

# GRÁFICAS CON MATLAB

Roberto Rodríguez del Río  
Departamento de Matemática Aplicada  
Universidad Complutense de Madrid

## INTRODUCCIÓN

1. Manejo elemental de Matlab.
  - 1.1. Interfaz de usuario. Variables.
  - 1.2. Vectores y Matrices.
2. Gráficas 2D.
  - 2.1. Funciones de la forma  $y = f(x)$
  - 2.2. Curvas en paramétricas.
  - 2.3. Curvas en polares.
  - 2.4. Cambios de coordenadas polares-cartesianas.
3. Gráficas 3D.
  - 3.1. Curvas en el espacio.
  - 3.2. Funciones de la forma  $z = f(x, y)$
  - 3.3. Manipulación de Gráficos 3D.
  - 3.4. Algunas superficies en el espacio.
  - 3.5. Gráficos de funciones complejas.
4. Gráficos estadísticos.
  - 4.1. Diagramas de sectores.
  - 4.2. Diagramas de Pareto.
  - 4.3. Diagramas de barras.
  - 4.4. Histogramas.
5. Gráficas en movimiento: “movies”.

## REFERENCIAS

## INTRODUCCIÓN

El nombre MatLab es una abreviatura de las palabras MATrix LABoratory. MatLab es un sistema interactivo para cálculos científicos y de ingeniería basado en las matrices. Con él se pueden resolver complejos problemas numéricos sin necesidad de escribir un programa específico para ello, aunque también es posible programar. Además, el programa MATLAB dispone, dependiendo de la versión, de diferentes módulos (*Toolboxes*) que permiten resolver problemas específicos.

Nosotros nos vamos a centrar en la capacidad de MatLab para generar gráficos, aunque, antes de llegar hasta este punto, haremos un rápido resumen de los comandos básicos del programa.

Debido a que MatLab es un programa de Cálculo Numérico, la forma de producir gráficos es completamente distinta de la de programas de Cálculo Simbólico como Derive, Mathematica o Maple. En MatLab, nosotros tenemos que calcular mediante comandos adecuados los puntos que después se representarán en la gráfica.

### 1. MANEJO ELEMENTAL DE MATLAB

Supongamos que hemos sido capaces de abrir el programa. En Matlab, las órdenes se introducen escribiéndolas una a una a continuación del *prompt* (`>>`) que aparece en la ventana del usuario. Veamos en primer lugar, algunas de las operaciones matemáticas más elementales.

Para sumar dos números:

```
>>2+2
ans =
    4
```

Después de escribir cada comando hay que pulsar Intro para que lo ejecute. Si después de esta agotadora primera sesión con MatLab queremos salir del programa, se puede hacer de dos formas, escribiendo `exit` a continuación del *prompt*, o bien con `File Exit MATLAB`.

El valor que queremos calcular también se puede asignar a una variable. Por ejemplo:

```
x=3^2
x=
    9
```

Hay que tener en cuenta que MatLab distingue entre mayúsculas y minúsculas, por lo tanto, se distingue entre la variable **X** y la variable **x**.

La notación para las operaciones matemáticas elementales es la habitual en todos los programas de Cálculo Simbólico:

suma	+
resta	-
división	/
exponenciación	^
multiplicación	*

También están definidas algunas de las funciones más comunes utilizadas en Matemáticas. Su sintaxis coincide también con la que se utiliza en la mayoría de los programas de Matemáticas, como, por ejemplo, el programa DERIVE, aunque hay algunas diferencias. Algunas de estas funciones son:

<b>sin</b>	seno
<b>sinh</b>	seno hiperbólico
<b>asin</b>	arcoseno
<b>cos</b>	coseno
<b>cosh</b>	coseno hiperbólico
<b>acos</b>	arcocoseno
<b>tan</b>	tangente
<b>atan</b>	arcotangente
<b>exp</b>	exponencial
<b>log</b>	logaritmo neperiano
<b>log10</b>	logaritmo decimal
<b>sqrt</b>	raíz cuadrada
<b>abs</b>	valor absoluto

Para obtener listas completas de todas las funciones que puede utilizar Matlab, así como para saber el uso de cada una de ellas o de cualquier comando, siempre se puede acudir al **help**. Esto se puede hacer de varias formas, poniendo **>>helpwin**, siendo el propio programa quien nos ofrece la ayuda (como en cualquier otro programa), o poniendo **>>helpdesk**, con lo que nos ofrece ayuda interactiva, conectándose a Internet si este recurso está disponible en nuestro ordenador.

Si conocemos el nombre del comando, pero queremos saber para qué sirve, se puede poner:

```
>>help comando
```

Y nos ofrecerá ayuda sobre el comando en cuestión, si éste existe.  
Por ejemplo,



```
>>help rotate3d
```

```
ROTATE3D Interactively rotate the view of a 3-D plot.  
ROTATE3D ON turns on mouse-based 3-D rotation.  
ROTATE3D OFF turns if off.  
ROTATE3D by itself toggles the state.  
  
See also ZOOM.
```

Nos ofrece información sobre el comando `rotate3d`, comando que sirve para rotar figuras tridimensionales utilizando el ratón.

Otra forma de buscar ayuda es utilizar el comando `lookfor`, por ejemplo, poniendo `>>lookfor cos`, nos aparecerá una lista con todos los comandos que tienen que ver con la función coseno.

### 1.1. Interfaz de usuario. Variables

Con las flechas del cursor:  y , se pueden recuperar las órdenes anteriores, sin tener que volver a teclearlas. Esto resulta útil en el caso de una equivocación o cuando se quiere repetir un comando con alguna pequeña modificación.

A veces, puede resultar necesario, hasta imprescindible, que el resultado de un cálculo no aparezca en pantalla. Por ejemplo, si generamos una matriz de orden muy alto con el objeto de hacer después una gráfica. El hecho de que aparezca la matriz en pantalla puede resultar un poco engorroso. Para conseguir esto se pone un punto y coma al final de la instrucción.

Por ejemplo,  
`x=sin(3);1`

No aparece ningún resultado, pero ha realizado el cálculo, porque si escribimos el valor de `x`, aparecerá el valor `0.1411`.

---

<sup>1</sup>El argumento de las funciones trigonométricas siempre se mide en radianes.

Los comandos se pueden ir escribiendo y ejecutando uno a uno, es decir, renglón a renglón, y también se pueden escribir uno a continuación de otro en una misma línea, en cuyo caso deben ir separados por comas. Si el comando o la cantidad de comandos es demasiado larga para que aparezca en un único renglón, se puede romper la cadena y seguir en el siguiente renglón, escribiendo tres puntos suspensivos. Por ejemplo,

```
>>x=sin(10),y=cos(10),...  
z=tan(10)
```

```
x =  
    -0.5440
```

```
y =  
    -0.8391
```

```
z =  
    0.6484
```

Los resultados numéricos que ofrece MatLab se pueden visualizar en diferentes formatos. Por defecto, si un resultado es un número entero, lo ofrecerá como tal. Si no lo es, lo hará con 4 cifras decimales (redondeando a la cuarta cifra). Si el resultado es un número grande, lo expresará en notación científica. Este formato que usa por defecto se puede modificar en el menú **File Preferences Numeric Format**, aunque también se puede hacer directamente escribiendo las órdenes a continuación de `>>`:

Si escribimos:

```
>>x=34/8449;
```

y vamos cambiando el formato como se indica en la siguiente tabla, volviendo a escribir `>>x`, cada vez se obtiene el mismo resultado en las distintas formas numéricas.

Formato	Variable x	Características
<code>format long</code>	0.00402414486922	16 dígitos
<code>format short e</code>	4.0241e-003	5 dígitos más exponente
<code>format long e</code>	4.024144869215292e-003	16 dígitos más exponente
<code>format hex</code>	3f707b9f29b8eae2	sistema hexadecimal
<code>format bank</code>	0.00	2 decimales
<code>format +</code>	+	signo +, - ó 0
<code>format rat</code>	2/497	aproximación racional
<code>format short</code>	0.0040	formato por defecto

No obstante, independientemente del formato que se esté utilizando, la representación interna del número siempre es la misma, lo único que cambia es la forma en que lo vemos en la pantalla.

En Matlab, lo normal es ir asignando valores escalares o matriciales a variables, si en un momento determinado queremos saber con qué variables estamos trabajando, se puede escribir `>>who`, que nos indica qué variables están en uso; el comando `>>whos`, nos indica lo mismo, pero además nos informa del tamaño y del tipo de variable. O bien, en el ítem **File** con **Show Workspace**, que produce el mismo resultado que `>>whos`. Para borrar una variable, se puede utilizar el comando `>>clear variable`, y borrará la variable que se indique, si se pone sólo `>>clear`, se borrarán todas las variables que se estén utilizando actualmente.

Las variables pueden contener hasta 19 caracteres, los caracteres más allá del 19 se ignoran. Las variables deben comenzar con una letra, seguida por letras, dígitos o guiones de subrayado.

Además hay algunas variables especiales que se utilizan por defecto:

- **ans**: Es la variable que se utiliza en los resultados. En la operación siguiente se puede recuperar este resultado volviendo a escribir **ans**. Esta variable se modificarán en cuanto haya un nuevo resultado.
- **pi**: El número  $\pi$ . (No hay una variable para el número  $e$ , pero se podría definir `>>e=exp(1)`).
- **eps**: Es el número más pequeño que utiliza el ordenador tal que, cuando se le suma 1, crea un número en coma flotante mayor que 1.
- **Inf**: Infinito, aparece si hacemos `1/0`.
- **NaN**: Mensaje de error (Not a Number), por ejemplo, `0/0`.

