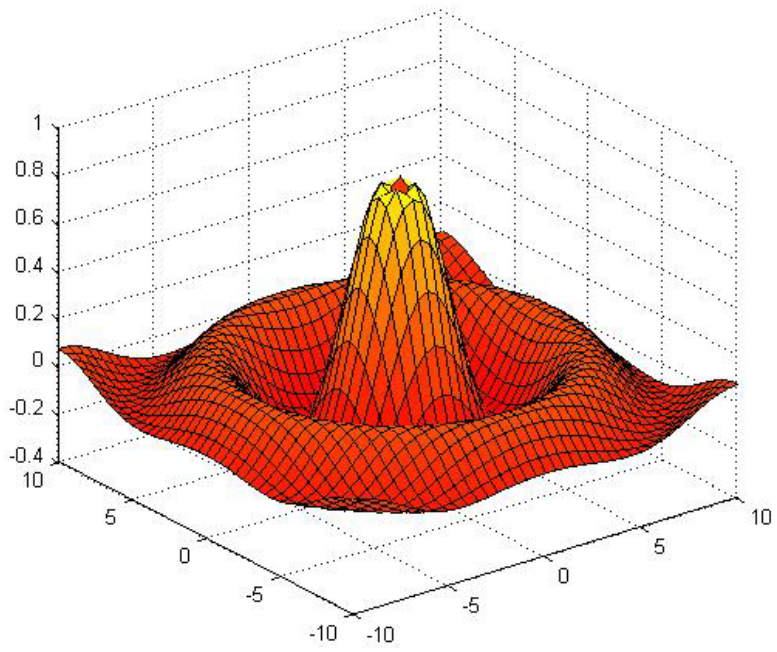
MANUAL BÁSICO

DE MATLAB



Mª Cristina Casado Fernández
Servicios Informáticos U.C.M
Apoyo a Investigación y Docencia

ÍNDICE

INTRODUCCIÓN

MATLAB es el nombre abreviado de "MATriz LABoratory". Es un programa para realizar cálculos numéricos con vectores y matrices, y por tanto se puede trabajar también con números escalares (tanto reales como complejos), con cadenas de caracteres y con otras estructuras de información más complejas.

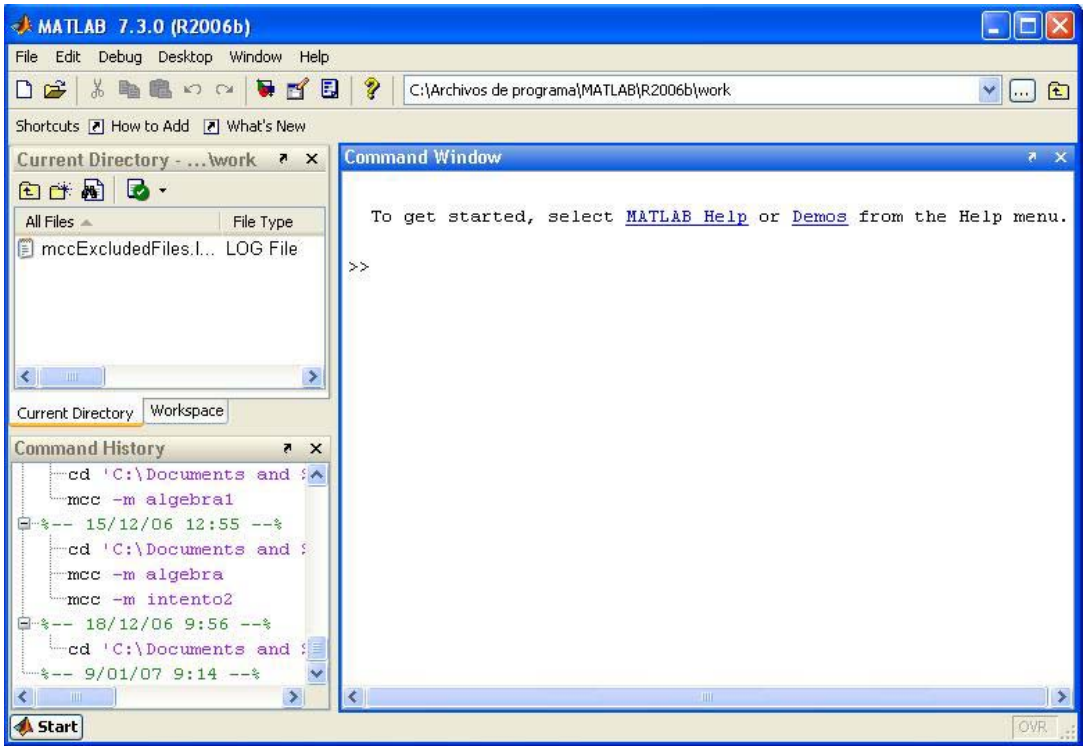
Matlab es un lenguaje de alto rendimiento para cálculos técnicos, es al mismo tiempo un entorno y un lenguaje de programación. Uno de sus puntos fuertes es que permite construir nuestras propias herramientas reutilizables. Podemos crear fácilmente nuestras propias funciones y programas especiales (conocidos como M-archivos) en código Matlab, los podemos agrupar en Toolbox (también llamadas librerías): colección especializada de M-archivos para trabajar en clases particulares de problemas.

Matlab, a parte del cálculo matricial y álgebra lineal, también puede manejar polinomios, funciones, ecuaciones diferenciales ordinarias, gráficos …

CARACTERÍSTICAS BÁSICAS

EL ESPACIO DE TRABAJO DE MATLAB

Nada más abrir Matlab (podemos hacerlo pinchando en el icono que aparece en el escritorio o en su defecto en Inicio->Todos los programas) aparecerá una pantalla como la siguiente:

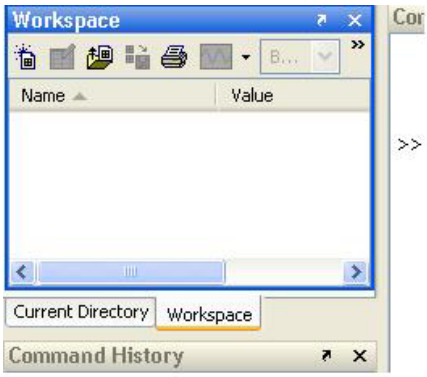


Todas las sentencias que vamos a utilizar las escribiremos en la ventana *Command Window* (ventana de comandos). Es la ventana de mayor tamaño.

Si queremos información acerca de las variables que estamos utilizando en Matlab podemos verlas en la ventana *Workspace* (espacio de trabajo) o usar:

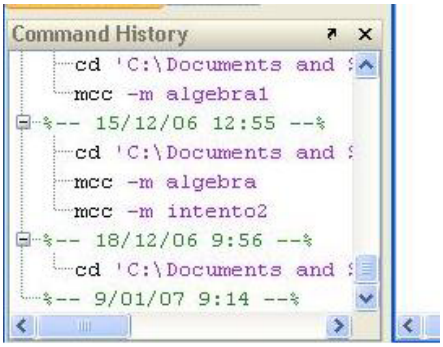
who para obtener la lista de las variables (no de sus valores)
whos para obtener la lista de las variables e información del tamaño, tipo y atributos (tampoco da valores)

Para ver esta ventana tenemos que pinchar en la pestaña que tienen este nombre. Está en la parte superior izquierda:



Si lo que queremos es conocer el valor que tiene una variable lo hacemos escribiendo el nombre de la variable y pulsando **Intro**.

Para recordar órdenes previas usamos las flechas del teclado ↑ y ↓. También podemos verlas en la ventana *Command History*, ventana situada en la parte inferior izquierda:



MATEMÁTICA SENCILLA

Matlab ofrece la posibilidad de realizar las siguientes operaciones básicas:

Operación	Símbolo	Expresión en Matlab
suma	+	a + b
resta	-	a - b
multiplicación	*	a * b
división	/	a / b
potencia	^	a ^ b

El orden de precedencia es:

Orden de precedencia de operaciones	
1º	^
2º	* /
3º	+ -

Matlab no tiene en cuenta los espacios.

Si queremos que Matlab evalúe la línea pero que no escriba la respuesta, basta escribir punto y coma (**;**) al final de la sentencia.

Si la sentencia es demasiado larga para que quepa en una sola línea podemos poner tres puntos (**…**) seguido de la tecla **Intro** para indicar que continúa en la línea siguiente.

Ejemplos:

```
>> a = 7        % damos valor a la variable a y la escribe por pantalla
a =
     7

>> b = 4;       % no escribe el valor de b por el ; del final

>> a + b        % realiza la suma de dos variables y guarda la solución en la variable ans
ans =
    11

>> a / b
ans =
    1.7500

>> a ^ b
ans =
        2401

>> 5 * a
ans =
    35

>> who                % da una lista de los nombres de las variables usadas

Your variables are:
a    ans  b

>> whos               % da una lista de las variables usadas más completa que la anterior

  Name      Size        Bytes  Class     Attributes
    a         1x1           8  double
  ans         1x1           8  double
    b         1x1           8  double
```

ALMACENAR Y RECUPERAR DATOS

Matlab permite guardar y cargar datos de los archivos del computador. En el menú **File**, la opción **Save Workspace as…** guarda todas las variables actuales y **Import Data…** carga variables de un espacio de trabajo guardado previamente.

Otra forma sería guardar el estado de una sesión de trabajo con el comando **save** antes de salir:

>> save

Al teclear esto, automáticamente se crea un fichero llamado *matlab.mat*. Puede recuperarse la siguiente vez que se arranque el programa con el comando **load**:

>> load

FORMATOS DE VISUALIZACIÓN DE NÚMEROS

Matlab no cambia la representación interna de un número cuando se escogen distintos formatos, sólo se modifica la forma de visualizarlo.

Tipo	Resultado	Ejemplo: >> pi
format short	Formato coma fija con 4 dígitos después de la coma (es el formato que viene por defecto)	3.1416
format long	Formato coma fija con 14 o 15 dígitos después de la coma	3.14159265358979
format short e	Formato coma flotante con 4 dígitos después de la coma	3.1416e+000
format long e	Formato coma flotante con 14 o 15 dígitos después de la coma	3.141592653589793e+000
format short g	La mejor entre coma fija o flotante con 4 dígitos después de la coma	3.1416
format long g	La mejor entre coma fija o flotante con 14 o 15 dígitos después de la coma	3.14159265358979
format short eng	Notación científica con 4 dígitos después de la coma y un exponente de 3	3.1416e+000
format long eng	Notación científica con 16 dígitos significantes y un exponente de 3	3.14159265358979e+000
format bank	Formato coma fija con 2 dígitos después de la coma	3.14
format hex	Hexadecimal	400921fb54442d18
format rat	Aproximación racional	355/113
format +	Positivo, negativo o espacio en blanco	+

ACERCA DE LAS VARIABLES

Matlab almacena el último resultado obtenido en la variable **ans**.

Las variables son sensibles a las mayúsculas, deben comenzar siempre con una letra, no pueden contener espacios en blanco y pueden nombrarse hasta con 63 caracteres (en versiones anteriores no permitía tantos caracteres). Si se nombra una variable con más de 63 caracteres truncará el nombre de dicha variable.

Algunas variables especiales de Matlab:

Variable	Definición	Valor
ans	Variable usada por defecto para almacenar el último resultado	? ? ?
pi	Razón de una circunferencia a su diámetro	3.1416
eps	Número más pequeño, tal que cuando se le suma 1, crea un número en coma flotante en el computador mayor que 1	2.2204e-016
inf	Infinito	Inf
nan	Magnitud no numérica	NaN
i y j	$i = j = \sqrt{-1}$	0 + 1.0000i
realmin	El número real positivo más pequeño que es utilizable	2.2251e-308
realmax	El número real positivo más grande que es utilizable	1.7977e+308

Tecleando **clear** podemos borrar todas las variables del espacio de trabajo, pero no borra lo de las demás ventanas, es decir, no desaparece lo que hay escrito en la ventana de comandos.

Tecleando **clc** borramos lo que hay en la ventana de comandos pero no borra las variables de la memoria del espacio de trabajo.

Algunos comandos de Matlab nos facilitan información sobre la fecha, como **clock**, **date** o **calendar**.

```
>> clock                    % año mes día hora minutos  y segundos, en este orden
ans =
  1.0e+003 *
    2.0060    0.0110    0.0140    0.0120    0.0190    0.0437

>> date                     % día-mes-año
ans =
14-Nov-2006

>> calendar                 % mes actual
             Nov 2006
    S    M   Tu    W   Th    F    S
    0    0    0    1    2    3    4
    5    6    7    8    9   10   11
   12   13   14   15   16   17   18
   19   20   21   22   23   24   25
   26   27   28   29   30    0    0
    0    0    0    0    0    0    0
```

OTRAS CARACTERÍSTICAS BÁSICAS

Los comentarios se escriben después del símbolo de tanto por ciento (**%**), de este modo todo lo que se escriba a continuación en la misma línea no será leído por Matlab. Podemos colocar varias órdenes en una línea si se separan correctamente, puede ser:

por *comas* (**,**) que hacen que se visualicen los resultados

o *puntos y comas* (**;**) que suprimen la impresión en pantalla

Para cerrar Matlab podemos hacerlo tecleando **quit**, cerrando con el aspa típico de Windows, entrando en **File->Exit Matlab** o con las teclas **Ctrl+Q**.

AYUDA EN LÍNEA

Matlab proporciona asistencia de varios modos.

Si queremos consultar un comando determinado podemos buscar información escribiendo en la ventana de comandos **help <comando a consultar>**, o simplemente **help**. También podemos abrir la ventana de ayuda con el ratón o con la tecla F1. Una vez abierta esta ventana podemos buscar por contenidos, palabras concretas, demostraciones…

Por último con la orden **lookfor <palabra>**, busca en todas las primeras líneas de las ayudas de los temas de Matlab y devuelve aquellos que contienen la palabra clave que hemos escrito. No es necesario que la palabra clave sea una orden de Matlab.

