Prácticas de laboratorio de Métodos Numéricos

Pedro Fortuny Ayuso

Curso 2020/21, EPIG, Gijón. Universidad de Oviedo
Email address: fortunypedro@uniovi.es

Índice general

CAPÍTULO 1

Introducción a las funciones y los archivos `.m`

En este curso vamos a utilizar con frecuencia la manera habitual de "programar" en Matlab/Octave, que es mediante *archivos* `.m` y funciones definidas en ellos.

Un archivo `.m` no es más que un archivo ordinario cuyo nombre termina en `.m` y que contiene una lista de instrucciones de Matlab/Octave con una estructura precisa. El código del listado 1.1 se puede incluir en un archivo `.m` que simplemente realizará una secuencia de instrucciones:

```
% Un archivo elemental
e = exp(1);
b = linspace(-2,2,1000);
plot(b, e.\^b)
```

LISTADO 1.1. Un archivo `.m` simple

La potencia de los archivos `.m` se debe a dos propiedades:

- Si se graban en un directorio directamente accesible a Matlab/Octave (por ejemplo, `Mis Documentos/Matlab`), se puede ejecutar un comando cuyo nombre sea el mismo que el de un fichero de ese directorio (sin la extensión `.m`) y Matlab/Octave hará las instrucciones pertinentes. Por ejemplo, si se ha grabado el archivo `ejemplo1.m` del código de 1.1, entonces puede ejecutarse la línea

  ```
  > trial1
  ```

  que dibujará la función e^x para $x \in [-2, 2]$ usando 1000 puntos.
- Se pueden usar para definir funciones complicadas.

En lugar de una mera secuencia de comandos, se pueden definir un programa ("una función", suele decirse) dentro de un archivo `.m`. Considérese el listado 1.2, que define una función que devuelve los dos máximos valores de una lista, en orden creciente.

La estructura es la siguiente:

(1) Varias líneas de comentarios (las que comienzan con `%`). Se utilizan para describir el programa definido después.
(2) Una línea como

```
function [y1, y2,...] = nombreFun(p1, p2,...)
```

donde la palabra `function` aparece justo al principio, después una lista de nombres (entre *corchetes*, `[]`), que designan las *variables de salida*, un signo igual =, después el *nombre de la función*, en este caso `nombreFun`, y una lista de nombres, que son los *parámetros de entrada*, entre paréntesis.

(3) Luego viene una sucesión de líneas de comandos de Matlab, que implementan justamente lo que hace la función (esto sería el programa en sí).

(4) La palabra clave `end` al final.

(5) Por último, es esencial que el nombre del archivo sea el mismo que el de la función junto con la extensión `.m`. Por ejemplo, para el listado 1.2, el nombre del archivo *debe ser* `max2.m`.

Un archivo que siga todas estas reglas definirá un comando de Matlab/Octave nuevo que se puede utilizar como cualquier otro.

```
% max2(x)
% devuelve los 2 valores máximos de x, en orden creciente
% si solo hay un valor (length(x) == 1), se devuelve [-inf, x].

function [y] = max2(x)

  % inicializar: el primer elemento es siempre -inf
  y = [-inf, x(1)];

  % si solo hay un número, ya hemos terminado
  if(length(x) == 1)
    return;
  end

  % para cada elemento, solo hay que hacer algo si es mayor que y(1)
  for X=x(2:end)
    if(X > y(1))
      if(X > y(2))
        y(1) = y(2);
        y(2) = X;
      else
        y(1) = X;
      end
    end
  end

end
```

LISTADO 1.2. Un primer fichero `.m` que define una función. Grábese como `max2.m`.

Por ejemplo, si se guarda el código del listado 1.2 en un archivo de nombre `max2.m` dentro del directorio `Mis Documentos/MATLAB`, entonces se puede ejecutar la siguiente cadena de comandos:

```
> x = [-1 2 3 4 -6 9 -8 11]
x =

   -1    2    3    4   -6    9   -8   11

> max2(x)
ans =

    9  11
```

que muestra cómo se ha definido una nueva función llamada `max2` que, cuando se corre, ejecuta las instrucciones del archivo `max2.m`.

Como se van a utilizar archivos `.m` y funciones definidas en ellos con bastante frecuencia, se sugiere al estudiante que haga tantos ejemplos como le sea posible. La programación en Matlab/Octave no es para nada más compleja que la de Python (y en muchos casos, es más sencilla, por el carácter vectorial del lenguaje).

Nota 1 Los principales elementos de programación que se requerirán están incluidos en la hoja de consulta rápida y son los siguientes:

- El enunciado `if...else`. Tiene la sintaxis siguiente:

  ```
  if CONDICIÓN
  ... % sucesión de comandos si se cumple CONDICIÓN
  else
  ... % sucesión de comandos si NO se cumple CONDICIÓN
  end
  ```

  Hay más posibilidades (con `elseif`) pero no vamos a entrar en demasiados detalles.
- El bucle `while`. Sintaxis:

  ```
  while CONDICIÓN
  ... % sucesión de comandos si se cumple CONDICIÓN
  end
  ```

  que ejecutará los comandos internos mientras (while) se cumpla la condición.
- El bucle `for`. Sintaxis:

  ```
  for var=LIST
  ... % sucesión de comandos
  end
  ```

asignará en la variable `var` secuencialmente cada elemento de la lista `LIST` y ejecutará los comandos internos del bucle hasta que se termine la lista.

- Expresiones lógicas. Las `CONDICIÓNes` (de `if` y `while`) pueden ser expresiones simples (como `x<3`) o complejas (construidas utilizando los operadores `and`, `or` y `not`, que se escriben como `&&`, `||` y `~`).

—

Ejercicio 1: Impleméntese una función `min3` que, dada una lista de números como entrada, devuelva los *tres* elementos mínimos. Úsese la función `max2` definida arriba como guía. —

Ejercicio 2: Impleméntese una función `es_creciente` que, dada una lista como entrada, devuelve un `1` si la lista está en orden no decreciente y un `0` si no. ¿Cómo lo harías? —

Ejercicio 3: Mejórese la función del Ejercicio 2 para que devuelva dos valores: primero, el mismo que la función `es_creciente`, y como segundo valor, la longitud de la "secuencia creciente" más larga que haya al principio de los valores de entrada. Llámese a la función `es_creciente2`. Por ejemplo, podría usarse así:

```
> [a,b] = es_creciente2([7, -1, 2, 3 4])
  a = 0
  b = 1
> [u,v] = es_creciente2([-1, 2, 5, 8, 9])
  u = 1
  v = 5
> [a,b] = es_creciente2([3, 4, 5, -6, 8, 10])
  a = 0
  b = 3
```

—

Ejercicio 4: Defínase una función llamada `positiva` que, dada como entrada una lista de números, devuelve dos valores: el número de elementos positivos de la lista (estrictamente mayores que 0) y la lista de dichos elementos como segundo valor de salida. Ejemplos de uso:

```
> [a,b] = positiva([-2, 3, 4, -5, 6, 7, 0])
  a = 4
  b = [3 4 6 7]
> [a,b] = positiva([-1, -2, -3, -exp(1), -pi])
```

```
  a = 0
  b = []
> [a,b] = positiva([4, 2, 1, 3 2])
  a = 5
  b = [4 2 1 3 2]
```

¿Usarías un bucle `while` o un `for`? ¿Por qué? —

Obsérvese que para las funciones que devuelven varios valores de salida, si se quiere que devuelvan más de uno, hay que pedirlo explícitamente asignando una lista de variables, como en el Ejercicio 4. Si solo se quiere un valor (el primero de la lista de salida), se llama con normalidad:

```
> positiva([1, 2, -2, 0, 4])
  3
> x = positiva([-2, 1, 3, 7, 2])
  x = 4
```

pero si se quiere que devuelva los dos, hay que hacer una asignación con corchetes:

```
> [n x] = positiva([pi, -2, 3, -5, 0])
  n = 2
  x = [pi 3]
```

Si la función se llama sin asignar el valor de salida, solo devuelve el primero. Este valor es habitualmente el más importante, pero hay ocasiones en que uno requiere acceder a más parámetros de salida.

Ejercicio 5: Defínase una función `secante` que reciba como entrada una función `f`, un número real `x0` y un número real positivo `epsilon`. Debe devolver dos números `a` y `b`: la pendiente `a` y la altura `b` del corte con el eje OY de la recta secante a la gráfica de `f` que pasa por $f(x_0 - \epsilon)$ y $f(x_0 + \epsilon)$. —

Ejercicio 6: Defínase una función `mav` que transforme una matriz `A` de cualquier tamaño en un vector `v`, de la siguiente manera: si `A` tiene n filas y m columnas, el vector `v` tendrá $n \times m$ componentes: las primeras m componentes serán la primera fila A, las siguientes m la segunda fila, etc...como en el ejemplo.

$$A = \begin{pmatrix} 1 & 0 & -3 \\ 2 & 4 & 7 \\ 3 & 2 & 1 \\ -2 & 0 & 6 \end{pmatrix} \longrightarrow v = (1, 0, -3, 2, 4, 7, 3, 2, 1, -2, 0, 6).$$

—

Ejercicio 7: Defínase una función `vam` que realice la operación inversa de `mav` (ver ejercicio 6): dado un vector `v` y dos dimensiones `n` y `m` (número de filas y número de columnas), se ha de devolver una matriz de tamamño $n \times m$ cuya primera fila sean los primeros m elementos del vector, cuya segunda fila sean los m siguientes, etc. Como en el ejemplo:

$$v = (1,2,0,4,5,7,-2,-3,1,0,2,-6), n = 3, m = 4$$
$$\longrightarrow A = \begin{pmatrix} 1 & 2 & 0 & 4 \\ 5 & 7 & -2 & -3 \\ 1 & 0 & 2 & -6 \end{pmatrix}.$$

Si $n \times m$ es mayor que la longitud de v, se ha de devolver una matriz vacía (como error). Si es menor, los elementos que sobren han de olvidarse. —

Ejercicio 8: Defínase una función `elementosM` que reciba como entrada una matriz `A` de cualquier tamaño y una matriz `P` de dos columnas (y cualquier número de filas). La función debe devolver los elementos de `A` que corresponden a las filas de `P` como si fueran coordenadas, en un vector. Por ejemplo:

$$A = \begin{pmatrix} 1 & 2 & 0 & 4 \\ 5 & 7 & -2 & -3 \\ 1 & 0 & 2 & -6 \end{pmatrix}, P = \begin{pmatrix} 2 & 3 \\ 1 & 4 \\ 1 & 1 \\ 3 & 2 \end{pmatrix} \longrightarrow (-2,4,1,0).$$

Si `P` no tiene dos columnas o alguna de sus filas es una coordenada que no existe en `A`, se debe devolver un vector vacío (como un error). —

Ejercicio 9: Defínase una función `darvalores` que reciba como entrada una matriz `A`, una matriz de dos columnas `P` y un vector `v` con tantas columnas como filas de `P`. La función debe devolver la matriz `A` en la que cada posición correspondiente a las filas de `P` tenga el valor

que corresponde de v, como en el ejemplo.

$$A = \begin{pmatrix} 1 & 2 & 0 & 4 \\ 5 & 7 & -2 & -3 \\ 1 & 0 & 2 & -6 \end{pmatrix}, P = \begin{pmatrix} 2 & 3 \\ 1 & 4 \\ 1 & 1 \\ 3 & 2 \end{pmatrix}, v = (3, 9, 2, 1)$$

$$\longrightarrow A = \begin{pmatrix} 2 & 2 & 0 & 9 \\ 5 & 7 & 3 & -3 \\ 1 & 1 & 2 & -6 \end{pmatrix}$$

En caso de que haya algún lugar inaccesible o de que v tenga un número distinto de elementos que las filas de P, se ha de devolver (a modo de error) una matriz vacía.

CAPÍTULO 2

Soluciones aproximadas de ecuaciones no lineales

Este capítulo estudia los métodos elementales para resolver ecuaciones no lineales de una variable:

$$f(x) = 0 \tag{1}$$

donde f es una función "razonable".

1. Bisección

El teorema de Bolzano dice que una función continua que cambia de signo en un intervalo cerrado tiene automáticamente un cero en dicho intervalo. Este resultado (que es uno de los más importantes del Cálculo) sirve para diseñar un procedimiento que aproxime una raíz dividiendo el intervalo en segmentos cada vez más pequeños. En concreto: dado un intervalo $[a, b]$ y una función $f(x)$ continua sobre él, si $f(a)f(b) < 0$ entonces se sabe que hay una raíz de f en (a, b). Tómese $c = (b + a)/2$ (el punto medio). Si $f(a)f(c) < 0$ entonces hay una raíz en $[a, c]$. Si no, tiene que haberla en $[c, b]$. En cualquiera de los dos casos, ahora se tiene un intervalo de longitud la mitad y puede repetirse el argumento. Esto se formaliza en el Algoritmo 1.

Ejercicio 10: Impleméntese el algoritmo de bisección en un fichero `.m` llamado `biseccion.m`. —

Ejercicio 11: Utilícese el algoritmo de bisección para calcular raíces aproximadas de las funciones que se indican en los intervalos correspondientes. Téngase en cuenta que el algoritmo usado "sin pensar" puede no funcionar para algunos casos, incluso aunque las funciones tengan raíces en dichos intervalos.

$$f(x) = \cos(e^x), x \in [0, 2]$$
$$g(x) = x^3 - 2, x \in [0, 3]$$
$$h(x) = e^x - 2\cos(x), x \in [0, \pi]$$
$$r(t) = t^2 - 1, x \in [-2, 2]$$
$$s(z) = \operatorname{sen}(z) + \cos(z), z \in [0, 2\pi]$$

Algoritmo 1 Algoritmo de bisección.

Input: una función $f(x)$, un par de números reales a, b con $f(a)f(b) < 0$, una tolerancia $\epsilon > 0$ y un límite de iteraciones $N > 0$.

Output: bien un mensaje de error o un número real c entre a y b tal que $|f(c)| < \epsilon$ (es decir, una raíz aproximada).

⋆CONDICIÓN PREVIA
if $f(a) \notin \mathbb{R}$ **ó** $f(b) \notin \mathbb{R}$ **then**
  return ERROR
end if
⋆INICIO
$i \leftarrow 0$
$c \leftarrow \dfrac{a+b}{2}$
while $|f(c)| \geq \epsilon$ **y** $i \leq N$ **do**
  if $f(a)f(c) < 0$ **then**
    $b \leftarrow c$ [`intervalo` $[a, c]$]
  else
    $a \leftarrow c$ [`intervalo` $[c, b]$]
  end if
  $i \leftarrow i + 1$
  $c \leftarrow \dfrac{a+b}{2}$ [`punto medio`]
end while
if $i > N$ **then**
  return ERROR
end if
return c

$$u(t) = t^2 - 3, t \in [-2, 2]$$
$$f(t) = \operatorname{atan}(t-2), t \in [-3, 3]$$

Explíquese cómo puede usarse para las funciones que tienen raíces pero para las cuales el algoritmo "sin pensar" no da resultado. —

2. El método de Newton-Raphson

El algoritmo de Newton-Raphson para encontrar raíces de una función $f(x)$ requiere que esta sea al menos derivable (y, por tanto, continua). Se puede describir formalmente como en el Algoritmo 2.

Algoritmo 2 Algoritmo de Newton-Raphson.

Input: una función diferenciable $f(x)$, una *semilla* $x_0 \in \mathbb{R}$, una tolerancia $\epsilon > 0$ y un límite para el número de iteraciones $N > 0$.

Output: Bien un error bien un número real c tal que $|f(c)| < \epsilon$ (es decir, una raíz aproximada).