- **realmin**, **realmax**: Son, respectivamente, el menor y el mayor de los números reales utilizables.

Poniendo el símbolo `%` se consigue que no se ejecute lo que venga a continuación, en el mismo renglón, sino que se interprete como un comentario, se suele utilizar para escribir comentarios aclaratorios en líneas de comandos de manera que no afecten a su ejecución. Por ejemplo, si ponemos,

```
>>sqrt(2)% Raíz cuadrada de 2
calculará la raíz de 2 y se saltará el comentario.
```

Una buena forma de acabar la lectura de esta primera introducción sería la de echar un vistazo a la **demo** que viene incorporada con el programa. Para activarla basta con teclear `>>demo`, aparecerá una ventana en la que se pueden ir viendo algunas de las capacidades del programa.

## 1.2. Vectores y matrices

Los vectores y las matrices son los elementos básicos con los que trabaja Matlab. Veamos cómo se introducen y cómo se pueden hacer algunas de las operaciones elementales con ellos.

**VECTORES.** Un vector se puede definir introduciendo sus coordenadas, separadas por espacios o por comas, entre corchetes:

```
>> x=[1 2 3]
```

```
x =
    1    2    3
```

Si queremos definir un vector columna, se separan las filas por puntos y comas, o bien se calcula el transpuesto de un vector fila con `>>x'`.

Otra forma de crear vectores es la siguiente:

```
>> x=1:0.5:3
```

```
x =
    1.0000    1.5000    2.0000    2.5000    3.0000
```

que genera un vector que va desde 1 hasta 10 con un paso de 0.5 unidades.

Exactamente el mismo resultado lo conseguiríamos con el comando **linspace**

```
>>x=linspace(1,3,5)
```

que produce 5 números igualmente espaciados entre 1 y 3.

**PRODUCTO ESCALAR.** Consideremos los dos vectores siguientes:

```
>>a=[1 2 3];b=[2 -3 5];
```

Si los multiplicamos de la forma

```
c=a.*b
c =
     2     -6     15
```

obtenemos el producto de los elementos del primero y del segundo vector elemento a elemento. Para obtener el valor del producto escalar

```
>>sum(c)
ans =
    11
```

El producto de dos vectores o dos matrices elemento a elemento será muy importante cuando queramos representar gráficas de funciones.

**MATRICES.** Para introducir una matriz, se separa cada fila con un punto y coma

```
A=[3 2 1; 6 5 4; 9 8 7]
A =
     3     2     1
     6     5     4
     9     8     7
```

**Ejercicio 1.1.** Después de definida la matriz, probar los siguientes comandos e intentar descubrir para qué sirven:

- a) `>>A(2,3)` o por ejemplo `>>A(1,2)`
- b) `A(:,1)` y también `A(2,:)`
- c) `A^2` y `A.^2`. ¿En qué se diferencian estos dos comandos?

Veamos algunas operaciones elementales con matrices. Definimos dos matrices  $3 \times 3$

```
>>A=[1 1 2; 3 4 6; 2 1 0];B=[-1 2 0; 2 0 0; -2 3 4];
```



Para sumarlas

```
>>C=A+B
C =
     0     3     2
     5     4     6
     0     4     4
```

Para multiplicarlas

```
>>D=A*B
D =
    -3     8     8
    -7    24    24
     0     4     0
```

Para elevar una matriz a una potencia

```
>>A^3
ans =
    45    44    58
   162   157   204
    43    39    46
```

Para calcular su determinante

```
>>det(A)
ans =
    -4
```

Para calcular su inversa, si existe

```
>>inv(A)
ans =
    1.5000   -0.5000    0.5000
   -3.0000    1.0000     0
    1.2500   -0.2500   -0.2500
```

**MATRICES PREDEFINIDAS.** En MatLab hay varios comandos que sirven para definir con gran facilidad matrices de tipos particulares. Algunas de estas funciones son las siguientes:

- `eye(n)`, matriz unidad de tamaño  $(n \times n)$

- `zeros(m,n)`, matriz de ceros de tamaño  $(m \times n)$
- `zeros(n)`, lo mismo, pero de orden  $(n \times n)$
- `ones(n)`, matriz de unos  $(n \times n)$
- `ones(m,n)`, lo mismo, pero de orden  $(m \times n)$
- `linspace(x1,x2,n)`, genera un vector con  $n$  valores igualmente espaciados entre `x1` y `x2`
- `logspace(d1,d2,n)`, genera un vector con  $n$  valores espaciados logarítmicamente entre  $10^{d1}$  y  $10^{d2}$
- `rand(n)`, matriz de números aleatorios entre 0 y 1, distribuidos uniformemente  $(n \times n)$
- `rand(m,n)`, lo mismo, de tamaño  $m \times n$
- `randn(n)`, matriz de números aleatorios  $(n \times n)$ , distribuidos según la normal estandar,  $N(0, 1)$
- `magic(n)`, crea una matriz en forma de cuadrado mágico de tamaño  $n \times n$

## 2. GRÁFICAS 2D

### 2.1. Funciones de la forma $y = f(x)$

Para hacer gráficas de funciones de una variable con MatLab, primero tenemos que crear una tabla de valores de la variable para después dibujar la función. Por ejemplo, queremos dibujar la gráfica de la función  $y = \sin(x)$ :

Primero creamos una tabla de valores para  $x$

```
>>x=0:pi/100:2*pi;
```

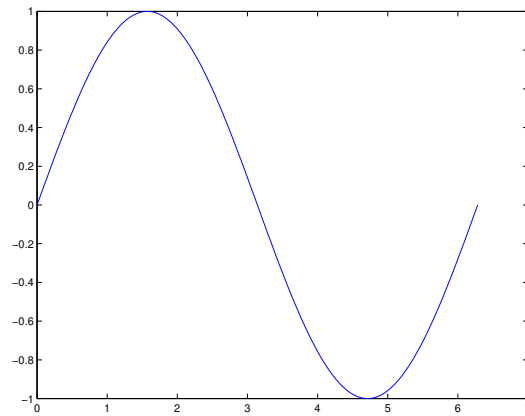
Con este comando hemos formado una tabla (el vector `x`) con 200 valores entre 0 y  $2 * \pi$ . Otra forma de conseguir el mismo resultado sería utilizar el comando

```
>>x=linspace(0,2*pi,200);
```

Ahora calculamos los valores de  $y$

```
>> y = sin(x);
```

y por último la dibujamos (ver figura 1)



**Figura 1.** Gráfica de  $y = \text{sen}(x)$ .

```
>>plot(x,y)
```

Realmente lo que hemos hecho es dibujar 200 puntos de la función en el intervalo  $[0, 2\pi]$ , y posteriormente el programa los ha unido mediante segmentos. Si el número de puntos es lo suficientemente grande, como en este caso, no se aprecian los vértices.

Veamos un ejemplo algo más complicado. Queremos dibujar ahora la gráfica de la función  $y = xe^{-x^2}$ .

Definimos los valores para los que queremos hacer la gráfica

```
>>x=-3:.01:3;
```

Es decir, que vamos a dibujar la gráfica en el intervalo  $[-3, 3]$  con un paso de longitud 0.01.

Definimos la función

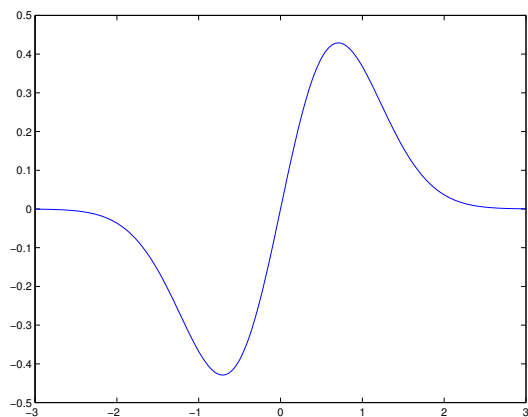
```
>>y=x.*exp(-x.^2);
```

(¿Por qué hay que poner los puntos antes de las operaciones?)

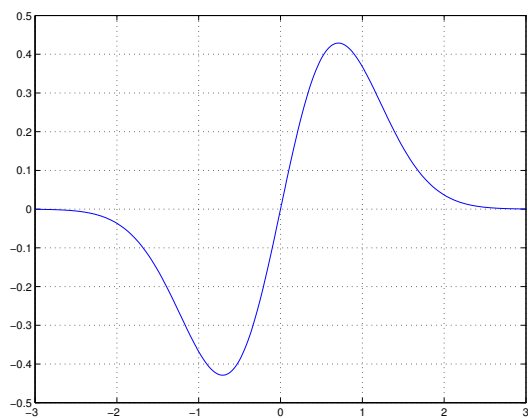
Y por último, se escribe el comando para que ejecute el dibujo (figura 2.)

```
>>plot(x,y)
```

El aspecto de la gráfica se puede modificar utilizando algunos comandos:



**Figura 2.** Gráfica de  $y = xe^{-x^2}$ .



**Figura 3.** Gráfica de  $y = xe^{-x^2}$  con cuadrícula.

- *Cuadrícula.* Si queremos que aparezca una cuadrícula sobre el dibujo, utilizaremos el comando `>>grid on`. El aspecto del dibujo sería ahora como el de la figura 3. Para desactivar la cuadrícula habrá que escribir `>>grid off`.

- *Color y trazo.* El comando `plot` ofrece múltiples posibilidades de color y forma de trazo de la gráfica. Por ejemplo, el comando `>>plot(x,y,'r*')`, nos dibujaría la gráfica en color rojo y con asteriscos. Para consultar todas las posibilidades, hacer `>>help plot`.

