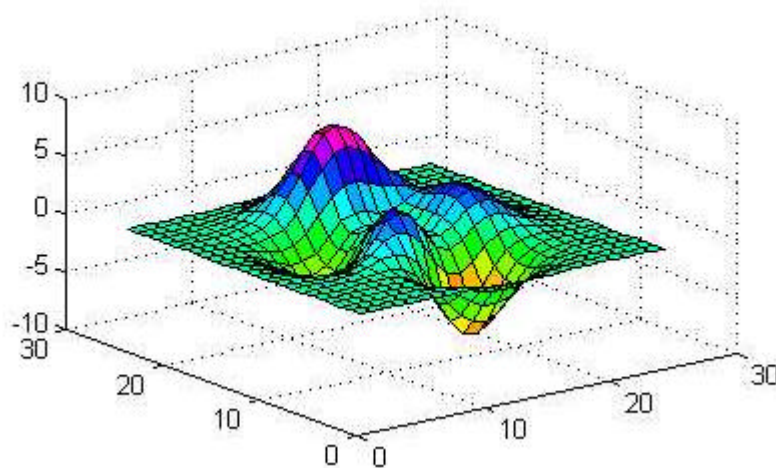


MANUAL BASICO

DE MATLAB



Apoyo a Investigación C.P.D.
Servicios Informáticos U.C.M.

1. INDICE :

1. INDICE	1
2. INTRODUCCION	3
3. CARACTERISTICAS BASICAS	5
3.1. Matemática sencilla	5
3.2. El espacio de trabajo de Matlab	5
3.3. Almacenar y recuperar datos	6
3.4. Formatos de visualización de números	6
3.5. Acerca de las variables	6
3.6. Otras características básicas	7
3.7. Ejemplos	7
4. CARACTERISTICAS CIENTIFICAS	8
4.1. Funciones matemáticas comunes	8
4.2. Números complejos	9
4.3. Ejemplos	9
5. AYUDA EN LINEA	10
5.1. La orden Help	10
5.2. La orden Lookfor	10
5.3. Ayuda conducida por menús	10
6. OPERACIONES CON ARRAYS	11
6.1. Arrays simples	11
6.2. Direccionamiento de arrays	11
6.3. Construcción de arrays	11
6.4. Matemáticas con arrays de escalares	12
6.5. Matemáticas con arrays de arrays	12
6.6. Orientación del array	12
6.7. Otras características	12
6.8. Ejemplos	13
7. GRAFICAS SIMPLES	14
8. ARCHIVOS SCRIPT	15
9. TEXTO	16
9.1. Ejemplos	16
10. OPERACIONES RELACIONALES Y LOGICAS	17
10.1. Operadores Relacionales	17
10.2. Operadores lógicos	17
10.3. Ejemplos	19
11. ALGEBRA LINEAL Y MATRICES	20
11.1. Características principales	20
11.2. Otras características	20
12. MANIPULACION MATRICIAL	21
12.1. Ejemplos	22
13. MATRICES ESPECIALES	25
13.1. Ejemplos	25
14. CONTROLES DE FLUJO	26
14.1. Ejemplos	27
15. FUNCIONES EN ARCHIVOS-M	28
16. ANALISIS DE DATOS	29

17. POLINOMIOS	30
17.1. Raíces	30
17.2. Otras características	30
17.3. Ejemplos	30
18. ANALISIS NUMERICO	32
18.1. Representación gráfica	32
18.2. Minimización	32
18.3. Localización de ceros	32
18.4. Integración	32
18.5. Diferenciación	33
18.6. Ecuaciones diferenciales	33
18.7. Ejemplos	33
19. GRAFICOS-2D	34
19.1. Utilización de la orden Plot	34
19.2. Estilo de líneas, marcadores y colores	34
19.3. Adición de rejillas y etiquetas	34
19.4. Ejes a medida	35
19.5. Impresión de figuras	35
19.6. Manipulación de datos	36
19.7. Otras características de los gráficos 2-D	37
19.8. Ejemplos	37
20. GRAFICOS 3-D	40
20.1. Gráficos de línea	40
20.2. Gráficos de malla y de superficie	40
20.3. Manipulación de gráficos	40
20.4. Otras características de los gráficos 3-D	40
20.5. Comprensión de los mapas de color	41
20.6. Utilización de mapas de color	41
20.7. Visualización de mapas de color	42
20.8. Creación y alteración de los mapas de color	42
20.9. Ejemplos	42

2. INTRODUCCION:

Matlab es al mismo tiempo un entorno y un lenguaje de programación. Uno de sus puntos fuertes es el hecho de que el lenguaje de Matlab permite construir nuestras propias herramientas reusables. Podemos fácilmente crear nuestras propias funciones y programas especiales (conocidos como archivos-M) en código Matlab. Los podemos agrupar en Toolbox: colección especializada de archivos-M para trabajar en clases particulares de problemas.

La manera más fácil de visualizar Matlab es pensar en él como en una calculadora totalmente equipada, aunque, en realidad, ofrece muchas más características y es mucho más versátil que cualquier calculadora. Matlab es una herramienta para hacer cálculos matemáticos. Es una plataforma de desarrollo de aplicaciones, donde conjuntos de herramientas inteligentes para la resolución de problemas en áreas de aplicación específica, a menudo llamadas toolboxes, se pueden desarrollar con facilidad relativa.

Se encuentra instalada en la máquina DEC-AXP_2100 Digital UNIX el programa MATLAB.

***** Entre sus utilidades, se encuentran: *****

- Cálculo matricial y Algebra lineal.
- Polinomios e interpolación.
- Regresión.
- Ajuste de funciones.
- Ecuaciones diferenciales ordinarias.
- Integración.
- Funciones.
- Gráficos bi y tridimensionales.

***** Además se encuentran disponibles los módulos (toolboxes) *****

- Optimización.
- Procesamiento de señales.
- Ecuaciones en derivadas parciales.
- Simulink: Simulación no lineal.

UTILIZACION DEL PROGRAMA MATLAB

Utilización en interactivo, dentro del entorno Matlab :

El comando de utilización es:

matlab

Es un entorno textual, con lo cual no es necesario habilitar una sesión o emulación X.

Sin embargo, para utilizar algunas ayudas y para realizar gráficos, se requiere una sesión o emulación X.

Para ello, debe prepararse el dispositivo gráfico de pantalla mediante el comando:

```
setenv DISPLAY dir-internet-terminal-o-Pc:0
```

y hay que habilitar una emulación X (p.e. Xwin), si no se trabaja desde una pantalla gráfica.

Utilización en interactivo, fuera del entorno de Matlab :

```
matlab <fichero_texto_matlab >fichero_salida_matlab
```

Utilización en BACH:

```
qsub -q cola_batch fichero_bat
```

donde:

cola_batch puede ser cualquier cola batch.

Ej: batch30min, batch4horas, batchduro...

fichero_bat es un fichero que contiene una línea:

```
matlab <fichero_texto_matlab >fichero_salida_matlab
```

En salida, además de *fichero_salida_matlab* se obtienen los ficheros:

```
fichero_bat.o_n_job  
fichero_bat.e_n_job
```

Nota Importante: Si los ficheros a los que se hace referencia en *fichero_bat* no se encuentran en el directorio de login del usuario, debe especificarse el camino de los mismos. (Se puede hacer uso de la variable de entorno **\$HOME**)

Para mayor información: AIDE

3. CARACTERISTICAS BASICAS :

3.1. MATEMATICA SENCILLA :

Matlab no tiene en cuenta los espacios.

El punto y coma al final de la línea le dice a Matlab que evalúe la línea, pero que no nos diga la respuesta.

Si la sentencia es demasiado larga para que quepa en una línea, una elipsis consistente en tres puntos (...) seguido por **Enter** indica que la sentencia continúa en la línea siguiente.

Matlab ofrece las siguientes operaciones básicas:

OPERACION	SIMBOLO
Suma, a+b	+
Resta, a-b	-
Multiplicación, a*b	*
División, a/b	/ o \
Potencia, a^b	^

Su precedencia es como sigue:

$$^ > / , * > + , -$$

3.2. EL ESPACIO DE TRABAJO DE MATLAB:

Para comprobar el valor de una variable, hay que preguntar a Matlab por ello introduciendo su nombre a continuación del indicativo de petición de orden.

Para obtener una lista de las variables usamos la orden: **who**.

Para recordar órdenes previas, usamos las teclas de cursor del teclado.

