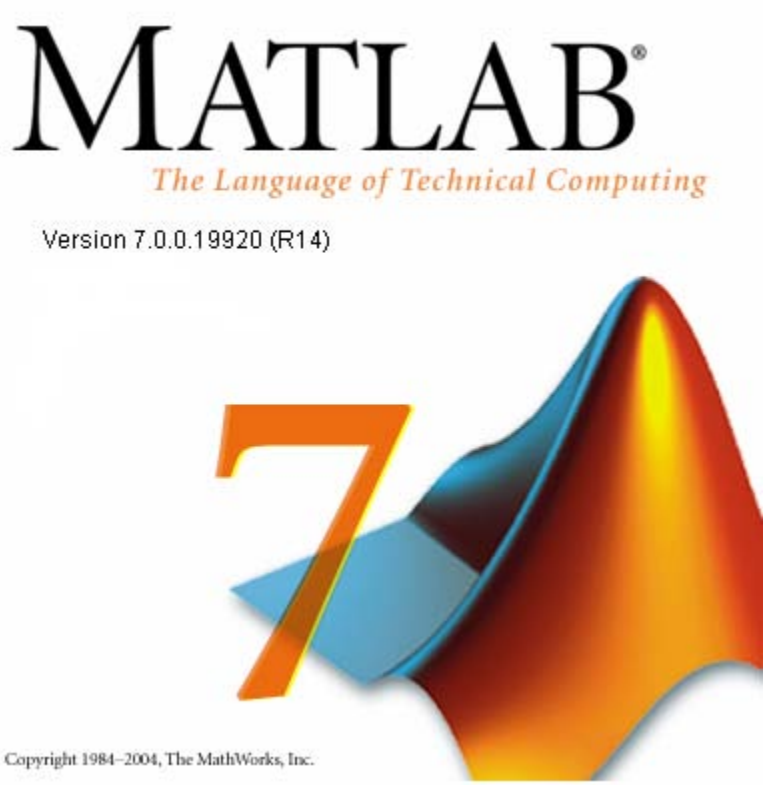
MATLAB

1. Introducción
2. Entorno de trabajo
3. Inicio
4. Ficheros y scripts
5. Vectores y Matrices
6. Gráficas
7. Cómo hacer un programa en Matlab
8. Funciones
9. Ficheros



1. Introducción

MATLAB = 'MATrix LABoratory'

MATLAB es un lenguaje para cálculos técnicos.

Permite realizar cálculos, visualización y programación de modo sencillo, donde los problemas y las soluciones se expresan en una notación matemática familiar.

Integra análisis numérico, matrices, procesamiento de señales y gráficas, todo esto en un ámbito donde los problemas y soluciones son expresados tal como se escriben matemáticamente.

Escrito inicialmente como auxiliar en la programación de cálculo con matrices.

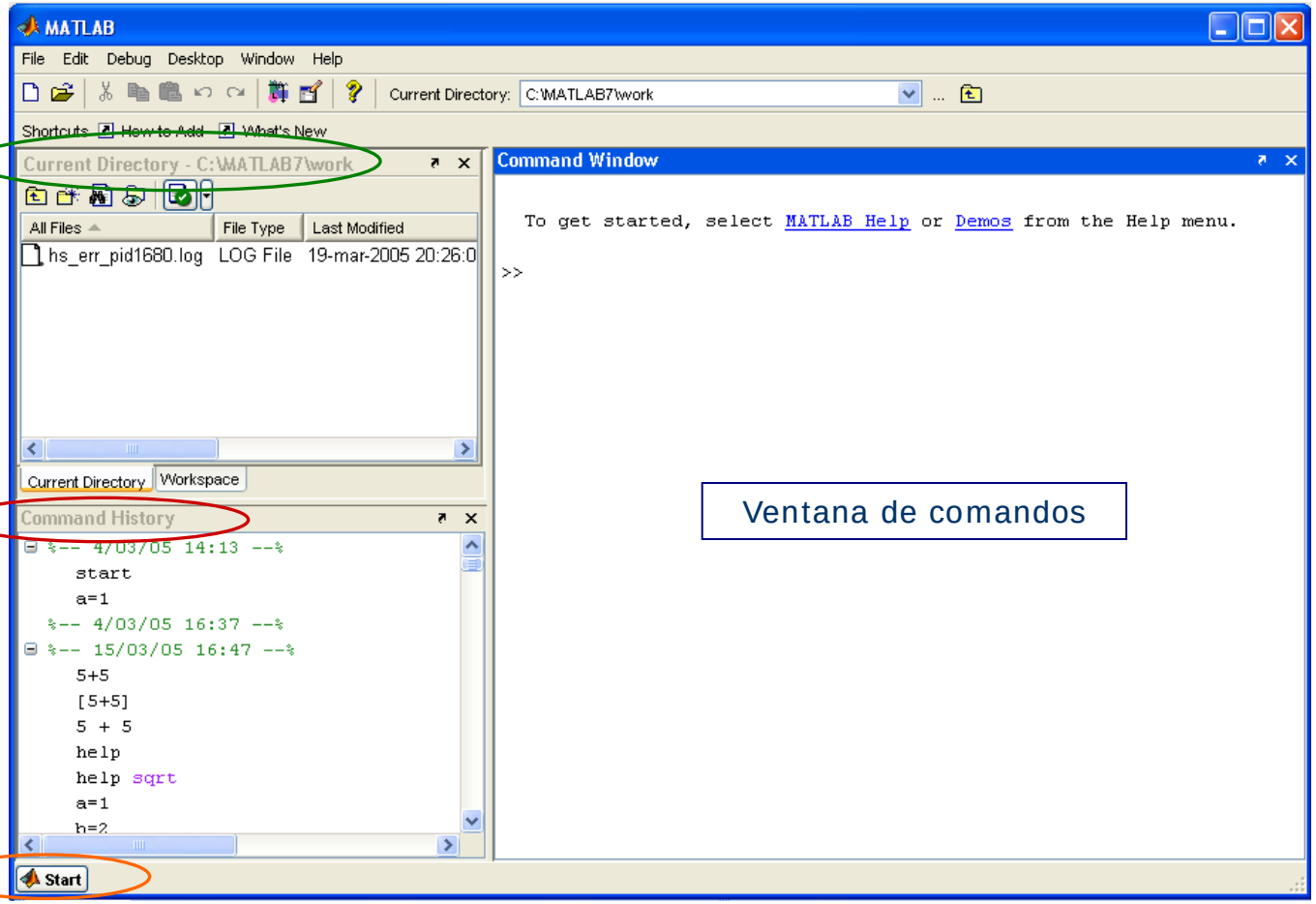
MATLAB es un lenguaje de programación amigable al usuario con características más avanzadas y mucho más fáciles de usar que los lenguajes de programación como basic, pascal o C.

2. Entorno de trabajo

Directorio actual y espacio de trabajo

Historial

Menú inicio



Ventana de comandos

2. Entorno de trabajo

Ventanas disponibles

- La ventana de comandos (**Command Window**),
- La ventana histórica de comandos (**Command History**),
- El espacio de trabajo (**Workspace**),
- La plataforma de lanzamiento (**Launch Pad**),
- El directorio actual (**Current Directory**),
- La ventana de ayuda (**Help**)
- El editor de ficheros y depurador (**Editor&Debugger**),
- El editor de vectores y matrices (**Array Editor**).

