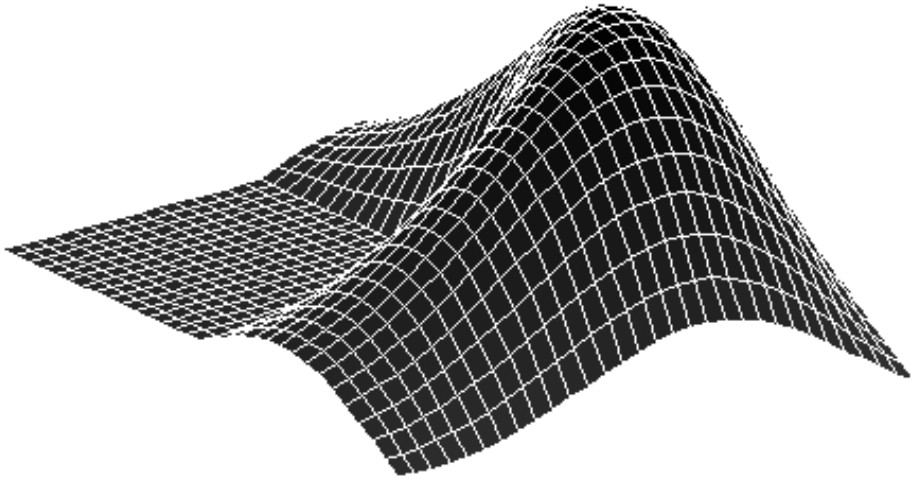
Programación: Introducción a

M



Primeros pasos.

Qué es MATLAB.

MATLAB es un lenguaje de alto nivel orientado al desarrollo de cálculos técnicos.

Integra cálculo, visualización y programación en un entorno interactivo de fácil manejo donde los problemas y las soluciones se expresan en la notación matemática habitual.

El elemento de información básico en MATLAB es una tabla a la que no hace falta asignar dimensión con antelación. Ésto permite abordar problemas que requieren una formulación vectorial o matricial en mucho menos tiempo de lo que se tardaría con un lenguaje escalar no interactivo tipo C o FORTRAN. De hecho, el nombre MATLAB es una abreviatura de MATrix LABoratory.

Cómo entrar y salir de MATLAB.

Para entrar en MATLAB basta pulsar dos veces seguidas el botón izquierdo del ratón sobre el icono correspondiente. Aparece entonces la **ventana de comandos** y, en su interior, el indicador (*prompt*) >>. MATLAB está listo para recibir nuestras instrucciones.

Para salir del programa basta teclear quit y pulsar la tecla Enter (toda instrucción en MATLAB debe concluir pulsando esta tecla por lo que, de ahora en adelante, omitiremos dicha acción). Otra opción es desplegar el menú File y desplazar la barra hasta la opción Exit MATLAB.

Para obtener ayuda.

El comando help constituye la forma más básica de conocer la sintaxis y el comportamiento de una función en particular. Su formato es

```
help nombre_de_funcion
```

y la información solicitada aparece directamente en la ventada de comandos. En dicha información MATLAB emplea letras mayúsculas para referirse a la función y a los nombres de variables, con objeto de destacarlos del resto del texto. Sin embargo, a la hora de utilizar la función en cuestión siempre escribiremos su nombre con letras minúsculas. MATLAB distingue entre ambos tipos de letras (*case sensitive*) y todos los nombres de función son, de hecho, en minúscula.

Si no queremos que la información aparezca en la ventana de comandos podemos recurrir a una ventana auxiliar, la **ventana de ayuda**. Ésta se invoca mediante la orden helpwin o seleccionando la opción Help Window del menú Help. Se puede acceder directamente a una función concreta mediante la instrucción helpwin nombre_de_funcion. Una ventaja de la ventana de ayuda es que provee enlaces con otras cuestiones relacionadas con la información que hayamos solicitado.

Otro comando útil es lookfor. La instrucción lookfor nombre muestra todas aquellas funciones de MATLAB en cuyo texto de ayuda asociado (concretamente en la primera línea), aparece la cadena nombre. Intercalando el modificador -all entre lookfor y nombre, la búsqueda se efectúa en todo el texto de ayuda.

Primeros pasos con matrices.

La mejor forma de empezar a trabajar con MATLAB es aprender a manejar matrices.

Una matriz es una tabla rectangular numérica. Casos particulares de matrices son los vectores (matrices con una sóla fila o una sóla columna). También los números o escalares pueden verse como matrices, con una sóla fila y una sóla columna. Pero, para mayor claridad, distinguiremos siempre entre escalares y matrices propiamente dichas.

La forma básica de introducir una matriz en MATLAB es dando explícitamente sus elementos. Por ejemplo,

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

Las reglas son sencillas:

1. Separar los elementos de cada fila por un espacio en blanco (o una coma).
2. Utilizar punto y coma para indicar el final de cada fila.
3. Rodear la lista con corchetes.

La matriz queda almacenada en el espacio de trabajo (*workspace*) de MATLAB y nos podemos referir a ella simplemente como A.

