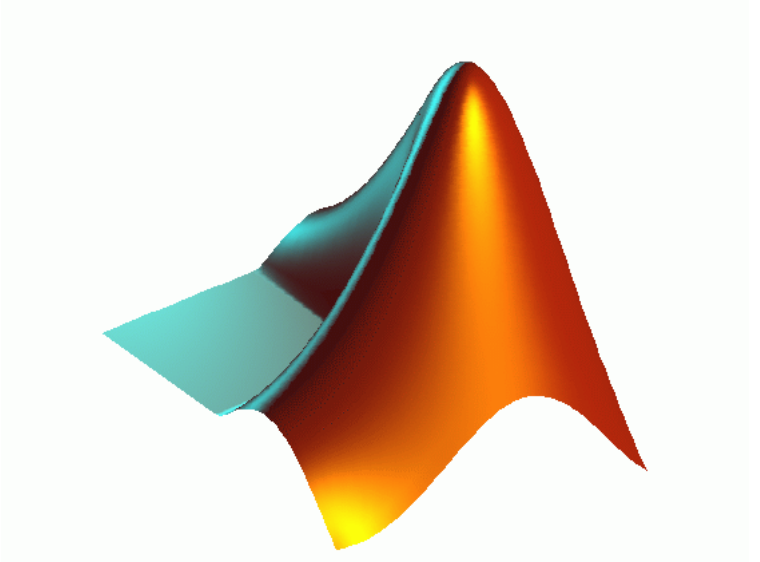


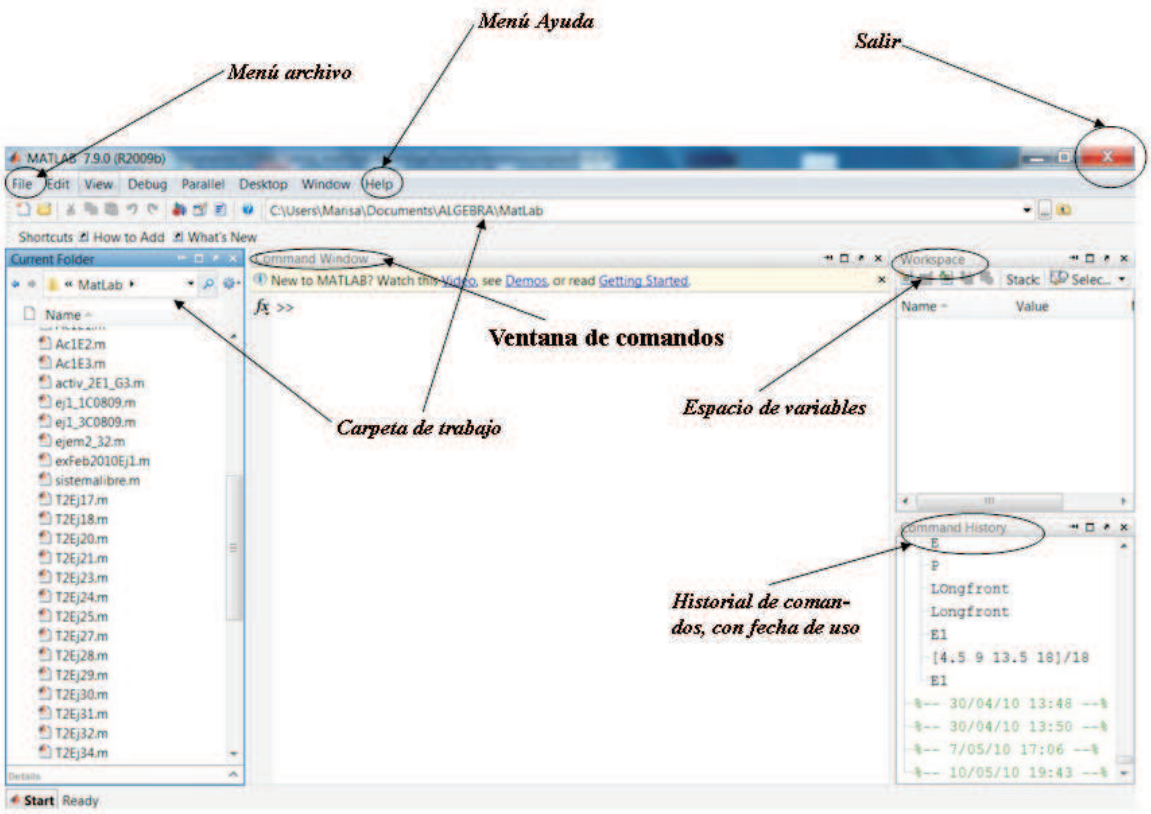
Manual de uso de Matlab

Curso 2010-2011.

Índice

1. Introducción

Este es el aspecto que presenta la versión R2009-b de MatLab, que será la que utilizaremos este curso:



En la ventana de comandos es donde podemos introducir las distintas expresiones para que MatLab las evalúe.

Para realizar los cálculos elementales con MATLAB es suficiente conocer la sintaxis de las distintas operaciones:

Suma	Resta	Multiplicación	División	Potenciación
+	-	*	/	^

Las operaciones se evalúan siguiendo un orden determinado. Primero se efectúan los paréntesis, luego las potencias, después productos y cocientes y, finalmente, sumas y restas. Dentro de un mismo nivel, se realizan de izquierda a derecha.

Ejemplo 1 *Obsérvese la diferencia entre las siguientes operaciones:*

$$3^2 - 5\left(2 - \frac{3}{4}7\right); \qquad 3^2 - 5*2 - \frac{3}{4*7}$$

```
>> 3^2-5*(2-3/4*7)
ans =
```

```
   25.2500
>> 3^2-5*2-3/(4*7)
ans =
   -6.2500
```

Para borrar la ventana de comandos se utiliza la orden `clc`. Esta acción no borra de la memoria nada que haya sido creado con anterioridad.

Las órdenes que han sido escritas previamente en un fichero ASCII se van a ejecutar secuencialmente. Los ficheros que reconoce MatLab reciben el nombre de ficheros `m`, debido a que su nombre tiene extensión .m.

Para crear un fichero `.m` se pincha con el ratón `File -> New -> M-File`, o bien se pincha el primer icono de la barra de herramientas  . Los dos caminos nos llevan a un editor de texto en el que se escriben las instrucciones que se quieren ejecutar posteriormente en el área de trabajo.

El signo % permite añadir comentarios, MATLAB obviará todo lo que esté escrito a la derecha de dicho símbolo. Además, si las primeras líneas van precedidas de este símbolo, MATLAB considerará éstas como la ayuda del fichero, y cuando en el área de trabajo tecleemos `help nombre_fichero` nos devolverá este comentario.

Una vez escrito el fichero, nos situamos en la opción `File` del menú del editor, se elige la opción `Save As` y aparece una ventana donde escribiremos el nombre del fichero `nombre_fichero.m`. Las reglas para dar nombre a un fichero son las siguientes: el primer carácter del nombre debe ser una letra, nunca un número, se pueden utilizar letras, números y el guión de subrayado, nunca signos de puntuación, ni los símbolos que indican operaciones y nunca pueden contener letras acentuadas ni espacios en blanco.

Para ejecutar un fichero `.m` se escribe el nombre de dicho fichero sin extensión en el área de trabajo, y se pulsa enter, ⏎.

2. Variables

Introducir variables nos ofrece nuevas posibilidades en MATLAB. Las reglas que se utilizan para nombrar las variables son las siguientes:

- MATLAB distingue entre letras mayúsculas y minúsculas. Las variables `area`, `Area`, `AREA`, `arEa` son variables distintas.
- El nombre de una variable puede contener un máximo de 31 caracteres ignorándose los posteriores.
- El nombre de una variable debe empezar necesariamente por una letra, aunque puede contener letras números y el guión de subrayado, nunca puede contener operadores (+,*,...), espacios en blanco ni signos de puntuación.
- No deben nombrarse variables con funciones con significado específico en MATLAB, por ejemplo `cos`=3 construye una variable `cos` cuyo valor es `3`, y a partir de este momento no podríamos calcular el coseno de un ángulo hasta que no borrásemos la variable `cos`.

Ejemplo 2 *Si queremos calcular el espacio recorrido por un móvil en movimiento rectilíneo y uniforme de velocidad* $v_0 = 5$ *m/s, para distintos tiempos, es necesario actualizar la variable espacio para cada valor del tiempo:*

```
>> v0=5, t=1, s=v0*t
>> t=3        %Cambiamos el valor de t
>> s          %s no se ha actualizado
>> s=v0*t    %actualización de s
```

Obsérvese, por un lado, que en la primera línea se han definido tres variables, sin más que separarlas por comas y, por otro, que hasta que no se actualice la definición de la variable `s` su valor no cambia.