FUNCIONES MATEMÁTICAS COMUNES

APROXIMACIONES

Función	¿Qué hace?	Ejemplo x = 5.92
ceil (x)	redondea hacia infinito	6
fix (x)	redondea hacia cero	5
floor (x)	redondea hacia menos infinito	5
round (x)	redondea hacia el entero más próximo	6

(con **x** escalar, vector o matriz, pero redondearía en cada caso los elemento individualmente)

Ejemplo:

```
>> round ( [19.54646 13.656 -2.1565 0.78] )
ans =
   20    14    -2     1
```

TRIGONOMETRÍA

Función	¿Qué hace?
… (x)	función trigonométrica con el ángulo expresado en radianes
sin (x)	seno (radianes)
cos (x)	coseno
tan (x)	tangente
csc (x)	cosecante
sec (x)	secante
cot (x)	cotangente
…d (x)	función trigonométrica con el ángulo expresado en grados
sind (x)	seno (grados)
…	…
…h (x)	función trigonométrica hiperbólica con el ángulo expresado en radianes
sinh (x)	seno hiperbólico (radianes)
…	…
a… (x)	inversa de la función trigonométrica con el resultado expresado en radianes
asin (x)	arco seno (radianes)
…	…
a…d (x)	inversa de la función trigonométrica con el resultado expresado en grados
asind (x)	arco seno (grados)
…	…
a…h (x)	inversa de la función trigonométrica hiperbólica con el resultado expresado en radianes
asinh (x)	arco seno hiperbólico (radianes)
…	…

Ejemplos:

```
>> sin (pi/2)
ans =
    1

>> sind (-90)
ans =
   -1

>> cosd (60)
ans =
   0.5000

>> asind (1)
ans =
   90
```

ALGUNAS OPERACIONES

Función	¿Qué hace?
abs (x)	valor absoluto o magnitud de un número complejo
sign (x)	signo del argumento si x es un valor real (-1 si es negativo, 0 si es cero, 1 si es positivo)
exp (x)	exponencial
gcd (m,n)	máximo común divisor
lcm (m,n)	mínimo común múltiplo
log (x)	logaritmo neperiano o natural
log2 (x)	logaritmo en base 2
log10 (x)	logaritmo decimal
mod(x,y)	módulo después de la división
rem (x,y)	resto de la división entera
sqrt (x)	raíz cuadrada
nthroot (x,n)	raíz n-ésima de x

(**x** e **y** cualquier escalar, **m** y **n** enteros)

Ejemplos:

```
>> abs (-7)          % valor absoluto de -7
ans =
    7

>> sign (10)         % signo del número 10
ans =
    1

>> gcd (9,12)        % máximo común divisor entre 9 y 12
ans =
    3

>> lcm (10,25)       % mínimo común múltiplo
ans =
   50

>> mod (-12,5)       % módulo de la división de -12 entre 5
ans =
    3

> rem (12,5)         % resto de la división de 12 entre 5
ans =
   2

>> nthroot (8,3)     % raíz cúbica de 8
ans =
    2
```

NÚMEROS COMPLEJOS

Función	¿Qué hace?	Ejemplos: x = 3 + 4i y = 2 z = 7
abs (x)	magnitud del número complejo x	5
angle (x)	ángulo (en radianes) del complejo x	0.9273
complex (y,z)	genera el complejo y + zi	2.0000 + 7.0000i
conj (x)	conjugado del número complejo x	3.0000 - 4.0000i
imag (x)	parte imaginaria del número complejo x	4
real (x)	parte real del número complejo x	3
sign (x)	divide el complejo x por su magnitud, devuelve un número complejo con el mismo ángulo de fase pero con magnitud 1	06000 + 0.8000i
isreal (x)	devuelve 1 si es real, y 0 si es complejo	0

(**x** número complejo, **y** y **z** números reales)

VECTORES Y MATRICES

CÓMO DEFINIRLOS

Para crear un vector introducimos los valores deseados separados por espacios (o comas) todo ello entre corchetes **[]**. Si lo que queremos es crear una matriz lo hacemos de forma análoga pero separando las filas con puntos y comas (**;**).

Generalmente usaremos letras mayúsculas cuando nombremos a las matrices y minúsculas para vectores y escalares. Esto no es imprescindible y Matlab no lo exige, pero resulta útil.

Ejemplos:

```
>> x = [5 7 -2 4 -6]    % es un vector, los elementos los separamos con espacios
x =
    5    7   -2    4   -6

>> y = [2,1,3,7]        % es otro vector, los elementos los separamos con comas
y =
    2    1    3    7

>> z = [0 1 2,3 4,5]    % es otro vector, da igual separar los elementos por comas o espacios
z =
    0    1    2    3    4    5

>> A = [1 2 3; 4 5 6]  % es una matriz con 2 filas y 3 columnas
A =
    1    2    3
    4    5    6
```

DIRECCIONAMIENTO DE ELEMETOS DE VECTORES Y MATRICES

Para acceder a los elementos individuales de un vector lo haremos utilizando subíndices, así x(n) sería el n-ésimo elemento del vector x. Si queremos acceder al último podemos indicarlo usando **end** como subíndice.

```
>> x = [5 7 -2 4 -6];
>> x (2)        % segundo elemento del vector x
ans =
    7

>> x (end)      % último elemento del vector x
ans =
   -6
```

Para acceder a un bloque de elementos a la vez, se usa la notación de dos puntos (**:**), así x (m:n) nos da todos los elementos desde el m-ésimo hasta el n-ésimo del vector x.

```
>> x (2:4)      % devuelve desde el segundo al cuarto elemento del vector x
ans =
    7   -2    4
```

Si introducimos un número entre el primero y el segundo también separado por dos puntos (**:**) se mostrarán los elementos del primero al último indicado, incrementados según el número que aparece en el centro (o decrementados si el número es negativo).

```
>> x (1:2:5)            % devuelve el primero, tercero y quinto elemento del vector x
ans =
    5    -2    -6
```

Otra forma de obtener un conjunto concreto de elementos del vector es indicando entre corchetes **[]** las posiciones de los elementos que queremos obtener poniendo paréntesis fuera de los corchetes.

```
>> x ( [3 5 1] )        % devuelve el tercer, quinto y primer elemento del vector x
ans =
   -2   -6    5
```

Para acceder a los elementos de una matriz necesitamos dar dos valores, el primero indica la fila y el segundo la columna.

```
>> A = [1 2 3; 4 5 6];
>> A (2,1)              % elemento de la matriz que está en la fila 2 y en la columna 1
ans =
    4
```

Si queremos que escriba toda una fila usaremos los dos puntos para indicar que queremos todos los elementos.

```
>> A (2,:)              % escribe la segunda fila de la matriz
ans =
    4    5    6
```

Y similar si queremos que escriba toda una columna pero ahora situamos los dos puntos en el lugar de las filas para indicar que queremos todas las filas de esa columna.

```
>> A (:,2)          % escribe la segunda columna de la matriz
ans =
    2
    5
```

Al igual que con los vectores podemos indicar que escriba una serie de filas o columnas, la manera de hacerlo sería muy parecido.

```
>> A (2,2:3)        % escribe de la segunda fila de la matriz, las columnas de la 2 a la 3
ans =
    5    6

>> A (2, [3 1] )    % escribe de la segunda fila de la matriz, las columnas 3 y 1
ans =
    6    4

>> A ( [2 1] , 2:3)  % escribe de las filas 2 y 1 de la matriz, las columnas de la 2 a la 3
ans =
    5    6
    2    3

>> A (end, [1 3] )   % escribe de la última fila, las columnas 1 y 3
ans =
    4    6
```

Matlab tiene además otra forma de identificar cada elemento de una matriz, de modo que podemos acceder a un elemento de una matriz indicando sólo un valor y no dos, pero debemos saber que el orden elegido por Matlab es por columnas así los elementos de la matriz A serían denominados:

A(1)	A(3)	A(5)
A(2)	A(4)	A(6)

Ejemplo:

Como la matriz A que teníamos era

```
A =
    1    2    3
    4    5    6
```

```
>> A (5)        % accede al elemento 5 de la matriz, es decir, igual que si escribiéramos A (1,3)
ans =
    3
```

Pero es preferible para evitar confusiones trabajar con los elementos de las matrices indicando la fila y la columna correspondiente.

CONSTRUCCIÓN ABREVIADA DE ALGUNOS VECTORES

A parte de definir un vector introduciendo cada uno de sus elementos, también podemos crearlo haciendo uso de las siguientes sentencias:

(a:b) crea un vector que comienza en el valor **a** y acaba en el valor **b** aumentando de 1 en 1.

(a:c:b) crea un vector que comienza en el valor **a** y acaba en el valor **b** aumentando de **c** en **c**.

linspace (a,b,c) genera un vector linealmente espaciado entre los valores **a** y **b** con **c** elementos.

linspace (a,b) genera un vector linealmente espaciado entre los valores **a** y **b** con 100 elementos.

logspace (a,b,c) genera un vector logarítmicamente espaciado entre los valores **10^a** y **10^b** con **c** elementos.

logspace (a,b) genera un vector logarítmicamente espaciado entre los valores **10^a** y **10^b** con 50 elementos.

Ejemplos:

```
>> (1:7)          % crea un vector que comienza en 1, aumenta de 1 en 1 y acaba en 7
ans =
    1    2    3    4    5    6    7

>> (1:3:10)      % crea un vector que comenzando en 1, aumenta de 3 en 3 hasta el 10
ans =
    1    4    7   10

>> (1:4:10)      % comenzando en 1, aumenta de 4 en 4 hasta el 10 y por eso acaba en 9
ans =
    1    5    9

>> (50:-7:1)    % crea un vector que comenzando en 50, disminuye de 7 en 7 hasta el 1
ans =
   50   43   36   29   22   15    8    1

>> linspace (2,6,3)     % genera un vector desde el 2 al 6 con 3 elementos equidistantes
ans =
    2    4    6

>> linspace (2,6,4)     % genera un vector desde el 2 al 6 con 4 elementos equidistantes
ans =
   2.0000   3.3333   4.6667   6.0000

>> logspace (0,2,4)    % genera un vector logarítmicamente espaciado entre 10^0 y 10^2 con 4
                         elementos
ans =
   1.0000   4.6416  21.5443  100.0000
```

CONSTRUCCIÓN DE ALGUNAS MATRICES

Al igual que pasa con los vectores, existen unas sentencias que nos ayudan a crear más rápidamente algunas matrices que Matlab ya tiene predefinidas (**m** y **n** deben tomar valores naturales):

zeros (n) crea una matriz cuadrada n x n de ceros.

zeros (m,n) crea una matriz m x n de ceros.

ones (n) crea una matriz cuadrada n x n de unos.

ones (m,n) crea una matriz m x n de unos.

rand (n) crea una matriz cuadrada n x n de números aleatorios con distribución uniforme (0,1).

rand (m,n) crea una matriz m x n de números aleatorios con distribución uniforme (0,1).

randn (n) crea una matriz cuadrada n x n de números aleatorios con distribución normal (0,1).

randn (m,n) crea una matriz m x n de números aleatorios con distribución normal (0,1).

eye (n) crea una matriz cuadrada n x n de unos en la diagonal y ceros el resto.

eye (m,n) crea una matriz m x n de unos en la diagonal y ceros el resto.

magic (n) crea una matriz cuadrada n x n de enteros de modo que sumen lo mismo las filas y las columnas.

hilb (n) crea una matriz cuadrada n x n de Hilbert, es decir, los elementos (i,j) responden a la expresión (1/(i+j-1)).

invhilb (n) crea una matriz cuadrada n x n que es la inversa de la matriz de Hilbert.

Ejemplos:

```
>> zeros (3)          % matriz cuadrada 3 x 3 de ceros
ans =
   0   0   0
   0   0   0
   0   0   0

>> zeros (2,5)        % matriz 2 x 5 de ceros
ans =
   0   0   0   0   0
   0   0   0   0   0

>> ones (2,3)         % matriz de unos
ans =
   1   1   1
   1   1   1
```

```
>> rand (2,4)          % matriz de valores aleatorios entre 0 y 1 según la uniforme (0,1)
ans =
   0.9355   0.4103   0.0579   0.8132
   0.9169   0.8936   0.3529   0.0099

>> randn (2,5)         % matriz de valores aleatorios según la normal (0,1)
ans =
   0.8156   1.2902   1.1908  -0.0198  -1.6041
   0.7119   0.6686  -1.2025  -0.1567   0.2573

>> eye (2)             % matriz  identidad o unidad
ans =
    1    0
    0    1

>> magic (4)           % matriz mágica 4 x 4
ans =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1

>> hilb (3)            % matriz de Hilbert 3 x 3
ans =
   1.0000   0.5000   0.3333
   0.5000   0.3333   0.2500
   0.3333   0.2500   0.2000

>> invhilb (3)         % inversa de la matriz de Hilbert 3 x 3
ans =
    9   -36    30
  -36   192  -180
   30  -180   180
```

OPERACIONES BÁSICAS CON MATRICES

Símbolo	Expresión	Operación
+	A + B	Suma de matrices
-	A - B	Resta de matrices
*	A * B	Multiplicación de matrices
.*	A .* B	Multiplicación elemento a elemento de matrices
/	A / B	División de matrices por la derecha
./	A ./ B	División elemento a elemento de matrices por la derecha
\	A \ B	División de matrices por la izquierda
.\	A .\ B	División elemento a elemento de matrices por la izquierda
^	A ^ n	Potenciación (n debe ser un número, no una matriz)
.^	A .^ B	Potenciación elemento a elemento de matrices
'	A '	Trasposición compleja conjugada
.'	A .'	Trasposición de matrices

Ejemplos:

Definimos tres matrices para poder hacer operaciones entre ellas.

```
A =                 B =                 C =
     1    2              1    1              1.0000 + 1.0000i   2.0000 + 2.0000i
     3    4              0    1              3.0000 + 1.0000i   4.0000 + 7.0000i
```

```
>> A * B       % multiplicación de matrices
ans =
    1    3
    3    7

>> A .* B      % multiplicación elemento a elemento
ans =
    1    2
    0    4

>> C '         % traspuesta conjugada
ans =
  1.0000 - 1.0000i   3.0000 - 1.0000i
  2.0000 - 2.0000i   4.0000 - 7.0000i

>> C .'        % traspuesta
ans =
  1.0000 + 1.0000i   3.0000 + 1.0000i
  2.0000 + 2.0000i   4.0000 + 7.0000i

>> A + 2       % si sumamos el número 2 a la matriz se suma ese número a cada elemento
ans =
    3    4
    5    6
```

FUNCIONES PARA OPERAR CON VECTORES

Función	¿Qué hace?
cross (x,y)	producto vectorial entre los vectores x e y
dot (x,y)	producto escalar entre los vectores x e y

Ejemplos:

```
>> x = [1 2 3]; y = [4 5 6];
>> cross (x,y)          % producto vectorial
ans =
   -3     6    -3

>> dot (x,y)            % producto escalar
ans =
   32
```

