

Sistema de formularios web con Scilab como servidor

Documentacion de usuario y tecnica

Diego Saravia Alia
Universidad Nacional de Salta

Julio 2026

Contents

1	Introduccion	2
2	Instalacion y uso rapido	2
2.1	Requisitos	2
2.2	Estructura de archivos	2
2.3	Iniciar el servidor	2
3	Manual de usuario	3
3.1	Completar el formulario	3
3.2	Obtener resultados	3
3.3	Interpretar los resultados	3
3.4	Comparar simulaciones	3
4	Documentacion tecnica	3
4.1	Arquitectura del sistema	3
4.2	servidor.py	4
4.3	formulario.html	6
4.4	calcular.sce	6
4.5	Generacion de graficas con gnuplot	7
4.6	Conversion de imagen a base64	8
4.7	Limitaciones conocidas	8
4.8	Diagnostico de errores	8
5	Adaptar el sistema a otra aplicacion	9
5.1	Ejemplo: aplicacion de estadistica	9
6	Codigo fuente del ejemplo: simulador de proyectil	11
6.1	formulario.html	11
6.2	calcular.sce	13

1 Introduccion

Este sistema permite crear aplicaciones web de calculo cientifico usando Scilab como motor de computo. El usuario completa un formulario en el navegador, los datos viajan al servidor, Scilab realiza el calculo y devuelve los resultados en la misma pagina, incluyendo graficas.

La arquitectura separa tres responsabilidades:

- **formulario.html**: la interfaz que el usuario ve y edita.
- **calcular.sce**: el calculo cientifico en Scilab.
- **servidor.py**: el mensajero HTTP, no sabe que calcula Scilab.

Para cambiar la aplicacion solo se editan `formulario.html` y `calcular.sce`. El servidor nunca se modifica.

2 Instalacion y uso rapido

2.1 Requisitos

- Scilab (rama main, abril 2024 o posterior)
- Python 3.6 o posterior
- gnuplot

Instalar gnuplot si no esta disponible:

```
1 sudo apt install gnuplot
```

2.2 Estructura de archivos

Los tres archivos deben estar en el mismo directorio:

```
1 mi_aplicacion/  
2   formulario.html      # interfaz web  
3   calcular.sce         # calculo en Scilab  
4   servidor.py          # servidor HTTP
```

2.3 Iniciar el servidor

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

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

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

3 Manual de usuario

3.1 Completar el formulario

La pagina principal muestra el formulario con los parametros de la simulacion. 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.
- **Metodo numerico:** Euler, Runge-Kutta 4 u ODE adaptativo.
- **Graficas:** seleccionar cuales generar.
- **Notas:** texto libre para registrar observaciones.

3.2 Obtener resultados

Presionar **Calcular**. El servidor llama a Scilab, que realiza la simulacion y devuelve la pagina con los resultados debajo del formulario. Los valores ingresados se conservan para facilitar la comparacion entre simulaciones.

3.3 Interpretar los resultados

Los resultados incluyen cuatro estadisticas numericas:

Estadistica	Descripcion
Alcance	Distancia horizontal al impacto [m]
Altura maxima	Punto mas 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 graficas seleccionadas: trayectoria (y vs x) y velocidad vs tiempo.

3.4 Comparar simulaciones

Para comparar el efecto de distintos parametros conviene abrir dos ventanas del navegador en `http://localhost:8080` y variar un parametro entre ambas.

4 Documentacion tecnica

4.1 Arquitectura del sistema

El flujo de una solicitud es el siguiente:

1. El navegador envia un `POST` con los datos del formulario.
2. `servidor.py` parsea los campos y escribe `/tmp/params.sce`.

3. `servidor.py` llama a `scilab-cli -e "exec('calcular.sce', -1); quit"`.
4. `calcular.sce` lee `params.sce`, simula, llama a `gnuplot`, convierte el PNG a base64 y escribe `/tmp/resultado.html`.
5. `servidor.py` lee `resultado.html` e inserta su contenido en `formulario.html` en el marcador `<!-- RESULTADOS -->`.
6. El navegador recibe la pagina completa.

