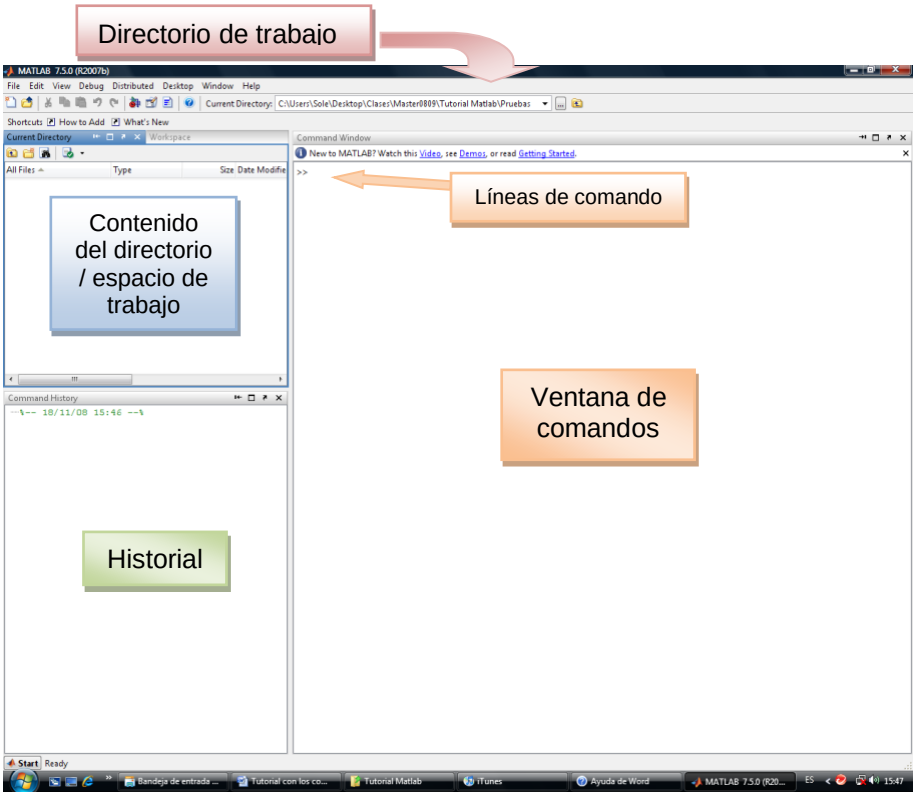
Introducción a Matlab



Directorio de trabajo (*Current Directory*): carpeta del sistema en el que vamos a guardar los archivos que escribamos o leamos con Matlab. Es lo primero que tenemos que elegir al empezar la sesión.

Espacio de trabajo (*Workspace*): espacio de memoria que gestiona Matlab en el que se guarda toda la información que se genera en la sesión. No se puede acceder directamente a él, sólo se puede exportar lo que contiene con la orden **save**.

Ventana de comandos (*Command Window*): ventana de comandos en el que el usuario se comunica con Matlab.

Historial (*Command History*): listado ordenado con fecha de todos los comandos que se han ido introduciendo en la ventana de comandos. Si se hace doble click sobre una orden se vuelve a ejecutar. Si se seleccionan varios comandos con el botón derecho se puede incluso crear un fichero (m-file) que las almacene.

Las primeras veces que se trabaja con Matlab es recomendable hacer uso de la orden

>>diary *nombre.txt*

con lo que se crea un fichero de texto *nombre.txt* en el que se guardan todas las órdenes de la sesión actual y así poder ver de nuevo lo que hicimos. Para dejar de escribir en dicho fichero y cerrarlo hay que escribir

>>diary off

Si más tarde en la misma sesión se quiere seguir escribiendo en el mismo fichero

```
>>diary on
```

No es muy útil en general pues es sólo un fichero texto descriptivo.

CÁLCULOS SIMPLES

```
>>2+3
>>3*4
>>5/3
>>3^5
>>cos(2+sqrt(3))
>>exp(3/log(5))
```

Como vemos, el último valor calculado lo almacena en la variable **ans**

```
>>cos(ans)
```

Para resolver una duda con la **ayuda online**

```
>>help sqrt
```

Si se escribe el principio de un comando y luego la tecla de tabulado Matlab te da las opciones que tiene para completarlo.

Toda sentencia que empieza por **%** se considera línea de comentario no ejecutable.

