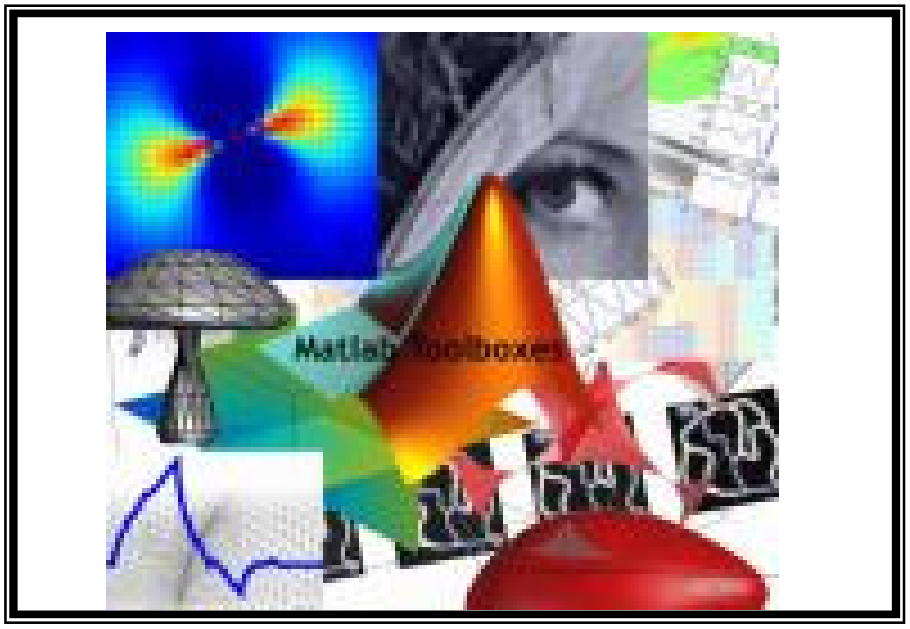
APUNTES
INTRODUCCION A MATLAB



DESARROLLADO POR: FRANCISCO CUBILLOS M.

2013

1. INTRODUCCIÓN A MATLAB.

1.1. Información sobre este documento.

Se presenta a continuación una introducción a Matlab, dirigida a usuarios que no tienen mayores conocimientos de este programa. Contiene explicaciones concisas de los comandos esenciales de Matlab, capacidades gráficas y de programación. Además incluye una introducción a Simulink, para simulación de sistemas dinámicos. El documento trata de introducir al usuario en los distintos temas explicados en una forma sencilla y que vayan acorde al grado de dificultad a tratar en el curso.

1.2. El programa MATLAB.

MATLAB ® es la abreviación de "MATrix LABoratory". MATLAB es un programa capaz de efectuar cálculos numéricos con ***vectores*** y ***matrices***, números escalares, tanto reales como complejos, con cadenas de caracteres y con otras estructuras de información más complejas. Una de los potenciales más atractivos de este programa es realizar una amplia variedad de ***gráficos*** en dos y tres dimensiones. Este manual fue escrito para ser usado con MATLAB 7, pero también puede usarse con versiones anteriores.

MATLAB® es un gran programa de cálculo técnico y científico, utilizado por profesionales de diversas áreas. Integra computación, visualización y programación en un ambiente fácil de usar, con soluciones expresadas en una notación matemática familiar. Se utiliza principalmente en: *Matemáticas y computación; Desarrollo de algoritmos; Modelación y simulación; Análisis de datos, exploración y visualización; Gráficos científicos y de ingeniería; Desarrollo de aplicaciones, entre otras.*

La manera más fácil de visualizar Matlab es pensar en él como en una calculadora programable totalmente equipada, aunque en realidad, ofrece muchas más características y es mucho más versátil que cualquier calculadora.

1.3. Entorno MATLAB ®

Para empezar a utilizar Matlab basta realizar un doble click en el icono correspondiente del escritorio. También puede usarse el menú de *Inicio*. Se abre una ventana como la exhibida en la Figura 1.1.

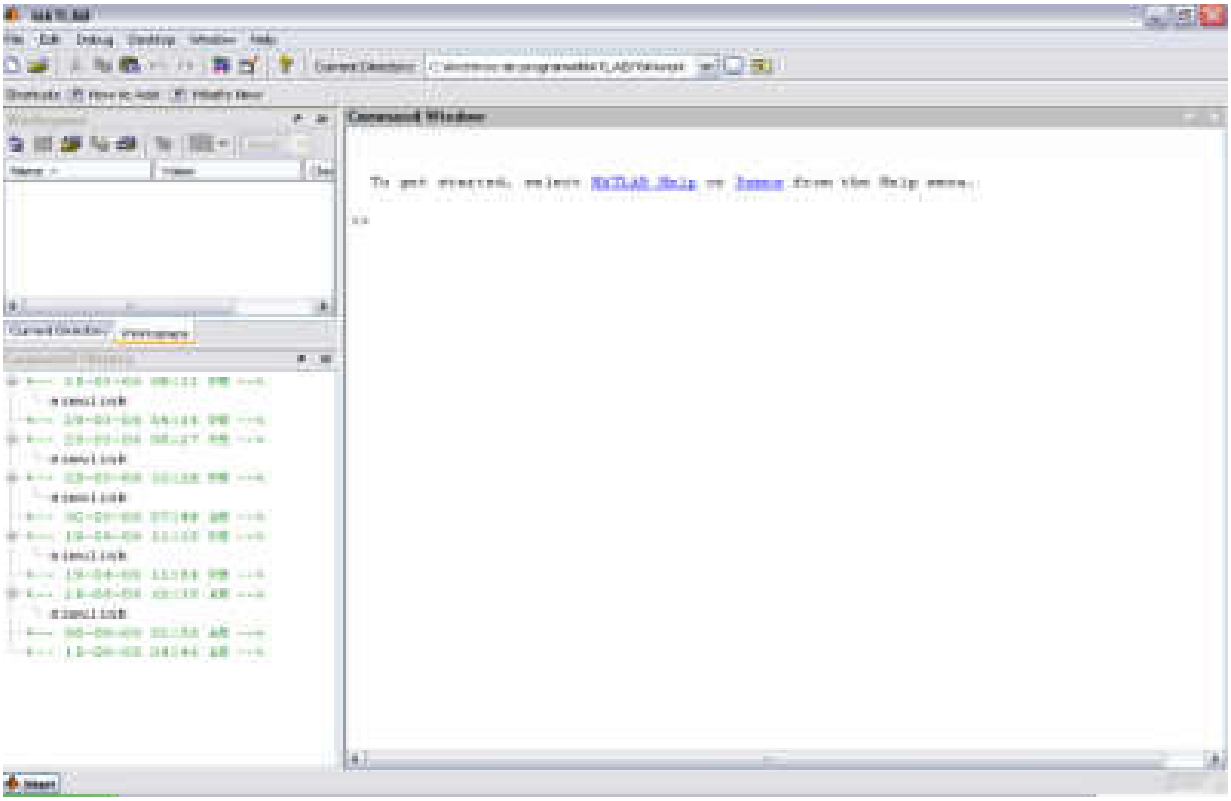


Figura 1.1: *Ventana de inicio Matlab 7.0*

El entorno operativo de Matlab consta de una serie de ventanas que se presentan a continuación. Aunque el reparto de estas ventanas en la pantalla puede cambiarse, generalmente aparecen en la misma posición. En cualquier caso, una vista similar se puede conseguir con ***View/Desktop Layout/Default***.

Las ventanas que forman parte del entorno de trabajo de Matlab son:

1.3.1. Command Window

Se utiliza para ingresar órdenes directamente por el usuario. Los resultados se muestran en esta misma pantalla. Cuando las órdenes se envían desde un programa previamente escrito, que en Matlab recibe el nombre de M-file, los resultados también aparecen en esta ventana.

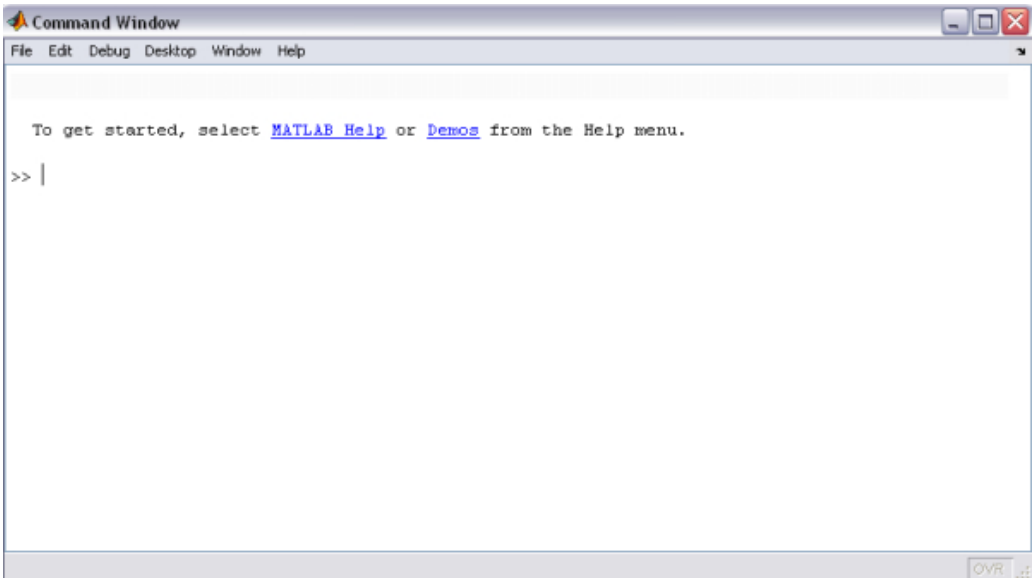


Figura 1.2: *Ventana de comandos de Matlab.*

1.3.2. Command history.

Las órdenes introducidas en la ventana *command window* quedan grabadas en esta ventana, de forma que haciendo doble click sobre ellas las podemos volver a ejecutar. Otra forma es usando la tecla ↑ en *command window,* las últimas órdenes ingresadas por el usuario, se pueden rescatar.

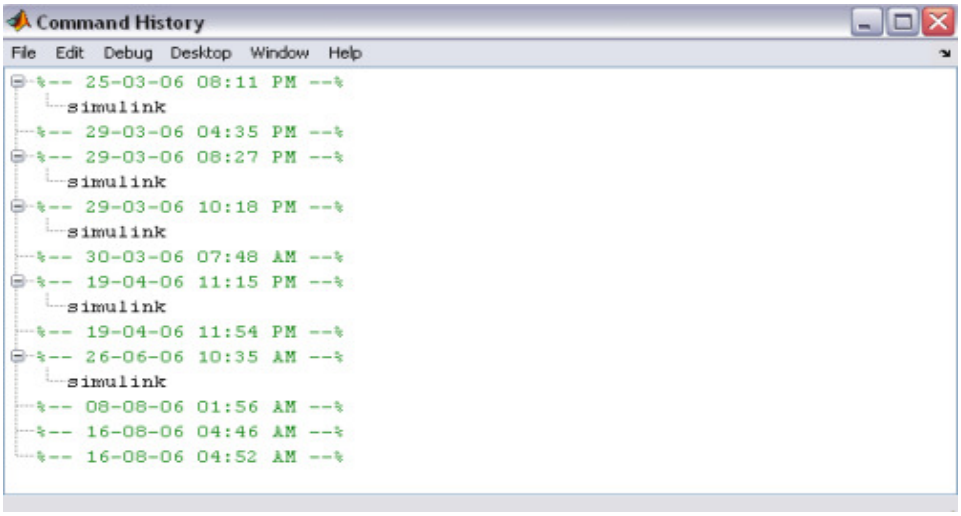


Figura 1.3: *Historial de comandos de Matlab*

1.3.3. *Workspace.*

Esta ventana contiene las variables (escalares, vectores, matrices, etc) creadas en la sesión de Matlab. La ventana workspace nos proporciona información sobre el nombre, dimensiones, tamaño y tipo de variable. Existen dos opciones para eliminar una variable:

(a) Introducir en *command window* el comando *clear* seguido del nombre de la variable.

(b) Seleccionar la variable en el *workspace* y borrarla directamente con la tecla *delete*.

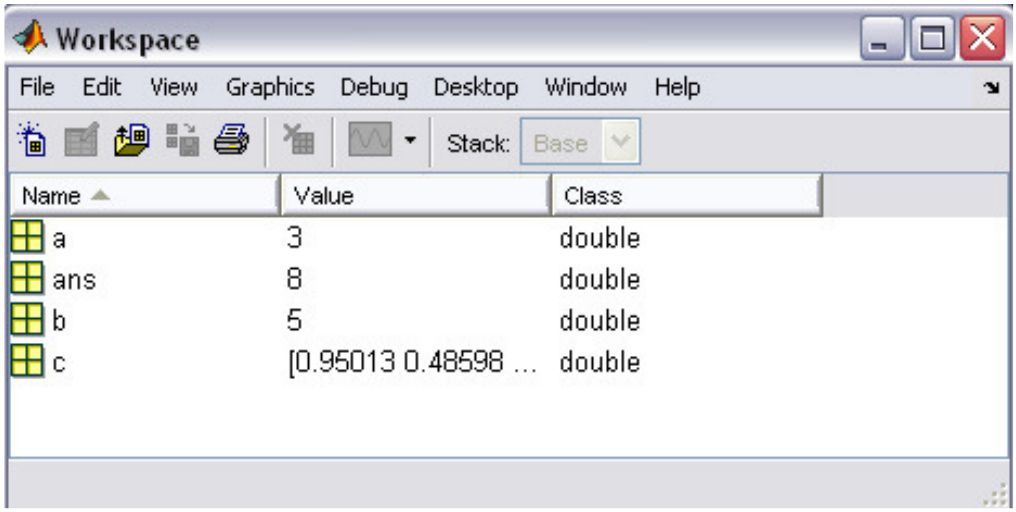


Figura1. 4: *Ventana Workspace*

Haciendo doble click en una variable se accede al contenido de dicha variable, pudiendo modificar sus valores.

1.3.3. Current directory.

Las operaciones de Matlab utilizan el directorio seleccionado en *current directory* (a través del botón para explorar) como punto de referencia. Por ejemplo, si se guarda una serie de variables con el comando *save*, se guardan en el directorio en el que estemos trabajando. Lo mismo sucede cuando se cargan datos con el comando *load*: el computador busca los datos en el archivo en el que estemos trabajando.