4.2 servidor.py

El servidor es completamente generico: no conoce el dominio de la aplicacion. Realiza tres tareas:

1. **Parseo HTTP:** extrae los campos del formulario del cuerpo del POST.
2. **Escritura de parametros:** convierte cada campo a una variable Scilab y la escribe en `params.sce`. Los valores numericos se escriben sin comillas; los textuales con comillas simples; los booleanos (`on/off`) como `%T/%F`.
3. **Ensamblado de la respuesta:** inserta el fragmento HTML generado por Scilab en el formulario y lo devuelve.

Los archivos de comunicacion son:

Archivo	Escribe	Lee
<code>/tmp/params.sce</code>	<code>servidor.py</code>	<code>calcular.sce</code>
<code>/tmp/resultado.html</code>	<code>calcular.sce</code>	<code>servidor.py</code>
<code>/tmp/proy_graph.png</code>	<code>gnuplot</code> (via Scilab)	base64 (Scilab)
<code>/tmp/proy_graph.b64</code>	Scilab	Scilab
<code>/tmp/error_scilab.txt</code>	<code>servidor.py</code> (log)	usuario

Listing 1: `servidor.py` completo

```

1  #!/usr/bin/env python3
2  from http.server import HTTPServer, BaseHTTPRequestHandler
3  from urllib.parse import unquote_plus
4  import subprocess, os
5
6  PUERTO = 8080
7  DIR = os.path.dirname(os.path.abspath(__file__))
8  FORM_FILE = os.path.join(DIR, 'formulario.html')
9  PARAMS_FILE = '/tmp/params.sce'
10 RESULT_FILE = '/tmp/resultado.html'
11 SCE_FILE = os.path.join(DIR, 'calcular.sce')
12 ERROR_FILE = '/tmp/error_scilab.txt'
13 MARCADOR = '<!-- RESULTADOS -->'
14
15 def leer_form():
16     with open(FORM_FILE, encoding='utf-8') as f:
17         return f.read()
18
19 def escribir_params(params):

```

```

20 with open(PARAMS_FILE, 'w') as f:
21     for k, v in params.items():
22         if v == 'on':
23             f.write(f"{k} = %T;\n")
24         elif v == 'off':
25             f.write(f"{k} = %F;\n")
26         elif v == '':
27             f.write(f"{k} = '';\n")
28         else:
29             try:
30                 float(v)
31                 f.write(f"{k} = {v};\n")
32             except ValueError:
33                 f.write(f"{k} = '{v}';\n")
34
35 def llamar_scilab():
36     for path in [RESULT_FILE, ERROR_FILE]:
37         if os.path.exists(path):
38             os.remove(path)
39     try:
40         r = subprocess.run(
41             ['scilab-cli', '-e',
42              f"exec('{SCE_FILE}', -1); quit"],
43             stdout=subprocess.PIPE,
44             stderr=subprocess.STDOUT,
45             timeout=60
46         )
47         salida = r.stdout.decode(errors='replace')
48     except subprocess.TimeoutExpired:
49         return '<p style=color:red>Timeout: Scilab tardo mas de 60s  
</p>'
50     with open(ERROR_FILE, 'w') as f:
51         f.write(salida)
52     if not os.path.exists(RESULT_FILE):
53         return (f'<h3>Error Scilab</h3>'
54                 f'<pre style="color:red">{salida}</pre>')
55     with open(RESULT_FILE, encoding='utf-8') as f:
56         return f.read()
57
58 class Handler(BaseHTTPRequestHandler):
59     def log_message(self, fmt, *args):
60         print(fmt % args)
61
62     def do_GET(self):
63         html = leer_form().replace(MARCADOR,
64                                     '<div id=resultados></div>')
65         self._enviar(html)
66
67     def do_POST(self):
68         n = int(self.headers.get('Content-Length', 0))
69         body = self.rfile.read(n).decode()