3.3. ALMACENAR Y RECUPERAR DATOS :

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

3.4. FORMATOS DE VISUALIZACION DE NUMEROS :

Matlab no cambia la representación interna de un número cuando se escogen distintos formatos; sólo se modifica la visualización del número.

A continuación se muestra la tabla con los formatos numéricos de Matlab:

ORDEN DE MATLAB	COMENTARIOS
Format long	16 dígitos
Format short e	5 dígitos más exponente
Format long e	16 dígitos más exponente
Format hex	Hexadecimal
Format bank	2 dígitos decimales
Format +	Positivo, negativo o cero
Format rat	Aproximación racional
Format short	Visualización por defecto

3.5. ACERCA DE LAS VARIABLES :

Por defecto, Matlab almacena resultados en la variable **ans**.

Las variables son sensibles a las mayúsculas y pueden contener hasta 19 caracteres. Deben comenzar con una letra.

Matlab tiene algunas variables especiales:

VARIABLE	VALOR
ans	Nombre por defecto de la variable usada para los resultados
pi	Razón de una circunferencia a su diámetro
eps	Número más pequeño tal que, cuando se le suma 1, crea un número en coma flotante en el computador mayor que 1
inf	Infinito
NaN	Magnitud no numérica
i y j	$i = j = \sqrt{-1}$
realmin	El número real positivo más pequeño que es utilizable
realmax	El número real positivo más grande que es utilizable

Cuando Matlab realiza un cálculo, lo hace utilizando los valores que conoce del momento en que se evaluó la orden pedida.

Mediante la orden **clear** podemos borrar las variables en el espacio de trabajo.

3.6. *OTRAS CARACTERISTICAS BASICAS :*

Los comentarios se escriben después del signo de tanto por ciento (%).

Podemos colocar órdenes múltiples en una línea si se separan por comas o puntos y comas. Las comas le dicen a Matlab que visualice los resultados; los puntos y comas suprimen la impresión.

Para interrumpir Matlab en cualquier momento: **Ctrl-C**.

Escribiendo la orden **quit** termina Matlab.

3.7. *EJEMPLOS :*

```
>> manzanas=4           % Número de manzanas.
manzanas =
     4
>> platanos=6, melones=2; % Número de plátanos y de melones.
platanos =
     6
>> fruta=manzanas+platanos+melones % Almacena el resultado en la variable fruta
fruta =
    12
>> coste=manzanas*25+platanos*22+melones*99
coste =
    430
>> coste_medio=coste/fruta
coste_medio =
    35.8333
>> who % Da una lista de los nombres de las variables de nuestro programa
```

Your variables are:

```
manzanas  platanos  melones  fruta
coste     coste_medio
>> clear manzanas % Borra la variable manzanas.
```


4. CARACTERISTICAS CIENTIFICAS :

4.1. FUNCIONES MATEMATICAS COMUNES :

A continuación se muestra una tabla con las funciones matemáticas en Matlab:

FUNCIONES MATEMATICAS ESPECIALES	
abs (x)	Valor absoluto o magnitud de un número complejo
acos (x)	Inversa del coseno
acosh (x)	Inversa del coseno hiperbólico
angle (x)	Angulo de un número complejo
asin (x)	Inversa del seno
asinh (x)	Inversa del seno hiperbólico
atan (x)	Inversa de la tangente
atan2 (x,y)	Inversa de la tangente en los cuatro cuadrantes
atanh (x)	Inversa de la tangente hiperbólica
ceil (x)	Redondea hacia más infinito
conj (x)	Complejo conjugado
cos (x)	Coseno
cosh (x)	Coseno hiperbólico
exp (x)	Exponencial
fix (x)	Redondea hacia cero
floor (x)	Redondea hacia menos infinito
imag (x)	Parte imaginaria de un número complejo
log (x)	Logaritmo natural
log10 (x)	Logaritmo decimal
real (x)	Parte real de un número complejo
rem (x,y)	Resto después de la división
round (x)	Redondea hacia el entero más próximo
sign (x)	Devuelve el signo del argumento
sin (x)	Seno
sinh (x)	Seno hiperbólico
sqrt (x)	Raíz cuadrada
tan (x)	Tangente
tanh (x)	Tangente hiperbólica

Matlab sólo opera en radianes.

4.2 NUMEROS COMPLEJOS :

Matlab sigue el convenio usual, donde un número complejo se escribe como $a+bi$. La terminación con los dos caracteres **i** y **j** sólo funciona con números simples, no con expresiones.

Las operaciones matemáticas sobre números complejos se escriben de la misma forma que con números reales.

Las funciones **real**, **imag**, **abs** y **angle** son útiles para la conversión entre las formas polar y rectangular.

4.3. EJEMPLOS :

Ejemplo 1:

```
>>a=1; b=4; c=13;
>>x1=(-b+sqrt(b^2-4*a*c))/(2*a)
x1 =
    -2.0000 + 3.0000i
>>x2=(-b-sqrt(b^2-4*a*c))/(2*a)
x2 =
    -2.0000 - 3.0000i
>> a*x1^2+b*x1+c      % Sustituimos x1 para comprobar la respuesta.
ans =
     0
>> a*x2^2+b*x2+c      % Sustituimos x2 para comprobar la respuesta.
ans =
     0
```

Ejemplo 2:

```
>> c1=1-2i           % Con j en lugar de i también funciona.
c1 =
    1.0000 - 2.0000i
>>c2=3*(2-sqrt(-1)*3); c3=sqrt(-2); c4=6+sin(.5)*j
c4 =
    6.0000 + 0.4794i
>>c5=(c1+c2)/c3
c5 =
   -7.7782 - 4.9497i
```

5. AYUDA EN LINEA :

Matlab proporciona asistencia a través de sus capacidades de **ayuda en línea**. Estas capacidades están disponibles en tres formas:

5.1. LA ORDEN *HELP* :

Escribiendo **help<tema>** visualiza la ayuda acerca de ese tema, si existe.

También proporciona asistencia escribiendo simplemente **help**.

5.2. LA ORDEN *LOOKFOR* :

Proporciona ayuda buscando a través de todas las primeras líneas de las ayudas a temas de Matlab y devolviendo aquellos que contienen una palabra clave que hay que especificar. Lo más importante es que la palabra clave no necesita ser una orden de Matlab.

5.3. AYUDA CONDUCTA POR MENUS :

Esta ayuda está disponible seleccionando **Table of Contents...** o **Index...** del menú **Help**.

6. OPERACIONES CON ARRAYS :

6.1. ARRAYS SIMPLES :

Para crear un array en Matlab comenzamos con un corchete de apertura, introducimos los valores deseados separados por espacios (o por comas) y cerramos el array con un corchete de cierre.

Variable=[(lista de números separados por espacios o comas)]

6.2. DIRECCIONAMIENTO DE ARRAYS :

Los elementos individuales de un array se acceden utilizando subíndices; así, $x(1)$ es el primer elemento en x .

Para acceder a un bloque de elementos a la vez, se usa la notación de dos puntos; así, $x(1:5)$ nos da los elementos del primero al quinto del array de elementos. Si introducimos un número entre el primero y el segundo, también separado por dos puntos (:), entonces se mostrarán los elementos del primero al último indicado, incrementados o decrementados el número que aparece en el centro; así, si ponemos $x(2:2:7)$, obtenemos el segundo, cuarto y sexto elemento del array.

Otra forma de obtener un conjunto concreto de elementos del array es indicando entre corchetes las posiciones de los elementos que queremos obtener; ponemos paréntesis fuera de los corchetes. Ejemplo: $y([8\ 2\ 9\ 1])$.

6.3. CONSTRUCCION DE ARRAYS :

Otras dos formas de introducir arrays son:

a.- Mediante la notación dos puntos, $(0:0.1:1)$ crea un array que comienza en cero, incrementa 0.1 y finaliza en 1.

b.- Mediante la función **linspace** :

$\text{linspace}(\text{primer_exponente}, \text{último_exponente}, \text{número_de_valores})$

Las dos formas anteriores crean arrays donde los elementos individuales están espaciados linealmente entre sí. Para espaciado logarítmico:

$\text{logspace}(\text{primer_exponente}, \text{último_exponente}, \text{número_de_valores})$

6.4. MATEMATICAS CON ARRAYS DE ESCALARES :

Las operaciones matemáticas sencillas entre escalares y arrays siguen una interpretación natural, es decir, se aplica la operación a todos los elementos del array.

6.5. MATEMATICAS CON ARRAYS DE ARRAYS :

Cuando dos arrays tienen la misma longitud y orientación, la suma, resta, multiplicación y división se aplican sobre la base de elemento-a-elemento.