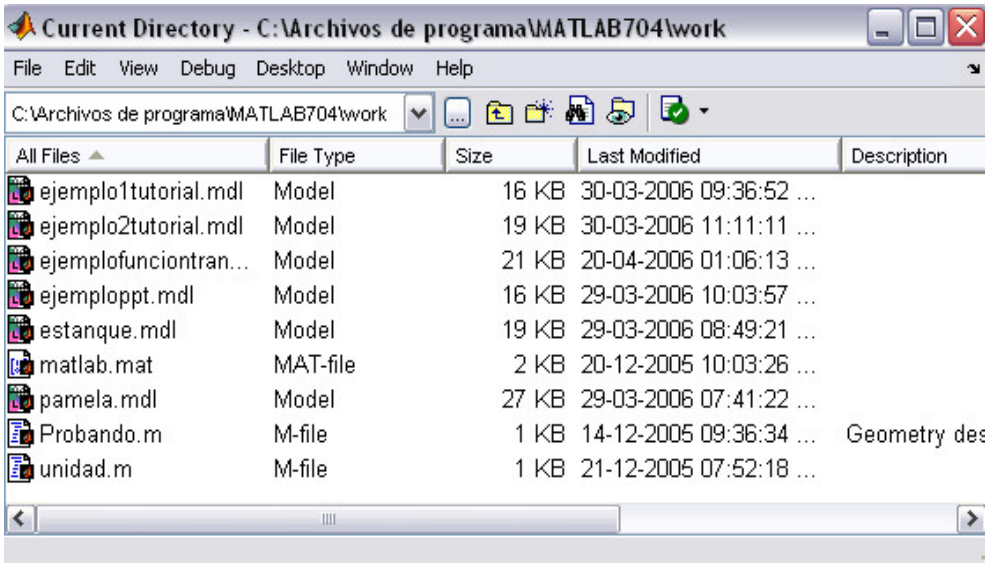


Figura 1.5: *Ventana Current Directory*

1.4. Path de Matlab: establecer el camino de búsqueda (*search path*)

Matlab puede llamar a una gran variedad de funciones, tanto propias como programadas por los usuarios. Puede incluso haber ***funciones distintas con el mismo nombre***. Interesa saber cuáles son las reglas que determinan qué función o qué archivo ****.m*** es el que se va a ejecutar cuando su nombre aparezca en una línea de comandos del programa. Esto queda determinado por el *camino de búsqueda* (***search path***) que el programa utiliza cuando encuentra el nombre de una función.

El ***search path*** de Matlab es una lista de directorios que se puede ver y modificar a partir de la línea de comandos, o utilizando el cuadro de diálogo ***Set Path***, del menú ***File***. El comando ***path*** hace que se escriba el ***search path*** de MATLAB (el resultado depende de en qué directorio esté instalado MATLAB; se muestran sólo unas pocas líneas de la respuesta del programa):

```
>> path
>> path
MATLABPATH
C:\MATLAB701\toolbox\matlab\general
C:\MATLAB701\toolbox\matlab\ops
C:\MATLAB701\toolbox\matlab\lang
C:\MATLAB701\toolbox\matlab\elmat
...
C:\MATLAB701\toolbox\matlab\helptools
C:\MATLAB701\toolbox\matlab\winfun
C:\MATLAB701\toolbox\matlab\
```

El cuadro de diálogo que se abre con el comando ***File/Set Path*** ayuda a definir la lista de directorios donde Matlab debe buscar los archivos de comandos y las funciones, tanto del sistema como del usuario. Al ejecutar dicho comando aparece el cuadro de diálogo de la ***Figura 1.6***, en el cual se muestra la lista de directorios en la que MATLAB buscará.

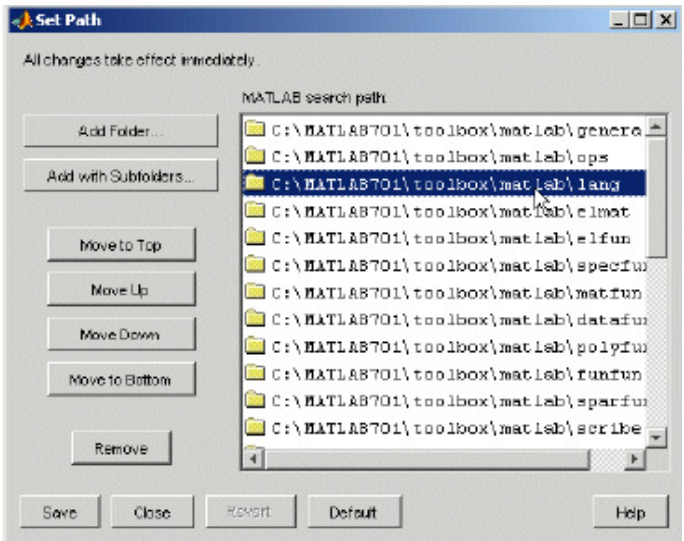


Figura 1. 6: *Search Path desde el menú File*

Para añadir (o quitar) un directorio a esta lista se debe realizar un clic sobre los botones ***Add Folder*** o ***Add with Subfolders***, con lo cual aparece un nuevo cuadro de diálogo, mostrado en la ***Figura 1.7***, que ayuda a elegir el directorio deseado. El cuadro de diálogo ***Set Path*** contiene los botones necesarios para realizar todas las operaciones que el usuario desee.

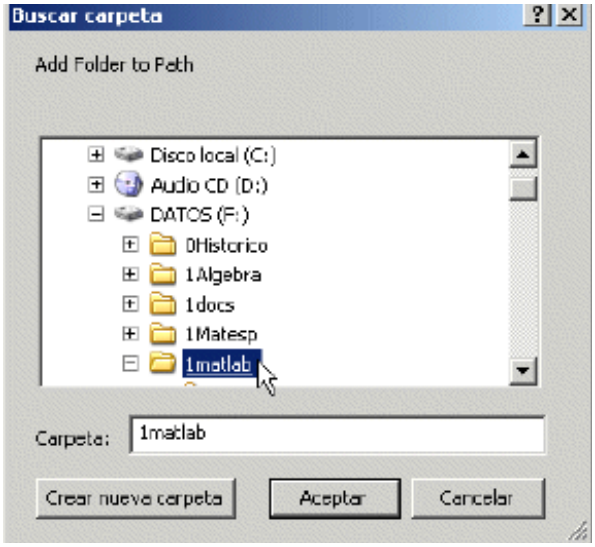


Figura 1. 7: *Ventana para elegir directorio deseado*

1.5. El Editor/Debugger

En Matlab son de vital importancia los ***archivos-M*** (o ***M-files***). Son archivos de texto ASCII, con la extensión ****.m***, que contienen ***un grupo de comandos*** o ***definición de funciones***. La importancia de estos ***archivos-M*** es que al escribir su nombre en la línea de comandos y pulsar ***Intro***, se ejecutan uno tras otro todos los comandos contenidos en dicho archivo. El poder guardar instrucciones y grandes matrices en un archivo permite ahorrar mucho trabajo de tecleado.

Los archivos ****.m*** se pueden crear con cualquier editor de archivos ASCII como por ejemplo el ***Notepad***, no obstante, Matlab dispone de un ***editor*** que permite crear y además modificar estos archivos, así como también ejecutarlos paso por paso para ver si contienen errores (proceso de ***Debug*** o depuración).

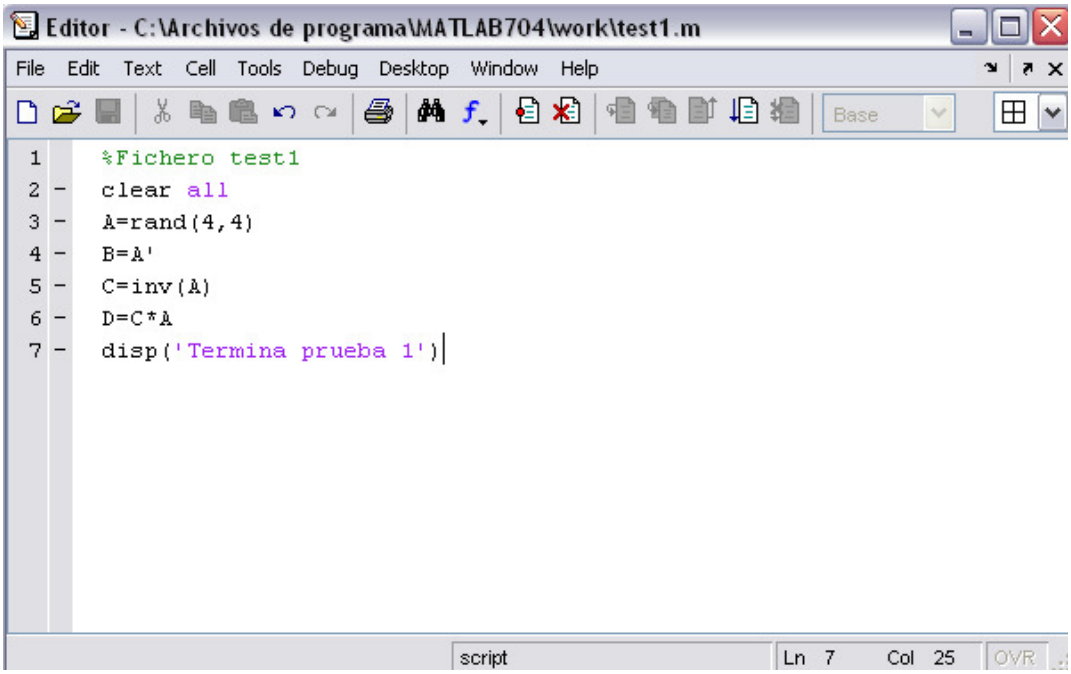


Figura 1.8: *Ventana del editor de archivos-M*

En la figura 1.8 se muestra la ventana principal del ***Editor/Debugger***, en la que se ha creado un ***archivo-M*** llamado ***Test1.m***, que contiene un comentario y seis sentencias. El ***Editor*** muestra con diferentes colores los diferentes tipos o elementos de los comandos (en *verde* los comentarios, en *violeta* las cadenas de caracteres, etc.).

El ***Editor*** se encarga también de que las comillas o paréntesis que se abren, no se queden sin el correspondiente elemento de cierre.

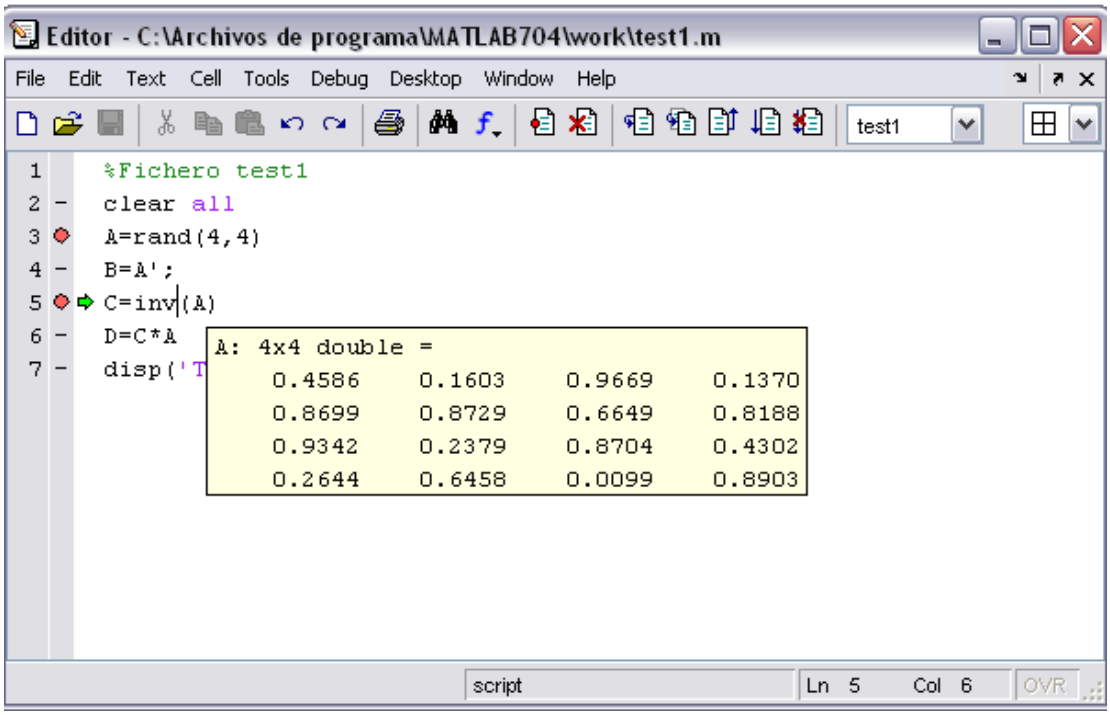


Figura 1. 9: *Ejecución del archivo test1.m*

La Figura 1.9 corresponde a una ejecución de este archivo ***test1.m*** controlada con el ***Debugger***. Esta ejecución se realiza con el comando ***Run*** en el menú ***Debug***, o pulsando la tecla **F5**, luego haciendo click en el botón ***Continue*** de la barra de herramientas del ***Editor*** o introduciendo el nombre del archivo en la línea de comandos de ***Command Window***. Los puntos rojos que se muestran al margen izquierdo son ***breakpoints*** (puntos en los que se detiene la ejecución de programa); la ***flecha verde*** en el borde izquierdo indica la sentencia en que está detenida la ejecución (antes de ejecutar dicha sentencia); cuando el cursor se coloca sobre una variable (en este caso sobre **A**) aparece una pequeña ***ventana con los valores numéricos*** de esa variable, tal como se ve en la Figura 1.9.

El ***Debugger*** es un programa complejo y que debe conocerse a cabalidad, puesto que es muy útil para detectar y corregir errores. Tiene una vital importancia para aprender métodos numéricos y técnicas de programación.

1.6. Utilización de la ayuda

Una de las principales ventajas de Matlab con respecto a similares programas consiste en la gran cantidad de información que el usuario puede obtener del funcionamiento del programa y de los comandos a través de la ayuda (Menú Help). Para programar en Matlab es necesario manejarse en el entorno de trabajo y saber utilizar la ayuda. Ante el surgimiento de cualquier duda sobre como utilizar una función o cual es el comando para realizar una determinada operación, la ayuda que incluye Matlab es extremadamente útil. El menú Help contiene, además de dos aplicaciones para introducir al usuario el entorno de trabajo (Help → Using the Desktop and Help → Using the Command Window), una pestaña llamada Matlab Help.

Matlab Help permite al usuario buscar información de tres formas diferentes:

1. La pestaña ***Contents*** permite ver un índice con todas las aplicaciones y Toolboxes de Matlab.

2. La pestaña ***Index*** permite buscar, por orden alfabético, en el índice de materias de Matlab, información sobre palabras clave (comandos, ordenes, etc). Por ejemplo, si escribimos ***plot*** obtenemos información sobre dicho comando, que se utiliza para hacer gráficos.

