BREVES APUNTES PARA COMENZAR CON MATLAB

1. SINTAXIS GENERAL

En MATLAB, en general, las letras minúsculas y mayúsculas NO SON IGUALES.
La ejecución de cualquier comando puede abortarse mediante **CONTROL + C**.
Se pueden escribir varios comandos en una misma línea, separándolos por "**coma**" o por "**punto y coma**".
Se pueden recuperar comandos anteriores, usando las teclas de flechas arriba y abajo. Con las flechas izquierda y derecha nos podemos desplazar sobre la línea de comando y modificarlo.

1.1 Constantes

Enteros: `12 -3`
Reales: `8.01 -5.2 .056 1.4e+5 -.567e-21`
Complejos: `1+2i -3+j` (`i j` son símbolos que representan la unidad imaginaria)
Caracteres (entre apóstrofes): `'esto es una cadena de caracteres' 'string'`

1.2 Operaciones aritméticas elementales

Suma:	`+`
Resta:	`-`
Multiplicación:	`*`
División:	`/`
Exponenciación:	`^`

Se puede utilizar MATLAB como simple calculadora, escribiendo expresiones aritméticas y terminando por **RETURN** (**<R>**). Se obtiene el resultado inmediatamente a través de la variable del sistema **ans** (de answer). Si no se desea eco en el terminal, deben terminarse las órdenes por "**punto y coma**".

1.3 Variables

Los nombres de variables pueden tener a lo sumo 19 caracteres, letras y números. El primero debe ser una letra. No se pueden utilizar los caracteres especiales:

`+ - = * ^ < > ...`

Las variables en MATLAB no necesitan ningún tipo de declaración y pueden almacenar sucesivamente distintos tipos de datos: enteros, reales, escalares, matriciales, caracteres, etc. Se crean, simplemente, asignándoles un valor.
Se pueden eliminar variables mediante el comando `clear`

`clear`	elimina todas las variables que existan en ese momento
`clear a,b,c`	elimina las variables a, b y c

Atención: recuérdese que las variables **AB ab Ab** y **aB** SON DISTINTAS.

Para conocer en cualquier instante el valor almacenado en una variable basta con teclear su nombre.
Se pueden conocer todas las variables definidas hasta el momento tecleando el comando

`who`	lista las variables actuales
`whos`	como el anterior, pero más detallado

EJEMPLO

```
>> a=10; <R>
>> pepito=2.4/3, <R>
>> b=a+pepito; <R>
>> b <R>
b =
     10.800
>> b=b+4-0.5i <R>
b =
     14.800 - 0.5000 i
```

1.4 Formatos

Por defecto, cuando MATLAB nos muestra un valor real, nos muestra sólo cinco cifras significativas (formato corto). Se puede modificar la forma de mostrar los valores mediante el comando **format**:

format long	14 cifras significativas
format short	vuelve al formato corto (5 cifras significativas)
format	vuelve al formato por defecto (corto)
format short e	formato corto y notación exponencial
format long e	formato largo y notación exponencial
format rat	formato racional: aproximación en forma de fracción

EJEMPLOS

```
>> a=.0001234567
a =
       1.2346e-004
>> format long
>> a
a =
       1.234567000000000e-004
>> format rat
>> a
a =
       1/8100
```

1.5 Variables predefinidas

Algunos nombres están pre-definidos por MATLAB:

ans	variable del sistema para almacenar el resultado de evaluar expresiones
i , j	unidad imaginaria : raiz cuadrada de -1
pi	número π
Inf	"Infinito": número mayor que el más grande que se puede almacenar
NaN	"Not a Number : magnitud no numérica resultado de cálculo indefinidos

EJEMPLOS

```
>> 5/3
ans =
    1.6667
>> b=5/0
Warning: Divide by zero.
b =
   Inf
>> b/b
ans =
   NaN
```

1.6 Funciones matemáticas elementales

sqrt(x)	raiz cuadrada	**sin(x)**	seno
abs(x)	módulo	**cos(x)**	coseno
conj(z)	complejo conjugado	**tan(z)**	tangente
real(z)	parte real	**asin(x)**	arcoseno
imag(z)	parte imaginaria	**acos(x)**	arcocoseno
angle(z)	argumento	**atan(x)**	arcotangente
exp(x)	exponencial	**rats(x)**	aprox. racional
log(x)	logaritmo natural	**rem(x,y)**	resto de dividir x por y
log10(x)	logaritmo decimal	**sign(x)**	signo (1 / -1 / 0)

Los argumentos pueden ser, siempre que tenga sentido, reales o complejos y el resultado se devuelve en el mismo tipo del argumento.

1.7 Algunos comandos utilitarios y de ayuda

El principal comando de ayuda en MATLAB es **help**, que nos da una lista de tópicos sobre las que pedir ayuda. También se puede pedir ayuda directamento sobre un comando:

`help`	Lista de los tópicos
`help ops`	Lista de operadores y caracteres especiales
`help lang`	Lista de comandos de programación
`help clear`	Ayuda sobre el comando clear
`lookfor texto`	Lista de los comandos/funciones en cuyas explicaciones Aparece la cadena texto
`clc`	"Limpia" la ventana de comandos

1.8 Elementos sobre ficheros

Podemos utilizar ficheros para:

- Guardar el valor de todas o algunas de las variables definidas en una sesión
- Agrupar un conjunto de sentencias que puedan utilizarse en cualquier momento
- Almacenar todas las sentencias ejecutadas durante una sesión de trabajo
- Definir nuevas funciones (se verá más adelante)

`save fich v1,v2`	Guarda en el fichero **`fich.mat`** los nombres y los valores de las variables especificadas. Los ficheros creados por este sistema son ficheros binarios, no de texto.
`load fichero.mat`	Recupera las variables almacenadas en un fichero ***.mat**

`diary fichero`
`… comandos …`
`diary off`
	Hace que se guarden en el fichero **`fichero`** todas las órdenes, con sus resultados, ejecutadas entre las dos órdenes diary . El fichero asi obtenido es ASCII: se puede editar e imprimir.

`dir`	Lista de todos los ficheros del directorio actual
`type fichero`	Muestra por pantalla el contenido del fichero indicado
`delete fichero`	Borra el fichero especificado
`cd path`	Cambia el directorio actual al indicado mediante el path
`pwd`	Indica el directorio actual

2. VECTORES Y MATRICES

El nombre MATLAB viene de **MAT**rix **LAB**oratory. En MATLAB todos los datos son matrices. Los vectores y escalares son casos particulares de matrices.
En MATLAB no es necesario especificar previamente las dimensiones de una matriz. Se definen de forma interactiva, al "crearla". De hecho, una vez creada, se pueden modificar sus dimensiones

2.1 Definición de vectores

Un vector-fila de dimension **n** (una matriz de dimensiones **1xn**) se puede definir en MATLAB escribiendo sus componentes entre corchetes rectos (**[ ]**) y separándolos por comas o espacios en blanco:

```
>> v=[1,-1,0,2.88]
```

La orden anterior crea en MATLAB una variable de nombre **v** que "contiene" un vector-fila de longitud 4 (una matriz 4x1).
Un vector-columna se crea igual, pero separando las componentes por "punto y coma":

```
>> w=[0;1;2;3;4;5]
```

crea una variable de nombre **w**, que "almacena" un vector-columna de longitud 6 (una matriz de dimensiones 6x1).

Otras órdenes para definir vectores son:

```
>> v1=a:h:b
```
define un vector-fila cuyas componentes van desde **a** hasta un número **c<b**, en incrementos de **h**

```
>> v2=linspace(a,b,n)
```
define un vector de longitud **n**, partición regular del intervalo **[a,b]**
(como **a:h:b**, con **h=(b-a/(n-1)**; la última componente es **=b**)

```
>> v'
```
es el vector traspuesto del vector **v** (ídem para matrices)

Las componentes de un vector se designan mediante el número de su subíndice:

```
v(1), w(4), v1(3)
```

ATENCIÓN: En MATLAB, los subindices de los vectores y matrices comienzan siempre por 1

Támbién se puede acceder a un bloque de componentes de un vector, indicando los subíndices mínimo y máximo, o indicando un subconjunto de índices

```
>> v(2:3)
>> w(1:4)
>> ii=[2,6,21,34]; wz(ii)
```

2.2 Operaciones con vectores y escalares

Si **v** es un vector (fila o columna) y **k** es un escalar, las operaciones siguientes dan el resultado que se indica:

v+k	vector de componentes $\{v_i+k\}$
v-k	vector de componentes $\{v_i-k\}$
k*v	vector de componentes $\{k*v_i\}$
v/k	vector de componentes $\{v_i/k\}$
k./v	vector de componentes $\{k/v_i\}$
v.^k	vector de componentes $\{(v_i)^k\}$
k.^v	vector de componentes $\{k^{(v_i)}\}$

2.3 Operaciones entre vectores

Si **v** y **w** son dos vectores (fila o columna) de las mismas dimensiones:

