf('pend_scilab_%03d.png',k-1))
end
host('magick -delay 5 -loop 0 pend_scilab_*.png
   pendulo_scilab.gif')
```

GNU Octave, gráficos nativos

```
t=linspace(0,8,160);[t,S]=ode45(@(t,y)[y(2);-.1*y(2)-9.8*sin(y(1))],t,[p
   k=1:numel(t),clf;plot(px(1:k),py(1:k),'color',[.7 .7
   .7]);hold on;plot([0 px(k)],[0
   py(k)],'k-');plot(px(k),py(k),'ro','markerfacecolor','r');hold
   off;axis([-1.2 1.2 -1.2 .2]);axis
   equal;print(sprintf('pend_octave_%03d.png',k-1),'-dpng');end;system('
   -delay 5 -loop 0 pend_octave_*.png pendulo_octave.gif');
```

Python/Matplotlib + Pillow

```
import numpy as np,matplotlib.pyplot as plt
```

```
from scipy.integrate import solve_ivp
from PIL import Image
t=np.linspace(0,8,160);s=solve_ivp(lambda
   t,y:(y[1],-.1*y[1]-9.8*np.sin(y[0])),(0,8),(np.pi/3,0),t_eval=t);px=n
for k in range(160):
 fig,ax=plt.subplots(figsize=(4,4));ax.plot(px[:k+1],py[:k+1],alpha=.25)
frames[0].save('pendulo_python.gif',save_all=True,append_images=frames[1
```

Perl, GD + Image::Magick

```
use strict;use warnings;use GD;use Image::Magick;sub
   F{my($q,$w)=@_;($w,-.1*$w-9.8*sin($q))}sub
   S{my($q,$w,$h)=@_;my($a,$b)=F($q,$w);my($c,$d)=F($q+$h*$a/2,$w+$h*$b/
   if $_<159}for
   my$k(0..159){my$im=GD::Image->new(500,500);my$wh=$im->colorAllocate(2
   X{250+180*$_[0]}sub Y{80-180*$_[0]}for
   my$i(1..$k){$im->line(X($x[$i-1]),Y($y[$i-1]),X($x[$i]),Y($y[$i]),$gr
   my$f,'>',sprintf('pend_perl_%03d.png',$k) or
   die$!;binmode$f;print$f$im->png;close$f}my$anim=Image::Magick->new;fo
```

JavaScript/Node.js, canvas + gifencoder

```
const
   fs=require('node:fs'),{createCanvas}=require('canvas'),GIFEncoder=req
   F(q,w){return[w,-.1*w-9.8*Math.sin(q)]}function
   S(q,w,h){let[a,b]=F(q,w),[c,d]=F(q+h*a/2,w+h*b/2),[e,f]=F(q+h*c/2,w+h
   x=[],y=[],q=Math.PI/3,w=0,h=8/159;for(let
   i=0;i<160;i++){x.push(Math.sin(q));y.push(-Math.cos(q));if(i<159)[q,w
   enc=new
   GIFEncoder(500,500);enc.createReadStream().pipe(fs.createWriteStream(
   c=createCanvas(500,500),g=c.getContext('2d'),X=a=>250+180*a,Y=a=>80-1
   k=0;k<160;k++){g.fillStyle='white';g.fillRect(0,0,500,500);g.strokeSt
   i=0;i<=k;i++){if(i===0)g.moveTo(X(x[i]),Y(y[i]));else
   g.lineTo(X(x[i]),Y(y[i]))}g.stroke();g.strokeStyle='black';g.beginPat
```

Gnuplot común a las cinco variantes lenguaje+Gnuplot. El archivo `ej31.dat` contiene t, θ, x, y. El mismo proceso genera los 160 PNG numerados.

```
set terminal pngcairo size 500,500
set xrange [-1.2:1.2];set yrange [-1.2:.2];set size ratio
   -1;unset key
N=160
do for [i=0:N-1] {
  stats 'ej31.dat' every ::i::i using 3 nooutput; X=STATS_min
  stats 'ej31.dat' every ::i::i using 4 nooutput; Y=STATS_min
  set output sprintf('pend_gp_%03d.png',i)
  set arrow 1 from 0,0 to X,Y nohead lw 2 front
  plot 'ej31.dat' every ::0::i u 3:4 w l lc rgb '#bbbbbb', \
       '' every ::i::i u 3:4 w p pt 7 ps 2 lc rgb 'red'
  unset arrow 1
}
```

```
unset output
```

Scilab + Gnuplot

```
function
    dy=f(t,y),dy=[y(2);-.1*y(2)-9.8*sin(y(1))],endfunction;t=linspace(0,8
    S(1,:)' sin(S(1,:))'
    -cos(S(1,:))'],'%15.10g');host('gnuplot
    ej31_frames.gnu');host('magick -delay 5 -loop 0
    pend_gp_*.png pendulo_gp_scilab.gif')
```

GNU Octave + Gnuplot

```
t=linspace(0,8,160);[t,S]=ode45(@(t,y)[y(2);-.1*y(2)-9.8*sin(y(1))],t,[p
    S(:,1) sin(S(:,1)) -cos(S(:,1))],'delimiter','
    ');system('gnuplot ej31_frames.gnu');system('magick
    -delay 5 -loop 0 pend_gp_*.png pendulo_gp_octave.gif');
```

Python + Gnuplot

```
import numpy as np,subprocess
from scipy.integrate import solve_ivp
t=np.linspace(0,8,160);s=solve_ivp(lambda
    t,y:(y[1],-.1*y[1]-9.8*np.sin(y[0])),(0,8),(np.pi/3,0),t_eval=t);q=s.
    -delay 5 -loop 0 pend_gp_*.png
    pendulo_gp_python.gif',shell=True,check=True)
```

Perl + Gnuplot

```
use strict;use warnings;sub
    F{my($q,$w)=@_;($w,-.1*$w-9.8*sin($q))}sub
    S{my($q,$w,$h)=@_;my($a,$b)=F($q,$w);my($c,$d)=F($q+$h*$a/2,$w+$h*$b/
    my$f,'>','ej31.dat' or
    die$!;my($q,$w)=(4*atan2(1,1)/3,0);my$h=8/159;for(0..159){print$f
    $_*$h," $q ",sin($q),'
    ',-cos($q),"\n";($q,$w)=S($q,$w,$h) if
    $_<159}close$f;system('gnuplot','ej31_frames.gnu')==0 or
    die;system('magick -delay 5 -loop 0 pend_gp_*.png
    pendulo_gp_perl.gif')==0 or die;
```

JavaScript/Node.js + Gnuplot

```
const
    fs=require('node:fs'),{spawnSync}=require('node:child_process');funct
    F(q,w){return[w,-.1*w-9.8*Math.sin(q)]}function
    S(q,w,h){let[a,b]=F(q,w),[c,d]=F(q+h*a/2,w+h*b/2),[e,f]=F(q+h*c/2,w+h
    q=Math.PI/3,w=0,h=8/159,s='';for(let
    i=0;i<160;i++){s+=`${i*h} ${q} ${Math.sin(q)}
    ${-Math.cos(q)}\n`;if(i<159)[q,w]=S(q,w,h)}fs.writeFileSync('ej31.dat
    r=spawnSync('gnuplot',['ej31_frames.gnu']);if(r.status!==0)throw
    Error('gnuplot');r=spawnSync('bash',['-lc','magick -delay
    5 -loop 0 pend_gp_*.png
```

```
    pendulo_gp_node.gif']);if(r.status!==0)throw
    Error('magick');
```

Extensión interactiva: Péndulo simple animado Construya una simulación interactiva de un sistema físico completo: el péndulo simple con visualización animada:

```
// Parametros del sistema
L = 1.0          // longitud [m]
g = 9.8          // gravedad [m/s^2]
b = 0.1          // amortiguamiento

// Pedir condiciones iniciales con dialogo
params = input_dialog({'Angulo inicial [grados]:', ...
                       'Velocidad angular inicial
                          [rad/s]:', ...
                       'Amortiguamiento b [0-1]:'}, ...
                      'Parametros del pendulo', ...
                      {'30', '0', '0.05'})
if isempty(params) then return, end

theta0 = strtod(params(1)) * %pi / 180
omega0 = strtod(params(2))
b      = strtod(params(3))

// EDO del pendulo: d2theta/dt2 = -g/L*sin(theta) -
   b*omega
function dydt = f_pendulo(t, y)
    dydt = [y(2); -(g/L)*sin(y(1)) - b*y(2)]
endfunction

// Resolver
t  = linspace(0, 10, 500)
y0 = [theta0; omega0]
sol = ode(y0, 0, t, f_pendulo)
theta = sol(1,:)

// Posicion cartesiana de la masa
px = L * sin(theta)
py = -L * cos(theta)

// Crear figura de animacion
f = figure('Figure_name', 'Pendulo animado', ...
           'position', [100, 100, 900, 500])

// Grafico de animacion (izquierda)
subplot(1, 2, 1)
title('Animacion del pendulo')
a_anim = gca()
```

```
// Grafico de theta(t) (derecha)
subplot(1, 2, 2)
plot(t, theta * 180/%pi, 'b-', 'LineWidth', 1.5)
xlabel('Tiempo [s]')
ylabel('Angulo [grados]')
title('Angulo vs tiempo')
xgrid()
h_marca = plot(t(1), theta(1)*180/%pi, 'ro', ...
               'MarkerSize', 8)

// Animacion
for i = 1:5:length(t)
    sca(a_anim)
    cla()

    // Dibujar el soporte
    plot([0, 0], [0.05, -0.05], 'k-', 'LineWidth', 3)

    // Dibujar la cuerda
    plot([0, px(i)], [0, py(i)], 'k-', 'LineWidth', 2)

    // Dibujar la masa
    plot(px(i), py(i), 'ro', 'MarkerSize', 20, ...
         'MarkerFaceColor', 'red')

    // Trayectoria
    if i > 5 then
        plot(px(max(1,i-50):i), py(max(1,i-50):i), ...
             'r:', 'LineWidth', 1)
    end

    a_anim.data_bounds = [-1.2*L, -1.2*L; 1.2*L, 0.2*L]
    a_anim.isoview = 'on'
    xlabel(sprintf('t = %.2f s   theta = %.1f deg', ...
                   t(i), theta(i)*180/%pi))

    // Mover el marcador en el grafico temporal
    set(h_marca, 'xdata', t(i), ...
                 'ydata', theta(i)*180/%pi)

    sleep(20)
    drawnow()
end

// Mostrar energia
E_cin = 0.5 * (L^2) * sol(2,:).^2
E_pot = g * L * (1 - cos(theta))
E_tot = E_cin + E_pot

figure()
plot(t, E_cin, 'r-', 'LineWidth', 1.5)
```

```
    plot(t, E_pot, 'b-', 'LineWidth', 1.5)
    plot(t, E_tot, 'k-', 'LineWidth', 2)
    legend('E cinetica', 'E potencial', 'E total')
    xlabel('Tiempo [s]')
    ylabel('Energia [J]')
    title('Energia del pendulo')
    xgrid()
endfunction

pendulo_animado()
```

a) Ejecute la simulación con ángulo inicial pequeño (5) y verifique que el período se aproxima a $T = 2\pi\sqrt{L/g}$.

b) Pruebe con ángulo grande (90). ¿El período es el mismo? ¿Por qué?

c) Aumente el amortiguamiento. ¿Cómo se refleja en el gráfico de energía?

d) Agregue un botón "Pausar/Reanudar" a la interfaz.

e) Agregue al diálogo inicial la opción de elegir entre péndulo simple y péndulo doble (con una segunda masa en el extremo del primero).

32. **Ejercicio:** Animación mediante actualización de objetos

Considere

$$x(t) = \cos(t), \qquad y(t) = \sin(2t), \qquad 0 \le t \le 8\pi.$$

Use un sistema gráfico que permita conservar y modificar objetos ya creados:

a) dibuje la trayectoria completa como referencia tenue;

b) cree un marcador que represente la posición actual y conserve el objeto gráfico correspondiente;

c) anime el movimiento actualizando los datos o propiedades de ese mismo objeto en cada cuadro, sin borrar y reconstruir toda la figura;

d) mantenga fijos los límites y la relación de aspecto;

e) explique por qué actualizar un objeto existente es preferible a recrear toda la figura cuando la escena contiene muchos elementos.

Scilab. Conserve el handle de la entidad y modifique sus datos; use `drawnow` para actualizar la figura.

Octave. Conserve el handle devuelto por el objeto gráfico, actualice `xdata`/`ydata` mediante `set` y fuerce el refresco con `drawnow`.

Python. Conserve el objeto `Line2D` o el marcador y actualícelo con `set_data`; refresque la figura sin volver a crear los ejes.

Perl. Use un backend interactivo que permita conservar y modificar el objeto, por ejemplo PDL::Graphics::Prima.

JavaScript. Cree la figura una sola vez y actualice la traza con `Plotly.restyle` o `Plotly.animate`, manteniendo fijo el `layout`.

Respuesta (N: cinco variantes nativas).

Este ejercicio no se resuelve con cinco variantes Gnuplot por lotes porque la consigna exige conservar **el mismo objeto gráfico** y modificarlo. En las cinco soluciones siguientes la trayectoria de referencia se crea una vez y el marcador conserva su identidad durante toda la animación.

Scilab

```
t=linspace(0,8*%pi,500);x=cos(t);y=sin(2*t);scf(320);clf();plot(x,y);a=g
    -1.1;1.1
    1.1];a.auto_clear='off';plot(x(1),y(1),'ro');h=gce();a.auto_clear='on
    k=1:length(t),h.data=[x(k) y(k)];drawnow();end
```

GNU Octave

```
t=linspace(0,8*pi,500);x=cos(t);y=sin(2*t);plot(x,y,'color',[.75
    .75 .75]);hold
    on;h=plot(x(1),y(1),'ro','markerfacecolor','r');hold
    off;axis([-1.1 1.1 -1.1 1.1]);axis equal;for
    k=1:numel(t),set(h,'xdata',x(k),'ydata',y(k));drawnow;end
```

Python/Matplotlib

```
import numpy as np,matplotlib.pyplot as plt
t=np.linspace(0,8*np.pi,500);x=np.cos(t);y=np.sin(2*t);fig,ax=plt.subplo
for X,Y in zip(x,y):
    marker.set_data([X],[Y]);fig.canvas.draw_idle();fig.canvas.flush_even
plt.show()
```

Perl/Tk. El identificador devuelto por `createOval` es el objeto que se actualiza.

```
use strict;use warnings;use
    Tk;my$mw=MainWindow->new;my$c=$mw->Canvas(-width=>500,-height=>500)->
    tick{my$X=$p[2*$i],$Y=$p[2*$i+1];$c->coords($m,$X-6,$Y-6,$X+6,$Y+6);$
```

JavaScript/Node.js con Plotly.js. La traza 1 es la referencia y la traza 2 es el mismo marcador, actualizado con `Plotly.restyle`.

```
const
    fs=require('node:fs'),P=fs.readFileSync(require.resolve('plotly.js-di
    t=Array.from({length:500},(_,i)=>8*Math.PI*i/499),x=t.map(Math.cos),y
    html><script>${P}</script><div id="p"></div><script>let
    x=${JSON.stringify(x)},y=${JSON.stringify(y)},i=0;Plotly.newPlot('p',
```

Actualizar un objeto evita reconstruir ejes, leyendas, textos y curvas estáticas en cada cuadro. La ventaja crece cuanto más compleja es la escena.

33. **Ejercicio:** Ajuste interactivo de curvas experimentales

Diseñe e instrumente una interfaz completa para el ajuste interactivo de curvas experimentales:

```
// Generar datos experimentales de ejemplo
x_real = linspace(0, 10, 15)
params_reales = [2.5, 0.3, 1.0]  // A, b, C
y_real = params_reales(1) *
   exp(-params_reales(2)*x_real) ...
         + params_reales(3) + 0.1*randn(1,15)

f = figure('Figure_name', ...
           'Ajuste interactivo de curvas', ...
           'position', [50, 50, 950, 650])

// Panel de modelo
uicontrol('style', 'text', ...
          'string', 'Modelo: A*exp(-b*x) + C', ...
          'position', [10, 610, 250, 25], ...
          'fontsize', 12)

// Sliders para los parametros
uicontrol('style', 'text', 'string', 'A:', ...
          'position', [10, 575, 30, 20])
h_A = uicontrol('style', 'slider', ...
                'position', [45, 575, 200, 20], ...
                'min', 0.1, 'max', 5.0, 'value', 1.0, ...
                'callback', 'actualizar_ajuste()')
h_A_txt = uicontrol('style', 'text', 'tag', 'txt_A', ...
                    'string', '1.00', ...
                    'position', [255, 575, 50, 20])

uicontrol('style', 'text', 'string', 'b:', ...
          'position', [10, 540, 30, 20])
h_b = uicontrol('style', 'slider', ...
                'position', [45, 540, 200, 20], ...
                'min', 0.01, 'max', 1.0, 'value', 0.5,
                   ...
                'callback', 'actualizar_ajuste()')
h_b_txt = uicontrol('style', 'text', 'tag', 'txt_b', ...
                    'string', '0.50', ...
                    'position', [255, 540, 50, 20])

uicontrol('style', 'text', 'string', 'C:', ...
          'position', [10, 505, 30, 20])
h_C = uicontrol('style', 'slider', ...
                'position', [45, 505, 200, 20], ...
                'min', -2.0, 'max', 3.0, 'value', 0.0,
                   ...
                'callback', 'actualizar_ajuste()')
h_C_txt = uicontrol('style', 'text', 'tag', 'txt_C', ...
```

```
                        'string', '0.00', ...
                        'position', [255, 505, 50, 20])

    // Boton de ajuste automatico
    uicontrol('style', 'pushbutton', ...
              'string', 'Ajuste automatico', ...
              'position', [10, 460, 150, 35], ...
              'callback', 'ajuste_automatico()')

    // Etiqueta de error
    uicontrol('style', 'text', 'tag', 'txt_error', ...
              'string', 'RMS: ---', ...
              'position', [10, 420, 200, 30], ...
              'BackgroundColor', [1, 1, 0.8])

    // Guardar datos en el entorno de la GUI
    global x_datos y_datos
    x_datos = x_real
    y_datos = y_real

    // Grafico
    ax = newaxes()
    ax.axes_bounds = [0.33, 0.08, 0.65, 0.88]

    actualizar_ajuste()
endfunction

function actualizar_ajuste()
    global x_datos y_datos
    h = findobj('style', 'slider')
    A = h(3).value
    b = h(2).value
    C = h(1).value

    set(findobj('tag','txt_A'), 'string', sprintf('%.2f', A))
    set(findobj('tag','txt_b'), 'string', sprintf('%.3f', b))
    set(findobj('tag','txt_C'), 'string', sprintf('%.2f', C))

    x = linspace(0, 10, 200)
    y_mod = A * exp(-b*x) + C
    y_datos_mod = A * exp(-b*x_datos) + C
    rms = sqrt(mean((y_datos - y_datos_mod).^2))

    set(findobj('tag','txt_error'), ...
        'string', sprintf('RMS = %.4f', rms))

    sca(findobj('type','Axes'))
    cla()
    plot(x_datos, y_datos, 'ro', 'MarkerSize', 8)
    plot(x, y_mod, 'b-', 'LineWidth', 2)
    legend('Datos', 'Modelo')
```

```
    xlabel('x')
    ylabel('y')
    title(sprintf('A=%.2f  b=%.3f  C=%.2f  RMS=%.4f', ...
                  A, b, C, rms))
    xgrid()
endfunction
```

a) Ejecute la GUI y ajuste manualmente los tres parámetros hasta minimizar el RMS. ¿Puede recuperar los valores reales $A = 2{,}5$, $b = 0{,}3$, $C = 1{,}0$?

b) Instrumente `ajuste_automatico()` usando `lsqrsolve` o `fminsearch` para minimizar el RMS automáticamente.

c) Agregue la posibilidad de cargar datos desde un archivo CSV usando un diálogo de selección.

d) Agregue un menú desplegable para elegir entre tres modelos distintos: exponencial, lineal y gaussiana.

e) Explique la diferencia entre una interfaz GUI de escritorio como esta y una interfaz web. ¿Cuáles son las ventajas y desventajas de cada una?

7.4.8. Sección 7: Exportación, persistencia y reproducibilidad

34. **Ejercicio:** Reproducibilidad y formatos de salida

Use una misma figura científica con al menos dos curvas, título, ejes, unidades, leyenda y cuadrícula. Organice el trabajo de manera que queden separados:

$$\texttt{cálculo} \longrightarrow \texttt{datos} + \texttt{especificación gráfica} \longrightarrow \texttt{archivo de salida}.$$

a) guarde los datos numéricos en un archivo independiente y la especificación de la figura en un archivo o script independiente;

b) genere la misma figura en PNG, PDF y SVG sin modificar los datos;

c) identifique cuál de los archivos es raster y cuáles son vectoriales;

d) amplíe una región con líneas y texto y compare la pérdida o conservación de nitidez;

e) compare los tamaños de archivo y explique por qué el tamaño, por sí solo, no determina qué formato es más apropiado;

f) borre los tres archivos gráficos y regenérelos exclusivamente a partir de los datos y la especificación gráfica, sin repetir el cálculo original;

g) modifique sólo una propiedad gráfica y regenere las tres salidas sin modificar los datos;

h) entregue los datos, la especificación y los tres archivos obtenidos.

Para visualizar las salidas utilice herramientas apropiadas para cada formato.

Nota Gnuplot: Para producir las tres salidas pueden utilizarse terminales como `pngcairo`, `pdfcairo` y `svg`. Si el cálculo se realiza en otro lenguaje, escriba primero

los datos y ejecute después el script gráfico; la regeneración de las figuras no debe exigir repetir el cálculo.

Scilab. Si el cálculo se realiza en Scilab, escriba primero el archivo de datos y ejecute después el script gráfico mediante `host`.

Octave. Separe de la misma forma el cálculo de los datos y la generación de la figura; para la vía directa, `print` permite distintos dispositivos de salida.

Python. Separe el cálculo de la especificación gráfica; en la vía directa, `savefig` permite generar formatos raster y vectoriales a partir de la misma figura.

Perl. Mantenga los datos independientes de la especificación gráfica y seleccione el dispositivo o formato de salida del backend sin recalcular los datos.

JavaScript. Conserve los datos y la configuración de la figura como origen reproducible; Plotly.js puede exportar PNG y SVG y el HTML puede conservar la figura interactiva.

Respuesta (T: diez modalidades).

Se adopta una figura con dos señales sobre $0 \le t \le 10$:

$$y_1(t) = e^{-t/4}\sin(2t), \qquad y_2(t) = e^{-t/4}\cos(2t).$$

El cálculo sólo crea `ej33.dat`. La especificación gráfica se mantiene en un archivo separado y puede volver a ejecutarse después de borrar PNG, PDF y SVG. PNG es raster; PDF y SVG son vectoriales. Al ampliar texto y líneas, los formatos vectoriales conservan la nitidez geométrica, mientras el PNG termina mostrando su resolución de píxeles. El tamaño de archivo depende de la complejidad y la codificación, no sólo de que el formato sea raster o vectorial.

Scilab, gráficos nativos

```
// calculo.sce
 t=linspace(0,10,501);e=exp(-t/4);fprintfMat('ej33.dat',[t(:)
    (e.*sin(2*t))' (e.*cos(2*t))'],'%15.10g')
// grafica.sce: puede ejecutarse sin repetir el bloque
   anterior
D=fscanfMat('ej33.dat');scf(330);clf();plot(D(:,1),D(:,2),'b-',D(:,1),D(
   amortiguadas');xlabel('t [s]');ylabel('amplitud
   [u]');legend('seno','coseno');xs2png(gcf(),'ej33_scilab.png');xs2pdf(
```

GNU Octave, gráficos nativos

```
% calculo.m
t=linspace(0,10,501)';e=exp(-t/4);dlmwrite('ej33.dat',[t
   e.*sin(2*t) e.*cos(2*t)],'delimiter',' ');
% grafica.m
D=load('ej33.dat');plot(D(:,1),D(:,2),'b-',D(:,1),D(:,3),'r--');grid
   on;xlabel('t [s]');ylabel('amplitud
   [u]');title('Oscilaciones
   amortiguadas');legend('seno','coseno');print('ej33_octave.png','-dpng
```

Python/Matplotlib

```
# calculo.py
import numpy as np
t=np.linspace(0,10,501);e=np.exp(-t/4);np.savetxt('ej33.dat',np.c_[t,e*r
# grafica.py: no recalcula los datos
import numpy as np,matplotlib.pyplot as plt
D=np.loadtxt('ej33.dat');plt.plot(D[:,0],D[:,1],label='seno');plt.plot(D
   [s]');plt.ylabel('amplitud [u]');plt.title('Oscilaciones
   amortiguadas');plt.legend()
for ext in
   ('png','pdf','svg'):plt.savefig('ej33_python.'+ext,dpi=150)
plt.close()
```

Perl/Cairo, gráficos nativos. La rutina `draw` es la especificación gráfica y se aplica sin cambios a tres superficies Cairo diferentes.

```
use strict;use warnings;use Cairo;open my$f,'<','ej33.dat'
   or die$!;my@d=map{[split]}<$f>;close$f;sub
   draw{my($c)=@_;$c->set_source_rgb(1,1,1);$c->paint;$c->set_source_rgb
   my$k(0..10){my$X=60+68*$k;$c->move_to($X,50);$c->line_to($X,450)}for
   my$k(-4..4){my$Y=250-45*$k;$c->move_to(60,$Y);$c->line_to(740,$Y)}$c-
   amortiguadas');$c->move_to(370,485);$c->show_text('t
   [s]');$c->move_to(5,250);$c->show_text('amplitud
   [u]');for
   my$j(1,2){$c->set_source_rgb($j==1?0:1,0,$j==1?1:0);my$i=0;for(@d){my
   / coseno')}
my$s=Cairo::ImageSurface->create('argb32',800,500);my$c=Cairo::Context->
   my$p(['pdf','ej33_perl.pdf'],['svg','ej33_perl.svg']){my$s=$p->[0]
   eq
   'pdf'?Cairo::PdfSurface->create($p->[1],800,500):Cairo::SvgSurface->c
```

JavaScript/Node.js con node-canvas

```
const
   fs=require('node:fs'),{createCanvas}=require('canvas'),D=fs.readFileS
   draw(g){g.fillStyle='white';g.fillRect(0,0,800,500);g.strokeStyle='#d
   k=0;k<=10;k++){let
   X=60+68*k;g.beginPath();g.moveTo(X,50);g.lineTo(X,450);g.stroke()}for
   k=-4;k<=4;k++){let
   Y=250-45*k;g.beginPath();g.moveTo(60,Y);g.lineTo(740,Y);g.stroke()}g.
   sans-serif';g.fillText('Oscilaciones
   amortiguadas',60,28);g.fillText('t
   [s]',380,485);g.fillText('amplitud [u]',5,250);for(let
   j=1;j<=2;j++){g.strokeStyle=j===1?'blue':'red';g.beginPath();D.forEac
   X=60+68*r[0],Y=250-180*r[j];i?g.lineTo(X,Y):g.moveTo(X,Y)});g.stroke(
   / coseno',640,60)}for(const [ext,type] of
   [['png',undefined],['pdf','pdf'],['svg','svg']]){let
   c=createCanvas(800,500,type),g=c.getContext('2d');draw(g);fs.writeFil
```

Gnuplot común a las cinco variantes lenguaje+Gnuplot. El archivo `ej33.gnu` es la especificación independiente. Cambiar, por ejemplo, sólo `TITULO` y volver a ejecutarlo no modifica los datos.

```
if (!exists('TITULO')) TITULO='Oscilaciones amortiguadas'
set title TITULO;set xlabel 't [s]';set ylabel 'amplitud
   [u]';set grid;set key top right
set terminal pngcairo size 800,500;set output
   'ej33.png';plot 'ej33.dat' u 1:2 w l lw 2 t 'seno','' u
   1:3 w l dt 2 lw 2 t 'coseno';unset output
set terminal pdfcairo size 8in,5in;set output
   'ej33.pdf';replot;unset output
set terminal svg size 800,500;set output
   'ej33.svg';replot;unset output
```

Scilab + Gnuplot

```
t=linspace(0,10,501);e=exp(-t/4);fprintfMat('ej33.dat',[t(:)
   (e.*sin(2*t))'
   (e.*cos(2*t))'],'%15.10g');[st,o,er]=host('gnuplot
   ej33.gnu');if st<>0 then error(er),end
```

GNU Octave + Gnuplot

```
t=linspace(0,10,501)';e=exp(-t/4);dlmwrite('ej33.dat',[t
   e.*sin(2*t) e.*cos(2*t)],'delimiter',' ');system('gnuplot
   ej33.gnu');
```

Python + Gnuplot

```
import numpy as np,subprocess
t=np.linspace(0,10,501);e=np.exp(-t/4);np.savetxt('ej33.dat',np.c_[t,e*n
```

Perl + Gnuplot

```
use strict;use warnings;open my$f,'>','ej33.dat' or
   die$!;for(0..500){my$t=10*$_/500,my$e=exp(-$t/4);print$f"$t
   ",$e*sin(2*$t),'
   ',$e*cos(2*$t),"\n"}close$f;system('gnuplot','ej33.gnu')==0
   or die;
```

JavaScript/Node.js + Gnuplot

```
const
   fs=require('node:fs'),{spawnSync}=require('node:child_process');let
   s='';for(let i=0;i<=500;i++){let
   t=10*i/500,e=Math.exp(-t/4);s+=`${t} ${e*Math.sin(2*t)}
   ${e*Math.cos(2*t)}\n`}fs.writeFileSync('ej33.dat',s);let
   r=spawnSync('gnuplot',['ej33.gnu'],{encoding:'utf8'});if(r.status!==0
   new Error(r.stderr);
```

Después de comprobar las tres salidas se pueden borrar y regenerar sólo con `ej33.dat` y la especificación gráfica. Para comparar tamaños debe consultarse el tamaño real

de cada archivo; no existe una regla general que obligue al vectorial a ser siempre menor o mayor que el raster.

==========

Capítulo 8

GUI-Web

UNIDAD 8 Interfaces gráficas de usuarios (GUI) y eventos

a) Terminal y Web. Terminal como interfaz. Entrada, salida y error estándar. Paso de parámetros y argumentos.

Web. Modelo cliente-servidor. HTTP. State-less. Formularios web. Métodos GET y POST. Paso de parámetros entre cliente y servidor. Estado. Uso de diálogos predefinidos. CGI y servidores de aplicaciones. Integración de una interfaz web con programas de cálculo.

b) Nativa de diferentes programas. Ventanas, controles y widgets. Creación y manipulación de objetos gráficos. Callbacks y procesamiento de eventos. Paso de parámetros y mantenimiento del estado. Uso de diálogos predefinidos. Eventos para modificar parámetros de una simulación. Separación entre modelo, interfaz y representación gráfica.

c) Jupyter como idea. Documentos interactivos: código, texto y resultados. Widgets y callbacks. Eventos para simular fenómenos físicos.

8.1. Interfaces Gráficas de Usuario (GUI)

8.1.1. Introducción

Hasta este punto de la asignatura los programas desarrollados interactúan con el usuario mediante la consola de Scilab. Los datos de entrada se ingresan por teclado y los resultados se muestran en forma de texto o mediante gráficos.

Sin embargo, muchas aplicaciones científicas y de ingeniería requieren una forma de interacción más amigable e intuitiva. Para ello se utilizan las **Interfaces Gráficas de Usuario** o **GUI** (*Graphical User Interface*), que permiten ingresar datos mediante cuadros de texto, botones, listas desplegables, controles deslizantes y otros elementos gráficos.

Una interfaz gráfica facilita el uso de un programa por parte de personas que no necesariamente conocen el lenguaje de programación con el que fue desarrollado. El usuario interactúa con la aplicación utilizando elementos visuales, mientras que el programa ejecuta internamente los cálculos correspondientes.

8.1.2. Concepto de control, evento y callback

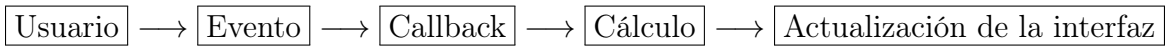
En una GUI hay **controles** que responden a determinadas Gamma et al. 1994 acciones realizadas por el usuario. Estas acciones desatan **eventos** en el sistema. Por ejemplo, presionar un botón, modificar el contenido de un cuadro de texto o mover un control deslizante desatan eventos que el programa puede detectar y procesar.

No se trata de una secuencia de código que se ejecuta, sino que el usuario decide qué código se ejecuta y cuándo.

La función que se ejecuta cuando ocurre un determinado evento se denomina **callback**. En otras palabras, un callback es el código asociado a un control gráfico que se ejecuta automáticamente cuando el usuario interactúa con dicho control.

Puede haber un callback único como en el caso web sin JavaScript (el que usamos preferentemente), en el que un solo callback maneja todos los eventos. O diferentes callbacks asociados a cada evento, como en el caso web con JavaScript o en el caso nativo de Scilab (el que tratamos, pero no usamos en forma predeterminada).

El funcionamiento general de una aplicación con interfaz gráfica puede resumirse mediante el siguiente esquema:



A diferencia de los programas desarrollados en los trabajos prácticos anteriores, donde las instrucciones se ejecutaban secuencialmente desde la primera hasta la última línea, las aplicaciones con interfaz gráfica están orientadas a eventos. El programa permanece esperando una acción del usuario y ejecuta únicamente el código asociado al evento ocurrido.

En este trabajo práctico se aprenderá a construir interfaces gráficas sencillas utilizando Scilab, eventualmente un servidor python, y Gnuplot, y a integrarlas con los programas desarrollados en los trabajos prácticos anteriores. El objetivo no es únicamente aprender a crear ventanas y botones, sino desarrollar aplicaciones que permitan resolver problemas científicos y de ingeniería de una manera más intuitiva para el usuario final.

8.1.3. Diálogos predefinidos

Los diálogos predefinidos son ventanas modales estándar Scilab Enterprises 2026c que el sistema operativo o el entorno de programación proporciona para las interacciones más comunes. Son más rápidos de usar que construir una GUI desde cero y producen interfaces más consistentes con el sistema.

Los tipos más comunes son:

Diálogo de mensaje: muestra información, advertencias o errores al usuario.

Diálogo de confirmación: pregunta sí/no o acepta/cancela.

Diálogo de entrada de texto: pide al usuario que escriba un valor.

Diálogo de selección de archivo: permite navegar el sistema de archivos para abrir o guardar un archivo.

Diálogo de selección de color: permite elegir un color de una paleta.

Diálogo de selección de fuente: permite elegir tipografía, tamaño y estilo.

```
// Catalogo de dialogos predefinidos en Scilab

// 1. Mensaje con icono
// Iconos disponibles: 'info', 'question', 'warning', 'error'
messagebox('La simulacion termino correctamente.', ...
           'Simulacion', 'info')

// 2. Confirmacion con botones personalizados
// Devuelve el indice del boton elegido (1, 2, ...)
opcion = messagebox('Guardar cambios antes de salir?', ...
                    'Confirmar salida', 'question', ...
                    ['Guardar' 'No guardar' 'Cancelar'])
select opcion
case 1 then disp('Guardando...')
case 2 then disp('Saliendo sin guardar')
case 3 then disp('Operacion cancelada')
end

// 3. Entrada de multiples valores
// Devuelve una lista de strings (o vacia si se cancela)
vals = input_dialog({'Radio [m]:'; ...
                     'Masa [kg]:'; ...
                     'Tiempo [s]:'}, ...
                    'Parametros de simulacion', ...
                    {'1.0'; '0.5'; '10.0'})  // valores por
                       defecto
if isempty(vals) then
    disp('El usuario cancelo')
else
    r = strtod(vals(1))
    m = strtod(vals(2))
    t = strtod(vals(3))
    printf('r=%.2f m, m=%.2f kg, t=%.1f s\n', r, m, t)
end

// 4. Seleccion de archivo para abrir
archivo = uigetfile({'*.csv', 'Datos CSV'; ...
                     '*.txt', 'Texto plano'; ...
                     '*.*',   'Todos'}, ...
                    pwd(), ...  // directorio inicial
                    'Seleccionar archivo de datos')
if archivo <> '' then
    printf('Archivo: %s\n', archivo)
end
```

```
// 5. Seleccion de archivo para guardar
destino = uiputfile({'*.pdf', 'PDF'; ...
                     '*.png', 'Imagen PNG'}, ...
                    'resultado', ...  // nombre por defecto
                    'Guardar grafico como')

// 6. Seleccion de color
// Devuelve [R, G, B] en [0,1]
color = uigetcolor([0, 0, 1], ...  // color inicial: azul
                   'Elegir color de la curva')
printf('Color elegido: R=%.2f G=%.2f B=%.2f\n', ...
       color(1), color(2), color(3))

// 7. Seleccion de directorio
directorio = uigetdir(pwd(), 'Seleccionar directorio de salida')
printf('Directorio: %s\n', directorio)

// 8. Lista de seleccion unica o multiple
opciones = ['Metodo de Euler'; ...
            'Runge-Kutta 4'; ...
            'Runge-Kutta-Fehlberg'; ...
            'Adams-Bashforth']
idx = listdialog('liststring', opciones, ...
                 'name', 'Seleccionar metodo numerico', ...
                 'selectionmode', 'single', ...
                 'initialvalue', 2)  // seleccionado por defecto
if ~isempty(idx) then
    printf('Metodo elegido: %s\n', opciones(idx))
end
```

8.1.4. Paso de parámetros en interfaces

El paso de parámetros es el mecanismo por el cual el Prasad et al. 2021; Preston-Werner, Gedam et al. 2021 usuario comunica los valores que el programa necesita para funcionar. Existe un espectro de alternativas según la complejidad y el uso previsto de la aplicación.

```
// NIVEL 1: parametros fijos en el codigo (sin interfaz)
// Solo sirve para uso personal y casos muy fijos
g = 9.8
L = 1.0
theta0 = 30 * %pi / 180
simular_pendulo(g, L, theta0)

// NIVEL 2: parametros en un archivo de configuracion
// Separa datos del codigo, reproducible, versionable
function p = leer_config(archivo)
    // Lee un archivo clave=valor
    p = struct('g', 9.8, 'L', 1.0, 'theta0', 30)
    if isfile(archivo) then
        lineas = mgetl(archivo)
```

```
        for i = 1:length(lineas)
            if length(lineas(i)) > 0 & lineas(i)(1) ~= '#' then
                partes = strsplit(lineas(i), '=')
                if length(partes) == 2 then
                    p(strtrim(partes(1))) =
                       strtod(strtrim(partes(2)))
                end
            end
        end
    end
endfunction

// NIVEL 3: parametros por linea de comandos (terminal)
// Maxima automatizacion y scriptabilidad
// En Scilab, usando argumentos de la funcion principal:
function main(varargin)
    // varargin permite numero variable de argumentos
    if nargin >= 1 then g = varargin(1); else g = 9.8; end
    if nargin >= 2 then L = varargin(2); else L = 1.0; end
    simular_pendulo(g, L)
endfunction

// NIVEL 4: dialogos interactivos
// Para uso esporadico o por usuarios no tecnicos
function params = pedir_parametros_dialogo()
    vals = input_dialog({'Gravedad [m/s^2]:', ...
                         'Longitud [m]:', ...
                         'Angulo inicial [grados]:', ...
                         'Amortiguamiento:'}, ...
                        'Parametros del pendulo', ...
                        {'9.8', '1.0', '30', '0.05'})
    if isempty(vals) then
        error('Simulacion cancelada por el usuario')
    end
    params.g      = strtod(vals(1))
    params.L      = strtod(vals(2))
    params.theta0 = strtod(vals(3)) * %pi / 180
    params.b      = strtod(vals(4))
endfunction

// NIVEL 5: controles GUI en tiempo real
// Para exploracion interactiva y visualizacion
// (sliders que actualizan el grafico inmediatamente)
```

La elección entre estos niveles depende del uso previsto: la línea de comandos y los archivos de configuración son ideales para producción y automatización; los diálogos y las GUI son preferibles para exploración interactiva y usuarios no especializados.

8.1.5. Interfaces de usuario

Una **interfaz de usuario** es el punto de contacto Norman 2013; Shneiderman et al. 2016 entre una persona y un sistema computacional. Define cómo el usuario comunica sus intenciones al sistema y cómo el sistema comunica sus respuestas al usuario. El diseño de la interfaz determina en gran medida la utilidad práctica de un programa: un algoritmo perfecto con una interfaz inutilizable es un programa inutilizable.

Las interfaces de usuario pueden clasificarse según el canal de comunicación que utilizan.

Cómo se evalúa una interfaz

Que una interfaz sea buena no es cuestión de gusto. Nielsen propuso diez heurísticas que funcionan como lista de control y que se aplican igual a una ventana, a un formulario web o a un programa de terminal Nielsen 1994: visibilidad del estado del sistema, correspondencia con el mundo real, control del usuario con una salida clara, consistencia, prevención de errores, reconocer antes que recordar, atajos para el usuario experto, diseño minimalista, mensajes de error comprensibles y ayuda accesible.

A esto se agrega la **accesibilidad**: la interfaz debe poder usarse con lector de pantalla, sólo con teclado y con contraste suficiente. En el caso del formulario web las reglas están normalizadas World Wide Web Consortium 2023 y lo mínimo es asociar cada campo con su etiqueta mediante `label for`, no transmitir información sólo por color, y mantener un orden de tabulación coherente.

La terminal como interfaz

La **interfaz de línea de comandos** (CLI, Prasad et al. 2021; Raymond 2003; The IEEE y The Open Group 2024; Ramey y Fox 2025 *Command Line Interface*) es la forma más antigua y más directa de interactuar con una computadora. El usuario escribe comandos en texto plano y el sistema responde con texto plano.

```
# La terminal es una interfaz de texto bidireccional
# Entrada: el usuario escribe
$ calcular_area 5.0

# Salida: el sistema responde
Area del circulo de radio 5.0 = 78.5398 m^2
```

A pesar de su aparente simplicidad, la terminal es una interfaz extraordinariamente poderosa porque:

- Es automatizable: los comandos pueden guardarse en scripts y ejecutarse sin intervención humana.
- Es componible: la salida de un programa puede ser la entrada de otro mediante tuberías.
- Es remota: funciona igual a través de SSH en un servidor en otra ciudad.
- Es reproducible: una secuencia de comandos documenta exactamente qué se hizo.

- Consume pocos recursos: no requiere entorno gráfico.

Los tres canales y el código de salida

Un programa de terminal no tiene una sola entrada y una sola salida, sino tres canales estándar, y esa separación es lo que hace posible la composición:

Entrada estándar (`stdin`, descriptor 0): por donde el programa lee datos, sea del teclado o de otro programa.

Salida estándar (`stdout`, descriptor 1): el resultado propiamente dicho, el que se encadena con una tubería.

Salida de error (`stderr`, descriptor 2): mensajes de diagnóstico y advertencias, que van a la pantalla aunque la salida esté redirigida.

Mezclar los dos últimos es el error más frecuente: si el programa escribe «Procesando archivo 3 de 10» en `stdout`, esa línea termina dentro del archivo de resultados y arruina la tubería.

```
# Los tres canales por separado
./simular < parametros.txt > resultados.dat 2> errores.log

# Encadenar: la salida de uno es la entrada del siguiente
./simular | grep 'convergio' | wc -l

# El codigo de salida indica exito (0) o fracaso (distinto de 0)
./simular parametros.txt
echo $?           # 0 si todo bien

# Y permite encadenar condicionalmente
./simular datos.txt && ./graficar resultados.dat
```

El **código de salida** es la forma en que un programa de terminal informa si tuvo éxito: cero significa que todo salió bien y cualquier otro valor indica un fallo. Es lo que permite que la shell decida si ejecuta o no el comando siguiente. Un programa que siempre devuelve cero es inutilizable dentro de un script.

Argumentos, opciones y convenciones

La forma de los argumentos no es arbitraria: hay convenciones que los usuarios esperan y que están normalizadas The IEEE y The Open Group 2024; Prasad et al. 2021.

Argumentos posicionales: los operandos sobre los que actúa el programa, como los nombres de archivo. Su significado depende de la posición.

Opciones cortas: una letra precedida de un guion, `-v`, agrupables como `-lah`.

Opciones largas: palabra precedida de dos guiones, `-verbose`, más legibles en un script.

Convenciones ineludibles: `-help` debe existir y describir el uso; `-version` debe imprimir la versión; un guion solo, `-`, suele significar «leer de la entrada estándar»; y `--` marca el fin de las opciones, para poder pasar un archivo que empieza con guion.

Una interfaz de terminal bien hecha cumple además otras dos reglas: si la salida no va a una terminal sino a un archivo o a una tubería, conviene omitir el color y las barras de progreso; y si el programa va a tardar, debe informar el avance por `stderr` y no quedarse mudo.

El paso de parámetros en la terminal se realiza mediante **argumentos de línea de comandos**:

```
# El programa recibe parametros como argumentos
./simular --temperatura 300 --presion 1.5 --tiempo 100

# O mediante variables de entorno
export TEMPERATURA=300
./simular

# O mediante archivos de configuracion
./simular --config parametros.yaml
```

```
// En Scilab, los argumentos pueden pasarse como variables
// globales o mediante un archivo de parametros

function params = leer_parametros(archivo)
    // Leer parametros desde archivo de texto
    lineas = mgetl(archivo)
    params = struct()
    for i = 1:length(lineas)
        if ~isempty(lineas(i)) & lineas(i)(1) ~= '#' then
            partes = strsplit(lineas(i), '=')
            if length(partes) == 2 then
                clave = strtrim(partes(1))
                valor = strtod(strtrim(partes(2)))
                params(clave) = valor
            end
        end
    end
endfunction

// Archivo parametros.txt:
// # Parametros de simulacion
// temperatura = 300
// presion = 1.5
// tiempo = 100

p = leer_parametros('parametros.txt')
printf('T=%.1f K, P=%.2f atm, t=%.0f s\n', ...
       p.temperatura, p.presion, p.tiempo)
```

Interfaz web

Revisar en tu02

Una **interfaz web** es una GUI que se presenta en un navegador. En lugar de un programa nativo instalado en la computadora, el usuario interactúa con páginas HTML que se muestran en el navegador.

Las interfaces web tienen ventajas importantes para aplicaciones científicas:

- No requieren instalación: el usuario solo necesita un navegador.
- Son multiplataforma: funcionan en cualquier sistema operativo y dispositivo.
- Facilitan la colaboración: múltiples usuarios pueden acceder al mismo sistema.
- Simplifican las actualizaciones: el servidor se actualiza una vez y todos los usuarios ven la versión nueva.

La arquitectura de una aplicación web tiene dos partes bien diferenciadas:

Frontend (cliente): el código que se ejecuta en el navegador del usuario. Está escrito en HTML, CSS y JavaScript. Se encarga de presentar la interfaz y capturar las interacciones del usuario.

Backend (servidor): el código que se ejecuta en el servidor. Puede estar escrito en cualquier lenguaje (Python, JavaScript/Node, Ruby, PHP, etc.). Se encarga de procesar los datos, acceder a la base de datos y generar las respuestas.

```
# Estructura tipica de una aplicacion web minima
mi_app/
    frontend/
        index.html       # estructura de la pagina
        estilo.css       # apariencia visual
        app.js           # comportamiento interactivo
    backend/
        servidor.py      # logica de negocio
        datos.db         # base de datos
    README.md
```

```
# Un servidor web minimo en Python (para desarrollo)
# servidor.py
python3 -m http.server 8080

# Acceder desde el navegador
http://localhost:8080
```

La comunicación entre el frontend y el backend se realiza mediante peticiones HTTP. El frontend envía una petición (por ejemplo, "dame los datos de temperatura de las últimas 24 horas"), el backend la procesa y devuelve una respuesta, generalmente en formato JSON.

```
# Formato JSON: intercambio de datos entre frontend y backend
# El servidor devuelve:
```

```
{
  "sensor": "temperatura_lab",
  "unidad": "celsius",
  "datos": [
    {"tiempo": "08:00", "valor": 22.3},
    {"tiempo": "09:00", "valor": 22.7},
    {"tiempo": "10:00", "valor": 23.1}
  ],
  "promedio": 22.7
}
```

La **interfaz web** usa el navegador como cliente universal: el programa no se instala en la máquina del usuario, se accede a él por una dirección. Eso cambia varias cosas respecto de una GUI de escritorio.

No hay instalación: cualquier máquina con navegador sirve, y la actualización es inmediata para todos porque el código vive en el servidor.

El cálculo ocurre en el servidor: la máquina del usuario sólo muestra y envía. Esto permite que una computadora modesta use un programa que necesita mucha memoria o mucho procesador.

La interacción es discontinua: entre una solicitud y la siguiente el servidor no recuerda nada, a diferencia de una GUI donde el programa está vivo todo el tiempo. Es la consecuencia directa de que HTTP no tenga estado.

Es multiusuario por naturaleza: varios usuarios pueden usar la misma instancia a la vez, lo que obliga a pensar la concurrencia y el aislamiento de los datos de cada uno.

En esta materia esa es la arquitectura del sistema que se presenta más adelante: un formulario HTML pide los parámetros, un servidor en Python recibe la solicitud, Scilab hace el cálculo y devuelve una página con los resultados y las gráficas.

Modelo cliente–servidor

El **modelo cliente–servidor** es una arquitectura R. T Fielding 2000 en la que las tareas se reparten entre dos roles:

Servidor: programa que espera solicitudes, las procesa y devuelve respuestas. Ofrece un servicio. Generalmente está siempre en ejecución.

Cliente: programa que envía solicitudes al servidor y usa los resultados. Inicia la comunicación. Puede conectarse, usar el servicio y desconectarse.

Esta división permite que muchos clientes compartan el mismo servidor, que el servidor se actualice sin afectar a los clientes, y que clientes en distintas plataformas accedan al mismo servicio.

```
# Ejemplos cotidianos del modelo cliente--servidor:

# Cliente web (navegador) TO  Servidor web (Apache, nginx)
```

```
# El navegador pide una pagina, el servidor la devuelve

# Cliente de correo (Thunderbird) TO Servidor SMTP/IMAP
# El cliente envia y recibe correos a traves del servidor

# Cliente SSH TO Servidor SSH (sshd)
ssh usuario@servidor.unsa.edu.ar
# El cliente envia comandos, el servidor los ejecuta
# y devuelve la salida

# En Scilab: cliente (me.py) TO servidor (mes.py)
# El cliente pide formularios, el servidor los sirve
```

La comunicación entre cliente y servidor sigue un protocolo: una especificación precisa del formato de las solicitudes y las respuestas.

```
# El protocolo HTTP es el ejemplo mas extendido:

# SOLICITUD del cliente:
GET /datos/temperatura HTTP/1.1
Host: servidor.lab.unsa.edu.ar
Accept: application/json

# RESPUESTA del servidor:
HTTP/1.1 200 OK
Content-Type: application/json

{"temperatura": 23.5, "unidad": "celsius"}
```

Protocolo sin estado (stateless)

HTTP, el protocolo de la web, es **sin estado** Roy T. Fielding et al. 2022; Gourley y Totty 2002 (*stateless*): cada solicitud es independiente y completa. El servidor no recuerda las solicitudes anteriores del mismo cliente.

Esto simplifica el diseño del servidor (no necesita mantener información sobre cada cliente conectado) pero plantea un problema: las aplicaciones web necesitan frecuentemente recordar al usuario entre páginas (carrito de compras, sesión iniciada, preferencias).

Ojo: el protocolo HTTP es stateless, pero nuestra implementación usa archivos temporales en `/tmp`. Mientras se utiliza `HTTPServer`, que atiende una solicitud por vez, los nombres fijos no se pisan entre dos cálculos simultáneos porque no hay ejecución concurrente. Si se reemplaza por un servidor con hilos o procesos, cada solicitud debe usar archivos temporales distintos o un identificador de sesión.

Obviamente, en una red abierta no se aconseja trabajar aún con este cambio porque un servidor así, sin identificación de usuario, puede ser demandado infinitamente sin posibilidad clara de defensa.

Las soluciones para mantener estado en un protocolo stateless son:

Cookies: pequeños archivos que el servidor pide al navegador que almacene y envíe en cada solicitud. Permiten identificar al usuario entre peticiones. El mecanismo está

definido en Barth 2011.

Sesiones: el servidor asigna un identificador único a cada usuario y lo almacena en una cookie. En el servidor se mantiene información asociada a ese identificador.

Tokens JWT (*JSON Web Token*): el servidor genera un token firmado que contiene información del usuario. El cliente lo envía en cada solicitud. El servidor puede verificar la autenticidad sin consultar una base de datos. El formato del token está normalizado M. B. Jones, Bradley y Sakimura 2015 y su contenido se codifica en JSON Bray 2017.

Parámetros en la URL: información de estado codificada en la URL. Simple pero visible e inseguro para datos sensibles.

```
# Ejemplo de cookie en HTTP:
# Primera solicitud: el usuario inicia sesion
POST /login HTTP/1.1
{"usuario": "garcia", "clave": "..."}

# El servidor responde asignando un ID de sesion:
HTTP/1.1 200 OK
Set-Cookie: sesion_id=abc123xyz; HttpOnly; Secure

# Solicitudes posteriores: el navegador envia la cookie
GET /datos HTTP/1.1
Cookie: sesion_id=abc123xyz
# El servidor identifica al usuario por el ID de sesion
```

```
// Simulacion del modelo stateless con archivos
// Cada "solicitud" es independiente y completa

function respuesta = solicitud_http(tipo, recurso, cuerpo)
    // Simula una solicitud HTTP
    // El servidor no recuerda solicitudes anteriores

    // Construir la solicitud completa (con todo el contexto)
    solicitud = struct()
    solicitud.tipo     = tipo       // GET, POST, PUT, DELETE
    solicitud.recurso  = recurso    // /temperatura, /formula
    solicitud.cuerpo   = cuerpo     // datos adicionales
    solicitud.tiempo   = date()

    // El servidor procesa cada solicitud de forma independiente
    respuesta = procesar_solicitud(solicitud)
endfunction

function resp = procesar_solicitud(sol)
    // El servidor no tiene memoria de solicitudes anteriores
    // Toda la informacion necesaria debe venir en la solicitud

    select sol.tipo + ' ' + sol.recurso
```

```
    case 'GET /temperatura' then
        resp.codigo = 200
        resp.datos  = struct('valor', 23.5, 'unidad', 'C')
    case 'POST /calcular' then
        resp.codigo = 200
        resp.datos  = struct('resultado', evstr(sol.cuerpo))
    else
        resp.codigo = 404
        resp.datos  = struct('error', 'Recurso no encontrado')
    end
endfunction
```

Diálogos predefinidos del navegador

El navegador dispone de diálogos modales predefinidos para interacciones simples. Los más comunes son `alert`, `confirm` y `prompt`. Estos diálogos interrumpen temporalmente la interacción con la página hasta que el usuario responde.

```
// Mensaje
alert("La simulacion termino")

// Confirmacion: devuelve true o false
if (confirm("Desea ejecutar nuevamente?")) {
    console.log("repetir")
}

// Entrada de texto: devuelve un string o null
let nombre = prompt("Nombre del experimento:", "prueba01")
```

En la arquitectura usada en este trabajo estos diálogos no son el mecanismo principal de entrada. Los parámetros se introducen mediante formularios HTML y se envían al servidor con una solicitud HTTP. Los diálogos predefinidos quedan para mensajes, confirmaciones o entradas puntuales del lado del navegador.

En una interfaz de terminal no existe un diálogo gráfico estándar: la interacción equivalente se realiza mediante argumentos, opciones, archivos de configuración o lectura desde la entrada estándar.

8.2. Formularios web y el protocolo CGI

8.2.1. Cómo funciona un formulario web

Un formulario HTML es un mecanismo que permite al usuario WHATWG 2026; Network 2026 enviar datos desde el navegador hacia un servidor. El proceso involucra dos partes bien definidas: la presentación del formulario (lado del cliente) y el procesamiento de los datos recibidos (lado del servidor).

Ciclo de vida de un formulario

El ciclo completo tiene tres pasos:

1. El navegador solicita la página con el formulario mediante `GET`. El servidor devuelve el HTML.
2. El usuario completa los campos y presiona el boton de envío. El navegador serializa los datos y los envía al servidor mediante `POST`.
3. El servidor procesa los datos y devuelve una nueva página HTML con el resultado.

Estructura del formulario HTML

Los atributos fundamentales de la etiqueta `form` son:

```
<form method="POST" action="/calcular">
  <!-- campos del formulario -->
  <button type="submit">Enviar</button>
</form>
```

`method`: define como se envían los datos. `GET` los incluye en la URL; `POST` los envía en el cuerpo de la solicitud HTTP. Para formularios con datos sensibles o volúmenes grandes se usa `POST`.

`action`: la URL a la que se envían los datos. Si se omite o se pone `/` se envía a la misma URL.

Tipos de controles del formulario

Control	HTML	Uso tipico
Texto de linea	`<input type=text>`	Nombre, path
Número	`<input type=number>`	Parámetros numéricos
Correo	`<input type=email>`	Correo electrónico
Color	`<input type=color>`	Selector de color
Fecha	`<input type=date>`	Fecha
Lista despleg.	`<select>`	Opciones exclusivas
Boton opción	`<input type=radio>`	Una opción del grupo
Casilla	`<input type=checkbox>`	Activar/desactivar
Area de texto	`<textarea>`	Texto largo
Oculto	`<input type=hidden>`	Datos sin visualizar

Serializacion de los datos: URL encoding

Cuando el navegador envía un formulario `POST`, serializa Berners-Lee, Roy T. Fielding y Masinter 2005; Masinter 2015 los campos en formato *URL encoding* (también llamado `application/x-www-form-urlencoded`):

```
# Formato del cuerpo del POST:
nombre=Garcia&v0=50&angulo=45&arr=bajo&met=rk4&g_tray=on

# Reglas de codificacion:
# - campos separados por &
# - clave y valor separados por =
# - espacios reemplazados por +  o  %20
# - caracteres especiales como %XX (hexadecimal)
# Por ejemplo: 'hola mundo' -> 'hola+mundo'
#              'a&b'        -> 'a%26b'
```

Los checkboxes tienen un comportamiento especial: si están marcados se envía `clave=on`; si no están marcados no se envía nada (la clave directamente no aparece en el cuerpo del POST). Por eso se usa un campo oculto con el mismo nombre y valor `off` antes de cada checkbox, de modo que siempre llegue algún valor al servidor.

Validación en el cliente

El navegador puede validar los datos antes de enviarlos, con los atributos del propio HTML —`required`, `min`, `max`, `pattern`, `type="number"`— o con código JavaScript asociado a los eventos del formulario Flanagan 2020. El modelo de eventos del navegador es el mismo que el de cualquier GUI: se registra un manejador para un tipo de evento sobre un elemento, y el navegador lo invoca cuando el evento ocurre.

Esa validación mejora la experiencia porque avisa sin ir y volver al servidor, pero **no reemplaza la validación en el servidor**: el cliente está bajo control del usuario y cualquiera puede enviar una solicitud sin pasar por el formulario.

La solicitud HTTP completa

Una solicitud POST se ve así a nivel de protocolo:

```
POST / HTTP/1.1
Host: localhost:8080
Content-Type: application/x-www-form-urlencoded
Content-Length: 52

nombre=Garcia&v0=50&angulo=45&arr=bajo&met=rk4
```

Y la respuesta del servidor:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 4821

<!DOCTYPE html>
<html>...resultado...</html>
```

8.2.2. El protocolo CGI

Qué es CGI

CGI (*Common Gateway Interface*) es el mecanismo D. Robinson y Coar 2004 original para que un servidor web ejecute programas externos y devuelva su salida como respuesta HTTP. Fue definido en 1993 y es el ancestro de todos los sistemas web dinámicos modernos.

La idea es simple: cuando el servidor recibe una solicitud para una URL especial (típicamente bajo `/cgi-bin/`), en lugar de devolver un archivo estático, ejecuta un programa, le pasa los datos de la solicitud y devuelve lo que el programa escribe en su salida estándar.

```
# Flujo CGI:
# 1. Navegador -> GET /cgi-bin/calcular.py?v0=50&ang=45
# 2. Servidor web ejecuta:  python3 /var/www/cgi-bin/calcular.py
# 3. El programa lee sus parametros de variables de entorno
# 4. El programa escribe el HTML en stdout
# 5. El servidor devuelve ese stdout al navegador
```

Variables de entorno CGI

El servidor web comunica los datos de la solicitud al programa CGI a través de variables de entorno. Las principales son:

Variable	Contenido
`REQUEST_METHOD`	GET o POST
`QUERY_STRING`	parámetros de la URL (en GET)
`CONTENT_LENGTH`	longitud del cuerpo (en POST)
`CONTENT_TYPE`	tipo de datos enviados
`HTTP_HOST`	nombre del servidor
`REMOTE_ADDR`	IP del cliente
`SCRIPT_NAME`	ruta del script CGI
`PATH_INFO`	información de ruta adicional

El cuerpo del POST se lee desde la entrada estándar (`stdin`) hasta `CONTENT_LENGTH` bytes.

Un programa CGI mínimo en Python

Listing 8.1: hola.py como script CGI

```
#!/usr/bin/env python3
import os

# El script CGI DEBE imprimir los headers primero
print("Content-Type: text/html; charset=UTF-8")
```

```
print("")    # linea en blanco obligatoria entre headers y cuerpo

# Leer el metodo
metodo = os.environ.get("REQUEST_METHOD", "GET")

# Leer parametros
if metodo == "GET":
    qs = os.environ.get("QUERY_STRING", "")
else:
    n  = int(os.environ.get("CONTENT_LENGTH", 0))
    qs = input()[:n]    # leer n bytes de stdin

# Parsear (simplificado)
params = {}
for par in qs.split("&"):
    if "=" in par:
        k, v = par.split("=", 1)
        params[k] = v

nombre = params.get("nombre", "mundo")

# Imprimir el HTML de respuesta
print(f"<!DOCTYPE html><html><body>")
print(f"<h1>Hola, {nombre}!</h1>")
print(f"<p>Metodo: {metodo}</p>")
print(f"</body></html>")
```

Para ejecutarlo como CGI con el servidor web Apache:

```
# Instalar apache
sudo apt install apache2

# Habilitar el modulo CGI
sudo a2enmod cgi

# Copiar el script
sudo cp hola.py /usr/lib/cgi-bin/
sudo chmod +x /usr/lib/cgi-bin/hola.py

# Probar
curl http://localhost/cgi-bin/hola.py?nombre=Diego
```

O con el servidor CGI incorporado en Python, sin necesitar Apache:

```
# Python incluye un servidor web con soporte CGI
python3 -m http.server --cgi 8080

# Los scripts deben estar en ./cgi-bin/
mkdir cgi-bin
cp hola.py cgi-bin/
chmod +x cgi-bin/hola.py
```

```
# Probar
curl "http://localhost:8080/cgi-bin/hola.py?nombre=Diego"
```

Un script CGI en Scilab

CGI no requiere Python ni ningun lenguaje particular. Cualquier programa ejecutable que lea de variables de entorno y escriba HTML en stdout puede ser un script CGI. Con Scilab:

Listing 8.2: calcular.sh envuelve Scilab como CGI

```
#!/bin/bash
# calcular.sh - wrapper CGI para Scilab

# Leer el cuerpo POST
read -n $CONTENT_LENGTH BODY

# Escribir los parametros en un archivo temporal
echo "$BODY" | python3 -c "
import sys
from urllib.parse import parse_qs, unquote_plus
body = sys.stdin.read()
for par in body.split('&'):
    if '=' in par:
        k, v = par.split('=', 1)
        k = unquote_plus(k)
        v = unquote_plus(v)
        try:
            float(v)
            print(f'{k} = {v};')
        except:
            print(f\"{k} = '{v}';\")
" > /tmp/params.sce

# Llamar a Scilab
scilab-cli -e "exec('/var/www/cgi-bin/calcular.sce', -1); quit" \
           2>/tmp/scilab_err.txt

# El header CGI va ANTES del HTML
echo "Content-Type: text/html; charset=UTF-8"
echo ""

# El resultado que genero Scilab
cat /tmp/resultado.html
```

Limitaciones del CGI clásico

CGI tiene una limitación fundamental de rendimiento: por cada Eby 2010 solicitud HTTP se crea un proceso nuevo del sistema operativo, se inicializa el intérprete (Python, Scilab, Perl, etc.), se ejecuta el programa y se destruye el proceso. Para aplicaciones con muchos usuarios simultáneos esto es muy costoso.

Las soluciones modernas son:

FastCGI: el proceso se mantiene vivo entre solicitudes, evitando el costo de inicialización. Se comunica con el servidor web por un socket en lugar de variables de entorno.

WSGI (Python): interfaz estándar entre servidores web y aplicaciones Python. Frameworks como Flask y Django usan WSGI.

Servidor propio: como nuestro `servidor.py`, que implementa HTTP directamente y llama a Scilab como subproceso.

8.2.3. Comparación de arquitecturas

Aspecto	CGI clásico	Nuestro servidor	Framework web
Por solicitud	Nuevo proceso	Nuevo proceso Scilab	En memoria
Rendimiento	Bajo	Bajo-medio	Alto
Simplicidad	Alta	Alta	Media
Dependencias	Solo el lenguaje	Python + Scilab	Framework
Separación	Ninguna	Python/Scilab/HTML	MVC
Estado	Sin estado	Sin estado	Sesiones

Para aplicaciones de cálculo científico de uso académico con pocos usuarios simultáneos, la arquitectura de nuestro `servidor.py` es adecuada y simple. La separación entre el servidor HTTP genérico, el formulario HTML editable libremente y el script de cálculo en Scilab refleja una separación de responsabilidades similar al patrón MVC (*Model-View-Controller*): Scilab es el modelo (cálculo), HTML es la vista (presentación) y Python es el controlador (enrutamiento).

8.2.4. Seguridad básica

Los formularios web exponen el servidor a entradas del OWASP Foundation 2021 usuario que pueden contener datos inesperados o maliciosos. Algunas precauciones minimas:

Validar los datos en el servidor: nunca confiar en que el cliente envió valores válidos. Verificar rangos, tipos y formatos antes de usarlos en el cálculo.

No ejecutar código del usuario: si el formulario permite ingresar texto libre y ese texto se usa en `execstr` o `eval`, un usuario malicioso podría ejecutar comandos arbitrarios en el servidor.

Sanitizar antes de insertar en HTML: si el valor ingresado por el usuario se incluye en el HTML de respuesta, caracteres como `<`, `>` y `&` deben reemplazarse por sus entidades HTML (`<`, `>`, `&`) para evitar inyeccion de HTML.

Validar según el lenguaje de destino: un dato puede ser válido como texto y, sin embargo, ser peligroso si después se inserta dentro de otro lenguaje. Por ejemplo, un color recibido desde un formulario puede terminar formando parte de un archivo de comandos de gnuplot. Antes de construir ese archivo debe comprobarse que el valor pertenece exactamente al formato admitido. La misma regla vale para texto que vaya a formar parte de una orden del shell, HTML, SQL, JavaScript u otro lenguaje: primero se valida el conjunto de valores permitidos y luego, cuando corresponda, se escapan o entrecomillan los caracteres según las reglas del lenguaje de destino.

Siempre que sea posible conviene aceptar un conjunto pequeño y explícito de valores válidos en lugar de intentar eliminar caracteres "peligrosos" de una cadena arbitraria. Así, para un color HTML puede aceptarse únicamente una cadena de la forma `#rrggbb`, donde cada componente es un dígito hexadecimal, antes de escribirla en el script de gnuplot.

Limitar el acceso: para uso interno, es suficiente con que el servidor escuche solo en `localhost` (no accesible desde la red). Para exponerlo en la red de la institución, agregar autenticación básica.

Listing 8.3: sanitizar HTML en Python

```
def sanitizar(texto):
    texto = texto.replace('&', '&amp;')
    texto = texto.replace('<', '&lt;')
    texto = texto.replace('>', '&gt;')
    texto = texto.replace('"', '&quot;')
    return texto

# usar antes de insertar texto del usuario en HTML
nombre_seguro = sanitizar(params.get('nombre', ''))
```

8.3. Sistema de formularios web con motores de cálculo externos

8.3.1. Introducción

Este sistema permite crear aplicaciones web de cálculo científico separando la interfaz web del programa que realiza el cálculo. El usuario completa un formulario en el navegador, los datos llegan a `servidor.py`, el servidor ejecuta el motor seleccionado y devuelve los resultados en la misma página, incluyendo gráficas cuando el motor las genera.

La arquitectura separa tres responsabilidades:

- **formulario.html**: la interfaz que el usuario ve y edita.

- **servidor.py**: recibe HTTP, entrega parámetros al motor y devuelve al navegador el HTML producido.
- **motor de cálculo**: programa independiente que puede estar escrito en Scilab, Octave, Python, Perl, JavaScript, C, Bash o PowerShell.

El servidor no contiene el algoritmo científico. Todos los motores reciben los parámetros mediante el mismo archivo de texto y escriben el resultado mediante el mismo protocolo. Para cambiar de lenguaje no se modifica `servidor.py`; se elige otro motor.

8.3.2. Instalación y uso rápido

Requisitos

- Python 3.6 o posterior para ejecutar `servidor.py`.
- Al menos uno de los motores de cálculo que se quiera utilizar: Scilab, Octave, Python, Perl, Node.js, un ejecutable C, Bash o PowerShell.
- gnuplot cuando el motor elegido lo utilice para generar gráficas.