3. La pestaña **Search** busca en toda la documentación de ayuda de Matlab las palabras introducidas.

Además Matlab dispone de Toolboxes, que comercializa de forma separada al programa, incluye una amplia gama de materias, tales como: Optimización, Finanzas, Ecuaciones diferenciales parciales, Estadísticas, por nombrar las más importantes. Cada toolbox contiene funciones predefinidas de la materia que trata, documentación (en PDF) de cada toolbox.

Otro aspecto destacable es la inclusión de Demos, los cuales resultan bastante útiles cuando se realizan trabajos de mayor complejidad.

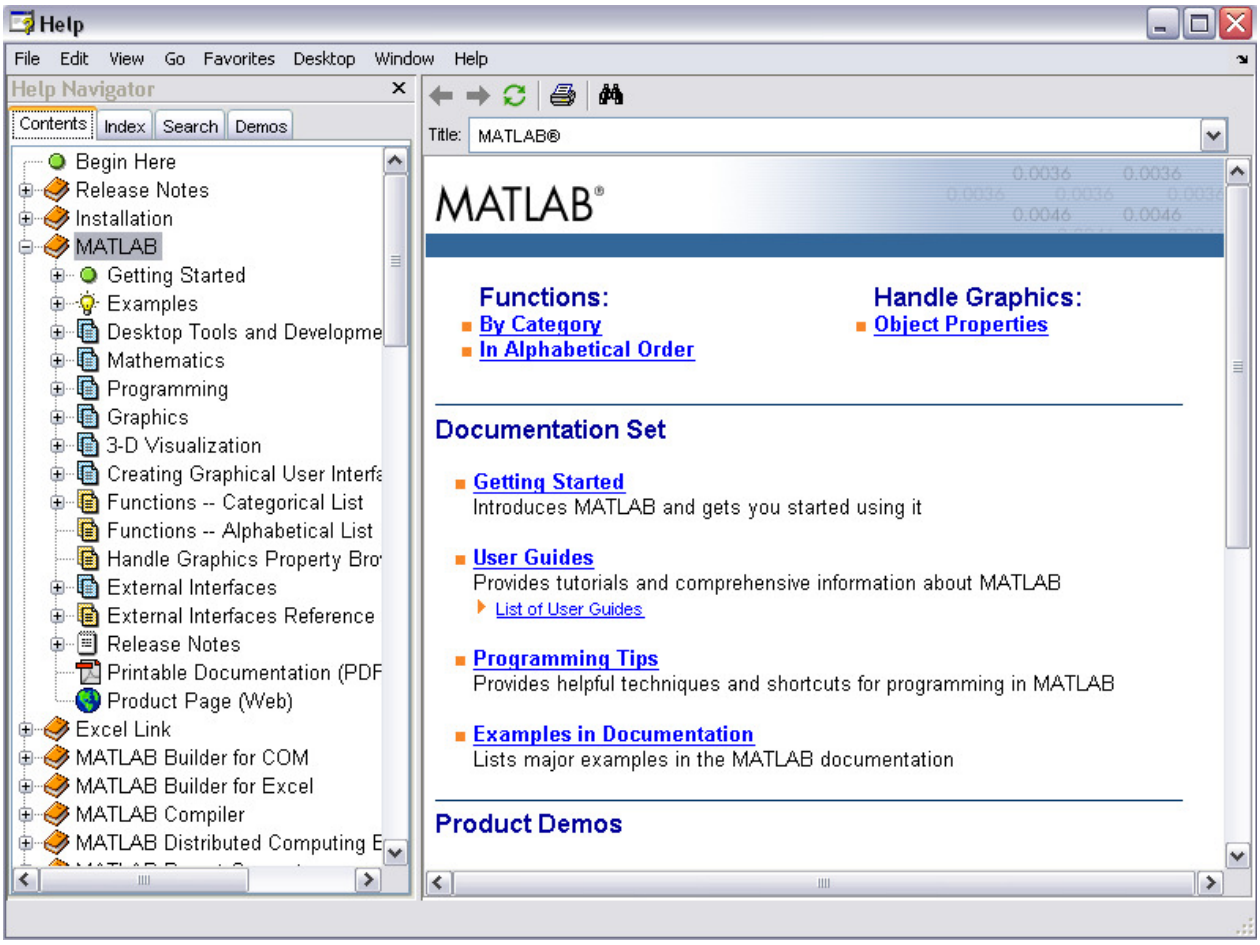


Figura 1.10 : *Vista de la ventana de Ayuda de Matlab*

1.7. Preferencias.

MATLAB 7.0 está provisto de un cuadro de diálogo desde el cual se establecen prácticamente todas las opciones que pueden ser modificadas por el usuario. Este cuadro de diálogo se activa a través de ***Preferences*** del menú ***File***.

En la Figura 1.11 se ilustra la ventana ***Preferences*** con todas las posibilidades que ofrece en el menú de la izquierda: en total son 24 cuadros de diálogo diferentes. La Figura 1.12 muestra la sección que permite elegir los colores generales del código.

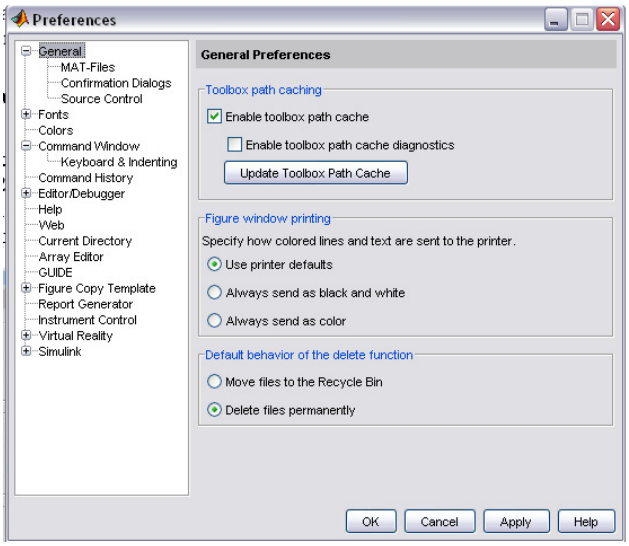


Figura 1.11: *Ventana Preferences*

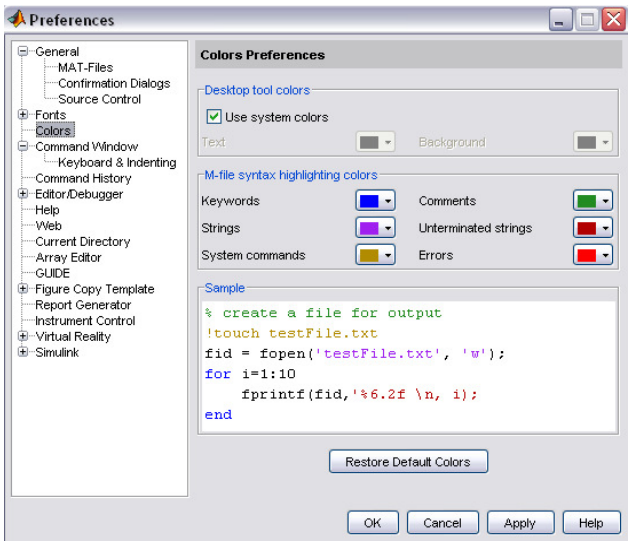


Figura 1.12 : *Otra opción de la ventana Preferences*

El cuadro de diálogo ***Command Window/Fonts*** nos entrega la posibilidad de elegir el tipo de letra –así como el tamaño y el color, tanto de las letras como del fondo– con la que se escribe en la ventana de comandos de Matlab. Respecto a los formatos numéricos con que Matlab muestra los resultados, las posibilidades existentes se muestran en la siguiente lista:

Comando	***Descripción***
short	*Coma fija con 4 decimales (**por defecto**)*
long	*Coma fija con 15 decimales*
hex	*Cifras hexadecimales*
bank	*Números con dos cifras decimales*
short e	*Notación científica con 4 decimales*
short g	*Notación científica o decimal, dependiendo del valor*
long e	*Notación científica con 15 decimales*
long g	*Notación científica o decimal, dependiendo del valor*
rational	*Expresa los números racionales como cocientes de enteros*

1.8. Líneas de comentarios

El carácter ***tanto por ciento*** (%) indica el comienzo de un comentario. Cada vez que aparece en una línea de comandos, el programa supone que todo lo que va desde ese carácter hasta el fin de la línea es un comentario. Pueden servir para definir *help*'s personalizados de las funciones que el usuario vaya creando.

1.9. Comandos *save* y *load*

En diversas situaciones nos vemos obligados a interrumpir el trabajo con MATLAB y por lo mismo, debemos ser capaces de recuperar más tarde en el mismo punto en el que se dejó (con las mismas variables definidas, con los mismos resultados intermedios, etc.). Se debe tener en cuenta que al salir del programa todo el contenido de la memoria se borra en forma automática. Para guardar el estado de una sesión de trabajo existe el comando ***save***. Si se teclea:

>> save antes de abandonar el programa, se crea en el ***directorio actual de trabajo*** un archivo binario llamado ***matlab. mat*** (o ***matlab***) con el estado de la sesión (excepto los gráficos, que por ocupar mucha memoria deben de guardarse por separado). Dicho estado puede recuperarse la siguiente vez que se arranque el programa con el comando:

>> load

Esta es la forma más simple de los comandos ***save*** y ***load***. Se pueden guardar también matrices y vectores de forma selectiva y en archivos con nombre definidos por el usuario. Por ejemplo, el comando (sin comas entre los nombres de variables):

>> save filename A x y guarda las variables **A**, **x** e **y** en un archivo binario llamado ***filename.mat*** (o ***filename***). Para recuperarlas en otra sesión basta teclear:

>> load filename Si no se indica ninguna variable, se guardan todas las variables creadas en esa sesión.

2. ESTRUCTURA DE DATOS.

2.1. Definición de variables

Para crear una variable en Matlab, simplemente se introduce, en la ventana ***command window***, el nombre de la variable y su valor. Por ejemplo:

```
a = 7;
```

Esta indicación hace que, en el espacio de trabajo ***workspace***, se cree una variable escalar con valor 7. Cualquier orden posterior puede hacer uso de esta variable por medio de una llamada a "*a*", por ejemplo:

```
b = exp(a);
```

Se crea así una nueva variable "b" cuyo valor es el exponencial de la variable "a". El punto y coma ***";"*** después de una orden no es necesario. Si no ponemos el punto y coma, el resultado de la orden aparece en la pantalla ***command window*** tal como se indica a continuación:

```
>> b=exp(a)
b =
1.0966e+003
```

2.1.1. El comando clear.

Para borrar variables que hemos creado en la sesión de trabajo, es conveniente conocer el comando ***clear***, el que posee distintas formas:

- ***clear:*** sin argumentos, ***clear*** elimina todas las variables creadas previamente (excepto las variables globales).
- ***clear A, b:*** borra las variables indicadas.
- ***clear global***: borra las variables globales.
- ***clear functions***: borra las funciones.
- ***clear all***: borra todas las variables, incluyendo las globales, y las funciones.

2.2. Operaciones con matrices y vectores.

En la introducción de este manual se ha dicho que Matlab es fundamentalmente un programa para cálculo matricial. Inicialmente se utilizará MATLAB como ***programa interactivo***, en el que se irán definiendo las matrices, los vectores y las expresiones que los combinan y obteniendo los resultados inmediatamente. Si estos resultados son asignados a otras variables podrán ser utilizados posteriormente en otras expresiones.

Antes de realizar cálculos complejos, se mostrará como ingresar matrices y vectores desde el teclado. Más adelante se verán otras formas más potentes de definir matrices y vectores.

2.3. Definición de matrices y vectores desde teclado.

De la misma forma que se definen variables escalares pueden definirse vectores (arrays de 1 dimensión) y matrices (arrays de 2 dimensiones). Por ejemplo, para definir la matriz:

$$M = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Se debe ingresar en la ventana de comandos:

```
>> M= [1, 4; 2, 5; 3, 6]
M =

    1    4
    2    5
    3    6
```

Donde los términos de una misma fila aparecen separados por comas, y el punto y coma hace de separador entre filas.

A partir de este momento la matriz **M** está disponible para hacer cualquier tipo de operación con ella (además de valores numéricos, en la definición de una matriz o vector se pueden utilizar expresiones y funciones matemáticas). Por ejemplo, una sencilla operación con **A** es encontrar su ***matriz transpuesta***, esto es, intercambiar filas por columnas.

En Matlab el apóstrofo (') es el símbolo de ***transposición matricial***. Para trasponer M, basta teclear M' en la ventana de comandos, a continuación se muestra la respuesta del programa:

```
>> M'
ans =

    1    2    3
    4    5    6
```

Como el resultado de la operación no ha sido asignado a ninguna otra matriz, Matlab utiliza un nombre de variable por defecto (***ans***, de *answer*), que contiene el resultado de la última operación. También podría haberse asignado el resultado a otra matriz llamada **N**:

```
>> N=M'

N =

    1    2    3
    4    5    6
```

En estos momentos tenemos definidas las matrices **M** y **N**, por lo que es posible seguir operando con ellas. Por ejemplo, se puede hacer el producto **N*M** (deberá resultar una matriz simétrica):

```
>> N*M
ans =
   14   32
   32   77
```

2.3.1 Funciones que definen matrices estándar

• Matriz de unos de dimensión N1xN2 : **ones(N1,N2)**

Ejemplo:

```
>> ones(2,3)
ans =
    1    1    1
    1    1    1
```

• Matriz inversa: **inv(Matriz Cuadrada)**

Ejemplo:

```
P =
    1    1    3
    2    3    1
    2    4    5

>> inv(P)
ans =

    1.2222    0.7778   -0.8889
   -0.8889   -0.1111    0.5556
    0.2222   -0.2222    0.1111
```

• Matriz de ceros de dimensión N1xN2 : **zeros(N1,N2)**

Ejemplo:

```
>> zeros(2,3)
ans =

    0    0    0
    0    0    0
```

• Matriz identidad de dimensión N1xN1 : **eye(N1)**

Ejemplo:

```
>> eye(3)
ans =
    1    0    0
    0    1    0
    0    0    1
```

• linspace(primer elemento, último elemento, número de elementos igualmente espaciados)

Ejemplo:

```
>>linspace (0, 1, 5) % crea el vector fila
ans =

         0    0.2500    0.5000    0.7500    1.0000
```

• La orden M = [primer elemento : paso : último elemento] : crea un vector fila que va desde "primer elemento" hasta "último elemento" dando saltos de la magnitud indicada por "paso". Por ejemplo:

```
>>M = [1 : 0.5 : 2]  %genera el vector fila

M =
    1.0000    1.5000    2.0000
```

Se deja como ejercicio crear las siguientes matrices.

