PRIMEROS PASOS con MATLAB[1]

Un cálculo elemental

Para hacernos una primera idea de cómo se trabaja con MATLAB vamos a sumar los números *a=2*, *b=3*: Para ello se puede escribir

```
>> 2+3
```

Otra posibilidad es

```
>> a=2

>> b=3

>> suma=a+b
```

Si no se realizan modificaciones, las variables *a*, *b*, *suma* permanecen en el espacio de trabajo (parte de la memoria del ordenador reservada a la sesión actual de trabajo) con los valores asignados.

A medida que se realizan cálculos, el área de trabajo se va llenando. Se puede borrar la ventana de comandos y situar el prompt(>>) en la primera línea con la orden

```
>> clc
```

Esta acción no borra las variables asignadas, como se puede comprobar escribiendo su nombre

```
>> a
```

Por otra parte se pueden ver todas las líneas de la sesión actual utilizando la barra de desplazamiento vertical

Utilización de la ayuda

• `help orden` Muestra en pantalla información sobre `orden`. En ocasiones esta información ocupa más de una pantalla. Para leerla toda se puede utilizar las barras de desplazamiento.

Ejemplo *Obtener información acerca de la orden* `clc` *utilizando la ayuda.*
Sol.:

```
>> help clc
```

[1] El contenido de estos apuntes está tomado de T. Aranda-J.G. García: “Notas sobre MATLAB”.

Cálculos básicos

Operaciones aritméticas con MATLAB

Para realizar los cálculos aritméticos (suma, resta, multiplicación, división y potenciación) con MATLAB es suficiente tener en cuenta:

• Símbolos utilizados para las operaciones:

SUMA	RESTA	MULTIPLICACIÓN	DIVISIÓN	POTENCIACIÓN
+	-	*	/	^

• Orden de prioridad de las operaciones: las expresiones se evalúan de izquierda a derecha; la operación potencia tiene el orden de prioridad más alto, seguida por multiplicación y división que tienen ambas igual prioridad y seguidas, finalmente, por suma y resta con igual prioridad.

Se pueden emplear paréntesis para alterar esta ordenación, en cuyo caso la evaluación se inicia dentro del paréntesis más interno y se prosigue hacia afuera.

Ejemplo *Realizar los siguientes cálculos:*

$$3^2-5-\frac{6}{3}\cdot 2, \qquad 3^2-5-\frac{6}{3\cdot 2}, \qquad 4\cdot 3^2+1, \qquad (4\cdot 3)^2+1.$$

Sol.: En la ventana de comandos se escribe:

```
>> 3^2-5-6/3*2
>> 3^2-5-6/(3*2)
>> 4*3^2+1
>> (4*3)^2+1
```

Variables

Hasta aquí los cálculos se realizan igual que en una calculadora. La introducción de variables ofrece nuevas posibilidades. Supongamos que queremos calcular el área de un triángulo de base 21.3 m y altura 12.6 m. De acuerdo con lo visto en la sección anterior, este cálculo se puede realizar así:

```
>> 1/2*21.3*12.6
```

Otra posibilidad es utilizar las variables *base*, *altura*, *area* y realizar los cálculos de la siguiente forma:

```
>> base=21.3      (Asigna a la variable base el valor 21.3)
>> altura=12.6    (Asigna a la variable altura el valor 12.6)
>> area=1/2*base*altura    (Asigna a la variable area el valor correspondiente)
```

Las variables *base*, *altura*, *area* permanecen en el espacio de trabajo con los últimos valores asignados y pueden ser llamadas o modificadas cuantas veces se desee.

Observación: La modificación de la variable *altura* no cambia el valor de la variable *area* si ésta no se vuelve a calcular.

```
>> altura=9.7
>> area
>> area=1/2*base+altura
```

Reglas para nombrar variables

• Las letras mayúsculas y minúsculas son distintas a efectos de nombrar variables. Por ejemplo, son diferentes las variables *base*, *Base*, *BASE*.

• El nombre de una variable puede tener hasta 31 caracteres; si hubiese más serían ignorados.

• El nombre de una variable debe comenzar obligatoriamente por una letra. Puede contener letras, números y el guión de subrayado (_); no se permiten espacios en blanco.

• No es conveniente nombrar variables mediante expresiones que tengan un significado específico en MATLAB: Por ejemplo, no es aconsejable utilizar `log` como nombre de variable ya que ésta es la designación de la función logarítmica en MATLAB.

Como regla general es aconsejable que el nombre de una variable sea indicativo de su contenido.

Algunas variables predefinidas en MATLAB

Hay algunas variables que, por defecto, tienen un valor asignado. Podemos citar:

• `ans` Es la variable que, por defecto, contiene los resultados.

• `pi` Contiene el valor del número real π.

• `eps` Es el número positivo más pequeño que sumado a 1 genera un número mayor que 1 en el ordenador.

• `Inf` o `inf` Representa el valor infinito. Se obtiene, por ejemplo, en caso de *overflow* o división por cero.

• `NaN` o `nan` (Not a Number) Representa una expresión indeterminada, por ejemplo: $\frac{0}{0}$.

• `i` o `j` Representa la unidad imaginaria $i = j = \sqrt{-1}$.

Borrar variables

Para borrar variables se utilizan las órdenes:

• `clear x y` Borra las variables *x* e *y*.

• `clear` Borra todas las variables de la sesión de trabajo.

Signos de puntuación, comentarios, movimientos del cursor

• En una misma línea pueden definirse varias variables separadas por coma (`,`) o por punto y coma (`;`). La diferencia consiste en que el punto y coma inhibe la visualización en pantalla. Por ejemplo:

```
>> a=2,b=3;c=5
>> d=a*b+c;
>> d
```

• Utilizando tres puntos se puede cambiar de línea sin ejecutar las órdenes escritas. Por ejemplo:

```
>> x=25,...
y=x/5
```

Esto puede resultar útil cuando una expresión no cabe en una sola línea de la ventana de comandos.

• MATLAB ignora lo que se encuentra a la derecha del símbolo %. Esto permite introducir comentarios.

```
>> a=2;b=3;  %a es la base, b es la altura
```

• Movimientos del cursor:

"Flecha arriba" (Recupera la línea anterior)
"Flecha abajo" (Recupera la línea siguiente)
"Flecha izquierda" (Mueve el cursor hacia la izquierda un carácter)
"Flecha derecha" (Mueve el cursor hacia la derecha un carácter)

Se recomienda hacer uso de esta utilidad siempre que sea posible ya que permite ahorrar mucho tiempo.

Ejemplo: *Calcular la suma* a=1+2+3+4+5+6+7+8+9+10. *Utilizando los movimientos del cursor, calcular* b=1-2+3-4+5-6+7-8+9-10.

