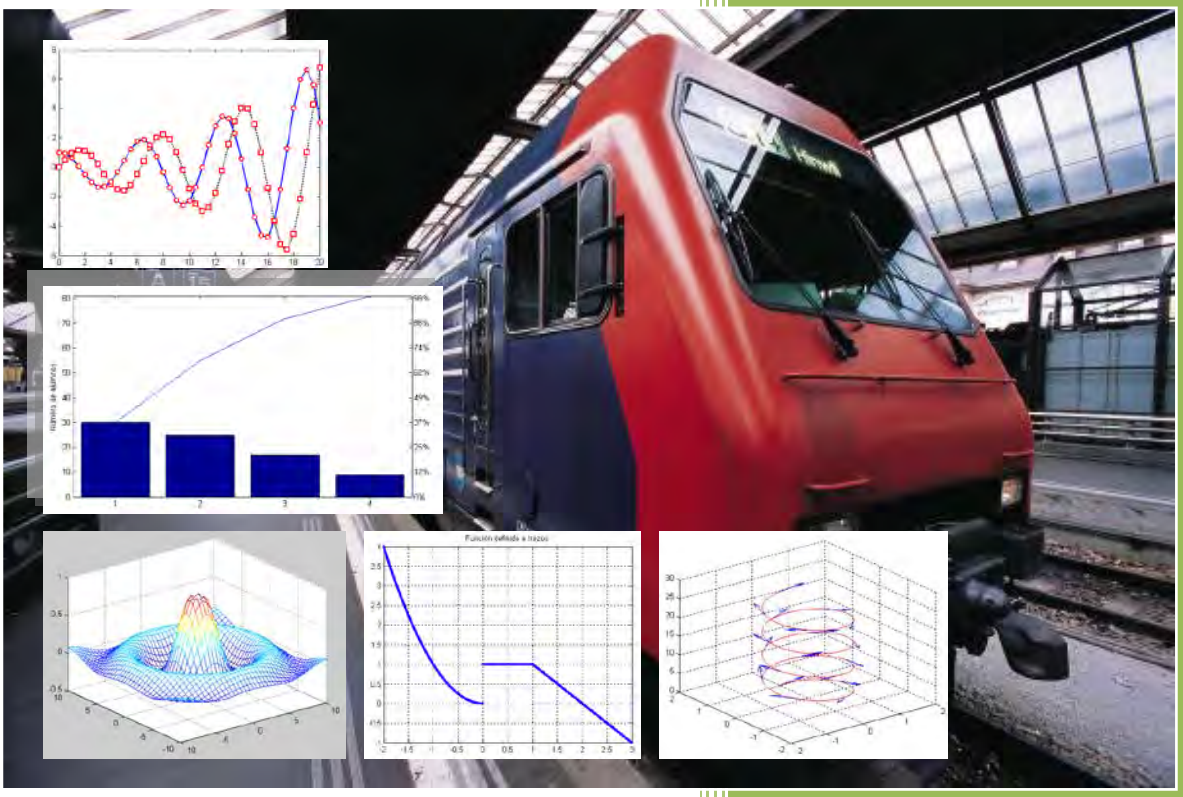
M A T L A B



L. Maria Pimentel Herrera

UNI - CEPS

Contenido

Introducción

MatLab significa MATrix LABoratory.

Es un programa para hacer computación numérica.

Fue diseñado para manipular matrices y ploteo de datos.

Ahora incluye funciones para: analizar datos, procesar señales, optimizar funciones.

Contiene funciones para los gráficos 2-Dy 3-D con su respectiva animación.

Matlab permite leer y escribir archivos .MAT, .TXT, etc.

Tiene interfaces con otros lenguajes.

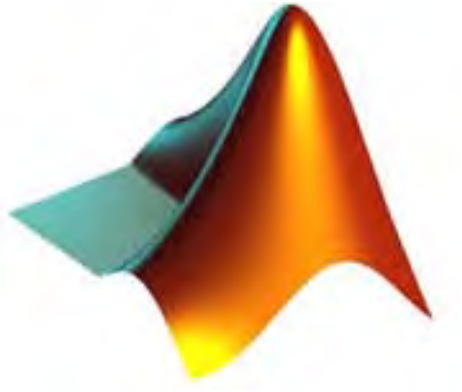
Permite la computación simbólica con el Maple.

Áreas de aplicación

- Ingeniería eléctrica y mecánica
- Informática
- Matemáticas
- Ingeniería aeroespacial y automotriz
- Ingeniería química y biomédica
- Economía y finanzas

Matlab

- Trabaja números escalares (reales y complejos), con caracteres y otras estructuras de datos.
- Tiene un lenguaje de programación propio.
- Permite un rápido prototipeo de aplicaciones científicas.
- Pero puede ser más lento que C/C++ o Fortran.
- Dispone de código básico y toolboxes.

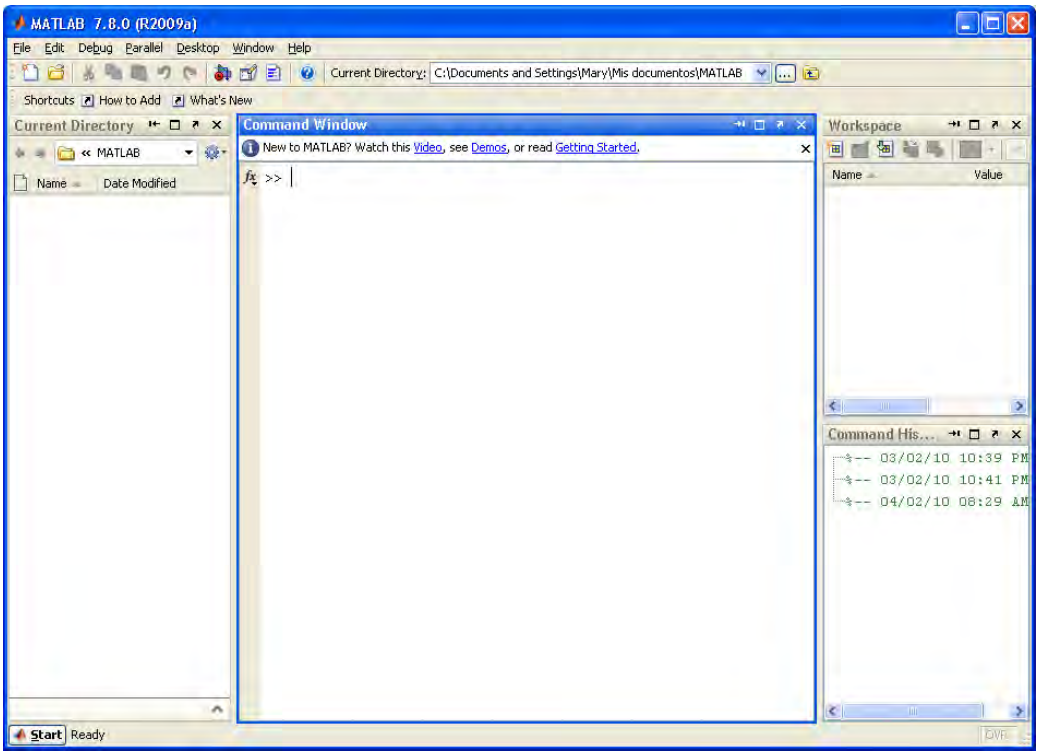


Entorno de trabajo de matlab

El entorno de trabajo de MATLAB es muy gráfico e intuitivo, similar al de otras aplicaciones profesionales de ***Windows***.

Las componentes más importantes del entorno de trabajo de MATLAB son las siguientes:

1. El ***Escritorio de Matlab*** (***Matlab Desktop***), que es la ventana o contenedor de máximo nivel en la que se pueden situar (*to dock*) las demás componentes.
2. Las componentes individuales, orientadas a tareas concretas, entre las que se puede citar:
    a. La ventana de comandos (***Command Window***),
    b. La ventana histórica de comandos (***Command History***),
    c. El espacio de trabajo (***Workspace***),
    d. La plataforma de lanzamiento (***Launch Pad***),
    e. El directorio actual (***Current Directory***),
    f. La ventana de ayuda (***Help***)
    g. El editor de ficheros y depurador de errores (***Editor&Debugger***),
    h. El editor de vectores y matrices (***Array Editor***).
    i. La ventana que permite estudiar cómo se emplea el tiempo de ejecución (***Profiler***).



Path de matlab

establecer el camino de búsqueda (search path)

MATLAB puede llamar a una gran variedad de funciones, tanto propias como programadas por los usuarios. Puede incluso haber ***funciones distintas con el mismo nombre***. Interesa saber cuáles son las reglas que determinan qué función o qué fichero ****.m*** es el que se va a ejecutar cuando su nombre aparezca en una línea de comandos del programa. Esto queda determinado por el *camino de búsqueda* (***search path***) que el programa utiliza cuando encuentra el nombre de una función.

El ***search path*** de MATLAB es una lista de directorios que se puede ver y modificar a partir de la línea de comandos, o utilizando el cuadro de diálogo ***Set Path***, del menú ***File***. El comando ***path*** hace que se escriba el ***search path*** de MATLAB (el resultado depende de en qué directorio esté instalado MATLAB):

```
>> path
```

Workspace browser y array editor

El espacio de trabajo de MATLAB (***Workspace***) es el conjunto de variables y de funciones de usuario que en un determinado momento están definidas en la memoria del programa o de la función que se está ejecutando. Para obtener información sobre el ***Workspace*** desde la línea de comandos se pueden utilizar los comandos ***who*** y ***whos***. El segundo proporciona una información más detallada que el primero.

```
>>x=9
x=
     9
>>y=15
y=
     15
>>whos
     Name  Size  Bytes      Class
        x   1x1    8     double array
        y   1x1    8     double array
     Grand total is 2 elements using 16 bytes
```

La ventana ***Workspace*** constituye un entorno gráfico para ver las variables definidas en el espacio de trabajo. Se activa con el comando ***View/Workspace***. La ventana ***Workspace*** cuando se abre desde un determinado programa. Haciendo doble clic por ejemplo sobre la matriz **BARS** aparece una nueva ventana (o pestaña, si la ventana ya existía) del ***Array Editor***, en la que se muestran y pueden ser modificados los elementos de dicha matriz.

Formatos de salida

Respecto a l os formatos numéricos con que MATLAB muestra los resultados (recuérdese que siempre calcula con doble precisión, es decir con unas 16 cifras decimales equivalentes), y son las siguientes:

short	coma fija con 4 decimales (***defecto***)
long	coma fija con 15 decimales
hex	cifras hexadecimales
bank	números con dos cifras decimales
short e	notación científica con 4 decimales
short g	notación científica o decimal, dependiendo del valor
long e	notación científica con 15 decimales
long g	notación científica o decimal, dependiendo del valor
rational	expresa los números racionales como cocientes de enteros

Estos formatos se pueden cambiar también desde la línea de comandos anteponiendo la palabra ***format***. Por ejemplo, para ver las matrices en formato ***long*** habrá que ejecutar el comando:

```
>>format long
>>1/3
ans =
        0.33333333333333
>>format      % Vuelve al formato estándar que es el de 4 cifras decimales
>> m=17/3;
>> c=9/1974;
```

Comando	Representación de **m**	Representación de **c**
format short	5.6667	
format short e	5.6667e+000	
format long	5.666666666666667	
format long e	5.666666666666667e+000	
format hex	4016aaaaaaaaaab	
format bank	5.67	
format rat	17/3	

Guardar variables y estados de una sesión

Comandos *save* y *load*

En muchas ocasiones puede resultar interesante interrumpir el trabajo con MATLAB y poderlo recuperar más tarde en el mismo punto en el que se dejó (con las mismas variables definidas, con los mismos resultados intermedios, etc.). Hay que tener en cuenta que al salir del programa todo el contenido de la memoria se borra automáticamente.

Para guardar el estado de una sesión de trabajo existe el comando ***save***. Si se teclea:

```
>> save
```

antes de abandonar el programa, se crea en el ***directorio actual*** un fichero binario llamado ***matlab.mat*** (o ***matlab***) con el estado de la sesión (excepto los gráficos, que por ocupar mucha memoria hay que guardar aparte). Dicho estado puede recuperarse la siguiente vez que se arranque el programa con el comando:

```
>> load
```

Esta es la forma más básica de los comandos ***save*** y ***load***. Se pueden guardar también matrices y vectores de forma selectiva y en f icheros con nombre especificado por el usuario. Por ejemplo, el comando (sin comas entre los nombres de variables):

```
>> save filename A x y
```

guarda las variables **A**, **x** e **y** en un f ichero binario llamado ***filename.mat*** (o ***filename***). Para recuperarlas

en otra sesión basta teclear:

```
>> load filename
```

Si no se indica ninguna variable, se guardan todas las variables creadas en esa sesión.

Guardar sesión y copiar salidas

Comando *diary*

Los comandos ***save*** y ***load*** crean ficheros binarios o ASCII con el estado de la sesión. Existe otra forma más sencilla de almacenar en un fichero un texto que describa lo que el programa va haciendo (la entrada y salida de los comandos utilizados). Esto se hace con el comando ***diary*** en la forma siguiente:

```
>> diary filename.txt

...
```

```
>> diary off
...
>> diary on
...
```

El comando ***diary off*** suspende la ejecución de ***diary*** y ***diary on*** la reanuda. El simple comando ***diary*** pasa de ***on*** a ***off*** y viceversa. Para poder acceder al fichero ***filename.txt*** con ***Notepad*** o ***Word*** es necesario que ***diary*** esté en ***off***. Si en el comando ***diary*** no se incluye el nombre del fichero se utiliza por defecto un fichero llamado ***diary*** (sin extensión).

Líneas de comentarios

El carácter ***tanto por ciento*** (%) indica comienzo de comentario. Cuando aparece en una línea de comandos, el programa supone que todo lo que va desde ese carácter hasta el fin de la línea es un comentario.

Otra forma de comentar bloques de sentencias (similar a la utilizada en C/C++ con /* y */) es encerrar las líneas que se desea inutilizar entre los caracteres **%{** y **%}**. Los bloques comentados pueden incluirse dentro de otros bloques comentados más amplios (bloques anidados).

Help!

Este comando nos permite solicitar ayuda sobre cualquier comando o función que se encuentre instalada en Matlab.

Escribiendo help en la línea de comando, el programa devuelve un listado de todas las librerías instaladas. Entonces:

```
>>help
HELP topics:
matlab\general       - General purpose commands.
matlab\ops           - Operators and special characters.
matlab\lang          - Programming language constructs.
matlab\elmat         - Elementary matrices and matrix manipulation.
matlab\randfun       - Random matrices and random streams.
matlab\elfun         - Elementary math functions.
matlab\specfun       - Specialized math functions.
matlab\matfun        - Matrix functions - numerical linear algebra.
matlab\datafun       - Data analysis and Fourier transforms.
matlab\polyfun       - Interpolation and polynomials.
```

matlab\funfun - Function functions and ODE solvers.
matlab\sparfun - Sparse matrices.
matlab\scribe - Annotation and Plot Editing.
matlab\graph2d - Two dimensional graphs.
matlab\graph3d - Three dimensional graphs.
……..

- *clic en matlab\general te mostrará*

General purpose commands.
MATLAB Version 7.8 (R2009a) 15-Jan-2009
General information.

syntax	- Help on MATLAB command syntax.
demo	- Run demonstrations.
ver	- MATLAB, Simulink and toolbox version information.
version	- MATLAB version information.
verLessThan	- Compare version of toolbox to specified version string.

Managing the workspace.

who	- List current variables.
whos	- List current variables, long form.
clear	- Clear variables and functions from memory.
onCleanup	- Specify cleanup work to be done on function completion.
pack	- Consolidate workspace memory.
load	- Load workspace variables from disk.
save	- Save workspace variables to disk.
saveas	- Save Figure or model to desired output format.
memory	- Help for memory limitations.
recycle	- Set option to move deleted files to recycle folder.
quit	- Quit MATLAB session.
exit	- Exit from MATLAB.

Managing commands and functions.

what	- List MATLAB-specific files in directory.
type	- List M-file.
open	- Open files by extension.
which	- Locate functions and files.

pcode	- Create pre-parsed pseudo-code file (P-file).
mex	- Compile MEX-function.
inmem	- List functions in memory.
namelengthmax	- Maximum length of MATLAB function or variable name.

Managing the search path.

path	- Get/set search path.
addpath	- Add directory to search path.
rmpath	- Remove directory from search path.
rehash	- Refresh function and file system caches.
import	- Import packages into the current scope.
finfo	- Identify file type against standard file handlers on path.
genpath	- Generate recursive toolbox path.
savepath	- Save the current MATLAB path in the pathdef.m file.

Managing the java search path.

javaaddpath	- Add directories to the dynamic java path.
javaclasspath	- Get and set java path.
javarmpath	- Remove directory from dynamic java path.

Controlling the command window.

echo	- Echo commands in M-files.
more	- Control paged output in command window.
diary	- Save text of MATLAB session.
format	- Set output format.
beep	- Produce beep sound.
desktop	- Start and query the MATLAB Desktop.
preferences	- Bring up MATLAB user settable preferences dialog.

Operating system commands.

cd	- Change current working directory.
copyfile	- Copy file or directory.
movefile	- Move file or directory.
delete	- Delete file or graphics object.
pwd	- Show (print) current working directory.

dir	- List directory.
ls	- List directory.
fileattrib	- Set or get attributes of files and directories.
isdir	- True if argument is a directory.
mkdir	- Make new directory.
rmdir	- Remove directory.
getenv	- Get environment variable.
!	- Execute operating system command (see PUNCT).
dos	- Execute DOS command and return result.
unix	- Execute UNIX command and return result.
system	- Execute system command and return result.
perl	- Execute Perl command and return the result.
computer	- Computer type.
isunix	- True for the UNIX version of MATLAB.
ispc	- True for the PC (Windows) version of MATLAB.

Debugging.

debug	- List debugging commands.

Tools to locate dependent functions of an M-file.

depfun	- Locate dependent functions of an M-file or P-file.
depdir	- Locate dependent directories of an M-file or P-file.

Loading and calling shared libraries.

calllib	- Call a function in an external library.
libpointer	- Creates a pointer object for use with external libraries.
libstruct	- Creates a structure pointer for use with external libraries.
libisloaded	- True if the specified shared library is loaded.
loadlibrary	- Load a shared library into MATLAB.
libfunctions	- Return information on functions in an external library.
libfunctionsview	- View the functions in an external library.
unloadlibrary	- Unload a shared library loaded with LOADLIBRARY.
java	- Using Java from within MATLAB.
usejava	- True if the specified Java feature is supported in MATLAB.

Controlling multithreading setting.

maxNumCompThreads - Controls the maximum number of computational threads.