FUNCIONES PARA EL ANÁLISIS DE MATRICES

Función	¿Qué hace?
cond (A)	número de condición
det (A)	determinante
diag (v)	crea una matriz diagonal con el vector v sobre la diagonal
diag (A)	extrae la diagonal de la matriz A como un vector columna
eig (A)	valores propios
inv (A)	matriz inversa
length (A)	máxima dimensión
norm (A)	norma
norm (A,n)	norma-n
normest (A)	estimación de la norma-2
null (A)	espacio nulo
orth (A)	ortogonalización
pinv (A)	pseudoinversa
poly (A)	polinomio característico
rank (A)	rango
rref (A)	reducción mediante la eliminación de Gauss de una matriz
size (A)	dimensiones
trace (A)	traza
tril (A)	matriz triangular inferior a partir de la matriz A
triu (A)	matriz triangular superior a partir de la matriz A

(Con **A** matriz, **v** vector y **n** número natural)

Ejemplos:

```
>> v = [1 2 3];
>> diag (v)              % crea una matriz diagonal a partir del vector v
ans =
    1    0    0
    0    2    0
    0    0    3

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

>> diag (A)              % crea un vector columna a partir de la diagonal de la matriz A
ans =
    1
    8
    6

>> size (A)              % devuelve las dimensiones de la matriz como un vector fila
ans =
    3    4
```

```
>> length (A)          % devuelve la mayor de las dos dimensiones de la matriz
ans =
    4

>> trace (A)           % traza de la matriz
ans =
   15

>> rank (A)            % rango de la matriz
ans =
    2

>> rref (A)            % reducción mediante Gauss
ans =
   1.0000         0   -1.0000   -4.6667
         0   1.0000    2.0000    4.3333
         0         0         0         0

>> l = tril (A), u = triu (A)
l =
    1    0    0    0          % convierte en ceros todos los elementos que quedan encima de
    7    8    0    0          % la diagonal principal y lo guarda en la variable l
    2    4    6    0
u =
    1    2    3    4          % convierte en ceros todos los elementos que quedan debajo de
    0    8    9    2          % la diagonal principal y lo guarda en la variable u
    0    0    6    8
```

OTRAS OPERACIONES CON MATRICES

Función	¿Qué hace?
find (A)	devuelve los índices donde las entradas de A son distinto de cero
fliplr (A)	intercambia la matriz de izquierda a derecha
flipud (A)	intercambia la matriz de arriba a abajo
reshape (A,m,n)	devuelve una matriz m x n cuyos elementos se toman por columnas de A, si A no contiene m x n elementos daría un error
rot90 (A)	gira la matriz 90º en sentido contrario a las agujas del reloj
rot90 (A,n)	gira la matriz n x 90º
expm (A)	matriz exponencial
logm (A)	matriz logarítmica
sqrtm (A)	matriz de raíces cuadradas
funm (A,@función)	evalúa la función que indiquemos en la matriz A
exp, log, sqrt…	operan elemento a elemento
[VE,VA] = eig (A)	VE son los vectores propios y VA son los valores propios
[L,U] = lu (A)	factorización LU
[Q,R] = qr (A)	factorización QR

(Con **A** matriz, **m** y **n** naturales)

Ejemplos:

```
>> A = [pi  0; pi/4 pi/3]
A =
   3.1416        0
   0.7854    1.0472

>> find (A)              % devuelve los índices como un vector columna
ans =
    1
    2
    4

>> reshape (A,1,4)
ans =
   3.1416    0.7854         0    1.0472

>> rot90 (A)             % gira la matriz 90º
ans =
         0    1.0472
   3.1416    0.7854

>> rot90 (A,3)           % gira la matriz 270º ( 90º x 3 = 270º )
ans =
   0.7854    3.1416
   1.0472         0

>> funm (A,@sin)         % calcula el seno de cada elemento de la matriz
ans =
   0.0000         0
  -0.3248    0.8660

>> expm (A)
ans =
  23.1407         0
   7.6091    2.8497
```

TEXTO

Una cadena de caracteres es texto rodeado por comillas simples (') y se manejan como vectores filas. Se direccionan y manipulan igual que los vectores. Son posibles las operaciones matemáticas sobre cadenas. Una vez hecha una operación matemática sobre una cadena, ésta se ve como un vector de números en ASCII.

Para ver la representación ASCII de una cadena, podemos utilizar las funciones **abs**, **double** o sumamos cero. Para restaurarla y verla de nuevo como cadena de caracteres, usamos la función **setstr**. Si queremos cambiar a minúsculas añadiremos la diferencia entre 'a' y 'A'.

Si queremos que escriba algo en pantalla podemos utilizar el comando **disp**.

Ejemplos:

```
>> a = 'casa'; b = 'gato';        % a y b son cadenas de caracteres (se manejarán como vectores)

>> a + b
ans =
  202   194   231   208

>> a + 0                          % vemos la representación ASCII de la cadena
ans =
   99    97   115    97

>> abs (a)                        % otra forma de ver la representación ASCII de la cadena
ans =
   99    97   115    97

>> double (a)                     % otra  tercera forma de ver la representación ASCII de la cadena
ans =
   99    97   115    97

>> setstr (ans)                   % convertimos un vector de número enteros en caracteres
ans =
casa

>> abs ('a') – abs ('A')          % calculamos la diferencia entre minúsculas y mayúsculas
ans =
   32

>> setstr (a-32)                  % escribimos los caracteres conociendo la representación ASCII
ans =
CASA

>> disp (a)                       % escribe el valor almacenado en la variable a
casa

>> disp ('escribe esto')          % escribe el texto que vaya entre las comillas
escribe esto
```

HIPERMATRICES

CÓMO DEFINIRLAS

Matlab permite trabajar con matrices de más de dos dimensiones. Los elementos de una hipermatriz pueden ser números, caracteres, estructuras y vectores o matrices de celdas. Las funciones que operan con matrices de más de dos dimensiones son análogas a las funciones vistas anteriormente aunque con algunas diferencias, por ejemplo, a la hora de definirlas:

```
>> HM(:,:,1) = [1 2 3; 4 5 6];          % definimos la primera capa
>> HM(:,:,2) = [7 8 9; 10 11 12]        % definimos la segunda capa
HM(:,:,1) =
    1    2    3
    4    5    6
HM(:,:,2) =
    7    8    9
   10   11   12
```

OPERACIONES CON HIPERMATRICES

Algunas funciones para generar matrices admiten más de dos subíndices y pueden ser utilizadas para generar hipermatrices como **rand**, **randn**, **zeros** y **ones**, también se pueden emplear con hipermatrices las funciones **size** y **reshape** entre otras. La función **cat** permite concatenar matrices según las distintas "dimensiones".

Ejemplos:

```
>> A = zeros (2,3); B = ones (2,3);          % definimos dos matrices de las mismas dimensiones
>> cat (1,A,B)                               % las concatena una debajo de la otra
ans =
    0    0    0
    0    0    0
    1    1    1
    1    1    1

>> cat (2,A,B)                               % las concatena una al lado de la otra
ans =
    0    0    0    1    1    1
    0    0    0    1    1    1

>> cat (3,A,B)                               % las concatena como distintas capas de una hipermatriz
ans(:,:,1) =
    0    0    0
    0    0    0
ans(:,:,2) =
    1    1    1
    1    1    1
```

Respecto al resto de funciones debemos tener en cuenta que:

1. Las funciones que operan sobre escalares, como **sin**, **cos**, etc., se aplican sobre hipermatrices elemento a elemento (igual que ocurre al aplicarlas sobre vectores y matrices).

2. Las funciones que operan sobre vectores, como **sum**, **max**, etc., se aplican a matrices e hipermatrices según la primera dimensión, resultando un array de una dimensión inferior.

3. Las funciones matriciales propias del álgebra lineal, como **det**, **inv**, etc., no se pueden aplicar a hipermatrices, para aplicarlas habría que extraer las matrices correspondientes.

ESTRUCTURAS

CÓMO DEFINIRLAS

Es una agrupación de datos de tipo diferente bajo un mismo nombre. A los datos les llamamos campos. No hace falta definir previamente el modelo de la estructura, podemos ir creando los distintos campos uno a uno o bien con el comando **struct**, donde los nombres de los campos se escriben entre apóstrofos (**'**) seguidos del valor que se les quiere asignar.

Ejemplos:

```
>> alumno.nombre = 'Pablo';          % introducimos el campo nombre en la estructura alumno
>> alumno.apellido1 = 'Fernández';  % introducimos el campo apellido1 en la estructura alumno
>> alumno.apellido2 = 'García';      % introducimos el campo apellido2 en la estructura alumno
>> alumno.edad = 15;                 % introducimos el campo edad en la estructura alumno
>> alumno              % escribe por pantalla la información almacenada en la estructura alumno
alumno =

     nombre: 'Pablo'
  apellido1: 'Fernández'
  apellido2: 'García'
      edad: 15

>> alumno2 = struct ('nombre','Fermín','apellido1','Martínez','apellido2','Gil','edad',16)
alumno2 =                          % otro modo de introducir los campos

     nombre: 'Fermín'
  apellido1: 'Martínez'
  apellido2: 'Gil'
      edad: 16
```

Pueden crearse vectores y matrices de estructuras, por ejemplo:

```
>> alumno (1) = struct ('nombre','Pablo','apellido1','fernández','apellido2','García','edad',15);
>> alumno (2) = struct ('nombre','Fermín','apellido1','Martínez','apellido2','Gil','edad',16);
>> alumno              % nos devuelve información sobre los campos que tiene la estructura alumno
alumno =
1x2 struct array with fields:
  nombre
  apellido1
  apellido2
  edad

>> alumno (1)          % nos devuelve los datos del primer elemento del vector de la estructura
ans =
     nombre: 'Pablo'
  apellido1: 'fernández'
  apellido2: 'García'
      edad: 15
```

```
>> alumno (2)          % nos devuelve los datos del segundo elemento del vector de la estructura
ans =
      nombre: 'Fermín'
   apellido1: 'Martínez'
   apellido2: 'Gil'
       edad: 16
```

Para ver un campo concreto de todos los alumnos bastaría teclear:

```
>> alumno.nombre    % escribe los datos de todos los campo nombre de la estructura en orden
ans =
Pablo
ans =
Fermín
```

OPERAR CON ESTRUCTURAS

Función	¿Qué hace?
fieldnames (E)	devuelve el nombre de los campos de la estructura E
isfield (E, 'c')	devuelve 1 si c es un campo de la estructura E y 0 si no lo es
isstruct (E)	devuelve 1 si E es una estructura y 0 si no lo es
rmfield (E, 'c')	elimina el campo c de la estructura E

(**E** es una estructura y **c** es un campo)

Ejemplos:

```
>> fieldnames (alumno)          % devuelve los campos de la estructura alumno
ans =
   'nombre'
   'apellido1'
   'apellido2'
   'edad'

>> isfield (alumno,'nombre')    % devuelve 1 por ser cierto que nombre es un campo de alumno
ans =
    1

>> isstruct (alumno)            % devuelve 1 porque es cierto que alumno es una estructura
ans =
    1

>> rmfield (alumno,'edad')      % elimina el campo edad de la estructura alumno
ans =
1x2 struct array with fields:
   nombre
   apellido1
   apellido2
```

VECTORES Y MATRICES DE CELDAS

CÓMO DEFINIRLOS

Un vector de celdas es un vector cuyos elementos son cada uno de ellos una variable de cualquier tipo. En todo vector sus elementos pueden ser números o cadenas de caracteres, pero en un vector de celdas el primer elemento puede ser un número, el segundo una matriz, el tercero una estructura, etc.

Para crear un vector de celdas usaremos llaves ({}).

```
>> celda (1) = {[0 1 2]};        % creamos un vector de celdas definiendo celda a celda
>> celda (2) = {'cadena de caracteres'};
>> celda (3) = {eye(2)};
>> celda (4) = {-7};
>> celda
celda =
   [1x3 double]    [1x20 char]    [2x2 double]    [-7]

>> cel {1} = [0 1 2];  % creamos otro vector de celdas definiendo celda a celda de forma distinta
>> cel {2} = 'cadena de caracteres';
>> cel {3} = eye (2);
>> cel {4} = -7;
>> cel
cel =
   [1x3 double]    [1x20 char]    [2x2 double]    [-7]

>> c = { [0 1 2] ,'cadena de caracteres',eye(2),-7};  % otra forma de crear un vector de celdas
```

Si queremos crear una matriz o una hipermatriz de celdas se haría de forma similar.

OPERAR CON VECTORES Y MATRICES DE CELDAS

Función	¿Qué hace?
cell (m,n)	crea una matriz de celdas con m filas y n columnas
celldisp (c)	muestra el contenido de todas las celdas de c
cellplot (c)	muestra la representación gráfica de las celdas de c
iscell (c)	devuelve 1 si es una matriz de celdas y 0 si no lo es
num2cell (x)	convierte el vector o matriz numérica en celdas

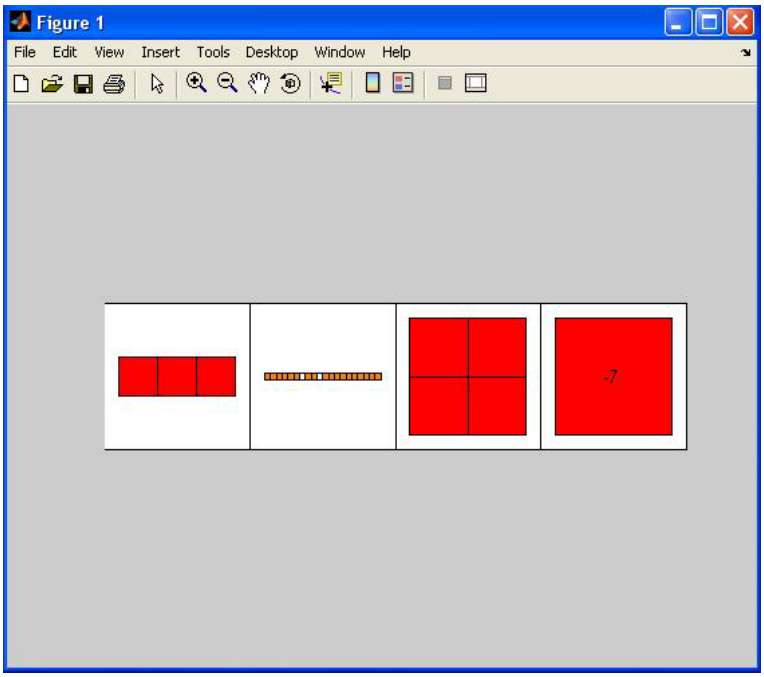
(**m** y **n** números naturales, **c** celdas y **x** vector o matriz)

Ejemplos:

```
>> cell (2,3)              % crea una matriz de celdas vacías
ans =
    []    []    []
    []    []    []
```

```
>> celldisp (c)          % escribe el contenido de las celdas de c
 c{1} =
     0     1     2
c{2} =
cadena de caracteres
c{3} =
     1     0
     0     1
c{4} =
    -7
```

```
>> cellplot (c)          % representa gráficamente cómo son las celdas de c
```




```
>> iscell (c)
ans =
     1

>> A = eye (3,2);
>> num2cell (A)
ans =
    [1]    [0]
    [0]    [1]
    [0]    [0]
```

OPERACIONES RELACIONALES Y LÓGICAS

Como entradas a las expresiones relacionales y lógicas, Matlab considera que cero es falso y que cualquier número distinto de cero es verdadero. La salida de expresiones de este tipo produce 1 si es verdadero y 0 si es falso.