Sol.:

```
>> a=1+2+3+4+5+6+7+8+9+10
>> "Flecha arriba"
(editando la línea)
>> b=1-2+3-4+5-6+7-8+9-10
```

Funciones predefinidas

MATLAB incorpora una serie de funciones que se corresponden con las funciones matemáticas más utilizadas.

Para las funciones elementales en MATLAB se utiliza la siguiente notación:

exp	sin	asin	sinh	asinh
log	cos	acos	cosh	acosh
log10	tan	atan	tanh	atanh
log2	cot	acot	coth	acoth
sqrt	sec	asec	sech	asech
	csc	acsc	csch	acsch

La correspondencia con la notación que se usa habitualmente en matemáticas es la siguiente:

MATLAB	MATEMÁTICAS
`exp(x)`	e^x
`log(x)`	$\ln(x)$
`log10(x)`	$\log_{10}(x)$
`log2(x)`	$\log_2(x)$
`sqrt(x)`	$\sqrt{x}$
`sin(x)`	$\text{sen}(x)$
⋮	⋮

En particular, `exp(1)` proporciona el número e=2.7182...
Otras funciones predefinidas en MATLAB de uso frecuente son:

`abs(x)`	Valor absoluto
`fix(x)`	Redondea hacia cero
`floor(x)`	Redondea hacia menos infinito
`ceil(x)`	Redondea hacia más infinito
`round(x)`	Redondea hacia el entero más próximo
`rem(m,n)`	Resto al dividir m entre n
`rand`	Número aleatorio entre 0 y 1

El uso de estas funciones se muestra en el siguiente ejemplo:

Ejemplo:

```
>> sqrt(16)
>> 16^(1/2)
>> (-27)^(1/3)
>> sin(pi/3)
>> x=cos(pi);y=log(25);z=exp(y);
>> t=25*x+z
>> abs(-3)
>> fix(5.7),floor(5.7),ceil(5.7),round(5.7)
>> fix(-5.7),floor(-5.7),ceil(-5.7),round(-5.7)
>> rem(20,3)
>> rand
```

Formatos numéricos

Cuando el resultado de un cálculo es un número entero, MATLAB lo muestra en pantalla como entero siempre que tenga menos de 10 cifras. Si tiene 10 o más cifras o es un número decimal, se puede expresar en alguno de los formatos que se muestran en la siguiente tabla:

	NÚMERO DE CIFRAS		
	PARTE ENTERA	PARTE DECIMAL	EXPONENTE
`format short`	3	4	
`format short e`	1	4	3
`format long`	2	14	
`format long e`	1	15	3

En todos los casos hay un espacio adicional para el signo. Por defecto MATLAB utiliza el formato `format short`. Cualquier otro se puede elegir escribiendo su nombre en la línea de comandos. El formato elegido solo afecta a la visualización en pantalla, no a la precisión de los cálculos que es siempre la misma.

Ejemplo: *Mostrar los números* $a = 1 + \frac{1}{3}$, $b = 123456789$, $c = 123451234512345$ *con diferentes formatos*.

Sol.:

```
>> a=1+1/3;b=123456789; c=123451234512345;
>> format short
>> a,b,c
>> format short e
>> a,b,c
>> format long
>> a,b,c
>> format long e
>> a,b,c
>> format short
```

Vectores y matrices

El tipo de dato básico en MATLAB es la matriz (MATLAB es abreviatura de MATrix LABoratory). Por esta razón, entender el contenido de este capítulo es esencial para familiarizarse con MATLAB.

Aunque los vectores, e incluso los escalares, pueden ser considerados como matrices, aquí se estudian separadamente como es habitual en Matemáticas.

Vectores

Cómo introducir vectores

Los vectores se introducen escribiendo, entre corchetes, cada una de sus coordenadas separadas por un espacio en blanco o una coma. Por ejemplo, para introducir el vector (1, 3, 5, -7, 0.33, π, e) se escribe

```
>> [1 3 5 -7 0.33 pi exp(1)]
```

o bien,

```
>> [1, 3, 5, -7, 0.33, pi, exp(1)]
```

Un vector puede asignarse a una variable, por ejemplo:

```
>> v = [1 3 5 -7 0.33 pi exp(1)]
```

En algunos casos se puede generar vectores sin necesidad de introducir explícitamente sus coordenadas:

• `x=[x1:h:xn]` Genera el vector $x = (x_1, x_1+h, x_1+2h, \ldots, x_1+kh)$, donde k es el mayor número entero tal que $x_1+kh \leq x_n$. Los corchetes pueden sustituirse por paréntesis o suprimirse. Por ejemplo,

```
>> [1:0.1:2]
```

genera el vector (1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2).

• `x=[x1:xn]` Hace lo mismo que `x=[x1:1:xn]`.

• `x=linspace(a,b,n)` Genera un vector de n coordenadas espaciadas

uniformemente, comenzando por *a* y terminando por *b*.

De acuerdo con esto, el vector *x* = (-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8) se puede generar de las siguientes formas:

```
>> x=[-3 -2 -1 0 1 2 3 4 5 6 7 8]
>> x=[-3:1:8]
>> x=[-3:8]
>> x=(-3:8)
>> x=-3:8
>> x=-3:8.7
>> x=linspace(-3,8,12)
```

Para apreciar la diferencia entre los dos modos descritos para generar vectores, observemos la diferencia entre los vectores `x` e `y`:

```
>> x=0:0.1:1.07
>> y=linspace(0,1.07,11)
```

Acceso a las coordenadas de un vector

Una vez definido un vector se puede acceder a sus coordenadas para conocer su valor, utilizarlo o modificarlo.

• `v(i)` Devuelve la coordenada *i*-ésima del vector *v*.

• `v(i1:h:i2)` Devuelve las coordenadas de *v* cuyos índices van desde i_1 hasta i_2, con incremento *h*.

• `v(i1:i2)` Hace lo mismo que `v(i1:1:i2)`.

• `v([i,j,k])` Devuelve las coordenadas *i*, *j* y *k* de *v*.

• `v(end)` Devuelve la última coordenada de *v*.

Ejemplo

```
>> v=[-1 3 2 4 -8 3]
>> v(2)          (Muestra la segunda coordenada de v)
>> v(end)        (Muestra la última coordenada de v)
>> v(2)+5        (Se utiliza la segunda coordenada de v en una operación)
>> v(2)=6        (Asigna el valor 6 a la segunda coordenada del vector v)
>> v(2:4)        (Muestra las coordenadas del vector v desde la segunda a la cuarta)
>> w=v(2:4)      (Define el vector w con las coordenadas segunda, tercera y cuarta de v)
>> u=v([1,5,6])  (Define u con las coordenadas primera, quinta y sexta de v)
```

Matrices

Cómo introducir matrices

Los elementos de la matriz se introducen por filas, separando cada fila de la siguiente por medio de un punto y coma (`;`), y los distintos elementos de una misma fila por espacios en blanco o comas. Por ejemplo, para introducir la matriz

$$A = \begin{pmatrix} 1 & 2 & 3 \\ -1 & 4 & 8 \\ 7 & 2 & 1 \end{pmatrix}$$

se escribe

```
>> A=[1 2 3; -1 4 8; 7 2 1];
```

Para visualizar la matriz A en pantalla escribimos:

```
>> A
```

Definición de matrices por cajas

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

```
>> A=[1 2 3; -1 4 8]
>> B=[5 5; 6 6]
>> C=[A B]
```

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

```
>> A=[1 2 3; -1 4 8]
>> B=[5 5 5; 6 6 6; 7 7 7]
>> C=[A;B]
```

Matrices especiales

Para introducir algunas matrices de uso frecuente (matrices de ceros, matrices de unos, matrices unidad, …) MATLAB dispone de órdenes específicas:

• `ones(n)` Genera una matriz cuadrada *n* x *n* formada por unos.