- *Ejes.* Los ejes que aparecen por defecto en una gráfica también se pueden modificar. Con el comando `>>axis([-2 2 -1 1])`, conseguiremos que la gráfica aparezca en la región  $-2 \leq x \leq 2$ ,  $-1 \leq y \leq 1$ . Con `>>axis square`, conseguiremos que la figura aparezca en un cua-

drado, sin cambiar el rango de los ejes. Con el comando `>>axis equal`, conseguiremos que los rangos de los ejes sean iguales.

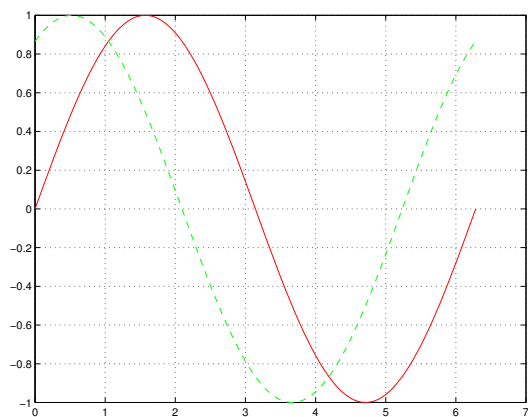
- *Zoom.* Utilizando el comando `>>zoom on`. Se puede agrandar la figura o alguna zona seleccionada de la figura. Hay que abrir la figura y utilizar los botones izquierdo y derecho del ratón. Para desactivarlo, habrá que escribir `>>zoom off`.

- *Varias gráficas en la misma figura.* Se pueden dibujar tantas gráficas como se quieran en una misma figura. Si ya tenemos dibujada una, y generamos una nueva gráfica, en principio la figura anterior es sustituida por la nueva. Sin embargo, utilizando el comando `>>hold on`, se mantendrá la anterior, con todas sus propiedades, y se podrá dibujar encima una nueva. Para desactivar el comando anterior: `>>hold off`. Otra forma de hacerlo es dibujar desde el principio dos gráficas juntas, por ejemplo, vamos a dibujar las gráficas de las funciones  $y = \sin(x)$  e  $y = \sin(x + \frac{\pi}{3})$  en la misma figura (4):

Generamos las tablas,

```
>>x=linspace(0,2*pi,300);  
>>y=sin(x);  
>>z=sin(x+pi/3);
```

Y ahora las dibujamos



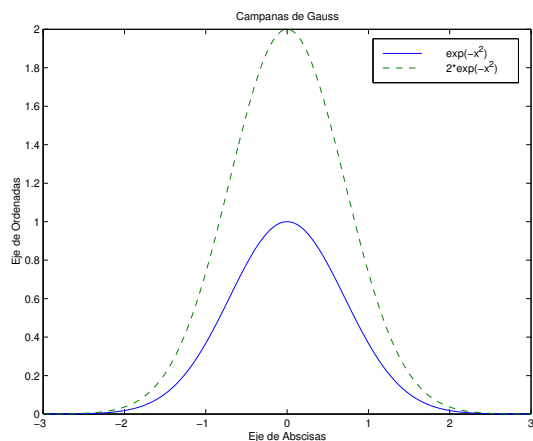
**Figura 4.** Gráficas de  $y = \sin(x)$  y de  $y = \sin(x + \frac{\pi}{3})$ .

```
>>plot(x,y,'r-',x,z,'g--'),grid on
```

(La primera en color rojo, con trazo continuo, y la segunda en verde, con trazo discontinuo).

- *Etiquetado de gráficas.* Existen diversas posibilidades para el etiquetado de las gráficas. Veámoslo con un ejemplo (ver figura 5):

```
>>x=linspace(-3,3,500);y=exp(-x.^2);z=2*exp(-x.^2);  
>>plot(x,y,'-',x,z,'--') % dibujamos dos funciones  
>>title('Campanas de Gauss')  
>>xlabel('Eje de Abscisas') % Etiqueta el eje horizontal  
>>ylabel('Eje de Ordenadas') % Etiqueta el eje vertical  
>>legend('exp(-x^2)', '2*exp(-x^2)') % Pone una leyenda
```



**Figura 5.** Etiquetado de gráficas.

Además de los comandos descritos antes para etiquetar gráficas, existe la posibilidad de poner un texto en algún otro lugar de la figura. Con el comando `>>gtext('texto')`, se abrirá la figura y podremos indicar con el ratón el lugar donde ha de ir el texto, que seleccionaremos con un clic.

- *Obtención de puntos desde el gráfico.* Una vez que se ha realizado una gráfica, podemos necesitar conocer las coordenadas de algunos puntos de la misma. Por ejemplo, el lugar aproximado en el que están los máximos y mínimos, o si queremos añadir alguna recta o una poligonal al dibujo. Para conseguir esto, se puede utilizar el comando `ginput`. Escribiendo

```
>>[x,y]=ginput(N)
```

Donde N es el número de puntos cuyas coordenadas queremos obtener. Después de ejecutado el comando habrá que pulsar con el botón izquierdo del ratón sobre el dibujo tantas veces como puntos hayamos especificado. Las coordenadas de esos puntos quedarán almacenadas en las variables [x, y].

Para dibujar gráficas de **funciones definidas a trozos**, necesitamos utilizar lo que vamos a denominar *índices o variables lógicas*. Veamos un ejemplo. Creamos un vector con los números del 1 al 7

```
>>x=1:7
x =
     1     2     3     4     5     6     7
```

Y ahora escribimos

```
>>x>4
ans =
     0     0     0     0     1     1     1
```

Observamos que donde no se cumple la condición, aparece 0 y donde se cumple, aparece 1. Para crear estas variables lógicas se pueden utilizar los siguientes *operadores relacionales*:

<	menor que
>	mayor que
<=	menor o igual
>=	mayor o igual
==	igual
~=	distinto

Estos operadores se pueden combinar utilizando los *operadores lógicos*:

&	y
	o
~	no

Así, por ejemplo, sobre el mismo x de antes, si escribimos

```
>>(2<x)&(x<=6)
ans =
     0     0     1     1     1     1     0
```

obtenemos unos en los valores que verifican  $2 < x \leq 6$ .

Ahora supongamos que queremos representar la función

$$f(x) = \begin{cases} x^2 & \text{si } x < 0 \\ 1 & \text{si } 0 \leq x < 1 \\ -x + 2 & \text{si } 1 \leq x \end{cases}$$

Generamos una tabla de valores en el dominio en el que queramos dibujar la función

```
>>x=linspace(-2,3,3000);
```

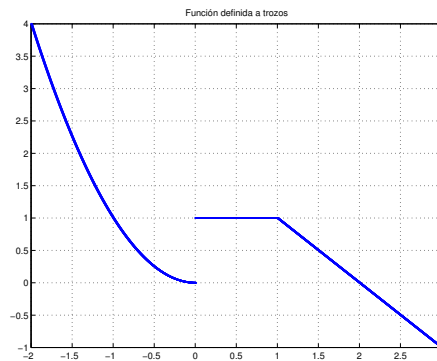
Y ahora definimos la función, multiplicando cada trozo por el índice lógico que describa el lugar en el que queremos dibujarlo,

```
>>y=(x.^2).*(x<0)+1.*((0<=x)&(x<1))+(-x+2).*(1<=x);
```

Y ahora la dibujamos. Resulta conveniente hacerlo con puntos, asteriscos o cruces porque, de otra forma, no aparecerán las discontinuidades

```
>>plot(x,y,'.'),grid on,title('Función definida a trozos')
```

Y obtenemos la gráfica de la figura 6.



**Figura 6.** Una función definida a trozos.

**Ejercicio 2.1.** Dibujar las gráficas de las siguientes funciones eligiendo, en cada caso, una tabla de valores adecuada para que aparezcan los aspectos más representativos de la función:



$$a) f(x) = x(x^2 + 4)^2$$

$$b) f(x) = x - \sqrt{x}$$

$$c) f(x) = \frac{\log x}{x}$$

$$d) f(x) = \frac{x(x-2)}{(x+1)(x-2)}$$

$$e) f(x) = \text{sen} \left( \frac{1}{x} \right)$$

$$f) f(x) = \frac{x}{e^{|x-1|}}$$

$$g) f(x) = \begin{cases} x^2 & \text{si } x < 0 \\ -1 & \text{si } x \geq 0 \end{cases}$$

$$h) f(x) = \begin{cases} -x & \text{si } x < -1 \\ 1 & \text{si } 0 < x < 2 \\ -x^2 & \text{si } x > 2 \end{cases}$$

$$i) f(x) = \begin{cases} \sqrt{1-x} & \text{si } x < -1 \\ 1-x^2 & \text{si } -1 < x < 1 \\ \sqrt{x-1} & \text{si } x > 1 \end{cases}$$

## 2.2. Curvas en paramétricas

Veamos ahora cómo se pueden representar curvas en el plano dadas en forma paramétrica, es decir, de la forma

$$\vec{r}(t) = (x(t), y(t)) \quad t \in [a, b]$$

Empecemos con un ejemplo: queremos dibujar la gráfica de la curva

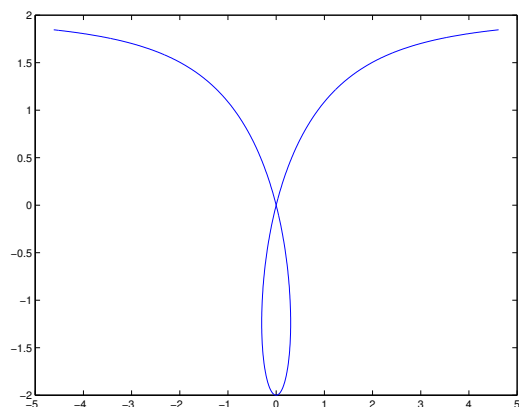
$$\vec{r}(t) = \left( \frac{t(t^2 - 1)}{t^2 + 1}, \frac{2(t^2 - 1)}{t^2 + 1} \right); \quad -5 \leq t \leq 5$$

En primer lugar generamos los valores de  $t$  en el intervalo indicado,

```
>>t=linspace(-5,5,1000);
```

Y ahora lo podemos dibujar de dos formas distintas:

```
>>plot((t.*(t.^2-1))./(t.^2+1),(2*(t.^2-1))./(t.^2+1))
```



**Figura 7.** Curva en paramétricas.

obtendremos la gráfica de la figura 7.