`>> v+w`	vector de componentes $\{v_i+w_i\}$
`>> v-w`	vector de componentes $\{v_i-w_i\}$
`>> v.*w`	vector de componentes $\{v_i*w_i\}$ (producto componente a componente)
`>> v./w`	vector de componentes $\{v_i/w_i\}$ (división componente a componente)
`>> v.^w`	vector de componentes $\{v_i^w_i\}$ (exponenc. componente a componente)
`>> v*w`	Si **v** es un vector-fila de dimension **n** y **w** es un vector columna de dimensión **n**, es el producto escalar de v y w

2.4 Funciones específicas de vectores

Las funciones matemáticas elementales admiten vectores como argumentos y se interpretan componente a componente.
Algunas funciones específicas para vectores son:

`sum(v)`	suma de las componentes del vector v
`prod(v)`	producto de las componentes del vector v
`dot(v,w)`	producto escalar de dos vectores del mismo tipo y las mismas dimensiones
`cross(v,w)`	producto vectorial de dos vectores del mismo tipo y dimensión 3
`max(v)`	máximo de las componentes del vector v (atención: sin valor absoluto)
`norm(v)`	norma euclídea del vector v
`norm(v,p)`	norma-p del vector v: sum(abs(v).^p)^(1/p)
`norm(v,inf)`	norma infinito del vector v

OBSERVACIÓN 2.4

Las funciones MATLAB pueden devolver un número variable de resultados. Cuando una función tiene, en su definición, más de un argumento de salida, puede ser utilizada de varias formas. La función **max** nos proporciona un ejemplo:
- si, simplemente, utilizamos la función en la forma:

```
>> max(v)
```

MATLAB nos devolverá el máximo valor de entre las componentes del vector **v**
- si la utilizamos en la forma:

```
>> [m,k]=max(v)
```

utilizando dos variables para almacenar la salida, en la variable **m** quedará almacenado el máximo de las componentes, y en la variable **k** se guardará el valor del subíndice de la componente que produce el máximo.

También pueden tener un número variable de argumentos de entrada: obsérvese que la función norm tiene distinto comportamiento según que la llamemos con un solo argumento (**norm(v)**) o con dos argumentos (**norm(v,p)**). Más adelante se explicará como escribir una función con esta característica.

EJEMPLO 2.4

Si **v** es un vector-fila (respectivamente columna) de componentes v_i, entonces **exp(v)** es otro vector fila (resp. columna) de componentes $\exp(v_i)$.

```
>> v=[1,2,4,-5,0,-1];
>> sum(v)
ans =
     1
>> max(v)
ans =
     4
>> [m,k]=max(abs(v))
m =
     5
k =
     4
>> sqrt(sum(v.^2))
ans =
    6.8557
```

2.5 Definición de matrices

Las matrices se definen de forma similar a los vectores, introduciendo sus filas como vectores-fila y separando unas filas de otras mediante punto y coma o saltos de línea.

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

Las componentes de una matriz se designan mediante los números de sus subíndices.
Un vector-fila de dimensión **n** es en realidad una matriz de dimensión **1xn**.
Un vector-columna de dimensión **m** es en realidad una matriz de dimensión **mx1**.

EJEMPLO 2.5

```
>> A=[1,2,3 ; 4,5,6 ; 7,8,9];
>> A(1,3)
ans =
     3
>> A(1,:)            % primera fila de A
ans =
     1     2     3
>> A(:,2)            % segunda columna de A
ans =
     2
     5
     8
>> A(1:2;2:3)        % submatriz de A
ans =
     2     3
     5     6
```

2.5 Operaciones con matrices

A+B	suma de matrices
A-B	diferencia de matrices
A*B	producto de matrices (habitual)
A^2	producto de la matriz A por si misma
A\B	A^{-1} B
A/B	A B^{-1}
A.*B	producto componente a componente (a_{ij} b_{ij})
A.^2	cuadrado componente a componente (a_{ij} ^2)
A./B	división componente a componente (a_{ij} / b_{ij})
A'	transpuesta de A (la adjunta si A es compleja)

Las matrices pueden también utilizarse como argumento de las funciones intrínsecas.

2.6 Funciones específicas para matrices

Algunas funciones específicas para manejo de matrices son:

diag(A)	vector conteniendo la diagonal principal de la matriz **A**
diag(A,k)	**k**-ésima sub o super diagonal de **A** (según sea **k**>0, **k**=0, **k**<0)
max(A)	Vector conteniendo el valor máximo de las componentes de cada columna. **[y,k]=max(A)** nos da, además, la fila en la que se produce el máximo de cada columna
zeros(n,m)	matriz **nxm** con todas sus componentes iguales a cero
zeros(n)	ídem **nxn**
ones(n,m)	matriz **nxm** con todas sus componentes iguales a uno
ones(n)	ídem **nxn**
eye(n,m)	matriz unidad: matriz **nxm** con diagonal principal =1 y el resto de las componentes =0
diag(v)	(donde **v** es un vector) matriz con la diagonal prinpicipal = **v** y ceros en el resto
diag(v,k)	(donde **v** es un vector) ídem con la **k**-ésima diagonal=**v** y ceros en el resto

(Obsérvese la diferencia de comportamiento de la función **diag**, en función de que el argumento de entrada sea una matriz o un vector).

EJEMPLO 2.6 (a)

```
>> diag([1,2,3,1])
ans =
     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     1
>> eye(3,3)+diag([-1,-1],1)
ans =
     1    -1     0
     0     1    -1
     0     0     1
```

También se pueden definir matrices por bloques:

`[A,B]`	es la matriz (A B)
`[A;B]`	es la matriz (A B) transpuesta (columna)
`[ ]`	representa la matriz "vacía" (0x0)
`A(:,3)=[]`	elimina la tercera columna de la matriz A
`A(1,:)=[]`	elimina la primera fila de A

EJEMPLO 2.6 (b)

```
>> A=diag([1,2,3,4]); A(:,3)=[];
>> A
A =
     1     0     0
     0     2     0
     0     0     0
     0     0     4
>> A(1,:)=[]
A =
     0     2     0
     0     0     0
     0     0     4
```

2.7 Matrices "sparse" (huecas)

MATLAB dispone de un sistema especial de almacenamiento y manipulación de matrices huecas.
La función

>> sparse(i,j,c,m,n)

donde:

i , **j** son vectores de subíndices, de la misma longitud
c es un vector de la misma longitud que los anteriores
n , **m** son números naturales,

genera una matriz hueca de dimensión **nxm**, cuyos únicos elementos no nulos son los de subíndices **(i(k),j(k))**, de valor **c(k)**

EJEMPLO 2.7 (a)

```
>> fil=[1,1,2,3,4];
>> col=[1,3,2,4,1];
>> val=[1,2,-1,3,4];
>> C=sparse(fil,col,val,4,4)
C =
   (1,1)    1
   (1,3)    2
   (2,2)   -1
   (3,4)    3
   (4,1)    4
```

C es la forma "hueca" de la matriz:

$$\begin{pmatrix} 1 & 0 & 2 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 3 \\ 4 & 0 & 0 & 0 \end{pmatrix}$$

full(A)	Si A es una matriz en forma hueca, devuelve su forma estándar
[i,j,c]=find(A)	Si A es una matriz (en forma hueca o no), devuelve los vectores i, j y c correspondientes a su almacenamiento sparse.
sparse(A)	Si A es una matriz en forma estándar, devuelve su forma esparse.

Las operaciones entre matrices pueden también realizarse entre matrices huecas.

EJEMPLO 2.7 (b)

Aquí C es la matriz del ejemplo anterior

```
>> CS=full(C)
CS =
      1      0      2      0
      0     -1      0      0
      0      0      0      3
      4      0      0      0
>> [i,j,c]=find(C)
i =
      1      1      2      3      4
j =
      1      3      2      4      1
c=
      1      2     -1      3      4
>> sparse(CS)
ans=
      (1,1)        1
      (1,3)        2
      (2,2)       -1
      (3,4)        3
      (4,1)        4
```

2.8 Polinomios

En MATLAB los polinomios se identifican con el vector-fila de sus coeficientes:

p=[3,5,0,1,2] representa el polinomio $3x^4 + 5x^3 + x + 2$

roots(p)	Calcula las raices del polinomio p (es un vector-columna y, en general, calcula aproximaciones)
poly(raices)	Si raices es un vector-columna, devuelve el polinomio que tienes dichas raices. Se obtiene normalizado y puede ser de coeficientes complejos
poly(A)	Si A es una matriz cuadrada, es el polinomio característico
polyval(p,x)	Calcula el valor del polinomio p en el punto x (x puede ser un vector
conv(p1,p2)	Producto de los polinomios p1 y p2
deconv(p1,p2)	División de polinomios
polyder(p)	Derivada del polinomio p

EJEMPLO 2.8

```
>> p=[3,5,0,1,2];polival(p,0)
ans =
      2
>> raices=roots(p)
raices =
     -1.6394
      0.3716 + 0.6243 i
      0.3716 - 0.6243 i
     -0.7704
>> poly(CS)  % CS es la del Ejemplo 2.7 (b)
ans=
      1      0     -1    -24    -24
>> polyder(p)
ans =
     12     15      0      1
```