See also lang, datatypes, iofun, graphics, ops, strfun, timefun,

matfun, demos, graphics, datafun, uitools, doc, punct, arith.

Para pedir mas detalles sobre las funciones que pertenecen a una l ibrería dada, ingresamos help seguido del nombre de la librería. Por ejemplo:

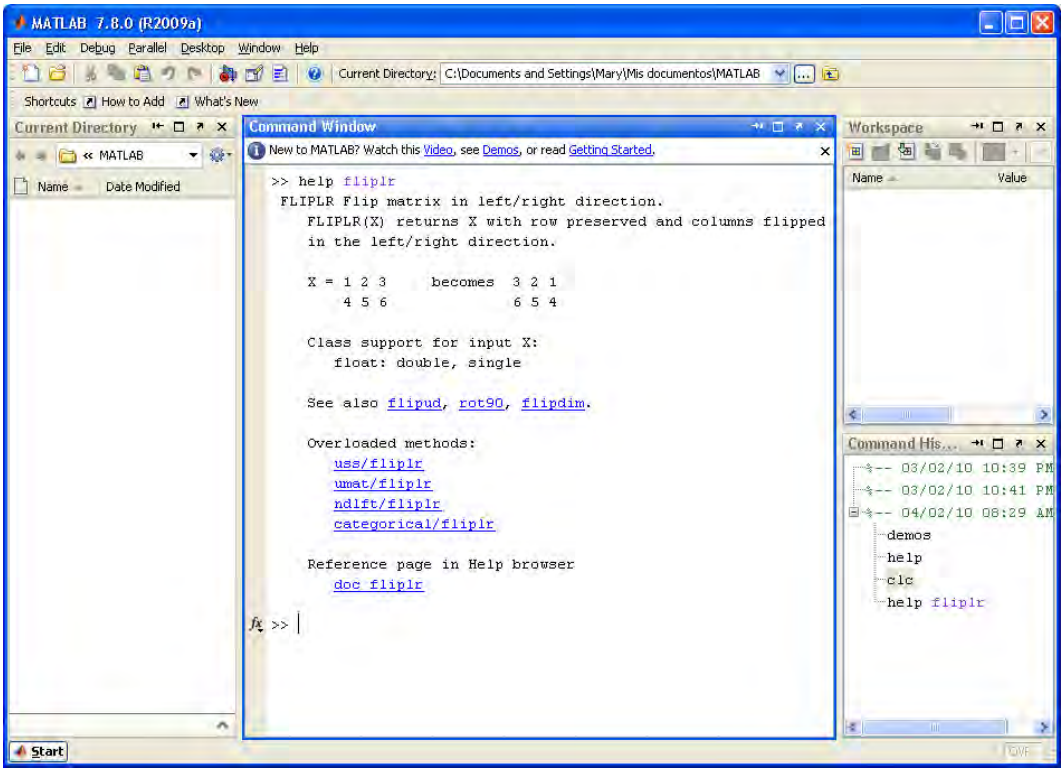
>>help stats

La librería stats agrupa diferentes rutinas útiles en probabilidad y estadística.

Al final de l a ayuda nos remite a al gunos temas relacionados para que podam os continuar la búsqueda, si es que no terminamos de encontrar lo que buscábamos.

Si quisiéramos ver con mas detalle de algún algún ítem de la lista, basta con escribir help <ítem>.

>>help fliplr

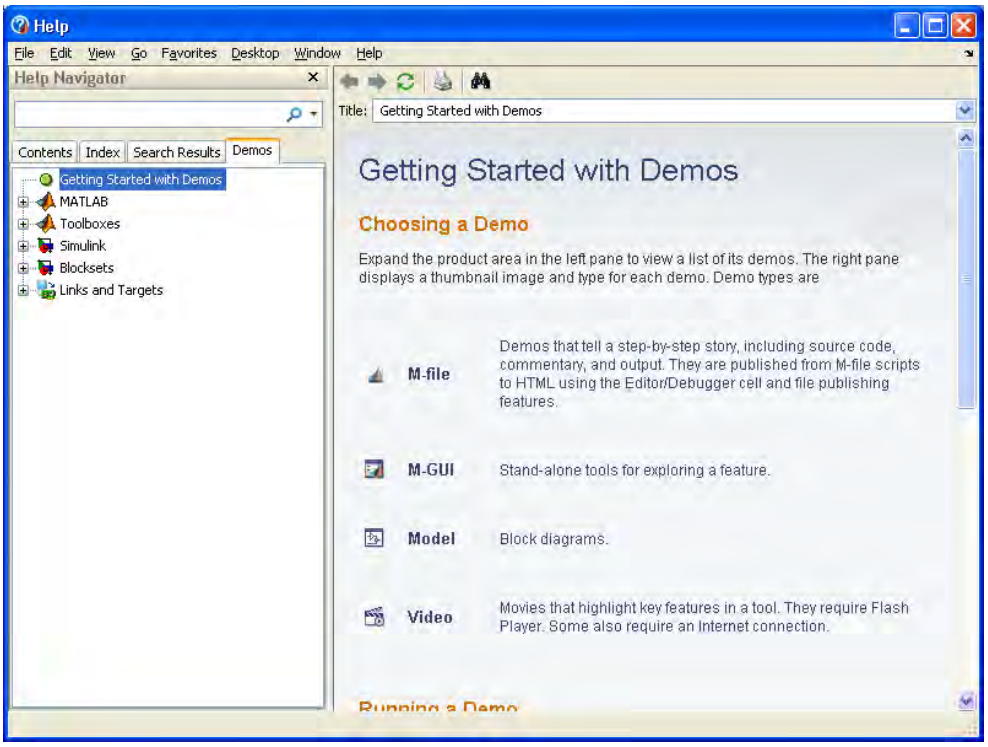


Ante cualquier duda sobre el help escribe:

>>help help

Hay otro modo de ayuda, un poco mas cómodo, que se puede acceder desde el menú desplegable Help. El contenido es el mismo que el de la línea de comandos, solo que disponemos de un pequeño navegador.

```
>>demos
```



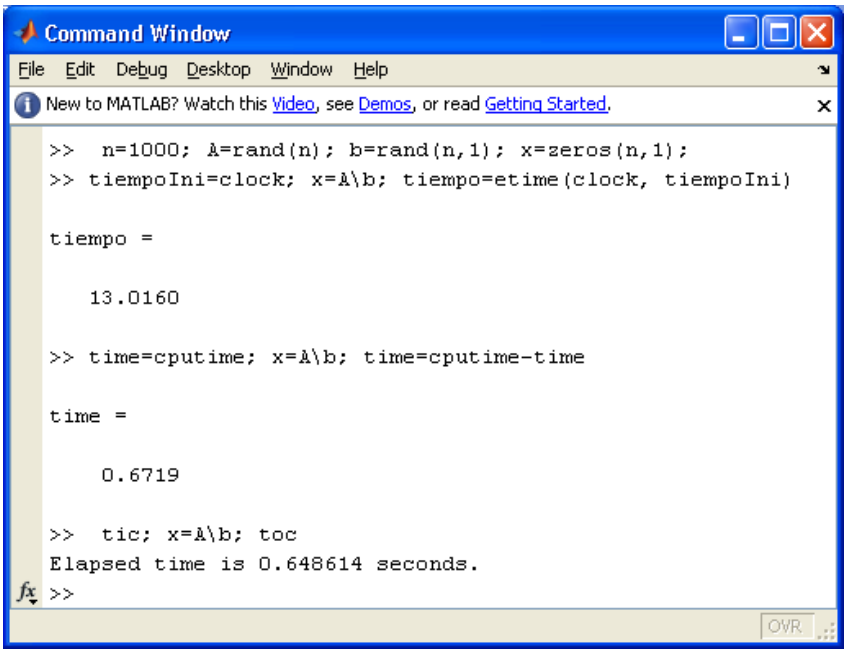
Medida de tiempos y de esfuerzo de cálculo

MATLAB dispone de funciones que permiten calcular el tiempo empleado en l as operaciones matemáticas realizadas. Algunas de estas funciones son las siguientes:

cputime	devuelve el tiempo de C PU (con precisión de centésimas de segundo) desdeque el programa arrancó. Llamando antes y después de realizar una oper ación y restando los valores devueltos, se puede saber el tiempo de CPU empleado en esa operación. Este tiempo sigue corriendo aunque MATLAB esté inactivo.
etime(t2, t1)	tiempo transcurrido entre los vectores **t1** y **t2** (¡atención al orden!), obtenidos como respuesta al comando ***clock***.
tic *ops* toc	imprime el tiempo en segundos requerido por ***ops***. El comando ***tic*** pone el reloj a cero y ***toc*** obtiene el tiempo transcurrido.

A modo de ejemplo, el siguiente código mide de varias formas el tiempo necesario para resolver un sistema de 1000 ecuaciones con 1000 incógnitas. Téngase en cuenta que los tiempos pequeños (del orden de las décimas o centésimas de segundo), no se pueden medir con gran precisión.

```
>> n=1000; A=rand(n); b=rand(n,1); x=zeros(n,1);

>> tiempoIni=clock; x=A\b; tiempo=etime(clock, tiempoIni)

>> time=cputime; x=A\b; time=cputime-time

>> tic; x=A\b; toc
```



donde se han puesto varias sentencias en la misma línea para que se ejecuten todas sin tiempos muertos al pulsar intro. Esto es especialmente importante en la línea de comandos en la que se quiere medir los tiempos. Todas las sentencias de cálculos matriciales van seguidas de punto y coma (;) con objeto de evitar la impresión de resultados. Conviene ejecutar dos o tres veces cada sentencia para obtener tiempos óptimos, ya que la primera vez que se ejecutan se emplea un cierto tiempo en cargar las funciones a memoria.

Nota: *Un punto y coma al final de una sentencia hace que no se vea el resultado de la operación. Esto es muy importante, porque Matlab tiene la costumbre de ir mostrando todos los resultados obtenidos. El punto y coma nos permite filtrar los resultados parciales y en todo caso solo mostrar aquellos que sean de nuestro interés.*

Operaciones con números reales

Los cálculos que no se asignan a una variable en concreto se asignan a la variable de respuesta por defecto que es **ans** (del inglés, answer):

```
>>2+3
ans =
        5
```

Sin embargo, si el cálculo se asigna a una variable, el resultado queda guardado en ella:

```
>>x=2+3
x =
        5
```

Para conocer el valor de una variable, basta teclear su nombre:

```
>>x
x =
        5
```

Si se añade un punto y coma (;) al final de la instrucción, la máquina no muestra la respuesta...

```
>>y=5*4;
```

... pero no por ello deja de realizarse el cálculo.

```
>>y
y =
        20
```

Operaciones aritméticas

Las operaciones se evalúan por orden de prioridad: primero las potencias, después las multiplicaciones y divisiones y, finalmente, las sumas y restas. Las operaciones de igual prioridad se evalúan de izquierda a derecha.

Prioridad	Operador	Significado	Ejemplo	Resultado
1	^	Exponente	78^2	
2	/	División derecha	29/23	
2	\	División izquierda	29\23	
3	*	Multiplicación	17*28	
4	+	Adición	15.3+19.5	
4	-	Sustracción	15.3-19.5	

```
>>2/4*3
ans =
        1.5000
>>2/(4*3)
ans =
        0.1667
```

Se pueden utilizar las funciones matemáticas habituales. Así, por ejemplo, la función coseno,

```
>>cos(pi)    % pi es una variable con valor predeterminado 3.14159...
ans =
        -1
```

o la función exponencial

```
>>exp(1)    % Función exponencial evaluada en 1, es decir, el número e
ans =
        2.7183
```

Además de la variable pi , MATLAB tiene otras variables con valor predeterminado; éste se pierde si se les asigna otro valor distinto. Por ejemplo:

```
>>eps      % Épsilon de la máquina. Obsérvese que MATLAB trabaja en doble precisión
ans =
        2.2204e-016
pero...
>>eps=7
eps =
        7
```

Otro ejemplo de función matemática: la raíz cuadrada; como puede verse, trabajar con complejos no da ningún tipo de problema. La unidad imaginaria se representa en MATLAB como i o j, variables con dicho valor como predeterminado:

```
>>sqrt(-4)
ans =
        0+ 2.0000i
```

El usuario puede controlar el número de decimales con que aparece en pantalla el valor de las variables, sin olvidar que ello no está relacionado con la precisión con la que se hacen los cálculos, sino con el aspecto con que éstos se muestran:

```
>>1/3
ans =
```

```
0.3333
```

Para conocer las variables que se han usado hasta el momento:

```
>>who

    Your variables are:
    ans    eps    x      y
```

o, si se quiere más información (obsérvese que todas las variables son arrays):

```
>>whos
     Name  Size  Bytes      Class
      ans   1x1    8     double array
      eps   1x1    8     double array
        x   1x1    8     double array
        y   1x1    8     double array
     Grand total is 4 elements using 32 bytes
```

Para deshacerse de una variable

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

Constantes matemáticas

Variable	Valor
eps	Número más pequeño tal que, cuando se le suma 1, crea un número en coma flotante en el computador mayor que 1.
pi	Relación entre la circunferencia del círculo a su diámetro
intmax	Mayor valor de tipo entero especificado.
intmin	Menor valor de tipo entero especificado.
inf	Infinito.
NaN	Magnitud no numérica.
i y j	i=j=√-1, unidad imaginaria.
realmin	El número real positivo más pequeño que es utilizable.
relamas	El número real positivo más grande que es utilizable.

Referencia de funciones

Escritorios y el Desarrollo para el Medio Ambiente	De inicio, ventana de comandos, ayuda, edición y depuración, tuning, otras funciones generales
Importación y exportación de datos	General y de bajo nivel / S de archivos, además de formatos de archivo específicos, como de audio, hoja de cálculo, HDF, imágenes
Matemáticas	Matrices y matrices, álgebra lineal, otras áreas de las matemáticas
Análisis de datos	Datos básicos de las operaciones, estadística descriptiva, covarianza y correlación, filtrado y convolución, derivadas numéricas e integrales, transformadas de Fourier, análisis de series temporales
De Programación y tipos de datos	La evaluación de funciones de expresión, el control del programa, la función de asas, programación orientada a objetos, el manejo de errores, los operadores, tipos de datos, fechas y horas, los temporizadores
Programación orientada a objetos	Funciones para trabajar con clases y objetos
Gráficos	Parcelas de la Línea, anotar los gráficos, gráficos especializados, las imágenes, la impresión, Handle Graphics
La visualización en 3-D	De superficie y las parcelas de malla, control de vista, la iluminación y la transparencia, la visualización de volúmenes
GUI para el Desarrollo	GUÍA, la programación de interfaces gráficas de usuario
Interfaces externos	Interfaces con librariess compartida, Java,. NET, COM y ActiveX, servicios Web y los dispositivos de puerto serie, y C y Fortran rutinas

Operaciones con Matrices y Vectores

Definición de variables

No es necesario definir variables.

Las variables mejoran la legibilidad del procedimiento y facilitan las correcciones y modificaciones.

En matlab las variables deben seguir las siguientes reglas:

- No pueden comenzar con un número.
- Puede tener números en la estructura del nombre de la variable.
- Las mayúsculas y minúsculas se diferencian en el nombre de las variables.
- Los nombres de las variables no pueden tener los siguientes símbolos: “+”, “-“, “*”, “/”, “.”, “,”, “;”, “^”, “~”, “&”, “|”, “\”.

Definición de matrices desde teclado

Para introducir una matriz:

- Se separan los números con espacio o comas.
- Se separan las columnas con punto y coma.
- Se agrupa toda la matriz entre corchetes.

Vectores fila

```
>> x=[1,2,3,5,7,11,13];          →    [ …. …. …. …. …. …. …. ]
>> x2=[1 2 3 5 7 11 13];         →    [ …. …. …. …. …. …. …. ]
>>a(5)=7;                        →    [ …. …. …. …. …. ]
```

Vectores columna

```
>> X=[1;2;3;5;7;11;13]
X =
    ….
    ….
    ….
    ….
    ….
    ….
    ….
```

….

Matriz

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
        ....  ....  ....  ....
        ....  ....  ....  ....
        ....  ....  ....  ....
        ....  ....  ....  ....
>> M = [1 2 3
4 5 6
7 8 9]
M=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
```

Tipos de matrices

Matriz diagonal

diag(v)	genera una matriz diagonal con el vector v como diagonal.
diag(A)	crea un vector con los elementos de la diagonal principal.

```
>> T=[17 25 29];
 >> S=diag(T)
S=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
>> diag(A)
ans =
        ....
        ....
        ....
        ....
```

Matriz simétrica diagonal constante

toeplitz(v) define una m atriz simétrica de diagonal constante con v como primera fila y primera columna.

```
>> v=[1 2 3]; toeplitz(v)
ans=
     .... .... ....
     .... .... ....
     .... .... ....
```

Matriz diagonal constante

toeplitz(w, v) define una matriz no simétrica de diagonal constante con w como primera columna y v como primera fila.

```
>> w=[1 2 3 4]; v=[1 5 7 8]; toeplitz(w,v)
ans=
     .... .... .... ....
     .... .... .... ....
     .... .... .... ....
     .... .... .... ....
```

Matriz de unos

ones(n) genera una matriz de n × n con todos los valores iguales a uno.

ones(n,m) genera una matriz de n × m con todos los valores iguales a uno.

```
>> unos=ones(3,4)
unos=
     .... .... .... ....
     .... .... .... ....
     .... .... .... ....
```

Matriz nula

zeros(n) genera una matriz de n × n con todos los valores iguales a cero.

Zeros(m,n) crea una matriz mxn, todos ceros.

```
>> ceros=zeros(2,5)
ceros=
     .... .... .... .... ....
     .... .... .... .... ....
```

Matriz identidad

eye(n) genera una matriz identidad de n × n.

eye(m,n) crea la matriz identidad de orden mxn con unos en la diagonal y ceros en los demás elementos.

```
>> ident=eyes(3)
ident=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
```

Matriz aleatoria uniforme

rand(n) genera una matriz de n × n con elementos de valor aleatorio entre 0 y 1 (distribución uniforme).

```
>> aleatorio=rand(2,3);
aleatorio=
.............  .............  .............
.............  .............  .............
```

Matriz aleatoria normal

randn(n) genera una matriz de n × n cu yos elementos siguen una distribución normal (media 0 y varianza 1).