2. Entorno de trabajo

Command Window

- Los comandos se teclean tras el prompt >>
- El punto y coma (;) al final de un comando inhibe mostrar el resultado
- Todo lo que se teclea tras % se considera un comentario

Algunos comandos interesantes

Comandos	
clc	limpia la pantalla de comandos
save	guarda en un fichero el estado de trabajo
load	lee de un fichero un estado de trabajo
diary	almacena en un fichero todo lo que va ocurriendo
help	Muestra la ayuda

3. Inicio

Cálculos numéricos. Las operaciones básicas entre números se escriben con la siguiente notación: suma (**+**), resta (**-**), producto (*), cociente (/) y potencia (^). Por ejemplo:

```
>>2+2
ans=
      4
>>2*(2+5*(1+2))
ans=
      34
```

```
>>3+2.1 (0.1*5)
ans=
      4.4491
```

Funciones matemáticas definidas:

Funciones en las calculadoras	Funciones trigonométricas, hiperb…	Funciones de números complejos
abs Valor absoluto.	**sin** Seno	**abs** Módulo de un número complejo
sqrt Raíz cuadrada.	**cos** Coseno	**angle** Ángulo de un número complejo
round Redondeo al entero más cercano	**tan** Tangente	**real** Parte real de un número complejo
fix Redondeo hacia el cero	**asin** Arcoseno	**imag** Parte imaginaria
floor Redondeo inferior	**acos** Arcocoseno	**conj** complejo conjugado
ceil Redondeo superior	**atan** Arcotangente	…
sign Signo de un número real	**atan2** Ángulo de un punto en el plano	…
rem Resto o módulo	**sinh** Seno hiperbólico	
exp función exponencial de base **e**	**cosh** Coseno hiperbólico	
log Logaritmo neperiano	**tanh** Tangente hiperbólica	
$\log_{10}$ Logaritmo en base 10	…	

3. Inicio

Variables:

```
>>a=4.35
a=
    4.3500
```

```
>>2*a
ans=
       8.700
```

```
>>a
a=
    4.3500
```

```
>>who
    Your variables are:
    a       ans
    leaving 291636 bytes of memory free
```

```
>>whos
Name   Size     Total   Complex
 a      1 by 1       1         No
 ans    1 by 1       1         No
Grand total is (2*8)=16 bytes,
leaving 291636 bytes of memory free
```

Los nombres de las variables deben empezar por una letra

Sólo se consideran los primeros 19 caracteres

MATLAB es sensible a las mayúsculas (case sensitive)

Comandos
who, **whos**. Lista las variables en el workspace
clear. Elimina una o varias variables del workspace

3. Inicio

Variables:

Existen variables especiales en MATLAB que están definidas al arrancar el programa:

ans.- Almacena el resultado de la última expresión a la que no se le ha asignado una variable. Se suele utilizar para referenciar el resultado de la operación previa.

eps.- Almacena la precisión de la máquina

pi.- Almacena la constante Π

i ó **j.-** Representa la unidad imaginaria.

Inf ó **inf.-** Representa el infinito.

NaN ó **nan.-** Representación para algo que no es un número, como una indeterminación.

flops.- Almacena el número de operaciones realizadas.

3. Inicio

Tipos de datos

Por defecto, se trabaja con variables numéricas en coma flotante de doble precisión (double), pero existen otros tipos

```
>>a=4.35
a=
    4.3500
```

```
>>b=4.35+9*i
b=
      4.3500 + 9.0000i
```

```
>>c= a>1
c=
    1
```

Tipos de datos simples
double
single
int8, int16, int32, int64
uint8, uint16, uint32, uint64
logical
complex
char

Las variables tipo caracter permiten definir 'strings'

Se utilizan las comillas simples para definir el inicio y el fin

```
>>x='hola mundo'
x=
hola mundo
```

Funciones
int2str, num2str
str2num

3. Inicio

Entrada y salida de datos por consola:

Puede utilizarse la consola para introducir datos en las variables

```
user_entry = input ('prompt')
user_entry = input ('prompt', 's')
```

Y también para mostrar resultados

```
disp (variable)
```

4. Ficheros y scripts

Nos centraremos en los cuatro siguientes tipos de ficheros en Matlab

1.- ficheros .m

Para problemas con gran cantidad de datos de entrada o en los que estos son complejos, la creación de ficheros .m es necesaria para evitar la prueba-error en la programación en el prompt de Matlab. Contendrá instrucciones y/o **funciones**. También pueden ser solo de comandos, ***scripts***.

Creación: con un editor de textos ascii y guardando el fichero con extensión m o utilizando el editor de textos del matlab.

Ejecución: en el prompt del sistema tecleando el nombre del fichero sin extensión, y teniendo en cuenta que se debe encontrar en el directorio actual o en uno de los directorios indicados en el path de matlab.

2.- ficheros .mat

Son ficheros binarios que se utilizan para guardar el estado de la sesión.

Creación: Utilizando el comando save.

Uso: Utilizando el comando load.

3.- ficheros .p

Precompilados. Los ficheros ****.p*** se ejecutan algo más rápidamente que los ****.m*** y permiten ocultar el código de los ficheros ASCII correspondientes a las funciones ****.m*** de MATLAB.

4.- ficheros .dat

Ficheros de datos que se pueden utilizar para leer datos de entrada e interpretarlos o representarlos con los comandos adecuados.

5. Vectores y matrices

Las variables no sólo pueden ser números, sino que también pueden representar vectores y matrices.

Es más, para MATLAB los números son simplemente matrices 1x1.

Recordemos, por otro lado, que un vector es un caso particular de matriz.

Para definir una matriz se abre un corchete "**[**" y, seguidamente, se introduce la matriz elemento a elemento y se cierra con otro corchete "**]**".

Las filas se separan mediante el simbolo ";" o la tecla enter
Las columnas se separan mediante un espacio o una coma

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

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

Definir un vector es introducir una matriz fila,

```
>>c=[1 2 3]
c=
    1 2 3
```

o columna:

```
>>d=[1;2;3]
d=
     1
     2
     3
```

```
>>cd=6; e=3; h=4;
>> M=[e, cd*h, cos(pi/3); h^2,
sqrt(h*h/cd), 14]

M=
    3.0000 24.0000  0.5000
   16.000    1.6300 14.0000
```

5. Vectores y matrices

Otras formas de crear vectores:

Utilizando el operador ':'. Se especifica el valor inicial, el espaciado y el valor final

```
>>x1 = 1 : 5          % Vector de 1 a 5, con incremento de 1
>>x2 = 10 : 5 : 100   % Vector de 10 a 100, con incrementos de 5.
```

Utilizando la funcion **linspace.** Se especifica el valor inicial, el valor final y el nº de elementos

```
>>x3 = linspace(1, 5, 8)        % Vector de 1 a 5, de 8 elementos
```

Si se quiere concatenar x1 y x2

```
>>C = [ x1 x2 ]
```

Acceso a los elementos:

```
Acceso a los elementos:

>>A(2,3)
   ans=5

>>B(1,1)
   ans=1
```

```
Acceso a los elementos:

>>c(2)
   ans=2

>>d(2)
   ans=2
```

5. Vectores y matrices

Operaciones con vectores (siendo 'a' y 'b', dos variables que contienen vectores):

```
>>a + 10          % Suma de un escalar con un vector
>>a * 10          % Multiplicación de un escalar con un vector
>>a + b           % Suma de dos vectores. La respuesta se guarda en ans.
>>a=b'            % Cargamos en 'a' la traspuesta de 'b'.
```

```
>>Z = 100 - 2 * a + b
```

La multiplicación de vectores se hace con (. *), ya que cuando se utiliza el asterisco sin punto indica multiplicación matricial, y además provoca un error.

```
>>Z = a .* b
```

La división también lleva un punto antes del signo, porque sino se utiliza el punto nos referimos a la división matricial que es muy diferente.

```
>>Z = a ./ b
```

La siguiente operación obtiene el cuadrado del vector " a ".

```
>>Z = a .^ 2
```

5. Vectores y matrices

Operaciones con matrices:

+	adición o suma
–	sustracción o resta
*	multiplicación
'	traspuesta
^	potenciación
\	división-izquierda
/	división-derecha

.*	producto elemento a elemento
./ .\	división elemento a elemento
.^	elevar a una potencia elemento a elemento

5. Vectores y matrices

Funciones para operar con matrices:

Funciones para operar con matrices	
det	Determinante de la matriz
inv	Matriz inversa
mod, rem	Módulo y resto de la división de los elementos de un vector y un valor
sum	Suma de los elementos de un vector
cumsum	Suma de los elementos de un array
max, min	Máximo y mínimo de los elementos de un vector
size	Número de filas y columnas de un array
length	Número de elementos deun vector
diag	Devuelve un vector on la diagonal principal de la matriz
rot90	Rota la matriz 90 grados
sort	Ordena los elementos de la matriz
find	Busca elementos en un array. Puede devolver los índices
trace	Suma de los elementos de la diagonal
fliplr	Invierte el orden de los elementos de la matriz de derecha a izquierda
flipud	Invierte el orden de los elementos de la matriz de arriba a abajo

5. Vectores y matrices

Operaciones con matrices (ejemplos):

Creamos la matriz A, utilizando incrementos

```
>>A=[1:0.1:2;3:0.1:4]      % ¿Qué elemento es A(8)?
```

Definamos las siguientes matrices ' g ' y ' h '.

```
>>g = [ 1 2 3; 4 5 6; 7 8 9]
>>h = [ 1 0 2 ; 11 2 3 ; 3 5 12 ]

>>det(g)                   % Determinante de la matriz g
>>a=g'                     % Guarda en la matriz a la traspuesta de g
>>inv(g)                   % Inversa de la matriz g
>>k = g + h                % Suma de matrices
>>k = g * h                % Multiplicación de dos matrices.
>>Z = a .* 2               % Multiplicación de los elementos de la matriz por un escalar
>>[L, U ] = lu (k)         % Calcula la factorización LU de la matriz cuadrada k
>>[d,e]= qr (k)            % Calcula la factorización QR de la matriz k
>>rank(k)                  % Devuelve el rango de la matriz k
>>cond(k)                  % Devuelve el número de condición de la matriz k.
```

5. Vectores y matrices

Funciones para generar matrices especiales:

Generación de matrices especiales	
eye(m,n)	Matriz de orden mxn con unos en la diagonal -identidad-
eye(n)	Equivale a la matriz identidad nxn
eye(A)	Si a es una matriz mxn entonces equivale a eye(mxn)
zeros(m,n)	Matriz de orden mxn cuyos elementos son ceros
ones(m,n)	Matriz de orden mxn cuyos elementos son unos
rand(m,n)	Matriz de orden mxn cuyos elementos son escogidos de forma aleatoria en el intervalo [0,1]
magic(n)	Matriz mágica de orden n
linspace(x,y,n)	Genera un vector con n puntos espaciados uniformemente entre x e y / linspace(x,y) = linspace(x,y,100)
logspace(a,b,c)	Genera un vector de componentes: 10 elevado a las componentes del vector que daría **linspace**
magic(n)	Genera una matriz tal que la suma de los elementos de cada fila y cada columna es la misma

5. Vectores y matrices

Modificaciones en matrices:

Definamos la matriz A

```
>>A = [ 1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1]   % Definimos la matriz A: identidad, de 4x4
>>A(5,5)=1                                    % Le añadimos un 1 en la fila 5, columna 5

>>B = A (1:3,1:3)                             % Definimos la matriz B a partir de las tres
>>C = B ( 3 : -1 : 1 , 1 : 3 )                % primeras filas y las tres primeras columnas de A
                                              % Definimos la matriz C a partir de la matriz B
                                              % tomando las filas en orden inverso
>>C = A ( : , [ 1 3 5 ] )                     % Definimos la matriz C tomando todas las filas
>>D=[A,C]                                     % de A y las columnas 1, 3, y 5
                                              % Definimos la matriz D concatenando A y C

>>A = [ 1 2 3 4 5; 1 2 3 4 5; 1 2 3 4 5; 1 2 3 4 5 ]   % Definimos la matriz D tomando las filas de A y
>>B = [ 6 7 8; 6 7 8; 6 7 8; 6 7 8 ]                   % las columnas 1, 2, y 5 y todas las filas de B y
>>D = [ A(:,[ 1 2 5]) B(:, [ 1 3])]                    % las columnas 1 y3

>>D( :,1)=[]                                  % Eliminamos la primera columna
```

5. Vectores y matrices

Resolución de sistemas de ecuaciones:

Dado el siguiente sistema de ecuaciones:

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

Capturamos los valores de x, y, z formando una matriz:

```
>>A = [ 2 0 5; 3 5 9; 1 5 7]   % Definimos la matriz de coeficientes.
>>b = [ 100 ; 251; 301 ]       % Definimos el vector de términos independientes, en este
                                % caso como un vector columna.
```

NOTA: Si lo definiéramos como un vector fila, debemos utilizar su traspuesta en el cálculo.

```
>>c = inv (A)* b        % Una manera de realizar el cálculo es mediante la inversa

>>det(A)                % Debemos recordar que el determinante de la matriz debe
                        % ser distinto de cero. Es decir el sistema de ecuaciones
                        % es linealmente independiente.
```

5. Vectores y matrices

Cadenas de caracteres o strings

Las cadenas de caracteres no son mas que vectores de elementos de tipo caracter
Pueden crearse nuevas cadenas mediante la concatenanció

```
>> x='hola'
x =
hola

>> y='mundo'
y =
mundo

>> z=[x ' ' y]
z =
hola mundo
```

Funciones con cadenas de caracteres	
length(b)	calcula el tamaño de la cadena b
strcmp(a,b)	Compara dos cadenas. Si son iguales devuelve 1 y s son diferentes 0
[p,r]=strtok(a)	Separa las palabras de una cadena. Devuelve la primera en p y el resto en r

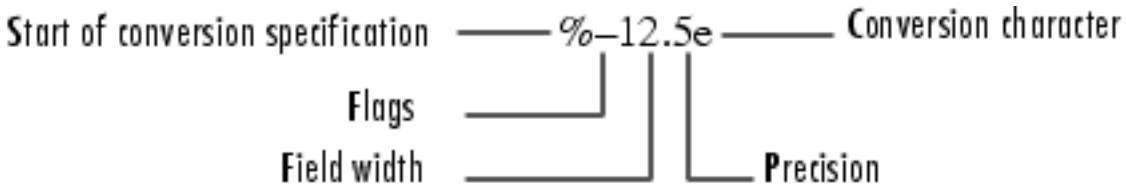
5. Vectores y matrices

Escritura formateada de datos sobre strings

Si deseamos convertir datos numéricos en cadenas, conun formato concreto, se utiliza la función sprintf

s = sprintf (format, A) % crea la variable cadena s, con los datos de A formateados
% según la cadena format

Format contiene las especificaciones de conversión



%d, %u	Notación decimal, con y sin signo
%f	Notación coma flotante
%e	Notación exponencial
%o, %x	Notaciones octal y hexadecimal
%c, %s	Caracter o cadena de caracteres

La combinación de sprintf y disp permite mostrar en pantalla datos de forma “elegante”:

disp(sprintf(‘pi=%1.4f’, pi);

6. Gráficas

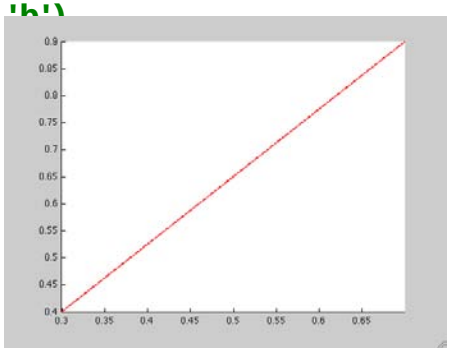
Funciones de bajo nivel:

Permiten dibujar gráficos elementales como líneas y polígonos. Dos de los más comandos útiles son los dedicados a dibujar líneas (**line**) y polígonos (**patch**)

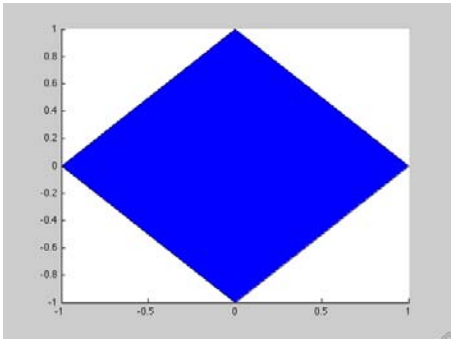
line(x,y,'Nombrepropiedad', valor, …)
patch (x,y, Color)

Entre las propiedades se encuentran el color (Color) y el tipo de línea (LineStyle)

>> line([.3 .7],[.4 .9], 'Color','r', 'LineStyle', '-.');



>>patch([-1 0 1 0] , [0 1 0 -1],



6. Gráficas

Matlab dispone de varios comandos de alto nivel para representar datos gráficamente. Veremos algunos que permiten realizar:

Gráficas en 2D:
- Funciones y=f(x)
- Curvas paramétricas
- Curvas en coordenadas polares

Gráficas en 3D:
- Funciones z=f(x,y)
- Curvas paramétricas en el espacio
- Curvas de nivel
- Superficies en el espacio
- Funciones complejas

Gráficas estadísticas:
- Diagramas de sectores
- Diagramas de pareto
- Diagramas de barra

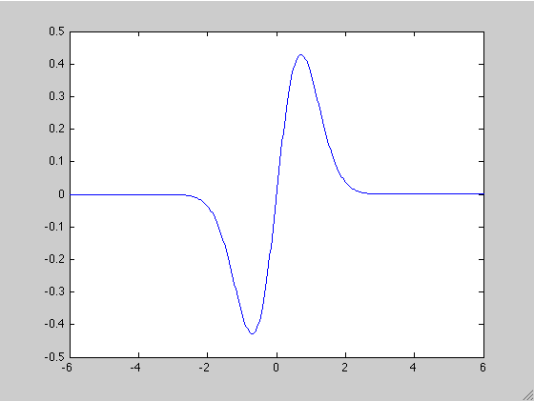
6. Gráficas en 2D

Gráficas en 2D: Funciones y=f(x). Comando plot

Permite representar los valores de un vector mediante una gráfica.

Ejemplo: dibujar la gráfica de $y = xe^{-x^2}$ para $x \in [-6,6]$

```
>> x=linspace(-6,6,400);
>> y=x.*exp(-x.^2);
>> plot(x,y);
```



Pueden modificarse diferentes aspectos de la gráfica mediante comandos:

Color y trazo (ver siguiente diapositiva)

Cuadrícula:

```
>> grid on;
>> grid off;
```

Ejes:

```
>> axis square;
>> axis [xmin xmax ymin ymax];
```

6. Gráficas en 2D

Gráficas en 2D: Comando plot en detalle

plot (x,y,'opciones')

Opciones: (Las opciones se pueden combinar)

Tipos de línea:

- **-** Continua
- **--** Discontinua
- **-:** Punteada
- **--.** Discontinua y punteada

Tipos de punto:

. Punto	**s** Cuadrado	**>** Triángulo (derecha)
+ Cruz	**d** Diamante	**p** Pentagrama
x Aspa	**v** Triángulo (abajo)	**h** Hexagrama
o Círculo	**^** Triángulo (arriba)	
***** Estrella	**<** Triángulo (izquierda)	

Colores

- **r** Rojo
- **g** Verde
- **b** Azul
- **w** Blanco
- **c1,c2,...**Otros colores

6. Gráficas en 2D

Gráficas en 2D: etiquetado de gráficas

Título de la gráfica:

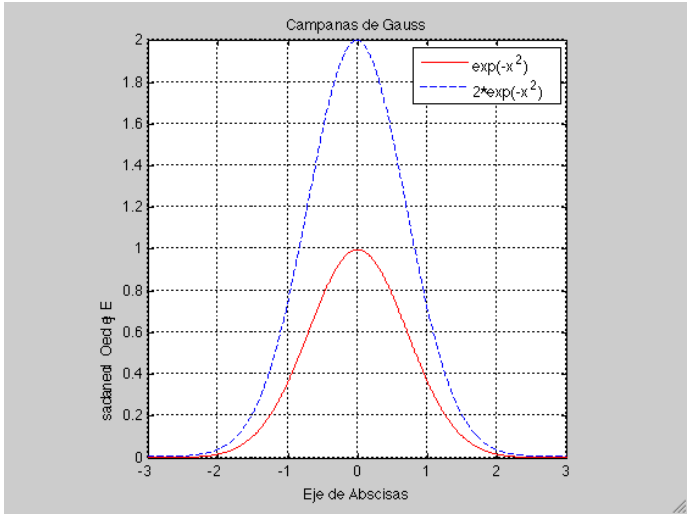
```
>> title('titulo de la gráfica');
```

Etiquetas en los ejes:

```
>> ylabel ('etiqueta eje vertical');
>> xlabel ('etiqueta eje horizontal');
```

Leyenda:

```
>> legend ('leyenda');
```



Superposición de gráficas

Plot permite dibujar tantas gráficas como se desee en la misma figura:

Mediante el comando hold:

```
>> plot (x, y);
>> hold on;
>> plot (x, z);
```

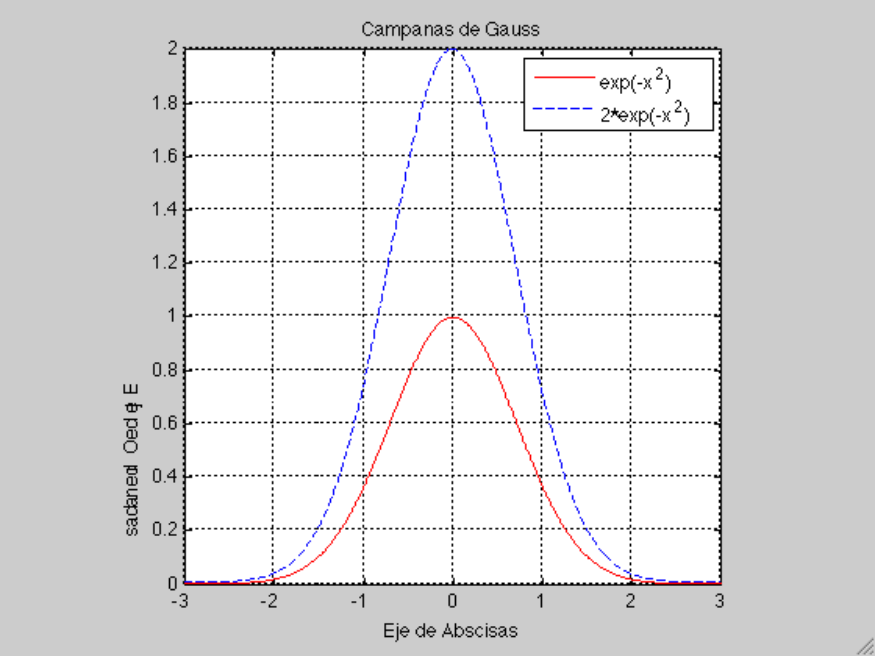
Dibujándolas juntas:

```
>> plot (x, y, 'r-', x, z, 'g—');
```

