El MATLAB al vuelo

1. Consideraciones previas 1
2. El entorno de trabajo 2
3. Funciones básicas 3
4. Vectores y matrices 5
5. Operaciones con vectores y matrices. 10
6. Variables lógicas. 12
7. Polinomios 13
8. Gráficas de funciones 14
9. Programación con MATLAB 17
9.1. La condiciones "if" y "switch" 17
9.2. Los bucles for y while 19
9.3. Funciones 20
10. Referencias 21
11. Apéndice A. Principales funciones de MATLAB. 22

1. Consideraciones previas

En lo que sigue, se supone que el usuario teclea lo que aparece en *cursiva* y precedido del símbolo >>, que hace el papel del *prompt* de la máquina (de hecho, lo óptimo sería que el aprendiz de MATLAB reprodujera éstos y parecidos ejemplos por sí mismo). En normal aparecerá la respuesta de MATLAB a la instrucción tecleada. Los comentarios aparecerán precedidos del símbolo %.

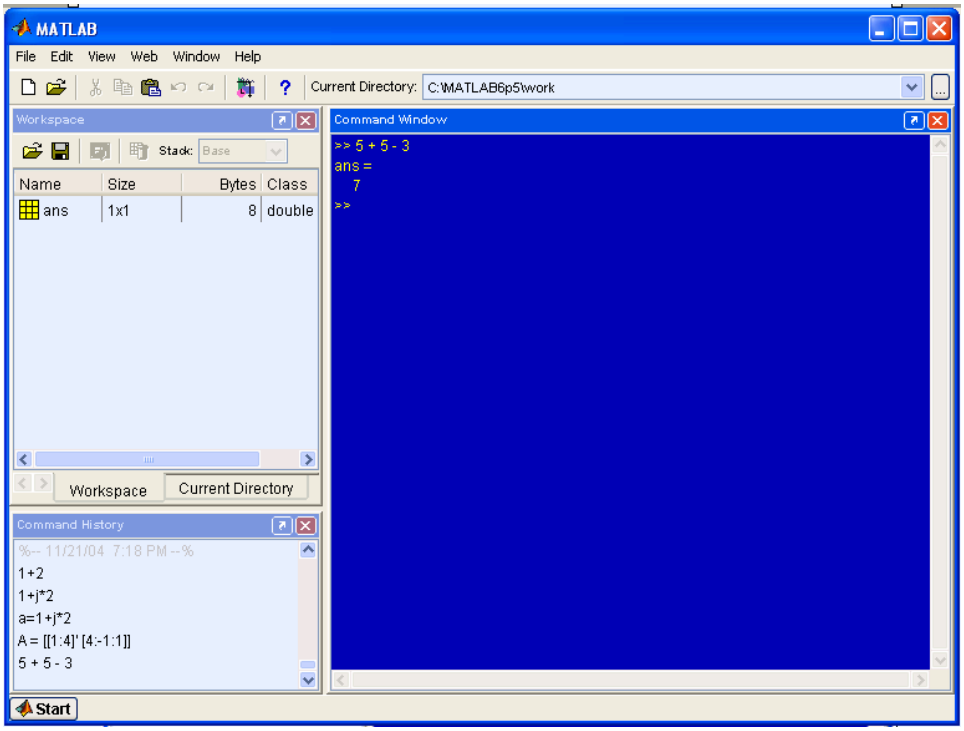
Antes de comenzar, hagamos algunas consideraciones generales:

- MATLAB distingue entre mayúsculas y minúsculas.
- La comilla ' es la que, en un teclado estándar, se encuentra en la tecla de la interrogación.
- Los comentarios deben ir precedidos por % o, lo que es lo mismo, MATLAB ignora todo lo que vaya precedido por el símbolo %.

La ayuda de MATLAB es bastante útil; para acceder a la misma basta teclear help o utilizar el menú del mismo nombre en la ventana de trabajo. Es recomendable usarlo para obtener una información más precisa sobre la sintaxis y diversas posibilidades de uso de los comandos. Al final de cada sección se presenta un EJERCICIO que el aprendiz debería realizar para poner a prueba los conocimientos adquiridos.