⋆INICIO
$i \leftarrow 0$
while $|f(x_i)| \geq \epsilon$ **y** $i \leq N$ **do**
$x_{i+1} \leftarrow x_i - \dfrac{f(x_i)}{f'(x_i)}$ [puede dar un NaN ó ∞]
$i \leftarrow i + 1$
end while
if $i > N$ **then**
return ERROR
end if
$c \leftarrow x_i$
return c

Ejemplo 1 Para implementar el método de Newton-Raphson *sin utilizar operaciones simbólicas* en Matlab, hace falta programar una función cuya entrada incluya la semilla `x0`, la función `f` (como una función anónima), su derivada `fp` (pues no se puede utilizar derivación simbólica) y tanto la tolerancia `epsilon` como el límite para el número de iteraciones `N`. El listado 2.1 muestra una tal implementación.

```
% Newton-Raphson implementation.
% Returns both the approximate root and the number of
% iterations performed.
% Input:
% f, fp (derivative), x0, epsilon, N
function [z n] = newtonraphson(f, fp, x0, epsilon, N)
  n = 0;
  xn = x0;
  % initialize z to a NaN (i.e. error by default)
  z = NaN;
  % Both f and fp are anonymous functions
  fn = f(xn);
  while(abs(fn) >= epsilon && n <= N)
    n = n + 1;
    fn = f(xn); % memorize to prevent recomputing
    % next iteration
    xn = xn - fn/fp(xn); % an exception might take place here
  end
```

```
  z = xn;
  if(n == N)
    warning('Tolerance not reached.');
  end
end
```

LISTADO 2.1. Una implementación (simple) del método de Newton-Raphson.

Obsérvese que, puesto que la función se llama `newtonraphson`, el archivo debe llamarse `newtonraphson.m`.

Ejercicio 12: Utilícese la función `newtonraphson` definida en el Ejemplo 1 para calcular raíces aproximadas de las funciones del Ejercicio 11. Utilícense diferentes valores para la semilla, el épsilon y el límite de iteraciones.

¿Converge siempre el algoritmo?

Ejercicio 13: Modifíquese el código de `newtonraphson` (escribiendo una nueva función llamada `newtonraphson_plot`, con su correspondiente fichero) de manera que, en cada paso, se marque el punto $(x_i, 0)$ en una gráfica, con símbolo suficientemente grande. Úsese con las funciones del Ejercicio 11.

Ejercicio 14: **Hágase este ejercicio sólo si se tiene tiempo suficiente.** Como en el Ejercicio 13, créese una función `newtonraphson_graph` que dibuje, no solo los puntos $(x_i, 0)$ sino también la recta tangente a $f(x)$ desde $(x_{i-1}, f(x_{i-1}))$ hasta $(x_i, 0)$, para visualizar el proceso. Utilícese el comando `pause` entre cada dibujo para esperar a que el usuario presione una tecla (así, el proceso puede verse paso a paso). Úsese esta función como en el Ejercicio 11.

3. El método de la secante

El método de la secante consiste en modificar del de Newton-Raphson que se utiliza cuando la derivada de f es, bien desconocida, bien computablemente pesada de calcular. En lugar de utilizarse $f'(x_i)$ en cada paso, este algoritmo aproxima dicho valor como

$$f'(x_i) \simeq \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}},$$

lo cual requiere llevar cuenta no solo de x_i sino también de x_{i-1} en cada paso.

Ejercicio 15: Escríbase una función `secant` en un archivo llamado `secant.m` que implemente el método de la secante. Téngase en cuenta que, a diferencia del método de Newton-Raphson, esta requiere como entrada solo una función anónima `f` pero *dos* semillas, `x0` y `x1`.

Utilícese esta función `secant` para calcular raíces aproximadas de las funciones del Ejercicio 11. —

4. El método del punto fijo

El método del punto fijo se basa en el siguiente resultado:

THEOREM 1. *Sea $g : [a,b] \to [a,b]$ una función de $[a,b]$ en sí mismo, continua y derivable en $[a,b]$. Si hay un número real positivo $\lambda < 1$ tal que para cada $x \in [a,b]$, $|g'(x)| \leq \lambda$, entonces existe un y solo un $\alpha \in [a,b]$ para el que $g(\alpha) = \alpha$.*

El resultado requiere verificar una buena colección de condiciones antes de que pueda ser utilizado para resolver una ecuación (de hecho, lo primero que hace falta habitualmente es transformar esa ecuación en una de punto fijo). Esencialmente, si uno comienza con la ecuación

$$f(x) = 0,$$

antes de utilizar el método del punto fijo, uno tiene que transformarla en un "problema de punto fijo", es decir, en una ecuación de la forma

$$\tilde{f}(x) = x$$

lo que significa que uno busca un valor c que queda fijo por $\tilde{f}$ (es decir, $\tilde{f}(c) = c$, de ahí el nombre). La transformación habitual es sencilla:

$$f(x) = 0 \Leftrightarrow f(x) + x = x.$$

Obviamente, $f(c) = 0$ si y solo si $f(c) + c = c$. Así, por lo general $\tilde{f}(x) = f(x) + c$.

En lugar de intentar verificar todas las condiciones del teorema 1, uno habitualmente trata de encontrar una solución de $f(x) + x = x$ directamente y, *a posteriori*, verificar que se ha encontrado

Ejercicio 16: Escríbase una función `fixed_point` que implemente el método del punto fijo. La entrada debería ser: una función anónima `f`, una semilla `x0`, una tolerancia `epsilon` y un límite para el número

de iteraciones `N`. La salida debería ser bien un mensaje de error o bien un número `c` tal que `abs(f(c))<epsilon`.

Nota: La iteración para este problema no es $x_{i+1} = f(x_i)$.

Utilícese esta función `fixed_point` para calcular (si se puede) raíces aproximadas de las funciones del Ejercicio 11. ¿Funciona siempre? ¿Cuándo lo hace? ¿Por qué? —

5. Algunos ejercicios más prácticos

Se presentan aquí algunos ejercicios de contenido más "práctico" para utilizar los métodos programados anteriormente. Se trata de que el alumno *plantee el problema* y lo transforme en uno en el que pueda utilizarse alguno (o varios) de los algoritmos previamente programados.

Ejercicio 17: Encontrar *todos* los puntos de corte de las funciones

$$f(x) = x^2 - 2,\ g(x) = e^{-x^2}.$$

—

Solución. Antes de hacer cálculos, es siempre conveniente hacerse una idea de las gráficas de las funciones. La función $f(x)$ tiene por gráfica una parábola con ramas hacia arriba, mientras que la función $g(x)$ tiene por límite 0 cuando $x \to \infty$. Por tanto, los puntos de corte estarán bastante acotados. De hecho, $f(x) > 1$ para $|x| > \sqrt{3}$ y $g(x) < 1$ siempre. Dibujemos ambas gráficas entre $-\sqrt{3}$ y $\sqrt{3}$:

```
u = [-sqrt(3):.01:sqrt(3)];
f = @(x) x.^2-2;
g = @(x) exp(-x.^2);
plot(u,f(u));
hold on;
plot(u,g(u));
```

Da la impresión de que solo hay dos puntos de corte, que son simétricos (ambas funciones son obviamente pares). Calcularemos solo el positivo. Para utilizar cualquiera de los algoritmos, necesitamos *una única función*. Tomemos, por ejemplo, $h(x) = f(x) - g(x)$ y dibujemos su gráfica entre 1 y $\sqrt{3}$:

```
clf;
h = @(x) f(x) - g(x);
v = [1:.001:sqrt(3)];
```

```
plot(v, h(v));
```

Se observa que cambia de signo en tal intervalo. Utilicemos el método de bisección primero, con una tolerancia de 10^{-6} y un número exagerado de iteraciones máximas:

```
a = 1;
b = sqrt(3);
e = 1e-6;
N = 1000;
r1 = biseccion(h, a, b, e, N)
```

El valor obtenido es $r1 = 1.4560$. Para dicho valor, $h(r1) = -7.1445 \times 10^{-7}$, que es aceptable (menor que la tolerancia). Veamos qué ocurre con el método de Newton-Raphson. Viéndo la gráfica de $h(x)$, parece razonable tomar $x_0 = 1$. La derivada de $h(x)$ es $h'(x) = 2x(1 + e^{-x^2})$. Utilizamos la misma tolerancia y el mismo número de pasos:

```
hp = @(x) 2*x.*(1+exp(-x.^2));
x0 = 1;
r2 = newtonraphson(h, hp, x0, e, N)
```

que da el mismo resultado.

En $r1$, el valor de $f(x)$ (que es el mismo que el de $g(x)$) es $f(r1) = 0.12003$, así que los puntos de corte pedidos son:

$$P_1 = (1.456, 0.12003), \quad P_2 = (-1.456, 0.12003).$$

Ejercicio 18: Modificar los programas ya hechos para Biseccion y Newton-Raphson de manera que devuelvan no solo la raíz aproximada, sino también el número de iteraciones realizadas. Llámese a estas funciones `biseccion2` y `newtonraphson2` (para no confundirlas con las otras) y compárese, en el ejercicio anterior, la velocidad de convergencia de ambas. —

Ejercicio 19: Una masa está sujeta al extremo de un muelle (horizontal) cuyo otro extremo está fijo. El movimiento de esta masa, respecto del punto de reposo del muelle, $x = 0$, sigue la ecuación

$$x(t) = e^{-0.1t}(2\,\mathrm{sen}(3t) + 3\cos(3t)).$$

Calcúlese con qué velocidad pasa la masa por el punto de reposo por tercera vez. —

SOLUCIÓN. Antes de nada, se define la función y se dibuja en un intervalo razonable (si no fuera suficiente, se cambiaría). Téngase en cuenta que, puesto que t indica el tiempo, se supone que comienza en $t = 0$.

```
m = @(t) exp(-0.1*t).*(2*sin(3*t)+3*cos(3*t));
T = [0:.001:10]; % si hace falta, se hace mas grande
plot(T, m(T));
```

En la gráfica se observa que la tercera vez que pasa por el origen está entre (aprox.) 2 y 4. Pero si calculamos `m(2)` y `m(4)`, ambos son positivos. Hagamos el intervalo más pequeño:

```
T =[2:.001:4];
plot(T, m(T));
```

Aquí se ve más claramente que entre 2.5 y 3 cambia de signo (ya sabemos que esta es la tercera vez). Ahora podemos aplicar Bisección (o Newton-Raphson):

```
a = 2.5;
b = 3;
e = 1e-6;
N = 100;
r1 = biseccion(m, a, b, e, N)
```

y $m(r1) = 8.1447 \times 10^{-7}$, que es aceptable. Para Newton-Raphson, hemos de tener cuidado de elegir una semilla que *no esté cerca de los extremos relativos*; por ejemplo, probemos con $x_0 = 2.9$. La derivada de $m(x)$ es (hay que saber hacerla):

$$m'(t) = -0.1e^{-0.1t}(2\,\mathrm{sen}(3t) + 3\cos(3t)) + e^{-0.1t}(6\cos(3t) - 9\,\mathrm{sen}(3t)).$$

Y el uso de Newton-Raphson puede hacerse como sigue:

```
mp = @(t) -0.1*exp(-0.1*t).*(2*sin(3*t) + 3*cos(3*t)) +
    exp(-0.1*t).*(6*cos(3*t) - 9*sin(3*t));
x0 = 2.9;
r2 = newtonraphson(m, mp, x0, e, N)
```

Como ya hemos calculado la derivada de $m(x)$, su valor en $r1$ (ó en $r2$) es la velocidad pedida:

$$m'(r1) = -8.1636$$

(en las unidades del problema).

Ejercicio 20: Calcular el radio de una esfera de acero (cuya densidad es $7850\text{kg}/\text{m}^3$) si tiene que tener la misma masa que un cilindro de mercurio (densidad $16.6\text{g}/\text{cm}^3$) de radio 1m y altura 2cm. —

Ejercicio 21: La altura de un cuerpo en caída libre con (un modelo de) rozamiento es

$$x(t) = \frac{-me^{-kt/m}(gm + kv_0) + gm(m - kt) + hk^2 + kmv_0}{k^2}$$

suponiendo que $x = 0$ es el suelo y que $x > 0$ es "hacia arriba". En esa ecuación, g es la constante gravitatoria, v_0 es la velocidad inicial, h es la altura inicial, m la masa y k la constante de rozamiento. Se lanza un cuerpo de masa 1kg hacia arriba desde 100m con una velocidad de $1\text{m}/\text{s}$ y una constante de rozamiento de 0.02. Un segundo más tarde, se lanza un cuerpo de masa 2kg hacia arriba a la misma velocidad inicial, desde 95m y con constante de rozamiento 0.015. Si $g = 10$, ¿hay algún momento en que se cruzan *mientras ambos caen*? Si esto es así, ¿a qué altura ocurre y en qué instante de tiempo? —

Ejercicio 22: Un objeto sigue un movimiento circular uniforme alrededor del punto $(2, 3)$ (en km) en el plano (x, y), con radio del movimiento $R = 20\text{km}$ y velocidad *lineal* de $20\text{m}/\text{s}$, comenzando en el punto $(22, 3)$. Se quiere lanzar un proyectil con movimiento rectilíneo uniforme de desde el origen de coordenadas, de tal manera que choque perpendicularmente con el primer objeto cuando este esté en el semiplano $x < 2$, *la primera vez que sea posible*. Ambos objetos se suponen puntuales. Calcular en qué punto deben chocar y a qué velocidad debe ir el proyectil. —

CAPÍTULO 3

Álgebra Lineal en Matlab

1. Introducción y ejercicios elementales

En las clases de Teoría se han explicado varios algoritmos de resolución aproximada de sistemas de ecuaciones lineales. Los sistemas lineales que aparecen en el mundo "real" tienen, por lo general, dimensiones enormes (del orden de los miles o millones de incógnitas) y, casi siempre son sistemas generados de manera automática. Solo en casos muy sencillos se pueden escribir explícitamente todas las ecuaciones.

En estas prácticas primero trataremos de familiarizar al alumno con la utilización de matrices y vectores para resolver problemas lineales, sin preocuparnos demasiado (al principio) de la utilización o implantación de algoritmos específicos.

Nota 2 El método que se utiliza en Matlab para resolver sistemas de ecuaciones es mediante el operador \, que se utiliza como sigue: si se quiere resolver el sistema

$$Ax = b$$

donde A es una matriz, b es el vector de términos independientes y x es el vector de incógnitas, se ha de escribir:

```
x = A\b
```

y x contendrá la solución. Por ejemplo,

$$\begin{pmatrix} 2 & 3 & 4 \\ 1 & 0 & -1 \\ 2 & 8 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix}$$

se resuelve así:

```
A = [2 3 4; 1 0 -1; 2 8 3];
b = [1; 2; 1];
x = A\b
x =
   1.06061
   0.21212
```

```
-0.93939
```

cuya exactitud puede verificarse escribiendo

```
A*x - b
```

y viendo que da el vector nulo (pero esta no es la mejor manera de hacerlo, como veremos). —

Nota 3 La forma de calcular la factorización LUP de una matriz usando Matlab es mediante el comando `lu`. Si utilizamos la matriz de antes,

```
[a b c] = lu(A)
```

se obtienen matrices: `a`, `b` y `c` tales que `c*A=a*b`, que son, respectivamente: la matriz `a` triangular inferior con unos en la diagonal (la L de la factorización), la matriz `b` triangular superior (la U), y la matriz `p` de permutación.

Importante: No hay una manera precisa de calcular sin más la factorización LU (nunca se usa porque no es óptima). —

Ejemplo 2 Resolver los siguientes sistemas de ecuaciones, comprobando que la solución obtenida es la correcta. Calcular las factorización LUP de la matriz correspondiente.

(1) El sistema:

$$\begin{aligned} y + z + x &= 33 \\ z - x + y &= 12 \\ z + 2x &= 1 \end{aligned}$$

Solución. Los siguientes comandos son suficientes (hay que tener cuidado con el orden de las variables, claro):

```
A=[1 1 1;-1 1 1;2 0 1];
b=[33;12;1];
v=A\b
% lo que sigue es la salida de matlab
  v =
   10.5000
   42.5000
  -20.0000
[l1 u1 p1] = lu(A)
```

Para comprobar que la solución es correcta, realizamos la operación de multiplicar A por v y comparamos con b:

```
A * v
% lo que sigue es la salida de matlab
```

```
ans =
 33
 12
  1
```

pero esta manera no es la mejor para verificarlo, como veremos.