Y otra forma de hacerlo es utilizar el comando

```
>>comet((t.*(t.^2-1))./(t.^2+1),(2*(t.^2-1))./(t.^2+1))
```

Los dos comandos producen el mismo resultado, sin embargo, la forma de ejecución es diferente, la segunda es más divertida, aparece un circulito (el cometa) que va dibujando la curva. La velocidad de ejecución depende del número de puntos que hayamos generado con el comando `linspace`.

Dibujada una curva en paramétricas existe la posibilidad de dibujar sobre la misma los **vectores velocidad**, utilizando el comando `quiver`.

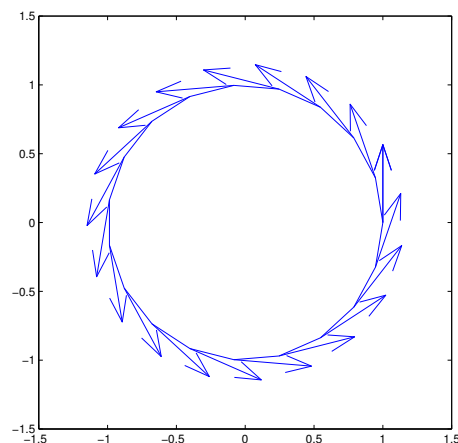
Por ejemplo, para dibujar los vectores velocidad sobre la curva

$$\vec{r}(t) = (\cos(t), \sin(t)), \quad t \in [0, 2\pi]$$

```
>>t=linspace(0,2*pi,20);
>>quiver(cos(t),sin(t),-sin(t),cos(t)),axis square
```

Produce la gráfica de la figura 8.

La sintaxis del comando es `>>quiver(r(t),r'(t))`. El número de vectores que aparecen en este caso es 20. Si el número de puntos que se indica con el comando `linspace` es demasiado grande, puede que no se aprecie con claridad la gráfica, ya que éste será el número de vectores que se dibujen.



**Figura 8.** Vectores velocidad sobre una circunferencia.

**Ejercicio 2.2.** Dibujar las curvas en paramétricas siguientes; en los apartados a) y b), dibujar además los vectores velocidad, utilizando el comando `quiver`:

$$a) \vec{r}(t) = (2 \cos^3 t, 2 \sin^3 t); \quad -\pi \leq t \leq \pi$$

$$b) \vec{r}(t) = (3 \sin t, 2 \sin(2t)); \quad -\pi \leq t \leq \pi$$

$$c) \vec{r}(t) = \left( \frac{t}{\pi} \left( 12 \left( \frac{t}{\pi} \right)^2 - 9 \right), \left( \left( \frac{t}{\pi} \right)^2 - 1 \right) 16 \left( \frac{t}{\pi} \right)^2 + 2 \right); \quad -3 \leq t \leq 3$$

$$d) \vec{r}(t) = \left( \frac{3}{2} \cos t (\cos t + 1), 2 \sin(2t) \right); \quad -\pi \leq t \leq \pi$$

$$e) \vec{r}(t) = (\sin(2t) + \sin t, -\cos(2t) - \cos t); \quad -\pi \leq t \leq \pi$$

$$f) \vec{r}(t) = \left( e^{\frac{t}{4}} \sin(2t), e^{\frac{t}{4}} \cos(2t) \right); \quad -\pi \leq t \leq \pi$$

$$g) \vec{r}(t) = \left( \frac{2}{3} t \cos\left(\frac{7t}{2}\right), \frac{2}{3} t \sin\left(\frac{7t}{2}\right) \right); \quad -\pi \leq t \leq \pi$$

$$h) \vec{r}(t) = \left( t - \frac{11}{10} \sin(3t), -\frac{22}{10} \cos(3t) \right); \quad -3\pi \leq t \leq 3\pi$$

### 2.3. Curvas en polares

Una curva en coordenadas polares es la imagen de la función

$$r = h(\theta), \quad \theta \in [\theta_1, \theta_2]$$

Un punto de la curva en polares  $(r_0, \theta_0)$  tiene distancia al origen  $r_0$  y el ángulo que forma el vector de posición del punto con el eje horizontal, medido en sentido positivo, es  $\theta_0$ .

Por lo tanto, la relación entre las coordenadas polares y las coordenadas paramétricas es

$$\begin{cases} x = r \cos(\theta) \\ y = r \sin(\theta) \end{cases}$$

Para dibujar una curva en polares con MatLab se utiliza el comando `polar`. Por ejemplo, para dibujar la gráfica de

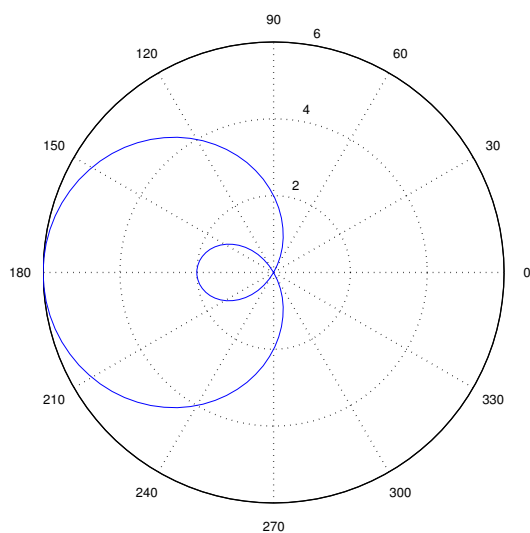
$$r = 2 - 4 \cos(\theta), \quad -\pi \leq \theta \leq \pi$$

Generamos los valores del ángulo `tetha`

```
>>tetha=linspace(-pi,pi,100);
```

Calculamos los valores de  $r$

```
>>r=2-4*cos(tetha);
```



**Figura 9.** Curva en polares.

Y dibujamos la gráfica

```
>>polar(tetha,r)
```

**Ejercicio 2.3.** Dibujar las gráficas de las siguientes funciones, dadas en coordenadas polares:

a)  $r = 7 - 7 \operatorname{sen}(\theta); \quad -\pi \leq \theta \leq \pi$

b)  $r = 3 - 6 \operatorname{sen}(\theta); \quad -\pi \leq \theta \leq \pi$

c)  $r = \operatorname{sen}(6\theta); \quad -\pi \leq \theta \leq \pi$

d)  $r = \cos(8\theta); \quad -\pi \leq \theta \leq \pi$

e)  $r = \sqrt{5 \cos(2\theta)}; \quad -\pi \leq \theta \leq \pi$

## 2.4. Cambios de coordenadas polares-cartesianas

Hay dos comandos que permiten hacer cambios de coordenadas. Si queremos cambiar de coordenadas polares a coordenadas cartesianas hay que utilizar el comando

```
>>[x,y]=pol2cart(theta,r);
```

Esto es, suponiendo que los puntos en coordenadas polares estén previamente almacenados en las variables `theta` y `r`. Los puntos ahora obtenidos se podrían dibujar utilizando el comando `plot`.

Para hacer el cambio de coordenadas cartesianas a coordenadas polares, habría que utilizar

```
>>[theta,r]=cart2pol(x,y);
```

**Ejercicio 2.4.** En los ejemplos del ejercicio anterior, utilizar el comando `pol2cart` para cambiar las coordenadas polares obtenidas a coordenadas cartesianas. Usar después el comando `plot` para obtener las gráficas en las nuevas coordenadas.

## 3. GRÁFICAS 3D

En esta sección vamos a ver cómo se pueden dibujar con MatLab gráficos de curvas en el espacio en forma paramétrica, gráficas de funciones de dos variables  $z = f(x, y)$ , y algunos ejemplos de superficies parametrizadas.

### 3.1. Curvas en el espacio

Se generan de una manera similar a las curvas en el plano, con la diferencia de que aquí se utilizan los comandos `plot3` o `comet3`, también existe un comando `quiver3` para dibujar vectores velocidad sobre las curvas.

Por ejemplo, queremos dibujar la hélice

$$\vec{r}(t) = (\sin(t), \cos(t), t) \quad 0 \leq t \leq 8\pi$$

y sobre ella los vectores velocidad.

Generamos los valores de  $t$ :

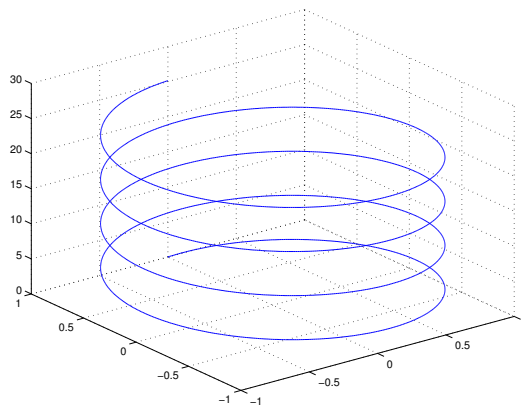
```
>>t=linspace(0,8*pi,2000);
```

Y ahora podemos utilizar dos comandos:

`plot3` lo que nos da el dibujo completo

```
>>plot3(sin(t),cos(t),t),grid on
```

con lo que obtendremos la gráfica de la figura 10.