Función	***Descripción***
logspace(d1,d2,n)	*Genera un vector con **n** valores espaciados logarítmicamente entre 10^d1 y 10^d2. Si **d2** es **pi9**, los puntos se generan entre 10^d1 y **pi***
rand(3)	*Forma una matriz de números aleatorios entre 0 y 1, con distribución uniforme, de tamaño (3×3)*
rand(2,5)	*Idem de tamaño (2×5)*
randn(4)	*Forma una matriz de números aleatorios de tamaño (4×4), con distribución normal, de valor medio 0 y varianza 1*
magic(4)	*Crea una matriz (4×4) con los números 1, 2, ... 4*4, con la propiedad de que todas las filas y columnas suman lo mismo*

Existen otras funciones para crear matrices de tipos particulares. Con ***Help/Matlab Help*** se puede obtener información sobre todas las funciones disponibles en Matlab, que aparecen agrupadas por categorías o por orden alfabético. En la categoría ***Mathematics*** aparecen la mayor parte de las funciones estudiadas.

2.3.2. Formación de una matriz a partir de otras

Matlab ofrece también la posibilidad de crear una matriz a partir de matrices previas ya definidas, por varios posibles caminos:

- Recibiendo alguna de sus propiedades (como por ejemplo el tamaño),
- Por composición de varias submatrices más pequeñas,
- Modificándola de alguna forma.

A continuación se describen algunas de las funciones que crean una nueva matriz a partir de otra o de otras, comenzando por dos funciones auxiliares:

Función	***Descripción***
[m,n]=size(A)	*Devuelve el número de filas y de columnas de la matriz A. Si la matriz es cuadrada basta recoger el primer valor de retorno*
n=length(x)	*Calcula el número de elementos de un vector x*
zeros(size(A))	*Forma una matriz de ceros del mismo tamaño que una matriz A previamente creada*
ones(size(A))	*Función idéntica a la anterior, pero con unos*
A=diag(x)	*Forma una matriz diagonal A cuyos elementos diagonales son los elementos de un vector ya existente x*
x=diag(A)	*Forma un vector x a partir de los elementos de la diagonal de una matriz ya existente A*
diag(diag(A))	*Crea una matriz diagonal a partir de la diagonal de la matriz A*
blkdiag(A,B)	*Crea una matriz diagonal de submatrices a partir de las matrices que se le pasan como argumentos*
triu(A)	*Forma una matriz triangular superior a partir de una matriz A (no tiene por qué ser cuadrada). Con un segundo argumento puede controlarse que se mantengan o eliminen más diagonales por encima o debajo de la diagonal principal.*
tril(A)	*Función identica a la anterior, con una matriz triangular inferior*
rot90(A,k)	*Gira k*90 grados la matriz rectangular A en sentido antihorario. k es un entero que puede ser negativo. Si se omite, el programa asume k = 1*
flipud(A)	*Encuentra la matriz simétrica de A respecto de un eje horizontal*
fliplr(A)	*Localiza la matriz simétrica de A respecto de un eje vertical*
reshape(A,m,n)	*Cambia el tamaño de la matriz A devolviendo una matriz de tamaño m×n cuyas columnas se obtienen a partir de un vector formado por las columnas de A puestas una a continuación de otra. Si la matriz A tiene menos de m×n elementos se produce un error.*

Un caso interesante es el de formar una nueva matriz ***componiendo como submatrices*** otras matrices definidas anteriormente. Como ejemplo, ingresar las siguientes líneas de comandos y ver los resultados obtenidos:

```
>> A=rand(3)
>> B=diag(diag(A))
>> C=[A, eye(3); zeros(3), B]
```

En el ejemplo anterior, la matriz **C** de tamaño 6×6 se forma por composición de cuatro matrices de tamaño 3×3. De la misma forma que con escalares, las submatrices que forman una fila se separan con ***blancos*** o ***comas***, mientras que las diferentes filas se separan entre sí con ***intros*** o ***puntos y comas***. Los tamaños de las submatrices deben de ser coherentes

2.3.3. Acceso a elementos de una matriz

La matriz M creada anteriormente, aparecerá en el *workspace* y puede ser utilizada hasta que se borre. Se trabajará identificando sus elementos.

• Para acceder al elemento (i, j) de la matriz M se ingresa: ***M (i, j)***

Si no ponemos punto y coma al final de la orden el valor obtenido aparecerá en pantalla. También podemos asignar dicho valor a un escalar:

a = M (i, j)

Ejemplo:

```
>> a=M(1,2)
a =
    4
```

• Para acceder a la fila i de la matriz M se teclea: ***M (i, :)***

Por ejemplo:

```
>> M(2,:)

ans =
    2    5
```

• Para acceder a la columna j de la matriz M ingresamos: ***M (:, j)***

Ejemplo:

```
>> M(:,1)
ans =
    1
    2
    3
```

2.3.4. Operaciones con matrices

Matlab puede operar con matrices por medio de ***operadores*** y por medio de ***funciones***. Se han visto ya los operadores *producto* (*) y *traspuesta* ('), así como la función *invertir* ***inv( )***.

Los operadores matriciales de Matlab son los siguientes:

Operador	***Significado/Descripción***
+	*Adición o suma*
-	*Sustracción o resta*
*****	*Multiplicación*
‘	*Transpuesta*
^	*Potenciación*
****	*División-izquierda*
/	*División-derecha*
.*	*Producto elemento a elemento*
./ y .	*División elemento a elemento*
.^	*Elevar a una potencia elemento a elemento*

Estos operadores se aplican también a las variables o valores escalares, aunque con algunas diferencias. Todos estos operadores son coherentes con las correspondientes operaciones matriciales: no se puede por ejemplo sumar matrices que no sean del mismo tamaño. Si los operadores no se usan de modo correcto Matlab envía un mensaje de error.

Los operadores anteriores se pueden aplicar también de modo ***mixto***, es decir con un operando escalar y otro matricial. En este caso la operación con el escalar se aplica a cada uno de los elementos de la matriz. Ejemplo:

```
>> A=[2 3; 1 5]
A =
2 3
1 5
>> A*2
ans =
4 6
2 10
>> A-4
ans =
-2 -1
-3 1
```

2.4. Tipos de datos

Matlab es un programa preparado para trabajar con vectores y matrices. Como caso particular también trabaja con variables escalares (matrices de dimensión 1). Estos aspectos se detallan a continuación con mayor precisión.

2.4.1. Números reales de doble precisión

Los elementos que conforman vectores y matrices son números reales almacenados en 8 bytes (entre 15 y 16 cifras decimales equivalentes). Es importante saber cómo trabaja Matlab con estos números y los casos especiales que presentan. Matlab mantiene una forma especial para los *números muy grandes* (más grandes que los que es capaz de representar), que son considerados como ***infinito***. Para ejemplificar esto, ingresar el siguiente comando y observar la respuesta del programa:

```
>> 1.0/0.0
Warning: Divide by zero
ans =
Inf
```

Para Matlab el ***infinito*** se representa como ***inf*** ó ***Inf***. Matlab posee además una representación especial para los resultados que no están definidos como números. Por ejemplo, si se ingresa el siguiente comando se obtiene:

```
>> 0/0
Warning: Divide by zero
ans =
NaN
>> inf/inf
ans =
NaN
```

En ambos casos la respuesta es ***NaN***, abreviación de ***Not a Number***. Este tipo de respuesta, así como la de ***Inf***, tienen enorme importancia en Matlab, debido a que permiten controlar la fiabilidad de los resultados de los cálculos matriciales.

Matlab dispone de tres funciones útiles relacionadas con las operaciones de coma flotante. Estas funciones, que no tienen argumentos, son las siguientes:

- ***eps:*** devuelve la diferencia entre 1.0 y el número de coma flotante inmediatamente superior. Da una idea de la precisión o número de cifras almacenadas. En un PC, ***eps*** vale 2.2204e-016.
- ***realmin:*** devuelve el número más pequeño con que se puede trabajar (2.2251e-308)
- ***relamax:*** devuelve el número más grande con que se puede trabajar (1.7977e+308)

2.4.2. Números complejos: función *complex*

En muchos cálculos matriciales los datos y/o los resultados no son reales sino ***complejos***, con ***parte real*** y ***parte imaginaria***. Matlab trabaja sin ninguna dificultad con números complejos. Para ver como se representan por defecto los números complejos, ingresar los siguientes comandos:

```
>> a=sqrt(-4)
a =
0 + 2.0000i
>> 3 + 4j
ans =
3.0000 + 4.0000i
```

En la entrada de datos de Matlab se pueden utilizar indistintamente la **i** y la **j** para representar el *número imaginario unidad* (en la salida, sin embargo, puede verse que siempre aparece la **i**). En general, cuando se está trabajando con números complejos, conviene no utilizar la **i** como variable ordinaria, pues puede dar lugar a errores y confusiones.

El programa Matlab cuenta con la función ***complex***, que crea un número complejo a partir de dos argumentos que representan la parte real e imaginaria, como se ve a continuación:

```
>> complex(1,2)
ans =
1.0000 + 2.0000i
```

2.4.3. Cadenas de caracteres

Matlab puede definir variables que contengan cadenas de caracteres. En Matlab las cadenas de texto van entre apóstrofos o comillas simples. Por ejemplo:

```
s = 'cadena de caracteres'
```

Las cadenas de texto tienen utilidad en temas que se verán más adelante.

2.5. El operador dos puntos (:)

Este operador es muy importante en Matlab y puede usarse de varias formas. Para empezar, defínase un vector **x** con el siguiente comando:

```
>> x=1:10
x =
1 2 3 4 5 6 7 8 9 10
```

En cierta forma se podría decir que el operador (:) representa un ***rango***: en este caso, los números enteros entre el 1 y el 10. Por defecto el incremento es 1, pero este operador puede también utilizarse con otros valores enteros y reales, positivos o negativos. En este caso el incremento va entre el valor inferior y el superior, en las formas que se muestran a continuación:

```
>> x=1:2:10
x =
1 3 5 7 9
>> x=1:1.5:10
x =
1.0000 2.5000 4.0000 5.5000 7.0000 8.5000 10.0000
>> x=10:-1:1
x =
10 9 8 7 6 5 4 3 2 1
```

El operador dos puntos (:) es aún más útil y potente –y también más complicado– con matrices. A continuación se va a definir una matriz **A** de tamaño 6×6 y después se realizarán diversas operaciones sobre ella con el operador (:).

```
>> A=magic(6)
A =
35     1     6    26    19    24
 3    32     7    21    23    25
31     9     2    22    27    20
 8    28    33    17    10    15
30     5    34    12    14    16
 4    36    29    13    18    11
```

Se debe recordar que Matlab accede a los elementos de una matriz por medio de los índices de fila y de columna encerrados entre paréntesis y separados por una coma. Por ejemplo:

```
>> A(2,3)
ans =
7
```

El siguiente comando extrae los 4 primeros elementos de la 6ª fila:

```
>> A(6, 1:4)
ans =
4 36 29 13
```

Los dos puntos aislados representan "todos los elementos". Por ejemplo, el siguiente comando extrae todos los elementos de la 3ª fila:

```
>> A(3, :)
ans =
31 9 2 22 27 20
```

Para acceder a la última fila o columna puede utilizarse la palabra ***end***, en lugar del número correspondiente. Por ejemplo, para extraer la sexta fila (la última) de la matriz:

```
>> A(end, :)
ans =
4 36 29 13 18 11
```

Se deja como sugerencia realizar las mismas operaciones, pero extrayendo columnas desde la matriz, en vez de filas.

2.6. Exportación/Importación de datos.

2.6.1. La function *xlswrite*

La function ***xlswrite*** escribe o traslada data almacenada en matrices desde Matlab a una hoja de Microsoft Excel. Si el archivo no existe, se crea el archivo en la carpeta de trabajo actual. A continuación se presentan y describen diversas variantes de la función *xlswrite:*

- ***xlswrite('nombre_del_archivo', M):*** escribe una matriz M a un archivo Excel. La matriz de entrada M posee una dimensión de *m x n* números, caracteres, arreglos, etc, donde m < 65536 y n < .Los datos de la matriz M son escritos en la primera fila de la hoja de Excel, partiendo en la celda A1.
- ***xlswrite('nombre_del_archivo', M, hoja):*** escribe la matriz M a la hoja especificada de Excel en el archivo de trabajo. El argumento “hoja” puede ser un entero positivo, o el nombre de la hoja de trabajo.

- ***xlswrite('nombre_del_archivo', M, 'rango'):*** escribe la matriz M en una región rectangular especificada por el rango, en la primera hoja de trabajo del archive excel. El

rango se especifica usando la siguiente notación: formato tipo “celda”, por ejemplo “D2”. Esto indica que la esquina superior izquierda recibe los datos de la matriz.

- ***xlswrite(‘nombre_del_ archivo', M, hoja, 'rango'):*** Escribe la matriz M a una región rectangular especificada por el rango en la hoja de trabajo de nuestro archivo excel.

Ejemplo del uso de la función *xlswrite*

- ***Comando: xlswrite('nombre_del_archivo', M)***

Para exportar datos a un archivo Excel, primero debemos tener creada la matriz **M**. Podemos utilizar las funciones para crear matrices especiales, como *magic(), rand(), eye(), etc.* Se usará la función ***magic(10)*** para crear una matriz de nombre exporta_excel de 10 x 10.

En la ventana de comandos se ingresa:

```
>> exporta_excel= magic(10)
```

```
exporta_excel =

   92   99    1    8   15   67   74   51   58   40
   98   80    7   14   16   73   55   57   64   41
    4   81   88   20   22   54   56   63   70   47
   85   87   19   21    3   60   62   69   71   28
   86   93   25    2    9   61   68   75   52   34
   17   24   76   83   90   42   49   26   33   65
   23    5   82   89   91   48   30   32   39   66
   79    6   13   95   97   29   31   38   45   72
   10   12   94   96   78   35   37   44   46   53
   11   18  100   77   84   36   43   50   27   59
```

Una vez que tenemos creada la matriz de datos a exportar, ingresamos el comando:

>> xlswrite('ejemploca2a',exporta_excel)

De esta forma se crea un archivo de nombre ejemplocap2a.xls en el directorio donde estamos trabajando.

:

2.6.2. La función *xlsread*

Esta función lee (importa hacia matlab) archivos de Excel (`.xls`). La función *xlsread* tiene las siguientes variantes. Los archivos Excel deben encontrarse en la carpeta actual de trabajo, o indicar a Matlab el camino de búsqueda.

- ***Nombre_de_la_variable = xlsread('nombre_del_ archivo'):*** Retorna la data numérica en un doble arreglo 'num' desde la primera hoja del archivo Excel de nombre 'nombre_del_archivo'. La función xlsread ignora columnas y filas que contienen texto.

- ***Nombre_de_la_variable = xlsread(' nombre_del_ archivo', -1)***: Abre el archivo en una ventana de Excel, permitiendo seleccionar las hojas de trabajo en una forma interactiva para leer la data en ella y así importarla.

