Prácticas de laboratorio de Métodos Numéricos –Soluciones a algunos ejercicios–

Pedro Fortuny Ayuso

Curso 2021/22, EPIG, Gijón. Universidad de Oviedo
Email address: fortunypedro@uniovi.es

Índice general

1. Capítulo 1

Ejercicio 1: La función `min3`:

```
% dada una lista v
% min3(v) devuelve una lista con tres elementos [m1 m2 m3]
% m1 es el minimo de v
% m2 es el siguiente
% m3 es el tercero
function [y] = min3(v)
  % inf es mayor que cualquier numero
  y = [inf inf inf];
  for k = v
    if k < y(1)
      y(3) = y(2);
      y(2) = y(1);
      y(1) = k;
    elseif k < y(2)
      y(3) = y(2);
      y(2) = k;
    elseif k < y(3)
      y(3) = k;
    end
  end
end
```

LISTADO 1. Función `min3`.

Ejercicio 2: Función `es_creciente`:

```
% es_creciente(v)
% devuelve 0 si la lista v NO es creciente, 1 si lo es
function [T] = es_creciente(v)
  T = 0;
  k = 1;
  while (k+1 <= length(v) && v(k+1) >= v(k))
    k = k +1 ;
  end
  if (k+1 > length(v))
    T = 1;
  end
end
```

LISTADO 2. Función `es_creciente`.

Ejercicio 3: Función `es_creciente2`:

```
% es_creciente2(v) devuelve [T k]
% T es 1 si v es creciente, 0 si no
% k es la longitud de la secuencia creciente inicial mas larga
function [T k] = es_creciente2(v)
  T = 0;
  k = 1;
  while(k+1 <= length(v) && v(k) <= v(k+1))
    k = k + 1;
  end
  if k + 1 > length(v)
    T = 1;
  end
end
```

LISTADO 3. Función `es_creciente2.m`.

Ejercicio 4: Función `positiva`:

```
% positiva(v)
% dada una lista v,
% devuelve dos valores: n, w
% n: es el numero de valores positivos de v (>0)
% w: la lista de dichos valores
function [n w] = positiva(v)
  n = 0;
  w = [];
  % bucle for: he de mirar TODA la lista v
  for k = v
    if (k > 0)
      n = n + 1;
      % "end" es el ultimo, asi que
      % "end+1" significa "el siguiente al ultimo"
      w(end+1) = k;
    end
  end
end
```

LISTADO 4. Función `positiva`.

Ejercicio 5: Función `secante`:

```
% secante(f,x0,ep) devuelve m, n donde
% Y= mX + n es la ecuacion de la curva que pasa por
% (x0-ep, f(x0-ep)) y (x0+ep, f(x0+ep))
function [m n] = secante(f, x0, ep)
  % no es mas que la ecuacion de la curva que pasa por dos ptos
  % lo IMPORTANTE es hacer un dibujo
```

```
  % estas asignaciones no son mas que para "aclarar" las cosas
  % punto (x1, y1)
  x1 = x0 - ep;
  y1 = f(x1);
  % punto (x2, y2)
  x2 = x0 + ep;
  y2 = f(x2);
  % pendiente
  m = (y2 - y1)./(x2 - x1);
  % intercepcion
  n = -m .* x1 + y1;
end
```

LISTADO 5. Función `secante`.

2. Capítulo 2

Ejercicio 10: Función que implementa el método de bisección:

```
% biseccion(f, a, b, e, N)
% dada f continua, que se supone que cambia de signo entre a y b
% devuelve un numero r:
% si |f(c)|< e antes de N pasos, entonces c = r
% si no, c = -inf
%
% esta funcion NO COMPRUEBA que f cambie de signo
function [c n] = biseccion(f, a, b, e, N)
  % vale empezar en 1 o en 0, no pasa nada
  n = 1;
  c = (b + a)/2;
  % valor absoluto: abs
  while (abs(f(c)) > e && n < N)
    if (f(c)*f(a) < 0)
      b = c;
    else
      a = c;
    end
    c = (b + a)/2;
    n = n + 1;
  end
  % OJO: lo contrario de n<N es n>=N
  if n >= N
    c = -inf;
  end
end
```

LISTADO 6. Función `biseccion`.