• `ones(m,n)` Genera una matriz *m* x *n* formada por unos.

• `zeros(n)` Genera una matriz cuadrada *n* x *n* formada por ceros.

• `zeros(m,n)` Genera una matriz *m* x *n* formada por ceros.

• `eye(n)` Genera una matriz unidad *n* x *n*.

• `eye(m,n)` Genera una matriz *m* x *n* con unos en la diagonal principal y ceros en el resto.

Ejemplo

```
>> A=ones(5)
>> B=ones(3,4)
>> C=zeros(5)
>> D=zeros(3,4)
>> E=eye(4)
>> F=eye(3,4), G=eye(4,3)
```

Acceso a los elementos de una matriz

Una vez definida una matriz se puede acceder a sus elementos para conocer sus valores, utilizarlos o modificarlos.

• `A(i,j)` Devuelve el elemento (*i, j*) de la matriz *A*.

• `A(i1:h:i2,j1:k:j2)` Devuelve la submatriz de *A* formada por las filas de la i_1 a la i_2 con incremento *h* y las columnas j_1 a la j_2 con incremento *k*.

• `A(i1:i2,j1:j2)` Devuelve la submatriz de *A* formada por las filas de la i_1 a la i_2 y las columnas j_1 a la j_2.

• `A(i1:h:i2,:)` Devuelve la submatriz de *A* formada por las filas de la i_1 a la i_2 con incremento *h*.

• `A(i1:i2,j)` Devuelve el elemento *j* de las filas comprendidas entre la i_1 y la i_2.

• `A(i,j1:j2)` Devuelve el elemento *i* de las columnas comprendidas entre la j_1 y la j_2.

• `A(:,j)` Devuelve el elemento *j* de todas las filas, o sea, la columna *j*.

• `A(i,:)` Devuelve el elemento *i* de todas las columnas, o sea, la fila *i*.

• `A(i,[j k])` Devuelve la submatriz (a_{ij}, a_{ik}).

El siguiente ejemplo aclara el funcionamiento de estas órdenes:

Ejemplo

```
>> A=[1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15; 16 17 18 19 20]
>> A(2,5)
>> A(1:2:4,1:3:5)
>> A(1:3,2:5)
>> A(:,1:2:5)
>> A(2:4,3)
>> A(2,1:4)
>> A(:,3)
>> B=[A(1:3,:) A(2:4,:)]
>> C=[A(1:3,1:3); A(:,[1 3 5])]
>> D=A([2 4], [3 5])
```

Información sobre las dimensiones

Las órdenes `length` y `size` ofrecen información sobre el tamaño de vectores y matrices. Conviene recordar que en la versión 5.0 de MATLAB el tamaño máximo de una matriz es de 16384 (= 128 x 128) elementos.

• `size(A)` Genera un vector con las dimensiones de la matriz *A*.

• `size(A,1)` Devuelve el número de filas de la matriz *A*.

• `size(A,2)` Devuelve el número de columnas de la matriz *A*.

• `[m n]=size(A)` Asigna a *m* el número de filas de *A* y a *n* el de columnas.

• `length(v)` Devuelve el número de coordenadas del vector *v*.

• `length(A)` Devuelve el valor de `max(size(A))`.

El funcionamiento de estas órdenes se puede ver en el siguiente ejemplo:

Ejemplo

```
>> A=ones(2,5), v=[1 2 3 4 5]
```

```
>> size(A)
>> size(A,1)
>> size(A,2)
>> [m n]=size(A)
>> length(v)
>> length(A)
```

Operaciones con vectores y matrices

Un vector de n coordenadas se puede considerar como una matriz 1 x n. Por esta razón vamos a describir simultáneamente las operaciones con vectores y matrices.

Operaciones algebraicas

Si *A* y *B* son matrices con dimensiones adecuadas y λ un escalar, las operaciones algebraicas en MATLAB se efectúan con las siguientes órdenes:

• `A+B` Suma *A* y *B*.

• `A-B` Resta *A* y *B*.

• `A*B` Multiplica *A* por *B*.

• `λ*A` Multiplica por λ todos los elementos de *A*.

• `A^n` Eleva la matriz *A* al número entero *n*.

• `A.'` Genera la traspuesta de *A*.

• `A'` Genera la conjugada traspuesta de *A*. Obsérvese que si *A* es una matriz real, `A'` y `A.'` coinciden.

• `inv(A)` Calcula la inversa de A.

Si *u* y *v* son dos vectores con el mismo número de coordenadas

• `dot(u,v)` Calcula el producto escalar.

• `cross(u,v)` Calcula el producto vectorial; es necesario que *u* y *v* tengan 3 coordenadas.

Ejemplo

```
>> A=[1 2; 3 4]
>> B=[-1 4; 1 -2]
>> C=[1 2 3; 4 5 6]
>> A+B
>> 3*A
>> A*C
>> C'
>> inv(B), B^(-1)
>> C'*(A-2*B)
>> u=[-1 2 3]; v=[5 -7 9];
>> dot(u,v)
>> cross(u,v)
```

Operaciones elemento a elemento

Además de las operaciones algebraicas mencionadas, en MATLAB se definen otro tipo de operaciones a las que denominamos operaciones elemento a elemento. Los operadores empleados son los de las operaciones aritméticas anteponiendo un punto (`.`), excepto en la suma de un escalar y una matriz.

Si $A=(a_{ij})$ y $B=(b_{ij})$ son matrices de iguales dimensiones y λ un escalar:

• `λ+A` Suma λ a cada elemento de la matriz A.

• `A.*B` Multiplica *A* por *B* elemento a elemento, es decir, define la matriz cuyo elemento *(i,j)* es $a_{ij}b_{ij}$.

• `A./B` Divide cada elemento de *A* por el correspondiente de *B*.

• `A.^λ` Eleva cada elemento de *A* a λ.

• `A.^B` Eleva cada elemento de *A* al correspondiente de *B*.