Funciones intrínsecas en Matlab:

Trigonométricas en radianes	`cos, sin, tan, csc, sec, cot`
Trigonométricas en grados	`cosd, sind, tand, cscd, secd, cotd`
Trigonométricas inversas en radianes	`acos, asin, atan, atan2, acsc, asec, acot`
Trigonométricas inversas en grados	`acosd, asind, atand, acscd, asecd, acotd`
Hiperbólicas	`cosh, sinh, tanh, csch, sech, coth`
Hiperbólicas inversas	`acosh, asinh, atanh, acsch, asech, acoth`
Logarítmicas y exponenciales	`log, log2, log10, log1p, exp, expm1, pow2, nextpow2, nthroot`
Redondeo	`ceil, fix, floor, round`
Complejos	`abs, angle, conj, imag, real`
Resto, signo	`mod, rem, sign`

CREACIÓN DE VARIABLES:

```
>>x=1.3
>>y=-3*x+sqrt(x)
```

Si se pone **;** al final se ejecuta pero no se ve el resultado

```
>>y=4*x-5*log(x);      %se borra el valor anterior
>>y
```

se pueden usar para hacer recursiones

```
>>y=2*y
```

Importante: Matlab distingue mayúsculas y minúsculas, es decir, x no es lo mismo que X

```
>>Y=3*x
```

Para limpiar una variable

```
>>clear Y
```

Para limpiar todas las variables (es como si volviesemos a empezar)

```
>>clear        %Cuidado! a veces no interesa limpiarlas todas
```

Para continuar un comando en la línea inferior hay que poner puntos suspensivos:

```
>>s=1+2+3+4+5+...
        +6+7
```

Ojo: no vale para cadenas de caracteres: hay que cerrarlas en la misma línea.

También se pueden colocar varias sentencias seguidas: o en la misma línea separadas por comas o con Mayúsculas+Enter en la siguiente línea de comando.

Para recuperar órdenes hay que usar las flechas arriba/abajo. Así podemos hacer recursivamente

```
>>x=0.2
>>x=cos(x)
>>x=cos(x)      %....(Repetir varias veces con las flechas)
```

Matlab **SIEMPRE TRABAJA EN DOBLE PRECISIÓN**, es decir, con 15-16 cifras decimales exactas. Otra cosa es como lo muestra en pantalla.
Por defecto muestra solo 4 cifras decimales. Para que muestre las 16 cifras se pone

```
>>format long
>>x=1.3
>>y=-3*x+sqrt(x)
```

Ejemplo de formatos: cómo aparece el número π

format short	3.1416
format long	3.141592653589793
format short e	3.1416e+00
format long e	3.141592653589793e+00
format short g	3.1416
format long g	3.14159265358979
format hex	400921fb54442d18
format bank	3.14
format rat	355/113

NÚMEROS COMPLEJOS

Para Matlab i=√-1 es la unidad imaginaria siempre que no se le asigne otro valor.
Los números complejos se introducen de distintas formas

```
>>z= 2-3i
>>z= 2-3*i     %esta no se prefiere por si le hemos dado a i otro valor
>>z= 2-3*sqrt(-1)
>>z= complex(2,-3)   %esta es la mejor forma siempre que sea posible
```

Maneja funciones específicas para complejos:

real(z)	parte real
imag(z)	parte imaginaria
abs(z)	módulo
conj(z)	conjugado

Además, las funciones trigonométricas, exponencial, etc., devuelven valores complejos si los argumentos que le ponemos son complejos. Y también se puede trabajar con vectores y matrices con elementos complejos.

VECTORES Y MATRICES

Para introducir vectores se puede hacer componente a componente

```
>>v(1)=0, v(2)=1, v(3)=4, v(4)=5
```

o en conjunto entre [ ] y separados por comas o espacios en blanco

```
>>v=[0,1,4,5]
>>v=[0 1 4 5]
```

Comando ':' i:INC:j varía de i a j con incremento INC.
Se usa para definir vectores, pero también bucles como veremos más tarde

```
>>v=[0:0.1:1]                % Va de 0 a 1 con incremento 0.1.
```

Puede ser negativo

```
>>w=[6:-0.2:4]
```

Observa este caso

```
>>z=[0:0.3:1]
```

Se pueden hacer operaciones entre componentes

```
>>w(1)*w(2)*z(4)
```

Importante: en Matlab no existe la componente 0-ésima. Con esto tendremos algún problema cuando intentemos pasar algunos algoritmos. Habrá que reajustar los subíndices en las fórmulas.