2. El entorno de trabajo

Una vez invocamos al MATLAB aparece la ventana de mostrada en la figura inferior.



La parte más importante de la ventana inicial es la **Command Window** o Ventana de Comandos, que aparece en la parte derecha. En esta ventana es donde se ejecutan los comandos de MATLAB, a continuación del *prompt* (aviso) característico "**>>**", que indica que el programa está preparado para recibir instrucciones. En la pantalla mostrada en la figura se ha ejecutado el comando *5+5-3 <enter>*, mostrándose a continuación el resultado proporcionado por MATLAB.

Otro elemento muy importante es la ventana de **Current Directory** o Carpeta de Trabajo donde se nos indica cual es la carpeta actual de trabajo, que es donde se guardarán por defecto nuestras funciones y variables. Para modificar la carpeta de trabajo hay que hacer "clic" sobre el botón  .

Otra ventana a considerar es la denominada "**Workspace**" donde aparecen las variables que estamos utilizando (haciendo "doble clic" sobre el nombre de la variable vemos su valor y tamaño). Por último tenemos la ventana "**Command History**" donde se muestra un histórico de las instrucciones que hayamos introducido en la ventana de comandos (*Command Window*).

EJERCICIO i.

Crear dentro de la carpeta de "Mis Documentos" una carpeta llamada "*LR*" y hacer que ésta sea la carpeta de trabajo del MATLAB.

3. Funciones básicas

Al invocar el MATLAB aparecerá la pantalla de comandos, algunas sugerencias y el símbolo >> , el cual indica donde se ha de realizar la entrada de instrucciones para ser evaluadas.

```
>> Comando o instrucción a evaluar < enter >
```

Para hacer la suma de dos números, escribimos :

```
>>5+5 < enter > % Presionamos la tecla entrar.

ans =

10
```

Los cálculos que no se asignan a una variable en concreto se asignan a la variable de respuesta por defecto que es `ans` (del inglés, *answer*):

```
>>2+3

ans =
5
```

Sin embargo, si el cálculo se asigna a una variable, el resultado queda guardado en ella:

```
>>x=2+3

x =
5
```

Para conocer el valor de una variable, basta teclear su nombre:

```
>>x

x =
5
```

O bien inspeccionarla desde la subventana WORKSPACE.

Si se añade un punto y coma "**;**" al final de la instrucción, la máquina no muestra la respuesta:

```
>>y=5*4;
```

Pero no por ello deja de realizarse el cálculo.

```
>>y
```

```
y =
20
```

Las operaciones se evalúan por orden de prioridad: primero las potencias "^", después las multiplicaciones "*" y divisiones "/" y, finalmente, las sumas "+" y restas "-". La prioridad se puede alterar mediante los paréntesis "()", mientras que las operaciones de igual prioridad se evalúan de izquierda a derecha:

```
>>2/4*3

ans =
1.5000

>>2/(4*3)

ans =
0.1667
```

Se pueden utilizar las funciones matemáticas habituales. Así, por ejemplo, la función coseno,

```
>>cos(pi) % pi es una variable con valor predeterminado 3.14159...

ans =
-1
```

o la función exponencial (e^x)

```
>>exp(1)     % Función exponencial evaluada en 1, es decir, el número e

ans =
2.7183
```

o la función exponente a^y

```
>>2^(1/3)     % Función 2 elevado a 1/3

ans =
1.2599
```

Otro ejemplo de función matemática: la raíz cuadrada; que, como puede verse, trabaja con complejos sin ningún tipo de problema. La unidad imaginaria[1] se representa en MATLAB como `i` o `j`, variables con dicho valor como predeterminado:

```
>>sqrt(-4)
```