6. Gráficas en 2D

Gráficas en 2D: Ejemplo

```
>> x=linspace(-3,3,500);
>> y=exp(-x.^2);
>> z=2*exp(-x.^2);
>> plot(x,y,'r-',x,z,'b--');    % 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)');   % Pone una leyenda
```



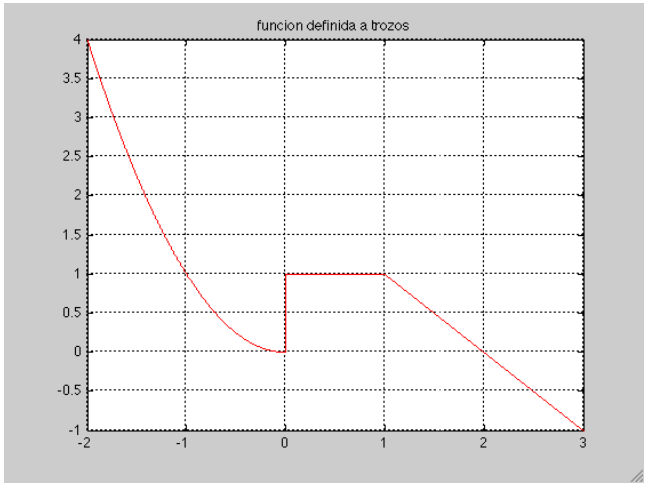
6. Gráficas en 2D

Gráficas en 2D: Ejemplo de una función definida a trozos

Representar la función f(x) para $x \in [-2,3]$

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

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

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

>> plot(x,y,'r');
>> grid on;
>> title('función definida a trozos');
```



6. Gráficas en 2D

Otros comandos relacionados con la visualización:

Funciones utilizadas en gráficos	
subplot	Permite mostrar en la misma ventana varias gráficas diferentes
figure	Abre una nueva ventana de gráficos
grid	Construye una malla sobre el dibujo
text(x,y,'string')	Añade la ristra 'string' en la posición x,y del gráfico. Texto colocado mediante coordenadas
gtext('string')	Añade la ristra 'string' en la posición indicada con el ratón. Texto colocado con el ratón
plottools	Muestra u oculta las herramientas de edición de un gráfico
semilogx semilogy	Como plot excepto que para los ejes x o y se utiliza una escala logarítmica (base 10)
loglog	Como plot excepto que para ambos ejes se utiliza una escala logarítmica (base 10)
clf	Borra todos los objetos de la gráfica.

6. Gráficas en 2D

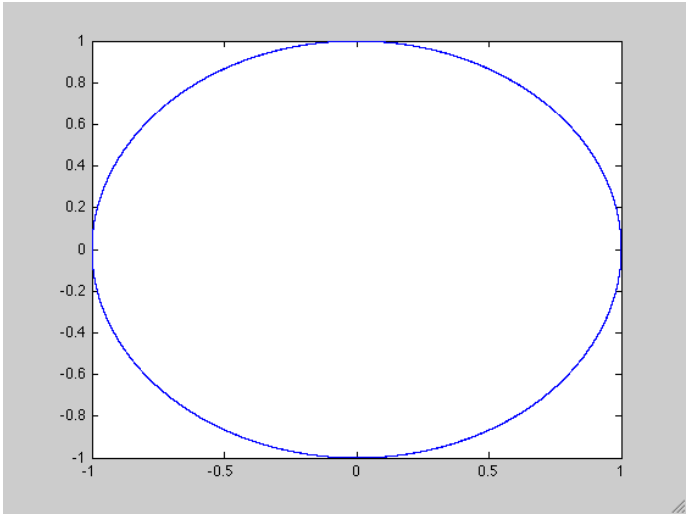
Gráficas en 2D: Curvas paramétricas

Matlab nos permite representar en el plano curvas dadas de forma paramétrica. Para ello podemos hacer uso de los comandos **plot** y **comet.**

Ejemplo: dibujar la gráfica 2D de

$$\vec{r}(t) = (sen(t), \cos(t)) \quad 0 \le t \le 2\pi$$

```
>> t=linspace(0, 2*pi, 3000);
>> plot(sin(t),cos(t));
>> comet(sin(t),cos(t));
```



6. Gráficas en 2D

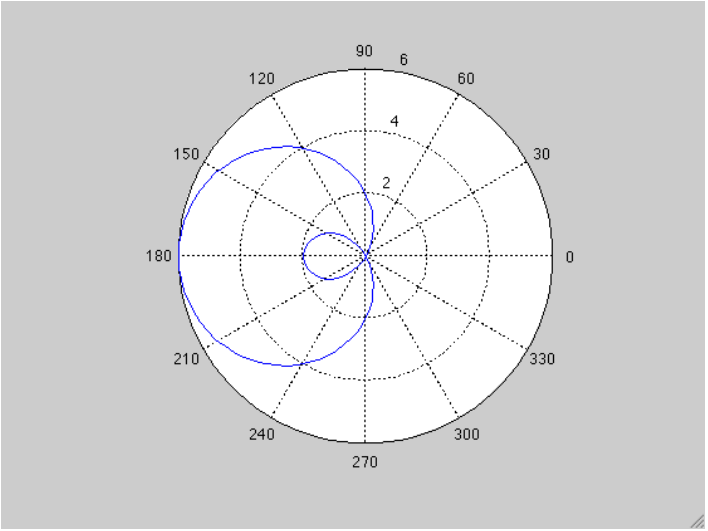
Gráficas en 2D: Curvas en coordenadas polares

Las curvas dadas en coordenadas polares se dibujan en Matlab utilizando el comando **polar**. Una curva en coordenadas polares viene dada por los puntos (r, θ), donde r varía según una función dependiente de θ.

Ejemplo: dibujar la gráfica 2D de

$$r = 2 - 4\cos(\theta) \quad -\pi \leq \theta \leq \pi$$

```
>> theta=linspace(-pi, pi, 100);
>> r=2-4*cos(theta);
>> polar(theta, r);
```



6. Gráficas en 3D

Gráficas en 3D: funciones de dos variables

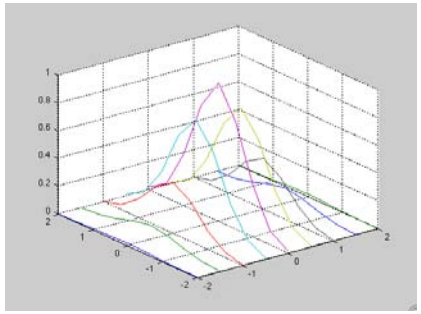
Pueden utilizarse varios comandos para mostrar funciones de dos variables: **plot3**, **mesh** y **surf**
Para generar el mallado (cto. de puntos del espacio), se utiliza el comando **meshgrid**

Ejemplo:

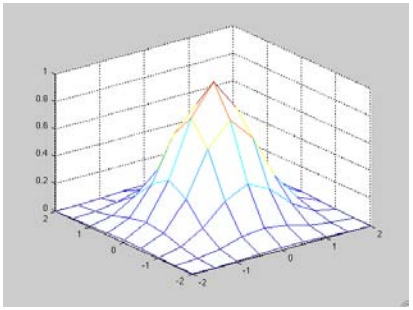
$$z = e^{-(x^2+y^2)} \quad para \quad -2 \le x \le 2, -2 \le y \le 2$$