Para multiplicar dos arrays elemento a elemento, escribimos `.*`, ya que si ponemos sólo `*`, sería multiplicación matricial. Lo mismo para la división de arrays y la potencia de un array.

Se pueden combinar operaciones escalares y de arrays.

6.6. ORIENTACION DEL ARRAY :

Separar los elementos por espacios o comas especifica elementos en distintas columnas (*vector fila*); separar elementos por puntos y comas especifica elementos en filas diferentes (*vector columna*).

Usando el operador *transpuesta* (`'`) de Matlab, podemos pasar de vector fila a vector columna, y viceversa.

En el caso de un array complejo, la *transpuesta* (`'`) da la transpuesta compleja conjugada. La *transpuesta con punto* (`.'`) transpone el array, pero no lo conjuga.

La creación de *matrices* (orientación *rectangular*) sigue la misma estructura de los vectores fila y columna.

Además de los puntos y comas, pulsando la tecla **Return** cuando se está introduciendo una matriz, también le dice a Matlab que comience una nueva fila.

Una matriz puede tener múltiples filas, pero cada fila debe tener un número igual de columnas.

6.7. OTRAS CARACTERISTICAS :

La orden **whos** proporciona información adicional sobre los arrays.

6.8. EJEMPLOS :

Ejemplo 1:

```
>> x=[0 .1*pi .2*pi .3*pi .4*pi .5*pi .6*pi .7*pi .8*pi .9*pi pi]
x =
Columns 1 through 7
    0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
Columns 8 through 11
    2.1991    2.5133    2.8274    3.1416
>> y=sin(x)
y =
Columns 1 through 7
    0    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511
Columns 8 through 11
    0.8090    0.5878    0.3090    0.0000
>> x(3)           % El tercer elemento de x.
ans =
    0.6283
>> x(1:5)         % Para obtener los elementos del primero al quinto en x.
ans =
    0    0.3142    0.6283    0.9425    1.2566
>> y(3:-1:1)      % Comienza con 3, disminuye en una unidad, y para en 1.
ans =
    0.5878    0.3090    0
>> y([8 2 9 11]) % Obtenemos los elementos 8°, 2°, 9° y 11° del array y.
ans =
    0.8090    0.3090    0.5878    0
```

Ejemplo 2 :

```
>> a=1:5,b=1:2:9
a =
     1     2     3     4     5
b =
     1     3     5     7     9
>> c=[b a]        % Crea un array de los elementos de b seguidos de los elementos de a
c =
     1     3     5     7     9     1     2     3     4     5
>> a-2            % Matemáticas con arrays de escalares.
ans =
    -1     0     1     2     3
>> a.*b           % Matemáticas con arrays de arrays.
ans =
     1     6    15    28    45
```

7. GRAFICAS SIMPLES :

Primero se crean los valores para el eje horizontal X (variable independiente); a continuación se calcula el eje vertical Y (variable dependiente); la orden **plot** genera la gráfica:

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

Opciones de la función **plot**:

- Superponer gráficas sobre los mismos ejes:

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

- Usar distintos tipos de líneas para el dibujo de la gráfica:

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

Además se pueden colocar etiquetas sobre los ejes:

- Etiqueta sobre el eje X de la gráfica actual:

```
>>xlabel('texto')
```

- Etiqueta sobre el eje Y de la gráfica actual:

```
>>ylabel('texto')
```

Un título en la cabecera de la gráfica actual:

```
>>title('texto')
```

Dibujar una rejilla:

```
>>grid
```

etc.

8. ARCHIVOS SCRIPT :

Matlab permite colocar órdenes en un simple archivo de texto y, a continuación, decirle a Matlab que lo abra y evalúe las órdenes exactamente como si hubiesen sido escritas desde la línea de orden de Matlab. Estos archivos se llaman archivos **scrip** o **archivos-M**, y deben finalizar 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 línea de orden de Matlab.

Las órdenes dentro del archivo-M tienen acceso a todas las variables en el espacio de trabajo de Matlab, y todas las variables creadas en el archivo-M se hacen parte del espacio de trabajo.

Normalmente, las órdenes leídas desde el archivo-M no se visualizan cuando se evalúan. La orden **echo on** le dice a Matlab que visualice o efectúe un eco de las órdenes en la ventana de **Orden** cuando se leen y evalúan. También existe la función **echo off**.

Ordenes de gestión de archivos:

ORDEN	DESCRIPCION
what	Devuelve un listado de todos los archivos-M en el directorio actual.
dir	Lista todos los archivos en el directorio o carpeta actual.
ls	Lo mismo que dir.
type test	Visualiza el archivo-M test.m en la ventana de orden.
delete test	Suprime 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.

9. TEXTO :

Una **cadena de caracteres** es texto rodeado por comillas simples ('). Se manejan como vectores filas.

Las cadenas se direccionan y manipulan igual que los arrays.

Son posibles las operaciones matemáticas sobre cadenas. Una vez hecha una operación matemática sobre una cadena, ésta se ve como un array de números en ASCII.

Para ver la representación ASCII de una cadena, tomamos su valor absoluto o sumamos cero.

Para restaurarla y verla de nuevo como cadena de caracteres, usamos la función **setstr(var)**.

Cambiamos a caracteres en minúsculas añadiendo la diferencia entre 'a' y 'A'.

9.1. EJEMPLOS :

Ejemplo 1:

```
>> t='Esto es una cadena de caracteres'
t =
    Esto es una cadena de caracteres
>> u=t(13:18)           % Los elementos que van del 13 al 18 son la palabra cadena.
u =
    cadena
>> u=t(18:-1:13)        % Palabra cadena deletreada de forma inversa.
u =
    anedac
```

Ejemplo 2:

```
>> s='ABCDEFGFG'
s =
    ABCDEFG
>> m=abs(s)
m =
    65    66    67    68    69    70    71
>> setstr(m)
ans =
    ABCDEFG
>> n=s+5; setstr(n)
ans =
    FGHIJKL
```

10. OPERACIONES RELACIONALES Y LOGICAS :

Como entradas a todas las expresiones relacionales y lógicas, Matlab considera que cualquier número distinto de cero es verdadero, y es falso si es igual a cero.

La salida produce 1 si es verdadero, y 0 si es falso.

10.1. OPERADORES RELACIONALES :

OPERADOR	DESCRIPCION
<	Menor que
<=	Menor que o igual a
>	Mayor que
>=	Mayor que o igual a
==	Igual a
~=	No igual a

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

10.2. OPERADORES LOGICOS :

Los operadores lógicos proporcionan un medio de combinar o negar expresiones relacionales.

OPERADOR	DESCRIPCION
&	AND
	OR
~	NOT

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

FUNCION	DESCRIPCION
xor(x,y)	Operación OR exclusiva.Devuelve unos donde o x o y es distinto de cero (verdadero). Devuelve ceros donde ambos x e y son ceros (falso) o ambos son distinto de cero (verdadero).
any(x)	Devuelve uno si algún elemento en un vector x es no nulo. Devuelve uno para cada columna en una matriz x que tiene elementos no nulos.
all(x)	Devuelve uno si todos los elementos en un vector x son no nulos. Devuelve uno para cada columna en una matriz x que tiene todos los elementos no nulos.
isnan(x)	Devuelve unos en magnitudes no numéricas (NaN) en x.
isinf(x)	Devuelve unos en magnitudes infinitas (inf) en x.
finite(x)	Devuelve unos en valores finitos en x.

La siguiente carta muestra el orden de precedencia para operadores aritméticos, lógicos y relacionales, con la fila superior teniendo máxima precedencia.