[1] La unidad imaginaria por defecto es `i` ó `j`, siempre que no se haya asignado alguna de esta variables a otro valor. Por ejemplo, el MATLAB admite la instrucción `>> j = 5`, con lo que `j` dejaría de ser la unidad imaginaria y pasaría a valer `5`.

```
ans =
0+ 2.0000i
```

De hecho se puede escribir directamente un número complejo como:

```
>>a = 1+j*3

a =
1.0000 + 3.0000i
```

El usuario puede controlar el número de decimales con que aparece en pantalla el valor de las variables (en el menú FILE > PREFERENCES > COMMAND WINDOW), sin olvidar que ello no está relacionado con la precisión con la que se hacen los cálculos, sino con el aspecto con que éstos se muestran.

Para deshacerse de una variable:

```
>>clear y
```

EJERCICIO ii.

Definir dos variables **x** cuyo valor es 2 e **y** que es el número complejo imaginario puro de módulo 5. Realizar la operación:

$$z = x \times y \times \text{sen}(x - y) + x/y \times \left[\cos(x - y)\right]^2$$

Obtener el resultado de la función para distintos valores de x e y probando que ocurre cuando **y** es igual a cero.

Comprobar el valor que tienen **x**, **y** y **z** inspeccionado la ventana WORKSPACE.

4. Vectores y matrices

Para definir un vector fila, basta introducir sus elementos separados por espacios entre los corchetes cuadrados "*[*" y "*]*":

```
>>v=[1 2 3]      % Vector de 3 coordenadas

v=
1 2 3

>>w=[4 5 6];
```

El operador "*'*" es el de transposición (en el caso de número complejos realiza la transposición y la conjugación simultáneas):

```
>>w'

ans =
4
5
6
```

Si queremos declarar un vector de coordenadas equiespaciadas entre dos dadas, por ejemplo, que la primera valga 0, la última 20 y la distancia entre coordenadas sea 2, basta poner:

```
>>vect1=[0:2:20]

vect1 =
0  2  4  6  8  10  12  14  16  18  20
```

Equivalentemente, si lo que conocemos del vector es que la primera coordenada vale 0, la última 20 y que tiene 11 en total, escribiremos:

```
>>vect2=linspace(0,20,11)

vect2 =
0  2  4  6  8  10  12  14  16  18  20
```

A los elementos de un vector se accede sin más que escribir el nombre del vector y, entre paréntesis, su índice:

```
>>vect2(3)

ans =
4
```

Se pueden extraer varios elementos a la vez, constituyéndose así un nuevo vector, por ejemplo:

```
>>vect2(2:5)

ans=
2  4  6  8
```

O, por ejemplo, utilizando el carácter ":" sin números a los lados, quiere decir todos los elementos de esa fila o columna:

```
>>vect1(:)

ans=
0
2
4
6
8
10
12
14
16
18
20
```

Las matrices se escriben como los vectores, pero separando las filas mediante un punto y coma; así una matriz 3x3:

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

M =
1 2 3
4 5 6
7 8 9

>>M'                    % Su transpuesta (su adjunta)

ans =
1 4 7
2 5 8
3 6 9

>>mat=[v;w;0 0 1]       % También es una matriz 3x3

mat =
1 2 3
4 5 6
0 0 1
```

A los elementos de una matriz se accede sin más que escribir el nombre de la matriz y, entre paréntesis, los respectivos índices:

```
>>mat(1,3)     % Elemento en la primera fila y tercera columna de la
matriz mat

ans =
3
```

También se puede acceder a un fila o columna completas, teniendo en cuenta que el carácter dos puntos "`:`" indica números consecutivos entre dos valores, `[2:5]` es equivalente a `[2 3 4 5]`

```
>>mat(:,2)     % Segunda columna de mat

ans =
2
5
0

>>mat(2,:)     % Su segunda fila

ans =
4 5 6
```

Para acceder a la matriz como si fuera una columna,

```
>>M(2:7)       % Los elementos segundo a séptimo de la matriz como
columna
```