**Figura 10.** Gráfica de una hélice.

O también `comet3`, que funciona de manera análoga a como lo hacía el comando `comet` en las curvas en el plano.

```
>>comet3(sin(t),cos(t),t)
```

Para dibujar algunos vectores velocidad sobre la curva hay que utilizar el comando `quiver3(vector posición,vector velocidad)`. Al

igual que con el comando `quiver`, también conviene volver a generar los valores de  $t$  de manera que no sean demasiados para que se pueda apreciar mejor la gráfica. Por ejemplo,

```
>>t=linspace(0,8*pi,30);
>>quiver3(sin(t),cos(t),t,cos(t),-sin(t),1)
```

**Ejercicio 3.1.** Representar las curvas siguientes y representar en gráfica aparte algunos vectores velocidad de la curva en los intervalos indicados:

$$a)\vec{r}(t) = (2\cos^3(t), 2\sin^3(t), t) \quad -4 \leq t \leq 3.$$

$$b)\vec{r}(t) = (\cos(t), 2\cos^2(t), \frac{1}{4}\sin t) \quad -\pi \leq t \leq \pi.$$

$$c)\vec{r}(t) = (\frac{t}{6}\cos t, \frac{t}{6}\sin t, \frac{t}{36}) \quad -12 \leq t \leq 19.$$

$$d)\vec{r}(t) = (e^{\frac{t}{4}}\sin(2t), e^{\frac{t}{4}}\cos(2t), \frac{t}{4}) \quad -10 \leq t \leq 4,8.$$

$$e)\vec{r}(t) = (\sin(2t) + \sin(t), -\cos(2t) - \cos(t), \frac{t}{6}) \quad -9 \leq t \leq 10.$$

$$f)\vec{r}(t) = (\cos(3t), 2\cos^2(t), \sin(2t)) \quad -\pi \leq t \leq \pi.$$

### 3.2. Funciones de la forma $z = f(x, y)$

Para dibujar gráficos de funciones de dos variables  $z = f(x, y)$ , al igual que para funciones de una variable, en primer lugar hay que generar tablas de valores para las variables  $x$  e  $y$ , en realidad, ahora lo que tenemos que hacer es generar un mallado sobre un rectángulo del plano  $XY$ . Para eso se utiliza el comando `meshgrid`.

Por ejemplo, si queremos dibujar la gráfica de la función

$$z = e^{-(x^2+y^2)}$$

en la región del plano  $D = \{(x, y) / -2 \leq x \leq 2, -2 \leq y \leq 2\}$ , habrá que efectuar los pasos siguientes:

Generamos el mallado

```
>>[x,y]=meshgrid(-2:.5:2);
```

Sustituimos en la función para calcular los valores de  $z$

```
>>z=exp(-x.^2-y.^2);
```

Y ahora podemos dibujar el gráfico con alguno de los siguientes comandos que producen los dibujos mostrados en la figura 11:

```
>>plot3(x,y,z)
```

```
>>mesh(x,y,z)
```

```
>>surf(x,y,z)
```

```
>>surf(x,y,z),shading flat %efecto de sombreado distinto
```

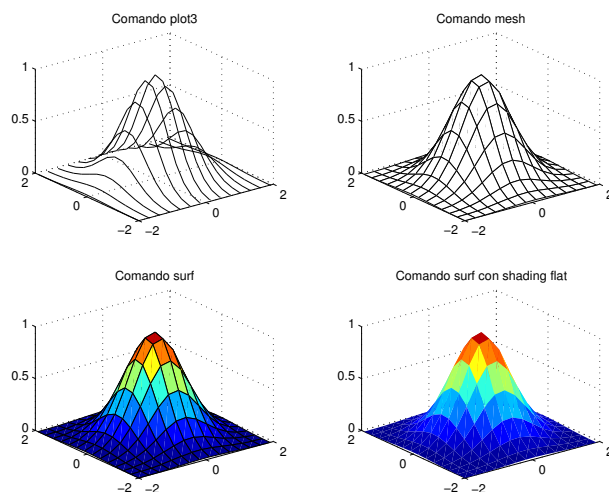


Figura 11. Gráficas 3D.

### 3.3. Manipulación de gráficos 3D

**MALLADO.** El comando `meshgrid` se puede utilizar también para generar mallados de regiones rectangulares. Por ejemplo, si queremos hacer un mallado para la región  $[0, 1] \times [0, 3]$ , tendremos que escribir

```
>>[x,y]=meshgrid(0:.1:1,0:.1:3);
```

La secuencia `0:.1:1` describe la variación de la variable  $x$ , y `0:.1:3` la de la variable  $y$ . Si sólo se utiliza un intervalo, éste se aplica a las dos variables. También se puede utilizar dentro de `meshgrid` el comando `linspace`.

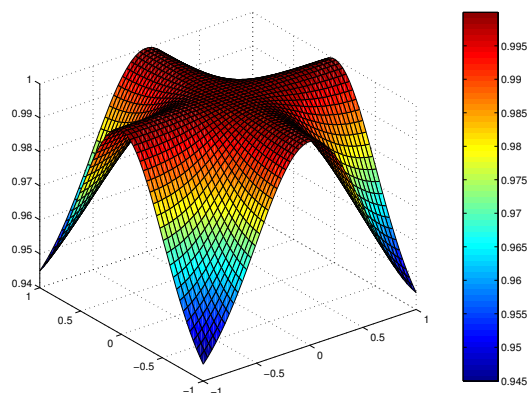
**SOMBRAS Y COLORES.** Para conseguir efectos de sombreados y colores diferentes se pueden consultar todas las posibilidades de los



comandos `colormap` y `shading`. Algo que resulta también interesante, es añadir una escala de colores al dibujo que nos permite conocer las alturas (coordenada  $z$ ) de los diferentes puntos de la gráfica, esto se consigue con el comando `colorbar` (después de dibujada la gráfica).

Para generar la gráfica de la figura 12 ha sido utilizada la siguiente secuencia de comandos:

```
>>[x,y]=meshgrid(linspace(-1,1,50));
>>z=cos((x.*y)./(x.^2+y.^2+1));
>>surf(x,y,z),colorbar
```



**Figura 12.** Gráfica 3D con escala de colores.

Como se puede observar, los puntos más altos corresponden a los colores más calientes y los puntos más bajos de la gráfica están coloreados con colores fríos.

**EJES.** Las longitudes de los ejes coordenados también se pueden modificar con el comando

```
>>axes([xmin xmax ymin ymax zmin zmax])
```

Los comandos `grid on` y `axis square` también funcionan en este tipo de gráficos.

**ROTACIÓN DE GRÁFICAS.** Otro comando interesante en las gráficas 3D es `rotate3d`, que nos permite, utilizando el ratón sobre la figura, rotarla de manera interactiva en tres dimensiones.

**CURVAS DE NIVEL.** Dada una función  $z = f(x, y)$ , las curvas sobre el plano  $XY$ , determinadas por  $f(x, y) = k$ , donde  $k$  es una constante se llaman *curvas de nivel*. Hay varias formas de obtenerlas usando MatLab.

Vamos a representar la gráfica de la función

$$z = x^2 + y^2,$$

dibujando algunas curvas de nivel.

Creamos el mallado,

```
>>[x,y]=meshgrid(-2:.1:2);
```

Sustituimos en la función, para calcular los valores de  $z$ ,

```
>>z=x.^2+y.^2;
```

Ahora, podemos dibujar la gráfica utilizando alguno de los comandos descritos anteriormente.

Las curvas de nivel se pueden hacer utilizando alguno de los comandos siguientes (ver figuras 13, 14 y 15):

```
>>contour(x,y,z,10) % dibuja 10 curvas de nivel
```

```
>>contour3(x,y,z,10) % lo mismo, pero en el espacio
```

```
>>pcolor(x,y,z),colorbar
```

Esta última orden dibuja un mapa de colores por niveles, la orden `colorbar` hace aparecer una escala de valores según el color, es decir, nos indica el valor de la variable  $z$ , como se describió antes.