Las matrices se pueden introducir por filas separadas por ;

```
>>A=[1,2,-3; 3,4,5; 6,7,8]
```

o dándole al Enter después de cada fila

```
>>A=[1,2,-3
      3,4,5
      6,7,8]
```

Y también componente a componente

```
>>A(1,1)=1, A(1,2)=2, A(1,3)=-3 ,....
```

Operaciones con vectores y matrices

La suma (resta) y producto siguen las reglas del álgebra lineal

```
>>B=[1,4,-5; 0,3,6;-1,-2,9]
>>z=[-3,4,5]
>>A+B
>>A*B
>>A*z
>>A*w      %da error pues las dimensiones no cuadran
>>A/B      %esto es A*B^(-1) (por la derecha)
```

Traspuesta: apostrofe '

```
>>C=A'
>>w=v'
```

Operaciones componente a componente

Las operaciones básicas se pueden hacer componente a componente poniendo un punto delante del operador:

```
>>A.*B
>>B./A
>>B.^2
```

Las funciones intrínsecas lo hacen componente a componente siempre, y también escalar*matriz

```
>>D=cos(A)
>>3*D
```

También es posible sumar a una matriz un escalar, en el sentido de que se suma el escalar a todos los elementos de la matriz:

```
>>A+2
```

Otras funciones matriciales

```
r=size(A)       devuelve las dimensiones de A
d=det(A)        determinante de A
E=inv(A)        matriz inversa numérica
F=diag([1,2,3])     matriz diagonal donde [1,2,3] es la diagonal principal
I1=zeros(3)         matriz nula de dimensión 3
I2=eye(4)           matriz identidad de dimensión 4
unos=ones(size(A))      matriz de la misma dimensión que A cuyos elementos son todos 1
```

Submatrices y subíndices

Se pueden extraer submatrices de una matriz dada con el comando i:j

```
A(1:2,2:3)    %elementos de las filas 1 y 2 y las columnas 2 y 3.
A(1:2,2)      %elementos de las filas 1 y 2 y la columna 2.
A(:,3)        %todos los elementos de la columna 3 (todas las filas)
A(2,:)        %todos los elementos de la fila 2 (todas las columnas)
```

Cuando se quieren extraer elementos de filas o columnas no consecutivos:

```
A([1,3],[2,3])  %elementos de las filas 1 y 3 y las columnas 2 y 3
```

Un comando muy útil cuando no conocemos exactamente la dimensión de la matriz es el comando **end**

```
A(end,:)        %todos los elementos de la última fila (todas las columnas)
A(1:2,2:end)  %los elementos de las filas 1 y 2 y las columnas desde la 2 a la última
```

Comando [ ]: denota una matriz vacía 0x0. Se puede usar para eliminar los elementos de una fila o una columna de una matriz

```
A(2,:)=[ ]      %elimina la segunda fila de A
```

Resolución de sistemas lineales Ax=r: lo hace mediante descomposiciones LU, ahorrando operaciones en el caso de matrices especiales como las diagonales simétricas (hay opciones para cambiar el algoritmo de resolución pero no las veremos)

```
>>r=[5,6,-4]'       %Ojo: hay que trasponer pues es un vector columna
>>x=A\r
```

Análisis de datos: Matlab tiene muchos comandos del análisis de datos. La siguiente tabla incluye los básicos. Además, tiene paquetes específicos para Estadística pero no entran en nuestro curso.

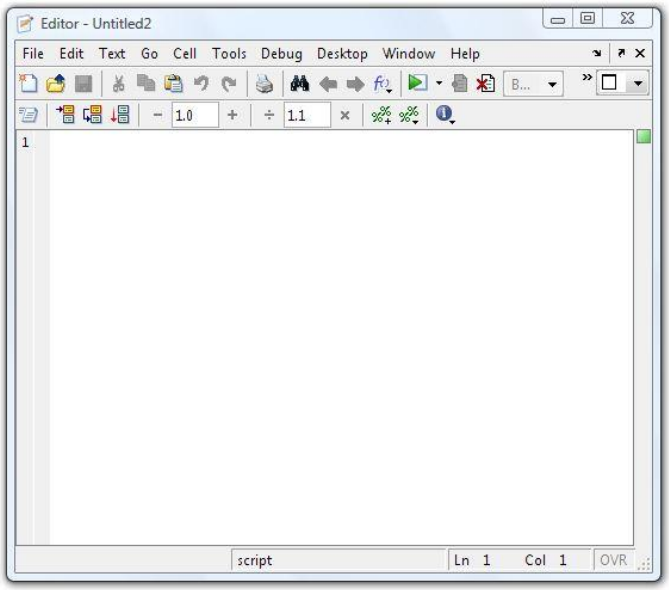
`max`	Mayor componente
`min`	Menor componente
`mean`	Media
`median`	Mediana
`std`	Desviación típica
`var`	Varianza
`sort`	Ordena en orden ascendente
`sum`	Suma de las componentes
`prod`	Producto de las componentes
`cumsum`	Suma acumulativa de las componentes
`cumprod`	Producto acumulativo de las componentes
`diff`	Diferencias de elementos