```

```

70     params = {}
71     for par in body.split('&'):
72         if '=' in par:
73             k, v = par.split('=', 1)
74             params[unquote_plus(k)] = unquote_plus(v)
75     escribir_params(params)
76     div = llamar_scilab()
77     html_final = leer_form().replace(MARCADOR,
78                                     f'<div id=resultados>{div}</div>')
79     self._enviar(html_final)
80
81     def _enviar(self, html):
82         data = html.encode('utf-8')
83         self.send_response(200)
84         self.send_header('Content-Type',
85                         'text/html; charset=UTF-8')
86         self.send_header('Content-Length', len(data))
87         self.end_headers()
88         self.wfile.write(data)
89
90 HTTPServer(('', PUERTO), Handler).serve_forever()

```

4.3 formulario.html

El formulario es HTML estandar. El unico requisito es incluir el marcador `<!-- RESULTADOS -->` donde se insertaran los resultados, y usar `method=POST action=/'`.

Para que los checkboxes funcionen correctamente se agrega un campo oculto con el mismo nombre antes de cada uno, con valor `off`. El navegador envia `off` cuando el checkbox esta desmarcado y `on` cuando esta marcado; el servidor convierte ambos a booleanos Scilab:

Listing 2: Patron de checkbox correcto

```

1 <input type=hidden name=g_tray value=off>
2 <label>
3     <input type=checkbox name=g_tray value=on checked>
4     Trayectoria
5 </label>

```

4.4 calcular.sce

`calcular.sce` es el unico archivo especifico de la aplicacion. Debe:

1. Cargar los parametros con `exec('/tmp/params.sce', -1)`.
2. Realizar el calculo.
3. Opcionalmente generar graficas con `gnuplot`.
4. Escribir el fragmento HTML completo en `/tmp/resultado.html`.
5. Terminar con `quit`.

Listing 3: Estructura minima de calcular.sce

```

1 RESULT_FILE = '/tmp/resultado.html'
2 GRAPH_FILE  = '/tmp/proy_graph.png'
3
4 // 1. Cargar parametros
5 exec('/tmp/params.sce', -1)
6
7 // 2. Calculo
8 resultado = v0 * sin(angulo * %pi / 180) / 9.8
9
10 // 3. Grafica con gnuplot
11 NL = ascii(10); Q = ascii(34)
12 gp = 'set terminal png size 800,400' + NL
13 gp = gp + 'set output ' + Q + GRAPH_FILE + Q + NL
14 gp = gp + 'plot sin(x)' + NL
15 fd = mopen('/tmp/mi_plot.gnu', 'w')
16 mfprintf(fd, '%s', gp)
17 mclose(fd)
18 unix('gnuplot /tmp/mi_plot.gnu')
19
20 // 4. Convertir grafica a base64 y armar HTML
21 res = '<h2>Resultado</h2>'
22 res = res + sprintf('<p>Tiempo de subida: %.2f s</p>', resultado)
23 if isfile(GRAPH_FILE) then
24     unix('base64 -w 0 ' + GRAPH_FILE + ' > /tmp/g.b64')
25     b64 = mgetl('/tmp/g.b64')
26     res = res + '<img src=data:image/png;base64,' + b64 + '>'
27 end
28
29 fd = mopen(RESULT_FILE, 'w')
30 mfprintf(fd, '%s', res)
31 mclose(fd)
32 quit

```

4.5 Generacion de graficas con gnuplot

Gnuplot genera el PNG sin necesidad de pantalla ni display. 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 4: Construccin del script gnuplot

```

1 NL = ascii(10)    // salto de linea
2 Q  = ascii(34)    // comilla doble  "
3
4 gp = 'set terminal png size 800,400' + NL
5 gp = gp + 'set output ' + Q + '/tmp/grafica.png' + Q + NL
6 gp = gp + 'set xlabel ' + Q + 'x [m]' + Q + NL
7 gp = gp + 'set ylabel ' + Q + 'y [m]' + Q + NL
8 gp = gp + 'plot ' + Q + '/tmp/datos.txt' + Q + ..
9         ' using 1:2 with lines lw 2' + NL

```

```

10
11 fd = mopen('/tmp/script.gnu', 'w')
12 mfprintf(fd, '%s', gp)
13 mclose(fd)
14 unix('gnuplot /tmp/script.gnu 2>/tmp/gnu_err.txt')