Si se usa el comando `contour`, después se pueden etiquetar las curvas con los valores correspondientes de la  $z$ . Para hacer esto:

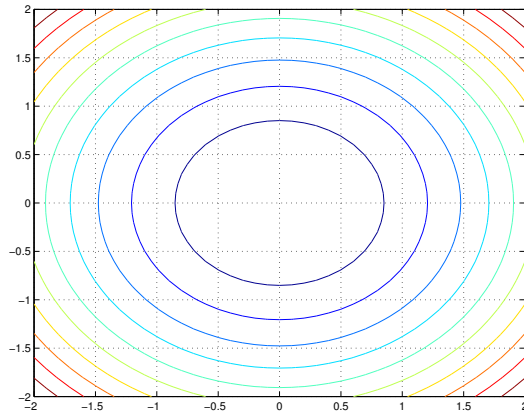
Primero dibujamos las curvas de nivel con

```
>>contour(x,y,z,10)
```

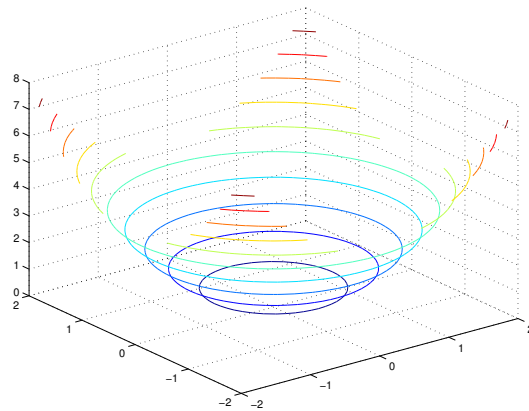
Después guardamos la información en una variable, por ejemplo,

```
>>cs=contour(x,y,z,30);
```

A continuación, tenemos dos opciones:



**Figura 13.** Curvas de nivel sobre el plano  $XY$ .



**Figura 14.** Curvas de nivel en el espacio.

```
>>clabel(cs) % etiqueta algunas aleatoriamente
```

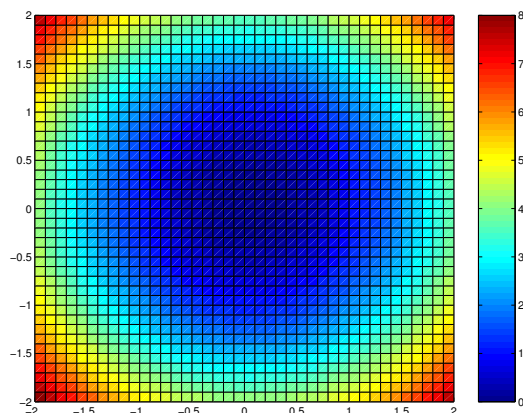
O bien

```
>>clabel(cs,'manual') % nos permite elegirlas con el ratón
```

Por otra parte, el comando `>>meshc(x,y,z)`, dibuja la gráfica, y por debajo, las curvas de nivel (algunas veces será necesario modificar los ejes para que la gráfica de la función no tape a las curvas de nivel).

**Ejercicio 3.2.** Representar las gráficas de las siguientes funciones de 2 variables, utilizando alguno de los comandos descritos anteriormente. Dibujar también algunas curvas de nivel:

$$a) z = \frac{1}{9 + x^2 + y^2}$$



**Figura 15.** Gráfica 3D con escala de colores.

$$b)z = -\sqrt{|xy|}$$

$$c)z = \frac{\cos(\frac{x^2+y^2}{4})}{3+x^2+y^2}$$

$$d)z = \frac{y^2}{5} - 3|x|$$

$$e)z = e^{-(x^2+y^2)}$$

### 3.4. Algunas superficies en el espacio

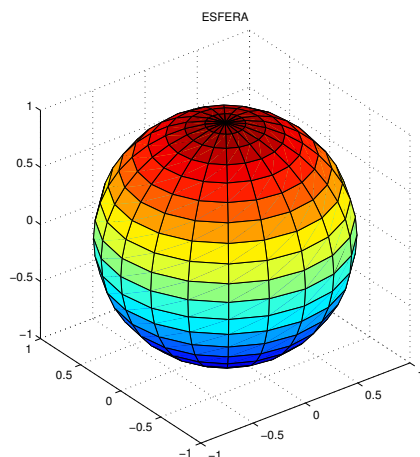
Hay varios comandos en MatLab que permiten generar las gráficas de superficies en  $\mathbb{R}^3$  (superficies que no son funciones.) Estos comandos son funciones que ya vienen programadas.

**ESFERA.** Se genera utilizando el comando `>>sphere(n)`, donde **n** es el número de puntos en los que queda dividido el ecuador de la esfera. Cuanto mayor sea **n**, mayor será la aproximación a la curvatura real de la esfera (de radio 1, centrada en el origen.) Poniendo sólo `>>sphere`, el valor que tomará **n** será 20, por defecto

```
>>sphere,axis square,title('ESFERA')
```

Obtenemos la gráfica de la figura 16.

**Ejercicio 3.3.** *Utilizando el comando `sphere`, dibujar varias esferas con diferentes valores de **n**. Probar, en particular, los valores 2, 3, 4, etc.*



**Figura 16.** Esfera de radio 1 centrada en el origen.

#### **Ejercicio 3.4. Vectores Normales a una superficie**

*Dibujar los vectores normales a la superficie de una esfera siguiendo los siguientes pasos:*

*Dibujar una esfera utilizando lo descrito anteriormente, pero guardando la información en tres variables*

```
>>[x,y,z]=sphere(n);
```

*Utilizar el comando >>surfnorm(x,y,z)*

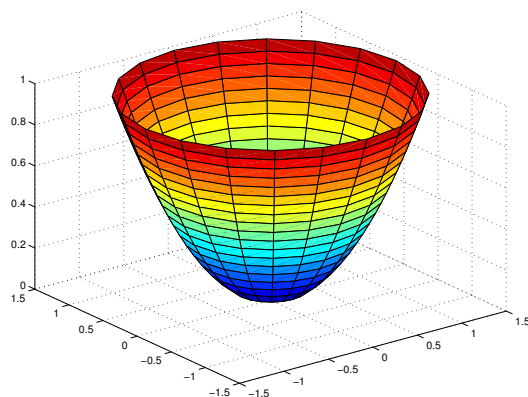
*Este comando también se puede utilizar para dibujar los vectores normales en superficies de funciones de la forma  $z = f(x,y)$ . Para dibujar las normales en el sentido opuesto habrá que poner `surfnorm(x',y',z')`.*

**CILINDRO.** El comando `>>cylinder(R,n)` genera automáticamente un cilindro de revolución de radio  $R$ , donde  $n$  es el número de puntos de la circunferencia de la base del cilindro. Como en el caso de la esfera, si usamos sólo `>>cylinder(R)`, el número  $n$  es, por defecto, 20.

Lo realmente interesante de este comando es que también admite radios variables  $R(t)$ , con  $t \in [a,b]$ . De esta forma, puede ser utilizado para obtener las gráficas de diferentes tipos de superficies de revolución, donde la generatriz es una función definida por  $R(t)$ . Por ejemplo, si queremos dibujar un paraboloide de revolución, podemos utilizar como generatriz la función  $r(t) = \sqrt{t}$ , con  $t \in [0,2]$

```
>>t=linspace(0,2,20);r=sqrt(t);cylinder(r)
```

Y obtendremos la gráfica de la figura 17. (No conviene poner demasiados puntos en `linspace` para que se pueda apreciar bien el dibujo.)



**Figura 17.** Paraboloide de revolución generado con `cylinder`.

**Ejercicio 3.5.** Dibujar las superficies generadas por `>>cylinder(R(t),30)`, en cada uno de los siguientes casos:

- a)  $R(t) = t, \quad t \in [-1, 1]$
- b)  $R(t) = t^2, \quad t \in [-1, 1]$
- c)  $R(t) = 2 + \sin(t), \quad t \in [-2\pi, 2\pi]$
- d)  $R(t) = e^t, \quad t \in [-3, 3]$

**MÁS SUPERFICIES DE REVOLUCIÓN.** El comando `>>makevase` hace aparecer una ventana interactiva que permite dibujar gráficas de superficies de revolución en las que la generatriz es una poligonal cuyos vértices se señalan con el ratón sobre el propio dibujo.

### 3.5. Gráficos de funciones complejas

El comando `cplxmap` permite representar gráficas de funciones complejas de variable compleja en el siguiente sentido:

Sea la función compleja de variable compleja

$$\begin{aligned} f: \mathbb{C} &\longrightarrow \mathbb{C} \\ z &\longmapsto w = f(z) \end{aligned}$$

El comando `>>cplxmap(z,f(z))` dibuja una gráfica tridimensional en la que el eje  $X$  es la parte real de la variable, es decir,  $\text{Real}(z)$ ; el eje

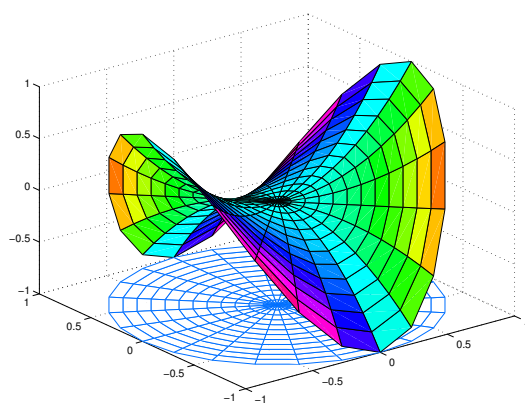
$Y$  es la parte imaginaria de la variable, es decir,  $\text{Im}(z)$  y el eje  $Z$  es la parte real de la imagen de la función, es decir,  $\text{Re}(f(z))$ .

La variable  $z$  va a pertenecer siempre al dominio constituido por el disco unidad centrado en el origen y las coordenadas de los puntos deben estar en forma polar. Esto se consigue utilizando previamente el comando `>>cplxgrid(n)`, donde  $n$  es el número entero positivo.