TABLA DE PRECEDENCIA DE OPERADORES						
^	.^	'	!			
*	/	\	.*	./	.\	
+	-	~	+(unario)	-(unario)		
:	>	<	>=	<=	==	~=
	&					

10.3. EJEMPLOS :

Ejemplo 1:

```
>> A=1:9,B=9-A
A =
    1    2    3    4    5    6    7    8    9
B =
    8    7    6    5    4    3    2    1    0
>> tf=A>4 % Encuentra elementos de A que son mayores que 4.
tf =
    0    0    0    0    1    1    1    1    1
>> tf=A==B % Encuentra elementos de A que son iguales a aquellos en B.
tf =
    0    0    0    0    0    0    0    0    0
>> tf=B-(A>2) % Encuentra dónde A>2 y resta el resultado de B.
tf =
    8    7    5    4    3    2    1    0    -1
```

Ejemplo 2:

```
>> A=1:9;B=9-A;
>> tf=A>4 % Encuentra dónde A es mayor que 4.
tf =
    0    0    0    0    1    1    1    1    1
>> tf=~(A>4) % Niega el resultado anterior.
tf =
    1    1    1    1    0    0    0    0    0
>> tf=(A>2)&(A<6) % Devuelve unos donde A es mayor que 2 y menor que 6.
tf =
    0    0    1    1    1    0    0    0    0
```

Ejemplo 3:

```
>> x=linspace(0,10,100); % Crear datos.
>> y=sin(x); % Calcular seno.
>> z=(y>=0).*y; % Fija a cero los valores negativos de sin(x).
>> z=z+0.5*(y<0); % Si sin(x) es negativo, sumar 1/2.
>> z=(x<=8).*z; % Fijar a cero los valores mayores que x=8.
>> plot(x,z) % Dibuja la gráfica de ejes x y z.
>> xlabel('x'), ylabel('z=f(x)'), % Etiquetas de ambos ejes.
>> title('Una señal discontinua') % Etiqueta de la gráfica.
```

11. ALGEBRA LINEAL Y MATRICES :

11.1. CARACTERISTICAS PRINCIPALES :

En Matlab, la multiplicación matricial se denota con la notación asterisco $*$.
La función **inv(A)** calcula la inversa de la matriz **A**.

En Matlab, cuando hay más ecuaciones que incógnitas (caso sobredeterminado), la utilización del operador de división \backslash o $/$ automáticamente encuentra la solución que minimiza el error al cuadrado en $Ax=b=0$. Esta solución se llama *solución de mínimos cuadrados*.

Cuando hay menos ecuaciones que incógnitas (caso indeterminado), existe un número infinito de soluciones. Matlab calcula dos de forma directa. El uso del operador de división da una solución que tiene ceros para algunos de los elementos de x . Alternativamente, calculando $x=pinv(A)*b$ se obtiene una solución donde la longitud o norma euclídea de x es más pequeña que todas las otras posibles soluciones. Esta solución se llama *solución de norma mínima*.

11.2. OTRAS CARACTERISTICAS :

- **A.'** es la transpuesta de la matriz **A**. La transpuesta compleja conjugada de la matriz **A** se escribe como **A'**.
- **d=eig(A)** devuelve los valores propios asociados con la matriz cuadrada **A** como un vector columna.
- **[V,D]=eig(A)** devuelve los vectores propios en la matriz **V** y los valores propios como los elementos diagonales en la matriz **D**.
- **[L,U]=lu(A)** calcula la factorización **LU** de la matriz cuadrada **A**.
- **[Q,R]=qr(A)** calcula la factorización **QR** de la matriz **A**.
- **[U,S,V]=svd(A)** calcula la descomposición en valores singulares de la matriz **A**.
- **rank(A)** devuelve el rango de la matriz **A**.
- **cond(A)** devuelve el número de condición de la matriz **A**.
- **norm(A)** calcula la norma de la matriz **A**. Admite el cálculo de norma-1, norma-2, norma-F y norma- ∞ .
- **poly(A)** encuentra el polinomio característico asociado con la matriz cuadrada **A**.
- **polyvalm(v,A)** evalúa el polinomio característico **v** usando la matriz cuadrada **A**.

12. MANIPULACION MATRICIAL :

Los elementos matriciales se direccionan en el formato fila, columna : $A(filas,columnas)$.

Valores internos a una matriz se acceden identificando los subíndices de los elementos deseados.

Utilizar el símbolo *dos puntos* como la designación de filas o columnas implica, respectivamente, todas las filas o columnas; por ejemplo, $A(:,1)$ representa todas las filas en la columna uno.

Fijar las filas o columnas de una matriz igual a la matriz vacía $[]$ elimina estas filas o columnas.

Usar sólo los *dos puntos*, por ejemplo, $A(:)$, reagrupa una matriz en un vector columna, tomando todas las columnas a un tiempo.

Vectores lógicos 0-1 pueden utilizarse también para direccionar partes de un vector. En este caso, los vectores lógicos 0-1 deben tener el mismo tamaño que el vector que direcciona. Los elementos falsos (0) se eliminan, los elementos verdaderos (1) se retienen.

La función **find** devuelve los subíndices o índices donde una expresión relacional es verdadera.

La función **size** devuelve el número de filas y de columnas de una matriz. La función **length** devuelve la longitud de un vector o la máxima dimensión de una matriz.

Otras características sobre la manipulación matricial son:

- **flipud(A)** intercambia una matriz de arriba abajo.
- **fliplr(A)** intercambia una matriz de izquierda a derecha.
- **rot90(A)** gira una matriz en dirección contraria a las agujas del reloj.
- **reshape(A,m,n)** devuelve una matriz $m \times n$ cuyos elementos se toman por columnas de **A**. **A** debe contener $m \times n$ elementos.
- **diag(v)** crea una matriz diagonal, con el vector **v** sobre la diagonal.
- **diag(A)** extrae la diagonal de la matriz **A** como un vector columna.

12.1. EJEMPLOS :

Ejemplo 1:

```
>> A=[1 2 3;4 5 6;7 8 9]           % Introducimos la matriz A.
A =
     1     2     3
     4     5     6
     7     8     9
>> A(3,3)=0           % Cambia a cero el elemento de la tercera fila y tercera columna.
A =
     1     2     3
     4     5     6
     7     8     0
>> A(2,6)=1           % Coloca 1 en la segunda fila, sexta columna.
A =
     1     2     3     0     0     0
     4     5     6     0     0     1
     7     8     0     0     0     0
>> B=A(3:-1:1,1:3)    % crea una matriz B tomando las filas de A en orden inverso.
B =
     7     8     9
     4     5     6
     1     2     3
>> B=A(3:-1:1,:);     % Hace lo mismo que el ejemplo anterior.
>> C=[A B(:,[1 3])]    % Crea C añadiendo todas las filas en la primera y tercera
columna de B a la derecha de A.
C =
     1     2     3     7     9
     4     5     6     4     6
     7     8     9     1     3
>> B=A(1:2,2:3)       % Crea B extrayendo las primeras dos filas y las últimas dos
columnas de A.
B =
     2     3
     5     6
>> C=[1 3]
C =
     1     3
>> B=A(C,C)           % Usa el array C para indexar la matriz A.
B =
     1     3
     7     9
>> B=A(:);            % Construye B al disponer A en un vector columna tomando todas
sus columnas a un tiempo.
>> B=B.'              % Operación punto-transpuesta.
B =
     1     4     7     2     5     8     3     6     9
```

```

>> B=A; B(:,2)=[]           % Redefine B eliminando todas las filas en la segunda
columna de la original B.
B =
     1     3
     4     6
     7     9
>> B=B.'; B(2,:)=[]         % Elimina la segunda fila de B.
B =
     1     4     7
>> A(2,:)=B                  % Sustituye la segunda fila de A con B.
A =
     1     2     3
     1     4     7
     7     8     9
>> B=A(:, [2 2 2 2])        % Crea B duplicando todas las filas en la segunda columna de
A cuatro veces.
B =
     2     2     2     2
     4     4     4     4
     8     8     8     8

```

Ejemplo 2:

```

>> x=-3:3                    % Introducimos datos.
x =
    -3    -2    -1     0     1     2     3
>> abs(x)>1                  % Da unos donde el valor absoluto de x es mayor que 1.
ans =
     1     1     0     0     0     1     1
>> y=x(abs(x)>1)            % Crea y al tomar aquellos valores de x donde su valor absoluto
es mayor que 1.
y =
    -3    -2     2     3
>> y=x([1 1 1 1 0 0 0])    % Crea y seleccionando sólo los primeros 4 valores, y
descartando los otros.
y =
    -3    -2    -1     0
>> y=x([1 1 1 1])          % Crea y tomando el primer elemento de x cuatro veces.
y =
    -3    -3    -3    -3
>> x(abs(x)>1)=[]           % Elimina valores de x donde abs(x)>1.
x =
    -1     0     1

```


Ejemplo 3:

```
>> b=[5 -3;2 -4]
b =
     5     -3
     2     -4
>> x=abs(b)>2    % La extracción de arrays lógicos 0-1 también funciona con matrices
x =
     1     1
     0     1
>> y=b(abs(b)>2)    % Los resultados se convierten a un vector columna.
y =
     5
    -3
     4
```