La matriz A tiene una propiedad interesante y es que sus columnas, sus filas y sus diagonales suman lo mismo: 34. Para comprobarlo con MATLAB vamos a introducir los comandos que aparecen en la siguiente tabla:

sum(A)	Si A es un vector, da la suma de sus componentes. Si es una matriz, da un vector fila con las sumas de cada columna.
A'	Da la matriz conjugada traspuesta de A.
diag(A)	Da la diagonal principal de A en forma de vector columna.
fliplr(A)	Da una matriz cuyas columnas son las de A en orden inverso.

Si en lugar de invertir el orden de todas las columnas quisiéramos, pongamos por caso, intercambiar las columnas segunda y tercera, utilizaríamos la instrucción `A(:,[1 3 2 4])`.

Para referirnos a un elemento concreto de una matriz utilizamos subíndices. Concretamente, el elemento común a la fila i y a la columna j de `A` se denota por `A(i,j)`. En el caso de un vector basta emplear un único subíndice. Ésto también es posible con una matriz bidimensional, siempre que la miremos como el vector columna que resulta de concatenar las columnas de la matriz original. Volviendo a nuestro ejemplo, las instrucciones `A(1,3)` y `A(9)` hacen referencia al mismo elemento de `A`.

Si intentamos utilizar el valor de un elemento fuera de la matriz, MATLAB produce un mensaje de error. Pero si almacenamos un valor en un elemento fuera de la matriz, ésta ajusta su tamaño automáticamente. Compruébalo con la instrucción `A(4,5)=17`.

Para especificar porciones de una matriz es muy útil el operador *dos puntos* `:`. He aquí algunos ejemplos de uso:

- `A(m:n,j)` indica los elementos que van desde el `m`-ésimo puesto hasta el `n`-ésimo puesto en la columna `j`.
- `A(:,j)` indica todos los elementos de la columna `j`.
- `A(:,end)` proporciona todos los elementos de la última columna.

El operador *dos puntos* también permite generar con comodidad vectores uniformemente espaciados. La expresión `1:10` da un vector fila que contiene todos los enteros del 1 al 10. Para que el espaciado no sea unitario se especifica un incremento. Por ejemplo, `100:-7:50` da un vector fila cuyos elementos son los enteros del 100 a 50 en intervalos decrecientes de 7 unidades.

La matriz que hemos elegido como ejemplo para desarrollar esta sección es lo que se conoce como un cuadrado mágico. MATLAB dispone de una función, `magic`, que genera este tipo de cuadrados. Así, `magic(n)` nos da una matriz `n` por `n` con propiedades análogas a las de `A`. Concretamente, `magic(4)` es otra forma de introducir la matriz A.

Pero `magic` no es la única instrucción que genera matrices de manera automática. Hay otras muchas. Por ejemplo, `eye(n)` da la matriz identidad `n` por `n`, `zeros(n,m)` da una matriz con `n` filas y `m` columnas cuyos elementos son todos nulos, `ones(n,m)` da una matriz con `n` filas y `m` columnas cuyos elementos son todos unos y `rand(n,m)` da una matriz con `n` filas y `m` columnas cuyos elementos son números aleatorios entre 0 y 1.

Expresiones.

Como otros lenguajes de programación, MATLAB proporciona expresiones matemáticas pero, a diferencia de la mayoría, estas expresiones tienen que ver con matrices. Cuatro son las componentes principales de dichas expresiones: variables, números, operadores y funciones.

Las variables en MATLAB no requieren declaraciones de tipo ni dimensionalización previa. Los nombres de variables consisten en una letra seguida de varias letras, dígitos o signos de subrayado. MATLAB sólo utiliza los 31 primeros caracteres del nombre de una variable y distingue, como ya comentamos, entre letras mayúsculas y minúsculas (A y a no representan a la misma variable). Para ver la matriz asignada a una variable simplemente introducimos el nombre de ésta.

Cuando ejecutamos una instrucción que no contiene ninguna asignación, MATLAB almacena el resultado en una variable temporal de nombre ans.

Con los números, MATLAB utiliza la notación decimal habitual. Pero también entiende la notación científica ($4.8\ 10^5$ se escribe 4.8e5) y la notación de número complejo ($3+2i$ se escribe 3+2i o 3+2j).

MATLAB permite efectuar con comodidad las operaciones habituales entre matrices y/o escalares. Las recogemos en la siguiente tabla:

Operaciones aritméticas con MATLAB		
Notación	*Tipo de datos*	*Efecto*
a+b	a y b escalares	a más b
a-b		a menos b
A+B	A y B matrices de la misma dimensión	A más B
A-B		A menos B
A+b	A matriz, b escalar	A cada elemento de A se le suma b
A-b		A cada elemento de A se le resta b
a*b	a y b escalares	a multiplicado por b
a/b		a dividido por b
A*B	A y B matrices (el número de columnas de A coincide con el filas de B)	A multiplicado por B
A.*B	A y B matrices de la misma dimensión	Cada elemento de A se multilplica por el correspondiente de B
b*A	A matriz, b escalar	Cada elemento de A se multiplica por b
A/b		Cada elemento de A se divide por b
A./B	A y B matrices de la misma dimensión	Cada elemento de A se divide por el correspondiente de B