(2) El sistema de ecuaciones

$$\begin{aligned} 2y + 3z + t + x &= -1 \\ 2y - 3z + 4t &= 2 \\ -x + 5y &= -3 \\ x + y - z + 2t &= 4 \end{aligned}$$

Igual que antes, se define la matriz de coeficientes y el vector de términos independientes y se resuelve con \:

```
A=[1 2 3 1; 0 2 -3 4; -1 5 0 0; 1 1 -1 2];
b=[-1;2;-3;4];
v=A\b
% lo que sigue es la salida de matlab
  v =
    3.7258
    0.1452
   -1.4516
   -0.6613
[l2 u2 p2] = lu(A)
```

Ahora, si multplicamos A por v, también obtenemos un resultado "normal":

```
A*v
% lo que sigue es la salida de matlab
  ans =
   -1.0000
    2.0000
   -3.0000
    4.0000
```

¿Ocurre que `u2 * v = b`? ¿Tiene que ocurrir? Ojo a esto...

(3) Resolver el siguiente sistema de ecuaciones en forma matricial:

$$\begin{pmatrix} 2 & 1 & -1 & 3 \\ 4 & 2 & 0.4 & 1 \\ -1 & 3 & 0 & 8 \\ 0 & 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}.$$

SOLUCIÓN. No hay más que hacerlo:

```
A=[2 1 -1 3; 4 2 0.4 1; -1 3 0 8; 0 1 2 3];
b=[1 2 3 4]';  % el vector traspuesto
v=A\b
% lo que sigue es la salida de matlab
  v =
    0.4737
   -0.5263
    1.3158
    0.6316
A*v
% lo que sigue es la salida de matlab
  ans =
    1.0000
    2.0000
    3.0000
    4.0000
[l3 u3 p3] = lu(A)
```

¿Ocurre que `u3 * v = b`? ¿Tiene que ocurrir?

(4) El sistema

$$\begin{pmatrix} 1 & 1 & 1 \\ -1 & 1 & 1 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 33 \\ 12 \\ 1 \end{pmatrix}.$$

SOLUCIÓN. Los comandos son siempre los mismos:

```
A = [1 1 1 ; -1 1 1 ; 2 0 1];
b = [33; 12; 1]; % tambien vale [22 12 1]'
s = A\b
[l4 u4 p4] = lu(A)
```

Comprobemos la solución restando, en lugar de "de memoria":

```
if (A*s - b == 0)
disp('ok')
```

```
else
disp('no ok')
end
```

Lo que se obtiene es `ok`. **PERO** esto está mal hecho. ¿Por qué?

(5) Finalmente, el sistema

$$\begin{aligned} x + 2y + 3z + 4t &= -1 \\ 2y - 3z + 4t &= 2 \\ -x + 5y &= -3 \\ x + y - z + 2t &= 4 \end{aligned}$$

Como en todos estos ejercicios,

```
A = [1 2 3 4 ; 0 2 -3 4 ; -1 5 0 0 ; 1 1 -1 2];
b = [-1; 2; -3; 4]; % tambien vale [-1 2 -3 4]'
s = A\b
% lo que sigue es la salida de matlab
  s =
    3.5455
    0.1091
   -1.0909
   -0.3727
```

Veamos qué pasa si comprobamos que la solución es correcta utilizando un `if`:

```
if (A*s - b == 0)
disp('OK')
else
disp('NO OK')
```

se debería obtener un `OK` pero resulta un `NO OK`. ¿Por qué? Veamos cuánto es `A*s - b`

```
A * s - b
% lo que sigue es la salida de matlab
  ans =
   1.0e-15 *
         0
   -0.4441
         0
   -0.4441
```

Que es un vector muy pequeño: está multiplicado todo por 10^{-15}. **NUNCA** se comparan números con igualdades, en Matlab, siempre hay que comparar con un error (una tolerancia). Si nuestra tolerancia es 10^{-6}, entonces:

```
if (max(abs(A*s-b)) < 10e-6)
disp('ok')
else
disp('no ok')
end
```

se obtiene lo que se desea (la solución es correcta hasta cierta tolerancia).

—

Ejemplo 3 Construyamos una matriz muy especial:

$$\begin{pmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{1+n} \\ \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{2+n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n} \end{pmatrix}$$

que se denomina *matriz de Hilbert de orden n*. Hagámoslo para $n = 12$, que ya es suficientemente grande:

```
A=zeros(12)
for k=1:12
    for l=1:12
        A(k,l)=1/(k+l);
    end
end
```

y planteemos un sistema de ecuaciones con ella: tomemos el vector

$$b = (200, 300, 123, 21, 10, 100, 256, 1040, 2300, 11, 1000, 300)$$

que asignamos en Matlab:

```
b = [200 300 123 21 10 100 256 1040 2300 11 1000 300]
```

y resolvamos el problema

$$A\overline{x} = \overline{b}^T$$

(tomamos la traspuesta de b para ponerlo en vertical). Esto no es más que la operación \:

```
v = A\b'
% lo que sigue es la salida de Matlab
v =
   1.0e+19 *
    0.0000
   -0.0001
    0.0024
   -0.0268
    0.1697
   -0.6676
    1.7045
   -2.8731
    3.1740
   -2.2102
    0.8799
   -0.1527
```

(es importante ver toda la salida). Fijaos que es posible que, antes de la salida, Matlab haya mostrado un mensaje de *advertencia* diciendo algo como "la matriz está cerca de ser singular...". Comprobemos si la solución es correcta:

```
A * v - b'
% lo que sigue es la salida de Matlab
ans =
   136
    52
   181
   331
   -74
   220
  -368
   240
   244
  -323
   120
   452
```

¿? ¿qué ha ocurrido? Ni siquiera se parece al vector cero...

El problema es que *la matriz A está muy mal acondicionada*: esto significa que un pequeño cambio en los datos de la matriz ó en los

términos independientes produce un gran cambio en las soluciones. Si se observa el valor de `v` mostrado por Matlab, se ve que todos los elementos de `v` están multiplicados por 10^{19}: en coma flotante de 64 bits, esto tiene una precisión menor que las milésimas (la coma flotante de 64 bits da una precisión de unos 16 dígitos *decimales*). Por tanto, la solución obtenida *no va a ser correcta prácticamente nunca*: se pierde precisión porque A es una matriz muy mala. Esto, sin más datos, es muy muy difícil de arreglar.

Conclusión: *no fiarse nunca de la salida de un programa*.

Ejercicio 23: Utilizar el comando `diag` para generar una matriz diagonal 10×10 con los números del uno al 10 en la diagonal. Lo mismo pero con 10 números entre $-e$ y π equiespaciados (utilizar el comando `linspace` para esto último).

Ejercicio 24: Utilizar el comando `diag` pero con dos parámetros para generar una matriz 11×11 que tenga los números pares del 2 al 20 en la primera diagonal hacia la derecha (a continuación se muestra el ejemplo 3×3):

$$\begin{pmatrix} 0 & 2 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix}.$$

Ejercicio 25: Utilizar los comandos `eye` y `diag` para construir una matriz de tamaño 30×30 que tenga:

- La diagonal principal llena de 3.14
- En la primera diagonal hacia la derecha, los números pares del 2 hasta el que corresponda (¿cuál es el que corresponde?)
- En la primera diagonal hacia abajo, los números impares del 3 hasta el que corresponda (¿cuál es?).

Se muestra a continuación el caso 5×5:

$$\begin{pmatrix} 3.14 & 2 & 0 & 0 & 0 \\ 3 & 3.14 & 4 & 0 & 0 \\ 0 & 5 & 3.14 & 6 & 0 \\ 0 & 0 & 7 & 3.14 & 8 \\ 0 & 0 & 0 & 9 & 3.14 \end{pmatrix}.$$

Ejercicio 26: Escribir una función `trasladaM` que realice la siguiente operación: dada una matriz `A` y dos números n, m, devuelve una matriz `B` que consiste en la traslación *circular* de los elementos de `A`: `n` unidades hacia la derecha y `m` unidades hacia abajo (izquierda, arriba, si son negativos). Por ejemplo, para la matriz

$$A = \begin{pmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \\ -1 & -2 & -3 & -4 \end{pmatrix}$$

la operación `trasladaM(A, 1, 2)` debe devolver la matriz

$$\begin{pmatrix} 8 & 2 & 4 & 6 \\ -4 & -1 & -2 & -3 \\ 7 & 1 & 3 & 5 \end{pmatrix}.$$

Realícense varias pruebas con `foto` para comprobar que funciona correctamente, tanto con valores positivos como negativos de `n` y de `m`.

Nota: Puede ser más sencillo hacer primero un programa llamado `trasladaMh` que traslade solo en horizontal y otro `trasladaMv` que traslade solo en vertical. El programa `trasladaM` será simplemente el primero seguido del segundo.

Ejercicio 27: Descárguese el archivo
`https://pfortuny.net/uniovi/metodos_numericos/foto.dat`
y guárdese en el directorio `\Documentos\MATLAB`. Cárguese su contenido con el comando `load('foto.dat')`. Al cargarlo, se crea una variable llamada `foto` que es una matriz (como se puede comprobar con el comando `size`, de tamaño 300×286). Lo interesante es que Matlab puede tratar las matrices como imágenes: escríbase

```
colormap(gray(256));
image(foto);
```

para ver una representación (en escala de 256 grises) de la matriz `foto`. Esto puede hacerse con cualquier matriz.

Ejercicio 28: Utilícese la manera propia de Matlab para extraer submatrices de una matriz para obtener partes de `foto`. Ejemplos:

- Las diez primeras filas enteras:

  ```
  foto(1:10,:)
  ```

- Las diez primeras filas y las quince primeras columnas:

```
foto(1:10, 1:15)
```

- Las filas pares y las columnas impares:

```
foto(2:2:end, 1:2:end)
```

- Las filas de la 20 a la 280 y las columnas de la 10 a la 270:

```
foto(20:280, 10:270)
```

2. Cadenas de Markov

Como aplicación de Matlab en problemas de álgebra lineal, se presentan brevemente los procesos que pueden modelarse como una cadena de eventos que solo dependen del estado anterior y cuya evolución viene determinada por un conjunto determinado de transiciones: lo que se denomina, técnicamente, como procesos de Markov.

Primero, un ejemplo:

Ejemplo 4 Un agricultor posee una cierta cantidad de hectáreas de tierra, que dedica a cebada, trigo o deja en barbecho, según el año. Se sabe lo siguiente:

- Si una hectárea ha estado un año en barbecho, al año siguiente no lo estará. Además, la probabilidad de que la dedique a cebada es de $2/5$ y de que la dedique a trigo es de $3/5$.
- Si una hectárea ha estado dedicada a cebada, la probabilidad de que la deje en barbecho al año siguiente es de $1/3$, la de que la dedique a cebada es $2/3$ y seguro que no la dedica a trigo.
- Finalmente, si ha estado dedicada a trigo, seguro que al año siguiente no se dedica a cebada y hay una probabilidad de $1/3$ de que se deje en barbecho (y, por tanto, de $2/3$ de que se dedique a trigo otra vez).

Se quieren estudiar los siguientes problemas:

(1) Si una hectárea está en una de las tres posibles dedicaciones, ¿cómo se espera que esté dentro de dos años? ¿dentro de 5? ¿dentro de 20?

(2) Si el agricultor dedica, en un año determinado, 36ha a trigo, 42ha a cebada y tiene 60 en barbecho, ¿cuál será la distribución esperada dentro de un año? ¿dentro de dos? ¿dentro de 5? ¿dentro de 10?

Este problema, que parece un largo ejercicio de estadística, puede enunciarse y resolverse muy simplemente con álgebra matricial.

Dispongamos en una tabla, *por columnas*, los valores de probabilidades dados arriba. La tabla ha de leerse, como se ha dicho, *por*

	B	T	C
B	0	1/3	1/3
T	3/5	2/3	0
C	2/5	0	2/3

TABLA 1. Probabilidad de pasar de un cultivo a otro.

columnas: por ejemplo, la columna primera representa que el barbecho no seguirá en barbecho y tendrá tres quintos de posibilidades de dedicarse a trigo y dos quintos a cebada, etc.

El esquema de distribución de un año al siguiente puede explicarse así, yendo tipo a tipo:

Barbecho: Si el agricultor posee b hectáreas en barbecho en un año determinado, al año siguiente tendrá $3b/5$ha de trigo y $2b/5$ha de cebada, y ninguna en barbecho (de esas a).

Trigo: Si posee t hectáreas de trigo en un año determinado, al siguiente, esas t se distribuirán así: $t/3$ha para barbecho y $2/3$ha para trigo.

Cebada: Si posee c hectáreas de cebada en un año, al siguiente, esas c se distribuirán así: $c/3$ha serán de barbecho y $2c/3$ seguirán con cebada.

Por tanto, si se considera el vector (b, t, c) de un año como la distribución en barbecho, trigo y cebada, al año siguiente se tendrán distribuidas las hectáreas según el vector $(t/3 + c/3, 3b/5 + 2t/3, 2b/5 + 2c/3)$, donde la primera componente es el barbecho, la segunda el trigo y la tercera la cebada.

Una mera inspección muestra que

$$\begin{pmatrix} t/3 + c/3 \\ 3b/5 + 2t/3 \\ 2b/5 + 2c/3 \end{pmatrix} = \begin{pmatrix} 0 & 1/3 & 1/3 \\ 3/5 & 2/3 & 0 \\ 2/5 & 0 & 2/3 \end{pmatrix} \begin{pmatrix} b \\ t \\ c \end{pmatrix}$$

así que la distribución de las hectáreas al año siguiente puede calcularse con una simple multiplicación matricial. Es decir, si denominamos A a la matriz de probabilidades, v_0 al vector de distribución de hectáreas en el año 0 y v_1 al del siguiente, entonces

$$v_1 = Av_0.$$

Si ahora quiere calcularse la distribución v_2 correspondiente al segundo año, es

$$v_2 = Av_1 = A \cdot (Av_0) = A^2 v_0$$

y, de manera similar, la distribución al cabo de n años será

$$v_n = A^n v_0$$

entendiendo siempre que v_0 es el estado inicial, la distribución de hectáreas con que se comenzó. Ahora, responder a las preguntas iniciales es bien sencillo. Para el caso de una hectárea de barbecho:

$$\begin{pmatrix} 0 & 1/3 & 1/3 \\ 3/5 & 2/3 & 0 \\ 2/5 & 0 & 2/3 \end{pmatrix}^2 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1/3 \\ 2/5 \\ 4/15 \end{pmatrix},$$

$$\begin{pmatrix} 0 & 1/3 & 1/3 \\ 3/5 & 2/3 & 0 \\ 2/5 & 0 & 2/3 \end{pmatrix}^5 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.247 \\ 0.452 \\ 0.301 \end{pmatrix},$$

$$\begin{pmatrix} 0 & 1/3 & 1/3 \\ 3/5 & 2/3 & 0 \\ 2/5 & 0 & 2/3 \end{pmatrix}^{20} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 0.45 \\ 0.3 \end{pmatrix}$$

que signfica que, tras cinco años, la probabilidad de que esa hectárea esté dedicada a barbecho es aproximadamente un cuarto; de que esté dedicada a trigo es un poco menos de la mitad y de que esté dedicada a cebada, cerca de un tercio. Como se ve, al cabo de veinte años es prácticamente la misma distribución de probabilidades; este fenómeno es habitual en procesos de este tipo con ciertas características que no mencionaremos ahora.

Por responder a alguna de las preguntas del segundo apartado, visto que la distribución inicial es $(36, 42, 60)$, al cabo de cinco años será:

$$\begin{pmatrix} 0 & 1/3 & 1/3 \\ 3/5 & 2/3 & 0 \\ 2/5 & 0 & 2/3 \end{pmatrix}^5 \begin{pmatrix} 36 \\ 42 \\ 60 \end{pmatrix} =$$

$$= \begin{pmatrix} 0.24691 & 0.25103 & 0.25103 \\ 0.45185 & 0.50206 & 0.37037 \\ 0.30123 & 0.24691 & 0.37860 \end{pmatrix} \begin{pmatrix} 36 \\ 42 \\ 60 \end{pmatrix} = \begin{pmatrix} 34.494 \\ 59.575 \\ 43.931 \end{pmatrix}$$

que da una idea aproximada (se supone que el agricultor no subdivide cada hectárea) del reparto final.