2.1. Información sobre las variables

Para obtener información sobre las variables definidas en una sesión de trabajo se utilizan las órdenes `who` y `whos`. La primera muestra las variables que tienen valores asignados, la segunda nos da además información sobre el tamaño y el tipo de dato.

```
>> who
>> whos
```

Puede observarse que MATLAB utiliza los escalares como matrices 1×1.

2.2. Cómo borrar variables

La orden `clear all` borra de la memoria todas las variables definidas hasta el momento; si a la orden `clear` se le añade una lista de variables (separadas por espacios en blanco) sólo se borrarán las variables de la lista.

```
>> clear t
>> s=v0*t
>> who
```

Como la variable `t` ha desaparecido MATLAB da un mensaje de error al recalcular `s`.

2.3. Algunas variables predefinidas en MATLAB

Algunas variables ya están definidas en MATLAB:

Nombre	**Significado**
`ans`	Almacena el último resultado no asignado a una variable
`pi`	π
`i` y `j`	Unidad imaginaria
`inf`	∞
`NaN`	No es un número

`NaN` (*Not a Number*) representa una expresión indeterminada, como puede verse en el siguiente ejemplo:

```
>> (2-2)/(3-3)
```

3. Trabajando con matrices

Como ya se ha comentado, el tipo básico de dato con el que MATLAB trabaja es la matriz, incluso los escalares son considerados como matrices 1×1, por lo que es esencial familiarizarse con esta sección.

3.1. Vectores

Los vectores se introducen escribiendo cada una de sus coordenadas entre corchetes, separadas por un espacio en blanco:

```
>> v=[1 3 pi 1/3]
```

o bien separadas por comas:

```
>> v=[1,3,pi,1/3]
```

No obstante, existen otras formas de introducir vectores, cuando sus coordenadas guardan alguna relación entre sí:

Orden	Salida
`[a:h:b]`	Vector $(a, a+h, a+2h, \ldots, a+nh)$, donde n es el mayor entero tal que $a+nh \in [a,b]$ si $h>0$ y $a+nh \in [b,a]$ si $h<0$. En este caso, los corchetes pueden sustituirse por paréntesis o incluso eliminarse
`linspace(a,b,n)`	Vector cuyas coordenadas son los puntos de una partición uniforme del intervalo $[a,b]$

```
>> v=[1:0.3:2]
>> v=(1:-0.4:-0.8)
```

Si se omite el incremento `h` MATLAB toma por defecto `h=1`

```
>> v=1:4
```

En la orden `linspace` el tercer argumento es opcional, y si no se introduce toma el valor 100:

```
>> v=linspace(0,10)
```

Ejemplo 3 *Supongamos ahora que en el ejemplo 2 queremos calcular los espacios recorridos por el móvil a velocidad* $v_0 = 5$ *m/s, para* 5 *instantes correspondientes a los 2 primeros segundos del movimiento:*

```
>> t=linspace(0,2,5)
>> v0=5
>> s=v0*t
```

Obsérvese que, como cabía esperar, el resultado del producto de un escalar por un vector es el vector de las posiciones en los instantes correspondientes.

Si nos interesa conocer las posiciones en instantes de tiempo separados por 0.3 *segundos*

```
>> t=[0:0.3:2]
>> s=v0*t
```

3.2. Matrices

Los elementos de una matriz se introducen entre corchetes. Las filas separadas mediante un punto y coma (;) y los elementos separados por espacios en blanco o comas.

```
>> A=[1 2 3; 3,1,2;1 1 0]
```

Una vez definida una matriz o un vector, se puede acceder a sus elementos o submatrices con las órdenes:

Orden	Salida
v(i)	Coordenada i del vector v
v(end)	Última coordenada del vector v
A(i,j)	Elemento de la matriz A que ocupa la posición i,j
A(:,j)	Columna j de la matriz A
A(i,:)	Fila i de la matriz A
A(v,w)	Submatriz de A que contiene las filas indicadas en las coordenadas de v y las columnas indicadas en w
A(i,:)=[ ]	Elimina la fila i de la matriz A
A(:,j)=[ ]	Elimina la columna j de la matriz A
A(:,end)	Última columna de la matriz A

Haciendo uso de estas órdenes pueden introducirse matrices y vectores elemento a elemento. Al asignarle un valor a una posición, se construye la matriz o vector de menor tamaño que contiene los elementos introducidos y hace ceros los no asignados.

```
>> B(5)=3
>> B(2,4)=5
```

O bien se puede utilizar para modificar posiciones de una matriz predefinida:

```
>> A=[1 2 3; 4 5 6]
>> A(1,2)=5
```

También puede eliminarse filas y columnas de matrices dadas.

```
>> A=[1 2 3 4 1; 3,1,2 0 2;1 1 0 1 3]
>> A(3,4)=100, A(2,5)=200
>> B=A
>> B(:,2)=[]
```

La matriz B coincide con la matriz obtenida de eliminar la columna 2 de A.

Pueden definirse ciertas matrices con las siguientes órdenes:

Orden	**Salida**
`ones(n)`	Matriz cuadrada $n \times n$ de unos.
`ones(m,n)`	Matriz $m \times n$ de unos.
`zeros(n)`	Matriz cuadrada $n \times n$ de ceros.
`zeros(m,n)`	Matriz $m \times n$ de ceros.
`eye(n)`	Matriz identidad $n \times n$.
`eye(m,n)`	Matriz $m \times n$ con unos en la diagonal principal y el resto ceros.

3.3. Definición de matrices por bloques

Dadas dos matrices A y B con el mismo número de filas, se puede definir una matriz C formada por todas las columnas de A y de B:

```
>> A=zeros(3)
>> B=eye(3,2)
>> C=[A B]
```

Análogamente, se puede definir una matriz a partir de otras dos con el mismo número de columnas:

```
>> A=eye(2,3)
>> B=ones(3)
>> C=[A;B]
```

Estas dos posibilidades pueden combinarse para formar matrices definidas por bloques:

```
>> A=[eye(3) ones(3,3);1:6;zeros(2) ones(2,1) eye(2,3)]
```

3.4. Operaciones con vectores y matrices

Si A y B son matrices con las dimensiones adecuadas y λ es un escalar, las operaciones habituales se efectúan con las siguientes órdenes:

Operación	Resultado
`A+B`	Suma A y B
`A-B`	Resta B de A
`A*B`	Multiplica A por B
`A/B`	Calcula AB^{-1}
`A\B`	Calcula $A^{-1}B$
λ`*A`	Multiplica todos los elementos de A por λ
`A^n`	Eleva la matriz A al entero n
`A.'`	Calcula la traspuesta de A
`A'`	Calcula la traspuesta de la conjugada de A

Además de las operaciones mencionadas, en MATLAB se definen otras operaciones a las que llamaremos operaciones elemento a elemento:

Operación	Resultado
λ`+A`	Suma a cada elemento de A el escalar λ
`A.*B`	Calcula una matriz que en la posición (i, j) contiene el producto $a_{ij}b_{ij}$ de los elementos que en A y B ocupan dicha posición
`A./B`	Calcula una matriz que en la posición (i, j) contiene el cociente a_{ij}/b_{ij} de los elementos que en A y B ocupan dicha posición
`A.^n`	Eleva cada elemento de la matriz A al entero n
`A.^B`	Calcula una matriz que en la posición (i, j) contiene $a_{ij}^{b_{ij}}$

3.5. Funciones que actúan sobre matrices

En MATLAB hay una colección de funciones que pueden utilizarse para obtener información y realizar cálculos. Por ejemplo, si se escribe

```
» A=eye(3,2)
```

se obtiene una matriz de tres filas y dos columnas con unos en la diagonal principal y ceros en el resto. El nombre de la función es `eye`, los argumentos de entrada son 3 y 2, la matriz resultante, que tiene por nombre A, es la salida.