```

El archivo de datos para gnuplot se escribe con `mfprintf`:

```

1 fd = mopen('/tmp/datos.txt', 'w')
2 for i = 1:length(x)
3     mfprintf(fd, '%.6f %.6f\n', x(i), y(i))
4 end
5 mclose(fd)

```

4.6 Conversion de imagen a base64

El PNG generado por gnuplot se convierte a base64 usando el comando del sistema y se incrusta en el HTML:

```

1 unix('base64 -w 0 /tmp/grafica.png > /tmp/grafica.b64')
2 b64 = mgetl('/tmp/grafica.b64')
3 img = '<img src=data:image/png;base64,' + b64 + ' alt=grafica>'

```

La opcion `-w 0` de `base64` produce una sola linea sin saltos, lo que permite leer el resultado directamente con `mgetl` como un scalar string.

4.7 Limitaciones conocidas

- El servidor atiende una solicitud a la vez. Para uso multiusuario simultaneo seria necesario un servidor con hilos o procesos.
- Scilab-cli no soporta la opcion `-nw` en esta version; no se debe incluir esa opcion al llamarlo.
- Las funciones graficas 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 (tipicamente 2-4 segundos). Para aplicaciones de alta frecuencia conviene mantener Scilab en memoria.

4.8 Diagnostico de errores

En caso de error, los archivos de log son:

Archivo	Contenido
/tmp/error_scilab.txt	Salida completa de Scilab
/tmp/params.sce	Parametros enviados a Scilab
/tmp/proy_gnuplot_err.txt	Errores de gnuplot

Flujo de diagnostico recomendado:

```
1 # 1. Ver que parametros recibio Scilab
2 cat /tmp/params.sce
3
4 # 2. Correr calcular.sce directamente
5 scilab-cli -e "exec('calcular.sce', -1); quit" 2>&1
6
7 # 3. Ver log completo
8 cat /tmp/error_scilab.txt
9
10 # 4. Ver errores de gnuplot
11 cat /tmp/proy_gnuplot_err.txt
12
13 # 5. Verificar que se genero el resultado
14 cat /tmp/resultado.html | head -5
15 ls -lh /tmp/proy_graph.png
```

5 Adaptar el sistema a otra aplicacion

Para crear una nueva aplicacion de calculo cientifico sobre este sistema:

1. Copiar los tres archivos a un nuevo directorio.
2. Editar `formulario.html`:
 - Cambiar el titulo y la descripcion.
 - 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. Reescribir `calcular.sce`:
 - Cargar parametros con `exec('/tmp/params.sce', -1)`.
 - Implementar el calculo cientifico.
 - Generar graficas con `gnuplot` si corresponde.
 - Escribir el HTML de resultados en `/tmp/resultado.html`.
 - Terminar con `quit`.
4. No modificar `servidor.py`.

5.1 Ejemplo: aplicacion de estadistica

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

Listing 5: formulario_estadistica.html (fragmento)

```

1 <form method=POST action=/>
2   <label>Datos (separados por comas)</label>
3   <input type=text name=datos value="1,2,3,4,5">
4   <label>Numero de clases del histograma</label>
5   <input type=number name=nclases value=5 min=2 max=20>
6   <input type=hidden name=g_hist value=off>
7   <label><input type=checkbox name=g_hist value=on checked>
8     Histograma</label>
9   <button type=submit>Calcular</button>
10 </form>
11 <!-- RESULTADOS -->

```

Listing 6: calcular_estadistica.sce (fragmento)

```

1 exec('/tmp/params.sce', -1)
2
3 // parsear el string de datos separados por coma
4 partes = strsplit(datos, ',')
5 n = size(partes, '*')
6 v = zeros(1, n)
7 for i = 1:n
8     v(i) = strtod(strtrim(partes(i)))
9 end
10
11 // calcular estadísticas
12 media = mean(v)
13 desv = stdev(v)
14 minimo = min(v)
15 maximo = max(v)
16
17 // grafica si se pidio
18 RESULT_FILE = '/tmp/resultado.html'
19 GRAPH_FILE = '/tmp/hist.png'
20
21 if g_hist then
22     NL = ascii(10); Q = ascii(34)
23     fd = mopen('/tmp/datos_hist.txt', 'w')
24     for i = 1:n
25         fprintf(fd, '%.6f\n', v(i))
26     end
27     mclose(fd)
28     gp = 'set terminal png size 700,350' + NL
29     gp = gp + 'set output ' + Q + GRAPH_FILE + Q + NL
30     gp = gp + 'binwidth = (' + sprintf('%.6f', maximo-minimo) + ')/'
31         + ..
32         string(nclases) + NL
33     gp = gp + 'set boxwidth binwidth' + NL
34     gp = gp + 'bin(x) = binwidth*floor(x/binwidth)' + NL
35     gp = gp + 'plot ' + Q + '/tmp/datos_hist.txt' + Q + ..
36         ' using (bin($1)):(1.0) smooth frequency with boxes'
37         + NL