Si, por ejemplo, desea saberse la distribución que había el año anterior si este año es $(12, 42, 18)$, habrá de resolverse el sistema:

$$\begin{pmatrix} 0 & 1/3 & 1/3 \\ 3/5 & 2/3 & 0 \\ 2/5 & 0 & 2/3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 12 \\ 42 \\ 18 \end{pmatrix}$$

cuya solución es $(x, y, z) = (36, 30.6, 5.4)$. Esto puede hacerse, como se dijo arriba, utilizando el operador \. Lo mismo, pero elevando A a una potencia adeucada, si quiere saberse la distribución n años antes.

—

Un caso más sencillo, en el que solo hay dos estados posibles, es el siguiente.

Ejemplo 5 El comportamiento de un alumno de la Universidad de Oviedo, en la asignatura de Historia del Álgebra, es el siguiente: cada día (con clase de esa asignatura) asiste a esa clase o no, dependiendo de sí ha ido o no el día anterior. En concreto:

- Si un día ha asistido, la probabilidad de que asista al día siguiente es 0.75, mientras que la probabilidad de que no lo haga es 0.25.
- Si un día no ha asistido, la probabilidad de que asista al día siguiente es 0.1, y la de que no asista es 0.9.

(Como se ve, la Historia del Álgebra no genera mucha adicción). En este ejemplo, hay dos estados posibles: *asistir* y *no asistir*. El estado siguiente depende exclusivamente del anterior, según la ley de probabilidades (por columnas) que se muestra en la tabla:

	Sí	No
Sí	0.75	0.1
No	0.25	0.9

TABLA 2. Asistencia a clases de Historia del Álgebra.

En lugar de escribir la tabla, llamamos directamente A a la matriz de "paso" de un estado al siguiente:

$$A = \begin{pmatrix} 0.75 & 0.1 \\ 0.25 & 0.9 \end{pmatrix}$$

Supongamos ahora que hay 200 matriculados en Historia del Álgebra y el primer día asisten 180 (porque los repetidores ya se conocen el percal). ¿Cuál es la distribución probable al día siguiente? ¿y al cabo de 10 días? ¿y al final del curso, si tiene 40 horas lectivas?

Igual que antes, si (s, n) es el vector de "sí asisten", "no asisten", la distribución al cabo de un día será

$$A \begin{pmatrix} s \\ n \end{pmatrix} = \begin{pmatrix} 0.75 & 0.1 \\ 0.25 & 0.9 \end{pmatrix} \begin{pmatrix} s \\ n \end{pmatrix} = \begin{pmatrix} 0.75s + 0.1n \\ 0.25s + 0.9n \end{pmatrix}$$

y, del mismo modo que en el Ejemplo 4, si se comienza con un vector v_0, la distribución esperada al cabo de n días es (llamamos v_0 al "primer" día y lo tratamos de manera especial para no complicarnos, como si no contara):

$$v_n = A^n v_0.$$

Por tanto, las distribuciones esperadas al cabo de 1 día, 10 y 40 son:

$$v_1 = \begin{pmatrix} 0.75 & 0.1 \\ 0.25 & 0.9 \end{pmatrix} \begin{pmatrix} 180 \\ 20 \end{pmatrix} = \begin{pmatrix} 137 \\ 63 \end{pmatrix}, \quad v_{10} = \begin{pmatrix} 0.75 & 0.1 \\ 0.25 & 0.9 \end{pmatrix}^{10} \begin{pmatrix} 180 \\ 20 \end{pmatrix} = \begin{pmatrix} 58.8 \\ 141.2 \end{pmatrix}$$

$$v_{40} = \begin{pmatrix} 0.75 & 0.1 \\ 0.25 & 0.9 \end{pmatrix}^{40} \begin{pmatrix} 180 \\ 20 \end{pmatrix} = \begin{pmatrix} 57.14 \\ 142.86 \end{pmatrix}$$

Se aprecia que v_{40} es muy parecido a v_{10}. Esto, que ya ocurría en el Ejemplo de las hectáreas de tierra dedicadas a cultivos, se enuncia diciendo que el sistema tiende a un *estado estacionario*: llega un momento en que el estado siguiente es muy parecido al presente, para cualquier estado inicial.

¿Cómo se calularía el número de alumnos inicial si se sabe que al cabo de dos semanas de clase (i.e. 10 clases) la distribución es $(32, 44)$? —

Ejemplo 6 El tiempo en Gijón sigue un patrón más o menos parecido al que se enuncia:

- Si un día llueve, la probabilidad de que llueva al día siguiente es de 0.6, la de que haga sol es 0.3 y la de que esté nublado es 0.1.
- Si un día está nublado, la probabilidad de que al día siguiente esté nublado es 0.6, la de que haga sol es 0.15 y la de que llueva es 0.25.
- Si un día hace sol, la probabilidad de que al día siguiente haga sol es 0, la de que llueva es 0.4 y la de que esté nublado es 0.6.

Un día determinado hizo sol. Calcular la distribución de probabilidades del tiempo al cabo de dos, cinco y siete días.

¿Hay alguna distribución estable a largo plazo del tiempo?

La matriz correspondiente a este problema es

$$A = \begin{pmatrix} 0.6 & 0.25 & 0.4 \\ 0.1 & 0.6 & 0.6 \\ 0.3 & 0.15 & 0 \end{pmatrix}$$

(nótese que hay que ordenar correctamente las columnas). Si comienza haciendo sol, quiere decir que se comienza con el vector

$$v_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

y para calcular los (posibles) estados correspondientes a dos, cinco y siete días se procede así:

$$v_2 = A^2 v_0 \simeq \begin{pmatrix} 0.39 \\ 0.40 \\ 0.21 \end{pmatrix}, \; v_5 = A^5 v_0 \simeq \begin{pmatrix} 0.43 \\ 0.39 \\ 0.18 \end{pmatrix}, \; v_7 = A^7 v_0 \simeq \begin{pmatrix} 0.43 \\ 0.39 \\ 0.18 \end{pmatrix}.$$

Parece que se estabiliza la distribución posible de estados. De hecho, si calculamos A^{20}, queda

$$A^{20} \simeq \begin{pmatrix} 0.428 & 0.428 & 0.428 \\ 0.386 & 0.386 & 0.386 \\ 0.186 & 0.186 & 0.186 \end{pmatrix}$$

una matriz con todas las columnas iguales. Multiplicar esta matriz por cualquier vector $(x, y, z)^T$ da

$$\begin{pmatrix} 0.428 & 0.428 & 0.428 \\ 0.386 & 0.386 & 0.386 \\ 0.186 & 0.186 & 0.186 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = (x + y + z) \begin{pmatrix} 0.428 \\ 0.386 \\ 0.186 \end{pmatrix},$$

lo cual significa: se empiece con la distribución de días que se empiece, a la larga, la distribución será de 43% de días de lluvia, 38% de días nublados y 19% de días de sol (aproximadamente). Esta es, como bien se sabe, la distribución más o menos general del tiempo en Gijón (quizás incluso demasiado optimista). —

La explicación sobre las matrices con todas las columnas iguales es la siguiente. Supongamos que una matriz de transición A puede escribirse

$$A = \begin{pmatrix} a_1 & a_1 & \dots & a_1 \\ a_2 & a_2 & \dots & a_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_n & a_n & \dots & a_n \end{pmatrix}$$

donde todas las columnas suman 1, es decir: $a_1 + \cdots + a_n = 1$ —esto es necesario para que sea una matriz de transición, pues las columnas son distribuciones de probabilidad. Con esta hipótesis, si el estado

inicial (o la distribución inicial de estados) viene dado por un vector

$$v_0 = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix},$$

resulta que

$$Av_0 = (x_1 + x_2 + \cdots + x_n) \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

que quiere decir que la distribución de estados siguientes es la descrita por cualquiera de las columnas de A, ponderada por el número de elementos considerados. Y, por tanto, lo mismo ocurre tras cada iteración. Nótese que la suma total de elementos en el vector Av_0 es la misma que la de los de v_0, que es $x_0 + \cdots + x_n$. Esto ocurre, como es evidente, en cualquier iteración de un proceso de este tipo.

Ejemplo 7 La playa de San Lorenzo, de Gijón se subdivide, para mejor gestión de los socorristas, en diez áreas. Suponemos, para simplificar, que la marea siempre está baja (de otro modo, como es bien sabido, nueve de esas diez áreas desaparecen bajo el agua...). Se divide el tiempo en intervalos de 1 minuto. En las ocho áreas "internas" (las que no están en los extremos), la gente permanece dentro con una probabilidad de 0.8 y sale hacia una de dos adyacentes con una probabilidad de 0.2. En las zonas de los extremos, la probabilidad de permanecer es 0.85 y la de salir es 0.15. La matriz de transición es la siguiente:

$$A = \begin{pmatrix} 0.85 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.15 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.15 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.85 \end{pmatrix}$$

que, en Matlab, puede generarse de manera sencilla así:

```
A = 0.8*eye(10) + 0.1*diag(ones(1,9),1) + ...
```

```
      0.1*diag(ones(1,9),-1)
A(1,1) = 0.85
A(2,1) = 0.15
A(9,10) = 0.15
A(10,10) = 0.85
```

La primera línea utiliza los comandos siguientes:

(1) `eye(10)`: construye una matriz cuadrada 10×10 con 1 en la diagonal y 0 fuera. Al multiplicarla por 0.8, queda este valor en la diagonal.
(2) `ones(1,9)`: genera una matriz de 1 fila y 9 columnas (un vector con nueve componentes) con unos.
(3) `diag(v, 1)`: genera una matriz cuadrada cuyo vector en la "diagonal 1" (es decir, la diagonal que está uno más arriba que la principal) contiene el vector `v`.
(4) `diag(v, -1)`: lo mismo que el anterior pero en la diagonal que está uno por debajo de la principal (la -1).

Y luego se ajustan los valores de los cuatro elementos diferentes de la regla general.

Si se comienza con una distribución de los bañistas como

$$\begin{bmatrix} 90 & 100 & 150 & 200 & 150 & 90 & 80 & 110 & 210 & 200 \end{bmatrix}$$

para calcular la distribución tras 15 minutos habría que calcular $A^{15}v$, siendo v el vector correspondiente a dicha distribución. Esto da, aproximadamente,

$$\begin{bmatrix} 87.2 & 135.1 & 142.0 & 143.9 & 136.7 & 128.2 & 132.1 & 154.4 & 184.9 & 135.0 \end{bmatrix}$$

entendiendo que los decimales son consecuencia de que no es un cálculo exacto sino una expectativa probabilística. Como puede verse, los valores máximos disminuyen y los mínimos han subido.

Tras media hora, la situación esperada será $A^{30}v$, que da

$$\begin{bmatrix} 90.6 & 136.7 & 137.9 & 138.2 & 138.3 & 140.6 & 147.7 & 159.5 & 171.5 & 118.4 \end{bmatrix}$$

que sigue mostrando una disminución de los máximos y un aumento de los mínimos. Los dos extremos van haciéndose menores que los valores intermedios.

¿Tiene este sistema un estado estacionario? Es decir: ¿llega un momento a partir del cual el sistema permanece en el mismo estado? Para que ello ocurra, tendría que existir un vector w tal que, cuando n tiende a infinito,

$$A^n v \to w.$$

Si esto pasa, es fácil comprobar que

$$Aw = w.$$

Es decir, si A (el sistema cuya matriz de transición es A) tiene un estado estacionario entonces existe un vector w no nulo tal que $Aw = w$. Un vector así se denomina un "autovector" del "autovalor" 1. El sistema

$$Aw = w$$

tiene por solución cualquier múltiplo del vector

$$w = \begin{pmatrix} 0.072 \\ 0.107 \\ 0.107 \\ 0.107 \\ 0.107 \\ 0.107 \\ 0.107 \\ 0.107 \\ 0.107 \\ 0.072 \end{pmatrix}$$

que, como puede comprobarse, es un vector la suma de cuyas componentes es 1 (así que representa una distribución de probabilidades). A la larga, dado cualquier estado inicial con N bañistas, se tiende a la distribución proporcional a esa: $0.107N$ en las zonas internas y $0.072N$ en las de los extremos.

Para visualizar la evolución del sistema puede utilizarse la siguiente secuencia de comandos:

```
for k=1:15:300
bar(A^k * v)
axis([1 10 0 1]);
pause(0.3)
end
```

que mostrará la distribución (esperada) como un diagrama de barras, a intervalos de 15 minutos (eso es lo que significa el `1:15:300`).

¿Cómo se calcularía la distribución inicial si se sabe que al cabo de media hora se tiene la siguiente?:

$$[50553260906040563244]$$

Ejemplo 8 Este ejemplo no posee un estado estacionario.

En una máquina con 6 cajetines, hay piezas distriubidas en ellos, que se pasan de un cajetín a otro. La transición es como sigue: todas las que están en el cajetín i pasan al $i+1$ para $i = 1, \ldots, 5$ y las que están en el 6 pasan las 1.

La matriz de transición es, claramente, la siguiente:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Si se empieza con una distribución $v_0 = (20, 10, 4, 30, 5, 2)^T$, la distribución en el siguiente instante es

$$v_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 20 \\ 10 \\ 4 \\ 30 \\ 5 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 20 \\ 10 \\ 4 \\ 30 \\ 5 \end{pmatrix},$$

que no es más que "rotar" los elementos del vector inicial. Al cabo de 6 iteraciones resulta $v_6 = A^6 v_0$, pero (como se puede comprobar fácilmente), A^6 es la identidad, así que $v_6 = v_0$: se tiene un ciclo de longitud 6. Esto impide que haya un estado estacionario, ¿por qué?

¿Cómo puede comprobarse que el ciclo no es de longitud más corta?

La matriz A de este ejemplo se construye fácilmente en Matlab:

```
A = diag(ones(5,1), -1)
A(1,6) = 1
```

Nótese que ha de usarse `ones(5,1)`, con un 5, pues la primera diagonal inferior tiene $6 - 1 = 5$ elementos. El segundo comando pone el 1 que hay en la primera fila, última columna.

Para atisbar cómo el modelo no tiene un estado estacionario, puede dibujarse un ejemplo:

```
v = [200 300 100 150 50 20]';
for k=1:100
bar(A^k*v);
axis([0 6 0 300]);
```

```
pause(0.3);
end
```

se observa que los individuos pasan de un estado a otro de manera cíclica, sin acercarse a una distribución fija. —

Ejemplo 9 (Una simplificación del sistema de puntos de los conductores). Un conductor posee, al obtener su carnet, 14 puntos, que puede ir perdiendo por infracciones o ganando haciendo cursillos en los que se le pide dinero (curioso...). Se sabe que, de mes a mes, el estado de puntos de un conductor sigue las siguientes reglas:

(1) Los que tienen entre 7 y 14 puntos, siguen con su misma cantidad con una probabilidad de 0.9. Pasan a tener un punto menos con una probabilidad de 0.08 y pasan a tener un punto más con una probabilidad de 0.02 (a estos les importa poco tener más puntos, pues "tienen muchos"). Los de 14 solo pasan a tener un punto menos con 0.1.
(2) Los que tienen entre 3 y 6 puntos, siguen con su misma cantidad con una probabilidad de 0.95. Pasan a tener un punto menos con una probabilidad de 0.025 (conducen con más cuidado) y pasan a tener un punto más con una probabilidad de 0.025 (están más dispuestos a hacer cursos).
(3) Los que tienen 1 ó 2 puntos, siguen con su misma cantidad con una probabilidad de 0.9. Pasan a tener un punto menos con una probabilidad de 0.01 (muy cuidadosos) y pasan a tener un punto más con una probabilidad de 0.09 (ponen mucho dinero para salir de la zona de riesgo).
(4) Los que tienen 0 puntos se quedan en 0 con una probabilidad de 0.1 (muy poca gente puede permitírselo, aunque si los puntos los regaló la abuelita Susana, entonces...), pasan a tener 7 con una probabilidad de 0.6 y pasan a tener 14 con una probabilidad de 0.3 (hay gente muy dispuesta a dejarse timar por la DGT).