Ejemplo *(Se utilizan las matrices y vectores del ejemplo anterior.)*

```
>> 2+A
>> A.*B
>> A./B
>> A.^3
>> A.^B
>> u.*v
```

Funciones con argumento matricial

Una característica singular de MATLAB es que las funciones elementales admiten argumento matricial. Si *f* es una función elemental, como las definidas anteriormente y *A* es una matriz

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

entonces MATLAB calcula *f(A)* como

$$A = \begin{pmatrix} f(a_{11}) & \cdots & f(a_{1n}) \\ \vdots & & \vdots \\ f(a_{m1}) & \cdots & f(a_{mn}) \end{pmatrix}.$$

Ejemplo *(Se utilizan las matrices y vectores del ejemplo anterior.)*

```
>> sin(A)
>> sqrt(A)
>> exp(u)
```

Otras funciones de MATLAB con argumento matricial son:

• `rank(A)` Calcula el rango de la matriz *A*.

• `det(A)` Calcula el determinante de la matriz cuadrada *A*.

• `trace(A)` Calcula la traza de la matriz *A*.

• `max(A)` Devuelve el mayor elemento de cada columna de *A*.

• `[m p]=max(A)` Guarda en *m* el mayor elemento de cada columna de *A* y en *p* la fila en la que se encuentra.

• `max(v)` Devuelve el mayor elemento del vector *v*.

• `[m p]=max(v)` Guarda en *m* el mayor elemento de *v* y en *p* su posición.

La función `min` es análoga a `max`, y selecciona el elemento mínimo, en lugar del máximo.

Gráficos 2D

Ventanas gráficas

Los gráficos generados por MATLAB se presentan en ventanas específicas, llamadas *ventanas gráficas*. Al realizar un gráfico con MATLAB se abre automáticamente una *ventana gráfica*.

La orden `plot`

La orden básica para trazar gráficos bidimensionales es `plot`, cuya sintaxis es:

• `plot(x,y)` Si x e y son números, dibuja el punto de coordenadas cartesianas (x, y). Si se trata de los vectores $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$, dibuja el conjunto de puntos $\{(x_1, y_1), \ldots, (x_n,y_n)\}$ y los enlaza con segmentos. Si x e y son matrices de la misma dimensión, dibuja un conjunto de puntos por cada par de columnas correspondientes.

• `plot(x,y,S)` Hace lo mismo que `plot(x,y)` pero con las opciones especificadas en *S*. En *S* puede aparecer, encerrados entre apóstrofos, ciertos símbolos que sirven para especificar el color y la forma de dibujar los puntos y los trazos de la gráfica. La tabla con los símbolos posibles y su significado se da a continuación:

COLOR		MARCA		TRAZO	
`b`	azul	`.`	punto	`-`	continuo
`g`	verde	`o`	círculo	`:`	discontinuo
`r`	rojo	`x`	aspa	`-.`	punto y guión
`c`	cyan	`+`	cruz	`--`	discontinuo
`m`	magenta	`*`	asterisco		
`y`	amarillo	`s`	cuadrado		
`k`	negro	`d`	diamond		
`w`	blanco	`v`	triángulo (abajo)		
		`^`	triángulo (arriba)		
		`<`	triángulo (izquierda)		
		`>`	triángulo (derecha)		
		`p`	estrella 5 puntas		
		`h`	estrella 6 puntas		

• `plot(x1,y1,S1,x2,y2,S2,x3,y3,S3,...)` Dibuja, en una misma ventana gráfica, los gráficos definidos por las ternas (x_i, y_i, S_i). Si no se especifican los parámetros S_i, el dibujo

se realiza con trazo continuo y un color distinto para cada gráfica, utilizando los siete primeros colores de la primera columna de la tabla, en el orden señalado. Si se necesitan más colores, se repiten cíclicamente.

Hay que tener en cuenta que cuando damos una nueva orden gráfica, el nuevo gráfico elimina el anterior. Si queremos que el nuevo aparezca en una nueva ventana, debemos usar:

• `figure(n)` Genera la ventana gráfica número *n*; si ya existe, la activa.

Si queremos que el nuevo gráfico se presente en la misma ventana gráfica, sin borrar el gráfico anterior, usamos

• `hold on` Mantiene activa la ventana gráfica actual.

Ejemplo *Observar el resultado de aplicar las siguientes órdenes:*

```
>> clear, clc
>> x=[1 3 4]; y=[1 2 4];
>> plot(x,y)
>> figure(2)
>> plot(x,y,'r*')
>> figure(3)
>> plot(x,y,'kp:')
>> figure(4)
>> plot(x,y)
>> hold on
>> plot(x,y,'r*')
>> hold off
>> close all
```

Gráficos de funciones en coordenadas cartesianas

Para dibujar la gráfica de una función de una variable $y = f(x)$ en el intervalo $[a, b]$ se siguen los siguientes pasos:

• Se genera un vector $x = (a = x_1, x_2, \ldots, x_n = b)$ mediante `x = linspace(a,b,n)` o bien `x=a:h:b`.

• Se genera el vector $y = (f(x_1), f(x_2), \ldots, f(x_n))$ escribiendo `y=f(x)`.

• Con `plot(x,y)` se dibuja el conjunto de puntos $\{(x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$, que constituye la gráfica de *f*.

Ejemplo *Dibujar la gráfica de* $f(x) = e^{-x^2}\text{sen}(\pi x^3)$ *en el intervalo* [-3, 3].

Sol.:

```
>> clear, clc
>> x=linspace(-3,3,400);
>> y=exp(-x.^2).*sin(pi*x.^3);
>> plot(x,y)
>> close all
```

Ejemplo *Representar en una misma ventana gráfica las funciones* $f(x) = e^{-x^2}$, $g(x) = e^{-5x^2}$ *en el intervalo* [-2, 2].

(a) Con trazo continuo.
(b) La gráfica de f con asteriscos rojos y la de g con cruces verdes.

Sol.:

```
>> clear, clc
>> x=-2:0.1:2;
>> y1=exp(-x.^2);
>> y2=exp(-5*x.^2);
>> plot(x,y1,x,y2)
>> figure(2)
>> plot(x,y1,'r*',x,y2,'g+')
>> close all
```

Otra orden que sirve para dibujar la gráfica de una función es `fplot`, cuya sintaxis es:

• `fplot('función',[xmin,xmax])` Dibuja la función en el intervalo de variación de *x* dado.

• `fplot('función',[xmin,xmax,ymin,ymax],S)` Dibuja la función en el intervalo de variación de *x* e *y* dados, con las opciones especificadas en *S*.

En las órdenes anteriores, `función` puede ser una función predefinida, una función definida por el usuario en un fichero *m* o una cadena de caracteres que defina una función.