Ejercicio 11: Dibujemos la gráfica de la función $f(x) = \cos(e^x)$ en el intervalo $x \in [0,2]$:

```
% definicion de la funcion
f = @(x) cos(exp(x));
% intervalo con bastantes puntos
U = [0:.01:2];
% grafica
plot(U, f(U));
% aunque en este caso esta claro, dibujo el eje OX para ver bien
hold on
% tengo que multiplicar por 0 para obtener una lista de ceros
plot(U, 0*U)
```

En la gráfica se ve que la función cambia dos veces de signo en el intervalo. Como $f(0) = \cos(1)$ y $f(2) = \cos(e^2)$ tienen el mismo signo, ha de subdividirse el intervalo para poder utilizar bisección. Como $\cos(e)$ es negativo, se puede tomar $x_1 = 1$ y estudiar los intervalos $I_1 = [0,1]$ e $I_2 = [1,2]$. Tomemos $\epsilon = 10^{-6}$ y pongamos un numero maximo de iteraciones grande $N = 100$. Utilizando la definición de f dada arriba, se trabaja así:

```
e = 1e-6;
N = 100;
r1 = biseccion(f, 0, 1, e, N)
r2 = biseccion(f, 1, 2, e, N)
```

(No se muestra la salida por pantalla). Para comprobar que los resultados tienen sentido, ha de verificarse que $r_1 \in [0,1]$ y $r_2 \in [1,2]$, lo cual es obvio a la vista de sus valores. Además, ha de comprobarse que $|f(r_1)| < 10^{-6}$ y que $|f(r_2)| < 10^{-6}$:

```
abs(f(r1)) < 1e-6
```

Si la salida es un 1, es cierto. Si la salida es un 0, es falso. Lo mismo pasa con

```
abs(f(r2)) < 1e-6
```

Ejercicio 12: Hagamos el mismo caso que para el Ejercicio 11: $\cos(e^x)$ para $x \in [0,2]$. Lo difícil en el caso de Newton-Raphson es encontrar una semilla que sirva para acercarse a la raíz deseada. Probemos primero, por ejemplo, con $x_0 = 0$ para ver si se acerca a la raíz de la izquierda. Utilizamos un N grande por si acaso y un error de 10^{-6}:

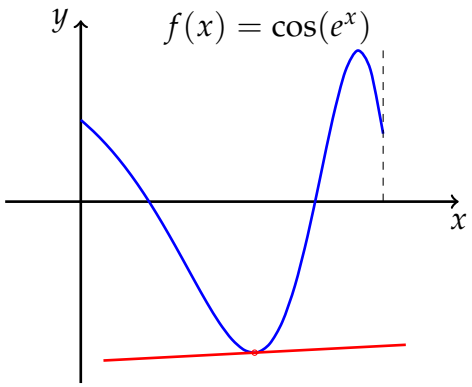

FIGURA 1. La función $\cos(e^x)$ para $x \in [0, 2]$ y su tangente para $x = 1.15$: mala semilla.

```
x0 = 0;
N = 100;
e = 1e-6;
f = @(t) cos(exp(t));
% hace falta la derivada, claro
fp = @(t) exp(t).*(-sin(exp(t))); % con punto por si acaso
% veamos que sale
r1 = newtonraphson(f, fp, x0, e, N)
```

se obtiene un valor parecido a $r_1 = 0.4516$. Para verificar que es una raíz aproximada,

```
abs(f(r1))<e
```

y se obtiene un 1, es decir: sí que el error es menor que 10^{-6}. Así que r_1 es la raíz de $f(x)$ que está entre 0 y más o menos 0.5.

Para calcular la segunda raíz, probemos primero con $x_0 = 1.15$ a ver qué ocurre:

```
x0 = 1.15;
% el resto de definiciones son iguales porque la funcion no cambia
r2 = newtonraphson(f, fp, x0, e, N)
```

se obtiene $r_2 \simeq 20.222$ que es una raíz aproximada pues $|f(r_2)| < 10^{-6}$ pero no nos sirve, pues estamos buscando las raíces entre 0 y 1. Lo que ha ocurrido es que la derivada es casi 0 y el valor de x_1 es enorme y está muy lejos del intervalo (ver Figura 1). Además, esta función tiene muchas raíces en $[0, \infty)$. Necesitamos una semilla mejor. Probemos con $x_0 = 1.5$.

```
x0 = 1.5;
r2 = newtonraphson(f, fp, x0, e, N)
```

Se obtiene $r_2 \simeq 1.5502$. Es raíz aproximada pues

```
abs(f(r2)) < 1e-6
```

devuelve un 1 (es verdad) y está en el intervalo. Hemos terminado el ejercicio. —