Las siguientes funciones permiten obtener información sobre las matrices o vectores que tienen como argumentos de entrada

Función	Salida
`size(A)`	Vector con las dimensiones de la matriz *A*
`size(A,1)`	Número de filas de la matriz *A*
`size(A,2)`	Número de columnas de la matriz *A*
`length(v)`	Número de coordenadas del vector *v*
`length(A)`	Mayor elemento del vector `size(A)`
`rank(A)`	Rango de la matriz *A*
`det(A)`	Determinante de la matriz *A*
`trace(A)`	Traza de la matriz *A*
`inv(A)`	devuelve la inversa de `A`, aunque también puede calcularse como `A^n`
`sum(A)`	devuelve un vector fila en el que el elemento `i` contiene la suma de todos los elementos de la columna `i` de `A`
`prod(A)`	devuelve un vector fila en el que el elemento `i` contiene el producto de todos los elementos de la columna `i` de `A`
`dot(u,v)`	Producto escalar de los vectores *u* y *v*
`cross(u,v)`	Producto vectorial de los vectores (de tres coordenadas) *u* y *v*
`max(A)`	devuelve un vector fila en el que el elemento `i` contiene el máximo de todos los elementos de la columna `i` de `A`
`[m,pos]=max(A)`	devuelve `m` vector fila en el que el elemento `i` contiene el máximo de todos los elementos de la columna `i` de `A`, y el vector fila `pos` en el que almacena la posición en la que se encuentra dicho máximo.
`min(A)`	devuelve un vector fila en el que el elemento `i` contiene el mínimo de todos los elementos de la columna `i` de `A`
`[m,pos]=min(A)`	devuelve `m` vector fila en el que el elemento `i` contiene el mínimo de todos los elementos de la columna `i` de `A`, y el vector fila `pos` en el que almacena la posición en la que se encuentra dicho mínimo.
`null(A)`	Devuelve una base del subespacio de las soluciones de un sistema homogéneo
`colspace`	Proporciona, por columnas, una base del subespacio generado por los vectores columna de la matriz *A*. Dado que es una función simbólica, el argumento debe ser una variable simbólica.
`rref(A)`	Calcula la matriz escalonada reducida de la matriz *A*
`poly(A)`	Calcula el polinomio `det(x-AI)`, expresado como un vector, según potencias decrecientes.
`poly(A,x)`	Calcula el polinomio `det(x-AI)`. `x` debe ser declarada antes como variable simbólica.
`eig(A)`	Calcula los valores propios de `A`.
`[P,D]=eig(A)`	Devuelve la matriz `P` cuyas columnas son los vectores propios, y la matriz diagonal `D` formada por los valores propios. Si la matriz `A` no es diagonalizable, Matlab devuelve una matriz `P` no regular y una matriz diagonal `D` formada por los valores propios, de modo que `AP=PD`.
`[P,D]=eig(sym(A))`	Hace lo mismo que la orden `[P,D]=eig(A)`, pero con la matriz `A` en formato simbólico.
`orth(A)`	Devuelve una matriz cuyas columnas forman una base ortonormal del subespacio engendrado por las columnas de A

Ejemplo 4 *Calcula el polinomio* $\det(A-xI)$ *para* $A=\begin{pmatrix} 1 & 2 & 5 \\ 2 & 1 & -1 \\ 3 & 0 & -3 \end{pmatrix}$

Solución

```
>> A=[1 2 5; 2 1 -1; 3 0 -3]
A =
       1             2             5
```

```
       2              1             -1
       3              0             -3
>> p=poly(A)
p =
       1              1            -24             12
```

y se obtiene que $\det(xI-A) = x^3+x^2-24x+12$. También se puede escribir:

```
>> syms lambda
>> A=[1 2 5; 2 1 -1; 3 0 -3]
A =
     1     2     5
     2     1    -1
     3     0    -3
>> p=poly(A,lambda)
p =
lambda^3 + lambda^2 - 24*lambda + 12
```

Obviamente, las funciones matemáticas habituales también están predefinidas en MATLAB, con la única particularidad de que actúan sobre vectores o matrices elemento a elemento.

MATLAB	**Función**	**MATLAB**	**Función**
`exp(x)`	e^x	`abs(x)`	$\|x\|$
`log(x)`	$\ln(x)$	`fix(x)`	Redondeo hacia cero
`log10(x)`	$\log_{10}(x)$	`floor(x)`	Redondeo hacia $+\infty$
`log2(x)`	$\log_2(x)$	`ceil(x)`	Redondeo hacia $-\infty$
`sqrt(x)`	$\sqrt{x}$	`round(x)`	Redondeo hacia el entero más próximo
		`rem(m,n)`	resto de dividir *m* entre *n*

y las funciones trigonométricas:

MATLAB	**Fun.**	**MATLAB**	**Fun.**	**MATLAB**	**Func.**	**MATLAB**	**Fun.**
`sin(x)`	sen(x)	`asin(x)`	asen(x)	`sinh(x)`	senh(x)	`asinh (x)`	asenh(x)
`cos(x)`	cos(x)	`acos(x)`	acos(x)	`cosh(x)`	cosh(x)	`acosh (x)`	acosh(x)
`tan(x)`	tan(x)	`atan(x)`	atan(x)	`tanh(x)`	tanh(x)	`atanh (x)`	atanh(x)
`cot(x)`	cot(x)	`acot(x)`	acot(x)	`coth(x)`	coth(x)	`acoth (x)`	acoth(x)
`sec(x)`	sec(x)	`asec(x)`	asec(x)	`sech(x)`	sech(x)	`asech (x)`	asech(x)
`csc(x)`	csc(x)	`acsc(x)`	acsc(x)	`csch(x)`	csch(x)	`acsch (x)`	acsch(x)

4. Operaciones básicas con números complejos

En MATLAB, por defecto, las letras *i* ó *j* representan la unidad imaginaria. Obsérvese cómo se introduce el complejo $1+i$

```
» z=1+i
» z=1+j
```

Las operaciones con complejos se realizan igual que con números reales:

Suma	Resta	Multiplicación	División	Potenciación
`+`	`-`	`*`	`/`	`^`

Ejemplo 5 *Calcular los siguientes complejos en forma binómica:*

$$(3+5i)(4-i), \qquad \frac{3-i}{4+5i}, \qquad (1+\sqrt{3}i)^3$$

Solución

```
» (3+5i)*(4-i)
» (3-i)/(4+5i)
```

Cuando la parte imaginaria del complejo se involucra alguna función u operación, debe escribirse `*` entre la parte imaginaria y la unidad imaginaria:

```
>> (1+sqrt(3)i)^3     %Devuelve un mensaje de error
>> (1+sqrt(3)*i)^3
>> 1+(1-1/3)i         %Devuelve un mensaje de error
>> 1+(1-1/3)*i
```

Otras funciones útiles para operar con complejos son las siguientes:

Orden	**Salida**
`real(z)`	Parte real de `z`
`imag(z)`	Parte imaginaria de `z`
`abs(z)`	Módulo de `z`
`conj(z)`	Conjugado de `z`
`angle(z)`	Argumento que se encuentra en el intervalo $]-\pi,\pi]$

Si las funciones anteriores trabajan sobre una matriz, devuelven otra matriz del mismo tipo que es el resultado de evaluar la función al actuar sobre el elemento.

Todas estas funciones, excepto angle, pueden actuar tanto sobre variables simbólicas como numéricas.

5. Programando bucles y condicionales

5.1. Operadores relacionales

A menudo, según sean los datos que se utilizan, es necesario tomar una decisión sobre las órdenes a ejecutar, por lo que resultan de gran utilidad los operadores y bucles que se mencionan a continuación.

MATLAB utiliza los operadores relacionales que se describen en la tabla adjunta:

Operador	Descripción
$<$	Menor
$<=$	Menor o igual
$>$	Mayor
$>=$	Mayor o igual
$==$	Igual
$\sim=$	Distinto
$\vert$	Operador lógico ó
&	Operador lógico y

El símbolo $\sim$ se obtiene pulsando Alt Gr y 4 simultáneamente y un espacio en blanco.

Es importante distinguir el símbolo $=$ de asignación de un valor a una variable, del símbolo $==$ que compara el valor de dos variables.

Los operadores relacionales permiten construir **expresiones lógicas** cuya estructura es

```
expresion1 OpR expresion2
```

donde `OpR` es un relacional y `expresion1` y `expresion2` son números, matrices (de igual dimensión) o cadenas de caracteres. La respuesta de estas expresiones lógicas es `1` si son verdaderas y `0` cuando son falsas.

Ejemplo 6 *Construiremos una variable x que almacene el complejo* $1+i$*, la compararemos con dicho complejo y con el complejo* $1+2i$*.*

```
» x=1+i
x =
   1.0000 + 1.0000i
» x==1+i
ans =
     1
» x==1+2i
ans =
     0
```

En la primera línea se asigna a `x` el valor `1+i`, en la siguiente se compara `x` con `1+i`, y nos devuelve un `1` debido a que la proposición lógica es cierta. En la tercera línea se compara la variable `x` con 1+2i y nos devuelve un `0` debido a que la proposición lógica es falsa.