```
>> normal=randn(2);
normal=
          .............  .............
          .............  .............
>> n1=randn(3,2)
n1=
          .............  .............
          .............  .............
          .............  .............
```

Nota

ones(m, n), zeros(m, n), rand(m, n) generan matrices de m × n.

ones(size(A)), zeros(size(A)), eye(size(A)) generan matrices de la misma forma que A.

Muchos problemas lineales implican el manejo de matrices de gran tamaño, en las que se da la circunstancia que la mayor parte de los elementos son nulos. Esto implica,

eventualmente, el almacenamiento de numerosos ceros y operaciones inútiles. Para resolver este problema, MATLAB dispone de una función (sparse) que permite prescindir de las cantidades nulas, almacenando la posición (fila y columna) de los elementos no nulos. Un ejemplo elemental se construye con la matriz unidad:

```
>> clear
>> I=eye(1000);
>> whos
     Name    Size            Bytes     Class
     I       1000x1000       8000000   double array
     Grand total is 1000000 elements using 8000000 bytes
```

Observamos que el almacenamiento de I consume aproximadamente 8 Mb.

Determinamos una expresión sparse de I:

```
>> I=eye(1000);
>> S=sparse(I);
>> whos
     Name    Size            Bytes     Class
     I       1000x1000       8000000   double array
     S       1000x1000       16004     sparse array
     Grand total is 1001000 elements using 8016004 bytes
```

Matrices especiales

Matriz Transpuesta

A’ El caracter ' (apóstrofe) denota la transpuesta de la matriz. Transponer significa intercambiar filas por columnas. Si tenemos la matriz A y llamamos B = A', B es la transpuesta de la matriz A.

```
>> A=[5 9 11; 6 8 10]; C=A'
C=
     ……   ……
     ……   ……
     ……   ……
```

Matriz Simétrica

Es una matriz cuadrada que cumple la condición: $A=A^T$.

```
>> M=[2 3 17; 3 -6 1; 17 1 7], Mt=M'
M=
```

....
....
....

Mt=

....
....
....

Matriz Antisimétrica

Es una matriz cuadrada que cumple la condición: $A=-A^T$, siendo ceros los elementos de la diagonal principal y sus demás elementos son de la forma $a_{ij}=a_{ji}$.

```
>> B=[0 1 -4; -1 0 -3; 4 3 0], Ba=-B'
```

B=

....
....
....

Ba=

....
....
....

Matriz de indices no nulos

Devuelve los índices de los elementos de la matriz A, que no son ceros, en forma vertical.

```
>> B=[0 1 -4; -1 0 -3; 4 3 0]; ind=find(B)
```

ind=

2.00
3.00
4.00
6.00
7.00
8.00

```
>> ind2=find(B>0)
```

Ind2 =

3.00
4.00
6.00

```
>> [f,c,v]=find(B>0); fila=f',columna=c',verdad=v'
fila =
        3.00        1.00        3.00
columna =
        1.00        2.00        2.00
verdad =
    1    1    1
```

Matriz Idempotente

Es una matriz cuadrada que cumple la condición: $A=A^2$.

```
>> V=[-1 2 4; 1 -2 -4; -1 2 4], Vi=V^2
V=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
Vi=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
```

Matriz Periódica

Es una matriz cuadrada que cumple la condición: $A=A^{k+1}$ entonces A es periódica y $k \in Z^+$. El periodo es igual a k.

```
>> P=[1/3 2; 1/9 2/3], Pp=P^2
P=
        ..............  ..............
        ..............  ..............
Pp=
        ..............  ..............
        ..............  ..............
```

Matriz Nilpotente

Es una matriz cuadrada que cumple la condición: $A^p=0$, donde $p \in Z$. A es nilpotente para p.

```
>> N=[1 1 3; 5 2 6; -2 -1 -3], Ni=N^3
N=
```

....
....
....

Ni=

....
....
....

Matriz Hilbert

hilb(N) es una matriz de N por N con elementos 1/(i+j-1).

```
>> format rat ; hilb(3)
ans =
    1      1/2     1/3
   1/2     1/3     1/4
   1/3     1/4     1/5
```

Matriz Hermitiana

Es una matriz cuadrada y compleja, que es igual a la transpuesta de su conjugada. Los elementos de su diagonal principal son números reales..

```
>> H=[1 3+i i; 3-i 3 1-i; -i 1+i 2], Ht=conj(H), He=transpose(Ht)
```

H=

....
....
....

Ht=

....
....
....

He=

....
....
....

Matriz Inversa

inv(A) A^{-1} si A es cuadrada e invertible, se cumple $A^{-1}*A=A*A^{-1}=I$.

```
>>  A=[1 1 1; 1 2 3; 1 3 4]; format rat; S1=inv(A)
S1=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
```

Matriz Seudoinversa

pinv(A) pseudoinversa de A , si X = pinv(A) produce una matriz X condimensiones de A', se cumple A*X*A = A, X*A*X = X y A*X y X*A son matrices hermitianas.

```
>>  K=magic(4); K1=det(K), K2=pinv(K)
K1=
        …
K2=
        .............  .............  .............  .............
        .............  .............  .............  .............
        .............  .............  .............  .............
        .............  .............  .............  .............
```

Matriz Ortogonal

Una matriz cuadrada es ortogonal si se cumple $A^{-1}= A^{T}$, es equivalente a $A*A^{T} =I$.

```
>> a=sqrt(2); M=[1/a 0 -1/a; 1/a 0 1/a; 0 1 0], L=M*M'
M=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
L=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
```

Operaciones de Matrices

Sumando y Restando Matrices

Las operaciones suma (+) y resta (-) son definidas para las matrices siempre y cuando éstas tengan la misma dimensión. Es decir, si A y B son matrices 3 x 3, entonces A + B se puede calcular.

Las operaciones suma y resta también están definidas si uno de los operadores es un escalar, es decir, una matriz 1 x 1.

```
>> A= [4 2 0 1; -2 3 6 5; 2 1 8 1];
>> B= [3 17 8 5; -7 12 15 10; 23 19 0 -25];
>> A+B
ans=
        .... .... .... ....
        .... .... .... ....
        .... .... .... ....
        .... .... .... ....
>> f=[2 4 6]; c=[1; 2; 3];
>>S1= f+c'
S1=
        .... .... .... ....
        .... .... .... ....
        .... .... .... ....
        .... .... .... ....
>>S2=f+25
S2=
        .... .... .... ....
        .... .... .... ....
        .... .... .... ....
        .... .... .... ....
>>S3=c-17
S3=
        .... .... .... ....
        .... .... .... ....
        .... .... .... ....
        .... .... .... ....
```

Multiplicando Matrices

La operación de multiplicación de matrices está definida siempre que el número de columnas de la primera matriz sea igual al número de filas de la segunda matriz.

A * B	da la matriz resultante del producto AB
	(si dicha operación es posible).

A. * B da el producto elemento por elemento

(si size(A) = size(B), es decir, si tienen el mismo tamaño)

```
>> A*B
ans=
      .... .... .... ....
      .... .... .... ....
      .... .... .... ....
      .... .... .... ....
```

Producto de una matriz por un vector

El producto de una matriz y un vector es un caso especial del producto matriz-matriz y naturalmente, un escalar como pi, puede multiplicar, ó ser multiplicado por, cualquier matriz.

```
>> 7*A
ans=
      .... .... .... ....
      .... .... .... ....
      .... .... .... ....
      .... .... .... ....
>> X*B
ans=
      ....
      ....
      ....
      ....
```

Dividiendo Matrices

En división de matrices, si A es una matriz cuadrada no-singular, entonces A\B y B/A corresponden a la multiplicación izquierda y derecha de B por el inverso de A, esto es, inv(A) * B y B * inv(A) respectivamente. El resultado es obtenido directamente sin la computación del inverso.

```
>> A=[0 1;1 0]; B=[1 2;3 4]; X=A\B
X =
      3     4
      1     2
```

```
>> Y=B/A
Y =
        2        1
        4        3
```

- X = A\B es una solución a A * X = B,

  es igual a inv(A) * B si existe inv(A): la barra inversa es la división por la izquierda.
- X = B/A es una solución a X * A = B

A\B es definido cuando B tiene la misma cantidad de filas que A. Si A es cuadrada, el método usado es la Eliminación Gaussiana. El resultado es una matriz X con las mismas dimensiones que B.

Si A no es cuadrada, se factoriza utilizando la ortogonalización de Householder con pivoteo de columnas.

Los factores son usados para resolver sistemas de ecuaciones sub-determinados y sobre -determinados. El resultado es una matriz X m-por-n donde m es el número de columnas de A y n es el número de columnas de B. Cada columna de X tiene, al menos, k componentes diferentes de cero, donde k es el rango efectivo de A.

B/A esta definido en términos de A\B por B/A = (A' \B') '.

Ejemplo: resolver el sistema:

$$2a + 3b + c = 6$$
$$4a + b + 2c = 7$$
$$6a + b + 7c = 4$$

```
>>A=[2 3 1; 4 1 2; 6 1 7];
>>B=[6; 7; 4]
>>X=A\B
x=
        .....
        .....
        .....
```

Exponentes con Matrices

La expresión A^n eleva A a la n-ésima potencia y está definido si A es una matriz cuadrada y n un escalar.

```
>>p1=A^2
p1 =
        .... .... ....
        .... .... ....
        .... .... ....
```

Funciones Matriciales Trascendentales y Elementales

MATLAB considera expresiones como exp(A) y sqrt(A) como operaciones de arreglos, definidas en los elementos individuales de A. También puede ca lcular funciones trascendentales de matrices, como la matriz exponencial y la matriz logarítmica. Estas operaciones especiales están definidas solamente para matrices cuadradas.

```
>>sqrt(A)
ans =
        .... .... ....
        .... .... ....
        .... .... ....
>>exp(A)
ans =
        .... .... ....
        .... .... ....
        .... .... ....
```

Funciones matemáticas elementales

Trigonométricas

sin(x)	Función seno
cos(x)	Función coseno
tan(x)	Función tangente
asin(x)	Inversa del seno
acos(x)	Inversa del coseno
atan(x)	Inversa de la tangente
atan2(x,y)	Inversa de la tangente de los cuatro cuadrantes
sinh(x)	Función seno hiperbólico

cosh(x)	Función coseno hiperbólico
tanh(x)	Función tangente hiperbólica
asin(x)	Inversa del seno hiperbólico
acosh(x)	Inversa del coseno hiperbólico
atanh(x)	Inversa de la tangente hiperbólica

También hay funciones para ángulos en sexagesimales.

Exponencial

exp(x)	Exponencial e^x.
log(x)	Logaritmo natural.
log10(x)	Logaritmo decimal.
log2(x)	Logaritmo en base 2.
sqrt(x)	Raíz cuadrada.

Complejo

abs(x)	Valor absoluto o magnitud de un número complejo.
real(x)	Parte real de un número complejo.
imag(x)	Parte imaginaria de un número complejo.
angle(x)	Ángulo de un número complejo.
conj(x)	Conjugado complejo

Redondeo y resto

ceil(x)	Redondeo hacia más infinito.
fix(x)	Redondeo hacia cero.
floor(x)	Redondeo hacia menos infinito.
round(x)	Redondea hacia el entero más próximo.
rem(x,y)	Resto después de la división.
mod(x,y)	Módulo después de la división.

Operaciones elemento a elemento

Matlab define algunas operaciones que serán realizadas elemento a elemento.

Ya vimos anteriormente que la operación A*B realizaba el producto matricial.

Pero ¿que pasa si queremos que cada elemento de A quede multiplicado por cada elemento de B (suponiendo que tienen las mismas dimensiones)?

Existe otro operador para tales fines:

Definimos las matrices a y b:

```
>>A = [1  2  3 ; 4  5  6; 7 8 9]; B= [9  8  7 ; 6  5  4; 3 2 1];
>>C1=A.*B
C1=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
```

Como se ve, si anteponemos un punto al operador, la operación se realiza elemento a elemento. La división (/) y la potencia (^) también permiten esta utilización.

```
>>C2=A./B
C2=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
>>C3=A.\B
C3=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
>>C4=B.^A
C4=
        ....  ....  ....
        ....  ....  ....
        ....  ....  ....
```

*Otras operaciones pueden ser consultadas en la librería **ops**.*

Generador de vectores

- Para generar un v ector cuyos elementos sean números crecientes o decrecientes en un intervalo regular existe el operador **:** (dos puntos).

  - El uso más sencillo de este operador sería:

    <Mínimo>**:**<Máximo>

  >>d = 1:4

  d =

  1 2 3 4

  - Especificar un intervalo determinado, escribimos:

    <Mínimo>**:**<Intervalo>**:**<Máximo>

  >>e = 1:0.5:4

  d =

  1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000

  - Si el rango es decreciente, sólo escribimos el intervalo negativo.

  >>f = 4:-0.5:1

  f =

  4.0000 3.5000 3.0000 2.5000 2.0000 1.5000 1.0000

- Genera un vector con n valores entre x1 y x2 igualmente espaciados.

  linspace(x1,x2,n)

  >>g = linspace(0,20,7)

  g=

  0 10/3 20/3 10 40/3 50/3 20

  >>h = linspace(0,pi,5)

  h=

  0 0.7854 1.5708 2.3562 3.1416

Comandos relacionados con tamaño de datos

length() se aplica solo a vectores. Devuelve el largo del vector (es igual para filas y columnas).

```
>>length(g)
ans=
      .....
```

size() se aplica tanto a vectores como matrices. Devuelve un vector de dos elementos: cantidad de filas y cantidad de columnas.

```
>>size(A)
ans=
      ..... .....
```

Cambiar elementos en una matriz A dada

A(3, 2) = 7	coloca un 7 en el elemento (3, 2).
A(3,:) = v	sustituye los valores de la tercera fila por los de v= [2; 4; 5].
A(:, 2) = w	sustituye los valores de la segunda columna por los de w= 2:4.
	*El símbolo de los dos puntos **:** significa todo (todas las columnas o todas las filas).*
A([2 3],:) = A([3 2],:)	intercambia las filas 2 y 3 de A.

Crear submatrices de una matriz A de m × n

- Matlab utiliza los paréntesis para acceder a elementos de la matriz.
- Los subíndices empiezan en 1, por lo tanto el primer elemento es a(1,1)
- *Nota: Matlab **no admite el cero como índice** de vectores ni matrices !!!!*

A(i, j)	muestra el elemento (i, j) de la matriz A (escalar = matriz de 1 × 1).
A(i, :)	muestra la fila i-ésima de A (como vector de fila).
A(:, j)	muestra la columna j-ésima de A (como vector de columna).
A(2: 4,3: 7)	muestra las filas de la 2 a la 4 y las columnas de la 3 a la 7 (en forma de matriz de 3 × 5).
A([2 4],:)	muestra las filas 2 y 4 y todas las columnas (en forma de matriz de 2 × n).
A(:)	muestra una sola columna larga formada a partir de las columnas de A (matriz de mn × 1).
triu(A)	coloca ceros en todos los elementos por debajo de la diagonal (triangular superior).
tril(A)	coloca ceros en todos lo elementos por encima de la diagonal (triangular inferior).

>> a=rand(4,5);

>> a(3,5)=56.8

0.1737	0.3421	0.6391	0.1632	0.2313
0.7858	0.7742	0.0934	0.2763	0.8453
0.3656	0.1478	0.9288	0.1310	0.7264
0.7769	0.1482	0.4851	0.0232	0.6947

0.1737	0.3421	0.6391	0.1632	0.2313
0.7858	0.7742	0.0934	0.2763	0.8453
0.3656	0.1478	0.9288	0.1310	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

- Se pueden utilizar vectores para definir índices

  >> a(2:3,1:4)=zeros(2,4);

  o bien: a(2:3,1:4)=0;

0.1737	0.3421	0.6391	0.1632	0.2313
0.7858	0.7742	0.0934	0.2763	0.8453
0.3656	0.1478	0.9288	0.1310	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

0.1737	0.3421	0.6391	0.1632	0.2313
0	0	0	0	0.8453
0	0	0	0	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

- Se pueden utilizar vectores para definir índices

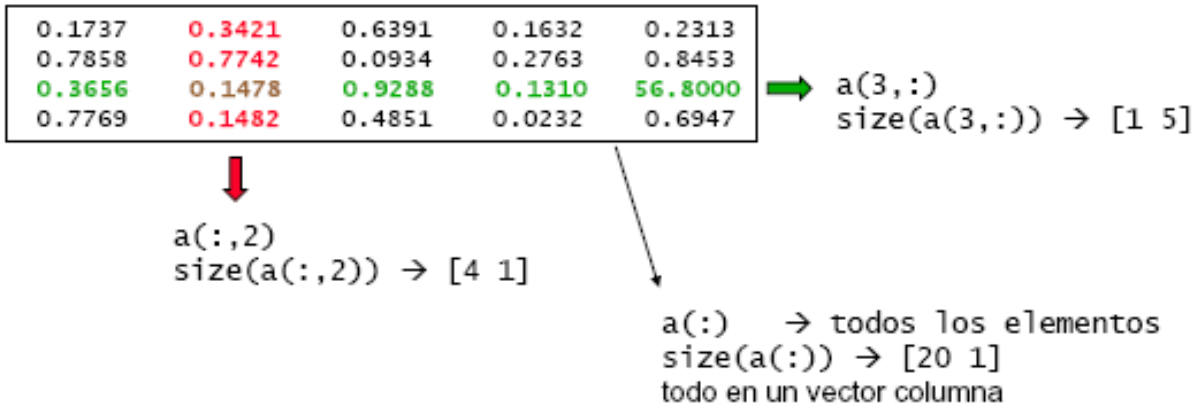
  >> a([2,3],[2,4])=ones(2,2);

  o bien: a([2,3],[2,4])=1;

0.1737	0.3421	0.6391	0.1632	0.2313
0	0	0	0	0.8453
0	0	0	0	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

0.1737	0.3421	0.6391	0.1632	0.2313
0	1.0000	0	1.0000	0.8453
0	1.0000	0	1.0000	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

- El operador ':' se utiliza para indicar "todos los elementos"



Triangular inferior

```
>>tril(M)
ans =
     .... .... ....
     .... .... ....
     .... .... ....
>>tril(M,-1)
ans =
     .... .... ....
     .... .... ....
     .... .... ....
```

Triangular superior

```
>>triu(M)
ans =
     .... .... ....
     .... .... ....
     .... .... ....
>>triu(M,1)
ans =
     .... .... ....
     .... .... ....
     .... .... ....
```

Concatenación de vectores y matrices

Supongamos que queremos formar una matriz con diferentes vectores y/o matrices de dimensiones compatibles, o queremos unir dos vectores para formar uno mas largo.

```
>>v = [1 2 3]; w=[4 5 6];
```

Podemos concatenarlos de varias maneras...

- Si los unimos uno a continuación del otro:

```
>>y = [v w]
y =
     1    2    3    4    5    6
```

Notar que al separar v y w por un espacio estamos diciendo que ambos pertenecen a la misma fila, por lo que es entendible el resultado obtenido.

- Si por el contrario, interponemos un punto y coma entre ambos:

```
>>y2 = [v ; w]
Y2 =
     1    2    3
     4    5    6
```

Matlab coloca cada vector en una fila diferente.

Función cat

cat(d,M,N) concatena matrices con una dimensión especificada.