Para el ejemplo principal con Scilab y gnuplot:

```
sudo apt install scilab gnuplot
```

Estructura de archivos

El formulario y el servidor deben estar en el mismo directorio. Se agrega el archivo del motor o de los motores que se quieran utilizar:

```
mi_aplicacion/
    formulario.html      # interfaz web
    servidor.py          # servidor HTTP comun
    calcular.sce         # motor Scilab
    calcular.m           # motor Octave, si se usa
    calcular.py          # motor Python, si se usa
    calcular.pl          # motor Perl, si se usa
    calcular.js          # motor JavaScript/Node, si se usa
    calcular             # motor C compilado, si se usa
    calcular.sh          # motor Bash, si se usa
    calcular.ps1         # motor PowerShell, si se usa
```

Iniciar el servidor

```
cd mi_aplicacion/
python3 servidor.py
```

Abrir en el navegador: `http://localhost:8080`

Para detener: `Ctrl+C` en la terminal.

8.3.3. Instalación del ejemplo Scilab como paquete

La variante original basada en Scilab se distribuye como paquete Debian. La instalación usa `dpkg` para desempaquetar y `apt` para resolver las dependencias faltantes; el manejo de ambas herramientas está descrito en Hertzog y Mas 2026.

```
$wget
   https://ututo.org/userfiles/downloads/servguiweb_0009-1_all.deb
$sudo dpkg -i servguiweb_0009-1_all.deb
$sudo apt-get install -f    # instala dependencias si faltan
$
$servguiweb proyectil     # abre el simulador en
   http://localhost:8080
$servguiweb estadistica   # abre el modulo de estadistica en
   http://localhost:8080
```

8.3.4. Manual de usuario

Completar el formulario

La página principal muestra el formulario con los parámetros de la simulación. Cada campo tiene un valor por defecto que puede modificarse:

- **Nombre**: identificacion del experimentador.
- **Velocidad inicial**: en metros por segundo (1 a 500).
- **Angulo de lanzamiento**: en grados (1 a 89).
- **Resistencia del aire**: cuatro niveles predefinidos.
- **Método numérico**: Euler, Runge-Kutta 4 u ODE adaptativo.
- **Motor de cálculo**: si el formulario ofrece el selector, elegir Scilab, Octave, Python u otro motor instalado.
- **Gráficas**: seleccionar cuáles generar.
- **Notas**: texto libre para registrar observaciones.

Obtener resultados

Presionar **Calcular**. El servidor recibe los datos del formulario, selecciona el motor de cálculo indicado y ejecuta el programa correspondiente. El motor realiza la simulación y escribe el fragmento HTML de resultados, que el servidor inserta en la misma página. Los valores ingresados se conservan para facilitar la comparación entre simulaciones.

Interpretar los resultados

Los resultados incluyen cuatro estadísticas numéricas:

Estadística	Descripción
Alcance	Distancia horizontal al impacto [m]
Altura máxima	Punto más alto de la trayectoria [m]
Tiempo de vuelo	Duracion total del vuelo [s]
Velocidad de impacto	Modulo de la velocidad al tocar el suelo [m/s]

Debajo aparecen las gráficas seleccionadas: trayectoria (y vs x) y velocidad vs tiempo.

Comparar simulaciones

Para comparar el efecto de distintos parámetros conviene abrir dos ventanas del navegador en `http://localhost:8080` y variar un parametro entre ambas.

8.3.5. Documentación técnica

Arquitectura del sistema

El flujo de una solicitud es el siguiente:

1. El navegador envía un `POST` con los datos del formulario.
2. `servidor.py` decodifica los campos HTTP.
3. El servidor separa el parámetro opcional `motor` y escribe los demás campos en `/tmp/params.txt`, usando una línea `clave=valor` por parámetro.
4. El servidor elige el comando correspondiente al motor y lo ejecuta como subproceso.
5. Las variables de entorno `PARAM_FILE` y `RESULT_FILE` indican al motor dónde leer la entrada y dónde escribir el fragmento HTML de salida.
6. El motor realiza el cálculo y escribe `/tmp/resultado.html`.
7. `servidor.py` inserta ese fragmento en `formulario.html`, en el marcador `<!-- RESULTADOS -->`, y devuelve la página.

El protocolo entre servidor y motor es deliberadamente pequeño. HTTP queda del lado de Python; el cálculo queda del lado del programa científico.

servidor.py

El servidor es ahora independiente del lenguaje de cálculo. Realiza cuatro tareas:

1. **Parseo HTTP**: extrae los campos enviados por el formulario.
2. **Protocolo neutro**: escribe los parámetros como texto `clave=valor`; ya no genera código Scilab.
3. **Selección y ejecución del motor**: elige una orden de una lista cerrada y ejecuta el programa correspondiente.
4. **Ensamblado de la respuesta**: lee el fragmento HTML creado por el motor, lo inserta en el formulario y lo devuelve.

Los archivos de comunicación son comunes a todos los lenguajes:

Archivo	Escribe	Lee
`/tmp/params.txt`	servidor.py	motor de cálculo
`/tmp/resultado.html`	motor	servidor.py
`/tmp/error_motor.txt`	servidor.py	usuario

Listing 8.4: servidor.py multilenguaje completo

```
#!/usr/bin/env python3
# servidor.py  -  mensajero HTTP puro, independiente del motor
   de calculo
# Requiere en el mismo directorio: formulario.html y algun
   calcular.*
# Protocolo con el motor (neutro, vale para cualquier lenguaje):
#   - el servidor escribe PARAM_FILE con lineas  clave=valor
#   - el motor lee PARAM_FILE y escribe RESULT_FILE (fragmento
   HTML)
#   - opcionalmente el motor escribe VERDICT_FILE (clave=valor)
   con el
#     veredicto que elige el proximo formulario del flujo
#   - las rutas se pasan por variables de entorno (PARAM_FILE,
#     RESULT_FILE, VERDICT_FILE y, si hay sesion, SESION_FILE)
# Sesiones: un archivo clave=valor por sesion en SESIONES_DIR,
   con el
# itinerario de formularios. Navegacion: campos ocultos
# _si_<veredicto>=<destino.html> en el formulario ('*' = lo
   elige el
# motor). Detalles en doc/secciones/documentacion_tecnica.tex.
VERSION = '0010'

from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import parse_qs
from html import escape
from datetime import datetime
```

```
import os
import re
import secrets
import subprocess
import tempfile

PUERTO       = 8080
DIR          = os.path.dirname(os.path.abspath(__file__))
FORM_FILE    = os.path.join(DIR, 'formulario.html')
PARAM_FILE   = '/tmp/params.txt'
RESULT_FILE  = '/tmp/resultado.html'
ERROR_FILE   = '/tmp/error_motor.txt'
VERDICT_FILE = '/tmp/veredicto.txt'
MARCADOR     = '<!-- RESULTADOS -->'

# Sesiones: un archivo <id>.txt con lineas clave=valor. Si el
# directorio por omision no es escribible se usa DIR/sesiones.
SESIONES_DIR = os.environ.get('SESIONES_DIR',
   '/var/lib/servguiweb/sesiones')
SESION_TTL   = int(os.environ.get('SESION_TTL', '3600'))

# Si el formulario no envia "motor", se usa este valor.
# Tambien puede elegirse al iniciar:
#   MOTOR=python python3 servidor.py
MOTOR_POR_DEFECTO = os.environ.get('MOTOR', 'scilab').lower()

# Archivo de calculo que espera cada motor en DIR.
ARCHIVO_MOTOR = {
    'scilab':     'calcular.sce',
    'octave':     'calcular.m',
    'python':     'calcular.py',
    'perl':       'calcular.pl',
    'javascript': 'calcular.js',
    'c':          'calcular',
    'bash':       'calcular.sh',
    'powershell': 'calcular.ps1',
}

MOTORES = {
    'scilab': [
        'scilab-cli', '-e',
        "exec('" + os.path.join(DIR, 'calcular.sce') + "', -1);
           quit"
    ],
    'octave': [
        'octave-cli', '--quiet', os.path.join(DIR, 'calcular.m')
    ],
    'python': [
        'python3', os.path.join(DIR, 'calcular.py')
    ],
    'perl': [
```

```
            'perl', os.path.join(DIR, 'calcular.pl')
        ],
        'javascript': [
            'node', os.path.join(DIR, 'calcular.js')
        ],
        'c': [
            os.path.join(DIR, 'calcular')
        ],
        'bash': [
            'bash', os.path.join(DIR, 'calcular.sh')
        ],
        'powershell': [
            'pwsh', '-File', os.path.join(DIR, 'calcular.ps1')
        ],
    }

    def motores_disponibles():
        """Motores cuyo archivo calcular.* existe en DIR."""
        return [m for m, arch in ARCHIVO_MOTOR.items()
                if os.path.exists(os.path.join(DIR, arch))]

    def verificar_archivos():
        faltantes = []
        if not os.path.exists(FORM_FILE):
            faltantes.append(FORM_FILE)
        if not motores_disponibles():
            faltantes.append(os.path.join(DIR, 'calcular.*  (algun
               motor)'))
        return faltantes

    def leer_form(nombre='formulario.html'):
        with open(os.path.join(DIR, nombre), encoding='utf-8') as f:
            return f.read()

    def dir_sesiones():
        """Directorio de sesiones; cae a DIR/sesiones si no es
           escribible."""
        global SESIONES_DIR
        try:
            os.makedirs(SESIONES_DIR, exist_ok=True)
        except OSError:
            SESIONES_DIR = os.path.join(DIR, 'sesiones')
            os.makedirs(SESIONES_DIR, exist_ok=True)
        return SESIONES_DIR

    def _ruta_sesion(sid):
        # los ids son hexadecimales: nunca pueden ser una ruta
        if not sid or not re.fullmatch(r'[0-9a-f]+', sid):
            return None
        return os.path.join(dir_sesiones(), sid + '.txt')
```

```
def _leer_lineas(sid):
    path = _ruta_sesion(sid)
    if not path or not os.path.exists(path):
        return None
    with open(path, encoding='utf-8') as f:
        return f.read().splitlines()

def _guardar_lineas(sid, lineas):
    with open(os.path.join(dir_sesiones(), sid + '.txt'), 'w',
              encoding='utf-8') as f:
        f.write('\n'.join(lineas) + '\n')

def _cabecera(lineas, clave):
    for l in lineas:
        if l.startswith(clave + '='):
            return l[len(clave) + 1:]
    return None

def nueva_sesion(form):
    """Crea una sesion con `form` como formulario abierto."""
    sid = secrets.token_hex(8)
    ahora = datetime.now().isoformat(timespec='seconds')
    _guardar_lineas(sid, [f'creada={ahora}', f'tocada={ahora}',
                          f'abierto={form}',
                             f'evento=servido:{form}'])
    return sid

def sesion_valida(sid):
    lineas = _leer_lineas(sid)
    if lineas is None:
        return False
    tocada = _cabecera(lineas, 'tocada')
    try:
        t = datetime.fromisoformat(tocada)
    except (TypeError, ValueError):
        return False
    return (datetime.now() - t).total_seconds() < SESION_TTL

def abierto_de(sid):
    """Formulario que la sesion tiene abierto, o None si no
       existe."""
    lineas = _leer_lineas(sid)
    if lineas is None:
        return None
    return _cabecera(lineas, 'abierto')

def _actualizar(sid, mutar):
    lineas = _leer_lineas(sid)
    if lineas is None:
        return
    _guardar_lineas(sid, mutar(lineas))
```

```
def registrar_evento(sid, texto):
    """Agrega evento=<texto> al itinerario y actualiza
       'tocada'."""
    ahora = datetime.now().isoformat(timespec='seconds')
    def m(lineas):
        return ['tocada=' + ahora if l.startswith('tocada=')
           else l
                for l in lineas] + ['evento=' + texto]
    _actualizar(sid, m)

def cambiar_abierto(sid, form):
    def m(lineas):
        out, puesto = [], False
        for l in lineas:
            if l.startswith('abierto='):
                l, puesto = 'abierto=' + form, True
            out.append(l)
        return out if puesto else out + ['abierto=' + form]
    _actualizar(sid, m)

# claves de la sesion que solo escribe el servidor
CLAVES_RESERVADAS = ('creada', 'tocada', 'abierto', 'evento')

def poner_en_sesion(sid, clave, valor):
    """Anota clave=valor en la sesion, a pedido del motor (lineas
    guardar_<clave>=<valor> en VERDICT_FILE). Valor vacio borra
       el
    dato. Las claves reservadas no se pueden pisar."""
    if clave in CLAVES_RESERVADAS or not
       re.fullmatch(r'[A-Za-z0-9_]+', clave):
        return
    valor = valor.replace('\r', ' ').replace('\n', ' ')
    def m(lineas):
        if valor == '':
            return [l for l in lineas if not l.startswith(clave
               + '=')]
        nueva = clave + '=' + valor
        out, puesto = [], False
        for l in lineas:
            if l.startswith(clave + '='):
                l, puesto = nueva, True
            out.append(l)
        return out if puesto else out + [nueva]
    _actualizar(sid, m)

def destino_valido(nombre):
    """Solo archivos .html del directorio de la aplicacion."""
    if not nombre or nombre != os.path.basename(nombre):
        return False
    if not nombre.endswith('.html'):
```

```
            return False
        return os.path.exists(os.path.join(DIR, nombre))

    def leer_veredicto():
        """VERDICT_FILE como dict clave=valor (vacio si el motor no
           lo escribio)."""
        if not os.path.exists(VERDICT_FILE):
            return {}
        out = {}
        with open(VERDICT_FILE, encoding='utf-8') as f:
            for linea in f:
                k, _, v = linea.rstrip('\n').partition('=')
                if k and k not in out:
                    out[k] = v
        return out

    def leer_transiciones(form):
        """Mapa veredicto -> destino declarado en el formulario
           (campos
        ocultos  <input type=hidden name=_si_<veredicto>
           value="...">).
        Se lee del archivo, no del POST: el cliente no puede inventar
        destinos."""
        try:
            html = leer_form(form)
        except OSError:
            return {}
        out = {}
        for m in
           re.finditer(r'name=_si_([A-Za-z0-9_]+)[^>]*value="([^"]*)"',
                       html):
            out[m.group(1)] = m.group(2)
        return out

    def inyectar_sesion(html, sid, form):
        """Agrega los campos ocultos _sesion y _form antes de
           </form>."""
        i = html.find('</form>')
        if i < 0:
            return html
        campos = (f'<input type=hidden name=_sesion value="{sid}">'
                  f'<input type=hidden name=_form value="{form}">')
        return html[:i] + campos + html[i:]

    def escribir_params(params):
        with open(PARAM_FILE, 'w', encoding='utf-8') as f:
            for k, v in params.items():
                # El archivo es datos, no codigo ejecutable.
                valor = str(v).replace('\r', ' ').replace('\n', ' ')
                f.write(f'{k}={valor}\n')
```

```
def llamar_motor(params, motor, sesion_file=None):
    motor = motor.lower()
    if motor not in MOTORES:
        return '<p>Motor desconocido: ' + escape(motor) + '</p>'

    arch = os.path.join(DIR, ARCHIVO_MOTOR[motor])
    if not os.path.exists(arch):
        return ('<p>Falta el archivo <code>' +
           escape(ARCHIVO_MOTOR[motor]) +
                '</code> para el motor ' + escape(motor) +
                   '.</p>')

    escribir_params(params)

    for path in [RESULT_FILE, ERROR_FILE, VERDICT_FILE]:
        if os.path.exists(path):
            os.remove(path)

    env = os.environ.copy()
    env['PARAM_FILE'] = PARAM_FILE
    env['RESULT_FILE'] = RESULT_FILE
    env['VERDICT_FILE'] = VERDICT_FILE
    if sesion_file:
        env['SESION_FILE'] = sesion_file

    try:
        r = subprocess.run(
            MOTORES[motor],
            stdout=subprocess.PIPE,
            stderr=subprocess.STDOUT,
            timeout=60,
            env=env,
            cwd=DIR
        )
        salida = r.stdout.decode(errors='replace')
    except FileNotFoundError:
        return ('<p>No esta instalado el motor ' + escape(motor)
           + '.</p>')
    except subprocess.TimeoutExpired:
        return '<p>Timeout: el motor tardo mas de 60 s.</p>'

    with open(ERROR_FILE, 'w', encoding='utf-8') as f:
        f.write(salida)

    if r.returncode != 0:
        return ('<h3>Error del motor ' + escape(motor) + '</h3>'
           +
                '<pre>' + escape(salida) + '</pre>')

    if not os.path.exists(RESULT_FILE):
        return ('<p>El motor ' + escape(motor) +
```

```
                ' no genero RESULT_FILE.</p>')

    with open(RESULT_FILE, encoding='utf-8') as f:
        return f.read()

def parsear_multipart(body, boundary):
    """multipart/form-data minimo: devuelve (campos, archivos),
       donde
    archivos es {campo: (nombre_original, contenido_bytes)}."""
    campos, archivos = {}, {}
    for parte in body.split(b'--' + boundary)[1:]:
        if parte[:2] == b'--':
            break
        cab, _, contenido = parte[2:].partition(b'\r\n\r\n')
        if contenido.endswith(b'\r\n'):
            contenido = contenido[:-2]
        m = re.search(br'name="([^"]*)"', cab)
        if not m:
            continue
        nombre = m.group(1).decode('utf-8', errors='replace')
        mf = re.search(br'filename="([^"]*)"', cab)
        if mf:
            archivos[nombre] = (mf.group(1).decode('utf-8',
               errors='replace'),
                                contenido)
        else:
            campos[nombre] = contenido.decode('utf-8',
               errors='replace')
    return campos, archivos

class Handler(BaseHTTPRequestHandler):
    def log_message(self, fmt, *args):
        print(fmt % args)

    def do_GET(self):
        faltantes = verificar_archivos()
        if faltantes:
            msg = '<h2>Error de configuracion</h2>'
            msg += '<p>Archivos no encontrados:</p><ul>'
            for f in faltantes:
                msg += f'<li><code>{f}</code></li>'
            msg += '</ul>'
            msg += '<p>El directorio debe contener:</p>'
            msg += '<pre>  formulario.html\n  calcular.sce  (o
               calcular.py, calcular.m, ...)\n
               servidor.py</pre>'
            msg += f'<p>Directorio actual de servidor.py:
               <code>{DIR}</code></p>'
            self._enviar(f'<!DOCTYPE
               html><html><body>{msg}</body></html>')
            return
```

```
        # la pagina inicial siempre abre una sesion nueva
        sid = nueva_sesion('formulario.html')
        html = leer_form().replace(MARCADOR, '<div
           id=resultados></div>')
        self._enviar(inyectar_sesion(html, sid,
           'formulario.html'))

    def do_POST(self):
        faltantes = verificar_archivos()
        if faltantes:
            self._enviar('<p style=color:red>Archivos
               faltantes.</p>')
            return
        n = int(self.headers.get('Content-Length', 0))
        body = self.rfile.read(n)

        ctype = self.headers.get('Content-Type', '')
        archivos = {}
        if ctype.startswith('multipart/form-data'):
            m = re.search(r'boundary=(.+)', ctype)
            if not m:
                self._enviar('<p>multipart sin boundary.</p>')
                return
            params, archivos = parsear_multipart(
                body, m.group(1).strip().encode())
        else:
            # parse_qs conserva todos los valores. Para un
               checkbox con
            # hidden=off seguido del checkbox=on interesa el
               ultimo.
            campos = parse_qs(body.decode('utf-8',
               errors='replace'),
                              keep_blank_values=True)
            params = {k: valores[-1] for k, valores in
               campos.items()}

        # Sesion (opcional): un POST sin _sesion se procesa sin
           sesion,
        # como en la version 0009.
        sid = params.pop('_sesion', '')
        form = params.pop('_form', 'formulario.html')
        if sid:
            if not sesion_valida(sid):
                self._enviar('<p>Sesion desconocida o expirada. '
                             'Recargar la pagina inicial.</p>')
                return
            if form != abierto_de(sid):
                self._enviar('<p>El formulario ' + escape(form) +
                             ' no esta abierto en esta
                                sesion.</p>')
                return
```

```
            registrar_evento(sid, 'post:' + form)

        # Los archivos subidos se guardan en un temporario
           propio y
        # viajan al motor como rutas (mas __nombre y __bytes).
        if archivos:
            updir = os.path.join(tempfile.gettempdir(),
                                 'servguiweb_' + (sid or 'anon'))
            os.makedirs(updir, exist_ok=True)
            for campo, (nombre, contenido) in archivos.items():
                base = os.path.basename(nombre) or 'archivo'
                ruta = os.path.join(updir, base)
                with open(ruta, 'wb') as f:
                    f.write(contenido)
                params[campo] = ruta
                params[campo + '__nombre'] = base
                params[campo + '__bytes'] = str(len(contenido))

        # El motor puede venir del formulario. Si no viene, se
           usa
        # MOTOR_POR_DEFECTO. El nombre se valida contra MOTORES.
        motor = params.pop('motor', MOTOR_POR_DEFECTO)

        div = llamar_motor(params, motor, _ruta_sesion(sid) if
           sid else None)

        # Navegacion: el veredicto del motor elige el proximo
           formulario
        # entre los declarados con _si_<veredicto>; '*' deja la
           eleccion
        # al motor. Por omision (sin sesion o sin veredicto) se
           vuelve
        # al mismo formulario.
        destino = form if sid else 'formulario.html'
        if sid:
            vdata = leer_veredicto()
            # el motor puede pedir anotar datos del negocio en
               la sesion
            for k, v in vdata.items():
                if k.startswith('guardar_'):
                    poner_en_sesion(sid, k[len('guardar_'):], v)
            veredicto = vdata.get('veredicto', '')
            if veredicto:
                registrar_evento(sid, 'veredicto:' + veredicto)
                declarado =
                   leer_transiciones(form).get(veredicto)
                if declarado == '*':
                    elegido = vdata.get('destino', '')
                    if not destino_valido(elegido):
                        self._enviar('<p>Destino no permitido: '
                           +
```

```
                                        escape(elegido or
                                            '(vacio)') + '</p>')
                            return
                        destino = elegido
                    elif declarado and destino_valido(declarado):
                        destino = declarado
                cambiar_abierto(sid, destino)
                registrar_evento(sid, 'servido:' + destino)

            html_final = leer_form(destino).replace(
                MARCADOR, f'<div id=resultados>{div}</div>')
            if sid:
                html_final = inyectar_sesion(html_final, sid,
                    destino)
            self._enviar(html_final)

        def _enviar(self, html):
            data = html.encode('utf-8')
            self.send_response(200)
            self.send_header('Content-Type', 'text/html;
                charset=UTF-8')
            self.send_header('Content-Length', len(data))
            self.end_headers()
            self.wfile.write(data)

    # verificar al arrancar
    faltantes = verificar_archivos()
    print(f'servguiweb {VERSION}')
    print('Pagina del proyecto: https://ututo.org/?SERVIDOR-GUI-WEB')
    print(f'Servidor en http://localhost:{PUERTO}')
    print(f'Directorio: {DIR}')
    if faltantes:
        print('ADVERTENCIA: archivos no encontrados:')
        for f in faltantes:
            print(f'  FALTA: {f}')
    else:
        print(f'Formulario : {FORM_FILE}')
        print(f'Motores    : {", ".join(motores_disponibles())}')
        print(f'Motor inicial: {MOTOR_POR_DEFECTO}')
    print(f'Log del motor -> {ERROR_FILE}')
    print(f'Sesiones    -> {dir_sesiones()}  (TTL {SESION_TTL} s)')
    print('Ctrl+C para detener')

    try:
        # Escucha solo en la maquina local.
        HTTPServer(('127.0.0.1', PUERTO), Handler).serve_forever()
    except OSError as e:
        if e.errno == 98:
            print()
            print(f'ERROR: el puerto {PUERTO} ya esta en uso.')
            print()
```

```
            print('Solucion A - matar el servidor anterior:')
            print('  pkill -f servidor.py')
            print(f'  servguiweb {"
               ".join(__import__("sys").argv[1:])}')
            print()
            print('Solucion B - usar otro puerto:')
            print(f'  sed -i "s/PUERTO.*= {PUERTO}/PUERTO =
               {PUERTO+1}/" {__file__}')
            print(f'  python3 {__file__}')
            print(f'  # abrir http://localhost:{PUERTO+1}')
            print()
            print('Ver que proceso usa el puerto:')
            print(f'  ss -tlnp | grep {PUERTO}')
        else:
            raise
```

El servidor escucha en `127.0.0.1:8080`; por lo tanto, por omisión, solo acepta conexiones desde la misma computadora. Para usarlo en una red hay que cambiar deliberadamente la dirección de escucha y agregar las medidas de seguridad correspondientes.

Tratamiento de errores del motor. El servidor distingue varios fallos antes de construir la respuesta HTML:

Motor desconocido: si el formulario solicita un nombre que no está en el diccionario `MOTORES`, se rechaza la selección y no se ejecuta ningún comando arbitrario.

Motor no instalado: si el ejecutable correspondiente no existe en el sistema, se informa que ese motor no está disponible.

Tiempo excedido: si el proceso tarda más de 60 segundos, se lo interrumpe y se devuelve un mensaje de *timeout*.

Código de salida distinto de cero: la salida del proceso se guarda en `/tmp/error_motor.txt` y se muestra como diagnóstico.

Resultado ausente: si el proceso termina sin crear el archivo indicado por `RESULT_FILE`, el servidor informa que el motor no generó el resultado esperado.

Esto mantiene separados tres tipos de información: el código de salida indica éxito o fracaso, la salida capturada sirve para diagnóstico y `RESULT_FILE` contiene únicamente el fragmento HTML que debe mostrarse al usuario cuando el cálculo termina correctamente.

Errores de entrada y fallas del programa. Conviene distinguir dos situaciones diferentes. Un **error de entrada** forma parte del funcionamiento normal de la aplicación: por ejemplo, una masa negativa, un ángulo fuera del intervalo permitido o un color con formato incorrecto. El motor debe detectar ese caso, escribir en `RESULT_FILE` un mensaje HTML comprensible para el usuario y terminar normalmente, con código de salida cero. De ese modo el servidor puede devolver la explicación en la misma página.

Una **falla inesperada del programa** es distinta: archivo inexistente, error de cálculo no previsto, excepción, orden externa que falla, etc. En ese caso el motor no debe presentar el fallo como si fuese un resultado científico. Debe enviar el diagnóstico a la salida de error y terminar con código de salida distinto de cero. `servidor.py` puede entonces conservar ese diagnóstico en `/tmp/error_motor.txt` y tratar la solicitud como fallida.

El mecanismo concreto depende del lenguaje:

Lenguaje	Mecanismo habitual para una falla inesperada
Scilab	`execstr(...,'errcatch')` y comprobación del código de error
Octave	`try`/`catch`; `rethrow` si el proceso debe fallar
Python	`try`/`except`; diagnóstico por `stderr` y salida no nula
Perl	`eval`; `warn`/`die` según corresponda
JavaScript	`try`/`catch`; excepción no resuelta para indicar fallo
C	códigos de retorno y comprobación explícita de errores
Bash	estado de salida; opcionalmente `set -e`, `trap ERR`
PowerShell	`try`/`catch` y `ErrorActionPreference=Stop`

No debe confundirse la *captura* de un error con su *ocultamiento*. Si una excepción se captura para producir un diagnóstico técnico pero representa una falla inesperada, el programa debe conservar finalmente un estado de salida no nulo. Si, en cambio, se trata de un dato inválido previsto por la interfaz, puede producirse una respuesta HTML normal explicando cómo corregirlo.

Archivos estáticos

Hasta aquí, el método `do_GET` del servidor devuelve siempre la página principal. Una aplicación web, sin embargo, suele utilizar también archivos que el navegador solicita por separado: hojas de estilo CSS, imágenes, código JavaScript, iconos, fuentes u otros recursos.

Se denomina **archivo estático** a un archivo que el servidor entrega como está almacenado, sin generarlo nuevamente para cada solicitud. El adjetivo *estático* no significa que el archivo nunca pueda modificarse: significa que su contenido no se calcula a partir de los datos particulares de esa petición.

Por ejemplo, una página puede contener

```
<link rel="stylesheet" href="estilo.css">
<img src="logo.png" alt="Logo">
<script src="interfaz.js"></script>
```

Después de recibir el HTML, el navegador realiza solicitudes GET separadas:

GET /estilo.css ⟶ servidor ⟶ archivo estilo.css ⟶ navegador.

En este caso el servidor no ejecuta `estilo.css`: simplemente lee sus bytes y los devuelve con el tipo MIME correspondiente. Lo mismo ocurre con una imagen PNG. Un archivo

JavaScript también puede servirse como archivo estático; quien ejecuta ese JavaScript es luego el navegador.

Esto es diferente del contenido dinámico producido por el cálculo:

POST del formulario ⟶ servidor.py ⟶ motor de cálculo ⟶ RESULT_FILE ⟶ HTML de r

Por lo tanto, una misma aplicación puede combinar recursos estáticos y contenido dinámico:

Recurso	Ejemplo	Tratamiento
HTML principal	`formulario.html`	leído y enviado por el servidor
CSS	`estilo.css`	archivo estático
Imagen	`logo.png`	archivo estático
JavaScript	`interfaz.js`	archivo estático; lo ejecuta el navegador
Resultado	`RESULT_FILE`	generado dinámicamente por el motor

Extensión sencilla de `do_GET`. Para el servidor didáctico conviene comenzar con una lista cerrada de recursos permitidos. Además de ser simple, evita que una URL construida por el cliente pueda utilizarse para leer archivos arbitrarios del sistema.

Puede agregarse, cerca de las constantes del servidor,

```
ARCHIVOS_ESTATICOS = {
    '/estilo.css':  ('estilo.css',  'text/css; charset=UTF-8'),
    '/logo.png':    ('logo.png',    'image/png'),
    '/interfaz.js': ('interfaz.js', 'text/javascript;
       charset=UTF-8'),
}
```

y reemplazar `do_GET` por la siguiente versión, agregando también el método auxiliar `_enviar_estatico` a la clase `Handler`:

```
def do_GET(self):
    if self.path in ARCHIVOS_ESTATICOS:
        nombre, tipo = ARCHIVOS_ESTATICOS[self.path]
        self._enviar_estatico(nombre, tipo)
        return

    if self.path == '/' or self.path == '/index.html':
        html = leer_form().replace(
            MARCADOR, '<div id=resultados></div>')
        self._enviar(html)
        return

    self.send_error(404)


def _enviar_estatico(self, nombre, tipo):
```

```
        archivo = os.path.join(DIR, nombre)

        if not os.path.isfile(archivo):
            self.send_error(404)
            return

        with open(archivo, 'rb') as f:
            data = f.read()

        self.send_response(200)
        self.send_header('Content-Type', tipo)
        self.send_header('Content-Length', len(data))
        self.end_headers()
        self.wfile.write(data)
```

Obsérvese que `_enviar` y `_enviar_estatico` no hacen lo mismo. El primero recibe texto HTML, lo codifica como UTF-8 y fija `text/html`; el segundo trabaja con bytes porque una imagen, por ejemplo, no es texto, y recibe explícitamente el tipo MIME que debe anunciar al navegador.

En un servidor de producción esta tarea suele delegarse a infraestructura especializada o a mecanismos generales para servir directorios estáticos. En este servidor pequeño la lista cerrada permite estudiar con claridad qué ocurre en cada solicitud sin mezclar ese problema con el motor de cálculo.

El mismo servidor con motores escritos en distintos lenguajes

El `servidor.py` anterior ya es multilenguaje. El navegador no necesita saber cómo se ejecuta Scilab, Octave, Python o cualquier otro motor: todos obedecen el mismo contrato.

navegador ⟶ servidor HTTP ⟶ params.txt ⟶ motor ⟶ resultado.html ⟶ navegador.

El archivo de parámetros es texto y no se ejecuta. Por ejemplo:

```
v0=50
angulo=45
arr=bajo
met=rk4
```

Las rutas de entrada y salida se comunican al motor mediante las variables de entorno `PARAM_FILE` y `RESULT_FILE`. El motor solo debe leer el primer archivo y crear el segundo.

El motor se elige de dos formas. Sin modificar el servidor se puede fijar al iniciarlo:

```
MOTOR=python python3 servidor.py
MOTOR=octave python3 servidor.py
MOTOR=scilab python3 servidor.py
```

También puede enviarse un campo `motor` desde el formulario. El servidor acepta únicamente los nombres presentes en el diccionario `MOTORES`; el formulario no puede enviar una

orden arbitraria al sistema operativo.

```
<label>Motor de calculo</label>
<select name=motor>
  <option value=scilab>Scilab</option>
  <option value=octave>Octave</option>
  <option value=python>Python</option>
  <option value=perl>Perl</option>
  <option value=javascript>JavaScript / Node.js</option>
  <option value=c>C</option>
  <option value=bash>Bash</option>
  <option value=powershell>PowerShell</option>
</select>
```

Motor	Archivo	Ejecución
Scilab	`calcular.sce`	`scilab-cli`
Octave	`calcular.m`	`octave-cli`
Python	`calcular.py`	`python3`
Perl	`calcular.pl`	`perl`
JavaScript	`calcular.js`	`node`
C	`calcular`	ejecutable compilado
Bash	`calcular.sh`	`bash`
PowerShell	`calcular.ps1`	`pwsh`

Todos los ejemplos siguientes realizan el mismo cálculo sencillo: leen `v0` y `angulo`, calculan el tiempo de subida de un proyectil sin resistencia del aire,

$$t_s = \frac{v_0 \sin(\alpha)}{g},$$

y escriben un fragmento HTML en el archivo indicado por `RESULT_FILE`. El objetivo no es comparar la sintaxis de los lenguajes sino mostrar que el protocolo con el servidor es el mismo.

Listing 8.5: calcular_generico.sce

Scilab

```
PARAM_FILE  = getenv('PARAM_FILE')
RESULT_FILE = getenv('RESULT_FILE')

lineas = mgetl(PARAM_FILE)
v0 = 50
angulo = 45

for i = 1:size(lineas, '*')
    p = strsplit(lineas(i), '=')
    if size(p, '*') == 2 then
        if p(1) == 'v0' then v0 = strtod(p(2)); end
        if p(1) == 'angulo' then angulo = strtod(p(2)); end
```

```
    end
end

ts = v0 * sin(angulo * %pi / 180) / 9.8
fd = mopen(RESULT_FILE, 'w')
mfprintf(fd, '<p>Tiempo de subida: %.3f s</p>', ts)
mclose(fd)
```

Listing 8.6: calcular.m

Octave

```
param_file = getenv('PARAM_FILE');
result_file = getenv('RESULT_FILE');

v0 = 50; angulo = 45;
fid = fopen(param_file, 'r');
while true
  linea = fgetl(fid);
  if ~ischar(linea), break; end
  p = strsplit(linea, '=');
  if numel(p) == 2
    if strcmp(p{1}, 'v0'), v0 = str2double(p{2}); end
    if strcmp(p{1}, 'angulo'), angulo = str2double(p{2}); end
  end
end
fclose(fid);

ts = v0 * sin(angulo*pi/180) / 9.8;
fid = fopen(result_file, 'w');
fprintf(fid, '<p>Tiempo de subida: %.3f s</p>', ts);
fclose(fid);
```

Listing 8.7: calcular.py

Python

```
import math, os

p = {}
with open(os.environ['PARAM_FILE'], encoding='utf-8') as f:
    for linea in f:
        if '=' in linea:
            k, v = linea.rstrip().split('=', 1)
            p[k] = v

v0 = float(p.get('v0', 50))
angulo = float(p.get('angulo', 45))
ts = v0 * math.sin(math.radians(angulo)) / 9.8

with open(os.environ['RESULT_FILE'], 'w', encoding='utf-8') as f:
    f.write(f'<p>Tiempo de subida: {ts:.3f} s</p>')
```

Listing 8.8: calcular.pl

Perl

```
use strict;
use warnings;

my %p;
open my $in, '<', $ENV{PARAM_FILE} or die $!;
while (<$in>) {
    chomp;
    my ($k, $v) = split(/=/, $_, 2);
    $p{$k} = $v if defined $v;
}
close $in;

my $v0 = $p{v0} // 50;
my $angulo = $p{angulo} // 45;
my $pi = 4 * atan2(1, 1);
my $ts = $v0 * sin($angulo*$pi/180) / 9.8;

open my $out, '>', $ENV{RESULT_FILE} or die $!;
printf $out '<p>Tiempo de subida: %.3f s</p>', $ts;
close $out;
```

Listing 8.9: calcular.js

JavaScript con Node.js

```
const fs = require('fs');

const p = {};
for (const linea of fs.readFileSync(process.env.PARAM_FILE,
   'utf8')
                        .split(/\r?\n/)) {
  const i = linea.indexOf('=');
  if (i >= 0) p[linea.slice(0,i)] = linea.slice(i+1);
}

const v0 = Number(p.v0 ?? 50);
const angulo = Number(p.angulo ?? 45);
const ts = v0 * Math.sin(angulo*Math.PI/180) / 9.8;

fs.writeFileSync(process.env.RESULT_FILE,
  `<p>Tiempo de subida: ${ts.toFixed(3)} s</p>`);
```

Listing 8.10: calcular.c

C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

int main(void) {
```

```
    const char *pf = getenv("PARAM_FILE");
    const char *rf = getenv("RESULT_FILE");
    FILE *f = fopen(pf, "r");
    char linea[256];
    double v0 = 50.0, angulo = 45.0;

    while (fgets(linea, sizeof linea, f)) {
        if (sscanf(linea, "v0=%lf", &v0) == 1) continue;
        if (sscanf(linea, "angulo=%lf", &angulo) == 1) continue;
    }
    fclose(f);

    double pi = acos(-1.0);
    double ts = v0 * sin(angulo*pi/180.0) / 9.8;

    f = fopen(rf, "w");
    fprintf(f, "<p>Tiempo de subida: %.3f s</p>", ts);
    fclose(f);
    return 0;
}
```

Se compila una vez:

```
gcc calcular.c -o calcular -lm
```

Listing 8.11: calcular.sh

Bash

```
v0=50
angulo=45

while IFS='=' read -r k v; do
    case "$k" in
        v0) v0="$v" ;;
        angulo) angulo="$v" ;;
    esac
done < "$PARAM_FILE"

ts=$(LC_ALL=C awk -v v="$v0" -v a="$angulo" \
     'BEGIN{pi=atan2(0,-1); printf "%.3f", v*sin(a*pi/180)/9.8}')

printf '<p>Tiempo de subida: %s s</p>' "$ts" > "$RESULT_FILE"
```

Listing 8.12: calcular.ps1

PowerShell

```
$p = @{}
Get-Content $env:PARAM_FILE | ForEach-Object {
    $k, $v = $_ -split '=', 2
    if ($null -ne $v) { $p[$k] = $v }
}
```

```
$v0 = if ($p.ContainsKey('v0')) { [double]$p['v0'] } else { 50 }
$a  = if ($p.ContainsKey('angulo')) { [double]$p['angulo'] }
   else { 45 }
$ts = $v0 * [Math]::Sin($a * [Math]::PI / 180) / 9.8

'<p>Tiempo de subida: {0:F3} s</p>' -f $ts |
    Set-Content -Encoding UTF8 $env:RESULT_FILE
```

La consecuencia importante es que el servidor HTTP deja de ser un "servidor para Scilab". Es un servidor para programas de cálculo. El lenguaje pasa a ser una decisión interna del motor. Esto permite conservar la misma interfaz web aunque el algoritmo se reescriba en otro lenguaje.

Sesiones y flujos de formularios

Hasta aquí cada aplicación tenía un único `formulario.html`: la respuesta del motor se insertaba siempre en la misma página. Desde la versión 0010 el servidor admite **flujos de varios formularios**: cada formulario puede declarar a qué otro formulario se pasa según el resultado del motor, y el servidor registra el recorrido de cada visitante en una **sesión** persistente. Las aplicaciones de un solo formulario siguen funcionando sin cambios: si un formulario no declara destinos, el próximo formulario es el mismo.

Lista blanca de formularios. El servidor solo puede mostrar los archivos `.html` presentes en el directorio de la aplicación. Ningún parámetro del navegador ni del motor puede hacerlo servir otro archivo: el destino de cada transición se valida contra esa lista cerrada, tanto si fue declarado en un formulario como si lo indicó el motor.

Sesiones. Al pedir la página inicial el servidor crea una sesión con un identificador aleatorio, que viaja en el campo oculto `_sesion`. El propio servidor inyecta ese campo en cada formulario que sirve, junto con `_form` (el nombre del formulario, usado para el control de saltos descripto más abajo). No se usan cookies.

Cada sesión es un archivo de texto con líneas `clave=valor` en el directorio de sesiones (por omisión `/var/lib/servguiweb/sesiones/`, configurable con la variable de entorno `SESIONES_DIR`; si no existe o no es escribible, se usa el subdirectorio `sesiones/` del directorio de la aplicación):

```
creada=2026-08-27T12:00:03
tocada=2026-08-27T12:04:41
abierto=bienvenido.html
evento=servido:inicio.html
evento=post:inicio.html
evento=veredicto:ok
evento=servido:bienvenido.html
```

Las primeras líneas describen la sesión (`creada`, `tocada`, `abierto`: el formulario que la sesión tiene abierto en este momento) y las líneas `evento=` forman el itinerario: qué

formulario se sirvió, qué formulario se recibió y qué veredicto devolvió el motor. El archivo es legible y editable a mano: el servidor lo relee en cada solicitud y conserva las líneas que no conoce, de modo que editarlo (por ejemplo, tocar `tocada` para revivir una sesión expirada o cambiar `abierto`) es la válvula de escape de cualquier bloqueo.

Las sesiones expiran pasado un tiempo configurable (`SESION_TTL`, una hora por omisión). Un `POST` con sesión desconocida o expirada se rechaza; pedir la página inicial siempre crea una sesión nueva.

Toda la escritura del archivo de sesión la hace el servidor. Si el motor necesita conocer la historia (qué se respondió tres pasos atrás), recibe la ruta en la variable de entorno `SESION_FILE` y solo lee.

El motor puede pedir que se anoten datos en la sesión. Además del veredicto, el motor puede escribir en `VERDICT_FILE` líneas `guardar_<clave>=<valor>`:

```
veredicto=ok
guardar_usuario=diego@example.com
```

El servidor es quien escribe esa línea en el archivo de sesión (como `usuario=diego@example.com`): el motor *pide*, el servidor *anota*. Así la sesión pasa a "saber" datos del negocio —por ejemplo, quién es el usuario tras un login— y cualquier paso posterior puede leerlos de `SESION_FILE`. Las claves de control del servidor (`creada`, `tocada`, `abierto`, `evento`) no se pueden pisar por esta vía: se ignoran. Pedir `guardar_usuario=` con el valor vacío borra el dato.

Un `POST` sin campo `_sesion` se procesa sin sesión, exactamente como en la versión 0009: sirve de nuevo el formulario principal con el fragmento insertado. Esto mantiene compatibles los clientes que no pasan por la página servida (scripts, pruebas).

Navegación por veredictos. Cada formulario declara sus transiciones posibles con campos ocultos `_si_<veredicto>`:

```
<input type=hidden name=_si_ok value="bienvenido.html">
<input type=hidden name=_si_error value="reintentar.html">
```

Esos campos llegan al motor como parámetros comunes, dentro de `PARAM_FILE`. El motor calcula y, además del fragmento HTML de siempre en `RESULT_FILE`, puede escribir el archivo indicado por la variable de entorno `VERDICT_FILE`:

```
veredicto=ok
```

El servidor busca entonces el veredicto entre las transiciones declaradas en el formulario abierto (leyendo el archivo, no los parámetros del `POST`: así el cliente no puede inventarse destinos) y sirve `bienvenido.html` con el fragmento insertado en el marcador. Si el motor no escribe veredicto, o el formulario no declaró ese veredicto, se sirve nuevamente el mismo formulario: ese es el comportamiento por omisión y el de las aplicaciones de la versión 0009.

La válvula: veredicto con destino dinámico. El mapa `_si_*` se escribe cuando se crea el formulario, pero hay flujos cuyo destino no se conoce de antemano: por ejemplo, un recorrido didáctico donde el docente agrega `paso7.html` al directorio y el próximo paso depende de lo que respondió el alumno. Para esos casos un veredicto puede declararse con destino `*`:

```
<input type=hidden name=_si_ir value="*">
```

y el motor indica el destino en `VERDICT_FILE`:

```
veredicto=ir
destino=paso7.html
```

El servidor valida `paso7.html` contra la lista blanca igual que cualquier otro destino: la válvula permite al motor *elegir* el formulario, no *servir* archivos ajenos a la aplicación.

Control de saltos. La sesión registra en `abierto` el último formulario servido. Un `POST` solo se acepta si su campo `_form` coincide con ese valor; cualquier otro `POST` se rechaza. Esto permite recargar el formulario actual, pero impide fabricar envíos de formularios que la sesión no tiene abiertos o volver atrás con el navegador después de avanzar: si retroceder es una operación válida, forma parte de la lógica del negocio y se declara como una transición más (`_si_volver`), no como un atajo automático.

Subida de archivos. Un formulario con `enctype="multipart/form-data"` puede incluir campos `<input type=file>`. El servidor guarda cada archivo recibido en un directorio temporario propio de la sesión y pasa al motor, dentro de `PARAM_FILE`, la ruta y los datos del archivo:

```
adjunto=/tmp/servguiweb_a1b2c3/archivo.csv
adjunto__nombre=archivo.csv
adjunto__bytes=1204
```

El motor lee el archivo desde la ruta indicada, como cualquier otro archivo local. El contenido nunca viaja por `PARAM_FILE`.

Funcionamiento sin usuarios. Las sesiones son anónimas: el servidor no pide login ni conoce cuentas. Si una aplicación necesita identificar usuarios, eso es lógica del negocio y vive en los formularios y el motor (por ejemplo, un primer formulario que pide `usuario` y un veredicto `ok` solo si la clave es válida); los valores enviados quedan registrados en el itinerario de la sesión como cualquier otro parámetro. El servidor no necesita saber nada de eso.

Qué hay debajo de http.server

El módulo `http.server` de Python esconde el trabajo con **sockets**: abrir un descriptor, asociarlo a un puerto, quedar a la escucha, aceptar conexiones y leer y escribir bytes. Ese nivel es el que se programa a mano en C y está descrito en Hall 2026. Conocerlo ayuda

a entender por qué el servidor atiende una solicitud por vez y qué significa que el puerto quede ocupado tras un cierre abrupto.

formulario.html

El formulario es HTML estándar. El único requisito es incluir el marcador `<!-- RESULTADOS -->` donde se insertarán los resultados, y usar `method=POST action=/`. Si incluye un campo `motor`, el servidor usa ese motor; si no lo incluye, usa el motor por omisión definido al iniciar `servidor.py`.

Para que los checkboxes funcionen correctamente se agrega un campo oculto con el mismo nombre antes de cada uno, con valor `off`. El navegador envía `off` cuando el checkbox está desmarcado y `on` cuando está marcado. El servidor conserva esos valores como texto; cada motor los convierte al tipo lógico de su lenguaje:

Listing 8.13: Patron de checkbox correcto

```
<input type=hidden name=g_tray value=off>
<label><input type=checkbox name=g_tray value=on checked>
   Trayectoria</label>
```

calcular.sce: motor Scilab

El motor Scilab ya no recibe un archivo ejecutable `params.sce`. Lee el archivo neutro indicado por `PARAM_FILE` y escribe el HTML en la ruta indicada por `RESULT_FILE`. Esto lo hace compatible con el mismo servidor que usan los demás lenguajes.

Listing 8.14: Estructura minima de calcular.sce

```
// calcular.sce  -  simulacion de proyectil (motor Scilab)
// Lee PARAM_FILE (lineas clave=valor, escrito por el servidor)
// Escribe RESULT_FILE (fragmento HTML) y archivos auxiliares en
   /tmp

PARAM_FILE  = getenv('PARAM_FILE')
RESULT_FILE = getenv('RESULT_FILE')
DATA_FILE   = '/tmp/proy_data.txt'
GRAPH_FILE  = '/tmp/proy_graph.png'

function h = box(val, lab)
    h = '<div class=sb><div class=sv>' + val + '</div>' + lab +
       '</div>'
endfunction

function escribir_error(msg)
    fd = mopen(RESULT_FILE, 'w')
    mfprintf(fd, '<p style=color:red><b>Error:</b> %s</p>', msg)
    mclose(fd)
endfunction

// ----- leer parametros (protocolo neutro clave=valor) -----
```

```
lineas  = mgetl(PARAM_FILE)
claves  = list()
valores = list()
for i = 1:size(lineas, '*')
    p = strindex(lineas(i), '=')
    if ~isempty(p) then
        claves($+1)  = part(lineas(i), 1:p(1)-1)
        valores($+1) = part(lineas(i), p(1)+1:length(lineas(i)))
    end
end

function v = pval(k, def)
    v = def
    for i = 1:length(claves)
        if claves(i) == k then v = valores(i); end
    end
endfunction

function b = pbool(k)
    b = pval(k, 'off') == 'on'
endfunction

function x = pnum(k, def)
    x = strtod(pval(k, string(def)))
    if isnan(x) then x = def; end
endfunction

v0     = pnum('v0', 50)
angulo = pnum('angulo', 45)
arr    = pval('arr', 'cero')
met    = pval('met', 'rk4')
g_tray = pbool('g_tray')
g_vel  = pbool('g_vel')

if v0     <= 0 then v0     = 50; end
if angulo <= 0 then angulo = 45; end
```

Generación de gráficas con gnuplot

Gnuplot genera el PNG sin necesidad de pantalla ni display. gnuplot Team 2026 El script gnuplot se construye en Scilab concatenando strings. Las comillas dobles que gnuplot requiere se insertan usando `ascii(34)` para evitar el error de comillas heterogeneas de Scilab:

Listing 8.15: Construccion del script gnuplot

```
    // construir script gnuplot  (sin comillas dobles en strings
       Scilab)
    NL = ascii(10)
    Q  = ascii(34)
```

```
gp = 'set terminal png size 800,400' + NL
gp = gp + 'set output ' + Q + GRAPH_FILE + Q + NL
gp = gp + 'set grid' + NL

if g_tray & g_vel then
    gp = gp + 'set multiplot layout 1,2' + NL
end

DAT = Q + '/tmp/proy_tray.txt' + Q

if g_tray then
    gp = gp + 'set xlabel ' + Q + 'x [m]'   + Q + NL
    gp = gp + 'set ylabel ' + Q + 'y [m]'   + Q + NL
    gp = gp + 'set title '  + Q + 'Trayectoria' + Q + NL
    gp = gp + 'plot ' + DAT + ' using 2:3 with lines lw 2 lc
       ' + Q + 'blue' + Q + ' notitle' + NL
end

if g_vel then
    gp = gp + 'set xlabel ' + Q + 't [s]'   + Q + NL
    gp = gp + 'set ylabel ' + Q + 'v [m/s]' + Q + NL
    gp = gp + 'set title '  + Q + 'Velocidad' + Q + NL
    gp = gp + 'plot ' + DAT + ' using 1:(sqrt($4*$4+$5*$5))
       with lines lw 2 lc ' + Q + 'blue'  + Q + ' title ' +
       Q + '|v|' + Q + ', \' + NL
    gp = gp + '     ' + DAT + ' using 1:4 with lines lw 1 lc
       '                  + Q + 'red'   + Q + ' title ' + Q
       + 'vx'  + Q + ', \' + NL
    gp = gp + '     ' + DAT + ' using 1:5 with lines lw 1 lc
       '                  + Q + 'green' + Q + ' title ' + Q
       + 'vy'  + Q + NL
end

if g_tray & g_vel then
    gp = gp + 'unset multiplot' + NL
end

// guardar script
fd = mopen('/tmp/proy_plot.gnu', 'w')
mfprintf(fd, '%s', gp)
mclose(fd)

// llamar gnuplot
ret = unix('gnuplot /tmp/proy_plot.gnu
   2>/tmp/proy_gnuplot_err.txt')
if ret ~= 0 then
    printf('Aviso gnuplot: codigo %d\n', ret)
end
```

El archivo de datos para gnuplot se escribe con `mfprintf`:

```
    // escribir datos separados para gnuplot
    fd = mopen('/tmp/proy_tray.txt', 'w')
    for i = 1:length(R.t)
        mfprintf(fd, '%.6f %.6f %.6f %.6f %.6f\n', ..
                 R.t(i), R.x(i), R.y(i), R.vx(i), R.vy(i))
    end
    mclose(fd)
```

Conversión de imagen a base64

El PNG generado por gnuplot se convierte a base64 usando el Josefsson 2006 comando del sistema y se incrusta en el HTML:

```
// agregar imagen si gnuplot la genero
if isfile(GRAPH_FILE) then
    B64_FILE = '/tmp/proy_graph.b64'
    unix('base64 -w 0 ' + GRAPH_FILE + ' > ' + B64_FILE)
    b64 = mgetl(B64_FILE)
    if length(b64) > 0 then
        res = res + '<img src=' + b64 + '
           alt=grafica>'
    end
end
```

La opción `-w 0` de `base64` produce una sola linea sin saltos, lo que permite leer el resultado directamente con `mgetl` como un scalar string.

Alternativa vectorial: SVG

El PNG incrustado es una imagen de mapa de bits: se ve igual en cualquier navegador pero se pixela al ampliar y no permite interacción. Gnuplot también genera SVG World Wide Web Consortium 2026, que es texto XML y se incrusta directamente en el HTML sin codificar en base64. Un SVG escala sin pérdida, se puede consultar con el navegador y sus elementos aceptan eventos, de modo que una curva puede responder al paso del puntero. El costo es un archivo más grande cuando la figura tiene muchos puntos.

Limitaciones conocidas

- El servidor atiende una solicitud a la vez. Por esa razón puede utilizar nombres fijos como `/tmp/params.txt` y `/tmp/resultado.html`: mientras se procesa una solicitud no hay otra escribiendo esos archivos. Si el servidor se reemplaza por una versión concurrente, con hilos, procesos o atención simultánea de varias solicitudes, cada solicitud debe usar archivos o un directorio temporal propio para evitar que los datos y resultados de distintos usuarios se mezclen o se sobrescriban.
- Para uso multiusuario simultáneo sería necesario un servidor con hilos o procesos y aislamiento por solicitud.

- Scilab-cli no soporta la opción `-nw` en esta version; no se debe incluir esa opción al llamarlo.

- Las funciones gráficas nativas de Scilab (`scf`, `driver`) no estan disponibles en modo CLI sin display. Se usa gnuplot como alternativa.

- Los strings de Scilab no deben mezclar comillas simples y dobles en el mismo contexto de concatenacion. Usar `ascii(34)` para obtener la comilla doble.

- El tiempo de respuesta incluye el tiempo de arranque de Scilab (típicamente 2-4 segundos). Para aplicaciones de alta frecuencia conviene mantener Scilab en memoria.

Medición del tiempo de respuesta

Cuando una aplicación combina un servidor, un proceso externo y un método numérico, no existe un "tiempo del programa" único. Conviene distinguir al menos dos mediciones:

Tiempo directo del motor: desde que se inicia el programa de cálculo hasta que termina para una entrada dada. Incluye el arranque del intérprete cuando el motor se ejecuta como un proceso nuevo.

Tiempo total de la solicitud: desde que el cliente envía la solicitud HTTP hasta que recibe la respuesta completa. Además del motor incluye el trabajo del servidor, la creación del proceso, la lectura y escritura de archivos y la transferencia local.

De manera esquemática puede pensarse

$$T_{\text{HTTP}} \simeq T_{\text{servidor}} + T_{\text{arranque}} + T_{\text{calculo}} + T_{\text{E/S}},$$

pero esos términos no se obtienen exactamente restando dos mediciones independientes. La resta entre el tiempo HTTP y el tiempo directo del motor es solamente una estimación del costo adicional de la cadena completa.

Para comparar motores o algoritmos las condiciones deben mantenerse iguales: misma entrada, misma máquina y, en lo posible, poca carga externa. Una sola medición puede ser engañosa; se deben realizar varias repeticiones y resumir los resultados mediante la mediana o el promedio, informando también la variabilidad cuando sea apreciable. La primera ejecución puede diferir de las siguientes por carga de bibliotecas, cachés o inicialización; conviene registrarla separadamente si el objetivo es estudiar ese costo.

En un lenguaje compilado como C, el tiempo de compilación no forma parte del tiempo de respuesta del motor ya instalado: el ejecutable se compila antes de la prueba. En cambio, si el sistema real crea un intérprete nuevo en cada solicitud, ese tiempo de arranque sí pertenece a la latencia observada por el usuario. Para comparar exclusivamente dos métodos numéricos dentro del mismo motor debe medirse el tramo de cálculo en condiciones equivalentes, separando en lo posible el costo fijo de inicialización.

Uso de los métodos de EDO en los distintos motores

Los métodos numéricos para ecuaciones diferenciales ordinarias ya fueron estudiados en un trabajo práctico anterior. En esta unidad no se vuelve a desarrollar su teoría: interesa aprender cómo expresar el mismo problema en cada motor y cuándo una función provista por el lenguaje representa, o no, el método solicitado en la consigna.

Hay que distinguir dos situaciones:

Método indicado explícitamente: si la consigna pide Euler o el Runge–Kutta clásico de orden 4 (RK4), y en particular si pide compararlos con el mismo paso h, se implementan las fórmulas del método ya estudiadas. Un integrador adaptativo de biblioteca no es el mismo experimento numérico.

Integración mediante una función del lenguaje: si la consigna pide simplemente resolver la EDO, o utilizar un integrador adaptativo, puede emplearse el integrador disponible en el entorno. En ese caso el paso interno puede variar y el usuario controla normalmente tolerancias en lugar de imponer un h fijo.

Para los motores utilizados en esta unidad la correspondencia es la siguiente:

Scilab. La función `ode` recibe la condición inicial `y0`, el tiempo inicial `t0`, los tiempos de salida `t` y la función `f`. Si se selecciona el tipo `rk`, utiliza un Runge–Kutta de orden 4 adaptativo. La función del modelo tiene la forma `f(t,y)` y la solución se devuelve por columnas para los tiempos solicitados. Si la consigna exige RK4 de paso fijo, se implementa el paso RK4 directamente.

Octave. La función `ode45` recibe la función del modelo, el intervalo de integración y la condición inicial. Resuelve problemas no rígidos mediante un Runge–Kutta Dormand–Prince de paso variable. La función del modelo tiene la forma `f(t,y)` y se obtienen los vectores `t` y `y`. Aunque su nombre contiene "45", no es el RK4 clásico de paso fijo. Por lo tanto, si se pide comparar Euler y RK4 con el mismo h, se implementa RK4 en lugar de sustituirlo por `ode45`. Octave dispone también de `lsode`, con otra interfaz y métodos Adams/BDF, pero tampoco representa el RK4 fijo pedido en esa comparación.

Python. Con SciPy puede usarse `scipy.integrate.solve_ivp`. La llamada con el método `RK45` utiliza un Dormand–Prince adaptativo de orden 5(4). La función del modelo es `f(t,y)`; los tiempos quedan en `sol.t` y los estados en `sol.y`. Si la consigna pide el RK4 clásico de paso fijo, se implementan sus cuatro etapas directamente.

Perl. El núcleo del lenguaje no proporciona un integrador de EDO. Para estos ejercicios se implementan directamente Euler y RK4. Pueden existir módulos externos, pero no forman parte del entorno mínimo supuesto por el práctico.

JavaScript/Node.js. El entorno estándar no proporciona un integrador de EDO. Los pasos de Euler o RK4 se implementan directamente usando números y arreglos para representar el estado.

C. La biblioteca estándar de C no incluye un integrador de EDO, por lo que el método se implementa directamente. Bibliotecas externas como GSL ofrecen integradores Runge–Kutta, pero no son necesarias para los ejercicios de esta unidad.

Bash. Bash no dispone de aritmética real adecuada ni de un integrador de EDO. El bucle numérico se delega normalmente a una sola ejecución de `awk`, dentro de la cual se implementa Euler o RK4.

PowerShell. No hay un integrador de EDO estándar. Los pasos se implementan directamente utilizando valores reales y arreglos para el estado.

En cualquier lenguaje conviene separar la función que representa el modelo, $f(t, y)$, de la rutina que avanza la solución. Una ecuación de orden mayor se escribe primero como un sistema de primer orden, tal como se hizo en los prácticos anteriores. De este modo la misma función del modelo puede usarse con Euler, RK4 o un integrador de biblioteca sin modificar la interfaz web o gráfica.

La opción "ODE adaptativo" del formulario debe interpretarse de acuerdo con el motor. En Scilab, Octave y Python existe un integrador adaptativo disponible directamente. En los demás motores esa opción exige implementar un método adaptativo o incorporar una biblioteca adicional; no debe cambiarse silenciosamente por RK4 de paso fijo.

Diagnostico de errores

En caso de error, los archivos de log son:

Archivo	Contenido
`/tmp/error_motor.txt`	Salida del motor ejecutado
`/tmp/params.txt`	Parámetros enviados al motor
`/tmp/proy_gnuplot_err.txt`	Errores de gnuplot

Flujo de diagnóstico recomendado:

```
# 1. Ver que parametros recibio el motor
cat /tmp/params.txt

# 2. Ver log del ultimo motor ejecutado
cat /tmp/error_motor.txt

# 3. Probar el servidor con un motor concreto
MOTOR=scilab python3 servidor.py

# 4. Ver errores de gnuplot
cat /tmp/proy_gnuplot_err.txt

# 5. Verificar que se genero el resultado
cat /tmp/resultado.html | head -5
ls -lh /tmp/proy_graph.png
```

Suite de tests del servidor

El proyecto incluye una suite de tests en `tests/` que cubre el servidor y todos los motores. Se corre completa con:

```
#!/bin/bash
# run_tests.sh  -  corre la suite de tests del servidor servguiweb
#
# uso:  bash servguiweb/tests/run_tests.sh
#
# Cada motor usa el comando del host si existe; si no, se genera un
# wrapper en tests/bin que llama a la imagen podman correspondiente
# (localhost/scilab-cli, localhost/octave-pkgs, etc.). Si no hay ni
# comando ni imagen, el test de ese motor se salta (SKIP).

set -u

DIR="$(cd "$(dirname "$0")" && pwd)"
ROOT="$(cd "$DIR/../.." && pwd)"
BIN="$DIR/bin"
cd "$ROOT"

# motor/comando -> imagen podman
imagen_de() {
    case "$1" in
        scilab-cli) echo localhost/scilab-cli ;;
        octave-cli) echo localhost/octave-pkgs ;;
        python3)    echo localhost/python-numpy ;;
        perl)       echo localhost/perl-pdl ;;
        node)       echo localhost/node-mathjs ;;
        bash)       echo localhost/bash-tools ;;
        *)          echo '' ;;
    esac
}

mkdir -p "$BIN"
WRAPPERS=0
for cmd in scilab-cli octave-cli python3 perl node bash; do
    if ! command -v "$cmd" > /dev/null 2>&1; then
        img=$(imagen_de "$cmd")
        if [ -n "$img" ] && podman image exists "$img"
           2>/dev/null; then
            cat > "$BIN/$cmd" <<EOF
#!/bin/bash
exec podman run --rm -e PARAM_FILE -e RESULT_FILE \
    -e VERDICT_FILE -e SESION_FILE -e SESIONES_DIR \
    -v /tmp:/tmp -v "\$PWD:\$PWD" -w "\$PWD" \
    $img $cmd "\$@"
```

```
EOF
            chmod +x "$BIN/$cmd"
            echo "wrapper podman: $cmd -> $img"
            WRAPPERS=1
        fi
    fi
done
[ "$WRAPPERS" = 1 ] && export PATH="$BIN:$PATH"

PASS=0; FAIL=0
correr() {
    local desc="$1"; shift
    echo "=== $desc"
    if "$@"; then
        PASS=$((PASS+1))
    else
        FAIL=$((FAIL+1)); echo "*** FALLO: $desc"
    fi
}

correr 'unit: funciones de servidor.py' \
    python3 "$DIR/unit/test_servidor_unit.py"

correr 'unit: sesiones, veredictos y destinos' \
    python3 "$DIR/unit/test_sesiones_unit.py"

correr 'http: todos los motores' \
    python3 "$DIR/http/test_motores_http.py"

correr 'http: caminos de error' \
    python3 "$DIR/http/test_errores_http.py"

correr 'http: flujos, sesiones y subida de archivos' \
    python3 "$DIR/http/test_flujos_http.py"

correr 'ejemplos del paquete y .deb' \
    python3 "$DIR/ejemplos/test_ejemplos_paquete.py"

echo
echo "==================================="
echo "grupos OK: $PASS   con fallos: $FAIL"
[ "$FAIL" = 0 ]
```

La suite tiene cuatro grupos:

- **Unitarios** (`unit/`): las funciones de `servidor.py` sin levantar HTTP — escritura del archivo de parámetros `clave=valor`, verificación de archivos, motores disponibles y todos los caminos de error de `llamar_motor` (motor desconocido, archivo faltante, código de salida distinto de cero, motor no instalado, falta de `RESULT_FILE`, escape de HTML).

- **Integración por motor** (`http/`): levanta el servidor y envía los mismos parámetros (`v0=80`, `angulo=30`) a cada motor disponible; todos deben responder `Tiempo de subida: 4.082 s`. También verifica la selección de motor por campo `motor` del formulario y por la variable de entorno `MOTOR`.

- **Caminos de error** (`http/`): aplicación sin archivos, motor desconocido, banner de arranque y que el servidor escuche solo en `127.0.0.1`.

- **Ejemplos del paquete** (`ejemplos/`): proyectil y estadística end-to-end con Scilab y Gnuplot (incluye el valor analítico exacto del alcance sin resistencia, 565,8 m), y que el contenido del `.deb` coincida con el árbol del proyecto.

Los motores de prueba son los ejemplos mínimos de `08/Motores/` (fuente única, no se duplican). Si un comando no está instalado en el host, `run_tests.sh` genera un wrapper que usa la imagen Podman correspondiente (`localhost/scilab-cli`, `octave-pkgs`, `python-numpy`, `perl-pdl`, `node-mathjs`, `bash-tools`); si tampoco hay imagen, ese motor se salta (`SKIP`). El motor PowerShell requiere `pwsh`.

8.3.6. Adaptar el sistema a otra aplicación

Para crear una nueva aplicación de cálculo científico sobre este sistema:

1. Copiar los tres archivos a un nuevo directorio.

2. Editar `formulario.html`:

   - Cambiar el título y la descripción.
   - Agregar, quitar o modificar los campos del formulario.
   - Mantener `method=POST action=/` en el formulario.
   - Mantener el marcador `<!-- RESULTADOS -->`.
   - Recordar agregar campos ocultos para cada checkbox.

3. Implementar o modificar el archivo del motor elegido (`calcular.sce`, `calcular.m`, `calcular.py`, etc.):

   - Leer los parámetros desde la ruta indicada por `PARAM_FILE`.
   - Implementar el cálculo científico.
   - Generar gráficas si corresponde.
   - Escribir el fragmento HTML en la ruta indicada por `RESULT_FILE`.
   - Devolver código de salida cero si terminó correctamente.

4. No modificar `servidor.py`; si se agrega un lenguaje nuevo, solamente se incorpora su comando al diccionario `MOTORES`.

Ejemplo: aplicación de estadistica

Un ejemplo alternativo que calcula estadísticas de un vector de datos ingresados por el usuario:

Listing 8.16: formulario.html de estadistica (fragmento)

```
<form method=POST action=/>

  <label>Datos (separados por comas)</label>
  <textarea name=datos
     rows=4>2.3,4.1,3.8,5.2,4.7,3.1,6.0,2.9,4.5,5.8,3.6,4.9</textarea>

  <div class=row>
    <div>
      <label>Numero de clases del histograma</label>
      <input type=number name=nclases value=8 min=2 max=30
         step=1>
    </div>
    <div>
      <label>Nivel de confianza</label>
      <select name=confianza>
        <option value=90>90%</option>
        <option value=95 selected>95%</option>
        <option value=99>99%</option>
      </select>
    </div>
  </div>

  <label>Graficas</label>
  <input type=hidden name=g_hist   value=off>
  <label><input type=checkbox name=g_hist   value=on checked>
     Histograma</label>
  <input type=hidden name=g_normal value=off>
  <label><input type=checkbox name=g_normal value=on checked>
     Curva normal superpuesta</label>
  <input type=hidden name=g_acum   value=off>
  <label><input type=checkbox name=g_acum   value=on> Frecuencia
     acumulada</label>

  <label>Titulo del grafico</label>
  <input type=text name=titulo value="Distribucion de datos">

  <label>Notas</label>
  <textarea name=notas rows=2></textarea>

  <button type=submit>Calcular</button>
</form>
```

Listing 8.17: calcular.sce de estadistica (fragmento)

```
// ----- leer parametros (protocolo neutro clave=valor) -----
```

```
lineas  = mgetl(PARAM_FILE)
claves  = list()
valores = list()
for i = 1:size(lineas, '*')
    p = strindex(lineas(i), '=')
    if ~isempty(p) then
        claves($+1)  = part(lineas(i), 1:p(1)-1)
        valores($+1) = part(lineas(i), p(1)+1:length(lineas(i)))
    end
end

function v = pval(k, def)
    v = def
    for i = 1:length(claves)
        if claves(i) == k then v = valores(i); end
    end
endfunction

function b = pbool(k)
    b = pval(k, 'off') == 'on'
endfunction

function x = pnum(k, def)
    x = strtod(pval(k, string(def)))
    if isnan(x) then x = def; end
endfunction