FICHEROS-M ("M-FILES"): SCRIPTS Y FUNCIONES

Aunque se puede trabajar directamente en las líneas de comando desde que tenemos que ejecutar muchas sentencias seguidas se hacen necesarios los ficheros-m. Se llaman así porque tienen que guardarse con la **extensión .m** y sirven para almacenar comandos Matlab**.**

Para crearlos **File -> New -> M-File** o picando el botón correspondiente. Así se abre el Editor de ficheros-m.

Editor de texto de los ficheros-m



Hay dos tipos de ficheros-m:

- **Scripts:** almacenan comandos Matlab que se ejecutarán en orden cuando desde la línea de comando escribamos el nombre del fichero-m sin extensión. No tienen argumentos de entrada ni de salida a diferencia del otro tipo de ficheros-m. Los comandos operan sobre las variables del espacio de trabajo de la sesión que tenemos abierto.
- **Funciones:** se diferencian porque la primera sentencia ejecutable es
  function [*arg_out***]=nombre(***arg_in***)**
  donde
  - **nombre** debe coincidir con el nombre del fichero-m en el que se almacena esta función (recuerda que Matlab distingue mayúsculas y minúsculas).
  - *arg_in* = lista de argumentos de entrada separados por comas y ordenados, que necesitaremos aportar a dicha función.
  - *arg_out*= lista de argumentos de salida ordenados y separados por comas.

  Se recomienda que el orden de colocación de los argumentos sea según su importancia, es decir, primero los argumentos imprescindibles para que trabaje la función y luego aquellos que pudieran ser prescindibles en algunos casos. En Matlab se puede trabajar con un número variable de argumentos aunque por ahora no lo veremos.

Este tipo de funciones son lo que en otros lenguajes se llaman rutinas, subrutinas o subprogramas.

Los ficheros script están cayendo en desuso. Incluso cuando no hay argumentos de entrada ni de salida se prefiere el uso de funciones sin argumentos (**function nombre**) pues permiten más aplicaciones que con los scripts, sobre todo en las últimas versiones de Matlab.

En el encabezado de un fichero-m se pueden poner líneas de comentario con % (antes de la primera sentencia ejecutable), lo que se usa para hacer un

pequeño resumen de lo que hace dicho fichero. Posteriormente, desde la interface de Matlab, si escribimos

>>**help** nombre %(del fichero-m)

nos aparecerá lo escrito en dichas líneas de comentario.

Ejemplo de script	Ejemplo de función
%PRUEBA.M: ejemplo de script % Suma dos matrices y da su dimensión A=[-2,3; 0,1; 3,5]; B=[-1,4;-8,3; 2,2]; C=A+B dim=size(C)	%SUMA.M: ejemplo de función % Dadas dos matrices nos devuelve su %suma si tienen igual dimensión function [C,err]=suma(A,B) if size(A) == size(B) C=A+B; err=0; else C=0; err=1; end

En el ejemplo prueba.m, para ejecutarlo en la línea de comando escribimos prueba sin la extensión. Así simplemente se ejecutan las sentencias que almacena para las matrices A y B que hemos definido.

En el ejemplo suma.m, para llamarlo en la línea de comando tenemos que poner

>> **[C1,err1]=suma([1,1;3,4],[-1,2;5,6])**

y nos devuelve en C1 el valor de la suma de las dos matrices que le hemos dado en los argumentos de entrada.

También se puede llamar esta función desde otro fichero-m (script o función) que esté en el mismo directorio.

La manera más rápida de interrumpir la ejecución de una función es apretando las teclas *Control+c.*

Los iconos  de la parte superior del editor de ficheros sirven para evaluar una parte seleccionada de la función, evaluarla y avanzar, o evaluar toda la función en la ventana de comandos.

Las variables en las funciones son todas locales, es decir, las funciones tienen su propio espacio de trabajo (workspace) independiente del espacio de trabajo de la sesión Matlab o del programa que las llame. En el ejemplo anterior las matrices A, B y C se usan sólo dentro de la rutina. Es decir, se puede poner en la línea de comando

>> [A,B]=suma(C,D)

pues las matrices A, B y C de la función no comparten memoria con las del espacio de trabajo. De todas formas esto no supone duplicar el consumo de memoria pues Matlab usa una optimización automática para evitarlo: ***los argumentos pasados a una función no se copian en el espacio de trabajo de la función a menos que se modifiquen dentro de la función*** (lo que no

es recomendable). Por tanto, no hay problemas de memoria al pasar variables grandes a funciones siempre que dichas funciones no las alteren.