```
ans =
4
7
2
5
8
3
```

o acceder a cualquiera de sus submatrices

```
>>mat(2:3,[1 3])   % Submatriz formada por los elementos que están en
      % "todas" las filas que hay entre la segunda y la tercera y
      % en las columnas primera y tercera

ans =
4 6
0 1
```

Existen algunas matrices definidas previamente; por ejemplo, la matriz identidad,

```
>>eye(5)    % eye se pronuncia en inglés como I

ans =
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

la matriz nula,

```
>>zeros(3)

ans =
0 0 0
0 0 0
0 0 0
```

o la matriz cuyos elementos valen todos 1:

```
>>ones(4)

ans =
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
```

Se puede conocer el tamaño de una matriz y la longitud de un vector:

```
>>size(mat) % Dimensiones de la matriz mat (número de filas
           % y de columnas)
```

```
ans =
3 3

>>size(v)

ans =
1 3

>>length(v) % Longitud del vector (número de coordenadas)

ans =
3
```

Existen comandos que permiten crear de forma sencilla matrices. Por ejemplo:

```
>>diag(v)  % Matriz diagonal cuya diagonal es el vector v

ans =
1 0 0
0 2 0
0 0 3

>>diag(diag(M)) % Matriz diagonal con la diagonal de M. La sentencia
       % diag(M) da el vector formado por la diagonal de la matriz M

ans =
1 0 0
0 5 0
0 0 9

>>diag(ones(1,4),1)+diag(ones(1,4),-1) % Matriz tridiagonal 5x5 con 0
              %en la diagonal principal y 1 en la sub y superdiagonal

ans =
0 1 0 0 0
1 0 1 0 0
0 1 0 1 0
0 0 1 0 1
0 0 0 1 0
```

Recuerde: Los paréntesis cuadrados "*[*" y "*]*" son empleados para matrices, mientras que los paréntesis "*(*" y "*)*" se emplean para funciones. El carácter coma "*,*" para separar elementos [1,2,3,4] y el punto y como "*;*" para separar filas en las matrices. [1 2; 3 4]. El carácter dos puntos "*:*" indica números consecutivos entre dos valores, `[2:5]` es equivalente a `[2 3 4 5]`.

EJERCICIO iii.

Crear en una sola instrucción una matriz llamada LR de tamaño 3 x 4, cuyas filas sean:

1ª. Los números del 1 al 7 de dos en dos.

2ª. Los números del 4 al 1.

3ª. Los números del 1, 7, 11 y 9

Emplear los siguientes caracteres especiales: "[", "]", ":", ";" y ","

Crear una matriz, LRb, que sea igual a la anterior excepto en que el elemento de la fila 1 y columna 2 sea igual a cero.

Comprobar el tamaño de la matriz con la instrucción "size" y el valor de la variable en la ventana WORKSPACE.

5. Operaciones con vectores y matrices.

Las funciones matemáticas elementales están definidas de forma que se pueden aplicar sobre matrices. El resultado es la *matriz* formada por la aplicación de la función a cada elemento de la *matriz*. Así:

```
>>log(v)

ans =
0 0.6931 1.0986

>>p=(0:0.1:1)*pi % Vector definido como el producto de un vector por
un escalar

p =
Columns 1 through 7
0 0.3142 0.6283 0.9425 1.2566 1.5708 1.8850
Columns 8 through 11
2.1991 2.5133 2.8274 3.1416

>>x=sin(p)

x =
Columns 1 through 7
0 0.3090 0.5878 0.8090 0.9511 1.0000 0.9511
Columns 8 through 11
0.8090 0.5878 0.3090 0.0000
```