// datos: lista de numeros separados por coma
ds     = pval('datos', '')
partes = strsplit(ds, ',')
v = []
for i = 1:size(partes, '*')
    x = strtod(partes(i))
    if ~isnan(x) then v($+1) = x; end
end
n = length(v)

nclases   = pnum('nclases', 8)
confianza = pval('confianza', '95')
g_hist    = pbool('g_hist')
g_normal  = pbool('g_normal')
g_acum    = pbool('g_acum')
titulo    = pval('titulo', 'Distribucion de datos')

if n < 2 then
    escribir_error('Se necesitan al menos 2 datos numericos
       validos.')
    quit
end
```

```
// estadisticas
media    = mean(v)
mediana  = median(v)
sigma    = stdev(v)
minv     = min(v)
maxv     = max(v)
rango    = maxv - minv
cv       = sigma / abs(media) * 100
err_std  = sigma / sqrt(n)
sesgo    = mean((v - media).^3) / sigma^3
curtosis = mean((v - media).^4) / sigma^4 - 3
v_ord    = gsort(v, 'g', 'i')
Q1       = v_ord(max(1, int(n*0.25)+1))
Q3       = v_ord(min(n, int(n*0.75)+1))
IQR      = Q3 - Q1

conf_s = confianza
select conf_s
case '90' then z = 1.645
case '99' then z = 2.576
else           z = 1.960
end
ic_inf = media - z * err_std
ic_sup = media + z * err_std
```

8.3.7. Código fuente del ejemplo: simulador de proyectil

Este es el ejemplo completo que se desarrolló y probó durante la elaboracion de este sistema. Incluye todos los tipos de controles HTML y una simulación numérica con gráficas.

formulario.html

Listing 8.18: formulario.html

```
<!DOCTYPE html>
<html lang=es>
<head>
<meta charset=UTF-8>
<title>Simulador de proyectil</title>
<style>
body{font-family:sans-serif;max-width:720px;margin:20px
   auto;padding:0 16px}
label{display:block;font-weight:bold;margin:10px 0 3px}
input,select,textarea{width:100%;padding:6px;box-sizing:border-box;
  border:1px solid #ccc;border-radius:4px}
.row{display:flex;gap:12px}
.row > div{flex:1}
button{margin-top:14px;padding:10px 24px;background:#2980b9;
  color:white;border:none;border-radius:4px;font-size:15px;cursor:pointer}
```

```
#resultados{margin-top:24px}
.stats{display:flex;gap:10px;flex-wrap:wrap;margin:12px 0}
.sb{flex:1;min-width:110px;background:#ecf0f1;padding:10px;
  border-radius:6px;text-align:center}
.sv{font-size:1.4em;font-weight:bold}
img{max-width:100%;border:1px solid
   #ccc;border-radius:6px;margin-top:8px}
</style>
</head>
<body>

<h2>Simulador de proyectil</h2>
<p>Complete los parametros y presione Calcular.</p>

<form method=POST action=/>

  <label>Nombre del experimentador</label>
  <input type=text name=nombre value="">

  <div class=row>
    <div>
      <label>Velocidad inicial [m/s]</label>
      <input type=number name=v0 value=50 min=1 max=500 step=1>
    </div>
    <div>
      <label>Angulo de lanzamiento [grados]</label>
      <input type=number name=angulo value=45 min=1 max=89
         step=1>
    </div>
  </div>

  <label>Resistencia del aire</label>
  <select name=arr>
    <option value=cero>Sin resistencia</option>
    <option value=bajo selected>Baja  (b=0.08)</option>
    <option value=medio>Media (b=0.30)</option>
    <option value=alto>Alta  (b=1.50)</option>
  </select>

  <label>Metodo numerico</label>
  <label><input type=radio name=met value=euler> Euler</label>
  <label><input type=radio name=met value=rk4 checked>
     Runge-Kutta 4</label>
  <label><input type=radio name=met value=ode> ODE
     adaptativo</label>

  <label>Graficas a mostrar</label>
  <input type=hidden name=g_tray value=off>
  <label><input type=checkbox name=g_tray value=on checked>
     Trayectoria</label>
  <input type=hidden name=g_vel value=off>
```

```
    <label><input type=checkbox name=g_vel value=on> Velocidad vs
       tiempo</label>

    <label>Notas del experimento</label>
    <textarea name=notas rows=3></textarea>

    <button type=submit>Calcular</button>

  </form>

  <!-- RESULTADOS -->

  </body>
  </html>
```

calcular.sce

Listing 8.19: calcular.sce

```
// calcular.sce  -  simulacion de proyectil (motor Scilab)
// Lee PARAM_FILE (lineas clave=valor, escrito por el servidor)
// Escribe RESULT_FILE (fragmento HTML) y archivos auxiliares en
   /tmp

PARAM_FILE  = getenv('PARAM_FILE')
RESULT_FILE = getenv('RESULT_FILE')
DATA_FILE   = '/tmp/proy_data.txt'
GRAPH_FILE  = '/tmp/proy_graph.png'

function h = box(val, lab)
    h = '<div class=sb><div class=sv>' + val + '</div>' + lab +
       '</div>'
endfunction

function escribir_error(msg)
    fd = mopen(RESULT_FILE, 'w')
    mfprintf(fd, '<p style=color:red><b>Error:</b> %s</p>', msg)
    mclose(fd)
endfunction

// ----- leer parametros (protocolo neutro clave=valor) -----

lineas  = mgetl(PARAM_FILE)
claves  = list()
valores = list()
for i = 1:size(lineas, '*')
    p = strindex(lineas(i), '=')
    if ~isempty(p) then
        claves($+1)  = part(lineas(i), 1:p(1)-1)
        valores($+1) = part(lineas(i), p(1)+1:length(lineas(i)))
```

```
    end
end

function v = pval(k, def)
    v = def
    for i = 1:length(claves)
        if claves(i) == k then v = valores(i); end
    end
endfunction

function b = pbool(k)
    b = pval(k, 'off') == 'on'
endfunction

function x = pnum(k, def)
    x = strtod(pval(k, string(def)))
    if isnan(x) then x = def; end
endfunction

v0     = pnum('v0', 50)
angulo = pnum('angulo', 45)
arr    = pval('arr', 'cero')
met    = pval('met', 'rk4')
g_tray = pbool('g_tray')
g_vel  = pbool('g_vel')

if v0     <= 0 then v0     = 50; end
if angulo <= 0 then angulo = 45; end

// no arrastrar una grafica de una corrida anterior
if isfile(GRAPH_FILE) then mdelete(GRAPH_FILE); end

select arr
case 'cero'  then b = 0
case 'bajo'  then b = 0.08
case 'medio' then b = 0.30
case 'alto'  then b = 1.50
else              b = 0
end

// ----- simulacion -----

function R = simular(v0, ang_deg, b, metodo)
    g   = 9.8
    m   = 1.0
    a   = ang_deg * %pi / 180
    y0  = [0; 0; v0*cos(a); v0*sin(a)]

    function dydt = fode(t, y)
        vmod = sqrt(y(3)^2 + y(4)^2)
        fd   = (b/m) * vmod
```

```
        dydt = [y(3); y(4); -fd*y(3); -g - fd*y(4)]
    endfunction

    t_max = 3 * v0 * sin(a) / g + 1
    n     = 400
    t     = linspace(0, t_max, n)

    select metodo
    case 'euler' then
        Y = zeros(4, n); Y(:,1) = y0; dt = t(2) - t(1)
        for i = 1:n-1
            Y(:,i+1) = Y(:,i) + dt * fode(t(i), Y(:,i))
            if Y(2,i+1) < -0.1 then break, end
        end
    case 'rk4' then
        Y = zeros(4, n); Y(:,1) = y0; dt = t(2) - t(1)
        for i = 1:n-1
            k1 = fode(t(i),      Y(:,i))
            k2 = fode(t(i)+dt/2, Y(:,i) + dt/2*k1)
            k3 = fode(t(i)+dt/2, Y(:,i) + dt/2*k2)
            k4 = fode(t(i)+dt,   Y(:,i) + dt*k3)
            Y(:,i+1) = Y(:,i) + dt/6*(k1 + 2*k2 + 2*k3 + k4)
            if Y(2,i+1) < -0.1 then break, end
        end
    else
        sol = ode(y0, 0, t, fode)
        Y   = sol
    end

    idx = find(Y(2,:) < 0)
    n2  = n
    if ~isempty(idx) then n2 = idx(1); end

    R.t  = t(1:n2);
    R.x  = Y(1,1:n2);  R.y  = Y(2,1:n2)
    R.vx = Y(3,1:n2);  R.vy = Y(4,1:n2)
    R.v  = sqrt(R.vx.^2 + R.vy.^2)
endfunction

ierr = execstr('R = simular(v0, angulo, b, met)', 'errcatch')
if ierr ~= 0 then
    escribir_error('simulacion: ' + lasterror())
    quit
end

// ----- grafica con gnuplot -----

if g_tray | g_vel then

    // escribir datos separados para gnuplot
    fd = mopen('/tmp/proy_tray.txt', 'w')
```

```
for i = 1:length(R.t)
    mfprintf(fd, '%.6f %.6f %.6f %.6f %.6f\n', ..
             R.t(i), R.x(i), R.y(i), R.vx(i), R.vy(i))
end
mclose(fd)

// construir script gnuplot  (sin comillas dobles en strings
   Scilab)
NL = ascii(10)
Q  = ascii(34)

gp = 'set terminal png size 800,400' + NL
gp = gp + 'set output ' + Q + GRAPH_FILE + Q + NL
gp = gp + 'set grid' + NL

if g_tray & g_vel then
    gp = gp + 'set multiplot layout 1,2' + NL
end

DAT = Q + '/tmp/proy_tray.txt' + Q

if g_tray then
    gp = gp + 'set xlabel ' + Q + 'x [m]'   + Q + NL
    gp = gp + 'set ylabel ' + Q + 'y [m]'   + Q + NL
    gp = gp + 'set title '  + Q + 'Trayectoria' + Q + NL
    gp = gp + 'plot ' + DAT + ' using 2:3 with lines lw 2 lc
       ' + Q + 'blue' + Q + ' notitle' + NL
end

if g_vel then
    gp = gp + 'set xlabel ' + Q + 't [s]'   + Q + NL
    gp = gp + 'set ylabel ' + Q + 'v [m/s]' + Q + NL
    gp = gp + 'set title '  + Q + 'Velocidad' + Q + NL
    gp = gp + 'plot ' + DAT + ' using 1:(sqrt($4*$4+$5*$5))
       with lines lw 2 lc ' + Q + 'blue'  + Q + ' title ' +
       Q + '|v|' + Q + ', \' + NL
    gp = gp + '     ' + DAT + ' using 1:4 with lines lw 1 lc
       '                    + Q + 'red'   + Q + ' title ' + Q
       + 'vx'  + Q + ', \' + NL
    gp = gp + '     ' + DAT + ' using 1:5 with lines lw 1 lc
       '                    + Q + 'green' + Q + ' title ' + Q
       + 'vy'  + Q + NL
end

if g_tray & g_vel then
    gp = gp + 'unset multiplot' + NL
end

// guardar script
fd = mopen('/tmp/proy_plot.gnu', 'w')
mfprintf(fd, '%s', gp)
```

```
    mclose(fd)

    // llamar gnuplot
    ret = unix('gnuplot /tmp/proy_plot.gnu
       2>/tmp/proy_gnuplot_err.txt')
    if ret ~= 0 then
        printf('Aviso gnuplot: codigo %d\n', ret)
    end

end

// ----- resultado HTML completo (numeros + imagen) -----

res = '<h2>Resultados</h2><div class=stats>'
res = res + box(sprintf('%.1f m',   R.x($)),   'Alcance')
res = res + box(sprintf('%.1f m',   max(R.y)), 'Altura max')
res = res + box(sprintf('%.2f s',   R.t($)),   'Tiempo vuelo')
res = res + box(sprintf('%.1f m/s', R.v($)),   'V impacto')
res = res + '</div>'
res = res + '<p>v0=' + sprintf('%.0f', v0)  + ' m/s  '
res = res + 'angulo=' + sprintf('%.0f', angulo) + ' grados  '
res = res + 'b=' + sprintf('%.2f', b) + '  metodo=' + met +
   '</p>'

// agregar imagen si gnuplot la genero
if isfile(GRAPH_FILE) then
    B64_FILE = '/tmp/proy_graph.b64'
    unix('base64 -w 0 ' + GRAPH_FILE + ' > ' + B64_FILE)
    b64 = mgetl(B64_FILE)
    if length(b64) > 0 then
        res = res + '<img src=' + b64 + '
           alt=grafica>'
    end
end

fd = mopen(RESULT_FILE, 'w')
mfprintf(fd, '%s', res)
mclose(fd)

quit
```

8.4. Interfaz nativa de Scilab

El método nativo de Scilab, es una interfaz gráfica formada por una ventana principal y un conjunto de controles denominados *widgets*, con funciones denominadas *UIcontrol* Scilab Enterprises 2026c. Entre los más utilizados se encuentran:

- botones (`pushbutton`);

- cuadros de texto (`edit`);
- etiquetas (`text`);
- casillas de selección (`checkbox`);
- listas desplegables (`popupmenu`);
- deslizadores (`slider`);
- ejes gráficos para representar resultados.

Una **GUI** (*Graphical User Interface*) presenta la información mediante elementos visuales (ventanas, botones, menús, controles deslizantes) y recibe las instrucciones del usuario principalmente mediante el ratón y el teclado.

El cambio respecto de la terminal no es cosmético. Shneiderman lo llamó *manipulación directa*: en lugar de escribir el nombre de una operación y sus argumentos, el usuario actúa sobre una representación visible del objeto y ve el efecto de inmediato Shneiderman 1983. Hutchins, Hollan y Norman analizaron después por qué eso se siente distinto, en términos de la distancia entre lo que el usuario quiere hacer y lo que la interfaz le ofrece Hutchins, Hollan y Norman 1985. La ventana, el widget y el evento tal como los usamos hoy son el resultado de décadas de investigación anteriores a su difusión Myers 1998.

El modelo de programación cambia fundamentalmente respecto a la terminal: en lugar de un flujo lineal de instrucciones, el programa define una colección de **elementos gráficos** y las **acciones** que se ejecutan cuando el usuario interactúa con ellos. El programa luego espera eventos y responde a medida que llegan.

Que la interfaz sea gráfica no la hace buena. Que un control se entienda depende de que su apariencia sugiera lo que hace y de que el sistema devuelva realimentación visible Norman 2013, y hay criterios verificables para revisar una ventana ya construida Shneiderman et al. 2016. Una ventana con doce botones sin rótulo es peor interfaz que una línea de comandos documentada.

Los elementos fundamentales de una GUI son:

Ventana (*window*): contenedor principal de la interfaz. Puede contener otros elementos.

Control o widget: elemento interactivo dentro de una ventana. Los más comunes son:

Botón (*button*): ejecuta una acción al ser presionado.

Etiqueta (*label*): muestra texto no interactivo.

Campo de texto (*text field*): permite al usuario ingresar texto.

Control deslizante (*slider*): permite seleccionar un valor numérico dentro de un rango.

Lista (*listbox*): permite seleccionar una o varias opciones de una lista.

Casilla de verificación (*checkbox*): representa un valor booleano.

Botón de opción (*radio button*): selección exclusiva entre varias opciones.

Menú: lista desplegable de opciones.

Barra de progreso: indica el avance de una tarea.

Área de dibujo o lienzo (*canvas*): superficie donde el programa puede dibujar gráficos y animaciones.

```
// Crear una GUI minima en Scilab
// Los elementos se crean con uicontrol()

f = figure('Figure_name', 'Mi primera GUI', ...
           'position', [200, 200, 400, 300], ...
           'BackgroundColor', [0.95, 0.95, 0.95])

// Etiqueta
uicontrol('style', 'text', ...
          'string', 'Ingrese un valor:', ...
          'position', [20, 230, 150, 25], ...
          'BackgroundColor', [0.95, 0.95, 0.95])

// Campo de texto para entrada
h_entrada = uicontrol('style', 'edit', ...
                      'string', '0', ...
                      'position', [180, 230, 100, 25])

// Boton que ejecuta una accion
uicontrol('style', 'pushbutton', ...
          'string', 'Calcular raiz', ...
          'position', [100, 170, 150, 35], ...
          'callback', 'calcular_raiz()')

// Etiqueta para mostrar el resultado
uicontrol('style', 'text', ...
          'tag', 'resultado', ...
          'string', 'Resultado: ---', ...
          'position', [20, 120, 350, 30], ...
          'fontsize', 12, ...
          'BackgroundColor', [0.95, 0.95, 0.95])

// La funcion que se ejecuta al presionar el boton
function calcular_raiz()
    valor = strtod(get(findobj('style', 'edit'), 'string'))
    if valor < 0 then
        set(findobj('tag', 'resultado'), ...
            'string', 'Error: numero negativo')
    else
        raiz = sqrt(valor)
        set(findobj('tag', 'resultado'), ...
            'string', sprintf('Resultado: sqrt(%.4f) = %.4f', ...
                              valor, raiz))
    end
endfunction
```

Para el primer contacto con el entorno, la sintaxis de funciones y los gráficos, conviene tener a mano un texto introductorio Baudin y Kroeter 2011; Gomez 2003. El tutorial de Openeering desarrolla una GUI completa de Scilab sobre un modelo dinámico y sirve como ejemplo extenso de todo lo que sigue Venturin 2013.

8.4.1. Uso de diálogos predefinidos

No toda interacción justifica construir una ventana. Scilab trae diálogos ya armados que resuelven los casos frecuentes Scilab Enterprises 2026c:

`x_dialog`**:** pide uno o varios valores de texto y devuelve lo ingresado.

`input_dialog`**:** variante para un único valor, con etiqueta y valor por omisión.

`listdialog`**:** presenta una lista y devuelve los índices de lo seleccionado.

`messagebox`**:** informa, advierte o pide confirmación, y devuelve qué botón se apretó.

`uigetfile` **y** `uiputfile`**:** selectores de archivo para abrir y para guardar.

`progressionbar` **y** `waitbar`**:** muestran avance durante un cálculo largo.

La diferencia con los controles de la sección anterior es de control del flujo. El diálogo predefinido es *modal* y bloqueante: el programa se detiene, espera la respuesta y sigue con el valor devuelto, igual que un `input()` en la terminal. Un `uicontrol`, en cambio, no interrumpe nada y sólo dispara su callback cuando el usuario decide usarlo.

Por eso el diálogo predefinido encaja bien en un guión secuencial que necesita un dato puntual —el nombre del archivo, el número de pasos—, y los widgets encajan en un programa que debe quedar disponible mientras el usuario prueba valores.

8.4.2. Paso de parámetros

Un programa con interfaz gráfica tiene un problema que el programa secuencial no tiene: los datos no llegan todos juntos al arrancar sino de a poco, en cualquier orden y en momentos que el programa no elige. Hay que decidir dónde vive el estado entre un evento y el siguiente. En Scilab hay tres caminos Scilab Enterprises 2026c:

En el propio widget. El valor se lee cuando hace falta con `get(h,'string')` o `get(h,'value')` y se escribe con `set`. La interfaz *es* el estado. Es lo que hace el ejemplo de la sección de UI Controls, donde el nombre del archivo se lee del campo de edición recién en el momento de guardar.

En variables del entorno. Declaradas `global`, o capturadas por la función manejadora. Es cómodo y frágil: cualquier `clear` las borra y no queda registro de quién las modificó.

Adosado al objeto gráfico. El campo `userdata` de una figura o de un control guarda cualquier valor de Scilab —una matriz, una lista, una estructura completa—, de modo que el dato viaja con el objeto al que pertenece y desaparece con él.

Hay además una cuarta vía que aparece apenas el callback necesita argumentos: la cadena de texto que se pasa en `'callback'` se evalúa en el momento del evento, así que `'f3(x,y)'` toma los valores que `x` e `y` tengan entonces, no los que tenían cuando se creó el botón. Es una fuente clásica de errores difíciles de encontrar, y la razón por la que conviene que el callback lea sus datos de los controles en vez de recibirlos congelados.

Para valores que no vienen del usuario sino de la configuración del programa —dónde están los datos, qué método numérico usar— conviene sacarlos del código a un archivo aparte, como ya vimos al tratar entrada y salida.

8.4.3. UI Controls

```
f=figure(1,'position',[100 100 300 300]);
t1=uicontrol(f,'style','text',...
'position', [10 10 150 50],...
'string','Nombre del archivo');

e1=uicontrol(f,'style','edit',...
'position', [150 20 100 30],...
'string','');

e2=uicontrol(f,'style','edit',...
'position', [150 150 100 30],...
'string','');

p1=uicontrol(f,'style','pushbutton',...
'position', [20 200 100 50],...
'string','Graficar',...
'callback','exec(foo1)');
p2=uicontrol(f,'style','pushbutton',...
'position', [150 200 100 50],...
'string','Guardar',...
'callback','exec(foo2)');
ll=uicontrol(f,'style','listbox', ...
'position', [20 100 100 80],...
'string', "seno|coseno|tangente",...
'value',1);

x=0:0.05*%pi:2*%pi;
funcprot(0)
function y=foo1(x)
set("current_figure",2)
q=get(ll,'value');
select q
  case 1 then
    y=sin(x);
    plot(x,y)
    xtitle("Funcion Seno","x","sin(x)")
  case 2 then
```

```
    y=cos(x);
    plot(x,y)
    xtitle("Funcion Coseno","x","cos(x)")
  case 3 then
    y=tan(x);
    plot(x,y)
    xtitle("Funcion Tangente","x","tan(x)")
end
endfunction

function foo2(x,y,q)
select q
case 1
  funcion="seno(x)"
case 2
   funcion="coseno(x)"
 case 3
   funcion="tangente(x)"
end
p=get(e1,'string');
nom=string(p);
u = mopen(nom,'w');
mfprintf(u,'%s\t\t%s\n','x',funcion);
mfprintf(u,'%f\t%f\n',x',y');
mclose(u);
set(e2,"string",'grabado!')
endfunction


clear;             //borra todas las variables del entorno
xdel(winsid());    //borra todas las ventanas graficas (winsid
   devuelve las ventanas abiertas)
exec('f.sci');     // carga las funciones que utilizara el
   programa
exec('g.sci');
exec('f2.sci');
exec('f3.sci');
x=linspace(0,3*%pi);
ventana=figure('position',[20,20,300,250],'backgroundcolor',[0.4,0.4,0.9]);
    //crea ventana grafica  position (x,y,ancho,largo)
marco=uicontrol(ventana,
   'style','frame','position',[10,10,280,230],...
'backgroundcolor',[0.6,0.6,0.6]);
color_marco=get(marco,'backgroundcolor');  //toma el color de
   fondo de ventana

uicontrol(ventana,'style','text','fontsize',12,'position',[40,210,220,20],..
   //pone dos textos
'string','titulo1','backgroundcolor', color_marco);
uicontrol(ventana,'style','text','fontsize',12,'position',[70,180,160,20],..
'string','titulo2','backgroundcolor', color_marco);
```

```
boton1=uicontrol('style','pushbutton','position',[20,130,110,20],'background
'string','seno(y)','callback','f(x)');   //crea boton que
   ejecuta la funcion f con el parametro x 'f(x)'
boton2=uicontrol('style','pushbutton','position',[20,100,110,20],'background
'callback','g(x)');   //crea boton que ejecuta la funcion g con
   el parametro x 'g(x)'

boton3=uicontrol('style','pushbutton','position',[20,70,110,20],...
'backgroundcolor',[0.6,0.6,0.6],'String','calcula y ',...
'callback','y=f2(x)');    //crea boton que ejecuta la funcion f2
   con el parametro x 'f2(x)' y se la asigna a la variable y

boton4=uicontrol('position',[20,40,110,20],'backgroundcolor',[0.6,0.6,0.6],'
   y',...
'callback','f3(x,y)')   //crea boton que ejecuta la funcion f3
   con los parametros x e y 'f3(x,y)'
```

8.4.4. Animación

Una animación numérica representa sucesivamente los distintos estados de un sistema. Si el estado de un fenómeno físico es $\mathbf{x}(t)$, una simulación produce una sucesión

$$\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$$

correspondiente a los tiempos

$$t_0, t_1, \ldots, t_n.$$

La animación transforma cada estado numérico en un estado gráfico. No conviene borrar la figura y construirla nuevamente en cada instante: se crean los objetos gráficos una sola vez y luego se modifican sus propiedades. Las funciones gráficas y de temporización utilizadas en estos ejemplos están documentadas en la ayuda oficial de Scilab Scilab Enterprises 2026c.

El esquema general es

estado físico $\longrightarrow$ actualizar objetos gráficos $\longrightarrow$ mostrar $\longrightarrow$ siguiente estado.

Ejemplo: animación de un péndulo

Para pequeñas oscilaciones, un péndulo ideal de longitud L cumple

$$\theta(t) = \theta_0 \cos\left(\sqrt{\frac{g}{L}}\, t\right),$$

y la posición de la masa es

$$x = L\sin\theta, \qquad y = -L\cos\theta.$$

El siguiente programa calcula primero los estados y luego anima el movimiento modificando los mismos objetos gráficos.

```
// Parametros fisicos
g = 9.81
L = 1
theta0 = 40 * %pi / 180

// Estados de la simulacion
dt = 0.02
t = 0:dt:10
theta = theta0 * cos(sqrt(g/L) * t)

// Preparar la figura
clf()
a = gca()
a.data_bounds = [-1.2*L -1.2*L ; 1.2*L 0.2*L]
a.isoview = "on"

// Estado inicial
x = L * sin(theta(1))
y = -L * cos(theta(1))

// Crear la barra una sola vez
xpoly([0 x], [0 y], "lines")
barra = gce()
barra.thickness = 2

// Crear la masa una sola vez
xpoly(x, y, "marks")
masa = gce()
masa.mark_style = 9
masa.mark_size = 4

// Asociar un intervalo real a cada paso
realtimeinit(dt)
realtime(0)

for k = 1:length(t)

    x = L * sin(theta(k))
    y = -L * cos(theta(k))

    // Actualizar los objetos existentes
    barra.data = [0 0 ; x y]
    masa.data = [x y]
```

```
    realtime(k)
end
```

Aquí el cálculo físico y la representación están separados: las variables `theta`, `x` e `y` describen el estado del sistema, mientras que `barra` y `masa` son solamente su representación gráfica.

8.4.5. Procesamiento de eventos

Un **evento** es una ocurrencia en el sistema a la que el programa puede responder. En una GUI, los eventos son generados por:

- El usuario: clic del ratón, movimiento del ratón, pulsación de una tecla, desplazamiento de un slider, escritura en un campo de texto.
- El sistema: apertura o cierre de una ventana, redimensionamiento, temporizador que expira.
- La red: llegada de un mensaje, fin de una descarga.
- Sensores: nueva lectura de temperatura, señal de un dispositivo externo.

El **modelo de programación orientada a eventos** funciona así:

1. El programa crea los elementos de la interfaz.
2. Para cada elemento y tipo de evento, registra una función *manejadora* (*event handler* o *callback*).
3. El programa entra en el **bucle de eventos** (*event loop*): espera eventos, y cuando llega uno, llama a la función manejadora correspondiente.
4. La función manejadora ejecuta la lógica de respuesta y puede modificar la interfaz.
5. El bucle continúa esperando el siguiente evento.

Este esquema tiene nombre propio en el diseño de programas: registrar una función para que otro la llame cuando algo ocurra es el patrón Observador, y empaquetar una acción como un valor que se puede guardar y pasar es el patrón Comando Gamma et al. 1994. El bucle que espera, despacha y vuelve a esperar es el mismo que estructura un videojuego, con su ciclo de estado, actualización y dibujo Nystrom 2014.

La alternativa sería un hilo de ejecución por cada fuente de sucesos. Ousterhout argumentó que para la mayoría de los casos eso complica más de lo que resuelve, porque obliga a sincronizar accesos concurrentes al estado compartido Ousterhout 1996, y el mismo argumento, con datos de sistemas reales, aparece en Dabek et al. 2002. Scilab adoptó el modelo de un solo hilo con callbacks, y de ahí se sigue una consecuencia práctica: un callback que tarda mucho congela toda la interfaz, porque mientras corre nadie atiende el bucle. Una simulación larga hay que partirla en pasos o avisar el avance con `progressionbar`.

```
// Ejemplo del flujo de eventos en Scilab

// 1. Crear elemento
h_slider = uicontrol('style', 'slider', ...
                     'min', 0, 'max', 100, 'value', 50, ...
                     'position', [50, 100, 300, 25], ...
                     'callback', 'manejador_slider()')
                     // ^--- registrar el manejador

// 2. Definir el manejador del evento
function manejador_slider()
    // Esta funcion se llama CADA VEZ que el slider se mueve
    valor = get(h_slider, 'value')
    printf('El slider cambio a: %.1f\n', valor)
    // Aqui va la logica de respuesta:
    // actualizar un grafico, recalcular algo, etc.
endfunction

// 3. El bucle de eventos es manejado automaticamente
//    por Scilab mientras la figura esta abierta
```

```
// Tipos de eventos en Scilab y sus callbacks

// Evento: boton presionado
uicontrol('style', 'pushbutton', ...
          'callback', 'funcion_al_presionar()')

// Evento: slider movido
uicontrol('style', 'slider', ...
          'callback', 'funcion_al_mover()')

// Evento: texto editado
uicontrol('style', 'edit', ...
          'callback', 'funcion_al_editar()')

// Evento: item seleccionado en una lista
uicontrol('style', 'listbox', ...
          'callback', 'funcion_al_seleccionar()')

// Evento: figura cerrada
set(gcf(), 'closerequestfcn', 'funcion_al_cerrar()')

// Evento: tecla presionada (en la figura)
set(gcf(), 'event_handler', 'manejador_teclado')
set(gcf(), 'event_handler_enable', 'on')

function manejador_teclado(win, x, y, ibut)
    // Para una tecla imprimible presionada, ibut es
    // su codigo ASCII. Por ejemplo: q=113, r=114.
    if ibut == ascii('q') then
```

```
        disp('Se presiono q')
    elseif ibut == ascii('r') then
        disp('Se presiono r')
    end
endfunction
```

En los eventos de la figura, `ibut` codifica el tipo de suceso. Para teclas imprimibles presionadas toma el código ASCII de la tecla; en cambio, `-5` indica que se liberó el botón izquierdo del ratón y `-1` indica movimiento del puntero. Por eso el ejemplo anterior compara con `ascii('q')` y `ascii('r')` Scilab Enterprises 2026c.

Eventos para simular fenómenos físicos

Un evento puede modificar un parámetro del modelo y provocar un nuevo cálculo. El ciclo completo es

evento del usuario $\longrightarrow$ cambio de parámetro $\longrightarrow$ modelo físico $\longrightarrow$ actualización de la vista.

En el siguiente ejemplo un deslizador controla el amortiguamiento b de un oscilador

$$x(t) = e^{-bt}\cos(2t).$$

Cada movimiento del deslizador genera un evento. El callback lee el nuevo valor de b, recalcula la solución y modifica la curva ya existente.

```
global h_b h_estado curva t

t = linspace(0, 10, 500)
b0 = 0.20
x = exp(-b0*t) .* cos(2*t)

clf()
a = gca()
a.data_bounds = [0 -1.1 ; 10 1.1]
xgrid()

// Crear la curva una sola vez
xpoly(t, x, "lines")
curva = gce()
curva.thickness = 2

f = gcf()

h_estado = uicontrol(f, ...
    "style", "text", ...
    "string", "b = 0.20", ...
    "position", [50 45 300 20])
```

```
h_b = uicontrol(f, ...
    "style", "slider", ...
    "min", 0, "max", 1, "value", b0, ...
    "position", [50 20 300 20], ...
    "callback", "actualizar_oscilador()")

function actualizar_oscilador()
    global h_b h_estado curva t

    b = get(h_b, "value")
    x = exp(-b*t) .* cos(2*t)

    curva.data = [t' x']

    set(h_estado, "string", sprintf("b = %.2f", b))
endfunction
```

Este ejemplo distingue dos escalas de tiempo. El tiempo t pertenece al fenómeno físico simulado; el evento del deslizador ocurre cuando el usuario decide modificar b. El callback no representa un paso del tiempo físico: cambia un parámetro, vuelve a calcular el modelo y actualiza la vista.

Cómo organizar el programa

Cuando la interfaz crece, la tentación es escribir toda la lógica dentro de los callbacks. El resultado es un programa donde la simulación, el dibujo y la lectura de los controles están mezclados, no se puede probar ninguna parte por separado y cambiar un botón obliga a tocar el cálculo.

La separación clásica se conoce como **modelo, vista y control** Krasner y Pope 1988. El *modelo* es el estado del sistema simulado y las funciones que lo hacen avanzar, que son las mismas que escribimos en la unidad de cálculo numérico y no saben nada de ventanas. La *vista* son los objetos gráficos que lo muestran. El *control* son los widgets y sus callbacks. Un callback bien escrito no calcula: lee los controles, llama a la función del modelo y actualiza la vista.

La ventaja es concreta y verificable: si el modelo no sabe que existe una GUI, el mismo código sirve para correr la simulación sin interfaz en un guión por lotes, para compararla contra una solución analítica y para probarla automáticamente.

8.5. Interfaces gráficas habituales en otros lenguajes

La GUI nativa de Scilab mostrada en la sección anterior no es una característica exclusiva de Scilab. Los otros lenguajes utilizados en la materia disponen de mecanismos equivalentes, aunque no todos los incluyen en el propio lenguaje.

La estructura conceptual es siempre la misma:

ventana ⟶ control ⟶ evento ⟶ callback ⟶ cálculo ⟶ actualización de la GUI.

Las herramientas elegidas aquí son deliberadamente simples y representativas de la forma habitual de construir una interfaz en cada lenguaje. El objetivo es reconocer el mismo modelo de programación orientada a eventos con sintaxis diferentes.

Lenguaje	**GUI utilizada en este tutorial**
Scilab	`figure` y `uicontrol` (ya estudiado)
Octave	`figure` y `uicontrol`
Python	`tkinter`/`ttk`
Perl	Perl/Tk
JavaScript	DOM del navegador; Electron para escritorio
C	GTK
Bash	Zenity, como programa gráfico externo
PowerShell	Windows Forms

Gnuplot no aparece como toolkit de GUI porque su función principal es la representación gráfica. Puede formar parte de una interfaz, como en los ejemplos web anteriores, pero no proporciona por sí solo el conjunto general de controles de una aplicación.

8.5.1. Cómo incorporar una gráfica a cada interfaz

En los ejercicios no basta con recibir parámetros: con frecuencia la interfaz debe mostrar una curva y actualizarla cuando cambia el modelo. La separación recomendable es siempre la misma:

controles ⟶ callback ⟶ datos calculados

datos calculados ⟶ actualización de la gráfica.

El modelo numérico debe producir datos sin depender de la tecnología usada para dibujarlos. La forma concreta de presentar esos datos cambia según la interfaz:

Lenguaje/interfaz	Forma habitual de mostrar la gráfica
Scilab	Ejes y funciones gráficas dentro de la propia `figure`; el callback modifica o vuelve a dibujar los objetos.
Octave	Ejes dentro de la `figure` y funciones como `plot`; los callbacks actualizan los datos o los objetos gráficos.
Python/Tkinter	Puede incrustarse una figura de Matplotlib en la ventana, por ejemplo con `FigureCanvasTkAgg`, o mostrarse una imagen generada por separado.
Perl/Tk	Puede utilizarse un `Canvas` o mostrar una imagen producida por una herramienta gráfica externa.
JavaScript	El navegador permite dibujar directamente en SVG o `canvas`; los eventos del DOM actualizan esos elementos.
C/GTK	Puede dibujarse en un `GtkDrawingArea` o mostrarse una imagen generada por el programa.
Bash/Zenity	Bash coordina herramientas externas; una opción simple es generar la curva con Gnuplot y mostrar o abrir el archivo resultante. La interacción suele organizarse como ciclos de diálogo.
PowerShell/Windows Forms	Puede utilizarse un control de gráficos de .NET o un `PictureBox` que muestre una imagen generada.

No es necesario que todas las interfaces dibujen de la misma manera. Lo que debe conservarse entre lenguajes es la separación entre el modelo, los datos calculados y la representación visual.

8.5.2. Octave: figure y uicontrol

Octave utiliza un modelo muy parecido al de Scilab: se crea una figura, se agregan controles y a cada control se le asocia un callback Eaton et al. 2024. El siguiente ejemplo lee un número y muestra su raíz cuadrada.

```
f = figure('name', 'GUI Octave', ...
           'position', [300 300 360 180]);

uicontrol(f, 'style', 'text', ...
          'string', 'Valor:', ...
          'position', [20 120 80 25]);

entrada = uicontrol(f, 'style', 'edit', ...
                    'string', '4', ...
                    'position', [100 120 100 25]);

salida = uicontrol(f, 'style', 'text', ...
                   'string', 'Resultado: ---', ...
                   'position', [20 50 300 25]);

uicontrol(f, 'style', 'pushbutton', ...
          'string', 'Calcular', ...
```

```
            'position', [220 118 100 30], ...
            'callback', @(src,event) calcular(entrada, salida));

function calcular(entrada, salida)
  x = str2double(get(entrada, 'string'));
  if x < 0
    set(salida, 'string', 'Error: numero negativo');
  else
    set(salida, 'string', sprintf('Resultado: %.4f', sqrt(x)));
  end
endfunction
```

La correspondencia con Scilab es directa: `uicontrol` crea el widget, `callback` registra la acción y `get`/`set` leen y modifican el estado de la interfaz.

8.5.3. Python: tkinter y ttk

Python incluye `tkinter`, la interfaz estándar de Python a Tcl/Tk. El módulo `ttk` proporciona widgets temáticos. La aplicación crea los controles y termina llamando a `mainloop()`, que ejecuta el bucle de eventos Python Software Foundation 2026b.

```
import tkinter as tk
from tkinter import ttk
import math

root = tk.Tk()
root.title('GUI Python')

x = tk.StringVar(value='4')
r = tk.StringVar(value='Resultado: ---')

def calcular():
    valor = float(x.get())
    if valor < 0:
        r.set('Error: numero negativo')
    else:
        r.set(f'Resultado: {math.sqrt(valor):.4f}')

ttk.Label(root, text='Valor:').grid(row=0, column=0)
ttk.Entry(root, textvariable=x).grid(row=0, column=1)
ttk.Button(root, text='Calcular', command=calcular).grid(row=0,
   column=2)
ttk.Label(root, textvariable=r).grid(row=1, column=0,
   columnspan=3)

root.mainloop()
```

Aquí `command=calcular` cumple el mismo papel que `callback` en Scilab.

8.5.4. Perl: Perl/Tk

Perl/Tk proporciona a Perl un toolkit gráfico basado en Tk. Su modelo es nuevamente ventana, widgets, callbacks y un bucle de eventos iniciado con `MainLoop` Ing-Simmons 2021.

```
use strict;
use warnings;
use Tk;

my $mw = MainWindow->new;
$mw->title('GUI Perl');

my $x = '4';
my $resultado = 'Resultado: ---';

$mw->Label(-text => 'Valor:')->pack;
$mw->Entry(-textvariable => \$x)->pack;

my $lab = $mw->Label(-textvariable => \$resultado)->pack;

$mw->Button(
    -text => 'Calcular',
    -command => sub {
        if ($x < 0) {
            $resultado = 'Error: numero negativo';
        } else {
            $resultado = sprintf('Resultado: %.4f', sqrt($x));
        }
    }
)->pack;

MainLoop;
```

8.5.5. JavaScript: el DOM como GUI

En JavaScript la GUI más natural es la que proporciona el navegador. Los elementos HTML actúan como controles y el DOM permite leerlos y modificarlos. `addEventListener` registra el manejador del evento MDN contributors 2026; Flanagan 2020.

```
<label>Valor:
  <input id="x" type="number" value="4">
</label>
<button id="calcular">Calcular</button>
<p id="resultado">Resultado: ---</p>

<script>
const x = document.getElementById('x');
const r = document.getElementById('resultado');
```

```
document.getElementById('calcular').addEventListener('click', ()
   => {
    const v = Number(x.value);
    if (v < 0)
        r.textContent = 'Error: numero negativo';
    else
        r.textContent = `Resultado: ${Math.sqrt(v).toFixed(4)}`;
});
</script>
```

Para aplicaciones de escritorio escritas con JavaScript puede usarse Electron, que construye aplicaciones de escritorio con JavaScript, HTML y CSS sobre Chromium y Node.js Electron contributors 2026. En la interfaz renderizada se siguen utilizando las herramientas de la plataforma web y sus eventos.

8.5.6. C: GTK

El lenguaje C no incorpora una biblioteca de GUI en el estándar. Una opción habitual es GTK, un toolkit de widgets implementado en C y orientado a eventos. Los eventos de los controles se conectan a funciones mediante señales The GTK Team 2026.

```
#include <gtk/gtk.h>
#include <math.h>

typedef struct {
    GtkWidget *entrada;
    GtkWidget *salida;
} Datos;

static void calcular(GtkButton *boton, gpointer p) {
    Datos *d = p;
    const char *s =
       gtk_editable_get_text(GTK_EDITABLE(d->entrada));
    double x = g_ascii_strtod(s, NULL);
    char r[128];

    if (x < 0)
        g_snprintf(r, sizeof r, "Error: numero negativo");
    else
        g_snprintf(r, sizeof r, "Resultado: %.4f", sqrt(x));

    gtk_label_set_text(GTK_LABEL(d->salida), r);
}

static void activar(GtkApplication *app, gpointer data) {
    GtkWidget *w = gtk_application_window_new(app);
    GtkWidget *caja = gtk_box_new(GTK_ORIENTATION_VERTICAL, 6);
    GtkWidget *boton = gtk_button_new_with_label("Calcular");
    Datos *d = g_new0(Datos, 1);
```

```
    d->entrada = gtk_entry_new();
    gtk_editable_set_text(GTK_EDITABLE(d->entrada), "4");
    d->salida = gtk_label_new("Resultado: ---");

    gtk_box_append(GTK_BOX(caja), d->entrada);
    gtk_box_append(GTK_BOX(caja), boton);
    gtk_box_append(GTK_BOX(caja), d->salida);
    gtk_window_set_child(GTK_WINDOW(w), caja);

    g_signal_connect(boton, "clicked", G_CALLBACK(calcular), d);
    gtk_window_present(GTK_WINDOW(w));
}

int main(int argc, char **argv) {
    GtkApplication *app = gtk_application_new("edu.unsa.gui", 0);
    g_signal_connect(app, "activate", G_CALLBACK(activar), NULL);
    int estado = g_application_run(G_APPLICATION(app), argc,
       argv);
    g_object_unref(app);
    return estado;
}
```

Una forma usual de compilarlo es

```
gcc gui.c -o gui $(pkg-config --cflags --libs gtk4) -lm
```

La llamada `g_signal_connect` cumple el papel de registrar el callback.

8.5.7. Bash: Zenity

Bash no posee widgets ni un bucle gráfico propios. Una forma habitual de agregar interacción gráfica a un script es ejecutar un programa externo como Zenity, diseñado para mostrar diálogos desde la línea de comandos y scripts GNOME Project s.f. En este ejemplo Bash espera a que el diálogo termine y luego continúa.

```
x=$(zenity --entry \
           --title="GUI Bash" \
           --text="Ingrese un valor" \
           --entry-text="4") || exit 1

if awk -v x="$x" 'BEGIN{exit !(x < 0)}'; then
    zenity --error --text="Numero negativo"
else
    r=$(awk -v x="$x" 'BEGIN{printf "%.4f", sqrt(x)}')
    zenity --info --text="Resultado: $r"
fi
```

Este caso es diferente de Scilab, Octave o Python: Bash coordina procesos externos en lugar de contener la GUI dentro de su propio runtime.

8.5.8. PowerShell: Windows Forms

En Windows, PowerShell puede utilizar las clases de .NET. Windows Forms permite construir ventanas y controles desde un script; este ejemplo es específico de Windows Microsoft 2022.

```
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

$form = New-Object System.Windows.Forms.Form
$form.Text = 'GUI PowerShell'
$form.Size = New-Object System.Drawing.Size(360,180)

$entrada = New-Object System.Windows.Forms.TextBox
$entrada.Text = '4'
$entrada.Location = New-Object System.Drawing.Point(20,20)

$boton = New-Object System.Windows.Forms.Button
$boton.Text = 'Calcular'
$boton.Location = New-Object System.Drawing.Point(140,18)

$salida = New-Object System.Windows.Forms.Label
$salida.Text = 'Resultado: ---'
$salida.AutoSize = $true
$salida.Location = New-Object System.Drawing.Point(20,70)

$boton.Add_Click({
    $x = [double]$entrada.Text
    if ($x -lt 0) {
        $salida.Text = 'Error: numero negativo'
    } else {
        $salida.Text = 'Resultado: {0:F4}' -f [Math]::Sqrt($x)
    }
})

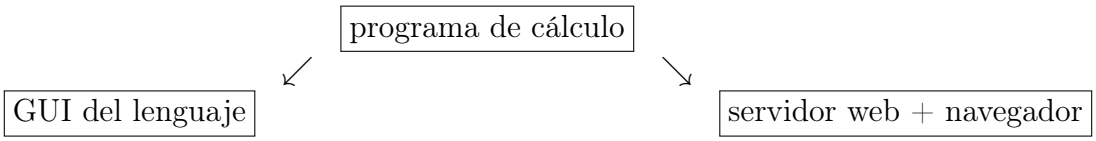
$form.Controls.AddRange(@($entrada, $boton, $salida))
[void]$form.ShowDialog()
```

El método `Add_Click` registra el callback del evento del botón.

8.5.9. Una misma lógica, distintas interfaces

Estos ejemplos muestran una separación que conviene mantener en los programas científicos. La función que calcula una raíz, resuelve una ecuación diferencial o procesa datos no debería depender de la GUI. La interfaz solamente obtiene parámetros, llama al modelo y presenta el resultado.

Por eso un mismo núcleo de cálculo puede exponerse de dos maneras:



La elección de la interfaz no obliga a reescribir el algoritmo. Este principio es el mismo que se utilizó antes al separar modelo, vista y control.

8.6. Jupyter como idea: documentos interactivos

Jupyter (anteriormente IPython Notebook) es un entorno de computación interactiva que implementa la idea de Donald Knuth de *programación literaria* D. E Knuth 1984: código, resultados y explicación en el mismo documento. El proyecto nació como una consola interactiva para Python Pérez y Granger 2007b y se convirtió después en un formato de documento con especificación propia Kluyver, Ragan-Kelley, Pérez et al. 2016a.

En esta unidad lo presentamos como ejemplo de documento interactivo y de programación orientada a eventos. En los ejercicios se retoma esta idea para construir en Scilab una interfaz tipo Jupyter, con evaluación de expresiones, resultados e historial.

Un cuaderno Jupyter está compuesto por **celdas** de dos tipos principales:

Celdas de código: contienen código ejecutable. Al ejecutar una celda, el resultado (texto, tabla, gráfico) aparece inmediatamente debajo.

Celdas de Markdown: contienen texto formateado, fórmulas LaTeX y explicaciones. Al ejecutar la celda, el Markdown se renderiza como texto formateado.

8.6.1. Estado, orden de ejecución e historial

Un cuaderno interactivo conserva un **estado de ejecución**: las variables y objetos creados por una celda continúan disponibles para las celdas que se ejecuten después. Por eso dos cuadernos con el mismo texto pueden producir resultados diferentes si las celdas se ejecutaron en un orden distinto. Esta dependencia del estado explica tanto la comodidad del modelo interactivo como una de sus principales dificultades para la reproducibilidad.

El **historial** registra la secuencia efectiva de trabajo. Para una interfaz tipo Jupyter simplificada conviene guardar, para cada entrada, al menos:

- el texto introducido por el usuario;
- si corresponde a código o a documentación;
- el resultado producido o el mensaje de error;
- el orden en que fue ejecutado.

Una implementación didáctica puede distinguir las entradas de texto por un prefijo, por ejemplo `#`, y no evaluarlas. Las demás entradas se ejecutan en el mismo entorno para que las definiciones anteriores permanezcan activas. Al exportar el historial a Markdown, las entradas de documentación se guardan como texto y las entradas de código y sus resultados se representan en bloques claramente diferenciados. Así el historial deja de ser solamente memoria de la sesión y se convierte en un documento legible.

Evaluar texto como código implica ejecutar instrucciones. Este modelo es apropiado para una herramienta local utilizada por su propio autor, pero no debe exponerse directamente como servicio web para usuarios no confiables sin un mecanismo específico de aislamiento y control.

```
# Instalar y lanzar Jupyter
pip install jupyter
jupyter notebook          # abre en el navegador

# Para trabajar con JupyterLab
pip install jupyterlab
jupyter lab
```

```
# Celda de codigo en un cuaderno Jupyter (Python)
import numpy as np
import matplotlib.pyplot as plt

# Los resultados aparecen debajo de cada celda
t = np.linspace(0, 10, 500)
y = np.exp(-0.3*t) * np.cos(2*t)

plt.figure(figsize=(10, 4))
plt.plot(t, y, 'b-', linewidth=2)
plt.xlabel('Tiempo [s]')
plt.ylabel('Amplitud')
plt.title('Oscilador amortiguado')
plt.grid(True)
plt.show()    # el grafico aparece embebido en el cuaderno
```

El ejemplo se apoya en dos bibliotecas que cumplen en Python el papel que en Scilab cumplen las matrices y los gráficos nativos: el arreglo n-dimensional Harris, Millman, Walt, Gommers, Virtanen, Cournapeau y Oliphant 2020 y la biblioteca de gráficos Hunter 2007a.

El cuaderno tiene también su versión de los widgets de esta unidad. Con `ipywidgets` se declaran deslizadores, botones y campos ligados a una función manejadora, de modo que el lector mueve un parámetro y ve cómo cambia la gráfica sin editar el código Jupyter Widgets Development Team 2026. Es el mismo modelo de eventos que vimos con `uicontrol`, sólo que dentro del documento.

La innovación conceptual de Jupyter no es técnica sino epistemológica: un cuaderno bien escrito es simultáneamente un programa ejecutable, un documento reproducible y una narración científica Granger y Pérez 2021. Quien lee el cuaderno puede reproducir exactamente el análisis, modificar los parámetros y verificar los resultados.

Ésa es la promesa, y conviene contrastarla con lo que ocurre. El formato tuvo una adopción amplísima en la práctica científica, junto con críticas persistentes sobre el orden de ejecución oculto y el estado invisible del núcleo Perkel 2018. Un estudio sobre más de un millón de cuadernos publicados en repositorios encontró que una fracción muy pequeña se ejecuta de principio a fin y reproduce sus propios resultados Pimentel et al. 2019. De ahí que existan recomendaciones explícitas sobre cómo escribir y compartir un cuaderno para que sea efectivamente reproducible Rule et al. 2019.

8.6.2. Eventos para simular fenómenos físicos en Jupyter

Los widgets de Jupyter permiten aplicar el mismo esquema orientado a eventos a un modelo físico. En el siguiente ejemplo, los deslizadores modifican la constante elástica k y la masa m de un oscilador armónico.

Cada cambio en un deslizador constituye un evento que vuelve a ejecutar la función y actualiza la gráfica.

$$x(t) = \cos\left(\sqrt{\frac{k}{m}}\, t\right).$$

```
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact

@interact(k=(0.5, 10.0, 0.1), m=(0.5, 5.0, 0.1))
def oscilador(k=4.0, m=1.0):

    t = np.linspace(0, 10, 500)

    x = np.cos(np.sqrt(k/m) * t)

    plt.figure(figsize=(9, 4))
    plt.plot(t, x)
    plt.xlabel("Tiempo [s]")
    plt.ylabel("x(t)")
    plt.title("Oscilador armonico")
    plt.grid(True)
    plt.show()
```

El evento no modifica directamente el dibujo: modifica los parámetros del modelo. La función calcula nuevamente $x(t)$ y la gráfica muestra el nuevo resultado.

El esquema es

$$\boxed{\text{widget}} \longrightarrow \boxed{\text{evento}} \longrightarrow \boxed{\text{callback}} \longrightarrow \boxed{\text{modelo físico}} \longrightarrow \boxed{\text{gráfica}}.$$

8.7. Ejercicios

8.8. Ejercicios para interfaz terminal

1. **Ejercicio:** Programa ejecutado desde la terminal

   Consigna general.

   Realice las siguientes tareas utilizando el lenguaje de trabajo que corresponda. La consigna es independiente del lenguaje: la terminal del sistema operativo se utiliza para iniciar el intérprete o ejecutar el programa, y la salida del programa se escribe en la salida estándar.

   a) Abra, cuando el lenguaje disponga de él, el intérprete interactivo en modo texto, sin depender de una interfaz gráfica.

   b) Cree un archivo de programa que asigne $x = 3$, calcule $y = x^2 + 1$ y muestre el valor de y. El resultado debe ser 10.

   c) Ejecute el archivo desde la terminal.

   d) Ejecútelo nuevamente redirigiendo la salida estándar al archivo `salida.txt` y compruebe su contenido.

   La diferencia entre las soluciones siguientes está en la forma de iniciar cada lenguaje y en su sintaxis; la operación de terminal `>salida.txt` es la misma porque la realiza el shell, no el lenguaje de cálculo.

   Scilab. Scilab dispone de `scilab-cli`, que abre la consola de texto. En un archivo `.sce` una expresión puede mostrarse con `disp`.

   Listing 8.20: programa.sce

```
x = 3
y = x^2 + 1
disp(y)
```

```
$ scilab-cli
$ scilab-cli -f programa.sce
$ scilab-cli -f programa.sce > salida.txt
$ cat salida.txt
```

   Respuesta en Bash. Bash es a la vez el shell de la terminal y un lenguaje de programación. Para este cálculo entero puede utilizar su aritmética propia.

   Listing 8.21: programa.sh

```
#!/usr/bin/env bash
x=3
y=$((x*x + 1))
printf '%d\n' "$y"
```

```
$ bash
$ bash programa.sh
$ bash programa.sh > salida.txt
$ cat salida.txt
```

Extensión gráfica de Bash con Zenity y YAD. La interfaz de un programa Bash no tiene por qué limitarse a la terminal. Programas auxiliares como **Zenity** y **YAD** permiten abrir cuadros de diálogo gráficos desde un script. Bash sigue realizando el cálculo; Zenity o YAD solamente se ocupan de la entrada y la salida gráfica.

Si no están instalados, en sistemas Debian/Ubuntu pueden instalarse con:

```
$ sudo apt install zenity yad
```

Con **Zenity**, el mismo cálculo puede escribirse así:

Listing 8.22: programa_zenity.sh

```
#!/usr/bin/env bash

x=$(zenity --entry \
    --title="Calculo con Bash" \
    --text="Ingrese un valor entero para x:" \
    --entry-text="3") || exit 1

y=$((x*x + 1))

zenity --info \
    --title="Resultado" \
    --text="Para x = $x, y = x^2 + 1 = $y"
```

```
$ bash programa_zenity.sh
```

YAD (*Yet Another Dialog*) cumple una función semejante y ofrece más opciones para construir formularios. El ejemplo equivalente es:

Listing 8.23: programa_yad.sh

```
#!/usr/bin/env bash

x=$(yad --entry \
    --title="Calculo con Bash" \
    --text="Ingrese un valor entero para x:" \
    --entry-text="3") || exit 1

y=$((x*x + 1))

yad --info \
    --title="Resultado" \
    --text="Para x = $x, y = x^2 + 1 = $y"
```

```
$ bash programa_yad.sh
```

En ambos casos, si se acepta el valor inicial `3`, la ventana de resultado muestra `10`. Esto permite comparar tres interfaces para el mismo lenguaje: Bash puramente en terminal, Bash + Zenity y Bash + YAD.

Octave. Octave dispone de `octave-cli`. En un archivo `.m` se usa `disp` o `fprintf` para producir salida.

Listing 8.24: programa.m

```
x = 3;
y = x^2 + 1;
disp(y)
```

```
$ octave-cli
$ octave-cli --quiet programa.m
$ octave-cli --quiet programa.m > salida.txt
$ cat salida.txt
```

Perl. El ejecutable `perl` se usa normalmente para ejecutar un archivo o una expresión indicada con `-e`. A diferencia de Scilab, Octave o Python, Perl no ofrece como interfaz principal un REPL simple; para este ejercicio basta ejecutar el script desde la terminal.

Listing 8.25: programa.pl

```
use strict;
use warnings;

my $x = 3;
my $y = $x**2 + 1;
print "$y\n";
```

```
$ perl -e 'my $x=3; print $x**2+1, "\n"'
$ perl programa.pl
$ perl programa.pl > salida.txt
$ cat salida.txt
```

Python. El comando `python3` abre el intérprete interactivo. En un archivo es necesario usar `print`, porque una expresión suelta no se muestra automáticamente al ejecutar un script.

Listing 8.26: programa.py

```
x = 3
y = x**2 + 1
print(y)
```

```
$ python3
$ python3 programa.py
```

```
$ python3 programa.py > salida.txt
$ cat salida.txt
```

a) Compare los cinco archivos de programa. ¿Qué parte del algoritmo es la misma y qué parte depende solamente de la sintaxis del lenguaje?

b) Explique por qué `>salida.txt` funciona sin modificar ninguno de los programas.

c) Identifique en cada caso qué programa actúa como intérprete y cuál es el archivo que contiene el código fuente.

8.9. Ejercicios para interfaz web

1. **Ejercicio:** Puesta en marcha del servidor multilenguaje

   Trabaje con la versión actual del servidor descripta en el tutorial. En un mismo directorio deben estar `formulario.html`, `servidor.py` y al menos un programa de cálculo compatible con el protocolo del servidor.

   Inicie el servidor sin fijar en la consigna un lenguaje particular para el motor de cálculo:

   ```
   $ cd mi_aplicacion
   $ python3 servidor.py
   ```

   Abra `http://localhost:8080`, complete el formulario y presione Calcular. Si el formulario contiene el selector `motor`, cambie de motor entre solicitudes sin reiniciar el servidor.

   a) Identifique qué programa cumple el papel de servidor HTTP y qué programa cumple el papel de motor de cálculo.

   b) Examine, cuando existan, los archivos de comunicación:

   ```
   cat /tmp/params.txt
   cat /tmp/resultado.html
   cat /tmp/error_motor.txt
   ```

   c) Explique por qué cambiar el motor no cambia HTTP ni obliga a reiniciar `servidor.py` cuando el motor se selecciona desde el formulario.

   Respuestas por lenguaje.

   Scilab. El motor se guarda como `calcular.sce` y se ejecuta con `scilab-cli`. El servidor sigue siendo el mismo; sólo cambia el valor del campo `motor`.

   ```
   curl -X POST http://localhost:8080 \
     -d "motor=scilab&v0=50&angulo=45"
   ```

Octave. El motor se guarda como `calcular.m` y se ejecuta con `octave-cli`. El mismo servidor despacha la solicitud.

```
curl -X POST http://localhost:8080 \
  -d "motor=octave&v0=50&angulo=45"
```

Python. El motor se guarda como `calcular.py` y se ejecuta con `python3`. Python actúa aquí como motor de cálculo, además de ser el lenguaje de `servidor.py`; son dos procesos y dos papeles distintos.

```
curl -X POST http://localhost:8080 \
  -d "motor=python&v0=50&angulo=45"
```

Perl. El motor se guarda como `calcular.pl` y se ejecuta con `perl`. Recibe los mismos archivos y variables de entorno que los demás motores.

```
curl -X POST http://localhost:8080 \
  -d "motor=perl&v0=50&angulo=45"
```

JavaScript. El motor se guarda como `calcular.js` y se ejecuta con `node`; JavaScript corre aquí fuera del navegador.

```
curl -X POST http://localhost:8080 \
  -d "motor=javascript&v0=50&angulo=45"
```

C. El fuente puede guardarse como `calcular.c`; se compila y el ejecutable esperado por el servidor se llama `calcular`.

```
gcc calcular.c -o calcular -lm
curl -X POST http://localhost:8080 \
  -d "motor=c&v0=50&angulo=45"
```

Bash. El motor se guarda como `calcular.sh` y se ejecuta con `bash`; el propio shell puede ser motor de cálculo.

```
curl -X POST http://localhost:8080 \
  -d "motor=bash&v0=50&angulo=45"
```

PowerShell. El motor se guarda como `calcular.ps1` y se ejecuta con `pwsh -File`.

```
curl -X POST http://localhost:8080 \
  -d "motor=powershell&v0=50&angulo=45"
```

2. **Ejercicio:** Parámetros enviados al motor

   Examine el archivo neutro indicado por `PARAM_FILE`. En la configuración del tutorial su valor es `/tmp/params.txt`:

```
cat /tmp/params.txt
```

El servidor escribe una línea `clave=valor` por parámetro. El archivo contiene datos y no código ejecutable del motor.

a) Identifique la clave y el valor de cada línea y explique por qué inicialmente todos los valores son texto, aunque representen números o valores lógicos.

b) Determine qué valor tiene `g_tray` cuando el checkbox está marcado y cuando no lo está.

c) Explique por qué se usa un campo oculto antes de cada checkbox.

d) Verifique que el campo `motor` no se copia al archivo de parámetros: lo consume `servidor.py` para decidir qué programa ejecutar.

e) Escriba en el lenguaje elegido un lector mínimo de `clave=valor`.

Respuestas por lenguaje.

Scilab. Una forma general es conservar dos vectores de strings y usar una función auxiliar para buscar una clave.

```
function [claves,valores] = leer_params()
    lineas = mgetl(getenv('PARAM_FILE'))
    claves=[]; valores=[]
    for i=1:size(lineas,'*')
        pos=strindex(lineas(i),'=')
        if ~isempty(pos) then
            k=pos(1)
            claves($+1,1)=part(lineas(i),1:k-1)
            valores($+1,1)=part(lineas(i),k+1:length(lineas(i)))
        end
    end
endfunction

function v = param(claves,valores,nombre,defecto)
    j=find(claves==nombre)
    if isempty(j) then v=defecto; else v=valores(j($)); end
endfunction
```

Octave. Una estructura permite conservar los valores como texto hasta que se conozca el tipo que necesita cada clave.

```
p = struct();
fid = fopen(getenv('PARAM_FILE'),'r');
while true
  s = fgetl(fid); if ~ischar(s), break; end
  k = find(s == '=',1);
  if ~isempty(k), p.(s(1:k-1)) = s(k+1:end); end
end
fclose(fid);
```

Python. Un diccionario representa directamente el conjunto de pares.

```
import os
p = {}
with open(os.environ['PARAM_FILE'], encoding='utf-8') as f:
    for linea in f:
        if '=' in linea:
            k, v = linea.rstrip('\n').split('=', 1)
            p[k] = v
```

Perl. Un hash cumple el mismo papel.

```
my %p;
open my $in, '<', $ENV{PARAM_FILE} or die $!;
while (<$in>) {
    chomp;
    my ($k,$v) = split(/=/,$_,2);
    $p{$k} = $v if defined $v;
}
close $in;
```

JavaScript. En Node.js se puede construir un objeto a partir de las líneas.

```
const fs = require('fs');
const p = {};
for (const s of
   fs.readFileSync(process.env.PARAM_FILE,'utf8')
                 .split(/\r?\n/)) {
  const i = s.indexOf('=');
  if (i >= 0) p[s.slice(0,i)] = s.slice(i+1);
}
```

C. En C se procesa una línea por vez. Para un programa grande conviene encapsular esta lógica en una función.

```
FILE *f = fopen(getenv("PARAM_FILE"), "r");
char linea[512], *eq;
while (fgets(linea, sizeof linea, f)) {
    linea[strcspn(linea,"\r\n")] = 0;
    eq = strchr(linea,'=');
    if (!eq) continue;
    *eq = 0;
    printf("clave=%s valor=%s\n", linea, eq+1);
}
fclose(f);
```

Bash. Un arreglo asociativo conserva los pares y permite recuperar luego cualquier clave.

```
declare -A p
while IFS='=' read -r clave valor; do
    p["$clave"]="$valor"
```

```
done < "$PARAM_FILE"
```

PowerShell. Se usa una tabla hash y `-split` limitado a dos partes.

```
$p = @{}
Get-Content $env:PARAM_FILE | ForEach-Object {
    $k,$v = $_ -split '=',2
    if ($null -ne $v) { $p[$k] = $v }
}
```

3. **Ejercicio:** Campos nuevos del formulario web

   Modifique `formulario.html` para agregar:

   a) un campo `type=number`, de nombre `masa`, en kg, con valor por defecto 1.0;

   b) un campo `type=color`, de nombre `color`, para elegir el color de la curva;

   c) un checkbox `leyenda`, con su campo oculto, para mostrar u ocultar la leyenda.

   No modifique `servidor.py` ni el programa del motor. Envíe el formulario y compruebe que las tres claves nuevas aparecen en el archivo indicado por `PARAM_FILE`. Explique por qué el servidor no necesita conocer de antemano esas claves.

   Respuesta común del formulario.

```
<label>Masa [kg]</label>
<input type=number name=masa value=1.0 step=0.1>

<label>Color</label>
<input type=color name=color value="#2980b9">

<input type=hidden name=leyenda value=off>
<label><input type=checkbox name=leyenda value=on checked>
  Mostrar leyenda</label>
```

   Respuestas por lenguaje.

   La modificación del formulario es la misma para todos los motores. Las siguientes líneas muestran cómo se convierten los tres valores una vez aplicado el lector del ejercicio anterior.

   Scilab.

```
[claves,valores]=leer_params()
masa=strtod(param(claves,valores,'masa','1.0'))
color=param(claves,valores,'color','#2980b9')
leyenda=(param(claves,valores,'leyenda','off')=='on')
```

   Octave.

```
masa = str2double(p.masa);
color = p.color;
leyenda = strcmp(p.leyenda,'on');
```

Python.

```
masa = float(p['masa'])
color = p['color']
leyenda = (p.get('leyenda') == 'on')
```

Perl.

```
my $masa = 0 + $p{masa};
my $color = $p{color};
my $leyenda = (($p{leyenda} // 'off') eq 'on');
```

JavaScript.

```
const masa = Number(p.masa);
const color = p.color;
const leyenda = (p.leyenda === 'on');
```

C. En C se guardan las claves necesarias al recorrer el archivo.

```
double masa = 1.0;
char color[32] = "#2980b9";
int leyenda = 0;
/* al leer cada par clave/valor: */
if (!strcmp(clave,"masa")) masa = strtod(valor,NULL);
if (!strcmp(clave,"color")) snprintf(color,sizeof
   color,"%s",valor);
if (!strcmp(clave,"leyenda")) leyenda = !strcmp(valor,"on");
```

Bash.

```
masa=${p[masa]:-1.0}
color=${p[color]:-#2980b9}
[[ ${p[leyenda]:-off} == on ]] && leyenda=1 || leyenda=0
```

PowerShell.

```
$masa = [double]$p['masa']
$color = $p['color']
$leyenda = ($p['leyenda'] -eq 'on')
```

4. **Ejercicio:** Masa en la simulación del proyectil

   Modifique el motor de cálculo seleccionado para utilizar la masa que llega mediante `PARAM_FILE`. No modifique `servidor.py`.

La ecuación del proyectil con resistencia cuadrática es

$$m\ddot{x} = -b\,v\,\dot{x}, \qquad m\ddot{y} = -mg - b\,v\,\dot{y}.$$

a) Lea `masa`, conviértala a número y valide que sea positiva.

b) Use la masa en las aceleraciones del modelo.

c) Agregue la masa al fragmento HTML escrito en `RESULT_FILE`.

d) Compare dos masas manteniendo constantes los demás parámetros.

Respuestas por lenguaje.

En todos los casos, si $v = \sqrt{v_x^2 + v_y^2}$, las aceleraciones son $a_x = -(b/m)vv_x$ y $a_y = -g - (b/m)vv_y$.

Scilab.

```
[claves,valores]=leer_params()
m=strtod(param(claves,valores,'masa','1.0'))
if isnan(m) | m <= 0 then error('masa invalida'), end
v = sqrt(vx^2 + vy^2)
ax = -(b/m) * v * vx
ay = -g - (b/m) * v * vy
mfprintf(fd,'<p>Masa: %.3f kg</p>',m)
```

Octave.

```
m = str2double(p.masa);
if ~isfinite(m) || m <= 0, error('masa invalida'); end
v = hypot(vx,vy);
ax = -(b/m)*v*vx;
ay = -g - (b/m)*v*vy;
fprintf(fid,'<p>Masa: %.3f kg</p>',m);
```

Python.

```
m = float(p['masa'])
if not math.isfinite(m) or m <= 0:
    raise ValueError('masa invalida')
v = math.hypot(vx, vy)
ax = -(b/m) * v * vx
ay = -g - (b/m) * v * vy
html += f'<p>Masa: {m:.3f} kg</p>'
```

Perl.

```
my $m = 0 + $p{masa};
die "masa invalida" unless $m > 0;
my $v = sqrt($vx*$vx + $vy*$vy);
my $ax = -($b/$m)*$v*$vx;
my $ay = -$g - ($b/$m)*$v*$vy;
printf $out '<p>Masa: %.3f kg</p>', $m;
```

JavaScript.