2.9 Sistemas lineales

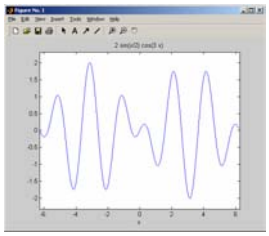
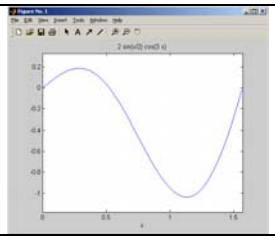
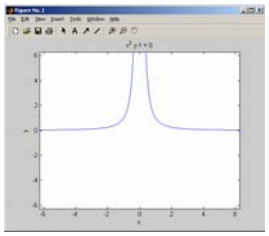
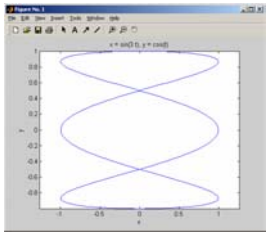
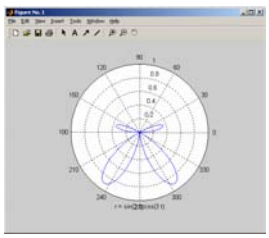
`det(A)`	Calcula el determinante de la matriz A
`inv(A)`	Calcula la inversa de la matriz cuadrada A
`rank(A)`	Calcula el rango de la matriz A
`cond(A)`	Número de condición de la matriz A: $\lambda_{max} / \lambda_{min}$ (Si es muy grande, la matriz está mal condicionada)
`A\b`	Calcula la solución del sistema lineal Ax=b. Previamente a los cálculos, MATLAB realiza un análisis de la matriz A para decidir cual es el método más adecuado: triangular, Cholesky, LU, QR, Gauss, etc., y además lleva a cabo un pre-ordenamiento si A es hueca.

3. REPRESENTACIÓN GRÁFICA DE FUNCIONES DEFINIDAS POR UNA FÓRMULA

Los comandos que se presentan en este apartado son funciones MATLAB "fáciles de usar" (easy-to-use) para representar gráficamente, de forma rápida, funciones definidas por una expresión matemática. Tienen sólo un pequeño número de parámetros que se pueden especificar. Todas ellas hacen uso de otras funciones MATLAB que disponen de un número superior de opciones y parámetros que podemos modificar. Cuando se necesite hacer gráficas de funciones que no vienen definidas por una fórmula (definidas a trozos, definidas a través de programas o por sus valores en un número finito de puntos, …) habrá que recurrir a dichas funciones más generales.

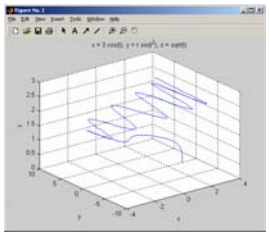
3.1 Curvas planas

El comando más fácil de usar de que dispone MATLAB para dibujar curvas planas definidas por una fórmula matemática (no por un conjunto de valores) es el comando **ezplot**, que puede ser usado de varias formas.

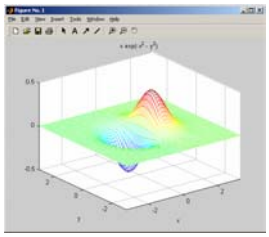
`ezplot(f)`	donde • **f** es una cadena de caracteres conteniendo la expresión de una función **y=f(x)** dibuja la función **y=f(x)** para x en el intervalo **[-2π,2π]** **Ejemplo:** `>> ezplot('2*sin(x/2)*cos(3*x)')`	
`ezplot(f,[a,b])`	lo mismo que la anterior para **x** variando en el intervalo **[a,b]** **Ejemplo:** `>> ezplot('2*sin(x/2)*cos(3*x)',[0,pi/2])`	
`ezplot(f)` `ezplot(f,[a,b])`	si **f** es una expresión de **(x,y)**, dibuja la curva implícitamente definida por **f(x,y)=0**, para **x** e **y** variando en el intervalo **[-2π,2π]** en el primer caso y para **x** e **y** variando en el intervalo **[a,b]** en el segundo caso. **Ejemplo:** `>> ezplot('x^2*y-1')`	
`ezplot(x,y)` `ezplot(x,y,[a,b])`	donde • **x** e **y** son dos cadenas de caracteres conteniendo las expresiones de dos funciones **x(t)** e **y(t)** dibuja la curva de ecuaciones paramétricas **x=x(t) y=y(t)** para **t** en el intervalo **[0,2π]**, en el primer caso y para **t** en el intervalo **[a,b]** en el segundo **Ejemplo:** `>> ezplot('sin(3*t)','cos(t)')`	
`ezpolar(f)` `ezpolar(f,[a,b])`	donde • **f** es una cadena de caracteres conteniendo la expresión de una función **f(θ)** dibuja la curva definida en coordenadas polares por **ρ= f(θ)** para θ variando en el intervalo **[0,2π]**, en el primer caso y en el intervalo **[a,b]** en el segundo **Ejemplo:** `>> ezpolar('sin(2*t)*cos(3*t)',[0,pi])`	

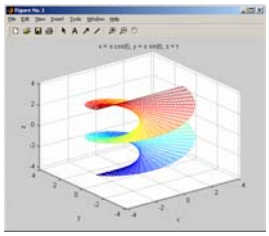
3.1 Curvas en el espacio

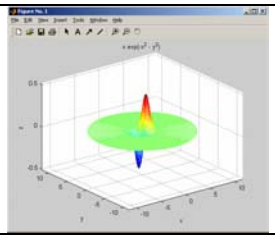
Para dibujar curvas en el espacio tridimensional, MATLAB dispone del comando **ezplot3**:

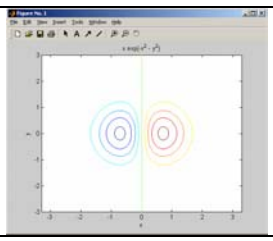
`ezplot3(x,y,z)` `ezplot3(x,y,z,[a,b])`	donde • **x** , **y** , **z** son tres cadenas de caracteres conteniendo las expresiones de tres funciones **x(t)** , **y(t)** , **z(t)** dibuja la curva de ecuaciones paramétricas **x=x(t) y=y(t) z=z(t)** para **t** en el intervalo **[0,2π]**, en el primer caso y para **t** en el intervalo **[a,b]** en el segundo **Ejemplo:** `>> ezplot3('3*cos(t)','t*sin(t^2)','sqrt(t)')`	

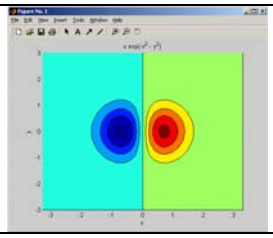
3.1 Superficies

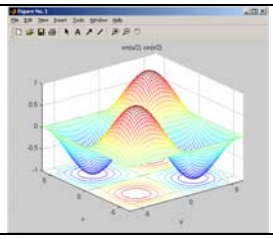
`ezmesh(f)`
`ezmesh(f,[a,b])`
`ezmesh(f,[a,b,c,d])`
	donde
	• **f** es una expresión de dos variables
	dibuja la superficie **z=f(x,y)** para **(x,y)** variando en el cuadrado **[-π,π]x[-π,π]** en el primer caso, en el cuadrado **[a,b]x[a,b]** en el segundo, y en el rectángulo **[a,b]x[c,d]** en el tercer caso. El método de dibujo es una malla con segmentos coloreados, en función de los valores en los extremos.
	Ejemplo:
	`>> ezmesh('x*exp(-x^2 - y^2)')`
		

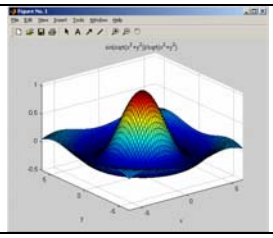
`ezmesh(x,y,z)`
`ezmesh(x,y,z,[a,b])`
`ezmesh(x,y,z,[a,b,c,d])`
	donde
	• **x**, **y**, **z** son expresiones de funciones de dos variables
	dibuja la superficie de coordenadas paramétricas **x=x(s,t) y=y(s,t) z=z(s,t)** para **(s,t)** variando en el cuadrado **[-2π,2π]x[-2π,2π]** en el primer caso, en el cuadrado **[a,b]x[a,b]** en el segundo, y en el rectángulo **[a,b]x[c,d]** en el tercer caso
	Ejemplo:
	`>> ezmesh('s*cos(t)','s*sin(t)','t',[-pi,pi])`
		

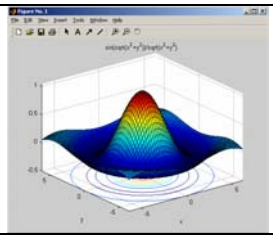
`ezmesh(... , 'circ')`	en cualquiera de los usos anteriores, dibuja la función correspondiente sobre un círculo centrado en el origen **Ejemplo:** `>> ezmesh('x*exp(-x^2 - y^2)','circ')`	

`ezcontour(f)`
`ezcontour(f,[a,b])`
`ezcontour(f,[a,b,c,d])`
	dibuja las líneas de nivel (isovalores) de la función z=f(x,y)
	Ejemplo:
	`>> ezcontour('x*exp(-x^2 - y^2)')`
		