Obsérvese que en las funciones es interesante poner **;** en todas las sentencias para evitar que salgan un montón de datos intermedios en la interface.

En Matlab las funciones pueden ser argumentos de entrada de otras funciones. Para ello hay que poner el símbolo @ precediendo el nombre.

Hay que tener en cuenta que hay una ligera diferencia en el manejo de funciones según la versión de Matlab que usemos. En Matlab 6 hay que usar el comando **feval**. Por ejemplo:

%FUNC1.M: función
%f(x)=sin(x)^2
 	
function f=func1(x)
f=sin(x)^2
	%DERIVA.M: function que
	%aproxima la derivada de una
	%función por su cociente
	%incremental con h=1.e-5
	%VERSIÓN MATLAB 6
	
	function y=deriva(f,x)
	h=1.e-5;
	y=(feval(f,x+h)-feval(f,x))/h;
		%DERIVA.M: function que
		%aproxima la derivada de una
		%función por su cociente
		%incremental con h=1.e-5
		%VERSIÓN MATLAB 7
		
		function y=deriva(f,x)
		h=1.e-5;
		y=(f(x+h)-f(x))/h;

Podemos calcular la aproximación de la derivada de la función $f(x)=\sin(x)^2$ en x=0.1 escribiendo en la línea de comando:

>>**y1=deriva(@func1,0.1)**

A partir de Matlab 7 aparecen otras formas de definir funciones sencillas y las **funciones anidadas** que son muy útiles cuando tenemos un gran número de argumentos compartidos entre dos funciones.

GRÁFICAS 2D

Matlab dibuja puntos (x,y) que tenemos que darle y luego los une con líneas

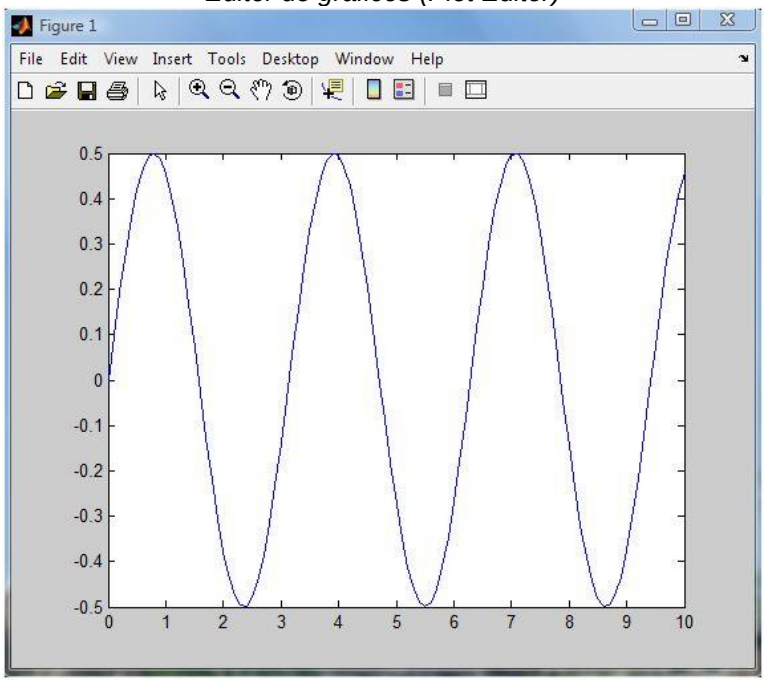
```
>>x=[1,2,3,4], y=[3,6,7,0]
>>plot(x,y)
```

Si queremos dibujar una función más compleja tenemos que darle muchos puntos:

```
>>x=[0:0.1:10];              %ponemos ; para que no salga toda la lista
>>y=cos(x).*sin(x);   %observa que es una operación por componentes
>>plot(x,y)
```

La ventana que se abre es el **Editor de gráficos** (Plot Editor):

Editor de gráficos (Plot Editor)



En la barra superior de este editor se abre un amplio menú con el que podremos modificar muchas características de este gráfico. En general, sobre todo a la hora de exportar los gráficos a otros formatos, suele ser mejor cambiar dichas características en la línea de comando en la que definimos el plot, como veremos a continuación. En la práctica usaremos las dos posibilidades según nos convenga.