```
const m = Number(p.masa);
if (!Number.isFinite(m) || m <= 0) throw Error('masa
   invalida');
const v = Math.hypot(vx,vy);
const ax = -(b/m)*v*vx;
const ay = -g - (b/m)*v*vy;
html += `<p>Masa: ${m.toFixed(3)} kg</p>`;
```

C.

```
double m = strtod(valor_masa,NULL);
if (!(m > 0.0) || !isfinite(m)) { /* error */ }
double v = hypot(vx,vy);
double ax = -(b/m)*v*vx;
double ay = -g - (b/m)*v*vy;
fprintf(out,"<p>Masa: %.3f kg</p>",m);
```

Bash. Para aritmética real se usa `awk`.

```
awk -v m="$masa" -v b="$b" -v vx="$vx" -v vy="$vy" -v g="$g"
   '
BEGIN {
  v=sqrt(vx*vx+vy*vy); ax=-(b/m)*v*vx; ay=-g-(b/m)*v*vy;
  printf "ax=%.8g ay=%.8g\n",ax,ay
}'
```

PowerShell.

```
$m = [double]$p['masa']
if ($m -le 0) { throw 'masa invalida' }
$v = [Math]::Sqrt($vx*$vx + $vy*$vy)
$ax = -($b/$m)*$v*$vx
$ay = -$g - ($b/$m)*$v*$vy
$html += '<p>Masa: {0:F3} kg</p>' -f $m
```

5. **Ejercicio:** Color de la gráfica desde el formulario

   Haga que el motor seleccionado utilice el valor del campo `color` recibido mediante `PARAM_FILE` para elegir el color de la trayectoria producida por gnuplot. El navegador envía un texto de la forma `#rrggbb`.

   a) Conserve el color como texto y valide obligatoriamente que tenga exactamente el formato `#[0-9A-Fa-f]{6}`. No confíe en la validación del navegador: la misma solicitud puede construirse con `curl` u otro cliente HTTP.

   b) Si el valor no es válido, rechace el dato antes de construir el archivo de comandos de gnuplot.

   c) Si es válido, insértelo entrecomillado en la orden de gnuplot.

d) Pruebe colores correctos y también entradas deliberadamente incorrectas.

Respuestas por lenguaje.

Suponga que `DAT` o su equivalente contiene los datos y que `gp` es el archivo de comandos de gnuplot. En todos los casos se acepta solamente un `#` seguido por seis dígitos hexadecimales.

Scilab. Se valida carácter por carácter para no depender de una expresión regular particular.

```
[claves,valores]=leer_params()
color=param(claves,valores,'color','#2980b9')
hex='0123456789abcdef'
ok=(length(color)==7 & part(color,1)=='#')
if ok then
    h=convstr(part(color,2:7),'l')
    for j=1:6
        ok = ok & ~isempty(strindex(hex,part(h,j)))
    end
end
if ~ok then error('color invalido: use #rrggbb'), end
mfprintf(gp,"plot '%s' using 2:3 with lines lc rgb
   '%s'\n",DAT,color)
```

Octave.

```
color = p.color;
if isempty(regexp(color,'^#[0-9A-Fa-f]{6}$','once'))
  error('color invalido: use #rrggbb');
end
fprintf(gp,"plot '%s' using 2:3 with lines lc rgb
   '%s'\n",DAT,color);
```

Python.

```
import re
color = p.get('color','#2980b9')
if re.fullmatch(r'#[0-9A-Fa-f]{6}', color) is None:
    raise ValueError('color invalido: use #rrggbb')
gp.write(f"plot '{DAT}' using 2:3 with lines lc rgb
   '{color}'\n")
```

Perl.

```
my $color = $p{color} // '#2980b9';
die "color invalido: use #rrggbb" unless $color =~
   /^#[0-9A-Fa-f]{6}$/;
print $gp "plot '$DAT' using 2:3 with lines lc rgb
   '$color'\n";
```

JavaScript.

```
const color = p.color ?? '#2980b9';
if (!/^#[0-9A-Fa-f]{6}$/.test(color))
  throw new Error('color invalido: use #rrggbb');
fs.appendFileSync(gpFile,
  `plot '${DAT}' using 2:3 with lines lc rgb '${color}'\n`);
```

C. En C se comprueba la longitud y que cada carácter posterior al # sea hexadecimal.

```
#include <ctype.h>
int color_valido(const char *s) {
    if (!s || strlen(s)!=7 || s[0]!='#') return 0;
    for (int i=1;i<7;i++)
        if (!isxdigit((unsigned char)s[i])) return 0;
    return 1;
}
if (!color_valido(color)) {
    fprintf(stderr,"color invalido: use #rrggbb\n");
    return 1;
}
fprintf(gp,"plot '%s' using 2:3 with lines lc rgb
   '%s'\n",dat,color);
```

Bash.

```
re='^#[[:xdigit:]]{6}$'
if [[ ! $color =~ $re ]]; then
  echo 'color invalido: use #rrggbb' >&2
  exit 1
fi
printf "plot '%s' using 2:3 with lines lc rgb '%s'\n" \
       "$DAT" "$color" >> "$GP_FILE"
```

PowerShell.

```
$color = $p['color']
if ($color -notmatch '^#[0-9A-Fa-f]{6}$') {
    throw 'color invalido: use #rrggbb'
}
Add-Content $gpFile "plot '$DAT' using 2:3 with lines lc rgb
   '$color'"
```

6. **Ejercicio:** Energía del proyectil

   Agregue una tercera gráfica al sistema con la energía cinética, potencial y total del proyectil:

$$E_c = \tfrac{1}{2}mv^2, \qquad E_p = mgy, \qquad E_t = E_c + E_p.$$

   a) Agregue un checkbox `g_ener` con su campo oculto.

b) Convierta `on/off` al tipo lógico del lenguaje.

c) Si está activado, escriba en un archivo de datos las columnas `t Ec Ep Et` y genere la gráfica con gnuplot.

d) Incluya la gráfica en el HTML escrito en `RESULT_FILE`.

e) Compare los casos con y sin disipación.

Respuestas por lenguaje.

Scilab.

```
[claves,valores]=leer_params()
g_ener=(param(claves,valores,'g_ener','off')=='on')
if g_ener then
    Ec = 0.5*m*(vx.^2 + vy.^2)
    Ep = m*g*y
    Et = Ec + Ep
    mfprintf(fd,'%f %f %f %f\n',t,Ec,Ep,Et)
end
```

Octave.

```
g_ener = strcmp(p.g_ener,'on');
if g_ener
  Ec = 0.5*m.*(vx.^2 + vy.^2);
  Ep = m*g.*y; Et = Ec + Ep;
  fprintf(fd,'%.9g %.9g %.9g
     %.9g\n',[t(:),Ec(:),Ep(:),Et(:)]');
end
```

Python.

```
g_ener = (p.get('g_ener') == 'on')
if g_ener:
    Ec = [0.5*m*(vx[i]**2 + vy[i]**2) for i in range(len(t))]
    Ep = [m*g*y[i] for i in range(len(t))]
    Et = [Ec[i] + Ep[i] for i in range(len(t))]
    for fila in zip(t,Ec,Ep,Et):
        fd.write(' '.join(f'{q:.9g}' for q in fila) + '\n')
```

Perl.

```
my $g_ener = (($p{g_ener}//'off') eq 'on');
if ($g_ener) {
    for my $i (0..$#t) {
        my $Ec = .5*$m*($vx[$i]**2+$vy[$i]**2);
        my $Ep = $m*$g*$y[$i];
        printf $fd "%.9g %.9g %.9g
           %.9g\n",$t[$i],$Ec,$Ep,$Ec+$Ep;
    }
}
```

JavaScript.

```
const gEner = p.g_ener === 'on';
if (gEner) {
  const filas = t.map((ti,i) => {
    const Ec=.5*m*(vx[i]*vx[i]+vy[i]*vy[i]);
    const Ep=m*g*y[i]; return `${ti} ${Ec} ${Ep} ${Ec+Ep}`;
  });
  fs.writeFileSync(dataEnergia, filas.join('\n')+'\n');
}
```

C.

```
int g_ener = !strcmp(valor_g_ener,"on");
if (g_ener) {
    for (size_t i=0; i<n; ++i) {
        double Ec=.5*m*(vx[i]*vx[i]+vy[i]*vy[i]);
        double Ep=m*g*y[i];
        fprintf(fd,"%.9g %.9g %.9g %.9g\n",t[i],Ec,Ep,Ec+Ep);
    }
}
```

Bash.

```
if [[ ${g_ener:-off} == on ]]; then
  awk -v m="$masa" -v g="$g" '{
    Ec=.5*m*($4*$4+$5*$5); Ep=m*g*$3;
    print $1,Ec,Ep,Ec+Ep
  }' "$DAT" > /tmp/energia.dat
fi
```

PowerShell.

```
$gEner = ($p['g_ener'] -eq 'on')
if ($gEner) {
    for ($i=0; $i -lt $t.Count; $i++) {
        $Ec=.5*$m*($vx[$i]*$vx[$i]+$vy[$i]*$vy[$i])
        $Ep=$m*$g*$y[$i]
        "$($t[$i]) $Ec $Ep $($Ec+$Ep)" | Add-Content
           $dataEnergia
    }
}
```

7. **Ejercicio:** Descarga de un capacitor con servidor web

   Cree una aplicación distinta reutilizando `servidor.py` sin modificarlo. La aplicación debe resolver

$$RC\,\frac{dV}{dt} + V = 0, \qquad V(0) = V_0,$$

   y comparar Euler con la solución exacta $V(t) = V_0 e^{-t/RC}$.

a) Cree un directorio independiente con `formulario.html`, `servidor.py` y el archivo del motor elegido.

b) El formulario debe pedir `R`, `C` y `V0`.

c) El motor debe leer exclusivamente mediante `PARAM_FILE` y escribir exclusivamente mediante `RESULT_FILE`.

d) Use $\Delta t = RC/100$ y simule hasta $5RC$. Genere un archivo de datos con `t V_Euler V_exacta` y una gráfica con gnuplot.

e) Verifique en la gráfica la constante de tiempo $\tau = RC$.

Respuestas por lenguaje.

Las respuestas siguientes muestran el núcleo numérico. Se supone usado el lector del ejercicio 2 y que `R`, `C` y `V0` ya fueron convertidos a número. El archivo generado puede graficarse con el mismo script gnuplot en todos los casos.

Scilab.

```
tau=R*C; dt=tau/100; n=round(5*tau/dt); V=V0
fd=mopen('/tmp/capacitor.dat','w')
for i=0:n
    t=i*dt; Ve=V0*exp(-t/tau)
    mfprintf(fd,'%.9g %.9g %.9g\n',t,V,Ve)
    V=V-dt*V/tau
end
mclose(fd)
```

Octave.

```
tau=R*C; dt=tau/100; n=round(5*tau/dt); V=V0;
fd=fopen('/tmp/capacitor.dat','w');
for i=0:n
  t=i*dt; Ve=V0*exp(-t/tau);
  fprintf(fd,'%.9g %.9g %.9g\n',t,V,Ve);
  V=V-dt*V/tau;
end
fclose(fd);
```

Python.

```
tau=R*C; dt=tau/100; n=round(5*tau/dt); V=V0
with open('/tmp/capacitor.dat','w') as fd:
    for i in range(n+1):
        t=i*dt; Ve=V0*math.exp(-t/tau)
        fd.write(f'{t:.9g} {V:.9g} {Ve:.9g}\n')
        V -= dt*V/tau
```

Perl.

```
my $tau=$R*$C; my $dt=$tau/100; my $n=int(5*$tau/$dt+.5); my
   $V=$V0;
```

```
open my $fd,'>','/tmp/capacitor.dat' or die $!;
for my $i (0..$n) {
    my $t=$i*$dt; my $Ve=$V0*exp(-$t/$tau);
    printf $fd "%.9g %.9g %.9g\n",$t,$V,$Ve;
    $V -= $dt*$V/$tau;
}
close $fd;
```

JavaScript.

```
const tau=R*C, dt=tau/100, n=Math.round(5*tau/dt); let V=V0;
let s='';
for (let i=0;i<=n;i++) {
  const t=i*dt, Ve=V0*Math.exp(-t/tau);
  s += `${t} ${V} ${Ve}\n`; V -= dt*V/tau;
}
fs.writeFileSync('/tmp/capacitor.dat',s);
```

C.

```
double tau=R*C, dt=tau/100.0, V=V0;
long n=lround(5*tau/dt);
FILE *fd=fopen("/tmp/capacitor.dat","w");
for (long i=0;i<=n;i++) {
    double t=i*dt, Ve=V0*exp(-t/tau);
    fprintf(fd,"%.9g %.9g %.9g\n",t,V,Ve);
    V -= dt*V/tau;
}
fclose(fd);
```

Bash.

```
awk -v R="$R" -v C="$C" -v V0="$V0" 'BEGIN {
  tau=R*C; dt=tau/100; n=int(5*tau/dt+.5); V=V0;
  for(i=0;i<=n;i++) {
    t=i*dt; Ve=V0*exp(-t/tau); print t,V,Ve; V-=dt*V/tau;
  }
}' > /tmp/capacitor.dat
```

PowerShell.

```
$tau=$R*$C; $dt=$tau/100; $n=[Math]::Round(5*$tau/$dt);
   $V=$V0
Remove-Item /tmp/capacitor.dat -ErrorAction Ignore
for ($i=0; $i -le $n; $i++) {
  $t=$i*$dt; $Ve=$V0*[Math]::Exp(-$t/$tau)
  "$t $V $Ve" | Add-Content /tmp/capacitor.dat
  $V -= $dt*$V/$tau
}
```

El script gnuplot común puede ser:

```
set terminal svg size 800,450
set output '/tmp/capacitor.svg'
set grid
set xlabel 't [s]'
set ylabel 'V [V]'
plot '/tmp/capacitor.dat' using 1:2 with lines title
   'Euler', \
     '/tmp/capacitor.dat' using 1:3 with lines title 'Exacta'
```

Cada motor ejecuta gnuplot con el mecanismo normal de su lenguaje, lee el SVG y lo concatena al fragmento HTML de `RESULT_FILE`.

8. **Ejercicio:** Validación y manejo de errores

   Agregue validación al motor seleccionado sin modificar el servidor. Los errores debidos a datos del usuario deben producir un fragmento HTML comprensible en `RESULT_FILE`. Las fallas inesperadas deben terminar el motor con estado de error y escribir un diagnóstico en la salida de error, de modo que `servidor.py` pueda conservarlo en `/tmp/error_motor.txt`.

   a) Valide que `v0` esté entre 1 y 500 y `angulo` entre 1 y 89.

   b) Ante un dato inválido escriba una explicación en `RESULT_FILE` y termine normalmente esa solicitud.

   c) Para una falla inesperada utilice el mecanismo de captura de errores del lenguaje. Si el lenguaje no posee excepciones, propague un código de retorno distinto de cero.

   d) Compruebe que los errores de usuario aparecen como HTML y que un error inesperado deja información en `error_motor.txt`.

   Respuestas por lenguaje.

   En los fragmentos siguientes `simular()` representa el cálculo principal ya probado. Primero se trata separadamente el error de entrada; luego se muestra la captura de fallas inesperadas.

   Scilab. `execstr(...,'errcatch')` permite ejecutar el cálculo sin abortar inmediatamente y devuelve un código de error.

```
if isnan(v0) | v0<1 | v0>500 | isnan(angulo) | angulo<1 |
   angulo>89 then
    fd=mopen(getenv('RESULT_FILE'),'w')
    mfprintf(fd,'<p class="error">Parametros fuera de
       rango</p>')
    mclose(fd); quit
end

ierr=execstr('simular()','errcatch')
if ierr<>0 then
    mfprintf(0,'Error inesperado del motor:
       %s\n',lasterror())
```

```
    error(lasterror())
end
```

Octave. Se usa `try/catch`; `rethrow` conserva el fallo como error del proceso para que el servidor lo registre.

```
if ~isfinite(v0) || v0<1 || v0>500 || ~isfinite(angulo) ||
   angulo<1 || angulo>89
  fid=fopen(getenv('RESULT_FILE'),'w');
  fprintf(fid,'<p class="error">Parametros fuera de
     rango</p>');
  fclose(fid); return;
end
try
  simular();
catch err
  fprintf(2,'Error inesperado del motor: %s\n',err.message);
  rethrow(err);
end
```

Python.

```
import os, sys
if not (1 <= v0 <= 500 and 1 <= angulo <= 89):
    with open(os.environ['RESULT_FILE'],'w') as f:
        f.write('<p class="error">Parametros fuera de
           rango</p>')
    raise SystemExit(0)
try:
    simular()
except Exception as e:
    print(f'Error inesperado del motor: {e}',
       file=sys.stderr)
    raise
```

Perl. `eval` captura la excepción; `warn` la envía a `stderr` y el proceso termina con código distinto de cero.

```
if ($v0<1 || $v0>500 || $angulo<1 || $angulo>89) {
    open my $o,'>',$ENV{RESULT_FILE} or die $!;
    print $o '<p class="error">Parametros fuera de
       rango</p>';
    close $o; exit 0;
}
eval { simular(); 1 } or do {
    my $e=$@ || 'error desconocido'; warn "Error inesperado:
       $e"; exit 1;
};
```

JavaScript.

```
if (!(v0>=1 && v0<=500 && angulo>=1 && angulo<=89)) {
  fs.writeFileSync(process.env.RESULT_FILE,
    '<p class="error">Parametros fuera de rango</p>');
  process.exit(0);
}
try {
  simular();
} catch (e) {
  console.error('Error inesperado del motor:', e);
  process.exit(1);
}
```

C. C no posee excepciones estándar. El cálculo debe devolver un código de estado y el programa principal debe propagarlo.

```
if (!(v0>=1 && v0<=500 && angulo>=1 && angulo<=89)) {
    FILE *o=fopen(getenv("RESULT_FILE"),"w");
    fputs("<p class=\"error\">Parametros fuera de
       rango</p>",o);
    fclose(o); return 0;
}
int rc=simular();
if (rc!=0) {
    fprintf(stderr,"Error inesperado del motor, codigo
       %d\n",rc);
    return rc;
}
```

Bash. Con `set -E -e` y `trap ERR` una orden fallida se informa por `stderr` y conserva un estado de salida no nulo.

```
if ! awk -v v="$v0" -v a="$angulo" \
  'BEGIN{exit !(v>=1&&v<=500&&a>=1&&a<=89)}'; then
  printf '%s\n' '<p class="error">Parametros fuera de
     rango</p>' \
    > "$RESULT_FILE"
  exit 0
fi
set -Ee
trap 'rc=$?; echo "Error inesperado del motor (rc=$rc)" >&2;
   exit $rc' ERR
simular
```

PowerShell. `try/catch` se combina con `ErrorActionPreference=Stop` para convertir errores no terminantes en excepciones capturables.

```
if ($v0 -lt 1 -or $v0 -gt 500 -or $angulo -lt 1 -or $angulo
   -gt 89) {
    '<p class="error">Parametros fuera de rango</p>' |
      Set-Content -Encoding UTF8 $env:RESULT_FILE
```

```
    exit 0
}
$ErrorActionPreference='Stop'
try {
    Simular
} catch {
    [Console]::Error.WriteLine("Error inesperado del motor:
       $_")
    exit 1
}
```

9. **Ejercicio:** Tiempo de respuesta del sistema

   Analice el tiempo de respuesta manteniendo `servidor.py` en ejecución y enviando desde otra terminal la misma solicitud a cada motor disponible.

   a) Mida cada motor varias veces y compare mediana o promedio.

   b) Mida primero el motor directamente y luego la solicitud HTTP completa. La diferencia aproxima el costo adicional del servidor, del protocolo y de la transferencia local.

   c) Compare métodos numéricos dentro de un mismo motor.

   d) Proponga una estrategia para reducir el tiempo sin cambiar el protocolo `PARAM_FILE`/`RESULT_`

   Prepare una entrada idéntica para las mediciones directas:

```
cat > /tmp/params_bench.txt <<'EOF'
v0=50
angulo=45
met=rk4
g_tray=off
g_vel=off
EOF
```

Respuestas por lenguaje.

En cada caso se muestra primero el tiempo propio del motor y luego el tiempo HTTP total. Use varias repeticiones; una medición aislada es poco confiable.

Scilab.

```
time env PARAM_FILE=/tmp/params_bench.txt \
  RESULT_FILE=/tmp/resultado_bench.html \
  scilab-cli -e "exec('calcular.sce',-1); quit"
time curl -s -o /dev/null -X POST http://localhost:8080 \
  -d 'motor=scilab&v0=50&angulo=45&met=rk4'
```

Octave.

```
time env PARAM_FILE=/tmp/params_bench.txt \
  RESULT_FILE=/tmp/resultado_bench.html octave-cli --quiet
     calcular.m
time curl -s -o /dev/null -X POST http://localhost:8080 \
  -d 'motor=octave&v0=50&angulo=45&met=rk4'
```

Python.

```
time env PARAM_FILE=/tmp/params_bench.txt \
  RESULT_FILE=/tmp/resultado_bench.html python3 calcular.py
time curl -s -o /dev/null -X POST http://localhost:8080 \
  -d 'motor=python&v0=50&angulo=45&met=rk4'
```

Perl.

```
time env PARAM_FILE=/tmp/params_bench.txt \
  RESULT_FILE=/tmp/resultado_bench.html perl calcular.pl
time curl -s -o /dev/null -X POST http://localhost:8080 \
  -d 'motor=perl&v0=50&angulo=45&met=rk4'
```

JavaScript.

```
time env PARAM_FILE=/tmp/params_bench.txt \
  RESULT_FILE=/tmp/resultado_bench.html node calcular.js
time curl -s -o /dev/null -X POST http://localhost:8080 \
  -d 'motor=javascript&v0=50&angulo=45&met=rk4'
```

C. Compile antes de medir; el tiempo de compilación no pertenece al tiempo de respuesta del motor ya instalado.

```
gcc calcular.c -O2 -o calcular -lm
time env PARAM_FILE=/tmp/params_bench.txt \
  RESULT_FILE=/tmp/resultado_bench.html ./calcular
time curl -s -o /dev/null -X POST http://localhost:8080 \
  -d 'motor=c&v0=50&angulo=45&met=rk4'
```

Bash.

```
time env PARAM_FILE=/tmp/params_bench.txt \
  RESULT_FILE=/tmp/resultado_bench.html bash calcular.sh
time curl -s -o /dev/null -X POST http://localhost:8080 \
  -d 'motor=bash&v0=50&angulo=45&met=rk4'
```

PowerShell.

```
time env PARAM_FILE=/tmp/params_bench.txt \
  RESULT_FILE=/tmp/resultado_bench.html pwsh -File
     calcular.ps1
time curl -s -o /dev/null -X POST http://localhost:8080 \
  -d 'motor=powershell&v0=50&angulo=45&met=rk4'
```

La resta *tiempo HTTP menos tiempo directo* no es una descomposición exacta, porque las mediciones no ocurren dentro de la misma ejecución, pero sirve para estimar si domina el arranque del motor o el resto de la cadena.

10. **Ejercicio:** Diseño de una aplicación web científica

    Diseñe y construya desde cero una aplicación web de cálculo científico sobre el mismo `servidor.py`. El servidor no debe modificarse.

    Requisitos mínimos:

    - al menos cinco campos de distinto tipo en `formulario.html`;
    - una ecuación diferencial resuelta con al menos dos métodos;
    - al menos una gráfica generada con gnuplot;
    - lectura exclusiva desde `PARAM_FILE`;
    - escritura exclusiva en `RESULT_FILE`;
    - manejo de errores visible en el navegador.

    Modelo de respuesta: crecimiento logístico.

    Como ejemplo, use $\dot{y} = ry(1 - y/K)$, con `r`, `K`, `y0`, `tmax` y `metodo` en el formulario. Compare Euler con Runge–Kutta de orden 4. Para una ecuación escalar, si $f(y) = ry(1 - y/K)$, RK4 usa

    $$k_1 = f(y), \quad k_2 = f(y + hk_1/2), \quad k_3 = f(y + hk_2/2), \quad k_4 = f(y + hk_3),$$

    $$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

    Scilab.

```
function q=f(y,r,K), q=r*y*(1-y/K), endfunction
if metodo=='euler' then
    y=y+dt*f(y,r,K)
else
    k1=f(y,r,K); k2=f(y+dt*k1/2,r,K)
    k3=f(y+dt*k2/2,r,K); k4=f(y+dt*k3,r,K)
    y=y+dt*(k1+2*k2+2*k3+k4)/6
end
```

    Octave.

```
f=@(q) r*q*(1-q/K);
if strcmp(metodo,'euler')
  y=y+dt*f(y);
else
  k1=f(y); k2=f(y+dt*k1/2); k3=f(y+dt*k2/2); k4=f(y+dt*k3);
  y=y+dt*(k1+2*k2+2*k3+k4)/6;
end
```

Python.

```
def f(q): return r*q*(1-q/K)
if metodo == 'euler':
    y += dt*f(y)
else:
    k1=f(y); k2=f(y+dt*k1/2); k3=f(y+dt*k2/2); k4=f(y+dt*k3)
    y += dt*(k1+2*k2+2*k3+k4)/6
```

Perl.

```
sub f { my($q)=@_; return $r*$q*(1-$q/$K); }
if ($metodo eq 'euler') { $y += $dt*f($y); }
else {
  my $k1=f($y); my $k2=f($y+$dt*$k1/2);
  my $k3=f($y+$dt*$k2/2); my $k4=f($y+$dt*$k3);
  $y += $dt*($k1+2*$k2+2*$k3+$k4)/6;
}
```

JavaScript.

```
const f=q => r*q*(1-q/K);
if (metodo === 'euler') y += dt*f(y);
else {
  const k1=f(y), k2=f(y+dt*k1/2), k3=f(y+dt*k2/2),
     k4=f(y+dt*k3);
  y += dt*(k1+2*k2+2*k3+k4)/6;
}
```

C.

```
double f(double q,double r,double K){ return r*q*(1.0-q/K); }
if (!strcmp(metodo,"euler")) y += dt*f(y,r,K);
else {
    double k1=f(y,r,K), k2=f(y+dt*k1/2,r,K);
    double k3=f(y+dt*k2/2,r,K), k4=f(y+dt*k3,r,K);
    y += dt*(k1+2*k2+2*k3+k4)/6;
}
```

Bash. `awk` puede ejecutar los dos métodos con aritmética real.

```
y=$(awk -v y="$y" -v r="$r" -v K="$K" -v h="$dt" -v
   m="$metodo" '
function f(q){return r*q*(1-q/K)}
BEGIN{
  if(m=="euler") y=y+h*f(y);
  else {k1=f(y);k2=f(y+h*k1/2);k3=f(y+h*k2/2);k4=f(y+h*k3);
        y=y+h*(k1+2*k2+2*k3+k4)/6}
  print y
}')
```

PowerShell.

```
function F([double]$q) { $r*$q*(1-$q/$K) }
if ($metodo -eq 'euler') { $y += $dt*(F $y) }
else {
  $k1=F $y; $k2=F ($y+$dt*$k1/2)
  $k3=F ($y+$dt*$k2/2); $k4=F ($y+$dt*$k3)
  $y += $dt*($k1+2*$k2+2*$k3+$k4)/6
}
```

En todos los lenguajes, ejecute ambos métodos con el mismo paso, escriba `t y_Euler y_RK4` en un archivo de datos, grafique las dos curvas con gnuplot y escriba el resumen y la gráfica en `RESULT_FILE`. La comparación entre ambas soluciones forma parte de la respuesta.

11. **Ejercicio:** Oscilador web con motor de cálculo y Gnuplot

    Construya una aplicación web para un oscilador utilizando el servidor actual y un motor elegido por el usuario. La consigna no fija el lenguaje del motor.

    a) El formulario debe incluir tipo de oscilador, masa, constante del resorte, amortiguamiento, fuerza, condiciones iniciales, tiempo de simulación y tipo de gráfica.

    b) El motor debe leer `clave=valor` desde `PARAM_FILE`.

    c) Resuelva el sistema $\dot{x} = v$, $\dot{v} = (F(t) - cv - kx)/m$ con RK4.

    d) Escriba los datos y genere con gnuplot al menos posición versus tiempo y retrato de fase.

    e) Escriba estadísticas y la gráfica en `RESULT_FILE`.

    f) `servidor.py` no se modifica.

Respuestas por lenguaje.

Suponga ya leídos `m`, `c`, `k`, `F0`, `omega`, `x0`, `v0`, `h` y `tmax` con el lector de parámetros de los ejercicios anteriores. Los siguientes programas implementan el paso RK4 completo y producen columnas `t x v energia`.

Scilab.

```
function dz=der(t,z,m,c,k,F0,w)
  F=F0*cos(w*t); dz=[z(2);(F-c*z(2)-k*z(1))/m]
endfunction
function zn=rk4(t,z,h,m,c,k,F0,w)
  k1=der(t,z,m,c,k,F0,w)
  k2=der(t+h/2,z+h*k1/2,m,c,k,F0,w)
  k3=der(t+h/2,z+h*k2/2,m,c,k,F0,w)
  k4=der(t+h,z+h*k3,m,c,k,F0,w)
  zn=z+h*(k1+2*k2+2*k3+k4)/6
endfunction
fd=mopen('/tmp/oscilador.dat','w'); z=[x0;v0]
for t=0:h:tmax
  E=.5*m*z(2)^2+.5*k*z(1)^2
```

```
  mfprintf(fd,'%.9g %.9g %.9g %.9g\n',t,z(1),z(2),E)
  z=rk4(t,z,h,m,c,k,F0,omega)
end
mclose(fd)
```

Octave.

```
f=@(t,z) [z(2);(F0*cos(omega*t)-c*z(2)-k*z(1))/m];
z=[x0;v0]; fd=fopen('/tmp/oscilador.dat','w');
for t=0:h:tmax
  E=.5*m*z(2)^2+.5*k*z(1)^2; fprintf(fd,'%.9g %.9g %.9g
     %.9g\n',t,z(1),z(2),E);
  k1=f(t,z); k2=f(t+h/2,z+h*k1/2); k3=f(t+h/2,z+h*k2/2);
     k4=f(t+h,z+h*k3);
  z=z+h*(k1+2*k2+2*k3+k4)/6;
end
fclose(fd);
```

Python.

```
def f(t,z):
    x,v=z; return (v,(F0*math.cos(omega*t)-c*v-k*x)/m)
def add(z,a,q): return (z[0]+a*q[0],z[1]+a*q[1])
def rk4(t,z):
    k1=f(t,z); k2=f(t+h/2,add(z,h/2,k1));
       k3=f(t+h/2,add(z,h/2,k2)); k4=f(t+h,add(z,h,k3))
    return (z[0]+h*(k1[0]+2*k2[0]+2*k3[0]+k4[0])/6,
            z[1]+h*(k1[1]+2*k2[1]+2*k3[1]+k4[1])/6)
z=(x0,v0)
with open('/tmp/oscilador.dat','w') as fd:
    t=0.0
    while t <= tmax+1e-12:
        E=.5*m*z[1]**2+.5*k*z[0]**2; fd.write(f'{t} {z[0]}
           {z[1]} {E}\n')
        z=rk4(t,z); t+=h
```

Perl.

```
sub f { my($t,$x,$v)=@_; return
   ($v,($F0*cos($omega*$t)-$c*$v-$k*$x)/$m); }
sub rk4 {
  my($t,$x,$v)=@_; my($a1,$b1)=f($t,$x,$v);
  my($a2,$b2)=f($t+$h/2,$x+$h*$a1/2,$v+$h*$b1/2);
  my($a3,$b3)=f($t+$h/2,$x+$h*$a2/2,$v+$h*$b2/2);
  my($a4,$b4)=f($t+$h,$x+$h*$a3,$v+$h*$b3);
  return ($x+$h*($a1+2*$a2+2*$a3+$a4)/6,
          $v+$h*($b1+2*$b2+2*$b3+$b4)/6);
}
open my $fd,'>','/tmp/oscilador.dat' or die $!;
   my($x,$v)=($x0,$v0);
```

```
for(my $t=0;$t<=$tmax+1e-12;$t+=$h){ my
   $E=.5*$m*$v*$v+.5*$k*$x*$x; print $fd "$t $x $v $E\n";
   ($x,$v)=rk4($t,$x,$v); }
close $fd;
```

JavaScript.

```
const f=(t,z)=>[z[1],(F0*Math.cos(omega*t)-c*z[1]-k*z[0])/m];
const add=(z,a,q)=>[z[0]+a*q[0],z[1]+a*q[1]];
function rk4(t,z){
  const k1=f(t,z), k2=f(t+h/2,add(z,h/2,k1)),
     k3=f(t+h/2,add(z,h/2,k2)), k4=f(t+h,add(z,h,k3));
  return [z[0]+h*(k1[0]+2*k2[0]+2*k3[0]+k4[0])/6,
          z[1]+h*(k1[1]+2*k2[1]+2*k3[1]+k4[1])/6];
}
let z=[x0,v0], s='';
for(let t=0;t<=tmax+1e-12;t+=h){ const
   E=.5*m*z[1]*z[1]+.5*k*z[0]*z[0]; s+=`${t} ${z[0]} ${z[1]}
   ${E}\n`; z=rk4(t,z); }
fs.writeFileSync('/tmp/oscilador.dat',s);
```

C.

```
typedef struct { double x,v; } Z;
Z der(double t,Z z){ Z
   q={z.v,(F0*cos(omega*t)-c*z.v-k*z.x)/m}; return q; }
Z add(Z z,double a,Z q){ Z r={z.x+a*q.x,z.v+a*q.v}; return
   r; }
Z rk4(double t,Z z){
  Z a=der(t,z), b=der(t+h/2,add(z,h/2,a)),
     d=der(t+h/2,add(z,h/2,b)), e=der(t+h,add(z,h,d));
  Z
     r={z.x+h*(a.x+2*b.x+2*d.x+e.x)/6,z.v+h*(a.v+2*b.v+2*d.v+e.v)/6};
     return r;
}
FILE *fd=fopen("/tmp/oscilador.dat","w"); Z z={x0,v0};
for(double t=0;t<=tmax+1e-12;t+=h){ double
   E=.5*m*z.v*z.v+.5*k*z.x*z.x; fprintf(fd,"%.9g %.9g %.9g
   %.9g\n",t,z.x,z.v,E); z=rk4(t,z); }
fclose(fd);
```

Bash. En Bash conviene delegar todo el bucle numérico a una sola invocación de `awk`; lanzar un proceso por paso sería innecesariamente costoso.

```
awk -v m="$m" -v c="$c" -v k="$k" -v F0="$F0" -v w="$omega" \
    -v x="$x0" -v v="$v0" -v h="$h" -v T="$tmax" '
function ax(t,x,v){return (F0*cos(w*t)-c*v-k*x)/m}
BEGIN{for(t=0;t<=T+1e-12;t+=h){
 E=.5*m*v*v+.5*k*x*x; print t,x,v,E;
 a1=v; b1=ax(t,x,v); a2=v+h*b1/2;
    b2=ax(t+h/2,x+h*a1/2,v+h*b1/2);
```

```
  a3=v+h*b2/2; b3=ax(t+h/2,x+h*a2/2,v+h*b2/2); a4=v+h*b3;
     b4=ax(t+h,x+h*a3,v+h*b3);
  x+=h*(a1+2*a2+2*a3+a4)/6; v+=h*(b1+2*b2+2*b3+b4)/6;
}}' > /tmp/oscilador.dat
```

PowerShell.

```
function Der($t,$z){
   @($z[1],($F0*[Math]::Cos($omega*$t)-$c*$z[1]-$k*$z[0])/$m)
   }
function Add($z,$a,$q){ @($z[0]+$a*$q[0],$z[1]+$a*$q[1]) }
function RK4($t,$z){
  $k1=Der $t $z; $k2=Der ($t+$h/2) (Add $z ($h/2) $k1)
  $k3=Der ($t+$h/2) (Add $z ($h/2) $k2); $k4=Der ($t+$h)
     (Add $z $h $k3)
  @($z[0]+$h*($k1[0]+2*$k2[0]+2*$k3[0]+$k4[0])/6,
    $z[1]+$h*($k1[1]+2*$k2[1]+2*$k3[1]+$k4[1])/6)
}
Remove-Item /tmp/oscilador.dat -ErrorAction Ignore;
   $z=@($x0,$v0)
for($t=0.0;$t -le $tmax+1e-12;$t+=$h){
   $E=.5*$m*$z[1]*$z[1]+.5*$k*$z[0]*$z[0]; "$t $($z[0])
   $($z[1]) $E" | Add-Content /tmp/oscilador.dat; $z=RK4 $t
   $z }
```

El archivo de datos tiene el mismo formato en todos los motores. Un script gnuplot común puede producir simultáneamente la historia temporal y el retrato de fase:

```
set terminal svg size 900,420
set output '/tmp/oscilador.svg'
set multiplot layout 1,2
set grid
set xlabel 't'; set ylabel 'x'
plot '/tmp/oscilador.dat' using 1:2 with lines title 'x(t)'
set xlabel 'x'; set ylabel 'v'
plot '/tmp/oscilador.dat' using 2:3 with lines title 'fase'
unset multiplot
```

El motor debe leer este SVG y escribirlo, junto con las estadísticas pedidas, en `RESULT_FILE`.

12. **Ejercicio:** Archivos estáticos entregados por el servidor

    Extienda `servidor.py` para que sirva archivos estáticos (CSS, imágenes) desde el mismo directorio, sin modificar su lógica principal. Este ejercicio modifica el servidor HTTP, no el motor de cálculo; por eso la respuesta corresponde al lenguaje en que está escrito `servidor.py`.

    Python. Una extensión mínima de `do_GET` puede distinguir la página principal de los archivos estáticos.

```
def do_GET(self):
    if self.path == '/' or self.path == '/index.html':
        html = leer_form().replace(MARCADOR,
                   '<div id=resultados></div>')
        self._enviar(html)
    elif self.path.endswith(('.css','.png')):
        archivo = os.path.join(DIR,self.path.lstrip('/'))
        if not os.path.exists(archivo):
            self.send_error(404); return
        tipo = 'text/css' if archivo.endswith('.css') else
           'image/png'
        with open(archivo,'rb') as f: data=f.read()
        self.send_response(200)
        self.send_header('Content-Type',tipo)
        self.send_header('Content-Length',len(data))
        self.end_headers(); self.wfile.write(data)
    else:
        self.send_error(404)
```

a) Mueva el CSS a `estilo.css` y enlácelo desde el formulario.

b) Verifique que el estilo se aplica correctamente.

c) Agregue un PNG estático y compruebe que se sirve con el tipo MIME correcto.

13. **Ejercicio:** Preguntas y problemas

a) ¿Por qué se dice que HTTP es un protocolo *stateless*?

b) ¿Dónde está el "estado" de la comunicación? ¿Qué pasaría si dos clientes envían solicitudes simultáneamente?

c) ¿Qué mecanismos usan las aplicaciones web reales para mantener estado entre peticiones?

d) El servidor actual usa nombres fijos como `/tmp/params.txt` y `/tmp/resultado.html`. ¿Qué problema aparecería si atendiera dos solicitudes simultáneamente? Proponga una forma de aislar los archivos de cada solicitud, por ejemplo mediante un directorio temporal o un identificador ùnico por petición.

Abra `http://localhost:8080` y compare GET con POST:

```
curl http://localhost:8080
curl -X POST http://localhost:8080 \
     -d
        "v0=80&angulo=30&arr=bajo&met=rk4&g_tray=on&g_vel=off"
```

8.10. Ejercicios para interfaces gráficas

En esta sección los problemas se formulan para cualquier interfaz gráfica de las estudiadas en el tutorial. El material específico de Scilab que ya formaba parte de la guía se conserva

íntegramente como implementación de referencia; para los otros lenguajes se pide reproducir la misma funcionalidad con su interfaz correspondiente. Por lo tanto, generalizar no significa reemplazar ni resumir los ejemplos existentes.

1. **Ejercicio:** Diálogos gráficos

   Consigna general. Explore en la interfaz gráfica del lenguaje elegido las operaciones de mensaje, confirmación, entrada de texto, selección de archivo y selección de color. Determine los valores devueltos, trate la cancelación y compare los diálogos modales con una ventana persistente.

   Implementación de referencia en Scilab (material original).

   Explore los diálogos predefinidos de Scilab ejecutando el siguiente programa y describiendo qué hace cada función:

```
// Dialogo de mensaje simple
messagebox('Bienvenido al laboratorio de fisica', ...
           'Inicio', 'info')

// Dialogo de pregunta si/no
respuesta = messagebox('Desea continuar con la simulacion?',
   ...
                       'Confirmacion', 'question', ...
                       ['Si' 'No'])
if respuesta == 1 then
    disp('El usuario eligio: Si')
else
    disp('El usuario eligio: No')
end

// Entrada de texto
nombre = input_dialog({'Ingrese su nombre:' ...
                       'Ingrese su legajo:'}, ...
                      'Identificacion')
printf('Usuario: %s, Legajo: %s\n', nombre(1), nombre(2))

// Seleccion de archivo
archivo = uigetfile({'*.txt', 'Archivos de texto'; ...
                     '*.csv', 'Archivos CSV'; ...
                     '*.*',   'Todos los archivos'}, ...
                    '', 'Seleccione el archivo de datos')
if archivo <> '' then
    printf('Archivo seleccionado: %s\n', archivo)
end

// Seleccion de color
color = uigetcolor('Elija el color de la curva')
disp(color)
```

   a) ¿Qué tipo de valor devuelve cada función de diálogo?

b) ¿Qué ocurre si el usuario cierra el diálogo sin responder? Verifíquelo y agregue validación.

c) ¿En qué situaciones es preferible pedir parámetros mediante diálogos en lugar de leerlos con `input` o `mscanf`?

Equivalencias en otras interfaces.

Octave. Use cuadros de diálogo equivalentes de Octave, como `msgbox`, `questdlg`, `inputdlg`, `uigetfile` y `uisetcolor`.

Python. Con Tkinter use `messagebox`, `simpledialog`, `filedialog` y `colorchooser`.

Perl. Con Perl/Tk use los diálogos y selectores provistos por Tk y compruebe el valor devuelto al cancelar.

JavaScript. En el navegador combine `alert`, `confirm`, `prompt`, `input type=file` e `input type=color`.

C. Con GTK utilice cuadros de diálogo y selectores y procese la respuesta mediante señales/callbacks.

Bash. Con Zenity use `-info`, `-question`, `-entry`, `-file-selection` y `-color-selection`.

PowerShell. Con Windows Forms use `MessageBox`, `OpenFileDialog`, `ColorDialog` y controles de entrada de texto.

2. **Ejercicio:** Ventana para seno y coseno

Consigna general. Construya con la interfaz del lenguaje elegido una ventana que permita seleccionar seno, coseno o ambas funciones y graficarlas. El archivo `ventana.sce` mencionado a continuación se conserva como la referencia específica de Scilab.

Implementación de referencia en Scilab (material original).

Ejecute el archivo `ventana.sce` proporcionado con esta guía y analice la salida de los distintos `uicontrols`. Modifique el archivo dado para escribir un programa que mediante un entorno gráfico permita elegir graficar la función seno, la función coseno, o ambas en el mismo gráfico.

Equivalencias en otras interfaces.

Octave. Reproduzca la funcionalidad de `ventana.sce` con `figure`, `uicontrol` y callbacks de Octave.

Python. Con Tkinter/ttk use botones de opción o `Combobox` y un callback que actualice la gráfica.

Perl. Con Perl/Tk use botones de opción/listas y un callback para elegir la curva.

JavaScript. Use controles HTML y eventos del DOM; dibuje en `canvas` o SVG.

C. Con GTK use botones o una lista y una señal que actualice un área de dibujo o una imagen generada.

Bash. Use Zenity para elegir seno, coseno o ambas y genere la gráfica con Gnuplot u otra herramienta externa.

PowerShell. Con Windows Forms use `RadioButton` o `ComboBox` y un evento de botón para actualizar la representación.

3. **Ejercicio:** Visualizador interactivo de funciones

   Consigna general. Realice el mismo visualizador interactivo con la interfaz correspondiente al lenguaje elegido: controles para parámetros, eventos/callbacks, actualización de la gráfica y exportación.

   Implementación de referencia en Scilab (material original).

   Construya una GUI completa en Scilab para visualizar y controlar una función matemática en tiempo real:

```
function gui_funcion()
    // Crear la figura principal
    f = figure('Figure_name', 'Visualizador de funciones',
       ...
              'position', [100, 100, 800, 600], ...
              'BackgroundColor', [0.9, 0.9, 0.9])

    // Area de grafico (axes)
    ax = newaxes()
    ax.axes_bounds = [0.05, 0.25, 0.90, 0.72]

    // Control deslizante para la frecuencia
    h_freq = uicontrol('style', 'slider', ...
                       'position', [50, 80, 300, 25], ...
                       'min', 0.1, 'max', 5.0, 'value', 1.0,
                          ...
                       'callback', 'actualizar_grafico()')

    // Etiqueta del slider
    uicontrol('style', 'text', ...
              'position', [50, 110, 300, 20], ...
              'string', 'Frecuencia [Hz]', ...
              'BackgroundColor', [0.9, 0.9, 0.9])

    // Control deslizante para la amplitud
    h_amp = uicontrol('style', 'slider', ...
                      'position', [400, 80, 300, 25], ...
                      'min', 0.1, 'max', 3.0, 'value', 1.0,
                         ...
                      'callback', 'actualizar_grafico()')
```

```
    uicontrol('style', 'text', ...
              'position', [400, 110, 300, 20], ...
              'string', 'Amplitud', ...
              'BackgroundColor', [0.9, 0.9, 0.9])

    // Boton para cambiar la funcion
    uicontrol('style', 'pushbutton', ...
              'position', [50, 30, 120, 35], ...
              'string', 'Seno', ...
              'callback', 'set_funcion(1)')

    uicontrol('style', 'pushbutton', ...
              'position', [190, 30, 120, 35], ...
              'string', 'Coseno', ...
              'callback', 'set_funcion(2)')

    uicontrol('style', 'pushbutton', ...
              'position', [330, 30, 120, 35], ...
              'string', 'Tangente', ...
              'callback', 'set_funcion(3)')

    // Boton para exportar el grafico
    uicontrol('style', 'pushbutton', ...
              'position', [580, 30, 150, 35], ...
              'string', 'Exportar PNG', ...
              'callback', 'exportar_grafico()')

    // Dibujar el grafico inicial
    actualizar_grafico()
endfunction

function actualizar_grafico()
    // Leer valores de los controles
    h = findobj('style', 'slider')
    freq = h(2).value
    amp  = h(1).value

    // Calcular y graficar
    t = linspace(0, 2, 500)
    tipo = evstr(get_tipo_funcion())

    select tipo
    case 1 then y = amp * sin(2*%pi*freq*t)
    case 2 then y = amp * cos(2*%pi*freq*t)
    case 3 then y = amp * tan(2*%pi*freq*t)
    end

    // Limitar rango para tangente
    y(abs(y) > 3*amp) = %nan

    sca(findobj('type', 'Axes'))
```

```
    cla()
    plot(t, y, 'b-', 'LineWidth', 2)
    xgrid()
    xlabel('Tiempo [s]')
    ylabel('Amplitud')
    title(sprintf('f=%.2f Hz, A=%.2f', freq, amp))
    a = gca()
    a.data_bounds = [0, -3.5; 2, 3.5]
endfunction
```

a) Ejecute la GUI y mueva los controles deslizantes. ¿El gráfico se actualiza en tiempo real?

b) Agregue un control deslizante para la fase inicial φ en $[0, 2\pi]$.

c) Agregue un campo de texto donde el usuario pueda escribir su propia expresión matemática.

d) ¿Qué es un evento en el contexto de interfaces gráficas? ¿Qué evento dispara `actualizar_grafico()` cuando se mueve el slider?

Equivalencias en otras interfaces.

Octave. Use sliders, botones y ejes mediante `figure`/`uicontrol`; los callbacks actualizan la gráfica.

Python. Con Tkinter/ttk use `Scale`, `Combobox`, `Entry` y callbacks; mantenga el cálculo de la función separado de los widgets.

Perl. Con Perl/Tk use `Scale`, listas, entradas y callbacks.

JavaScript. Use `input type=range`, `select` y eventos `input`/`change`; actualice `canvas` o SVG.

C. Con GTK use escalas, entradas y botones conectados mediante señales.

Bash. Zenity no ofrece una ventana persistente equivalente; use diálogos sucesivos para cambiar parámetros y regenere la gráfica después de cada confirmación.

PowerShell. Con Windows Forms use `TrackBar`, `ComboBox`, `TextBox` y eventos de cambio.

4. **Ejercicio:** Simulador interactivo de proyectiles

Consigna general. Implemente el simulador con la interfaz gráfica del lenguaje elegido, manteniendo separado el modelo numérico de la capa de interfaz.

Implementación de referencia en Scilab (material original).

Instrumente una simulación interactiva del movimiento de un proyectil con controles GUI:

```
function simulador_proyectil()
    f = figure('Figure_name', 'Simulador de proyectil', ...
               'position', [50, 50, 900, 650])

    // Panel de parametros
    uicontrol('style', 'text', 'string', 'Velocidad inicial
       [m/s]:', ...
              'position', [10, 580, 180, 20])
    h_v0 = uicontrol('style', 'edit', 'string', '50', ...
                     'position', [10, 555, 180, 25])

    uicontrol('style', 'text', 'string', 'Angulo [grados]:',
       ...
              'position', [10, 520, 180, 20])
    h_ang = uicontrol('style', 'slider', ...
                      'position', [10, 495, 180, 25], ...
                      'min', 1, 'max', 89, 'value', 45)

    uicontrol('style', 'text', 'string', 'Resistencia del
       aire:', ...
              'position', [10, 460, 180, 20])
    h_b = uicontrol('style', 'slider', ...
                    'position', [10, 435, 180, 25], ...
                    'min', 0, 'max', 0.5, 'value', 0)

    // Botones
    uicontrol('style', 'pushbutton', ...
              'string', 'Lanzar', ...
              'position', [10, 390, 85, 35], ...
              'callback', 'lanzar_proyectil()')

    uicontrol('style', 'pushbutton', ...
              'string', 'Limpiar', ...
              'position', [105, 390, 85, 35], ...
              'callback', 'clf_proyectil()')

    // Etiquetas de resultados
    uicontrol('style', 'text', 'tag', 'resultado', ...
              'string', 'Alcance: --   Altura max: --', ...
              'position', [10, 350, 200, 40], ...
              'BackgroundColor', [0.8, 1.0, 0.8])

    // Area de grafico
    ax = newaxes()
    ax.axes_bounds = [0.27, 0.05, 0.71, 0.92]
    xlabel('Distancia horizontal [m]')
    ylabel('Altura [m]')
    title('Trayectoria del proyectil')
    xgrid()
endfunction
```

```
function lanzar_proyectil()
    // Leer parametros
    v0  = strtod(get(findobj('style','edit'), 'string'))
    ang = get(findobj('style','slider','min',1), 'value')
    b   = get(findobj('style','slider','min',0), 'value')

    ang_rad = ang * %pi / 180
    g   = 9.8
    m   = 1.0

    // Condiciones iniciales: [x, y, vx, vy]
    y0 = [0; 0; v0*cos(ang_rad); v0*sin(ang_rad)]

    // EDO del proyectil con resistencia
    function dydt = f_proyectil(t, y)
        vx = y(3); vy = y(4)
        v  = sqrt(vx^2 + vy^2)
        dydt = [vx; vy; -(b/m)*vx; -g-(b/m)*vy]
    endfunction

    // Integrar hasta que y < 0 (suelo)
    t_max = 2*v0*sin(ang_rad)/g * 1.5
    t = linspace(0, t_max, 1000)
    sol = ode(y0, 0, t, f_proyectil)

    x_tray = sol(1,:)
    y_tray = sol(2,:)

    // Encontrar donde toca el suelo
    idx = find(y_tray < 0)
    if ~isempty(idx) then
        n = idx(1)
        x_tray = x_tray(1:n)
        y_tray = y_tray(1:n)
    end

    // Estadisticas
    alcance = x_tray($)
    h_max   = max(y_tray)

    // Graficar
    sca(findobj('type','Axes'))
    plot(x_tray, y_tray, 'LineWidth', 2)
    plot(x_tray($), 0, 'ro', 'MarkerSize', 8)

    // Actualizar resultado
    msg = sprintf('Alcance: %.1f m   Altura max: %.1f m', ...
                  alcance, h_max)
    set(findobj('tag','resultado'), 'string', msg)
endfunction
```

a) Ejecute el simulador y lance proyectiles con distintos ángulos. ¿Qué ángulo maximiza el alcance sin resistencia del aire?

b) Aumente gradualmente la resistencia del aire. ¿Cómo cambia el ángulo óptimo?

c) Agregue un checkbox para mostrar u ocultar la trayectoria sin resistencia del aire como referencia.

d) Agregue una etiqueta que muestre en tiempo real el valor del ángulo mientras se mueve el slider, sin necesidad de presionar "Lanzar".

e) ¿Qué ocurre si el usuario ingresa un valor no numérico en el campo de velocidad? Agregue validación.

Equivalencias en otras interfaces.

Octave. Use campos, sliders, botones y ejes de Octave; el callback llama a una función numérica independiente.

Python. Con Tkinter/ttk use `Entry`, `Scale`, `Checkbutton`, etiquetas y botones.

Perl. Con Perl/Tk use entradas, escalas, checkbox y botones.

JavaScript. Use controles HTML y eventos del DOM; el modelo numérico devuelve los puntos y la interfaz actualiza la gráfica y las estadísticas.

C. Con GTK use entradas, escalas, checkbox, etiquetas y botones conectados a una rutina numérica independiente de GTK.

Bash. Use un formulario de Zenity para parámetros, ejecute el cálculo y genere la gráfica externamente; la interacción será por ciclos de diálogo.

PowerShell. Con Windows Forms use cajas de texto, `TrackBar`, `CheckBox`, botones y etiquetas.

5. **Ejercicio:** Regresión lineal o exponencial con interfaz gráfica

Escriba un programa que permita leer una serie de datos (x, y) desde un archivo de texto, obtener la curva de regresión lineal o exponencial y graficar los datos y la curva obtenida. La selección de la operación que realice el programa debe hacerse a partir de una ventana gráfica.

Orientaciones por interfaz.

Scilab. Use un selector de archivo y `uicontrol` para elegir el tipo de regresión; mantenga el ajuste numérico separado de la GUI.

Octave. Use selector de archivo, controles de selección y callbacks.

Python. Con Tkinter use `filedialog`, botones de opción o `Combobox` y etiquetas para mostrar coeficientes.

Perl. Con Perl/Tk use selector de archivo, botones de opción y callbacks.

JavaScript. Use `input type=file`, controles de selección y una gráfica en `canvas` o SVG.

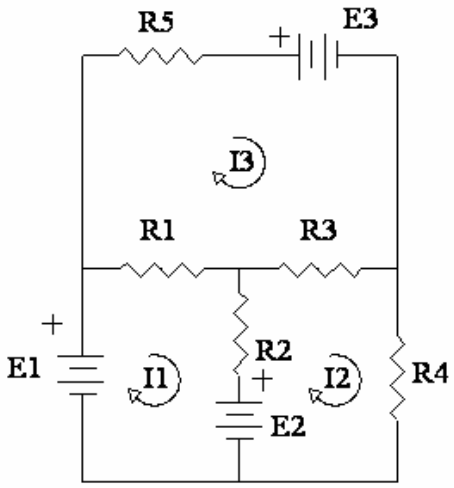
C. Con GTK use selector de archivo, controles de selección y un botón.

Bash. Use Zenity para seleccionar archivo y modelo; muestre luego los parámetros y la gráfica.

PowerShell. Use `OpenFileDialog`, radio buttons o `ComboBox` y controles de salida.

6. **Ejercicio:** Corrientes de un circuito con interfaz gráfica

   Realizar un programa que encuentre el valor de las corrientes del siguiente circuito eléctrico. Producir las entradas y las salidas en la pantalla mediante una ventana gráfica adecuada.



Utilizando la ley de voltajes de Kirchhoff (LVK) se obtiene el siguiente sistema de ecuaciones:

$$-E1+E2+(R1+R2)*I1-R2*I2-R1*I3=0$$
$$-E2-R2*I1+(R2+R3+R4)*I2-R3*I3=0$$
$$E3-R1*I1-R3*I2+(R1+R3+R5)*I3=0$$

Coloque los siguientes valores de resistencias y voltajes por defecto:

$$R1=4\Omega \quad R2=8\Omega \quad R3=7\Omega \quad R4=9\Omega \quad R5=2\Omega$$
$$E1=9V \quad E2=6V \quad E3=8V$$

Para los cuales la solución del sistema de ecuaciones debe ser:

$$I_1 = 0{,}2446381, \qquad I_2 = 0{,}2064343, \qquad I_3 = 0{,}4289544.$$

Orientaciones por interfaz.

Scilab. Use `uicontrol` para entradas y etiquetas; el callback arma y resuelve el sistema y actualiza las corrientes.

Octave. Use entradas y etiquetas en una `figure`; el callback construye y resuelve el sistema lineal.

Python. Con Tkinter use `Entry` para resistencias y voltajes y etiquetas para las corrientes.

Perl. Con Perl/Tk use entradas y etiquetas; encapsule la resolución del sistema en una función independiente.

JavaScript. Use un formulario HTML y escriba los resultados en elementos del DOM.

C. Con GTK use entradas, etiquetas y una función C separada para resolver el sistema.

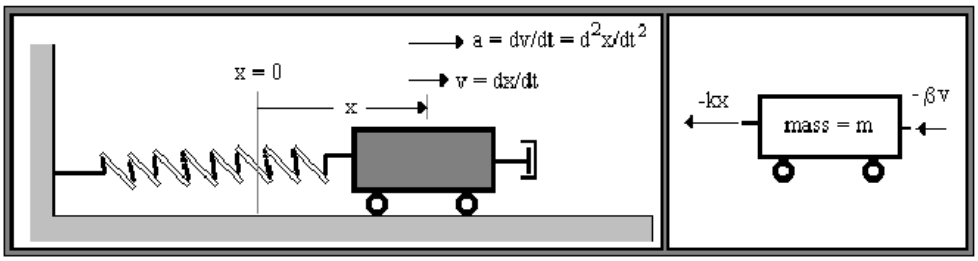
Bash. Use un formulario de Zenity y delegue la aritmética matricial a la herramienta numérica disponible.

PowerShell. Use `TextBox` y etiquetas; el evento del botón valida, resuelve y actualiza la ventana.

7. **Ejercicio:** Sistema masa-resorte con interfaz gráfica

   Considere el sistema masa-resorte que se muestra en la figura. Se saca la masa m de su posición de equilibrio ($x = 0$) y al tiempo $t = 0$, se suelta en la posición $x = x_0$. En ese instante el cuerpo se mueve con velocidad $v = v_0$. Además, sobre la masa actúa la fuerza de restauración del resorte, $F_s = -kx$, y la fuerza de roce, $F_v = -\beta v = -\beta(dx/dt)$. Aplicando al sistema la segunda ley de Newton se obtiene:

$$-kx - \beta\frac{dx}{dt} = m\frac{d^2x}{dt^2}.$$



Confeccione un programa que resuelva y grafique la variación de la posición, velocidad y aceleración del sistema masa-resorte. El programa debe tener un entorno gráfico y los `uicontrols` que correspondan para ingresar los valores de los parámetros. Ensaye el programa con las siguientes entradas para dichos parámetros:

(a) $m = 2\,\text{kg}$, $\beta = 0{,}01\,\text{Ns/m}$, $k = 2\,\text{N/m}$, $x_0 = 0{,}2\,\text{m}$, $v_0 = 1{,}2\,\text{m/s}$.

(b) $m = 4\,\text{kg}$, $\beta = 0{,}10\,\text{Ns/m}$, $k = 2\,\text{N/m}$, $x_0 = 0{,}2\,\text{m}$, $v_0 = 1{,}2\,\text{m/s}$.

(c) $m = 1\,\text{kg}$, $\beta = 0{,}02\,\text{Ns/m}$, $k = 2\,\text{N/m}$, $x_0 = 0{,}2\,\text{m}$, $v_0 = 1{,}2\,\text{m/s}$.

(d) $m = 0{,}5\,\text{kg}$, $\beta = 0{,}25\,\text{Ns/m}$, $k = 2\,\text{N/m}$, $x_0 = 0{,}2\,\text{m}$, $v_0 = 1{,}2\,\text{m/s}$.

Generalización de la interfaz.

En lenguajes distintos de Scilab, la expresión `uicontrols` del enunciado original debe entenderse como los controles equivalentes de la interfaz elegida.

Scilab. Conserve los `uicontrols` indicados en el enunciado.

Octave. Use `figure`/`uicontrol` y callbacks.

Python. Con Tkinter/ttk use entradas, checkboxes o listas y botones; la rutina de integración debe devolver t, x, v, a sin depender de la GUI.

Perl. Con Perl/Tk use entradas, controles de selección y callbacks.

JavaScript. Use un formulario HTML, controles de selección y eventos del DOM.

C. Con GTK use entradas, controles de selección y señales.

Bash. Use Zenity para reunir los parámetros y genere las curvas mediante una herramienta gráfica externa.

PowerShell. Con Windows Forms use `TextBox`, `CheckBox` o listas y un botón que actualice las curvas.

8.11. Ejercicios para interfaz Jupyter

1. **Ejercicio:** Cuaderno interactivo tipo Jupyter en Scilab

   Cree una interfaz tipo Jupyter en Scilab: un programa que permite al usuario ingresar expresiones, las evalúa y muestra el resultado, manteniendo el historial:

```
function jupyter_scilab()
    // Nucleo del cuaderno interactivo
    historial = list()
    resultados = list()

    printf('\n')
    printf('============================================\n')
    printf('  Cuaderno interactivo de Scilab\n')
    printf('  Comandos especiales:\n')
```

```
printf('    %s: salir\n', 'salir')
printf('    %s: mostrar historial\n', 'hist')
printf('    %s: limpiar\n', 'clear')
printf('    plot(expr): graficar una expresion\n')
printf('================================================\n\n')

n = 0
while %T
    n = n + 1
    printf('In  [%d]: ', n)
    cmd = input('', 's')

    select cmd
    case 'salir' then
        printf('\nFin de sesion. %d celdas
           ejecutadas.\n', n-1)
        break

    case 'hist' then
        printf('\n--- Historial ---\n')
        for i = 1:length(historial)
            printf('In  [%d]: %s\n', i, historial(i))
            printf('Out [%d]: %s\n\n', i, resultados(i))
        end

    case 'clear' then
        n = 0
        historial = list()
        resultados = list()
        printf('Historial limpiado.\n')

    else
        // Evaluar la expresion
        ierr = execstr(cmd, 'errcatch')
        if ierr <> 0 then
            printf('Out [%d]: ERROR: %s\n\n', n,
               lasterror())
            historial($+1) = cmd
            resultados($+1) = 'ERROR: ' + lasterror()
        else
            // Intentar obtener el resultado
            try
                res = evstr(cmd)
                res_str = string(res)
                if length(res_str) > 60 then
                    res_str = res_str(1:60) + '...'
                end
                printf('Out [%d]: %s\n\n', n, res_str)
                historial($+1) = cmd
                resultados($+1) = res_str
            catch
```

```
                    printf('Out [%d]: (ejecutado)\n\n', n)
                    historial($+1) = cmd
                    resultados($+1) = '(sin valor de
                       retorno)'
                end
            end
        end
    end