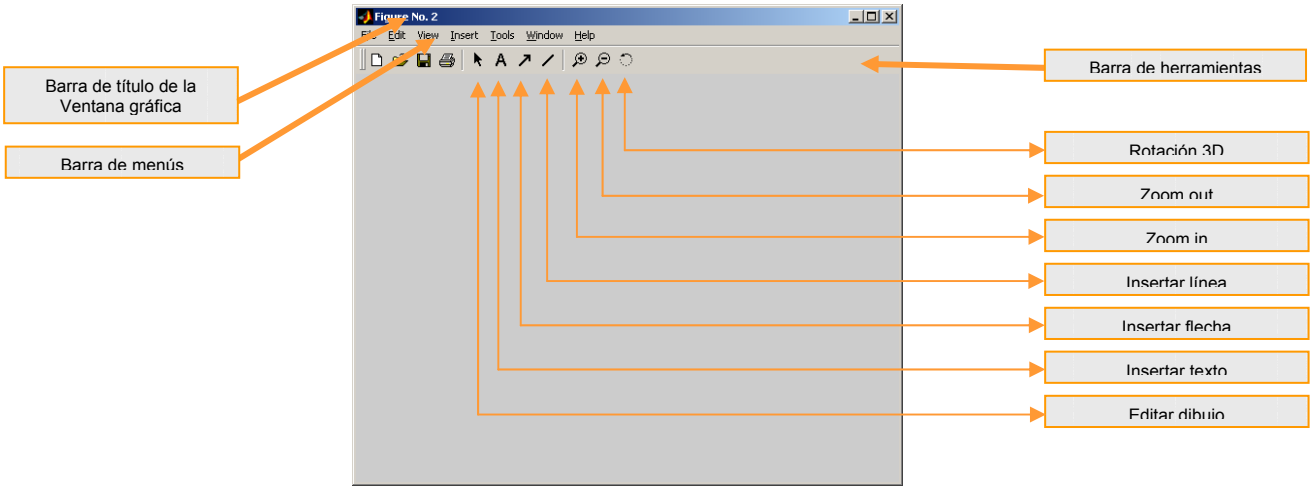
`ezcontourf(. . . )`	hace lo mismo que **ezcontour**, pero rellenando con un color sólido las distintas zonas determinadas por las líneas de nivel **Ejemplo:** `>> ezcontourf('x*exp(-x^2 - y^2)')`	

`ezmeshc(f)`	con los mismos argumentos que **ezmesh**, dibuja simultáneamente las líneas de nivel y la superficie **Ejemplo:** `>> ezmeshc('sin(u/2)*sin(v/2)')`	

`ezsurf(f)`	dibuja una superficie coloreada **z=f(x,y)**. Sus argumentos son como en **ezmesh** **Ejemplo:** `>> ezsurf('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)')`	

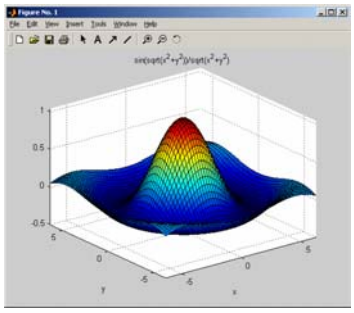
`ezsurfc(f)`	como la anterior pero, además, dibuja las líneas de nivel **Ejemplo:** `>>ezsurfc('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)')`	

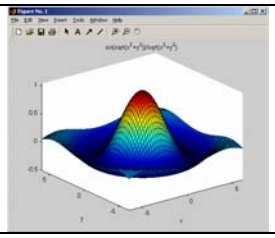
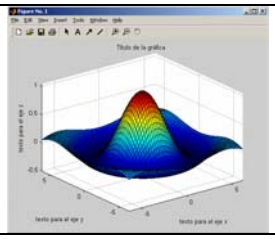
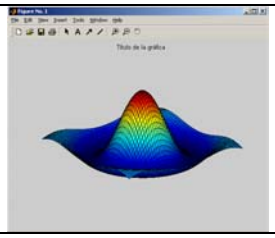
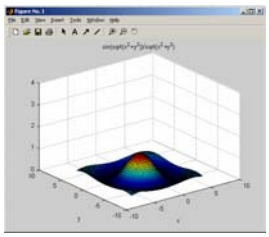
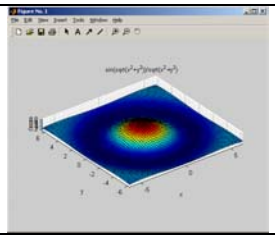
4. LA VENTANA GRÁFICA DE MATLAB



5. ALGUNOS COMANDOS GRÁFICOS AUXILIARES

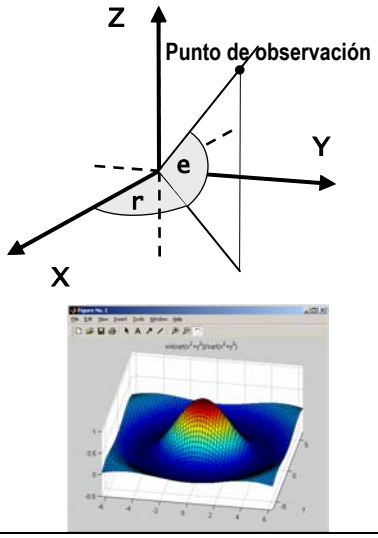
Vamos a ver algunos comandos que modifican el aspecto de un gráfico. Veremos su efecto sobre la gráfica. En estas notas no se exponen todas las posibilidades de estos comandos. Utilizando el **help** de MATLAB, se pueden ver el resto de las opciones.



grid on **grid off** pone – elimina el "enrejado" de los ejes	
xlabel('texto para el eje x') **ylabel('texto para el eje y')** **zlabel('texto para el eje z')** **title('Título de la gráfica')**	
axis on **axis off**	
axis([x1,x2,y1,y2,z1,z2]) Determina los límites de la gráfica. En gráficos bi-dimensionales no se incluyen **z1**, **z2.** **axis auto** impone los límites establecidos por defecto por MATLAB ([-10,10,-10,10,-0.5,1]) **Ejemplo:** **axis([-10,10,-10,10,0,4])**	
axis equal determina los mismos factores de escala para todos los ejes	

view(r,e)
en las gráficas tridimensionales permite cambiar el punto de observación del objeto representado.
Los valores por defecto son r=-37.5, e=30

```
>> view(10,50)
```

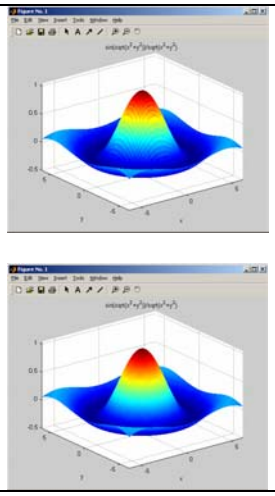


shading
determina la forma de utilizar los colores de las superficies coloreadas:

shading flat
la función color es constante a trozos (en cada segmento o panel)

shading interp
la función color es bilineal a trozos (en cada segmento o panel)

shading faceted (es la opción por defecto)
la función color es constante a trozos (como en flat), pero además se superpone el dibujo de la malla con los segmentos en negro

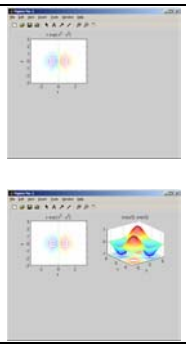


hold on . . . **hold off**	Todas las órdenes de dibujo entre ambos comandos se ejecutan sobre la misma ventana gráfica, sin borrar lo anterior.
figure **figure(h)**	Crea una nueva ventana gráfica, la activa y la trae al frente, delante de todas las ventanas abiertas. En el primer caso le asigna un número de forma automática. En el segundo caso le asigna el número h, es decir, el nombre "Figure No. h"
gcf	Devuelve el número de la ventana gráfica activa en ese momento.
shg	Trae la ventana gráfica activa al frente de todas.
clf	Borra la figura de la ventana gráfica activa. No cierra la ventana; sólo borra su contenido.
close **close(h)**	Cierra la ventana gráfica activa, en el primer caso, o la de número h, en el segundo.

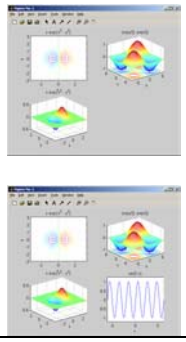
subplot(m,n,p)
Este comando permite dividir la ventana gráfica en una matriz **mxn** de sub-ventanas gráficas, activando para dibujar la **p**-ésima de ellas. Ver ejemplo siguiente:

```
>> subplot(2,2,1); ezcontour('x*exp(-x^2 - y^2)');
```

```
>> subplot(2,2,2); ezmeshc('sin(u/2)*sin(v/2)');
```

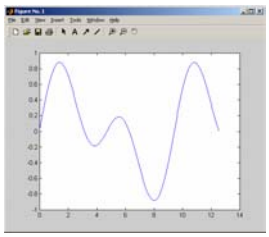


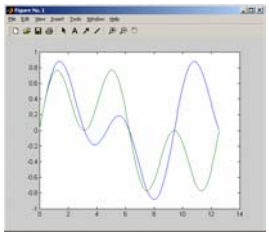
```
>> subplot(2,2,3); ezmesh('x*exp(-x^2 - y^2)');

>> subplot(2,2,4); ezplot('sin(3*x)');
```

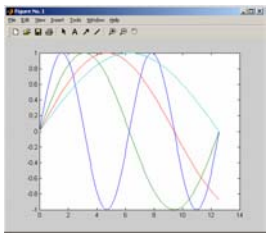


6. LOS COMANDOS BÁSICOS DE DIBUJO 2D Y 3D

Vemos aquí los comandos más completos de que dispone MATLAB para dibujar curvas planas y en el espacio, superficies, lineas de nivel, etc.