A^b	A matriz cuadrada, b escalar (entero positivo)	Potencia b-ésima de A
A.^B	A y B matrices de la misma dimensión	Cada elemento de A se eleva al correspondiente de B
A.^b	A matriz, b escalar	Cada elemento de A se eleva a b
b.^A		b elevado a cada uno de los elementos de A
a'	a escalar (complejo)	Conjugado de a
A'	A matriz	Traspuesta conjugada de A
A.'	A matriz	Traspuesta sin conjugar de A

El orden de prioridad entre las operaciones es el usual, pudiendo alterarse a voluntad introduciendo paréntesis adecuadamente.

MATLAB proporciona un gran número de funciones matemáticas elementales (para obtener una lista completa de las mismas teclea help elfun). Dichas funciones pueden actuar sobre escalares y sobre matrices (elemento a elemento). Así mismo, pueden operar sobre argumentos complejos. Tomar la raíz cuadrada o el logaritmo de un número negativo no produce un error en MATLAB sino el correspondiente resultado complejo.

MATLAB también posee funciones matemáticas más avanzadas (teclea help specfun) y funciones específicas para matrices (teclea help elmat).

Ciertas funciones proveen valores especiales:

pi	Número Π.
i, j	Unidad imaginaria.
eps	Precisión de la máquina.
realmin	Menor número manejable.
realmax	Mayor número manejable.
Inf	Infinito. Aparece al dividir un valor no nulo por cero o al evaluar expresiones matemáticas que exceden realmax.
NaN	No es un número. Aparece al evaluar expresiones del tipo 0/0 o Inf-Inf (indeterminaciones).

Los nombres de funciones no están reservados. No es aconsejable, pero podemos utilizar cualquiera de ellos para designar una nueva variable. Por ejemplo, podemos hacer abs=2, pi=4 o sin=8. Para restaurar las funciones originales utilizamos el comando clear seguido del nombre o los nombres (separados por un espacio) de las funciones que queramos restaurar. Por ejemplo, clear abs pi sin.

La ventana de comandos.

Podemos modificar a nuestro antojo diversos aspectos de la ventana de comandos.

Uno de ellos es el formato numérico con que los valores son mostrados en pantalla. Para controlarlo utilizamos el comando format:

format short	Formato por defecto. Notación decimal con cinco dígitos en total.
format short e	Notación científica con cinco dígitos en total para la mantisa.
format short g	Alterna entre las dos anteriores.
format long	Notación decimal con quince dígitos en total.
format long e	Notación científica con quince dígitos en total para la mantisa.
format long g	Alterna entre las dos anteriores.
format bank	Notación decimal con tres dígitos en total.
format rat	Notación racional.
format hex	Notación hexadecimal.

Coge un vector, por ejemplo x = [4/3 1.2345e-6], y observa las diferencias entre los distintos formatos. Conviene dejar claro que el formato elegido no afecta para nada a la manera en que MATLAB trabaja internamente con los números.

Por lo que hemos visto hasta ahora, si escribimos una instrucción y pulsamos la tecla Enter, MATLAB muestra automáticamente los resultados en pantalla. Podemos evitarlo poniendo un punto y coma al final de la instrucción, lo cual es de utilidad cuando el resultado que se espera es muy extenso o simplemente no es relevante que aparezca en pantalla. Compara entre introducir A = magic(100) y A = magic(100);. Si alguna vez olvidamos poner punto y coma en una instrucción de este tipo, podemos cortar el proceso pulsando a la vez Ctrl y c.

Si una instrucción no cabe en una sola línea, podemos utilizar tres puntos seguidos para cortarla donde queramos y continuarla después en la siguiente línea. Por ejemplo:

```
S = 1 - 1/2 + 1/3 - 1/4 + 1/5 -1/6 + 1/7 …
-1/8 +1/9 - 1/10 + 1/11-1/12;
```

Las teclas con flechas permiten recuperar comandos que hayamos escrito con anterioridad en la misma sesión. Por ejemplo, supongamos que queremos escribir rho = (1+sqrt(5))/2 pero nos equivocamos al teclear y escribimos sqt. MATLAB da un mensaje de error indicando que la variable o función sqt no está definida. En lugar de volver a escribir la orden anterior, presionamos la tecla ↑, con lo que dicha orden aparece en la línea de

comandos. A continuación, utilizamos la tecla ← para situarnos delante de la letra te e insertamos la erre que faltaba.

El espacio de trabajo.

El espacio de trabajo es la zona de memoria donde se almacenan las variables que vamos introduciendo desde la línea de comandos. Podemos utilizar las órdenes who y whos para mostrar el contenido actual del espacio de trabajo. La primera da una lista abreviada mientras que la segunda proporciona el tamaño y el tipo de datos asociados a cada variable.

Para borrar todas las variables almacenadas en el espacio de trabajo, utilizamos el comando clear sin más.