- ***Nombre_de_la_variable = xlsread('nombre_del_arhcivo', hoja)***: Lee la hoja especificada. El argumento 'hoja' se ingresa de la misma forma vista anteriormente.

- ***Nombre_de_la_variable = xlsread('nombre_del_archivo','rango')***: Lee la data desde una región rectangular específica. Análoga a la función ***xlswrite('nombre_del_ archivo', M, 'rango')***

Ejemplo del uso de la función *xlsread*

- ***Comando: Nombre_de_la_variable = xlsread('nombre_del_ archivo')***

Se tiene un archivo Excel, de nombre ***ejemplocap2.xls,*** el que contiene los siguientes datos para la presión de vapor del monóxido de carbono (CO)

P (pa)	T (K)
133	51,2
667	56
1330	58,2
2670	60,4
5330	63,2
8000	65,1
13300	67,5
26700	71,9
53300	76,9
101300	81,9
203000	89,7
507000	102,7
1013000	112,2
2030000	123,5
3040000	131,3

Se desea llevar esta tabla de datos a Matlab, ya que en el podemos realizar un sinnúmero de operaciones, como graficar, ajustar una curva, trabajar con los datos, etc.

De acuerdo a los comandos descritos, para importar esta tabla se debe escribir en la ventana de comandos lo siguiente:

```
>> importe_excel=xlsread('ejemplocap2')
```

En este caso el nombre de la variable elegido fue *importe_excel*, y de esta forma Matlab guardará en una matriz de este nombre los datos del archivo Excel. La función *xlsread* solo reconoce datos numéricos.

Al apretar la tecla ***enter*** se obtiene:

```
1.0e+006 *

  0.00013300000000   0.00005120000000
  0.00066700000000   0.00005600000000
  0.00133000000000   0.00005820000000
  0.00267000000000   0.00006040000000
  0.00533000000000   0.00006320000000
  0.00800000000000   0.00006510000000
  0.01330000000000   0.00006750000000
  0.02670000000000   0.00007190000000
  0.05330000000000   0.00007690000000
  0.10130000000000   0.00008190000000
  0.20300000000000   0.00008970000000
  0.50700000000000   0.00010270000000
  1.01300000000000   0.00011220000000
  2.03000000000000   0.00012350000000
  3.04000000000000   0.00013130000000
```

- ***Comando: Nombre_de_la_variable = xlsread(' nombre_del_ archivo', -1)***

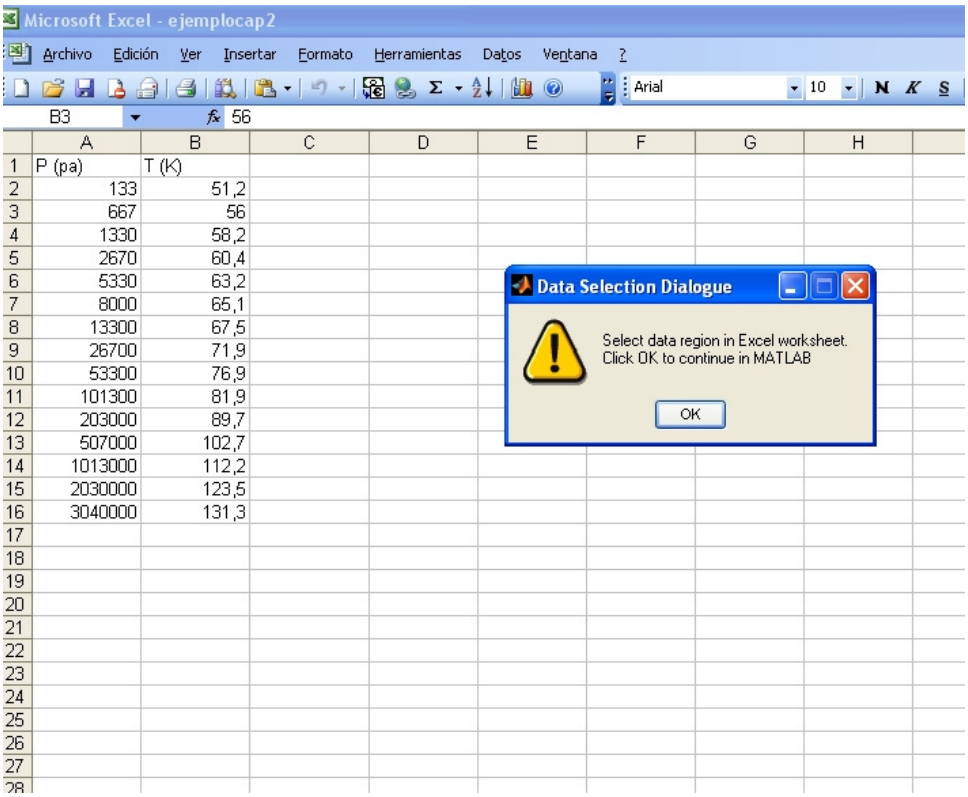
Esta función es particularmente útil, ya que permite interactuar con Excel de una forma sencilla, y así extraer los datos necesarios sin tener que importar todo el archivo Excel.

Por ejemplo, si queremos almacenar sólo los datos de la columna 1 en una variable llamada Presión, en la ventana de comandos se escribe:

```
>> presion=xlsread('ejemplocap2', -1)
```

Al apretar la tecla *enter*, se abrirá el archivo Excel y una ventana nos indica que seleccionemos la data a importar. Se selecciona la columna A con el mouse y damos click a OK en la ventana que nos retorna a Matlab

A continuación una captura de pantalla de lo descrito anteriormente:



	A	B
1	P (pa)	T (K)
2	133	51,2
3	667	56
4	1330	58,2
5	2670	60,4
6	5330	63,2
7	8000	65,1
8	13300	67,5
9	26700	71,9
10	53300	76,9
11	101300	81,9
12	203000	89,7
13	507000	102,7
14	1013000	112,2
15	2030000	123,5
16	3040000	131,3

***Figura 2.1**: Captura de pantalla al tipear el comando >> presion=xlsread('ejemplocap2', -1)*

Al hacer click en OK, se vuelve a Matlab y se tiene:

```
presion =
     133
     667
    1330
    2670
    5330
    8000
   13300
   26700
   53300
  101300
  203000
  507000
 1013000
 2030000
 3040000
```

De esta forma, se almacenan los datos en un vector columna de nombre ***presion.*** Se puede seguir el mismo procedimiento para almacenar los datos de la segunda columna:

```
>> temperatura=xlsread('ejemplocap2', -1)

temperatura =

  1.0e+002 *

   0.51200000000000
   0.56000000000000
   0.58200000000000
   0.60400000000000
   0.63200000000000
   0.65100000000000
   0.67500000000000
   0.71900000000000
   0.76900000000000
   0.81900000000000
   0.89700000000000
   1.02700000000000
   1.12200000000000
   1.23500000000000
   1.31300000000000
```

Teniendo las variables ***presion*** y ***temperatura*** podemos realizar múltiples cálculos y gráficos.

3. GRÁFICOS EN MATLAB

Uno de los grandes potenciales de Matlab es la realización de gráficos, tanto en dos como en tres dimensiones. La función *plot* se utiliza para hacer gráficos en dos dimensiones y las funciones *plot3, mesh, surface, contour*, entre otras, se utilizan para hacer gráficos en tres dimensiones.

Matlab utiliza un tipo especial de ventanas para realizar las operaciones gráficas. Ciertos comandos abren una nueva ventana y otros dibujan sobre la ventana activa, sustituyendo lo que hubiera en ella o agregando nuevos elementos gráficos a un dibujo anterior. Todo esto se verá con más detalle en las siguientes secciones.

:

3.1. Gráficos Bidimensionales

3.1.1 Funciones básicas para gráficos 2D

Matlab posee cinco funciones básicas para la creación de gráficos en 2 dimensiones. La diferencia principal entre ellas es el *tipo de escala* que utilizan en los ejes de abscisas y de ordenadas. Las funciones son las siguientes:

Función	***Descripción***
plot()	*Crea un gráfico a partir de vectores y/o columnas de matrices, con escalas lineales sobre ambos ejes*
plotyy()	*Dibuja dos funciones con dos escalas diferentes para las ordenadas, una a la derecha y otra a la izquierda de la figura.*
loglog()	*Función idéntica pero con escala logarítmica en ambos ejes*
semilogx()	*Función idéntica pero la escala es lineal en el eje de ordenadas y logarítmica en el eje de abscisas*
semilogy()	*Función idéntica pero con escala lineal en el eje de abscisas y logarítmica en el eje de ordenadas*

Se enfocará esta parte del curso a las diversas posibilidades de la función ***plot***. Las demás pueden utilizarse de una forma muy similar, y se deja planteado como desafío para practicar.

A continuación se presentan otras funciones usadas en la presentación de los gráficos, a modo de ejemplo se pueden añadir títulos al gráfico, a cada uno de los ejes, dibujar una cuadrícula auxiliar, a introducir texto, etc. Estas funciones son:

Función	***Descripción***
title('título')	*Añade un título al dibujo*
xlabel('descripción eje x')	*Añade una etiqueta al eje de abscisas. Con xlabel off desaparece*
ylabel('descripción eje y')	*Añade una etiqueta al eje de ordenadas. Con ylabel off desaparece*
text(x,y,'texto')	*Introduce 'texto' en el lugar especificado por las coordenadas x e y. Si x e y son vectores, el texto se repite por cada par de elementos. Si texto es también un vector de cadenas de texto de la misma dimensión, cada elemento se escribe en las coordenadas correspondientes*
gtext('texto')	*Introduce texto con ayuda del ratón: el cursor cambia de forma y se espera un clic para introducir el texto en esa posición*
legend()	*Define rótulos para las distintas líneas o ejes utilizados en la figura. Para más detalle, recordar siempre consultar el Help*
grid	*Activa la inclusión de una cuadrícula en el dibujo. Con grid off desaparece la cuadrícula*

3.1.2. Función *plot*

Esta es la función primordial de todos los gráficos 2-D en Matlab. La función ***plot()***, en sus diversas variantes, tiene como tarea principal dibujar argumentos almacenados ya sea en forma vectorial o matricial.

Ejemplo 3.1

Supongamos que se desea graficar la función $y = x^3$ en el siguiente intervalo: $-1 < x < 1$

Solución: El primer paso es generar la data a graficar. Para ello, en la ventana de comandos ingresamos las siguientes órdenes:

```
>> x=-1:0.1:1; % Define el rango de X
>> y=x.^3; % Eleva cada elemento de la variable x a la tercera potencia
```

La primera orden crea la variable x que contiene los valores entre -1 y 1, mediante incrementos de 0,1. También se puede usar la función linspace vista anteriormente. La segunda orden calcula la tercera potencia de *cada elemento* de ***x*** y guarda los valores en ***y***

Ahora que generamos los datos deseados, es posible graficar la función $y = x^3$ utilizando los comandos descritos en las tablas anteriores.

Se utiliza el comando plot(x,y) para graficar las variables y además con los comandos title, xlabel, ylabel, etc, podemos personalizar nuestro gráfico. A continuación se presentan las instrucciones introducidas en la ventana de comandos.

```
plot(x,y);
>> grid
>> title('Ejemplo grafico 3.1')
>> xlabel('Eje x')
>> ylabel('Eje y')
```

Con estas instrucciones Matlab crea la siguiente ventana:

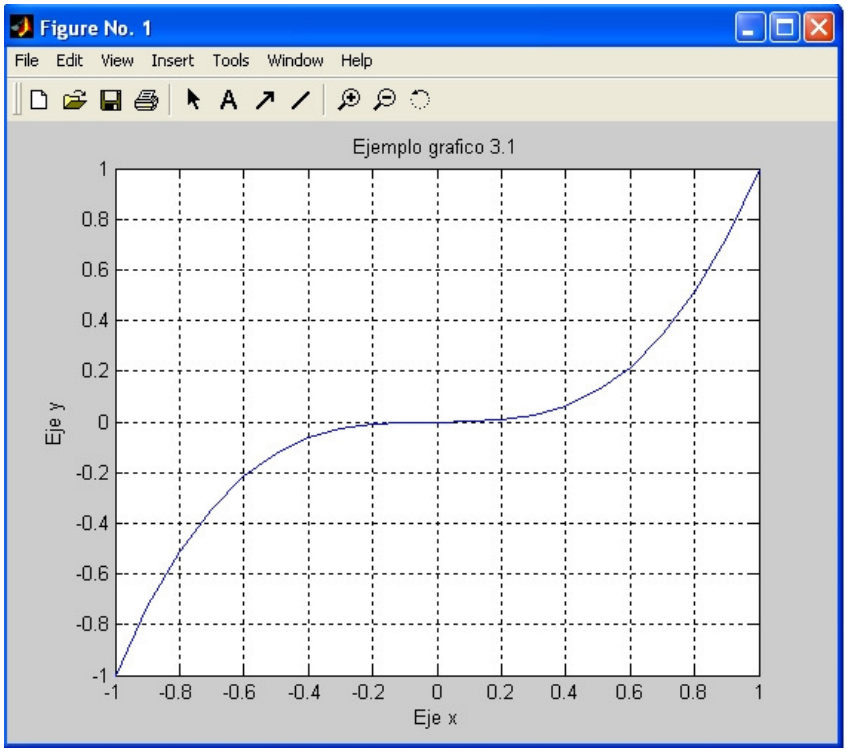


Figura 3.1: *Ejemplo creado en Matlab de la función* $y = x^3$

Ejemplo 3.2

La función ***plot()*** también permite al usuario dibujar múltiples curvas introduciendo varias parejas de vectores como argumentos. En este caso, cada uno de los segundos vectores se dibujan en ordenadas como función de los valores del primer vector de la pareja, que se representan en abscisas. Si no se modifica, para las sucesivas líneas se utilizan colores que son permutaciones cíclicas del ***azul***, ***verde***, ***rojo***, ***cyan***, ***magenta***, ***amarillo*** y ***negro***.