Ejercicio 13: Solo hace falta cambiar el nombre a la función (y, obviamente, al archivo, que debe llamarse `newtonraphson_plot.m`) y añadir una línea con el `plot` (justo después del `while`). Hemos incluido un `hold on` también antes del while porque se supone que el usuario ha dibujado antes la gráfica.

```
% Newton-Raphson con plot
% Devuelve la raiz aproximada y el numero de iteraciones
% Ademas, dibuja cada xi en el eje OX
% Input:
% f, fp (derivative), x0, epsilon, N
function [z n] = newtonraphson_plot(f, fp, x0, epsilon, N)
  n = 0;
  xn = x0;
  % inicializamos z (la salida) a NaN (i.e. por defecto, error)
  z = NaN;
  % suponemos que el usuario ha dibujado la grafica, claro
  hold on
  while(abs(f(xn)) >= epsilon && n <= N)
    plot(xn, 0, '*'); % para que se vea bien
    n = n + 1;
    % siguiente paso
    xn = xn - f(xn)/fp(xn); % podria haber error (1/0) pero no nos preocupa
  end
  z = xn;
  if(n == N)
    warning('No se ha alcanzado la tolerancia');
  end
end
```

—

Ejercicio 15: El código es muy parecido al de Newton-Raphson aunque hay que ser cuidadoso con la entrada (que ahora tiene una sola función pero dos semillas). Podría ser así:

```
% Secant: metodo de la secante
% devuelve
% z: raiz aproximada
% n: numero de iteraciones
% Entrada:
% f, x0, x1, epsilon, N
function [z n] = secant(f, x0, x1, epsilon, N)
```

```
  n = 0;
  % por defecto, damos a z un valor de "error" (Not a Number)
  z = NaN;
  while(abs(f(x1)) >= epsilon && n <= N)
    n = n + 1;
    % recuerdo x1 para despues
    xT = x1;
    % siguiente punto, lo llamo x1 otra vez
    x1 = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0));
    x0 = xT;
  end
  z = x1;
  if(n == N)
    warning('No se ha alcanzado la tolerancia');
  end
end
```

Ejercicio 20: Cuidado con las unidades.

Un cilindro tiene volumen $V_c = \pi r^2 h$, donde r es su radio y h su altura. Por tanto, la masa del cilindro es

$$M_c = \pi r^2 h \rho = \pi \cdot 2^2 \text{cm}^2 \cdot 100\text{cm} \cdot 16.6\text{g/cm}^3 = 20860.17\text{g} = 20.860\text{kg}.$$

Una esfera tiene volumen $V_e = \frac{4}{3}\pi r^3$, así que la masa (en kilogramos) de una esfera de acero de radio r es

$$M_e(r) = \frac{4}{3}\pi \cdot r^3 \cdot 7850\text{kg}/m^3.$$

Se nos pide que $M_e(r)$ sea igual a 20.860kg (veinte y pico, ojo, no veinte mil). Hemos de resolver la ecuación, para r:

$$7850\text{kg/m}^3 \frac{4}{3} r^3 = 20.860\text{kg}$$

donde r está en metros, o lo que es lo mismo

$$r^3 - \frac{3}{4}\frac{20.860}{7850} = 0.$$

Si tomamos $f(r) = r^3 - 20.860 * 3/(7850 * 4)$, hemos de calcular una raíz de esta función. Utilizando Newton-Raphson, $f'(r) = 3r^2$, podemos tomar como semilla $x_0 = 1$, una tolerancia de 10^{-5} y 100 pasos. Veamos qué se obtiene:

```
f = @(r) r.^3 - 20.860*3/(7850*4);
fp = @(r) 3*r.^2;
R = newtonraphson(f, fp, 1, 10^-5, 100)
```

R es 0.12584 (en unidades de metros, ojo), aproximadamente, unos 12cm. La masa de esta esfera es, por hacer la comprobación, en kilogramos:

```
format long % pone muchos decimales
4/3*R^3*7.850
```

la respuesta es 20.8601 . . ., que es lo que se buscaba (aprox). —

Ejercicio 21: Este ejercicio es mucho más sencillo de lo que parece. Lo único que hay que hacer es definir las constantes, la función cuya solución quiere calcularse y buscar su raíz. Las constantes:

```
g = 10;
% cuerpo 1
m1 = 1;
x1 = 100; % altura inicial
v1 = 1; % hacia arriba: +1
k1 = 0.02;

% cuerpo 2
m2 = 2;
x2 = 95;
v2 = 1;
k2 = 0.015;
```