OPERADORES RELACIONALES

Operador	¿Qué significa?
<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que
==	igual a
~=	distinto de

La salida de las operaciones lógicas se puede utilizar también en operaciones matemáticas.

OPERADORES LÓGICOS

Operador	¿Qué significa?
&	y
|	o
~	no

Además de los operadores relacionales y lógicos básicos anteriores, Matlab proporciona una serie de funciones relacionales y lógicas adicionales que incluyen:

Función	¿Qué significa?
xor (x,y)	operación "o" exclusiva, devuelve 0 si ambas son falsas o ambas verdaderas y devuelve 1 si una es falsa y la otra verdadera
any (x)	devuelve 1 si algún elemento en un vector x es no nulo y devuelve 0 si son todos nulos, si se trata de una matriz da una respuesta por cada columna
all (x)	devuelve 1 si todos los elementos en un vector x son no nulos y 0 si existe alguno nulo y si se trata de una matriz da una respuesta por cada columna
exist ('x')	devuelve uno si existe y cero si no existe
isnan (x)	devuelve unos en magnitudes no numéricas (NaN) en x
isinf (x)	devuelve unos en magnitudes infinitas (Inf) en x
isfinite (x)	devuelve unos en valores finitos en x

Podemos ver muchos más casos pero todos serían similares: **ischar**, **isempty**, **isequal**, **isfloat**, **isinteger**, **islogical**, **isnumeric**, **isprime**, **isreal**, **isscalar**, **isspace**, …

Existe un orden de precedencia para operadores aritméticos, lógicos y relacionales, en la siguiente tabla van de mayor a menor precedencia:

Orden de precedencia de operadores	
1º	^ .^ ' .'
2º	* / \ .* ./ .\
3º	+ - ~ +(unario) -(unario)
4º	: > < >= <= == ~=
5º	| &

Ejemplos:

```
>> a = 1:9, b = 5-a              % definimos dos vectores
a =
    1    2    3    4    5    6    7    8    9
b =
    4    3    2    1    0   -1   -2   -3   -4

>> r1 = a<6    % pregunta si a es menor que 6, devuelve 1 cuando es verdadero y 0 cuando es falso
r1 =
    1    1    1    1    1    0    0    0    0

>> r2 = a==b  % pregunta si a es igual a b, devuelve 1 cuando es verdadero y 0 cuando es falso
r2 =
    0    0    0    0    0    0    0    0    0

>> r3 = a~=b  % pregunta si a es distinto a b, devuelve 1 cuando es verdadero y 0 cuando es falso
r3 =
    1    1    1    1    1    1    1    1    1

>> r4 = (a>b)&(b>-3)% pregunta si a>b y b>-3, devuelve 1 cuando es verdadero y 0 cuando es falso
r4 =
    0    0    1    1    1    1    1    0    0

>> c = [Inf 0 5 -8 NaN 94];
>> exist ('c')    % pregunta si existe alguna variable llamada c
ans =
    1

>> isnan (c)    % pregunta cuando c es NaN, devuelve 1 cuando es verdadero y 0 cuando es falso
ans =
    0    0    0    0    1    0

>> isinf (c)     % pregunta cuando c es Inf, devuelve 1 cuando es verdadero y 0 cuando es falso
ans =
    1    0    0    0    0    0

>> isfinite (c)  % pregunta cuando c es finito, devuelve 1 cuando es verdadero y 0 cuando es falso
ans =
    0    1    1    1    0    1
```

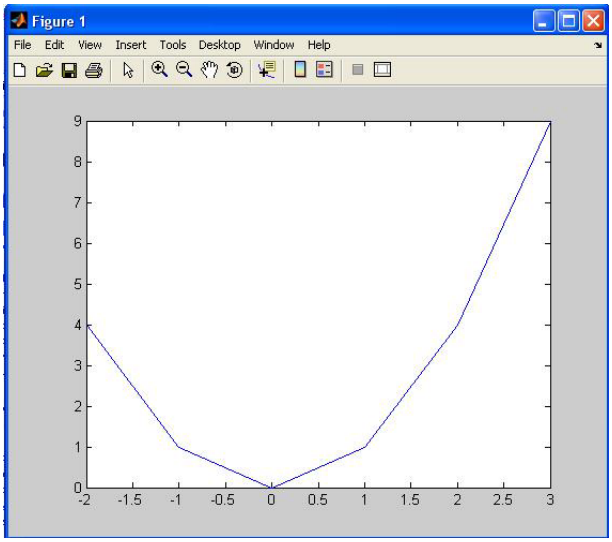
GRÁFICAS

2-D

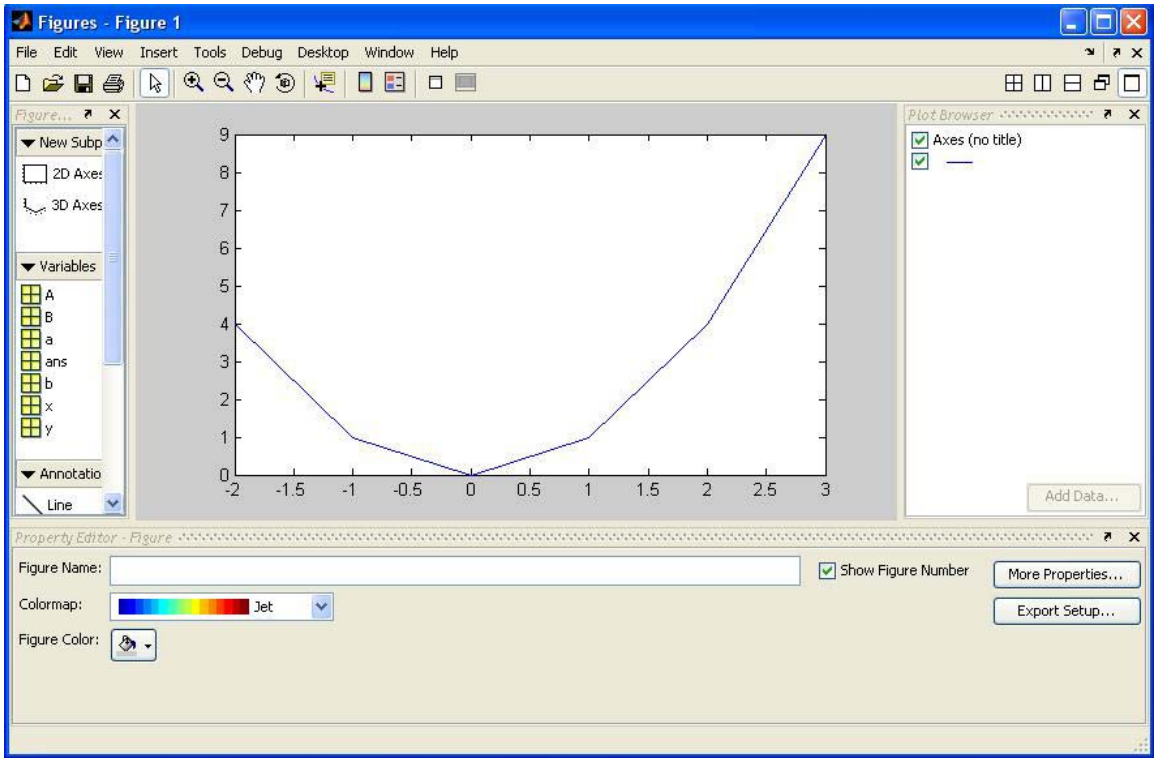
La orden **plot** genera una gráfica. Los argumentos deben ser vectores de la misma longitud.

Ejemplo:

```
>> x = [-2 -1 0 1 2 3]; y = [4 1 0 1 4 9];
>> plot (x,y)
```

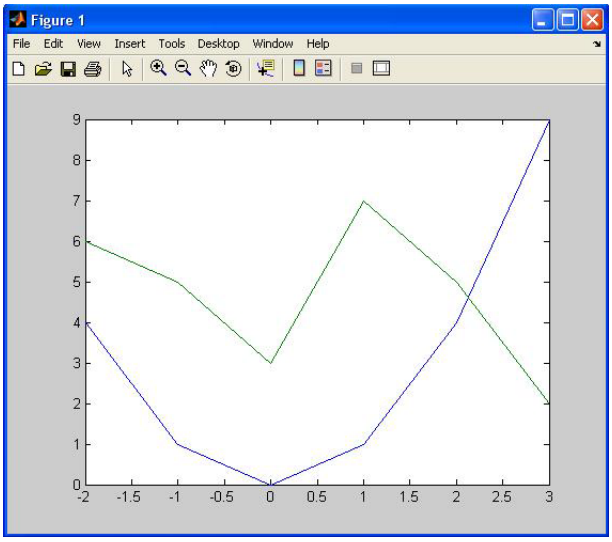


Si queremos cambiar la apariencia de la gráfica basta pinchar en el último botón de la barra de herramientas  y se abrirán unos cuadros en los laterales que nos permitirán ir haciendo los cambios deseados como darle nombre a los ejes.



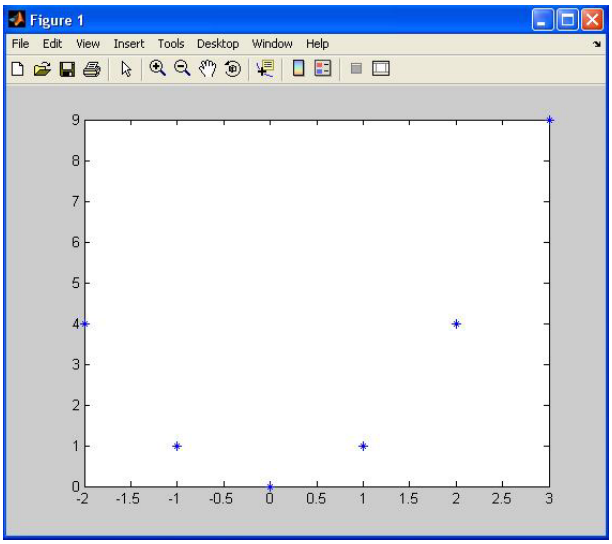
La función **plot** nos permite otras opciones como superponer gráficas sobre los mismos ejes:

```
>> x = [-2 -1 0 1 2 3]; y = [4 1 0 1 4 9]; z = [6 5 3 7 5 2];
>> plot (x,y,x,z)
```



También podemos usar distintos tipos de líneas para el dibujo de la gráfica:

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



Además podemos colocar etiquetas o manipular la gráfica:

etiqueta sobre el eje X de la gráfica actual:	>> **xlabel('texto')**
etiqueta sobre el eje Y de la gráfica actual:	>> **ylabel('texto')**
título en la cabecera de la gráfica actual:	>> **title('texto')**
texto en el lugar especificado por las coordenadas:	>> **text(x,y, 'texto')**
texto, el lugar lo indicamos después con el ratón:	>> **gtext('texto')**
dibujar una rejilla:	>> **grid**
fija valores máximo y mínimo de los ejes:	>> **axis([xmin xmax ymin ymax])**
fija que la escala en los ejes sea igual:	>> **axis equal**
fija que la gráfica sea un cuadrado:	>> **axis square**
desactiva **axis equal** y **axis square:**	>> **axis normal**
abre una ventana de gráfico:	>> **hold on**
borra lo que hay en la ventana de gráfico:	>> **hold off**

Todas estas órdenes se las podemos dar desde la propia ventana de la gráfica una vez que hemos abierto las opciones con el botón indicado anteriormente.

Otros comandos relacionados con las gráficas son los siguientes:

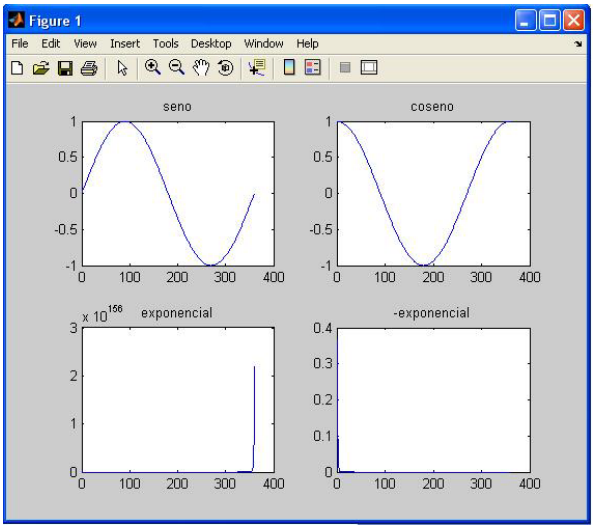
Orden	¿Qué hace?	Imagen
area	colorea el area bajo la gráfica	
bar	diagrama de barras (verticales)	
barh	diagrama de barras (horizontales)	
hist	histograma	
pie	sectores	
rose	histograma polar	
stairs	gráfico de escalera	
stem	secuencia de datos discretos	
loglog	como plot pero con escala logarítmica en ambos ejes	
semilogx	como plot pero escala logarítmica en el eje x	
semilogy	como plot pero escala logarítmica en el eje y	

Para obtener una información más detallada se recomienda utilizar la ayuda de Matlab:

>> help <orden>

Una ventana gráfica se puede dividir en **m** particiones horizontales y en **n** verticales, de modo que cada subventana tiene sus propios ejes, y para hacer esto vamos a usar **subplot (m,n,p)** donde **p** indica la subdivisión que se convierte en activa.