Ejemplo 4:

```
>> x=-3:3
x =
    -3    -2    -1     0     1     2     3
>> k=find(abs(x)>1)    % Encuentra aquellos subíndices donde abs(x)>1.
k =
     1     2     6     7
>> y=x(k)    % Crea y utilizando los índices en k.
y =
    -3    -2     2     3
```

Ejemplo 5:

```
>> A=[1 2 3 4; 5 6 7 8]; B=pi:0.01:2*pi;
>> s=size(A)    % devuelve un vector fila cuyo primer elemento es el número de
filas y cuyo segundo elemento es el número de columnas.
s =
     2     4
>> [r,c]=size(A)    % Devuelve el número de filas en la primera variable y el número
de columnas en la segunda variable.
r =
     2
c =
     4
>> length(A)    % Devuelve el número de filas o de columnas, cualquiera que sea mayor.
ans =
     4
>> size (B)    % Muestra que B es un vector fila.
ans =
     1    315
>> length(B)    % Devuelve la longitud del vector (315).
```

13. MATRICES ESPECIALES :

- **zeros(n)** Matriz de ceros ($n \times n$).
- **ones(n,m)** Matriz de unos ($n \times m$).
- **rand(n,m)** Matriz ($n \times m$) de números aleatorios distribuidos uniformemente entre cero y uno.
- **randn(n,m)** Matriz ($n \times m$) de números aleatorios distribuidos normalmente con media cero y varianza unidad.
- **eye(n,m)** Matriz identidad ($n \times m$).

13.1 EJEMPLOS :

```
>> zeros(3)                % Una matriz 3×3 de ceros.
ans =
    0    0    0
    0    0    0
    0    0    0
>> ones(2,4)               % Una matriz 2×4 de unos.
ans =
    1    1    1    1
    1    1    1    1
>> rand(3,1)
ans =
    0.2190
    0.0470
    0.6789
>> randn(2)
ans =
    1.1650    0.0751
    0.6268    0.3516
>> eye(3)
ans =
    1    0    0
    0    1    0
    0    0    1
>> A=[1 2 3;4 5 6];
>> ones(size(A))           % Una matriz de unos del mismo tamaño que A.
ans =
    1    1    1
    1    1    1
```

14. CONTROLES DE FLUJO :

Las características de control de flujo de Matlab se resumen en la siguiente tabla:

ESTRUCTURA DE CONTROL DE FLUJO	DESCRIPCION
for x = array órdenes end	Un bloque <i>for</i> que en cada iteración asigna <i>x</i> a la columna <i>i</i> -ésima de <i>array</i> y ejecuta <i>órdenes</i> .
while expresión órdenes end	Un bloque <i>while</i> que ejecuta <i>órdenes</i> mientras todos los elementos de <i>expresión</i> son verdaderas o distinto de cero.
if expresión órdenes end	Una simple estructura <i>if-else-end</i> donde <i>órdenes</i> se ejecutan si todos los elementos en <i>expresión</i> son verdaderas.
if expresión órdenes evaluadas si exp \equiv verdadero else órdenes evaluadas si exp \equiv falso end	Una estructura <i>if-else-end</i> con dos caminos. Un grupo de órdenes se ejecuta si <i>expresión</i> es verdadera. El otro conjunto se ejecuta si <i>expresión</i> es falsa o cero.
if expresión1 órdenes evaluadas si expresión1 es verdadera elseif expresión2 órdenes evaluadas si expresión2 es verdadera elseif _ . . . else órdenes evaluadas si ninguna otra expresión es verdadera end	La estructura más general <i>if-else-end</i> . Sólo se evalúan las órdenes asociadas con la primera expresión verdadera.
break	Termina la ejecución de bucles <i>for</i> y bucles <i>while</i> .

14.1. EJEMPLOS :

Ejemplo 1 : (bucle for)

```
>> for n=1:5
    for m=5:-1:1
        A(n,m)=n^2+m^2;
    end
    disp(n)
end
1
2
3
4
5
>> A
A =
     2     5    10    17    26
     5     8    13    20    29
    10    13    18    25    34
    17    20    25    32    41
    26    29    34    41    50
```

Ejemplo 2 : (bucle while)

```
>> num=0;EPS=1;          % Forma de calcular el valor especial eps de Matlab.
>> while (1+EPS)>1
    EPS=EPS/2;
    num=num+1;
end
>> num
num =
    53
>> EPS=2*EPS
EPS =
 2.2204e-16
```

Ejemplo 3 : (estructura if-else-end)

```
>> manzanas=10; coste=manzanas*25    % Número y coste de manzanas.
coste =
    250
>> if manzanas>5    % Aplicar 20% de descuento por cantidad.
    coste=(1-20/100)*coste;
end
>> coste
coste =
    200
```

15. FUNCIONES EN ARCHIVOS - M :

Un *archivo-M* de función es similar a un archivo *script*.

El nombre de la función y el nombre del archivo deben ser idénticos.

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

Cuando una función tiene más de una variable de salida, éstas se encierran entre corchetes.

El número de variables de entrada pasadas a una función está disponible dentro de la función en la variable **nargin**. El número de variables de salida solicitadas cuando una función se llama, está disponible dentro de la función en la variable **nargout**.

16. ANALISIS DE DATOS :

Matlab ejecuta análisis estadístico sobre conjuntos de datos.

Los conjuntos de datos se almacenan en matrices orientadas por columnas.

Matlab incluye las siguientes funciones estadísticas:

FUNCION ESTADISTICA	DESCRIPCION
corrcoef(x)	Coeficientes de correlación.
cov(x)	Matriz de covarianza.
cumprod(x)	Producto acumulativo de columnas.
cumsum(x)	Suma acumulativa de columnas.
diff(x)	Calcula las diferencias entre elementos.
hist(x)	Histograma o diagrama de barras.
mean(x)	Valor medio de columnas.
median(x)	Valor de la mediana de columnas.
prod(x)	Producto de elementos en columnas.
rand(x)	Números aleatorios distribuidos uniformemente.
randn(x)	Números aleatorios distribuidos normalmente.
sort(x)	Ordenar columnas en orden ascendente.
std(x)	Desviación estándar de columnas.
sum(x)	Suma de elementos en cada columna.

17. POLINOMIOS :

17.1. RAICES :

Un polinomio se representa por un vector fila con sus coeficientes en orden descendente; se deben incluir los términos con coeficientes nulos.

Las raíces de un polinomio se encuentran utilizando la función **roots(p)**.

Matlab adopta el convenio de que los polinomios son vectores fila y las raíces son vectores columna.

Dadas las raíces de un polinomio, es posible construir los polinomios asociados mediante la función **poly(r)**.

17.2. OTRAS CARACTERISTICAS:

Matlab ofrece muchas capacidades para la manipulación de polinomios:

- **conv(a,b)** multiplica los dos polinomios **a** y **b**.
- **deconv(c,b)** divide el polinomio **b** entre **c**.
- **polyder(p)** calcula la derivada del polinomio **p**.
- **polyval(p,x)** evalúa el polinomio **p** en todos los valores de **x**.
- **residue(n,d)** calcula el desarrollo en fracciones simples del cociente de **n** a **d**, donde **n** y **d** son polinomios.
- **polyder(n,d)** calcula la derivada del cociente de **n** a **d**, donde **n** y **d** son polinomios.

Matlab no tiene incorporada una función para sumar polinomios. Sin embargo, es fácil construir un archivo-M de función que lo haga.