Ejemplo *Gráfica de la función* $f(x) = x\,\text{sen}\left(\frac{1}{x}\right)$ *en el intervalo* [-10,10]. *Dependiendo del resultado obtenido, hacer tanteos en los intervalos de variación de x e y hasta obtener una gráfica mejorada.*

Sol.:

```
>> clear, clc
>> fplot('x.*sin(1./x)',[-10,10])
>> fplot('x.*sin(1./x)',[-0.5,0.5,-0.4,0.4])
>> close
```

Manipulaciones geométricas

Las siguientes órdenes permiten situar una malla sobre el gráfico, manipular de diversos modos sus ejes y ampliar una parte de la figura.

• `grid on` Dibuja una malla en un gráfico 2D ó 3D.

• `grid off` Elimina la malla.

• `grid` Conmuta entre `on` y `off`.

• `axis equal` Obliga a usar el mismo factor de escala para ambos ejes.

• `axis normal` Elimina las opciones `square` y `equal`.

• `zoom on` Permite ampliar un gráfico o una parte de él seleccionada con el ratón.

• `zoom off` Desactiva el zoom. Es la opción por defecto.

• `zoom` Conmuta entre `on` y `off`.

El siguiente ejemplo sugiere un procedimiento gráfico para determinar las coordenadas del punto de intersección de las curvas $y = \text{sen}\, x$, $y = \ln x$ utilizando algunas de las órdenes anteriores.

Ejemplo *Dibujar las gráficas de las funciones* $y = \text{sen}\, x$, $y = \ln x$ *en el intervalo* [-5,5]. *Utilizar la orden* `grid` *para estimar un intervalo en el que se encuentre el punto de intersección. Con la orden* `axis` *para ver con mayor claridad la parte de la figura en la que se encuentra el punto de intersección. Haciendo uso del* `zoom` *comprobar que las coordenadas del punto de intersección son* (2.218, 0.796).

Sol.:

```
>> x=-5:.1:5;
>> y=sin(x);
>> plot(x,y)
>> hold on
>> a=eps:0.1:5;
```

```
>> b=log(a);
>> plot(a,b)
>> grid on
>> axis([0,3,-2,2])
>> zoom on
>> close
```

Gráficos de curvas en paramétricas

La orden plot permite dibujar una curva γ dada por sus ecuaciones paramétricas.

Ejemplo *Representar la curva*

$$\gamma = \begin{cases} x(t) = e^{-\frac{t}{20}}\cos(t) \\ y(t) = e^{-\frac{t}{20}}\operatorname{sen}(t) \end{cases}, \qquad t \in [0,80]$$

Sol.:

```
>> clear, clc
>> t=0:.01:80;
>> x=exp(-t/20).*cos(t);
>> y=exp(-t/20).*sin(t);
>> plot(x,y)
>> close
```

Gráficos 3D

La orden `plot3`

La orden básica para trazar gráficas en tres dimensiones es `plot3`:

• `plot3(x,y,z)` Si x, y, z son números, dibuja el punto de coordenadas cartesianas (x, y, z). Si se trata de los vectores $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$, $z = (z_1, \ldots, z_n)$, dibuja el conjunto de puntos $\{(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)\}$ y los enlaza con segmentos. Si x, y, z son matrices de la misma dimensión, dibuja un conjunto de puntos por cada terna de columnas correspondientes.

• `plot3(x,y,z,S)` Hace lo mismo que `plot3(x,y,z)` pero con las opciones especificadas en S (color y tipo de trazo que se utiliza para dibujar).

• `plot3(x1,y1,z1,S1,x2,y2,z2,S2,x3,y3,z3,S3,...)` Dibuja, en una misma ventana gráfica, los gráficos obtenidos por las cuaternas (x_i, y_i, z_i, S_i). Si no se especifican los parámetros S_i, se asigna un color diferente a cada terna. Un efecto parecido se consigue con la orden `hold on`.

Curvas alabeadas

Curvas alabeadas definidas por sus ecuaciones paramétricas

Para representar una curva alabeada

$$\begin{cases} x = x(t) \\ y = y(t) \\ z = z(t) \end{cases} \qquad t \in [t_0, t_1]$$

se define el vector t adecuado, se calculan los vectores x, y, z y se dibuja la curva mediante `plot3(x,y,z)`.

Ejemplo *Representar la curva*

$$\begin{cases} x(t) = e^{-0.2t}\cos\left(\dfrac{\pi}{2}t\right) \\ y(t) = e^{-0.2t}\,\text{sen}\left(\dfrac{\pi}{2}t\right) \\ z(t) = t \end{cases} \qquad t \in [0,20]$$

Sol.:

```
clear, clc, close all
t=0:0.1:20;
x=exp(-0.2*t).*cos(0.5*pi*t);
y=exp(-0.2*t).*sin(0.5*pi*t);
z=t;
plot3(x,y,z)
close
```

Superficies

MATLAB permite representar superficies, tanto si están definidas por una ecuación cartesiana explícita como si lo están por sus ecuaciones paramétricas. En cada caso puede realizar tres tipos de representación, a los que denominamos gráficos de malla, gráficos continuos y gráficos de líneas de nivel.

En el dibujo de superficies el color es un aspecto importante. En este capítulo se emplea siempre la paleta de colores que utiliza MATLAB por defecto.

Superficies definidas por su ecuación cartesiana explícita

Para representar una superficie $z = f(x, y)$ es necesario, en primer lugar, comprender el funcionamiento de la orden `meshgrid`.

• `[X Y]=meshgrid(x,y)` A partir de los vectores $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_m)$, genera una matriz X de dimensión m x n cuyas filas son m copias del x, y una matriz Y de dimensión m x n cuyas columnas son n copias del vector y, es decir:

$$X = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_1 & x_2 & \cdots & x_n \end{pmatrix}_{m\times n}, \qquad Y = \begin{pmatrix} y_1 & y_1 & \cdots & y_1 \\ y_2 & y_2 & \cdots & y_2 \\ \vdots & \vdots & & \vdots \\ y_m & y_m & \cdots & y_m \end{pmatrix}_{m\times n}$$

Teniendo esto en cuenta, para representar la superficie de ecuación cartesiana $z = f(x, y)$; $(x, y) \in [a,b]\times[c,d]$, se procede así:

• Se definen los vectores `x=linspace(a,b,n)`, `y=linspace(c,d,m)`.

• Se definen las matrices *X* e *Y* con la orden `[X Y]=meshgrid(x,y)`.

• A partir de las matrices *X* e *Y*, se genera la matriz *Z*:

$$Z = f(X,Y) = \begin{pmatrix} f(x_1,y_1) & f(x_2,y_1) & \cdots & f(x_n,y_1) \\ f(x_1,y_2) & f(x_2,y_2) & \cdots & f(x_n,y_2) \\ \vdots & \vdots & & \vdots \\ f(x_1,y_m) & f(x_2,y_m) & \cdots & f(x_n,y_m) \end{pmatrix}_{m\times n}$$

Una vez que se han obtenido las matrices *X*, *Y*, *Z* se pueden generar los diferentes tipos de gráficos que se estudian a continuación.