```
>>M=[2 3; 1 9]; N=[8 4; 5 7]; MN=cat(1,M,N)
MN =
    2    3
    1    9
    8    4
    5    7
>>NM= cat(2,B,C,B)
NM =
   35    1   14   16   35    1
    3   32   18   11    3   32
>> A=magic(6)
A =
```

```
    35     1     6    26    19    24
     3    32     7    21    23    25
    31     9     2    22    27    20
     8    28    33    17    10    15
    30     5    34    12    14    16
     4    36    29    13    18    11
>> B=A(1:2,1:2); C=A(5:6,5:6); cat(1,B,C)
ans =
    35     1
     3    32
    14    16
    18    11
>> cat(2,B,C)
ans =
    35     1    14    16
     3    32    18    11
```

Función horzcat

concatena matrices horizontalmente.

```
>> A = magic(5); A(4:5,:) = []
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
>> B = magic(3)*17
B =
   136    17   102
    51    85   119
    68   153    34
>> C = horzcat(A, B)
C =
    17    24     1     8    15   136    17   102
    23     5     7    14    16    51    85   119
```

```
     4     6    13    20    22    68   153    34
```

Función vertcat

concatena matrices verticalmente.

```
>> A = magic(5); A(:, 4:5) = []
A =
    17    24     1
    23     5     7
     4     6    13
    10    12    19
    11    18    25
>> B = magic(3)*7
B =
    56     7    42
    21    35    49
    28    63    14

>> C = vertcat(A,B)
C =
    17    24     1
    23     5     7
     4     6    13
    10    12    19
    11    18    25
    56     7    42
    21    35    49
    28    63    14
```

Función repmat

crea una nueva matriz de copias, m veces verticalmente y n veces horizontalmente.

repmat(A,m,n)

```
>> K = repmat([1 2; 3 4],2,3)
```

```
K =
    1    2    1    2    1    2
    3    4    3    4    3    4
    1    2    1    2    1    2
    3    4    3    4    3    4
```

Función blkdiag

crea una nueva matriz de bloque diagonal a partir de matrices existentes, los demás elementos de la matriz son ceros.

```
>> A=7;B=[9 8; 2 5];C=pascal(3);D=[1 7 12]; blkdiag(A,B,C,D)

ans =
    7    0    0    0    0    0    0    0    0
    0    9    8    0    0    0    0    0    0
    0    2    5    0    0    0    0    0    0
    0    0    0    1    1    1    0    0    0
    0    0    0    1    2    3    0    0    0
    0    0    0    1    3    6    0    0    0
    0    0    0    0    0    0    1    7   12
```

Números y matrices asociados a *A*

Determinante

det(A) es el determinante (si A es una matriz cuadrada).

```
>> A=[3 2 1; 4 3 0; 7 9 15]; det(A)

ans =
   30

>> det(magic(4))

ans =
    0
```

Rango

rank(A) es el rango (número de pivotes = dimensión del espacio de filas y del espacio de columnas).

```
>> rank(A)
```

```
ans =
    3
>> rank(magic(4))
ans =
    3
```

Orden

size(A) es el par de números [m n].

```
>> size(A)
ans =
    3    3
```

Traza

trace(A) es la traza = suma de los elementos de la diagonal = suma de autovalores.

```
>> trace(A)
ans =
   21
>> trace(magic(4))
ans =
   34
```

Producto Escalar

El producto interior (producto escalar ó producto punto) se consigue de la siguiente manera: **x' * y** asumiendo que x y y son vectores columnas. Note que y' * x produce el mismo resultado.

```
>> X=[1, 2, 3, 5];
>> Y=[4, 7, -8, 3];
>> X'*Y
ans=
        .....
```

dot(u,v) calcula el producto escalar de dos vectores **u** y **v**.

```
>> u=[-7 17 5]; v=[-2 3 4]; dot(u,v)
```

```
ans =
   85
```

Producto Vectorial

crossl(u,v) calcula el producto vectorial de dos vectores **u** y **v** en R^3.

```
>> u=[1 7 5]; v=[2 3 8]; cross(u,v), cross(v,u)
ans =
   41    2  -11
ans =
  -41   -2   11
```

Norma

norm(A,p) calcula la norma de A, donde p puede ser 1, 2 o inf.

Si p=1, calcula la suma de l os valores absolutos de todos los elementos de A por columna su equivalentes es `max(sum(abs(A)))`.

Si p=2, norma Euclidiana si es un vector, opción por defecto, su equivalente es `max(svd(A))`.

Si p=inf, calcula el máximo valor absoluto de sus elementos, su equivalente es `max(sum(abs(A')))`.

```
>> v=[3 4 5]; norm(v,1)
ans =
   12
>> norm(v,2), norm(v)
ans =
   7.0711
ans =
   7.0711
>> norm(v,inf)
ans =
    5
>> norm(v,-inf)
ans =
    3
```

```
>> N=[8 5; 1 7]; norm(N)
ans =
  10.8035
```

Base ortogonal para el espacio nulo

null(A) es una matriz cuyas columnas n - r forman una base ortogonal para el espacio nulo de A.

```
>> A = [1   2   3
     1   2   3
     1   2   3];
Z = null(A);
A*Z
ans =
  1.0e-015 *
    0.2220   0.2220
    0.2220   0.2220
    0.2220   0.2220
>> Z'*Z
ans =
    1.0000   -0.0000
   -0.0000    1.0000
```

Base ortogonal para el espacio de columnas

B=orth(A) es una matriz cuyas columnas r forman una base ortogonal para el espacio de columnas de A, se cumple B'*B=eye(rank(A)).

```
>> A=[3 2 1; 4 3 0; 7 9 15]; B=orth(A)
B =
   -0.1548   -0.4852   -0.8606
   -0.1622   -0.8468    0.5066
   -0.9745    0.2180    0.0524
```

Operadores relacionales

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

Operadores lógicos

&	Conjunción, devuelve el valor de 1 en cada posición, donde los elementos de ambas matrices no son ceros, 0 para los otros casos.
|	Disyunción, devuelve el valor de 1 en cada posición, donde los elementos de por lo menos de una matriz no es nula, 0 en otros casos.
~	Negación, complementa cada elemento de la matriz de entrada.
xor	Disyunción exclusiva, devuelve el valor de 1, en cada posición donde los elementos de ambas matrices son diferentes, 0 para los otros casos.

Inputs		**and**	**or**	**not**	**xor**
A	**B**	**A & B**	**A | B**	**~A**	**xor(A,B)**
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

```
>> A=3*ones(2), B=eye(2)*3
A =
    3    3
    3    3
B =
    3    0
```

```
    0     3
>> A&B
ans =
    1     0
    0     1

>> A|B
ans =
    1     1
    1     1
>> ~B
ans =
    0     1
    1     0
>> xor(A,B)
ans =
    0     1
    1     0
```

Variables lógicas

También existen variables lógicas que toman los valores 0 (falso) o 1 (verdadero)

```
>>v=[-7 3 17];
>>abs(v)>=2            % Vector lógico cuyas coordenadas valen 1 si la
                       % coordenada correspondiente de v es >= 2 y 0 si no lo es
ans =
        0 1 1
>>vector=v(abs(v)>=2) % Vector formado por la coordenadas de v que
                       % verifican la desigualdad
vector =
        2 3
>>v2=[23 3 12]
v2 =
        23 3 12
```

```
>>logica=v==v2   % Asignación de un valor lógico (el doble signo igual
                % es el igual lógico)
logica =
        0 1 0
>>logic2=v~=v2   % Distinto (~ es el operador de negación)
logic2 =
        1 0 1
```

Ejercicios

1. Crear los siguientes vectores:
   a. X=[3 $\sqrt{7}$ π e^{12}]
   b. Y=[0 0.1π 0.2π 0.3π 0.4π 0.5π 0.6π 0.7π 0.8π 0.9π π]
   c. K=$\begin{bmatrix} 7 & 6 & 5 \\ 2 & 4 & 16 \\ 9 & 3 & 1 \end{bmatrix}$
2. Crear un vector Z de cuatro números complejos.
3. Listar el tercer elemento del vector.
4. Listar los 5 primeros elementos del vector Y.
5. Listar los 5 últimos elementos del vector Y.
6. Listar los elementos de posiciones impares del vector Y.
7. Listar los elementos de posiciones 2, 4, 5 y 7 del vector Y.
8. Crear los vectores M=[5 4 3 2 1] y C=[17 9 8 25 12].
9. Fusionar los vectores M y C en un vector J.
10. Obtener la transpuesta del vector K.
11. Obtener la transpuesta del vector Z.
12. Crear las siguientes matrices:
    a. R=$\begin{bmatrix} 4 & 3 & 2 & 1 \\ 5 & 6 & 7 & 8 \end{bmatrix}$

b. $S=\begin{bmatrix} 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \end{bmatrix}$

13. Sumar las matrices R y S.

14. Multiplicar las matrices R y S.

15. Multiplicar R con la transpuesta de S.

16. Multiplicar R y S componente a componente.

17. Eleve 3 a cada elemento de R.

18. Obtener la inversa de cada elemento de R.

19. Hallar la matriz inversa de K.

20. E = eye(4); E(2, 1) = -3 crea una matriz de eliminación elemental de 4 × 4. E*A resta 3 veces la fila 1 de la fila 2 de A.

21. B = [A b] crea una matriz aumentada con b como columna adicional.

22. E = eye(3); P = E([2 1 3],:) genera una matriz de permutación.

23. Nótese que triu(A) + tril(A) - diag(diag(A)) es igual a A.

24. Sea E=[-1 2 4; 1 -2 -4; -1 2 4], expresar la matriz F con las columnas de izquierda a derecha de la matriz E.

....
....
....

25. Expresar E como la suma de una matriz simétrica y una antisimétrica

SIMÉTRICA	ANTISIMÉTRICA
....	....
....	....
....	....

26. Determinar si E es idempotente.

27. Sea A=[-3 -6 2; 2 4 -1; 2 3 0], expresar la matriz A con las filas de arriba abajo.

....
....
....

28. Determinar si la matriz A es involuta

....
....
....

29. Hallar el rango de la matriz A …….

30. Hallar el determinante de la matriz A ……

31. Hallar la inversa de la matriz A

....
....
....

32. Sea C=[1 1 3; 5 2 6; -2 -1 -3], determinar si la matriz C es nilpotente.

....
....
....

33. Hallar el determinante de la matriz C ……

34. Hallar el rango de la matriz C …….

35. Resolver el sistema:

2 a + 3 b + c = 6

4 a + b + 2 c = 7

6 a + b + 7 c = 4

Mediante la función inv.

36. Resolver el siguiente sistema de ecuaciones.

2x + 0y + 5z = 100
3x + 5y + 9z = 251
1x + 5y + 7z = 301

Polinomios

conv	Producto de polinomios.
deconv	División de polinomios.
poly	Polinomio con raíces especificado.
polyder	Polinomio derivados
polyeig	Polinomial problema de valores propios
polyfit	Ajuste de la curva polinómica
polyint	Integrar polinomio analíticamente
polyval	Evaluación Polynomial.
polyvalm	Matriz de evaluación polinomio
residue	Convertir entre la expansión de l as fracciones parciales y polinomio de coeficientes
roots	Raíces del polinomio.

```
>>p=[1 0 2 0 3]      % Polinomio x^4+2*x^2+3
p =
        1 0 2 0 3
>>q=[2 1 0]          % Polinomio 2*x^2+x
q =
        2 1 0
>>polyval(p,-1)  % Evaluación del polinomio x^4+2x^2+3 en x=-1
ans =
        6
>>pro=conv(p,q)  % Producto de los polinomios p y q
pro =
        2 1 4 2 6 3 0
>>deconv(pro,p)  % Cociente entre pro y p; obviamente el resultado es q
ans =
        2 1 0
>>roots(pro)     % Raíces del polinomio pro
ans =
            0
```

```
          0.6050+1.1688i
          0.6050-1.1688i
         -0.6050+1.1688i
         -0.6050-1.1688i
         -0.5000
>>poly([i -i 1/2 pi])    % Polinomio mónico que tiene por raíces a los
                         % números i, -i, 0.5 y pi
ans =
-3.6416 2.5708 -3.6416 1.5708
>> A=[4,2;3,3]; p=poly(A)
p =
          1 -7 6
```

El resultado son los coeficientes del polinomio característico ordenado de acuerdo a las potencias decrecientes de la variable, es decir: λ

$P(\lambda) = \lambda^2 - 7 + 6$

Otra forma de calcular el polinomio característico es usando el comando:

vpa(polynsym(p)), donde " n" indica el número de cifras decimales con que se quiere obtener los coeficientes del polinomio.

```
>> vpa(poly2sym(p))
ans =
          x^2-7.*x+6
```

Expresa el polinomio característico en la variable x.

Sean $p(x)=3x^2+6x+9$ y $q(x)=x^2+2x$)

```
>> px=[3 6 9], qx=[1 2 0]
> polyder(px,qx)
ans =
   12    36    42    18
>> conv(px,qx)
ans =
    3    12    21    18     0
>> prod=conv(px,qx)
prod =
    3    12    21    18     0
>> deconv(prod,px)
```

```
ans =
    1     2     0
>> polyder(prod)
ans =
   12    36    42    18
>>polyint(qx)
ans =
     1/3          1          0          0
```

Ejercicios

37. Definir los siguientes polinomios: p(x)=3*x^4+5*x^3+2*x^2+8*x+6 y q(x)=6*x^4+2*x^3+x^2+7*x+8 y hallar
    a. El valor del polinomio p(-1) y q(-3)
       p(-1)=
       q(-3)=
    b. r(x) es el producto de los polinomios p(x) y q(x)
       r(x)=
    c. d(x) es el cociente de los polinomios p(x) y q(x)
       d(x)=
    d. las soluciones de los polinomios p(x) y q(x)
       p(x)
       x1= x2= x3= x4=
       q(x)=
       x1= x2= x3= x4=
38. Hallar las soluciones de los polinomios
    a. $f(s)=s^4 + 3s^3 – 15s^2 – 2s + 9$
       s1= s2= s3= s4=
    b. $g(s)=s^4 + 1$
       s1= s2= s3= s4=
    c. $h(x)=x^3 + 5x^2 – 2$
       x1= x2= x3=

Álgebra Lineal

Matriz de logaritmos y exponenciales

expm	Matriz exponencial.
logm	Logaritmo Matrix.
sqrtm	Raíz matriz cuadrada.

```
>> A = [1      1      0;    0      0      2;    0      0      -1 ]
A =
    1    1    0
    0    0    2
    0    0   -1
>>  Y=expm(A)
Y =
   2.7183    1.7183    1.0862
         0    1.0000    1.2642
         0         0    0.3679
>>  logm(Y)              % A=logm(Y)
ans =
   1.0000    1.0000    0.0000
         0         0    2.0000
         0         0   -1.0000
>> X=[10 7; 15 22]
X =
   10    7
   15   22
>> Y=sqrtm(X)
Y =
   2.8347    0.9575
   2.0518    4.4761
>> Y*Y                  % X=Y*Y
```

```
ans =
  10.0000    7.0000
  15.0000   22.0000
```

Análisis de matriz

cond	Número de condición con respecto a la inversión.
condeig	Número de condición con respecto a los valores propios.
det	Determinante de la matriz.
norm	Vector y matriz de las normas.
normest	2-estimación de la norma.
null	Espacio nulo.
orth	Rango de espacio de la matriz.
rank	Rango de la matriz.
rcond	Matriz de estimación de número de condición de reciprocidad.
rref	Reducción de forma escalonada por fila.
subspace	Ángulo entre dos subespacios.
trace	Suma de los elementos de la diagonal.

```
>> A=magic(3);
>> trace(A)
ans=
        .......
>>rank(A)
ans=
        .......
>>detA)
ans=
        .......
>>norm(A)
ans=
        .......
>> B=magic(4);
```

```
A =

   8   1   6
   3   5   7
   4   9   2
```

```
B =

  16   2   3  13
   5  11  10   8
   9   7   6  12
   4  14  15   1
```

```
>> rank(B)
ans =
    3
>> rref(B)
ans =
    1    0    0    1
    0    1    0    3
    0    0    1   -3
    0    0    0    0
>> c=sqrt(2); C=[1/c 0 -1/c; 1/c 0 1/c; 0 1 0], OR=orth(C)
C =
   0.7071         0   -0.7071
   0.7071         0    0.7071
        0    1.0000         0
OR =
  -0.7071         0    0.7071
  -0.7071         0   -0.7071
        0   -1.0000         0
>> OR'*OR
ans =
   1.0000         0    0.0000
        0    1.0000         0
   0.0000         0    1.0000
```

Valores propios y valores singulares

balance	Diagonal de escala para mejorar la precisión de valores propios
cdf2rdf	Cambio de forma diagonal complejo para bloquear real forma diagonal
eig	Valores y vectores propios
eigs	Valores propios más grande y vectores propios de la matriz
gsvd	Generalizada descomposición de valor singular
hess	Hessenberg forma de matriz de

ordeig	Valores propios de las matrices de quasitriangular
ordgz	Reordenar valores propios en QZ factorización
ordschur	Reordenar valores propios en la factorización Schur
poly	Polinomio con raíces especificado
polyeig	Polynomial problema de valores propios
rsf2csf	Cambio de forma real de Schur para formar complejos de Schur
schur	Descomposición de Schur
sqrtm	Raíz matriz cuadrada
ss2tf	Cambio de estado de los parámetros de filtro de espacio para la transferencia de forma función de
svd	Descomposición del valor singular
vds	Encuentre los valores singulares y de vectores

$P(\lambda)=\det(A-\lambda I)$ donde A es una matriz cuadrada, I es la matriz identidad y λ es un parámetro. Al desarrollar $\det(A-\lambda I)$ obtenemos el polinomio característico.

```
>>A=[5 4 2; 4 5 2; 2 2 2];
>> p=poly(A)
p =
     1    -12     21    -10
```

Al resolver $\det(A-\lambda I)=0$ obtenemos los valores propios o característicos del polinomio, y

$(A-\lambda I)v=0$ obtenemos los vectores propios asociados a los valores propios.

En Matlab tenemos eig, si [V,D]=eig(A) produce una matriz diagonal D de los valores propios y una matriz V de columnas que corresponden a los vectores propios. De tal forma que se cumpla: AV=VD.