La posición del primer cuerpo es directamente la función que se nos da:

```
pos1 = @(t) (-m1 *exp(-k1*t/m1)*(g*m1+k1*v1) + g*m1*(m1-k1*t) + x1*k1^2 + k1*m1*
    v1)/k1^2;
```

La posición del segundo cuerpo es la misma (luego nos preocuparemos del "un segundo más tarde"):

```
pos2 = @(t)(-m2 *exp(-k2*t/m2)*(g*m2+k1*v2) + g*m2*(m2-k2*t) + x2*k2^2 + k2*m2*
    v2)/k2^2;
```

Lo que queremos saber es si "coinciden", es decir, si en algún momento *relativo al lanzamiento del primer objeto*, la posición del primero en el instante t es la misma que la posición del segundo para tiempo $t-1$ (pues sale un segundo más tarde, "vuela menos tiempo"). Así que queremos encontrar una raíz de $pos_1(t) - pos_2(t-1)$:

```
f = @(t) pos1(t) - pos2(t-1);
```

Antes de nada, dibujamos la gráfica en un tiempo grande:

```
T = [0:0.1:10];
plot(T, f(t));
```

Se ve que hay un cero entre 4 y 8. Utilicemos bisección (por no derivar la expresión tan grande de la función $f(t)$):

```
r1 = biseccion(f, 4, 8, 1e-6, 100)
```

El resultado es más o menos 5.87. Comprobemos que es realmente una raíz aproximada:

```
f(r1)
```

da como salida un número menor que 10^{-6}, así que esa es la solución.

—

Ejercicio 22: Lo único necesario es *tener cuidado* y plantear bien el problema. Para ello, es importante *hacer un dibujo* como en la Figura 2.

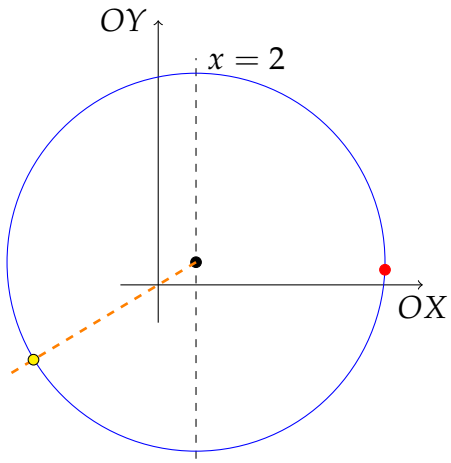

FIGURA 2. Esquema del ejercicio 22. La circunferencia rodea el origen.

El punto rojo es el inicio del movimiento, el negro es el centro de la circunferencia y el amarillo es la solución del problema. ¿Cómo se conoce el punto amarillo? Es el punto de corte entre la recta que une el origen con el centro de la circunferencia y está en el semiplano $x < 2$. ¿Por qué? Porque una recta corta a una circunferencia perpendicularmente solo si la recta pasa por el origen. La recta que pasa por $(2, 3)$ y $(0, 0)$ tiene ecuación

$$Y = \frac{3}{2}X$$

y la circunferencia recorrida por la partícula tiene ecuación

$$(X - 2)^2 + (Y - 3)^2 = 400.$$

Los puntos de corte son, sustituyendo, las soluciones de:

$$(X-2)^2 + (3/2X-3)^2 = 400.$$

Hace falta resolver esta ecuación. Para ello utilizamos, por ejemplo, bisección (así nos evitamos las derivadas). La función que usaremos es:

$$f(X) = (X-2)^2 + (3/2X-3)^2 - 400.$$

Y buscamos una solución con $x < 2$ (ojo con esto!). Sabemos, por ejemplo, que $f(-25) > 0$ y $f(0) < 0$ (compruébese). Con una tolerancia de 10^{-6} y un número máximo de iteraciones de 100, se obtiene

```
f = @(x) (x-2)^2 + (3/2*x -3).^2 - 400;
f(0)*f(-25) % da negativo, ok
x0 = biseccion(f, -25, 0, 1e-6, 100) % 0 es -9.0940
```

y, por tanto, $y_0 = 3/2x_0 = -13.641$. Ahora hemos de calcular el instante.

