****ITC **INSTITUTO TECNOLÓGICO**

**DE CULIACÁN**

DEPARTAMENTO DE INGENIERÍA

ELÉCTRICA-ELECTRÓNICA

**DIEE**

Curso:

**MATLAB**

Material compilado por:

MC. Omar Iván Gaxiola Sánchez

Dr. Guillermo Rubio Astorga

Dr. Modesto Guadalupe Medina Melendrez

Septiembre de 2019.

Contenido.

1. Introducción.
2. Álgebra lineal y matrices.
3. Gráficas.
4. Programación en MATLAB
5. **Introducción.**

**Ventana inicial de MATLAB.**

Esta es la vista que se obtiene eligiendo la opción ***Desktop Layout/Default***, en el menú ***View***. Como esta configuración puede ser cambiada fácilmente por el usuario, es posible que en muchos casos concretos lo que aparezca sea muy diferente.

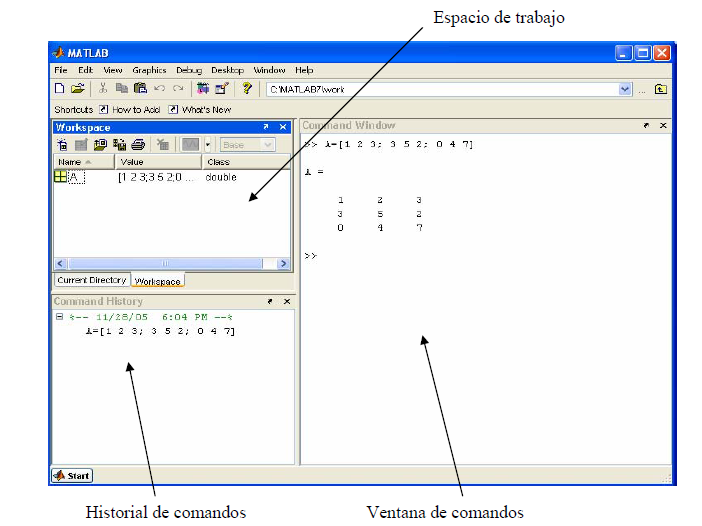
****

Figura 1 Ventana default de MATLAB.

La parte más importante de la ventana inicial es la ventana de comandos (***Command Window***), que aparece en la parte derecha. En esta sub-ventana es donde se ejecutan loscomandos de MATLAB, a continuación del ***prompt*** (aviso) característico (>>), queindica que el programa está preparado para recibir instrucciones. En la pantalla mostradaen la **Figura 1** se ha ejecutado el comando A = [1 2 3; 3 5 2; 0 4 7] mostrándose acontinuación el resultado proporcionado por MATLAB.

En la parte superior izquierda de la pantalla aparece la ventana espacio de trabajo (***Workspace***), contiene información sobre todas las variables que se hayan definido en la sesión.

En la parte inferior derecha aparecen otras dos ventanas: historial de comandos (***Command History***) y directorio actual (***Current Directory***), que se pueden mostrar alternativamente por medio de las pestañas correspondientes. La ventana del historial de comandos muestra los últimos comandos ejecutados en la ventana de comandos. Estos comandos se pueden volver a ejecutar haciendo doble clic sobre ellos. Haciendo clic sobre un comando con el botón derecho del ratón se muestra un menú contextual con las posibilidades disponibles en ese momento. Para editar uno de estos comandos hay que copiarlo antes a la ventana de comandos. Por otra parte, la ventana directorio actual muestra los ficheros del directorio activo o actual. Haciendo doble clic sobre uno de los ficheros ***\*.m*** del directorio activo se abre el ***editor de ficheros*** de MATLAB, herramienta fundamental para la programación que se verá más delante.

**Uso del Help**

MATLAB 7.0 dispone de un excelente ***Help*** con el que se puede encontrar la información que se desee. La **Figura 2** muestra las distintas opciones que aparecen en el menú ***Help*** de la ventana principal de la aplicación:

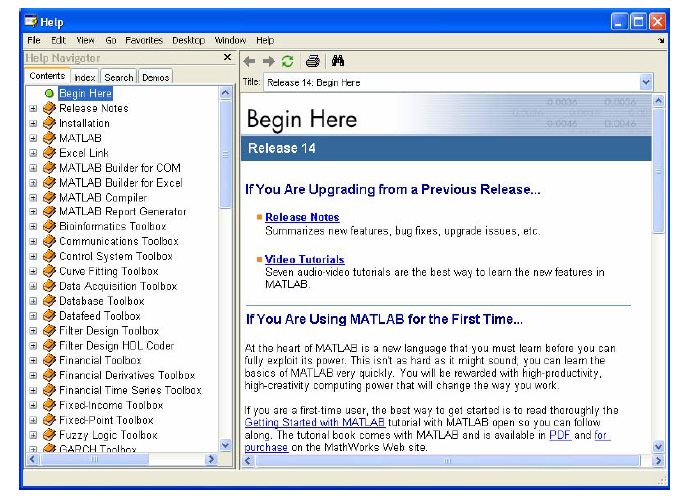
****

Figura 2. Ventana inicial de la ayuda.

**1.** ***Matlab Help***. Se abre la ventana de la **Figura 3**, en la que se puede buscar ayuda general sobre MATLAB o sobre la función o el concepto que se desee. La portada de esta ayuda tiene tres capítulos principales: ***Learning Matlab***, que contiene distintos apartados de introducción al programa; ***Finding Functions and Properties***, que permite acceder a información concreta sobre las distintas funciones o propiedades de los objetos gráficos; y ***Printing the Documentation***, que da acceso a versiones completas e imprimibles de los manuales del programa en formato PDF (Portable Document Format), que precisa del programa ***Adobe Acrobat Reader*** 4.0 o superior.