Queremos tratar este problema con un modelo de Markov, pues encaja precisamente en él (es un sistema con un número finito de estados y la evolución de un individuo depende únicamente del estado en que se encuentra). La matriz de transición, tal como está descrita, es (téngase en cuenta que la matriz es 15×15 pues hay desde 0 hasta

14 puntos):

$$A = \left(\begin{array}{c|cc|cccc|cccccccc}
0.10 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
\hline
0.00 & 0.90 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.09 & 0.90 & 0.025 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
\hline
0.00 & 0.00 & 0.09 & 0.95 & 0.025 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.025 & 0.95 & 0.025 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.025 & 0.95 & 0.025 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.025 & 0.95 & 0.08 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
\hline
0.60 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.025 & 0.90 & 0.08 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 0.90 & 0.08 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 0.90 & 0.08 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 0.90 & 0.08 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 0.90 & 0.08 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 0.90 & 0.08 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 0.90 & 0.10 \\
0.30 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 0.90
\end{array}\right)$$

Esta matriz es un buen ejemplo para aprender a utilizar índices en Matlab. Se han añadido líneas que dividen A en zonas más o menos regulares. Hay cuatro submatrices cuadradas en la diagonal, de tamaños sucesivos 1, 2, 4 y 8. Además, se han indicado en negrita los valores no nulos que hay fuera de dichas matrices, que introduciremos *a mano*.

Para empezar, construyamos una matriz 15×15 llena de ceros:

```
A = zeros(15);
```

Modifiquemos ahora la casilla $A_{1,1}$:

```
A(1,1) = 0.1;
```

La submatriz de tamaño 2×2 está en las filas $2, 3$ y en las columnas $2, 3$. Como solo tiene cuatro elementos, la escribimos completa:

```
A([2 3], [2 3]) = [0.9 0.01; 0.09 0.9];
```

La siguiente submatriz, de tamaño 4×4, la generamos de la siguiente manera: primero los elementos de la diagonal, que valen 0.95,

```
A([4:7], [4:7]) = A([4:7], [4:7]) + 0.95*eye(4);
```

y a continuación las dos diagonales arriba y abajo:

```
A([4:7],[4:7])=A([4:7],[4:7])+diag(0.025*ones(3,1),1);
A([4:7],[4:7])=A([4:7],[4:7])+diag(0.025*ones(3,1),-1);
```

La construcción de la última matriz es similar a la recién hecha

```
A([8:15],[8:15])=A([8:15],[8:15])+0.9*eye(8);
A([8:15],[8:15])=A([8:15],[8:15])+diag(0.08*ones(7,1),1);
A([8:15],[8:15])=A([8:15],[8:15])+diag(0.02*ones(7,1),-1);
```

Salvo el elemento $A_{14,15}$ que es 0.1:

```
A(14,15)=0.1;
```

Finalmente, hay que dar su valor a los elementos en negrita:

```
A(1,2) = 0.01;
A(8,1) = 0.6;
A(15,1) = 0.3;
A(3,4) = 0.025;
A(4,3) = 0.09;
A(7,8) = 0.08;
A(8,7) = 0.025;
```

Hecho esto, se puede proceder al estudio del sistema.

(1) Un conductor comienza con 14 puntos. ¿cuál es la probabilidad de que tenga menos puntos al cabo de un año?

```
v = zeros(15,1);
v(15) = 1;
p = A^12*v;
p(14)
```

da más o menos 0.33 de que siga con 14, así que la probabilidad de que tenga menos es aproximadamente 0.67.

(2) Cuál es la probabilidad de que un conductor con 6 puntos tenga 5 al cabo de dos años.

```
v = zeros(15,1);
v(7) = 1;
p=A^24*v;
p(6)
```

da más o menos 0.24.

(3) Se comienza el sistema con 1000 conductores con 6 puntos y 2000 con 14 puntos. Calcular la distribución esperada al cabo de 5 años.

```
v=zeros(15,1);
v(7)=1000;
v(15)=2000;
p=A^60*v
```

en horizontal, este vector es, más o menos,

```
p' = [0 1 13 63 139 276 473 268 253 313 361 350 274 163
    53]
```

así que, por ejemplo, se espera que haya 473 conductores con 5 puntos, 361 con 10, etc.

—

Ejemplo 10 Si se derrama una gota de tinta en un recipiente con agua, la tinta tiende a difundirse (disolverse) homogéneamente por todo el volumen. Este proceso puede modelarse con una ecuación en derivadas parciales, la *ecuación de difusión*, que es la misma que la *ecuación del calor*. Si se considera el problema en una dimensión (piénsese en la concentración de tinta en la superficie, proyectada en el eje OX, por ejemplo), es posible construir un modelo discreto bastante simple basado en las cadenas de Markov. Una manera de hacerlo es así:

- Se discretiza la anchura del recipiente en, digamos 300 elementos.
- Se postula que la concentración de tinta cumple la ley siguiente: una partícula tiene probabilidad 0.8 de quedarse en su lugar, 0.1 de ir hacia la derecha y 0.1 de ir hacia la izquierda.
- En las dos casillas de los bordes, la probabilidad de quedarse es 0.8 y la de ir a la contigua es 0.2.

Con estos tres principios, se tiene un proceso de Markov. La situación inicial vendrá dada por el vector v_0, sea el que sea.

La matriz de transición es sencilla de construir: 0.8 en la diagonal principal y 0.1 en las dos diagonales contiguas, excepto en los lugares $A_{2,1}$ y $A_{299,300}$ que es 0.2:

```
A = 0.8*eye(300);
A = A + 0.1*diag(ones(299,1),1) + ...
      0.1*diag(ones(299,1),-1);
A(2,1)=0.2;
A(299,300)=0.2;
```

La evolución del sistema (que es muy lenta, pues la ley que hemos impuesto hace que la tinta tarde mucho en disolverse) puede observarse como se indica a continuación. Partimos de una mancha de tinta que llena solo la primera casilla:

```
v = zeros(300,1);
v(1) = 1000;
for k=1:100:10000
plot(A^k*v);
axis([0 300 0 80]);
```

```
pause(0.1);
end
```

se observa cómo la tinta se distribuye poco a poco homogéneamente. Al cabo de un millón de pasos, la cosa queda más clara:

```
bar(A^1000000*v);
```

Puede comenzarse con una distribución diferente, por ejemplo con varias manchas de tinta en diversos lugares:

```
w = zeros(300,1);
w(3)=1000;
w(100)=5000;
w(200)=7000;
w(295)=1000;
for k=1:1000:100000
plot(A^k*w);
axis([0 300 0 80]);
pause(0.1);
end
```

Esta manera de entender la difusión es moralmente la misma que Einstein describió en su trabajo de 1905 sobre el movimiento Browniano. La diferencia estriba, más que nada, en la ley de transición, que para él era más compleja —la que hemos descrito nosotros en una simplificación muy elemental pero que posee propiedades similares a la suya. —

Ejemplo 11 Un sistema algo más complejo que el Ejemplo 8 sin estado estacionario es el dado por la matriz de transición

$$A = \begin{pmatrix} 0 & 0.4 & 0 & 0.2 \\ 0.3 & 0 & 0.9 & 0 \\ 0 & 0.6 & 0 & 0.8 \\ 0.7 & 0 & 0.1 & 0 \end{pmatrix}$$

Si se comienza con el vector $v = (200, 300, 150, 400)$, la sucesión de estados termina siguiendo un patrón similar a

$$\begin{pmatrix} 118.75 \\ 487.50 \\ 231.25 \\ 212.50 \end{pmatrix} \to \begin{pmatrix} 237.50 \\ 243.75 \\ 462.50 \\ 106.25 \end{pmatrix} \to \begin{pmatrix} 118.75 \\ 487.50 \\ 231.25 \\ 212.50 \end{pmatrix} \to \cdots$$

que, como se ve, no es estacionario. Puede comprobarse que el sistema tiene un comportamiento similar si se comienza con cualquier otro vector v_0.

Para los siguientes ejercicios, es interesante la secuencia de comandos que se indica: se construye una matriz (la de paso) M, se parte de un vector inicial v y se quiere representar la evolución del sistema. Por ejemplo:

$$M = \begin{pmatrix} 0.9 & 0.1 & 0 & 0.1 \\ 0.1 & 0.8 & 0 & 0 \\ 0 & 0 & 0.8 & 0.1 \\ 0 & 0.1 & 0.2 & 0.8 \end{pmatrix}, \; v = \begin{pmatrix} 100 \\ 200 \\ 10 \\ 90 \end{pmatrix}.$$

```
M = [0.9 0.1 0 0.1; 0.1 0.8 0 0; 0 0 0.8 0.1; 0 0.1 0.2
   0.8];
v = [100 200 10 90]';
r =[];
for j=1:50 ; r = [r M^j * v]; end;
bar(r);
```

La gráfica obtenida presenta en la primera columna el número de elementos que habrá en cada paso del primer tipo. En la segunda, del segundo tipo, en la tercera del tercero y en la cuarta, del cuarto.

Ejercicio 29: Modelar, mediante una cadena de Markov, el proceso correspondiente a una infección. Se suponen los siguientes estados: sano, enfermo, curado. La probabilidad de pasar de sano a enfermo es 1/10, la de pasar de sano a curado es 0 (claro); la de pasar de enfermo a curado es 1/10 y la de pasar de enfermo a sano es 0. Finalmente, la probabilidad de pasar de curado a enfermo ó sano es 0. El resto de probabilidades son las correspondientes al quedarse como uno está.

Estudiar el comportamiento del sistema para distintos valores iniciales. ¿Converge a algún punto?

Se sabe que al cabo de 25 etapas, el estado es: sanos 4000, enfermos 2000 y curados 6000. ¿cuál era la distribución inicial?

Ejercicio 30: Modelar el mismo proceso que en el Ejercicio 29 pero suponiendo que los enfermos curados *sí pueden volver a enfermar*, pero con menos probabilidad que los sanos. Estudiar el proceso y ver si tiene o no un punto de estabilidad.

¿Cuál es el estado inicial si se conoce que al cabo de 100 etapas es: sanos 4500, enfermos 3000, curados 3500? —

Ejercicio 31: Modelar el Ejemplo 10 pero con los extremos "muy reflectantes": es decir, los dos puntos de los extremos tienen una probabilidad muy baja de quedarse con materia mientras que la tienen muy alta de mandársela a los puntos adyacentes. Comparar la evolución y el estado estacionario (si lo hay) de este sistema con el del Ejemplo 10.

¿Cómo se calcularía el estado previo a uno dado? —

Ejercicio 32: Modelar un sistema "circular" de 6 estados. La probabilidad de pasar del i al $i+1$ es 0.5. La probabilidad de quedarse en el i es 0.5 y la probabilidad de pasar del 6 al 1 es 0.5. Con diferentees condiciones iniciales, observar la evolución del sistema. ¿Tiene estado estacionario? —

CAPÍTULO 4

Ecuaciones Diferenciales Ordinarias

En las siguientes sesiones se estudiarán ejemplos de ecuaciones diferenciales ordinarias; sobre todo problemas de una variable pero, si hay tiempo, se introducirán algunos ejemplos de varias variables.

1. Perfil de una etapa ciclista

El concepto más importante de la asignatura de Cálculo (y de todo el Cálculo Infinitesimal) es el de *derivada* $y'(x)$ de una función $y(x)$ en un punto x_0: la *pendiente de la gráfica de* $y(x)$ *en el punto* $(x_0, y(x_0))$. Lo usaremos con mucha frecuencia.

Sean $\mathbf{y}' = (y'_0, y'_1, \ldots, y'_{n-1})$ las distintas pendientes de la carretera en diferentes puntos de una etapa ciclista y sean $\mathbf{x} = (x_0, x_1, \ldots, x_n)$ las coordenadas x (horizontales) de cada uno de dichos puntos (téngase en cuenta que $\mathbf{x}$ no son los puntos kilométricos, sino las coordenadas en el eje OX de cada punto del trayecto). Obsérvese que hay un elemento menos en la lista de pendientes que en la de valores de x.

Si la carrera comienza en una altura y_0, ¿cómo podría calcularse de manera aproximada la altura en la que se está en cada punto x_i, para $i = 1, \ldots, n$?

Este problema tiene diversas soluciones (depende de cómo se plantee la aproximación). Parece que la manera más obvia de resolverlo es suponer que en cada intervalo $[x_i, x_{i+1})$ la pendiente es constante e igual a y'_i. De este modo, la altura en x_1 se aproximaría como $y_1 = y_0 + y'_0(x_1 - x_0)$ (la altura que se sube o baja es la pendiente por el incremento en horizontal). Una manera de comprobarlo es viendo que la ecuación de la recta que pasa por (x_0, y_0) y tiene pendiente y'_0 es

$$Y = y_0 + y'_0(x - x_0).$$

Repitiendo este argumento, se calcularía la altura aproximada en x_2 (que llamamos y_2), en x_3, etc. hasta x_n.

Es importante darse cuenta de que para resolver el problema se requiere la altura inicial y_0: sin ella no se puede ni siquiera intentar aproximar y_1.

Ejemplo 12 Hagamos un ejemplo con solo 5 nodos. Supongamos que la carretera tiene las indicaciones de pendiente: $+4\%$, -2%, $+7\%$, $+12\%$ en las coordenadas horizontales (en km): $0, 2, 5, 7p$ y que la carrera termina en el kilómetro 8. La etapa comienza a 850m. Dibújese un perfil aproximado.

Este ejemplo se puede hacer a mano completamente:

(1) Como el primer tramo tiene 2km de longitud *horizontal* y la pendiente es del 4%, suponiendo que esta es constante en todo el tramo, la carretera subirá $2\text{km} \times 4\% = 80\text{m}$. Como la etapa comienza a 850m, se alcanzará (aproximadamente) una nueva altura de $850\text{m} + 80\text{m} = 930\text{m}$.

(2) El segundo tramo tiene 3km de longitud (horizontal) y su pendiente es -2% al comienzo; suponiendo que es constante, el tramo bajará aproximadamente $3\text{km} \times 2\% = 60\text{m}$. Por tanto, termina a $930\text{m} - 60\text{m} = 870\text{m}$ de altura.

(3) El tercer tramo tiene 2km de longitud y suponemos que la pendiente es constante e igual a 7%. Así que se suben 140m y termina a $870\text{m} + 140\text{m} = 1010\text{m}$.

(4) Finalmente, hay un tramo de 1km (en horizontal) de pendiente que aproximamos como 12%: se suben 120m y la carrera termina a $1010\text{m} + 120\text{m} = 1130\text{m}$

El perfil aproximado se muestra en la Figura 1. —

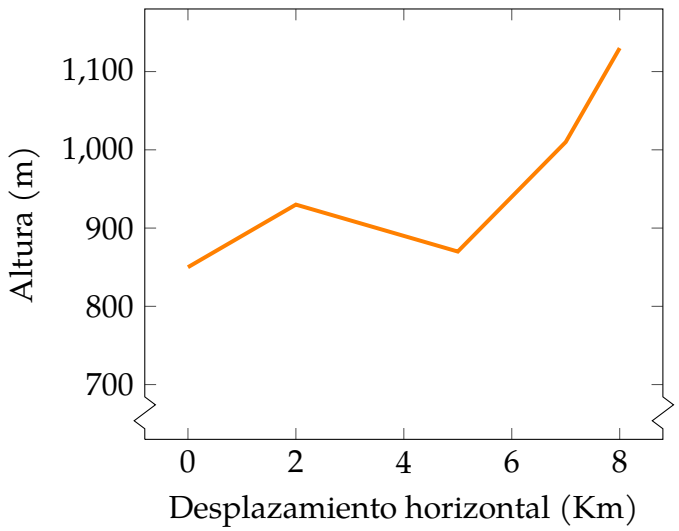

FIGURA 1. Perfil aproximado de una carrera.

Ejemplo 13 El mismo ejemplo con muchos más datos. Constrúyase un vector **x** de 25 componentes entre 0 y 200 (estas serán las posiciones en horizontal). Constrúyase un vector aleatorio $\mathbf{y}'$ (désele un nombre como yp) con valores entre -15% and 15% (razonables para una carretera). Fíjese un valor y_0 como altura inicial. Calcúlense los valores aproximados sucesivos para cada x_i y dibújese el perfil (aproximado) de la etapa.