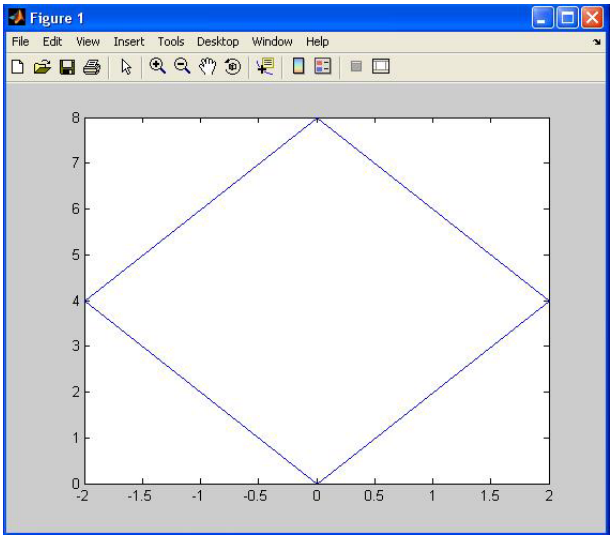
```
>> x = 1:360; y1 = sind (x); y2 = cosd (x); y3 = exp (x); y4 = exp (-x);
>> subplot (2,2,1), plot (x,y1), title ('seno')
>> subplot (2,2,2), plot (x,y2), title ('coseno')
>> subplot (2,2,3), plot (x,y3), title ('exponencial')
>> subplot (2,2,4), plot (x,y4), title ('-exponencial')
```



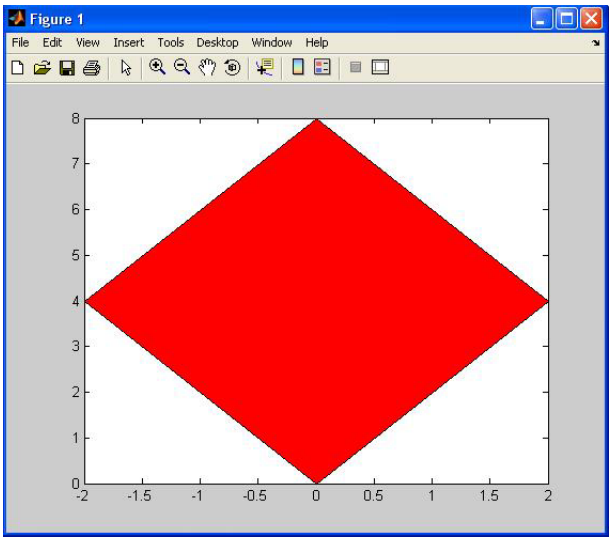
Para volver al modo por defecto basta escribir: **subplot (1,1,1)**.

Para dibujar polígonos podemos usar la función **plot** pero teniendo en cuenta que el último punto de ambos vectores deben coincidir para que la gráfica quede cerrada. Pero si lo que queremos es que quede coloreado todo el interior del polígono debemos usar mejor la función **fill**, tiene tres argumentos, los dos vectores que forman los puntos y un tercer argumento para indicar el color.

```
>> x = [-2 0 2 0 -2]; y = [4 8 4 0 4];
>> plot (x,y)
```




```
>> x = [-2 0 2 0 -2]; y = [4 8 4 0 4];
>> fill (x,y,'r')          % dibuja el polígono, 'r' indica el color rojo
```

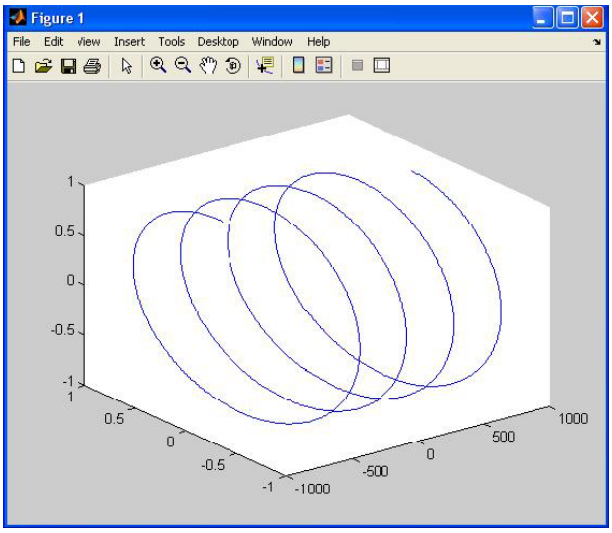


3-D

Gráficos de línea:

También podemos crear gráficas en 3 dimensiones, se trata de extender la orden de **plot** (2-D) a **plot3** (3-D) donde el formato será igual pero los datos estarán en tripletes:

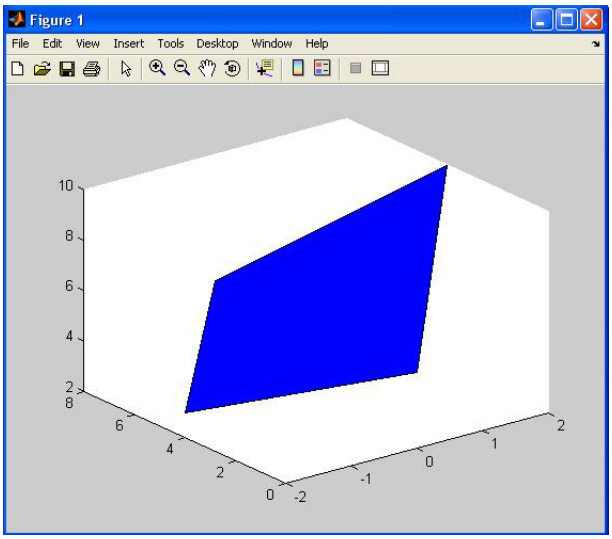
```
>> x = -720:720; y = sind (x); z = cosd (x);

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



Podemos hacer girar la gráfica usando de la barra de herramientas el botón  o hacerla más grande o más pequeña con  . Al igual que ocurría con las gráficas en dos dimensiones podemos nombrar los ejes o hacer modificaciones entrando en opciones con el botón  .

Si queremos representar un polígono en 3 dimensiones lo haremos con la función **fill3** de forma similar a **fill** pero ahora con 4 argumentos, siendo el cuarto el que indica el color.

```
>> x = [-2 0 2 0 -2];
>> y = [4 8 4 0 4];
>> z = [3 5 10 5 3];
>> fill3 (x,y,z,'b')       % dibuja en 3-D, 'b' indica el color azul
```

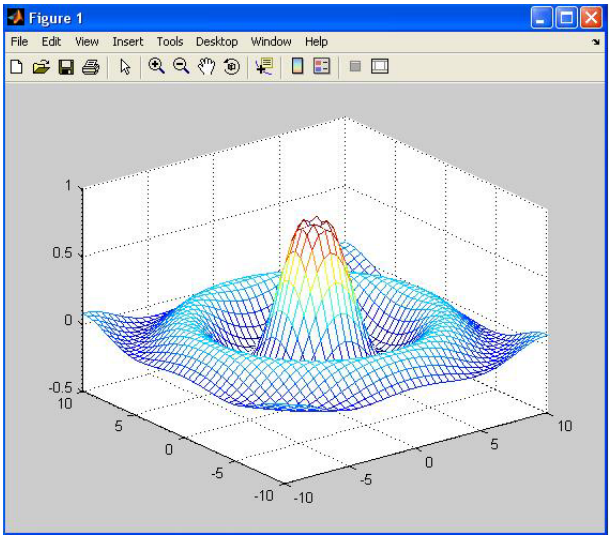


Superficie de malla:

La orden **[X,Y]=meshgrid(x,y)** crea una matriz **X** cuyas filas son copias del vector x y una matriz **Y** cuyas columnas son copias del vector y. Para generar la gráfica de malla se usa la orden **mesh(X,Y,Z)**, **mesh** acepta un argumento opcional para controlar los colores. También puede tomar una matriz simple como argumento: **mesh(Z)**.

Ejemplo:

```
>> x = -10:0.5:10; y = -10:0.5:10;
>> [X,Y] = meshgrid (x,y);                    % crea matrices para hacer la malla
>> Z = sin (sqrt (X .^2 + Y .^2)) ./ sqrt (X .^ 2 + Y .^ 2 + 0.1);
>> mesh (X,Y,Z)                               % dibuja la gráfica
```



Hubiera dado igual si hubiéramos escrito:

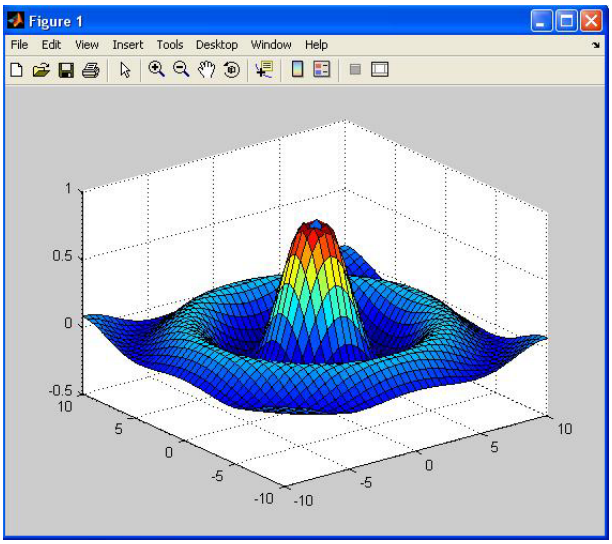
```
>> [X,Y] = meshgrid (-10:0.5:10);
>> Z = sin (sqrt (X .^2 + Y .^ 2)) ./ sqrt (X .^ 2 + Y .^ 2 + 0.1);
>> mesh (X,Y,Z)
```

Gráfica de superficie:

Es similar a la gráfica de malla, pero aquí se rellenan los espacios entre líneas. La orden que usamos es **surf** con los mismos argumentos que para **mesh**.

Ejemplo:

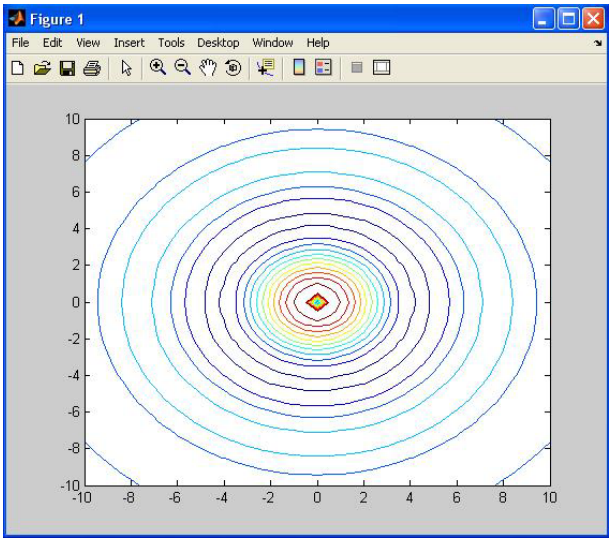
```
>> surf (X,Y,Z)
```



Las gráficas de contorno en 2-D y 3-D se generan usando respectivamente las funciones **contour** y **contour3**.

Ejemplo:

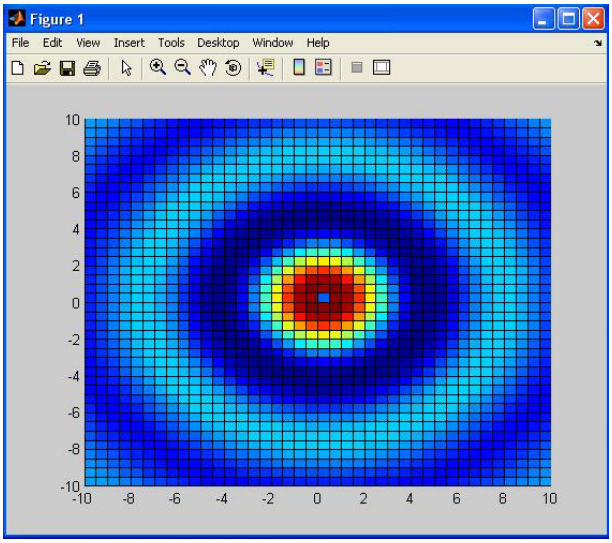
>> contour (X,Y,Z) % dibuja las líneas de contorno



La función **pcolor** transforma la altura a un conjunto de colores.

Ejemplo:

>> pcolor (X,Y,Z)

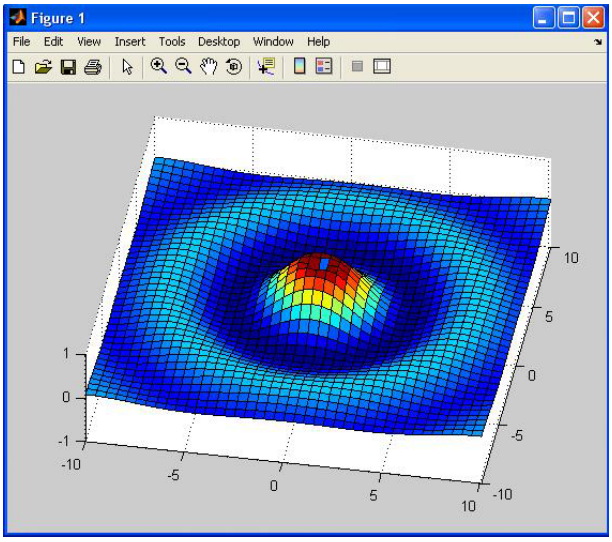


Manipulación de gráficos:

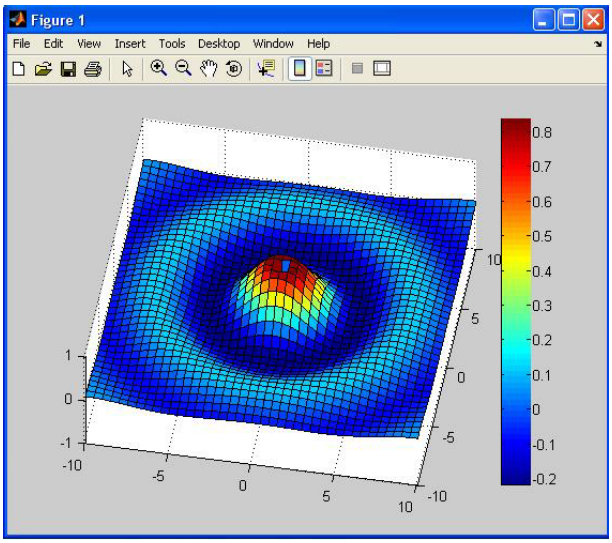
fija el ángulo de visión especificando el azimut y la elevación:	>> **view(az,el)**
coloca su vista en un vector de coordenada cartesiana (x,y,z) en el espacio 3-D:	>> **view([x,y,z])**
almacena en **az** y **el** los valores del azimut y de la elevación de la vista actual:	>> **[az,el]=view**
añade etiquetas de altura a los gráficos de contorno:	>> **clabel(C,h)**
añade una barra de color vertical mostrando las transformaciones:	>> **colorbar**

Ejemplos:

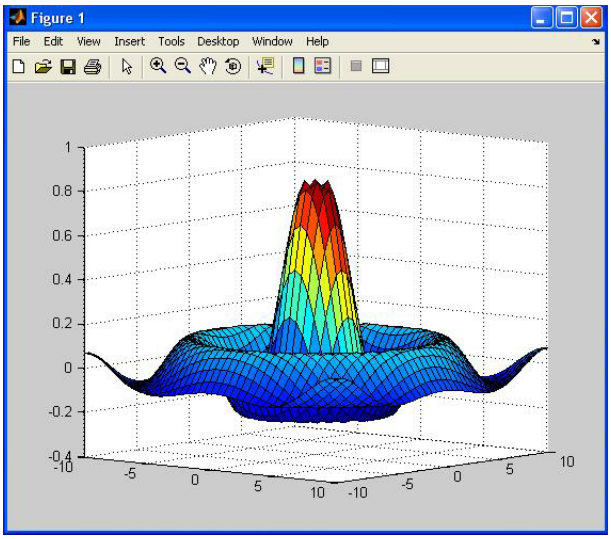
```
>> surf (X,Y,Z)
>> view (10,70)
```



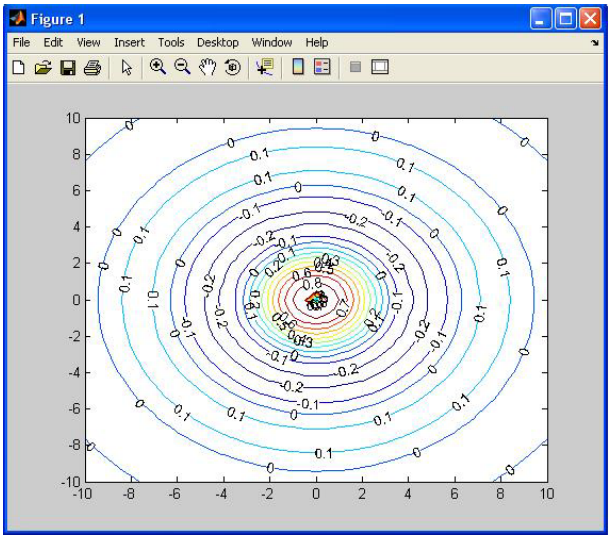