En la parte izquierda de la ventana, cuando está seleccionada la pestaña ***Contents***, aparece un índice temático estructurado en forma de árbol que puede ser desplegado y recorrido con gran facilidad. Las restantes pestañas de esta ventana dan acceso a un índice por palabras (***Index***), a un formulario de búsqueda (***Search***) y a una sección en la que el usuario puede almacenar enlaces a las páginas que más vaya a utilizar (***Favorites***).

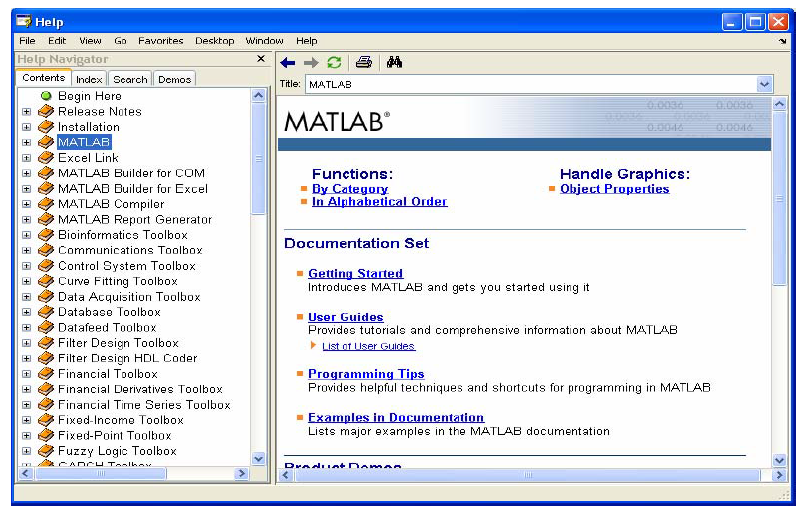
****

Figura 3. Ayuda de MATLAB.

**2.** ***Using the Desktop***. Se abre una ventana de ayuda con un formato similar al de las Figuras anteriores con información detallada sobre cómo utilizar y configurar el entorno de desarrollo. Las distintas herramientas disponibles se describen sucesivamente. Cada página dispone de flechas y enlaces que permiten ir a la página siguiente o volver a la anterior.

**3.** ***Using the Command Window***. Esta opción del menú ***Help*** da acceso a la información necesaria para aprovechar las capacidades de la ***Command Window***, que es el corazón de MATLAB.

**4.** ***Demos***. Se abre una ventana que da acceso a un buen número de ejemplos resueltos con MATLAB, cuyos resultados se presentan gráficamente de diversas formas. Es bastante interesante recorrer estos ejemplos para hacerse idea de las posibilidades del programa. Es asimismo muy instructivo analizar los ficheros ***\*.m*** de los ejemplos que reúnen características similares a las de la aplicación de se desea desarrollar.

Además, se puede también recurrir al ***Help*** desde la línea de comandos de la ventana de comandos. Por ejemplo, obsérvese la respuesta al siguiente uso del comando ***help:***

>> help

>> help and

& Logical AND.

A & B is a matrix whose elements are 1's where both A and B have non-zero elements, and 0's where either has a zero element.

A and B must have the same dimensions unless one is a scalar.

C = AND(A,B) is called for the syntax 'A & B' when A or B is an object.

Overloaded functions or methods (ones with the same name in other directories)

help conboolean/and.m

help conbase/and.m

**Valores especiales**

MATLAB incluye varias constantes predefinidas y valores especiales, que pueden ser usados para hacer programas y se definen en la lista siguiente:

**pi** Representa π

**i, j** Representa el valor

**Inf** Representa infinito, que normalmente ocurre al dividir entre cero. Se imprimirá un mensaje de advertencia cuando se calcule este valor.

**NaN** Representa No-es-un-número, ocurre cuando una operación no está definida, como la división de cero entre cero.

**clock** Representa la hora actual en un vector de fila de seis elementos que contiene año, mes, día, hora, minuto y segundos.

**date** Representa la fecha actual en formato de cadena de caracteres, como 22-jun-2006.

**ans** Representa un valor calculado por una expresión pero no almacenado en un nombre de variable.

**Operaciones con escalares.**

Las operaciones aritméticas entre dos escalares se muestran en la tabla siguiente:

|  |  |  |
| --- | --- | --- |
| **Operación** | **Forma algebraica** | **MATLAB** |
| Suma |  | **a + b** |
| Resta |  | **a - b** |
| Multiplicación |  | **a \* b** |
| División |  | **a / b** |
| Exponenciación |  | **a ^ b** |

**Funciones matemáticas comunes**

Las funciones matemáticas comunes incluyen funciones para calcular el valor absoluto de un valor o la raíz cuadrada de un valor o para redondear un valor. La tabla siguiente muestra una lista de estas funciones con su descripción.

abs (x) Calcula el valor absoluto de .

sqrt (x) Calcula la raíz cuadrada de .

round (x) Redondea al entero más cercano.

fix (x) Redondea (o trunca) al entero más cercano a 0.

floor (x) Redondea al entero más cercano a .

ceil (x) Redondea al entero más cercano a .

sign (x) Devuelve un valor de -1 si es menor que 0, un valor 0 si es igual a 0 y un valor de 1 si es mayor que 0.

exp (x) Calcula , donde es la base de los algoritmos naturales (aproximadamente 2.718282)

log (x) Calcula , el logaritmo natural de con base .

log10 (x) Calcula , el logaritmo común de con base 10.

**Ejercicio.**

Evalúe las expresiones siguientes, y luego verifique sus respuestas introduciendo expresiones en MATLAB.

1. round (-2.6).
2. fix (-2.6).
3. floor (-2.6).
4. ceil (-2.6).
5. sign (-2.6).
6. floor(ceil(10.8))
7. log10(100) + log10(0.001)
8. abs (-5.5)
9. abs(6+j\*7) => sqrt(6**^2+7^2)**

**2. Álgebra lineal y Matrices.**

Una *matriz* en MATLAB es un arreglo rectangular de números de *m* renglones por *n* columnas. Una matriz de 1x1 recibe el nombre de escalar y una matriz de 1 columna por *m* renglones el nombre de *vector*.