Es posible guardar en un archivo el contenido del espacio de trabajo y recuperarlo después durante la misma sesión u otra posterior.

Para guardar todo el contenido del espacio de trabajo utilizamos el comando save. Por ejemplo, save [a:\]sesion (lo que aparece entre corchetes es opcional) guarda todo el contenido del espacio de trabajo en el archivo sesion.mat (MATLAB añade automáticamente la extensión mat, de ahí que los archivos así generados se conozcan como archivos MAT). Si no pones a:\ el archivo se almacenará en tu directorio de trabajo, pero si pones a:\ el archivo se guardará en un disquete que, previamente, deberás haber introducido. Es posible guardar sólo variables concretas especificando el nombre de las mismas después del nombre del archivo. Por ejemplo, save [a:\]sesion x y z sólo guarda las variables x, y y z en el archivo sesion.mat. Por defecto los datos se almacenan en formato binario (no legible por nosotros), pero es posible hacerlo en formato ASCII (legible por nosotros) añadiendo los modificadores -ascii o -ascii -double al final de la instrucción. El primero utiliza 8 dígitos para cada número y el segundo utiliza 16. Si elegimos este formato conviene:

1. Especificar una extensión para el nombre del archivo distinta de mat. Las más usuales son dat y txt.
2. Guardar a lo sumo una variable.

Por ejemplo, save [a:\]sesion.dat x -ascii. Otro modificador que podemos utilizar con save es -append. Con él, los datos se agregan a un archivo MAT ya existente. Por ejemplo, save [a:\]sesion x -append añade la variable x al archivo sesion.mat (que ya debe existir).

Para recuperar la información almacenada en un archivo MAT utilizamos el comando load. Por ejemplo, load [a:\] sesion carga el contenido de sesion.mat en el espacio de trabajo. Pondremos a:\ o no según el archivo esté en un disquete o en nuestro directorio de trabajo. Si el archivo MAT contiene tres variables llamadas A, B y C, al cargarlo dichas variables retornan al espacio de trabajo sobreescribiendo las que ya pudieran existir con el mismo nombre.

El efecto del comando load sobre un archivo en formato ASCII es completamente distinto, de ahí las recomendaciones anteriores para guardar datos en dicho formato. Por ejemplo, load [a:\]sesion.dat crea una nueva variable en el espacio de trabajo llamada sesion. Esto obliga a que el contenido del fichero sesion.dat tenga formato de matriz numérica. Si no es así, se producirá

un error de lectura. El contenido es admisible (es una matriz 2 por 4), pero no lo es (la segunda fila tiene una columna menos que la primera).

Finalizamos este apartado con una breve descripción del comando diary. Si escribimos diary [a:\]sesion.txt se creará el archivo sesion.txt, en el cual se hará una copia literal de cuanto escribamos desde ese momento así como de los resultados que se vayan obteniendo. El proceso de copiado finalizará cuando escribamos diary off. Luego, con un procesador de texto, podremos abrir el archivo sesion.txt y repasar todos los pasos que hemos dado en la sesión.

Archivos M.

Los archivos M son archivos de texto (archivos ASCII) que contienen instrucciones propias de MATLAB. Se llaman así porque su nombre tiene (obligatoriamente) la extensión m.

Como archivos de texto, los archivos M se pueden crear con cualquier procesador. Recomendamos, no obstante, utilizar el editor que trae incorporado MATLAB. Para ello, cuando queramos crear un nuevo archivo M, debemos seguir el siguiente esquema:

Creación de un nuevo archivo M	
Llamada al editor	Si todavía no estamos en nuestro directorio de trabajo, nos pasamos a él.
	Tecleamos edit (aparece una ventana llamada MATLAB Editor/Debugger y en su interior otra de nombre Untitled1).
Creación del archivo	Escribimos el texto del archivo.
Guardamos lo escrito	En el menú File elegimos Save As (aparece una ventana llamada Guardar Como).
	Si queremos guardar el archivo en un disquete, ascendemos de nivel hasta llegar a Mi Pc. Una vez ahí pinchamos en Disco de 3½ (A:).
	En el campo Nombre de archivo escribimos el nombre que queramos ponerle, sin la extensión.
	Pulsamos el botón Guardar.
Salimos del editor	En el menú File elegimos Exit Editor/Debugger

Para editar un archivo ya existente, con objeto de revisarlo y/o modificarlo, seguimos el mismo esquema pero con las siguientes variantes:

- Invocamos el editor con edit [a:\]nombre_del_archivo.
- Si cambiamos el texto y queremos guardar las modificaciones en el mismo archivo, elegimos Save en el menú File. Si queremos guardarlas en otro archivo, elegimos Save As en el menú File y seguimos los mismos pasos del esquema anterior.

En cualquier momento podemos conocer los archivos M existentes en nuestro directorio de trabajo tecleando what. También podemos revisar el contenido de un archivo M sin necesidad de llamar al editor. Basta teclear type [a:\]nombre_del_archivo.