Se graficará la función seno y coseno. Para ello se debe ingresar las siguientes líneas en la ventana de comandos:

```
>> x=0:pi/25:6*pi;
>> y=cos(x); z=sin(x);
>> plot(x,y,x,z)
```

La gráfica correspondiente es:

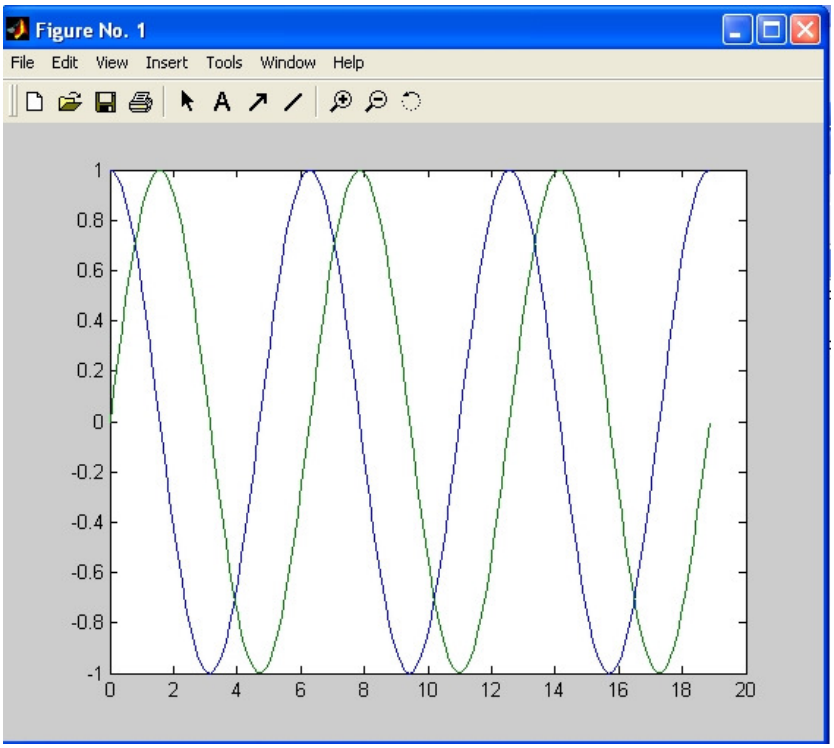


Figura 3.2: *Ejemplo 3.2*

El usuario puede personalizar el gráfico con las opciones que más le acomoden, tal como se indicó en el ejemplo 3.1.

El comando ***plot*** puede utilizarse también con matrices como argumentos. Se recomienda practicar realizando estos sencillos ejemplos:

Función	***Descripción***
plot(A)	*Dibuja una línea por cada columna de **A** en ordenadas, frente al índice de los elementos en abscisas*
plot(x,A)	*Dibuja las columnas (o filas) de **A** en ordenadas frente al vector **x** en abscisas. Las dimensiones de **A** y **x** deben ser coherentes: si la matriz **A** es cuadrada se dibujan las columnas, pero si no lo es y la dimensión de las filas coincide con la de **x**, se dibujan las filas*
plot(A,x)	*Análogo al anterior, pero dibujando las columnas (o filas) de **A** en abscisas, frente al valor de **x** en ordenadas*
plot(A,B)	*Dibuja las columnas de **B** en ordenadas frente a las columnas de **A** en abscisas, dos a dos. Las dimensiones deben coincidir*
plot(A,B,C,D)	*Análogo al anterior para cada par de matrices. Las dimensiones de cada par deben coincidir, aunque pueden ser diferentes de las dimensiones de los demás pares*

Una excelente y breve descripción de la función ***plot()*** se puede obtener con el comando ***help plot*** o ***helpwin plot***.

3.1.3. Estilos de línea y marcadores en la función *plot*

Hasta el momento se ha visto cómo la tarea fundamental de la función ***plot()*** era dibujar los valores de un vector en ordenadas, frente a los valores de otro vector en abscisas. En la tabla siguiente se pueden observar las distintas posibilidades con las que se puede personalizar el estilo del gráfico, ya sea el color y tipo de línea o el maker a utilizar.

Símbolo	***Color***	***Símbolo***	***Marcador (maker)***
y	*Amarillo*	**.**	*Puntos*
m	*Magenta*	**o**	*Círculos*
c	*Calipso*	**x**	*Marcas en x*
r	*Rojo*	**+**	*Marcas en +*
g	*Verde*	*****	*Marcas en **
b	*Azul*	**s**	*Marcas cuadradas*
w	*Blanco*	**d**	*Marcas en diamantes*
k	*Negro*	**^**	*Triángulo apuntando arriba*
		v	*Triángulo apuntando abajo*
Símbolo	***Estilo de línea***	**>**	*Triángulo apuntando a la derecha*
-	*Líneas continuas*	**<**	*Triángulo apuntando a la izquierda*
:	*Líneas a puntos*	**p**	*Estrella de 5 puntas*
-.	*Líneas a barra-punto*	**h**	*Estrella de 6 puntas*
_	*Líneas a trazos*		

A modo de ejemplo, se graficará una ecuación de la recta y se personalizarán las opciones de línea, marcadores, colores, etc.

```
>> x=1:0.2:2;
>> y=x.*4+2;
>> plot(x,y,'-.rs', 'LineWidth',4, 'MarkerEdgeColor','k', 'MarkerFaceColor', 'g',
'MarkerSize',10)
>> title('Ejemplo ecuación de la recta')
>> xlabel('Eje x')
>> ylabel('Eje y')
```

El resultado obtenido es:

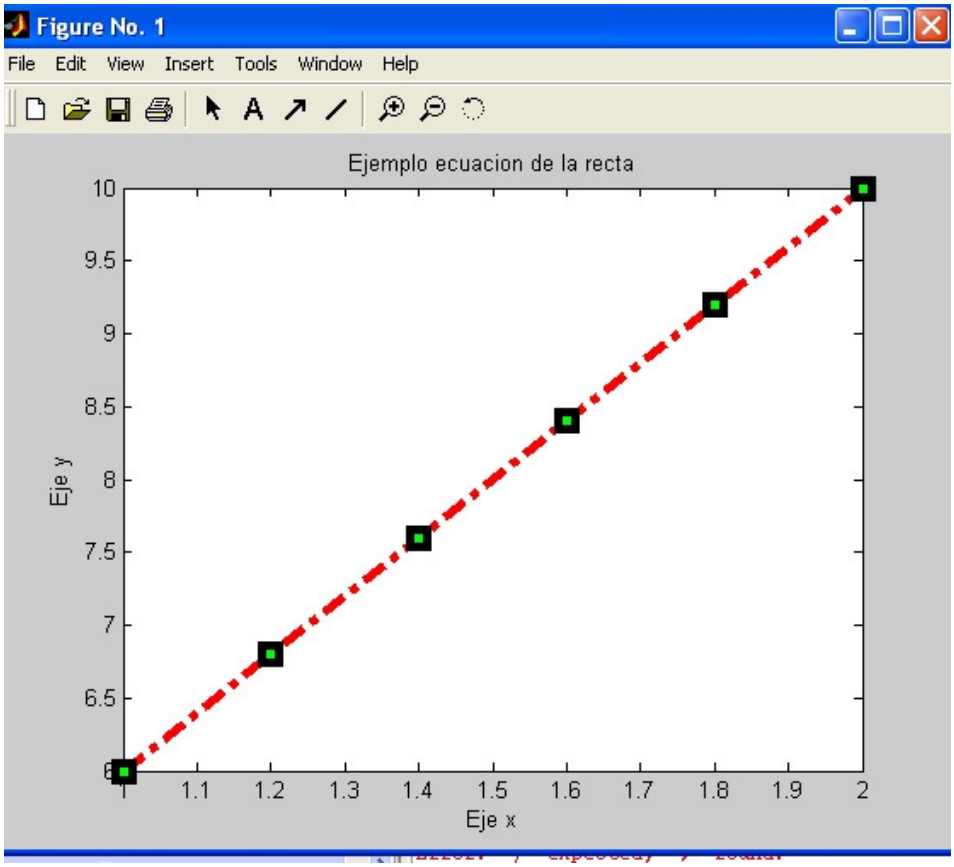


Figura 3.3: *Ejemplo de utilización de marcadores y estilos de línea*

3.1.4. Otras funciones gráficas 2D

Matlab posee también otras funciones gráficas bidimensionales que reproducen otro tipo de gráficos, distintos de los que produce la función ***plot()*** y sus análogas. Algunas de estas funciones son las siguientes (para más información sobre cada una de ellas en particular, utilizar ***help nombre_función***:

Función	***Descripción***
bar()	*Crea diagramas de barras*
barh()	*Diagramas de barras horizontales*
bar3()	*Diagramas de barras con aspecto 3-D*
bar3h()	*Diagramas de barras horizontales con aspecto 3-D*
pie()	*Gráficos con forma de "tarta"*
pie3()	*Gráficos con forma de "tarta" y aspecto 3-D*
area()	*Similar **plot()**, pero rellenando en ordenadas de 0 a y*
errorbar()	*Representa sobre una gráfica –mediante barras– valores de errores*
hist()	*Dibuja histogramas de un vector*

A modo de ejemplo, construya un vector de valores aleatorios entre 0 y 10, y ejecútense los comandos:

```
>> x=[rand(1,100)*10];
>> bar(x)
>> hist(x)
```

Los resultados serán los siguientes:

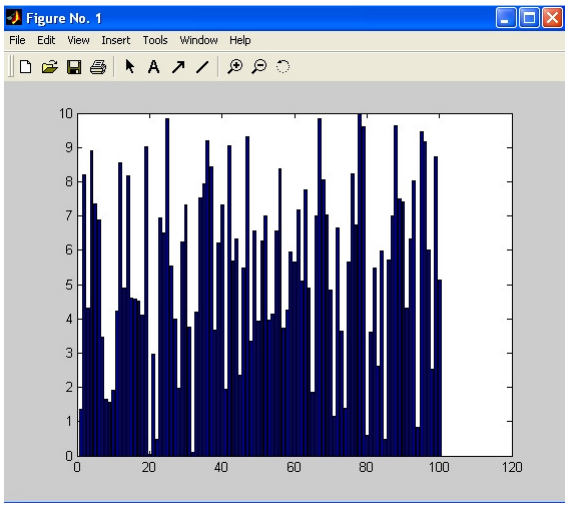


Figura 3.4: *Gráfico de la función bar(x)*

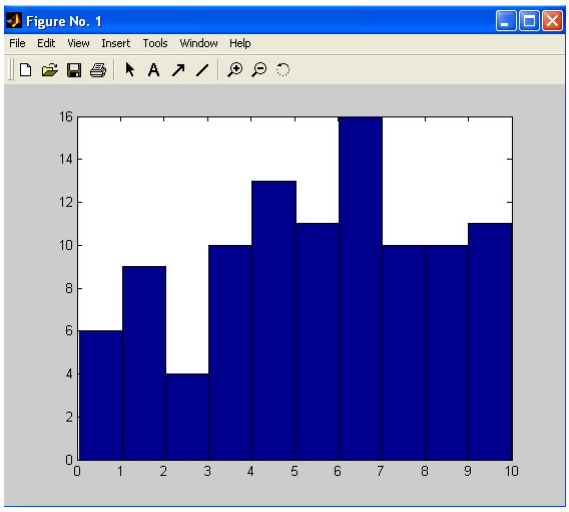


Figura 3.5 : *Gráfico de la función hist(x)*

3.2. Gráficos Tridimensionales

3.2.1 Tipos de funciones gráficas tridimensionales

Matlab tiene el potencial de realizar varios tipos de gráficos en 3D. La primera forma de gráfico 3D es la función ***plot3***, análogo a la función *plot* utilizada en gráficos 2D. Esta función dibuja puntos cuyas coordenadas están contenidas en 3 vectores. A modo de ejemplo, ingresar el siguiente comando:

```
>> fi=[0:pi/20:6*pi]; plot3(cos(fi),sin(fi),fi,'r'), grid
```

Se obtiene:

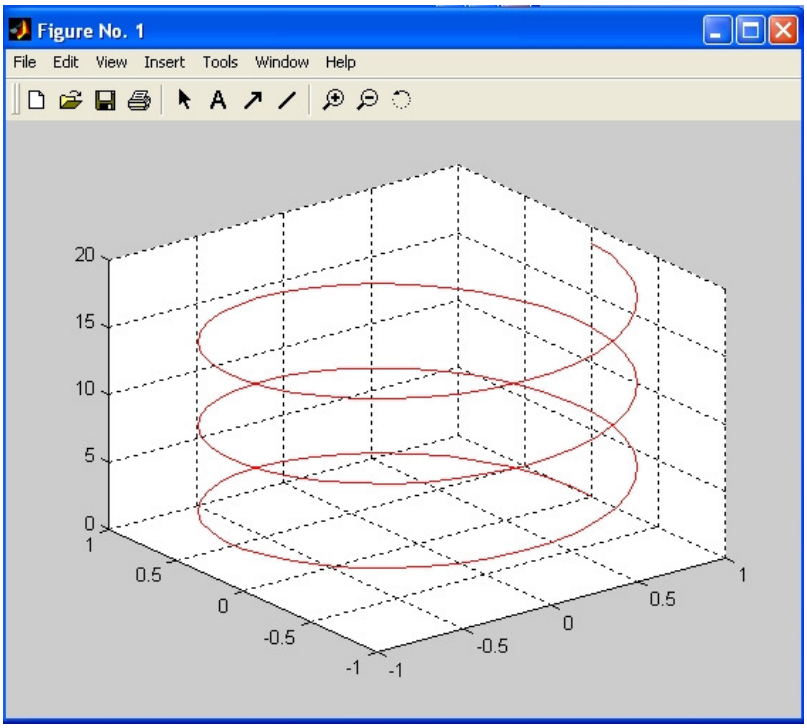


Figura 3.6: *Gráfico en 3D*

Se realizará el ejemplo para una función de dos variables. Para ello se va a definir una función de este tipo en un archivo llamado ***test3d.m***. La fórmula será la siguiente:

$$z = 3\cdot(1-x)^2\cdot e^{-x^2-(y+1)^2} - 10\cdot\left(\frac{x}{5} - x^3 - y^5\right)\cdot e^{-x^2-y^2} - \frac{1}{3}\cdot e^{-(x+1)^2-y^2}$$

El archivo ***test3d.m*** debe tener los siguientes comandos:

```
function z=test3d(x,y)
z = 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) - 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) -
1/3*exp(-(x+1).^2 - y.^2);
```

Ahora, escribir los siguientes comandos y ejecutar:

```
>> x=[-3:0.4:3]; y=x;
>> close
>> subplot(2,2,1)
```

```
>> figure(gcf),fi=[0:pi/20:6*pi];
>> plot3(cos(fi),sin(fi),fi,'r')
>> grid
>> [X,Y]=meshgrid(x,y);
>> Z=test3d(X,Y);
>> subplot(2,2,2)
>> figure(gcf), mesh(Z)
>> subplot(2,2,3)
>> figure(gcf), surf(Z)
>> subplot(2,2,4)
>> figure(gcf), contour3(Z,16)
```

Se obtendrá:

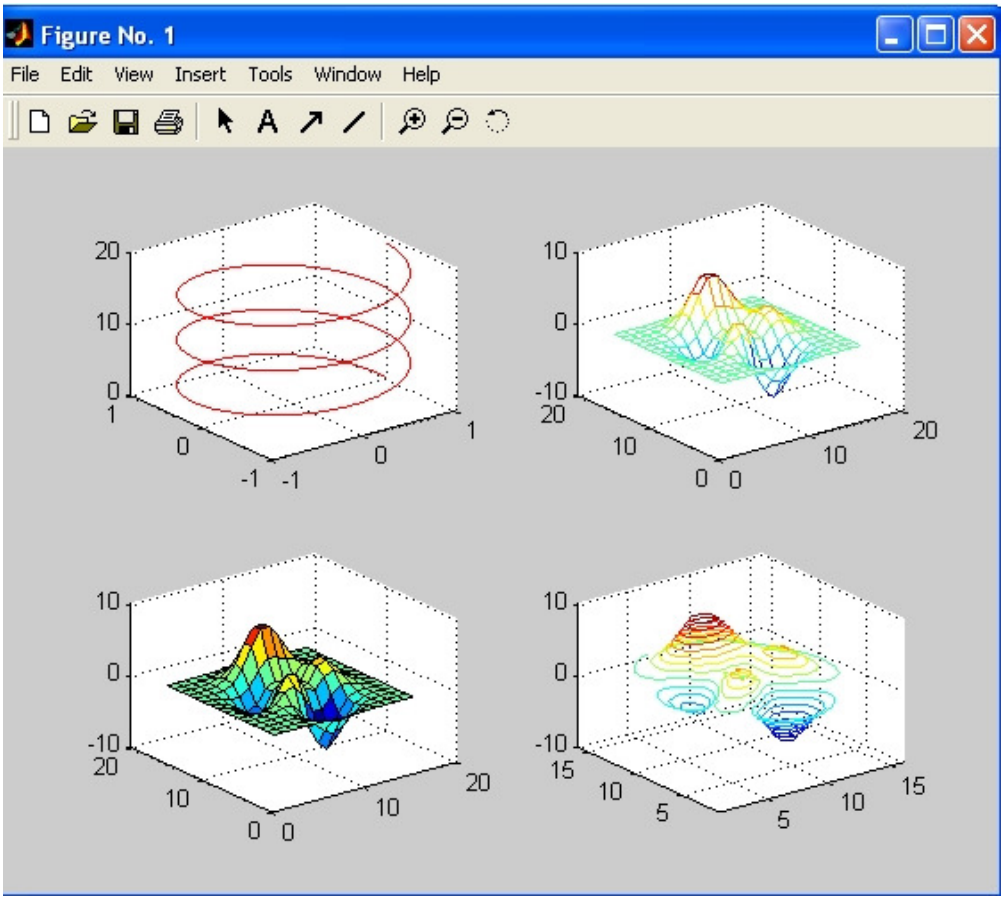


Figura 3.7: *Distintas funciones en 3D*

4. FUNCIONES Y ARCHIVOS '*.m*'

Matlab incorpora un gran número de ***funciones intrínsecas*** (escritas en el propio código ejecutable del programa). Estas funciones son particularmente rápidas y eficientes. Para problemas complejos, las herramientas provistas por la ventana de comandos y su mecanismo de guardado de variables son insuficientes. Una mejor opción es crear los llamados archivos de instrucciones de Matlab, que poseen la extensión ****.m*** (m-files o archivos .m). Estas funciones potencian en gran manera las capacidades del programa.

4.1. Aspectos generales de las funciones de Matlab

El concepto de función en Matlab es similar a ***C*** y otros lenguajes de programación, no obstante posee algunas diferencias importantes. Al igual que en ***C***, una función tiene ***nombre***, ***valor de retorno*** y ***argumentos***. Una función *se llama* mediante su nombre en una expresión o usando éste como un comando más. Las funciones se pueden definir en archivos de texto ****.m*** como se verá en este capítulo. A continuación, un ejemplo sencillo:

```
>> x=2 ; y=7;
>> r = sqrt(x^2+y^2)
>> a = log(x) - log(y)
r =
    7.2801

a =
   -1.2528
```

En el ejemplo anterior se utilizaron dos funciones matemáticas conocidas como la raíz cuadrada y el logaritmo natural. Se pueden identificar los componentes principales de las funciones en Matlab:

- Los ***nombres*** de las funciones se han puesto en **negrita**.
- Los ***argumentos*** de cada función van a continuación del ***nombre***, entre paréntesis, y separados por comas si hay más de uno.
- Los ***valores de retorno*** son el resultado de efectuar las instrucciones de la función sobre los *argumentos.*

Los nombres de las funciones de Matlab ***no son palabras reservadas*** del lenguaje. Se puede crear una variable llamada ***sqrt*** o ***log***, que ocultan las funciones correspondientes. Para poder acceder a las funciones hay que eliminar (usando el comando ***clear***) las variables del mismo nombre que las ocultan.

4.2. Funciones predefinidas

Matlab posee diversos tipos de funciones**, se enumeran a continuación las más importantes, clasificadas según su finalidad:

1.- Funciones matemáticas elementales.
2.- Funciones especiales.
3.- Funciones matriciales elementales.
4.- Funciones matriciales específicas.
5.- Funciones para la descomposición y/o factorización de matrices.
6.- Funciones para análisis estadístico de datos.
7.- Funciones para análisis de polinomios.
8.- Funciones para integración de ecuaciones diferenciales ordinarias.
9.- Resolución de ecuaciones no-lineales y optimización.
10.- Integración numérica.
11.- Funciones para procesamiento de señal.

**En este manual se verán sólo algunas de estas funciones, por lo tanto, el lector deberá profundizar por su cuenta el uso de alguna de ellas, en caso de ser necesario. Para ello tiene disponible el manual del usuario de Matlab y otros recursos disponibles en Internet.

4.2.1 Características generales de todas las funciones de Matlab:

- Las operaciones de suma y/o resta de una matriz con un escalar consisten en sumar y/o restar el escalar a todos los elementos de la matriz.
- Introduciendo en la ventana de comandos **>> *help nombre_funcion*** se obtiene de inmediato información sobre la función de ese nombre. En el ***Help Desk*** aparecen enlaces a "*Functions - By Cathegory*" y "*Functions – Alphabetical List*", en donde aparecen relaciones completas de las funciones disponibles en Matlab.

- Existe una equivalencia entre las funciones y los comandos con argumentos de Matlab. Así, un comando en la forma: **>> comando arg1 arg2** es equivalente a una función con el mismo nombre que el comando a la que los argumentos se le pasan como cadenas de caracteres, **>> comando('arg1', 'arg2')** Esta dualidad entre comandos y funciones es sobre todo útil en programación, porque permite "*construir*" los argumentos con las operaciones propias de las cadenas de caracteres.

4.2.2. Funciones matemáticas elementales que operan de modo escalar

Existen funciones que sólo operan sobre escalares, pero que pueden operar componente a componente cuando se aplican sobre matrices y/o vectores creando otra matriz y/o vector. Por tanto, se aplican de la misma forma a escalares, vectores y matrices. Estas funciones comprenden las funciones matemáticas trascendentales y otras funciones básicas. A continuación ejemplos de esta categoría:

Función	***Descripción***
sin(x)	*Seno*
cos(x)	*Coseno*
tan(x)	*Tangente*
log(x)	*Logaritmo natural*
log10(x)	*Logaritmo decimal (en base 10)*
exp(x)	*Función exponencial*
sqrt(x)	*Raíz cuadrada*
sign(x)	*Devuelve -1 si <0, 0 si =0 y 1 si >0.*
rem(x,y)	*Resto de la división (2 argumentos que no tienen que ser enteros)*
abs(x)	*Valores absolutos*
angle(x)	*Ángulos de fase*
round(x)	*Redondeo hacia el entero más próximo*

4.2.3. Funciones que actúan sobre vectores

Otras funciones Matlab **sólo operan sobre vectores** (fila o columna) creando un escalar, pero **pueden operar columna a columna cuando se aplican sobre matrices,** creando un vector fila que contiene los resultados de su aplicación sobre cada columna. Si queremos que actúe fila a fila hacemos que opere sobre la transpuesta de la matriz. Algunos ejemplos:

Función	***Descripción***
[xm,im]=max(x)	*Máximo elemento de un vector. Devuelve el valor máximo **xm** y la posición que ocupa **im***
min(x)	*Mínimo elemento de un vector. Devuelve el valor mínimo y la posición que ocupa*
sum(x)	*Suma de los elementos de un vector*
cumsum(x)	*Devuelve el vector suma acumulativa de los elementos de un vector (cada elemento del resultado es una suma de elementos del original)*
mean(x)	*Valor medio de los elementos de un vector*
std(x)	*Desviación típica*
prod(x)	*Producto de los elementos de un vector*
cumprod(x)	*Devuelve el vector producto acumulativo de los elementos de un vector*
[y,i]=sort(x)	*Ordenación de menor a mayor de los elementos de un vector **x**. Devuelve el vector ordenado **y**, y un vector **i** con las posiciones iniciales en **x** de los elementos en el vector ordenado **y**.*

4.2.4. Funciones que actúan sobre matrices

Las siguientes funciones sólo operan sobre el/los argumento/s que sean matrices. En este grupo aparecen algunas de las funciones más útiles y potentes de Matlab. Se subdividen en varios grupos:

- ***Funciones matriciales elementales:*** son funciones básicas, algunas ya se habían revisado en el capítulo 2.

Función	***Descripción***
B = A'	*calcula la traspuesta (conjugada) de la matriz* ***A***
B = A.'	*calcula la traspuesta (sin conjugar) de la matriz* ***A***
v = poly(A)	*devuelve un vector* ***v*** *con los coeficientes del polinomio característico de la matriz cuadrada* ***A***
t = trace(A)	*devuelve la traza* ***t*** *(suma de los elementos de la diagonal) de una matriz cuadrada* ***A***
[m,n] = size(A)	*devuelve el número de filas* ***m*** *y de columnas* ***n*** *de una matriz rectangular* ***A***
n = size(A)	*devuelve el tamaño de una matriz cuadrada* ***A***
f = size(A,1)	*devuelve el número de filas de* ***A***
nc = size(A,2)	*devuelve el número de columnas de* ***A***

- ***Funciones matriciales especiales:*** Las funciones ***exp()***, ***sqrt()*** y ***log()*** se aplican ***elemento a elemento*** a las matrices y/o vectores que se les ingresan como argumentos. Existen otras funciones similares que tienen también sentido cuando se aplican a una matriz como una única entidad. Estas funciones son las siguientes (se distinguen porque llevan una "m" adicional en el nombre):

Función	***Descripción***
expm(A)	*Si* ***A=XDX'****, expm(****A****) =* ***X*****diag(exp(diag(****D****)))*****X'***
sqrtm(A)	*Devuelve una matriz que multiplicada por sí misma da la matriz* ***A***
logm()	*Es la función recíproca de expm(****A****)*

Aunque no pertenece a esta familia de funciones, se puede considerar al ***operador potencia*** (^) dentro de esta clasificación. Así:

A^n está definida si **A** es cuadrada y **n** un número real. Si **n** es entero, el resultado se calcula por multiplicaciones sucesivas. Si **n** es real, el resultado se calcula como: **A^n=X*D**.^n***X'** siendo [**X**,**D**]=eig(**A**)

- ***Funciones de factorización y/o descomposición:*** esta categoría se puede sub-dividir en 4 niveles. Estas funciones no serán tratadas en profundidad, y el lector deberá profundizar en ellas, si lo requiere.

  - Funciones basadas en la factorización triangular (eliminación de Gauss).

:

  - Funciones basadas en el cálculo de valores y vectores propios.

:

  - Funciones basadas en la descomposición QR.

  - Funciones basadas en la descomposición de valores singulares.

- ***Función linsolve():*** la función ***linsolve*** es la forma más eficiente de que dispone Matlab para resolver sistemas de ecuaciones lineales. La forma general de la función ***linsolve*** para resolver **Ax=b** es: **>> x = linsolve(A,b)**

4.3. Archivos '*.m'

Los *archivos .m* pueden ser ***scripts***, que simplemente ejecutan una serie de órdenes o instrucciones de MATLAB; o pueden ser **funciones**, que además aceptan argumentos y producen resultados. Se crea un *M-file* utilizando un editor de textos. Desde la versión 5.3 en adelante, Matlab tiene su propio editor (*medit*). Se utiliza el editor para escribir el *archivo .m*, posteriormente se graba y se llama directamente desde la línea de comandos de Matlab como si fuera cualquier otra orden de las que ya se han revisado en este manual.

• Archivos ***Script.*** Son archivos .m que no constituyen funciones y que se construyen mediante una secuencia de instrucciones. El contenido de un archivo de programas Matlab de nombre

nombre.m se ejecuta tecleando simplemente su nombre en la ventana de comandos. En el próximo capítulo se profundizará este tema, junto con la programación en lenguaje Matlab.

• ***Archivos de función***: Son también archivos .m, pero a diferencia de los anteriores, se le pueden pasar argumentos y pueden devolver resultados. La mayoría de los archivos contenidos en los *toolboxes* son funciones.

4.3.1. Archivos de función

Son aquellos cuya primera línea ejecutable (no de comentario) comienza con la palabra ***function.*** Una función se define con un m-archivo, cuyo nombre coincide con el de la función. La primera línea ejecutable es

```
function argumentos salida = nombre_función (argumentos entrada)
```

A continuación siguen las instrucciones necesarias. Cuando hay más de un argumento de salida, éstos deben ir entre corchetes y separados por comas. Por ejemplo:

```
function y=f(x)
function [a,b,c]=g(x,y)
```

Es altamente conveniente comenzar las primeras líneas del archivo con un comentario (iniciándolas con el símbolo %), explicando cómo debe usarse la función y sus argumentos (tanto de entrada como de salida). De esta manera, dicha explicación será visible al ingresar la instrucción ***help nombre_función*** en la ventana de comandos. La función puede finalizarse en cualquier momento utilizando la instrucción *return*. En el editor de archivos m, se vería en forma similar a la siguiente:

```
function [out1,out2,...] = nombre_archivo (in1,in2,...)
% Comentarios adicionales para el help
comandos de Matlab
return;
```

4.3.2. Instrucciones de entrada y salida

Es necesario comentar que existen una serie de utilidades a la hora de programar una función en Matlab. Las más comunes son:

- ***input:*** Muestra una cadena de caracteres por pantalla y espera a que el usuario introduzca un valor, que generalmente será asignado a una variable. La instrucción es: x=input('mensaje'[,'s']). La opción 's' se emplea para leer una variable de tipo carácter ('string'), evitando los apóstrofos.
  Ejemplos:

  a=input('Número de filas: ').
  m=input('Nombre del archivo: ','s').

- ***disp:*** Muestra una cadena de caracteres por pantalla. La instrucción es: disp('mensaje') ´o disp('texto').

- ***pause***: Detiene la ejecución del programa

4.3.3 Aplicaciones.

- ***Ejemplo 4.1:*** en reiteradas ocasiones interesa utilizar funciones que Matlab no tiene predefinidas. Por ejemplo, suponga que se necesita evaluar, en repetidas ocasiones la siguiente expresión.

$$fun(x, y) = x^3 + \frac{3}{4} \cdot y^2 + \sqrt{\frac{\left(x^2 + y^3\right)}{5}}$$

Se definirá una función propia en un archivo .m nuevo, que debe guardarse o añadirse al directorio de trabajo (current directory) cada vez que se evalúe la función.