Para definir una matriz *no hace falta establecer de antemano su tamaño* (de hecho, se puede definir un tamaño y cambiarlo posteriormente). MATLAB determina el número de renglones y de columnas en función del número de elementos que se proporcionan (o se utilizan). *Las matrices se definen por renglones*; los elementos de un mismo renglón están separados por ***espacios en blanco* o por comas** (,), mientras que las filas están separadas por caracteres ***punto y coma*** (;). Por ejemplo, el siguiente comando define una matriz **A** de dimensión (3x3):

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

La respuesta del programa es la siguiente:

A =

1 2 3

4 5 6

7 8 9

A partir de este momento la matriz **A** 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).

La transpuesta de una matriz es una nueva matriz en la que las filas de la matriz original son las columnas de la nueva. Por ejemplo, una sencilla operación con **A** es hallar su *matriz transpuesta*, . En MATLAB el apóstrofe (') es el símbolo de *transposición* *matricial*. Para calcular **A'** (transpuesta de **A**) basta teclear lo siguiente:

**>> A'**

ans =

1 4 7

2 5 8

3 6 9

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. La variable ***ans*** puede ser utilizada como operando en la siguiente expresión que se introduzca. También podría haberse asignado el resultado a otra matriz llamada **B**:

**>> B=A'**

B =

1 4 7

2 5 8

3 6 9

Ahora ya están definidas las matrices **A** y **B**, y es posible seguir operando con ellas. Por ejemplo, se puede hacer el producto **B\*A** (deberá resultar una matriz simétrica):

**>> B\*A**

ans =

66 78 90

78 93 108

90 108 126

También puede definirse una matriz listando cada fila en una línea aparte, como se muestra a continuación:

>> c= [-1, 0, 0

1, 1, 0

1,-1, 0

0, 0, 2]

c =

-1 0 0

1 1 0

1 -1 0

0 0 2

Si hay demasiados números en una fila de una matriz para que quepan en una línea, se puede continuar la instrucción en la siguiente línea, pero se requieren tres puntos (puntos suspensivos) al final de la línea para indicar que la fila continúa. Por ejemplo, si se quiere definir el vector fila F con 10 valores, se puede usar cualquiera de las instrucciones siguientes:

>> F= [1 52 64 197 42 -42 55 82 22 109]

F=

1 52 64 197 42 -42 55 82 22 109

>> F= [1,52,64,197,42,-42,...

55,82,22,109]

F=

1 52 64 197 42 -42 55 82 22 109

**Operador de dos puntos.**

Es un operador muy potente para crear matrices nuevas. Se usa el operador de dos puntos en una referencia de matriz en lugar de un subíndice específico, el signo de dos puntos representa la fila o columna. Por ejemplo, después de definir una matriz , los siguientes comandos almacenan la primera columna de en , la segunda en , y la tercera en :

>>C = [-1 0 0;1 1 0;1 -1 0; 0 0 2]

C=

-1 0 0

1 1 0

1 -1 0

0 0 2

>> x = C(:,1) >> y = C(:,2) >> z = C(:,3)

x= y= z=

-1 0 0

1 1 0

1 -1 0

0 0 2

Si se usa un signo de dos puntos para separar dos enteros, el operador de dos puntos generará todos los enteros entre los dos enteros especificados. Por ejemplo H=1:8, genera un vector llamado H que contiene los números del 1 al 8.

Si se usan signos de dos puntos para separar tres números, el operador de dos puntos generará valores entre el primer número y el tercero, usando el segundo número como incremento. Por ejemplo, la notación siguiente genera un vector fila llamado tiempo que contiene los números de 0.0 a 5.0 en incrementos de 0.5:

>>tiempo = 0.0:0.5:5.0

El incremento también puede ser negativo, como se muestra en el ejemplo siguiente que genera los números 10, 9, 8,… 0 en el vector fila llamado valores:

>>valores = 10:-1:0;

**Accesando a los elementos de una matriz**

Para referenciar un elemento particular de una matriz, se usa la referencia del renglón y columna del elemento empleando la sintaxis siguiente:

**A (renglón, columna)**

Donde A es una variable que contiene una matriz. Siempre se especifica primero el renglón y luego la columna. Por ejemplo para la matriz,

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

A=

16 2 3 13

5 11 10 8

9 7 6 12

4 14 15 1

Para accesar al elemento del renglón 4 columna 2 se emplea

>>A(4,2)

ans =

14

En MATLAB se accede a los elementos de un vector poniendo el índice entre paréntesis (por ejemplo x(3) ó x(i)). Los elementos de las matrices se acceden poniendo los dos índices entre paréntesis, separados por una coma (por ejemplo A(1,2) ó A(i,j)). Las matrices se *almacenan por columnas* (aunque se *introduzcan por filas*, como se ha dicho antes), y teniendo en cuenta esto puede accederse a cualquier elemento de una matriz con un solo subíndice. Por ejemplo, si **A** es una matriz (3x3) se obtiene el mismo valor escribiendo A(1,2) que escribiendo A(4). A continuación se va a definir una nueva matriz **A** -no singular- en la forma:

**>> A= [1 4 -3; 2 1 5; -2 5 3]**

A =

1 4 -3

2 1 5

-2 5 3

De forma análoga a las matrices, es posible definir un *vector renglón* en la forma siguiente (si los tres números están separados por *blancos* o *comas*, el resultado será un vector fila):

**>> x= [10 20 30]** % vector renglón

x =

10 20 30

MATLAB considera comentarios todo lo que va desde el *carácter tanto por ciento* (%) hasta el final de la línea.