Se da una solución a este enunciado en el listado 4.1 y se presenta un perfil aproximado en la Figura 2.

```
% Una 'etapa' de una carrera ciclista con pendientes aleatorias.

% Tengase en cuenta que hay una pendiente menos que coordenadas x
x  = linspace(0, 200, 25);

% Explicacion de la siguiente linea:
%   Tomense 24 valores aleatorios entre 0 y 1
%   Multipliquense por 0.30 para que esten entre 0 y 0.30
%   Restese 0.15 para que queden entre -0.15 y 0.15, como dice el enunciado
yp = rand(1,24) * .30 - .15;

% Altura inicial (elijase una)
y0 = 870;

% Construccion del perfil aproximado
% 1) Se fabrica la lista de "incrementos de altura"
%    (x(i+1) - x(i)) * yp(i)
h  = diff(x).*yp;

% 2) Se fabrica primero una lista de ceros para las alturas
%    y se pone el primer valor
y    = zeros(1, 25);
y(1) = y0;

% 3) Para cada segmento,
%      Anyadase a la lista de alturas el valor anterior mas
%      el incremento correspondiente
for s = 1:length(h)
    y(s+1) = y(s) + h(s);
end
% Seria mucho mas propio y claro escribir:
%  y(2:end) = y(1:end-1) + h;

plot(x,y);
```

LISTADO 4.1. Perfil aproximado de una etapa ciclista, primera versión.

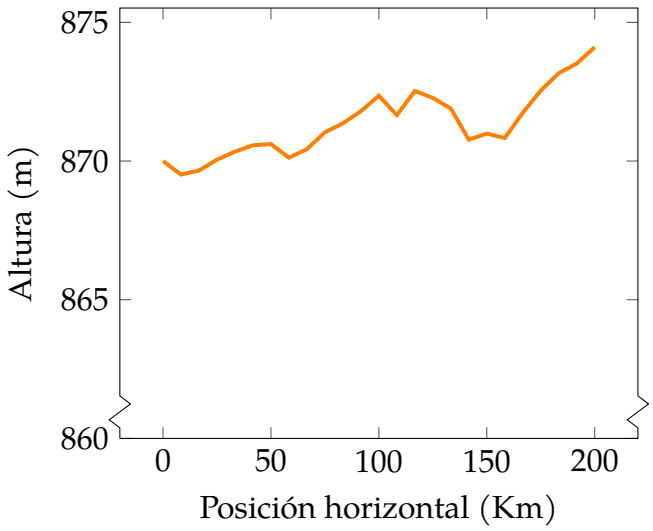

FIGURA 2. Perfil aproximado de una etapa (más larga).

Por supuesto, este método solo es *una* manera posible de resolver el problema del perfil de manera aproximada; hay más, aunque no muchas más razonables con los datos de que se dispone.

Ejemplo 14 Se puede afrontar el problema de un modo distinto. En el Ejemplo 13, se utiliza la pendiente *al comienzo del intervalo* como si fuera constante en todo él; esto puede ser poco adecuado. De hecho, uno podría pensar "¿por qué se va a usar el valor en el extremo izquierdo y no el del extremo derecho?" Quizás una manera más razonable de enfrentarse al problema sería usar, de algún modo, la información en ambos extremos para cada intervalo. En lugar de tomar y'_{i-1} o y'_i como la pendiente para *todo el intervalo* $[x_{i-1}, x_i]$, uno podría elegir utilizar el valor medio como "pendiente media" en tal intervalo:

$$\tilde{y}'_{i-1} = \frac{y'_{i-1} + y'_i}{2}$$

y calcular cada altura sucesiva y_i como:

$$y_i = y_{i-1} + (x_i - x_{i-1})\tilde{y}'_{i-1}.$$

Ha de tenerse en cuenta (y esto es importante) que para poder hacer este cálculo, es preciso conocer de antemano *la pendiente en el último punto*, así que ahora $\mathbf{y}'$ debe tener tantas componentes como $\mathbf{x}$ para que este método funcione.

Este argumento se utiliza en el listado 4.2.

```
% Etapa ciclista usando valores medios de pendientes

% Comenzamos con las coordenadas x equidistribuidas
x  = linspace(0, 200, 25);

% Creamos una lista de pendientes:
yp = rand(1,25) * .30 - .15;

% Pendiente inicial (a elegir)
y0 = 870;

% Pendientes medias en cada intervalo
yp_means = (yp(1:end-1) + yp(2:end))/2;

% Vector de "incrementos de alturas"
h  = diff(x).*yp_means;

% Vector de alturas "reales": la primera es y0
y    = zeros(1, 25);
y(1) = y0;

% Forma propia de Matlab de hacer la cuenta entera:
for s = 1:length(h)
    y(s+1) = y(s) + h(s);
end

plot(x,y);
```

LISTADO 4.2. Perfil aproximado de una etapa ciclista, versión del "valor medio".

Ejercicio 33: Usando los códigos de los listados 4.1 y 4.2, compárense las soluciones que se obtienen para el mismo problema: hágase "analíticamente" (es decir, utilizando diferencias absolutas y relativas) y gráficamente.

Ejercicio 34: Escribir dos archivos de Matlab, uno para cada uno de los métodos de los ejemplos 13 y 14. En cada uno de ellos se ha de definir una función que recibe como argumentos dos vectores de la misma longitud `x` e `yp` y un número real `y0`. La salida será un vector `y` que contiene la lista de alturas de la etapa en cada valor de `x`. Llámese a dichas funciones `profile_euler.m` y `profile_mean.m`.

Úsense varias veces por pares (con los mismos datos las dos) y compárense las gráficas. ¿Cuáles son más suaves? ¿Por qué?

Se da un ejemplo del contenido del archivo `profile_euler.m` en el listado 4.3.

```
% profile_euler(x, yp, y0):
%
% Dadas listas de coordenadas x, de pendientes yp y una altura inicial y0
% devuelve las alturas en cada tramo de una carretera que tenga pendientes
% yp (para cada valor de x), comenzando en altura y0.
% Se supone que la pendiente es constante e igual a la del extremo izquierdo
% en cada tramo.
function [y] = profile_euler(x, yp, y0)
    y = zeros(size(x));
    y(1) = y0;
    for s = 1:length(x)-1
        y(s+1) = y(s) + yp(s)*(x(s+1) - x(s))
    end
end
```

LISTADO 4.3. Código de ejemplo para el archivo `profile_euler.m`.

2. Integración Numérica de EDO

La sección anterior era solo una introducción al problema de la integración (resolución) numérica de Ecuaciones Diferenciales Ordinarias (EDO de ahora en adelante). Solo consideraremos, en este curso, ecuaciones de la forma:

$$y' = f(x, y).$$

Y ahora vamos a explicar qué significa esta fórmula.

La derivada de una función $y(x)$ es (repetimos) la *pendiente de la gráfica* $(x, y(x))$ *en cada punto*: si $y(x)$ es derivable y x_0 es un número, entonces $y'(x_0)$ es justamente la *pendiente* (como la pendiente de una *carretera*, es lo mismo) de la gráfica de $y(x)$ en el punto $(x_0, y(x_0))$.

En la Figura 3 se muestra la gráfica de la función $y(x) = \operatorname{sen}(x)$ y la recta tangente a dicha gráfica en el punto $(\frac{2\pi}{6}, \frac{\sqrt{3}}{2})$. Obsérvese que la pendiente de esta recta es 0.5 (i.e. está inclinada $\pi/6$, 30 grados): se puede ver que sube dos unidades en vertical en un desplazamiento horizontal de 4. Este valor 0.5 es justamente lo mismo que el valor de $y'(x) = \cos(x)$ en el punto $x_0 = \frac{2\pi}{6}$. Así incidimos en la idea de que "derivada es lo mismo que pendiente".

Con esta idea clara, la expresión (que es una ecuación):

$$y' = f(x, y)$$

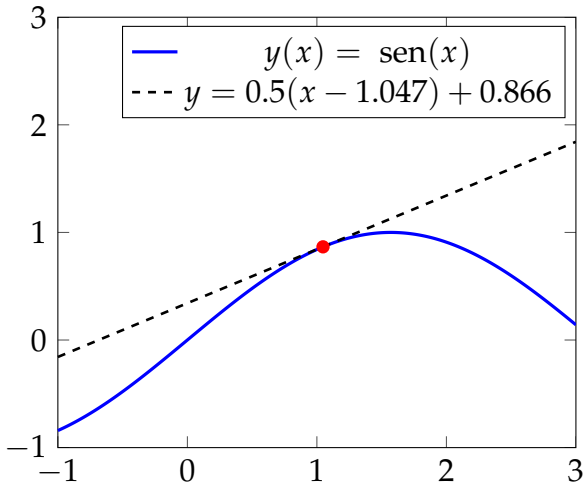

FIGURA 3. Gráfica de $y(x) = \operatorname{sen}(x)$ y su tangente en $\left(\frac{2\pi}{6}, \frac{\sqrt{3}}{2}\right)$.

solo puede tener una interpretación: "la función $y(x)$ cumple que su pendiente en cada punto $(x, y(x))$ es justamente $f(x, y(x))$". Dicho brevemente, "la pendiente de y en x es $f(x, y)$".

En el ejemplo de la carrera ciclista, dar las pendientes de la carretera en cada punto no basta para calcular las alturas correspondientes: hace falta un dato inicial (la altura a la que se comienza). Esto pasa también con una EDO: plantear la ecuación $y' = f(x, y)$ no es bastante para tener una solución *concreta*, hace falta dar una condición inicial más: la "altura incial" de la gráfica de $y(x)$. Así se llega a la noción de *problema de condición inicial*:

DEFINICIÓN 1. Un *problema de condición inicial* es una EDO $y' = f(x, y)$ y un par (x_0, y_0) que se llama *condición inicial*.

Una vez que tal condición inicial $y(x_0)$ se da, la EDO $y' = f(x, y)$ tiene una única solución [1].

Ejemplo 15 El problema de condición inicial

$$y' = y, \;\; y(0) = 1,$$

tiene como solución la función $y(x) = e^x$. Compruébese.

Si en lugar de $y(0) = 1$ se toma $y(0) = K$, entonces la solución es

$$y(x) = Ke^x.$$

[1] Bajo ciertas condiciones que son "habituales".

¿Qué ocurre si el valor inicial es $y(1) = 0.5$? ¿Se puede calular la solución?

Por tanto, para enunciar un problema de condición inicial, se necesitan los siguientes datos:

(1) La función $f(x, y)$.
(2) El par (x_0, y_0): un valor de la x y el correspondiente valor $y(x)$.

Pero, puesto que vamos a calcular una solución aproximada, se requiere un dato adicional: la lista de valores x_i en los que se desea calcular el valor aproximado de la solución (igual que en el ejemplo de la carrera ciclista). Este dato será habitualmente un vector de "coordenadas x". También es frecuente, en lugar de dar el vector $(x_0, x_1, \ldots, x_n)$, fijar una distancia h constante entre x_i y x_{i+1} y especificar cuántos intervalos hay: de esta manera se tendrá una lista de puntos $(x_0, x_0 + h, x_0 + 2h, \ldots, x_0 + nh)$.

La "solución" (aproximada) del problema de condición incial será la lista de valores aproximados $y(x)$ de una función que cumpla la ecuación. Cómo puede calcularse es el contenido del resto del capítulo.

2.1. Método de Euler. El primer método que a uno se le ocurre para aproximar la solución de un problema de condición inicial es el de Euler, que consiste en aplicar el mismo principio que en el perfil de la carrera ciclista, como en el Ejemplo 13. Si n es la longitud del vector $\mathbf{x}$, se puede describir el método de Euler como en el Algoritmo 2.1: tan fácil como hacerlo. Téngase en cuenta que x_0 e y_0 son *parte de los datos iniciales*.

Algoritmo 3 Método de Euler.

for $i = 1 \ldots n$ **do**
  $y_i = y_{i-1} + f(x_i, y_i)(x_i - x_{i-1})$
end for
return $(y_0, y_1, \ldots, y_n)$

La línea interior del bucle **for** es exactamente la misma que para el perfil de la carrera ciclista: la "altura" y_i en la posición horizontal x_i se aproxima como la altura y_{i-1} en el punto anterior x_{i-1} más la pendiente que indica la ecuación, $f(x_{i-1}, y_{i-1})$ multiplicada por el desplazamiento horizontal $(x_i - x_{i-1})$. El algoritmo devuelve la lista de alturas al final.

La implementación del algoritmo 2.1 como una función en un archivo-`m` de Matlab debería ser directa. Incluimos el código en el

Ejemplo 16 como ayuda al lector. La función ha sido llamada `euler` y, como consecuencia, el archivo ha de llamarse `euler.m`.

Ejemplo 16 Dados una función $f(x,y)$ (que supondremos que es una *función anónima*), un vector **x** de posición horizontales y un valor inicial y_0, el listado 4.4 sirve como código para implementar el algoritmo de Euler con dichos datos. Recuérdese que, para que funcione, el nombre del archivo ha de ser euler.m. —

```
% Metodo de Euler 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] = euler(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 Euler
    for s = 2:length(x)
         y(s) = y(s-1) + f(x(s-1), y(s-1)).*(x(s) - x(s-1));
    end
end
```

LISTADO 4.4. Código de Matlab para el Algoritmo de Euler.

Ejercicio 35: Utilícese la función `euler` que se ha implementado para resolver numéricamente los siguientes problemas de condición inicial. Úsense tantos puntos como se desee (no más de 100 ni menos de 10), cuando no s especifiquen. Dibújense las soluciones que se encuentran.

- Para $x \in [0,1]$, resolver $y' = 2x$ con $y(0) = 1$. Úsense 10 puntos. Dibújese, en la misma gráfica, la solución real: $y(x) = x^2 + 1$.
- Para $x \in [0,1]$, resolver $y' = y$ con $y(0) = 1$. Úsense 15 puntos. Dibujar, en la misma gráfica, la solución real: $y(x) = e^x$.
- Para $x \in [0,1]$, resuélvase $y' = xy$ con $y(0) = 1$. Dibujar, en la misma gráfica, la solución real: $y(x) = e^{\frac{x^2}{2}}$.
- Para $x \in [-1,1]$, resuélvase $y' = x + y$ con $y(-1) = 0$.

- Para $x \in [-\pi, 0]$, resuélvase $y' = \cos(y)$ con $y(-\pi) = -1$.
- Para $x \in [1, 3]$, resuélvase $y' = y - x$ con $y(1) = -1$.

—

El método de Euler no es, en general, la *mejor manera* de aproximar soluciones de EDOs. Como se ve en los tres primeros ejemplos del Ejercicio 35, las soluciones que se calculan con este método están siempre debajo de la solución real si esta es convexa (la derivada es creciente), y están siempre encima si esta en cóncava (la derivada es decreciente). Esta falta de precisión puede evitarse de diversas maneras. La que vamos a estudiar en detalle es el *método de Heun*.

2.2. Método de Heun. Este método consiste en aplicar la idea que se explicó en el Ejemplo 14 para "mejorar" la aproximación al perfil de una carrera: usar el valor medio entre las pendientes inicial y final en un tramo.

Sin embargo, hay una diferencia importante con respecto al problema de la carrera ciclista: en esta, la pendiente al final del tramo se sabe con exactitud porque se conoce la carretera (i.e. por donde pasa realmente la carrera) mientras que en una ecuación diferencial, precisamente porque no se conoce la solución, no puede conocerse su pendiente al final de un tramo. Dada la EDO:

$$y' = f(x, y)$$

y el valor inicial (x_0, y_0), la pendiente de la solución en este punto es conocida (es $f(x_0, y_0)$) pero aun no se conoce el valor aproximado de y_1, y por tanto, $f(x_1, y_1)$ no se conoce. Lo que se hace es una "predicción" de cuál sería el valor de y_1 utilizando el método de Euler, que llamamos $\tilde{y}_1$ y, usando esta predicción, calcular el valor medio entre la pendiente en el punto (x_0, y_0) y la pendiente en este punto predicho $(x_1, \tilde{y}_1)$. Con este valor medio se *corrige* la predicción un poco. Más o menos, como se explica:

(1) Póngase uno en (x_0, y_0).
(2) Calcúlese $(x_1, \tilde{y}_1)$ usando el método de Euler como "predicción": para ello se utiliza $k_1 = f(x_0, y_0)$ como pendiente.
(3) Calcúlese $k_2 = f(x_1, \tilde{y}_1)$: la pendiente en el punto predicho por Euler.
(4) En lugar de k_1 ó k_2, tómese como pendiente el valor medio $k = (k_1 + k_2)/2$.
(5) Úsese k como pendiente (x_0, y_0). Es decir, calcúlese:

$$y_1 = y_0 + k(x_1 - x_0).$$

Y este argumento se repite en cada iteración.