```
>> [V,D]=eig(A)
V =
  -601/1157     503/941       2/3
   112/1181   -2107/2850      2/3
    383/451    1514/3697      1/3
D =
     1     0     0
     0     1     0
     0     0    10
```

Análisis de datos

Operaciones básicas

prod	Producto de elementos de la matriz.
sum	Suma de los elementos de la matriz.
cumprod	Producto acumulado.
cumsum	Suma acumulada.
sort	Ordenar elementos de la matriz en or den ascendente o descendente.
sortrows	Ordenar filas en orden ascendente.

```
>>x=[15 2 -7 5 6 3 4 1 8 9 21 17.5];
>>sum(x)      % suma de los elementos de un vector.
ans =
          84.50
>>cumsum(x)      % devuelve el vector suma acumulativa de los elementos de un vector.
                 % producto de los elementos de un vector
.x =
  Columns 1 through 7
      15.00      17.00      10.00      15.00      21.00      24.00      28.00
  Columns 8 through 12
      29.00      37.00      46.00      67.00      84.50
>>x=[15 3 4 1 8 9];
>> cumprod(x)  % devuelve el vector producto acumulativo de los elementos de un vector.
ans =
      15.00      45.00     180.00     180.00    1440.00   12960.00
>> [y,I]=sort(x)        % ordena de menor a mayor los elementos de un vector x.
y =
       1.00       3.00       4.00       8.00       9.00      15.00
I =
    4.00    2.00    3.00    5.00    6.00    1.00    5    6    8    9    15
```

Estadística Descriptiva

Puede utilizar las funciones de MATLAB siguiente para calcular las estadísticas descriptivas para los datos.

Nota: Para la matriz de datos, estadísticas descriptivas para cada columna se calculan de forma independiente.

corrcoef	los coeficientes de correlación
cov	covarianza.
max	mayor valor.
mean	promedio o valor medio.
median	mediana.
min	menor valor.
mode	los valores más frecuentes en la matriz.
std	desviación estándar.
var	varianza.

```
>> x=[15 2 -7 5 6 3 4 1 8 9 21 17.5];
>>[xm,im]=max(x)
% máximo elemento de un vector. Devuelve el valor máximo xm y la posición que ocupa im.
xm =
          21
im =
          11
>>[xmi,mi]=min(x)
% mínimo elemento de un vector. Devuelve el valor mínimo xmi y la posición que ocupa mi.
xmi =
          -7
mi =
          3
>> mean(x)          % valor medio de los elementos de un vector.
ans =
          169/24
>> std(x)       % desviación típica.
```

```
ans =

        644/83
```

***Nota**: en realidad estas funciones se pueden aplicar también a matrices, pero en este caso se aplican por separado a cada columna de la matriz, dando como valor de retorno un vector resultante de aplicar la función a cada columna de la matriz considerada como vector. Si estas funciones se quieren aplicar a las filas de la matriz basta aplicar dichas funciones a la matriz transpuesta.*

Derivadas e integrales

cumtrapz	Acumulativa de integración numérica trapezoidal
del2	Integral laplaciano
diff	Las diferencias y aproximar los derivados
gradient	Gradiente numérico
int	Integral
polyder	Polinomio derivados
polyint	Integrar polinomio analíticamente
trapz	La integración numérica trapezoidal

```
>>f='sin(x)'   % Función sin(x) definida mediante una cadena de caracteres

f =

        sin(x)

, >>diff(f)          % calcular derivadas

ans =
        cos(x)

>>diff(f,2)    % Derivada segunda de f

ans =
-sin(x)

>>int('log(x)') % Primitiva de la función logaritmo

ans =
        x*log(x)-x
```

```
>>diff('x*log(x)-x') % Comprobación

ans =
        log(x)
```

Ejercicios

Las notas obtenidas por 10 alumnos en Física 1 y Física 2 son:

F 1	11	9	13	10	8	12	10	15	10	9
F 2	13	11	15	12	10	14	12	9	12	16

Completar el cuadro

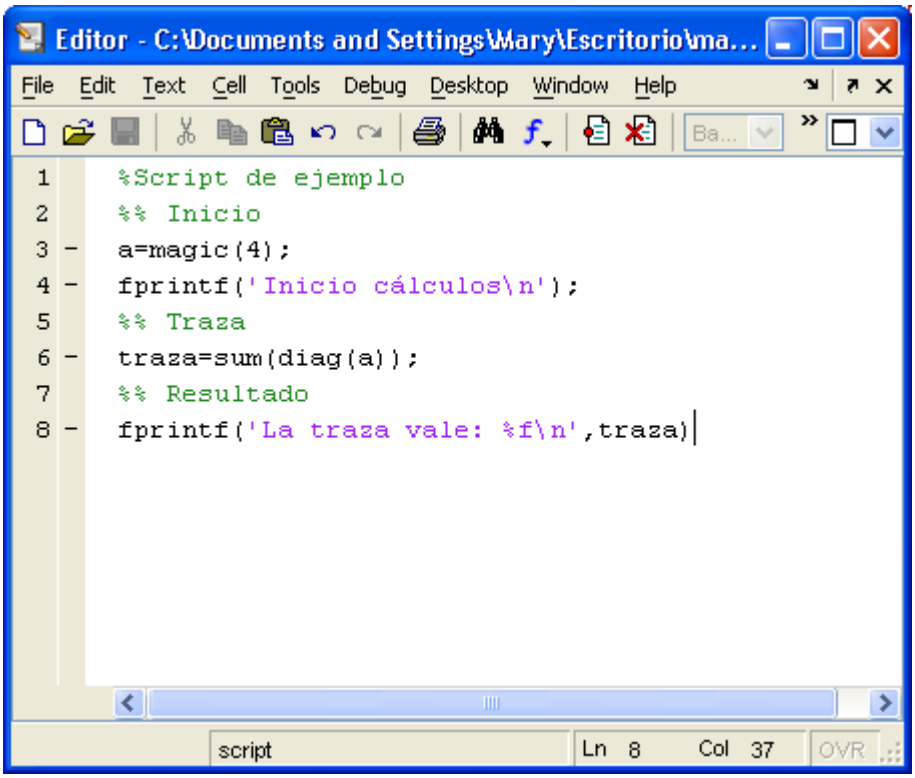
	F 1	F 2
Máxima nota		
Mínima nota		
Acumulado (suma)		
Media aritmética		
Desviación estándar		
Varianza		
Covarianza		

Programación

MATLAB es una aplicación que se puede programar muy fácilmente. Se comenzará viendo las bifurcaciones y bucles, y la lectura y escritura interactiva de variables, que son los elementos básicos de cualquier programa de una cierta complejidad.

Es posible hacer una colección de comandos y agruparlos en un archivo de tipo texto y de extensión m (.m) llamado archivo-m. Estos archivos pueden ser scripts o funciones. El script es un archivo-m que contiene una serie de comandos que se ejecutarán al ejecutar dicho archivo en MatLab. La función, es un archivo-m que permite la entrada y salida de argumentos además de la ejecución de comandos. Para crear un archivo-m se usa cualquier editor de textos, asegurándose de almacenar dicho archivo con la extensión (.m).

Para crear un archivo-M escogemos New del menú File y seleccionamos M-file. Una vez guardado este archivo-M en el disco, Matlab ejecutará las órdenes en dicho archivo simplemente escribiendo su nombre (sin extensión) en la ventana de comandos de Matlab.



Guardar el archivo con el nombre ejemplo.m

Defina o agregue la ruta donde esta guardando sus archivos:

```
>> path(path,'C:\Documents and Settings\Mary\Escritorio\matlab\CEPSUNI2010')
```

Ejecute el programa de la línea de comando de la ventana Command Window:

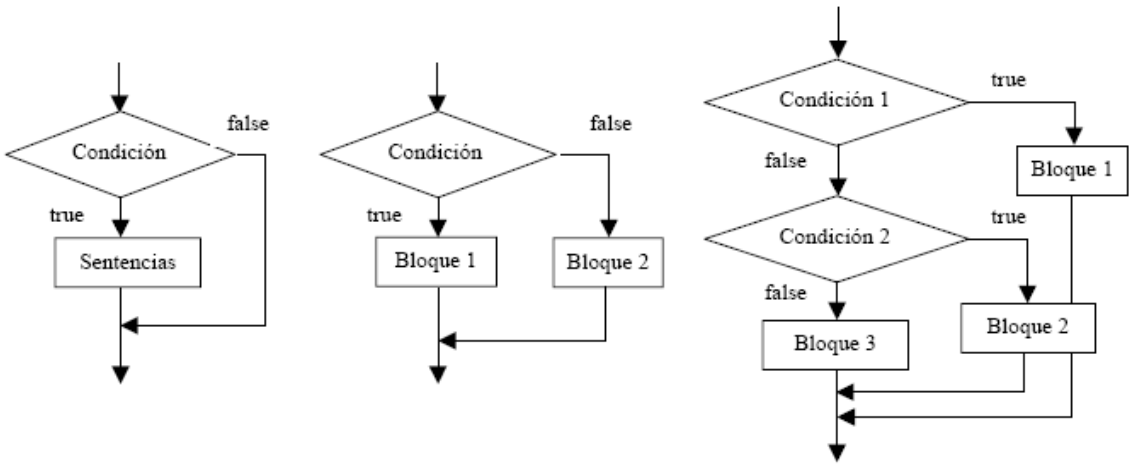
```
>> ejemplo

Inicio cálculos

La traza vale: 34.000000
```

Ordenes de gestión de archivos

what	devuelve un listado de todos los archivos-M del directorio actual.
dir	lista todos los archivos en el directorio o carpeta actual.
ls	contenido de la carpeta, igual a dir.
type test	visualiza el archivo-M test.m en la ventana de comando.
delete test	elimina el archivo-M test.m.
cd path	cambia al directorio o carpeta dada por path.
chdir path	lo mismo que cd path.
cd	muestra el directorio o carpeta de trabajo presente.
chdir	lo mismo que cd.
pwd	lo mismo que cd.
which test	visualiza el camino del directorio de test.m.

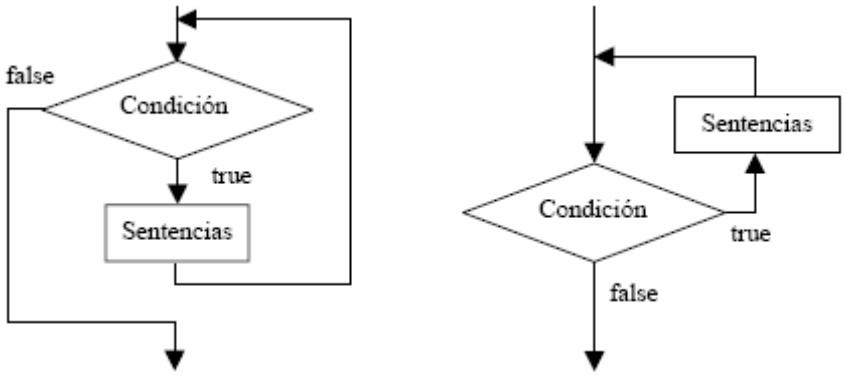
Bifurcaciones y bucles

MATLAB posee un l enguaje de pr ogramación que –como cualquier otro lenguaje– dispone de sentencias para realizar bifurcaciones y bucles. Las bifurcaciones permiten realizar una u otra operación según se cumpla o no una determinada condición.



Ejemplos gráficos de bifurcaciones.

Los bucles permiten repetir las mismas o análogas operaciones sobre datos distintos. Mientras que en C /C++/ Java el "cuerpo" de estas sentencias se determinaba mediante llaves {...}, en MATLAB se utiliza la palabra end con análoga finalidad. Existen también algunas otras diferencias de sintaxis.



Bucles con control al principio y al final.

La Figura muestra dos posibles formas de bucle, con el control situado al principio o al final del mismo. Si el control está situado al comienzo del bucle es posible que las sentencias no se ejecuten ninguna vez, por no haberse cumplido la condición cuando se llega al bucle por primera vez. Sin embargo, si la condición está al final del bucle las sentencias se ejecutarán por lo menos una vez, aunque la condición no se cumpla. Muchos lenguajes de programación disponen de bucles con control al principio (for y while en C /C++/Java) y al final (do … while en C/C++/Java). En MATLAB no hay bucles con control al final del bucle, es decir, no existe construcción análoga a do ... while.

Sentencia if

En su forma más simple, la sentencia if se escribe en la forma siguiente:

```
if condicion sentencias
end
```

Existe también la bifurcación múltiple, en l a que pueden concatenarse tantas condiciones como se desee, y que tiene la forma:

```
if condicion1 bloque1
      elseif condicion2 bloque2
      elseif condicion3 bloque3
else % opción por defecto para cuando no se cumplan las condiciones 1,2,3
      bloque4
end
```

donde la opción por defecto else puede ser omitida: si no está presente no se hace nada en caso de que no se cumpla ninguna de las condiciones que se han chequeado.

Una observación muy importante: la condición del if puede ser una condición matricial, del tipo A==B, donde A y B son matrices del mismo tamaño. Para que se considere que la condición se cumple, es necesario que sean iguales dos a dos todos los elementos de las matrices A y B (aij=bij, $1 \le i \le m$, $1 \le j \le n$). Basta que haya dos elementos aij y bij diferentes para que las matrices ya no sean iguales, y por tanto las sentencias

del if no se ejecuten. Análogamente, una condición en la forma A~=B exige que todos los elementos sean diferentes dos a dos (aij≠bij, 1≤i≤m, 1≤j≤n). Bastaría que hubiera dos elementos aij y bij iguales para que la condición no se cumpliese. En resumen:

if A==B exige que todos los elementos sean iguales dos a dos if A~=B exige que todos los elementos sean diferentes dos a dos como se ha di cho, MATLAB dispone de funciones especiales para ayudar en el chequeo de co ndiciones matriciales. Por ejemplo, la función isequal(A, B) devuelve un uno si las dos matrices son idénticas y un cero en caso de que difieran en algo.

Ejemplo 1. Dados dos números a y b si a es mayor que b entonces intercambiar los valores.

```
if a > b
     tmp=a;
     a=b;
     b=tmp;
end
```

prog01.m

```
>> a=16; b=-7;
>> prog01
>> a
a =
  -7
>> b
b =
  16
```

Si a es mayor que b entonces intercambia el valor de las variables.

Ejemplo 2. Si el número dado es 7 entonces lo cambia por cero, en caso contrario lo cambia a 1.

```
>>n=17;
```

```
if n==17
    n=0
else
    n=1
end;
```

prog02.m

```
>>prog02
n =
   0
```

Donde el 0 proviene de entrar al primer if, y el 1, de entrar al else del segundo if.

```
>>n=7
>>prog02
n =
    1
```

Ejemplo 3. Si el número dado es diferente de cero entonces lo cambia por uno, en caso contrario lo cambia a 3.

```
if m~=0
   m=1
else
   m=3
end;
```

prog03.m

```
>>m=9
>>prog03
m =
    1
>>m=0
>>prog03
m =
    3
```

Donde el 1proviene de entrar al primer if, y el 3, de entrar al else del segundo if.

Ejemplo 4. Si nota es mayor o igual a 13 imprimir aprobado en caso contrario desaprobado.

```
if nota>=13
   fprintf('aprobado\n')
else
   fprintf('desaprobado\n')
end;
```

prog04.m

```
>>nota=11
>>prog04
desaprobado
>>nota=15
>>prog04
aprobado
```

Ejemplo 5. Imprimir una matriz magic de orden:

n si el resto de dividir entre 2 es diferente de cero,

n-2 si el resto de dividir entre 3 es diferente de cero,

n-1 en otro caso.

```
if rem(n,2) ~= 0
        M = magic(n)
elseif rem(n,3) ~= 0
        M = magic(n-2)
else
        M = magic(n-1)
end
```

prog05.m

```
>>n=7
>>prog05
```

```
>>n=8
>>prog05
```

```
>>n=6
>>prog05
```

Sentencia switch

La sentencia switch realiza una función análoga a un conjunto de if...elseif concatenados. Su forma general es la siguiente:

```
switch switch_expresion
case case_expr1,bloque1
case {case_expr2, case_expr3, case_expr4,...}
bloque2 ...
otherwise, % opción por defecto
bloque3
end
```

Al principio se evalúa la switch_expresion, cuyo resultado debe ser un número escalar o una cadena de caracteres. Este resultado se compara con las case_expr, y se ejecuta el bloque de sentencias que corresponda con ese resultado. Si ninguno es igual a switch_expresion se ejecutan las sentencias correspondientes a otherwise. Según puede verse en el ejemplo anterior, es posible agrupar varias condiciones dentro de unas llaves (constituyendo lo que se llama un cell array o vector de celdas); basta la igualdad con cualquier elemento del cell array para que se ejecute ese bloque de sentencias. La "igualdad" debe entenderse en el sentido del operador de igualdad (==) para escalares y la función strcmp() para cadenas de caracteres). MATLAB sólo se ejecuta uno de los bloques relacionado con un case.

Ejemplo 6. Ingrese un valor del 1 al 4, para cada caso imprima un mensaje; en caso contrario imprimir ninguna de las anteriores

```
s=7;
switch(s)
   case 1,
      resp='windows'
   case 2,
      resp='matlab'
   case 3,
      resp='octave'
   case 4,
      resp='linux'
   otherwise,
      resp='Ninguna de las anteriores'
end
```

prog06.m

```
>>prog06
resp =
Ninguna de las anteriores
```

Sentencia for

La sentencia for repite un conjunto de sentencias un número predeterminado de veces. La siguiente construcción ejecuta sentencias con valores de i de 1 a n, variando de uno en uno.

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

o bien,

```
for i=vectorValores sentencias
end
```

donde vectorValores es un vector con los distintos valores que tomará la variable i.

Ejemplo 7 Imprime valores del 1 al 9.

```
for i=1:9
   i
end
```

prog07.m

>>prog07

Ejemplo 8 Imprime la matriz magic de orden 3 al 7

```
for i=3:7
   magic(i)
end
```

prog08.m

>>prog08

En el siguiente ejemplo se presenta el caso más general para la variable del bucle (valor_inicial: incremento: valor_final); el bucle se ejecuta por primera vez con i=n, y luego i se va reduciendo de 0.2 en 0.2 hasta que llega a ser menor que 1, en cuyo caso el bucle se termina:

```
for i=n:-0.2:1 sentencias
end
```

Ejemplo 9.

```
for i=k:-0.2:1
   i
end
```

prog09.m

```
>>k=4
>>prog09
```

En el siguiente ejemplo se presenta una es tructura correspondiente a dos bucles anidados. La variable j es la que varía más rápidamente (por cada valor de i, j toma todos sus posibles valores):

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

Ejemplo 10.

```
m=4; n=5;
for i=1:m
   for j=1:n
      A(i,j)=i+j;
   end
   end
A
```

prog10.m

```
>>prog10
```

Una última forma de interés del bucle for es la siguiente (A es una matriz):

```
for i=A sentencias
end
```

en la que la variable i es un vector que va tomando en cada iteración el valor de una de las columnas de A.

Cuando se introducen interactivamente en la línea de comandos, los bucles for se ejecutan sólo después de introducir la sentencia end que los completa.

Ejemplo 11.