Hay dos tipos de archivos M: de guión y de función. Los archivos M de guión son los más simples:

- No tienen argumentos de entrada ni de salida.
- Son útiles para automatizar bloques de instrucciones y cálculos que deben efectuarse repetidamente.
- Pueden operar sobre datos ya existentes en el espacio de trabajo o sobre datos que ellos mismos introduzcan.
- Cualquier variable creada por uno de estos archivos permanece en el espacio de trabajo una vez que finaliza su lectura.

En cambio, los archivos M de función:

- Aceptan argumentos de entrada y salida.
- Sirven para extender el lenguaje de MATLAB creando nuestras propias funciones.
- Tienen su propio espacio de trabajo reservado, donde pueden definirse variables propias (locales) que no afectan al espacio de trabajo general.

La estructura del texto de un archivo M de guión es totalmente libre (siempre que se ajuste, por supuesto, a las instrucciones de MATLAB). Además, podemos y debemos incluir comentarios que aclaren el contenido del archivo. Tales comentarios han de ir precedidos por el símbolo de tanto por ciento: %. Por ejemplo, para crear un archivo M de guión que reproduzca el ejemplo de construcción por bloques, escribiremos lo siguiente:

```
% Ejemplo de construccion de una matriz por bloques.
% Apuntes sobre MATLAB, pagina 9.
B1 = [1 -3 5; 2 3 4; -7 8 9]
B2 = 5*eye(3)
% Como B1 y B2 van a estar a un mismo nivel,
% deben tener el mismo numero de filas.
B3 = zeros(2,1)
B4 = ones(2,3)
B5 = fliplr(2*eye(2))
%  Como B3, B4 y B5 van a estar a un mismo nivel,
% deben tener el mismo numero de filas.
A = [B1 B2; B3 B4 B5]
% Los dos niveles deben tener el mismo numero de columnas.
```

Si hemos guardado el archivo con el nombre bloques.m, bastará escribir bloques en la línea de comandos para que se ejecuten de una vez todas las instrucciones.

La estructura de un archivo M de función es un poco más complicada. Sus elementos básicos son una línea de definición, una línea H1, texto de ayuda, el cuerpo de la función y comentarios adicionales. Por ejemplo, para crear una función que calcule la superficie de un círculo, éste podría ser el contenido del archivo:

```
function s = supcirc(r)
% SUPCIRC Superficie de un circulo
% SUPCIRC(R), donde R es una matriz, calcula la superficie de los círculos
% cuyos radios son los coeficientes de la matriz.
s = pi * r.^2;
```

La primera línea de texto es la línea de definición. En ella hay que destacar cuatro cosas:

- La palabra clave `function`. Indica a MATLAB que el archivo M contiene una función.
- El argumento de salida `s`. En este caso sólo hay uno, pero podría haber más. Cuando ésto ocurre, deben encerrarse todos entre corchetes y separarse por comas. También puede ocurrir que no haya argumentos de salida y en tal caso utilizaremos dos corchetes vacíos: `[ ]`.
- El nombre de la función `supcirc`. Las reglas para los nombres de las funciones son las mismas que para las variables. Importante: el nombre del archivo M debe coincidir con el de la función.
- El argumento de entrada `r`. Va encerrado entre paréntesis. Puede haber más de uno, en cuyo caso irán separados por comas.

Éstos son otros ejemplos válidos de líneas de definición:

```
function [x,y,z] = esfera(theta, phi, rho)

function [ ] = sinsalida(x)
```

La segunda línea de texto es la línea H1. Se llama así porque constituye la primera línea del texto de ayuda. Es la línea de comentarios que sigue inmediatamente a la línea de definición y, como tal, debe ir precedida por el símbolo `%`. Cuando tecleemos `help supcirc` ésta será la primera línea que aparezca. Además, el comando `lookfor` sólo busca por defecto en dicha línea. Es importante que sea lo más descriptiva posible.

El texto de ayuda (líneas tres y cuatro en nuestro ejemplo) lo forman todas las líneas de comentario que siguen inmediatamente a la línea H1. En él se explica de forma más detallada lo que hace la función.

El cuerpo de la función se reduce en nuestro caso a una línea de asignación, la quinta. Pero en general puede contener también llamadas a otras funciones, instrucciones de control de flujo, instrucciones interactivas de entrada y salida de datos, cálculos, comentarios y líneas en blanco.

Una vez creada la función, si queremos calcular las superficies de los círculos de radio 6, 7 y 24 bastará escribir lo siguiente en la línea de comandos:

```
x = [6 7 24]
y = supcirc(x)
```

Observa que los nombres de las variables de entrada y salida no tienen por qué coincidir con los que aparecen en el archivo. Las variables r y s son locales y no intervienen para nada en el espacio de trabajo general.

Durante la ejecución de un archivo M, ya sea de guión o de función, puede ser muy útil mostrar determinados mensajes por pantalla. Ésto se consigue con la instrucción disp('mensaje')[1]. Si dicho mensaje es consecuencia de un error que obliga a finalizar la ejecución, emplearemos la orden error('mensaje'). El comando return permite igualmente finalizar la ejecución antes de alcanzar el final de archivo.