5.2. Estructuras if-elseif-else-end

En ocasiones se quiere ejecutar un conjunto de órdenes solo en el caso de que se verifique cierta condición. Esto se consigue con las combinaciones de órdenes siguiente `if-end`, `if-else-end` e `if-elseif-else-end`.

La estructura `if-elseif-else-end` se utiliza como sigue:

```
if expresión lógica 1
     conjunto de órdenes 1
elseif expresión lógica 2
     conjunto de órdenes 2
elseif expresión lógica 3
     conjunto de órdenes 3
               .
               .
else
     conjunto de órdenes
end
```

El *conjunto de órdenes 1* se ejecuta si la *expresión lógica 1* es verdadera, el *conjunto de órdenes 2* se ejecuta si la *expresión lógica 2* es verdadera, etc. Cuando todas las expresiones lógicas son falsas, se ejecuta el *conjunto de órdenes* que sigue a `else` y *la expresión lógica 1* es falsa.

La orden `else` puede aparecer o no. También puede aparecer sólo la combinación `if-end` o la combinación `if-else-end`.

Ejemplo 7 *Escribir un fichero* `m` *llamado* `ejem1_6.m` *que calcule la imagen de un complejo* $z=x+yi$ *por la función:*

$$f(x+yi)=\begin{cases} 0 & \text{si} \quad x=y=0 \\ y+xi & \text{si} \quad xy<0 \\ 1 & si \quad x=0,\ y<0 \\ y-xi & \text{si} \quad xy>0 \\ y & \text{si} \quad x=0,\ y>0 \end{cases}$$

```
z=3-i;
if (imag(z)==0)&(real(z)==0)
    imagen=0;
elseif real(z)*imag(z)<0
    imagen=imag(z)+real(z)*i;
elseif (real(z)==0)&(imag(z)<0)
    imagen=1;
elseif real(z)*imag(z)>0
    imagen=imag(z)-real(z)*i;
elseif (real(z)==0)&(imag(z)>0)
    imagen=imag(z);
end
imagen
```

5.3. La estructura for-end

Hasta ahora las órdenes se ejecutaban de forma secuencial, pero pueden existir procesos en los que un conjunto de órdenes se deban ejecutar varias veces, para ello existen los bucles `for-end`. La sintaxis es la siguiente:

```
for k=x
    conjunto de órdenes
end
```

donde `k` es una variable y `x` es un vector.

Al llegar el programa a la orden `for` la variable `k` toma como valor la primera coordenada del vector `x` y se ejecuta el `conjunto de órdenes`. A continuación `k` toma como valor la segunda coordenada de `x` y se vuelve a ejecutar `conjunto de órdenes`. El bucle se repite tantas veces como coordenadas tenga `x`. Cuando `k` ha recorrido todas las posiciones de `x` el programa seguirá con las órdenes posteriores a `end`. En el caso de que `x` sea una matriz, la variable `k` tomará como valor las distintas columnas de `x`.

Por ejemplo si queremos calcular el valor de k^2, cuando $k = 4, 5, 6$, escibiríamos:

```
>> for k=[4,5,6]
 k^2
end
```

la salida sería

```
ans =
    16
ans =
    25
ans =
    36
```

5.4. Bucles while-end

Su sintaxis es la siguiente:

```
while expresión lógica
    conjunto de órdenes
end
```

Hacen que un conjunto de órdenes se ejecute mientras una expresión lógica sea verdadera.

Por ejemplo, si escribimos en un fichero:

```
x=1
while x<=11
    x=2*x
end
```

Al ejecutarlo la salida obtenida es

```
x =
     1
x =
     2
x =
     4
x =
     8
x =
    16
```

Cuando $x = 16$ ya no se verifica la condición y el bucle termina.

6. Ficheros *function*

Dentro de la organización de un programa es muy común la realización de tareas que pueden servir para diferentes programas o simplemente la separación en etapas del programa global que se pueden abordar independientemente. Una de las formas de realizar esta división en MATLAB es a través de las `function`.

La característica de la `function` respecto a los ficheros de órdenes es la utilización de `argumentos`. Su funcionamiento es análogo a muchas de las órdenes del MATLAB, por ejemplo, cuando nosotros ejecutamos

```
>> x = sqrt(16)
x =
    4
```

la orden `sqrt` funciona como una `function` con `argumento de entrada (16)` y obtenemos un `argumento de salida` que asociamos a `x`.

Las `function` se construyen en ficheros **.m** . Se distinguen de los guiones en la primera orden en donde se deben especificar los argumentos

```
function   [Argumento(s) de Salida] = nombrefuncion (Arg. Entrada)
%          líneas de comentarios
%          que aparecen al ejecutar
%           help nombrefuncion

           Órdenes que hacen los cálculos
                 ⋮
```

Por norma los nombres de la function y del fichero coincidirán. Es decir, el fichero lo llamamos `nombrefuncion.m`

Para llamar o ejecutar la `function` se realiza como las órdenes normales

» ***[Argumento(s) de Salida] = nombrefuncion (Arg. Entrada)***

Los `argumentos` tanto de entrada como salida pueden ser varios y se separan por comas. Incluso puede que no los haya. Los `nombres de los argumentos` en el fichero `function` son variables ficticias puesto que esos nombres van a ser sustituidos por los utilizados en la llamada a la `function` que son las variable verdaderamente reales.

Ejemplo 8 *Construir una `function raices.m` que calcule las raíces de un polinomio de segundo grado* ax^2+bx+c

SOLUCIÓN:

Se escribe en el fichero `raices.m`

```
function [x1, x2] = raices(a,b,c)
%
%     Función que calcula en x1 y x2 las raíces
%        de un polinomio de segundo grado
%          a x^2 + b x + c = 0
%
disc=sqrt(b*b-4*a*c)
x1 = (-b + disc)/(2*a)
x2 = (-b - disc)/(2*a)
```

Una vez guardado el fichero `raices.m`. Se puede llamar

```
>> [x,y] = raices(1,-3,2)
x =
    2
y =
    1
>> help  raices

     Función que calcula en x1 y x2 las raíces
        de un polinomio de segundo grado
          a x^2 + b x + c = 0
```

Debe mencionarse que a las variables de salida se les puede asignar un nombre cualquiera. En este caso, se les han asignado los nombres `x` e `y`. Si se escribe `x1` o `y1`, estas variables no existen y lo mismo sucede con la variable `disc`. Estas variables sólo están activas dentro de la función `raices`.

7. Cálculo simbólico

7.1. Creamos objetos simbólicos y operamos con ellos

Función	Salida
`syms`	crea variables simbólicas
`sym(x)`	devuelve x simbólicamente

Si se utiliza la instrucción syms para declarar variables, estas se introducen con un espacio en blanco entre ellas. Por ejemplo, `syms s t` declara simbólicas las variables `s` y `t`. Si al finalizar la lista se escribe real, MatLab considerará que estas variables no tienen parte imaginaria, en caso contrario se presuponen complejas.

Pueden crearse objetos simbólicos y aplicar las funciones habituales:

```
» syms x y real %crea las variables simbólicas reales x e y.
» f=(x+i*y)^3 %crea la variable simbólica f.
» u=imag(f)
» v=real(f)
```

En la instrucción anterior las variables `x` e `y` tienen el sentido de variables independientes habitual en matemáticas. La variable f sería la variable dependiente, y, como puede observarse, no es necesario declararla.

Con el comando `sym` se pueden obtener constantes simbólicas:

```
» x=sym(pi)
```

y también puede aplicarse a matrices, en cuyo caso trabaja elemento a elemento:

```
» A=sym([1 2/3;pi sqrt(2)])
```

7.2. Cómo borrar variables simbólicas

La orden `clear`, utilizada sobre variables simbólicas, presenta algunas limitaciones. Por ejemplo, si la variable está declarada como simbólica real, al borrarla con `clear`, queda en memoria su carácter real. Veamos un ejemplo:

```
>> syms x real
>> imag(x)%Será 0 por ser real
ans =
0
>> clear x
>> x % Aparentemente borrada
??? Undefined function or variable 'x'
>> syms x
>> imag(x) % vemos que es 0, luego sigue siendo real
ans =
0
```

Para borrar el carácter real de una lista de variables escribiremos `syms lista clear`. Por ejemplo

```
>> syms x y real
>> syms x y clear % Son simbólicas pero ya no son reales.
>> imag(x),imag(y)
ans =
(i*conj(x))/2 - (i*x)/2
ans =
(i*conj(y))/2 - (i*y)/2
>> clear x y % ya quedan perfectamente borradas
```

Si queremos borrar todas las variables de golpe podemos seguir utilizando `clear all`, pues esta orden también borra el carácter real de las variables.