Por el contrario, si los números están separados por *puntos* y *coma* (;) se obtendrá un *vector columna*:

**>> y= [11; 12; 13]** % vector columna

y =

11

12

13

MATLAB tiene en cuenta la diferencia entre vectores renglón y vectores columna. Por ejemplo, si se intenta sumar los vectores ***x*** e ***y*** se obtendrá el siguiente mensaje de error:

**>> x+y**

??? Error using ==> +

Matrix dimensions must agree.

Estas dificultades desaparecen si se suma ***x*** con el vector transpuesto de ***y***:

**>> x+y'**

ans =

21 32 43

**Ejercicio.**

Indique el contenido de las siguientes matrices. Luego verifique sus respuestas introduciendo los comandos en MATLAB.

1. A = G(:,2);
2. C = 10:15;
3. D = [4:9;1:6];
4. F = 0.0:0.1:1.0;
5. T1 = G(4:5,1:3);
6. T2 = G(1:2:5,:);

**Operaciones con matrices**

MATLAB puede operar con matrices por medio de *operadores* y por medio de *funciones*. Se han visto ya los operadores *suma* (+), *producto* (\*) y *transpuesta* (').

Los operadores matriciales de MATLAB son los siguientes:

+ adición o suma

– sustracción o resta

\* multiplicación

' transpuesta

^ potenciación

\ división-izquierda

.\* producto elemento a elemento

./ 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 se obtiene 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. Considérese el siguiente ejemplo:

**>> A= [1 2; 3 4]**

A =

1 2

3 4

**>> A\*2**

ans =

2 4

6 8

**>> A-4**

ans =

-3 -2

-1 0

Los *operadores de división* requieren una cierta explicación adicional. Considérese el siguiente sistema de ecuaciones lineales, **Ax = b** (1) en donde **x** y **b** son vectores columna, y **A** una matriz cuadrada invertible. La resolución de este sistema de ecuaciones se puede escribir en las 2 formas siguientes (¡Atención a la 2ª forma, basada en la *barra* *invertida* (\), que puede resultar un poco extraña!):

**x = inv(A)\*b** (2a)

**x = A\b** (2b)

Así pues, el operador *división-izquierda* por una matriz (barra invertida \) equivale a premultiplicar por la inversa de esa matriz.

La inversa de una matriz en MATLAB se calcula con la función ***inv( )***, por ejemplo defina la matriz **A** en MATLAB como sigue:

**>> A= [1 4 -3; 2 1 5; -2 5 3]**

A =

1 4 -3

2 1 5

-2 5 3

Ahora se emplea la función ***inv( )*** para calcular la inversa de **A** y el resultado se guarda en **B**:

**>> B=inv(A)**

B =

0.1803 0.2213 -0.1885

0.1311 0.0246 0.0902

-0.0984 0.1066 0.0574

Para comprobar que este resultado es correcto basta pre-multiplicar **A** por **B**;

**>> B\*A**

ans =

1.0000 0.0000 0.0000

0.0000 1.0000 0.0000

0.0000 0.0000 1.0000

Con la función ***rank ( )*** se puede calcular el rango de una matriz, que es el número de ecuaciones independientes representadas por las filas de la matriz. Por lo tanto, si el rango de una matriz es igual al número de las filas que tiene, la matriz no es singular y existe su inversa. Considere las siguientes instrucciones que usan la matriz de la explicación anterior:

**>> A = [2 1; 4 3];**

**>> rank (A)**

ans =

2

**>>B = inv(A)**

B=

1.5 -0.5

-2 1

Como el rango de la matriz es igual al número de filas que tiene, sí existe su inversa.

Ahora encuentre el rango de la siguiente matriz y calcule su inversa:

**>> A= [-2 1; -2 1]**

A=

-2 1

-2 1

>> **rank(A)**

ans =

1

**>> inv(A)**

Warning: Matrix is singular to working precision.

ans =

Inf Inf

Inf Inf

La matriz no se puede invertir debido a que el número de ecuaciones es diferente al rango, la matriz es singular.

Para calcular el determinante de una matriz cuadrada se emplea la función ***det( )***. Un determinante es un escalar calculado a partir de los elementos de una matriz cuadrada.

Los determinantes tienen varias aplicaciones en ingeniería, incluido el cálculo de inversas y la resolución de sistemas de ecuaciones simultaneas. Por ejemplo, para calcular el determinante de la matriz **A** () siguiente se emplea la sintaxis que se muestra a continuación:

**>> A = [1 3; -1 5];**

**>> det(A)**

ans =

8

En MATLAB existe también la posibilidad de aplicar ***elemento a elemento*** los operadores matriciales (\*, ^, \ y /). Para ello basta precederlos por un punto (.). Por ejemplo:

**>> [1 2 3 4]^2**

??? Error using ==> ^

Matrix must be square.

Esto es debido a que para elevar al cuadrado un vector se tiene que hacer elemento por elemento como sigue:

**>> [1 2 3 4].^2**

ans =

1 4 9 16

Lo mismo pasa para el producto:

**>> [1 2 3 4]\*[1 -1 1 -1]**

??? Error using ==> \*

Inner matrix dimensions must agree.

**>> [1 2 3 4].\*[1 -1 1 -1]**

ans =

1 -2 3 -4

La diferencia entre usar y no usar el producto (\*) o producto elemento por elemento (.\*) se muestra en el ejemplo siguiente: considérese las matrices **A**= [1 2; 3 4] y **B**= [1 2; 3 4], el resultado de hacer el producto normal entre ellas es:

**>> A\*B**

ans =

7 10

15 22

y el resultado del producto elemento por elemento es:

**>> A.\*B**

ans =

1 4

9 16

**Solución de sistemas de ecuaciones lineales**

Considere el siguiente sistema de tres ecuaciones con tres incógnitas:

Podemos reescribir este sistema de ecuaciones usando las siguientes matrices:

Usando multiplicación de matrices, el sistema de ecuaciones puede escribirse de esta forma:

En MATLAB se calcula esta solución usando el siguiente comando:

**x = inv(A) \* B**

Defina la matriz A y el vector B en MATLAB y resulta el sistema de ecuaciones como sigue:

**>> a**

A=

3 2 -1

-1 3 2

1 -1 -1

**>> B= [10; 5; -1]**

B=

10

5

-1

**>> x=inv(A)\*B**

x=

-2.0000

5.0000

-6.0000

**Ejercicio.**

Use MATLAB para definir las siguientes matrices, y luego calcule las matrices especificadas, si es que existen.