El movimiento circular de la partícula es, en las unidades del problema:

$$x(t) = 2\text{km} + 20km \cdot \cos(\omega t), y(t) = 3\text{km} + 20\text{km} \cdot sin(\omega t),$$

donde hemos de calcular ω para que la velocidad lineal sea de $20\text{m}/s$. La velocidad vectorial es

$$\dot{x}(t) = -20\text{km}\omega \text{ sen}(\omega t)/\text{s}, \dot{y}(t) = 20\text{km}\omega \cos(\omega t)/\text{s}$$

y por tanto, la velocidad lineal instantánea es

$$|v| = 20\omega\text{km}/s.$$

Como se quiere que sea $0.02\text{m}/s$, el valor de ω es $\omega = 2 \cdot 10^{-3}$.

Necesitamos conocer la longitud del arco descrito por la partícula. Como comienza en $(22, 3)$, el ángulo inicial respecto de su centro es 0. El ángulo final es el que forma el punto $(2, 3)$ con el (x_0, y_0) que, por construcción, es el mismo que forma el origen con (x_0, y_0), es decir $\tan(\alpha) = 3/2$, pero "en el tercer cuadrante". Si usamos bisección otra vez, sabemos que $\tan(pi) = 0$ y $\tan(pi + pi/3) > 3/2$, así que el ángulo lo podemos calcular así:

```
f = @(x) tan(x) - 3/2;
a = biseccion(f, pi, pi+pi/3, 1e-6, 100); % a es 4.1244
```

(ojo, esto es en radianes, algo más de 180 grados, claro). Para llegar a este ángulo, ha de ocurrir que $\omega t = 4.124$, es decir $t = 4.124/0.002 \simeq 2062$ segundos (algo más de media hora).

3. Capítulo 4

Ejercicio 33: Aquí no hay más que comparar las salidas de cada ejercicio. Para ello, en lugar de utilizar la misma variable y en ambos, se requiere utilizar en uno una, digamos y y en el otro otra, digamos z. Para comparar las diferencias, por ejemplo, puede hacerse entonces

```
d = abs(y-z); % esto da los valores absolutos de las diferencias
plot(d) % para hacerse una idea
max(d) % para ver el error maximo
d(end) % para conocer el error al final
r = d./abs(y) % si y fuera "exacta", el error relativo
plot(r) % dibuja los errores relativos
```

por ejemplo. Hay muchas maneras de hacer este ejercicio. —

Ejercicio 35: Solo el primero, el resto son iguales (definiendo la función correctamente y eligiendo el intervalo adecuado, claro).

```
f1 = @(x,y) 2*x; % OJO: la funcion es de (x,y) aunque solo haya 'x'
% 10 puntos entre 0 y 1
v = linspace(0, 1, 10);
y0 = 1;
s1 = euler(f1, v, y0)
plot(v, s1, '*')
```

Se han dibujado puntos porque lo que obtiene el método de Euler son valores discretos. Si ahora queremos dibujar la solución exacta en dichos puntos:

```
hold on
plot(v, v.^2+1,'o')
```

Obsérvese cómo la opción `'o'` hace que solo aparezcan puntos. Sin ella, Matlab dibujaría una sucesión de segmentos uniéndolos y no queremos eso. Ahora dibujemos la función $x^2 + 1$ con más puntos (y en azul):

```
v2 = linspace(0,1,100);
plot(v2, v2.^2+1,'b.')
```

—

Ejercicio 36: El listado sigue. Obsérvese cómo, en cada paso, que corresponde a `x(s-1)`, la pendiente en el punto "final de Euler" se ha de calcular utilizando `x(s)` (el tiempo actualizado), este es quizás el punto más delicado de todo el código y del algoritmo.

```
% Metodo de Heun para integrar EDOs numericamente
% ENTRADA:
% 1) Una funcion anonima f, de dos variables (importante)
% 2) Un vector con las coordenadas x de los puntos en que aproximar la solucion
% 3) un valor inicial y0 correspondiente a x0 (que es x(1))
% SALIDA:
% un vector de valores y(i) que aproximan la solucion en x(i)
function [y] = heun(f, x, y0)
  % se creea la lista y con la misma longitud que x
  y = zeros(size(x));
  % se almacena la condicion inicial en el primer punto
  y(1) = y0;
  % Bucle de Heun
  for s = 2:length(x)
    h = x(s) - x(s-1); % este es el paso
    pe = f(x(s-1), y(s-1)); % pte. euler
    ye = y(s-1) + pe * h;   % valor predicho por euler
    pf = f(x(s), ye);       % pte. "final": OJO a x(s)!
    pm = (pe + pf)/2;       % pte. media
    y(s) = y(s-1) + h * pm; % valor de heun
  end
end
```

La utilización como en el otro ejercicio es idéntica y no la incluimos.