`plot(x,y)`	Si **x** e **y** son dos vectores de la misma dimensión, **n**, dibuja una curva (lineal a trozos) que pasa por los puntos (x_i , y_i), **i=1,… n**
	
	Ejemplo:
	
	`>> x=0:pi/100:4*pi; y=sin(x).*cos(x/3);`
	`>> plot(x,y)`
		

`plot(x1,y1,x2,y2)`	Dibuja las dos curvas $(x1_i , y1_i)$, **i=1,… n1** y $(x2_i , y2_i)$, **i=1,… n2** en la misma ventana y en los mismos ejes
	
	Ejemplo:
	
	`>> x=0:pi/100:4*pi;`
	`>> y=sin(x).*cos(x/3); z=sin(x).*cos(x/2);`
	`>> plot(x,y,x,z)`
		

`plot(x,A)`	donde x es un vector-columna (resp. fila) y A es una matriz, dibuja tantas curvas (x_i , A_{ij}), **i=1,… n** (como columnas (resp. filas) tenga la matriz A
	
	Ejemplo:
	
	`>> x=0:pi/100:4*pi;`
	`>> A=[sin(x);sin(x/2);sin(x/3);sin(x/4)];`
	`>> plot(x,A)`
		

El comando plot asigna, por defecto, determinados colores a las gráficas. Estos colores, así como el tipo de línea a utilizar para dibujar (continua, de puntos, con marcadores, etc.) se pueden modificar.

plot(x,y,param)

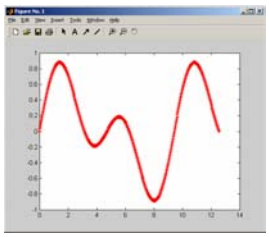
donde param es una cadena de caracteres como máximo, cada uno a elegir de una de las columnas siguientes:

b	amarillo	**.**	asteriscos	-	línea sólida
b	azul	**.**	puntos	-	línea sólida
g	verde	**o**	círculo	**:**	l. de puntos
r	rojo	**x**	aspas	**-.**	punto-guión
c	cyan	**+**	cruces	**--**	guiones
m	magenta	*****	asteriscos		
y	amarillos	**s**	square		
k	negro	**d**	diamantes		
		v	triángulos		

etc. (consultar help plot)

Ejemplo:

```
>> x=0:pi/100:4*pi; y=sin(x).*cos(x/3);
>> plot(x,y,'r*')
```



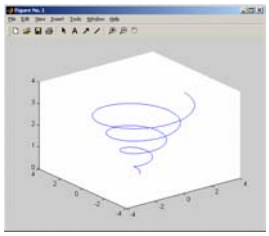
plot3(x,y,z)

plot3(x,y,z,param)

Si **x** , **y** , **z** son tres vectores de la misma dimensión, **n**, dibuja una curva tridimensional (lineal a trozos) que pasa por los puntos (x_i , y_i , z_i), **i=1,… n**

Ejemplo:

```
>> alpha=0:pi/80:8*pi; z=alpha/8;
>> x=z/(8*pi)+z.*cos(alpha);
>> y=z/(8*pi)+z.*sin(alpha);
>> plot3(x,y,z)
```



meshgrid(xp,yp)

mesh(x,y,z)

meshc(x,y,z)

Representa una superficie **z=f(x,y)** sobre una malla rectangular. Los argumentos **x** , **y** , **z** son matrices de la misma dimensión conteniendo, respectivamente, las coordenadas **x** , **y** , **z** de los nodos de la malla. Los segmentos de la malla se colorean según los valores de la función (coordenada **z**)

La función **meshc** hace lo mismo, pero dibujando además las líneas de nivel en el plano XY

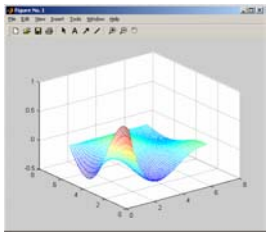
La funcion **meshgrid** sirve para construir la malla de base, en el plano XY:
Si **xp** es una partición del intervalo **[x0,x1]** e **yp** es una partición del intervalo **[y0,y1]**, entonces

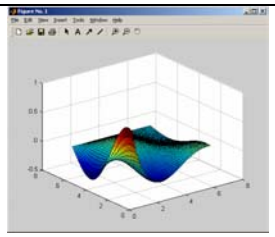
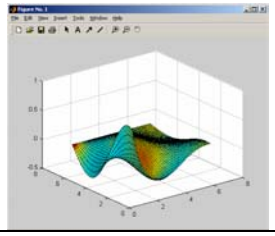
```
>> [x,y]=meshgrid(xp,yp)
```

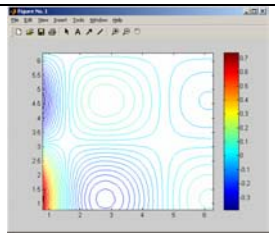
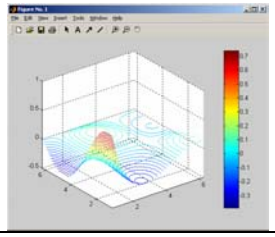
construye dos matrices, **x** e **y**, que definen una malla del rectángulo **[x0,x1]x[y0,y1]** . La matriz **x** contiene las coordenadas X de los puntos de la malla y la matriz **y** sus coordenadas Y

Ejemplo:

```
>> xp=linspace(pi/4,2*pi,50);
>> [x,y]=meshgrid(xp,xp);
>> z=(cos(x)./x).*(sin(y)./sqrt(y));
>> mesh(x,y,z)
```



`surf(x,y,z)`
`surfc(x,y,z)`
`surfl(x,y,z)`
	La función **surf** hace lo mismo que **mesh**, pero dibujando los segmentos de la malla en color negro y rellenando los "rectángulos" de la malla de color, según los valores de la función
	
	La función **surfc** hace lo mismo, pero dibujando además las líneas de nivel en el plano XY
	
	La funcion **surfl** hace lo mismo que surf, pero además añade una fuente de luz lateral
		
		 

`contour(x,y,z,n)` `contour3(x,y,z,n)`	La función **contour** dibuja las proyecciones sobre el plano XY de las líneas de nivel (isovalores) La función **contour3** dibuja las líneas de nivel sobre la misma superficie En ambos casos **n** es el número de líneas a dibujar. Se puede usar la función **colorbar** para añadir a la gráfica una barra con la graduación de colores y la correspondencia con los valores de la función representada. (Esta función puede ser usada con cualquier otra función gráfica que utilice colores).	 

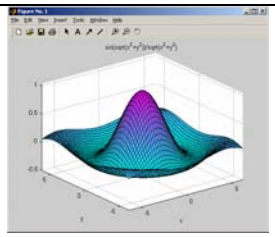
colormap(m)
permite cambiar el mapa de colores que se utiliza en la representación. En general, m es una matriz con tres columnas, de modo que la i-ésima fila determina las proporciones, en la escala RGB, del i-ésimo color utilizado.

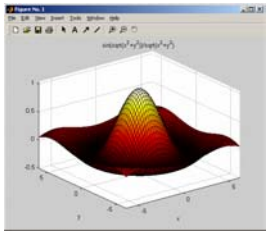
Para más comodidad, MATLAB dispone de una serie de mapas de colores pre-definidos, que se pueden imponer mediante

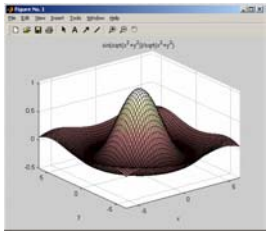
colormap(mapa)

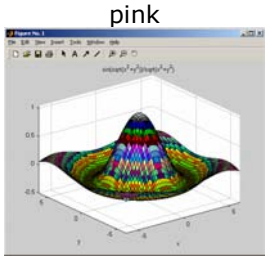
mapa puede tomar uno de los valores siguientes:

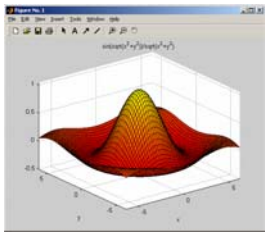
```
hsv         - la escala por defecto (hue-saturation-value).
cool        - Gama de colores entre cyan y magenta.
hot         - Gama de colores entre rojo oscuro y amarillo.
gray        - Gama de colores grises.
copper      - Gama de colores cobrizos.
pink        - Gama de colores rosados.
flag        - Alterna rojo – blanco – azul - negro.
colorcube   - Contraste de colores.
autumn      - Colores entre el rojo y el amarillo.
spring      - Colores entre el magenta y el amarillo.
winter      - Colores entre el azul y el verde.
summer      - Colores entre el verde y el amarillo.
. . .
para más mapas ver help graph3d
```

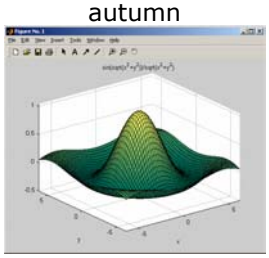

cool


hot


pink


colorcube


autumn


summer

7. PROGRAMAR CON MATLAB

Los programas o funciones de MATLAB se guardan en ficheros de nombre ****.m
Son ficheros ASCII (de texto) que contienen secuencias de comandos de MATLAB.