También podemos, con la orden input, pedir que se introduzcan datos por teclado. Si el archivo contiene la instrucción

```
n = input('mensaje');
```

al ejecutarse aparecerá mensaje en la ventana de comandos y MATLAB esperará que escribamos un dato. Cuando lo escribamos (y pulsemos Enter) dicho dato se almacenará en la variable n. El dato en cuestión debe ser una matriz, aunque también es admisible una cadena de caracteres. En este último caso, debemos escribir la cadena encerrada entre apóstrofes.

En realidad, una cadena de caracteres es un vector donde cada componente es un caracter. Si respondemos a la pregunta con 'ejemplo' y luego le pedimos a MATLAB que nos diga quién es n(5), nos responderá que p.

Las cadenas de caracteres, combinadas con las funciones eval y feval, dan una flexibilidad muy grande al lenguaje de MATLAB. Lo veremos en el último apartado.

Es posible detener temporalmente la ejecución de un archivo M mediante el comando pause. Si escribimos pause sin más, la ejecución no volverá a empezar hasta que pulsemos una tecla. Si escribimos pause(n), la ejecución se reanudará tras n segundos. Ésto puede ser muy útil cuando el archivo contenga comandos que generen gráficos, pues nos permitirá ir viéndolos poco a poco.

[1] En general, la orden disp(x), donde x es una matriz, sirve para que el contenido de dicha matriz aparezca en pantalla sin ningún nombre antecediéndolo.

Control de flujo.

MATLAB tiene cinco construcciones de control de flujo:

- enunciados if,
- enunciados switch,
- bucles while,
- bucles for y
- enunciados break.

El enunciado if evalúa una expresión lógica y ejecuta un grupo de instrucciones según el valor de la misma. En su forma más simple, su estructura es

```
if expresion_logica
        instrucciones
end
```

Si la expresión es verdadera, MATLAB ejecuta todas las órdenes entre las líneas if y end. Si la expresión es falsa, MATLAB no ejecuta dichas instrucciones. Los comandos opcionales elseif y else permiten la ejecución alternativa de varios grupos de instrucciones. Por ejemplo,

```
if expresion_1
        instrucciones_1
elseif expresion_2
        instrucciones_2
else
        instrucciones_3
end
```

hace lo siguiente:

1. Evalúa *expresion_1*.
2. Si es verdadera, ejecuta *instrucciones_1* y termina. Si es falsa, evalúa *expresion_2*.
3. Si *expresion_2* es verdadera, ejecuta *instrucciones_2* y termina. Si es falsa, ejecuta *instrucciones_3*.

En toda expresión lógica aparecen los llamados operadores relacionales: < (menor que), <= (menor o igual que), > (mayor que), >= (mayor o igual que), == (igual que) y ~= (distinto de). Estos operadores permiten comparar dos escalares, dos matrices de la misma dimensión (elemento a elemento) o un escalar con una matriz (cada elemento de la matriz se compara con el escalar). En el primer caso, la expresión vale 1 si es verdadera y 0 si es falsa. En los otros dos, la expresión da un matriz de ceros y unos de igual dimensión que la o las matrices implicadas. Se entenderá que la expresión es verdadera cuando todos los coeficientes sean unos. Dos expresiones lógicas pueden a su vez

compararse entre si mediante los operadores lógicos & y |. El enunciado expresion_1 & expresion_2 es verdadero si y sólo si ambas expresiones lo son, mientras que el enunciado expresion_1 | expresion_2 es verdadero si y sólo si al menos una de las dos expresiones lo es. El operador ~ sirve para negar una expresión. Así, ~expresion es verdadera si y sólo si expresion es falsa y viceversa.

El enunciado switch ejecuta ciertas instrucciones en función del valor de una variable o expresión. Su estructura es la siguiente:

```
switch expresion
        case valor_1
                instrucciones_1
        case valor_2
                instrucciones_2
        •
        •
        •
        otherwise
                otras_instrucciones
end
```

Como vemos el enunciado consta de:

1. La palabra switch seguida de una expresión a evaluar.
2. Varios grupos case. Cada grupo consiste en una primera línea con la palabra clave case seguida de un posible valor para *expresion*. Las líneas siguientes del grupo contienen las instrucciones a ejecutar en caso de que *expresion* tome ese valor. Sólo se ejecuta el primer grupo case cuyo valor coincide con el de la expresión.
3. Un grupo otherwise. Consiste en una primera línea con dicha palabra clave seguida de otras líneas con las órdenes a ejecutar en caso de que el valor de *expresion* no coincida con ninguno de los establecidos en los grupos case.
4. Una línea con la palabra clave end.