```
>> [x, y]=meshgrid(-2:0.5:2, -2:0.5:2);
>> z=exp(-x.^2-y.^2);
>> grid on;
```

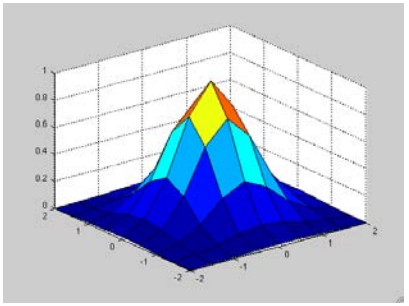
>>plot3(x,y,z);



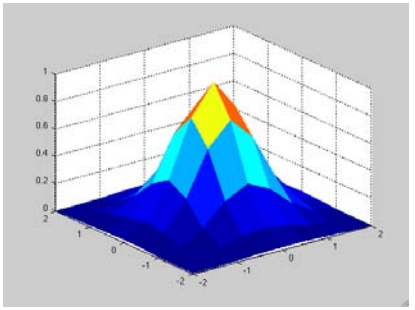
>>mesh(x,y,z);



>>surf(x,y,z);



>> surf(x,y,z), shading flat;



6. Gráficas en 3D

Gráficas en 3D: curvas paramétricas

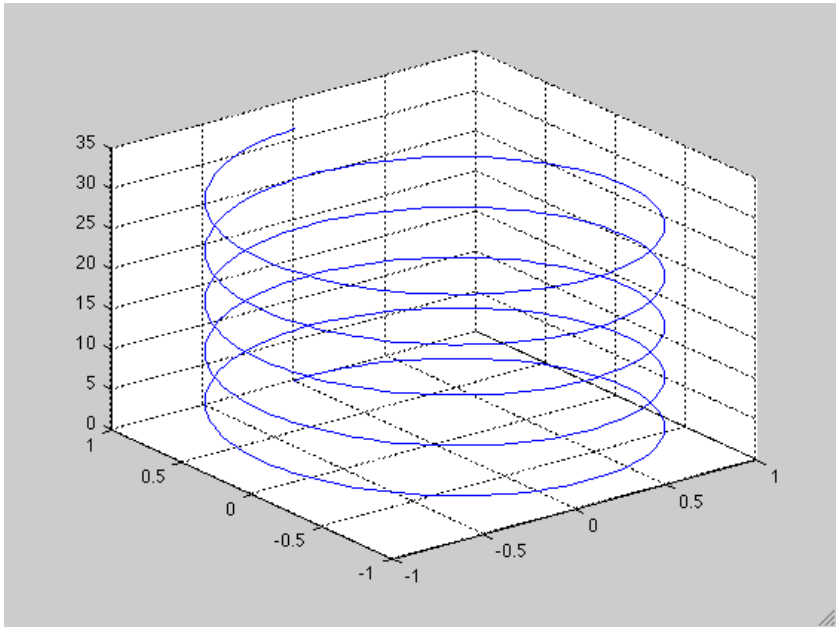
Los comandos **plot** y **comet** se puede extender a 3 dimensiones con los comandos **plot3** y **comet3**.

El siguiente ejemplo dibuja la gráfica de una espiral en tres dimensiones:

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

```
>> t=linspace(0, 8*pi, 3000);
>> plot3(sin(t),cos(t),t);
>> grid on;
```

```
>> t=linspace(0, 8*pi, 3000);
>> comet3(sin(t),cos(t),t);
>> grid on;
```



6. Gráficas en 3D

Gráficas en 3D: curvas de nivel

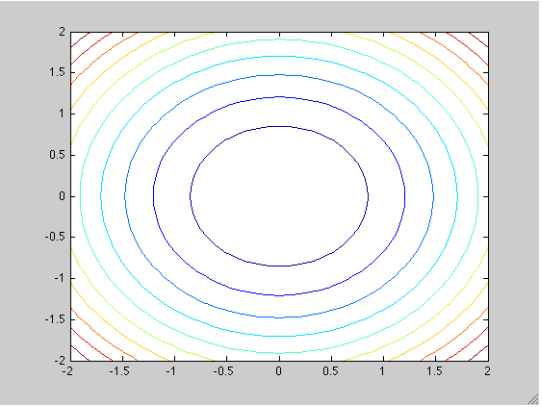
Dada una función z=f(x,y), pueden representarse las curvas sobre el plano dadas por f(x,y)=K, donde K es una constante. Se utilizan las funciones **contour** y **contour3.**

Ejemplo: dibujar la gráfica de la siguiente función utilizando curvas de nivel

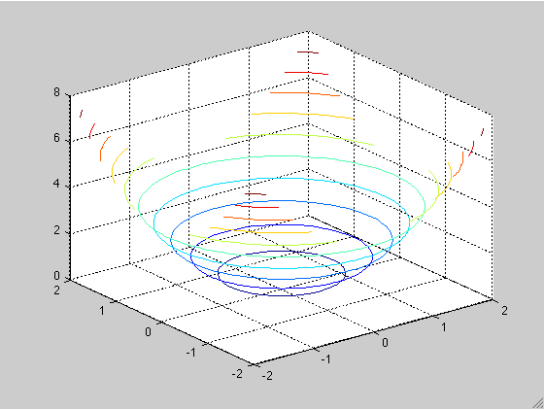
$$z = x^2 + y^2 \quad para \quad -2 \le x \le 2, -2 \le y \le 2$$

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

```
>>contour(x,y,z,10);
```



```
>>contour3(x,y,z,10);
```



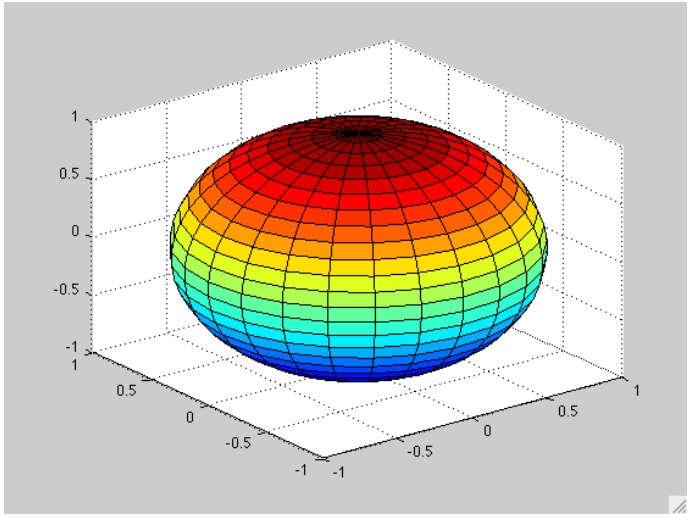
6. Gráficas en 3D

Gráficas en 3D: superficies en el espacio

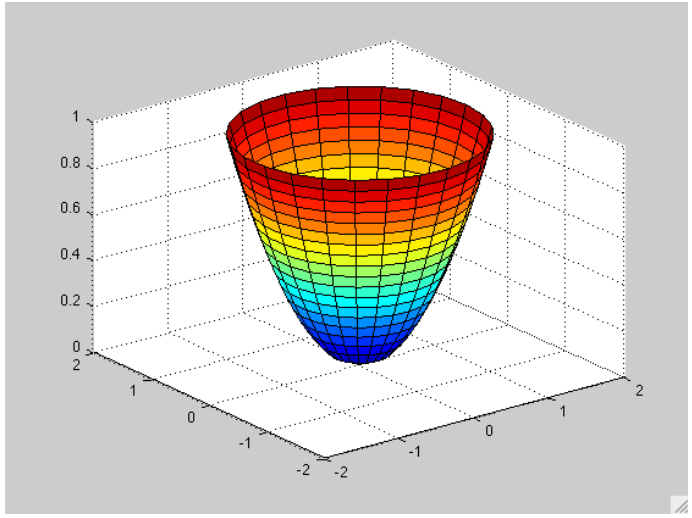
Matlab cuenta con varios comandos para reprentar superficies en el espacio: **sphere**, **cylinder**. Este último comando permite obtener gráficas de diferentes superficies de revolución.

Ejemplo: gráficas de una esfera y un cilindro