```
for i=A
   fprintf('mi valor es: %f\n',i)
end
```

prog11.m

```
>>prog11
```

- Elimine la fila de fprintf y escriba i, explique que hace el programa.

  >>prog11

Sentencia while

La estructura del bucle while es muy similar a la de C/C++/Java. Su sintaxis es la siguiente:

```
while condicion sentencias
end
```

donde condicion puede ser una expresión vectorial o matricial. Las sentencias se siguen ejecutando mientras haya elementos distintos de cero en condicion, es decir, mientras haya algún o algunos elementos true. El bucle se termina cuando todos los elementos de condicion son false (es decir, cero).

Ejemplo 12.

```
p=5;
while p
   fprintf('mi valor es: %f\n',p)
end
```

Prog12.m

>>prog12

Ejemplo 13.

```
p=5;
while p>0
   fprintf('mi valor es: %f\n',p);
   p=p-1;
end
```

Prog13.m

>>prog13

Sentencia break

La sentencia break hace que se termine la ejecución del bucle for y/o while más interno de los que comprenden a dicha sentencia.

Sentencia continue

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

Aplicación

1. Calcular la suma de los n primeros términos de la sucesión 1, 2x, 3x^2, 4x^3, ...

```
n=input('¿Cuántos términos quieres sumar?  ');
x=input('Dame el valor del numero x  ');
suma=1;
for i=2:n
  suma=suma+i*x^(i-1);
end
disp('El valor pedido es')
disp(suma)
```

2. Escribir un número natural en una base dada (menor que diez).

```
n=input('Dame el número que quieres cambiar de base ');
base=input('¿En qué base quieres expresarlo? ');
i=1;
while n>0
  c(i)=rem(n,base);
  n=fix(n/base);   % Parte entera de n/base
  i=i+1;
```

```
end
disp('La expresión en la base dada es:')
i=i-1;
disp(c(i:-1:1))
```

3. Decidir si un número natural es primo.

```
n=input('Número natural que deseas saber si es primo ');
i=2;
primo=1;
while i<=sqrt(n)
 if rem(n,i)==0   % Resto de dividir n entre i
     primo=0;
   break
   end
   i=i+1;
end
if primo
   disp('El número dado es primo.')
else
   disp('El número dado no es primo.')
   disp('De hecho, es divisible por:')
   disp(i)
end
```

Funciones

También pueden programarse funciones. La primera instrucción de un fichero que contenga una función de nombre fun debe ser:

```
function [argumentos de salida]=fun(argumentos de entrada)
```

Es conveniente que el fichero que contenga la función se llame como ella; así, la función anterior debería guardarse en el fichero fun.m; por ejemplo, si se desea programar una función que calcule, mediante el algoritmo de Euclides, el máximo común divisor de dos números naturales, basta escribir un fichero euclides.m cuyo contenido sea:

```
function m=euclides(a,b)
% Cálculo del máximo común divisor de dos números naturales
% mediante el algoritmo de Euclides
if a<b
  c=b;
  b=a;
  a=c;
end
while b>0
  c=rem(a,b);
  a=b;
  b=c;
end
m=a;
```

Si, una vez escrito el fichero anterior, en el espacio de trabajo o en un programa se escribe la instrucción

```
mcd=euclides(33,121)
```

en la variable mcd se almacenará el valor 11.

Las variables de una función son siempre locales. Por tanto, aunque en el seno de la función se modifiquen los argumentos de entrada, el valor de las variables correspondientes queda inalterado. Por ejemplo, en la función euclides.m se modifica el valor de los argumentos de entrada, pero, sin embargo:

```
>>x=15;
>>mcd=euclides(x,3);
>>x
x =
      15
```

Si se pretende que las modificaciones de un argumento de entrada afecten a la variable correspondiente, deberá situarse dicho argumento, además, en la lista de argumentos de salida.

- Calcular el promedio de los elementos de un vector y dibujar dicho vector.

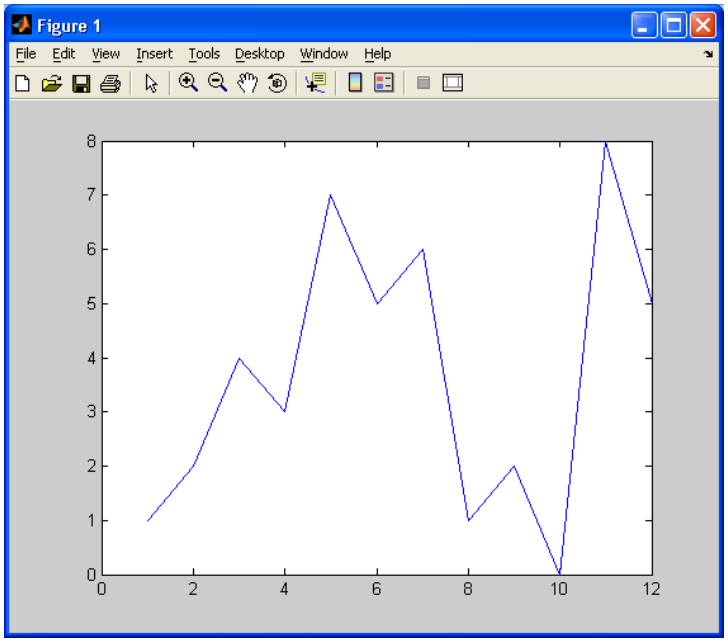
```
% Calcula el promedio de los elementos de un vector
% y dibuja dicho vector
% Sintaxis: promedio(x) donde x es el vector a promediar
```

```
function p = promedio(x)
n=length(x);
p=0;
for i=1:n
p=p+x(i);
end
p=p/n;
plot(x);
```

Para ejecutar la función, se hace el llamado en la línea de comandos incluyendo el parámetro. La función promedio usa por parámetro un vector. Este vector debe ser definido previamente.

```
>>A=[1 2 4 3 7 5 6 1 2 0 8 5];
>>promedio(A)
ans =
3.6667
```

MatLab despliega las imágenes en una ventana de figuras. Al observar el contenido de dicha ventana luego de ejecutar la función promedio, se tiene:



Funciones de usuario propias

Función que devuelve una sola variable

Consideremos un archivo M de función para la siguiente ecuación:

$$f(x) = \frac{2x^3 + 7x^2 + 3x - 1}{x^2 - 3x + 5e^{-x}} \quad \ldots(\alpha)$$

Suponiendo que el archivo M se guarda como fun, su guión sería el siguiente:

```
function y=fun(x)
y=(2*x.^3+7*x.^2+3*x-1)./(x.^2-3*x+5*exp(-x));
```

Usemos el comando fun aplicado para el valor de x=5

```
>> fun(5)

y =
  43.7526
```

Si el argumento es una matriz, por ejemplo:

```
>> fun([4 7; 2 -2])
ans =
  61.3455   37.4582
 -37.0280    0.1065
```

Función que devuelve múltiples variables

Una función puede devolver más de una variable. Supongamos una función que evalúa la media y la desviación estándar de una serie de datos. Para devolver las dos variables utilizamos un vector en el miembro izquierdo del enunciado de la función; por ejemplo,

```
function [media, dvstd]=media_ds(x)
n = length(x);
media = sum(x)/n;
dvstd = sqrt(sum(x.^2)/n - media.^2);
```

Para utilizar esta function, el miembro derecho del enunciado de llamada también debe ser un vector.

```
>> x=[1 5 3 4 6 5 8 9 2 4];
>> [m, d] = media_ds(x)
Produce
m =
   4.7000
d =
   2.3685
```

Función que utiliza otra función

El argumento de una función puede ser el nombre de otra función.

Ejm supongamos una función que evalúa la media ponderada de una función en tres puntos como

$$f_{av} = \frac{f(a) + 2f(b) + f(c)}{4}$$

Donde f(x) es la función que se nombrará en el argumento. Realizar el guión fav que calcula la función anterior.

```
function mp = fav(nomf,a,b,c)
mp = (feval(nomf,a) + 2*feval(nomf,b)+feval(nomf,c))/4;
```

En el guión anterior, nomf (una variable de cadena) es el nombre de la función f(x).

Si f(x) es la función seno, nomf será 'sin'.

feval (nomf,x) es un comando de Matlab que evalúa la función llamada nomf para el argumentos.

Por ejemplo, y=fevcal(´sin',x) equivale a y=sin(x).

Evaluar la ecuación para la función (α) con a=1, b=2 y c=3.

```
>> A = fav('fun', 1,2,3)
Produce
A =
  89.8976
```

Gráficos

Matlab permite crear gráficos de varios tipos, que se utilizan para:

– visualizar el contenido de las variables

– crear imágenes/películas/VR/GIS

– generar interfaces de usuario

Graficar: suponga que desea graficar un conjunto de punto de datos, (xi,yi), i=1,2,…n. Es necesario preparar x y y en forma de arreglo idéntica, es decir, convertirlos en arreglos de fila o de columna de la misma longitud. Los datos se grafican con plot.

Gráficos 2D

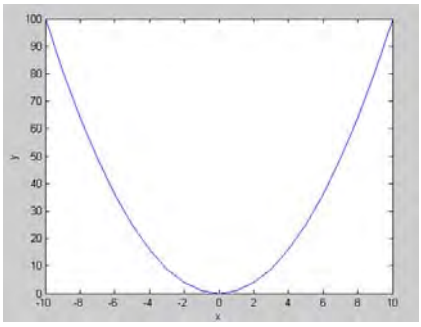
- $y=x^2$, $-10 \leq x \leq 10$

  >> x=-10:10;

  >> y=x.^2;

  >> plot (x,y)

  >> xlabel('x'); ylabel('y');



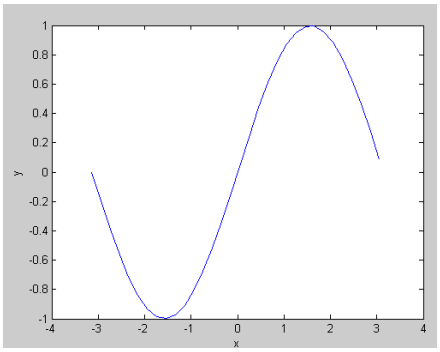
- y=seno(x), $-\pi \leq x \leq \pi$

  >> x=……;

  >> y=……;

  >> plot (x,y)

  >> xlabel('x'); ylabel('y');



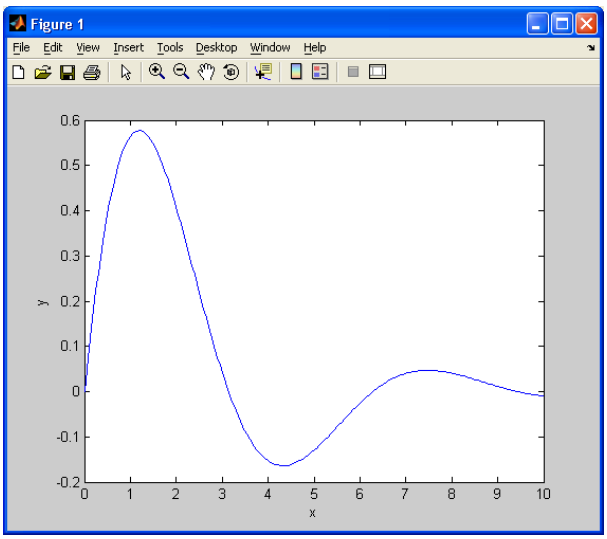
- y=sen(x)exp(-0.4x), $0 \leq x \leq 10$

  >> x=0:0.05:10;

  >> y=sin(x).*exp(-0.4*x);

  >> plot (x,y)

  >> xlabel('x'); ylabel('y');



Funciones para gráficos

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 mouse	>> 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

Tipos de colores de líneas

Tipos de línea	Símbolo
Continua	-
Guiones	--
Punteada	:
Guiones y punto	-.

El tipo de línea por omisión es el continuo. Si desea graficar con un mismo tipo de línea en particular especifique la marca de línea después de las coordenadas; por ejemplo, plot(x,y,'—')

Se dispone los siguientes colores:

Color de línea	Símbolo
rojo	r
amarillo	y
magenta	m
turquesa	c
verde	g
azul	b
blanco	w
negro	k

Utilice el símbolo del color igual que los tipos de línea en el argumento de plot; por ejemplo, plot(x,y,'g')

También es posible combinar marcas y colores: plot(x,y,'+g') grafica los datos con marcas + de color verde.

Borrados de gráficas

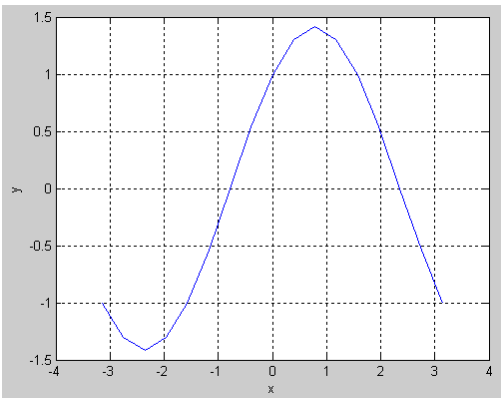
clf borra todo lo que haya en la ventana de gráficos.

cla borra las curvas graficadas y redibuja los ejes

Retícula

Se puede agregar una retícula a la gráfica con grid on. Por otro lado, grid off elimina la retícula. El empleo de grid por sí solo activa y desactiva la retícula alternadamente. El siguiente guión es un ejemplo del empleo de gris on:

```
>> u=-pi:pi/8:pi;
>> h=sin(u)+cos(u);
>> plot(u,h)
>> grid
>> xlabel('x'); ylabel('y')
```



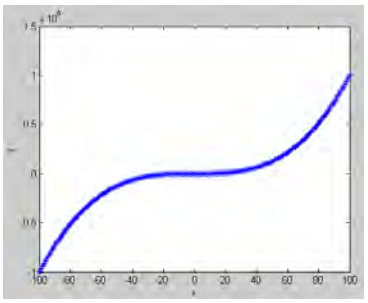
Graficación únicamente con marcas

Los datos pueden graficarse sólo con marcar sin estar conectados por líneas.

Tipos de marca	Símbolo
Punto	.
Más	+
Estrella	*
Círculo	o
Marca x	x

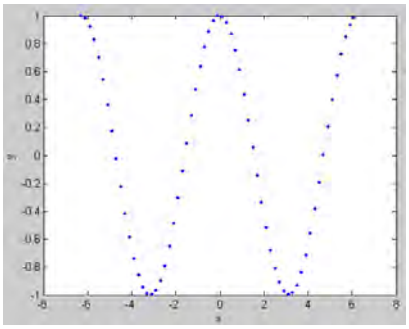
- $y=x^3$, $-100 \leq x \leq 100$

```
>> x=-10:10;
>> y=x.^3+1;
>> plot (x,y,'*')
>> xlabel('x'); ylabel('y');
```



- y=coseno(x), $-2\pi \leq x \leq 2\pi$

```
>> x=……;
>> y=……;
>> plot (x,y,'.')
>> xlabel('x'); ylabel('y');
```



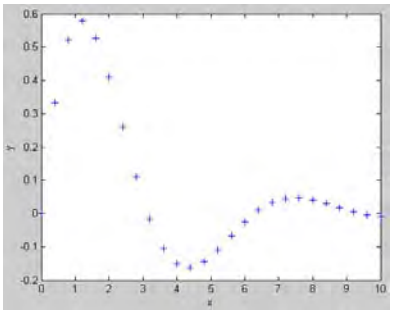
- y=sen(x)exp(-0.4x), $0 \leq x \leq 10$

  >> x=(0:0.4:10)';

  >> y=sin(x).*exp(-0.4*x);

  >> plot(x,y,'+')

  >> xlabel('x'); ylabel('y')



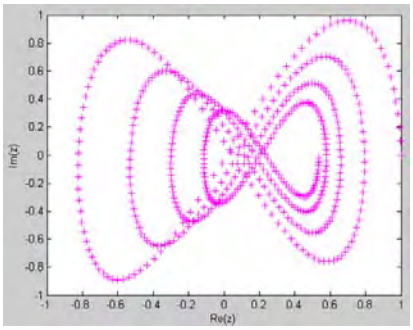
- z=coseno(p)+***i***seno(2p)exp(-0.05p)+0.01p

  >> p=0: 0.05: 8*pi;

  >> z=(cos(p) + i*sin (2*p)).*exp(-0.05*p)+0.01*p;

  >> plot(real(z), imag(z),'g')

  >> xlabel('Re(z)'); ylabel('Im(z)')



- Grafica de la campana de gauss

  >>x=linspace(-3,3,500); y=exp(-x.^2); z=2*exp(-x.^2);

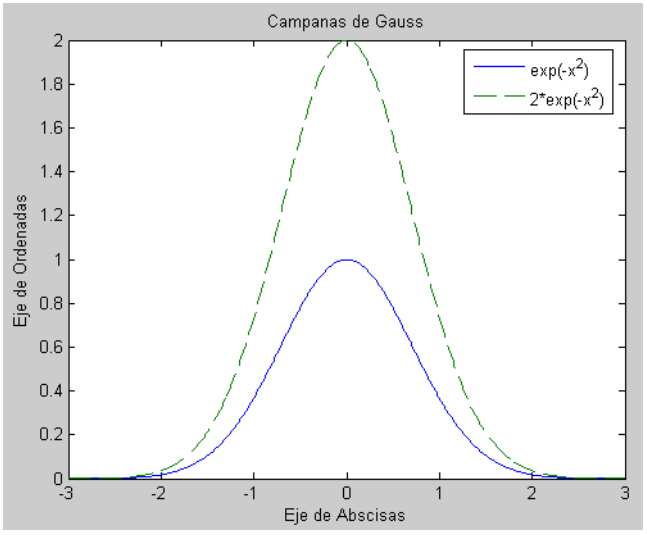
  >>plot(x,y,'-',x,z,'--') % Dibujamos dos funciones

  >>title('Campanas de Gauss')

  >>xlabel('Eje de Abscisas') % Etiqueta el eje horizontal

  >>ylabel('Eje de Ordenadas') % Etiqueta el eje vertical

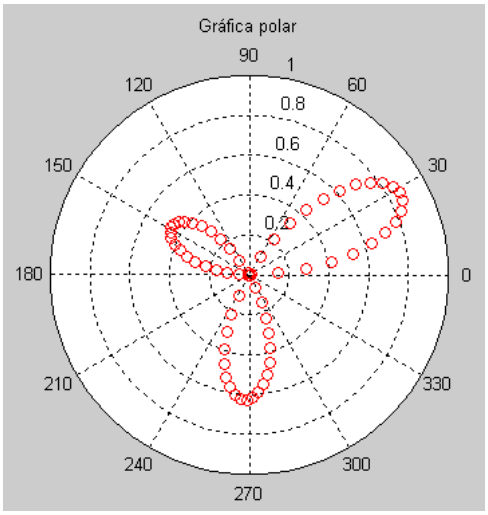
  >>legend('exp(-x^2)', '2*exp(-x^2)') % Leyenda



Gráficas polares

Podemos graficar una función en coordenadas polares con polar.