Las operaciones habituales entre *matrices* (suma, resta y producto escalar de vectores; suma, resta, producto y potencia de matrices) se representan con los operadores habituales:

```
>>v,w     % Recordamos los valores de v y w

v =
1 2 3
w =
4 5 6

>>z=v*w'  % Producto escalar (producto de matrices 1x3 por 3x1)

z =
32
```

```
>>Z=w'*v  % Producto de matrices 3x1 por 1x3 = Matriz 3x3

Z =
4  8  12
5  10 15
6  12 18

>>v*w     % Los vectores v y w no se pueden multiplicar

??? Error using ==> *
Inner matrix dimensions must agree.

>>mat     % Recordamos el valor de la matriz mat

mat =
1  2  3
4  5  6
0  0  1

>>mat^2   % Matriz mat elevada al cuadrado

ans =
 9  12 18
24  33 48
 0   0  1
```

También pueden efectuarse multiplicaciones, divisiones y potencias de *matrices*, entendiéndolas como elemento a elemento (como, de hecho, se realizan la suma y la resta). El operador utilizado para ellas es el habitual precedido por un punto; es decir:

```
>>v.*w % Vector formado por los productos de los elementos
       % ans(i)=v(i)*w(i)

ans =
4  10 18

>>w./v  % Vector formado por el cociente de cada coordenada de w
        %entre la coordenada correspondiente de v: ans(i)=w(i)/v(i)

ans =
4.0000  2.5000  2.0000

>>mat.^2 % Matriz cuyos  elementos son los de mat elevados
          % al cuadrado: ans(i,j)=mat(i,j)^2

ans =
 1   4   9
16  25  36
 0   0   1
```

Finalmente, pueden calcularse determinantes:

```
>>det(mat)
```

```
ans =
-3
```

y resolverse sistemas de ecuaciones lineales con el versátil comando \:

```
>>mat\v'

ans =
2.6667
-5.3333
3.000
```

EJERCICIO iv.

Crear una matriz compleja LRc cuya parte real sea la matriz LR del ejemplo anterior y su parte imaginaria los elementos al cuadrado de dicha matriz.

Obtener una matriz LR2 que sea resultado del producto matricial de LRc y la transpuesta-conjugada de LR.

Calcular la matriz inversa de LR2 y llamarle LRi. Comprobar el resultado haciendo el producto matricial de LR2 y LRi.

6. Variables lógicas.

También existen variables lógicas que toman los valores 0 (falso) o 1 (verdadero). Los caracteres más empleados en las condiciones lógicas son:

- "**>**" mayor que ...
- "**<**" menor que ...
- "**==**" igual a ...
- "**>=**" mayor o igual que ...
- "**<=**" menor o igual que ...
- "**~=**" distinto de ...
- "**&&**" condición "AND" ó "Y"
- "**||**" condición "OR" ó "O"

Por ejemplo:

```
>>abs(v)>=2      % Vector lógico cuyas coordenadas valen 1 si la
             %coordenada correspondiente de v es >= 2 y 0 si no lo es

ans =
0 1 1

>>vector=v(abs(v)>=2) % Vector formado por la coordenadas de v que
                      % verifican la desigualdad
```

```
vector =
2 3

>>v2=[3 2 1]

v2 =
3 2 1

>>logica=v==v2    % Asignación de un valor lógico (el doble signo
                  %igual es el igual lógico)

logica =
0 1 0

>>logic2=v~=v2    % Distinto (~ es el operador de negación)

logic2 =
1 0 1
```

EJERCICIO v.

Obtener el número de elementos de LR mayores que tres y menores que diez empleando para ellos las operaciones lógicas "<" y ">" además del comando "sum".

(Consultar la ayuda Help para obtener el funcionamiento del comando sum)

7. Polinomios

Se puede trabajar con polinomios: basta tener en cuenta que un polinomio no es más que un vector. El orden de los coeficientes es de mayor a menor grado, por ejemplo:

```
>>p=[1 0 2 0 3]        % Polinomio x^4+2*x^2+3

p =
1 0 2 0 3

>>q=[2 1 0]            % Polinomio 2*x^2+x

q =
2 1 0
```