```
>> sphere(25);
```



```
>> t=linspace(0,2,20);
 >> r=sqrt(t), cylinder(r,25);
```



6. Gráficas en 3D

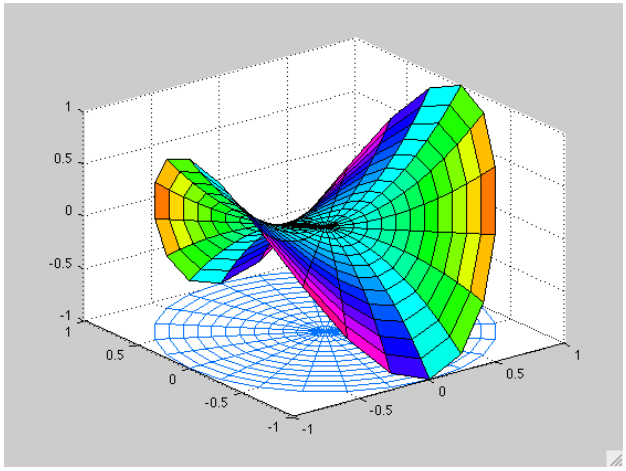
Gráficas en 3D: funciones complejas

Las funciones complejas de variable compleja se pueden representar mediante el comando **cplxmap.**

Ejemplo: gráfica de la función

$$f(z) = z^2$$

```
>> z=cplxgrid(12);
>> cplxmap(z, z.^2);
```



Puede encontrarse mas información ejecutando el comando **cplxdemo**

6. Gráficas estadísticas

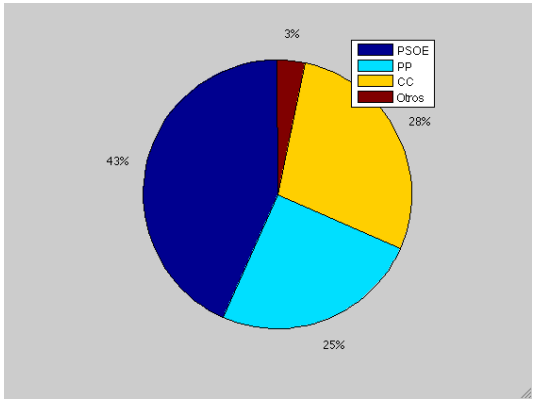
Gráficas estadísticas:

Matlab permite representar datos estadísticos mediante diagramas de sectores (**pie**, **pie3**), diagramas de Pareto (**pareto**) o diagramas de barras (**bar**, **barh**, **bar3**, **bar3h**)

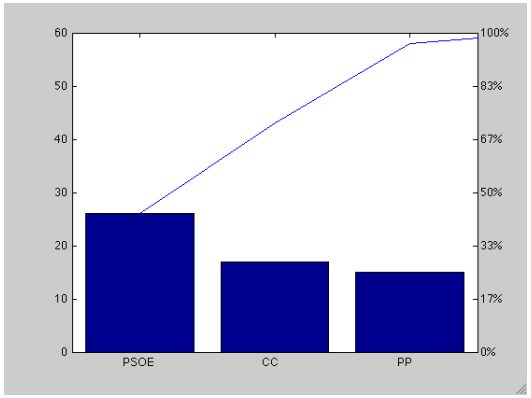
Ejemplo: representar los resultados de las últimas elecciones al Gobierno de Canarias. En escaños, el PSOE obtuvo 26, el PP 15, CC 17 y otros 2.

```
>> x=[26 15 17 2];
>> leyenda={'PSOE'; 'PP'; 'CC'; 'Otros'}
```

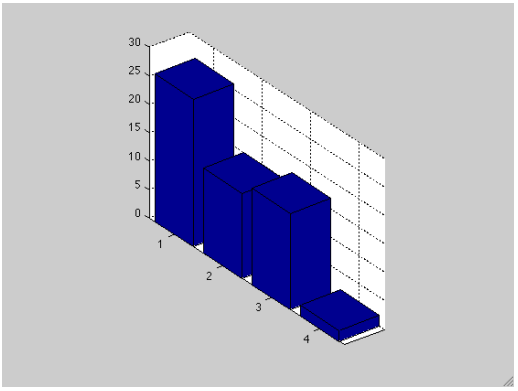
>> pie(x), legend(leyenda);



>> pareto(x, leyenda);



>>bar3(x);



6. Gráficas

Cómo guardar gráficos en un fichero:

Las gráficas pueden almacenarse en diversos formatos con el comando

Saveas (gcf, 'nombredelaimagen', formato);

Formato:

- Jpg
- eps
- bmp
- tif

7. Cómo hacer un programa en Matlab

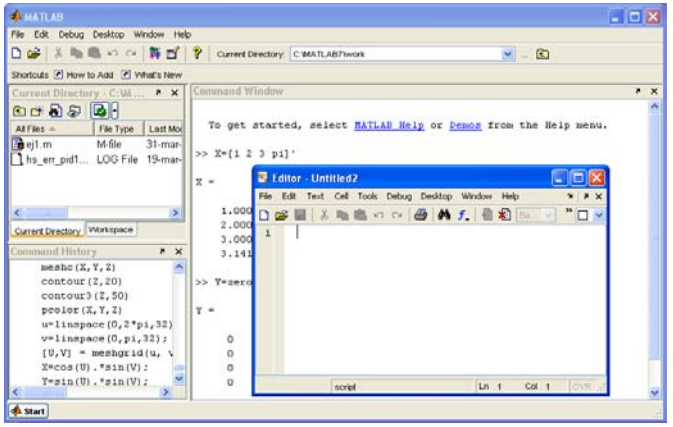
Es posible realizar un programa en Matlab tal como se hace en otros lenguajes como el basic, pascal o el lenguaje C.

Es necesario utilizar un editor para escribir el código, y guardar el fichero con extensión *.m

Se puede ejecutar el programa desde Matlab simplemente escribiendo el nombre del archivo que fue creado.

Es posible abrir programas con
la terminación *.m desde Matlab,
en el menú file, open m.file.

Para empezar, no declararemos variables



7. Cómo hacer un programa en Matlab

Sentencias de lectura y escritura

Escribir “Introduzca un valor”
Leer a

```
a=input(‘Introduzca un valor);
```

Como puedes ver, la instrucción de lectura ya permite incorporar un mensaje

Escribir “El resultado es “a “ metros”

```
fprintf(‘El resultado es %d metros',a);
ó
disp(‘El resultado es lo que sigue ’);
```

%d indica que se escribe una variable entera

Para una variable real usaremos %f

7. Cómo hacer un programa en Matlab

Sentencia de asignación

a ⟵ 32+4

b ⟵ a – 4*34

```
a=32+4;
b=a-4*34;
```

La asignaciçon de un valor a una variable se realiza con el carácter ‘=‘.

Por otro lado a la hora de comparar utilizaremos ‘==‘

7. Cómo hacer un programa en Matlab

Flujo de control: CONDICIONAL

Si *condición* entonces
  ...
Fin si

```
if condición1
   sentencia-1
end
```

```
if condición1
   sentencia-1
else
   if condición2
      sentencia-2
   else
      sentencia-3
end
```

según valor

```
switch valor
  case v1
    sentencia-1,...,sentencia-n

  case {v2,v3,v4}
    sentencia-a,...,sentencia-na
  ...

  otherwise
    sentencia-b,...,sentencia-nb

end
```

7. Cómo hacer un programa en Matlab

Flujo de control: BUCLES

Para x desde n_inicial hasta n_final hacer
  ...
Fin para

```
for x=n_inicial:n_final
   sentencias
end
```

Mientras condición hacer
  ...
Fin mientras