17.3. EJEMPLOS :

Ejemplo 1 :

```
>> p=[1 -12 0 25 116]      % Incluimos términos con coeficientes nulos.  
p =  
    1   -12    0   25  116
```

```

>> r=roots(p)                % raíces del polinomio p.
r =
    11.7473
     2.7028
   -1.2251 + 1.4672i
   -1.2251 + 1.4672i
>> pp=poly(r)                % Polinomios asociados.
pp =
    1.0e+02 *
Columns 1 through 4
    0.0100    -0.1200    -0.0000     0.2500
Columns 5
    1.1600 + 0.0000i
>> pp=real(pp)               % Extrae la parte real.
pp =
    1.0000    -12.0000    -0.0000    25.0000   116.0000

```

Ejemplo 2 :

```

>> a=[1 2 3 4]; b=[1 4 9 16];
>> c=conv(a,b)               % Multiplicación.
c =
     1     6    20    50    75    84    64
>> d=a+b                     % Suma.
d =
     2     6    12    20
>> e=c+[0 0 0 d]             % Porque son de distinto grado.
e =
     1     6    20    52    81    96    84
>> f=c+[0 0 0 -d]           % Resta.
f =
     1     6    20    48    69    72    44
>> [q,r]=deconv(c,b)         % División.
q =
     1     2     3     4
r =
     0     0     0     0     0     0     0
>> g=polyder(f)              % Derivada.
g =
     6    30    80    144    138    72

```

Ejemplo 3 :

```

>> x=linspace(-1,3);         % Se generan 100 puntos de datos entre -1 y 3.
>> p=[1 4 -7 -10];           % Definimos el polinomio p.
>> v= polyval(p,x);          % Evaluamos p(x) en los valores de x y lo almacenamos en v.
>> plot(x,v), title('x^3 + 4x^2 - 7x - 10'), xlabel('x')

```


18. ANALISIS NUMÉRICO :

18.1. REPRESENTACION GRAFICA :

Existe una función que evalúa cuidadosamente la función que se va a representar, y asegura que todas sus peculiaridades se representan en la gráfica de salida. Como entrada, esta función necesita conocer el nombre de la función como una cadena de caracteres y el rango de representación como un array de dos elementos:

fplot('nombre',[a,b])

18.2. MINIMIZACION :

Para encontrar mínimos de funciones unidimensional y n-dimensional usamos, respectivamente, las funciones:

fmin('nombre_función',a,b)

fmins('nombre_función',a,b)

18.3. LOCALIZACION DE CEROS :

Para buscar el cero de una función unidimensional usamos:

fzero('nombre_función',a)

donde **a** es el punto cerca del cual se busca el cero.

A la función **fzero** debe darse el nombre de una función cuando se la llama.

También puede utilizarse para encontrar donde una función es igual a cualquier constante.

18.4. INTEGRACION :

Matlab proporciona tres funciones para calcular numéricamente el área bajo una función sobre un rango finito: **trapz**, **quad** y **quad8**.

La función **trapz(x,y)** aproxima la integral bajo una función al sumar el área de los trapecoides formados con los puntos.

Las funciones **quad** y **quad8** realizan aproximaciones de un orden más elevado que un simple trapecoide. Funcionan igual que **fzero**.

18.5. DIFERENCIACION :

Matlab proporciona una función para calcular una aproximación de la derivada, muy poco precisa, dados los datos que describen alguna función. Esta función, llamada **diff**, calcula la diferencia entre los elementos de un array.

18.6. ECUACIONES DIFERENCIALES :

Cuando las ecuaciones diferenciales ordinarias se pueden resolver analíticamente, se pueden utilizar características del *toolbox de matemática simbólica* para encontrar soluciones exactas.

Si no se pueden resolver directamente de forma analítica, es conveniente resolverlas numéricamente.

En Matlab, las derivadas se dan mediante un vector columna.

ode23(function_name, to,tf,yo) y **ode45(function_name,to,tf,yo)** integra un conjunto de ecuaciones diferenciales descritas en la función **function_name** desde un instante inicial **to**, a un instante final **tf**, comenzando con la condición inicial **yo**.

18.7. EJEMPLOS :

Ejemplo 1 :

```
>> f='2*exp(-x).*sin(x)';           % Definimos la función f.
>> fplot(f,[0 8])                   % Representa la función sobre el rango 0≤x≤8.
>> title(f), xlabel('x')
```

Ejemplo 2 :

```
>> x=[0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1];           % Datos.
>> y=[-.447 1.987 3.28 6.16 7.08 7.34 7.66 9.56 ... 9.48 9.30 11.2]; % Datos.
>> n=2;                                           % Orden del ajuste.
>> p=polyfit(x,y,n)                             % Encuentra los coeficientes del polinomio.
p =
    -9.8108    20.1293    -0.0317
>> xi=linspace(0,1,100);
>> z=polyval(p,xi);                             % Evalúa el polinomio.
>> plot(x,y,'o',x,y,xi,z,':')
>> xlabel('x'), ylabel('y=f(x)'),
>> pd=polyder(p)                                 % Encuentra la derivada.
pd =
    -19.6217    20.1293
```

19. GRAFICOS 2-D :

19.1. UTILIZACION DE LA ORDEN PLOT :

Si uno de los argumentos es una matriz y el otro un vector, la orden **plot** representa cada columna de la matriz respecto del vector. Si cambia el orden de los argumentos, la gráfica girará 90 grados.

19.2. ESTILO DE LINEAS, MARCADORES Y COLORES :

Se pueden especificar los colores y estilos de línea dando un argumento adicional a **plot** después de cada pareja de arrays de datos. El argumento opcional adicional es una cadena de caracteres formada por uno, dos o tres caracteres de la tabla siguiente:

SIMBOLO	COLOR	SIMBOLO	ESTILO DE LINEA
y	amarillo	•	punto
m	magenta	o	círculo
c	cien	×	marca-x
r	rojo	+	más
g	verde	*	estrella
b	azul	-	línea sólida
w	blanco	:	línea punteada
k	negro	-.	línea punto-rama
		--	línea de trazos

19.3. ADICION DE REJILLAS Y ETIQUETAS :

La orden **grid on** añade una rejilla a la gráfica actual en las marcas. La orden **grid off** elimina la rejilla. **grid** sin ningún argumento, las conmuta. Los ejes horizontal y vertical se pueden etiquetar, respectivamente, con las órdenes **xlabel** e **ylabel**. La orden **title** añade una línea de texto en la parte superior de la gráfica.

Podemos añadir también cualquier cadena de texto a cualquier localización específica en la gráfica con la orden:

text(x,y,'string'),

donde (x,y) representa las coordenadas de la arista del centro izquierda de la cadena de texto en unidades tomadas de los ejes de la gráfica. También se puede poner una cadena de texto con el ratón:

gtext('string')

19.4. EJES A MEDIDA :

ORDENES	DESCRIPCION
axis([xmin xmax ymin ymax])	Fija los valores máximo y mínimo de los ejes usando los valores dados en el vector fila.
axis auto	Devuelve el escalado de los ejes a sus valores por defecto: $xmin=min(x)$, $xmax=max(x)$, etc.
axis('auto')	
axis(axis)	Congela el escalado de los ejes en los límites actuales, así que si se activa <i>hold</i> , las gráficas subsiguientes usan los mismos límites de los ejes.
axis xy	Usa coordenadas cartesianas (por defecto), donde el <i>origen del sistema</i> está en el ángulo inferior izquierdo. El eje horizontal aumenta de izquierda a derecha y el eje vertical de abajo hacia arriba.
axis('xy')	
axis ij	Usa coordenadas de <i>matriz</i> , donde el origen del sistema está en el ángulo superior izquierdo. El eje horizontal aumenta de izquierda a derecha, pero el eje vertical de arriba hacia abajo.
axis('ij')	
axis square	Fija que la gráfica actual sea en un cuadrado en lugar del rectángulo que utiliza por defecto.
axis('square')	
axis equal	Fija que los factores de escala para ambos ejes sean iguales.
axis('equal')	
axis normal	Desactiva <i>axis equal</i> y <i>axis square</i> .
axis('normal')	
axis off	Desactiva todos los etiquetados de ejes, rejillas y marcas. Deja el título y las etiquetas colocadas por las órdenes <i>text</i> y <i>gtext</i> .
axis('off')	
axis on	Activa el etiquetado de ejes, marcas y rejilla.
axis('on')	

19.5. IMPRESIÓN DE FIGURAS :

- print** imprime la gráfica actual en la impresora.
- orient** cambia el modo de orientación.

Por defecto, dentro de la máquina **DEC-AXP-2100 Digital UNIX**, Matlab almacena las gráficas en formato EPS (PostScript). Podemos, transformarlas en otro tipo de formato mediante el driver adecuado. De esta forma podremos visualizarlas e imprimirlas desde nuestro PC con cualquier programa básico para Windows.

Así, por ejemplo, para pasarlas a formato JPG debemos seguir los siguientes pasos:

- 1.- Nombrar la gráfica dentro del comando **print** (no en **Save as ...**).
- 2.- Poner la orden:

print -djpeg nombre_gráfica.jpg

- 3.- Hacer un FTP a nuestro PC en modo binario, auto.

Otros drivers de transformación de formato son :

- **-dtiff** Formato TIFF comprimido.
- **-dtiffnocompression** Formato TIFF no comprimido.
- **-dbitmap** Formato bitmap.

Para más información sobre la impresión de gráficas, teclea (dentro del entorno de Matlab de la máquina) la orden:

help print

19.6. MANIPULACION DE DATOS :

Se pueden añadir líneas a una gráfica existente usando la orden **hold**.

figure con ningún argumento, crea una nueva ventana de figura. Utilizando **figure(n)** elegimos una ventana **n** de figura específica.