Gráficos de malla (`mesh`)

La superficie se representa mediante una malla, de aspecto parecido a una red de pesca, cuyos nudos están situados sobre la superficie correspondiente. Se generan con las siguientes órdenes:

• `mesh(X,Y,Z,C)` Representa el gráfico de malla de la función $z = f(x,y)$, dibujando las líneas de la rejilla que componen la malla con los colores especificados en el parámetro opcional *C*. *C* es una matriz con las mismas dimensiones que *X*, *Y*, *Z*, formada por números c_{ij} que especifican el color correspondiente al punto (x_{ij}, y_{ij}, z_{ij}). Si se omite este argumento se toma $C = Z$, con lo cual el color está relacionado con la cota del punto.

• `meshz(X,Y,Z,C)` Representa el gráfico de malla de la función $z = f(x,y)$ con una especie de cortina o telón en la parte inferior.

• `meshc(X,Y,Z,C)` Representa el gráfico de malla de la función $z = f(x,y)$ junto con las curvas de nivel proyectadas sobre el plano *XY*.

Gráficos continuos (`surf`)

En estos gráficos la superficie se representa como una lámina continua. Se generan con las siguientes órdenes:

• `surf(X,Y,Z,C)` Representa el gráfico de la función $z = f(x,y)$, realizando el dibujo con los colores especificados en el argumento opcional *C*.

• `surfc(X,Y,Z,C)` Representa el gráfico de la función $z = f(x,y)$ junto con las curvas de nivel proyectadas sobre el plano *XY*.

• `surfl(X,Y,Z,S)` Representa el gráfico de la función $z = f(x,y)$, iluminado con la fuente de luz especificada en el parámetro opcional *S*. *S* puede ser un vector `S=[Sx,Sy,Sz]` que indica la dirección de la fuente de iluminación o un vector `S=[az,el]` que señala el azimut y la elevación de la dirección de iluminación. Por defecto, la fuente de luz está desplazada respecto al punto de vista 45º en sentido contrario a las agujas del reloj.

El aspecto de una superficie generada con una orden `surf` se puede modificar con la orden `shading`:

• `shading faceted` La superficie aparece descompuesta en elementos delimitados por líneas; es la opción por defecto.

• `shading flat` Hace desaparecer la línea que delimita los elementos de la superficie.

• `shading interp` Difumina los bordes de los elementos.

Gráficos de curvas de nivel (`contour`)

En este tipo de gráficos la superficie se representa por sus cartas de nivel. Se generan con las siguientes órdenes:

• `contour(Z)` Dibuja el gráfico de curvas de nivel para la matriz $Z = f(X, Y)$. El número de curvas que se utilizan se elige automáticamente.

• `contour(Z,n)` Dibuja el gráfico de curvas de nivel para la matriz *Z*, usando *n* curvas.

• `contour3(Z)` Dibuja el gráfico de curvas de nivel para la matriz *Z*, en 3D. El número de curvas que se utilizan se elige automáticamente.

• `contour3(Z,n)` Dibuja el gráfico de curvas de nivel para la matriz *Z*, en 3D, usando *n* curvas.

• `pcolor(X,Y,Z)` Dibuja un gráfico de curvas de nivel correspondiente a las matrices *X*, *Y*, *Z*, utilizando una representación basada en densidades de colores. El aspecto del gráfico se puede modificar con la orden `shading`.

Ejemplo *Obtener los gráficos de malla, continuos y de curvas de nivel de la superficie de ecuación*

$$z = xe^{-(x^2+y^2)}, \qquad x,y \in [-2,2].$$

Observar los efectos que produce la orden `shading`.

Sol.:

```
clear, clc, close all
x=-2:0.1:2; y=x;
[X Y]=meshgrid(x,y);
Z=X.*exp(-(X.^2+Y.^2));
mesh(X,Y,Z), title('mesh')
meshz(X,Y,Z), title('meshz')
meshc(X,Y,Z), title('meshc')
surf(X,Y,Z), title('surf')
shading flat, title('flat')
shading interp, title('shading')
shading faceted, title('shading faceted')
surfc(X,Y,Z), title('surfc')
surfl(X,Y,Z), title('surfl')
contour(Z,20), title('contour')
contour3(Z,20), title('contour3')
pcolor(X,Y,Z), title('pcolor')
shading interp, title('shading interp')
close
```

En algunas ocasiones el procedimiento descrito (`[X Y]=meshgrid(x,y), Z=f(X,Y)`), no define una matriz *Z* con las mismas dimensiones que *X* e *Y*. Esta situación se presenta con la gráfica de los planos perpendiculares a los ejes cartesianos. El ejemplo siguiente muestra un procedimiento para salvar esta dificultad.

Ejemplo *Dibujar el plano* $z = 3$.

Sol.:

```
clear, clc, close all
x=-5:.1:5;
y=x;
[X Y]=meshgrid(x,y);
```

```
Z=3*ones(size(X));
surf(X,Y,Z), close
```

Superficies definidas por sus ecuaciones paramétricas

Para representar gráficamente una superficie de ecuaciones paramétricas

$$\begin{cases} x = f(u,v) \\ y = g(u,v) \\ z = h(u,v) \end{cases} \qquad (u,v) \in [a,b] \times [c,d],$$

se procede del modo siguiente:

• Se definen los vectores `u=linspace(a,b,m)`, `v=linspace(c,d,n)`.

• Se generan las matrices U y V, de dimensión m x n, con la orden `[U V]=meshgrid(u,v)`.

• Se definen las matrices $X = f(U,V)$, $Y = g(U,V)$ y $Z = h(U,V)$, teniendo en cuenta que las operaciones que se realicen sobre U y V deben ser elemento a elemento.

Ejemplo *Representar la esfera de ecuaciones paramétricas*

$$\begin{cases} x = \cos u \operatorname{sen} v \\ y = \operatorname{sen} u \operatorname{sen} v \\ z = \cos v \end{cases} \qquad (u,v) \in [0,2\pi] \times [0,\pi]$$

Sol.:

```
clear, clc, close all
u=linspace(0,2*pi,32);
v=linspace(0,pi,32);
[U V]=meshgrid(u,v);
X=cos(U).*sin(V);
Y=sin(U).*sin(V);
Z=cos(V);
surf(X,Y,Z)
close
```

Ficheros *m*

(Para la realización de los ejemplos de esta sección se deberá insertar un diskette floppy y teclear `path(path,'a:')` en la ventana de comandos detrás del "prompt" >>)

Hasta ahora se ha utilizado MATLAB en *modo interactivo*: las órdenes se escriben en la ventana de comandos y, a continuación, se ejecutan pulsando <*retorno*>. Otra posibilidad es trabajar en modo *batch*, esto es, ejecutando secuencialmente un conjunto de órdenes que han sido escritas previamente en un fichero ASCII. Estos ficheros reciben el nombre de **ficheros** *m*, debido a que su nombre tienen obligatoriamente extensión *m*. Se trata de ficheros permanentes, es decir, no se borran cuando se apaga el ordenador y pueden ser modificados cuantas veces se desee.