—

4. Capítulo 5

Ejercicio 38: Como se verá, la implementación es *casi igual* que la función de Heun del capítulo anterior excepto que hay que indicarle a Matlab que la función tiene varias filas (hay varias variables dependientes). Esto se hace con la expresión `y(:,s-1)` (para el "momento actual") y con `y(:,s)` (para el momento predicho) en donde corresponde.

En el fondo, lo que se hace es calcular las pendientes de Euler, la predicción, la pendiente final y la media *en todas las variables a la vez*. La diferencia entre `eulervector` y `heunvector` es la misma que entre `euler` y `heun`, si se miran los ejercicios del capítulo anterior. El código sigue.

```
% Metodo de Heun para integracion de EDO de varias variables
% ENTRADA:
% 1) una funcion anonima f(x, y)
%    de dos variables
%    x: numerica
%    y: vector columna
```

```
% 2) Un vector fila x que es la lista de valores xi
% 3) Un vector columna y0, que es la lista de condiciones iniciales

% SALIDA:
% una matriz con mismo numero de filas que y0 y de columnas que x
% con la solucion aproximada dada por heun para el problema de c.i.
% Y' = f(X,Y); Y(0) = y0
function [y] = heunvector(f, x, y0)
  % creamos la matriz de salida con el tamanyo requerido
  y = zeros(length(y0),length(x));
  % ponemos en la primera posicion las condiciones iniciales
  y(:,1) = y0;
  % pucle de Heun, "todas las coordenadas a la vez"
  for s = 2:length(x)
    % pendiente de Euler
    pe = f(x(s-1), y(:,s-1));
    % prediccion de Euler
    ye = y(:,s-1) + (x(s) - x(s-1))*pe;
    % pendiente final
    pf = f(x(s), ye);
    % pendiente media
    pm = (pe + pf)/2;
    % punto siguiente de Heun
    y(:, s) = y(:,s-1) + (x(s) - x(s-1))*pm;
  end
end
```

—

Ejercicio 39: Hagámoslo con los dos métodos: Euler y Heun (pero la versión vectorial de ambos, claro).

```
% la funcion es la misma para ambos metodos
f = @(t, x) [
  0.8*x(1) - 0.4*x(1)*x(2) ;
  -2*x(2) + 0.2*x(1)*x(2)
  ];
% tiempo y condicion inicial
T=[0:.1:12];
X0 = [18;3];
% con euler
Xe = eulervector(f, T, X0);
plot(T, Xe)
hold on
% con heun
Xh = heunvector(f, T, X0);
plot(T, Xh)
```

Es interesante ver cómo cambia la precisión de Euler/Heun para tiempos más largos:

```
clf % limpiamos la grafica
T2 = [0:.1:36]; % bastante mas tiempo
Xe2 = eulervector(f, T2, X0);
Xh2 = heunvector(f, T2, X0);
plot(T, Xe2);
hold on
plot(T, Xh2);
```

El método de Euler en seguida "enloquece" mientras que Heun se mantiene coherente. ¿Qué pasa si hacemos lo mismo pero utilizando pasos de `0.01` en el método de Euler, en lugar de `0.1`?

Para calcular máximos y mínimos, Matlab posee las instrucciones `max` (máximo) y `min` (mínimo), pero dejo esta parte del problema para quien desee hacerla. —

Ejercicio 40: Este problema solo requiere expresar correctamente la función del modelo, utilizando un vector columna de tamaño 3 en la variable dependiente. Se definen primero los parámetros:

```
a = 0.1;   % no es necesario llamarlos "alfa" y "beta"
b = 0.05;
% ahora puedo definir f
% y(1) es S
% y(2) es I
% y(3) es R
f = @(t, y) [
  -a*y(1).*y(2);
  -b*y(2) + a*y(1).*y(2);
  b*y(2)
];
```

Ya tenemos el modelo. Ahora se resuelve el problema de condición inicial. Primero con Euler y luego con Heun:

```
T   = linspace(0, 1500, 2000); % me dicen linspace
Y0  = [1; 0.001; 0];
Se = eulervector(f, T, Y0);
Sh = heunvector(f, T, Y0);
plot(T, Se);
hold on
plot(T, Sh);
```

Se trata ahora de que el alumno realice más pruebas (cambiando `T`, cambiando `Y0`, cambiando ambas cosas, comparando Euler y Heun...).