Para crear y modificar estos ficheros, la versión para Windows de MATLAB dispone de su propio **EDITOR** (aunque puede usarse cualquier otro editor ASCII).

Existen dos tipos de ficheros ****.m** :

- los **ficheros de comandos** o **scripts**
  Son ficheros que contienen un conjunto de comandos que se ejecutan sucesivamente cuando se teclea el nombre del fichero en la línea de comandos.

- las **funciones**
  Son ficheros con una estructura especial, que definen funciones análogas a las de MATLAB. Con carácter general, una función recibe unos **DATOS DE ENTRADA**, ejecuta unas **ÓRDENES** y produce unos **DATOS DE SALIDA**.

La primera línea de un fichero **nombre.m** que define una función debe tener la forma

```
function [variables de salida]=nombre(argumentos de entrada)
```

Los argumentos de entrada y las variables de salida, cuando hay varios, se separan por comas.
Puede no haber unos y otros.
Si no hay argumentos de entrada, no es necesario poner los paréntesis.
Si no hay variables de salida, no es necesario poner los corchetes ni el signo =
Si sólo hay una variable de salida, no es necesario poner los corchetes.

7.1. Sintaxis general

Las líneas que comienzan por el carácter **%** son líneas de comentario, es decir, son ignoradas al ejecutar la función o script.
Las primeras líneas de comentario de un fichero ****.m** pueden utilizarse para construir un help del programa.
Si una expresión es demasiado larga, se puede continuar en la línea siguiente, indicándolo por "tres puntos": **...**
En los scripts y funciones conviene terminar cada línea por "punto y coma", ya que sino aparecerá un exceso de información en el terminal.

7.2. Ejecución de las funciones y scripts

Las funciones y los scripts pueden ser **"llamados"** desde la línea de comandos y desde otro fichero ***.m**.

Para ejecutar un script llamado **program.m** basta con teclear su nombre **program** (sin el punto ni la m).
La ejecución del programa termina cuando se llega a su última sentencia ejecutable.
En cualquier punto del programa se puede forzar el fin de la ejecución mediante la instrucción **return**.

El número de argumentos de salida de una función no tiene que ser fijo. Puede depender de cómo se **"llame"** a la función (ver Ejemplo 2.1).

Una función llamada **fulanita.m** se llama como el resto de funciones MATLAB: por su nombre (sin el punto ni la m) y proporcionando los datos de entrada necesarios, a través de variables o de constantes.

Observación: en un fichero ***.m** que contenga una **función** pueden incluirse **sub-funciones**. Son funciones con nombre diferente del nombre del fichero (y por tanto de la función principal), y que sólo son "visibles" para las funciones del mismo fichero.

EJEMPLO 7.2

Crear un fichero con la función siguiente.
Ensayar lo que sucede cuando se utiliza el comando help y cuando se llama a la función con distinto número de argumentos de salida

```
>> help suma_positiv
>>suma_positiv(x)
>>sum=suma_positiv(x)
>>[sum,num]=suma_positiv(x)
```

```
function [suma,numero]=suma_positiv(x)
% Suma componentes positivos
%
% [suma,numero]=suma_positiv(x)
%
%  suma      es la suma de las componentes positivas
%            del vector x
%  numero    es el numero de componentes positivas de x
%
numero=0;
suma=0;
for i=1:length(x)
   if x(i)>0
      numero=numero+1;
      suma=suma+x(i);
   end;
end;
```

7.3. El entorno de trabajo de MATLAB

- **Directorio actual (Working directory)**
  Es el directorio en que, por defecto y en primer lugar, MATLAB va a buscar y crear cualquier fichero que necesite.

```
>> pwd              % Print Working Directory
                    % Dice cual es el directorio actual
>> dir              % DIRectory
                    % Dice el contenido del directorio actual
>> dir subsub       % Dice el contenido del subdirectorio subsub
>> ls               % LiSt directory
                    % Análogo a dir; funciona como el comando ls de unix
>> cd               % Change Directory
                    % Para cambiar el directorio actual. Necesita argumentos
>> cd ..            % Cambia al directorio "padre" del actual
>> cd subsub        % Cambia al subdirectorio "subsub" del actual
>> cd /a1/a2/a3     % Cambia al directorio indicado por el "path" dado
```

- **Camino de búsqueda (Path Browser)**
  Es la "lista" de directorios en los cuales MATLAB "buscará" los ficheros que le falten. Por defecto en dicha lista están todos los directorios necesarios de la instalación de MATLAB. Se pueden añadir directorios a la lista, con el comando `path` o usando la opcion "**Set Path...**" del menú **"File".**

- **Espacio de trabajo (Workspace)**
  Es el conjunto de variables que en un momento dado están definidas en la memoria del programa. Para obtener información se pueden usar los comandos `who` y `whos`. También se puede obtener en la ventana auxiliar correspondiente de MATLAB.

  Las variables creadas desde la linea de comandos de MATLAB pertenecen al **base workspace** (espacio de trabajo base). Los mismo sucede con las variables creadas por **scripts** que se ejecutan desde la linea de comandos. Estas variables permanecen en el **base workspace** cuando se termina la ejecuación del **script**.

  Sin embargo, las variables creadas por una **función** pertenecen al **espacio de trabajo de dicha función**, que es independiente del espacio de trabajo base. Es decir, las variables de las funciones son **LOCALES**.

  Para hacer que una variable de una funcion pertenezca al **base workspace**, hay que declararla **GLOBAL**: la orden

```
global a suma error
```

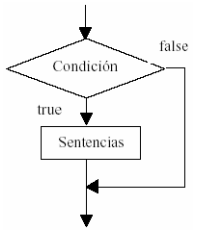
en una función hace que las variables **a** , **suma** y **error** pertenezcan al **base workspace.**

7.4. Decisiones y bucles

Las sentencias MATLAB para implementar las tomas de decisión (bifurcaciones) y los bucles (repeticiones) son las siguientes:

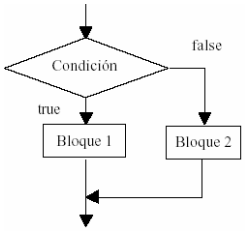
- **Sentencia IF**
  En su forma más simple se escribe en la forma siguiente:

```
if condicion
   sentencias
end
```



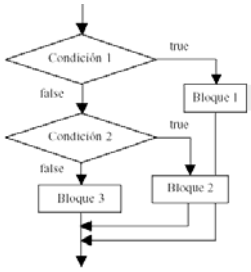
También se puede utilizar en la forma:

```
if condicion
   sentencias-1
else
   sentencias-2
end
```



Su forma más compleja es:

```
if condicion-1
   sentencias-1
elseif condicion-2
   sentencias-2
elseif condicion-3
   sentencias-3
else
   sentencias-4
end
```



En estas instrucciones, **condicion** es una expresión con valor lógico (true o false). Si el valor de la expresión es una matriz, sólo se ejecuta el conjunto de sentencias si todos los elementos son **true** (o, igualmente, distintos de cero).

- **Sentencia SWITCH**
  Es análoga a un conjunto de if ... elseif concatenados. Su forma general es:

```
switch expresion
   case caso-1
      sentencias-1
   case caso-2
      sentencias-2
   case {caso-3, caso-4,caso-5,...}
      sentencias-3
   otherwise
      sentencias-4
end
```

Aquí, expresion debe tener un valor numérico o de cadena de caracteres.

EJEMPLO 7.4

Ejemplo tonto de uso de SWITCH. Ver más adelante el uso de las instrucciones input y disp

```
resp=input('escribe un numero del 0 al 9 :: ');
switch resp
case 0
    disp('Has escrito 0 (cero)')
case 1
    disp('Has escrito 1')
case {2,3,4,5}
    disp('Has escrito un numero entre 2 y 5')
otherwise
    disp('has escrito un numero mayor que 5')
```

- **Sentencia FOR**
  Repite un conjunto de sentencias un numero predeterminado de veces. Su forma general es la siguiente:

```
for variable=vector-fila
   sentencias
end
```

  Se ejecuta el conjunto de **sentencias** tantas veces como elementos tenga el **vector-fila**, y en cada repetición, la **variable** toma el valor del elemento correspondiente. Por ejemplo, se puede usar en la forma:

```
for i=1:n
   sentencias
end
```

  que ejecutará el conjunto de sentencias **n** veces, para **i=1,2,3,...,n.**
  En la forma:

```
for i=n:-1:1
   sentencias
end
```

  se ejecuta el conjunto de sentencias también **n** veces, pero en orden inverso: **i=n,n-1,...,1.**

- **Sentencia WHILE**
  Repite la ejecución de un conjunto de sentencias mientras que una determinada condición se verifique. Su forma general es:

```
while condicion
   sentencias
end
```

  Se ejecutará el bloque de sentencias mientras que **condicion** tome el valor **true**, o mientras que **condicion** sea distinto de cero.