La orden **subplot(m,n,p)** subdivide la ventana de la figura actual en una matriz **m´n** de las áreas de representación gráfica y escoge como activa el área p-ésima. Para retornar al modo por defecto: **subplot(1,1,1)**.

Podemos ampliar una región: **zoom on**; el cual se desactiva con **zoom off**. También podemos ampliar y reducir la gráfica creada usando el archivo-M llamado **peaks.m**.

19.7. OTRAS CARACTERISTICAS DE LOS GRAFICOS 2-D :

ORDENES	DESCRIPCION
loglog	Es lo mismo que <i>plot</i> , excepto que se usan escalas logarítmicas para ambos ejes.
semilogx	Es lo mismo que <i>plot</i> , excepto que usa escala logarítmica en el eje x y escala lineal en el eje y . Lo mismo con <i>semilogy</i> .
polar(t,r,S)	Crea gráficos en coordenadas polares, donde t es el vector de ángulos en radianes, r es el radio vector y S es una cadena de caracteres opcional que describe color, símbolo que se emplea para marcar y/o estilo de línea.
bar	Generan, respectivamente, los gráficos de barras y escaleras.
stairs	
hist(y)	Dibuja un histograma.
hist(y,n)	Dibuja un histograma con n elementos (n es un escalar).
hist(y,x)	Dibuja un histograma usando los elementos especificados en x (vector).
stem	Representa la secuencia de datos discretos.
errorbar(x,y,e)	Representa la gráfica del vector x frente al vector y con barras de error especificadas por el vector e .
compass	Representan datos complejos.
feather	
rose	Dibuja un histograma polar.
ginput	Proporciona un medio de seleccionar puntos de la gráfica actual usando el ratón.
fplot	Representa automáticamente una función de una variable entre límites especificados sin crear un conjunto de datos para la variable.
fill(x,y,'c')	Rellena el polígono 2-D definido por los vectores columna x e y con el color especificado por c .

19.8. EJEMPLOS :

Ejemplo 1 :

```
>> x=linspace(0,2*pi,30);      % Crea 30 datos en el intervalo  $0 \leq x \leq 2\pi$ 
>> y=sin(x);                  % Vector que contiene el seno de los datos en x.
>> plot(x,y)                   % Representa x frente a y.

>> z=cos(x);
>> plot(x,y,x,z)              % Representa un seno frente a un coseno en la misma gráfica.
```

```

>> W=[y;z]; % Crea una matriz con las funciones seno y coseno.
>> plot(x,W) % Representa las columnas de W frente a x.
>> plot(W,x) % Representa x frente a las columnas de W.

>> plot(x,y,'g:',x,z,'r--',x,y,'wo',x,z,'c+') % Usa diferentes estilos de línea, colores
y marcas de puntos.

>> plot(x,y,x,z)
>> grid % Activa la rejilla.
>> xlabel('Variable Independiente X') % Etiqueta del eje x.
>> ylabel('Variables dependientes Y y Z') % Etiqueta del eje y.
>> title('Curvas de seno y coseno') % Título de la gráfica.
>> text(2.5,0.7,'sin(x)') % Añade una etiqueta identificando la curva del seno en
la localización (2.5,0.7).
>> gtext('cos(x)') % Etiqueta la gráfica del coseno con el ratón.

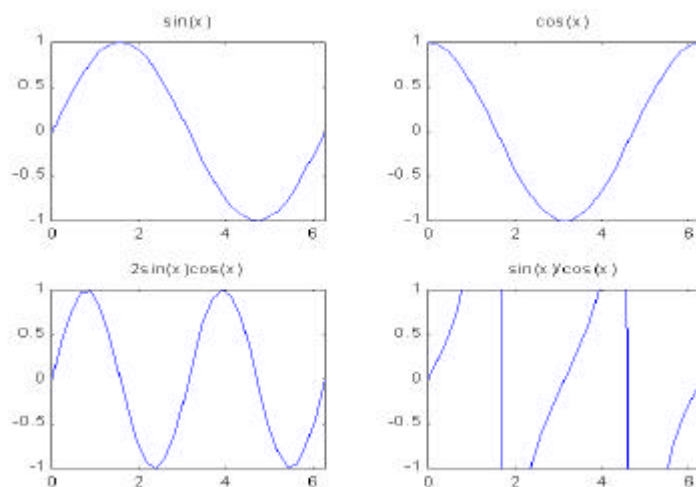
```

Ejemplo 2 :

```

>> x=linspace(0,2*pi,30);
>> y=sin(x);
>> z=cos(x);
>> a=2*sin(x).*cos(x);
>> b=sin(x)./(cos(x)+eps);
>> subplot(2,2,1) % Selecciona la subgráfica superior izquierda.
>> plot(x,y),axis([0 2*pi -1 1]),title('sin(x)')
>> subplot(2,2,2) % Selecciona la subgráfica superior derecha.
>> plot(x,z),axis([0 2*pi -1 1]),title('cos(x)')
>> subplot(2,2,3) % Selecciona la subgráfica inferior izquierda.
>> plot(x,a),axis([0 2*pi -1 1]),title('2sin(x)cos(x)')
>> subplot(2,2,4) % Selecciona la subgráfica inferior derecha.
>> plot(x,b),axis([0 2*pi -1 1]),title('sin(x)/cos(x)')

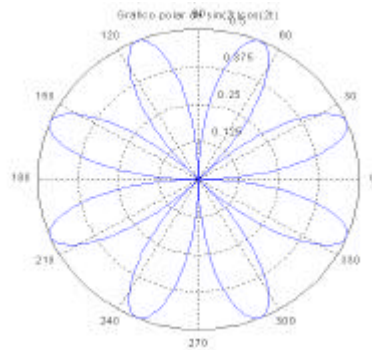
```



```
>> subplot(1,1,1) % Retorna a una única gráfica en la ventana de figura.
```

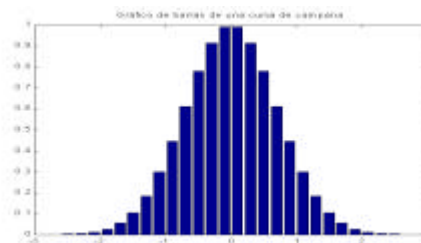
Ejemplo 3 :

```
>> t=0:.01:2*pi;  
>> r=sin(2*t).*cos(2*t);  
>> polar(t,r)  
>> title('Gráfico polar de sin(2t)cos(2t)')
```

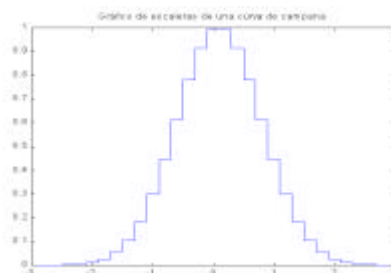


Ejemplo 4 :

```
>> x=-2.9:0.2:2.9;  
>> y=exp(-x.*x);  
>> bar(x,y)  
>> title('Gráfico de barras de una curva de campana')
```

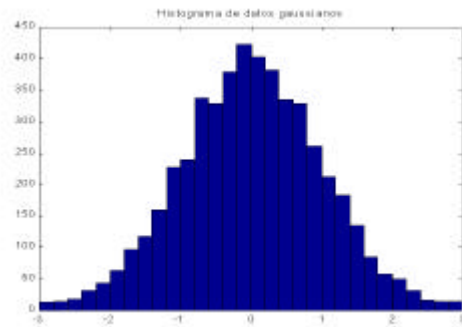


```
>> stairs(x,y)  
>> title('Gráfico de escaleras de una curva de campana')
```

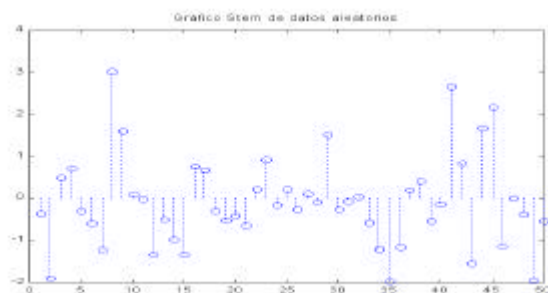


Ejemplo 5 :

```
>> x=-2.9:0.2:2.9;           % Especifica los elementos a usar.  
>> y=randn(5000,1);         % Crea 5000 puntos aleatorios.  
>> hist(y,x)                 % Dibuja el histograma.  
>> title('Histograma de datos gaussianos')
```