Por ejemplo, con los comandos

```
>>z=cplxgrid(12);
>>cplxmap(z,z.^2)
```

obtenemos la gráfica de la función  $f(z) = z^2$  (figura 18)

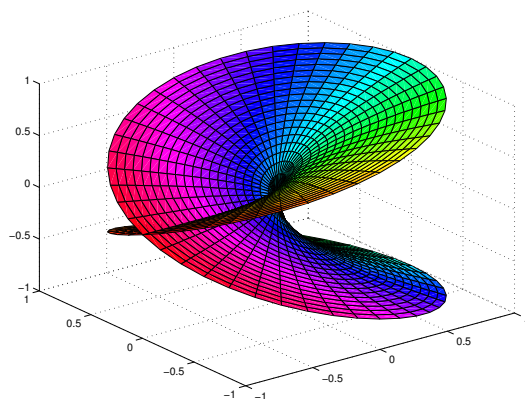


**Figura 18.** Gráfica de  $f(z) = z^2$ .

Obsérvese que para cada valor de  $z$ , su imagen  $f(z)$ , es única. Esto no es así para cualquier función compleja. Por ejemplo, la función  $f(z) = z^{1/2}$  es una función bivaluada, la función  $g(z) = z^{1/3}$  es una función trivaluada, cada  $z$  puede producir tres valores distintos para  $g(z)$ , y así sucesivamente. Para obtener las gráficas de estas funciones especiales, que se denominan *Superficies de Riemann*, MatLab dispone de un comando que las dibuja automáticamente, es el comando `cplxroot(n)`, donde  $n$  es el índice de la raíz.

El comando `>>cplxroot(2)` generaría la superficie de la figura 19.

Para obtener más información, se pueden ejecutar los comandos `cplxdemo` y `grafcplx`, que contienen sendas demostraciones de gráficas de funciones complejas.



**Figura 19.** Gráfica de  $f(z) = z^{1/2}$ .

## 4. GRÁFICOS ESTADÍSTICOS

A pesar de que no se puede decir que MatLab sea el programa ideal para hacer cálculos relacionados con la Estadística<sup>2</sup>, dispone de algunos comandos que nos permiten calcular algunos de los parámetros estadísticos básicos, así como comandos para generar bastantes gráficos.

Dependiendo del tipo de datos estadísticos de los que dispongamos, resulta conveniente utilizar uno u otro tipo de gráfico. Vamos a ir viendo los que se pueden hacer con MatLab, que son: *diagramas de Sectores*, *diagramas de Pareto*, *diagramas de barras e histogramas*.

### 4.1. Diagramas de sectores

Resultan útiles para representar datos de tipo cualitativo, en los que tenemos varias opciones, el diagrama de sectores permite compararlas en un círculo con sectores cuyo ángulo es directamente proporcional al porcentaje de cada opción.

**Ejemplo 4.1** *Los resultados de las elecciones generales del 12 de marzo de 2000 al Congreso de los Diputados fueron los siguientes:*

---

<sup>2</sup>No al menos como programas especializados en cálculos estadísticos, como puede ser el programa STATGRAPHICS.



Formación Política	Número de Escaños
<i>Partido Popular</i>	<i>183</i>
<i>Partido Socialista Obrero Español</i>	<i>124</i>
<i>Convergència i Unió</i>	<i>15</i>
<i>Izquierda Unida</i>	<i>8</i>
<i>Partido Nacionalista Vasco</i>	<i>7</i>
<i>Otros</i>	<i>12</i>
<i>Total</i>	<i>350</i>

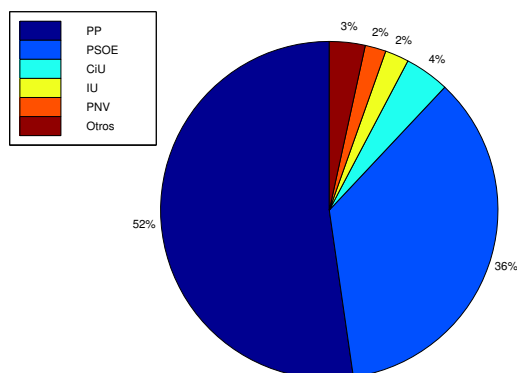
Para dibujar un diagrama de sectores de los resultados de las elecciones, procedemos como sigue. Introducimos los datos en un vector

```
>>x=[183 125 15 8 7 12]
x =
    183    125     15      8      7     12
```

Y ahora, dibujamos el diagrama. Se puede poner una leyenda que nos indique qué sector corresponde a cada partido político. Como se puede observar en el gráfico (figura 20), MatLab calcula automáticamente los porcentajes correspondientes y los pone junto a su sector

```
>>pie(x),legend('PP', 'PSOE', 'CiU', 'IU', 'PNV', 'Otros')
```

(Nota: si la leyenda no sale en el lugar deseado, se puede mover utilizando el botón izquierdo del ratón y colocándola en el lugar adecuado.)



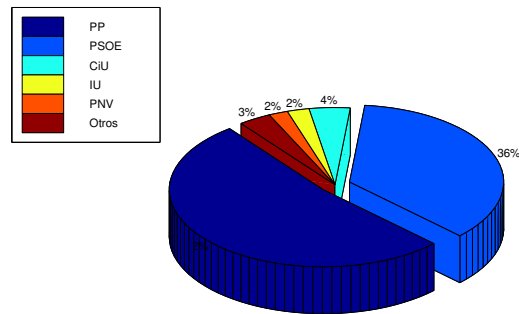
**Figura 20.** Diagrama de sectores.

Con el comando `pie3` se obtiene también un diagrama de sectores, pero en versión tridimensional (ver figura 21).

Tanto para el comando `pie`, como para el comando `pie3` existe la posibilidad de separar uno o más sectores para destacarlos con respecto de los demás. Por ejemplo, si queremos separar los sectores correspondientes a los dos primeros datos

```
>>pie3(x,[1 1 0 0 0 0])
```

El vector que se pone a continuación de `x` debe tener la misma longitud que el `x`, los unos y los ceros indican, respectivamente, los sectores que queremos separar y los que no.



**Figura 21.** Diagrama de sectores 3D.

#### 4.2. Diagramas de Pareto

Vamos a utilizar el ejemplo 4.1, pero ligeramente modificado:

Formación Política	Número de Escaños
<i>Partido Popular</i>	<i>183</i>
<i>Partido Socialista Obrero Español</i>	<i>124</i>
<i>Otros</i>	<i>42</i>
<i>Total</i>	<i>350</i>

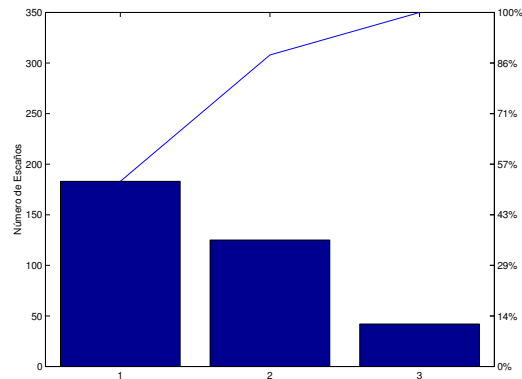
El *diagrama de Pareto* que produce MatLab constará de barras cuyas alturas son el número de escaños, ordenadas en forma decreciente y sobre las barras, un polígono con las frecuencias acumuladas de los escaños. Además, en el eje vertical derecho aparece una escala de porcentajes.

Para generarlo, escribimos

```
>>x=[183 125 42]
x =
    183    125     42
```

```
>>pareto(x),ylabel('Número de Escaños')
```

Y obtenemos el gráfico de la figura 22.



**Figura 22.** Diagrama de Pareto.

Este comando tiene un pequeño problema y es que si la frecuencia de uno de los datos es pequeña en comparación con las otras, puede no aparecer en el dibujo. Por ejemplo, si hubiésemos utilizado los datos tal y como aparecían en el ejemplo 4.1, algunas de las barras correspondientes a los partidos políticos que habían obtenido un número bajo de escaños no habrían aparecido.

### 4.3. Diagramas de barras

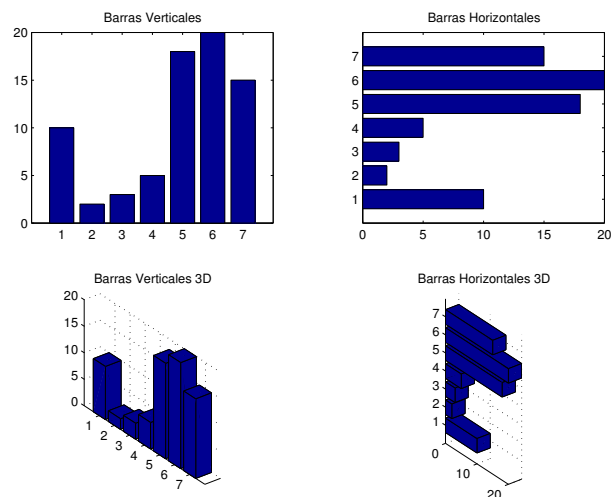
Existen varias posibilidades para representar diagramas de barras. Supongamos que queremos representar los siguientes datos en un diagrama de barras:

Introducimos los datos en un vector

```
>>x=[10 2 3 5 18 20 15 ];
```

Y ahora usamos los comandos `bar`, `barh`, `bar3` y `bar3h` para generar los gráficos. (Usando el comando `subplot` podemos conseguir que aparezcan todos en la misma figura.)

```
>>subplot(2,2,1),bar(x),title('Barras Verticales')
>>subplot(2,2,2),barh(x),title('Barras Horizontales')
>>subplot(2,2,3),bar3(x),title('Barras Verticales 3D')
>>subplot(2,2,4),bar3h(x),title('Barras Horizontales 3D')
```



**Figura 23.** Diagramas de barras.

Obtenemos los gráficos de la figura 23.

Hay que observar que las gráficas 3D se pueden modificar utilizando el comando `rotate3d` descrito en las secciones anteriores.

Los datos pueden estar agrupados, en este caso, las órdenes anteriores los dibujan también agrupados de manera que resulte fácil compararlos. Veamos el siguiente ejemplo:

```
>>x=[1 2 3;4 3 6; 10 9 8; 4 2 7;12 10 7];
```

Ahora, utilizando los mismos comandos que antes, obtenemos los gráficos de la figura 24.