Respuestas:

**1.** -14 62 **2.** 9 -7 6 **3.** 39 **4.** 8 -4 2

7 -9 10 -25 -2 2 -2

18 9 -3 0

1

**5.** Inf Inf Inf **6.** 8 **7.** 0 **8.** -0.2500 0.2500

Inf Inf Inf 0.6250 -0.3750

Inf Inf Inf

1

**Inf- Infinity**, es la representación matemática de la IEEE para Infinito positivo, resultado de dividir por cero o sobreflujo como

**Resuelva los siguientes sistemas de ecuaciones.**

Respuestas:

1

**1.** **2.** **3.**

**4.** **5.**

1

**NaN- Not a Number**, es la representación aritmética de la IEEE para operaciones matemáticas indefinidas como 0/0.

**3. GRÁFICAS.**

Los gráficos 2-D de MATLAB están fundamentalmente orientados a la representación gráfica de vectores (y matrices). En el caso más sencillo los argumentos básicos de la función ***plot*** van a ser vectores. MATLAB utiliza un tipo especial de ventanas para realizar las operaciones gráficas.

Ciertos comandos abren una ventana nueva y otros dibujan sobre la ventana activa, bien sustituyendo lo que hubiera en ella, bien añadiendo nuevos elementos gráficos a un dibujo anterior. Todo esto se verá con más detalle en las siguientes secciones.

**3.1. Funciones gráficas 2D elementales**

MATLAB dispone de cuatro funciones básicas para crear gráficos 2-D. Estas funciones se diferencian principalmente por el *tipo de escala* que utilizan en los ejes de abscisas y de ordenadas.

Estas cuatro funciones son las siguientes:

**plot(x,y)** Genera una gráfica lineal con los valores de e .

**semilogx(x,y)** Genera una gráfica de los valores de e usando una escala logarítmica para y una lineal para .

**semilogy(x,y)** Genera una gráfica de los valores de e usando una escala logarítmica para y una lineal para .

**loglog(x,y)** Genera una gráfica de los valores de e usando escalas logarítmicas tanto para como para .

Existen además otras funciones orientadas a añadir títulos al gráfico, a cada uno de los ejes, a dibujar una cuadrícula auxiliar, a introducir texto, etc. Estas funciones son las siguientes:

title('título') añade un título al dibujo.

xlabel('tal') añade una etiqueta al eje de abscisas.

ylabel('cual') añade una etiqueta al eje de ordenadas.

legend() define rótulos para las distintas líneas o ejes utilizados en la figura. Para más detalle, consultar el ***Help.***

grid activa la inclusión de una cuadrícula en el dibujo. Con ***grid off*** desaparece la cuadrícula.

La función ***plot*** dibuja una nueva figura en la ventana activa (en todo momento MATLAB tiene una ventana activa de entre todas las ventanas gráficas abiertas), o abre una nueva figura si no hay ninguna abierta, sustituyendo cualquier cosa que hubiera dibujada anteriormente en esa ventana. Para verlo, se comenzará creando un par de vectores e con los que trabajar:

>> x= [-10:0.2:10]; y=sin(x);

Ahora se deben ejecutar los comandos siguientes (se comienza cerrando la ventana activa, para que al crear la nueva ventana aparezca en primer plano):

>> close % se cierra la ventana gráfica activa anterior

>> grid % se crea una ventana con una cuadrícula

>> plot(x,y) % se dibuja la función seno borrando la cuadrícula

Se puede observar la diferencia con la secuencia que sigue:

>> close

>> plot(x,y) % se crea una ventana y se dibuja la función seno

>> grid % se añade la cuadrícula sin borrar la función seno

En el primer caso MATLAB ha creado la cuadrícula en una ventana nueva y luego la ha borrado al ejecutar la función ***plot***. En el segundo caso, primero ha dibujado la función y luego ha añadido la cuadrícula. Esto es así porque hay funciones como ***plot*** que por defecto crean una nueva figura, y otras funciones como ***grid*** que se aplican a la ventana activa modificándola, y sólo crean una ventana nueva cuando no existe ninguna ya creada. Más adelante se verá que con la función ***hold*** pueden añadirse gráficos a una figura ya existente respetando su contenido.

**3.1.1. FUNCIÓN *PLOT***

Esta es la función clave de todos los gráficos 2-D en MATLAB. Ya se ha dicho que el elemento básico de los gráficos bidimensionales es el **vector**. Se utilizan también cadenas de 1, 2 ó 3 caracteres para indicar *colores y tipos de línea*. La función ***plot()***, en sus diversas variantes, no hace otra cosa que dibujar vectores. Un ejemplo muy sencillo de esta función, en el que se le pasa un único vector como argumento, es el siguiente:

**>> x= [1 3 2 4 5 3]**

x =

1 3 2 4 5 3

**>> plot(x)**

El resultado de este comando es que se abre una ventana mostrando el gráfico de la Figura 3.1. Por defecto, los distintos puntos del gráfico se unen con una línea continua. También por defecto, el color que se utiliza para la primera línea es el azul. Cuando a la función ***plot()*** se le pasa un único vector –real– como argumento, dicha función dibuja en ordenadas el valor de los **n** elementos del vector frente a los índices 1, 2,... **n** del mismo en abscisas.

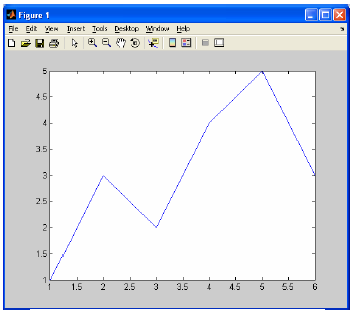


Figura 3.1. Gráfico del vector x= [1 3 2 4 5 3].