    // Guardar historial en archivo
    fd = mopen('sesion.ipynb.txt', 'w')
    mfprintf(fd, '# Sesion Scilab - %s\n\n', date())
    for i = 1:length(historial)
        mfprintf(fd, '## Celda %d\n', i)
        mfprintf(fd, 'In:  %s\n', historial(i))
        mfprintf(fd, 'Out: %s\n\n', resultados(i))
    end
    mclose(fd)
    printf('Sesion guardada en sesion.ipynb.txt\n')
endfunction

jupyter_scilab()
```

a) Ejecute el cuaderno interactivo e ingrese al menos diez expresiones distintas: operaciones matemáticas, definición de variables, gráficos.

b) ¿En qué se parece y en qué se diferencia de un cuaderno Jupyter real?

c) ¿Qué ventajas tiene el modelo de cuaderno (código + resultado + texto) respecto al modelo de script tradicional?

d) Agregue al cuaderno la capacidad de ejecutar celdas de texto (comenzando con #) que se guarden como documentación pero no se evalúen.

e) Modifique la función para que guarde el historial en formato Markdown, de forma que pueda leerse directamente en un visor de Markdown.

Capítulo 9

Laboratorio de toma de datos

9.1. Medición y control

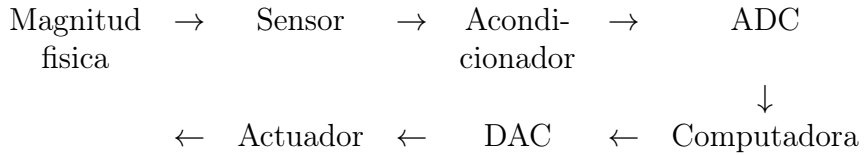
Esta unidad recorre la cadena completa que va del fenómeno físico al número guardado en un archivo, y de vuelta al proceso a través de un actuador. El orden es el de esa cadena: primero qué significa medir y contra qué se compara, después el sensor, la señal, su digitalización y sus errores, luego las formas concretas de adquirir datos con la computadora, y por último el control y las plataformas de hardware.

9.1.1. El proceso de medida

Medir es comparar una magnitud fisica desconocida con un patron de referencia y expresar el resultado como un numero acompanado de una unidad. El proceso de medida no es instantaneo ni perfecto: involucra una cadena de transformaciones fisicas y matematicas, cada una de las cuales introduce incertidumbre.

El **mesurado** es la magnitud que se desea conocer: temperatura, presion, fuerza, velocidad, concentracion quimica. El **instrumento de medida** es el conjunto de elementos que transforma el mesurado en una representacion comprensible, generalmente un numero en una pantalla o un archivo digital.

La cadena de medida tipica es:



Esta cadena es bidireccional: en un sistema de control, la computadora no solo lee datos sino que tambien envia comandos a actuadores mediante conversores digital-analogicos (DAC) o senales digitales de control.

Los elementos de la cadena

Medir es asignar un número a una magnitud física, y ese número sólo se obtiene atravesando una cadena de elementos que transforma la magnitud de interés en una representación digital. La cadena y su tratamiento general están descritos en Universidad de Alicante 2014; Wikipedia contributors 2023a.

Sensor o transductor: convierte la magnitud fisica (temperatura, presion, luz, sonido, aceleracion) en una senal electrica, generalmente una tension o una corriente. Ejemplos: termopar (temperatura → tension), microfono (presion sonora → tension), fotodiodo (luz → corriente), acelerometro (aceleracion → tension).

Acondicionador de senal: amplifica, filtra y adapta la senal del sensor para que sea adecuada para el conversor. Incluye amplificadores de instrumentacion, filtros anti-aliasing y circuitos de proteccion.

Conversor analogico-digital (ADC): transforma la senal electrica continua en una secuencia de numeros enteros que la computadora puede almacenar y procesar.

Computadora: almacena, procesa, visualiza y transmite los datos digitales.

9.1.2. Unidades, patrones y trazabilidad

Medir es comparar con un patrón, de modo que el resultado sólo tiene sentido si se sabe con qué se comparó. Desde la revisión de 2019, las siete unidades básicas del Sistema Internacional están definidas a partir de constantes físicas fijadas por convención —la velocidad de la luz, la constante de Planck, la carga del electrón, la constante de Boltzmann— y ya no a partir de artefactos materiales como el antiguo kilogramo patrón Bureau International des Poids et Mesures 2019. El fundamento físico de esas magnitudes puede repasarse en cualquier curso general de física Eisberg y Lerner 1981; Feynman 1963.

La **trazabilidad** es la cadena ininterrumpida de comparaciones que vincula el instrumento del laboratorio con el patrón nacional o internacional. Cada eslabón agrega incertidumbre. En la práctica del curso la cadena es corta y modesta: el termómetro de referencia con el que se calibra el termopar, o el multímetro con el que se verifica la tensión de la placa.

Expresión de la incertidumbre

La forma normalizada de expresar el resultado de una medición está definida en la *Guía para la expresión de la incertidumbre de medida*, conocida como GUM Joint Committee for Guides in Metrology 2008. Sus ideas centrales, que conviene aplicar en los informes de esta materia, son tres.

Incertidumbre tipo A: la que se evalúa por métodos estadísticos a partir de mediciones repetidas. Es la desviación estándar de la media, $s/\sqrt{N}$.

Incertidumbre tipo B: la que se evalúa por otros medios: especificaciones del fabricante, certificados de calibración, resolución del instrumento, experiencia previa. El

error de cuantización del ADC es un caso típico: si sólo se sabe que el valor está en un intervalo de ancho Δ, la incertidumbre estándar asociada es $\Delta/\sqrt{12}$.

Incertidumbre expandida: $U = k\,u_c$, con $k = 2$ para un nivel de confianza aproximado del 95 %. Es el valor que se reporta junto al resultado: $T = (24{,}3 \pm 0{,}4)$ °C, $k = 2$.

La distinción importa porque el error de cuantización, el ruido y la deriva del sensor no se evalúan igual, pero se combinan en la misma incertidumbre combinada u_c.

9.1.3. Magnitudes físicas como señales eléctricas: sensores

Un **sensor** o **transductor** convierte una magnitud fisica en una senal electrica medible. La senal de salida puede ser una tension, una corriente, una resistencia, una capacitancia, una frecuencia o un tren de pulsos digitales. El catalogo razonado de sensores por principio fisico esta en Fraden Fraden 2016.

Clasificación de sensores por principio físico

Resistivos: la magnitud fisica cambia la resistencia electrica del sensor.

- *Termistor*: resistencia que varia fuertemente con la temperatura. Muy sensible en rango reducido (−50 a 150 °C tipico).
- *RTD* (*Resistance Temperature Detector*): resistencia de platino de variacion lineal con la temperatura. Precision de milesimas de grado.
- *Galga extensiometrica* (*strain gauge*): la deformacion mecanica cambia la resistencia. Usada en balanzas, celdas de carga, sensores de presion.
- *LDR* (*Light Dependent Resistor*): resistencia que disminuye con la iluminacion.
- *Potenciometro*: posicion angular o lineal.

Generadores de tension: producen una tension proporcional a la magnitud.

- *Termopar*: union de dos metales distintos que genera una diferencia de potencial proporcional a la diferencia de temperatura entre sus extremos (efecto Seebeck). Los tipos mas comunes son K (cromo-alumel, -200 a 1350 °C) y J (hierro-constantan). La tension es del orden de decenas de microvolts por grado.
- *Acelerometro piezoeléctrico*: genera carga electrica proporcional a la aceleracion.
- *Celda fotovoltaica*: genera tension bajo iluminacion.

Capacitivos: la magnitud fisica cambia la capacitancia de un condensador. Usados para medir presion, nivel de liquidos, humedad y proximidad.

Inductivos: la magnitud fisica cambia la inductancia o el acoplamiento magnetico. Los *LVDT* (*Linear Variable Differential Transformer*) miden desplazamiento lineal con alta precision.

De frecuencia: la magnitud se convierte en una frecuencia de oscilacion facilmente medible con precision. El cristal de cuarzo de los relojes oscila a una frecuencia que varia con la masa depositada sobre el: fundamento de la balanza de cuarzo.

Digitales: producen directamente una senal digital.

- *Encoder rotatorio*: genera pulsos al girar. El conteo de pulsos da posicion; la frecuencia, velocidad.
- *Sensor de efecto Hall*: detecta campos magneticos; salida digital (presente/ausente).
- *DHT11/DHT22*: sensor de temperatura y humedad con salida serie digital.
- *I2C, SPI*: protocolos digitales usados por muchos sensores modernos (barometros, IMUs, magnetometros).

Parámetros característicos de un sensor

Rango: los valores minimo y maximo de la magnitud que puede medir correctamente.

Sensibilidad: cambio en la salida por unidad de cambio en la entrada. Por ejemplo, 10 mV/°C.

Resolucion: el cambio mas pequeno detectable en la entrada.

Linealidad: cuanto se aparta la respuesta real de una recta. Se expresa como porcentaje del fondo de escala.

Tiempo de respuesta: cuanto tarda el sensor en responder a un cambio escalon en la magnitud. Se especifica como constante de tiempo τ o como tiempo al 90 % del valor final.

Repetibilidad: variacion de la salida al medir el mismo valor en condiciones identicas.

Deriva (*drift*): cambio lento de la salida en el tiempo sin cambio en la magnitud, causado por envejecimiento o cambios de temperatura.

9.1.4. Señales eléctricas analógicas y digitales

Señales analógicas

Una señal analógica es una función continua del tiempo que puede tomar cualquier valor dentro de un rango. La temperatura ambiente, la presión atmosférica o la voz humana son señales analógicas: varían de forma continua y suave. Matemáticamente es $x(t)$ con t continuo y x con infinitos valores posibles en $[x_{min}, x_{max}]$.

En instrumentacion, los niveles estandar de tension para senales analogicas son:

- 0 a 5 V: microcontroladores y logica de baja tension.
- 0 a 10 V: instrumentacion industrial.

- −10 a +10 V: senales bipolares (vibracion, audio).
- −5 a +5 V: tarjetas de adquisicion de datos.

El estandar industrial de corriente 4 a 20 mA tiene la ventaja de que la corriente es inmune al ruido en cables largos y un valor de 0 mA indica rotura del cable (distinguible de 4 mA que indica cero de la magnitud).

Señales digitales

Una senal digital toma solo dos valores: alto (1) y bajo (0). Los niveles de tension que representan cada estado dependen de la familia logica:

Familia	VCC	V alto	V bajo
TTL	5 V	>2.4 V	<0.8 V
CMOS (5 V)	5 V	>3.5 V	<1.5 V
CMOS (3.3 V)	3.3 V	>2.0 V	<0.8 V
Arduino (Uno)	5 V	>3.0 V	<1.5 V
Arduino (3.3 V)	3.3 V	>2.0 V	<0.8 V

Atencion: conectar un dispositivo de 5 V a un pin de 3.3 V sin adaptador de nivel puede daniarlo permanentemente.

Protocolos de comunicación digital

Los sensores digitales modernos usan protocolos estandarizados. Cada uno resuelve un compromiso distinto entre cantidad de cables, velocidad y número de dispositivos por bus:

UART (*Universal Asynchronous Receiver-Transmitter*): comunicacion serie asincrona. Dos cables (TX, RX). Velocidad configurable (baud rate). El mas simple.

SPI (*Serial Peripheral Interface*): comunicacion serie sincrona. Cuatro cables (MOSI, MISO, SCK, CS). Alta velocidad, hasta decenas de MHz. Un maestro puede controlar varios esclavos con senales CS separadas.

I2C (*Inter-Integrated Circuit*): comunicacion serie sincrona con solo dos cables (SDA, SCL). Permite conectar varios dispositivos en el mismo bus, identificados normalmente mediante direcciones de 7 bits.

Velocidades de 100 kHz, 400 kHz o 1 MHz. Muy comun en sensores de bajo costo. El direccionamiento, el arbitraje entre maestros y los modos de velocidad están definidos en la especificación del bus NXP Semiconductors 2021.

1-Wire: utiliza una sola linea de datos, además de masa. Algunos dispositivos pueden alimentarse en modo parásito por la propia linea de datos.

Usado por el sensor de temperatura DS18B20, que además incorpora su propio conversor y entrega la temperatura ya digitalizada con resolución configurable Analog Devices 2019.

PWM (*Pulse Width Modulation*): senal digital cuyo ciclo de trabajo (fraccion del periodo en estado alto) codifica un valor analogico. Usado para controlar motores, servos y leds.

9.1.5. Conversión analógica-digital

La conversion analogico-digital (ADC) transforma la senal continua en una secuencia discreta de numeros enteros. Este proceso tiene dos dimensiones: discretizacion en el tiempo (muestreo) y discretizacion en amplitud (cuantizacion).

Muestreo

El muestreo consiste en tomar muestras de la senal analogica en instantes de tiempo regularmente espaciados. La frecuencia de muestreo f_s (en Hz o muestras por segundo) determina cuantas veces por segundo se lee el valor de la senal.

Teorema de Nyquist-Shannon Shannon 1948; S. W. Smith 1997:

para una senal estrictamente limitada en banda, la reconstruccion ideal requiere una frecuencia de muestreo superior al doble de la frecuencia maxima presente en la senal:

$$f_s > 2f_{max}$$

La frecuencia $f_{Nyquist} = f_s/2$ es la maxima frecuencia representable. Componentes de la senal por encima de este limite no se representan correctamente.

Aliasing

Si la senal contiene frecuencias superiores a $f_s/2$, estas aparecen en el espectro digital como frecuencias mas bajas, un fenomeno llamado *aliasing* S. W. Smith 1997. Una frecuencia $f > f_s/2$ aparece en la frecuencia:

$$f_{alias} = |f - k \cdot f_s|$$

donde k es el entero que minimiza el resultado.

Para evitar el aliasing se usa un **filtro anti-aliasing**: un filtro pasa-bajos analogico que elimina todas las frecuencias superiores a $f_s/2$ antes de que la senal llegue al ADC.

Ejemplo: el audio de telefonia usa $f_s = 8000$ Hz, suficiente para la voz humana (hasta 4000 Hz). El audio de CD usa $f_s = 44100$ Hz, suficiente para el oido humano (hasta 20000 Hz).

Tratamiento formal del muestreo

La exposición anterior es operativa. El desarrollo formal —transformada de Fourier de tiempo discreto, reconstrucción por interpolación, diseño de filtros digitales, decimación e interpolación— está en Oppenheim y Schafer Oppenheim y Schafer 2009. Para el nivel

de esta materia alcanza con la presentación de Smith S. W. Smith 1997, que es libre, está en línea y explica los mismos resultados sin el aparato matemático completo.

Una consecuencia práctica que conviene retener: sobremuestrear y promediar mejora la resolución efectiva. Promediar M muestras independientes reduce el ruido en $\sqrt{M}$, lo que equivale a ganar $\log_2(\sqrt{M})$ bits. Cuatro muestras promediadas dan un bit adicional; esta es la técnica que usan los ADC sigma-delta llevada al extremo.

Cuantización

La cuantizacion asigna a cada muestra un valor entero dentro de un conjunto finito de niveles. Un ADC de N bits puede representar 2^N niveles distintos. El rango de entrada $[V_{min}, V_{max}]$ se divide en 2^N intervalos iguales de ancho:

$$\Delta = \frac{V_{max} - V_{min}}{2^N}$$

Δ se llama **LSB**Kester 2005; Wikipedia contributors 2023b (*Least Significant Bit*, bit menos significativo) o resolucion del ADC.

Bits	Niveles	Resolucion	Ejemplo
8	256	0.39 % del fondo de escala	Microcontroladores
12	4096	0.024 %	Instrumentacion
16	65536	0.0015 %	Audio, medicion
24	16777216	0.000006 %	Instrumentacion de precision

Para un ADC de 12 bits con rango 0-10 V: $\Delta = 10/4096 \approx 2{,}44$ mV.

Dos tensiones que difieren menos que Δ pueden producir el mismo codigo digital. El tamaño del escalon de cuantizacion es Δ.

Arquitecturas de ADC

Los conversores analogico-digitales se clasifican segun su arquitectura interna, que determina el compromiso entre velocidad, resolucion y costo: Kester 2005

Flash: el mas rapido. Compara la entrada con $2^N - 1$ referencias simultaneamente. Requiere muchos comparadores (255 para 8 bits). Velocidades de GHz pero limitado a 8-10 bits. Usado en osciloscopios digitales y adquisicion de radiofrecuencia.

SAR (*Successive Approximation Register*): realiza una busqueda binaria: prueba sucesivamente si la entrada es mayor o menor que la mitad del rango restante. Requiere N ciclos de reloj para N bits. Equilibrio entre velocidad (hasta 5 MSPS) y resolucion (hasta 18 bits). El mas comun en microcontroladores.

Sigma-Delta ($\Sigma\Delta$): muestrea a frecuencia muy alta (oversampling) con baja resolucion y luego filtra digitalmente para obtener alta resolucion a menor velocidad. Resolucion de 16 a 32 bits pero frecuencias de pocos kHz. Usado en audio, medicion de precision y sensores lentos.

Integrador de doble rampa: integra la senal de entrada durante un tiempo fijo y luego descarga a velocidad conocida. Extremadamente lento (decenas de Hz) pero excelente rechazo al ruido de 50/60 Hz. Usado en multimetros digitales.

Acondicionamiento de señal

Entre el sensor y el ADC frecuentemente se utiliza un circuito de acondicionamiento con una o varias de las siguientes funciones: Horowitz e Hill 2015

Amplificacion: la senal del sensor suele ser muy pequena (microvolts para un termopar). El amplificador de instrumentacion la eleva al rango del ADC. Caracteristica clave: alto rechazo al modo comun (CMRR) para rechazar interferencias que aparecen en ambas entradas.

Filtro anti-aliasing: filtro analogico pasa-bajos que atenúa suficientemente las componentes que podrían producir aliasing antes del ADC. Su frecuencia de corte y su orden dependen del ancho de banda de la señal, de la frecuencia de muestreo y de la atenuacion requerida.

Proteccion: diodos y resistencias que protegen el ADC de tensiones fuera de rango, descargas electrostaticas (ESD) e inversiones de polaridad.

Filtraje de alimentacion: condensadores de desacoplo que evitan que el ruido de la fuente de alimentacion contamine la senal analogica.

Conversion de nivel: adaptar la senal del sensor al rango de entrada del ADC. Por ejemplo, una senal de -10 a $+10$ V se convierte a 0 a 3,3 V.

9.1.6. Simulación del circuito de acondicionamiento

Antes de armar un circuito de acondicionamiento conviene simularlo. El comportamiento de un filtro anti-aliasing, la respuesta de un divisor resistivo o el punto de trabajo de un amplificador se verifican en minutos y sin soldar nada.

ngspice es el simulador de circuitos libre derivado de SPICE ngspice Project 2026. Se describe el circuito en un archivo de texto —la *netlist*— y se piden los análisis: punto de operación, transitorio, barrido en frecuencia. El marco teórico, con el análisis nodal y las ecuaciones de estado que están detrás del simulador, se desarrolla en Chua, Desoer y Kuh Chua, Desoer y Kuh 1987.

Listing 9.1: filtro RC pasa-bajos anti-aliasing en ngspice

```
* Filtro RC pasa-bajos: fc = 1/(2*pi*R*C)
* R = 1.6k, C = 100n  ->  fc ~ 1 kHz
V1 in 0 AC 1 SIN(0 1 100)
R1 in out 1.6k
C1 out 0 100n

.ac dec 100 10 100k       ; respuesta en frecuencia
.tran 10u 20m             ; respuesta temporal
```

```
.control
run
plot vdb(out)              ; ganancia en dB
plot v(in) v(out)          ; senales en el tiempo
.endc
.end
```

```
# ejecutar la simulacion y volcar los resultados a un archivo
ngspice -b filtro.cir -o filtro.log

# modo interactivo
ngspice filtro.cir
```

El flujo natural con el resto de la materia es exportar los resultados de ngspice a un archivo de texto y graficarlos con Gnuplot desde Scilab, igual que se hace con los datos adquiridos. Así la respuesta simulada del filtro y la respuesta medida quedan en la misma gráfica, que es la forma de validar el diseño.

9.1.7. Errores en la medición

Toda medicion tiene errores. El tratamiento clasico de errores en el laboratorio esta en Taylor J. R. Taylor 1997; el tratamiento estadistico con ajuste de curvas, en Bevington y Robinson Bevington y D. K. Robinson 2003. Conocerlos y cuantificarlos es tan importante como el valor medido.

Errores estáticos

Son los que aparecen en condiciones de equilibrio, cuando la magnitud no cambia.

Offset (error de cero): el instrumento indica un valor distinto de cero cuando la magnitud es cero: $\hat{x} = x + b$. Se corrige restando la constante b determinada en la calibración.

Error de ganancia: la sensibilidad no es la nominal, $\hat{x} = G \cdot x$ con $G \neq 1$. Se corrige multiplicando por un factor de calibración.

No linealidad: la relación entre la magnitud y la salida no es perfectamente lineal. Se especifica como INL (*Integral Non-Linearity*) y DNL (*Differential Non-Linearity*), en LSB o como porcentaje del fondo de escala.

Histéresis: la lectura depende de la historia: el valor medido al subir la magnitud difiere del medido al bajar. Típico en sensores mecánicos y magnéticos.

Deriva: cambio lento de la salida en el tiempo sin cambio en la magnitud, por envejecimiento o por temperatura. Obliga a recalibrar cada cierto tiempo.

Error de cuantización

Es el error inherente al proceso de discretizacion en amplitud. Cuando el valor analogico x se redondea al nivel de cuantizacion mas cercano $\hat{x}$, el error es:

$$e_q = x - \hat{x}$$

Para una senal que varía aleatoriamente, el error de cuantizacion se modela como ruido uniforme en el intervalo $[-\Delta/2, +\Delta/2]$, con potencia:

$$\sigma_q^2 = \frac{\Delta^2}{12}$$

La relacion senal a ruido de cuantizacion (SQNR) para un ADC de N bits es aproximadamente:

$$\text{SQNR} \approx 6{,}02N + 1{,}76 \text{ dB}$$

Cada bit adicional agrega aproximadamente 6 dB de relacion senal a ruido.

Ruido y resolución efectiva

Las senales reales contienen ruido de diversas fuentes: ruido termico (Johnson), ruido de interferencia electromagnetica (EMI), ruido del amplificador. El ruido limita la resolucion efectiva del sistema independientemente de la resolucion nominal del ADC.

El numero efectivo de bits (ENOB) describe la resolucion real considerando el ruido:

$$\text{ENOB} = \frac{\text{SQNR} - 1{,}76}{6{,}02}$$

Un ADC de 16 bits con ruido puede tener ENOB de 14 bits.

Errores dinámicos

Aparecen cuando la magnitud cambia con rapidez.

Tiempo de respuesta: el sensor o el ADC no siguen cambios rápidos de la magnitud. Se caracteriza por la constante de tiempo τ del sensor.

Aliasing: las frecuencias superiores a $f_s/2$ aparecen como frecuencias falsas en el espectro digital. Se elimina con el filtro anti-aliasing, no con procesamiento posterior: una vez muestreada, la señal ya perdió la información.

Apertura: el ADC necesita un tiempo finito para tomar la muestra, durante el cual la señal puede cambiar.

Jitter: incertidumbre Δt en el instante exacto de muestreo. Para una señal de frecuencia f y amplitud A produce un error de tensión

$$\Delta V = 2\pi f A \cdot \Delta t$$

Para una señal de audio de 20 kHz digitalizada con 24 bits, el jitter debe mantenerse en el orden de los picosegundos.

Errores de interferencia

Ruido termico (Johnson-Nyquist): tension de ruido en toda resistencia, con valor cuadratico medio $v_n = \sqrt{4k_B T R \Delta f}$ donde k_B es la constante de Boltzmann, T la temperatura, R la resistencia y Δf el ancho de banda.

Ruido electromagnetico (EMI): interferencia de campos electricos o magneticos externos. Mitigado con blindaje y cables trenzados.

Ruido de alimentacion: perturbaciones en la fuente de alimentacion que contaminan la senal.

Interferencia de 50/60 Hz: la frecuencia de la red electrica se acopla inductiva o capacitivamente a los cables del sensor. Se rechaza con filtros notch o con ADC de integracion de doble rampa.

Bucles de masa: diferencias de potencial entre los planos de masa de distintos equipos crean corrientes que generan tension de ruido. Se resuelven con aislamiento galvanico (optoacopladores, transformadores).

De todos los errores tratados, éstos son los únicos que no se corrigen con software: no hay calibración ni promediado que arregle un blindaje mal conectado o una masa compartida con un motor. Se resuelven en el cableado y en la disposición física del montaje, y el tratamiento sistemático de esas técnicas —apantallamiento, trenzado, punto único de masa, separación de retornos analógicos y digitales— es materia de la compatibilidad electromagnética Ott 2009.

Errores sistemáticos y aleatorios

Errores sistematicos: reproducibles, con signo y magnitud definidos. Causados por imperfecciones del instrumento (offset, ganancia, no linealidad) o errores en el metodo de medicion. Se reducen mediante calibracion.

Errores aleatorios: fluctuaciones impredecibles en cada medicion, causadas por ruido. Se reducen promediando multiples mediciones: el error de la media cae como $1/\sqrt{N}$ para N mediciones independientes.

Errores de cuantizacion: inherentes a la conversion digital. Se reducen aumentando la resolucion del ADC o usando tecnicas de promediado (oversampling).

Errores de aliasing: producidos por frecuencias superiores a $f_s/2$. Se eliminan con filtro anti-aliasing.

Propagación de errores

Si una magnitud Z se calcula a partir de mediciones con incertidumbres, la incertidumbre de Z se propaga:

Para $Z = f(x_1, x_2, \ldots, x_n)$:

$$\sigma_Z = \sqrt{\sum_{i=1}^{n} \left(\frac{\partial f}{\partial x_i} \right)^2 \sigma_{x_i}^2}$$

Por ejemplo, si $Z = x + y$ J. R. Taylor 1997, entonces $\sigma_Z = \sqrt{\sigma_x^2 + \sigma_y^2}$. Si $Z = x \cdot y$, entonces $\sigma_Z / Z = \sqrt{(\sigma_x/x)^2 + (\sigma_y/y)^2}$.

Incertidumbre combinada

La incertidumbre total de una medicion se calcula combinando todas las fuentes de error. Para errores independientes:

$$u_{total} = \sqrt{u_{offset}^2 + u_{ganancia}^2 + u_{linealidad}^2 + u_{cuantizacion}^2 + u_{ruido}^2}$$

La incertidumbre se expresa con un factor de cobertura k: $U = k \cdot u_{total}$. Para $k = 2$, U cubre el 95 % de los resultados posibles (distribucion normal).

9.1.8. Calibración

Todo sistema de adquisicion debe calibrarse antes de usar. La calibracion consiste en medir senales de referencia conocidas y ajustar los parametros del modelo.

Conviene ser preciso con los términos, porque el uso corriente los confunde: calibrar es establecer la relación entre lo que el instrumento indica y el valor que corresponde a un patrón, con su incertidumbre; ajustar es modificar el instrumento para que indique lo correcto. La calibración produce información, el ajuste modifica el aparato, y el vocabulario internacional de metrología fija esas definiciones junto con las de trazabilidad, exactitud y veracidad Joint Committee for Guides in Metrology 2012.

La referencia contra la que se calibra debe estar a su vez calibrada, y así hacia arriba hasta el patrón nacional: eso es la cadena de trazabilidad que vimos al principio. Para que un certificado de calibración valga fuera del laboratorio que lo emite, ese laboratorio debe demostrar competencia técnica, trazabilidad documentada e intervalos de recalibración definidos International Organization for Standardization 2017. En el laboratorio de la materia no llegamos a eso, pero conviene saber que la diferencia entre «lo calibré con un multímetro que tenía a mano» y «está calibrado» es exactamente ésa.

Listing 9.2: calibracion lineal de dos puntos

```
// calibracion de dos puntos:
// se miden dos referencias conocidas y se ajusta una recta
```

```
// valores de referencia (valores verdaderos)
V_ref = [0.0,  5.0]

// lecturas del sistema (valores medidos)
V_lec = [102, 4198]   // en cuentas del ADC (0-4095 para 12 bits)

// ajuste lineal: V_real = a * V_lectura + b
// sistema de ecuaciones: V_ref = [V_lec; 1] * [a; b]
A = [V_lec(1), 1; V_lec(2), 1]
b_vec = V_ref'
coefs = A \ b_vec
a = coefs(1); b = coefs(2)

printf('Calibracion: V_real = %.6f * lectura + %.6f\n', a, b)
printf('Ganancia: %.4f V/cuenta\n', a)
printf('Offset:   %.4f V\n', b)

// aplicar calibracion a nuevas lecturas
function V_real = calibrar(lectura, a, b)
    V_real = a * lectura + b
endfunction

// ejemplo de uso
lectura_nueva = 2048
printf('Lectura %d -> %.4f V\n', lectura_nueva, ..
       calibrar(lectura_nueva, a, b))
```

****

9.1.9. Sistemas multicanales

Muchas aplicaciones requieren medir simultaneamente varias magnitudes: temperatura en multiples puntos, fuerzas en varias direcciones, senales de distintos sensores.

Multiplexor analógico

La forma mas economica de multicanal usa un solo ADC y un multiplexor analogico que conecta secuencialmente cada sensor a la entrada del ADC.

$$\begin{array}{l}\text{Sensor 1} \rightarrow \\ \left.\begin{array}{l}\text{Sensor 2} \rightarrow \\ \text{Mux} \rightarrow \text{ADC} \rightarrow \text{CPU} \\ \vdots \\ \text{Sensor N} \rightarrow\end{array}\right.\end{array}$$

Las limitaciones son:

- Las muestras de distintos canales no son simultaneas.

Si f_s es la frecuencia total de conversion del ADC, las conversiones consecutivas están separadas aproximadamente por $1/f_s$, mientras que cada canal se muestrea con una frecuencia aproximada f_s/N.

- Los transitorios de conmutacion del multiplexor pueden contaminar la muestra siguiente si el tiempo de establecimiento no es suficiente.
- Las impedancias de los sensores pueden interactuar.

ADC simultáneos

Para aplicaciones donde la sincronizacion es critica (medicion de potencia electrica, analisis modal de vibraciones, sistemas de control rapido) se usa un ADC por canal, todos disparados por la misma senal de reloj.

Sistemas de adquisición de datos (DAQ)

Las tarjetas de adquisicion de datos integran todos los elementos en un solo modulo: multiplexor, amplificador de instrumentacion, filtros, ADC y DAC, con interfaz USB o PCIe hacia la computadora.

Las familias mas conocidas son National Instruments (NI USB-6001, NI USB-6341), Measurement Computing (MCC USB-1208) y OpenDAQ. Tipicamente ofrecen:

- 8 a 64 canales analogicos de entrada.
- Resolucion de 12 a 24 bits.
- Velocidad de 10 kSPS a 2 MSPS por canal.
- 2 a 4 canales de salida analogica (DAC).
- Entradas y salidas digitales.
- Contadores y temporizadores.

9.1.10. Adquisición con la computadora

Las tres formas de adquirir datos que se usan en el curso —la placa de sonido, el puerto serie sobre USB y una tarjeta dedicada— cubren rangos distintos de frecuencia, resolución y costo. Las dos primeras no requieren comprar nada.

La placa de sonido como sistema de adquisición

La tarjeta de sonido de una computadora es un sistema de Ramey y Fox 2025; Scilab Enterprises 2026c adquisicion de datos de proposito general para senales en el rango de audio (20 Hz a 20 kHz). Tiene un ADC de 16 o 24 bits con frecuencias de muestreo tipicas de 8000, 22050, 44100 o 96000 Hz.

La entrada de audio puede utilizarse para adquirir señales alternas dentro de su banda de frecuencia. No es una entrada de tension de proposito general: normalmente está acoplada en alterna y puede incorporar una polarizacion destinada a microfonos. Por ello no permite medir correctamente tensiones continuas ni señales muy lentas. Toda señal electrica externa debe acondicionarse para respetar el rango y la impedancia de entrada de la placa utilizada.

Se puede adquirir cualquier senal dentro del ancho de banda de audio. El microfono tiene un rango tipico de ±1 V (o similar segun la configuracion) y una impedancia de entrada de varios kilohms.

Grabación desde la terminal en Linux

```
# listar los dispositivos de audio
arecord -l

# grabar 5 segundos en formato WAV (stereo, 44100 Hz, 16 bits)
arecord -d 5 -f cd -t wav /tmp/senal.wav

# grabar en mono, 8000 Hz (calidad telefonica)
arecord -d 5 -c 1 -r 8000 -f S16_LE -t wav /tmp/mono.wav

# ver informacion del archivo grabado
soxi /tmp/senal.wav

# reproducir
aplay /tmp/senal.wav
```

Lectura y análisis en Scilab Scilab puede leer archivos WAV y procesarlos numericamente:

Listing 9.3: leer y analizar un WAV en Scilab

```
// leer el archivo WAV
[x, fs] = wavread('/tmp/senal.wav')
// x: matriz de muestras (filas=canales, columnas=muestras)
// fs: frecuencia de muestreo [Hz]

printf('Frecuencia de muestreo: %d Hz\n', fs)
printf('Numero de canales: %d\n', size(x,1))
printf('Numero de muestras: %d\n', size(x,2))
printf('Duracion: %.3f s\n', size(x,2)/fs)

// tomar el canal 1 si es stereo
if size(x,1) > 1 then
    senal = x(1,:)
else
    senal = x
end

// vector de tiempo
t = (0:length(senal)-1) / fs
```

```
// graficar la senal en el tiempo
clf()
subplot(2,1,1)
plot(t, senal)
xlabel('Tiempo [s]')
ylabel('Amplitud')
title('Senal temporal')
xgrid()

// calcular y graficar el espectro de frecuencias (FFT)
N   = length(senal)
Y   = fft(senal)
f   = (0:N/2-1) * fs / N
amp = abs(Y(1:N/2)) * 2/N

subplot(2,1,2)
plot(f, amp)
xlabel('Frecuencia [Hz]')
ylabel('Amplitud')
title('Espectro de frecuencias')
xgrid()
```

Captura en tiempo real desde Scilab Scilab puede llamar a `arecord` para capturar datos y luego procesarlos:

Listing 9.4: captura y analisis en tiempo real

```
// parametros de captura
fs       = 44100    // Hz
duracion = 2        // segundos
archivo  = '/tmp/captura.wav'

printf('Capturando %d segundos...\n', duracion)
cmd = sprintf('arecord -d %d -c 1 -r %d -f S16_LE -t wav %s', ..
              duracion, fs, archivo)
unix(cmd)
printf('Captura completada.\n')

// leer y analizar
[x, fs_leido] = wavread(archivo)
senal = x(1,:)
t     = (0:length(senal)-1) / fs_leido

// frecuencia dominante
N   = length(senal)
Y   = fft(senal)
amp = abs(Y(1:N/2)) * 2/N
f   = (0:N/2-1) * fs_leido / N

[amp_max, idx_max] = max(amp)
```

```
printf('Frecuencia dominante: %.1f Hz\n', f(idx_max))
printf('Amplitud maxima: %.4f\n', amp_max)
```

Aplicaciones con el micrófono como sensor El microfono de PC puede usarse para medir fenomenos fisicos que generen variaciones de presion sonora o que puedan conectarse electricamente a la entrada de audio:

- **Velocidad del sonido**: medir el retardo entre un impulso generado y su eco en una pared conocida.
- **Frecuencia de resonancia**: golpear un objeto y analizar el espectro de las vibraciones.
- **Nivel de sonido**: medir la amplitud RMS de la senal como aproximacion al nivel de presion sonora.
- **Sensores con salida de tension alternas**: sensores cuya salida se encuentre dentro de la banda de audio pueden adquirirse mediante la entrada de sonido si se utiliza el acondicionamiento adecuado.

Puerto serie y USB

El puerto serie (UART) El puerto serie o UART (*Universal Asynchronous Receiver-Transmitter*) es el interfaz de comunicacion digital mas antiguo y universal. Transmite datos bit a bit en forma asincrona: cada byte va precedido de un bit de inicio y seguido de uno o dos bits de parada. Desde un programa, abrir el puerto es sólo el primer paso: hay que fijar velocidad, formato y modo de lectura sobre la estructura `termios`, y elegir entre el modo canónico —que entrega líneas completas— y el modo crudo, que entrega los bytes a medida que llegan Sweet 1999.

Los parametros de configuracion son:

Velocidad (baud rate): bits por segundo. Valores tipicos: 9600, 19200, 38400, 57600, 115200 baud.

Bits de datos: generalmente 8.

Paridad: ninguna (N), par (E) o impar (O).

Bits de parada: 1 o 2.

Se especifica como `9600 8N1`: 9600 baud, 8 bits, sin paridad, 1 bit de parada.

USB como puerto serie virtual Los dispositivos modernos (Arduino, microcontroladores, instrumentos de medida) usan USB con un chip convertidor USB-a-serie (CH340, CP2102, FTDI).

En Linux pueden aparecer como `/dev/ttyUSB0`, `/dev/ttyUSB1`, etc., o como `/dev/ttyACM0`, `/dev/ttyACM1`, etc., según el dispositivo.

Los convertidores mas antiguos con conector DB9 aparecen como `/dev/ttyS0`.

Conviene notar que ahi no hay ningun puerto serie: el sistema enumera el dispositivo USB, lee sus descriptores y, al reconocer la clase de comunicaciones, carga un controlador que expone un puerto serie virtual USB Implementers Forum 2000.

En un conversor USB–UART, parámetros como velocidad, bits de datos y paridad siguen siendo necesarios porque configuran la comunicacion UART del lado del microcontrolador. En dispositivos USB nativos el mecanismo puede ser diferente.

```
# listar puertos serie disponibles
ls -la /dev/ttyUSB* /dev/ttyACM* /dev/ttyS* 2>/dev/null

# identificar el dispositivo al conectarlo
dmesg | tail -10

# leer datos del puerto serie en la terminal
# (parametros: 9600 baud, 8N1)
stty -F /dev/ttyUSB0 9600 cs8 -cstopb -parenb
cat /dev/ttyUSB0

# alternativa mas comoda: minicom
sudo apt install minicom
minicom -D /dev/ttyUSB0 -b 9600

# o screen
screen /dev/ttyUSB0 9600
```

Permisos de acceso al puerto En Debian/Ubuntu, el usuario debe pertenecer al grupo `dialout` para acceder al puerto serie sin sudo:

```
# agregar el usuario al grupo dialout
sudo usermod -a -G dialout $USER

# cerrar sesion y volver a entrar para que tenga efecto
# verificar
groups | grep dialout
```

Protocolo de comunicación El microcontrolador o instrumento envia datos en formato de texto, generalmente una linea por medicion:

```
# ejemplo de datos que envia un Arduino cada 100 ms:
# tiempo_ms,temperatura,presion
1000,23.45,1013.2
1100,23.46,1013.1
1200,23.44,1013.3
...
```

Lectura desde Scilab vía puerto serie Scilab puede leer el puerto serie usando `unix_g` o pipes, o mediante el modulo `serialport` si esta disponible. Desde Python —opción natural en la Raspberry Pi— el camino habitual es la biblioteca pySerial, que abre el puerto, lo configura y devuelve las líneas ya separadas Liechti 2026:

Listing 9.5: leer datos del puerto serie via bash

```
// metodo 1: leer N lineas con timeout usando bash
function datos = leer_serie(puerto, baud, n_lineas, timeout_s)
    // configurar el puerto
    unix(sprintf('stty -F %s %d cs8 -cstopb -parenb raw', ..
                 puerto, baud))

    // leer n lineas con timeout
    cmd = sprintf('timeout %d head -n %d %s', ..
                  timeout_s, n_lineas, puerto)
    [lineas, ret] = unix_g(cmd)

    // parsear cada linea como vector de numeros
    datos = zeros(n_lineas, 3)
    for i = 1:size(lineas, '*')
        partes = strsplit(lineas(i), ',')
        if size(partes,'*') >= 3 then
            for j = 1:3
                datos(i,j) = strtod(partes(j))
            end
        end
    end
endfunction

// leer 50 muestras del puerto USB
D = leer_serie('/dev/ttyUSB0', 9600, 50, 10)
t    = D(:,1) / 1000   // ms a segundos
temp = D(:,2)
pres = D(:,3)

printf('Temperatura media: %.2f C\n', mean(temp))
printf('Temperatura min:   %.2f C\n', min(temp))
printf('Temperatura max:   %.2f C\n', max(temp))
```

Listing 9.6: adquisicion continua con graficado en tiempo real

```
// adquisicion continua: leer una linea a la vez
// y actualizar la grafica

PUERTO = '/dev/ttyUSB0'
BAUD   = 9600
N_MAX  = 200   // maximo de puntos en pantalla

unix(sprintf('stty -F %s %d cs8 -cstopb -parenb raw', PUERTO,
   BAUD))
```

```
t_vec = zeros(1, N_MAX)
y_vec = zeros(1, N_MAX)
n     = 0

clf()
h_curva = plot(0, 0, 'b-', 'LineWidth', 2)
xlabel('Tiempo [s]'); ylabel('Valor')
title('Adquisicion en tiempo real')
xgrid()

printf('Adquiriendo... Ctrl+C para detener\n')

t0 = getdate()

while %T
    // leer una linea del puerto
    [linea, ret] = unix_g(sprintf('head -n 1 %s', PUERTO))
    if ret ~= 0 | isempty(linea) then continue, end

    // parsear
    partes = strsplit(linea(1), ',')
    if size(partes,'*') < 2 then continue, end

    valor = strtod(partes(2))
    if isnan(valor) then continue, end

    t_actual = getdate()
    dt = (t_actual(9) - t0(9)) + 60*(t_actual(8)-t0(8))

    // desplazar el buffer
    n = n + 1
    if n > N_MAX then
        t_vec = [t_vec(2:$), dt]
        y_vec = [y_vec(2:$), valor]
    else
        t_vec(n) = dt
        y_vec(n) = valor
    end

    // actualizar la grafica
    set(h_curva, 'xdata', t_vec(1:min(n,N_MAX)), ..
                 'ydata', y_vec(1:min(n,N_MAX)))
    drawnow()
end
```

Guardar datos a archivo y graficar con Gnuplot El patron mas robusto para adquisicion larga es guardar los datos a archivo mientras se adquieren y graficar al terminar:

Listing 9.7: adquisicion a archivo y graficado posterior

```
// adquirir durante N segundos y guardar
```

```
PUERTO   = '/dev/ttyUSB0'
BAUD     = 9600
DURACION = 60   // segundos
ARCHIVO  = '/tmp/datos_serie.csv'

unix(sprintf('stty -F %s %d cs8 -cstopb -parenb raw', PUERTO,
   BAUD))

// redirigir el puerto a archivo durante N segundos
cmd = sprintf('timeout %d cat %s > %s', DURACION, PUERTO,
   ARCHIVO)
printf('Adquiriendo %d segundos...\n', DURACION)
unix(cmd)
printf('Adquisicion completada. Procesando...\n')

// leer el archivo
lineas = mgetl(ARCHIVO)
n      = size(lineas, '*')
datos  = zeros(n, 3)
validos = 0

for i = 1:n
    partes = strsplit(lineas(i), ',')
    if size(partes,'*') >= 2 then
        validos = validos + 1
        for j = 1:min(size(partes,'*'), 3)
            datos(validos, j) = strtod(partes(j))
        end
    end
end
datos = datos(1:validos, :)

t    = datos(:,1)' / 1000
temp = datos(:,2)'

printf('Muestras validas: %d\n', validos)
printf('Frecuencia media: %.1f Hz\n', validos/DURACION)

// graficar con Gnuplot
fd = mopen('/tmp/datos_gnuplot.txt', 'w')
for i = 1:validos
    mfprintf(fd, '%.3f  %.4f\n', t(i), temp(i))
end
mclose(fd)

NL = ascii(10); Q = ascii(34)
gp = 'set terminal png size 900,400' + NL
gp = gp + 'set output ' + Q + '/tmp/serie.png' + Q + NL
gp = gp + 'set grid' + NL
gp = gp + 'set xlabel ' + Q + 'Tiempo [s]' + Q + NL
gp = gp + 'set ylabel ' + Q + 'Temperatura [C]' + Q + NL
```

```
gp = gp + 'set title  ' + Q + 'Adquisicion via puerto serie' + Q
   + NL
DAT = Q + '/tmp/datos_gnuplot.txt' + Q
gp = gp + 'plot ' + DAT + ' with lines lw 1.5 lc ' + ..
          Q + 'blue' + Q + ' notitle' + NL

fd = mopen('/tmp/serie.gnu', 'w')
mfprintf(fd, '%s', gp)
mclose(fd)
unix('gnuplot /tmp/serie.gnu')
printf('Grafica guardada en /tmp/serie.png\n')
```

Comparación de métodos de adquisición

Aspecto	Microfono	Puerto serie	Tarje
Costo	Incluido en PC	∼$5 (Arduino)	$100-$
Frecuencia max	48000 Hz	Segun ADC externo	Hasta
Resolucion	16-24 bits	Segun ADC externo	12-24
Canales	1-2 (L/R)	Depende del sistema conectado	8-64 t
Aislamiento	No	Depende del hw	Dispo
Sincronizacion	Precisa	Depende firmware	Precis
Rango de tension	Depende de la entrada de audio	Depende del ADC y acondicionamiento	±10 V
Uso tipico	Audio, acustica	Microcontroladores	Labor

9.1.11. Registro, transmisión y visualización

El programa de la unidad menciona cuatro pasos después de la digitalización: transmisión, visualización, registro y actuación. Los dos primeros ya aparecieron en los ejemplos; conviene ordenarlos.

Transmisión: el enlace entre el dispositivo y la computadora. Puerto serie sobre USB para distancias cortas; WiFi con el ESP32 cuando no se puede cablear; RS-485 o el lazo de 4 a 20 mA en ambientes industriales o con cables largos, porque la corriente es inmune al ruido inducido y una lectura de 0 mA distingue el cable cortado del cero de la magnitud.

Registro: escribir a archivo mientras se adquiere, no al final. Un archivo de texto con una línea por muestra, con marca de tiempo y separador claro, sobrevive a la caída del programa y se lee con cualquier herramienta. El formato CSV con encabezado es suficiente y se procesa igual desde Scilab, Gnuplot o una planilla Ramey y Fox 2025.

Visualización: durante la adquisición, una gráfica simple que permita advertir que algo anda mal —saturación, cable suelto, ruido de 50 Hz—. Después de la adquisición, la gráfica cuidada para el informe, con ejes rotulados, unidades y barras de incertidumbre.

Actuación: cerrar el lazo enviando comandos al actuador, que es el tema de la sección de control.

Conviene guardar junto a los datos un archivo de metadatos con la fecha, el instrumento, la frecuencia de muestreo, los parámetros de calibración vigentes y las condiciones del ensayo. Sin eso, una serie de números deja de ser una medición al cabo de unos meses.

9.1.12. Control

Un sistema de control utiliza la medicion del estado actual del proceso para calcular y aplicar una accion que lo lleva al estado deseado. Es el paso que cierra la unidad: hasta acá el sistema sólo observaba, y ahora actúa sobre lo que observa. La realimentación tiene consecuencias que no son obvias —puede estabilizar un sistema inestable, y también desestabilizar uno estable— y ése es el asunto central de la teoría de control Åström y Murray 2021.

Control de lazo abierto

El actuador aplica una accion fija sin realimentacion. Simple pero sensible a perturbaciones y cambios en el proceso. Ejemplo: encender un horno durante 30 minutos sin medir la temperatura.

Control de lazo cerrado (realimentado)

El sistema mide la salida del proceso, la compara con el valor deseado (setpoint) y calcula el error. El controlador usa el error para calcular la accion correctiva:

$$e(t) = r(t) - y(t)$$

donde $r(t)$ es el setpoint e $y(t)$ la salida medida.

El controlador **PID** (*Proportional-Integral-Derivative*) es el mas utilizado en la industria: Su analisis formal, los criterios de estabilidad y los metodos de sintonia estan en Ogata Ogata 2010.

$$u(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de}{dt}$$

Termino proporcional K_p: correccion proporcional al error actual. Mayor K_p da respuesta mas rapida pero puede causar oscilaciones.

Termino integral K_i: elimina el error en estado estacionario acumulando el error en el tiempo. Puede causar sobrealcance (*windup*).

Termino derivativo K_d: anticipa cambios futuros mirando la tasa de cambio del error. Reduce el sobrealcance pero amplifica el ruido.

Implementación digital del PID

En un microcontrolador o computadora, el PID se implementa en tiempo discreto con periodo de muestreo T_s. El paso del integrador al acumulador y de la derivada a la diferencia dividida no es inocente: la derivada discreta amplifica el ruido de medición —que es justamente lo que estuvimos midiendo en la sección de errores—, y el término integral sigue acumulando mientras el actuador está saturado, de modo que cuando el error cambia de signo el controlador tarda en reaccionar. Por eso el código que sigue satura la salida y descuenta la integral. Las variantes prácticas para los dos problemas —filtrado del término derivativo, derivada sobre la medición en vez del error, anti-windup por retrocálculo— están tratadas en Åström y Hägglund 2006.

Listing 9.8: PID digital en Scilab

```
function [u, integral, error_prev] = pid_paso(r, y, ..
                                     Kp, Ki, Kd, Ts, ..
                                     integral, error_prev, ..
                                     u_min, u_max)
    e          = r - y                          // error actual
    integral   = integral + e * Ts              // suma del error
    derivada   = (e - error_prev) / Ts          // tasa de cambio
    error_prev = e

    u = Kp*e + Ki*integral + Kd*derivada        // accion PID

    // saturar la salida
    u = max(u_min, min(u_max, u))

    // anti-windup: no acumular integral si esta saturado
    if u ~= Kp*e + Ki*integral + Kd*derivada then
        integral = integral - e * Ts
    end
endfunction

// simulacion de control PID de temperatura
Kp = 2.0; Ki = 0.5; Kd = 0.1; Ts = 0.1
r  = 50   // setpoint: 50 grados
y  = 20   // temperatura inicial

integral   = 0
error_prev = 0
T_sim = 30

t_vec = []; y_vec = []; u_vec = []

for t = 0:Ts:T_sim
    [u, integral, error_prev] = pid_paso(r, y, ..
                                Kp, Ki, Kd, Ts, ..
                                integral, error_prev, 0, 100)

    // modelo simple del proceso: primer orden
    tau_proceso = 5.0
```

```
    y = y + Ts * (u/tau_proceso - (y-20)/tau_proceso)

    t_vec = [t_vec, t]
    y_vec = [y_vec, y]
    u_vec = [u_vec, u]
end
```

Sintonía del controlador

Elegir K_p, K_i y K_d a ojo funciona en simulación y rara vez en un proceso real. Los métodos de sintonía dan un punto de partida razonable Ogata 2010.

Ziegler-Nichols en lazo cerrado: Ziegler y Nichols 1942 se anulan los términos integral y derivativo y se aumenta K_p hasta que el sistema oscila de forma sostenida. Con la ganancia crítica K_u y el período de oscilación T_u se obtiene, para un PID clásico, $K_p = 0{,}6\,K_u$, $K_i = 2K_p/T_u$ y $K_d = K_p T_u/8$.

Ziegler-Nichols en lazo abierto: se aplica un escalón en la entrada y se mide la respuesta. De la curva se extraen el retardo L y la constante de tiempo T, y las tablas dan los parámetros.

Ajuste manual: subir K_p hasta obtener una respuesta rápida con algo de sobrepaso; agregar K_i hasta eliminar el error en régimen permanente; agregar K_d sólo si el ruido lo permite, porque el término derivativo lo amplifica.

Conviene registrar la respuesta al escalón antes y después de sintonizar, con los tiempos de subida y establecimiento y el sobrepaso medidos sobre la gráfica. Esa comparación es el resultado que se informa, no los valores de las tres constantes.

Representación en espacio de estados

El PID trata al proceso como una caja negra con una entrada y una salida. Cuando el sistema tiene varias entradas y salidas acopladas —o cuando se dispone de un modelo físico, como los de la unidad de simulación— conviene la representación en espacio de estados:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \qquad \mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

donde $\mathbf{x}$ es el vector de estado, $\mathbf{u}$ el de entradas y $\mathbf{y}$ el de salidas. Es el mismo formalismo con el que se escriben las ecuaciones de un circuito Chua, Desoer y Kuh 1987 y el que usan las herramientas de control de Scilab Scilab Enterprises 2026c. El tratamiento completo, con controlabilidad, observabilidad y diseño por asignación de polos, está en Ogata Ogata 2010.

Actuadores

Los actuadores convierten la senal de control en una accion fisica sobre el proceso. Son el espejo del sensor: donde el sensor transforma una magnitud física en señal eléctrica, el actuador transforma la señal eléctrica en magnitud física Hughes y Drury 2019.

Relevador (relay): interruptor electromecanico. Solo control on/off. Bajo costo, adecuado para cargas de alta potencia.

Transistor y MOSFET: control de corriente continua. El PWM del microcontrolador controla la tension media aplicada a la carga (motor DC, LED, resistencia calefactora). La carga suele ser inductiva, y entonces hace falta un diodo de retorno y cuidado con los transitorios de conmutacion, que son la fuente de interferencia mas comun del montaje Mohan, Undeland y Robbins 2003.

Servo motor: motor con realimentacion de posicion. Recibe un pulso PWM cuyo ancho codifica el angulo deseado. Muy usado en robotica y modelos RC.

Motor paso a paso (*stepper*): motor que gira un angulo fijo por cada pulso. Control de posicion sin sensor de realimentacion. Usado en impresoras 3D y CNC.

Valvulas: electrovalvulas para control de fluido. Valvulas proporcionales controladas por corriente.

DAC: conversor digital-analogico que produce una tension proporcional al valor digital. Controla actuadores con entrada de tension (variadores de frecuencia, actuadores proporcionales).

9.2. Hardware de adquisición y control

En muchos sistemas modernos el sensor no entrega directamente una tensión analógica. El elemento sensible, el acondicionamiento de señal, la conversión analógico–digital y parte del procesamiento se encuentran integrados en un mismo circuito. El microcontrolador recibe entonces directamente un valor digital mediante un protocolo de comunicación.

La cadena de medida puede representarse como

magnitud física $\longrightarrow$ sensor $\longrightarrow$ acondicionamiento $\longrightarrow$ ADC $\longrightarrow$ interfaz digital $\longrightarrow$ microcontrolador.

Esto no elimina los problemas de calibración, resolución, ruido o tiempo de respuesta: simplemente parte de la instrumentación queda incorporada dentro del propio sensor.

Categorías de lo que veremos y de otros dispositivos que no veremos pero sirven para éstas tareas:

```
PLATAFORMAS DE ADQUISICIÓN Y CONTROL
|
|-- Microcontroladores
```

```
|     |-- Arduino
|     `-- ESP32
|
|-- Computadoras embebidas
|     `-- Raspberry Pi
|
|-- Módulos de adquisición industriales
|     `-- ADAM
|
`-- Instrumentos digitales completos
      `-- TEMPer2
```

Y aparte, porque no son plataformas:

```
SENSORES
|-- LM35
|-- LDR
|-- BH1750
|-- VEML7700
|-- SHT31
|-- BME280
`-- DS18B20
```

Y otro nivel distinto:

```
PROTOCOLOS / INTERFACES
|-- UART / serie
|-- USB
|-- HID
|-- I2C
|-- SPI
|-- 1-Wire
|-- RS-485
`-- WiFi / HTTP
```

Y finalmente:

```
SOFTWARE
|-- Scilab
|-- Python
|-- Tcl
|-- Gnuplot
`-- temper.py
```

9.2.1. LM35: sensor analógico de temperatura

El LM35 es un sensor integrado de temperatura cuya salida es una tensión aproximadamente proporcional a la temperatura en grados Celsius.

Su sensibilidad nominal es

$$10 \text{ mV/°C},$$

por lo que

$$V_{\text{out}} = 0{,}010\, T$$

con V_{out} expresada en volts y T en grados Celsius. Por tanto,

$$T = 100\, V_{\text{out}}.$$

Por ejemplo, una temperatura de 25 °C produce aproximadamente

$$V_{\text{out}} = 0{,}25 \text{ V}.$$

En el montaje utilizado en el laboratorio el sensor se alimenta desde el Arduino y su salida se conecta a una entrada analógica. Un condensador de desacoplo de aproximadamente 100 nF entre alimentación y masa, ubicado cerca del sensor, ayuda a reducir perturbaciones de alta frecuencia.

La temperatura calculada depende también de la referencia utilizada por el ADC. Si se supone una referencia de 5 V y un ADC de 10 bits,

$$V_{\text{out}} \simeq \frac{n}{1024}\, 5 \text{ V},$$

donde n es el código entregado por el ADC. Para mediciones de mayor exactitud conviene conocer experimentalmente la tensión real de referencia.

9.2.2. LDR y divisor resistivo

Una LDR (*Light Dependent Resistor*) es una resistencia cuyo valor disminuye cuando aumenta la iluminación.

Como el ADC mide tensión y no resistencia, la LDR puede utilizarse formando un divisor resistivo. Por ejemplo:

$$V_o = V_{CC} \frac{R}{R + R_{\text{LDR}}},$$

donde R es una resistencia fija.

Al aumentar la iluminación disminuye R_{LDR} y, con esta conexión, aumenta V_o.

La relación entre resistencia e iluminancia no es lineal y depende del modelo de LDR. Por eso una lectura expresada como porcentaje del rango del ADC representa un nivel relativo de iluminación y no directamente una medida calibrada en lux.

9.2.3. BH1750 y VEML7700: iluminancia – Sensores digitales integrados

El BH1750 y el VEML7700 son sensores digitales de luz ambiente. Ambos incorporan fotodetector, electrónica de acondicionamiento, conversión A/D y una interfaz I2C.

El microcontrolador no mide una tensión producida por el sensor sino que interroga registros digitales y obtiene un número relacionado con la iluminancia.

El bus I2C utiliza fundamentalmente dos líneas:

SDA: datos;

SCL: reloj.

Además son necesarias alimentación y masa. Varios dispositivos pueden compartir el mismo bus siempre que puedan distinguirse mediante sus direcciones.

El BH1750 permite obtener directamente medidas de iluminancia y posee distintos modos de medida relacionados con el tiempo de integración.

El VEML7700 permite además modificar parámetros tales como la ganancia y el tiempo de integración. Una ganancia elevada o un tiempo de integración largo permiten detectar niveles pequeños de iluminación, mientras que para iluminación intensa deben reducirse para evitar saturación.

En una experiencia deben registrarse no solamente las lecturas sino también las condiciones de configuración del sensor.

La iluminancia se expresa en lux y es una magnitud fotométrica, es decir, ponderada de acuerdo con la sensibilidad visual humana. No debe confundirse directamente con la irradiancia, expresada en W/m^2. Por esta razón estos sensores pueden utilizarse para estudiar variaciones de iluminación pero no constituyen, sin una caracterización adicional, un piranómetro.

9.2.4. DHT11 y DHT22

Los DHT11 y DHT22 son sensores digitales integrados de temperatura y humedad relativa.

El DHT22 posee un rango y una resolución mejores que el DHT11 y es el utilizado en uno de los prácticos de esta unidad.

El dispositivo realiza internamente la medición y la conversión digital. El microcontrolador obtiene los valores mediante un protocolo serie propio del sensor, por lo que normalmente se utiliza una biblioteca que implementa la temporización de la comunicación.

Estos sensores son relativamente lentos. Por ello no tiene sentido interrogarlos a frecuencias elevadas: la frecuencia útil de adquisición está limitada por el propio sensor y no por la velocidad de cálculo del ESP32.

9.2.5. SHT31, BMP280 y BME280: sensores ambientales digitales

El SHT31, el BMP280 y el BME280 son ejemplos de sensores digitales integrados utilizados para adquisición de variables ambientales.

El SHT31 mide temperatura y humedad relativa.

El BMP280 mide temperatura y presión atmosférica.

El BME280 agrega a esas magnitudes la medición de humedad relativa, por lo que permite obtener temperatura, presión y humedad con un único dispositivo.

Estos sensores incorporan internamente el elemento sensible, el acondicionamiento de señal, la conversión analógico–digital y parte del procesamiento necesario para obtener la magnitud medida.

La comunicación con el microcontrolador es digital. En los montajes utilizados habitualmente puede realizarse mediante I^2C; algunos modelos también permiten SPI.

Su interés experimental reside en que una única interfaz digital puede proporcionar una o varias magnitudes físicas ya convertidas a valores numéricos.

9.2.6. DS18B20 y el bus 1-Wire – Sensores digitales integrados

El DS18B20 es un termómetro digital muy utilizado, disponible también en sondas encapsuladas.

A diferencia de los sensores I2C, utiliza el protocolo 1-Wire. Cada dispositivo posee un identificador propio, por lo que varios sensores pueden compartir una misma línea de datos.

La arquitectura

$$\text{varios DS18B20} \longrightarrow \text{1-Wire} \longrightarrow \text{microcontrolador}$$

permite realizar medidas de temperatura distribuidas utilizando muy poco cableado.

9.2.7. Medición de distancia y tiempo de vuelo por ultrasonido

Un sensor ultrasónico permite determinar distancias midiendo el tiempo transcurrido entre la emisión de un pulso acústico y la recepción de su eco.

El HC-SR04 es un ejemplo habitual. Posee dos señales digitales principales:

TRIG: recibe del microcontrolador un pulso que inicia la medición;

ECHO: produce un pulso cuya duración representa el tiempo de ida y vuelta de la onda ultrasónica.

Si Δt es el tiempo entre emisión y recepción y c es la velocidad del sonido, la distancia al reflector es

$$d = \frac{c\,\Delta t}{2}.$$

El factor 1/2 aparece porque la onda recorre la distancia dos veces: desde el emisor hasta el obstáculo y desde éste nuevamente hasta el receptor.

La misma relación puede utilizarse en sentido inverso. Si la distancia d se conoce, puede determinarse experimentalmente la velocidad del sonido:

$$c = \frac{2d}{\Delta t}.$$

La velocidad del sonido depende de las propiedades del medio y, en el aire, varía principalmente con la temperatura. Cerca de temperatura ambiente puede utilizarse como aproximación

$$c \simeq 331 + 0{,}6\,T$$

con c en m/s y T en grados Celsius.

La medición de tiempos cortos exige considerar la resolución temporal del sistema, los retardos electrónicos, la geometría del reflector y la identificación correcta del eco.

9.2.8. Arduino – Microcontrolador

Arduino es una plataforma de hardware y software libre Banzi y Shiloh 2014 diseñada para facilitar la programacion de microcontroladores. El microcontrolador mas usado es el ATmega328P en el Arduino Uno, un AVR de 8 bits a 16 MHz.

Características del Arduino Uno

Parámetro	**Valor**
Microcontrolador	ATmega328P
Tensión de trabajo	5 V
Frecuencia de reloj	16 MHz
Memoria de programa	32 KB flash
RAM	2 KB SRAM
Pines digitales	14 (6 con PWM)
Entradas analógicas	6 (ADC de 10 bits)
Rango ADC	0 a 5 V con referencia por defecto
Corriente por pin	20 mA recomendados; 40 mA máximo absoluto
Comunicación	UART, SPI, I^2C y USB mediante conversor USB–serie

Programación del Arduino El Arduino se programa en C++ con la biblioteca de Arduino. El programa tiene dos funciones obligatorias:

Listing 9.9: estructura de un programa Arduino

```
// programa Arduino: leer temperatura y enviar por serie
// sensor LM35: salida 10 mV/grado, referencia 5V

const int PIN_TEMP = A0;    // pin analogico
const int BAUD = 9600;

void setup() {
    Serial.begin(BAUD);     // inicializar comunicacion serie
}

void loop() {
    int lectura = analogRead(PIN_TEMP);
    // ADC de 10 bits: 0-1023 para 0-5V
    float tension  = lectura * 5.0 / 1023.0;
    float temp_C   = tension * 100.0;  // LM35: 10 mV/grado

    Serial.print(millis());
    Serial.print(",");
    Serial.println(temp_C, 2);  // dos decimales

    delay(100);  // esperar 100 ms
}
```

Listing 9.10: control de un LED por PWM

```
const int PIN_LED = 9;    // pin PWM

void setup() {
    pinMode(PIN_LED, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    // leer el valor de brillo desde el puerto serie
    if (Serial.available() > 0) {
        int brillo = Serial.parseInt();  // 0-255
        brillo = constrain(brillo, 0, 255);
        analogWrite(PIN_LED, brillo);    // PWM
        Serial.print("Brillo: ");
        Serial.println(brillo);
    }
}
```

Recibir datos del Arduino en Scilab

Listing 9.11: lectura de Arduino desde Scilab

```
// el Arduino envia: tiempo_ms,temperatura cada 100 ms
```

```
PUERTO = '/dev/ttyUSB0'
BAUD   = 9600

// configurar el puerto serie
unix(sprintf('stty -F %s %d cs8 -cstopb -parenb raw', ..
             PUERTO, BAUD))

// adquirir 100 muestras
N       = 100
t_vec   = zeros(1, N)
T_vec   = zeros(1, N)

for i = 1:N
    [linea, ret] = unix_g(sprintf('head -n 1 %s', PUERTO))
    if ret == 0 & ~isempty(linea) then
        partes = strsplit(linea(1), ',')
        if size(partes,'*') >= 2 then
            t_vec(i) = strtod(partes(1)) / 1000  // ms a s
            T_vec(i) = strtod(partes(2))
        end
    end
end

printf('Temperatura media: %.2f C\n', mean(T_vec))
printf('Temperatura min:   %.2f C\n', min(T_vec))
printf('Temperatura max:   %.2f C\n', max(T_vec))
```

Listing 9.12: enviar comandos al Arduino

Enviar comandos al Arduino desde Scilab

```
// enviar el valor de brillo al Arduino
function enviar_al_arduino(puerto, valor)
    cmd = sprintf('echo -n "%d" > %s', valor, puerto)
    unix(cmd)
endfunction

PUERTO = '/dev/ttyUSB0'
unix(sprintf('stty -F %s 9600 cs8 -cstopb -parenb raw', PUERTO))

// secuencia de brillo
for brillo = [0, 64, 128, 192, 255, 128, 0]
    enviar_al_arduino(PUERTO, brillo)
    sleep(500)
end
```

9.2.9. ESP32 – microcontrolador

El ESP32 es un microcontrolador de 32 bits con WiFi y Espressif Systems 2026 Bluetooth integrados, fabricado por Espressif Systems. Es mas potente que el Arduino Uno y mucho

mas barato que la Raspberry Pi.

Características del ESP32

Parametro	Valor
Procesador	Xtensa LX6 dual-core 32 bits
Frecuencia	240 MHz
RAM	520 KB SRAM
Flash	4 MB (tipico)
WiFi	802.11 b/g/n (2.4 GHz)
Bluetooth	BT 4.2 y BLE
GPIO	hasta 34, con restricciones según el pin
ADC	ADC SAR de 12 bits; canales y rango dependen del modelo y de la atenuacion c
DAC	2 canales de 8 bits
Comunicacion	UART, SPI, I^2C, I^2S, TWAI (CAN), PWM
Tension de trabajo	3.3 V (**NO conectar 5V a pines**)

Servidor web en el ESP32 Una de las aplicaciones mas utiles del ESP32 es servir datos de sensores via HTTP sin necesidad de una computadora intermedia:

Listing 9.13: ESP32 como servidor web de sensores

```
// ESP32 con sensor DHT22 publicando datos via HTTP

#include <WiFi.h>
#include <WebServer.h>
#include <DHT.h>

const char* SSID     = "mi_red";
const char* PASSWORD = "mi_clave";

DHT dht(4, DHT22);   // sensor en pin 4
WebServer server(80);

void handleRoot() {
    float T = dht.readTemperature();
    float H = dht.readHumidity();
    String resp = "{\"temperatura\":"  + String(T, 2) +
                  ",\"humedad\":"      + String(H, 2) + "}";
    server.send(200, "application/json", resp);
}

void setup() {
    dht.begin();
    WiFi.begin(SSID, PASSWORD);
    while (WiFi.status() != WL_CONNECTED) delay(500);
    server.on("/", handleRoot);
    server.begin();
```

```
}

void loop() {
    server.handleClient();
}
```

Listing 9.14: Scilab lee sensores del ESP32 via HTTP

```
// Scilab lee datos del ESP32 via HTTP
IP_ESP32 = '192.168.1.45'
N        = 60   // mediciones
T_vec    = zeros(1, N)
H_vec    = zeros(1, N)

for i = 1:N
    // descargar el JSON
    archivo = '/tmp/esp32_dato.json'
    cmd = sprintf('curl -s http://%s/ > %s', IP_ESP32, archivo)
    unix(cmd)

    // leer y parsear el JSON manualmente
    linea = mgetl(archivo)
    if ~isempty(linea) then
        // extraer temperatura: buscar "temperatura":VALOR
        idx_t = strindex(linea, '"temperatura":')
        if ~isempty(idx_t) then
            sub = part(linea, idx_t(1)+14:$)
            idx_c = strindex(sub, ',')
            T_vec(i) = strtod(part(sub, 1:idx_c(1)-1))
        end
        // extraer humedad
        idx_h = strindex(linea, '"humedad":')
        if ~isempty(idx_h) then
            sub2 = part(linea, idx_h(1)+10:$)
            idx_f = strindex(sub2, '}')
            H_vec(i) = strtod(part(sub2, 1:idx_f(1)-1))
        end
    end
    sleep(1000)   // una medicion por segundo
end

printf('Temperatura media: %.2f C\n', mean(T_vec))
printf('Humedad media:     %.2f %%\n', mean(H_vec))
```

9.2.10. Raspberry Pi – computadora embebida

La Raspberry Pi es una computadora completa en una placa del Raspberry Pi Ltd. 2026 tamanio de una tarjeta de credito. A diferencia del Arduino y el ESP32, ejecuta un sistema operativo Linux completo (Raspberry Pi OS, basado en Debian).

Diferencias fundamentales con microcontroladores

Aspecto	Arduino/ESP32	Raspberry Pi
Tipo	Microcontrolador	Computadora
Sistema operativo	Ninguno (bare metal)	Linux
Procesador	8-240 MHz 8/32 bits	1.8 GHz 64 bits
RAM	2 KB - 520 KB	1-8 GB
Almacenamiento	32 KB - 4 MB flash	Tarjeta SD
Tiempo de inicio	Instantaneo	20-40 segundos
Consumo	0.1-0.3 W	3-7 W
ADC	Si (integrado)	No (requiere externo)
GPIO	Arduino Uno: 5 V; ESP32: 3.3 V	Raspberry Pi: 3.3 V; no tolera 5 V
WiFi	En ESP32 si	Si (Pi 3 y 4)

La Raspberry Pi es adecuada cuando se necesita procesamiento pesado (vision artificial, aprendizaje automatico, servidor web complejo) o una interfaz grafica. No tiene ADC propio: para medir senales analogicas se conecta un ADC externo por SPI o I2C (el MCP3008 es popular: 8 canales de 10 bits).

Acceso a GPIO desde Python en Raspberry Pi

Listing 9.15: leer y controlar GPIO en Raspberry Pi

```
import RPi.GPIO as GPIO
import time

# configurar el modo de numeracion
GPIO.setmode(GPIO.BCM)    # usar numeracion BCM

# configurar pines
GPIO.setup(17, GPIO.IN,  pull_up_down=GPIO.PUD_UP)  # entrada
GPIO.setup(18, GPIO.OUT)                              # salida
   LED

try:
    while True:
        boton = GPIO.input(17)     # leer boton
        GPIO.output(18, not boton)  # encender LED si boton
           presionado
        time.sleep(0.01)
except KeyboardInterrupt:
    GPIO.cleanup()
```

Listing 9.16: leer ADC MCP3008 por SPI en Raspberry Pi

```
import spidev
import time

spi = spidev.SpiDev()
spi.open(0, 0)        # bus 0, chip select 0
```

```
spi.max_speed_hz = 1000000

def leer_adc(canal):
    # protocolo MCP3008: enviar 3 bytes, recibir 3 bytes
    r = spi.xfer2([1, (8 + canal) << 4, 0])
    valor = ((r[1] & 3) << 8) + r[2]   # 10 bits: 0-1023
    return valor

try:
    while True:
        v = leer_adc(0)
        tension = v * 3.3 / 1023.0
        print(f"ADC canal 0: {v:4d}  {tension:.3f} V")
        time.sleep(0.1)
except KeyboardInterrupt:
    spi.close()
```

9.2.11. Módulos ADAM – Adquisición industrial distribuida

Los módulos ADAM constituyen un ejemplo de instrumentación industrial distribuida. Según el modelo pueden disponer de entradas analógicas, salidas analógicas, entradas y salidas digitales, contadores o relés.

En la serie ADAM-4000 la comunicación se realiza tradicionalmente mediante RS-485.

RS-485 define principalmente características eléctricas de la comunicación. Permite distancias considerablemente mayores que buses destinados a comunicación dentro de un equipo, como I2C, y permite conectar varios instrumentos sobre un mismo bus.

La arquitectura típica es

$$\text{sensores} \longrightarrow \text{ADAM} \longrightarrow \text{RS-485} \longrightarrow \text{computadora}.$$

Cada módulo posee una dirección. La computadora actúa normalmente como maestro: envía una orden dirigida a un módulo y éste devuelve una respuesta.

En diversos módulos ADAM-4000 se utiliza un protocolo ASCII. Una orden puede contener conceptualmente

$$\text{delimitador} + \text{dirección} + \text{comando} + \text{datos} + \text{terminador}.$$

Algunos modos utilizan además mecanismos de comprobación tales como checksum. La sintaxis exacta depende del modelo y debe consultarse siempre en su manual.

Una computadora moderna puede conectarse al bus mediante un conversor USB–RS-485, que en Linux suele aparecer como un dispositivo del tipo

```
/dev/ttyUSB0
```

El conversor USB–RS-485 actúa como puente entre la computadora y el bus RS-485; no modifica el protocolo de comandos utilizado por el módulo ADAM.

Comunicación con ADAM desde Scilab mediante Tcl

En programas históricos de laboratorio puede encontrarse Tcl utilizado como capa de comunicación entre Scilab y el sistema operativo.

Scilab realiza el cálculo, tratamiento y representación de los datos, mientras Tcl abre el puerto serie, transmite órdenes y recibe respuestas.

Las operaciones fundamentales son conceptualmente:

open: abrir el puerto;

fconfigure: establecer velocidad, paridad y formato;

puts: transmitir una orden;

gets: recibir una respuesta;

close: cerrar el puerto.

Desde Scilab, funciones como

```
TCL_EvalStr(...)
TCL_GetVar(...)
```

permiten ejecutar instrucciones Tcl y recuperar posteriormente los datos.

Por ejemplo, la secuencia

```
Scilab -> Tcl -> puerto serie -> RS-485 -> ADAM
ADAM -> RS-485 -> puerto serie -> Tcl -> Scilab
```

separa claramente tres niveles diferentes:

1. el programa que analiza los datos;
2. el mecanismo utilizado para acceder al puerto;
3. el protocolo propio del instrumento.

Esta separación es general y continúa siendo válida aunque Tcl sea sustituido por Python, C u otra biblioteca de comunicaciones.

Funciones de comunicación utilizadas en el laboratorio Las siguientes rutinas corresponden al sistema ADAM disponible en el laboratorio y muestran cómo Scilab utiliza Tcl para acceder al puerto serie y enviar comandos al módulo.