```
>> t=0:.05:pi+.01;
>> b=sin(3*t).*exp(-0.3*t);
>> polar(t,y)
>> title('Gráfica polar')
>> polar(t,b,'+c')
>> polar(t,b,'+r')
>> polar(t,b,':r')
>> polar(t,b,'or')
```



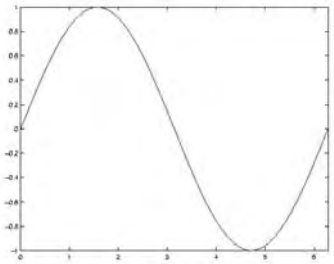
Limpiar la memoria

```
Clear
```

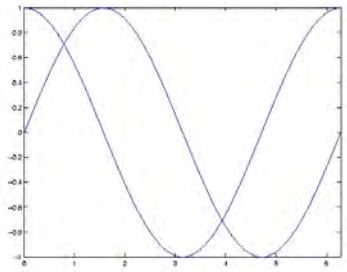
Dos gráficos en uno

También pueden dibujarse funciones. Así:

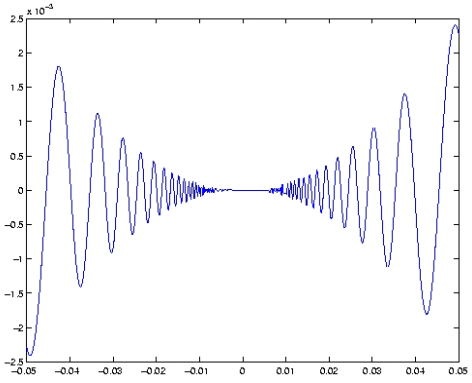
```
>>fplot('sin(x)',[0 2*pi])
% Dibuja la función seno en el intervalo [0,2*pi]
```



```
>>hold on
% Mantiene en la ventana gráfica los dibujos anteriores

>>fplot('cos(x)',[0 2*pi])
% Dibuja sobre la gráfica anterior la función cos(x)
>>hold off      % Con esto olvida los dibujos
                % anteriores  y dibuja en una ventana nueva
```

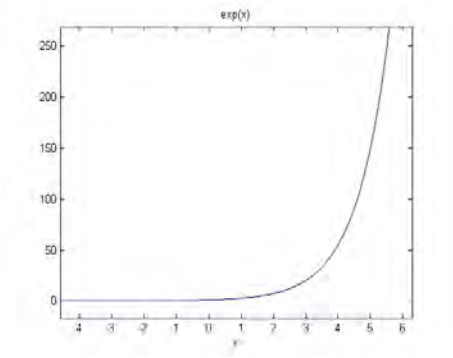


```
>>fplot('x^2*sin(1/x)',[-0.05 0.05])
 % Dibuja la función x^2*sin(1/x)
```



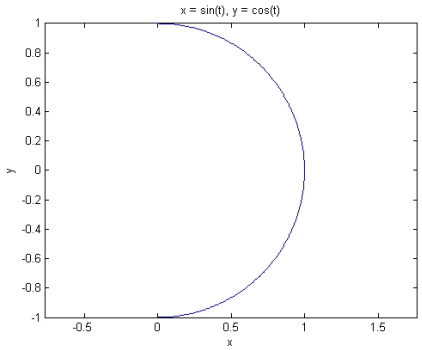
Grafica de función

```
>>ezplot('exp(x)')
% Dibuja la función exponencial en un intervalo adecuado a la función
```



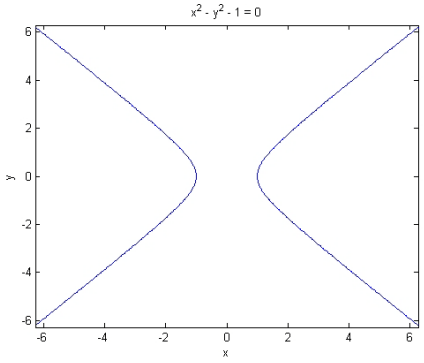
Grafica de paramétrica

```
>>ezplot('sin(t)','cos(t)',[0 pi])
```



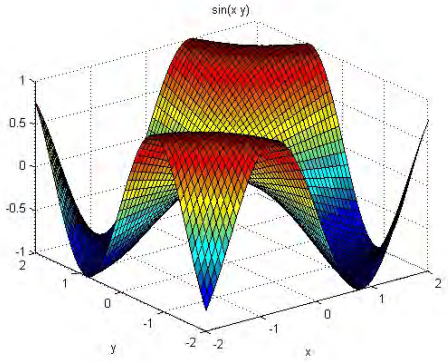
Grafica de función implícita

```
>>ezplot('x^2 - y^2 - 1')
```



También permite dibujar superficies. La forma más sencilla es mediante el comando ezsurf,

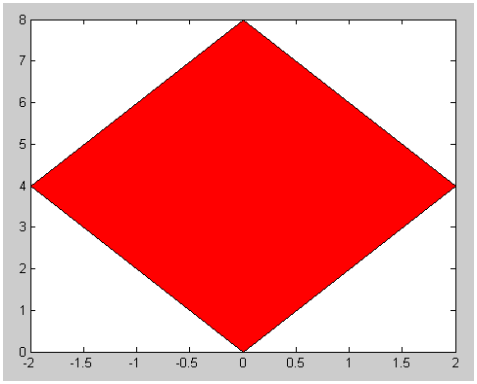
```
>>ezsurf('sin(x*y)',[-2 2 -2 2])
```



Polígonos

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.

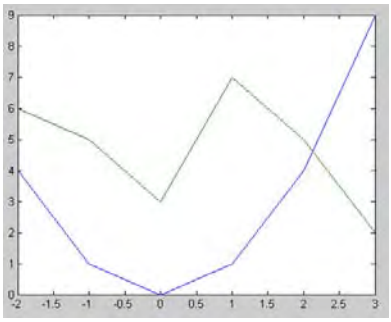
```
>> x=[-2 0 2 0 -2]; y=[4 8 4 0 4];
>> plot(x,y)
% dibuja el polígono, 'r' indica el color rojo
>> fill(x,y,'r') %
```



Gráficos con el mismo eje

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

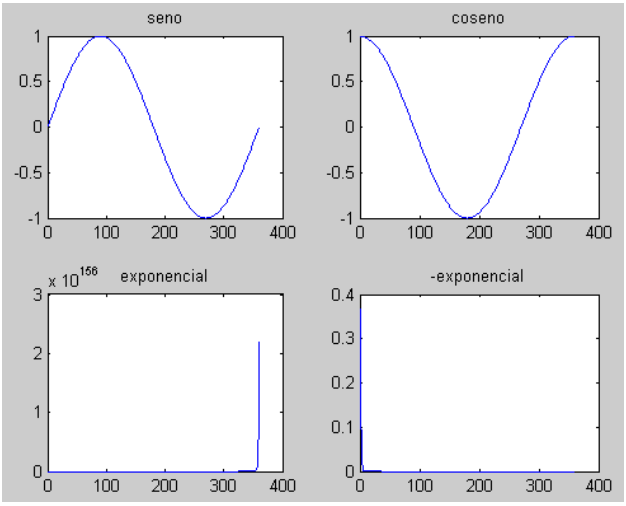
```
>> x=[-2 -1 0 1 2 3]; y=[4 1 0 1 4 9]; z=[6 5 3 7 5 2];
>> plot(x,y,x,z)
```



Múltiples graficas

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).

Otras propiedades gráficas

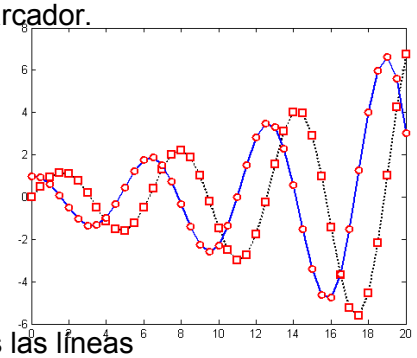
También se pueden e specificar otras propiedades gráficas de l a línea mediante parejas de nombre de propiedad y valor. Estas propiedades se asignarán a todas las líneas del gráfico. Algunas de las propiedades más habituales son:

• LineWidth: Especifica el grosor de la línea en píxeles.

• MarkerEdgeColor: Especifica el color del borde del marcador.

• MarkerFaceColor: Especifica el color del interior del marcador.

• MarkerSize: Especifica el tamaño del marcador.

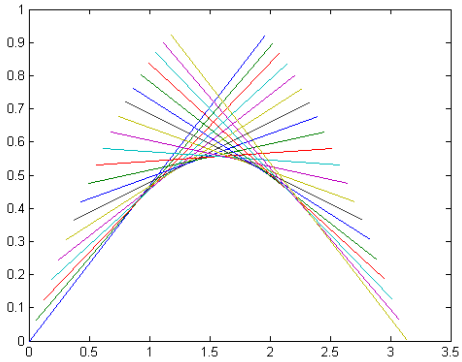
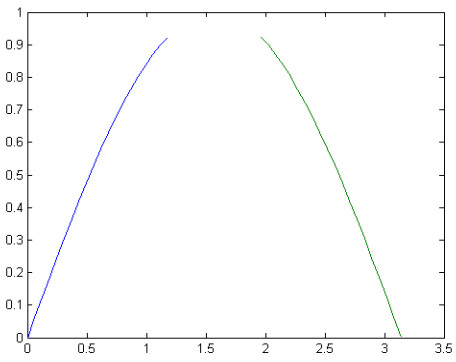


```
>> x = 0:0.5:20;
>> y = exp(0.1*x);
>> y1 = y.*sin(x);
>> y2 = y.*cos(x);
% Las propiedades especificadas afectan a todas las líneas
>> plot (x, y1, ':ks', x, y2, '-bo', 'LineWidth', 2, 'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'w', 'MarkerSize', 7)
```

Gráfico de función a trozos

Gráfico de la función seno en dos tramos [0, 3π/8] y [5π/8, π]

```
>> X = [linspace(0,pi*3/8,20); linspace(pi*5/8, pi, 20)];
>> Y = sin(X);
>> plot(X,Y) %Hay que tener en cuenta que se imprime por columnas
```

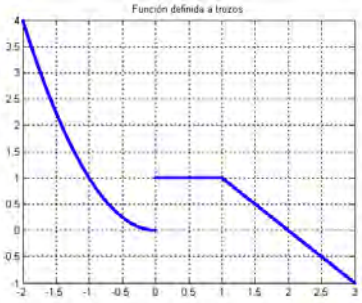



>> plot(X',Y') % Ahora sí

% Es equivalente a: plot(X(1, :), Y(1, :), X(2, :), Y(2, :))

- Graficar la función

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

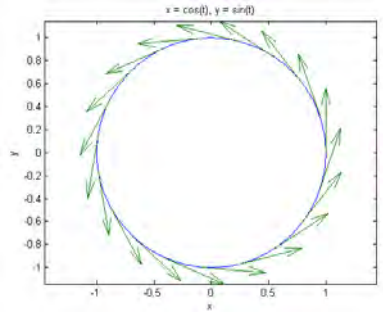


>> x=linspace(-2,3,3000);

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

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

Campo vectorial 2D



Dibujar los vectores velocidad sobre la curva

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

>> ezplot('cos(t)','sin(t)',[0 2*pi])

>> hold on

>> t=linspace(0,2*pi,20);

>> quiver(cos(t),sin(t),-sin(t),cos(t)),axis square

Cambios de coordenadas polares-cartesianas

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

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

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

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

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

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	

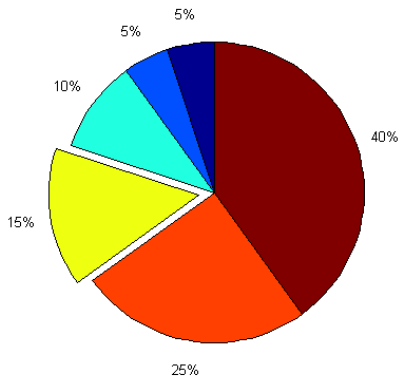
Gráficos Estadísticos

Gráfico de sectores

```
>> x = [1 1 2 3 5 8];
>> pie(x);
```

Gráfico de sectores con elementos resaltado

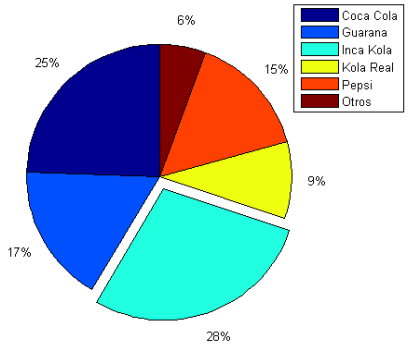
```
>> x = [1 1 2 3 5 8];
>> ex = [0 1 0 0 1 0];
>> pie(x, ex);
```



Ejercicio

Realizar un grafico de sectores en el cual se resalte el sector de Inca Kola y se indique la leyenda por color del sector correspondiente.

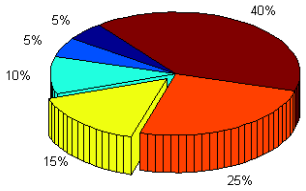
Gaseosa	Coca Cola	Guarana	Inca Kola	Kola Real	Pepsi	Otros	Total
Preferencia	13	9	15	5	8	3	53




```
>> g=[13 9 15 5 8 3];
>> gg=[0 0 1 0 0 0];
>> pie(g, gg);
>> legend('Coca Cola', 'Guarana','Inca Kola','Kola Real','Pepsi','Otros')
```

Gráfico de sectores tridimensional

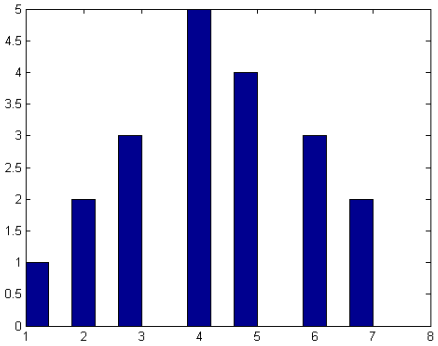
```
>> x = [1 1 2 3 5 8];
>> ex = [0 0 0 1 0 0];
>> pie3(x, ex);
```



Histograma

Para generar histogramas se utiliza el comando hist. Por ejemplo, generamos 1000 números aleatorios siguiendo la normal N(0; 1)

```
>> x = randn(1,10000);
>> hist(x);
```



Histograma de 50 barras

```
>> x = randn(1,10000);
```

```
>> hist(x, 50);
```

Ejercicio

Con los siguientes datos construya un histograma

1	3	7	5	2	2	3	4	4	4	3	5	5	6	7	6	5	6	4	4

```
>> f = [1 3 7 5 2 2 3 4 4 4 3 5 5 6 7 6 5 6 4 4];
>> hist(f ,15);
```

Diagrama de pareto

El diagrama de Pareto que produce MatLab constará de barras cuyas alturas son el número de alumnos, ordenadas en forma decreciente y sobre las barras, un po lígono con las frecuencias acumuladas del número de alumnos.

Además, en el eje vertical derecho aparece una esca la de porcentajes.

De un grupo de alumnos se encontró las siguientes preferencias:

Deporte	**No.**
Futbol	30
Básquet	25
Tennis	17
Voley	9
Total	81

```
>> x=[30 25 17 9];
>> pareto(x),ylabel('Número de alumnos')
```

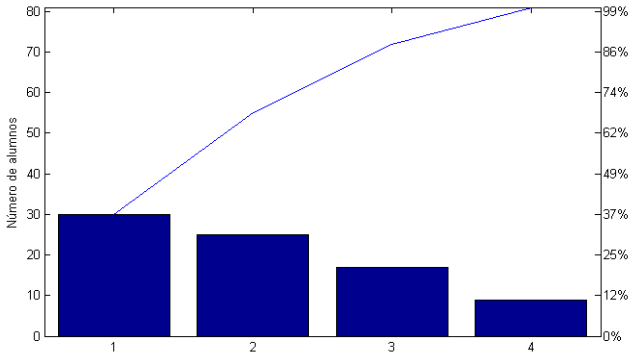


Gráfico de barras

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

Introducimos los datos en un vector

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

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

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

Rotar

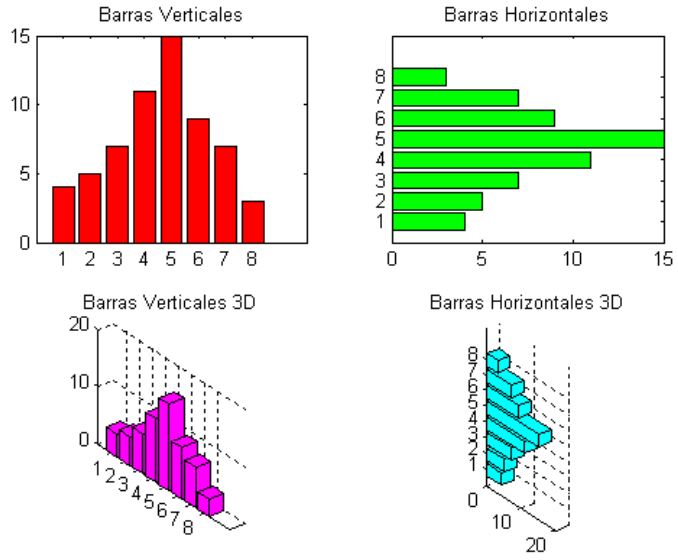
Hay que observar que las gráficas 3D se pueden modificar utilizando el comando rotate3d

Ejercicio

Realizar un grafico de barras de Días transcurridos vs el número de reclamos resueltos.

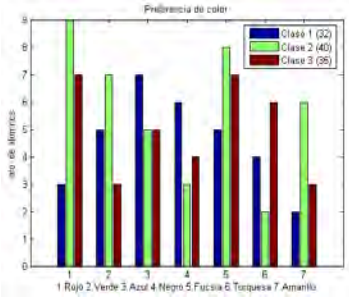
Dias	1	2	3	4	5	6	7	8
N° de reclamos	4	5	7	11	15	9	7	3

```
>> x = [1 2 3 4 5 6 7 8];
>> y = [4 5 7 11 15 9 7 3];
>> subplot(2,2,1),bar(x,y,'r'),title('Barras Verticales')
>> subplot(2,2,2),barh(x,y,'g'),title('Barras Horizontales')
>> subplot(2,2,3),bar3(x,y,'m'),title('Barras Verticales 3D')
>> subplot(2,2,4),bar3h(x,y,'c'),title('Barras Horizontales 3D')
```



Grupos de barras