```

```

36     fd = mopen('/tmp/hist.gnu', 'w')
37     mfprintf(fd, '%s', gp)
38     mclose(fd)
39     unix('gnuplot /tmp/hist.gnu')
40 end
41
42 res = '<h2>Estadísticas</h2>'
43 res = res + sprintf('<p>n=%d  media=%.4f  desvio=%.4f', n, media,
44     desv)
45 res = res + sprintf('  min=%.4f  max=%.4f</p>', minimo, maximo)
46
47 if isfile(GRAPH_FILE) & g_hist then
48     unix('base64 -w 0 ' + GRAPH_FILE + ' > /tmp/hist.b64')
49     b64 = mgetl('/tmp/hist.b64')
50     res = res + '<img src=data:image/png;base64,' + b64 + '>'
51 end
52
53 fd = mopen(RESULT_FILE, 'w')
54 mfprintf(fd, '%s', res)
55 mclose(fd)
56 quit

```

6 Código fuente del ejemplo: simulador de proyectil

Este es el ejemplo completo que se desarrollo y probó durante la elaboracion de este sistema. Incluye todos los tipos de controles HTML y una simulacion numerica con graficas.

6.1 formulario.html

Listing 7: formulario.html

```

1 <!DOCTYPE html>
2 <html lang=es>
3 <head>
4 <meta charset=UTF-8>
5 <title>Simulador de proyectil</title>
6 <style>
7 body{font-family:sans-serif;max-width:720px;
8     margin:20px auto;padding:0 16px}
9 label{display:block;font-weight:bold;margin:10px 0 3px}
10 input,select,textarea{width:100%;padding:6px;
11     box-sizing:border-box;border:1px solid #ccc;border-radius:4px}
12 .row{display:flex;gap:12px}
13 .row > div{flex:1}
14 button{margin-top:14px;padding:10px 24px;background:#2980b9;
15     color:white;border:none;border-radius:4px;
16     font-size:15px;cursor:pointer}
17 #resultados{margin-top:24px}
18 .stats{display:flex;gap:10px;flex-wrap:wrap;margin:12px 0}

```

```

19 .sb{flex:1;min-width:110px;background:#ecf0f1;padding:10px;
20   border-radius:6px;text-align:center}
21 .sv{font-size:1.4em;font-weight:bold}
22 img{max-width:100%;border:1px solid #ccc;
23   border-radius:6px;margin-top:8px}
24 </style>
25 </head>
26 <body>
27
28 <h2>Simulador de proyectil</h2>
29 <p>Complete los parametros y presione Calcular.</p>
30
31 <form method=POST action=/>
32
33   <label>Nombre del experimentador</label>
34   <input type=text name=nombre value="">
35
36   <div class=row>
37     <div>
38       <label>Velocidad inicial [m/s]</label>
39       <input type=number name=v0 value=50
40         min=1 max=500 step=1>
41     </div>
42     <div>
43       <label>Angulo de lanzamiento [grados]</label>
44       <input type=number name=angulo value=45
45         min=1 max=89 step=1>
46     </div>
47   </div>
48
49   <label>Resistencia del aire</label>
50   <select name=arr>
51     <option value=cero>Sin resistencia</option>
52     <option value=bajo selected>Baja (b=0.08)</option>
53     <option value=medio>Media (b=0.30)</option>
54     <option value=alto>Alta (b=1.50)</option>
55   </select>
56
57   <label>Metodo numerico</label>
58   <label><input type=radio name=met value=euler>
59     Euler</label>
60   <label><input type=radio name=met value=rk4 checked>
61     Runge-Kutta 4</label>
62   <label><input type=radio name=met value=ode>
63     ODE adaptativo</label>
64
65   <label>Graficas a mostrar</label>
66   <input type=hidden name=g_tray value=off>
67   <label><input type=checkbox name=g_tray value=on checked>
68     Trayectoria</label>
69   <input type=hidden name=g_vel value=off>