```
function habilitar_ADAM()
    TCL_EvalStr(["set leer [ open /dev/ttyS0 r+]"
                 "fconfigure $leer -mode 9600,n,8,1 -buffering
                    line -eofchar {}"])
endfunction

function deshabilitar_ADAM()
    TCL_EvalStr("close $leer")
endfunction

function b=leer_configuracion_ADAM(numero)
    num = sprintf("puts $leer \$0%d2", numero)
    TCL_EvalStr(num)
    TCL_EvalStr("set b [ gets $leer]")
    b = TCL_GetVar("b")
endfunction

function b=conf_almacenamiento_ADAM(numero)
    num = sprintf("puts $leer @0%dCFF1000001", numero)
    TCL_EvalStr(num)
    TCL_EvalStr("set b [ gets $leer]")
    b = TCL_GetVar("b")
endfunction

function b=medir_ADAM(canal, numero)
    num = sprintf("puts $leer #0%d%d", numero, canal)
    TCL_EvalStr(num)
    TCL_EvalStr("after 100")
    TCL_EvalStr("set b [ gets $leer]")
    b = TCL_GetVar("b")
endfunction
```

```
function b=conf_almacenamiento_ADAM(numero)
num= sprintf("puts $leer @0%dCFF1000001",numero)
TCL_EvalStr(num)
TCL_EvalStr("set b [ gets $leer]")
b=TCL_GetVar('b')
endfunction

function b=leer_configuracion_ADAM(numero)
num= sprintf("puts $leer \$0%d2",numero)
TCL_EvalStr(num)
TCL_EvalStr("set b [ gets $leer]")
b=TCL_GetVar('b')
endfunction
```

```
function deshabilitar_ADAM()
TCL_EvalStr("close $leer ")
endfunction

function habilitar_ADAM()
TCL_EvalStr(["set leer [ open /dev/ttyS0 r+]"
             "fconfigure $leer -mode 9600,n,8,1 -buffering line -eofchar {} "
    ])
endfunction

function b=leer_configuracion_ADAM(numero)
num= sprintf("puts $leer \$0%d2",numero)
TCL_EvalStr(num)
TCL_EvalStr("set b [ gets $leer]")
b=TCL_GetVar('b')
endfunction

function b=medir_ADAM(canal,numero)
num= sprintf("puts $leer #0%d%d",numero,canal)
TCL_EvalStr(num)
TCL_EvalStr("after 100")
TCL_EvalStr("set b [ gets $leer]")
b=TCL_GetVar('b')
endfunction
```

9.2.12. TEMPer2 – Instrumento digital completo

El **TEMPer2** es una familia de termómetros USB de bajo costo. A diferencia de un LM35 conectado a un ADC, el dispositivo contiene el sensor, la electrónica de conversión y la interfaz USB; para la computadora aparece como un periférico digital. Muchas versiones tienen un sensor interno y una entrada para una sonda externa, de modo que pueden obtenerse dos temperaturas.

No existe un único TEMPer2 completamente uniforme. Se comercializaron varias revisiones de hardware y firmware, con distintos identificadores USB y protocolos. Dos instrumentos con el mismo nombre comercial pueden, por tanto, requerir programas o procedimientos de comunicación diferentes. Por eso el primer paso en Linux debe ser **identificar experimentalmente el dispositivo** y no suponer un protocolo a partir del nombre comercial.

Guía para usar el termómetro USB TEMPer2

Nos referimos específicamente al (ID `3553:a001`, firmware `TEMPer2_V4.1`) en Ubuntu/-Debian.

Qué hay que bajar de internet

La página del fabricante es `https://pcsensor.com/product/pcsensor-usb-temperature-sensor-wa`

Para usarlo en GNU/Linux Solo se necesita **un programa**: `temper.py`, del repositorio `greg-kodama/temper` en GitHub. Es un script en Python, no requiere compilación.

git clone https://github.com/greg-kodama/temper.git cd temper

Y para instalar la dependencia:

```
sudo apt install python3-serial
```

y se realiza una lectura mediante

```
sudo ./temper.py --force 3553:a001
```

Cuando la sonda externa está conectada se obtienen dos temperaturas. La primera corresponde al sensor interno del dispositivo USB y la segunda a la sonda externa.

También puede solicitarse la salida en formato JSON:

```
sudo ./temper.py --force 3553:a001 --json
```

lo que facilita incorporar las mediciones a programas de adquisición y almacenarlas posteriormente en archivos CSV.

Requisitos del sistema:

- `git` — para bajar el repositorio.
- `python3` — ya viene instalado.
- `python3-serial` — única dependencia extra.

Nota: el driver `pcsensor` (en C) **no sirve** para este modelo: solo reconoce los ID viejos `0c45:7401/7402`. El TEMPer2 nuevo tiene ID `3553:a001`.

1. Paso 1 — Bajar al disco local

   Elegir un directorio de trabajo y clonar el repositorio:

   ```
   cd ~/mi/directorio/de/trabajo
   git clone https://github.com/greg-kodama/temper.git
   cd temper
   ```

   Eso crea la carpeta `temper/` con el programa `temper.py` adentro.

2. Paso 2 — Instalar la dependencia y probar

   ```
   sudo apt install python3-serial
   ```

3. Y ya se puede usar (hay que ejecutarlo como `root`):

   ```
   sudo ./temper.py --force 3553:a001
   ```

Salida esperada (dos lecturas si el sensor externo está conectado):

```
Bus 001 Dev 006 3553:a001 TEMPer2_V4.1 24.18C 75.52F -
   22.31C 72.16F -
```

- La primera temperatura es el **sensor interno** del stick USB.
- La segunda es el **sensor externo** (el cable).

Solución de problemas

En algunas oportunidades el dispositivo puede no responder correctamente a la consulta inicial del firmware y aparecer el mensaje

`Cannot read device firmware identifier`

`Cannot read device firmware identifier`

La respuesta del dispositivo a veces llega corrupta. Desenchufar el TEMPer2, esperar 5 segundos, reenchufar y reintentar:

En tal caso debe desconectarse el instrumento durante algunos segundos, volver a conectarlo y repetir la lectura. La respuesta USB de esta familia no siempre resulta estable en el primer intento.

```
for i in 1 2 3 4 5; do
    sudo ./temper.py --force 3553:a001 && break
    sleep 1
done
```

Verificar que la PC lo detecta

```
lsusb | grep -i temper
```

Debe aparecer: `ID 3553:a001 PCsensor TEMPer2`.

No usar `pip install`

En Debian/Ubuntu modernos el sistema bloquea la instalación de paquetes Python sobre el sistema (*externally managed*, PEP 668). Esta guía no necesita `pip`.

Registrar mediciones en CSV

Crear una vez el archivo `~/log_temper.py`:

```
import sys, json, datetime

d = json.load(sys.stdin)[0]
print(
    datetime.datetime.now().isoformat(sep=" ",
       timespec="seconds"),
    d.get("internal temperature"),
```

```
    d.get("external temperature"),
    sep=",",
)
```

Y para grabar una medición cada 3 segundos en `~/temperaturas.csv`:

```
cd ~/mi/directorio/de/trabajo/temper
while true; do
    sudo ./temper.py --force 3553:a001 --json \
        | python3 ~/log_temper.py >> ~/temperaturas.csv
    sleep 3
done
```

Se detiene con `Ctrl+C`. El CSV resultante (fecha, temperatura interna, temperatura externa) se puede leer en Scilab con `csvRead`.

Identificación en Linux

El dispositivo puede identificarse mediante

```
lsusb
```

En la unidad disponible en el laboratorio se obtiene:

```
ID 3553:a001 PCsensor TEMPer2
```

El instrumento utiliza USB/HID para comunicarse con la computadora. El sistema operativo proporciona la interfaz de comunicación y el programa `temper.py` implementa el protocolo necesario para obtener las mediciones.

Sensor interno y sonda externa. La unidad estudiada dispone de una sonda externa además del sensor contenido en el propio instrumento. El programa de adquisición permite registrar simultáneamente las temperaturas de ambos sensores.

Antes de realizar medidas cuantitativas deberá determinarse experimentalmente qué dato corresponde al sensor interno y cuál a la sonda externa. Una forma simple de hacerlo consiste en mantener uno aproximadamente a temperatura ambiente y producir una variación claramente identificable sobre el otro.

También deberán determinarse o comprobarse:

- resolución de la lectura;
- exactitud mediante comparación con un termómetro de referencia;
- tiempo de respuesta;
- repetibilidad;
- posible diferencia sistemática entre sensor interno y externo.

La resolución numérica proporcionada por el instrumento no debe confundirse con su exactitud.

Interés experimental. El TEMPer2 constituye un ejemplo de **instrumento USB completo**. En contraste con una experiencia construida a partir de un sensor analógico, un ADC y un microcontrolador separados, aquí gran parte de la cadena de medida está encerrada dentro del instrumento.

Esto permite estudiar otro aspecto de la adquisición experimental: **identificación del hardware, interfaces USB, protocolos digitales, compatibilidad del software y extracción de datos de un instrumento real**.

9.2.13. Comparación de plataformas

Criterio	Arduino Uno	ESP32	Raspberry Pi 4
Costo aproximado	$5-15	$5-10	$35-80
Uso tipico	Medicion y control simple, bajo consumo	Medicion con WiFi, IoT	Procesamiento complejo, servidor, vision
Lenguaje	C++	C++, MicroPython	Python, C y otros disponibles en GNU/Linux
ADC integrado	10 bits	12 bits	No
Tiempo real	Si	Si (RTOS)	No garantizado
Comunidad	Muy grande	Grande	Muy grande

Elección de la plataforma

Arduino: cuando se necesita un sistema simple, confiable y de bajo consumo para una tarea especifica: leer un sensor y enviar los datos por serie, controlar un motor, hacer parpadear un LED. Sin sistema operativo, tiempo de inicio instantaneo, puede funcionar anos sin mantenimiento.

ESP32: cuando se necesita conectividad WiFi sin una computadora adicional. Ideal para IoT: sensor que publica datos en un servidor web, actuador controlado remotamente, sistema de monitoreo sin cables.

Raspberry Pi: cuando se necesita procesamiento pesado, una interfaz grafica, varios servicios simultaneos o un lenguaje de alto nivel con bibliotecas ricas. Ideal como servidor central de un sistema de adquisicion, como plataforma de vision artificial o como computadora embebida con pantalla.

Combinacion Arduino + Raspberry Pi: muy comun en laboratorio. El Arduino se ocupa de la adquisicion en tiempo real y el control preciso; la Raspberry Pi recibe los datos por serie, los procesa, los almacena y los publica en la red.

9.3. Sistema completo de adquisición y control

Un sistema tipico de laboratorio integra todos los elementos descritos:

1. **Sensores** convierten las magnitudes fisicas en senales electricas.
2. **Acondicionadores** amplifican y filtran las senales.
3. **Arduino o ADC externo** digitaliza las senales y las envia por puerto serie.
4. **Computadora o Raspberry Pi** recibe los datos, los almacena y puede procesarlos con las herramientas disponibles. En una computadora de laboratorio puede utilizarse Scilab para el analisis numerico y la representacion grafica.
5. El usuario puede enviar comandos de vuelta al Arduino para controlar actuadores.
6. Opcionalmente, un servidor web (como el desarrollado en este practico) publica los datos y permite el control remoto desde cualquier navegador.

Listing 9.17: sistema completo de adquisicion y control

```
// sistema de control de temperatura
// Arduino mide la temperatura y controla un calefactor via PWM
// Scilab recibe los datos, implementa el PID y envia la accion

PUERTO = '/dev/ttyUSB0'
BAUD   = 9600
unix(sprintf('stty -F %s %d cs8 -cstopb -parenb raw', PUERTO,
   BAUD))

SETPOINT = 40   // temperatura deseada en grados
Kp = 5.0; Ki = 0.2; Kd = 0.5; Ts = 0.5

integral   = 0
error_prev = 0
N_pasos    = 200

t_vec   = zeros(1, N_pasos)
T_vec   = zeros(1, N_pasos)
u_vec   = zeros(1, N_pasos)

for i = 1:N_pasos
    // leer temperatura del Arduino
    [linea, ret] = unix_g(sprintf('head -n 1 %s', PUERTO))
    T = SETPOINT   // valor por defecto si falla la lectura
    if ret == 0 & ~isempty(linea) then
        partes = strsplit(linea(1), ',')
        if size(partes,'*') >= 2 then
            T = strtod(partes(2))
        end
    end
```

```
    // calcular accion PID
    [u, integral, error_prev] = pid_paso(SETPOINT, T, ..
                                Kp, Ki, Kd, Ts, ..
                                integral, error_prev, 0, 255)

    // enviar accion al Arduino (PWM del calefactor)
    unix(sprintf('echo -n "%d" > %s', int(u), PUERTO))

    t_vec(i) = i * Ts
    T_vec(i) = T
    u_vec(i) = u

    printf('t=%.1f s  T=%.2f C  u=%.0f\n', t_vec(i), T, u)
    sleep(int(Ts * 1000))
end

// graficar con Gnuplot
fd = mopen('/tmp/control.txt', 'w')
for i = 1:N_pasos
    mfprintf(fd, '%.2f  %.4f  %.2f\n', t_vec(i), T_vec(i),
       u_vec(i))
end
mclose(fd)

NL = ascii(10); Q = ascii(34)
DAT = Q + '/tmp/control.txt' + Q
gp = 'set terminal png size 900,400' + NL
gp = gp + 'set output ' + Q + '/tmp/control.png' + Q + NL
gp = gp + 'set multiplot layout 1,2' + NL
gp = gp + 'set grid' + NL
gp = gp + 'set title ' + Q + 'Temperatura' + Q + NL
gp = gp + 'set xlabel ' + Q + 't [s]' + Q + NL
gp = gp + 'set ylabel ' + Q + 'T [C]' + Q + NL
gp = gp + 'plot ' + DAT + ' using 1:2 with lines lw 2 lc '+ ..
          Q+'blue'+Q+' title '+Q+'medida'+Q+', \' + NL
gp = gp + '     ' + sprintf('%g', SETPOINT) + ..
          ' with lines lt 2 lc '+Q+'red'+Q+ ..
          ' title '+Q+'setpoint'+Q + NL
gp = gp + 'set title ' + Q + 'Accion de control' + Q + NL
gp = gp + 'set ylabel ' + Q + 'u [PWM]' + Q + NL
gp = gp + 'plot ' + DAT + ' using 1:3 with lines lw 2 lc ' + ..
          Q+'green'+Q+' notitle' + NL
gp = gp + 'unset multiplot' + NL
fd = mopen('/tmp/control.gnu', 'w')
mfprintf(fd, '%s', gp)
mclose(fd)
unix('gnuplot /tmp/control.gnu')
printf('Grafica: /tmp/control.png\n')
```

9.4. Laboratorios

Los practicos que siguen tienen tres niveles de equipamiento: **Nivel A** requiere solo la computadora y sus puertos integrados (microfono, USB); **Nivel B** agrega un Arduino o ESP32 con componentes basicos; **Nivel C** agrega sensores y actuadores adicionales. Cada practico indica el nivel requerido.

1. **Ejercicio:** Adquisicion de audio y analisis espectral

   Nivel A

   Objetivo.

   Familiarizarse con el proceso de adquisicion digital de señales utilizando como entrada el microfono de una computadora. Analizar la señal en los dominios temporal y frecuencial, verificar experimentalmente el teorema de Nyquist y observar el fenomeno de aliasing.

   Materiales.

   - Computadora con microfono integrado o externo.
   - Herramientas para adquisicion, procesamiento numerico y representacion grafica de los datos.
   - Opcionalmente: diapason, flauta u otro instrumento que produzca tonos aproximadamente puros.

   Parte A — Captura y analisis de una señal de audio.

   Registrar aproximadamente tres segundos de una señal de audio monoaural con frecuencia de muestreo de 44100 Hz.

   En GNU/Linux puede utilizarse, por ejemplo:

   ```
   arecord -d 3 -c 1 -r 44100 -f S16_LE -t wav /tmp/audio.wav
   ```

   Silbar, cantar una nota o hacer sonar un diapason durante la captura.

   A partir de la señal registrada determinar:

   - frecuencia de muestreo;
   - duracion de la señal;
   - amplitud maxima;
   - valor RMS;
   - espectro de frecuencias;
   - frecuencia dominante.

   Representar la señal en funcion del tiempo y su espectro de frecuencias.

   Scilab. Una posible implementacion del analisis anterior en Scilab es la siguiente.

```
// leer y analizar el archivo WAV
[x, fs] = wavread('/tmp/audio.wav')

if size(x,1) > 1 then
    senal = x(1,:)
else
    senal = x
end

t = (0:length(senal)-1) / fs

// estadisticas basicas
printf('Frecuencia de muestreo: %d Hz\n', fs)
printf('Duracion: %.3f s\n', length(senal)/fs)
printf('Amplitud maxima: %.4f\n', max(abs(senal)))
printf('Valor RMS: %.4f\n', sqrt(mean(senal.^2)))

// FFT y espectro
N   = length(senal)
Y   = fft(senal)
N2  = floor(N/2)
f   = (0:N2-1) * fs / N
amp = abs(Y(1:N2)) * 2/N

// frecuencia dominante
[amp_max, idx] = max(amp)
printf('Frecuencia dominante: %.1f Hz\n', f(idx))

// datos para graficar con Gnuplot
fd = mopen('/tmp/audio_t.txt', 'w')
n_plot = min(4000, length(t))
for i = 1:n_plot
    mfprintf(fd, '%.6f  %.6f\n', t(i), senal(i))
end
mclose(fd)

fd = mopen('/tmp/audio_f.txt', 'w')
for i = 1:length(f)
    mfprintf(fd, '%.2f  %.6f\n', f(i), amp(i))
end
mclose(fd)

NL = ascii(10)
Q  = ascii(34)

gp = 'set terminal png size 900,500' + NL
gp = gp + 'set output ' + Q + '/tmp/audio.png' + Q + NL
gp = gp + 'set multiplot layout 2,1' + NL
gp = gp + 'set title ' + Q + 'Senal en el tiempo' + Q + NL
gp = gp + 'set xlabel ' + Q + 'Tiempo [s]' + Q + NL
gp = gp + 'set ylabel ' + Q + 'Amplitud' + Q + NL
```

```
gp = gp + 'set grid' + NL
gp = gp + 'plot ' + Q+'/tmp/audio_t.txt'+Q + ..
          ' with lines notitle' + NL
gp = gp + 'set title ' + Q + 'Espectro de frecuencias' + Q +
   NL
gp = gp + 'set xlabel ' + Q + 'Frecuencia [Hz]' + Q + NL
gp = gp + 'set ylabel ' + Q + 'Amplitud' + Q + NL
gp = gp + 'set xrange [0:5000]' + NL
gp = gp + 'plot ' + Q+'/tmp/audio_f.txt'+Q + ..
          ' with lines notitle' + NL
gp = gp + 'unset multiplot' + NL

fd = mopen('/tmp/audio.gnu', 'w')
mfprintf(fd, '%s', gp)
mclose(fd)
unix('gnuplot /tmp/audio.gnu')
```

Parte B — Verificacion del teorema de Nyquist.

Capturar una señal semejante utilizando diferentes frecuencias de muestreo, por ejemplo:

```
arecord -d 2 -c 1 -r 44100 -f S16_LE -t wav /tmp/fs44100.wav
arecord -d 2 -c 1 -r 22050 -f S16_LE -t wav /tmp/fs22050.wav
arecord -d 2 -c 1 -r  8000 -f S16_LE -t wav /tmp/fs8000.wav
```

Obtener el espectro de cada registro y comparar los resultados. Determinar para cada frecuencia de muestreo la frecuencia de Nyquist.

Parte C — Demostracion de aliasing.

Generar numericamente muestras de una señal sinusoidal de 20 kHz utilizando una frecuencia de muestreo de 22050 Hz, sin aplicar un filtro anti-aliasing.

La frecuencia aparente esperada es

$$f_{\text{alias}} = |20000 - 22050| = 2050 \text{ Hz}.$$

Calcular el espectro de las muestras y comprobar la aparicion de un pico alrededor de 2050 Hz.

Scilab. Una posible implementacion en Scilab es la siguiente.

```
fs = 22050
f0 = 20000
N = fs

t = (0:N-1)/fs
x = sin(2*%pi*f0*t)
Y = fft(x)
N2 = floor(N/2)
f  = (0:N2-1)*fs/N
amp = abs(Y(1:N2))*2/N
```

```
[a, idx] = max(amp)
printf('Frecuencia aparente: %.1f Hz\n', f(idx))
```

Preguntas.

1. ¿Que frecuencias maximas pueden representarse con cada una de las frecuencias de muestreo utilizadas?
2. ¿Como cambia el espectro al disminuir la frecuencia de muestreo? ¿Se pierde informacion? ¿Cual?
3. Mida la frecuencia de una nota conocida, por ejemplo el La de un diapason de 440 Hz, y determine el error relativo.
4. Calcule la potencia de la señal grabada correspondiente a la banda de frecuencias de la voz humana, aproximadamente 300–3400 Hz.
5. Explique por que la componente de 20 kHz aparece como una frecuencia de aproximadamente 2050 Hz al muestrear a 22050 Hz.

2. **Ejercicio:** Calibracion y errores de medicion

Nivel A

Objetivo.

Comprender los distintos tipos de error de un sistema de medicion, estudiar estadisticamente mediciones repetidas y aplicar un procedimiento de calibracion de dos puntos.

Parte A — Simulacion de un sensor imperfecto.

Simular un sensor cuya respuesta tenga error de ganancia, error de offset y ruido aleatorio. Utilizar, por ejemplo, el modelo

$$T_m = 1{,}05\,T + 2{,}3 + n,$$

donde T es el valor verdadero, T_m el valor indicado por el instrumento y n un termino aleatorio.

Realizar 50 mediciones para

$$T = 25\,^{\circ}\mathrm{C}.$$

Calcular:

- media;
- desviacion estandar;
- desviacion estandar de la media;
- incertidumbre expandida utilizando $k = 2$.

Scilab. Una posible implementacion de la simulacion en Scilab es la siguiente.

```
function T_med = sensor_imperfecto(T_real)
    ganancia = 1.05
    offset   = 2.3
    // ruido uniforme entre -0.25 y +0.25 grados
    ruido = 0.5*(rand()-0.5)
    T_med = ganancia*T_real + offset + ruido
endfunction

N = 50
mediciones = zeros(1,N)

for i = 1:N
    mediciones(i) = sensor_imperfecto(25.0)
end

media = mean(mediciones)
s     = stdev(mediciones)
u     = s/sqrt(N)

printf('Valor verdadero:           25.0000 C\n')
printf('Media de las mediciones: %.4f C\n', media)
printf('Desvio estandar:         %.4f C\n', s)
printf('Desvio de la media:      %.4f C\n', u)
printf('Incertidumbre k=2:       %.4f C\n', 2*u)
```

Parte B — Calibracion de dos puntos.

Utilizar dos referencias conocidas, por ejemplo

$$T_1 = 0\,^\circ\mathrm{C}, \qquad T_2 = 100\,^\circ\mathrm{C}.$$

Realizar varias mediciones en cada punto y determinar una correccion lineal de la forma

$$T = aT_m + b.$$

Verificar posteriormente la calibracion utilizando varios valores intermedios.

Scilab. Una posible implementacion del procedimiento en Scilab es la siguiente.

```
T_ref = [0.0, 100.0]
N_cal = 30

med_baja = zeros(1,N_cal)
med_alta = zeros(1,N_cal)

for i = 1:N_cal
    med_baja(i) = sensor_imperfecto(T_ref(1))
    med_alta(i) = sensor_imperfecto(T_ref(2))
end
```

```
m_baja = mean(med_baja)
m_alta = mean(med_alta)

printf('Punto bajo: T_ref=%.1f T_medida=%.4f\n', ..
       T_ref(1), m_baja)
printf('Punto alto: T_ref=%.1f T_medida=%.4f\n', ..
       T_ref(2), m_alta)

a = (T_ref(2)-T_ref(1))/(m_alta-m_baja)
b = T_ref(1)-a*m_baja

printf('T = %.6f*T_medida + %.6f\n', a, b)

T_test = [25,50,75]

for i = 1:length(T_test)
    lecturas = zeros(1,20)
    for j = 1:20
        lecturas(j) = sensor_imperfecto(T_test(i))
    end
    T_corregida = a*mean(lecturas)+b
    error = T_corregida-T_test(i)
    printf('T_ref=%5.1f  T_corr=%.4f  error=%+.4f C\n', ..
           T_test(i), T_corregida, error)
end
```

Parte C — Efecto del promediado sobre el ruido.

Aplicar la correccion de calibracion obtenida anteriormente. Estudiar como disminuye la incertidumbre estadistica de la media al aumentar el numero de mediciones. Para ruido independiente se espera aproximadamente

$$u_{\bar{x}} \simeq \frac{s}{\sqrt{N}}.$$

Distinguir este efecto de los errores sistematicos, que no desaparecen mediante promediado.

Scilab. El siguiente ejemplo realiza muchas repeticiones para cada valor de N y calcula experimentalmente la dispersion de las medias.

```
N_vec = [1 2 4 8 16 32 64 128 256 512]
M = 200
T_real = 37.0
sigma_medias = zeros(1,length(N_vec))

for k = 1:length(N_vec)
    N = N_vec(k)
    promedios = zeros(1,M)
    for j = 1:M
        lecturas = zeros(1,N)
        for i = 1:N
```

```
            y = sensor_imperfecto(T_real)
            lecturas(i) = a*y+b
        end
        promedios(j) = mean(lecturas)
    end
    sigma_medias(k) = stdev(promedios)
end

sigma_teo = sigma_medias(1)./sqrt(N_vec)
disp([N_vec' sigma_medias' sigma_teo'])
```

Representar $\sigma_{\bar{x}}$ en funcion de N y comparar con la dependencia $1/\sqrt{N}$.

Preguntas.

1. ¿Cual es la incertidumbre de una sola medicion?
2. ¿Cuantas mediciones son necesarias para reducir a la mitad la incertidumbre de la media?
3. Despues de calibrar, ¿que tipos de error pueden permanecer?
4. ¿Por que el promediado reduce el ruido aleatorio pero no elimina los errores sistematicos?
5. Si el sensor presenta no linealidad ademas de offset y error de ganancia, ¿es suficiente una calibracion de dos puntos?
6. Proponga un procedimiento de calibracion utilizando cinco puntos de referencia.

3. **Ejercicio:** Adquisicion de temperatura con LM35

   Nivel B

   Objetivo.

   Construir un sistema de adquisicion de temperatura utilizando un sensor LM35, digitalizar su señal, registrar las mediciones en una computadora y analizar los datos obtenidos.

   Materiales.

   - Plataforma de adquisicion con conversor A/D.
   - Sensor LM35.
   - Condensador de desacoplo de 100 nF.
   - Cables y protoboard.

   Como implementacion de referencia se utilizara un Arduino Uno.

   Conexion.

   Para la implementacion con Arduino:

```
LM35

Pin 1 (VCC)  -> 5 V Arduino
Pin 2 (OUT)  -> A0 Arduino
Pin 3 (GND)  -> GND Arduino

Condensador de 100 nF entre VCC y GND,
ubicado cerca del LM35.
```

Adquisicion.

El sistema debe realizar periodicamente las siguientes operaciones:

1. leer la salida del LM35 mediante el ADC;
2. promediar varias conversiones para reducir el efecto del ruido;
3. convertir el resultado a temperatura;
4. asociar cada medicion con una referencia temporal;
5. transmitir o almacenar los datos utilizando el formato

   `tiempo_ms,temperatura`

Como implementacion de referencia se proporciona el siguiente programa para Arduino.

```
// arduino_temperatura.ino
const int PIN_T  = A0;
const int N_PROM = 16;

void setup() {
    Serial.begin(115200);
}

void loop() {
    long suma = 0;
    for (int i = 0; i < N_PROM; i++) {
        suma += analogRead(PIN_T);
        delay(2);
    }

    float lectura = suma/(float)N_PROM;
    float tension = lectura*5.0/1024.0;
    float temp = tension*100.0;

    Serial.print(millis());
    Serial.print(",");
    Serial.println(temp,3);

    delay(100);
}
```

Registro y analisis de los datos.

Registrar aproximadamente 120 segundos de mediciones. A partir de los datos determinar:

- numero de muestras validas;
- frecuencia media de adquisicion;
- temperatura media;
- desviacion estandar;
- temperatura minima;
- temperatura maxima;
- rango observado;
- resolucion teorica debida al ADC.

Representar la temperatura en funcion del tiempo.

Scilab. Una posible implementacion del registro y analisis mediante Scilab en GNU/Linux es la siguiente.

```
PUERTO   = '/dev/ttyACM0'
BAUD     = 115200
DURACION = 120
ARCHIVO  = '/tmp/temperatura.csv'

unix(sprintf('stty -F %s %d cs8 -cstopb -parenb raw', ..
             PUERTO, BAUD))
unix(sprintf('timeout %d cat %s > %s', ..
             DURACION, PUERTO, ARCHIVO))

lineas = mgetl(ARCHIVO)
n = size(lineas,'*')
datos = zeros(n,2)
validos = 0

for i = 1:n
    partes = strsplit(lineas(i),',')
    if size(partes,'*') >= 2 then
        t_ms = strtod(partes(1))
        temp = strtod(partes(2))
        if ~isnan(t_ms) & ~isnan(temp) then
            validos = validos+1
            datos(validos,1) = t_ms/1000
            datos(validos,2) = temp
        end
    end
end

datos = datos(1:validos,:)
t    = datos(:,1)'
temp = datos(:,2)'
```

```
printf('Muestras validas:   %d\n', validos)
printf('Frecuencia media:   %.2f Hz\n', validos/DURACION)
printf('Temperatura media:  %.4f C\n', mean(temp))
printf('Desvio estandar:    %.4f C\n', stdev(temp))
printf('Temperatura minima: %.4f C\n', min(temp))
printf('Temperatura maxima: %.4f C\n', max(temp))
printf('Rango total:        %.4f C\n', max(temp)-min(temp))

resolucion = 5.0/1024.0*100
printf('Resolucion teorica: %.4f C/cuenta\n', resolucion)
printf('Ruido observado 3 sigma: %.4f C\n', 3*stdev(temp))
```

Para una medicion de mayor exactitud no debe suponerse que la referencia del ADC vale exactamente 5,000 V. Puede medirse la tension real de referencia y utilizar ese valor en la conversion.

Preguntas.

1. ¿Cuantas cuentas del ADC corresponden aproximadamente a un grado Celsius?
2. ¿Cual es la resolucion teorica del sistema expresada en grados Celsius?
3. Compare el ruido observado con la resolucion calculada. ¿Por que pueden ser diferentes?
4. Caliente suavemente el sensor con los dedos. ¿Cuanto tarda en registrarse el cambio? ¿Que propiedad del sensor determina principalmente ese retardo?
5. Cambie el promedio de $N = 16$ a $N = 64$ lecturas. ¿Como deberia cambiar el ruido si las perturbaciones fueran independientes? Compruebelo experimentalmente.
6. Mida la tension utilizada como referencia por el ADC y recalcule las temperaturas. ¿Que diferencia aparece respecto de suponer exactamente 5 V?

4. **Ejercicio:** Adquisicion multicanal de temperatura y luz

   Nivel B

   Objetivo.

   Adquirir dos magnitudes fisicas mediante un sistema de adquisicion multicanal con ADC multiplexado y estudiar la diferencia temporal entre las mediciones correspondientes a distintos canales.

   Materiales.

   - Plataforma de adquisicion con por lo menos dos entradas analogicas.
   - Sensor LM35 para temperatura.
   - LDR y resistencia de 10 kΩ para formar un divisor resistivo.
   - Cables y protoboard.

Como implementacion de referencia se utilizara el ADC multiplexado de un Arduino Uno.

Adquisicion.

Conectar el LM35 a un canal analogico y el divisor resistivo con la LDR a otro canal.

El sistema debe:

- adquirir periodicamente ambos canales;
- convertir la lectura del LM35 a temperatura;
- expresar la lectura de la LDR como nivel relativo de iluminacion;
- registrar una referencia temporal;
- determinar el intervalo temporal entre las conversiones de ambos canales;
- transmitir o almacenar los resultados.

Puede utilizarse el formato

```
tiempo_ms,temperatura,luz,dt_canales_us
```

donde `dt_canales_us` es una estimacion del intervalo entre el comienzo de las conversiones de los dos canales.

Implementacion de referencia con Arduino.

```
// arduino_multicanal.ino
const int PIN_TEMP = A0;
const int PIN_LUZ  = A1;
const int N_PROM = 16;

void setup() {
    Serial.begin(115200);
}

void loop() {
    long suma_t = 0;
    long suma_l = 0;
    unsigned long suma_dt = 0;

    for (int i = 0; i < N_PROM; i++) {
        unsigned long t_temp = micros();
        suma_t += analogRead(PIN_TEMP);
        unsigned long t_luz = micros();
        suma_l += analogRead(PIN_LUZ);
        suma_dt += t_luz-t_temp;
        delay(1);
    }

    float lectura_t = suma_t/(float)N_PROM;
    float lectura_l = suma_l/(float)N_PROM;
    float temp = lectura_t*5.0/1024.0*100.0;
```

```
    float luz  = lectura_l*100.0/1023.0;
    float dt_canales = suma_dt/(float)N_PROM;

    Serial.print(millis());
    Serial.print(",");
    Serial.print(temp,3);
    Serial.print(",");
    Serial.print(luz,1);
    Serial.print(",");
    Serial.println(dt_canales,1);

    delay(200);
}
```

Analisis de los datos.

Registrar aproximadamente 120 segundos de mediciones. A partir de los datos:

- representar temperatura en funcion del tiempo;
- representar nivel relativo de iluminacion en funcion del tiempo;
- determinar el intervalo medio entre las mediciones de ambos canales;
- calcular la correlacion entre temperatura e iluminacion;
- estudiar si las mediciones de los dos canales pueden considerarse simultaneas para las escalas temporales de los fenomenos observados.

Scilab. Una posible implementacion del registro y analisis mediante Scilab en GNU/Linux es la siguiente.

```
PUERTO   = '/dev/ttyACM0'
BAUD     = 115200
DURACION = 120
ARCHIVO  = '/tmp/multicanal.csv'

unix(sprintf('stty -F %s %d cs8 -cstopb -parenb raw', ..
             PUERTO, BAUD))
unix(sprintf('timeout %d cat %s > %s', ..
             DURACION, PUERTO, ARCHIVO))

lineas = mgetl(ARCHIVO)
n = size(lineas,'*')
datos = zeros(n,4)
validos = 0

for i = 1:n
    p = strsplit(lineas(i),',')
    if size(p,'*') >= 4 then
        tv = strtod(p(1))
        T  = strtod(p(2))
        L  = strtod(p(3))
        dt = strtod(p(4))
```

```
        if ~isnan(tv) & ~isnan(T) & ~isnan(L) & ~isnan(dt)
           then
            validos = validos+1
            datos(validos,1) = tv/1000
            datos(validos,2) = T
            datos(validos,3) = L
            datos(validos,4) = dt
        end
    end
end

datos = datos(1:validos,:)
t    = datos(:,1)'
temp = datos(:,2)'
luz  = datos(:,3)'
dt   = datos(:,4)'

printf('Muestras validas: %d\n', validos)
printf('Separacion media entre canales: %.1f us\n', mean(dt))

r = corr(temp,luz,0)
printf('Correlacion temperatura-luz: %.4f\n', r(1))

fd = mopen('/tmp/multi.txt','w')
for i = 1:validos
    mfprintf(fd,'%.3f %.4f %.2f\n', t(i), temp(i), luz(i))
end
mclose(fd)

NL = ascii(10)
Q  = ascii(34)
DAT = Q+'/tmp/multi.txt'+Q

gp = 'set terminal png size 900,400'+NL
gp = gp+'set output '+Q+'/tmp/multicanal.png'+Q+NL
gp = gp+'set grid'+NL
gp = gp+'set xlabel '+Q+'Tiempo [s]'+Q+NL
gp = gp+'set ylabel '+Q+'Temperatura [C]'+Q+NL
gp = gp+'set y2label '+Q+'Luz [%]'+Q+NL
gp = gp+'set ytics nomirror'+NL
gp = gp+'set y2tics'+NL
gp = gp+'plot '+DAT+ ..
     ' using 1:2 with lines axis x1y1 title '+ ..
     Q+'Temperatura'+Q+', \' + NL
gp = gp+'     '+DAT+ ..
     ' using 1:3 with lines axis x1y2 title '+ ..
     Q+'Luz'+Q+NL

fd = mopen('/tmp/multi.gnu','w')
mfprintf(fd,'%s',gp)
mclose(fd)
```

```
unix('gnuplot /tmp/multi.gnu')
```

El puerto serie puede aparecer como `/dev/ttyACM0`, `/dev/ttyUSB0` u otro nombre, segun la plataforma utilizada.

Preguntas.

1. ¿Las muestras de temperatura y luz son exactamente simultaneas?
2. ¿Que intervalo temporal existe entre las conversiones de los dos canales?
3. Compare ese intervalo con los tiempos caracteristicos de variacion de la temperatura y de la iluminacion. ¿Es importante en esta experiencia?
4. Tape y destape la LDR con la mano. ¿Cuanto tarda en registrarse el cambio? Compare con el tiempo de respuesta del sensor de temperatura.
5. Calcule la correlacion entre temperatura y nivel de iluminacion. ¿Existe correlacion? ¿Tiene significado fisico en la experiencia realizada?
6. La variable denominada "luz" esta expresada entre 0 y 100 %. ¿Puede interpretarse como iluminancia en lux? Justifique.
7. ¿Que modificaciones serian necesarias para medir dos señales cuya variacion fuese tan rapida que la separacion temporal entre canales ya no pudiera despreciarse?

5. **Ejercicio:** Control PID de temperatura

   Nivel B

   Objetivo.

   Implementar un sistema de control realimentado de temperatura, estudiar el comportamiento de los terminos proporcional, integral y derivativo y analizar experimentalmente la respuesta del sistema.

   El controlador debe ejecutarse en el dispositivo que realiza el control, de manera que el periodo de muestreo no dependa de la computadora utilizada para registrar posteriormente los datos.

   Materiales.

   - Plataforma con entrada analogica y salida PWM.
   - Sensor LM35.
   - Resistencia calefactora de potencia adecuada a la fuente utilizada.
   - MOSFET de potencia de nivel logico adecuado a la corriente del calefactor.
   - Resistencia de 100 a 330 Ω en la compuerta del MOSFET.
   - Resistencia de aproximadamente 10 kΩ entre compuerta y masa.
   - Fuente de alimentacion adecuada para el calefactor.

   Como implementacion de referencia se utilizara un Arduino Uno. El calefactor no debe alimentarse directamente desde una salida del microcontrolador.

   Control.

   El sistema debe realizar periodicamente las siguientes operaciones:

1. medir la temperatura;
2. calcular el error $e = r - T$;
3. calcular la accion de control PID;
4. limitar la accion al rango permitido por el actuador;
5. evitar la acumulacion excesiva del termino integral cuando el actuador se encuentra saturado;
6. aplicar la accion de control al calefactor;
7. registrar el tiempo, la temperatura y la accion de control.

Los datos pueden transmitirse utilizando el formato

```
tiempo_ms,temperatura,pwm
```

Como implementacion de referencia se proporciona el siguiente programa para Arduino.

```
// arduino_pid.ino
const int PIN_TEMP = A0;
const int PIN_PWM  = 9;

const float SETPOINT = 35.0;
const float Ts = 0.2;

float Kp = 8.0;
float Ki = 0.3;
float Kd = 1.5;

float integral  = 0.0;
float error_ant = 0.0;
unsigned long ultimo_control = 0;

void setup() {
    Serial.begin(115200);
    pinMode(PIN_PWM, OUTPUT);
    analogWrite(PIN_PWM, 0);
}

float medirTemperatura() {
    long suma = 0;
    for (int i = 0; i < 8; i++) {
        suma += analogRead(PIN_TEMP);
    }
    float lectura = suma/8.0;
    float tension = lectura*5.0/1024.0;
    return tension*100.0;
}

void loop() {
    unsigned long ahora = millis();
```

```
    if (ahora-ultimo_control >= (unsigned long)(Ts*1000.0)) {
        ultimo_control = ahora;

        float T = medirTemperatura();
        float error = SETPOINT-T;
        float integral_nueva = integral + error*Ts;
        float derivada = (error-error_ant)/Ts;

        float u_sin_saturar =
              Kp*error
            + Ki*integral_nueva
            + Kd*derivada;

        float u = constrain(u_sin_saturar,0.0,255.0);

        if (u_sin_saturar >= 0.0 && u_sin_saturar <= 255.0) {
            integral = integral_nueva;
        }

        error_ant = error;
        analogWrite(PIN_PWM,(int)u);

        Serial.print(ahora);
        Serial.print(",");
        Serial.print(T,3);
        Serial.print(",");
        Serial.println(u,1);
    }
}
```

Registro de los datos.

Registrar por lo menos 60 segundos de funcionamiento del sistema. El programa utilizado para el registro debe mantener los datos en un archivo con las columnas t, T y u.

En GNU/Linux, para la implementacion de referencia, puede primero identificarse el puerto mediante

```
ls /dev/ttyACM* /dev/ttyUSB* 2>/dev/null
```

y registrar los datos mediante, por ejemplo,

```
stty -F /dev/ttyACM0 115200 cs8 -cstopb -parenb raw
timeout 60 cat /dev/ttyACM0 > /tmp/pid.csv
```

El nombre del puerto depende del dispositivo utilizado.

Analisis de la respuesta.

A partir de los datos registrados determinar:

- temperatura inicial;

- tiempo necesario para alcanzar el 95 % del cambio requerido;
- sobrealcance;
- error estacionario;
- evolucion de la accion de control.

Representar la temperatura y el setpoint en funcion del tiempo y la accion de control en funcion del tiempo.

Scilab. Una posible implementacion del analisis anterior en Scilab es la siguiente.

```
lineas = mgetl('/tmp/pid.csv')
n = size(lineas,'*')

t_v = []
T_v = []
u_v = []

for i = 1:n
    p = strsplit(lineas(i),',')
    if size(p,'*') >= 3 then
        t = strtod(p(1))/1000
        T = strtod(p(2))
        u = strtod(p(3))
        if ~isnan(t) & ~isnan(T) & ~isnan(u) then
            t_v($+1) = t
            T_v($+1) = T
            u_v($+1) = u
        end
    end
end

if isempty(T_v) then
    error('No se obtuvieron muestras validas')
end

t_v = t_v-t_v(1)
SETPOINT = 35.0

printf('Numero de muestras: %d\n',length(T_v))
printf('Temperatura maxima: %.3f C\n',max(T_v))

sobrealcance = max(T_v)-SETPOINT
printf('Sobrealcance: %.3f C\n',sobrealcance)

N_final = min(20,length(T_v))
error_final = mean(T_v($-N_final+1:$))-SETPOINT
printf('Error estacionario: %.3f C\n',error_final)

T0  = T_v(1)
T95 = T0+0.95*(SETPOINT-T0)
idx = find(T_v >= T95)
```

```
if ~isempty(idx) then
    printf('Tiempo al 95%%: %.2f s\n',t_v(idx(1)))
end
```

Ajuste de los parametros.

Realizar varios experimentos modificando los parametros del controlador:

Experimento	K_p	K_i	K_d	**Observar**
Solo P	8.0	0	0	Error estacionario y velocidad de respuesta
PI	8.0	0.3	0	Reduccion del error estacionario y posible oscilacion
PD	8.0	0	1.5	Efecto del termino derivativo
PID	8.0	0.3	1.5	Compromiso entre rapidez, sobrealcance y estabilidad

Los valores anteriores son solamente puntos de partida y deben ajustarse al sistema termico construido.

Preguntas.

1. Compare las respuestas P, PI, PD y PID. ¿Cual presenta menor tiempo de subida? ¿Cual presenta menor error estacionario?
2. ¿Que ocurre al aumentar demasiado K_p?
3. ¿Que efecto produce el termino integral?
4. Desactive temporalmente el mecanismo anti-windup y compare la respuesta cuando el actuador permanece saturado.
5. ¿Que efecto produce el termino derivativo sobre el ruido de la medicion?
6. Estime la constante de tiempo termica del sistema mediante una respuesta al escalon del calefactor.
7. ¿Por que conviene ejecutar el controlador en el dispositivo de control y utilizar la computadora solamente para registrar y analizar los resultados?
8. ¿Que ocurre si el periodo real del controlador difiere significativamente del valor T_s utilizado en las ecuaciones?

6. **Ejercicio:** ESP32 como servidor de sensores

   Nivel B/C

   Objetivo.

   Construir un sistema de adquisicion capaz de publicar mediciones de sensores a traves de una red WiFi utilizando HTTP y recuperar posteriormente esos datos desde otra computadora.

Integrar conceptualmente este sistema con el servidor web estudiado en el practico sobre interfaces que precede al presente.

Materiales.

- Placa ESP32.
- Sensor DHT22 de temperatura y humedad.
- Sensor BMP280 de presion y temperatura.
- Red WiFi local.
- Computadora conectada a la misma red.

Servidor de adquisicion.

El sistema debera:

1. adquirir temperatura y humedad del DHT22;
2. adquirir presion del BMP280;
3. conectarse a una red WiFi;
4. implementar un servidor HTTP;
5. proporcionar una direccion, por ejemplo `/datos`, que devuelva las mediciones utilizando un formato estructurado independiente del programa cliente.

En este practico se utilizara JSON, con una respuesta del tipo

```
{
  "tiempo_s": 12.5,
  "temperatura": 24.31,
  "humedad": 51.2,
  "presion": 1012.6
}
```

El lenguaje utilizado para implementar el servidor no es parte del objetivo experimental.

Como implementacion de referencia se proporciona el siguiente programa para ESP32 utilizando el entorno Arduino.

```
// esp32_estacion.ino
#include <WiFi.h>
#include <WebServer.h>
#include <DHT.h>
#include <Wire.h>
#include <Adafruit_BMP280.h>

const char* SSID     = "nombre_de_tu_red";
const char* PASSWORD = "contrasena";

DHT dht(4,DHT22);
Adafruit_BMP280 bmp;
```

```
WebServer server(80);

void handleDatos() {
    float T = dht.readTemperature();
    float H = dht.readHumidity();
    float P = bmp.readPressure()/100.0;
    unsigned long ms = millis();

    String json = "{";
    json += "\"tiempo_s\":" + String(ms/1000.0,2) + ",";
    json += "\"temperatura\":" + String(T,3) + ",";
    json += "\"humedad\":" + String(H,1) + ",";
    json += "\"presion\":" + String(P,2);
    json += "}";

    server.send(200,"application/json",json);
}

void handleRaiz() {
    server.send(200,"text/html",
        "<h2>Estacion ESP32</h2>"
        "<p>GET /datos para obtener JSON</p>");
}

void setup() {
    dht.begin();
    bmp.begin(0x76);
    WiFi.begin(SSID,PASSWORD);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
    }

    server.on("/",handleRaiz);
    server.on("/datos",handleDatos);
    server.begin();
}

void loop() {
    server.handleClient();
}
```

Determinar la direccion IP asignada al ESP32 y verificar desde otra computadora que el servidor responde. Por ejemplo:

```
curl http://DIRECCION_IP/datos
```

Adquisicion remota.

Realizar 120 consultas al servidor separadas aproximadamente cinco segundos. Para cada consulta registrar t, T, H y P y guardar las mediciones en un archivo CSV.

El programa utilizado para realizar las consultas y almacenar los datos puede imple-

mentarse con cualquier herramienta capaz de realizar una peticion HTTP y procesar la respuesta JSON.

Scilab. Una posible implementacion del cliente de adquisicion en Scilab es la siguiente.

```
IP = '192.168.1.X'
N = 120
archivo = '/tmp/esp32.csv'
fd = mopen(archivo,'w')
mfprintf(fd, '# tiempo_s,temperatura,humedad,presion\n')

function v = extraer(json,clave)
    idx = strindex(json,clave+':')
    if isempty(idx) then
        v = %nan
        return
    end

    sub = part(json,idx(1)+length(clave)+1:$)
    idx2 = strindex(sub,',')
    idx3 = strindex(sub,'}')

    if isempty(idx2) then
        fin = idx3(1)-1
    else
        fin = min(idx2(1),idx3(1))-1
    end

    v = strtod(part(sub,1:fin))
endfunction

for i = 1:N
    tmp = '/tmp/esp32_dato.json'
    unix(sprintf('curl -s --max-time 2 http://%s/datos >
       %s', ..
                 IP,tmp))

    linea = mgetl(tmp)
    if ~isempty(linea) then
        linea = linea(1)
        t_s = extraer(linea,'"tiempo_s"')
        T   = extraer(linea,'"temperatura"')
        H   = extraer(linea,'"humedad"')
        P   = extraer(linea,'"presion"')

        mfprintf(fd,'%.2f,%.3f,%.1f,%.2f\n', t_s,T,H,P)
        printf('t=%6.0f s T=%.2f C H=%.1f%% P=%.2f hPa\n', ..
               t_s,T,H,P)
    end

    sleep(5000)
```

```
end

mclose(fd)
```

Analisis.

Representar en funcion del tiempo temperatura, humedad relativa y presion atmosferica. Calcular ademas la correlacion entre temperatura y humedad.

Latencia y frecuencia util.

Medir el tiempo transcurrido entre una peticion HTTP y la recepcion de la respuesta. En GNU/Linux puede utilizarse, por ejemplo,

```
time curl -s http://DIRECCION_IP/datos
```

Comparar esta latencia con el tiempo propio de actualizacion de los sensores.

Preguntas.

1. ¿Que informacion intercambian el servidor y el cliente? ¿Depende esa informacion del lenguaje con que fueron programados?
2. ¿Que funcion cumple HTTP en este sistema?
3. ¿Que funcion cumple JSON?
4. ¿Cual es la latencia aproximada de una consulta?
5. Compare la latencia de la comunicacion con el tiempo de actualizacion de los sensores. ¿Que elemento limita realmente la frecuencia util de adquisicion?
6. Grafique las tres magnitudes medidas. ¿Existe correlacion entre temperatura y humedad?
7. Acceda simultaneamente al servidor desde dos dispositivos. ¿Continua respondiendo correctamente?
8. Proponga como integrar estas mediciones con el servidor web del practico de interfaces para permitir elegir que magnitud representar y durante cuanto tiempo.

7. **Ejercicio:** Sensores digitales con microcontrolador

   Nivel C

   Objetivo.

   Comparar sensores digitales que integran la conversion y entregan sus mediciones mediante protocolos de comunicacion. Identificar que tareas se realizan dentro del sensor y cuales quedan a cargo del microcontrolador y de la computadora.

   Equipo.

   Segun la disponibilidad del laboratorio, utilizar uno o mas de los siguientes sensores:

   - BH1750 o VEML7700 para iluminancia;
   - SHT31, BMP280 o BME280 para variables ambientales;

- DS18B20 para temperatura.

Como plataforma de referencia puede utilizarse un ESP32 o un Arduino.

Procedimiento.

Para cada sensor utilizado:

1. identificar las magnitudes que mide y sus unidades;
2. identificar el protocolo de comunicacion utilizado;
3. determinar alimentacion y niveles logicos admitidos;
4. identificar direccion o identificador cuando corresponda;
5. realizar una serie temporal de mediciones;
6. registrar los datos en un formato independiente del lenguaje, preferentemente CSV;
7. representar las magnitudes obtenidas en funcion del tiempo.

Cuando se utilicen varios sensores sobre el mismo bus, comprobar si pueden compartir las lineas de comunicacion y explicar como se los distingue.

Comparacion de arquitecturas.

Comparar, cuando corresponda,

$$\text{LM35} \rightarrow \text{ADC del microcontrolador}$$

con

$$\text{sensor digital} \rightarrow \text{I2C o 1-Wire} \rightarrow \text{microcontrolador.}$$

Identificar en cada caso donde se realiza la conversion A/D y donde se aplica la calibracion proporcionada por el fabricante.

Preguntas.

1. ¿Que ventaja presenta un sensor digital integrado frente a uno analogico cuando la distancia entre sensor y controlador aumenta?
2. ¿Que limitaciones introduce el protocolo de comunicacion?
3. ¿Que diferencias existen entre I2C y 1-Wire en numero de lineas, direccionamiento y posibilidad de conectar varios sensores?
4. ¿Una salida numerica implica necesariamente mayor exactitud?
5. ¿Que parte del procedimiento experimental es independiente del lenguaje utilizado para leer el sensor?

8. **Ejercicio:** Medicion de la velocidad del sonido

   Nivel A/B

   Objetivo.

Determinar experimentalmente la velocidad del sonido en el aire mediante una medicion de tiempo de vuelo.

Se proponen dos metodos que utilizan el mismo principio fisico:

$$v = \frac{2d}{\Delta t},$$

donde d es la distancia entre el emisor y el reflector y Δt es el tiempo de ida y vuelta de la onda.

Opcion A — Impulso sonoro y microfono.

Ubicarse frente a una pared plana a una distancia conocida d. Generar un impulso breve, por ejemplo mediante un aplauso, y registrar el sonido y su eco. La frecuencia de muestreo debe ser suficientemente alta para resolver el retardo temporal.

En GNU/Linux puede utilizarse, por ejemplo,

```
arecord -d 1 -c 1 -r 44100 -f S16_LE -t wav /tmp/eco.wav
```

A partir del archivo registrado:

1. identificar el instante correspondiente al impulso inicial;
2. identificar el instante correspondiente al eco;
3. calcular $\Delta t = t_{\text{eco}} - t_{\text{impulso}}$;
4. determinar $v = 2d/\Delta t$.

El procedimiento de identificacion de los eventos puede realizarse mediante inspeccion de la señal, deteccion por umbral, correlacion u otro metodo apropiado.

Scilab. Una posible implementacion mediante deteccion por umbral en Scilab es la siguiente.

```
[x,fs] = wavread('/tmp/eco.wav')

if size(x,1) > 1 then
    senal = x(1,:)
else
    senal = x
end

t = (0:length(senal)-1)/fs
umbral = 0.3*max(abs(senal))
idx_eventos = find(abs(senal) > umbral)

if length(idx_eventos) >= 2 then
    t_impulso = t(idx_eventos(1))
    gap = find(diff(idx_eventos) > 0.002*fs)

    if ~isempty(gap) then
        t_eco = t(idx_eventos(gap(1)+1))
        dt = t_eco-t_impulso
```

```
        d = input('Distancia a la pared [m]: ')
        v = 2*d/dt
        printf('Retardo: %.6f s\n',dt)
        printf('Velocidad del sonido: %.1f m/s\n',v)
    end
end
```

Opcion B — Sensor ultrasonico.

Puede realizarse la misma experiencia utilizando un sensor HC-SR04. El sensor genera un pulso ultrasonico y proporciona una señal cuyo ancho temporal corresponde aproximadamente al tiempo transcurrido hasta recibir el eco.

Para medir la velocidad del sonido no debe utilizarse como dato la distancia calculada suponiendo previamente un valor de la velocidad. Debe registrarse directamente el tiempo del eco.

Como implementacion de referencia:

```
// arduino_hcsr04.ino
const int PIN_TRIG = 7;
const int PIN_ECHO = 8;

void setup() {
    Serial.begin(9600);
    pinMode(PIN_TRIG,OUTPUT);
    pinMode(PIN_ECHO,INPUT);
}

void loop() {
    digitalWrite(PIN_TRIG,LOW);
    delayMicroseconds(2);
    digitalWrite(PIN_TRIG,HIGH);
    delayMicroseconds(10);
    digitalWrite(PIN_TRIG,LOW);

    unsigned long duracion = pulseIn(PIN_ECHO,HIGH,30000);

    Serial.print(millis());
    Serial.print(",");
    Serial.println(duracion);

    delay(100);
}
```

Si el reflector se encuentra a una distancia conocida d y el instrumento entrega un tiempo Δt en microsegundos,

$$v = \frac{2d}{\Delta t \, 10^{-6}}.$$

Mediciones.

Repetir la experiencia para varias distancias, por ejemplo

$$1, \qquad 1{,}5, \qquad 2, \qquad 2{,}5 \text{ m}.$$

Para cada distancia realizar varias mediciones del retardo y calcular su valor medio y dispersion. Representar Δt en funcion de d.

Si el modelo es

$$\Delta t = \frac{2}{v} d + b,$$

la pendiente m del ajuste lineal permite obtener $v = 2/m$. El termino independiente b permite ademas detectar retardos sistematicos aproximadamente constantes del sistema de medicion.

Scilab. Una vez obtenidos experimentalmente los vectores de distancias y tiempos, el ajuste puede realizarse en Scilab. Por ejemplo:

```
// reemplazar por los valores experimentales
d  = [1.0 1.5 2.0 2.5]
dt = [0.0059 0.0088 0.0117 0.0146]

A = [d' ones(length(d),1)]
coef = A\dt'

m = coef(1)
b = coef(2)
v = 2/m

printf('Pendiente: %.6f s/m\n',m)
printf('Retardo constante: %.6f s\n',b)
printf('Velocidad: %.2f m/s\n',v)
```

Preguntas.

1. Compare el resultado con $v \simeq 331{,}5 + 0{,}607\,T$, con v en m/s y T en grados Celsius.
2. Mida la temperatura ambiente. ¿La correccion por temperatura es significativa respecto de la incertidumbre experimental?
3. ¿Cuales son las principales fuentes de error de la medicion?
4. ¿Que incertidumbre introduce un error δd en la medicion de la distancia?
5. ¿Que incertidumbre temporal minima introduce la frecuencia de muestreo cuando se utiliza el metodo acustico?
6. ¿Por que es preferible determinar v a partir de varias distancias y un ajuste lineal en lugar de una unica medicion?
7. ¿Que interpretacion fisica puede tener un termino independiente b distinto de cero en el ajuste?

9. **Ejercicio:** Raspberry Pi como centro de adquisicion y servidor

   Nivel C

   Objetivo.

   Construir un sistema distribuido de adquisicion en el que un dispositivo realiza el muestreo de los sensores y una Raspberry Pi recibe los datos, los almacena y los publica mediante un servidor accesible desde la red.

   La arquitectura general sera

   $$\text{sensores} \longrightarrow \text{unidad de adquisicion} \longrightarrow \text{Raspberry Pi} \longrightarrow \begin{matrix}\text{almacenamiento} \\ \text{servidor HTTP}\end{matrix}$$

   Como implementacion de referencia se utilizara un Arduino como unidad de adquisicion.

   La Raspberry Pi no debe determinar los instantes criticos de muestreo. La unidad de adquisicion mide los sensores y transmite posteriormente los resultados.

   Parte A — Comunicacion con la unidad de adquisicion.

   Conectar el Arduino por USB a la Raspberry Pi e identificar el puerto serie disponible. En GNU/Linux puede utilizarse:

   ```
   ls /dev/ttyACM* /dev/ttyUSB* 2>/dev/null
   ```

   La unidad de adquisicion debe transmitir una linea por medicion, por ejemplo:

   ```
   tiempo_ms,temperatura
   1000,23.45
   2000,23.51
   3000,23.48
   ```

   Parte B — Recepcion y almacenamiento.

   Implementar en la Raspberry Pi un programa que:

   1. abra el puerto serie;
   2. reciba las mediciones;
   3. las muestre por pantalla;
   4. las almacene inmediatamente en un archivo CSV;
   5. mantenga abierto el puerto durante toda la adquisicion.

   El lenguaje utilizado para implementar estas tareas no forma parte del objetivo experimental.

   Python. Como implementacion de referencia se utilizara Python.

   Instalar:

   ```
   sudo apt install python3 python3-serial
   ```

Crear:

```
# adquirir.py
import serial
import time

puerto = "/dev/ttyACM0"
baud = 115200

s = serial.Serial(puerto, baud, timeout=2)

# algunas placas se reinician al abrir el puerto
time.sleep(2)
s.reset_input_buffer()

with open("datos.csv","a") as f:
    while True:
        linea = s.readline().decode(errors="ignore").strip()
        if linea:
            print(linea)
            f.write(linea+"\n")
            f.flush()
```

Verificar que las mediciones se almacenan correctamente.

Parte C — Servidor web.

Implementar un servidor HTTP que permita consultar desde otro dispositivo las ultimas mediciones almacenadas.

Python. Como implementacion de referencia puede utilizarse el servidor incluido en la biblioteca estandar de Python.

```
# servidor.py
from http.server import BaseHTTPRequestHandler, HTTPServer

class Handler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:
            with open("datos.csv","r") as f:
                datos = f.readlines()

            ultimas = datos[-20:]

            cuerpo = """
            <html>
            <head><title>Laboratorio</title></head>
            <body>
            <h1>Datos adquiridos</h1>
            <pre>
            """ + "".join(ultimas) + """
            </pre>
            </body>
```

```
                </html>
                """

                contenido = cuerpo.encode()
                self.send_response(200)
                self.send_header("Content-Type", "text/html;
                    charset=utf-8")
                self.send_header("Content-Length",
                    str(len(contenido)))
                self.end_headers()
                self.wfile.write(contenido)

            except Exception as e:
                self.send_error(500,str(e))

HTTPServer(("0.0.0.0",8000), Handler).serve_forever()
```

Ejecutar:

```
python3 servidor.py
```

Determinar la direccion IP de la Raspberry Pi:

```
hostname -I
```

Desde otra computadora o telefono conectado a la misma red acceder a

`http://DIRECCION_IP:8000`

y comprobar que aparecen las mediciones.

Parte D — Analisis de los datos.

El archivo CSV debe poder analizarse independientemente del programa utilizado para adquirir o publicar los datos.

Representar la temperatura en funcion del tiempo y determinar:

- numero de mediciones;
- duracion total del registro;
- periodo medio de muestreo;
- temperatura media;
- valor minimo y maximo.

Scilab. Si el archivo contiene dos columnas numericas `tiempo_ms,temperatura`, una posible lectura en Scilab es la siguiente.

```
d = csvRead("datos.csv")

t = d(:,1)/1000
T = d(:,2)
```

```
t = t-t(1)

printf('Muestras: %d\n',length(T))
printf('Duracion: %.1f s\n',t($))
printf('Temperatura media: %.3f C\n',mean(T))
printf('Minimo: %.3f C\n',min(T))
printf('Maximo: %.3f C\n',max(T))

plot(t,T)
xlabel("Tiempo [s]")
ylabel("Temperatura [C]")
```

Parte E — Comportamiento distribuido.

Mantener simultaneamente en funcionamiento:

- la unidad de adquisicion;
- el programa que recibe y almacena los datos;
- el servidor HTTP;
- uno o mas clientes conectados por red.

Observar que ocurre cuando cada uno de esos elementos es detenido temporalmente.

Preguntas.

1. ¿Que tareas realiza la unidad de adquisicion y cuales realiza la Raspberry Pi?
2. ¿Por que conviene que el instante de muestreo sea determinado por la unidad de adquisicion y no por el servidor web?
3. Desconecte temporalmente el Arduino. ¿Que ocurre con el programa de adquisicion? ¿Que ocurre con el servidor?
4. Acceda simultaneamente al servidor desde una computadora y un telefono. ¿La adquisicion continua normalmente?
5. Determine aproximadamente cuanto espacio de almacenamiento ocuparia un dia de mediciones con el periodo utilizado.
6. Modifique el servidor para que muestre separadamente la ultima temperatura recibida.
7. Implemente una alerta: si la temperatura supera un determinado umbral, la pagina debe presentar una advertencia.
8. Utilice `top` u otra herramienta equivalente para observar el uso de CPU durante la adquisicion y durante las consultas HTTP. ¿Que proceso utiliza mas recursos?
9. Proponga como agregar otros sensores sin modificar la arquitectura general del sistema.
10. ¿Que partes de esta arquitectura dependen del protocolo entre los dispositivos y cuales dependen solamente del programa elegido para implementarlas?

10. **Ejercicio:** Instrumentacion USB con TEMPer2

    Nivel A

    Objetivo.

    Utilizar un termometro USB TEMPer2 como instrumento digital de adquisicion, identificar sus sensores interno y externo, estudiar su repetibilidad y respuesta temporal y registrar automaticamente las mediciones para su posterior analisis.

    En este practico el programa utilizado para comunicarse con el instrumento actua como controlador o *driver* del dispositivo. El tratamiento y analisis posterior de los datos puede realizarse con cualquier herramienta apropiada.

    Equipo.

    - Termometro USB PCsensor TEMPer2.
    - Sonda externa.
    - Computadora con GNU/Linux.
    - Termometro de referencia.

    La unidad disponible en el laboratorio se identifica como

    ```
    3553:a001 PCsensor TEMPer2
    ```

    y utiliza el firmware `TEMPer2_V4.1`.

    Preparacion.

    Para esta unidad se proporciona una version del programa `temper.py` adaptada al firmware disponible en el laboratorio.

    Puede obtenerse directamente mediante

    ```
    wget https://ututo.org/downloads/temper_vdsa.py
    chmod a+x temper_vdsa.py
    ```

    Alternativamente puede descargarse el repositorio original para examinar su implementacion:

    ```
    git clone https://github.com/greg-kodama/temper.git
    ```

    Instalar las dependencias necesarias para el programa utilizado. Para la implementacion proporcionada:

    ```
    sudo apt install python3 python3-serial
    ```

    Conectar el instrumento y verificar su identificacion:

    ```
    lsusb | grep -i temper
    ```

    Debe aparecer una linea que contenga

    ```
    ID 3553:a001 PCsensor TEMPer2
    ```

Realizar una primera lectura:

```
sudo ./temper_vdsa.py --force 3553:a001
```

Con la sonda externa conectada deben obtenerse dos temperaturas.

Si aparece

```
Cannot read device firmware identifier
```

desconectar el TEMPer2, esperar aproximadamente cinco segundos, volver a conectarlo y repetir la lectura.

Experiencia 1 — Identificacion de los sensores.

Mantener inicialmente ambos sensores a temperatura ambiente y registrar sus indicaciones. Calentar suavemente solamente la sonda externa y observar cual de las dos temperaturas cambia. Repetir calentando el cuerpo del TEMPer2 sin modificar deliberadamente la temperatura de la sonda externa.

Identificar de esta forma:

- sensor interno;
- sensor externo.

Experiencia 2 — Comparacion y repetibilidad.

Colocar ambos sensores junto a un termometro de referencia y esperar hasta alcanzar aproximadamente el equilibrio termico. Realizar por lo menos 20 lecturas.

Para cada sensor determinar:

- valor medio;
- valor maximo;
- valor minimo;
- desviacion estandar;
- diferencia respecto del termometro de referencia.

Comparar tambien las indicaciones de ambos sensores entre si. Discutir las diferencias entre resolucion, repetibilidad y exactitud.

Experiencia 3 — Respuesta temporal.

Colocar la sonda externa en un medio cuya temperatura sea claramente diferente de la temperatura ambiente. Puede utilizarse, por ejemplo, agua templada.

Registrar la temperatura durante la aproximacion al equilibrio. Retirar luego la sonda y registrar su retorno hacia la temperatura ambiente. Representar $T(t)$.

Si el comportamiento puede aproximarse mediante un sistema de primer orden,

$$T(t) = T_a + (T_0 - T_a)e^{-t/\tau},$$

determinar experimentalmente la constante de tiempo τ. Comparar los tiempos caracteristicos del sensor interno y de la sonda externa.

Experiencia 4 — Adquisicion automatica.

El programa proporcionado puede entregar las mediciones en formato JSON:

```
sudo ./temper_vdsa.py --force 3553:a001 --json
```

La salida contiene, entre otros, los campos

```
internal temperature
external temperature
```

El objetivo es construir un registro con la estructura

```
tiempo,temperatura_interna,temperatura_externa
```

donde el tiempo sea numerico para facilitar su procesamiento posterior. El lenguaje utilizado para extraer los campos JSON y generar el archivo no forma parte del objetivo experimental.

Python. Como implementacion de referencia puede utilizarse Python.

```
# log_temper.py
import sys
import json
import time

d = json.load(sys.stdin)[0]

temp_interna = d["internal temperature"]
temp_externa = d["external temperature"]

print(
    f"{time.time():.3f},"
    f"{temp_interna},"
    f"{temp_externa}"
)
```

Realizar una adquisicion cada tres segundos:

```
while true; do
    sudo ./temper_vdsa.py --force 3553:a001 --json \
      | python3 log_temper.py >> temperaturas.csv
    sleep 3
done
```

Detener mediante `Ctrl+C`. El archivo resultante debe contener exclusivamente datos de la forma

```
tiempo,temperatura_interna,temperatura_externa
```

El programa utilizado para comunicarse especificamente con el TEMPer2 disponible en el laboratorio es:

Listing 9.18: Programa de comunicacion con el TEMPer2 V4.1

```
#!/usr/bin/env python3
# temper.py -*-python-*-
# Copyright 2018 by Pham Urwen (urwen@mail.ru)
#
# Permission is hereby granted, free of charge, to any
   person obtaining a copy
# of this software and associated documentation files (the
   "Software"), to
# deal in the Software without restriction, including
   without limitation the
# rights to use, copy, modify, merge, publish, distribute,
   sublicense, and/or
# sell copies of the Software, and to permit persons to whom
   the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice
   shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
   KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
   MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN
   NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
   DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
   OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
   OR OTHER DEALINGS
# IN THE SOFTWARE.