También está la alternativa de **linspace** para definir el vector x, que suele ser más cómoda en los gráficos

```
>>x=linspace(0,10,100);      %crea 100 puntos de 0 a 10
>>y1=cos(x).*sin(x);         % se pone el punto pues es producto matricial
>>y2=cos(3*x)/4;
>>plot(x,y1,x,y2)            %mezcla de dos o mas curvas
```

Nota: no es lo mismo linspace(0,10,100) que 0:0.1:10.

Si x es un vector de m componentes e Y es una matriz mxn, **plot(x,Y)** dibuja en el mismo gráfico los n grafos formados por los elementos de x frente a cada columna de Y:

```
>>x=[1,2,3,4];        %da igual que x sea vector fila o vector columna
>>Y=[1,2; 2,-2; 4,2; 5,-2];
>>plot(x,Y)
```

Si X e Y son dos matrices mxn, **plot(X,Y)** dibuja en el mismo gráfico los n grafos formados por las columnas de X frente a las columnas de Y:

```
>>X=[1,2; 3,4; -1,-2; -3,-4];
>>Y=[1,2; 2,-2; 4,2; 5,-2];
>>plot(X,Y)
```

Opciones básicas:

```
>>x=linspace(0,10,100);
>>y1=cos(x).*sin(x);
>>y2=cos(3*x)/4;
>>plot(x,y1,'r',x,y2,'mo'), title('Funciones'), text(5,0.3,'Puntos'),…
xlabel('x'), ylabel('y=f(x)'), axis('equal'), legend('Funcion 1','Funcion 2')
```

Esta línea dibuja la función y1 en línea continua roja (‘r’), la y2 en línea punteada de color magenta ('m') con el símbolo 'o'.
Le pone el título 'Funciones', en el punto (5,0.3) pone la etiqueta 'Puntos', en el eje de las x pone la etiqueta 'x', en el eje de las y la etiqueta 'y=f(x)'.
La orden axis('equal') hace que la escala en el eje x y en el y sean iguales, o sea, una escala 1:1.
Además pone una leyenda en el borde superior izquierdo, identificando cada curva.

Un comando útil cuando queremos añadir un gráfico a otro ya existente es la opción **hold on … hold off.** Por ejemplo

```
>>x=linspace(0,1,20);
>>for i=1:4, plot(x, i*x), ylim([0,4]), hold on, pause, end
```

genera las cuatro rectas x, 2x, 3x, 4x en [0,1] una detrás de otra. Obsérvese el uso de **pause**, que provoca la parada temporal hasta que le demos al Enter, para ir viéndolas una tras otra. Es importante que finalicemos con

```
>>hold off
```

para que las siguientes gráficas se dibujen aparte.

Es recomendable usar la ayuda (help plot, help title,...).

A lo largo del curso iremos viendo más opciones según nos vayan haciendo falta.

Opciones para el color, el marcador y el estilo de línea:

Color		Marcador		Estilo de línea	
r	rojo	o	círculo	-	sólida (por defecto)
g	verde	*	asterisco	--	a rayas
b	azul	.	punto	:	punteada
c	cian	+	más	-.	punto-raya
m	magenta	x	cruz		
y	amarillo	s	cuadrado		
k	negro	d	diamante		
w	blanco	^	triángulo hacia arriba		
		v	triángulo hacia abajo		
		>	triángulo hacia la derecha		
		<	triángulo hacia la izquierda		
		p	estrella de 5 puntas		
		h	estrella de 6 puntas		

Opciones para los ejes:

`axis([xmin,xmax,ymin,ymax])`	especifica los límites de los ejes x e y
`axis auto`	vuelve a los límites de los ejes por defecto
`axis equal`	usa la misma escala en los ejes x, y y z
`axis off`	elimina los ejes
`axis square`	reescala para que los ejes formen un cuadrado
`axis tight`	ajusta los límites de los ejes a los datos
`xlim([xmin,xmax])`	especifica los límites del eje x
`ylim([ymin,ymax])`	especifica los límites del eje y

Gráficos múltiples

Podemos dibujar varios gráficos distintos en la misma figura con el comando

subplot(m,n,p)

Este comando divide la ventana en mxn zonas, y en cada una de ellas vamos a dibujar un gráfico. El tercer parámetro p nos dice en qué región estamos dibujando.

```
>>x=linspace(-2*pi,2*pi,100);      %pi=3.14… lo computa Matlab
>>y1=cos(x); y2=sin(x);
>>subplot(2,2,1), plot(x,y1,'r')    %región (1,1)
>>subplot(2,2,2), plot(x,y2,'g')    %región (1,2)
>>subplot(2,2,3), plot(x,y1.*y2,'k')        %región (2,1)
>>subplot(2,2,4), plot(x,y1+y2,'b')         %región (2,2)
```