- **Sentencia BREAK**
  Detiene la ejecuación de un bucle **for** o **while**. Si hay bucles anidados, se detiene la ejecución del más interno.

7.5. Operadores de relación y lógicos

- **Operadores de relación o comparación**
  Los siguientes operadores producen como resultado un valor lógico:

```
true   (cualquier valor distinto de cero)

false  (cero)
```

  Cuando estos operadores se utilizan para comparar dos matrices de las mismas dimensiones, el resultado es otra matriz de la misma dimensión y la comparación se realiza elemento a elemento. Cuando se utilizan para comparar un escalar con una matriz, el resultado es una matriz, y se compara el escalar con cada uno de los elementos.

```
==     Igual a

~=     No igual a

<      Menor que

>      Mayor que

<=     Menor o igual que

>=     Mayor o igual que
```

- **Operadores lógicos**
  Son operadores que actuan entre valores lógicos:

  `&` Y lógico
  A&B produce los resultados que se reflejan en la tabla
  A&B es equivalente a **and(A,B)**

  `|` O lógico
  A|B produce los resultados que se reflejan en la tabla
  A|B es equivalente a **or(A,B)**

  `~` Negación lógica
  ~A produce los resultados que se reflejan en la tabla
  ~A es equivalente a **not(A)**

&	A	
	T	F
B T	T	F
B F	F	F

|	A	
	T	F
B T	T	T
B F	T	F

A	T	F
NOT(A)	F	T

7.6. Instrucciones sencillas de entrada/salida

- **Función INPUT**
  Permite imprimir un mensaje en la linea de comandos y leer datos desde el teclado. La instrucción

```
>> resp=input('Mensaje que se imprime')
```

  imprime **Mensaje que se imprime** en una linea de la ventana de comandos y se queda esperando a que el usuario teclee un valor o una expresión.
  Si se teclea un valor (escalar o matricial), MATLAB lo almacena en la variable **resp.**
  Si se teclea una expresión, MATLAB la evalua con los valores actuales de las variables del Workspace, y el resultado se almacena en la variable **resp.**
  Si en el mensaje se incluye el comando **\n**, se produce un salto de linea.

- **Función DISP**
  Permite imprimer en la pantalla un mensaje de texto:

```
>> disp('Mensaje que se imprime')
>> disp(['el valor de z es: ',num2str(8)])
```

  También permite imprimir el valor de una variable, sin imprimir su nombre:

```
>> a=[1,2;3,4];
>> disp(a)
     1     2
     3     4
```

8. RESOLUCIÓN NUMÉRICA DE ECUACIONES DIFERENCIALES

MATLAB dispone de varias funciones para resolver numéricamente Problemas de Valor Inicial para Ecuaciones Diferenciales Ordinarias:

$$\begin{cases} y' = f(t,y) & \text{en} \quad [t_0, t_f] \\ y(t_0) = y_0 \end{cases}$$

ode45 **ode23**	están basadas en algoritmos explícitos de tipo Runge-Kutta
ode113	implementa un algoritmo PECE (predictor-corrector) de orden variable.
ode15s	es un método de multipaso y orden variable basado en fórmulas de diferenciación numérica.
ode23s	está basado en una fórmula modificada de Rosenbrock de orden 2.
ode23t	es una implementación de la regla del trapecio usando un interpolante "libre".
ode23tb	es una implementación de TR-BDF2, una fórmula de Runge-Kutta implícita.

Para la mayoría de los casos será suficiente con utilizar **ode45** o bien **ode23**. Para casos difíciles, ver la documentación de MATLAB.

La utilización de todas estas funciones **ode**** es similar. Lo indicado a continuación para **ode23** es válido para cualquier otra

```
>> ode23(funcion,[t0,tf],y0)
```

funcion es el nombre de una función que evalúa el segundo miembro de la ecuacion, **f(t,y)**
Puede ser - un objeto **inline** o bien
- una referencia a una m-funcion fun.m: **@fun**

[t0,tf] es el intervalo en el que se quiere calcular la solución

y0 es el valor de la condición inicial

Usada de esta forma, la función **ode23** calcula la solución numérica y la dibuja

```
>> [t,y]=ode23(funcion,[t0,tf],y0)
```

Usada en esta forma, la función **ode23**, calcula (numéricamente) la solución del problema, devolviendo dos vectores, **t** e **y**, que proporcionan la aproximación de la solución del problema:

$$y(k) \approx \varphi(t(k)),\ k = 1, 2, \ldots \text{length}(y)$$

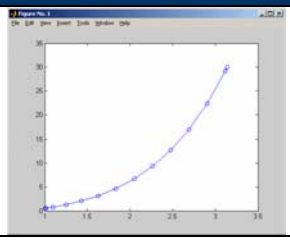
```
nombre_funcion = inline('expresion de la funcion')
nombre_funcion = inline('expresion de la funcion', 'var1', 'var2', …)
nombre_funcion = inline('expresion de la funcion', N)
```

Construye la "expresion de una función" y le asigna el nombre **nombre_funcion**
En el primer caso, los argumentos son determinados automáticamente.
En el segundo caso, los argumentos de la funcion son especificados: **f(var1,var2,...)**
En el tercer caso, se construye una función cuyos argumentos son: **x, P1,...,PN**

EJEMPLO

Calcular la solución de $\begin{cases} y' = 2t + y & \text{en } [1,\pi] \\ y(1) = 0.5 \end{cases}$

```
>> f=inline('2*t+y','t','y');
>> ode23(f,[1,pi],0.5);
```



EJEMPLOS

Calcular la solución de $\begin{cases} y' = 2t + y & \text{en } [1,\pi] \\ y(1) = 0.5 \end{cases}$

```
>> f=inline('2*t+y','t','y');
>> [t,y]=ode23(f,[1,pi],0.5)
t=  .....
y=  .....
```

Para conocer el valor (aprox.) de la solución en los puntos intermedios del intervalo [t0,tf], basta interpolar los valores

$$(t(k), y(k)), \quad k = 1, 2, \ldots \text{length}(y)$$

La gráfica se puede dibujar con

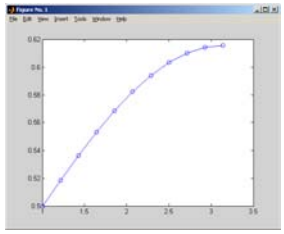
```
>> plot(t,y);
```

Calcular la solución de $\begin{cases} y' = 0.2\cos(\frac{t}{2})y & \text{en } [1,\pi] \\ y(1) = 0.5 \end{cases}$

```
>> ode23(@mifun,[1,pi],0.5);
```

donde mifun.m esla m-función:

```
function [dydt]=mifun(t,y)
dydt=0.2*cos(t/2)*y;
```



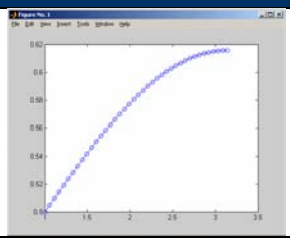
```
>> ode23(funcion,[t0,t1,t2,...,tf],y0)
```

Si se desea **expresamente** conocer los valores de la solución en unos puntos determinados, **[t0,t1,t2,...,tf],** se puede usar la orden **ode**** en la forma anterior.

EJEMPLO

Calcular la solución de $\begin{cases} y' = 0.2\cos(\frac{t}{2})y & \text{en } [1,\pi] \\ y(1) = 0.5 \end{cases}$

```
>> tin=linspace(1,pi,40);
>> f=inline('0.2*cos(t/2)*y','t','y');
>> ode23(f,tin,0.5);
```



```
>> ode23(@fun,[t0,tf],y0,options,p1,p2,...)
```

Las funciones **ode**** se pueden utilizar, también, en esta forma, donde:

options permite dar valores a una serie de parámetros que intervienen en el cálculo. Ver documentación para más detalles. Si no se necesita, poner una matriz vacía: **[]** en su lugar.

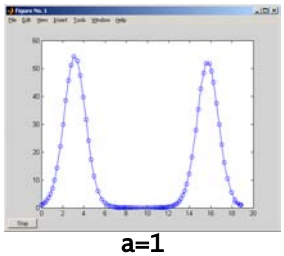
p1,p2,... son parámetros que serán pasados como argumentos a la m-función **fun.m** cuando sea llamada.

EJEMPLO

Escribir un programa que permita resolver el problema siguiente para distintos valores del parámetro a

$$\begin{cases} y' = a\cos(\frac{t}{2})y & \text{en } [0, 6\pi] \\ y(0) = 1 \end{cases}$$

```
function prog1(a)
ode23(@mifun,[0,6*pi],1,[],a)
%
function [dydt]=mifun(t,y,a)
dydt=a*cos(t/2)*y;
```



a=1

Resolución de sistemas diferenciales

Se resuelven exactamente igual, teniendo en cuenta que, en este caso, la función del segundo miembro y la condición inicial toman valores vectoriales.