7.3. Sustituciones en una expresión simbólica y conversión a numérico

Función	**Salida**
`subs`	substituye una expresión
`compose(f,g,x,y,z)`	compone dos funciones simbólicas, f y g, donde la variable independiente de la composición será *z* y las independientes de f y de g serán respectivamente x e y. Las variables x, y y z son opcionales
`double`	obtiene el valor numérico
`digits`	especifica el número de dígitos
`vpa`	evalúa una expresión con la precisión deseada

Ejemplo 9 *Construir* $f = ax^2 + bx + c$ *y sustitúyase x por* s^2. *Haciendo* $a = 1$, $b = 2$ *y* $c = \sqrt{3}$, *obténgase el valor de* f *para* $s = 1$ *y* $s = 4$.

```
» syms x a b c real
» f=a*x^2+b*x+c
» syms s real
» g=subs(f,x,s^2) %en f sustituye x por s^2
» h=subs(g,{a,b,c},{1,2,sqrt(3)})  %substitución múltiple.
» k=subs(h,s,[1;4])   %sustituye s por una matriz.
» a=1;b=2;c=3;f=subs(f) %Cambia en f a, b, c, por los valores dados.
                              a,b,c dejan de ser simbólicos.
```

Ejemplo 10 *Obtener el valor de* $f(x,y) = \sqrt[3]{2x+5y+3}$ *en los puntos* $(0,0)$, $(0,1)$, $(0,2)$, *y* $(0,3)$.

```
» syms x y
» f=(2*x+5*y+3)^(1/3)
» v=[0 1 2 3];
» val=subs(f,{x,y},{0*v,v}) %sustituye (x,y) por (0,v(i))
» val=double(val) %pasa val a variable numérica
```

En la parte básica, MATLAB utiliza la aritmética de punto flotante y trabaja con 16 dígitos. Por este motivo, si se manejan números de más decimales, lo que sucede siempre con números irracionales, en cada operación se produce un error llamado de redondeo. En cálculo simbólico no se produce este tipo de error pues MATLAB no realiza cálculos numéricos, trabaja simbólicamente. Sí puede producirse un error de redondeo cuando se usa la instrucción `double` para convertir un resultado simbólico a numérico.

7.4. Límites, derivadas e integrales simbólicas

MatLab calcula límites, suma de expresiones, derivadas e integrales de variables simbólicas. Al hacerlo, si no especificamos otra cosa, considera como variable independiente la variable preferente de la expresión simbólica con la que está trabajando. La variable preferente en una expresión simbólica es la letra `x`. Si ésta no interviene en la expresión , será el carácter más próximo a x en el orden lexicográfico que no sea ni la i ni la j. Por esto, no conviene omitir la variable respecto de la cual se va a realizar la operación.

Orden	Salida
`limit(f,x,a)`	Calcula $\lim_{x\to a} f$, siendo x variable simbólica; a puede ser la variable `inf`.
`limit(f,x,a,'right')`	Calcula $\lim_{x\to a^+} f$, siendo x variable simbólica.
`limit(f,x,a,'left')`	Calcula $\lim_{x\to a^-} f$, siendo x variable simbólica.
`symsum(f,n,a,b)`	Calcula $\sum_{n=a}^{b} f$, siendo f una variable simbólica dependiente de la variable simbólica n; a y b son los límites donde varía n. (b puede ser la variable `inf`)
`symsum(f,n)`	Calcula $\sum_{n=0}^{\infty} f$, siendo f una variable simbólica dependiente de la variable simbólica n
`diff(f,u,n)`	Halla la derivada de orden `n` (n número entero) respecto a `u`
`diff(f,u)`	Halla la derivada respecto a `u`
`int(f,s)`	Calcula una primitiva de f respecto a s.
`int(f)`	Calcula una primitiva de f respecto a la variable por defecto.
`int(f,s,a,b)`	Calcula la integral definida respecto a s entre a y b.
`int(f,a,b)`	Calcula la integral definida respecto a la variable por defecto entre a y b
`int(f,s,a,b)`	Calcula la integral definida respecto a s entre a y b.
`taylor(f,n,s,a)`	Calcula el desarrollo de Taylor de f en potencias de `s-a` de orden `n-1`. `n`, `s` y `a` pueden omitirse . Si se omite `s`, considera como variable independiente la preferente. Si se omite `a`, interpreta `a=0`. Si se omite `n`, toma `n=5`.

Conviene señalar que `diff` puede actuar sobre una matriz. También hay un operador `diff` que actúa sobre variables numéricas (obviamente no calcula la derivada).

7.5. Manipulación de expresiones simbólicas

En una expresión simbólica f se pueden realizar, entre otras, las siguientes transformaciones:

Función	Salida
`collect(f)`	Agrupa términos mostrando la expresión como un polinomio en la variable preferente
`collect(f,'s')`	Agrupa términos mostrando la expresión como un polinomio en la variable s
`expand(f)`	Desarrolla la expresión
`factor(f)`	Factoriza la expresión
`simplify(f)`	Simplifica la expresión
`simple(f)`	Busca la forma más simple de la expresión f. Prueba distintas órdenes de simplificación y muestra la forma de expresión de f con menor número de caracteres.
`pretty(f)`	Muestra f en forma parecida a la tipografía matemática

8. Solución de ecuaciones

En esta sección vamos a resolver ecuaciones simbólicas mediante la función **solve**. Mediante esta instrucción MATLAB obtiene soluciones de ecuaciones. MATLAB busca soluciones en el campo de los números complejos y cuando no puede obtener soluciones simbólicas intenta obtener soluciones numéricas.

Función	Salida
`solve(p)`	encuentra soluciones de la ecuación `p=0`

Ejemplo 11 *Calcúlense todas las raíces del polinomio* z^3+z^2-4z+6

La instrucción

```
» syms z
» p=z^3+z^2-4*z+6
» sol=solve(p)
```

nos da las soluciones de la ecuación. La solución es un vector de tres componentes que hemos guardado con el nombre `sol`.

La orden `solve` no siempre da todas las soluciones como puede comprobarse si se escribe

```
» sol=solve('sin(x)')
```

Sólo devuelve `sol = 0` .

Ejemplo 12 *Hallar la solución general de la ecuación* $az^4+bz^2+c=0$.

```
» sol=solve('a*z^4+b*z^2+c','z')
```

En el comando `solve` podemos especificar cual es la variable que deseamos despejar, en el caso de que halla varias. Así, en la ecuación del ejemplo anterior podemos despejar la *b* escribiendo

```
» sol=solve('a*z^4+b*z^2+c','b')
```

También podemos resolver un sistema de ecuaciones.

Ejemplo 13 *Hallar las soluciones del sistema* $-90z+12w+45z^2+6z^2w-12zw+z^2w^2-2zw^2+2w^2=-90$, $w^2-2w=-5$.

MATLAB almacena la solución del sistema en una estructura de datos:

```
» sol=solve('90-90*z+12*w+45*z^2+6*z^2*w-12*z*w+z^2*w^2
    -2*z*w^2+2*w^2','w^2-2*w+5')
```

Para obtener los valores de la solución escribimos

```
» sol.w,sol.z
```

En el caso de que en el sistema aparezcan más variables que ecuaciones también podemos elegir qué variables deseamos despejar.

Ejemplo 14 *En el sistema* $ax+by=0$, $bx-ay+1=0$, *despejar las variables a y x en función de las variables y y b.*

```
>> syms a x b y real
>> ec1=a*x+b*y
ec1 =
a*x+b*y
>> ec2=b*x-a*y+1
ec2 =
b*x-a*y+1
```

```
>> S=solve(ec1,ec2,'a','x')
S =
    a: [2x1 sym]
    x: [2x1 sym]
>> solucion=[S.a S.x]
solucion =
[ (1/2+1/2*(1-4*b^2*y^2)^(1/2))/y,  1/2/b*(-1+(1-4*b^2*y^2)^(1/2))]
[ (1/2-1/2*(1-4*b^2*y^2)^(1/2))/y,  -1/2*(1+(1-4*b^2*y^2)^(1/2))/b]
```

9. Resolución de ecuaciones y sistemas de ecuaciones diferenciales

9.1. Resolución de ecuaciones diferenciales

Una ecuación diferencial la escribiremos siempre entre comillas simples. En MATLAB, y' se representa por `Dy`, y'' se representa por `D2y`, y''' por `D3y`, etc.