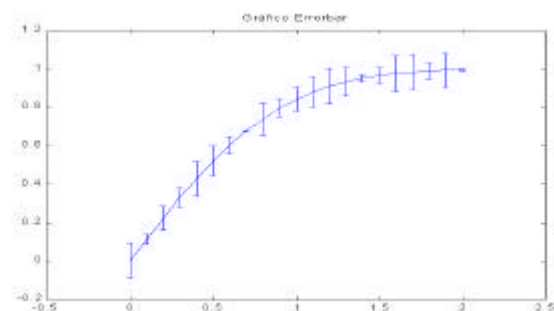


```
>> y=randn(50,1);           % Crea algún dato aleatorio.  
>> stem(y,':')              % Dibuja una gráfica stem con línea punteada.  
>> title('Gráfico Stem de datos aleatorios')
```



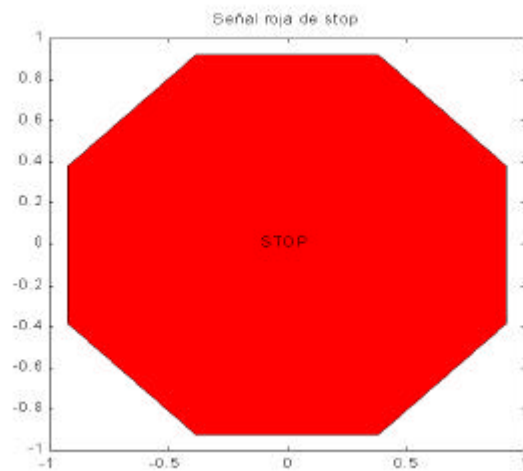
Ejemplo 6 :

```
>> x=0:0.1:2;               % Crea un vector.  
>> y=erf(x)                  % y es la función error de x.  
>> e=rand(size(x))/10;      % Genera valores de error aleatorios.  
>> errorbar(x,y,e)          % Crea la gráfica.  
>> title('Gráfico Errorbar')
```



Ejemplo 7:

```
>> t=(1/8:2/8:15/8)*pi;          % Vector columna.  
>> x=sin(t);  
>> y=cos(t);  
>> fill(x,y,'r')      % Un círculo rojo relleno usando sólo 8 puntos.  
>> axis('square')  
>> text(-.11,0,'STOP')  
>> title('Señal roja de stop')
```



20. GRAFICOS 3-D :

20.1. GRAFICOS DE LINEA :

Extendemos la orden **plot** de 2-D a 3-D con **plot3**. El formato es el mismo, excepto que los datos están en tripletes.

20.2. GRAFICOS DE MALLA Y DE SUPERFICIE :

20.2.1. Superficie de malla :

[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**.

Una vez dada esta orden, la gráfica de malla se genera mediante: **mesh(X,Y,Z)**.

mesh acepta un argumento opcional para controlar los colores. También puede tomar una matriz simple como un argumento: **mesh(Z)**.

20.2.2. Gráfica de superficie :

Es como la gráfica de malla, excepto que se rellenan los espacios entre líneas. Las gráficas de este tipo se generan usando la función **surf** (mismos argumentos que la función **mesh**).

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

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

20.3. MANIPULACION DE GRAFICOS :

La función **view(azimut,elevación)** fija el ángulo de visión especificando el *azimut* y la *elevación*. **view([x y z])** coloca su vista en un vector que contiene la coordenada cartesiana (x,y,z) en el espacio 3-D. El azimut y la elevación de la vista actual se pueden obtener utilizando **[az,el]=view**.

La orden **hidden** controla la eliminación de líneas escondidas. Si activamos **hidden off** veremos todas las partes a través de la malla.

20.4. OTRAS CARACTERISTICAS DE LOS GRAFICOS 3.D :

La función **clabel** añade etiquetas de altura a los gráficos de contorno.

fill3 es la versión 3-D de **fill**.

20.5. COMPRESION DE LOS MAPAS DE COLOR :

ROJO	VERDE	AZUL	COLOR
0	0	0	negro
1	1	1	blanco
1	0	0	rojo
0	1	0	verde
0	0	1	azul
1	1	0	amarillo
1	0	1	magenta
0	1	1	cian
0.5	0.5	0.5	gris medio
0.5	0	0	rojo oscuro
1	0.62	0.4	cobre
0.49	1	0.83	agua marina

FUNCION	DESCRIPCION DEL MAPA DE COLOR
hsv	Matiz-saturación-valor
hot	Negro-rojo-amarillo-blanco
cool	Sombras de cyan y magenta
pink	Sombras de rosa pastel
gray	Escala lineal de gris
bone	Escala de gris con un tinte de azul
jet	Una variante de HSV
copper	Tono cobre lineal
prism	Prisma
flag	Alternar rojo, blanco, azul y negro

20.6. UTILIZACION DE MAPAS DE COLOR :

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

20.7. VISUALIZACION DE MAPAS DE COLOR :

Se puede hacer de varias formas:

- Se pueden visualizar los elementos en una matriz del mapa de color mediante **hot(m)**.
- También mediante la función **pcolor** se puede visualizar un mapa de color.
- La función **colorbar** añade una barra de color vertical u horizontal (escala de color) a la ventana de figura actual mostrando las transformaciones de color para el eje actual.

20.8. CREACION Y ALTERACION DE LOS MAPAS DE COLOR :

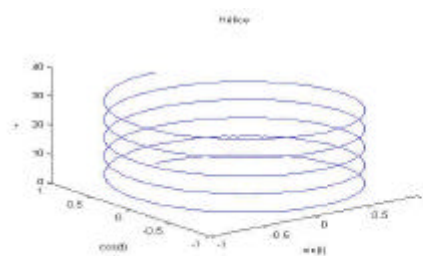
La función **brighten** ajusta un mapa de color dado para aumentar o disminuir la intensidad de los colores oscuros.

Podemos crear un mapa de color mediante **colormap(mymap)**.

20.9. EJEMPLOS :

Ejemplo 1 :

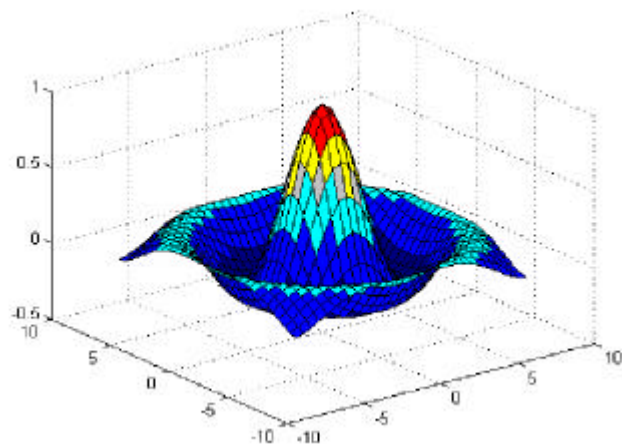
```
>> t=0:pi/50:10*pi;  
>> plot3(sin(t),cos(t),t)  
>> title('Hélice'),xlabel('sin(t)'),ylabel('cos(t)'),zlabel('t')
```



```
>> axis('ij') % Hace que el eje y aumente de atrás hacia adelante.
```

Ejemplo 2 :

```
>> x=-7.5:7.5;
>> y=x;
>> [X,Y]=meshgrid(x,y);    % Genera puntos igualmente espaciados en el plano xy
entre -7.5 y 7.5 en ambos x e y.
>> R=sqrt(X.^2+Y.^2)+eps;    % Distancia desde el origen (0,0).
>> Z=sin(R)./R;
>> mesh(X,Y,Z)              % Genera la gráfica de malla.
>> surf(X,Y,Z)              % Genera la gráfica de superficie.
```



Ejemplo 3 :

```
>> view([-7 -9 7])          % Vista a través de (-7,-9 7) al origen.
>> [az,el]=view             % Encuentra el azimuth y la elevación
```

1	1	5
1	2	0
1	3	11
1	4	13
1	5	0
1	6	72
1	7	27
2	1	7
2	2	4
2	3	4
2	4	0
2	5	0
2	6	27
2	7	22
3	1	4
3	2	11
3	3	0
3	4	0
3	5	4
3	6	20
3	7	39
4	1	8
4	2	13
4	3	7
4	4	5
4	5	0
4	6	16
4	7	24
5	1	4
5	2	9
5	3	4
5	4	5
5	5	4
5	6	23
5	7	10
6	1	2
6	2	4
6	3	7
6	4	20
6	5	5
6	6	4
6	7	23