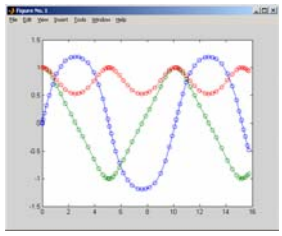
EJEMPLO

Resolver

$$\begin{cases} y_1' = y_2 y_3 \\ y_2' = -0.7\, y_1 y_3 \\ y_3' = -0.51\, y_1 y_2 \\ y_1(0) = 0 \\ y_2(0) = 1 \\ y_3(0) = 1 \end{cases} \qquad Y' = F(t,Y) = \begin{pmatrix} y_2 y_3 \\ -0.7\, y_1 y_3 \\ 0.51\, y_1 y_2 \end{pmatrix}$$

$$Y(0) = Y_0 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

```
>> f=inline('[y(2)*y(3);-0.7*y(1)*y(3); ...
            -0.51*y(1)*y(2)]','t','y');
>> ode23(f,[0,5*pi],[0;1;1])
```



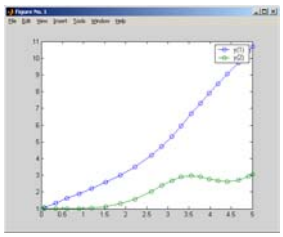
EJEMPLO

Resolver el Pb. de segundo orden

$$\begin{cases} y'' = \frac{1}{y'} - \text{sen}(y) \\ y(0) = 1 \\ y'(0) = 1 \end{cases}$$

$$Z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} y \\ y' \end{pmatrix} \qquad \begin{cases} Z' = F(t,Z) = \begin{pmatrix} z_2 \\ \frac{1}{z_2} - \text{sen}(z_1) \end{pmatrix} \\ Z(0) = Z_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \end{cases}$$

```
>> f=inline('[y(2);1/y(2)-sin(y(1))]','t','y')
>> ode23(f,[0,5],[1;1])
```



9. RAÍCES DE ECUACIONES, CÁLCULO DE MÍNIMOS Y DE INTEGRALES DEFINIDAS

9.1. Raíces de ecuaciones

La búsqueda de soluciones de ecuaciones escalares no lineales **f(x)=0** se hace, en MATLAB, mediante la función

```
>> fzero(funcion,x0)
```

funcion es el nombre de una función que evalúa la función **f(x)**
Puede ser - un objeto **inline** o bien
- una referencia a una m-funcion fun.m: **@fun**

x0 es un valor "próximo" al cero que se busca

```
>> [x,fval]=fzero(funcion,x0)
```

devuelve la raíz encontrada, **x**, y el valor de la función en **x** (si el algoritmo no converge, **x=NaN**)

```
>> fzero(funcion,[a,b])
```

[a,b] tiene que ser un intervalo donde la función cambia de signo. Tiene que ser

$$\text{signo}(f(a)) \neq \text{signo}(f(b))$$

```
>> fzero(@fun,x0,options,p1,p2,. . .)
```

options permite dar valores a una serie de parámetros que intervienen en el cálculo. Ver documentación de MATLAB para más detalles. Si no se necesita, poner una matriz vacía: **[]** en su lugar.

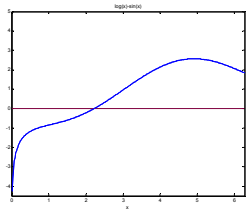
p1,p2,... son parámetros que serán pasados como argumentos a la m-función **fun.m** cuando sea llamada.

EJEMPLO

Resolver

$$\ln(x) - \text{sen}(x) = 0$$

```
>> f=inline('log(x)-sin(x)');
>> x=fzero(f,pi)
x = 2.2191
```



EJEMPLO

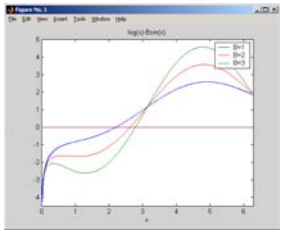
Resolver

$$\ln(x) - \beta\,\text{sen}(x) = 0$$

donde β es un parámetro, desconocido "a priori"

```
>> f=inline('log(x)-sin(x)');
>> x=fzero(f,pi)
x = 2.2191
```

```
function prog2(beta)
fzero(@fun,pi,[],beta)
%
function z=fun(t,param)
z=log(t)-param*sin(t);
```



9.2. Búsqueda de mínimos

El cálculo de mínimos relativos de funciones no lineales $f : \mathbf{R}^N \longrightarrow \mathbf{R}$ se hace, con MATLAB, mediante la función:

`>> fminsearch(funcion,x0)`
funcion es el nombre de una función que evalúa la función **f(x)** Puede ser - un objeto **inline** o bien - una referencia a una m-funcion fun.m: **@fun**
x0 es un valor "próximo" al mínimo que se busca

`>> [x,fvalor]=fminsearch(funcion,x0)`
devuelve el minimizador encontrado, **x**, y el valor de la función en **x** (si el algoritmo no converge, **x=NaN**)

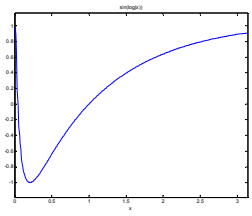
`>> fminsearch(@fun,x0,options,p1,p2,...)`
options permite dar valores a una serie de parámetros que intervienen en el cálculo. Ver documentación de MATLAB para más detalles. Si no se necesita, poner una matriz vacía: **[]** en su lugar.
p1,p2,... son parámetros que serán pasados como argumentos a la m-función **fun.m** cuando sea llamada

EJEMPLO

Hallar un mínimo relativo de la función

$$f(x) = \text{sen}(\ln(x))$$

```
>> f=inline('sin(log(x))')
>> fminsearch(f,1)
x =
       0.2079
```

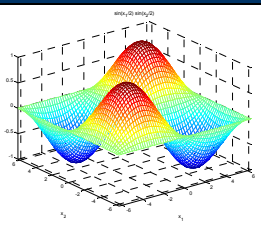


EJEMPLO

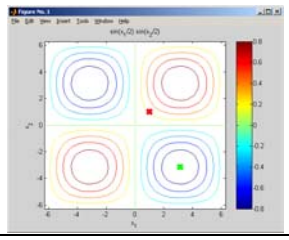
Hallar un mínimo relativo de la función

$$f : \mathbf{R}^2 \longrightarrow \mathbf{R}$$

$$f(\vec{x}) = \text{sen}(\frac{x_1}{2})\,\text{sen}(\frac{x_2}{2})$$



```
>> f=inline('sin(x(1)/2)*sin(x(2)/2)','x')
>> fminsearch(f,[1,1])
ans =
       3.1416   -3.1416
```



La búsqueda de mínimos absolutos en un intervalo acotado de funciones escalares se hace mediante la función

```
>> fminbnd(funcion,x1,x2)
```

funcion es el nombre de una función que evalúa la función **f(x)**
Puede ser - un objeto **inline** o bien
- una referencia a una m-funcion fun.m: **@fun**

x1, x2 son los extremos del intervalo

```
>> [x,fvalor]=fminbnd(funcion,x1,x2)
```

devuelve el minimizador,**x**, y el valor de la función en **x**

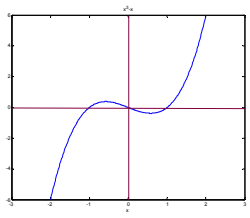
```
>> fminbnd(@fun,x1,x2,options,p1,p2,...)
```

mismo significado que en los casos anteriores

EJEMPLO

$$\min_{x\in[-2,2]} f(x) = x^3 - x$$

```
>> f=inline('x^3-x')
>> fminbnd(f,-2,2)
ans =
      -2
```



9.3. Cálculo de integrales definidas

El cálculo numérico de integrales definidas de funciones de una variable real

$$\int_a^b f(x)\,dx$$

se hace en MATLAB mediante la función:

```
>> quad(fun,a,b)
```

funcion es el nombre de una función que evalúa la función **f(x)**
Puede ser - un objeto **inline** o bien
- una referencia a una m-funcion fun.m: **@fun**
Debe aceptar un argumento vectorial, **x**, y devolver un argumento vectorial: el integrando evaluado en cada elemento de **x**

```
>> quad(@fun,a,b,[],[],p1,p2,..)
```

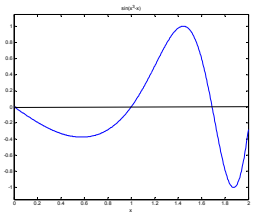
permite pasar **p1,p2,...** como argumentos a la función: **fun(x,p1,p2,...)**

EJEMPLO

Calcular

$$\int_0^2 sen(x^3 - x)\,dx$$

```
>> f=inline('sin(x.^3-x)')
>> quad(f,0,2)
ans =
       -0.0445
```

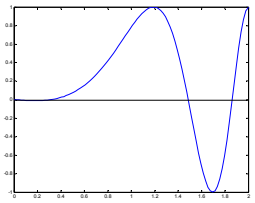


EJEMPLO

Calcular

$$\int_0^2 sen(x^3 - \frac{x}{\alpha})\,dx$$

```
function prog5(a)
quad(@fun,0,2,[],[],a)
%
function [z]=fun(t,alpha)
z=sin(t.^3-(t/alpha));
```



α=10