Por ejemplo, la ecuación $\frac{d^3y}{dt^3} + 4\frac{dy}{dt} = \text{sen}^2 t$ se escribiría en MATLAB como 'D3y+4*Dy=sin(t)^2'.

Las condiciones iniciales también van entre comillas simples. Por ejemplo, las condiciones $y(0) = 1,\ y'(0) = 2,$ $y''(0) = 3$ se escriben $'y(0) = 1'$, $'Dy(0) = 2'$, $'D2y(0) = 3'$.

Orden	Descripción
`dsolve(ecuacion,'x')`	Devuelve la solución general de una ecuación diferencial respecto de la variable independiente x
`dsolve(ecuacion,condicion1,condicion2,...,'x')`	Devuelve la solución de la ecuación diferencial respecto de la variable independiente x verificando las condiciones iniciales indicadas

Ejemplo 15 *Resolver el problema* $y'' - 4y' + 3y = 9x^2 + 4,\ y(0) = 6,\ y'(0) = 8$

```
>> y=dsolve('D2y-4*Dy+3*y=9*x^2+4','y(0)=6','Dy(0)=8','x')
y =
2*exp(3*x)-6*exp(x)+10+8*x+3*x^2
```

Si se omite la variable, el programa interpreta que se trabaja con variable independiente t.

Ejemplo 16 *Resolver la ecuación* $\frac{d^3y}{dt^3} + 4\frac{dy}{dt} = \text{sen}^2 t$

```
>> clear, syms t
y=dsolve('D3y+4*Dy=sin(t)^2')
y =
-1/2*cos(2*t)*C2+1/2*sin(2*t)*C1-3/64*sin(2*t)+1/16*cos(2*t)*t+1/8*t+C3
```

9.2. Resolución de sistemas de ecuaciones diferenciales

Orden	Descripción
`dsolve(ec1,ec2,...,'x')`	Devuelve la solución general del sistema de ecuaciones diferenciales de variable independiente x
`dsolve(ec1,ec2,...,cond1,cond2,..., 'x')`	Devuelve la solución del sistema respecto de la variable independiente x verificando las condiciones iniciales indicadas

Ejemplo 17 *Hallar la solución del problema de valores iniciales*

$$\begin{cases} u' = u + 2v \\ v' = -2u + v + 2e^x \end{cases} \qquad u(0) = 2,\ v(0) = 0$$

Utilizando **dsolve**, haríamos

```
>> syms x, ed1='Du=u+2*v'; ed2='Dv=-2*u+v+2*exp(x)';
 S= dsolve(ed1,ed2,'u(0)=2','v(0)=0','x')
 S =
    v: [1x1 sym]
    u: [1x1 sym]
```

y esto nos indica que MATLAB ha calculado las soluciones y las ha almacenado en la estructura S. De este modo, para definirlas bastará escribir:

```
   >> u=S.u,v=S.v
u =
exp(x) + cos(2*x)*exp(x)
v =
-sin(2*x)*exp(x)
```

También en este caso, si se omite la variable, el programa interpreta que se trabaja con variable independiente t.

10. Funciones de tipo numérico

10.1. Funciones anónimas

Además de definirlas por medio de un archivo, como ya se ha visto, las funciones de tipo numérico pueden construirse directamente en la ventana de comandos utilizando las funciones anónimas.

Función	**Salida**
`f=@(variables)expresión`	Almacena en f la función definida en expresión, utilizando como variables independientes las que aparecen en `variables`
`matlabFunction(g)`	Convierte la función simbólica g en una función anónima

Ejemplo 18
```
>> f=@(x)x.^2   %función de la variable x
f =
    @(x)x.^2
>> f(3),f([1 2 3])% puedo evaluarla sobre escalares y matrices
ans =
```

```
     9
ans =
     1     4     9
>> syms x y, g=x^2+3*y %es una función simbólica
g =
x^2 + 3*y
>> g_num=matlabFunction(g) %es una función anónima
g_num =
    @(x,y)y.*3.0+x.^2
```

10.2. Polinomios

En Matlab, un polinomio se define mediante un vector fila cuyas coordenadas son los coeficientes del polinomio según potencias decrecientes. Por ejemplo, el polinomio $p = x^2 + 2x + 5$ se escribiría p=`[1 2 5]` . Para trabajar con fracciones polinómicas, serán útiles la órdenes siguientes:

Función	**Salida**
`conv(p,q)`	Múltiplica los polinomios p y q
`[c,R]=deconv(p,q)`	Divide q por p y devuelve en c el cociente y en R el resto de la divisón
`[A,r,k]=residue(p,q)`	Devuelve los elementos de la descomposición en fracciones simples de la fracción p/q

Ejemplo 19 *Descomponemos en fracciones simples* $\dfrac{x^3+7x^2-12x}{x^3-2x^2-x+2}$

```
>> p=[1 7 -12 0  ];q=[1 -2 -1 2];
>> [A r k]=residue(p,q)
r =
    4.0000
    3.0000
    2.0000
p =
    2.0000
   -1.0000
    1.0000
k =
     1
```

lo que significa que

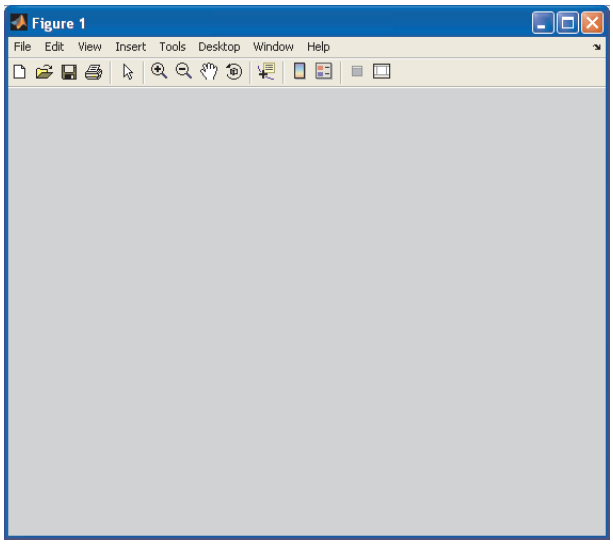
$$\frac{p}{q} = 1 + \frac{4}{x-2} + \frac{3}{x+1} + \frac{2}{x-1}$$

En `A` se almacenan los coeficientes de las fracciones, en `r` las raíces del denominador y en `k` el cociente de la división.

11. Gráficos con MatLab

11.1. Gráficos 2D

MatLab genera los gráficos, tanto 2D como 3D, en una ventana distinta del área de trabajo y del editor de ficheros, es lo que se llama una *ventana gráfica* o *figura*, que, por defecto, tiene este aspecto.



La mayor parte de los comandos que se utilizan para construir gráficos llevan implícita la orden de abrir una ventana gráfica, no obstante, existen instrucciones que permiten abrir (o cerrar) las ventanas gráficas antes de construir los gráficos. Además, se pueden mantener abiertas varias ventanas gráficas a la vez, una de ellas es la que llamaremos *ventana activa*, que será la última ventana gráfica abierta, aunque esto puede modificarse a partir de ciertas órdenes o simplemente, pinchando con el ratón en la que queremos que sea la activa. Todas las instrucciones gráficas serán enviadas a la que en ese momento es la ventana activa. Las instrucciones básicas son las siguientes:

`figure`	Genera una nueva ventana gráfica
`figure(n)`	Genera la ventana activa que numerará como n y si ya está creada, esta será la activa desde este momento
`close`	Cierra la ventana gráfica activa
`close(n)`	Cierra la ventana gráfica número n
`close all`	Cierra todas las ventanas gráficas abiertas
`clf`	Borra el contenido de la ventana gráfica activa, manteniéndola abierta
`hold on`	Todos los gráficos de la ventana activa se superpondrán, sin borrar los ya dibujados
`hold off`	Reemplaza el gráfico antiguo por el nuevo (esta es la opción por defecto)

El comando `ezplot` permite representar curvas utilizando directamente la expresión simbólica de la curva. La curva puede venir expresada de tres formas:

- en forma explícita: $y = f(x)$, con $x \in [a,b]$
- en forma paramétrica: $(x(t),y(t))$, con $t \in [a,b]$
- en forma implícita: $f(x,y) = 0$, con $(x,y) \in [a,b] \times [c,d]$

Su sintaxis básica es:

Explícita `ezplot(f,[a,b])`	donde `f` contiene la expresión de $f(x)$ y `[a,b]` es el dominio de la variable x. Si se omite este último argumento se toma por defecto el intervalo $[-2\pi,2\pi]$
Paramétrica `ezplot(xt,yt,[a,b])`	donde `xt` e `yt` contienen la expresión de $x(t)$ e $y(t)$ respectivamente y `[a,b]` es el dominio de la variable t. Si se omite este último argumento se toma por defecto el intervalo $[-2\pi,2\pi]$
Implícita `ezplot(f,[a,b,c,d])`	donde `f` contiene la expresión de $f(x,y)$ y `[a,b,c,d]` es el dominio de (x,y). Si se omite este último argumento se toma por defecto el intervalo $[-2\pi,2\pi] \times [-2\pi,2\pi]$

La instrucción `plot` es la función clave de la mayor parte de los gráficos 2D en MATLAB y su sintaxis es la siguiente: `plot(x,y,s)` (la variable `s` es opcional). Si queremos dibujar n puntos $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, ... ,$P_n = (x_n, y_n)$, `x` sería $[x_1, x_2, \ldots, x_n]$ e `y` sería $[y_1, y_2, \ldots, y_n]$. Si la variable `s` no aparece, dibujaría los puntos unidos por segmentos. La variable `s` puede contener un símbolo de cada una de las columnas de la siguiente tabla, encerrados entre apóstrofos:

Color	**Marca**	**Trazo**
`b` azul	`.` punto	`-` continuo
`g` verde	`o` círculo	`:` discontinuo
`r` rojo	`x` aspa	`-.` punto y guión
`y` amarillo	`*` asterisco	`--` discontinuo
`m` magenta	`s` cuadrado	
`k` negro	`d` rombo	
`w` blanco	`v` triángulo (abajo)	
	`\^` triángulo (arriba)	
	`<` triángulo (izquierda)	
	`>` triángulo (derecha)	
	`p` estrella 5 puntas	
	`h` estrella 6 puntas	

Ejemplo 20 *Dibujar la gráfica de la función* $f(x) = x^2 + 1$, *en el intervalo* $[-3, 3]$

Primero se construye un vector con las coordenadas *x*,

```
» x=linspace(-3,3);
```

Es recomendable recordar el ; al finalizar la instrucción, ya que esto evita que aparezca información innecesaria por pantalla.

A continuación se construye el vector que contiene las imágenes de dichos valores por la función f, es decir, el vector de las coordenadas *y*

```
» y=x.^2+1;
```

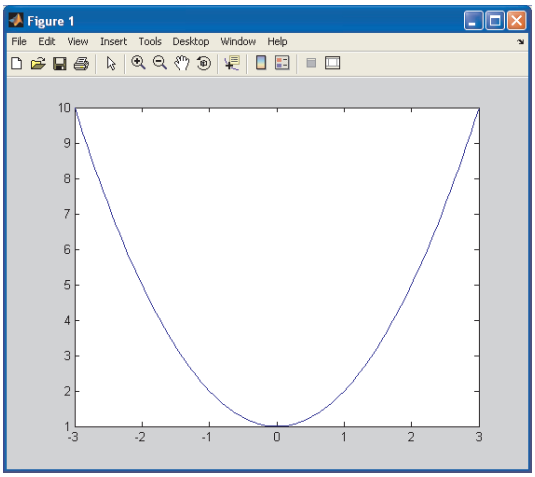
Obsérvese que a la operación elevado a `^` la hemos antecedido de un punto, ya que lo que queremos no es elevar a 2 la matriz *x* (que ni tan siquiera estaría definido), sino elevar a 2 cada elemento de la matriz x.

A continuación utilizamos el `plot` para dibujar la gráfica pedida

```
» plot(x,y)
```

Como no hemos incluido la variable `s`, la gráfica resulta ser en azul (color por defecto), con trazo continuo (uniendo los

puntos por una poligonal) y sin marcas. La gráfica obtenida es:



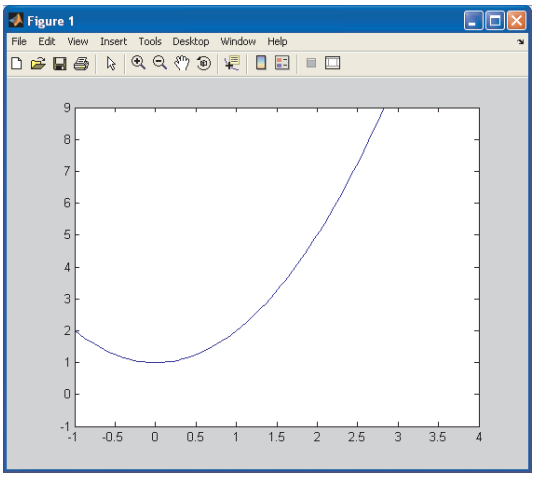
La gráfica aparece en un rectángulo blanco que en el lenguaje de MATLAB se llama 'eje'. Una figura puede tener varios ejes, al último eje utilizado es al que llamaremos eje activo. Se pueden modificar los ejes a partir de las siguientes funciones:

Orden	**Salida**
`axis([xmin xmax ymin ymax])`	Los números reales xmin y xmax definen los límites inferior y superior de la coordenada *x* mientras que ymin e ymax hacen lo propio para la coordenada *y*
`axis opción`	Genera cambios en las escalas de un eje. Si opción es `equal` utiliza la misma escala en ambas coordenadas, si es `square` ajusta la figura a un cuadrado, si es `off` oculta el eje.
`zoom`	Activa la utilidad `zoom` sobre el gráfico, permitiendo realizar una ampliación (reducción) al pulsar el botón izquierdo (derecho) del ratón en una parte del gráfico. La utilidad se desactiva volviendo a ejecutar `zoom`
`grid on` `grid off`	agrega las líneas coordenadas a la representación gráfica elimina las líneas coordenadas a la representación gráfica, esta es la opción por defecto

Estas funciones alteran la visualización de gráfica, pero no lo que se ha dibujado. Obsérvese lo que ocurre con la gráfica anterior si en el área de trabajo tecleamos:

```
» axis([-1 4 -1 9])
```

se obtiene:



prueba ahora con las siguientes instrucciones:

```
» axis([-3 3 -1 9])
» axis square
» axis equal
```

La ventana gráfica dispone de un menú que permite modificar el estilo de las líneas, añadir textos, borrar partes de la gráfica...

Veamos algunas instrucciones útiles para dibujar complejos:

Función	**Salida**
`plot(z,s)`	Dibuja el complejo `z`. La variable `s` es opcional. (Ver `plot` sección anterior)
`polar(a,r,s)`	La variable `s` es opcional, `a` y `r` son las variables que contienen las coordenadas polares de los puntos (ángulo y radio) que se quieren dibujar. Si la variable s no aparece, dibujaría los puntos unidos por segmentos.
`compass(a,b,s)`	Dibuja los vectores con origen en el (0,0) y extremos en los puntos de coordenadas `(a(i),b(i))`. Aquí `s` es opcional al igual que lo era en `plot` y `polar`
`compass(z)`	Idéntico a `compass(real(z),imag(z))`
`quiver(x,y,u,v,m)`	Representa el vector `(u,v)` con origen en el punto `(x,y)`. La variable `m` es opcional, y representa una graduación para la longitud del vector, 1 si queremos la longitud real, 0 escalado automático. El valor 0 es el que toma por defecto. Si `x,y,u,v` son matrices del mismo tipo, dibujará varios vectores a la vez

Ejemplo 21 *Dibújense, en tres ventanas gráficas distintas, los complejos* 0, $1+i$ *y* $-1+3i$ *unidos por segmentos, marcados con puntos azules y vectorialmente.*

```
» z=[0 1+i -1+3i]
» plot(z)
» figure
» plot(z,'b.')
» figure
» compass(z)
```

Ejemplo 22 *Dibujar en la misma ventana gráfica el triángulo* T *cuyos vértices son los del ejemplo 21 en azul, en rojo*