Se debe iniciar Matlab desde el icono del escritorio, luego ir a File/New/m-file. Se abrirà el editor de archivos m, donde se ingresa el siguiente código.

La siguiente figura contiene el código del archivo .m que define la función fun(x, y)

```
%Primer ejemplo creación de funciones propias.
function resultado= fun(x,y)
% Variables de entrada: x, y
% Variable de salida: resultado

%operaciones
resultado=(x^3)+3/4*(y^2)+sqrt(((x^2)+(y^3))/5);
return;
```

El archivo .m debe guardarse en el directorio de trabajo actual, con el mismo nombre de la función, en es este caso el archivo debe llamarse: ***fun.m***

Una vez está definido el archivo .m, en la ventana de comandos se puede trabajar con la función de distintas formas:

```
>> b=5; c=3;
>> a=fun(b,c)
a =
  134.9749

>> a=fun(5,3)
a =
  134.9749

>> x=5;y=3;
>> fun(x,y)
ans =
  134.9749
```

Como puede observarse, no es necesario, al evaluar una función, usar las variables con los mismos nombres que aparecen en la definición de la función en el archivo m.

- ***Ejemplo 4.2:*** Crear una archivo .m que convierta la temperatura desde grados Celsius, a Fahrenheit. Además que solicite al usuario ingresar la temperatura.

Para crear esta función, se siguen los mismos pasos que el ejemplo anterior, pero ahora usamos las utilidades de la sección 4.3.2. El código resulta:

```
function y = tconvert(x)
% Función que convierte grados Celcius (°C) en grados Farenheit (°F)
% Requiere que el usuario ingrese la temperatura en °C

x=input('Ingrese temperatura en °C:');

y = 1.8*x + 32;

disp('Aprete cualquier tecla para continuar')
pause
disp('La temperatura en °F es')
return;
```

Se guarda el archivo con el nombre de la función, esto es**: tconvert.m**. Recordar siempre guardar este archivo en el directorio actual de trabajo. Para llamar a esta función desde la ventana de comandos, simplemente ingresar ***tconvert*** y seguir las instrucciones:

```
>> tconvert
Ingrese temperatura en ºC:100   ← Esto lo ingresa el usuario y a continuación apreta ENTER.
Aprete cualquier tecla para continuar  ← Apretar enter o cualquier otra tecla.
La temperatura en ºF es
ans =
        212
```

5. PROGRAMACIÓN EN MATLAB.

Cada vez que creamos un archivo .m (m-file), estamos escribiendo un programa computacional, usando el lenguaje de programación de Matlab. Se puede hacer mucho en Matlab, usando solamente las técnicas más sencillas de programación que el lector ya debe conocer. En este capítulo se cubrirán los comandos y técnicas de programación que son útiles para abordar problemas un poco más complejos

5.1. Operadores relacionales y lógicos

Matlab es capaz de respondernos si ciertas expresiones son verdaderas o falsas. Los operadores relacionales y lógicos permiten la comparación de escalares (o de matrices elemento a elemento). Si el resultado de la comparación es verdadero, devuelven un 1, en caso contrario devuelven un 0. Los operadores elementales son:

<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual que
~=	No igual que
&	Conjunción (AND)
|	Disjunción (OR)
~	Negación (NOT)

Es importante no dejar espacios entre los operadores formados por dos símbolos. Si los datos a comparar son matrices, la comparación se hace elemento a elemento, devolviendo una matriz binaria.

- ***Ejemplo a:*** Si comparamos dos matrices del mismo orden, Matlab nos responde con la matriz que resulta al comparar elementos con mismos índices:

```
>> [1 5; 2 4] >= [1 2; 5 7]

ans =

   1   1
   0   0
```

- ***Ejemplo b:*** Si comparamos una matriz con un número, el resultado es una matriz que muestra el valor lógico de la comparación de cada elemento de la matriz con dicho número.

```
>> [1 2; 3 7] >= 3

ans =

   0   0
   1   1
```

- ***Ejemplo C:*** utilización de los operadores lógicos &, | y ~

```
>> p=[1 0 1 0]; q=[1 0 1 1];
>>  [p&q;p|q;~p]

ans =

      1   0   1   0
      1   0   1   1
      0   1   0   1
```

5.2. Ciclos y estructuras condicionales

Es habitual que al resolver cierto problema se tenga que repetir varias veces un cierto número de instrucciones, como ocurre, por ejemplo, al programar el método de Newton para aproximar raíces. Para realizar esta operación de forma cómoda, los lenguajes de programación disponen de ciertas estructuras que reciben el nombre de ***Ciclos (o loops).*** Matlab tiene un lenguaje de programación que –como cualquier otro lenguaje– dispone de sentencias para realizar bifurcaciones y ciclos. Las bifurcaciones permiten realizar una u otra operación según se cumpla o no una determinada condición. La Figura 5.1 muestra tres posibles formas de bifurcación.

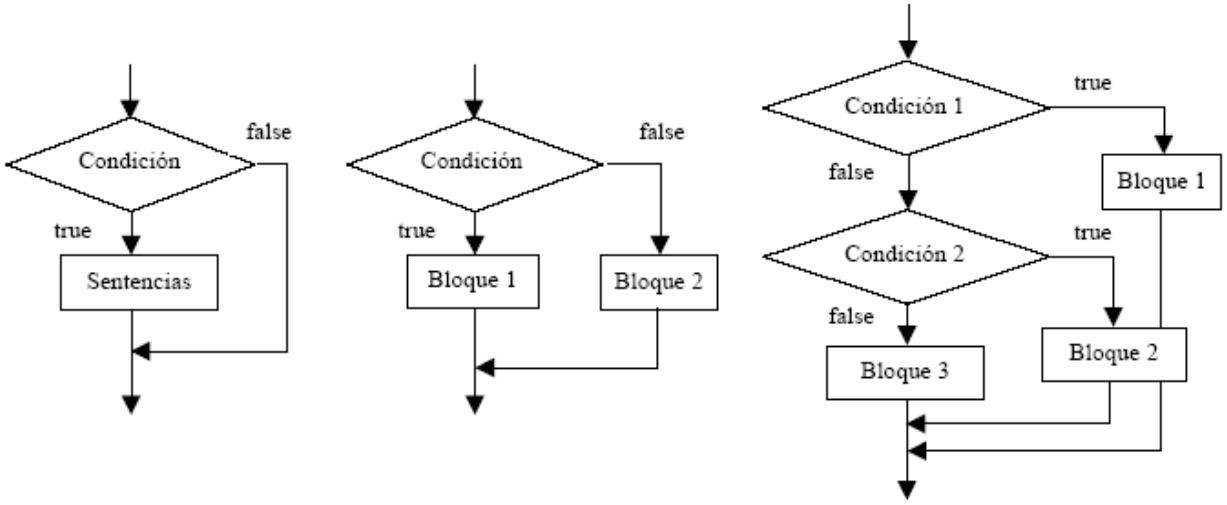


Figura 5.1: *Ejemplos de bifurcaciones*

Los ***Ciclos*** permiten repetir las mismas o análogas operaciones sobre datos distintos. Mientras que en C/C++/Java el "cuerpo" de estas sentencias se determinaba mediante llaves {...}, en Matlab se utiliza la palabra ***end*** con análoga finalidad. Existen también algunas otras diferencias de sintaxis.

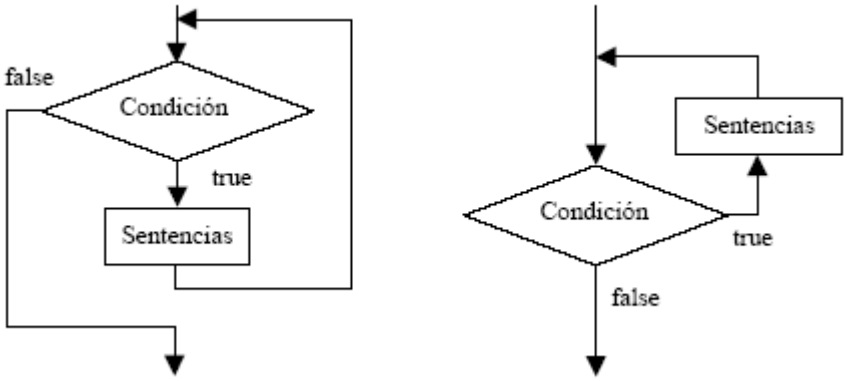


Figura 5.2: Ciclos con control al principio y al final.

La Figura 5.2 muestra dos posibles formas de ciclo, con el control situado al principio o al final del mismo. Si el control está situado al comienzo del ciclo es posible que las sentencias no se ejecuten ninguna vez, por no haberse cumplido la condición cuando se llega al ciclo por primera vez. Sin embargo, si la condición está al final del ciclo las sentencias se ejecutarán por lo menos una vez, aunque la condición no se cumpla. Muchos lenguajes de programación disponen

de Ciclos con control al principio (***for*** y ***while*** en C/C++/Java) y al final (***do … while*** en C/C++/Java). En Matlab no hay Ciclos con control al final del ciclo, es decir, no existe construcción análoga a ***do ... while***. Las bifurcaciones y Ciclos no sólo son útiles en la preparación de programas o de archivos ****.m***. También se aplican con frecuencia en el uso interactivo de Matlab.

5.2.1. Sentencia *if*

La forma general incluye *bifurcaciones múltiples*, en la que pueden encadenarse tantas condiciones como se necesiten, y tiene la siguiente estructura:

```
if condicion1
bloque1
elseif condicion2
bloque2
elseif condicion3
bloque3
.
.
.
else % opción por defecto para cuando no se cumplan las condiciones 1,2,3….
bloque4
end
```

La opción por defecto ***else*** puede ser omitida: si no está presente no se hace nada en caso de que no se cumpla ninguna de las condiciones que se han chequeado.

Y para el caso más sencillo, se puede ver el siguiente ejemplo:

```
if (a < b)
c = a;
else
c = b
end
```

5.2.2. Sentencia *switch*

La sentencia switch es útil cuando se tienen varios if anidados. El comando switch evalúa una variable, y en función del valor de dicha variable, realiza un conjunto de acciones u otro. Forma general:

```
switch switch_expresion
case case_expr1,
bloque1
case {case_expr2, case_expr3, case_expr4,...}
bloque2
...
otherwise, % opción por defecto
bloque3
end
```

En el siguiente ejemplo, la variable *a* es evaluada por switch en la primera línea del código. Si el valor de la variable ***a*** es 1, se realizan las operaciones incluidas en el apartado case 1, si el valor de la variable ***a*** es 2, se realizan las operaciones incluidas en el apartado case 2, ... Si la variable ***a*** no toma ninguno de los valores especificados en los distintos *case* se ejecutan las órdenes que aparecen en el apartado otherwise.

```
switch a
case 1
órdenes a ejecutar si a = 1
case 2
órdenes a ejecutar si a = 2
case 3
órdenes a ejecutar si a = 3
...
otherwise
órdenes a ejecutar si a no toma ninguno
de los valores especificados en los distintos case
end
```

5.2.3. Sentencia *for*

La sentencia *for* permite que una sentencia, o grupo de sentencias, pueda ser repetida un numero fijo y predeterminado de veces. Pueden incluirse Ciclos anidados (unos dentro de otros). Su sintaxis es la siguiente:

```
for i=1:n
sentencias
end
```

O también:

```
for i=vectorValores
sentencias
end
```

Ejemplo, incluyendo dos sentencias *for*

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

Ahora, un ejemplo de la creación de un archivo .m de funciones, utilizando la sentencia *for* e *if*

```
function h = fun_factorial(n)
%fun_factorial calcula el factorial de n

if (n == 0)
        h = 1;
else
        h = 1;
for i=1:n
        h = h*i;
end %final de la sentencia for
end % final de la sentencia if
```

Se debe guardar como fun_factorial.m y al llamarlo desde la ventana de comandos, ingresar fun_factorial(cualquier número entero).

5.2.4. Sentencia *while*

El ciclo *while* permite que una sentencia o grupo de sentencias sean ejecutadas un número indefinido de veces mientras una expresión lógica sea verdadera. Su sintaxis es la siguiente:

```
while condición
sentencias
end
```

En que *condición* puede ser una expresión vectorial o matricial. Las *sentencias* se siguen ejecutando mientras haya elementos distintos de cero en *condición*, es decir, mientras haya algún o algunos elementos ***true***. El ciclo se termina cuando *todos los elementos* de *condición* son ***false*** (es decir, cero).

Si por ejemplo se quiere averiguar cuál es el mayor entero cuyo factorial es menor que 100:

```
n=1;
while prod(1:n)<100
n=n+1;
end
```

5.2.4. Sentencia *Break*

Al igual que en C/C++/Java, la sentencia *break* permite que se termine la ejecución del ciclo *for* y/o *while* más interno de los que comprenden a dicha sentencia.

5.2.5. Sentencia *Continue*

La sentencia ***continue*** hace que se pase inmediatamente a la siguiente iteración del ciclo ***for*** o ***while***, saltando todas las sentencias que hay entre el ***continue*** y el fin del ciclo en la iteración actual.

5.3. Órdenes de entrada /salida.

Cuando se quiere indicar el valor de alguna variable o expresión en la ejecución de un programa, es conveniente utilizar la orden *disp* para evitar que aparezca además el nombre de la variable. Matlab ofrece la posibilidad de asignar un valor a una variable desde el teclado en el transcurso de la ejecución de un programa. Con var= input(' mensaje') se muestra en pantalla la cadena de caracteres mensaje y aparece un cursor parpadeante en la misma hasta que introduzcamos una expresión, que se convertirá en el valor de la variable var.

5.4. Órdenes de ruptura.

Puede ser que al ejecutar un programa, se desee detener la ejecución del mismo definitivamente o hasta que pase cierto intervalo de tiempo. Para esto, el lenguaje de programación de Matlab cuenta con las siguientes órdenes:

5.4.1. La orden *break*.

Detiene la ejecución de todos los archivos .m que se estén ejecutando en ese momento y regresa a MATLAB con el símbolo >> y un mensaje de ruptura.

5.4.2. La orden *return.*

Detiene la ejecución del archivo .m donde se halle esta instrucción, es decir, si el archivo donde se encuentra la orden *return* había sido llamado por uno anterior, continua la ejecución de éste.

5.4.3. La instrucción *pause(x)*

Realiza una pausa de *x* segundos antes de ejecutar la siguiente orden de programa.

5.4.4. La orden *error('x')*

Detiene el desarrollo de la ejecución y muestra el mensaje ***x*** en pantalla acompañado de un mensaje de error.