```
>> colorbar     % añade la barra de color a la figura actual
```




```
>> surf (X,Y,Z)
>> view ( [10,-12,2] )
```




```
>> surf (X,Y,Z)
>> [az,el] = view
az =
  -37.5000
el =
    30

>> [C,h] = contour (X,Y,Z);
>> clabel (C,h)
```



Comprensión de los mapas de color:

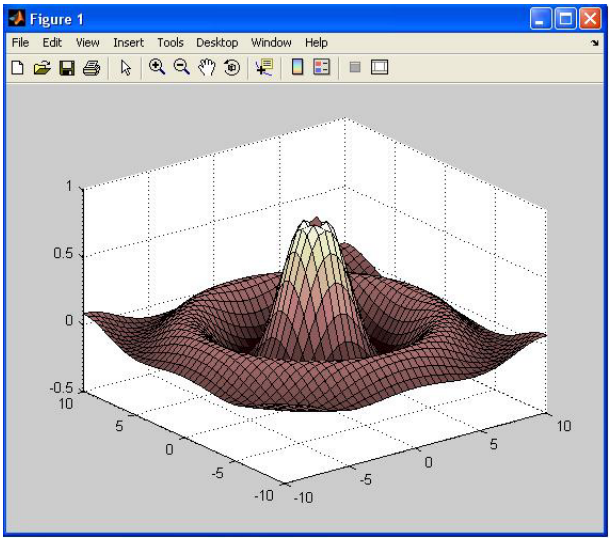
Color	Nombre corto	Rojo/Verde/Azul
Negro	**k**	[0 0 0]
Blanco	**w**	[1 1 1]
Rojo	**r**	[1 0 0]
Verde	**g**	[0 1 0]
Azul	**b**	[0 0 1]
Amarillo	**y**	[1 1 0]
Magenta	**m**	[1 0 1]

La sentencia **colormap (M)** instala al matriz M como el mapa de color a utilizar por la figura actual.

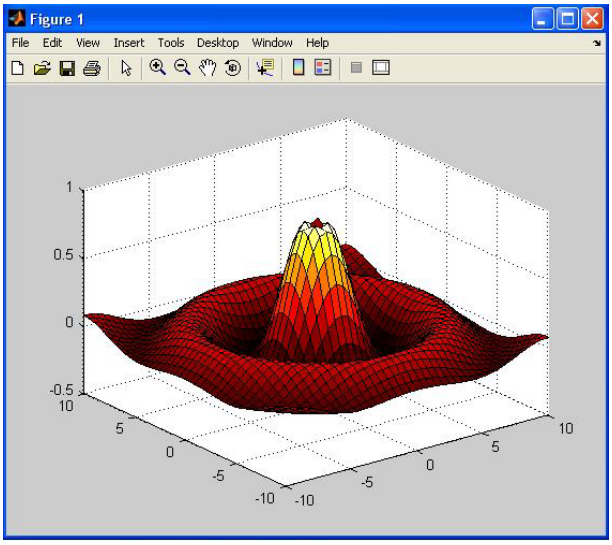
Función	Colores
Jet	
HSV	
Hot	
Cool	
Spring	
Summer	
Autumn	
Winter	
Gray	
Bone	
Copper	
Pink	
Lines	

Ejemplos:

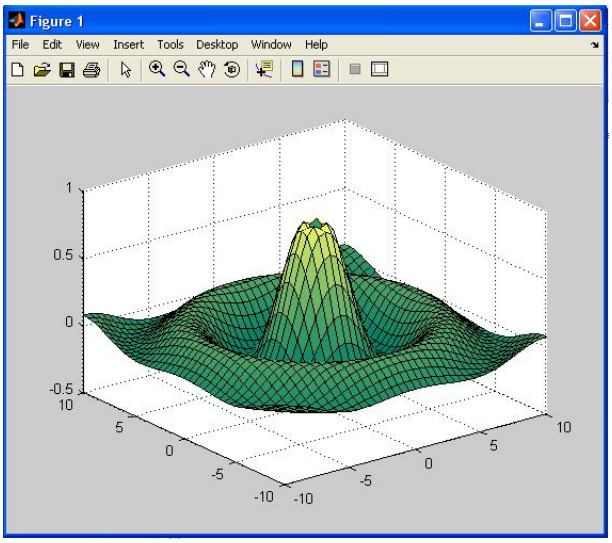
```
>> surf (X,Y,Z)
>> colormap (pink)
```



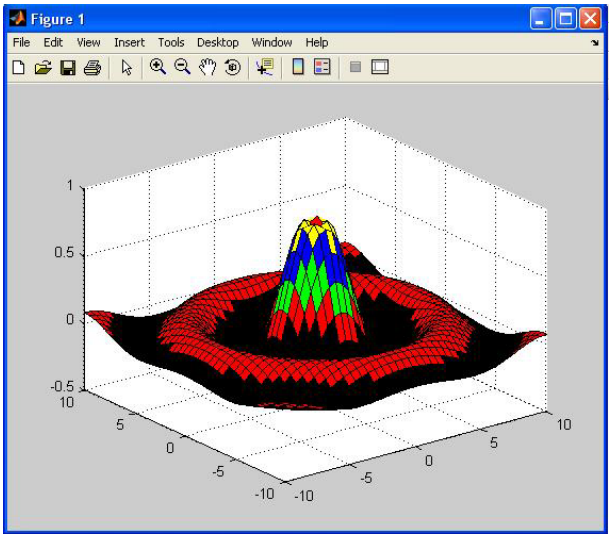
>> colormap (hot)



>> colormap (summer)



>> M = [0 0 0; 1 0 0; 0 1 0; 0 0 1; 1 1 0]; % creamos una matriz de colores
>> colormap (M)



PROGRAMACIÓN DE MATLAB

Matlab es una aplicación que permite programar fácilmente.

SENTENCIA FOR

Un bloque **for** en cada iteración asigna a la variable la columna i-ésima de la expresión y ejecuta las órdenes. En la práctica las expresiones suelen ser del tipo escalar:escalar en cuyo caso las columnas son escalares.

for variable = expresión
    <orden>
    <orden>
        …
    <orden>
end

Ejemplo:

```
>> for x = 1:5
        disp ('x toma el valor')        % escribe por pantalla el texto que se indica entre las comillas
        disp (x)                        % escribe el valor de la variable x
   end

x toma el valor          % es lo que devuelve por pantalla
    1
x toma el valor
    2
x toma el valor
    3
x toma el valor
    4
x toma el valor
    5
```

SENTENCIA WHILE

Un bloque **while** ejecuta las órdenes mientras todos los elementos de la expresión sean verdaderos.

while <expresión>
    <orden>
    <orden>
        …
    <orden>
end

Ejemplo:

```
>> a=3;
>> while a < 5
        disp ('a es menor que 5 ya que vale')
        disp (a)
        a = a + 1;
    end

a es menor que 5 ya que vale        % es lo que devuelve por pantalla
    3
a es menor que 5 ya que vale
    4
```

SENTENCIA IF

Un bloque **if** puede escribirse de varias maneras distintas. Lo que hace es evaluar una expresión lógica y si es cierta ejecuta las órdenes que encuentre antes del **end**.

```
if <expresión>
        <órdenes evaluadas si la expresión es verdadera>
end
```

Puede que nos interese que en caso de no ejecutar dicha orden ejecute otra distinta. Esto se lo indicaremos usando **else** dentro del bloque.

```
if <expresión>
        <órdenes evaluadas si la expresión es verdadera>
else
        <órdenes evaluadas si la expresión es falsa>
end
```

Si queremos dar una estructura mucho más completa, usaremos la más general donde sólo se evalúan las órdenes asociadas con la primera expresión verdadera de todas. En cuanto la evalúe deja de leer el resto y se dirige directamente al **end**.

```
if <expresión1>
        <órdenes evaluadas si la expresión1 es verdadera>
elseif <expresión2>
        <órdenes evaluadas si la expresión2 es verdadera>
elseif <expresión3>
        <órdenes evaluadas si la expresión3 es verdadera>
elseif
          …
          …
else
        <órdenes evaluadas si ninguna otra expresión es verdadera>
end
```

Ejemplo:

```
>> b = 2;
>>      if b == 0        % ponemos == porque no es una asignación sino una expresión lógica
            disp ('b vale 0')
        elseif b == 1
            disp ('b vale 1')
        elseif b == 2
            disp ('b vale 2')
        elseif b == 3
            disp ('b vale 3')
        else
            disp ('b no vale ni 0 ni 1 ni 2 ni 3')
        end

b vale 2                 % es lo que devuelve por pantalla
```

SENTENCIA BREAK

Si queremos que en un momento dado termine la ejecución de un bucle **for** o un bucle **while** usaremos **break.**

SENTENCIA CONTINUE

La sentencia **continue** hace que se pase inmediatamente a la siguiente iteración del bucle **for** o del bucle **while** saltando todas las órdenes que hay entre el **continue** y el fin del bucle en la iteración actual.

Ejemplo:

Podemos mezclar en un programa varias sentencias de este estilo. Aquí podemos ver un programa que escribe por pantalla los primos del 1 al 100 usando las sentencias if, while y for.

```
disp('Estos son los números primos menores de 100')
disp(2)
for i=2:100
    n=2;
    while n <= sqrt(i)
        if rem(i,n)==0
            n=i;
        else n=n+1;
        end
    end
    if n~=i disp(i)
    end
end
```

FUNCIONES EN M-ARCHIVOS

Existen dos tipos de M-archivo, es decir, de archivos con extensión ***.m**. Un tipo son los *ficheros de comandos* (es un archivo *stript*) y el otro son la *funciones.*

Un *fichero de comandos* contiene simplemente un conjunto de comandos que se ejecutan sucesivamente cuando se teclea el nombre del fichero en la línea de comandos de Matlab o se incluye dicho nombre en otro fichero ***.m**.

Las *funciones* permiten definir funciones análogas a las de Matlab, con su nombre, argumentos y valores de salida. La primera línea que no sea comentario debe empezar por la palabra **function**, seguida por los valores de salida (entre corchetes **[ ]** y separados por comas si hay más de uno), el signo igual (=) y el nombre de la función seguido de los argumentos (entre paréntesis **( )** y separados por comas):

function [a,b,c] = nombre_función (x,y,z)

En las líneas siguientes escribimos los argumentos de salida a partir de los de entrada. El nombre de la función y el nombre del archivo deben ser idénticos y no empezar por cifra sino por letra.

Todas las variables dentro de una función se aíslan del espacio de trabajo de Matlab. Las únicas conexiones entre las variables dentro de una función y el espacio de trabajo de Matlab son las variables de entrada y salida.

El número de variables de entrada pasadas a una función está disponible dentro de la función en la variable **nargin** y el número de variables de salida solicitadas cuando una función es llamada, está disponible dentro de la función en la variable **nargout**.

Debemos tener siempre en cuenta que los argumentos pueden ser vectores, luego si queremos que las operaciones se hagan elemento a elemento y no vectorialmente debemos usar el punto.

Ejemplo:

En un M-archivo guardamos lo siguiente:

```
function [suma,resta] = calculos (x,y)          % la función se llama calculos
suma = x + y;
resta = x - y;
```

Ahora si queremos usarlo, debemos escribir por ejemplo en la ventana de comandos:

```
>> x = 1:10; y = 16:-2:-2;

>> [a,b] = calculos (x,y)
a =
   17   16   15   14   13   12   11   10    9    8
b =
  -15  -12   -9   -6   -3    0    3    6    9   12
```

```
>> x = [1 5; 3 -2; 3 7; 4 -1; 0 2]; y = [16 -1; 0 4; 1 5; -1 0; -1 3];