En la pantalla de su ordenador se habrá visto que MATLAB utiliza por defecto color blanco para el fondo de la pantalla y otros colores más oscuros para los ejes y las gráficas. Una segunda norma de utilizar la función ***plot()*** es con dos vectores como argumentos. En este caso los elementos del segundo vector se representan en ordenadas frente a los valores del primero, que se representan en abscisas. Para poner los valores del eje x que se deseen se puede hacer de la siguiente manera:

>> x= [1 3 2 4 5 3]

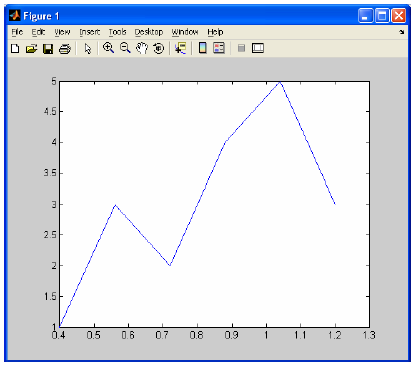
>> xmin=0.4; %valor mínimo de x

>> xmax=1.2; %valor máximo de x

>> n=length(x);

>> L= [xmin:(xmax-xmin)/(n-1):xmax];

>>plot(L,x)

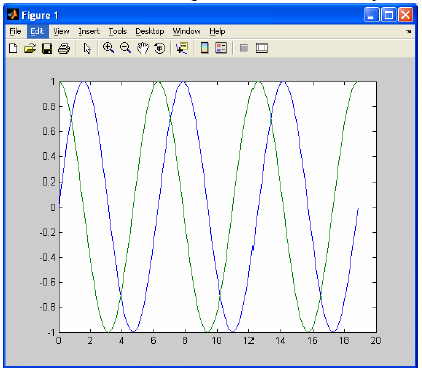


La función ***plot()*** permite también dibujar múltiples curvas introduciendo varias parejas de vectores como argumentos. En este caso, cada uno de los segundos vectores se dibuja en ordenadas como función de los valores del primer vector de la pareja, que se representan en abscisas. Si el usuario no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

>> x=0:pi/25:6\*pi;

>> y=sin(x); z=cos(x);

>> plot(x,y,x,z)



**3.1.2. ESTILOS DE LÍNEA Y MARCADORES EN LA FUNCIÓN *PLOT***

En la sección anterior 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 el caso general esto exige que se pasen como argumentos un par de vectores. En realidad, el conjunto básico de argumentos de esta función es una *tripleta* formada por dos vectores y una cadena de 1, 2 ó 3 caracteres que indica el color y el tipo de línea o de *marker*. En la tabla siguiente se pueden observar las distintas posibilidades.

plot(x,y,'r-.','linewidth',2.5)



Cuando hay que dibujar varias líneas, por defecto se van cogiendo sucesivamente los colores de la tabla comenzando por el azul, hacia arriba, y cuando se terminan se vuelve a empezar otra vez por el azul. Si el fondo es blanco, este color no se utiliza para las líneas.

**3.1.3. AÑADIR LÍNEAS A UN GRÁFICO YA EXISTENTE**

Existe la posibilidad de añadir líneas a un gráfico ya existente, sin destruirlo o sin abrir una nueva ventana. Se utilizan para ello los comandos ***hold on*** y ***hold off***. El primero de ellos hace que los gráficos sucesivos respeten los que ya se han dibujado en la figura (es posible que haya que modificar la escala de los ejes); el comando ***hold off*** deshace el efecto de ***hold on***. El siguiente ejemplo muestra cómo se añaden las gráficas de **x2** y **x3** a la gráfica de **x** previamente creada (cada una con un tipo de línea diferente):

>> x=0:pi/25:6\*pi;

>> y=sin(x);

>> z=cos(x);

>> w=tan(x);

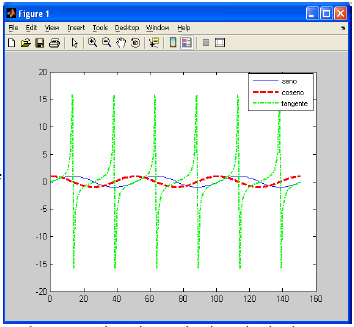
>> hold on

>> plot(x,y,'m:', 'linewidth',3.5)

>> plot(x,z,'r--', 'linewidth',2.5)

>> plot(x,w,'g-.', 'linewidth',1.5)

>> legend('seno','coseno','tangente');

>> hold off

Para definir el rango de valores de los ejes se utiliza el comando axis con la sintaxis siguiente:

axis([xmin,xmax,ymin,ymax])

**3.1.4. COMANDO SUBPLOT**

Una ventana gráfica se puede dividir en **m** particiones horizontales y **n** verticales, con objeto de representar múltiples gráficos en ella. Cada una de estas subventanas tiene sus propios ejes, aunque otras propiedades son comunes a toda la figura. La forma general de este comando es:

subplot(m,n,i)

donde **m** y **n** son el número de subdivisiones en filas y columnas, e **i** es la subdivisión que se convierte en activa. Las subdivisiones se numeran consecutivamente empezando por las de la primera fila, siguiendo por las de la segunda, etc. Por ejemplo, la siguiente secuencia de comandos genera cuatro gráficos en la misma ventana:

>> x=0:pi/25:6\*pi;

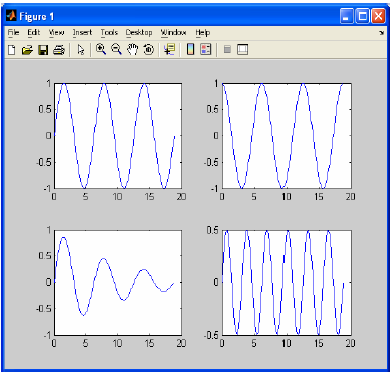
>> y=sin(x); z=cos(x); w=exp(-x\*.1).\*y; v=y.\*z;

>> subplot(2,2,1), plot(x,y)

>> subplot(2,2,2), plot(x,z)

>> subplot(2,2,3), plot(x,w)

>> subplot(2,2,4), plot(x,v)



**3.2. GRÁFICOS TRIDIMENSIONALES**

MATLAB tiene posibilidades de realizar varios tipos de gráficos 3D.