Veamos un ejemplo:

Ejemplo 17 Dado el problema de condición inicial (PCI):

$$y' = x - y \;\; y(2) = 0.$$

sea $\mathbf{x} = (2, 2.25, 2.5)$ una secuencia de coordenadas x. Usemos el método de Heun para calcular una aproximación de la solución del PCI en $x = 2.25$ y $x = 2.5$.

Debemos realizar dos pasos del algoritmo de Heun. Haremos el primero en detalle y el segundo lo mostraremos rápidamente.

Primer paso: $x_0 = 2, y_0 = 0$.

- Según el método de Euler, la pendiente sería $k_1 = f(x_0, y_0) = 2$ (este punto (x_0, y_0) es la condición inicial), así que se tendría $\tilde{y}_1 = 0 + 2 \times 0.25 = 0.5$.
- Con estos valores, $k_2 = f(2.25, 0.5) = 1.75$.
- El valor medio entre k_1 y k_2 es $k = 1.875$.
- Por tanto, el valor de y_1 dado por Heun es $y_1 = y_0 + k(x_1 - x_0) = 0 + 1.875 \times 0.25 = 0.46875$.

En fin, $(x_1, y_1) = (2.25, 0.46875)$.

Segundo paso: $x_1 = 2.25, y_1 = 0.46875$.

De estos datos sale $k_1 = f(x_1, y_1) = 1.7812$ y $\tilde{y}_2 = 0.46875 + 1.7812 \times 0.25 = 0.91405$, de donde $k_2 = f(x_2, \tilde{y}_2) = 1.5859$. Así pues, $k = 1.6835$ y el resultado final es $(x_2, y_2) = (2.5, 0.88962)$.

La solución real del problema es $y(x) = x - e^{2-x} - 1$, que vale en $x = 2.5$: $y(2.5) = 0.89347$, el error relativo que se ha cometido es $\frac{|0.89347 - 0.88962|}{0.89347} \simeq 0.004$, que es bastante pequeño (téngase en cuenta que los pasos en la variable x son muy grandes: 0.25 cada uno). —

En el Algoritmo 2.2 se expresa este método con precisión. Se supone que $f(x, y)$ y (x_0, y_0) son datos.

Algoritmo 4 Método de Heun.

---

for $i = 1 \ldots n$ **do**
  $k_1 = f(x_{i-1}, y_{i-1})$
  $\tilde{y}_i = y_{i-1} + k_1(x_i - x_{i-1})$
  $k_2 = f(x_i, \tilde{y}_i)$
  $k = \frac{k_1 + k_2}{2}$
  $y_i = y_{i-1} + k(x_i - x_{i-1})$
end for
return $(y_0, \ldots, y_n)$

Ejercicio 36: Implementar el método de Heun en un archivo `m`, usando una función que se llame `heun` (recuérdese que el archivo ha de llamarse `heun.m`). Utilícese para encontrar soluciones aproximadas a los PCI del Ejercicio 35 y compárense estas con las encontradas usando el algoritmo de Euler. ¿Cuáles son mejores? ¿Siempre es mejor un método que otro? —

Ejercicio 37: Impleméntense los métodos de Euler y Heun utilizando una hoja de cálculo cualquiera (Excel, Numbers, LibreOffice...). Explíquese cómo se haría y utilícense para los ejemplos del Ejercicio 35. Úsese la hoja de cálculo para dibujar las soluciones. —

CAPÍTULO 5

Ecuaciones Diferenciales Ordinarias (II)

En este capítulo vamos a desarrollar en algo de detalle unos ejemplos elementales de EDOs de varias variables, que son más gráficos y realistas que los del capítulo anterior.

1. El modelo predador/presa de Lotka-Volterra

El primer modelo diferencial que modela un sistema biológico complejo es el de "Lotka-Volterra": se utiliza para describir la evolución de un entorno en el que dos especies conviven; una de ellas actúa como depredador y otra como presa (el ejemplo más habitual que se da es el de zorros y conejos).

Supongamos que $x(t)$ denota la población de la especie "presa" en el momento t y que $y(t)$ denota la población de la especie "predador". La ecuación diferencial de Lotka-Volterra que describe la evolución conjunta de las dos especies se basa en las siguientes suposiciones (simplistas, claro está):

(1) La población de presas crece en proporción a su tamaño.
(2) Una presa muere solo como consecuencia de ser víctima de un depredador. Esto ocurre con una probabilidad constante.
(3) La población de depredadores muere en proporción a su tamaño.
(4) Los predadores solo se multiplican en proporción a las presas que comen.

Estas cuatro reglas elementales (insistimos, muy simples), se trasladan a ecuaciones como se explica a continuación.

Del punto 1 se deduce que existe una constante $\alpha > 0$ tal que

$$\dot{x}(t) = \alpha x(t) + \dots$$

donde los puntos denotan otra expresión que habrá que descubrir. Si solo se tuviera esta ecuación, habría un crecimiento de presas exponencial (la ecuación $\dot{x}(t) = \alpha x(t)$ tiene como solución $x(t) = Ke^{\alpha t}$, donde K es el valor inicial).

Del punto 3 se deduce que existe un número $\gamma > 0$ tal que

$$\dot{y}(t) = -\gamma y(t) + \dots$$

igual que antes para las presas. Esto hace que, de por sí, los depredadores decrezcan según una ley exponencial negativa.

Es fácil convencerse de que la probabilidad de que un depredador se encuentre con una presa en algún lugar del terreno es proporcional al producto del número de predadores y de presas. Pero como no todo encuentro mutuo termina en una caza, se modela la probabilidad de que un predador coma a una presa como una constante β por dicho producto: $\beta x(t)y(t)$ (con $\beta > 0$). Por otro lado, el hecho de que un depredador se alimente no garantiza que se reproduzca; haremos que esto ocurra con un factor δ. Así, los puntos suspensivos que hemos dejado arriba han de sustituirse por $-\beta x(t)y(t)$ (en la parte de las presas) y por $\delta x(t)y(t)$ (en la parte de los depredadores). Se obtiene el sistema de ecuaciones diferenciales de Lotka-Volterra:

$$\begin{aligned} \dot{x}(t) &= \alpha x(t) - \beta x(t)y(t) \\ \dot{y}(t) &= -\gamma y(t) + \delta x(t)y(t) \end{aligned} \tag{2}$$

Ejemplo 18 Modélese un sistema del tipo Lotka-Volterra con parámetros $\alpha = 0.8$, $\beta = 0.4$, $\gamma = 2$, $\delta = 0.2$ y con poblaciones iniciales $x(0) = 18$, $y(0) = 3$. Utilícese el método de Euler para calcular una aproximación de su evolución con un paso temporal de 0.1, durante 12 unidades de tiempo.

Primero lo haremos a mano y después intentaremos escribir un programa que sirva para el caso general. El listado 5.1 muestra una manera (complicada) de hacer este ejemplo y de dibujar la evolución de ambas poblaciones en el tiempo.

```
% Simulacion de Lotka-Volterra, version 1
% El tiempo va de 0 a 12 en incrementos de 0.1
t=[0:.1:12];
% Se crea una lista de ceros del tamanyo adecuado para ambas variables
x=zeros(size(t));
y=zeros(size(t));
% Valores inciales
x(1)=18;
y(1)=3;

% Estas son las ecuaciones de Lotka-Volterra con los parametros del ejercicio
xp = @(t,x,y) 0.8*x - 0.4*x*y;
yp = @(t,x,y) -2*y + 0.2*x*y;

% El paso de Euler para ambas variables
for k=1:length(t)-1
    x(k+1) = x(k) + xp(t(k),x(k),y(k)) * (t(k+1) - t(k));
    y(k+1) = y(k) + yp(t(k),x(k),y(k)) * (t(k+1) - t(k));
```

```
end

% Dibujo de ambas especies en la misma grafica
plot(t,x)
hold on
plot(t,y,'r')
```

LISTADO 5.1. Una primera manera de resolver la ecuación de Lotka-Volterra.

Obsérvese (esto es quizás lo más importante de este sistema) que ambas poblaciones tienen un comportamiento creciente y decreciente con un desfase entre ellas.

Con algo de esfuerzo se puede verificar que los puntos críticos de la función $x(t)$ se alcanzan cuando la función $y(t)$ vale γ/δ y que los puntos críticos de $y(t)$ se alcanzan cuando la función $x(t)$ vale α/β (¿cómo se puede comprobar esto? es fácil pero requiere una explicación).

Sin embargo, se sabe (desde el punto de vista teórico) que el sistema de Lotka-Volterra *es periódico*. Las soluciones que se han dibujado no lo son (si uno calculara las soluciones para tiempos mucho más largos, vería cómo las funciones se vuelven cada vez más exageradamente "puntiagudas"). Esta anomalía se debe a la utilización del método de Euler: es demasiado simple, y esto se "paga". —

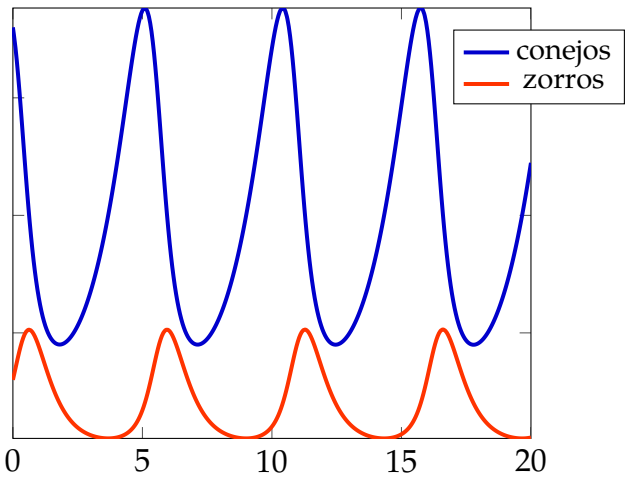

FIGURA 1. Ejemplo de sistema predador-presa. En este sistemas se supone siempre que los valores son "densidades de población".

Debería ser fácil darse cuenta de que el proceso llevado a cabo en el ejemplo anterior es generalizable si uno escribe un programa que ejecute el algoritmo de Euler para EDOs de varias variables.

La expresión

$$\dot{x}(t) = \alpha x(t) - \beta x(t)y(t)$$
$$\dot{y}(t) = -\gamma y(t) + \delta x(t)y(t)$$

se puede reescribir, en forma vectorial, como

$$\begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \end{pmatrix} = \begin{pmatrix} f_1(t,x,y) \\ f_2(t,x,y) \end{pmatrix}$$

donde $f_1(t,x,y) = \alpha x - \beta xy$ y $f_2(t,x,y) = -\gamma y + \delta xy$. En el caso general, se tendrá un vector de más de dos variables y uno podría escribir el sistema, de manera totalmente genérica como:

$$\begin{pmatrix} \dot{x}_1(t) \\ \vdots \\ \dot{x}_n(t) \end{pmatrix} = \begin{pmatrix} F_1(t,x_1,\ldots,x_n) \\ \vdots \\ F_n(t,x_1,\ldots,x_n) \end{pmatrix}$$

donde $F_1,\ldots,F_n$ son funciones de $n+1$ variables. Así pues, en el caso más general, para describir el problema de condición inicial, hace falta:

- Un *vector columna* de n componentes: las n funciones de $n+1$ variables.
- Los n valores iniciales para $t = 0$: $x_1(0),\ldots,x_n(0)$.

Para programar el método de Euler en este caso, hace falta por tanto definir una función, digamos `eulervector` que tiene por entrada:

- Una función anónima que devuelve un vector columna (la lista de $F_1,\ldots,F_n$ de arriba),
- Una lista de valores de tiempo en que calcular las soluciones aproximadas,
- Un vector columna con los n valores iniciales.

Dicha función debe devolver una lista de vectores columna de la misma longitud que la lista de valores temporales (por tanto, una matriz de n filas —número de variables— y l columnas —valores temporales)

Ejemplo 19 El código del listado 5.2 es una posible implementación de la función `eulervector` descrita arriba.

```
% Metodo de Euler para integrar EDOs, version vectorial.
% ENTRADA:
% 1) una funcion anonima f(t,x)
```

```
%    de dos variables:
%    t: numerica
%    x: vector columna
% 2) un vector T de valores temporales en que calcular la solucion
% 3) un vector x0 de valores iniciales de la x para t = T(1)

% SALIDA:
% una matriz de tantas filas como x0 y tantas columnas como T, que
% aproxima la solucion de la EDO dada por f

function [y] = eulervector(f, x, y0)
    % creamos y lleno de ceros con el tamanyo adecuado
    y = zeros(length(y0),length(x));
    % almacenamos las condiciones iniciales en la primera posicion
    y(:,1) = y0;

    % Bucle de Euler (es igual que siempre porque Matlab es vectorial)
    for s = 2:length(x)
         y(:,s) = y(:,s-1) + (x(s) - x(s-1)) * f(x(s-1),y(:,s-1));
    end
end
```

LISTADO 5.2. Implementación del método de Euler para funciones vectoriales.

Obsérvese en dicho código como la única diferencia entre nuestra implementación anterior del algoritmo de Euler en el listado 4.4, es la aparición explícita de las filas en el vector y tanto al inicio como al final de la línea:

```
    y = zeros(length(y0), length(x));
```

y en el resto de ocasiones, con los puntos suspensivos en la expresión `y(:, ...)`.

Usando esta función, el modelo de Lotka-Volterra del Ejemplo 18 se puede resolver con el código del listado 5.3.

```
% Definase la funcion (con los mismos parametros)
% Observese que, como x es un vector (columna)
% uno puede hacer referencia a cada componente con un solo indice:
f = @(t, x) [0.8*x(1) - 0.4*x(1)*x(2) ; -2*x(2) + 0.2*x(1)*x(2)];

% Inicializamos la lista de tiempos
T = [0:.1:12];
% Condiciones iniciales de poblacion
X0 = [18; 3];
% Resolvemos el problema
X = eulervector(f, T, X0);
% Hacemos la grafica (que Matlab se encarga de colorear)
```

```
plot(T, X)
```

LISTADO 5.3. Lotka-Volterra usando `eulervector`.

Se ve también cómo no hace falta utilizar dos variables diferentes `x` e `y`, sino un simple vector "columna" `x` y que las referencias a cada componente se hacen con los índices correspondientes. Así, al definir `f`, en lugar de usar `x` como anteriormente, se usa `x(1)` y, en lugar de `y`, se usa `x(2)`. También se ve esto en la solución, que se almacena en una sola variables (en el ejemplo, `X`) que (en el ejemplo) tiene dos columnas, una para cada componente del sistema.

En fin, nótese que Matlab dibuja cada fila de una matriz con un color distinto, como listas diferentes. Esto hace que sea innecesario el `hold on`. —

Ejercicio 38: Escríbase una función `heunvector` que implemente el método de Heun para ecuaciones diferenciales vectoriales. Debería ser obvio que la única diferencia con el listado 5.2 ha de consistir en cambiar la línea

```
y(:,s) = y(:,s-1) + (x(s) - x(s-1)) * f(x(s-1),y(:,s-1));
```

por algo un poco más complicado (el paso de Heun de "predecir y corregir"). —

Ejercicio 39: Utilizando la función `heunvector` recién definida, dibújense las soluciones aproximadas de la ecuación de Lotka-Volterra del Ejemplo 18. Dibújese también la solución dada por el método de Euler y compárense. ¿Cuál parece más razonable y por qué? ¿Cuáles son las diferencias en valor absoluto y relativo?