>> [a,b] = calculos (x,y)
a =
    17     4
     3     2
     4    12
     3    -1
    -1     5
b =
   -15     6
     3    -6
     2     2
     5    -1
     1    -1
```

ANÁLISIS DE DATOS

Matlab ejecuta análisis estadístico sobre conjuntos de datos. Estos conjuntos de datos se almacenan en matrices orientadas por columnas. Matlab incluye, entre otras, las siguientes funciones estadísticas:

Función	¿Qué hace?
corrcoef (X)	coeficientes de correlación
cov (X)	matriz de covarianzas
cumprod (X)	producto acumulativo de columnas
cumsum (X)	suma acumulativa de columnas
diff (X)	diferencias entre elementos adyacentes de X
hist (X)	histograma o diagrama de barras
iqr (X)	rango intercuartílico de la muestra
max (X)	máximo de cada columna
mean (X)	media de los valores de vectores y columnas
median (X)	mediana de los valores de vectores y columnas
min (X)	mínimo de cada columna
prod (X)	producto de elementos en columnas
rand (n)	números aleatorios distribuidos uniformemente
randn (n)	números aleatorios distribuidos normalmente
range (X)	rango de cada columna
sort (X)	ordena columnas en orden ascendente
std (X)	desviación estándar de la muestra
sum (X)	suma de elementos en cada columna
tabulate (v)	tabla de frecuencias del vector
var (X)	varianza de la muestra

Ejemplos:

```
>> X = [5 7 9 2 9; 3 1 7 5 1; 3 9 2 7 5; 1 5 5 1 8]
X =
    5    7    9    2    9
    3    1    7    5    1
    3    9    2    7    5
    1    5    5    1    8

>> cumprod (X)                % matriz de productos acumulados
ans =
     5     7     9     2     9
    15     7    63    10     9
    45    63   126    70    45
    45   315   630    70   360

>> cumsum (X)                 % matriz de sumas acumuladas
ans =
     5     7     9     2     9
     8     8    16     7    10
    11    17    18    14    15
    12    22    23    15    23

>> mean (X)                   % media de cada columna
ans =
   3.0000    5.5000    5.7500    3.7500    5.7500

>> median (X)                 % mediana de cada columna
ans =
   3.0000    6.0000    6.0000    3.5000    6.5000

>> prod (X)                   % producto de todos los elementos de cada columna
ans =
   45   315   630    70   360

>> sort (X)                   % ordena los valores de cada columna
ans =
    1    1    2    1    1
    3    5    5    2    5
    3    7    7    5    8
    5    9    9    7    9

>> sum (X)                    % suma de todos los elementos de cada columna
ans =
   12    22    23    15    23

>> var (X)                    % varianza de los elementos de cada columna
ans =
   2.6667   11.6667    8.9167    7.5833   12.9167
```

```
>> max (X)                    % valor máximo de cada columna
ans =
    5     9     9     7     9

>> min (X)                    % valor mínimo de cada columna
ans =
    1     1     2     1     1

>> iqr (X)                    % rango intercuartílico de cada columna
ans =
   2.0000    5.0000    4.5000    4.5000    5.5000

>> Y = [5 7 9 2 9 3 1 7 5 1 3 9 2 7 5 1 5 5 1 8];

>> tabulate (Y)               % tabla de frecuencias generada a partir de una serie de valores
  Value    Count   Percent
      1        4    20.00%
      2        2    10.00%
      3        2    10.00%
      4        0     0.00%
      5        5    25.00%
      6        0     0.00%
      7        3    15.00%
      8        1     5.00%
      9        3    15.00%

>> range (X)                  % rango de cada columna (diferencia  entre el máximo y el mínimo)
ans =
    4     8     7     6     8
```

POLINOMIOS

RAÍCES

Un polinomio se representa por un vector fila con sus coeficientes en orden descendiente, no debemos olvidar colocar los términos con coeficiente nulo.

Así por ejemplo si queremos indicar el polinomio $5x^4 + 2x^2 – x + 7$ escribiríamos [5 0 2 -1 7].

Para encontrar las raíces de un polinomio **p** usaremos la función **roots (p)**.

Si conocemos las raíces de un polinomio es posible construir el polinomio asociado mediante la función **poly (r)**.

Matlab trabaja con los polinomios como vectores fila y con las raíces como vectores columnas.

Ejemplos:

```
>> p = [1 -9 13 9 -14];        % representa al polinomio x^4-9x^3+13x^2-9x-14

>> roots (p)                   % calcula sus raíces
ans =
   7.0000
  -1.0000
   2.0000
   1.0000

>> poly (ans)                  % devuelve el polinomio generado por  esas cuatro raíces
ans =
   1.0000   -9.0000   13.0000    9.0000  -14.0000
```

OTRAS CARACTERÍSTICAS

Función	¿Qué es?
conv (p,q)	multiplica los dos polinomios p y q
deconv (c,q)	divide el polinomio c entre q
polyder (p)	calcula la derivada del polinomio p
polyder (p,q)	calcula la derivada del producto de los polinomios p y q
polyval (p,A)	evalúa el polinomio p en todos los valores de la matriz A

Matlab no tiene incorporada una función para sumar polinomios.

Ejemplos:

```
>> p = [1 2 7];
>> q = [1 3 6];                % polinomios

>> c = conv (p,q)              % producto de los polinomios p y q
c =
    1     5    19    33    42

>> deconv (c,q)                % cociente de dividir el polinomio c entre el polinomio q
ans =
    1     2     7

>> polyder (p)                 % derivada del polinomio p
ans =
    2     2

>> polyder (p,q)               % derivada del producto de los polinomios p y q
ans =
    4    15    38    33
```

```
>> polyval (p, [0 1 5] )          % evalúa el polinomio en 0, 1 y 5, es decir, halla p(0), p(1) y p(5)
ans =
     7    10    42

>> polyval (p, [0 1 2; -1 -2 -3; 4 0 7] )          % igual pero toma los valores de una matriz
ans =
     7    10    15
     6     7    10
    31     7    70
```

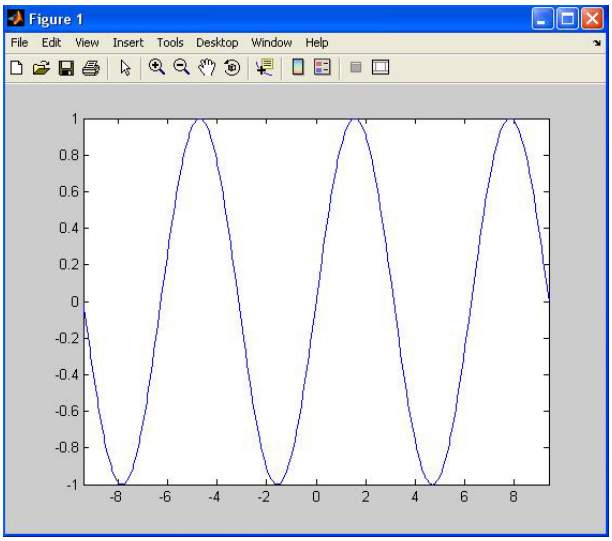
ANÁLISIS NUMÉRICO

REPRESENTACIÓN GRÁFICA

Existe la función **fplot** que evalúa la función que se desea representar en la gráfica de salida. Como entrada, necesita conocer el nombre de la función como una cadena de caracteres y el rango de representación como un vector de dos elementos: **fplot ('nombre', [valor min, valor max])**.

Ejemplo:

```
>> fplot ('sin', [-3*pi,3*pi] )
```



OTRAS CARACTERÍSTICAS

Función	¿Qué hace?
diff ('f')	derivada de la función respecto a x
diff ('f',t)	derivada parcial de la función respecto a t
diff ('f',n)	derivada n-ésima de la función respecto a x
feval ('f',a)	evalúa la función en a
fminbnd ('f',a,b)	calcula el mínimo de una función de una variable
fzero ('f',a)	busca el cero de una función unidimensional f más próximo al punto a
quad ('f',a,b)	aproxima la integral definida (según la cuadratura de Simpson)
trapz (x,y)	integral numérica trapezoidal de la función formada al emparejar los puntos de los vectores x e y

(**f** función, **n** número natural, **a** y **b** valores numéricos, **x** e **y** vectores del mismo tamaño)

Matlab incorpora una serie de funciones para resolver ecuaciones diferenciales ordinarias. Si se trata de un problema rígido deberíamos usar: **ode15s**, **ode23s**, **ode23t** u **ode23tb**, si por el contrario se trata de un problema sin rigidez: **ode113**, **ode 23** y **ode45**. Para saber más de estas funciones consultar la ayuda de Matlab.

Ejemplos:

```
> diff ('sin (7*x) ')                    % derivada respecto a x
ans =
7*cos(7*x)

>> diff ('(exp (x) * cos (3*x*y))','y')          % derivada parcial respecto a y
 ans =
 -3*exp(x)*sin(3*x*y)*x

>> diff ('(sin (x^2))',2)                 % segunda derivada respecto a x
 ans =
 -4*sin(x^2)*x^2+2*cos(x^2)

>> feval ('cos',pi)                       % evalúa el coseno en el valor pi
ans =
   -1

>> feval ('cos', [0 pi/3 pi] )            % para evaluar en varios puntos debemos darlo como un vector
ans =
   1.0000    0.5000   -1.0000

>> feval (@cos, [0 pi/3 pi] )             % es lo mismo que lo anterior, da igual comillas que el @
ans =
   1.0000    0.5000   -1.0000

>> fminbnd (@sind,0,360)                  % valor del dominio donde la función toma el mínimo
ans =
  270.0000
```

```
>> fzero ('sind',100)          % el valor más próximo a 100 donde la función seno vale cero
ans =
   180

>> quad ('sin',0,pi)           % integral definida del seno desde 0 hasta pi
ans =
    2.0000

> x = 0:4; y = [0 2 2 1 6];
>> trapz (x,y)
ans =
     8
```

CONVERTIR UN FICHERO (*.m) EN UN EJECUTABLE (*.exe)

Si tenemos un fichero *.m, lo primero que debemos hacer es asegurarnos de que sea una función, para ello en la primera línea del fichero debe aparecer:

function nombre (el nombre de la función debe coincidir con el nombre del fichero)

Ahora debemos situarnos en el directorio donde tengamos el fichero que queremos transformar usando el comando **cd**, por ejemplo:

```
>> cd 'C:\Documents and Settings\Escritorio\Prueba'
```

Lo que debemos escribir a continuación es el comando **mcc** seguido de **–m** y el nombre del fichero:

```
>> mcc -m nombre
```

Con esto nos aparecerá en el mismo directorio donde estamos un ejecutable con el mismo nombre. También aparecerán una carpeta y varios archivos.

Ejemplo:

Creamos un fichero que va a ser una función que a su vez va a llamar a otras dos funciones que también hemos creado nosotros:

Fichero algebra.m:

```
% algebra
function algebra
x = input ('Escribe un número:  ');
y = input ('Escribe otro número:  ');
disp ('La suma es...')
suma (x,y)
disp ('La resta es...')
resta (x,y)
pause                       %para que no se cierre la ventana automáticamente al ejecutarse
```

Fichero suma.m:

```
% suma
function m = suma (tt,xx)
m = tt + xx;
```

Fichero resta.m:

```
% resta
function m = resta (tt,xx)
m = tt - xx;
```

Después de situarnos en el directorio correspondiente escribimos:

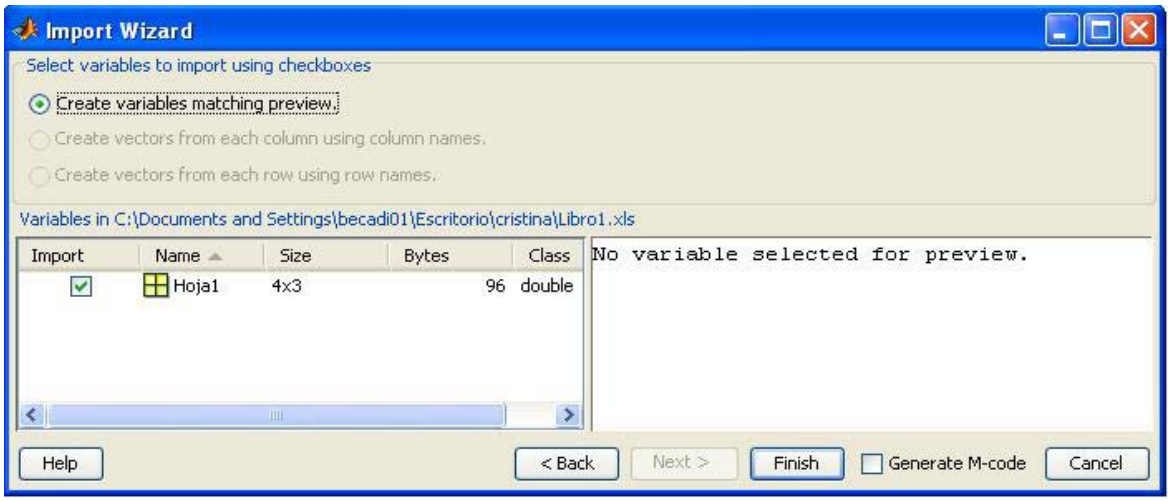
```
>> mcc -m álgebra
```

IMPORTAR Y EXPORTAR FICHEROS DE DATOS

Si tenemos un fichero de datos y queremos usarlo en Matlab podemos importarlo para evitarnos copiar de nuevo todos los datos.

Para ello usaremos un botón  que se encuentra en la ventana workspace. Vemos que al situar el ratón sobre él aparece un letrero diciendo para lo que sirve (import data):

Al pinchar en él se abre una ventana. Debemos localizar el fichero que queremos importar y pinchar en el botón donde pone Abrir. Aparecerá una nueva ventana similar a ésta:

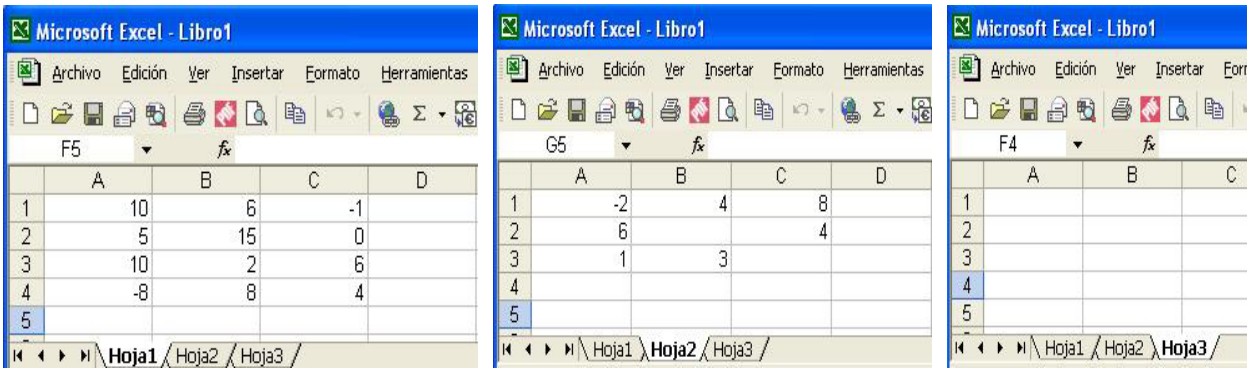


(Aquí estamos importando un fichero de datos .xls de Excel con el nombre Libro1 pero al importarlo lo renombra como Hoja1 ya que el fichero en cuestión tenía 3 hojas, pero sólo la Hoja1 tenía datos)

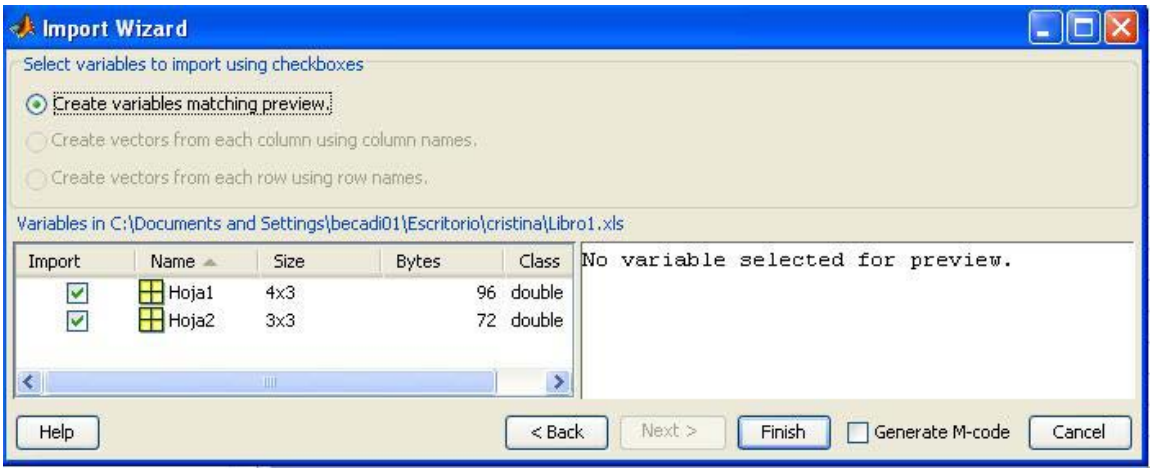
Pinchamos en el botón Finish y ya tendremos el fichero de datos convertido en una matriz en Matlab. Si queremos visualizarla sólo tenemos que llamarla ya que se almacenará con el mismo nombre.

Ejemplo:

Queremos importar un fichero de datos de Excel con dos hojas (la Hoja3 está vacía):



Al importar el fichero nos aparece la ventana siguiente (sólo aparecen dos matrices porque la Hoja3 está vacía):



Pinchamos en Finish y aceptamos. Si queremos ver cómo ha guardado los datos basta llamar a las matrices con el nombre que hayan sido almacenadas. (Los espacios en blanco los ha guardado como NaN).