# Standard python3 modules
import argparse
import binascii
import json
import os
import re
import select
import struct
import sys

# Non-standard modules
try:
  import serial
```

```
except ImportError:
  print('Cannot import "serial". Please sudo apt-get install
     python3-serial')
  sys.exit(1)
# Standard, but linux-specific
try:
  import fcntl
except ImportError:
  fcntl = None


# "Constants"
# Equivalent of the _IO('U', 20) constant in the linux
   kernel.
USBDEVFS_RESET = ord('U') << (4*2) | 20
USE_RESET = True


class USBList(object):
  '''Get a list of all of the USB devices on a system, along
     with their
  associated hidraw or serial (tty) devices.
  '''

  SYSPATH = '/sys/bus/usb/devices'

  def _readfile(self, path):
    '''Read data from 'path' and return it as a string.
       Return the empty string
    if the file does not exist, cannot be read, or has an
       error.
    '''
    try:
      with open(path, 'r') as fp:
        return fp.read().strip()
    except:
      return ''

  def _find_devices(self, dirname):
    '''Scan a directory hierarchy for names that start with
       "tty" or "hidraw".
    Return these names in a set.
    '''
    devices = set()
    for entry in os.scandir(dirname):
        if entry.is_dir() and not entry.is_symlink():
          devices |=
             self._find_devices(os.path.join(dirname,
             entry.name))
        if re.search('tty.*[0-9]', entry.name):
          devices.add(entry.name)
```

```
          if re.search('hidraw[0-9]', entry.name):
            devices.add(entry.name)
      return devices

    def _get_usb_device(self, dirname):
      '''Examine the files in 'dirname', looking for files
         with well-known
      names expected to be in the /sys hierarchy under Linux
         for USB devices.
      Return a dictionary of the information gathered. If no
         information is found
      (i.e., because the directory is not for a USB device)
         return None.
      '''
      info = dict()
      vendorid = self._readfile(os.path.join(dirname,
         'idVendor'))
      if vendorid == '':
        return None
      info['vendorid'] = int(vendorid, 16)
      productid = self._readfile(os.path.join(dirname,
         'idProduct'))
      info['productid'] = int(productid, 16)
      info['manufacturer'] =
         self._readfile(os.path.join(dirname,
                                                    'manufacturer'))
      info['product'] = self._readfile(os.path.join(dirname,
         'product'))
      info['busnum'] =
         int(self._readfile(os.path.join(dirname, 'busnum')))
      info['devnum'] =
         int(self._readfile(os.path.join(dirname, 'devnum')))
      info['devices'] = sorted(self._find_devices(dirname))
      return info

    def get_usb_devices(self):
      '''Scan a well-known Linux hierarchy in /sys and try to
         find all of the
      USB devices on a system. Return these as a dictionary
         indexed by the path.
      '''
      info = dict()
      for entry in os.scandir(Temper.SYSPATH):
          if entry.is_dir():
            path = os.path.join(Temper.SYSPATH, entry.name)
            device = self._get_usb_device(path)
            if device is not None:
              device['port'] = entry.name
              info[path] = device
      return info
```

```
class USBRead(object):
  '''Read temperature and/or humidity information from a
     specified USB device.
  '''
  def __init__(self, device, verbose=False):
    self.device = device
    self.verbose = verbose

  def _parse_bytes(self, name, offset, divisor, bytes, info,
     verbose = False):
    '''Data is returned from several devices in a similar
       format. In the first
    8 bytes, the internal sensors are returned in bytes 2
       and 3 (temperature)
    and in bytes 4 and 5 (humidity). In the second 8 bytes,
       external sensor
    information is returned. If there are only external
       sensors, then only 8
    bytes are returned, and the caller is expected to use
       the correct 'name'.
    The caller is also expected to detect the firmware
       version and provide the
    appropriate divisor, which is usually 100 or 256.

    There is no return value. Instead 'info[name]' is update
       directly, if a
    value is found.
    '''
    try:
      if bytes[offset] == 0x4e and bytes[offset+1] == 0x20:
        return
    except:
      return
    try:
    # Big endian, short (signed) integer (2 Bytes)
      if verbose:
        print('Converted value: %s' %
           binascii.hexlify(bytes[offset:offset+2]))
      info[name] = struct.unpack_from('>h', bytes,
         offset)[0] / divisor
    except:
      return

  def _read_hidraw_firmware(self, fd, verbose = False):
    ''' Get firmware identifier'''
    query = struct.pack('8B', 0x01, 0x86, 0xff, 0x01, 0, 0,
       0, 0)
    if verbose:
      print('Firmware query: %s' % binascii.b2a_hex(query))
```

```
      # Sometimes we don't get all of the expected information
         from the
      # device.  We'll retry a few times and hope for the best.
      # See: https://github.com/urwen/temper/issues/9
      for i in range(0, 10):
        os.write(fd, query)

        firmware = b''
        while True:
          r, _, _ = select.select([fd], [], [], 0.2)
          if fd not in r:
            break
          data = os.read(fd, 8)
          firmware += data

        if not len(firmware):
          os.close(fd)
          raise RuntimeError('Cannot read device firmware
             identifier')

        if len(firmware) > 8:
          break

      if verbose:
        print('Firmware value: %s %s'
           %(binascii.b2a_hex(firmware), firmware.decode()))

      return firmware

    def _read_hidraw(self, device):
      '''Using the Linux hidraw device, send the special
         commands and receive the
      raw data. Then call '_parse_bytes' based on the firmware
         version to provide
      temperature and humidity information.

      A dictionary of temperature and humidity info is
         returned.
      '''
      path = os.path.join('/dev', device)
      fd = os.open(path, os.O_RDWR)

      firmware = self._read_hidraw_firmware(fd, self.verbose)
      #print(firmware[:12])

      # Get temperature/humidity
      os.write(fd, struct.pack('8B', 0x01, 0x80, 0x33, 0x01,
         0, 0, 0, 0))
      bytes = b''
      while True:
        r, _, _ = select.select([fd], [], [], 0.1)
```

```
        if fd not in r:
          break
        data = os.read(fd, 8)
        bytes += data

      os.close(fd)
      if self.verbose:
        print('Data value: %s' % binascii.hexlify(bytes))

      info = dict()
      info['firmware'] = str(firmware, 'latin-1').strip()
      info['hex_firmware'] = str(binascii.b2a_hex(firmware),
         'latin-1')
      info['hex_data'] = str(binascii.b2a_hex(bytes),
         'latin-1')

      if info['firmware'][:10] in [ 'TEMPerF1.4', 'TEMPer1F1.'
         ]:
        info['firmware'] = info['firmware'][:10]
        self._parse_bytes('internal temperature', 2, 256.0,
           bytes, info)
        return info

      if info['firmware'][:15] in [ 'TEMPerGold_V3.1',
         'TEMPerGold_V3.4' ]:
        info['firmware'] = info['firmware'][:15]
        self._parse_bytes('internal temperature', 2, 100.0,
           bytes, info)
        return info

      if info['firmware'][:12] in [ 'TEMPerX_V3.1',
         'TEMPerX_V3.3' ]:
        info['firmware'] = info['firmware'][:12]
        self._parse_bytes('internal temperature', 2, 100.0,
           bytes, info)
        self._parse_bytes('internal humidity', 4, 100.0,
           bytes, info)
        self._parse_bytes('external temperature', 10, 100.0,
           bytes, info)
        self._parse_bytes('external humidity', 12, 100.0,
           bytes, info)
        return info

      if info['firmware'][:16] == 'TEMPer2_M12_V1.3':
        info['firmware'] = info['firmware'][:16]
        self._parse_bytes('internal temperature', 2, 256.0,
           bytes, info)
        self._parse_bytes('external temperature', 4, 256.0,
           bytes, info)
        return info
      if info['firmware'][:12] in [ 'TEMPer2_V3.7',
```

```
          'TEMPer2_V3.9']:
        info['firmware'] = info['firmware'][:12]
        #Bytes 3-4 hold the device temp, divide by 100
        self._parse_bytes('internal temperature', 2, 100.0,
            bytes, info, self.verbose)
        #Bytes 11-12 hold the external temp, divide by 100
        self._parse_bytes('external temperature', 10, 100.0,
            bytes, info, self.verbose)
        return info
      if info['firmware'].startswith('TEMPer2_V4.1'):
        info['firmware'] = 'TEMPer2_V4.1'
        #Bytes 3-4 hold the internal temperature, divide by 100
        self._parse_bytes('internal temperature', 2, 100.0,
            bytes, info, self.verbose)
        #Bytes 11-12 hold the external temperature, divide by
            100
        self._parse_bytes('external temperature', 10, 100.0,
            bytes, info, self.verbose)
        return info
      if info['firmware'][:16] == 'TEMPerHUM_V3.9':
        info['firmware'] = info['firmware'][:16]
        #Bytes 3-4 hold the device temp, divide by 100
        self._parse_bytes('internal temperature', 2, 100.0,
            bytes, info, self.verbose)
        #Bytes 11-12 hold the external temp, divide by 100
        self._parse_bytes('external temperature', 10, 100.0,
            bytes, info, self.verbose)
        #Bytes 5-6 hold the device humidity, divide by 100
        self._parse_bytes('internal humidity', 4, 100.0,
            bytes, info)
        return info
      info['error'] = 'Unknown firmware %s: %s' %
          (info['firmware'],
                                                  binascii.hexlify(bytes)
      return info

    def _read_serial(self, device):
      '''Using the Linux serial device, send the special
          commands and receive the
      text data, which is parsed directly in this method.

      A dictionary of device info (like that returned by
          USBList) combined with
      temperature and humidity info is returned.
      '''

      path = os.path.join('/dev', device)
      s = serial.Serial(path, 9600)
      s.bytesize = serial.EIGHTBITS
      s.parity = serial.PARITY_NONE
      s.stopbits = serial.STOPBITS_ONE
```

```
      s.timeout = 1
      s.xonoff = False
      s.rtscts = False
      s.dsrdtr = False
      s.writeTimeout = 0

      # Send the "Version" command and save the reply.
      s.write(b'Version')
      firmware = str(s.readline(), 'latin-1').strip()

      # Send the "ReadTemp" command and save the reply.
      s.write(b'ReadTemp')
      reply = str(s.readline(), 'latin-1').strip()
      reply += str(s.readline(), 'latin-1').strip()
      s.close()

      info = dict()
      info['firmware'] = firmware
      m = re.search(r'Temp-Inner:([0-9.]*).*, ?([0-9.]*)',
         reply)
      if m is not None:
        info['internal temperature'] = float(m.group(1))
        info['internal humidity'] = float(m.group(2))
      m = re.search(r'Temp-Outer:([0-9.]*)', reply)
      if m is not None:
        try:
          info['external temperature'] = float(m.group(1))
        except:
          pass
      return info

    def read(self):
      '''Read the firmware version, temperature, and humidity
         from the device and
      return a dictionary containing these data.
      '''
      # Use the last device found
      if self.device.startswith('hidraw'):
        return self._read_hidraw(self.device)
      if self.device.startswith('tty'):
        return self._read_serial(self.device)
      return {'error': 'No usable hid/tty devices available'}


  class Temper(object):
    SYSPATH = '/sys/bus/usb/devices'

    def __init__(self, verbose=False):
      usblist = USBList()
      self.usb_devices = usblist.get_usb_devices()
      self.forced_vendor_id = None
```

```
    self.forced_product_id = None
    self.verbose = verbose

  def _is_known_id(self, vendorid, productid):
    '''Returns True if the vendorid and product id are valid.
    '''
    if self.forced_vendor_id is not None and \
       self.forced_product_id is not None:
      if self.forced_vendor_id == vendorid and \
         self.forced_product_id == productid:
        return True
      return False

    if vendorid == 0x0c45 and productid == 0x7401:
      return True
    if vendorid == 0x413d and productid == 0x2107:
      return True
    if vendorid == 0x1a86 and productid == 0x5523:
      return True
    if vendorid == 0x1a86 and productid == 0xe025:
      return True

    # The id is not known to this program.
    return False

  def list(self, use_json=False):
    '''Print out a list all of the USB devices on the
       system. If 'use_json' is
    True, then JSON formatting will be used.
    '''
    if use_json:
      print(json.dumps(self.usb_devices, indent=4))
      return

    for _, info in sorted(self.usb_devices.items(),
                          key=lambda x: x[1]['busnum'] *
                             1000 + \
                          x[1]['devnum']):
      print('Bus %03d Dev %03d %04x:%04x %s %s %s' % (
        info['busnum'],
        info['devnum'],
        info['vendorid'],
        info['productid'],
        '*' if self._is_known_id(info['vendorid'],
           info['productid']) else ' ',
        info.get('product', '???'),
        list(info['devices']) if len(info['devices']) > 0
           else ''))

  def read(self, verbose=False):
    '''Read all of the known devices on the system and
```

```
        return a list of
      dictionaries which contain the device information,
        firmware information,
      and environmental information obtained. If there is an
        error, then the
      'error' field in the dictionary will contain a string
        explaining the
      error.
      '''
      results = []
      for _, info in sorted(self.usb_devices.items(),
                            key=lambda x: x[1]['busnum'] *
                               1000 + \
                            x[1]['devnum']):
        if not self._is_known_id(info['vendorid'],
           info['productid']):
          continue
        if len(info['devices']) == 0:
          info['error'] = 'no hid/tty devices available'
          results.append(info)
          continue
        usbread = USBRead(info['devices'][-1], verbose)
        results.append({ **info, **usbread.read() })
        if USE_RESET:
            send_reset(info)
      return results

    def _add_temperature(self, name, info):
      '''Helper method to add the temperature to a string in
        both Celsius and
      Fahrenheit. If no sensor data is available, then '- -'
        will be returned.
      '''
      if name not in info:
        return '- -'
      degC = info[name]
      degF = degC * 1.8 + 32.0
      return '%.2fC %.2fF' % (degC, degF)

    def _add_humidity(self, name, info):
      '''Helper method to add the humidity to a string. If no
        sensor data is
      available, then '-' will be returned.
      '''

      if name not in info:
        return '-'
      return '%d%%' % int(info[name])

    def print(self, results, use_json=False):
      '''Print out a list of all of the known USB sensor
```

```
        devices on the system.
      If 'use_json' is True, then JSON formatting will be used.
      '''

      if use_json:
        print(json.dumps(results, indent=4))
        return

      for info in results:
        s = 'Bus %03d Dev %03d %04x:%04x %s' % (info['busnum'],
                                                info['devnum'],
                                                info['vendorid'],
                                                info['productid'],
                                                info.get('firmware'))
        if 'error' in info:
          s += ' Error: %s' % info['error']
        else:
          s += ' ' + self._add_temperature('internal
             temperature', info)
          s += ' ' + self._add_humidity('internal humidity',
             info)
          s += ' ' + self._add_temperature('external
             temperature', info)
          s += ' ' + self._add_humidity('external humidity',
             info)
        print(s)

    def main(self):
      '''An example 'main' entry point that can be used to
         make temper.py a
      standalone program.
      '''

      parser = argparse.ArgumentParser(description='temper')
      parser.add_argument('-l', '--list', action='store_true',
                          help='List all USB devices')
      parser.add_argument('--json', action='store_true',
                          help='Provide output as JSON')
      parser.add_argument('--force', type=str,
                          help='Force the use of the hex id;
                             ignore other ids',
                          metavar=('VENDOR_ID:PRODUCT_ID'))
      parser.add_argument('--verbose', action='store_true',
                          help='Output binary data from
                             thermometer')
      args = parser.parse_args()
      self.verbose = args.verbose

      if args.list:
        self.list(args.json)
        return 0
```

```
    if args.force:
      ids = args.force.split(':')
      if len(ids) != 2:
        print('Cannot parse hexadecimal id: %s' % args.force)
        return 1
      try:
        vendor_id = int(ids[0], 16)
        product_id = int(ids[1], 16)
      except:
        print('Cannot parse hexadecimal id: %s' % args.force)
        return 1
      self.forced_vendor_id = vendor_id;
      self.forced_product_id = product_id;

    # By default, output the temperature and humidity for
       all known sensors.
    results = self.read(args.verbose)
    self.print(results, args.json)
    return 0


def send_reset(dev_info):
    """Sends the USBDEVFS_RESET IOCTL to a USB device.

    Uses the usb bus and id to reset the device under
       /dev/bus/usb/

    dev_info - The devfs path to the USB device (under
       /dev/bus/usb/)
    """
    bus, dev = dev_info['busnum'], dev_info['devnum']
    fd = os.open(f"/dev/bus/usb/{bus:03}/{dev:03}",
       os.O_WRONLY)
    try:
        fcntl.ioctl(fd, USBDEVFS_RESET, 0)
    finally:
        os.close(fd)


def main():
  temper = Temper()
  sys.exit(temper.main())

if __name__ == "__main__":
  main()
```

Experiencia 5 — Analisis de los datos.

A partir del archivo registrado:

- representar simultaneamente las temperaturas interna y externa;

- determinar la temperatura media de cada sensor;
- determinar las amplitudes de las fluctuaciones;
- calcular la diferencia entre ambos sensores;
- comparar sus tiempos de respuesta;
- buscar posibles retardos entre ambas señales.

Scilab. Si el archivo contiene las tres columnas numericas indicadas anteriormente, una posible implementacion del analisis en Scilab es la siguiente.

```
d = csvRead("temperaturas.csv")

t  = d(:,1)
Ti = d(:,2)
Te = d(:,3)

t = t-t(1)

printf('Sensor interno\n')
printf('Media:  %.3f C\n',mean(Ti))
printf('Minimo: %.3f C\n',min(Ti))
printf('Maximo: %.3f C\n',max(Ti))
printf('Desvio: %.3f C\n',stdev(Ti))

printf('\nSensor externo\n')
printf('Media:  %.3f C\n',mean(Te))
printf('Minimo: %.3f C\n',min(Te))
printf('Maximo: %.3f C\n',max(Te))
printf('Desvio: %.3f C\n',stdev(Te))

diferencia = Ti-Te
printf('\nDiferencia media: %.3f C\n', mean(diferencia))

plot(t,[Ti Te])
xlabel("Tiempo [s]")
ylabel("Temperatura [C]")
legend(["Interno","Externo"])
```

Preguntas.

1. ¿Que elementos de la cadena de medida estan contenidos dentro del TEMPer2?
2. ¿En que se diferencia este instrumento de un LM35 conectado a un conversor A/D?
3. ¿Que ventajas tiene obtener directamente una medida digital mediante USB?
4. ¿Por que una gran cantidad de cifras decimales en la salida del programa no garantiza igual exactitud?
5. ¿Los sensores interno y externo presentan el mismo tiempo de respuesta? Explique las diferencias observadas.
6. ¿Que frecuencia de muestreo resulta razonable para este instrumento?

7. ¿Por que no tendria sentido adquirir temperaturas a una frecuencia de varios kHz?
8. ¿Que ocurre si el intervalo entre mediciones es mucho menor que la constante de tiempo termica del sensor?
9. ¿Como utilizaria este instrumento para estudiar la ley de enfriamiento de Newton?
10. Compare las arquitecturas

    $$\text{sensor} \rightarrow \text{conversion interna} \rightarrow \text{USB} \rightarrow \text{computadora}$$

    y

    $$\text{LM35} \rightarrow \text{ADC} \rightarrow \text{interfaz digital} \rightarrow \text{computadora.}$$

11. ¿Que parte de esta experiencia depende especificamente del programa `temper_vdsa.py` y que partes son independientes del lenguaje utilizado?

11. **Ejercicio:** Adquisicion industrial con modulos ADAM

    Nivel C

    Objetivo.

    Utilizar un modulo ADAM como sistema de adquisicion industrial, comunicarse con el mediante RS-485, interpretar su protocolo de comandos y realizar adquisiciones automaticas de uno o varios canales.

    El objetivo es estudiar la arquitectura y el protocolo del instrumento. El lenguaje utilizado por la computadora para enviar y recibir los comandos no forma parte esencial del experimento.

    Equipo.

    - Modulo ADAM disponible en el laboratorio.
    - Fuente de alimentacion adecuada.
    - Conversor USB–RS-485.
    - Computadora con GNU/Linux.
    - Sensor o fuente de señal compatible con las entradas del modulo.
    - Multimetro para realizar mediciones de referencia.

    Parte A — Identificacion del modulo.

    Identificar el modelo exacto del modulo ADAM disponible. A partir de la informacion indicada en el equipo y de su documentacion determinar:

    - numero y tipo de entradas;
    - numero y tipo de salidas;
    - rangos de entrada;
    - resolucion;
    - alimentacion requerida;

- interfaz electrica de comunicacion;
- velocidad y formato de comunicacion;
- estructura de los comandos;
- direccion del modulo.

Registrar estos datos en el informe.

Parte B — Conexion RS-485.

Conectar el modulo al conversor USB–RS-485 respetando la polaridad de las lineas diferenciales, la masa de referencia cuando corresponda y la alimentacion indicada por el fabricante.

Conectar el conversor a la computadora e identificar el dispositivo serie. En GNU/-Linux:

```
ls -la /dev/ttyUSB* /dev/ttyACM* 2>/dev/null
```

Determinar que dispositivo corresponde al conversor utilizado.

Parte C — Protocolo de comunicacion.

Consultar la documentacion del modelo disponible y determinar el comando necesario para consultar la configuracion y leer un canal de entrada. Transmitir una consulta al modulo y observar la respuesta sin convertirla todavia a un valor fisico.

Identificar en las cadenas intercambiadas:

- delimitador o caracter inicial;
- direccion del modulo;
- comando;
- numero de canal, cuando corresponda;
- datos;
- terminador;
- mecanismo de comprobacion, si existe.

Distinguir claramente RS-485 como especificacion de la capa electrica del protocolo de comandos ADAM como formato de los mensajes intercambiados.

Scilab. Para el sistema ADAM utilizado historicamente en el laboratorio se encuentran disponibles en la unidad teorica funciones de comunicacion desde Scilab mediante Tcl. Si corresponden al modelo instalado, pueden utilizarse de la siguiente forma.

```
habilitar_ADAM()

b = leer_configuracion_ADAM(numero)
disp(b)
```

El valor de `numero`, la configuracion del puerto y la sintaxis de los comandos deben corresponder al modulo realmente utilizado.

Parte D — Medicion de un canal.

Conectar una señal compatible con uno de los rangos de entrada. Realizar una lectura del canal y conservar inicialmente la respuesta original del instrumento. Interpretar posteriormente esa respuesta segun la documentacion del modulo y obtener el valor fisico correspondiente.

Scilab. Con las funciones de referencia del laboratorio, cuando sean compatibles con el modulo disponible, puede utilizarse:

```
respuesta = medir_ADAM(canal,numero)
disp(respuesta)
```

La conversion de `respuesta` a numero debe realizarse segun el formato exacto indicado en el manual del modulo. No debe utilizarse una expresion generica que evalue arbitrariamente el texto recibido.

Medir la misma señal mediante un multimetro de referencia y calcular

$$e = x_{\mathrm{ADAM}} - x_{\mathrm{ref}}$$

y, para $x_{\mathrm{ref}} \neq 0$,

$$e_r = \frac{x_{\mathrm{ADAM}} - x_{\mathrm{ref}}}{x_{\mathrm{ref}}}\,100\,\%.$$

Parte E — Adquisicion automatica.

Realizar por lo menos 100 mediciones periodicas del mismo canal. Para cada muestra registrar t_k y x_k.

El programa de adquisicion debe:

1. mantener abierta la comunicacion durante toda la experiencia;
2. enviar el comando correspondiente al canal;
3. esperar la respuesta;
4. validar la respuesta;
5. convertirla al valor numerico segun el protocolo;
6. asociarla a una referencia temporal;
7. almacenarla.

Scilab. La estructura general de una adquisicion con las funciones proporcionadas puede escribirse en Scilab como sigue.

```
N  = 100
dt = 1

t = zeros(N,1)
```

```
x = zeros(N,1)

habilitar_ADAM()

for k = 1:N
    t(k) = (k-1)*dt
    respuesta = medir_ADAM(canal,numero)

    // Interpretar aqui la respuesta
    // segun el manual del modulo.
    // x(k) = ...

    sleep(dt*1000)
end

deshabilitar_ADAM()
csvWrite([t x],"adam.csv")
```

Completar la conversion solamente despues de identificar el modelo y el formato exacto de su respuesta. Representar la magnitud medida en funcion del tiempo.

Parte F — Adquisicion multicanal.

Si el modulo dispone de varias entradas, conectar señales a por lo menos dos canales. Adquirir sucesivamente $x_1, x_2, \ldots, x_n$ y registrar para cada conversion el instante correspondiente.

Determinar experimentalmente:

- si los canales son adquiridos simultanea o secuencialmente;
- tiempo necesario para recorrer los canales utilizados;
- frecuencia efectiva de adquisicion de cada canal.

Parte G — Analisis de la arquitectura.

Identificar fisicamente donde se realizan:

- el acondicionamiento de la señal;
- la conversion analogico–digital;
- la comunicacion digital;
- el almacenamiento;
- el procesamiento de los datos.

Representar la cadena de medida como

$$\text{sensor} \longrightarrow \text{ADAM} \longrightarrow \text{RS-485} \longrightarrow \text{USB–RS-485} \longrightarrow \text{computadora}.$$

Indicar que informacion atraviesa cada una de las interfaces.

Preguntas.

1. ¿Que diferencia existe entre RS-485 y el protocolo de comandos utilizado por el modulo ADAM?
2. ¿Por que RS-485 utiliza señalizacion diferencial?
3. ¿Que ventaja presenta frente a una conexion serie TTL en instalaciones industriales?
4. ¿Como pueden conectarse varios modulos al mismo bus?
5. ¿Como se determina a que modulo esta dirigido un comando?
6. ¿La conversion A/D se realiza en la computadora o en el modulo ADAM?
7. ¿Las mediciones de diferentes canales son simultaneas?
8. ¿Que factores limitan la frecuencia maxima de adquisicion?
9. Compare
$$\text{sensor} \rightarrow \text{ADAM} \rightarrow \text{RS-485} \rightarrow \text{computadora}$$
con
$$\text{sensor} \rightarrow \text{microcontrolador} \rightarrow \text{USB} \rightarrow \text{computadora}.$$
10. ¿En que situaciones resulta conveniente utilizar un sistema industrial como ADAM en lugar de un Arduino o ESP32?
11. ¿Que parte del procedimiento depende del protocolo ADAM y que parte depende solamente del lenguaje utilizado en la computadora?

12. **Ejercicio:** Sistema de adquisicion de datos multicanal

Nivel C

Objetivo.

Construir y caracterizar un sistema general de adquisicion capaz de medir varias señales analogicas y determinar experimentalmente sus principales limitaciones.

Este practico integra los conceptos desarrollados en los ejercicios anteriores y no presupone una plataforma ni un lenguaje de programacion determinados.

El sistema puede implementarse mediante:

- una tarjeta de adquisicion dedicada;
- un modulo externo de conversion A/D y D/A;
- un microcontrolador;
- un modulo industrial;
- otro hardware disponible en el laboratorio que permita realizar las experiencias propuestas.

Parte A — Caracterizacion del sistema.

Identificar:

- numero de entradas analogicas;
- numero de salidas analogicas, si existen;

- resolucion de los conversores A/D y D/A;
- rangos de tension;
- frecuencia maxima de conversion;
- metodo de adquisicion de varios canales;
- interfaz con la computadora;
- mecanismo utilizado para transferir los datos.

Distinguir la frecuencia maxima del conversor de la frecuencia efectiva que puede obtenerse para cada canal.

Parte B — Adquisicion multicanal.

Conectar por lo menos dos señales diferentes a distintos canales. Pueden utilizarse, segun la disponibilidad:

- una tension continua;
- una señal sinusoidal;
- un sensor de temperatura;
- una LDR u otro sensor analogico.

Para cada canal construir una serie (t_i, x_i). Registrar, cuando sea posible, el instante correspondiente a cada conversion.

Determinar si las muestras de distintos canales son realmente simultaneas o si se obtienen mediante multiplexacion temporal. En el segundo caso determinar el intervalo temporal entre canales.

Parte C — Frecuencia de muestreo y aliasing.

Aplicar una señal periodica cuya frecuencia pueda variarse. Realizar adquisiciones utilizando distintas frecuencias de muestreo.

Determinar experimentalmente:

- numero de muestras por periodo;
- frecuencia maxima que puede representarse adecuadamente;
- aparicion de aliasing;
- comportamiento al aproximarse a la frecuencia de Nyquist.

Comparar los resultados con $f_{\text{Nyquist}} = f_s/2$. Cuando corresponda, analizar el espectro de la señal adquirida.

Parte D — Resolucion del conversor.

Aplicar una señal continua de amplitud conocida y variar lentamente su valor. Determinar el menor cambio de tension que produce un cambio observable en el codigo del ADC.

Comparar el resultado con

$$\Delta V = \frac{V_{\text{máx}} - V_{\text{mín}}}{2^N},$$

donde N es el numero de bits del conversor.

Distinguir entre resolucion teorica, ruido observado y exactitud de la medida.

Parte E — Ruido y repetibilidad.

Mantener constante la señal de entrada y adquirir un numero grande de muestras. Calcular:

- valor medio;
- desviacion estandar;
- valor maximo;
- valor minimo;
- rango observado.

Representar la distribucion de las lecturas. Comparar la dispersion observada con la resolucion nominal del ADC. Promediar grupos de N muestras y estudiar como cambia la dispersion de la media al aumentar N.

Parte F — Conversion D/A, si esta disponible.

Si el sistema posee una salida analogica, programar varios valores conocidos mediante el conversor D/A y medirlos posteriormente con una entrada A/D o con un instrumento externo.

Comparar el valor programado con el valor medido y determinar que errores pueden atribuirse al conversor D/A, al conversor A/D, a la referencia de tension y al ruido y acondicionamiento.

Parte G — Registro prolongado.

Realizar una adquisicion prolongada de varios canales y almacenar los datos en un formato independiente del programa utilizado para obtenerlos. Se recomienda un archivo CSV del tipo

```
tiempo,canal1,canal2,canal3
```

Registrar junto con los datos, en el informe o en un archivo de metadatos:

- fecha y hora;
- identificacion del sistema de adquisicion;
- frecuencia de muestreo;
- rango utilizado;
- resolucion;
- identificacion de los sensores;
- parametros de calibracion;
- unidades de cada canal.

Parte H — Analisis de los datos.

El programa utilizado para el analisis queda a eleccion. Obtener como minimo:

- graficos de cada variable en funcion del tiempo;
- valores medios;
- desviaciones estandar;
- valores maximos y minimos;
- correlaciones entre canales;
- espectros de frecuencia cuando corresponda.

Scilab. Si el archivo CSV contiene exclusivamente columnas numericas, una posible base para el analisis en Scilab es la siguiente.

```
d = csvRead("datos.csv")

t = d(:,1)
t = t-t(1)
ncanales = size(d,2)-1

for k = 1:ncanales
    x = d(:,k+1)
    printf('\nCanal %d\n',k)
    printf('Media:  %.6f\n',mean(x))
    printf('Desvio: %.6f\n',stdev(x))
    printf('Minimo: %.6f\n',min(x))
    printf('Maximo: %.6f\n',max(x))
end
```

Para una señal periodica puede calcularse tambien su transformada discreta de Fourier.

Scilab. Por ejemplo, para un canal muestreado uniformemente puede utilizarse:

```
x = d(:,2)

dt = mean(diff(t))
fs = 1/dt
N = length(x)
x = x-mean(x)
Y = fft(x)
N2 = floor(N/2)
f = (0:N2-1)*fs/N
A = 2*abs(Y(1:N2))/N

plot(f,A)
xlabel("Frecuencia [Hz]")
ylabel("Amplitud")
```

Parte I — Comparacion de arquitecturas.

Comparar las distintas arquitecturas utilizadas durante la unidad:

$$\text{sensor analogico} \rightarrow \text{ADC} \rightarrow \text{computadora},$$
$$\text{sensor} \rightarrow \text{microcontrolador} \rightarrow \text{computadora},$$
$$\text{sensor} \rightarrow \text{ADAM} \rightarrow \text{RS-485} \rightarrow \text{computadora},$$
$$\text{TEMPer2} \rightarrow \text{USB/HID} \rightarrow \text{computadora}.$$

Para cada arquitectura indicar donde se realizan:

- la transduccion;
- el acondicionamiento;
- la conversion A/D;
- el muestreo temporal;
- el almacenamiento temporal;
- la comunicacion digital;
- el almacenamiento permanente;
- el procesamiento final.

Preguntas.

1. ¿Que diferencia existe entre frecuencia de conversion del ADC, frecuencia de muestreo del sistema y frecuencia efectiva por canal?
2. ¿En que condiciones puede considerarse que una adquisicion multicanal multiplexada es aproximadamente simultanea?
3. ¿Como puede detectarse experimentalmente el aliasing?
4. ¿Por que aumentar el numero de bits del ADC no garantiza por si solo una mayor exactitud?
5. ¿Que relacion existe entre resolucion nominal y ruido observado?
6. ¿Que informacion debe conservarse junto con un archivo de datos para que las mediciones puedan interpretarse posteriormente?
7. Compare las ventajas y limitaciones de una adquisicion basada en microcontrolador, un modulo industrial y un instrumento USB integrado.
8. ¿Que partes de este practico dependen del hardware utilizado?
9. ¿Que partes dependen del protocolo de comunicacion?
10. ¿Que partes son completamente independientes del lenguaje de programacion empleado?

Capítulo 10

Integración

10.1. Programa

UNIDAD 10: No es unidad, es un contenedor para hacer el proyecto

Como cierre de la asignatura se debe completar un proyectoque integre los contenidos desarrollados durante el curso.

No tiene unidad temática, pues es un práctico integrador.

10.2. Proyecto final integrador

10.2.1. Descripcion general

Consiste en el desarrollo completo de un sistema computacional de aplicacion cientifica o de ingenieria que integre los contenidos trabajados a lo largo del curso: hardware, sistema operativo, redes, programacion, simulacion numerica, adquisicion de datos, visualizacion y documentacion.

El proyecto no es un ejercicio academico cerrado sino un trabajo con valor real: el resultado debe ser un sistema que funcione, sea utilizable por otras personas y este documentado de forma que pueda ser mantenido y extendido.

La filosofia del proyecto es la misma que guio la materia: la vagancia bien entendida, automatizar todo lo que sea posible, escribir codigo reproducible y documentado, y usar herramientas libres.

En lo específico el procesamiento será en Scilab, la visualizacion gráfica con Gnuplot y una interface gráfica de usuario, preferiblemente vía web. Cuando hay comunicación con dispositivos externos ésto se maneja desde el calculador de scilab.

10.2.2. Tipos de proyecto

Se proponen cuatro tres de proyecto segun los intereses y la disponibilidad de equipamiento, sin ser una lista cerrada:

Tipo A — Simulacion y modelado: desarrollo de un sistema que incluya un simulador numerico de un sistema fisico, quimico, biologico o economico. Incluye la formulacion matematica del modelo, la implementacion de los metodos numericos, la validacion y la interfaz web para interactuar con el simulador.

No requiere hardware adicional.

Ejemplos:

- Simulador de la propagacion de calor en una barra (ecuacion de difusion en 1D y 2D).
- Simulacion del doble pendulo y visualizacion del comportamiento caotico.
- Modelo epidemiologico SIR/SEIR con parametros variables y graficado de curvas de contagio.
- Simulacion de orbitas planetarias con el metodo de Verlet.
- Modelo de dinamica poblacional con depredadores, presas y recursos limitados.
- Simulacion de circuitos RC, RL y RLC con distintas formas de onda de excitacion.
- Modelo de difusion de contaminantes en un rio.
- Simulación térmica de un tanque solar.
- Procesamiento de datos experimentales en general.
- Adquisición, análisis y representación de mediciones realizadas.

Un ejemplo de dimensionamiento vía simulación de un sistema fotovoltaico.

Dimensionar sistema fotovoltaico a partir de datos de radiación.

Para ello simularlo, y calcular el ahorro y la inversión, ante diferentes dimensiones de paneles y baterias.

Se requiere: irradiancia, temperatura, y demanda, a lo largo del día.

Se prueban diferentes dimensiones de paneles y baterias, y se elije la óptima.

Tipo B — Adquisicion y analisis: sistema de medicion y analisis de datos experimentales.

Incluye hardware especial.

Ejemplos:

- Estacion meteorologica con temperatura, humedad, presion y visualizacion historica.
- Medicion de la constante de tiempo termica de distintos materiales.
- Analizador de vibraciones de una estructura usando un acelerometro.
- Monitor de calidad del aire (CO2, particulas).
- Registrador de consumo electrico de un hogar.
- Medicion de curvas I-V de diodos y paneles solares.

Tipo C — Control automatico: sistema de control en lazo cerrado de un proceso fisico real.

Incluye harware especial.

Ejemplos:

- Control PID de temperatura de un horno o bano termico.
- Control de nivel de liquido en un tanque.
- Control de velocidad de un motor DC.
- Sistema de riego automatico con sensor de humedad del suelo.
- Seguidor solar: plataforma que orienta un panel solar hacia la luz.

Cuando dice que incluye hardware especial se piensa en la construccion de un circuito de adquisicion via USB, por ejemplo con Arduino, ESP32 u otro hardware, la comunicacion con la computadora.

Y en punto del controlador la estructuración del controlador en Scilab y el envio de comandos al actuador.

10.2.3. Seleccion del proyecto

Criterios de seleccion

Un buen proyecto debe cumplir los siguientes criterios:

Viabilidad: se puede completar en el tiempo disponible con los recursos accesibles. Es preferible un proyecto modesto completado que uno ambicioso a medias.

Originalidad: no es una repeticion exacta de los practicos ni una copia de un proyecto existente. Puede basarse en ellos pero debe agregar algo propio.

Relevancia: tiene aplicacion en ciencia, ingenieria, educacion o la vida cotidiana. No es un ejercicio puramente academico sino un sistema con utilidad real.

Integración: usa al menos tres de las siguientes areas: hardware/SO, redes, programación estructurada, metodos numericos, adquisicion de datos, visualizacion, documentacion en LaTeX, control de versiones.

Reproducibilidad: alguien que lea la documentacion y tenga el repositorio puede reproducir el proyecto completo sin ayuda adicional.

Proceso de seleccion

1. **Semana 1 — Propuesta inicial:** escribir en media pagina la descripcion del proyecto, el tipo, los objetivos especificos y los recursos disponibles. Presentarla al docente para recibir retroalimentacion.

   Puede hacerse en simultaneo con otros prácticos de la materia.

2. **Semana 2 — Propuesta aprobada:** incorporar las observaciones del docente y entregar la propuesta definitiva de una pagina con: objetivo general, tres objetivos especificos medibles, lista de componentes fisicos necesarios, descripcion de la arquitectura del software y cronograma tentativo.

   Puede hacerse en simultaneo con otros prácticos de la materia.

3. **Semana 3 — Prototipo minimo:** tener funcionando la parte mas critica o mas incierta del proyecto. Si el proyecto depende de un sensor especifico, haberlo probado. Si depende de un metodo numerico, haberlo implementado. El prototipo minimo evita descubrir problemas criticos en la ultima semana.

   Se dedicara una semana exclusiva de la materia a esto.

Otra semana para su finalización y presentación.

10.2.4. Estructura del repositorio

Todo el proyecto debe estar en un repositorio Git o similar Wilson, Bryan et al. 2017; Chacon y Straub 2014 con la siguiente estructura:

```
mi_proyecto/
    README.md            # descripcion breve y como ejecutar
    CLAUDE.md            # contexto para IA (si se usa Claude u
       similar si se usa otra)
    documentacion/
        informe.tex      # informe completo en LaTeX
        informe.pdf      # PDF compilado
        diagramas/       # imagenes y diagramas
    src/
        servidor.py      # servidor HTTP (si aplica)
        calcular.sce     # calculo en Scilab
        formulario.html  # interfaz web (si aplica)
        arduino/
        firmware.ino   # codigo del Arduino (si aplica)
    datos/
        calibracion.csv    # datos de calibracion
        mediciones/        # datos adquiridos
    tests/
        test_calculo.sce   # tests del modulo de calculo
    resultados/
        graficas/          # PNGs generados por Gnuplot
        logs/              # logs de sesiones de adquisicion
```

```
# inicializar el repositorio desde el primer dia
mkdir mi_proyecto
cd mi_proyecto
git init
git config user.name "Nombre Apellido"
git config user.email "correo@unsa.edu.ar"
```

```
# crear el README inicial
cat > README.md << 'EOF'
# Nombre del proyecto

**Autor:** Nombre Apellido
**DNI:** XXXXX
**Materia:** Laboratorio de Computacion
**Fecha:** La que corresponda

## Descripcion breve

Una o dos oraciones describiendo el proyecto.

## Como ejecutar

```bash
python3 src/servidor.py
# abrir http://localhost:8080
```

Requisitos

- Scilab branch-main (2024)
- Python 3.8+
- gnuplot
- (hardware: Arduino Uno, sensor LM35)
EOF

git add README.md
git commit -m "inicio del proyecto"
```

Los commits deben ser frecuentes Perez-Riverol et al. 2016b; Blischak, Davenport y Wilson 2016 (al menos uno por sesion de trabajo) y con mensajes descriptivos que expliquen que se cambio y por que, no solo que archivos:

```
# mal: no dice nada util
git commit -m "cambios"

# bien: describe el progreso real
git commit -m "implementar metodo RK4 para la EDO del pendulo"
git commit -m "agregar validacion de parametros en calcular.sce"
git commit -m "corregir bug: el grafico fallaba con datos vacios"
git commit -m "documentar la funcion de calibracion"
```

10.2.5. Metodologia de desarrollo

Desarrollo incremental

El proyecto se desarrolla en etapas McConnell 1996; Balaban et al. 2021, verificando el funcionamiento en cada una antes de avanzar a la siguiente. No se pasa a la etapa N+1 sin que la etapa N funcione correctamente.

Las etapas podrian ser:

1. Formulario HTML estatico que se muestra correctamente.
2. Servidor Python que sirve el formulario y parsea el POST.
3. Script Scilab minimo que lee los parametros y escribe un resultado de prueba.
4. Calculo real implementado y verificado.
5. Gráfica con Gnuplot funcionando.
6. Integracion completa: formulario + servidor + Scilab + Gnuplot.
7. Documentacion y tests.

Para un proyecto con arduino o similar:

1. Circuito construido y sensor verificado con multimetro.
2. Codigo Arduino que lee el sensor y envia por serie.
3. Lectura del puerto serie/usb desde la terminal (sin Scilab).
4. Lectura desde Scilab y visualizacion basica.
5. Actuadores, de existir.
6. Calculo y analisis de los datos adquiridos.
7. Gráfica con Gnuplot.
8. Calibracion y cuantificacion de errores.
9. Documentacion.

Test-driven development

Para cada funcion o modulo que se implemente, escribir primero un test que defina el comportamiento esperado, luego implementar hasta que el test pase. El método se debe a Beck Beck 2003; las reglas prácticas para que el código funcione en una máquina que no sea la propia están en Taschuk y Wilson Taschuk y Wilson 2017.

Listing 10.1: tests del proyecto en Scilab

```
// tests/test_calculo.sce
// ejecutar con: scilab-cli -e
   "exec('tests/test_calculo.sce',-1);quit"

OK   = 0
FAIL = 0

function check(desc, cond)
    if cond then
        printf('  ok: %s\n', desc)
        OK = OK + 1
    else
        printf('  FALLO: %s\n', desc)
        FAIL = FAIL + 1
    end
endfunction

// cargar las funciones a testear
exec('src/calcular.sce', -1)

// test 1: casos conocidos
check('temperatura en t=0 es la inicial', ..
      abs(simular(20, 0, 0.05) - 20) < 0.01)
check('temperatura tiende a T_amb en t grande', ..
      abs(simular(20, 1000, 0.05) - 20) < 0.1)

// test 2: casos borde
check('v0=0 no da error', ~isnan(simular(0, 1, 0.05)))
check('angulo=90 da alcance=0', ..
      abs(alcance(50, 90, 0)) < 0.1)

// test 3: conservacion de energia (sin rozamiento)
E0 = energia(1, 0, 9.8)     // E inicial
Et = energia(1, 10, 9.8)    // E final
check('energia conservada sin rozamiento', ..
      abs(E0 - Et)/E0 < 0.01)

printf('\nResultado: %d ok, %d fallos\n', OK, FAIL)
quit
```

Uso de inteligencia artificial

Se puede usar IA (Claude, ChatGPT u otra) como herramienta de apoyo, con las siguientes reglas:

- Todo codigo generado por IA debe ser leido, entendido y verificado por el estudiante antes de incorporarlo al proyecto. No se acepta copiar codigo sin comprender que hace.

- Si se uso IA para generar una funcion, el estudiante debe poder explicar cada linea en la defensa.
- Las ideas centrales del proyecto, la seleccion de metodos y la validacion de resultados son responsabilidad del estudiante, no de la IA.
- Se recomienda documentar explicitamente en el informe que partes se desarrollaron con ayuda de IA y que verificaciones se hicieron.

10.2.6. Informe tecnico

El informe es el documento central del proyecto Whitesides 2004. Se escribe en LaTeX y debe poder leerse de forma independiente del codigo.

Estructura del informe

1. **Caratula:** titulo, autor, legajo, materia, fecha.
2. **Resumen** (media pagina): que hace el sistema, que metodos usa, que resultados se obtuvieron. Alguien que lea solo el resumen debe entender el proyecto.
3. **Introduccion:** descripcion del problema que resuelve el proyecto, motivacion, objetivos especificos medibles, alcance y limitaciones.
4. **Marco teorico:** la teoria necesaria para entender el proyecto. No copiar el apunte sino seleccionar y explicar lo relevante para el caso especifico. Incluir las ecuaciones del modelo con sus variables definidas y sus unidades.
5. **Diseno del sistema:** arquitectura general, diagrama de bloques, descripcion de cada componente (hardware y software), decisiones de diseno y justificacion de las mismas.
6. **Implementacion:** descripcion del codigo con los fragmentos mas importantes comentados, el circuito (si hay hardware), el protocolo de comunicacion, los metodos numericos implementados con su orden de error y criterio de convergencia.
7. **Calibracion y validacion:** para proyectos con medicion, el procedimiento de calibracion y la cuantificacion de los errores. Para proyectos de simulacion, la comparacion con soluciones analiticas o datos experimentales de referencia. Para proyectos de control, la respuesta al escalon y los parametros del controlador.
8. **Resultados:** los resultados concretos del sistema funcionando. Gráficas con etiquetas de ejes, unidades y leyendas. Tablas de mediciones con incertidumbres. No solo mostrar que funciona sino analizar los resultados.
9. **Discusión:** que funciona bien, que limitaciones tiene, que se haria diferente si se comenzara de nuevo, como podria extenderse el trabajo.
10. **Conclusiones:** respuesta concreta a los objetivos planteados en la introduccion. Los objetivos se plantearon como preguntas; las conclusiones son las respuestas.

11. **Referencias:** en formato APA, incluyendo los manuales de los componentes usados, las fuentes de los metodos numericos y el material de la materia.

12. **Apendices:** codigo fuente completo (puede referenciarse al repositorio en lugar de incluirlo completo), esquematico del circuito, hoja de datos de los componentes, datos de calibracion completos.

Plantilla LaTeX del informe

Listing 10.2: informe.tex plantilla

```
\documentclass[12pt,a4paper]{article}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{geometry}
\usepackage{listings}
\usepackage{xcolor}
\usepackage{graphicx}
\usepackage{booktabs}
\usepackage{amsmath}
\usepackage{amssymb}
\usepackage{hyperref}
\usepackage{parskip}

\geometry{margin=2.5cm}

\title{\textbf{Titulo del proyecto}\\
       \large Proyecto Final Integrador\\
       Laboratorio de Computacion}
\author{Nombre Apellido\\
        Legajo: XXXXX\\
        \texttt{correo@estudiante.unsa.edu.ar}}
\date{Julio 2026}

\begin{document}
\maketitle
\tableofcontents
\newpage

\begin{abstract}
Descripcion del proyecto en tres a cinco oraciones.
Que hace, como lo hace, que resultados se obtienen.
\end{abstract}

\section{Introduccion}
...

\section{Marco teorico}
...

\section{Diseno del sistema}
```

```
...

\section{Implementacion}
...

\section{Resultados}
...

\section{Conclusiones}
...

\bibliographystyle{apalike}
\bibliography{referencias}

\appendix
\section{Codigo fuente}
Disponible en: \url{https://github.com/usuario/proyecto}

\end{document}
```

10.2.7. Citando software

La práctica de tratar el software como una referencia propia, distinta de un artículo o manual que lo describa, está formulada en los principios de citación de software de FORCE11 (Smith, Katz, Niemeyer y el FORCE11 Software Citation Working Group, 2016, PeerJ Computer Science 2:e86, doi 10.7717/peerj-cs.86).

El primero de los seis principios es el de "importancia": el software debe considerarse un producto de investigación legítimo y citable en sí mismo. La recomendación derivada es citar el software —código, scripts, notebooks, bibliotecas— del mismo modo que se citan artículos y libros.

El documento se ocupa de cómo citar el software; que su documentación vaya como entrada aparte.

Del lado de la herramienta, el tipo @software está definido en el manual de biblatex, que sería la referencia técnica.

10.2.8. Presentacion y defensa

La presentacion es oral con demostracion en vivo Bourne 2007 del sistema funcionando. Tiene una duracion de 20 minutos:

5 minutos — Presentacion del problema: que problema resuelve el proyecto, por que es interesante, que metodos se usaron. Sin entrar en detalles tecnicos. La audiencia no necesita conocer el proyecto previamente.

10 minutos — Demostracion en vivo: el sistema funcionando frente al tribunal. No una captura de pantalla ni un video: el sistema real ejecutandose. Se demuestran

los casos de uso principales y al menos un caso borde o de error.

5 minutos — Preguntas: el tribunal pregunta sobre cualquier aspecto del proyecto: decisiones de diseno, alternativas consideradas, limitaciones, como se implemento una funcion especifica, que pasaria si se cambia un parametro.

Preparacion para la defensa

Antes de la defensa verificar que:

```
# el repositorio esta actualizado y limpio
git status       # no debe haber cambios sin commitear
git log --oneline | head -10   # historia de commits visible

# el sistema arranca desde cero en una maquina limpia
git clone https://github.com/usuario/proyecto /tmp/prueba
cd /tmp/prueba
python3 src/servidor.py &
curl http://localhost:8080 | head -5   # verifica que responde