Verifíquese (de alguna manera razonable) que los máximos y mínimos de cada variable (los puntos críticos) corresponden con los cocientes de los parámetros de la otra. ¿Cómo puede hacerse esto?

Debería ser claro que las soluciones calculadas con el método de Heun tienen un aspecto más periódico que las calculadas con el método de Euler: esto refleja el hecho de que el método de Heun es mejor que el de Euler para aproximar soluciones de ecuaciones de Lotka-Volterra. —

2. Modelos de epidemias (SIR)

Ejercicio 40: El modelo epidémico SIR describe la evolución de una enfermedad infecciosa mediante la siguiente ecuación diferencial:

$$\begin{aligned}\dot{S}(t) &= -\alpha S(t)I(t)\\ \dot{I}(t) &= -\beta I(t) + \alpha S(t)I(t)\\ \dot{R}(t) &= \beta I(t)\end{aligned}$$

donde α y β son dos constantes. Las variables S, I y R indican, respectivamente, el número de individuos susceptibles S de contraer la enfermedad, el de infectados I y el de recuperados R (de "recovered", eliminados bien por inmunidad o bien por curación y subsiguiente inmunidad). Úsese la función de Matlab `heunvector` definida en el Ejercicio 38 para dibujar diversos modelos de esta ecuación, tomando valores diferentes de los parámetros α y β y/o valores diferentes de las condiciones iniciales. Un ejemplo interesante se da con $S(0) = 1$, $I(0) = 0.001$, $R(0) = 0$ y $\alpha = 0.1$, $\beta = 0.05$, para un intervalo de tiempo de 0 a 1500 usando 2000 puntos, por ejemplo. Compárese la evolución de este sistema con uno que tenga $\alpha = 0.05$ y $\beta = 0.01$. ¿Se comportan de manera distinta si $\alpha > \beta$ o si $\beta > \alpha$? ¿Cómo cambian?

Cómparense los resultados que se obtienen usando `heunvector` y `eulervector`. ¿Cuál parece más preciso? —

Ejercicio 41: Modifíquese el modelo del Ejercicio 40 para permitir que algunos de los infectados puedan convertirse en susceptibles (i.e. individuos que se curan pero pueden volver a contagiarse), mediante un coeficiente γ. Compárese la evolución de este sistema con la del anterior, utilizando `heunvector` en ambos casos. —

Ejercicio 42: En el modelo del Ejercicio 41, considérese un posible crecimiento de la población susceptible (digamos, por nacimiento), que siga una ley proporcional (es decir, $S(t)$ crece proporcionalmente a la suma de $S(t), I(t)$ y $R(t)$ con un pequeño coeficiente δ). Compárese este modelo con los de los Ejercicios 41 y 40. Explíquense las gráficas que se obtienen y la (notable) diferencia que se ve en $R(t)$ entre los dos ejercicios previos y en este. —

Ejercicio 43: Además del nacimiento, considérese la posibilidad de muerte en el Ejercicio 42: cada uno de los grupos $S(t), R(t)$ e $I(t)$ decrecen con diferentes probabilidades ϵ_1, ϵ_2 y ϵ_3, suponiendo que

$\epsilon_1 = \epsilon_3$ y que $\epsilon_2 > \epsilon_1$. ¿Cambia mucho el modelo? Compárese con los anteriores. —

3. Sistemas Físicos

Ejemplo 20 Se puede simular un péndulo simple sin rozamiento utilizando la siguiente ecuación (en coordenadas polares) con centro el extremo del hilo:

$$l\ddot{\theta} = -g \operatorname{sen}(\theta)$$

donde θ es el ángulo con respecto a la vertical y l es la longitud (constante) del péndulo. Como no hay rozamiento, la masa del péndulo es irrelevante. Usaremos `heunvector` para calcular una aproximación a la solución de esta ecuación, usando por ejemplo $l = 1$ y $g = 9.81$ y dos condiciones iniciales distintas: $\theta(0) = \pi/4$ y $\theta(0) = \pi/3$, ambas con velocidad inicial 0. Dejaremos que el tiempo vaya de 0 a 50 en pasos de 0.01. El código del Listado 5.4 muestra cómo hacerlo.

```
% Simulacion de un pendulo, para dos condiciones iniciales diferentes.
% La masa es irrelevante en este problema.

% La ecuacion del pendulo es theta'' = -sen(theta)
% que se escribe con DOS variables:
% theta=X(1)
% theta'=X(2)
P = @(t, X) [X(2) ; -sin(X(1))];

% Tiempo
T=[0:.01:50];
% Condicion inicial pi/4
X0=[pi/4; 0];

% Resolvemos con Heun
Y = heunvector(P, T, X0);
% Dibujamos (el angulo y la velocidad, las dos cosas)
plot(T, Y)
hold on

% Condicion inicial pi/6
X1=[pi/6;0];

% Resolvemos y dibujamos como antes
Y2 = heunvector(P, T, X1);
plot(T, Y2)
```

LISTADO 5.4. Simulación de un péndulo utilizando el método de Heun.

Ejercicio 44: Calcúlense soluciones aproximadas a los mismos sistemas que en el Ejemplo 20 pero utilizando `eulervector`; compárense con las obtenidas en dicho ejemplo. ¿Cuáles parecen mejores y por qué?

Ejercicio 45: Considérese el péndulo del Ejemplo 20 pero con rozamiento. Supóngase que la fuerza de rozamiento es proporcional a la velocidad angular (y, obviamente, de signo contrario), con constante de proporcionalidad 0.1. La ecuación debe escribirse ahora teniendo en cuenta la masa del sistema (póngase $m = 0.5$, por ejemplo). Calcúlese la solución aproximada (utilizando `heunvector`) y compárese con la del ejercicio 20. El efecto que se observa se denomina "amortiguamiento exponencial". Téngase en cuenta que (si se hace un dibujo) la ecuacióon del movimiento queda, en las polares del Ejemplo 20:

$$ml\ddot{\theta} = -g(\theta) - k\dot{\theta}.$$

Ejercicio 46: El movimiento armónico simple viene descrito por la EDO de segundo orden

$$\ddot{x} = -\frac{k}{m}x$$

para cierta constante de elasticidad k y una masa m del cuerpo en movimiento. Se dice que está *amortiguado* si hay una fuerza de amortiguamiento que va contra el movimiento, que da lugar a la ecuación

$$\ddot{x} = -\frac{k}{m}x - r\dot{x}$$

para cierta constante $r > 0$. Estúdiese el comportamiento de un oscilador armónico con y sin rozamiento utilizando tanto *eulervector* como *heunvector* y compárense los resultados. ¿Qué ocurre si $r < 0$?

Ejercicio 47: La ecuación balística describe el movimiento de un cuerpo en libertad bajo la acción de la gravedad. Si $(x(t), y(t))$ es el vector de posición, la ecuación es:

$$\ddot{x} = 0$$
$$\ddot{y} = -g.$$

Dadas una posición inicial $(0, 0)$ y una velocidad inicial $(1, 1)$, dibújese la gráfica $(x(t), y(t))$ de posiciones del cuerpo en movimiento para t de 0 a 20 en pasos de 0.01. Úsese `heunvector`.

¿Cuál es la ecuación diferencial si hay fricción y es proporcional a la velocidad (obviamente, en sentido contrario)? Plantéese y dibújese la trayectoria correspondiente para las mismas condiciones iniciales de antes.

Finalmente, hágase el mismo cálculo con el rozamiento proporcional al *cuadrado* de la velocidad. —

CAPÍTULO 6

Interpolación por mínimos cuadrados

Dada una nube C de N puntos y un *espacio vectorial* V de funciones que "representan adecuadamente la nube de puntos", el problema de encontrar la *mejor* aproximación a la nube de puntos dentro de las funciones de V se puede comprender de diversos modos. El más común es la *aproximación por mínimos cuadrados*, que se explicó en la teoría y que puede enunciarse así:

Sean $(x_1, y_1), \ldots, (x_N, y_N)$ los puntos de la nube C y sea $\{f_1, \ldots, f_n\}$ una base de V. El *problema de interpolación por mínimos cuadrados* para C y V consiste en encontrar los coeficientes $a_1, \ldots, a_n$ tales que el valor

$$E(a_1, \ldots, a_n) = \sum_{i=1}^{N} (a_1 f_1(x_i) + \cdots + a_n f_n(x_i) - y_i)^2$$

es mínimo (ese valor se llama *error cuadrático total* y, obviamente, depende de los coeficientes).

Hay diversas maneras de resolverlo. Quizás la más sencilla sea utilizar cálculo diferencial en varias variables. Al final, los coeficientes $a_1, \ldots, a_n$ se calculan como la solución del sistema de ecuaciones lineales:

$$XX^t \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} = X \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \tag{3}$$

(nótese que ambos miembros de la igualdad terminan siendo vectores columna de n componentes). La matriz X es

$$X = \begin{pmatrix} f_1(x_1) & f_1(x_2) & \ldots & f_1(x_N) \\ \vdots & \vdots & \ddots & \vdots \\ f_n(x_1) & f_2(x_2) & \ldots & f_n(x_N) \end{pmatrix}$$

y X^t es su traspuesta. La ecuación (3) siempre (en condiciones muy generales) da un sistema compatible, pero habitualmente está mal condicionado.

Así pues, el problema de interpolación por mínimos cuadrados parte de un conjunto de N puntos (la nube) y una familia de n funciones (con $n < N$, de hecho N suele ser muy grande y n muy pequeño).

Ejemplo 21 El ejemplo más sencillo es la interpolación de una nube de puntos mediante una función lineal (la llamada *recta de regresión*). Las funciones lineales tienen la forma $a + bx$, así que el espacio vectorial que se estudia está generado por $f_1(x) = 1$ y $f_2(x) = x$ (y por tanto hay dos valores que calcular, la a y la b). Tómese por ejemplo la nube dada por $(1, 2), (2, 2.1), (3, 2.15), (4, 2.41), (5, 2.6)$. Es fácil darse cuenta de que la recta de regresión será más o menos $y = 0.1(x - 1) + 2 = 1.9 + 0.1x$ (puesto que la pendiente es más o menos 0.1 y la línea pasará más o menos poor $(1, 2)$). Usando Matlab para resolver este problema:

```
> x=[1 2 3 4 5];
> y=[2 2.1 2.15 2.41 2.6];
> f1=@(x) 1 + x.*0;
> f2=@(x) x;
> X=[f1(x); f2(x)]
X =

1 1 1 1 1 1 2 3 4 5

> A=X*X'
A =

5 15 15 55

> Y=X*y'
Y =

11.260 35.290

> coefs=A\Y
coefs =

1.79900 0.15100
```

Por tanto, la función lineal (recta) que interpola la nube de puntos por mínimos cuadrados es $y = 1.799 + 0.151x$, que está de acuerdo con nuestra suposición. —

Nota 4 Hay que tener cuidado con la definición de la función constante $f_1(x) = 1$, en Matlab: hay que hacerla "vectorial" añadiéndole,

X	Y
1.0	6.20
1.5	9.99
2.0	15.01
2.5	23.23
3.0	32.70
3.5	43.08
4.0	54.01
4.5	65.96
5.0	80.90

TABLA 1. Datos que siguen una fórmula cuadrática.

por ejemplo, un vector nulo (en este caso, $0. * x$). Esto es un apaño para un defecto de Matlab, no algo mágico. —

Ejemplo 22 Los datos de la Tabla 1 vienen de un experimento. Se sabe que la variable Y depende cuadráticamente de la variable X (es decir, hay una fórmula de grado dos que relaciona Y con X). Usando mínimos cuadrados, encuéntrense los coeficientes que mejor ajustan esta nube.

Como se sabe que Y depende de X como un polinomio de grado 2, el espacio vectorial V de funciones está generado por $1, x, x^2$. Sean $f_1 = 1, f_2 = x, f_3 = x^2$. Se puede usar Matlab como sigue para resolver el problema:

```
> x=[1:.5:5];
> y=[6.2 9.99 15.01 23.23 32.7 43.08 54.01 65.96
80.9];
> % definimos las funciones:
> f1=@(x) 1+0.*x;
> f2=@(x) x;
> f3=@(x) x.^2;
> % main matrix
> X=[f1(x); f2(x); f3(x)];
> A=X*X';
> Y=X*y';
> % the system is A*a'=Y, use matlab to solve it:
> A\Y ans =

0.55352 2.27269 2.75766
> % this means that the least squares interpolating
> % polynomial of degree 2 is
> % 0.55352 + 2.27269*x + 2.75766*x.^2
> % plot both the cloud of points and the polynomial
```

X	Y
2.0	11.39
2.7	15.31
3.4	18.18
4.1	19.80
4.8	19.76
5.5	15.03
6.2	1.74
6.9	-29.67

TABLA 2. Datos que siguen una fórmula compleja.

```
> plot(x,y,'*');
> hold on
> u=[1:.01:5];
> plot(u, 0.55352 + 2.27269.*u + 2.75766*u.^2, 'r')
```

Obsérvese cómo el polinomio interpolador por mínimos cuadrados *no pasa por todos los puntos de la nube* (de hecho, es muy posible que no pase por ninguno). —

Nota 5 El problema de interpolación por mínimos cuadrados no tiene por qué ser solo de encontrar polinomios. Puede haber funciones más complejas que aproximen la nube de puntos, pero siempre habrá de tratarse de un problema *lineal*: si no, la técnica expuesta no sirve. —

Ejercicio 48: Un nuevo desarrollo teórico muestra que los datos de la Tabla 1 se describen mejor mediante una función de grado 3. Úsese la interpolación por mínimos cuadrados para calcular el polinomio cúbico que mejor se ajusta a la nube. ¿Se parecen los coeficientes de grados $0, 1$ y 2 de este polinomio a los del Ejemplo 22? ¿Es el ajuste mejor o peor?

Este ejercicio es interesante porque muestra que, por lo general, usar un polinomio para interpolar una nube de puntos, no es una buena idea, *salvo que haya una explicación clara* de por qué la nube está relacionada con un polinomio (como en los ejemplos y ejercicios que siguen). —

Ejercicio 49: La Tabla 2 representa datos de un experimento. Se sabe que siguen una función de la forma $y(x) = a\log(x) + bx + ce^x$, para ciertos a, b y c. Usando mínimos cuadrados, calcular estos a, b y c. —

X	Y	X	Y	X	Y	X	Y
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.10	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.10	4	5.39	19	12.50
12	10.84	12	9.13	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89

TABLA 3. Cuarteto de Anscombe.

v	E
1.0	8.05
1.5	16.97
2.3	39.69
2.7	55.58
3.0	66.91

TABLA 4. Energía cinética frente a velocidad para el mismo objeto.

Ejercicio 50: La Tabla 3 contiene lo que se conoce como el *Cuarteto de Anscombe*: es una lista de datos interesante por las propiedades que comparten las cuatro colecciones. Nosotros nos fijaremos solo en la recta de regresión $a + bx$ para cada nube de puntos. Calcúlese (hay cuatro nubes de puntos, por tanto, cuatro rectas de regresión). Una vez calculadas las cuatro rectas, dibújense las cuatro nubes de puntos y las rectas (todo en la misma figura) y compárense. ¿Qué se puede deducir de esto? —

Ejercicio 51: Un experimento sirve para calcular el valor de la energía cinética a partir de la velocidad de un objeto en movimiento. La Tabla 4 muestra los resultados de llevar a cabo dicho experimento 5 veces con el mismo objeto. Dése un valor razonable para la masa del objeto. La velocidad se da en m/s y la energía en julios. —

Ejercicio 52: La Tabla 5 lista los valores del cómputo de la distancia recorrida por un objeto en un tiempo determinado. Se sabe que dicho objeto sigue un movimiento uniformemente acelerado que comienza con la misma velocidad inicial. Calcúlense valores razonables para esta velocidad inicial y para la acelaración. —

t (s)	d (m)
1	4.89
2	11.36
3	21.64
4	34.10
5	50.05
6	69.51

TABLA 5. Distancia frente a tiempo de un m.u.a.