```
>> Hoja1
Hoja1 =
    10     6    -1
     5    15     0
    10     2     6
    -8     8     4

>> Hoja2
Hoja2 =
    -2     4     8
     6   NaN     4
     1     3   NaN
```

Para exportar una matriz podemos convertirla en texto haciendo lo siguiente:

Primero escribimos:

>> diary my_data.out

Después escribimos bien la matriz (o bien la llamamos si ya estuviera almacenada en el workspace). Es este caso creamos la matriz A:

>> A = [0 1;2 3;4 5;6 7;8 9]

A =

0 1
2 3
4 5
6 7
8 9

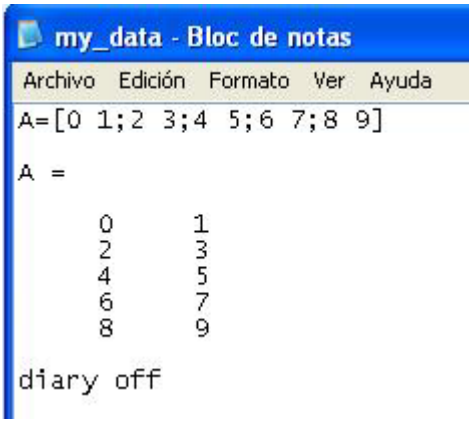
Escribimos:

>> diary off

Ahora para abrirlo buscamos el documento llamado my_data.out que se encuentra en:

C:\Archivos de programa\MATLAB\R2006b\work

Podemos abrirlo con cualquier editor de texto.



EJERCICIOS PROPUESTOS

Calcula el resultado de sumar 15 y 6:

```
>> 15+6
ans =
   21
```

Guarda en la variable x el resultado de sumar 15 y 6:

```
>> x=15+6
x =
   21
```

Haz que aparezca por pantalla el valor almacenado en la variable x:

```
>> x
x =
   21
```

Guarda en la variable y el resultado de multiplicar 12 y 2:

```
>> y=12*2
y =
   24
```

Realiza la suma de las variables x e y:

```
>> x+y
ans =
   45
```

Guarda en la variable z el resultado de restarle a la variable y la variable x:

```
>> z=y-x;
```

Haz que aparezca por pantalla el valor almacenado en la variable z:

```
>> z
z =
    3
```

Calcula el coseno de π (tomando el ángulo en radianes):

```
>> cos(pi)
ans =
   -1
```

Calcula el coseno de 180º (tomando el ángulo en grados sexagesimales):

```
>> cosd(180)
ans =
   -1
```

Calcula la exponencial en 1 (es decir, el número e):

```
>> exp(1)
ans =
   2.7183
```

Calcula la raíz cuadrada de -16:

```
>> sqrt(-16)
ans =
       0 + 4.0000i
```

Calcula el resultado de la división de 2 entre 3:

```
>> 2/3
ans =
   0.6667
```

Cambia a formato con 15 decimales:

```
>> format long
```

Vuelve a calcular el resultado de la división de 2 entre 3:

```
>> 2/3
ans =
  0.666666666666667
```

Cambia a formato con solo 4 decimales:

```
>> format short
```

Vuelve a calcular el resultado de la división de 2 entre 3:

```
>> 2/3
ans =
   0.6667
```

Haz que aparezcan por pantalla las variables que estás utilizando:

```
>> who
Your variables are:
ans  x    y    z
>> whos
  Name      Size            Bytes  Class     Attributes
  ans       1x1                 8  double
  x         1x1                 8  double
  y         1x1                 8  double
  z         1x1                 8  double
```

Borra la variable z:

```
>> clear z
```

Vuelve a hacer que aparezcan por pantalla las variables que estás utilizando:

```
>> who
Your variables are:
ans  x    y
```

Crea el vector v = (1,2,3,4) de modo que no se vuelva a escribir en pantalla:

```
>> v=[1 2 3 4];
```

Crea el vector w = (5,6,7,8) y deja que lo vuelva a escribir en pantalla:

```
>> w=[5 6 7 8]
w =
    5     6     7     8
```

Calcula el vector traspuesto de v:

```
>> v'
ans =
    1
    2
    3
    4
```

Crea un vector llamado v2 donde sus elementos vayan desde el 2 al 17 creciendo de 3 en 3:

```
>> v2=2:3:17
v2 =
    2    5    8   11   14   17
```

Crea un vector v3 donde sus elementos vayan desde el 2 al 20 y que en total tenga 10 elementos:

```
>> v3=linspace(2,20,10)
v3 =
    2    4    6    8   10   12   14   16   18   20
```

Averigua cuál es el cuarto elemento del vector v3:

```
>> v3(4)
ans =
    8
```

Crea la matriz M= $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$**:**

```
>> M=[1 2 3 4;5 6 7 8;9 10 11 12]
M =
    1    2    3    4
    5    6    7    8
    9   10   11   12
```

Calcula la traspuesta de la matriz M:

```
>> M'
ans =
    1    5    9
    2    6   10
    3    7   11
    4    8   12
```

Halla la fila 2 de la matriz M:

```
>> M(2,:)
ans =
    5    6    7    8
```

Calcula el rango de M:

```
>> rank(M)
ans =
    2
```

Calcula la traza de la matriz M:

```
>> trace(M)
ans =
   19
```

Crea la matriz identidad de tamaño 4:

```
>> eye(4)
ans =
    1    0    0    0
    0    1    0    0
    0    0    1    0
    0    0    0    1
```

Crea la matriz nula de tamaño 3x3:

```
>> zeros(3)
ans =
    0    0    0
    0    0    0
    0    0    0
```

Crea la matriz cuadrada de unos de tamaño 2x2:

```
>> ones(2)
ans =
    1    1
    1    1
```

Averigua las dimensiones de la matriz M:

```
>> size(M)
ans =
    3    4
```

Crea una matriz llamada M2 que tenga por diagonal el vector v y el resto sean todo ceros:

```
>> M2=diag(v)
M2 =
    1    0    0    0
    0    2    0    0
    0    0    3    0
    0    0    0    4
```

Haz que aparezca por pantalla la matriz triangular inferior a partir de M:

```
>> tril(M)
ans =
    1    0    0    0
    5    6    0    0
    9   10   11    0
```

Haz que aparezca por pantalla la matriz triangular superior a partir de M:

```
>> triu(M)
ans =
    1    2    3    4
    0    6    7    8
    0    0   11   12
```

Calcula una matriz que tenga por elementos todos los elementos de la matriz M elevados al cuadrado:

```
>> M.^2
ans =
    1    4    9   16
   25   36   49   64
   81  100  121  144
```

Elimina de la matriz M su tercera columna:

```
>> M(:,3)=[]
M =
    1    2    4
    5    6    8
    9   10   12
```

Calcula el determinante de M:

```
>> det(M)
ans =
    0
```

Guarda en p el polinomio $x^3 - x^2 - 26x - 24$:

```
>> p=[1 -1 -26 -24];
```

Calcula las raíces del polinomio p:

```
>> roots(p)
ans =
   6.0000
  -4.0000
  -1.0000
```

Evalúa el polinomio p cuando x = 1:

```
>> polyval(p,1)
ans =
  -50
```

Evalúa el polinomio p en todos los valores del vector w:

```
>> polyval(p,w)
ans =
  -54    0   88  216
```

Crea un polinomio q que tenga por raíces los elementos del vector v:

```
>> q=poly(v)
q =
    1  -10   35  -50   24
```

Calcula la multiplicación de los polinomios p y q:

```
>> conv(p,q)
ans =
    1  -11   19  151 -596  436  576 -576
```

Calcula la división del polinomio obtenido como solución entre el polinomio q:

```
>> deconv(ans,q)
ans =
    1    -1   -26   -24
```

Escribe por pantalla el valor de los vectores v y w:

```
>> v
v =
    1     2     3     4
>> w
w =
    5     6     7     8
```

Calcula el producto de los vectores elemento a elemento:

```
>> v.*w
ans =
    5    12    21    32
```

Calcula el producto escalar de los vectores v y w:

```
>> dot(v,w)
ans =
   70
```

Calcula el producto del vector traspuesto de v con el vector w:

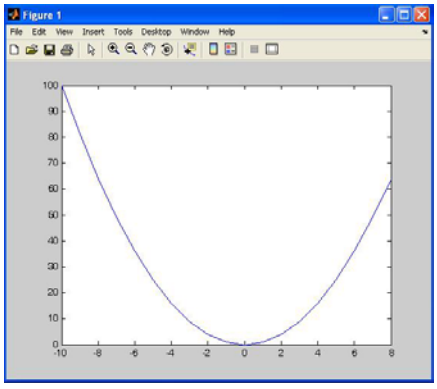
```
>> v'*w
ans =
    5     6     7     8
   10    12    14    16
   15    18    21    24
   20    24    28    32
```

Define el vector x = (-10,-9,-8,…,6,7,8) y el vector y que sea el cuadrado de cada elemento:

```
>> x=(-10:8); y=x.^2;
```

Ahora dibuja la gráfica formada por esos dos vectores:

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



Guarda en la variable a la palabra guacamole:

```
>> a='guacamole';
```

Haz que aparezca en pantalla la representación ASCII del valor almacenado en la variable a:

```
>> abs(a)
ans =
  103   117    97    99    97   109   111   108   101
```

Crea un pequeño programa que escriba por pantalla BUENOS DÍAS y a continuación los 15 primeros números pares (usando un for):

```
disp('BUENOS DÍAS')
for i=1:15
    disp(2*i)
end
```

Calcula la derivada de la función f(x) = sin (2x) + tan (x^2):

```
>> diff('sin(2*x)+tan(x^2)')
ans =
2*cos(2*x)+2*(1+tan(x^2)^2)*x
```

COMANDOS QUE APARECEN EN ESTE MANUAL

abs
all
angle
ans
any
area
asin
asind
asinh
axis

bar
barh
break

calendar
cat
ceil
cell
celldisp
cellplot
clabel
clc
clear
clock
colorbar
colormap
complex
cond
conj
continue
contour
contour3
conv
corrcoef
cos
cot
cov
cross
csc
cumprod
cumsum

date
deconv
det
diag
diff
disp
dot
double

eig
else
elseif
end
eps
exist
exp
expm
eye

feval
fieldnames
fill
fill3
find
fix
fliplr
flipud
floor
fminbnd
for
format bank
format hex
format long
format long e
format long eng
format long g
format rat
format short
format short e
formatshort eng
format short g
format +
fplot
function
funm
fzero

gcd
grid

help
hilb
hist
hold on
hold off

i
if
imag
inf
inv
invhilb
iqr
iscell
ischar
isempty
isequal
isfield
isfinite
isfloat
isinf
isinteger
islogical
isnan
isnumeric
isprime
isreal
isscalar
isspace
isstruct

j

lcm
length
linspace
load
log
loglog
logm
logspace
log2
log10
lookfor
lu

magic
max
mcc
mean
median
mesh
meshgrid
min
mod

nan
nargin
nargout
norm
normest
nthroot
null
num2cell

ones
orth
ode113
ode15s
ode23
ode23s
ode23t
ode23tb
ode45
pcolor
pi
pie
pinv
plot
poly
polyder
polyval
prod

qr
quad
quit

rand
randn
range
rank
real
realmax
realmin
rem
reshape
rmfield
roots
rot90
rose
round
rref

save
sec
semilogx
semilogy
setstr
sign
sin
sind
sinh
size
sort
sqrt
sqrtm
stairs
std
stem
struct
subplot
sum
surf

tabulate
tan
text
title
trace
trapz
triu
tril

var
view

xlabel
xor

ylabel

while
who
whos

zeros