```
while condición
   sentencias
end
```

break provoca la salida del bucle, o del último bucle si están anidados.

continue pasa el control a la siguiente iteración del bucle for o while en el cual aparece evitando cualquier sentencia del cuerpo del bucle.

return causa un retorno normal al entorno de llamada.

7. Cómo hacer un programa en Matlab

Predicados.- Función que permite expresar que una condición se verifica o no en un entorno dado.

Predicados
any(x).- (fv) devuelve cierto si algún elemento del vector x no es cero
all(x).- (fv) devuelve cierto solo si los elementos de x son distintos de cero
find(x).- (fv) devuelve las posiciones de los elementos distintos de cero del vector x
exist(var).- devuelve cierto si la variable existe
isnan(A).- devuelve cierto donde vale NaN y cero donde no
finite(A).- devuelve cierto en los valores finitos y 0 donde no lo sea
isempty(A).- devuelve cierto si es una matriz vacía
…

Operadores relacionales

Operadores relacionales	
<	menor que
>	mayor que
<=	menor o igual
>=	mayor o igual
==	igual que
~=	distinto que

Operadores lógicos

Operadores lógicos	
&	y
|	o
~	no

7. Cómo hacer un programa en Matlab

La construcción try…catch…end permite gestionar los errores que se pueden producir en tiempo de ejecución. Su forma es la siguiente:

try

```
try
    sentencias1
    ...
catch
    sentencias2
    ...
end
```

En caso de que durante la ejecución del bloque *sentencias1* se produzca un error, el control de la ejecución se transfiere al bloque *sentencias2*.

Si la ejecución transcurriera normalmente, *sentencias2* no se ejecutaría nunca. MATLAB dispone de la función ***lasterr*** que devuelve una cadena de caracteres con el mensaje correspondiente al último error que se ha producido.

8. Funciones

El usuario de Matlab puede definir sus propias funciones y asignarle el nombre que quiera.

Definir una función propia consiste sencillamente en la creación de un fichero m que ha de tener por nombre **el mismo nombre que el de la función**

Ejemplo: Crear la función de nombre sumaycuadrado que calcule la media y la suma de los cuadrados de los datos dados por las componentes de un vector

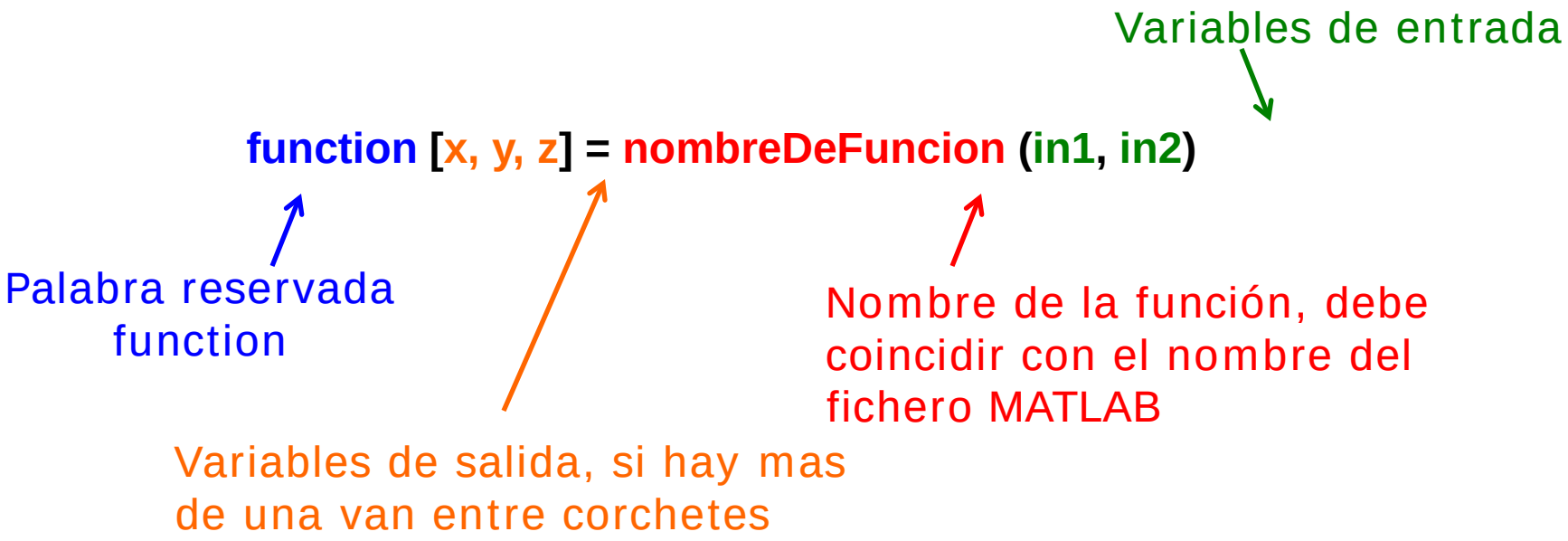
```
% sumaycuadrado: calcula la suma y la suma de los
% cuadrados de un vector dado
%
% [media, cuadrado]= sumaycuadrado (x)

function [media, cuadrado]= sumaycuadrado (x)

        n=length(x);
        media=sum(x)/n;
        cuadrado=sum(x.^2);
```

Uso

```
>> [m, c]=sumaycuadrado ( [1 3 4 5 ])
```

8. Funciones

Algunos detalles de las funciones



Todos los comentarios que se encuentren antes de la palabra reservada función, aparecerán al teclear el comando **help nombreDeFuncion**

8. Funciones

EJEMPLOS

```
function y=ordenacion(x)
temp=0;
for j=1:(length(x)-1)
   for k=j+1:(length(x))
      if (x(j)>x(k))
         temp=x(j);
         x(j)=x(k);
         x(k)=temp;
      end
   end
end
y=x;
```

```
function y=ordenacion(x)
temp=0;
for j=1:(length(x)-1)
   for k=j+1:(length(x))
      if (x(j)>x(k))
         temp=x(j);
         x(j)=x(k);
         x(k)=temp;end, end, end
y=x;
```

```
function y=ordenacion(x)
temp=0;
for j=1:(length(x)-1)
   for k=j+1:(length(x))
      if (x(j)>x(k))
         disp([num2str(x(j)),'>',num2str(x(k))]);
         temp=x(j);
         x(j)=x(k);
         x(k)=temp;
      end
   end
end
y=x;
```

¿Qué realiza el siguiente código?

EJEMPLOS

for k=1:4, y=x(k+2)-x(k), end

k=1:4, for k, y=x(k+2)-x(k), end

k=1, while k~=4, y=x(k+2)-x(k), k=k+1, end

¿Funcionan todos igual, muestran el mismo resultado?

9. Ficheros

Comandos:

Funciones para trabajar con ficheros	
[fi,texto] = fopen('filename','c') .	Abre el fichero *"filename"* en modo *"c"* (r: lectura, w: escritura, a: añadir, r+: lectura-escritura)
st = fclose(fi). **st = fclose('all')**	Cierra el fichero Cierra los ficheros abiertos
[var1,var2,...] = fscanf(fi,'cadena de control',size).-	Lee de fichero
fprintf(fi,'cadena de control',var1,var2,...) **fprintf(fi,'El determinante es: %lf10.4\n',n)**	Escribe en fichero
what	ficheros *.m en el directorio actual
delete *filename*	Borra el fichero llamado *filename*
copyfile(sc,dst)	Copia el fichero sc en el fichero dst
type file.txt	Muestra en pantalla el contenido del fichero de texto file.txt
which func	Localiza una función llamada *func*
lookfor palabra	Busca *palabra* en todas las primeras líneas de los ficheros *.m

Referencias

- Tutorial Matlab
- Matemáticas avanzadas para Ingenieros