—

Ejercicio 41: Se pide cambiar la evolución de los infectados *¡y de los susceptibles!*, así que hay que modificar el valor de $\dot{I}(t)$ y de $\dot{S}(t)$ como sigue (en negrita lo nuevo):

$$\begin{cases} \dot{S}(t) = -\alpha S(t)I(t) + \boldsymbol{\gamma I(t)} \\ \dot{I}(t) = -\beta I(t) + \alpha S(t)I(t) - \boldsymbol{\gamma I(t)} \end{cases}$$

y para expresar esto en Matlab, poniendo $\gamma = 0.005$ (recuérdese que $R(t)$ sigue existiendo):

```
c = 0.005 % parametro gamma
f = @(t, y) [
  -a*y(1).*y(2) + c*y(2);
  -b*y(2) + a*y(1).*y(2) - c*y(2);
  b*y(2)
];
```

El resto del ejercicio es igual que el anterior (se sugiere, claro, hacer varias gráficas, comparar con el ejercicio anterior...). —

Ejercicio 42: La modificación, ahora, solo afecta a la evolución de $S(t)$, así que el valor de $\dot{S}(t)$ nuevo (respecto del ejercicio anterior) es:

$$\dot{S}(t) = -\alpha S(t)I(t) + \gamma I(t) + \boldsymbol{\delta(S(t) + I(t) + R(t))}$$

donde δ es el parámetro de crecimiento de la población. En Matlab, si $\delta = 0.002$:

```
d = 0.002 % parametro gamma
f = @(t, y) [
  -a*y(1).*y(2) + c*y(2) + d*(y(1) + y(2) + y(3));
  -b*y(2) + a*y(1).*y(2) - c*y(2);
  b*y(2)
];
```

etc. —

Ejercicio 45: Puesto que ya nos dan la ecuación de segundo orden, no tenemos más que convertirla en un par de ecuaciones de primer

orden. Llamando ω a la velocidad angular (es decir, a $\dot{\theta}$) y despejando:

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = \dfrac{1}{ml}\left(-g\ \text{sen}(\theta) - k\omega\right) \end{cases}$$

y los datos son $g = 9.81$, $k = 0.1$, $m = 0.5$ y $l = 1$ (del ejercicio anterior, algunas). Por tanto, en Matlab:

```
g = 9.81;
k = 0.1;
m = 0.5;
l = 1;
P2 = @(t, y) [
  y(2);
  1/(m*l) * (-g*sin(y(1)) - k*y(2))
];
```

y ahora hemos de resolver para las mismas condiciones iniciales, etc.

```
T = [0:.01:50];
Y0 = [pi/4; 0];
Yroz = heunvector(P2, T, Y0);
plot(T, Yroz);
```

Se observa la pérdida de energía con el paso del tiempo, debida al rozamiento. —

Ejercicio 46: Solo voy a expresar las funciones correspondientes de Matlab. Hay que poner las ecuaciones de segundo orden como sistemas de ecuaciones de primer orden. En ambos casos se puede definir v (la velocidad) como la derivada de x (es decir, $v = \dot{x}$) y por ello, los sistemas correspondientes son:

$$\begin{cases} \dot{x} = v \\ \dot{v} = -\dfrac{k}{m}x \end{cases}$$

que, en Matlab, para nuestras funciones, se pondría como (suponiendo que $k = 0.2$ y $m = 3$, por ejemplo):

```
k = 0.2;
m = 3;
f = @(t, x)[
  x(2);
  -k/m * x(1) ];
```

Para el caso con rozamiento:

$$\begin{cases} \dot{x} = v \\ \dot{v} = -\frac{k}{m}x - \frac{r}{m}v \end{cases}$$

y esto en Matlab, si por ejemplo $r = 0.01$:

```
r = 0.01;
f = @(t, x)[
  x(2);
  1/m*(-k*x(1) - r*x(2)) ];
```