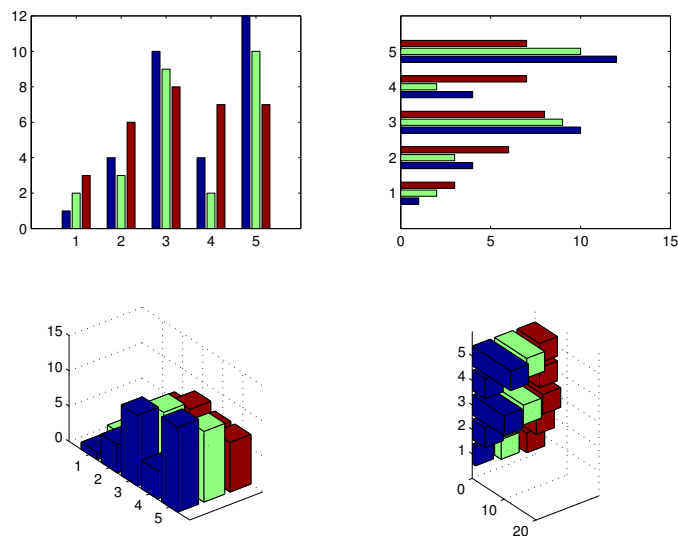
Y por último, también se pueden agrupar en 3D, de forma diferente a la anterior, con la orden `bar3(x,'group')` y se puede hacer que aparezcan las barras apiladas con `bar3(x,'stack')` (ver figura 25).

#### 4.4. Histogramas

Para generar histogramas se utiliza el comando `hist`. Por ejemplo, generamos 1000 números aleatorios siguiendo la normal  $\mathcal{N}(0,1)$

```
>>x=randn(1000,1);
```

Con la orden `hist(x)`, obtenemos (figura 26) un histograma en el que los datos aparecen agrupados en 10 intervalos. Si queremos que



**Figura 24.** Diagramas de barras con datos agrupados.

aparezcan más o menos intervalos, habrá que indicarlo con `>>hist(x,N)`, donde  $N$  es el número de intervalos.

## 5. GRÁFICAS EN MOVIMIENTO: “MOVIES”

Entre las múltiples posibilidades del programa MatLab está la de producir gráficas en movimiento. Se trata de pequeños programas, llamados “movies”, que elaboran una “película” fotograma a fotograma. Estos fotogramas, una vez visualizados, producen la sensación de movimiento.

Veamos un ejemplo: queremos dibujar la gráfica de la curva  $y = \lambda \sin(x)$  para varios valores de  $\lambda$  contenidos en el intervalo  $[-1, 1]$ .

Veamos en primer lugar el programa:

En primer lugar, abrimos el editor de programas de MatLab, con `File New M-File`. Se abre un editor en el que escribiremos lo siguiente,

### Ejemplo 1

```
function cuerda

% movie cuerda

x=linspace(0,2*pi,1000); n=50;
% n numero de fotogramas
```

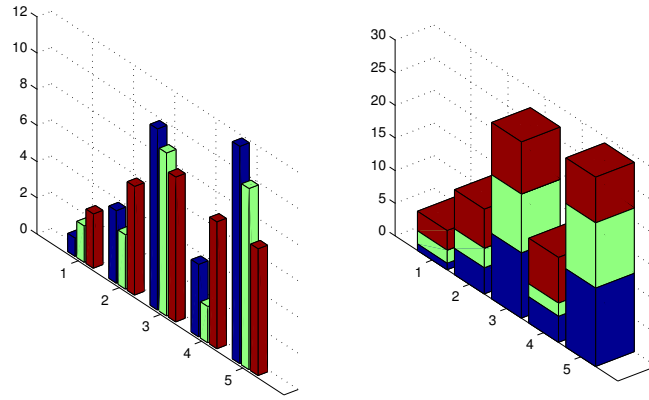


Figura 25. Datos agrupados en 3D y barras apiladas.

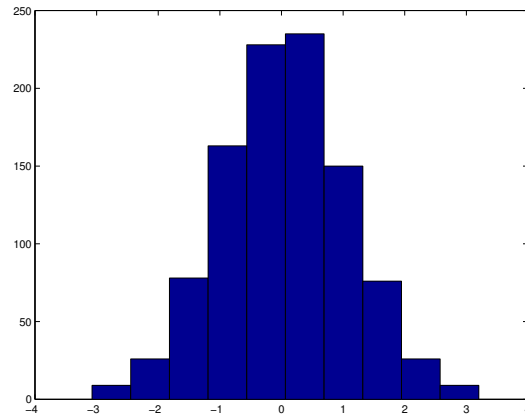
```
for j = 1:n
    t=(2*pi/49)*(j-1);
    y=sin(t)*sin(x);
    plot(x,y,'*'),axis([0 2*pi -1.2 1.2])
    F(j) = getframe;
end
movie(F,2) % veces que queremos ver la peli
```

A continuación lo guardamos (en el directorio que aparece por defecto, `Work`) con el nombre `cuerda`. Si se pone otro nombre, habrá que cambiar la primera línea del programa. Para ejecutarlo, basta con escribir el nombre del programa, `cuerda`, en la línea de comandos.

El núcleo del programa lo constituyen el conjunto de comandos:

```
for j = 1:n
    t=(2*pi/49)*(j-1);
    y=sin(t)*sin(x);
    plot(x,y,'*'),axis([0 2*pi -1.2 1.2])
    F(j) = getframe;
end
```

Es lo que en programación se de denomina un **bucle**, esto es, un conjunto de instrucciones, en este caso, comandos gráficos que se ejecutan varias veces, dependiendo del valor de  $j$ . A medida que  $j$  varía de 1 a



**Figura 26.** Histograma.

50,  $t$  varía, de 0 a  $2\pi$  y, por tanto,  $\lambda = \sin(t)$  varía entre -1 y 1. Para cada valor de  $j$  se realiza un gráfico/fotograma que se almacena con la instrucción `F(j) = getframe;`. Por último, el comando `movie(F,2)` permite visualizar la película el número de veces que se le indique.

A continuación se incluyen algunos ejemplos más de “movies”:

### Ejemplo 2

```
function ellipse

% movie
n=30; x=linspace(0,2*pi,200);

for j = 1:n
    t=(pi/29)*(j-1);
    plot(cos(x),sin(t)*sin(x),'rs'),
    axis([-1 1 -1 1]);
    F(j)=getframe;
end
movie(F,5)
```

### Ejemplo 3

```
function colores

% movie
```

```

n=30;

for j = 1:n
    x=rand(10);
    imagesc(x)
    F(j) = getframe;
end
movie(F,5)

```

#### Ejemplo 4

```

function membrana

% movie membrana
[x,y]=meshgrid(-1:.1:1); n=20;

for j = 1:n
    t=(2*pi/19)*(j-1);
    z=2*sin(t)*exp(-x.^2-y.^2);
    surf(x,y,z),axis([-1 1 -1 1 -2 2])
    F(j) = getframe;
end
movie(F,6)

```

#### Ejemplo 5

```

function picos

% movie

[x,y,z]=peaks; n=20;

for j = 1:n
    t=(2*pi/19)*(j-1);
    z1=sin(t)*z;
    surf(x,y,z1),axis([-3 3 -3 3 -5 5])
    F(j) = getframe;
end
movie(F,3)

```

#### Ejemplo 6



```

function reloj

% movie reloj

n=100;

for j = 1:n;
    t=linspace(0,2*pi,1000);
    plot(cos(t),sin(t)),axis square hold on horas=0:12;
    plot(.9*cos(horas*2*pi/12),... .9*sin(horas*2*pi/12),'k*')
    hor=pi/2-(j-1)*2*pi/(n-1); %horaria
    plot([0 .5*cos(hor)],[0 .5*sin(hor)]),
    min=pi/2-(j-1)*12*2*pi/(n-1); % minuteria
    plot([0 .8*cos(min)],[0 .8*sin(min)]) hold off
    F(j) = getframe;
end
movie(F)

```

## REFERENCIAS

CHEN, K., GIBLIN, P. e IRVING, A. *Mathematical Explorations with MATLAB*. Cambridge University Press. Cambridge, 1999.

DUOANDIKOETXEA, J. “*Análisis de Fourier: historia y aplicaciones recientes*”. En ZUAZUA, E. (Director) *Temas relevantes de la Matemática actual: el reto de la Enseñanza Secundaria*. Centro de publicaciones del Ministerio de Educación, Cultura y Deporte/UIMP. Madrid, 2000. Págs. 11-43.

HARMAN, Th. L., DABNEY, J. y RICHERT, N. *Advanced Engineering Mathematics using MATLAB*. Vol. 4. PWS. Boston, 1997.

HIGHAM, D.J. y HIGHAM, N.J. *Matlab guide*. SIAM. Philadelphia, 2000.

RODRÍGUEZ DEL RÍO, R. “*Matemáticas en el Aula de Informática*”. En ZUAZUA, E. (Director) *Temas relevantes de la Matemática actual: el reto de la Enseñanza Secundaria*. Centro de publicaciones del Ministerio de Educación, Cultura y Deporte/UIMP. Madrid, 2000. Págs. 145-210.

ZUAZUA, E. (Director) *Temas relevantes de la Matemática actual: el reto de la Enseñanza Secundaria*. Centro de publicaciones del Ministerio de Educación, Cultura y Deportes/UIMP. Madrid, 2000.