```
>> subplot(1,1,1)
>> x = 0.5:0.5:4;
>> y = 1./x;
>> Y = [y' fliplr(y)'];
%en 2D dimensiones
>> bar(x, Y)
%en 3D dimensiones
>> bar3(x, Y,'group')
```



Ejercicio

De tres salones de clase, se encuesto la preferencia por los colores, según la tabla realizar un grafico de barras

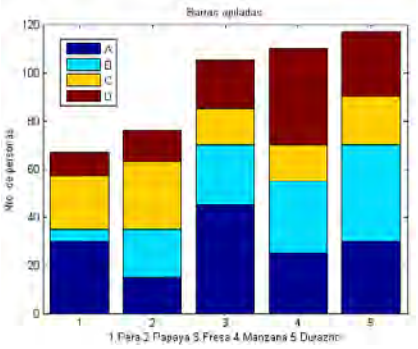
Color	r	v	az	n	m	f	am
Clase 1 (32)	3	5	7	6	5	4	2
Clase 2 (40)	9	7	5	3	8	2	6
Clase 3 (35)	7	3	5	4	7	5	3

```
>> X= [1 2 3 4 5 6 7];
>> c1 = [3 5 7 6 5 4 2];
>> c2 = [9 7 5 3 8 2 6];
>> c3 = [7 3 5 4 7 6 3];
>> Y = [c1' c2' c3'];
>> bar(X,Y);
>> title('Preferencia de color');
>> xlabel('1.Rojo 2.Verde 3.Azul 4.Negro 5.Fucsia 6.Turquesa 7.Amarillo');
>> ylabel('nro. de alumnos');
>> legend('Clase 1 (32)','Clase 2 (40)','Clase 3 (35)');
```

Barras apiladas

```
>> x = 0.5:0.5:4;
>> y = 1./x;
>> Y = [y' fliplr(y)'];
```

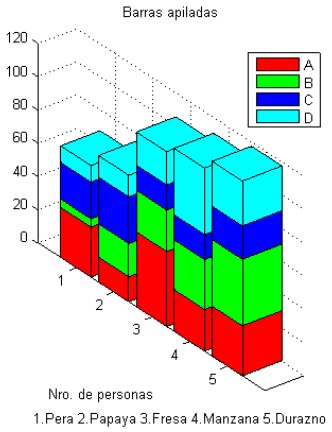
```
%en 2D dimensiones
>> bar(x, Y, 'stacked')
>> colormap([1 1 0;0 1 0])
%en 3D dimensiones
>> bar3(x, Y, 'stacked')
>> colormap([1 0 0;0 0 1])
```



Ejercicio

De cuatro distritos, se encuesto la preferencia por las frutas, según la tabla realizar un grafico de barras apiladas.

Distrito	Pera	Papaya	Fresa	Manzana	Durazno
A (145)	30	15	45	25	30
B (120)	5	20	25	30	40
C (100)	22	28	15	15	20
D (110)	10	13	20	40	27




```
>> X = 1:5;
>> d1 = [30 15 45 25 30];
>> d2 = [5 20 25 30 40];
>> d3 = [22 28 15 15 20];
>> d4 = [10 13 20 40 27];
>> Y = [d1' d2' d3' d4'];
>> bar(X, Y, 'stacked');          % 2D
>> bar3(X, Y, 0.5,'stacked');     % 3D
>> title('Barras apiladas');
>> colormap([1 0 0;0 1 0;0 0 1;0 1 1])
>> legend('A','B','C','D');
>> ylabel('Nro. de personas');
>> xlabel('1.Pera 2.Papaya 3.Fresa 4.Manzana 5.Durazno');
```

Gráfico de dispersión

Las formas más habituales de esta función son:

scatter(x, y)

scatter(x, y, size)

scatter(x, y, size, color)

Hace un gráfico de dispersión con las coordenadas de x e y. Los vectores deben ser del mismo tamaño. También se puede especificar el tamaño de los puntos y su color. Tanto el tamaño como el color se pueden e specificar para todos los puntos en conjunto o para cada punto individualmente. Para lo primero basta con especificar con un escalar el tamaño y con un vector de 1x3 el color (en RGB). Para lo segundo hay que proporcionar para el tamaño de los puntos un vector (de igual longitud que x e y) y para el color, o bien, un vector de índices (que harán referencia al colormap), o bien, una matriz de length(x)x3 con los colores (en RGB).

```
%Puntos distribuidos aleatoriamente con distribución normal en 2D
>> x = randn(1,100);
>> y = randn(1,100);
%..con puntos de color aleatorio usando el colormap
>> scatter(x, y, 100, rand(1, 100))
```

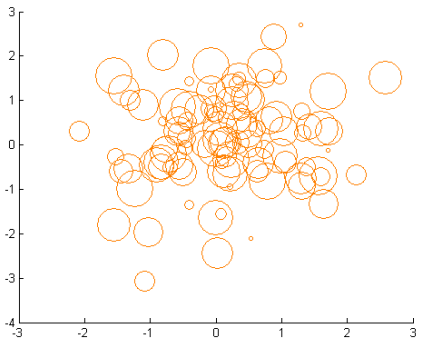


Gráfico utilizando dos ejes verticales distintos

plotyy(X1,Y1,X2,Y2)

plotyy(X1,Y1,X2,Y2,'function1','function2')

Realiza un gráfico utilizando dos ejes verticales distintos, uno para la serie Y1 (eje de la izquierda) y otro para la serie Y2 (eje de la derecha). Esta función es útil cuando se desean visualizar dos series con distinta escala y/o unidades en un mismo gráfico, por ejemplo en gráficos de temperatura y pluviometría. Se pueden especificar los nombres de función con que se quiere mostrar el gráfico. Se puede usar cualquier función que acepte llamadas del tipo function(x, y).

Gráfico con dos ejes verticales con distinta escala

```
>> x = 1:12;
```

```
>> pluv = sin(linspace(pi/2,pi+3*pi/2,12))*100+100;
>> temp = sin(linspace(0, pi, 12))*40;
>> plotyy(x, pluv, x, temp)
```

Gráfico con dos ejes verticales usando distintas funciones para dibujarlo

```
>> plotyy(x, pluv, x, temp, 'bar', 'plot')
```

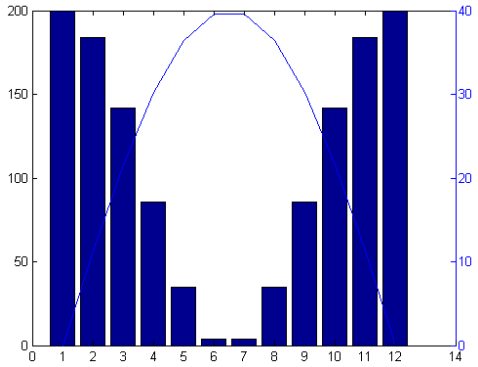
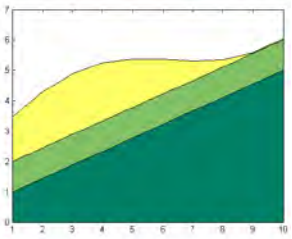


Gráfico Apilado

area(X,Y)

Esta función genera un gráfico apilado, útil para mostrar las contribuciones de distintas componentes de un todo. Si X e Y son vectores el resultado es igual a plot excepto que se rellena el área bajo la curva. Si Y es una matriz, la función acumula los distintos valores de las columnas de Y. X debe ser monótona.



Creamos una columna creciente, otra constante y otra que varia con el seno

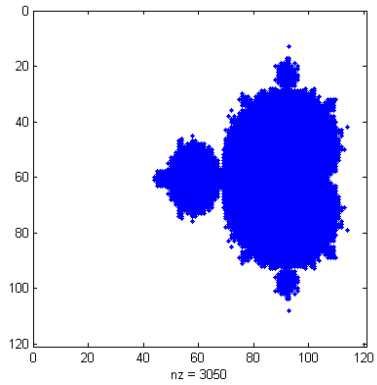
```
>> x = 1:10;
>> Y = [linspace(1,5,10)' ones(10, 1) 1+sin(0.5:0.5:5)'];
>> area(x, Y)
>> colormap summer
```

Ejemplo de Análisis Numérico

Dibujo del conjunto de Mandelbrot para la ecuación

$$f(z) = z^2 + c, \quad z, c \in \mathcal{C}$$

obtenido con mandel(120).



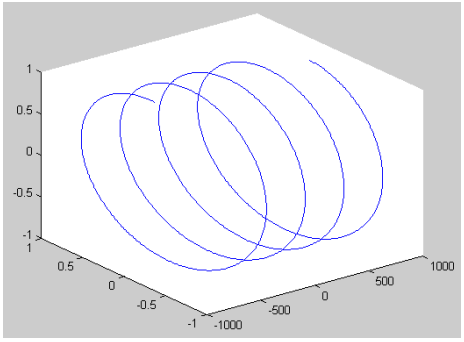

```
function mandel(n)
x=linspace(-2.5,0.6,n);
y=linspace(-1.2,1.2,n);
[X,Y] = meshgrid(x,y);
C = complex(X,Y);
Zmax = 1e6; itemax = 50;
Z=C;
for k=1:itemax
Z = Z.^2 + C;
end
% la orden spy dibuja los elementos no nulos de
% una matriz. Suele usarse para matrices ‘‘huecas"
spy( Z < Zmax);
```

3-D

Gráficos de línea

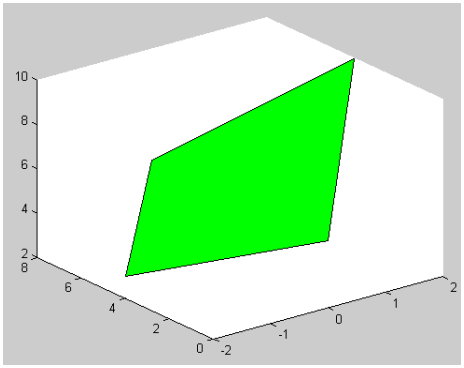
También podemos crear gráficas en 3 dimensiones, se trata de extender la orden de plot (2-D) a plot3 (3-D) donde el formato será igual pero los datos estarán en tripletes:

```
>> x=-720:720; y=sind(x); z=cosd(x);
>> plot3(x,y,z)
```



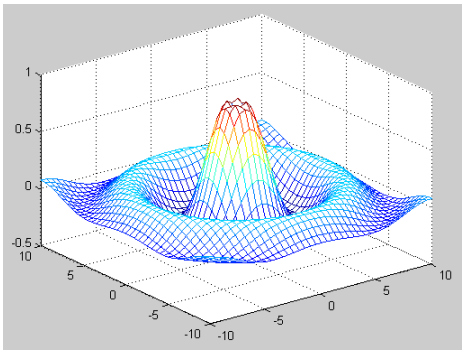
Si queremos representar un polígono en 3 dimensiones lo haremos con la función fill3 de forma similar a fill pero ahora con 4 argumentos, siendo el cuarto el que indica el color.

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



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).




```
>> 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.

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

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

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

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

```
>> 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`

```
>> 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	

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

creamos una matriz de colores

```
>> M = [0 0 0; 1 0 0; 0 1 0; 0 0 1; 1 1 0];
>> colormap (M)
```

Campo vectorial 3D

Dibujar los vectores velocidad sobre la hélice

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

```
>> t=linspace(0,8*pi,20);
>> quiver3(sin(t),cos(t),t,cos(t),-sin(t),1,1.5);
>> hold on
>> t=linspace(0,8*pi,2000);
>> plot3(sin(t),cos(t),t,'r');
>> grid on;
>> hold off;
```

Superficie

Dibujar la grafica de la función de diferentes formas $z = e^{-(x^2+y^2)}$

para -2 ≤ x ≤ 2, -2 ≤ y ≤ 2

```
>> [x,y]=meshgrid(-2:.5:2);
>> z=exp(-x.^2-y.^2);
>> subplot(2,2,1); plot3(x,y,z);
>> subplot(2,2,2); mesh(x,y,z);
>> subplot(2,2,3); surf(x,y,z);
>> subplot(2,2,4); surf(x,y,z),shading flat;
```

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

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

Vectores Normales a una superficie

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

Dibujar una esfera utilizando

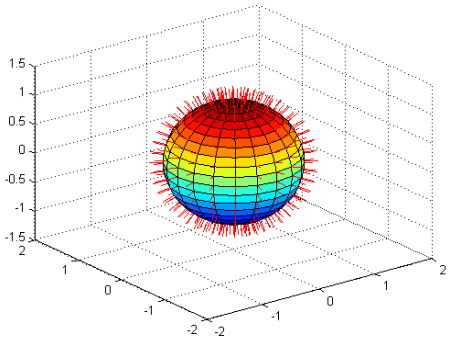
```
>> sphere,axis square,title('ESFERA')
```

luego guarde la información en tres variables

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

Utilizar el comando

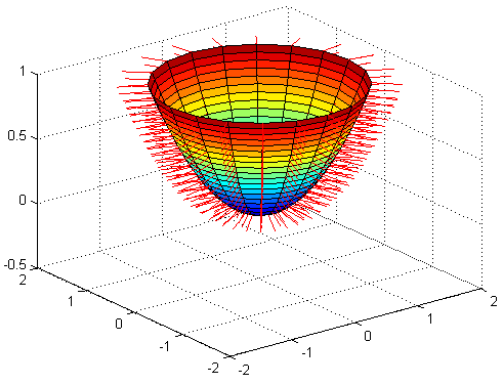
```
>> surfnorm(x,y,z)
```



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

- Dibujar un paraboloide de revolución y sus vectores normales, utilizar como generatriz la función:

$$r(t) = \sqrt{t}\ , \text{ con } t \in [0,2]$$



```
>> t=linspace(0,2,20);
>> r=sqrt(t); cylinder(r);
>> [x y z]=cylinder(r);
>> surfnorm(x,y,z)
```

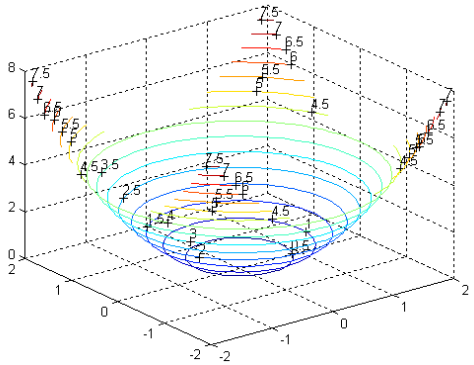
Curvas de nivel

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

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

Dibujar las curvas de nivel y etiquetarlas.

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



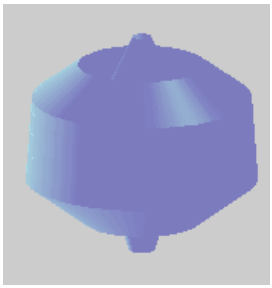
2D

```
>> contour(x,y,z,10) % curvas de nivel
>> cs=contour(x,y,z,15);
>> clabel(cs)
```

3D

```
>> contour3(x,y,z,10)
>> cs=contour3(x,y,z,15);
>> clabel(cs)
```

Superficies de revolución



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

Gráficos de funciones complejas

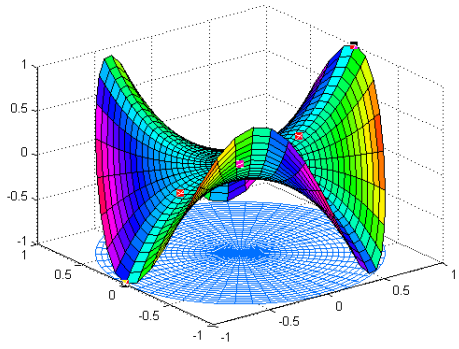
El comando cplxmap permite representar gráficas de funciones complejas de variable compleja en el siguiente sentido:

Sea la función compleja de variable compleja

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

El comando cplxmap(z,f(z)) dibuja una gráfica tridimensional en la que el eje X es la parte real de la variable, es decir, Real(z); el eje Y es la parte imaginaria de la variable, es decir, Im(z) y el eje Z es la parte real de la imagen de la función, es decir, Re(f(z)).

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



Ejemplo : la gráfica de la función $f(z) = z^3$

```
>>z=cplxgrid(25);
>>cplxmap(z,z.^3)
```

Gráficas en movimiento

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

Estos fotogramas, una vez visualizados, producen la sensación de movimiento.

- Dibujar la gráfica de la curva y =λsin(x) para varios valores de λ contenidos en el intervalo [-1;1].

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

El núcleo del programa lo constituyen el conjunto de comandos

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

En programación se denomina un bucle, esto es, un conjunto de instrucciones, en este caso, comandos gráficos que se ejecutan varias veces, dependiendo del valor de j. A medida que j varía de 1 a 50, t varía, de 0 a 2π y, por tanto, λ = sin(t) varía entre -1 y 1. Para cada valor de j se realiza un g ráfico/fotograma que se almacena con la instrucción F(j) = getframe;. Por último, el comando movie(F,2) permite visualizar la película el número de veces que se le indique.

```
>>movie(F,2)
```

- 2D Función elipse en movimiento

```
function elipse
n=30; x=linspace(0,2*pi,200);
for j = 1:n
t=(pi/29)*(j-1);
plot(cos(x),sin(t)*sin(x),'rs'),
axis([-1 1 -1 1]);
F(j)=getframe;
end
movie(F,5)
```

- 2D colores en movimiento

```
function colores
n=30;
for j = 1:n
x=rand(10);
imagesc(x)
F(j) = getframe;
end
movie(F,5)
```

- 3D membrana en movimiento

```
function membrana
 [x,y]=meshgrid(-1:.1:1); n=20;
for j = 1:n
t=(2*pi/19)*(j-1);
z=2*sin(t)*exp(-x.^2-y.^2);
surf(x,y,z),axis([-1 1 -1 1 -2 2])
F(j) = getframe;
end
movie(F,6)
```

- Función 3D de picos en movimiento

```
function picos
[x,y,z]=peaks; n=20;
for j = 1:n
t=(2*pi/19)*(j-1);
z1=sin(t)*z;
surf(x,y,z1),axis([-3 3 -3 3 -5 5])
F(j) = getframe;
end
>> movie(F,3)
```

MATLAB

Primera edición digital

Noviembre, 2012

Lima - Perú

PROYECTO LIBRO DIGITAL

PLD 0618

Editor: Víctor López Guzmán

Guzlop editoras

http://www.guzlop-editoras.com/
guzlopster@gmail.com
guzlopnano@gmail.com
facebook.com/guzlop
twitter.com/guzlopster
731 2457 - 999 921 348
Lima - Perú