—

Ejercicio 47: Este ejercicio es más complicado porque es un sistema de dos ecuaciones de segundo orden (una para cada coordenada), así que hemos de convertirlo en uno de cuatro variables, haciendo $v_x = \dot{x}$ y $v_y = \dot{y}$ (las componentes x e y de la velocidad). Con esto, el sistema entero queda sencillo (las ecuaciones no tienen nada de complicado, claro):

$$\begin{cases} \dot{x} = v_x \\ \dot{v}_x = 0 \\ \dot{y} = v_y \\ \dot{v}_y = -g \end{cases}$$

donde $g = 9.81$. La posición inicial es $x = 0, y = 0$ y la velocidad inicial es $v_x = 1$, $v_y = 1$. Por tanto, en Matlab:

```
g = 9.81;
f = @(t, x)[
  x(2);
  0;
  x(4);
  -g ];
x0 = [0; 0; 1; 1];
T = [0:.01:20];
s = heunvector(f, T, x0);
plot(T, s);
```

Se obtiene el dibujo de una parábola. Con el rozamiento que se indica, la ecuación sería:

$$\ddot{x} = -kv_x$$
$$\ddot{y} = -kv_y$$

donde k es la constante de rozamiento. La función de Matlab, si $k = 0.05$:

```
k = 0.05;
f = @(t, x)[
  x(2);
  -k*x(2);
  x(4);
  -g - k*x(4) ];
```

y ahora la curva no es tan parabólica (precisamente por el rozamiento). —

5. Capítulo 6

Ejercicio 48: No hay más que definir las funciones necesarias, en este caso $1, t, t^2$ y t^3 y hacer las mismas operaciones:

```
f1 = @(t) 1+0.*t;
f2 = @(t) t;
f3 = @(t) t.^2;
f4 = @(t) t.^3;
% se supone que las listas de valores x, y ya estan asignadas
X = [f1(x); f2(x); f3(x); f4(x)];
A = X*X';
Y = X*y';
coefs = A\Y
```

El alumno verá si se parecen o no los coeficientes. —

Ejercicio 49: Para introducir la tabla:

```
x = [2.0 2.7 3.4 4.1 4.8 5.5 6.2 6.9]; % lo que corresponda
y = [11.39 15.31 18.18 19.80 19.76 15.03 1.74 -29.67];
```

Como nos dicen cuáles son las funciones del modelo: $\log(x), x, e^x$, las definimos:

```
f1 = @(x) log(x);
f2 = @(x) x;
f3 = @(x) exp(x);
```

Y ahora no hay más que realizar las mismas operaciones que en el ejercicio anterior, con solo 3 funciones:

```
X = [f1(x); f2(x); f3(x)];
A = X*X';
Y = X*y';
coefs = A\Y
```

—

Ejercicio 51: Hay que tener cuidado porque en este ejercicio *solo hay una función*. La energía cinética de un cuerpo se calcula, físicamente, como

$$E = \frac{1}{2}mv^2$$

donde v es la velocidad y m la masa. En este problema, se tiene una correspondencia entre velocidades y energías, y se quiere conocer el parámetro que falta, la masa. Para ello se puede tomar como función $f(v) = v^2$, y el parámetro que se obtenga será $a = m/2$, por tanto, m será el doble del coeficiente. Así pues,

```
v = [1.0 1.5 2.3 2.7 3.0];
E = [8.05 16.97 39.69 55.58 66.91];
f = @(t) t.^2;
```

y las operaciones son las mismas, aunque aquí en lugar de un sistema queda una ecuación:

```
X = [f(v)];
A = X*X';
Y = X*E';
A\Y
```

se obtiene el coeficiente $a = m/2$. La masa es, como ya se ha dicho, el doble. —

Ejercicio 52: Este ejercicio debe ser fácil. Un m.u.a. sigue una fórmula $d = d_0 + v_0 t + \frac{a}{2}t^2$, donde d_0 es la posición inicial, v_0 la velocidad inicial y a es la aceleración. Nuestro modelo, tiene, por tanto, tres funciones: 1, t y t^2. El resto es mecánico:

```
t = [1:6];
d = [4.89 11.36 21.64 34.10 50.05 69.51];
f1 = @(t) 1 + 0.*t;
f2 = @(t) t;
f3 = @(t) t.^2;
X = [f1(t); f2(t); f3(t)];
A = X*X';
Y = X*d';
coefs = A\Y
```

Ahora `coefs(1)` es la distancia inicial, `coefs(2)` la velocidad inicial y `coefs(3)` la *mitad de la* aceleración. —