servguiweb

Servidor web científico multi-motor

Diego Saravia Alia
Universidad Nacional de Salta

Versión 0008

Índice

1. El sistema servguiweb

1.1. 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.

1.2. Instalación y uso rápido

1.2.1. 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
```

1.2.2. 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
```

1.2.3. Iniciar el servidor

```
cd mi_aplicacion/
python3 servidor.py
```

Abrir en el navegador: `http://localhost:8080`

Para detener: `Ctrl+C` en la terminal.

1.3. 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/articulos/servguiweb_0008-1_all.deb
$sudo dpkg -i servguiweb_0008-1_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
```

1.4. Manual de usuario

1.4.1. 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.

1.4.2. 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.

1.4.3. 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.

1.4.4. 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.

1.5. Documentación técnica

1.5.1. 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.

1.5.2. 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 1: 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)
#   - ambas rutas se pasan por variables de entorno
VERSION = '0008'

from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import parse_qs
```

```
from html import escape
import os
import subprocess

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'
MARCADOR    = '<!-- RESULTADOS -->'

# 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():
    with open(FORM_FILE, encoding='utf-8') as f:
        return f.read()

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):
    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]:
        if os.path.exists(path):
            os.remove(path)

    env = os.environ.copy()
```

```
        env['PARAM_FILE'] = PARAM_FILE
        env['RESULT_FILE'] = RESULT_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()

    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
        html = leer_form().replace(MARCADOR, '<div
            id=resultados></div>')
        self._enviar(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).decode('utf-8',
            errors='replace')

        # parse_qs conserva todos los valores. Para un checkbox
            con
        # hidden=off seguido del checkbox=on interesa el ultimo.
        campos = parse_qs(body, keep_blank_values=True)
        params = {k: valores[-1] for k, valores in
            campos.items()}

        # 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)
        html_final = leer_form().replace(
            MARCADOR, f'<div id=resultados>{div}</div>')
        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(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('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.

1.5.3. 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.

1.5.4. 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 $\longrightarrow$ servidor HTTP $\longrightarrow$ params.txt $\longrightarrow$ motor $\longrightarrow$ resultado.html $\longrightarrow$ 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 2: 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 3: 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 4: 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 5: 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 6: 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 7: 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: 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 9: 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.

1.5.5. 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.

1.5.6. 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 10: Patron de checkbox correcto

```
<input type=hidden name=g_tray value=off>
<label><input type=checkbox name=g_tray value=on checked>
   Trayectoria</label>
```

1.5.7. 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 11: 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
```

1.5.8. 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 12: 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)
```

1.5.9. 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.

1.5.10. 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.

1.5.11. 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.

1.5.12. 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.

1.5.13. 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.

1.5.14. 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
```

1.5.15. 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
   serv guiweb
#
# uso:  bash serv guiweb/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 \
    -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 'http: todos los motores' \
    python3 "$DIR/http/test_motores_http.py"

correr 'http: caminos de error' \
    python3 "$DIR/http/test_errores_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`.

1.6. 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`.

1.6.1. Ejemplo: aplicación de estadistica

Un ejemplo alternativo que calcula estadísticas de un vector de datos ingresados por el usuario:

Listing 13: 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 14: 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
```

1.7. 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.

1.7.1. formulario.html

Listing 15: 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>
```

1.7.2. calcular.sce

Listing 16: 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
```

2. 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.

```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.

```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.

```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.

```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.

```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.

```python
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"
```

Referencias

gnuplot Team (2026). *gnuplot Documentation*. Documentacion oficial de gnuplot. URL: https://gnuplot.sourceforge.net/documentation.html (visitado 02-08-2026).

Hall, Brian (2026). *Beej's Guide to Network Programming*. Programacion de sockets en C. URL: https://beej.us/guide/bgnet/ (visitado 02-08-2026).

Hertzog, R. y R Mas (2026). *The Debian administrator's handbook*. tu01.tex. Freexian. URL: https://debian-handbook.info/ (visitado 19-07-2026).

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).

World Wide Web Consortium (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).