**DIBUJO DE MALLADOS: FUNCIONES *MESHGRID*, *MESH* Y *SURF***

Ahora se verá con detalle cómo se puede dibujar una función de dos variables (z=f(x,y)) sobre un dominio rectangular. Se verá que también se pueden dibujar los elementos de una matriz como función de los dos índices. Sean e dos vectores que contienen las coordenadas en una y otra dirección de la retícula (*grid*) sobre la que se va a dibujar la función. Después hay que crear dos matrices **X** (cuyas filas son copias de ) e **Y** (cuyas columnas son copias de ). Estas matrices se crean con la función ***meshgrid***. Estas matrices representan respectivamente las coordenadas e de todos los puntos de la retícula. La matriz de valores **Z** se calcula a partir de las matrices de coordenadas **X** e **Y**.

Finalmente hay que dibujar esta matriz **Z** con la función ***mesh***, cuyos elementos son función elemento a elemento de los elementos de **X** e **Y**. Véase como ejemplo el dibujo de la función siendo ; para evitar dividir por 0 se suma al denominador el número pequeño (***eps***).

>> close all

>> x=-8:0.5:8;

>> y=x;

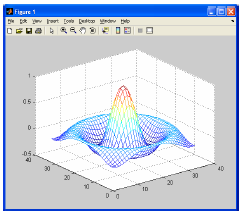
>> [X,Y]=meshgrid(x,y);

>> R=sqrt(X.^2+Y.^2)+eps;

>> W=sin(R)./R;

>> mesh(W)

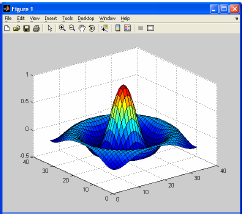
Ejecutando este fichero se obtiene el gráfico mostrado en la Figura.



Se habrá podido comprobar que la función ***mesh*** dibuja *en perspectiva* una función en base a una retícula de líneas de colores, rodeando cuadriláteros del color de fondo, con eliminación de líneas ocultas. Más adelante se verá cómo controlar estos colores que aparecen. Baste decir por ahora que el color depende del valor **z** de la función.

Ejecútese ahora el comando:

>> surf(W)



y obsérvese la diferencia en la Figura. En vez de líneas aparece ahora una superficie *faceteada*, también con eliminación de líneas ocultas. El color de las facetas depende también del valor de la función.

**4. PROGRAMACIÓN DE MATLAB**

MATLAB es una aplicación que se puede programar muy fácilmente. De todas formas, como lenguaje de programación pronto verá que no tiene tantas posibilidades como otros lenguajes (ni tan complicadas...). Se comenzará viendo las bifurcaciones y bucles, y la lectura y escritura interactiva de variables, que son los elementos básicos de cualquier programa de una cierta complejidad.

**4.1 SENTENCIA IF**

En su forma más simple, la sentencia ***if*** se escribe en la forma siguiente (obsérvese que –a diferencia de C/C++/Java– la condición no va entre paréntesis, aunque se pueden poner si se desea):

if condición

sentencias 1

else

sentencias 2

end

Existe también la *bifurcación múltiple*, en la que pueden concatenarse tantas condiciones como se desee, y que tiene la forma:

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

donde 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.

**4.2. SENTENCIA FOR**

La sentencia ***for*** repite un conjunto de sentencias un número predeterminado de veces. La sentencia ***for*** de MATLAB es muy diferente y no tiene la generalidad de la sentencia ***for*** de C/C++/Java. La siguiente construcción ejecuta *sentencias* con valores de **i** de **1** a **n**, variando de uno en uno.

N=10;

for i=1:N

sentencias

end

o bien,

for i=vectorValores

sentencias

end

donde **vectorValores** es un vector con los distintos valores que tomará la variable **i**.

En el siguiente ejemplo se presenta el caso más general para la variable del bucle (*valor\_inicial: incremento: valor\_final*); el bucle se ejecuta por primera vez con **i=n**, y luego **i** se va decrementando de 0.2 en 0.2 hasta que llega a ser menor que 1, en cuyo caso el bucle se termina:

for i=n:-0.2:1

sentencias

end

En el siguiente ejemplo se presenta una estructura correspondiente a dos ***bucles anidados***. La variable **j** es la que varía más rápidamente (por cada valor de **i**, **j** toma todossus posibles valores):

for i=1:m

for j=1:n

sentencias

end

end

Cuando se introducen interactivamente en la línea de comandos, los bucles ***for*** se ejecutan sólo después de introducir la sentencia ***end*** que los completa.

**4.3. SENTENCIA WHILE**

La estructura del bucle ***while*** es muy similar a la de C/C++/Java. Su sintaxis es la siguiente:

while condición

sentencias

end

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

**4.4. FUNCIÓN INPUT**

La función ***input*** permite imprimir un mensaje en la línea de comandos de MATLAB y recuperar como valor de retorno un valor numérico o el resultado de una expresión tecleada por el usuario. Después de imprimir el mensaje, el programa espera que el usuario teclee el valor numérico o la expresión. Cualquier expresión válida de MATLAB es aceptada por este comando. El usuario puede teclear simplemente un vector o una matriz. En cualquier caso, la expresión introducida es evaluada con los valores actuales de las variables de MATLAB y el resultado se devuelve como valor de retorno. Véase un ejemplo de uso de esta función:

>> n = input('Teclee el número de ecuaciones')

Otra posible forma de esta función es la siguiente (obsérvese el parámetro 's'):

>> nombre = input('¿Cómo te llamas?','s')

En este caso el texto tecleado como respuesta se lee y se devuelve sin evaluar, con lo que se almacena en la cadena ***nombre***. Así pues, en este caso, si se teclea una fórmula, se almacena como texto sin evaluarse.