```

```

70  <label><input type=checkbox name=g_vel value=on>
71  Velocidad vs tiempo</label>
72
73  <label>Notas del experimento</label>
74  <textarea name=notas rows=3></textarea>
75
76  <button type=submit>Calcular</button>
77
78 </form>
79
80 <!-- RESULTADOS -->
81
82 </body>
83 </html>

```

6.2 calcular.sce

Listing 8: calcular.sce

```

1  RESULT_FILE = '/tmp/resultado.html'
2  GRAPH_FILE  = '/tmp/proy_graph.png'
3
4  function h = box(val, lab)
5      h = '<div class=sb><div class=sv>' + val + ..
6          '</div>' + lab + '</div>'
7  endfunction
8
9  function escribir_error(msg)
10     fd = mopen(RESULT_FILE, 'w')
11     fprintf(fd, '<p style=color:red><b>Error:</b> %s</p>', msg)
12     fclose(fd)
13 endfunction
14
15 // ----- cargar parametros escritos por Python -----
16
17 exec('/tmp/params.sce', -1)
18
19 if isnan(v0)      | v0      <= 0 then v0      = 50; end
20 if isnan(angulo) | angulo <= 0 then angulo = 45; end
21
22 select arr
23 case 'cero'   then b = 0
24 case 'bajo'   then b = 0.08
25 case 'medio'  then b = 0.30
26 case 'alto'   then b = 1.50
27 else          b = 0
28 end
29
30 // ----- simulacion -----
31
32 function R = simular(v0, ang_deg, b, metodo)

```

```

33 g = 9.8; m = 1.0
34 a = ang_deg * %pi / 180
35 y0 = [0; 0; v0*cos(a); v0*sin(a)]
36
37 function dydt = fode(t, y)
38     vmod = sqrt(y(3)^2 + y(4)^2)
39     fd = (b/m) * vmod
40     dydt = [y(3); y(4); -fd*y(3); -g-fd*y(4)]
41 endfunction
42
43 t_max = 3 * v0 * sin(a) / g + 1
44 n = 400
45 t = linspace(0, t_max, n)
46
47 select metodo
48 case 'euler' then
49     Y = zeros(4,n); Y(:,1) = y0; dt = t(2)-t(1)
50     for i = 1:n-1
51         Y(:,i+1) = Y(:,i) + dt*fode(t(i), Y(:,i))
52         if Y(2,i+1) < -0.1 then break, end
53     end
54 case 'rk4' then
55     Y = zeros(4,n); Y(:,1) = y0; dt = t(2)-t(1)
56     for i = 1:n-1
57         k1 = fode(t(i), Y(:,i))
58         k2 = fode(t(i)+dt/2, Y(:,i)+dt/2*k1)
59         k3 = fode(t(i)+dt/2, Y(:,i)+dt/2*k2)
60         k4 = fode(t(i)+dt, Y(:,i)+dt*k3)
61         Y(:,i+1) = Y(:,i) + dt/6*(k1+2*k2+2*k3+k4)
62         if Y(2,i+1) < -0.1 then break, end
63     end
64 else
65     sol = ode(y0, 0, t, fode); Y = sol
66 end
67
68 idx = find(Y(2,:) < 0)
69 n2 = n
70 if ~isempty(idx) then n2 = idx(1); end
71 R.t = t(1:n2)
72 R.x = Y(1,1:n2); R.y = Y(2,1:n2)
73 R.vx = Y(3,1:n2); R.vy = Y(4,1:n2)
74 R.v = sqrt(R.vx.^2 + R.vy.^2)
75 endfunction
76
77 ierr = execstr('R = simular(v0, angulo, b, met)', 'errcatch')
78 if ierr ~= 0 then
79     escribir_error('simulacion: ' + lasterror())
80     quit
81 end
82
83 // ----- grafica con gnuplot -----