El bucle while ejecuta una instrucción o un grupo de instrucciones mientras que cierta expresión de control sea verdadera. Su estructura es la siguiente:

```
while expresion
        instrucciones
end
```

El bucle for repite una instrucción o grupo de instrucciones un número predeterminado de veces. Su estructura es la siguiente:

```
for indice = comienzo : incremento : fin
        instrucciones
end
```

Puede especificarse cualquier incremento, incluso negativo (si no se especifica ninguno se toma por defecto 1). Veamos un ejemplo de utilización construyendo la sucesión de Fibonacci. Ésta se define por recurrencia como sigue:

$$\begin{cases} a_1=1,\ a_2=2 \text{ y} \\ a_n=a_{n-1}+a_{n-2} \text{ para cada n mayor o igual que 3.} \end{cases}$$

Para construir con MATLAB los primeros 20 términos de dicha sucesión bastará escribir:

```
a(1)=1;
a(2)=2;
for k = 3:20
        a(k) = a(k-1)+a(k-2);
end
a
```

Podemos anidar múltiples bucles for. Supongamos, por ejemplo, que queremos construir una matriz 15x11 con la siguiente estructura: todos los elementos de la diagonal principal toman el valor uno, los que hay justo por encima el valor dos, los que hay justo por encima de éstos el valor tres y así sucesivamente; los que hay debajo de la diagonal principal valen cero. Entonces deberemos escribir:

```
for m = 1:15
        for n = 1:11
                if  n >= m
                        B(m,n)=n-m+1;
                else
                        B(m,n)=0;
                end
        end
end
```

El comando break permite interrumpir la ejecución de un bucle for o while. En bucles anidados tan sólo se interrumpe la ejecución del bucle más próximo.

Gráficos.

MATLAB dispone de un gran número de herramientas que permiten visualizar vectores y matrices como gráficos. Aquí tan sólo describimos, brevemente, algunos de ellos.

Para crear un gráfico bidimensional utilizamos el comando plot, el cual admite distintos formatos. Si y es un vector, plot(y) produce un gráfico lineal a trozos de los elementos de y frente a los índices de dichos elementos. Si x e y son dos vectores de la misma longitud, plot(x,y) produce un gráfico lineal a trozos de y frente a x. Por ejemplo, para dibujar la gráfica de la función seno en el intervalo [0,2Π] escribiremos

```
t = 0: pi/100:2*pi;
y = sin(t);
plot(t,y)
```

Hemos construido una partición de diámetro Π/100. Lógicamente cuanto más fina sea la partición (es decir, cuanto menor sea su diámetro) mejor será la gráfica obtenida. Múltiples parejas x-y permiten crear varias gráficas, cada una de un color distinto, con una sola llamada a plot. Probemos con

```
y2 = sin(t-0.25);
y3 = sin(t-0.5);
plot(t,y,t,y2,t,y3)
```

Es posible especificar el color, cómo se unen los puntos (estilo de línea) y cómo se marcan los puntos (marcadores) con

```
plot(x,y,'color_estilo_marcador')
```

donde color_estilo_marcador es una cadena de uno, dos o tres caracteres construida en base a las siguientes posiblidades:

- Los colores disponibles son c, m, y, r, g, b, w y k, que corresponden a turquesa, magenta, amarillo, rojo, verde, azul, blanco y negro.
- Los estilos disponibles son -, --, :, -. y none, que corresponden a continuo, con guiones, punteado, con guiones y puntos y sin línea.
- Los marcadores más habituales son +, o, * y x.

La instrucción

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

dibuja la gráfica en color amarillo, punteada y con el marcador + en cada uno de los puntos correspondientes a los datos. Si especificamos un marcador pero no un estilo de línea, MATLAB sólo dibuja los marcadores.

La función plot abre automáticamente una ventana gráfica (*figure window*) si no hay ninguna abierta en la pantalla. Si ya existe una ventana

gráfica, plot la utiliza por defecto, borrando todo lo que en ella hubiera dibujado. Para abrir una nueva ventana gráfica y que pase a ser la ventana en uso, escribimos figure. Para que una ventana gráfica ya existente pase a ser la ventana en uso escribimos figure(n), donde n es el número que aparece en la barra superior de la ventana. Los resultados de los comandos gráficos que ejecutemos a continuación se mostrarán en dicha ventana.

El comando hold permite añadir elementos diversos a un gráfico ya existente. Cuando tecleamos hold on MATLAB protege el gráfico de la ventana actual y superpone al mismo los efectos de los nuevos comandos gráficos que ejecutemos, reescalando si es necesario. Por ejemplo, escribamos

```
hold on
y4 = cos(t);
plot(t,y4,'b:*')
```

y veremos cómo la gráfica del coseno se dibuja sobre el último gráfico creado. La orden se desactiva con hold off.

La función subplot permite dividir una ventana gráfica en varias ventanas más pequeñas y dibujar en cada una de ella gráficos distintos. Al escribir

```
subplot(m,n,p)
```

la ventana gráfica actual se divide en m por n subventanas, numeradas de izquierda a derecha y de arriba abajo, y selecciona la ventana número p como ventana actual. Por ejemplo, con las instrucciones siguientes representamos en una misma ventana pero de manera independiente, las gráficas de $g_k(t) = \text{sen}(kt)$ para k=1,...,9 en $[0,2\Pi]$:

```
for k = 1:9
        subplot(3,3,k)
        plot(t,sin(k*t))
end
```

La función axis permite modificar el aspecto de la *caja* donde se va a dibujar el gráfico. Por defecto, MATLAB encuentra el máximo y el mínimo entre los datos que le damos y elige unas dimensiones y un etiquetado de los ejes adecuado. Pero podemos establecer los límites que queramos para los ejes escribiendo axis([xmin xmax ymin ymax]). Con axis square conseguimos que las dimensiones de los ejes sean las mismas. Con axis equal logramos que incrementos iguales en ambos ejes midan lo mismo. La orden axis auto nos devuelve al escalado por defecto.