**4.5. FUNCIÓN *fprintf***

Además de exhibir el contenido de alguna variable, también permite controlar el formato con que se usara al exhibir los valores, y también saltos de línea. La forma general de este comando es la siguiente:

fprintf (formato, variable)

El formato contiene el texto y las especificaciones de formato para las salidas, y va seguido de los nombres de las variables a mostrar. Dentro del formato se usan los especificadores %f y %e. Si se usa %e los valores se exhibirán en notación exponencial y con %f en notación de punto fijo o decimal. Un ejemplo sencillo del comando es:

>> temperatura=58;

>> fprintf('la temperatura es %f grados F \n',temperatura);

la temperatura es 58.000000 grados F

Si se modifica el comando así:

>> temperatura=58;

>> fprintf('la temperatura es \n %f grados F \n',temperatura);

la temperatura es

58.000000 grados F

**Archivos \*.m**

Los ficheros con extensión (***.m***) son ficheros de texto sin formato (ficheros ASCII) que constituyen el centro de la programación en MATLAB. Ya se han utilizado en varias ocasiones. Estos ficheros se crean y modifican con un editor de textos cualquiera. En el caso de MATLAB ejecutado en un PC bajo ***Windows***, lo mejor es utilizar su propio editor de textos, que es también ***Debugger***. Existen dos tipos de ficheros ***\*.m***, los ***ficheros de comandos*** (llamados *scripts* en inglés) y las ***funciones***.

Para comprender mejor el ambiente de programación de MATLAB vamos a revisar los ejemplos siguientes:

Teclee las líneas siguientes y grabe el archivo con el nombre de raices.m.

a = 0; fa = -Inf;

b = 3; fb = Inf;

n=1;

while b-a > eps\*b

x = (a+b)/2;

fx = x^3-2\*x-5;

if sign(fx) == sign(fa)

a = x; fa = fx;

else

b = x; fb = fx;

end

n=n+1,

end

x

En el editor de MATLAB teclee las líneas siguientes de comandos y guárdelas con el nombre de ejercicio1.

clear all % limpia el workspace

clc %limpia la linea de comandos

n1= input('Teclea un número: ');

n2= input ('Teclea otro: ');

reply=4;

while (reply~=1)&(reply~=2)&(reply~=3)

fprintf('Selecciona la operación que deseas: \n' );

fprintf('1 Suma \n' );

fprintf('2 Resta \n' );

fprintf('3 Multiplicacion \n' );

reply = input('...');

end

if reply==1 %la suma de los dos números

resultado=n1+n2;

elseif reply==2 %la resta de los dos números

resultado=n1-n2;

elseif reply==3 %la multiplicación de los dos números

resultado=n1\*n2;

end

fprintf('El resultado es = %f \n', resultado);

Las variables se pierden una vez se cierra el programa, para recuperar los valores se puede hacer con el comando save:

save variables <variable> <variable>

para recuperar estas variables se debe utilizar el comando load

load variables

El ejemplo siguiente muestra el procedimiento antes mencionado:

u=-8:0.5:8;

v=u;

[U,V]=meshgrid(u,v);

R=sqrt(U.^2+V.^2)+eps;

W=sin(R)./R;

mesh(W)

save variables W

**Funciones**

En MATLAB se soporta la generación de funciones con las mismas propiedades de otros lenguajes de programación, un ejemplo de función se muestra en el siguiente código:

% Una función sencilla

% un argumento

function y=ud1\_f1(x)

y=x.ˆ2-log(x);

El nombre de la función, para evitar confusiones, debe coincidir con el nombre del archivo .m donde esta función se encuentra. Por tanto, como este primer ejemplo es la función ud1\_f1, debemos guardarla en el archivo ud1\_f1.m. La primera línea comienza con la palabra reservada function lo cual es común a todas las funciones que veremos en el curso.

Lo más interesante es entender a partir de este ejemplo cómo se pasan los argumentos desde la línea de comandos hasta las funciones. Cuando invocamos esta función desde la línea de comandos podríamos hacerlo del siguiente modo, obteniendo la siguiente respuesta:

>> y=ud1\_f1(5)

y = 23.3906

Cuando se pulsa Enter, tecla de retorno de carro, se carga en memoria RAM la función ud1\_f1.m, y se crea espacio en memoria para la variable x. En ese espacio se coloca el valor 5. Se crea espacio también para y. Las variables x e y se llaman variables locales de la función; el adjetivo locales procede de que viven en el espacio de memoria de la función. Una vez hecho esto, el ordenador ejecuta las sentencias de la función (en este caso una sola) de arriba abajo. Durante esta ejecución se deﬁne la variable de salida y, en la cual al ﬁnal de la misma está el resultado, 23.3906. Una vez terminada la ejecución se devuelve el control a la línea de comandos.

Si necesitamos más argumentos de entrada, simplemente los colocamos uno tras otro separados por comas dentro de los paréntesis a la derecha del nombre de la función. Si se van a generar más salidas se agregan a la declaración de la función como un arreglo de variables de salida del modo:

function [u,v]=ud2\_f2(x,y)

u=x+y;

v=y\*x;

También es posible mandar llamar una función previamente generada dentro de otra función.

A continuación se muestra un ejemplo de función:

% ud1\_ftrozos2

% primera función que llama a otra función

% ud1\_ftrozos2(x). Devuelve el valor de la función:

% f(x)=x si x<1

% f(x)=xˆ2-ln(x) si x>=1

function [u,v]=ud1\_ftrozos2(x,y)

if x<1

u=x+y;

v=x\*y;

else

u=ud1\_f1(x);

v=ud1\_f1(y);

end