Gráficos simples

Matlab tiene una orden para dibujar gráficas simples:

>>**ezplot**(*fun*,[xmin,xmax])

dibuja la función fun en el intervalo [xmin,xmax], dónde la función *fun* se puede introducir de dos formas: directamente entre apóstrofes o si está almacenada en un fichero function mediante la arroba @. Ejemplos:

```
>>ezplot('x^2+sin(x)',[-pi,pi])
>>ezplot(@cuad,[-pi,pi])
```

cuando la función está almacenada en un fichero-m cuad.m y está escrita de la forma

```
function f=cuad(x)
f=x.^2+sin(x)                %observa el punto para la operación matricial
end
```

Tiene el inconveniente de que tiene muy pocas opciones y la función a dibujar ha de ser simple por lo que no se suele utilizar mucho.

Guardar e imprimir los gráficos

La manera más sencilla de guardar e imprimir los gráficos es desde el Plot Editor:

File->Save as: se guardan los gráficos en los siguientes formatos:
-Figura Matlab .fig
-PostScript (encapsulado): .eps (recomendado, es el mejor)
-Pdf
-Bitmap .bmp
-JPEG .jpg
-TIFF .tif

File->Print: para imprimir directamente a impresora.

También existe el comando **print** para exportar directamente desde una función un gráfico en un formato específico y con un nombre. Para ello si tenemos una figura abierta y ponemos

```
>>print -depsc2 figura1.eps
```

se crea en el directorio de trabajo el fichero figura1.eps con formato Postcript encapsulado de nivel 2 en color (Encapsulated Level 2 Color Postcript), que es el mejor cuando se quiere insertar el gráfico en algún documento.

Otros formatos habituales disponibles son

```
-deps2    % Encapsulated Level 2 PostScript (blanco y negro)
-djpeg    % Imagen JPEG (hay una opción para aumentar la resolución)
-dtiff    % Imagen TIFF
```

Para ver el listado total de formatos basta poner >>help print.

Este comando es muy útil cuando un programa genera varias figuras a lo largo de su ejecución. Por ejemplo, el bucle

```
x=linspace(0,2*pi,50);
for i=1:5
        plot(x,sin(i*x))
        print('-depsc2',['fig', int2str(i), '.eps'])
end
```

genera 5 gráficos diferentes de las funciones sin(x), sin(2x),…, sin(5x), en formato Postcript en color almacenados en sendos ficheros fig1.eps, fig2.eps,…, fig5.eps.

GRÁFICOS 3D

La opción más simple para crear gráficos 3D es la de

plot3(x,y,z)

donde x,y,z son tres vectores de la misma dimensión. Esta sentencia une los puntos (x_i,y_i,z_i) en orden en el espacio tridimensional.

```
>>t=-5:.005:5;
>>x=(1+t.^2).*sin(20*t);
>>y=(1+t.^2).*cos(20*t);
>>z=t;
>>plot3(x,y,z)
>>grid on
>>xlabel('x(t)','FontSize',14), ylabel('y(t)','FontSize',14)
>>zlabel('z(t)','FontSize',14,'Rotation',0)
```

```
>>title('Ejemplo plot3','Fontsize',14)
```

Observa en qué afectan las opciones grid on, FontSize y Rotation.

Las opciones de color, marcador y estilo de línea son las mismas que para plot.

Gráficos de funciones de dos variables z=f(x,y)

Para dibujar los puntos

$$C=\{(x,y,z) \text{ tal que } z=f(x,y),\ a\le x\le b,\ c\le y\le d\}$$

hay que hacer previamente un mallado en (x,y), es decir, crear una malla ("grid") de puntos en el rectángulo [a,b]x[c,d] con **ndgrid.** Para ver cómo funciona primero veamos un ejemplo simple:

```
clear
x=[1,2,3,4];  y=[-1,-2,-3];
[X,Y]=ndgrid(x,y);
```

Así X,Y son matrices de la misma dimensión 4x3, donde X almacena 3 copias de x en columnas e Y almacena 4 copias de y en filas.
Por tanto, si x es un vector de dimensión m e y es un vector de dimensión n, X e Y son matrices de dimensión n x m tales que $X_{ij}=x_i$, $Y_{ij}=y_j$, para cada $1\le i\le n$, $1\le j\le m$. Como consecuencia, variando i,j en cada par (X_{ij}, Y_{ij}) tenemos todas los posibles pares de puntos de la malla (x_i, y_j).
Si calculamos, por ejemplo, Z=X+Y, Z es una matriz que almacena todas las posibles sumas de elementos de x y de y, es decir, $Z_{ij}=X_{ij} + Y_{ij}= x_i + y_j$.