Si escribimos grid on el gráfico aparece en una cuadrícula. Con grid off la cuadrícula desaparece. Las funciones xlabel, ylabel y zlabel (ésta última para gráficos tridimensionales) permiten poner nombres a los ejes coordenados. Si queremos, por ejemplo, llamar al eje x por su nombre escribiremos

```
xlabel('Eje x')
```

La orden title permite poner nombre al gráfico representado. Su formato es

```
title('titulo_del_grafico')
```

MATLAB también permite representar superficies de la forma z=f(x,y). Para ello disponemos de dos comandos, mesh y surf. El funcionamiento es análogo en ambos: sobre una malla de una determinada parte del plano levantamos una serie de puntos y los unimos. La diferencia entre uno y otro comando está en la forma en que dicha unión se lleva a cabo. Mientras que mesh sólo colorea las líneas de unión, surf también da color a las porciones planas que aquellas delimitan. Supongamos que queremos representar la superficie $z = xe^{-x^2-y^2}$ sobre el cuadrado [-2,2]x[-2,2]. Éstos son los pasos a seguir:

1. Construcción de la malla. Comenzamos creando una partición uniforme sobre el intervalo [-2,2] del eje x:

   ```
   px = -2:0.2:2;
   ```

   A continuación creamos otra sobre el intervalo [-2,2] del eje y:

   ```
   py = -2:0.1:2;
   ```

   Con estas particiones la malla tendría 21x41 puntos. Por último creamos dos matrices x e y con las siguientes características: las 21 filas de x son iguales y consisten en las abcisas de los puntos de la malla; las 41 columnas de y son iguales y consisten en las ordenadas de los puntos de la malla. Para ello escribimos [x,y] = meshgrid(px,py). Si px y py fueran iguales, bastaría escribir [x,y] = meshgrid(px).
2. Levantamiento de la función. Escribimos

   ```
   z = x.*exp(-x.^2-y.^2);
   ```

3. Construcción de la gráfica. Escribimos mesh(x,y,z) o surf(x,y,z).

Antes de finalizar este apartado queremos insistir en que aquí está recogida tan sólo una ínfima parte de las posibilidades gráficas de MATLAB. Nos dejamos atrás aspectos como la representación de curvas en el espacio, la coloración, textura, iluminación, enfoque y proyección de superficies, la colocación de cámaras alrededor de una superficie que nos permitan movernos entorno a ella, la creación de gráficos de barras, áreas y sectores, la representación de campos vectoriales, la creación de contornos de superficies (curvas de nivel), el tratamiento de imágenes digitales, la modelización 3D, etc.

Otros aspectos interesantes.

Finalizamos esta introducción a MATLAB con algunas cuestiones que puedan sernos de utilidad. Empezamos con el control del tiempo.

La orden clock muestra un vector de 6 componentes en notación decimal. Las tres primeras corresponden a la fecha actual y las tres últimas a la hora en curso. Si lo queremos en notación entera escribiremos fix(clock). Con esta orden podemos controlar el tiempo que tarda un programa en ejecutarse. Bastará hacer, por ejemplo,

```
tp1 = fix(clock(3:6));
```

antes de la ejecución,

```
tp2 = fix(clock(3:6));
```

después de ella y luego comparar tp1 con tp2.

Otro aspecto al que queremos hacer referencia es la evaluación de cadenas de caracteres, pues añade potencia y flexibilidad al lenguaje MATLAB.

La función eval evalúa una cadena que contiene una expresión de MATLAB o un enunciado. En su forma más simple, su sintaxis es eval('cadena'). Por ejemplo, las siguientes instrucciones generan una matriz de orden n:

```
t = '1/(i+j-1)';
for i = 1:n
        for j = 1:n
                a(i,j) = eval(t);
        end
end
```

La función feval ejecuta una función cuyo nombre aparece en una cadena. Su sintaxis es feval('cadena',dato_numerico) y su efecto es evaluar la función de nombre cadena en dato_numerico. El siguiente ejemplo permite elegir entre tres funciones distintas a evaluar:

```
fun = ['sin','cos','log'];
k = input('Elige un numero de funcion:');
x = input('Elige un punto para evaluar la funcion:');
feval(fun(k,:),x)
```

El argumento de la función eval puede ser una concatenación de cadenas de caracteres. El siguiente ejemplo muestra cómo crear 10 variables de nombres P1, P2,..., P10 y que cada una tenga un valor distinto:

```
for k = 1:10
        eval(['P',int2str(k),'=k^2'])
end
```

La orden int2str(k) transforma el valor numérico de k en una cadena de caracteres.