Creación de un fichero *m*

Para crear un fichero *m* se procede como sigue:

• Se elige en el menú la opción **File>New>M-file**, o bien se pica en el primer icono de la barra de herramientas. Esto nos lleva a un editor de texto en el que se escribe el fichero *m*.

• En la ventana del editor se escribe el conjunto de órdenes que van a constituir el fichero *m*. Recordemos que el signo `%` permite añadir comentarios.

• 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 se escribe el nombre del fichero, `nom_fichero.m`. No es necesario especificar la extensión *m*. Las reglas para dar nombre a un fichero son las mismas que las que rigen para las variables, excepto que no se distingue entre mayúsculas y minúsculas.

Ejecución de un fichero *m*

• Para comprobar el funcionamiento del fichero creado se activa la ventana de comandos de MATLAB, picando con el ratón en cualquier punto de la misma. A continuación se escribe el nombre del fichero **sin la extensión**.

```
>> nom_fichero
```

con lo cual se ejecutan secuencialmente las órdenes escritas en el fichero *m*.

• La orden `echo` permite visualizar las instrucciones de un fichero *m* a medida que se ejecutan.

• `echo on` Activa el `echo`.

• `echo off` Desactiva el `echo`. Es la opción por defecto.
• `echo` Conmuta entre `on` y `off`.

• La ejecución de un fichero *m* se puede interrumpir temporalmente con la orden

• `pause` Interrumpe la ejecución hasta que el usuario pulsa una tecla.
• `pause(n)` Interrumpe la ejecución durante *n* segundos.

• Se puede acceder al primer bloque de comentarios del fichero escribiendo

```
>> help nom_fichero
```

Modificación de un fichero *m*

• Si hay que hacer alguna modificación se vuelve al editor activando su ventana, se realizan los cambios oportunos y se guardan eligiendo en **File** la opción **Save**, o bien picando en el tercer icono de la barra de herramientas.

Una vez comprobado el funcionamiento del fichero se puede cerrar la ventana del editor, maximizar la de MATLAB y continuar trabajando en la forma habitual.

Los ficheros m se guardan, por defecto, en el directorio de trabajo actual. Es posible guardarlos en cualquier otro sitio indicando el *path* (trayectoria) correspondiente cuando se guarda el fichero.

• Para modificar un fichero *m* preexistente, se elige la opción **File** del menú y aquí la opción **Open...** Aparece una ventana en la que se selecciona el fichero que se quiere modificar.

Ejemplo *(a) Crear un fichero* ***matrices.m*** *en el que se defina la matriz*

$$A = \begin{pmatrix} 1 & 2 \\ -1 & 3 \end{pmatrix}$$

y se realicen los cálculos: A^2, A^t, A^{-1}, $|A|$.
(b) Observar el efecto de las órdenes ***echo*** *y* ***pause*** *en el fichero* `m` *anterior.*

Sol.:
(a)

```
A=[1 2; -1 3]
A^2
A'
inv(A)
det(A)
```

(b)

```
echo on
A=[1 2; -1 3], pause(1)
A^2, pause(1)
A', pause(1)
inv(A), pause(1)
det(A), pause(1)
echo off
```

Al ejecutar un fichero *m* las variables definidas en el mismo pasan a formar parte del espacio de trabajo, igual que si hubiesen sido definidas desde la línea de comandos.

Ficheros *m* de función

Tal como se ha visto, MATLAB dispone de un gran número de funciones predefinidas que pueden ser llamadas escribiendo su nombre. El usuario tienen la posibilidad de definir otras funciones utilizando unos ficheros *m* especiales, a los que denominamos *ficheros* m *de función*.

Para definir una función

$$\begin{aligned} f:\quad & \Re^n \rightarrow \Re^m \\ & (x_1,...,x_n) \mapsto f(x_1,...,x_n) = (y_1,...,y_m) \end{aligned}$$

se escribe un fichero *m* **cuyo nombre debe coincidir con el de la función** (en este caso `f.m`) con las siguientes órdenes:

```
function [y1,...,ym]=f(x1,...,xn)
y1=f1(x1,...,xn)
y2=f2(x1,...,xn)
... ... ... ... ... ... ...
ym=fm(x1,...,xn)
```

donde $f_1, ..., f_m$ indican las relaciones funcionales que definen las y_i a partir de las x_i.

Una vez guardado el fichero `f.m` la función `f` puede ser utilizada del mismo modo que las funciones predefinidas.

Es conveniente colocar en la primera línea del fichero un comentario explicativo de la función precedido del símbolo `%` en la línea o líneas que ocupe. La ejecución desde la ventana de comandos de la orden

```
>> help nom_fun
```

muestra en pantalla el comentario explicativo correspondiente a la función `nom_fun`.

Ejemplo *Definir la función fun1(x) =* x^3 *- 2x +cos 3x mediante un fichero* m.

Sol.:

```
% fun1 define la funcion
% x^3-2*x+cos(3*x)
function y=fun1(x)
y=x.^3-2*x+cos(3*x);
```

Tal como ha sido definida (utilizando `.^` en lugar de `^`), la función `fun1` admite argumento vectorial.

Una vez definida la función, se puede evaluar en un punto o en varios, pedir ayuda acerca de esta función desde la ventana de comandos...

```
>> fun1(3)
>> fun1(1:5)      (fun1 admite argumento vectorial)
>> help fun1
```

Ejemplo *Definir la función fun2(x,y)=* $\frac{x^2+y^2}{\sqrt{xy}}$ *mediante un fichero* m.

Sol.:

```
% fun2 define la funcion
% (x^2+y^2)/sqrt(x*y)
function z=fun2(x,y)
z=(x.^2+y.^2)/sqrt(x.*y);
```

Para evaluar esta función en el punto (2,3) se escribe, en la ventana de comandos:

```
>> fun2(2,3)
```

Ejemplo *Definir mediante un fichero* m *una función que admita como argumentos los coeficientes a, b, c de la ecuación de segundo grado* $ax^2 + bx + c = 0$ *y devuelva sus raíces.*

Sol.:

```
% fun3(a,b,c) calcula las raíces
% de la ecuación de segundo grado a*x^2+b*x+c=0
function [r1,r2]=fun3(a,b,c)
r1=(-b+sqrt(b.^2-4*a.*c))/(2*a);
r2=(-b-sqrt(b.^2-4*a.*c))/(2*a);
```

Podemos utilizar esta función para calcular las raíces de la ecuación $x^2 + 4x - 5 = 0$; para ello se escribe en la ventana de comandos

```
>> [x1,x2]=fun3(1,4,-5)
```

Organización de ficheros y directorios

MATLAB, como cualquier otro programa de aplicación, consta de un gran número de ficheros de diferentes tipos, agrupados en diferentes directorios.