# los tests pasan todos
scilab-cli -e "exec('tests/test_calculo.sce',-1);quit"
# debe terminar con 0 fallos

# el PDF del informe compila
cd documentacion
pdflatex informe.tex && pdflatex informe.tex
ls -lh informe.pdf

# el hardware funciona (si aplica)
ls /dev/ttyUSB*    # Arduino conectado
# probar la adquisicion basica
```

Preparar respuestas para las preguntas tipicas:

- Por que eligio este metodo numerico y no otro?
- Cual es el error de su sistema de medicion?
- Que pasa si se le da un valor fuera de rango?
- Como verifico que el resultado es correcto?
- Que cambiaria si tuviera que rehacerlo?
- Que pasaria si la frecuencia de muestreo fuera la mitad? El doble?
- Puede mostrar un caso donde el sistema falla?

10.2.9. Criterios de evaluacion

La evaluacion tiene cuatro dimensiones con igual peso:

Dimension	Que se evalua	Peso
Funcionamiento	El sistema funciona correctamente, maneja casos borde, tiene tests que pasan, es reproducible desde el repositorio.	25 %
Calidad tecnica	Codigo legible y documentado, modularidad, uso correcto de los metodos numericos, cuantificacion de errores, control de versiones con historia significativa.	25 %
Informe	Claridad, completitud, correcto uso de LaTeX, ecuaciones con variables definidas, graficas con ejes etiquetados, referencias en formato APA, conclusiones alineadas con los objetivos.	25 %
Defensa	Claridad en la explicacion, capacidad de responder preguntas sobre cualquier parte del proyecto, dominio del tema, demostracion en vivo exitosa.	25 %

Indicadores de excelencia

Un proyecto sobresaliente tiene alguna de estas caracteristicas:

- El sistema es genuinamente util y podria usarse fuera del contexto academico.
- La documentacion es tan clara que otra persona podria extender el proyecto sin ayuda.
- Los tests cubren sistematicamente todos los casos incluidos los bordes y los de error.
- El estudiante puede demostrar que comprende las limitaciones del sistema y propone mejoras concretas.
- El proyecto va mas alla de los requisitos minimos en algun aspecto bien fundamentado.

Causas de desaprobacion

Un proyecto se desaprueba si:

- No funciona en la demostracion en vivo.
- El estudiante no puede explicar una funcion que aparece en su propio codigo.

- El repositorio no tiene historia de commits (todo subido de una vez el dia antes de la entrega).
- El informe no compila con `pdflatex`.
- Los resultados no son validados (el sistema produce numeros pero no se verifica que sean correctos).
- Se detecta copia de un proyecto de otro estudiante.

10.2.10. Consejos varios

Alcance, plazos y la trampa del proyecto ambicioso

El error más común no es técnico sino de dimensionamiento: elegir un proyecto que no entra en el tiempo disponible. McConnell McConnell 1996 cataloga los errores clásicos que hunden un proyecto con fecha de entrega y desarrolla la estrategia del recorte de alcance y las entregas por etapas. Brooks Brooks 1995 explica por qué agregar gente a un proyecto atrasado lo atrasa todavía más, algo que conviene leer antes de armar los grupos y no después. Balaban y colaboradores Balaban et al. 2021 tratan específicamente el caso de este práctico: cómo programar rápido sin programar mal, con prototipos mínimos centrados en la pregunta del momento y sin implementar funciones especulativas.

Organización del proyecto

Wilson y colaboradores Wilson, Bryan et al. 2017 definen el mínimo viable de organización: estructura de directorios, nombres de archivos y manejo de los datos. Está escrito para gente que no viene de computación, que es la situación de esta materia; la estructura de repositorio propuesta más arriba sigue ese criterio. El tratamiento más amplio de buenas prácticas está en Wilson, Aruliah et al. 2014: escribir para personas antes que para la máquina, automatizar lo repetitivo y no optimizar antes de tiempo.

Escribir el código

Kernighan y Pike Kernighan y Pike 1999 es el más cercano al tamaño de este proyecto: programas chicos y bien hechos, con capítulos de estilo, depuración, pruebas y portabilidad. Hunt y Thomas A. Hunt y Thomas 2019 reúne consejos aplicables sin adoptar ninguna metodología. Martin Martin 2008 se ocupa de nombres, funciones y comentarios; Fowler Fowler 2018 de qué hacer cuando el prototipo crece y hay que reorganizarlo sin romperlo, que es exactamente lo que ocurre entre la semana 3 y la 5.

Pruebas y validación

Beck Beck 2003 presenta el desarrollo guiado por pruebas con ejemplos completos. Taschuk y Wilson Taschuk y Wilson 2017 apuntan a algo distinto y complementario: que el programa funcione en una máquina que no sea la del autor, con mensajes de error útiles,

parámetros por línea de comandos y dependencias declaradas. Es el criterio con el que se evalúa la reproducibilidad desde el repositorio.

Control de versiones

Blischak y colaboradores Blischak, Davenport y Wilson 2016 es una introducción breve con tutorial, suficiente para empezar. Perez-Riverol y colaboradores Perez-Riverol et al. 2016b cubren el flujo de trabajo en repositorios públicos, y su primera regla es el archivo de licencia. Chacon y Straub Chacon y Straub 2014 es la referencia completa, libre y con traducción al castellano.

Reproducibilidad

Sandve y colaboradores Sandve et al. 2013 formulan las reglas de fondo: registrar cómo se produjo cada resultado, evitar los pasos manuales, archivar las versiones de los programas externos y guardar los resultados intermedios. Boettiger Boettiger 2015 presenta los contenedores para cuando el problema es reproducir el entorno completo y no sólo distribuir el programa. Si el proyecto usa notebooks, Rule y colaboradores Rule et al. 2019 dan las reglas equivalentes para ese formato.

Documentación, licencia y publicación

Lee Lee 2018 trata el README, los ejemplos de uso y la documentación de la interfaz: es lo que falta cuando el proyecto se entrega y nadie sabe cómo ejecutarlo. Morin y colaboradores Morin, Urban y Sliz 2012 explican cómo elegir licencia, con la diferencia entre copyleft y permisivas, tema que enlaza con la unidad sobre software libre. List y colaboradores List, Ebert y Albrecht 2017 cubren el paso del prototipo a algo que otra persona pueda instalar y usar, y Prlić y Procter Prlić y Procter 2012 el desarrollo abierto. Si el sistema se publica y se quiere que sea citable, los principios están en A. M. Smith, Katz y Niemeyer 2016; para empaquetarlo como paquete nativo, en el tutorial de Debian Nussbaum 2026 junto con el manual de políticas Debian Project 2026. La discusión sobre infraestructura y entornos está en Stodden y Miguez 2014.

Comunicar el resultado

Whitesides Whitesides 2004 son tres páginas sobre cómo se arma un texto técnico, empezando por el esquema y no por el texto. Bourne Bourne 2007 da las reglas para la defensa oral: hablarle a la audiencia, menos es más y pensar la presentación como un relato. Para las gráficas del informe, Tufte Tufte 2001 y Wilke Wilke 2019.

Referencia de conjunto

Si hace falta una visión general de ingeniería del software —requerimientos, diseño, verificación, gestión— el manual es Sommerville Sommerville 2015. Sobre método de trabajo, el manifiesto ágil Beck, Beedle et al. 2001 son cuatro valores en una página, la guía de

Scrum Schwaber y Sutherland 2020a define un marco de iteraciones con entregas parciales, y Beck y Andres Beck y Andres 2004 desarrollan las prácticas de equipos chicos: ciclos cortos, integración continua y programación de a pares.

10.2.11. Cronograma sugerido

Semana	Hito
1	Seleccion del tipo y tema. Propuesta de media pagina. Repositorio creado con README.
2	Propuesta aprobada. Marco teorico escrito en LaTeX. Estructura de directorios completa.
3	Prototipo minimo funcionando. Primera version de tests. Al menos 5 commits significativos.
4	Implementacion principal completa. Calibracion o validacion realizada. Resultados preliminares.
5	Sistema completo integrado. Informe con todas las secciones (pueden estar incompletas). Tests pasando.
6	Informe terminado y compilando. Defensa preparada. Repositorio limpio y documentado.
7	Presentaciones y defensas.

10.2.12. Lista de verificacion final

Antes de la entrega, verificar cada item:

- ☐ El repositorio Git tiene al menos 10 commits con mensajes descriptivos distribuidos a lo largo del tiempo.
- ☐ El `README.md` describe como ejecutar el proyecto desde cero.
- ☐ El sistema funciona ejecutando los comandos del README en una terminal limpia.
- ☐ Los tests corren sin fallos.
- ☐ El informe compila con `pdflatex informe.tex` sin errores.
- ☐ El informe tiene todas las secciones requeridas.
- ☐ Las graficas tienen titulo, ejes etiquetados con unidades y leyenda.
- ☐ Las ecuaciones tienen todas las variables definidas con sus unidades.
- ☐ Las referencias estan en formato APA.
- ☐ Los errores de medicion estan cuantificados (para proyectos con medicion).
- ☐ Los resultados son validados con al menos un caso conocido.
- ☐ El hardware funciona y esta listo para la demostracion (si aplica).
- ☐ La URL del repositorio esta en el informe.

10.3. Ejercicios

Se plantean varios ejercicios que simplemente agregan consignas a los desarrollados en el práctico 4, cubriendo temas que en ese momento no se habian visto: interfaces web y gráficas. Son complementos.

No se incluyen soluciones por lenguaje de lo solicitado, ya hay en prácticos anteriores suficientes ejemplos de uso de graficos y del servidor web para las GUI.

El último es un modelo para el proyecto que debe desarrollar el estudiante.

1. **Ejercicio:** Ecuaciones diferenciales ordinarias

   Corresponde y continua el ejercicio homónimo del práctico 4.

   1. Graficar el error en $t = 5$ vs el paso h en escala log-log y ajustar la pendiente con una recta. Verificar que Euler tiene pendiente 1 y RK4 pendiente 4.
   2. Diseñe y complete el siguiente estudio de convergencia de métodos para la EDO $\dot{y} = -y$, $y(0) = 1$, solución exacta $y(t) = e^{-t}$, evaluando el error en $t = 5$:

      ```
      pasos = [0.5, 0.25, 0.1, 0.05, 0.01, 0.005, 0.001]
      for j = 1:length(pasos)
          h = pasos(j)
          // implementar Euler y RK4 y calcular el error
          // en t=5 respecto a exp(-5)
      end
      // graficar log(error) vs log(h)
      // ajustar una recta y obtener la pendiente
      // comparar con el orden teórico
      ```

   3. Cree las gráficas que no hizo en el practico 4.
   4. ¿Qué pendiente obtiene para Euler? ¿Y para RK4?
   5. ¿Coinciden con los órdenes teóricos 1 y 4?
   6. Controle el programa mediante la interfaz web, del practico 8.
   7. Cree las gráficas que no hizo en el practico 4.

      Para cada gráfica: escriba los datos desde el programa, genere el script para graficar y ejecutelo desde consola del lenguaje.

      Verifique que el PNG se crea correctamente antes de seguir.

2. **Ejercicio:** Enfriamiento de Newton

   Corresponde y continua al ejercicio homónimo del práctico 4.

   Corresponde y continua a “Enfriamiento de Newton”

   a) Graficar la solución para distintos pasos h mostrando cuando se vuelve inestable.

   b) Controle el programa mediante la interfaz web, del practico 8.

c) Cree las gráficas que no hizo en el practico 4.
   Para cada gráfica: escriba los datos desde el programa, genere el script para graficar y ejecutelo desde consola del lenguaje.
   Verifique que el PNG se crea correctamente antes de seguir.

3. **Ejercicio:** Modelo depredador–presa de Lotka–Volterra

   Corresponde y continua al ejercicio homónimo del práctico 4.

   Simule el modelo depredador–presa de Lotka–Volterra:

$$\frac{dx}{dt} = ax - bxy, \qquad \frac{dy}{dt} = -cy + dxy$$

   con $a = 1$, $b = 0{,}1$, $c = 1{,}5$, $d = 0{,}075$, $x(0) = 10$ (presas), $y(0) = 5$ (depredadores).

   - Grafique $x(t)$ e $y(t)$ en el mismo gráfico y luego el retrato de fase y vs x.
   - Trayectorias $x(t)$ e $y(t)$ en una subfigura y el retrato de fase y vs x (orbita cerrada) en otra.
   - Controle el programa mediante la interfaz web, del practico 8.
   - Cree las gráficas que no hizo en el practico 4.
     Para cada gráfica: escriba los datos desde el programa, genere el script para graficar y ejecutelo desde consola del lenguaje.
     Verifique que el PNG se crea correctamente antes de seguir.

4. **Ejercicio:** Crecimiento logístico poblacional

   Corresponde y continua el ejercicio homónimo del práctico 4.

   a) Controle el programa mediante la interfaz web, del practico 8.
   b) Cree las gráficas que no hizo en el practico 4.
      Para cada gráfica: escriba los datos desde el programa, genere el script para graficar y ejecutelo desde consola del lenguaje.
      Verifique que el PNG se crea correctamente antes de seguir.

5. **Ejercicio:** Modelo de proyecto integrador

   Desarrolle un programa completo que lea datos experimentales desde un archivo, valide y limpie los registros, realice cálculos, ajuste un modelo, produzca gráficos y guarde un informe de resultados.

   El proyecto deberá incluir como mínimo:

   a) Lectura de uno o más archivos.
   b) Validación de los datos.
   c) Registro de filas aceptadas y rechazadas.
   d) Una estructura de datos apropiada.

e) Al menos cuatro funciones propias.
f) Al menos un algoritmo implementado manualmente.
g) Comparación con una función nativa de un lenguaje estudiado.
h) Cálculo de indicadores estadísticos.
i) Al menos dos gráficos correctamente rotulados.
j) Escritura de un archivo de resultados.
k) Tratamiento de errores.
l) Al menos ocho pruebas automáticas.
m) Medición de rendimiento de una parte relevante.
n) Documentación de las funciones.
ñ) Un archivo README con instrucciones de ejecución.
o) Un repositorio con historial significativo.
p) Ejecución reproducible desde una sesión limpia.

El informe final deberá contener:

- descripción del problema;
- descripción de los datos;
- decisiones de diseño;
- algoritmos empleados;
- resultados;
- análisis de errores;
- pruebas realizadas;
- limitaciones;
- posibles mejoras.

La evaluación del trabajo por parte de los profesors, considerará principalmente:

- corrección;
- claridad;
- modularidad;
- validación;
- pruebas;
- reproducibilidad;
- calidad de la documentación;
- interpretación de los resultados.

Referencias

Abelson, Harold, Gerald Jay Sussman y Julie Sussman (1996). *Structure and Interpretation of Computer Programs*. 2.ª ed. Programacion, abstraccion, interpretes y modelos computacionales. MIT Press. URL: https://mitpress.mit.edu/9780262510875/structure-and-interpretation-of-computer-programs/ (visitado 02-08-2026).

Aho, Alfred V. et al. (2006). *Compilers: Principles, Techniques and Tools*. 2.ª ed. Referencia clásica sobre análisis léxico, análisis sintáctico y descenso recursivo. Los capítulos 3 y 4 cubren exactamente el camino de texto a estructura. Addison-Wesley.

Analog Devices (2019). *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*. Hoja de datos. El sensor 1-Wire que nombra el capítulo. Sirve como ejemplo concreto de sensor digital con protocolo propio y conversión interna. Analog Devices (ex Maxim Integrated). URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf (visitado 14-08-2026).

Apache Software Foundation (2026). *Apache Subversion Documentation*. Referencia de comandos y notas de versión del proyecto. Apache Software Foundation. URL: https://subversion.apache.org/docs/ (visitado 14-08-2026).

Apple Inc. (18 de dic. de 2002). *acl(3) — Introduction to the POSIX.1e ACL Security API*. Página de manual de macOS sobre listas de control de acceso. URL: https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/acl.3.html (visitado 22-08-2026).

Ascher, Uri M. y Linda R. Petzold (1998). *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. EDO, ecuaciones algebraico-diferenciales, rigidez y metodos implicitos. SIAM.

Ashby, W. Ross (1956). *An Introduction to Cybernetics*. Trad. esp.: *Introducción a la cibernética*, Nueva Visión, 1960. London: Chapman & Hall.

Åström, Karl Johan y Tore Hägglund (2006). *Advanced PID Control*. El tratamiento de referencia del PID real: forma discreta, filtrado del término derivativo, anti-windup y saturación del actuador. Es lo que Ogata no cubre. ISA — The Instrumentation, Systems y Automation Society.

Åström, Karl Johan y Richard M. Murray (2021). *Feedback Systems: An Introduction for Scientists and Engineers*. 2.ª ed. Lazo abierto contra lazo cerrado, estabilidad y espacio de estados, con nivel de física e ingeniería general. Disponible en línea. Princeton University Press. URL: https://fbswiki.org/ (visitado 14-08-2026).

Balaban, Gabriel et al. (2021). «Ten Simple Rules for Quick and Dirty Scientific Programming». En: *PLOS Computational Biology* 17.3. Cómo programar rápido sin programar mal, bajo presión de tiempo. Prototipos mínimos, reutilización de código y modularización, e1008549. DOI: 10.1371/journal.pcbi.1008549.

Banzi, Massimo y Michael Shiloh (2014). *Getting Started with Arduino*. 3.ª ed. Del cofundador del proyecto. Nivel introductorio, acorde al «conceptos de uso de Arduino» del programa. Maker Media.

Barth, Adam (abr. de 2011). *HTTP State Management Mechanism*. Inf. téc. RFC 6265. Cookies: el mecanismo con el que se agrega estado a un protocolo que no lo tiene. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc6265 (visitado 13-08-2026).

Bassi, Anna y Manolo Venturin (2026). *Scilab as a Calculator*. URL: http://www.openeering.com/sites/default/files/Scilab_calculator.pdf (visitado 03-08-2026).

Baudin, Michaël (2010). *Scilab is not naive*. Inf. téc. Sobre aritmética de punto flotante en Scilab. Scilab Consortium. URL: https://www.scilab.org/sites/default/files/scilabisnotnaive.pdf (visitado 15-08-2026).

Baudin, Michaël y Stanislav Kroeter (2011). *Programming in Scilab*. URL: https://www.scilab.org/sites/default/files/progscilab-v.0.10_en.pdf.

Beck, Kent (2003). *Test-Driven Development by Example*. Desarrollo guiado por pruebas, con ejemplos completos. Corresponde al ítem «testear, validar» de la unidad 5. Addison-Wesley.

Beck, Kent y Cynthia Andres (2004). *Extreme Programming Explained: Embrace Change*. 2.ª ed. Ciclos cortos, integración continua, programación de a pares. Pensado para equipos chicos, que es la situación del curso. Addison-Wesley.

Beck, Kent, Mike Beedle et al. (2001). *Manifesto for Agile Software Development*. Cuatro valores en una página; sirve como lectura de arranque y como objeto de discusión. URL: https://agilemanifesto.org/.

Berners-Lee, Tim, Roy T. Fielding y Larry Masinter (ene. de 2005). *Uniform Resource Identifier (URI): Generic Syntax*. Inf. téc. RFC 3986. Sintaxis del URI y codificación por porcentaje. Es la norma detrás del *URL encoding* del formulario. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc3986 (visitado 13-08-2026).

Bevington, Philip R. y D. Keith Robinson (2003). *Data Reduction and Error Analysis for the Physical Sciences*. 3.ª ed. Tratamiento estadístico de datos experimentales, con ajuste de curvas. Complementa a Taylor por el lado computacional. McGraw-Hill.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Aprendizaje estadistico, modelos probabilisticos y reconocimiento de patrones. Springer.

Blaze, Matt et al. (2019). *DEF CON 27 Voting Village Report*. tu01.tex. URL: https://media.defcon.org/DEF%5C%20CON%5C%2027/voting-village-report-defcon27.pdf.

Blischak, John D., Emily R. Davenport y Greg Wilson (2016). «A Quick Introduction to Version Control with Git and GitHub». En: *PLOS Computational Biology* 12.1. Introducción breve con tutorial; sirve como lectura previa al trabajo colaborativo del proyecto, e1004668. DOI: 10.1371/journal.pcbi.1004668.

Boettiger, Carl (2015). «An Introduction to Docker for Reproducible Research». En: *ACM SIGOPS Operating Systems Review* 49.1. Contenedores como alternativa al empaquetado nativo cuando el problema es reproducir el entorno, no distribuir el programa, págs. 71-79. DOI: 10.1145/2723872.2723882.

Borland, David y Russell M. Taylor (2007). «Rainbow Color Map (Still) Considered Harmful». En: *IEEE Computer Graphics and Applications* 27.2. Por qué la paleta arcoíris, que es la de uso más extendido en mapas de color y superficies, introduce fronteras que no están en los datos y oculta variaciones que sí están, págs. 14-17. DOI: 10.1109/MCG.2007.323435.

Bourne, Philip E. (2007). «Ten Simple Rules for Making Good Oral Presentations». En: *PLOS Computational Biology* 3.4. Para la defensa del proyecto: hablarle a la audiencia, menos es más, pensar la presentación como un relato, e77. DOI: 10.1371/journal.pcbi.0030077.

Box, G. E. P. y Mervin E. Muller (1958). «A Note on the Generation of Random Normal Deviates». En: *The Annals of Mathematical Statistics* 29.2. Referencia original de la transformacion de Box–Muller usada para generar normales a partir de uniformes, págs. 610-611. DOI: 10.1214/aoms/1177706645.

Bray, Tim (dic. de 2017). *The JavaScript Object Notation (JSON) Data Interchange Format*. Inf. téc. RFC 8259. Formato de intercambio entre el navegador y el servidor. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc8259 (visitado 13-08-2026).

Bridgman, Percy Williams (1927). *The Logic of Modern Physics*. Operacionalismo y definicion de conceptos por operaciones de medicion. Macmillan.

Brooks, Frederick P. (1995). *The Mythical Man-Month: Essays on Software Engineering*. 2.ª ed. Edición del vigésimo aniversario, incluye «No Silver Bullet». Por qué agregar gente a un proyecto atrasado lo atrasa más: conviene leerlo antes de armar los grupos. Addison-Wesley.

Brown, Tom B. et al. (2020). «Language Models are Few-Shot Learners». En: *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. GPT-3: modelos de lenguaje de gran tamaño y aprendizaje con pocos ejemplos. URL: https://arxiv.org/abs/2005.14165.

Bryant, Randal E. y David R. O'Hallaron (2016). *Computer Systems: A Programmer's Perspective*. 3.ª ed. Representacion de datos, memoria, ensamblador, procesos, E/S y redes desde la perspectiva del programador. Pearson.

Bunge, Mario (1967). *Scientific Research: Strategy and Philosophy*. Realismo cientifico, modelos, metodo y filosofia de la ciencia. Springer.

Burden, Richard L. y J. Douglas Faires (2011). *Numerical Analysis*. 9.ª ed. Metodos numericos generales; raices, interpolacion, integracion, sistemas y EDO. Brooks/Cole.

Bureau International des Poids et Mesures (2019). *The International System of Units (SI)*. 9.ª ed. Definición actual de las unidades a partir de constantes fundamentales. Pertinente al ítem «magnitud física». BIPM. URL: https://www.bipm.org/en/publications/si-brochure.

Busaniche, Beatriz (2014). *Voto electrónico: los riesgos de una ilusión*. tu01.tex.

Callen, Herbert B. (1985). *Thermodynamics and an Introduction to Thermostatistics*. 2.ª ed. New York: John Wiley & Sons. ISBN: 978-0-471-86256-7.

Calvo Rolle, José Luis (2003). *Scilab: Simulación y programación*. Separada de la entrada compuesta Baudin. Ra-Ma.

Campbell, Stephen L., Jean-Philippe Chancelier y Ramine Nikoukhah (2006). *Modeling and Simulation in Scilab/Scicos*. Simulación en Scilab con interfaz gráfica y por bloques: conecta la GUI nativa con Xcos. Springer. DOI: 10.1007/0-387-30486-X.

Carnap, Rudolf (1937). *The Logical Syntax of Language*. Reconstruccion logica del lenguaje cientifico. Kegan Paul.

Cartwright, Nancy (1983). *How the Laws of Physics Lie*. Leyes, modelos, idealizaciones y explicacion cientifica. Oxford University Press.

Cellier, Francois E. y Ernesto Kofman (2006). *Continuous System Simulation*. Simulacion de sistemas continuos, modelado y metodos numericos. Springer.

Chacon, Scott y Ben Straub (2014). *Pro Git*. 2.ª ed. Libre y en línea, con traducción al castellano. Referencia completa de control de versiones para el trabajo colaborativo. Apress. URL: https://git-scm.com/book/.

Chapra, Steven C. (2007). *Métodos numéricos para ingenieros*. Separada de la entrada compuesta Baudin. McGraw-Hill.

Chapra, Steven C. y Raymond P. Canale (2015). *Numerical Methods for Engineers*. 7.ª ed. Metodos numericos aplicados a ingenieria. McGraw-Hill Education.

Chen, Mark et al. (2021). *Evaluating Large Language Models Trained on Code*. Codex: modelos de lenguaje para generar código, y sus limitaciones. URL: https://arxiv.org/abs/2107.03374.

Christiansen, Tom et al. (2012). *Programming Perl*. 4.ª ed. Edición actualizada de la referencia del lenguaje: contexto escalar y de lista, referencias, autovivificación y clausuras. O'Reilly.

Chua, Leon O., Charles A. Desoer y Ernest S. Kuh (1987). *Linear and Nonlinear Circuits*. Teoria de circuitos lineales y no lineales. McGraw-Hill.

Cleveland, William S. (1994). *The Elements of Graphing Data*. 2.ª ed. Reglas concretas para ejes, escalas, relación de aspecto y superposición de curvas. Es el complemento operativo de las buenas prácticas que enumera el capítulo. Hobart Press.

Cleveland, William S. y Robert McGill (1984). «Graphical Perception: Theory, Experimentation and Application to the Development of Graphical Methods». En: *Journal of the American Statistical Association* 79.387. Evidencia experimental de qué codificaciones gráficas se leen con menos error: posición mejor que longitud, longitud mejor que área o color. Justifica por qué ciertas decisiones de gráfico no son cuestión de gusto, págs. 531-554. DOI: 10.1080/01621459.1984.10478080.

Collins-Sussman, Ben, Brian W. Fitzpatrick y C. Michael Pilato (2011). *Version Control with Subversion*. El libro de referencia de Subversion, disponible en línea y en castellano. Cubre el modelo centralizado, las propiedades y el uso de ramas por copia. O'Reilly. URL: https://svnbook.red-bean.com/ (visitado 14-08-2026).

Condorí, Miguel, Mariela Finetti y Luis Saravia (2011). *Introducción al uso del Scilab: Lenguaje libre para la programación científica*. Apuntes de clase. Se tomaron textos y ejemplos de este material.

Cooper, Mendel (2014). *Advanced Bash-Scripting Guide*. Guia de scripting Bash. URL: https://tldp.org/LDP/abs/html/ (visitado 02-08-2026).

Crameri, Fabio, Grace E. Shephard y Philip J. Heron (2020). «The Misuse of Colour in Science Communication». En: *Nature Communications* 11. Revisión del problema con paletas alternativas listas para usar, uniformes en percepción y legibles con daltonismo y en escala de grises. Se aplican directamente a colormap() en Scilab, pág. 5444. DOI: 10.1038/s41467-020-19160-7. URL: https://www.fabiocrameri.ch/colourmaps/ (visitado 14-08-2026).

Crockford, Douglas (2008a). *JavaScript: The Good Parts*. Sebastopol, CA: O'Reilly. ISBN: 978-0-596-51774-8.

– (2008b). *JavaScript: The Good Parts*. Argumenta qué subconjunto del lenguaje conviene usar y cuál evitar. Buen caso de estudio de un lenguaje con decisiones de diseño heredadas que no se pueden revertir por compatibilidad. O'Reilly.

Dabek, Frank et al. (2002). «Event-Driven Programming for Robust Software». En: *Proceedings of the 10th ACM SIGOPS European Workshop*. Por qué el bucle de eventos, y no los hilos, es el esquema natural de una interfaz. ACM, págs. 186-189. DOI: 10.1145/1133373.1133410.

Debian Project (2026). *Debian Policy Manual*. Politicas del sistema Debian; paquetes, archivos, scripts y estructura del sistema. URL: https://www.debian.org/doc/debian-policy/ (visitado 02-08-2026).

DeVoe, Howard (2020). *Thermodynamics and Chemistry*. 2.ª ed. Libro de acceso libre. URL: https://www2.chem.umd.edu/thermobook/ (visitado 15-08-2026).

Dewey, John (1938). *Logic: The Theory of Inquiry*. Trad. esp.: *Lógica: teoría de la investigación*, Fondo de Cultura Económica, 1950. New York: Henry Holt.

Downey, Allen B. (2015). *Think Python: How to Think Like a Computer Scientist*. 2.ª ed. Introduccion a programacion con Python. O'Reilly Media. URL: https://greenteapress.com/wp/think-python-2e/ (visitado 02-08-2026).

Duhem, Pierre (1954). *The Aim and Structure of Physical Theory*. Traduccion inglesa de la obra francesa de 1914; teorias fisicas, holismo y representacion. Princeton University Press.

Eaton, John W. (2001). «Octave: Past, Present, and Future». En: *Proceedings of the 2nd International Workshop on Distributed Statistical Computing (DSC 2001)*. Ed. por Kurt Hornik y Friedrich Leisch. Historia y diseño de GNU Octave por su autor. Vienna. URL: https://www.r-project.org/conferences/DSC-2001/Proceedings/Eaton.pdf (visitado 16-08-2026).

Eaton, John W. et al. (2024). *GNU Octave: A high-level interactive language for numerical computations*. Edition 9 for Octave version 9.2.0. Manual oficial de GNU Octave. Free Software Foundation. URL: https://docs.octave.org/latest/ (visitado 16-08-2026).

Eby, Phillip J. (2010). *PEP 3333: Python Web Server Gateway Interface v1.0.1*. WSGI: la interfaz que reemplazó al CGI en Python, sin lanzar un proceso por solicitud. Python Software Foundation. URL: https://peps.python.org/pep-3333/ (visitado 13-08-2026).

Echevarría Líbano, Rosa (2013). *Una introducción rápida a Scilab*. URL: http://personal.us.es/echevarria/documentos/ApuntesScilab.pdf (visitado 03-08-2026).

Ecma International (2025). *ECMAScript 2025 Language Specification*. ECMA-262, 16th edition. La norma del lenguaje. JavaScript es el nombre corriente de una implementación de ECMAScript. Ecma International. Geneva. URL: https://ecma-international.org/publications-and-standards/standards/ecma-262/ (visitado 15-08-2026). Texto actualizado de la especificación: https://tc39.es/ecma262/.

Eisberg, Robert y Lawrence Lerner (1981). *Physics*. McGraw-Hill.

Electron contributors (2026). *Electron Documentation*. Documentacion oficial de Electron. Consultada el 2026-08-23. OpenJS Foundation.

Espressif Systems (2026). *ESP32 Technical Reference Manual*. Manual del fabricante: periféricos, ADC, temporizadores, comunicación. Espressif Systems. URL: https://www.espressif.com/en/support/documents/technical-documents.

Feigenbaum, Edward A. (1977). «The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering». En: *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. William Kaufmann, págs. 1014-1029.

Feldman, Ariel J., J. Alex Halderman y Edward W. Felten (2007). «Security analysis of the Diebold AccuVote-TS voting machine». En: *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology (EVT '07)*. tu01.tex. USENIX. URL: https://citp.princeton.edu/our-work/voting/.

Feldman, Stuart I. (1979). «Make — A Program for Maintaining Computer Programs». En: *Software: Practice and Experience* 9.4. El artículo original: describir las dependen-

cias entre archivos y dejar que la herramienta decida qué hay que rehacer. Es una forma declarativa de especificar un cálculo, págs. 255-265. DOI: 10.1002/spe.4380090402.

Fermi, Enrico (1937). *Thermodynamics*. Reimpresión Dover, 1956. New York: Prentice-Hall.

Feynman, Richard (1963). *The Feynman Lectures on Physics*. California Institute of Technology. URL: https://www.feynmanlectures.caltech.edu/I_09.html.

Fielding, R. T (2000). *Architectural styles and the design of network-based software architectures [Tesis doctoral, University of California, Irvine]*. tu02.tex. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm.

Fielding, Roy T. et al. (2022). *RFCs 9110–9114: HTTP semantics, HTTP/1.1, HTTP/2 y HTTP/3*. Inf. téc. RFC 9110. Semantica actual de HTTP, tu02.tex. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc9110 (visitado 02-08-2026).

Flanagan, David (2020). *JavaScript: The Definitive Guide*. 7.ª ed. Modelo de eventos del navegador: registro de manejadores, propagación, DOM. Es el lado cliente del formulario. O'Reilly.

Foley, James D. et al. (1990). *Computer Graphics: Principles and Practice*. 2.ª ed. Escena retenida: la figura como conjunto de objetos con propiedades modificables, que es lo que hacen posible los handles y set() de Scilab. Addison-Wesley.

Fowler, Martin (2018). *Refactoring: Improving the Design of Existing Code*. 2.ª ed. Qué hacer cuando el prototipo del proyecto empieza a crecer y hay que reorganizarlo sin romperlo. Addison-Wesley.

Fraassen, Bas C. van (1980). *The Scientific Image*. Empirismo constructivo y adecuacion empirica. Oxford University Press.

Fraden, Jacob (2016). *Handbook of Modern Sensors: Physics, Designs and Applications*. 5.ª ed. Catálogo razonado de sensores por principio físico: temperatura, presión, posición, luz, humedad. Springer.

freedesktop.org (2026). *xdg-utils*. freedesktop.org. URL: https://wiki.freedesktop.org/www/Software/xdg-utils/ (visitado 22-08-2026).

Freedman, David y Persi Diaconis (1981). «On the Histogram as a Density Estimator: L2 Theory». En: *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 57.4. Cómo elegir el ancho de clase de un histograma con un criterio y no a ojo. El ancho de la caja cambia la forma de la distribución que se ve, págs. 453-476. DOI: 10.1007/BF01025868.

Friedl, Jeffrey E. F. (2006). *Mastering Regular Expressions*. 3.ª ed. Tratamiento práctico y exhaustivo: cómo funciona el motor por retroceso, por qué un patrón mal escrito puede tardar un tiempo exponencial, y las diferencias entre dialectos. O'Reilly.

Gadamer, Hans-Georg (1975). *Truth and Method*. Hermeneutica filosofica; traduccion inglesa de Wahrheit und Methode, 1960. Continuum.

Galassi, Mark et al. (2009). *GNU Scientific Library Reference Manual*. 3.ª ed. Integración, raíces, derivadas, interpolación, Monte Carlo y EDO usados desde PDL y Math::GSL. Network Theory Ltd. ISBN: 978-0-9546120-7-8. URL: https://www.gnu.org/software/gsl/doc/html/ (visitado 16-08-2026).

Gamma, Erich et al. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. El patrón Observador es la formulación canónica del mecanismo de eventos y callbacks; el patrón Comando, la de las acciones asociadas a un control. Addison-Wesley.

Giere, Ronald N. (1988). *Explaining Science: A Cognitive Approach*. Modelos, representacion y practica cientifica. University of Chicago Press.

Git Project (2026). *Git Reference Manual*. Referencia completa de comandos, opciones y formatos de configuración. Software Freedom Conservancy. URL: https://git-scm.com/docs (visitado 14-08-2026).

Glazebrook, Karl y Frossie Economou (sep. de 1997). «PDL: The Perl Data Language». En: *Dr. Dobb's Journal*. Special issue on Perl.

GNOME Project (2026). *Document Viewer Help*. GNOME Project. URL: https://help.gnome.org/evince/ (visitado 22-08-2026).

– (s.f.). *Zenity Manual*. Documentacion de Zenity para dialogos desde la linea de comandos y scripts. Consultada el 2026-08-23. GNOME Project.

GNU nano Project (2026). *GNU nano Documentation*. GNU nano Project. URL: https://www.nano-editor.org/docs.php (visitado 22-08-2026).

gnuplot development team (2025). *gnuplot 6.0: An Interactive Plotting Program*. URL: https://gnuplot.sourceforge.net/docs_6.0/gnuplot.pdf.

– (2026). *gnuplot Demos*. URL: https://gnuplot.sourceforge.net/demo/.

gnuplot Team (2026). *gnuplot Documentation*. Documentacion oficial de gnuplot. URL: https://gnuplot.sourceforge.net/documentation.html (visitado 02-08-2026).

Goldberg, David (1991a). «What Every Computer Scientist Should Know About Floating-Point Arithmetic». En: *ACM Computing Surveys* 23.1, págs. 5-48. DOI: 10.1145/103162.103163.

– (1991b). «What Every Computer Scientist Should Know About Floating-Point Arithmetic». En: *ACM Computing Surveys* 23.1. Representación, redondeo, cancelación catastrófica y por qué comparar reales por igualdad es un error. Vale para todos los lenguajes de la unidad, pero es en C donde la representación queda a la vista, págs. 5-48. DOI: 10.1145/103162.103163.

Golub, Gene H. y Charles F. Van Loan (2013). *Matrix Computations*. 4.ª ed. Computacion matricial y algebra lineal numerica. Johns Hopkins University Press.

Gomez, Christine (2003). *Scilab for Beginners: Tutorial*. tu04b.tex. URL: https://www.scilab.org/scilab-for-beginners-tutorial (visitado 03-08-2026).

Goodfellow, Ian, Yoshua Bengio y Aaron Courville (2016). *Deep Learning*. Libro en linea sobre aprendizaje profundo. MIT Press. URL: https://www.deeplearningbook.org/ (visitado 02-08-2026).

Google (2024). *Shell Style Guide*. Convenciones de escritura y, sobre todo, el criterio para decidir cuándo un guión de shell dejó de ser la herramienta adecuada. URL: https://google.github.io/styleguide/shellguide.html (visitado 14-08-2026).

Gould, Harvey, Jan Tobochnik y Wolfgang Christian (2007). *An Introduction to Computer Simulation Methods: Applications to Physical Systems*. 3.ª ed. Simulación de sistemas físicos con animación: el uso de la figura que se actualiza como parte del método, no como adorno. Enlaza con la unidad de simulación. Addison-Wesley.

Gourley, D. y B Totty (2002). *HTTP: The definitive guide*. tu02.tex. O'Reilly.

Goyal, Rajan y Mansi Dhingra (2019). *Programming in Scilab*. Edición en tapa dura. Edición en rústica: Narosa Publishing House, 2019, ISBN 9788184876475. Alpha Science International. ISBN: 9781783324019.

Granger, Brian E. y Fernando Pérez (2021). «Jupyter: Thinking and Storytelling with Code and Data». En: *Computing in Science & Engineering* 23.2. La tesis de los propios autores: el cuaderno resuelve un problema humano, no técnico. Es el respaldo directo de la afirmación sobre la innovación epistemológica, págs. 7-14. DOI: 10.1109/MCSE.2021.3059263.

Hairer, Ernst, Syvert P. Norsett y Gerhard Wanner (1993). *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2.ª ed. Metodos numericos para ecuaciones diferenciales ordinarias no rigidas. Springer.

Hall, Brian (2026). *Beej's Guide to Network Programming*. Programacion de sockets en C. URL: https://beej.us/guide/bgnet/ (visitado 02-08-2026).

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau y Travis E. Oliphant (2020). «Array Programming with NumPy». En: *Nature* 585.7825. El arreglo n-dimensional sobre el que se apoya el ejemplo del cuaderno. Equivalente en Python a las matrices de Scilab, págs. 357-362. DOI: 10.1038/s41586-020-2649-2.

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser et al. (2020). «Array programming with NumPy». En: *Nature* 585.7825, págs. 357-362. DOI: 10.1038/s41586-020-2649-2.

Harrower, Mark y Cynthia A. Brewer (2003). «ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps». En: *The Cartographic Journal* 40.1. Paletas secuenciales, divergentes y cualitativas probadas para impresión en blanco y negro y para visión deficiente del color. Resuelve el ítem de daltonismo de las buenas prácticas, págs. 27-37. DOI: 10.1179/000870403235002042. URL: https://colorbrewer2.org/ (visitado 14-08-2026).

Heer, Jeffrey y George Robertson (2007). «Animated Transitions in Statistical Data Graphics». En: *IEEE Transactions on Visualization and Computer Graphics* 13.6. Cuándo la transición animada sí ayuda: continuidad del objeto entre cuadros, que es exactamente lo que logra actualizar xdata/ydata en vez de redibujar, págs. 1240-1247. DOI: 10.1109/TVCG.2007.70539.

Hempel, Carl G. (1965). *Aspects of Scientific Explanation*. Explicacion cientifica, leyes y confirmacion. Free Press.

Hertzog, R. y R Mas (2026). *The Debian administrator's handbook*. tu01.tex. Freexian. URL: https://debian-handbook.info/ (visitado 19-07-2026).

Higham, Nicholas J. (2002). *Accuracy and Stability of Numerical Algorithms*. 2.ª ed. Error numerico, estabilidad y condicionamiento. SIAM.

Holen, Vidar (2026). *ShellCheck: A Shell Script Static Analysis Tool*. Detecta los errores clásicos del shell, sobre todo las variables sin comillas. Es la herramienta que conviene correr sobre todo guión antes de usarlo. URL: https://www.shellcheck.net/ (visitado 14-08-2026).

Hopcroft, John E., Rajeev Motwani y Jeffrey D. Ullman (2006). *Introduction to Automata Theory, Languages and Computation*. 3.ª ed. Por qué una expresión regular no puede reconocer paréntesis anidados: la jerarquía de lenguajes formales y qué autómata corresponde a cada nivel. Pearson.

Horowitz, Paul y Winfield Hill (2015). *The Art of Electronics*. 3.ª ed. Referencia general de electrónica práctica; capítulos sobre conversión analógico-digital y acondicionamiento de señal. Cambridge University Press.

Hughes, Austin y Bill Drury (2019). *Electric Motors and Drives: Fundamentals, Types and Applications*. 5.ª ed. Motor de continua, servo, paso a paso y su control: cubre la sección de actuadores, que hoy no tiene ninguna fuente. Newnes.

Hunt, Andrew y David Thomas (2019). *The Pragmatic Programmer: Your Journey to Mastery*. 2.ª ed. Edición del vigésimo aniversario; la primera es de 1999. Consejos sueltos y aplicables sin adoptar ninguna metodología. Addison-Wesley.

Hunt, Doug, Rafael Laboissiere y David Mertens (2026). *PDL::Graphics::PLplot: Object-oriented interface from Perl/PDL to the PLplot plotting library*. Documentacion de la interfaz grafica PLplot para PDL. URL: `https://metacpan.org/pod/PDL::Graphics::PLplot` (visitado 22-08-2026).

Hunter, John D. (2007a). «Matplotlib: A 2D Graphics Environment». En: *Computing in Science & Engineering* 9.3. La biblioteca de gráficos del ejemplo del cuaderno. Apareció en el mismo número que el artículo de IPython, págs. 90-95. DOI: `10.1109/MCSE.2007.55`.

– (2007b). «Matplotlib: A 2D graphics environment». En: *Computing in Science & Engineering* 9.3, págs. 90-95. DOI: `10.1109/MCSE.2007.55`.

Hutchins, Edwin L., James D. Hollan y Donald A. Norman (1985). «Direct Manipulation Interfaces». En: *Human–Computer Interaction* 1.4. Distancia entre la intención del usuario y la acción que la interfaz ofrece. Marco conceptual del widget, págs. 311-338. DOI: `10.1207/s15327051hci0104_2`.

IEEE (2019). *IEEE Standard for Floating-Point Arithmetic*. IEEE 754-2019. La norma que fija cómo se representan y se operan los números reales en las máquinas actuales, y que todos los lenguajes de la unidad heredan. IEEE. DOI: `10.1109/IEEESTD.2019.8766229`.

IEEE and The Open Group (2018). *IEEE Std 1003.1-2017 – Standard for Information Technology – Portable Operating System Interface (POSIX)*. Inf. téc. IEEE. URL: `https://pubs.opengroup.org/onlinepubs/9699919799/` (visitado 15-08-2026).

ImageMagick Studio LLC (2026a). *ImageMagick Command-Line Tools: Display*. URL: `https://imagemagick.org/display/`.

– (2026b). *ImageMagick: Command-line Processing*. Documentacion general de las herramientas de linea de comandos de ImageMagick, incluidas conversion y procesamiento de secuencias de imagenes. URL: `https://imagemagick.org/command-line-processing/` (visitado 22-08-2026).

Ing-Simmons, Nick (2021). *Tk – a Graphical User Interface Toolkit for Perl*. Documentacion de Perl/Tk. Consultada el 2026-08-23. MetaCPAN.

INRIA (2026). *Scilab: Sitio del proyecto*. URL: `https://www.scilab.org/` (visitado 03-08-2026).

International Organization for Standardization (2004). *Information Technology — Computer Graphics and Image Processing — Portable Network Graphics (PNG): Functional Specification*. ISO/IEC 15948:2004. Especificación del formato rasterizado que usa el capítulo: compresión sin pérdida, canal alfa, profundidad de color. ISO/IEC. URL: `https://www.w3.org/TR/png/` (visitado 14-08-2026).

– (2017). *General Requirements for the Competence of Testing and Calibration Laboratories*. ISO/IEC 17025:2017. Qué se le exige a un laboratorio para que su certificado de calibración valga: trazabilidad documentada, intervalos y competencia técnica. ISO.

– (2018). *Information Technology — Programming Languages — C*. ISO/IEC 9899:2018. La norma del lenguaje. Define además qué es comportamiento indefinido, que es la categoría que explica por qué un programa en C puede hacer cualquier cosa cuando se sale de las reglas. ISO/IEC.

– (2020). *Document Management — Portable Document Format — Part 2: PDF 2.0*. ISO 32000-2:2020. El formato vectorial de salida para incluir figuras en LaTeX con includegraphics, y el que preserva el texto del gráfico como texto. ISO.

International Telecommunication Union (1992). *Information Technology — Digital Compression and Coding of Continuous-Tone Still Images: Requirements and Guidelines*. ITU-T T.81 (ISO/IEC 10918-1). El formato con pérdida: transformada coseno discreta

y cuantización. Explica por qué cada guardado degrada la imagen y por qué no sirve para gráficos con líneas. ITU-T. URL: https://www.w3.org/Graphics/JPEG/itu-t81.pdf (visitado 14-08-2026).

ISO/IEC JTC 1/SC 22 (2018). *ISO/IEC 9899:2018 – Information technology – Programming languages – C*. Inf. téc. Geneva: International Organization for Standardization.

Ji, Ziwei et al. (2023). «Survey of Hallucination in Natural Language Generation». En: *ACM Computing Surveys* 55.12. Alucinaciones: por qué los modelos inventan y cómo se detecta, págs. 1-38. DOI: 10.1145/3571730.

Joint Committee for Guides in Metrology (2008). *Evaluation of Measurement Data: Guide to the Expression of Uncertainty in Measurement*. JCGM 100:2008. La GUM: el documento normativo sobre incertidumbre de medida. Hay traducción al castellano del CEM. BIPM. URL: https://www.bipm.org/en/committees/jc/jcgm/publications.

– (2012). *International Vocabulary of Metrology: Basic and General Concepts and Associated Terms (VIM)*. 3.ª ed. JCGM 200:2012. Define con precisión calibración, trazabilidad, exactitud, veracidad, error e incertidumbre. Es el complemento del GUM para el vocabulario del capítulo. BIPM. URL: https://www.bipm.org/en/committees/jc/jcgm/publications (visitado 14-08-2026).

Jones, Michael B., John Bradley y Nat Sakimura (mayo de 2015). *JSON Web Token (JWT)*. Inf. téc. RFC 7519. Tokens firmados: cómo se mantiene una sesión sin guardar estado en el servidor. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc7519 (visitado 13-08-2026).

Jong, Jos de y math.js contributors (2026). *math.js: an extensive math library for JavaScript and Node.js*. URL: https://mathjs.org/docs/ (visitado 15-08-2026).

Josefsson, Simon (oct. de 2006). *The Base16, Base32, and Base64 Data Encodings*. Inf. téc. RFC 4648. Codificación base64, usada para incrustar las gráficas PNG en la respuesta HTML. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc4648 (visitado 13-08-2026).

Jupyter Widgets Development Team (2026). *Jupyter Widgets Documentation*. El equivalente a uicontrol dentro del cuaderno: sliders, botones y campos ligados a callbacks. Hace comparable la GUI nativa de Scilab con la de Jupyter. URL: https://ipywidgets.readthedocs.io/ (visitado 13-08-2026).

Karnopp, Dean C., Donald L. Margolis y Ronald C. Rosenberg (2012). *System Dynamics: Modeling, Simulation, and Control of Mechatronic Systems*. 5.ª ed. Dinamica de sistemas, simulacion, control y bond graphs. Wiley.

Kernighan, Brian W. y Rob Pike (1999). *The Practice of Programming*. Programas chicos y bien hechos: estilo, depuración, pruebas, portabilidad y herramientas. Es el más cercano al tamaño de un práctico integrador. Addison-Wesley.

Kernighan, Brian W. y Dennis M. Ritchie (1978). *The C Programming Language*. Prentice Hall.

– (1988). *The C Programming Language*. 2.ª ed. Englewood Cliffs, NJ: Prentice Hall. ISBN: 978-0-13-110362-7.

Kester, Walt, ed. (2005). *The Data Conversion Handbook*. Conversión A/D y D/A en detalle: arquitecturas, resolución, errores de cuantización, muestreo. Disponible libre en el sitio de Analog Devices. Analog Devices / Newnes.

Khalil, Hassan K. (2002). *Nonlinear Systems*. 3.ª ed. Sistemas dinamicos no lineales y estabilidad. Prentice Hall.

King, K. N. (2008). *C Programming: A Modern Approach*. 2.ª ed. Curso completo del lenguaje con ejercicios, apoyado en la norma vigente. Más didáctico que el libro original de Kernighan y Ritchie para quien empieza. W. W. Norton.

Kleene, Stephen C. (1956). «Representation of Events in Nerve Nets and Finite Automata». En: *Automata Studies*. Ed. por Claude E. Shannon y John McCarthy. El origen de las expresiones regulares: la equivalencia entre una notación de patrones y un autómata finito. Princeton University Press, págs. 3-41.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez et al. (2016a). «Jupyter Notebooks—A Publishing Format for Reproducible Computational Workflows». En: *Positioning and Power in Academic Publishing*. IOS Press, págs. 87-90. DOI: 10.3233/978-1-61499-649-1-87.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez et al. (2016b). «Jupyter Notebooks – a publishing format for reproducible computational workflows». En: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. por Fernando Loizides y Birgit Schmidt. IOS Press, págs. 87-90. DOI: 10.3233/978-1-61499-649-1-87.

Knuth, D. E (1984). *Literate programming. The Computer Journal, 27(2), 97–111*. tu03.tex. URL: https://doi.org/10.1093/comjnl/27.2.97.

Knuth, Donald E. (2022). *The Art of Computer Programming*. 5 vols. Estuche con los volúmenes 1–4B. Vol. 1 *Fundamental Algorithms* (3ª ed.), vol. 2 *Seminumerical Algorithms* (3ª ed.), vol. 3 *Sorting and Searching* (2ª ed.), vol. 4A y 4B *Combinatorial Algorithms*, partes 1 y 2. Addison-Wesley Professional. ISBN: 9780137935109.

Kowalski, Robert (1979). *Logic for Problem Solving*. North-Holland.

Krasner, Glenn E. y Stephen T. Pope (1988). «A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80». En: *Journal of Object-Oriented Programming* 1.3. MVC: separación entre el estado simulado, su representación gráfica y los controles que lo cambian. Estructura implícita del ejemplo masa-resorte, págs. 26-49.

Kuhn, Thomas S. (1962). *The Structure of Scientific Revolutions*. Paradigmas, ciencia normal y revoluciones cientificas. University of Chicago Press.

Lamport, Leslie (1994). *LaTeX: A document preparation system*. 2.ª ed. tu03.tex. Addison-Wesley.

LeCun, Yann, Yoshua Bengio y Geoffrey Hinton (2015). «Deep Learning». En: *Nature* 521.7553, págs. 436-444. DOI: 10.1038/nature14539.

Lee, Benjamin D. (2018). «Ten Simple Rules for Documenting Scientific Software». En: *PLOS Computational Biology* 14.12. README, ejemplos de uso, documentación de la interfaz. Es lo que falta cuando el proyecto se entrega y nadie sabe cómo correrlo, e1006561. DOI: 10.1371/journal.pcbi.1006561.

Lenin, Vladimir Ilyich (1909). *Materialism and Empirio-Criticism*. Materialismo filosofico, critica del empiriocriticismo y teoria del conocimiento. Zveno Publishers. URL: https://www.marxists.org/archive/lenin/works/1908/mec/ (visitado 02-08-2026).

Lessig, L (2006). *Code: Version 2.0*. tu01.tex. Basic Books. URL: http://codev2.cc/.

Leto, Jonathan Duke y Thierry Ju (2026). *Math::GSL: Perl interface to the GNU Scientific Library*. Incluye Math::GSL::ODEIV, Roots, Integration. CPAN. URL: https://metacpan.org/pod/Math::GSL (visitado 16-08-2026).

Libre, Fundación Vía (2015). *Informe técnico sobre el sistema Vot.ar utilizado en la Ciudad Autónoma de Buenos Aires*. tu01.tex. URL: https://www.vialibre.org.ar/.

Liechti, Chris (2026). *pySerial Documentation*. Biblioteca estándar de facto para leer el puerto serie desde Python, incluida la lectura del Arduino. URL: https://pyserial.readthedocs.io/ (visitado 14-08-2026).

Linux Foundation (2015). *Filesystem Hierarchy Standard*. Estructura estandar de directorios en sistemas tipo Unix/Linux. URL: https://refspecs.linuxfoundation.org/fhs.shtml (visitado 02-08-2026).

List, Markus, Peter Ebert y Felipe Albrecht (2017). «Ten Simple Rules for Developing Usable Software in Computational Biology». En: *PLOS Computational Biology* 13.1. Del prototipo a algo que otro pueda usar: interfaz, instalación, documentación mínima, e1005265. DOI: 10.1371/journal.pcbi.1005265.

Martin, Robert C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Nombres, funciones, comentarios y estructura. Cubre el ítem «escribir código legible» de la unidad 5. Prentice Hall.

Marx, Karl (1845). *Theses on Feuerbach*. Practica, transformacion del mundo y critica del materialismo contemplativo. URL: https://www.marxists.org/archive/marx/works/1845/theses/ (visitado 02-08-2026).

Masinter, Larry (jul. de 2015). *Returning Values from Forms: multipart/form-data*. Inf. téc. RFC 7578. La otra codificación del formulario, la que se usa cuando hay campos de archivo. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc7578 (visitado 13-08-2026).

McCarthy, John (1960). «Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I». En: *Communications of the ACM* 3.4, págs. 184-195. DOI: 10.1145/367177.367199.

McConnell, Steve (1996). *Rapid Development: Taming Wild Software Schedules*. Cómo terminar a tiempo: estimación, recorte de alcance, entregas por etapas y los errores clásicos que hunden un proyecto corto. Microsoft Press.

McKinney, Wes (2022). *Python for Data Analysis*. 3.ª ed. Tablas con columnas heterogéneas, series temporales y datos faltantes: la estructura que el arreglo numérico no cubre. O'Reilly.

MDN contributors (2026). *JavaScript reference*. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (visitado 15-08-2026).

Mertens, David (2026). *PDL::Graphics::Prima: an interactive plotting widget and library for PDL and Prima*. Documentacion del sistema de graficos interactivos de PDL basado en Prima. URL: https://metacpan.org/pod/PDL::Graphics::Prima (visitado 22-08-2026).

Miao, Fengchun y Wayne Holmes (2023). *Guidance for generative AI in education and research*. Inf. téc. Hay versión en español. Paris: UNESCO. DOI: 10.54675/EWZM9535. URL: https://unesdoc.unesco.org/ark:/48223/pf0000386693.

Microsoft (2022). *Creating a Graphical Date Picker – PowerShell*. Ejemplo oficial de uso de System.Windows.Forms desde PowerShell. Consultada el 2026-08-23. Microsoft Learn.

– (9 de jun. de 2025). *icacls*. Documentación oficial del comando para mostrar y modificar listas de control de acceso discrecional en Windows. URL: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/icacls (visitado 22-08-2026).

Mitchell, Tom M. (1997). *Machine Learning*. Aprendizaje automatico clasico; hipotesis, generalizacion y algoritmos. McGraw-Hill.

Modelica Association (2026). *Modelica Language Specification*. Lenguaje de modelado fisico orientado a ecuaciones. URL: https://specification.modelica.org/ (visitado 02-08-2026).

Mohan, Ned, Tore M. Undeland y William P. Robbins (2003). *Power Electronics: Converters, Applications and Design*. 3.ª ed. El lado de potencia del actuador: conmutación con MOSFET, PWM y manejo de cargas inductivas como relés y motores. Wiley.

Mora Escobar, Héctor Manuel (2001). *Introducción al Scilab 8*. Material docente. URL: http://mmc.geofisica.unam.mx/acl/anum/Ejemplitos/SciLab/Intro_Spanish.pdf.

Moreland, Kenneth (2009). «Diverging Color Maps for Scientific Visualization». En: *Advances in Visual Computing (ISVC 2009)*. Vol. 5876. Lecture Notes in Computer Science. Cómo se construye un mapa de color perceptualmente uniforme y por qué el punto medio importa cuando el cero tiene significado. Es el criterio detrás de elegir bluered en vez de hotcolormap. Springer, págs. 92-103. DOI: 10.1007/978-3-642-10520-3_9.

Morin, Andrew, Jennifer Urban y Piotr Sliz (2012). «A Quick Guide to Software Licensing for the Scientist-Programmer». En: *PLOS Computational Biology* 8.7. Elegir licencia: copyleft, permisivas, compatibilidad. Complemento práctico de los textos de software libre de la unidad 1, e1002598. DOI: 10.1371/journal.pcbi.1002598.

Murdock, I. A (jul. de 2007). *How package management changed everything [Blog]. ianmurdock.com*. tu01.tex. URL: https://ianmurdock.debian.net/index.html%5C%3Fp=437.html.

Myers, Brad A. (1998). «A Brief History of Human-Computer Interaction Technology». En: *interactions* 5.2. De dónde salen la ventana, el widget y el evento: contexto histórico de la GUI que construye la unidad, págs. 44-54. DOI: 10.1145/274430.274436.

Nakamura, Shoichiro (1997). *Análisis numérico y visualización gráfica con MATLAB*. 1.ª ed. Traducción de *Numerical Analysis and Graphic Visualization with MATLAB*. Serie Pearson Educación. México D.F.: Prentice-Hall Hispanoamericana. ISBN: 9688808601.

Network, Mozilla Developer (2026). *HTTP — MDN Web Docs*. tu02.tex. URL: https://developer.mozilla.org/es/docs/Web/HTTP.

Newham, Cameron y Bill Rosenblatt (2005). *Learning the bash Shell*. 3.ª ed. Introducción ordenada al shell como lenguaje: variables, control de flujo, funciones y manejo de trabajos. O'Reilly.

ngspice Project (2026). *ngspice Manual*. Simulacion de circuitos tipo SPICE. URL: https://ngspice.sourceforge.io/docs.html (visitado 02-08-2026).

Nielsen, Jakob (1994). *10 Usability Heuristics for User Interface Design*. Lista breve para revisar una interfaz antes de darla por terminada: visibilidad del estado, prevención de errores, mensajes comprensibles. Nielsen Norman Group. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (visitado 13-08-2026).

Norman, Donald A. (2013). *The Design of Everyday Things*. 2.ª ed. Edición revisada y ampliada. Affordances, modelos mentales, visibilidad y realimentación: por qué una interfaz se entiende sola o no. Basic Books.

NumPy Developers (2026). *NumPy Reference*. URL: https://numpy.org/doc/stable/reference/ (visitado 15-08-2026).

Nussbaum, Lucas (2026). *Debian Packaging Tutorial*. Del código fuente al paquete .deb. El Debian Policy Manual ya está en el archivo como lab1_extra_debian_policy. Debian Project. URL: https://www.debian.org/doc/manuals/packaging-tutorial/packaging-tutorial.en.pdf.

NXP Semiconductors (2021). *I2C-bus Specification and User Manual*. UM10204. Ver. 7.0. Especificación formal del bus: direccionamiento de 7 bits, arbitraje, modos de velocidad. El protocolo más usado por los sensores de bajo costo del capítulo. NXP Semiconductors. URL: https://www.nxp.com/docs/en/user-guide/UM10204.pdf (visitado 14-08-2026).

Nystrom, Robert (2014). *Game Programming Patterns*. Libre y en línea. Los capítulos sobre el bucle de juego y el método de actualización describen exactamente el patrón de animación de la figura: estado, actualización, dibujo, espera. Genever Benning. URL: https://gameprogrammingpatterns.com/ (visitado 13-08-2026).

Octave Forge community (2026). *Octave Packages*. Paquetes adicionales de GNU Octave (control, signal, statistics, etc.) URL: https://gnu-octave.github.io/packages/ (visitado 16-08-2026).

Ogata, Katsuhiko (2010). *Modern Control Engineering*. 5.ª ed. Sistemas dinamicos, espacio de estados, funciones de transferencia y control. Prentice Hall.

Okabe, Masataka y Kei Ito (2008). «Color Universal Design: How to Make Figures and Presentations That Are Friendly to Colorblind People». En: La paleta de ocho colores distinguibles bajo las formas más frecuentes de daltonismo, muy usada en publicaciones científicas. Se traduce directamente a las definiciones de linetype de gnuplot. URL: https://jfly.uni-koeln.de/color/ (visitado 14-08-2026).

OpenJS Foundation (2026a). *Node.js Documentation*. El lenguaje fuera del navegador: archivos, procesos y red. Es lo que lo convierte en un lenguaje de propósito general y no sólo en el de las páginas web. URL: https://nodejs.org/docs/latest/api/ (visitado 14-08-2026).

– (2026b). *Node.js documentation*. OpenJS Foundation. URL: https://nodejs.org/docs/latest/api/ (visitado 15-08-2026).

Oppenheim, Alan V. y Ronald W. Schafer (2009). *Discrete-Time Signal Processing*. 3.ª ed. Tratamiento formal, por si la unidad necesita respaldo teórico sobre muestreo y filtrado. Pearson.

Ott, Henry W. (2009). *Electromagnetic Compatibility Engineering*. La referencia canónica sobre interferencia: lazos de masa, blindaje, apantallamiento y cableado. Cubre la sección de errores de interferencia, que hoy no tiene fuente. Wiley. DOI: 10.1002/9780470508510.

Ousterhout, John (1996). *Why Threads Are a Bad Idea (for Most Purposes)*. Charla invitada, USENIX Annual Technical Conference. El argumento clásico a favor del bucle de eventos con callbacks, que es el modelo que usa Scilab.

OWASP Foundation (2021). *OWASP Top Ten Web Application Security Risks*. Las diez categorías de riesgo más frecuentes. Pertinente a la sección de seguridad básica: validación de entradas, inyección de comandos. Open Worldwide Application Security Project. URL: https://owasp.org/www-project-top-ten/ (visitado 13-08-2026).

PDL Developers (2026a). *PDL Book*. The Perl Data Language project. URL: https://pdl.perl.org/PDLdocs/PDL/Book.html (visitado 15-08-2026).

– (2026b). *PDL::GSL modules: INTEG, MROOT, DIFF, INTERP, MC*. Envoltorios de la GNU Scientific Library incluidos en PDL. The Perl Data Language project. URL: https://metacpan.org/pod/PDL::GSL::INTEG (visitado 16-08-2026).

Peer 2 Peer University (2014). *Getting Started with Scilab: Plotting Graphs*. URL: https://p2pu.org/he/groups/getting-started-with-scilab/content/session-8-plotting-graphs/.

Peirce, Charles S. (1878). «How to Make Our Ideas Clear». En: *Popular Science Monthly* 12. Trad. esp.: "Cómo esclarecer nuestras ideas", págs. 286-302.

Pérez, Fernando y Brian E. Granger (2007a). «IPython: A System for Interactive Scientific Computing». En: *Computing in Science & Engineering* 9.3, págs. 21-29. DOI: 10.1109/MCSE.2007.53.

– (2007b). «IPython: A System for Interactive Scientific Computing». En: *Computing in Science & Engineering* 9.3. El origen del proyecto: la shell interactiva de la que salió el cuaderno en 2011, págs. 21-29. DOI: 10.1109/MCSE.2007.53.

Perez-Riverol, Yasset et al. (2016a). «Ten Simple Rules for Taking Advantage of Git and GitHub». En: *PLOS Computational Biology* 12.7. Reglas prácticas para usar control de versiones en investigación: qué versionar, cómo escribir los mensajes y cómo colaborar, e1004947. DOI: 10.1371/journal.pcbi.1004947.

– (2016b). «Ten Simple Rules for Taking Advantage of Git and GitHub». En: *PLOS Computational Biology* 12.7. Flujo de trabajo en repositorios públicos. La regla 1 es el archivo de licencia, que conecta con la unidad 1, e1004947. DOI: 10.1371/journal.pcbi.1004947.

Perkel, Jeffrey M. (2018). «Why Jupyter Is Data Scientists' Computational Notebook of Choice». En: *Nature* 563.7732. Panorama de la adopción del formato y de sus límites, págs. 145-146. DOI: 10.1038/d41586-018-07196-1.

Peters, Tim (2004). *PEP 20: The Zen of Python*. Los principios de diseño del lenguaje en veinte aforismos. Sirve para discutir qué se considera código claro y por qué. URL: https://peps.python.org/pep-0020/ (visitado 14-08-2026).

Pimentel, João Felipe et al. (2019). «A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks». En: *Proceedings of the 16th International Conference on Mining Software Repositories (MSR)*. Contrapeso empírico: de más de un millón de cuadernos publicados en GitHub, muy pocos se ejecutan de principio a fin y reproducen sus propios resultados. IEEE, págs. 507-517. DOI: 10.1109/MSR.2019.00077.

Plotly Technologies Inc. (2026). *Plotly.js graphing library*. Plotly Technologies Inc. URL: https://plotly.com/javascript/ (visitado 15-08-2026).

Poincare, Henri (1905). *Science and Hypothesis*. Convencionalismo, geometria, hipotesis y ciencia. Walter Scott Publishing.

Popper, Karl (1959). *The Logic of Scientific Discovery*. Falsacionismo y demarcacion cientifica. Hutchinson.

Prasad, Aanand et al. (2021). *Command Line Interface Guidelines*. Guía de diseño de programas de línea de comandos: argumentos, salida, errores, interactividad. El equivalente de las heurísticas de usabilidad para la terminal. URL: https://clig.dev/ (visitado 13-08-2026).

Press, William H. et al. (2007). *Numerical Recipes: The Art of Scientific Computing*. 3.ª ed. Capítulo 15: ajuste por mínimos cuadrados no lineal y algoritmo de Levenberg-Marquardt, que es exactamente lo que hace el comando fit de gnuplot, junto con el significado de la matriz de covarianza que informa. Cambridge University Press.

Preston-Werner, Tom, Pradyun Gedam et al. (2021). *TOML: Tom's Obvious Minimal Language, v1.0.0*. Formato normalizado de archivo de configuración clave=valor. Es la versión formal del nivel 2 del paso de parámetros. URL: https://toml.io/en/v1.0.0 (visitado 13-08-2026).

Prlić, Andreas y James B. Procter (2012). «Ten Simple Rules for the Open Development of Scientific Software». En: *PLOS Computational Biology* 8.12. Conecta con software

libre y desarrollo colaborativo de la unidad 1, e1002802. DOI: 10.1371/journal.pcbi.1002802.

Python Software Foundation (2026a). *Python 3 Documentation*. Documentación oficial de Python 3. URL: https://docs.python.org/3/ (visitado 02-08-2026).

– (2026b). *The Python Language Reference and Standard Library*. Referencia del lenguaje y de la biblioteca estándar. Es la fuente a consultar mientras se programa. URL: https://docs.python.org/3/ (visitado 14-08-2026).

– (2026c). *The Python Language Reference*. Python Software Foundation. URL: https://docs.python.org/3/reference/ (visitado 15-08-2026).

Quarteroni, Alfio, Riccardo Sacco y Fausto Saleri (2007). *Numerical Mathematics*. 2.ª ed. Fundamentos matematicos de calculo numerico. Springer.

Quarteroni, Alfio, Fausto Saleri y Paola Gervasio (2014). *Scientific Computing with MATLAB and Octave*. 4.ª ed. Vol. 2. Texts in Computational Science and Engineering. Cálculo científico con la misma sintaxis en MATLAB y Octave. Berlin: Springer. DOI: 10.1007/978-3-642-45367-0.

Radford, Alec et al. (2018). *Improving Language Understanding by Generative Pre–Training*. Preprint. OpenAI.

Ramalho, Luciano (2022). *Fluent Python: Clear, Concise and Effective Programming*. 2.ª ed. El nivel intermedio: modelo de datos, iterables, decoradores, clausuras y funciones de orden superior tal como el lenguaje los implementa. O'Reilly.

Ramey, Chet y Brian Fox (18 de mayo de 2025). *Bash Reference Manual*. Ver. 5.3. Free Software Foundation. URL: https://www.gnu.org/software/bash/manual/bash.html (visitado 02-08-2026).

Raspberry Pi Ltd. (2026). *Raspberry Pi Documentation*. Documentación oficial, incluye el uso de GPIO para entrada y salida digital. Raspberry Pi Ltd. URL: https://www.raspberrypi.com/documentation/.

Raymond, Eric S. (2003). *The Art of UNIX Programming*. Addison-Wesley.

Ritchie, Dennis M. y Ken Thompson (1974). «The UNIX Time-Sharing System». En: *Communications of the ACM* 17.7, págs. 365-375. DOI: 10.1145/361011.361061.

Rivest, R. L (2008). *On the notion of "software independence" in voting systems. Philosophical Transactions of the Royal Society A, 366(1881), 3759–3767*. tu01.tex.

Robinson, David y Ken Coar (oct. de 2004). *The Common Gateway Interface (CGI) Version 1.1*. Inf. téc. RFC 3875. Especificación del CGI: variables de entorno, entrada estándar, formato de la respuesta. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc3875 (visitado 13-08-2026).

Romero Mora, Gabriel (2014). *Fundamentos de programación en Scilab*. SlideShare. URL: http://www.slideshare.net/AstroPhoenix/fundamentos-de-programacin-en-scilab.

Rossum, Guido van, Barry Warsaw y Nick Coghlan (2001). *PEP 8: Style Guide for Python Code*. Convenciones de escritura adoptadas por toda la comunidad: sangría, nombres, longitud de línea. Un caso claro de la legibilidad tratada como norma. URL: https://peps.python.org/pep-0008/ (visitado 14-08-2026).

Rule, Adam et al. (2019). «Ten Simple Rules for Writing and Sharing Computational Analyses in Jupyter Notebooks». En: *PLOS Computational Biology* 15.7. Aplicable a la unidad 8 también. Contar la historia del análisis, documentar el entorno, dividir el notebook en pasos, e1007007. DOI: 10.1371/journal.pcbi.1007007.

Russell, Stuart J. y Peter Norvig (2021). *Artificial Intelligence: A Modern Approach*. 4.ª ed. Pearson.

Sandve, Geir Kjetil et al. (2013). «Ten Simple Rules for Reproducible Computational Research». En: *PLOS Computational Biology* 9.10. Registrar cómo se produjo cada resultado, evitar los pasos manuales, archivar versiones de programas externos, guardar resultados intermedios, e1003285. DOI: 10.1371/journal.pcbi.1003285.

Saravia Alía, Diego (2002). *Introducción a la programación*. Apuntes de clase.

– (jun. de 2005). *Programar es una habilidad cultural fundamental*. tu01.tex.

– (2015). *Simusol*. Software y material de simulación usado en Laboratorio 1 para interacción con datos, simulaciones y gráficos de Gnuplot. URL: http://simusol.org.

– (2019). «Simulación de sistemas físicos: Comportamiento, ecuaciones diferenciales, metodologías». Archivo PDF: B26CSimula.pdf. Falta la ruta o URL del archivo.

Schneier, B (nov. de 2004). *The problem with electronic voting machines. Schneier on Security*. tu01.tex. URL: https://www.schneier.com/blog/archives/2004/11/the_problem_wit.html.

Schulz, Markus (2026). *LaTeX::Table: Perl extension for the automatic generation of LaTeX tables*. CPAN. URL: https://metacpan.org/pod/LaTeX::Table (visitado 16-08-2026).

Schwaber, Ken y Jeff Sutherland (2020a). *The Scrum Guide*. Definición corta y oficial del marco. Útil si organizás el práctico integrador en iteraciones con entregas parciales. URL: https://scrumguides.org/.

– (2020b). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. Hay traducción al español en el mismo sitio. URL: https://scrumguides.org/ (visitado 16-08-2026).

Schwartz, Randal L., brian d foy brian d y Tom Phoenix (2021). *Learning Perl*. 8.ª ed. Introducción por problemas: escalares, listas, hashes, expresiones regulares y archivos. El camino más corto para llegar a procesar texto. O'Reilly.

Scilab (2024). *Getting Started with Scilab*. URL: https://www.scilab.org/tutorials/getting-started (visitado 03-08-2026).

– (2026). *Tutorials*. URL: https://www.scilab.org/tutorials (visitado 03-08-2026).

Scilab Enterprises (2026a). *fprintfMat — Writes a Matrix in a File*. Página de ayuda de la función fprintfMat. URL: https://help.scilab.org/docs/2026.0.0/en_US/fprintfMat.html (visitado 13-08-2026).

– (2026b). *host — Executes a Windows, Linux, or MacOS Command*. Incluye el alias unix. URL: https://help.scilab.org/docs/2026.0.0/en_US/host.html (visitado 13-08-2026).

– (2026c). *Scilab Online Help*. Inglés. Ver. 2026.0.0. Documentación oficial de Scilab. Ayuda oficial de Scilab. Scilab. URL: https://help.scilab.org/docs/2026.0.0/en_US/index.html (visitado 13-08-2026).

– (2026d). *Xcos Documentation*. Documentacion de Xcos para simulacion grafica. URL: https://help.scilab.org/docs/2026.0.0/en_US/xcos.html (visitado 02-08-2026).

Scilab Group (2000). *Introduction to Scilab: User's Guide*. hay dos índices de comandos, uno por tema y otro alfabético, en Word, útiles para buscar comandos. Eso no está en el .bib y es información práctica para el laboratorio. URL: https://cermics.enpc.fr/scilab_new/site/Liens/intro/intro.html (visitado 03-08-2026).

– (2026). *Scilab Reference Manual*. Manual correspondiente a Scilab 2026.1.0, publicado el 19 de mayo de 2026. URL: https://epdf.pub/scilab-reference-manual.html (visitado 03-08-2026).

Shafranovich, Yakov (oct. de 2005). *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. Inf. téc. RFC 4180. RFC Editor. DOI: 10.17487/RFC4180. URL: https://www.rfc-editor.org/rfc/rfc4180 (visitado 22-08-2026).

Shannon, Claude E. (1948). «A Mathematical Theory of Communication». En: *The Bell System Technical Journal* 27. Fuente original del teorema de muestreo. Enlaza con la unidad 1 por el lado de la representación de la información, págs. 379-423, 623-656.

Shneiderman, Ben (1983). «Direct Manipulation: A Step Beyond Programming Languages». En: *Computer* 16.8. Fundamenta el contraste que abre la unidad: manipular un objeto visible no es lo mismo que escribir el comando que lo modifica, págs. 57-69. DOI: 10.1109/MC.1983.1654471.

Shneiderman, Ben et al. (2016). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 6.ª ed. Manual de referencia de interacción persona-computadora. Incluye las ocho reglas de oro del diseño de interfaces. Pearson.

Simons, B. y D. W Jones (2012). *Broken ballots: Will your vote count?* tu01.tex. CSLI Publications, Stanford University.

Smaldone, J (2017). *Voto electrónico: una solución en busca de problemas*. tu01.tex. URL: https://blog.smaldone.com.ar/.

Smith, Arfon M., Daniel S. Katz y Kyle E. Niemeyer (2016). «Software Citation Principles». En: *PeerJ Computer Science* 2, e86. DOI: 10.7717/peerj-cs.86. URL: https://peerj.com/articles/cs-86/.

Smith, Steven W. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing*. Libre y en línea. Explicación accesible del muestreo, el teorema de Nyquist y el aliasing. California Technical Publishing. URL: https://www.dspguide.com/.

Sommerville, Ian (2015). *Software Engineering*. 10.ª ed. Manual general, por si hace falta una referencia de conjunto sobre requerimientos, diseño, verificación y gestión. Pearson.

Stallman, Richard M., Roland McGrath y Paul D. Smith (2023). *GNU Make Manual*. Compilación separada, dependencias y reglas: la herramienta con la que se construye un programa en C formado por varios archivos. Free Software Foundation. URL: https://www.gnu.org/software/make/manual/ (visitado 14-08-2026).

Stodden, Victoria y Sheila Miguez (2014). «Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research». En: *Journal of Open Research Software* 2.1. Infraestructura y entornos; licencias y publicación del código. DOI: 10.5334/jors.ay.

Suppes, Patrick (1960). «A Comparison of the Meaning and Uses of Models in Mathematics and the Empirical Sciences». En: *Synthese* 12.2–3. Modelos en matematica y ciencias empiricas, págs. 287-301.

Sussman, Gerald Jay y Guy L. Steele (1975). *Scheme: An Interpreter for Extended Lambda Calculus*. Inf. téc. AI Memo 349. Donde la clausura aparece como mecanismo de implementación: la función junto con el entorno en que fue definida. MIT Artificial Intelligence Laboratory.

Sweet, Michael R. (1999). *Serial Programming Guide for POSIX Operating Systems*. Cómo se abre y configura un puerto serie desde un programa: la estructura termios, velocidad, bits de datos, paridad y modos canónico y crudo. URL: https://www.cmrr.umn.edu/~strupp/serial.html (visitado 14-08-2026).

Taschuk, Morgan y Greg Wilson (2017). «Ten Simple Rules for Making Research Software More Robust». En: *PLOS Computational Biology* 13.4. Que el programa ande en otra máquina que no sea la del autor: mensajes de error útiles, parámetros por línea de comandos, dependencias declaradas, e1005412. DOI: 10.1371/journal.pcbi.1005412.

Taylor, John R. (1997). *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*. 2.ª ed. El clásico de propagación de errores a nivel de laboratorio de grado. Cubre el ítem «Errores en general» del programa. University Science Books.

– (2005). *Classical Mechanics*. Oscilador amortiguado y forzado: el sistema masa-resorte que la unidad integra con ode() y anima. University Science Books.

Text::CSV maintainers (3 de mar. de 2025). *Text::CSV — Comma-Separated Values Manipulator*. Ver. 2.06. Documentación de la distribución Text::CSV en MetaCPAN. URL: `https://metacpan.org/dist/Text-CSV` (visitado 22-08-2026).

The GTK Team (2026). *GTK 4 Documentation: Getting Started with GTK*. Documentacion oficial de GTK 4. Consultada el 2026-08-23. GNOME Project.

The IEEE y The Open Group (2024). *The Open Group Base Specifications Issue 8, IEEE Std 1003.1-2024*. Estándar POSIX.1-2024 sobre interfaces del sistema operativo, shell, utilidades y API. URL: `https://pubs.opengroup.org/onlinepubs/9799919799/` (visitado 02-08-2026).

The Linux Kernel Organization (2026). *The Linux Kernel Documentation*. Documentacion oficial del nucleo Linux. URL: `https://docs.kernel.org/` (visitado 02-08-2026).

The Linux man-pages project (22 de abr. de 2026). *The Linux man-pages project*. Ver. 6.18. Documentación de las interfaces del kernel de Linux y de la biblioteca C utilizadas por programas de espacio de usuario. URL: `https://www.kernel.org/doc/man-pages/` (visitado 02-08-2026).

The Matplotlib development team (2026). *Matplotlib documentation*. URL: `https://matplotlib.org/stable/` (visitado 15-08-2026).

The Perl Documentation Team (2026a). *perldata – Perl data types*. The Perl Foundation. URL: `https://perldoc.perl.org/perldata` (visitado 15-08-2026).

– (2026b). *perldebug – Perl debugging*. The Perl Foundation. URL: `https://perldoc.perl.org/perldebug` (visitado 15-08-2026).

– (2026c). *perlfunc – Perl builtin functions*. The Perl Foundation. URL: `https://perldoc.perl.org/perlfunc` (visitado 15-08-2026).

– (2026d). *perlintro – a brief introduction and overview of Perl*. The Perl Foundation. URL: `https://perldoc.perl.org/perlintro` (visitado 15-08-2026).

The Perl Foundation (2026). *Perl Documentation*. Referencia del lenguaje, incluida perlre sobre expresiones regulares y perldsc sobre estructuras de datos anidadas. URL: `https://perldoc.perl.org/` (visitado 14-08-2026).

Torvalds, Linus (2007). «Tech Talk: Linus Torvalds on Git». En: *Google Tech Talks*. Por qué se escribió Git y qué problemas del modelo centralizado buscaba resolver. Útil para entender las decisiones de diseño, más allá del tono. URL: `https://www.youtube.com/watch?v=4XpnKHJAok8` (visitado 14-08-2026).

Trefethen, Lloyd N. y David Bau (1997). *Numerical Linear Algebra*. Algebra lineal numerica; estabilidad, factorizaciones y valores propios. SIAM.

Tufte, Edward R. (2001). *The Visual Display of Quantitative Information*. 2.ª ed. Gráficos del informe. Enlaza con la unidad 7 y con Wilke, que ya está en el archivo. Graphics Press.

Tversky, Barbara, Julie Bauer Morrison y Mireille Bétrancourt (2002). «Animation: Can It Facilitate?» En: *International Journal of Human-Computer Studies* 57.4. Contrapeso empírico: revisión de estudios donde animar no mejora la comprensión frente a imágenes fijas. Sirve para discutir cuándo conviene animar, págs. 247-262. DOI: `10.1006/ijhc.2002.1017`.

Universidad de Alicante (2014). *Sistemas de adquisición y procesamiento de datos*. Archivo PDF. URL: http://rua.ua.es/dspace/bitstream/10045/19119/1/Sistemas%5C%20de%5C%20adquisici%5C%C3%5C%B3n%5C%20y%5C%20Procesamiento%5C%20de%5C%20datos.pdf.

Urroz, Gilberto E. (2001). *Numerical and Statistical Methods with Scilab for Science and Engineering*. Utah State University.

USB Implementers Forum (2000). *Universal Serial Bus Specification, Revision 2.0*. Qué hay debajo del /dev/ttyUSB: enumeración, descriptores y transferencias. Explica por qué el adaptador USB-serie aparece como puerto serie. USB Implementers Forum. URL: https://www.usb.org/documents (visitado 14-08-2026).

Vaswani, Ashish et al. (2017). «Attention Is All You Need». En: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. La arquitectura Transformer, base de los modelos de lenguaje actuales, págs. 5998-6008. URL: https://arxiv.org/abs/1706.03762.

Venturin, Manolo (2013). *How to Develop a Graphical User Interface (GUI) in Scilab*. Tutorial de 18 páginas basado en el modelo LHY. Licencia Creative Commons BY-NC-ND 3.0. Openeering. URL: http://www.openeering.com/sites/default/files/LHY_Tutorial_Gui.pdf (visitado 03-08-2026). Espejo en Scilab.org: https://www.scilab.org/sites/default/files/prg/att/1710/LHY_Tutorial_Gui.pdf.

Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy y David Cournapeau (2020). «SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python». En: *Nature Methods* 17. Integración, optimización, álgebra lineal, ecuaciones diferenciales: el equivalente en Python de lo que en el entorno de cálculo viene incorporado, págs. 261-272. DOI: 10.1038/s41592-019-0686-2.

Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau et al. (2020). «SciPy 1.0: fundamental algorithms for scientific computing in Python». En: *Nature Methods* 17, págs. 261-272. DOI: 10.1038/s41592-019-0686-2.

Wall, Larry, Tom Christiansen y Jon Orwant (2000a). *Programming Perl*. 3.ª ed. Sebastopol, CA: O'Reilly. ISBN: 978-0-596-00027-1.

– (2000b). *Programming Perl*. 3.ª ed. O'Reilly.

WHATWG (2026). *HTML Living Standard*. Norma vigente de HTML. La sección de formularios define los tipos de control, la validación y la serialización. Web Hypertext Application Technology Working Group. URL: https://html.spec.whatwg.org/multipage/ (visitado 13-08-2026).

Whitesides, George M. (2004). «Whitesides' Group: Writing a Paper». En: *Advanced Materials* 16.15. Tres páginas sobre cómo se arma un texto técnico: el esquema primero, el texto después, págs. 1375-1377. DOI: 10.1002/adma.200400767.

Wikipedia contributors (2023a). *Adquisición de datos*. URL: https://es.wikipedia.org/wiki/Adquisici%5C%C3%5C%B3n_de_datos.

– (2023b). *Conversión analógica-digital*. URL: https://es.wikipedia.org/wiki/Conversi%5C%C3%5C%B3n_anal%5C%C3%5C%B3gica-digital.

Wilke, Claus O. (2019). *Fundamentals of Data Visualization*. O'Reilly Media. URL: https://clauswilke.com/dataviz/.

Williams, Thomas, Colin Kelley y many others (2026). *gnuplot 6.0: An Interactive Plotting Program*. URL: http://www.gnuplot.info/documentation.html (visitado 15-08-2026).

Wilson, Greg, D. A. Aruliah et al. (2014). «Best Practices for Scientific Computing». En: *PLOS Biology* 12.1. Escribir para personas antes que para la máquina, automatizar

lo repetitivo, no optimizar antes de tiempo, documentar el propósito, e1001745. DOI: 10.1371/journal.pbio.1001745.

Wilson, Greg, Jennifer Bryan et al. (2017). «Good Enough Practices in Scientific Computing». En: *PLOS Computational Biology* 13.6. Mínimo viable de organización: estructura de directorios, nombres de archivos, manejo de datos, colaboración. Pensado para quien no viene de computación, e1005510. DOI: 10.1371/journal.pcbi.1005510.

Wolchok, Scott et al. (2010). «Security Analysis of India's Electronic Voting Machines». En: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. Clave conservada como halderman2015security por compatibilidad con cita existente. ACM, págs. 1-14.

World Wide Web Consortium (2023). *Web Content Accessibility Guidelines (WCAG) 2.2*. Accesibilidad del formulario: etiquetas asociadas a los campos, contraste, navegación por teclado. W3C. URL: https://www.w3.org/TR/WCAG22/ (visitado 13-08-2026).

– (2026). *Scalable Vector Graphics (SVG) 2*. Gráficos vectoriales en el navegador: alternativa al PNG incrustado cuando la figura debe ser interactiva o escalable. W3C. URL: https://www.w3.org/TR/SVG2/ (visitado 13-08-2026).

Wu, Jonathan Duke Leto (2026). *Math::ODE: Solve N-th Order Ordinary Differential Equations*. Integrador Runge–Kutta de orden 4 en Perl puro. CPAN. URL: https://metacpan.org/pod/Math::ODE (visitado 16-08-2026).

Yergeau, François (nov. de 2003). *UTF-8, a Transformation Format of ISO 10646*. Inf. téc. RFC 3629. RFC Editor. DOI: 10.17487/RFC3629. URL: https://www.rfc-editor.org/rfc/rfc3629 (visitado 22-08-2026).

Ziegler, John G. y Nathaniel B. Nichols (1942). «Optimum Settings for Automatic Controllers». En: *Transactions of the ASME* 64.11. El trabajo original de donde salen las reglas de sintonía que usa el capítulo, págs. 759-768.