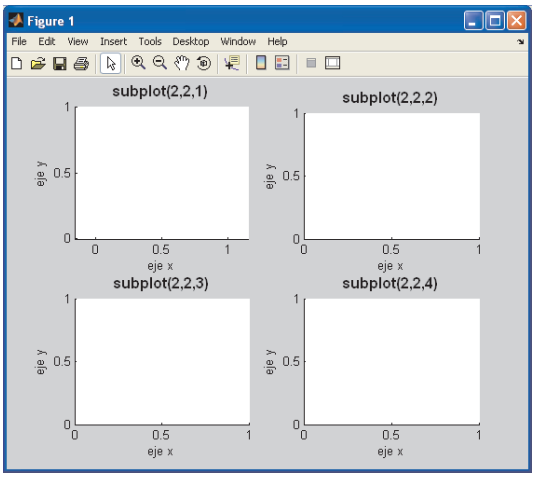
el triángulo girado en torno al 0 *un ángulo de* $\frac{\pi}{2}$*, en verde el homotético de* T *de centro* 0 *y razón* 1.5*, y en negro su trasladado por el vector* $(0,-1)$.

```
» z=[z 0];
» plot(z)
» hold on
» plot(i*z,'r')
» plot(1.5*z,'g')
» plot(z-i,'k')
```

Otras utilidades gráficas pueden ser las que nos permiten dividir la pantalla en varias subventanas, esto se realiza con la siguiente instrucción:

Función	**Salida**
`subplot(n,m,k)`	Divide la ventana gráfica activa en $n \times m$ subventanas y envía el gráfico a la subventana número k (se cuenta de izquierda a derecha y de arriba hacia abajo)

La forma de activar uno de los ejes generados en la ventana es con la orden subplot. `n` es el número de filas en que se divide la pantalla, `m` es el número de columnas, y `k` se refiere al eje sobre el que se va a enviar la gráfica, numera por orden los ejes de izquierda a derecha y de arriba abajo. Obsérvese la numeración en la gráfica siguiente:



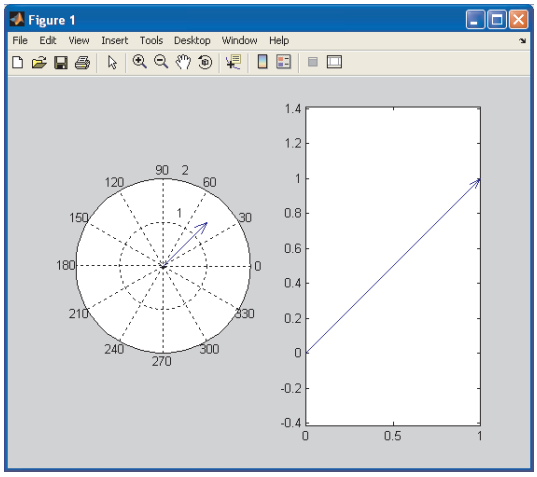
Las gráficas de los distintos ejes pueden ser de distinto tipo:

Ejemplo 23 *Dibujar en dos subventanas de la misma ventana gráfica el complejo* $z = 1 + i$ *con la instrucción* `compass` *y con la instrucción* `quiver`.

Creamos un fichero `m` con las instrucciones siguientes:

```
z=1+i
subplot(1,2,1)
compass(z)
subplot(1,2,2)
quiver(0,0,1,1,0)
axis equal
```

obteniendo la siguiente gráfica:



Se observa que en la misma ventana gráfica tenemos dos ejes, en cada momento se activa el que indica el subplot, es decir `subplot(1,2,1)` activa el primer eje `subplot(1,2,2)` activa el segundo eje.

11.2. Gráficos 3D

11.2.1. Dibujo de curvas

La función `plot3` es análoga a su homóloga bidimensional `plot`. Su forma de uso más sencilla es

```
» plot3(x,y,z)
```

dibuja una línea que une los puntos $(x(1),y(1),z(1))$, $(x(2),y(2),z(2))$, $(x(3),y(3),z(3))$, etc. Al igual que en el caso plano se puede añadir una cadena con 1, 2 ó 3 caracteres para determinar el color, los marcadores y el tipo de línea. Básicamente, el uso de esta instrucción es como sigue:

Función	Salida
`plot3(x,y,z)`	Si x,y,z son números dibuja el punto de coordenadas (x,y,z), si son vectores dibuja el conjunto de puntos $\{(x_1,y_1,z_1),\ldots,(x_n,y_n,z_n)\}$ y los enlaza con segmentos.
`plot3(x,y,z,S)`	Hace lo mismo que la instrucción anterior, pero con las opciones especificadas en la variable de carácter S (color, marcas y tipo de trazo) vistas en la sesión 5.

Ejemplo 24 *Representar la trayectoria* $\sigma(t) = ((2-2t)\cos(4\pi t),(2-2t)\,\mathrm{sen}(4\pi t),2t)$ *con* $t \in [0,2\pi]$.

SOLUCIÓN:

```
» t=linspace(0,2*pi,500);
» x=(2-2*t).*cos(4*pi*t);
» y=(2-2*t).*sin(4*pi*t);
» z=2*t;
» plot3(x,y,z)
```

11.2.2. Superficies

Vamos a representar a continuación superficies. En general, se puede determinar una superficie por una función de la forma:

$$f(u,v) = (x(u,v), y(u,v), z(u,v)) \text{ con } (u,v) \in I = [a,b] \times [c,d]$$

los puntos imágenes serían los que formarían la superficie.

La representación de una superficie en MATLAB se realiza básicamente generando una *malla* de puntos sobre ella y *uniéndolos* mediante segmentos o planos para obtener el aspecto de superficie en el sentido habitual. Una *malla* sobre la superficie se construye a partir de una partición del intervalo I. Si $a = u_1 < u_2 < ... < u_n = b$ y $c = v_1 < v_2 < ... < v_m = d$ el conjunto de puntos (u_i, v_j) con $i = 1, ..., n$ y $j = 1, ..., m$ define la malla en I y los puntos $(x(u_i, v_j), y(u_i, v_j), z(u_i, v_j))$ la malla sobre la superficie.

La malla sobre I se genera con el siguiente comando:

Función	**Salida**
`[U,V]=meshgrid(u,v)`	A partir de dos vectores `u` de dimensión n y `v` de dimensión m. `U` es una matriz $m \times n$, cuyas filas son m copias del vector `u` y `V` es una matriz $m \times n$, cuyas columnas son n copias del vector `v`

Los puntos donde MATLAB dibuja la función de forma exacta son los $f(U_{ij}, V_{ij})$ para cada $i = 1, \ldots, m$ y para cada $j = 1, \ldots, n$, y a dichos puntos les llamaremos nudos de la malla.

La forma habitual de proceder es la siguiente:

- Se definen los vectores `u=linspace(a,b,n)` y `v=linspace(c,d,m)`.
- Se generan las matrices `[U,V]=meshgrid(u,v)`
- Se definen las matrices `X=x(U,V)`, `Y=y(U,V)`, `Z=z(U,V)`, siempre teniendo en cuenta que las operaciones que se realicen con `U` y `V` deben realizarse elemento a elemento.

Una vez generadas las matrices de coordenadas de los puntos de la malla existen distintos tipos de gráficos que se pueden realizar, con distintos aspectos.

Gráficos de malla La superficie se representa mediante una malla, con un aspecto similar al de una red de pesca, cuyos nudos están situados sobre la superficie correspondiente. La orden básica para este tipo de gráficos es `mesh` y sus distintas opciones:

Función	**Salida**
`mesh(X,Y,Z,C)`	Dibuja el gráfico con las líneas de rejilla que componen la malla con los colores especificados en `C`, que debe ser una matriz del mismo tamaño que `X`, `Y` y `Z`. Si se omite este último argumento `C = Z`
`meshz(X,Y,Z,C)`	Representa el gráfico anterior, con plano de referencia en el valor mínimo y una especie de 'cortina' en los bordes del dominio de la función
`meshc(X,Y,Z,C)`	Representa el gráfico de malla junto con las curvas de nivel proyectadas en el plano *OXY*

Para observar la diferencia entre los distintos gráficos, ejecuta en el área de trabajo el fichero `ej_mesh`.

Gráficos continuos En este tipo de gráficos, la superficie se representa como una lámina continua, y se genera con las siguientes órdenes:

Función	**Salida**
`surf(X,Y,Z,C)`	Dibuja el gráfico con los colores especificados en `C`, que debe ser una matriz del mismo tamaño que `X`, `Y` y `Z`. Si se omite este último argumento `C = Z`
`surfc(X,Y,Z,C)`	Representa el gráfico junto con las curvas de nivel proyectadas en el plano *OXY*

Ejemplo 25 *Dibujar la gráfica de la función que a cada complejo le asigna su módulo, para complejos con módulo en el intervalo* $[0, 2\pi]$.

SOLUCIÓN:

```
» r=linspace(0,4);
» t=linspace(0,2*pi);
» [r,t]=meshgrid(r,t);
» X=r.*cos(t);
» Y=r.*sin(t);
» Z=r;
» surf(X,Y,Z)
```

Índice alfabético