• `path` Muestra el *path* de MATLAB, es decir, todos los directorios donde MATLAB busca los ficheros que necesita.

• `path(path,'a:')` Añade al *path* el directorio del disco floppy.

Cálculo Numérico

Cálculos en Álgebra Lineal

Sistemas de ecuaciones lineales

Consideremos un sistema de *m* ecuaciones lineales con *n* incógnitas

$$\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ \vdots \qquad\qquad \vdots \\ a_{m1}x_1 + \ldots + a_{mn}x_n = b_m \end{cases}$$

Brevemente, este sistema lo podemos escribir como

$$Ax = b$$

donde

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

son, respectivamente, la matriz de coeficientes del sistema, el vector de las incógnitas y el vector de los términos independientes.

Una vez introducida la matriz *A* y el vector *b*, la solución del sistema se obtiene escribiendo:

```
>> A\b
```

que es equivalente a

```
>> inv(A)*b
```

Pueden presentarse tres casos:

• Si el sistema $Ax = b$ es compatible determinado, MATLAB calcula su solución. Por ejemplo:

```
>> A=[2 1 1;1 -1 1;3 -1 2]
>> b=[3;0;2]
>> rA=rank(A); rAB=rank([A b])      (rA = rAb ⇒ Sistema Comp. Det.)
>> x=A\b
```

• Si el sistema $Ax = b$ es compatible indeterminado, MATLAB calcula una sola de las infinitas soluciones. Por ejemplo:

```
>> A=[1 1 1;1 -1 0]
>> b=[1;0]
>> rA=rank(A); rAB=rank([A b])      ( Sistema Compatible Indeterminado)
>> x=A\b
```

• Si el sistema $Ax = b$ es incompatible, MATLAB devuelve un vector x que hace mínima la cantidad $||Ax - b||$. Por ejemplo:

```
>> A=[2 1;1 -1;-1 -1]
>> b=[1;0;3]
>> rA=rank(A); rAB=rank([A b])      ( Sistema Incompatible)
>> x=A\b
```

Alternativamente, si el sistema es compatible determinado se puede usar la función `rref`, del siguiente modo:

```
>> A=[2 1 1;1 -1 1;3 -1 2]
>> b=[3;0;2]
>> rA=rank(A); rAB=rank([A b])      (rA = rAb ⇒ Sistema Comp. Det.)
>> rref([A b])
```

El resultado es una matriz cuya última columna contiene la solución del sistema de ecuaciones.

Resolución de ecuaciones

Para determinar las raíces de una ecuación $f(x) = 0$, se dispone de la orden `fzero` cuya sintaxis es:

• `fzero('f',x0)` Calcula una raíz de la ecuación $f(x) = 0$ próxima al punto x_0. f es una función predefinida de MATLAB, una función definida en un fichero *m* o una cadena de caracteres sin igualar a cero.

• `fzero('f',x0,tol)` Realiza el mismo cálculo con la tolerancia indicada por el parámetro opcional *tol*, cuyo valor por defecto es *eps*.

• `fzero('f',x0,tol,traza)` Si el parámetro opcional *traza* es distinto de cero, muestra información en cada iteración.

Ejemplo *Calcular las raíces de la ecuación* $x - e^{-x} = 0$.

Sol.: Para conocer el número y valor aproximado de las raíces se dibuja la gráfica de la función $f(x) = x - e^{-x}$ y se le añade una malla:

```
>> fplot('x-exp(-x)',[-3 3]), grid on
```

Se observa que hay una raíz próxima a cero. Para calcular su valor utilizamos la orden `fzero`:

```
>> fzero('x-exp(-x)',0)
```

Interpolación y ajuste

Interpolación unidimensional

Para interpolar la función $y = f(x)$ que toma los valores $y_0, y_1, \ldots, y_n$, en los puntos $x_0, x_1, \ldots, x_n$, se dispone de la orden `interp1` que tiene la siguiente sintaxis:

• `interp1(x,y,v,'método')` Realiza la interpolación de los datos definidos por los vectores $x = (x_0, x_1, \ldots, x_n)$ e $y = (y_0, y_1, \ldots, y_n)$ en los puntos especificados en el vector *v*, cuyas coordenadas han de estar comprendidas entre x_0 y x_n. El parámetro opcional `método` indica el tipo de interpolación; puede tomar los siguientes valores:

- `linear` Realiza una interpolación lineal a trozos. Es la opción por defecto.

- `spline` Realiza una interpolación con splines cúbicos.

- `cubic` Realiza un tipo de interpolación con polinomios de tercer grado. Sólo se puede utilizar con nodos equidistantes.

Ejemplo *Interpolar los datos*

x	2	4	6	8	10
y	7	20	47	42	70

mediante interpolación lineal a trozos, splines cúbicos y polinomios de tercer grado. En una misma figura, obtener la gráfica de los datos y las de las diferentes funciones de interpolación utilizando distintos colores.

Sol.:

```
clear, clc
x=2:2:10;
y=[7 20 47 42 70];
a=2:.1:10;
s1=interp1(x,y,a);
a2=interp1(x,y,a,'spline');
a3=interp1(x,y,a,'cubic');
plot(x,y,'ko',a,s1,'r',a,s2,'b',a,s3,'m')
close
```

Ajuste por mínimos cuadrados

Dados n puntos (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), es posible encontrar un polinomio de ajuste en el sentido de los mínimos cuadrados. Si el grado del polinomio de ajuste es $n - 1$, se obtiene el polinomio de interpolación. Estos polinomios se obtienen con la orden:

• `polyfit(x,y,n)` Devuelve el polinomio de mínimos cuadrados de grado n que se ajusta a los datos definidos por los vectores x e y.

Ejemplo *Determinar los polinomios de mínimos cuadrados de grados* 1, 2, 6 *y* 9 *para*

los datos

x	1	2	3	4	5	6	7	8	9	10
y	3	6	7	12	9	12	19	15	18	20

y representar gráficamente los resultados.

Sol.:

```
clear, clc, close all
x=1:10; y=[3 6 7 12 9 12 19 15 18 20];
u=1:.1:10; % variable para la representación gráfica.
subplot(2,2,1)
p=polyfit(x,y,1); v=polyval(p,u); plot(x,y,'ro',u,v)
title('Polinomio de grado 1')
subplot(2,2,2)
p=polyfit(x,y,2); v=polyval(p,u); plot(x,y,'ro',u,v)
title('Polinomio de grado 2')
subplot(2,2,3)
p=polyfit(x,y,6); v=polyval(p,u); plot(x,y,'ro',u,v)
title('Polinomio de grado 6')
subplot(2,2,4)
p=polyfit(x,y,9); v=polyval(p,u); plot(x,y,'ro',u,v)
title('Polinomio de interpolación')
close
```