Para dibujar z=f(x,y) entonces usamos el comando **mesh:**

```
>>x=0:0.1:pi;  y=0:0.1:pi;
>> [X,Y]=ndgrid(x,y);
>>Z=sin(Y.^2+X)-cos(Y-X.^2);
>>mesh(Z)
```

También se puede usar el comando **meshgrid,** con el que se computan las traspuestas de las anteriores, es decir, si hacemos [X,Y]=meshgrid(x,y), X e Y son ahora matrices de dimensión 3x4, donde X almacena 3 copias de x en filas e Y almacena 4 copias de y en columnas. Por tanto, $X_{ij}=x_j$, $Y_{ij}=y_i$, para cada $1\le i\le n$, $1\le j\le m$, con lo que variando i,j en cada par (X_{ij}, Y_{ij}) tenemos todas los posibles pares de puntos de la malla (x_j, y_i).

También podemos usar el comando **surf(Z)**, que rellena la superficie formada por z=f(x,y), coloreando los diferentes niveles. Además, para ver la escala de colores que ha usado se puede añadir la orden

```
colorbar
```

que coloca al lado de la gráfica 3D una barra escalada con dichos colores.

Otra alternativa es **waterfall(Z).**

Curvas de nivel ("contour plots")

Cuando trabajamos con funciones de dos variables z=f(x,y) es muy útil dibujar las curvas de nivel, es decir las curvas C_z={(x,y) tal que f(x,y)=z} para cada z fijo.
Usando el mismo ejemplo anterior hacemos

```
>>x=0:0.1:pi;  y=0:0.1:pi;
>> [X,Y]=ndgrid(x,y);
>>Z=sin(Y.^2+X)-cos(Y-X.^2);
>>contour(x,y,Z)
```

Aquí Matlab elige automáticamente los niveles.

La opción **contour(x,y,Z,20)** hace que tome 20 niveles diferentes. Añadiendo colorbar vemos qué valores toman esos niveles.

También se pueden especificar qué valores han de tomar los niveles almacenándolos en un vector.

```
>>cvals=[-1.5:0.2:1.5];
>>contour(x,y,Z,cvals), colorbar
```

Incluso podemos hacer que sobre cada curva aparezca el valor que toma:

```
>> [C,h]=contour(x,y,Z,cvals)
>>clabel(C,h)
```

En este comando C almacena los valores de los niveles y h es lo que se llamará manipulador del gráfico (“graphic object handle”).

Los comandos mesh y surf tienen unas versiones alternativas

```
meshc(Z)
surfc(Z)
```

que incluyen en el plano XY del gráfico tridimensional las curvas de nivel.

MATLAB SIMBÓLICO

Uno de los paquetes incluidos en Matlab es el Symbolic Math Toolbox que le permite trabajar simbólicamente. Suele estar incluido en la versión para estudiantes, pero a veces hay que comprarlo aparte. Si se pone en la línea de comandos la sentencia **ver** podrás ver el listado de los paquetes incluidos en tu versión. Este paquete está basado en Maple. De hecho se puede usar cualquier comando de Maple escribiendo

```
>>maple(‘funcion’,arg1,arg2,…)
```

Por ejemplo,

```
>>maple(‘evalf(Pi,15)’)
```

De todas formas es mejor usar los comandos simbólicos directamente en Matlab.

Para que Matlab distinga si una variable es simbólica o no hay que definirla como tal con la orden **syms**:

```
>>syms a,b,c
>>y=solve(a*x^2+b*x+c=0,x)       %resuelve la ecuación de 2º grado
```

Matlab tiene muchos comandos simbólicos más. Debajo aparecen algunos ejemplos. Si se quiere ver en más detalle todos los comandos que incluye basta poner **help symbolic.**

```
syms x,y
int(sqrt(tan(x)))        %integral indefinida
int(sqrt(tan(x)),0,5)  %integral definida de 0 a 5
diff(x^5)                %derivada primera
diff(x^5,3)              %derivada tercera
taylor(log(1+x),n,a) %polinomio de Taylor de grado n en a
A=[1,2;3,4], As=sym(A)     %hace simbólica la matriz A
inv(As)                  %matriz inversa simbólica
eig(As)                  %autovalores de A
poly(As)                 %polinomio característico de la matriz A
```