```

```

84
85 if g_tray | g_vel then
86     fd = mopen('/tmp/proy_tray.txt', 'w')
87     for i = 1:length(R.t)
88         fprintf(fd, '%.6f %.6f %.6f %.6f %.6f\n', ..
89             R.t(i), R.x(i), R.y(i), R.vx(i), R.vy(i))
90     end
91     mclose(fd)
92
93     NL = ascii(10); Q = ascii(34)
94     DAT = Q + '/tmp/proy_tray.txt' + Q
95
96     gp = 'set terminal png size 800,400' + NL
97     gp = gp + 'set output ' + Q + GRAPH_FILE + Q + NL
98     gp = gp + 'set grid' + NL
99     if g_tray & g_vel then
100         gp = gp + 'set multiplot layout 1,2' + NL
101     end
102     if g_tray then
103         gp = gp + 'set xlabel ' + Q+'x [m]'+Q + NL
104         gp = gp + 'set ylabel ' + Q+'y [m]'+Q + NL
105         gp = gp + 'set title ' + Q+'Trayectoria'+Q + NL
106         gp = gp + 'plot ' + DAT + ..
107             ' using 2:3 with lines lw 2 lc ' + ..
108             Q+'blue'+Q+' notitle' + NL
109     end
110     if g_vel then
111         gp = gp + 'set xlabel ' + Q+'t [s]'+Q + NL
112         gp = gp + 'set ylabel ' + Q+'v [m/s]'+Q + NL
113         gp = gp + 'set title ' + Q+'Velocidad'+Q + NL
114         gp = gp + 'plot ' + DAT + ..
115             ' using 1:(sqrt($4*$4+$5*$5))' + ..
116             ' with lines lw 2 lc '+Q+'blue'+Q+..
117             ' title '+Q+'|v|'+Q+', \' + NL
118         gp = gp + ' ' + DAT + ..
119             ' using 1:4 with lines lw 1 lc ' + ..
120             Q+'red'+Q+' title '+Q+'vx'+Q+', \' + NL
121         gp = gp + ' ' + DAT + ..
122             ' using 1:5 with lines lw 1 lc ' + ..
123             Q+'green'+Q+' title '+Q+'vy'+Q + NL
124     end
125     if g_tray & g_vel then
126         gp = gp + 'unset multiplot' + NL
127     end
128
129     fd = mopen('/tmp/proy_plot.gnu', 'w')
130     fprintf(fd, '%s', gp)
131     mclose(fd)
132     unix('gnuplot /tmp/proy_plot.gnu 2>/tmp/gnu_err.txt')
133 end
134

```

```

135 // ----- resultado HTML completo -----
136
137 res = '<h2>Resultados</h2><div class=stats>'
138 res = res + box(sprintf('%.1f m', R.x($)), 'Alcance')
139 res = res + box(sprintf('%.1f m', max(R.y)), 'Altura max')
140 res = res + box(sprintf('%.2f s', R.t($)), 'Tiempo vuelo')
141 res = res + box(sprintf('%.1f m/s', R.v($)), 'V impacto')
142 res = res + '</div>'
143 res = res + '<p>v0=' + sprintf('%.0f', v0) + ' m/s '
144 res = res + 'angulo=' + sprintf('%.0f', angulo) + ' grados '
145 res = res + 'b=' + sprintf('%.2f', b) + ' '
146 res = res + 'metodo=' + met + ' </p>'
147
148 if isfile(GRAPH_FILE) then
149     unix('base64 -w 0 ' + GRAPH_FILE + ' > /tmp/graph.b64')
150     b64 = mgetl('/tmp/graph.b64')
151     if length(b64) > 0 then
152         res = res + '<img src=data:image/png;base64,' + ..
153             b64 + ' alt=grafica>'
154     end
155 end
156
157 fd = mopen(RESULT_FILE, 'w')
158 mfprintf(fd, '%s', res)
159 mclose(fd)
160 quit

```