MATLAB tiene funciones específicas para polinomios como:

```
>>polyval(p,-1)  % Evaluación del polinomio x^4+2x^2+3 en x=-1

ans =
6

>>pro=conv(p,q)  % Producto de los polinomios p y q
```

```
pro =
2 1 4 2 6 3 0

>>deconv(pro,p)  % Cociente entre pro y p obviamente el resultado es q

ans =
2 1 0

>>roots(pro)     % Raíces del polinomio pro

ans =
     0
 0.6050+1.1688i
 0.6050-1.1688i
-0.6050+1.1688i
-0.6050-1.1688i
-0.5000

>>poly([i -i 1/2 pi])  % Polinomio mónico que tiene por raíces a los
                      % números i, -i, 0.5 y pi

ans =
1.0000 -3.6416 2.5708 -3.6416 1.5708
```

EJERCICIO vi.

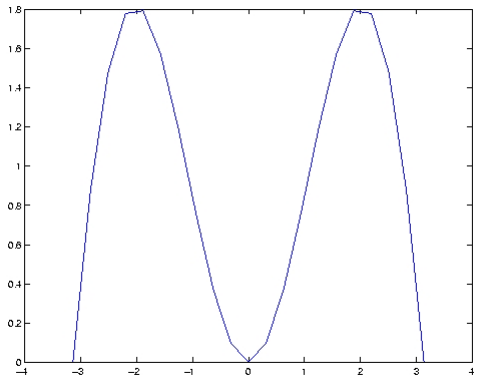
Crear un polinomio "p" de forma que sus coeficientes sean la segunda fila de la matriz LR (utilizar los caracteres "=" y ":").

Calcular las raíces de dicho polinomio.

8. Gráficas de funciones

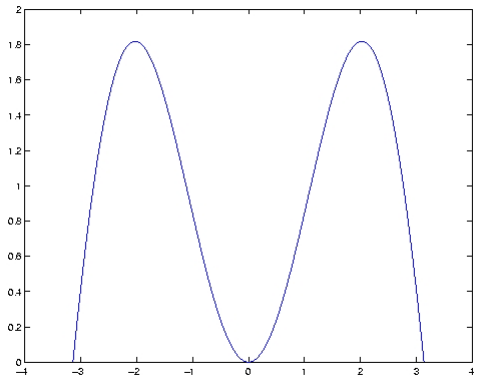
MATLAB tiene un gran potencial de herramientas gráficas. Se pueden dibujar los valores de un vector frente a otro (de la misma longitud):

```
>>x=pi*(-1:0.1:1);
>>y=x.*sin(x);
>>plot(x,y)        % Por defecto une los puntos (x(i),y(i)) mediante
                   % una poligonal
```

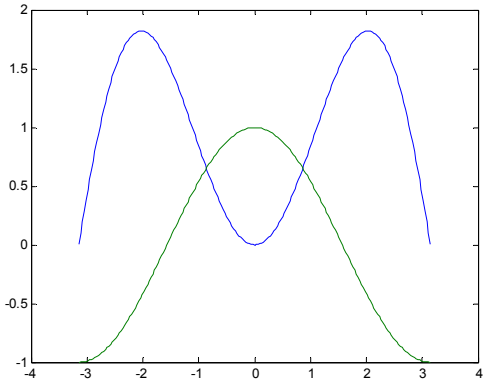


Como se ve, con pocos puntos la gráfica tiene un aspecto demasiado lineal a trozos. Para "engañar" al ojo, basta tomar más puntos.

```
>>x=pi*(-1:0.01:1);
>>y=x.*sin(x);
>>plot(x,y)
```



```
>>hold on      % Mantiene en la ventana gráfica los dibujos anteriores
>>plot(x,cos(x))  % Dibuja sobre la gráfica anterior la función cos(x)
```



```
>>hold off                    % Con esto olvida los dibujos anteriores
                             % y dibuja en una ventana nueva
```

El mismo gráfico de la figura superior puede obtenerse con una única orden, dibujándose cada curva de un color distinto:

```
>>plot(x,cos(x),x,y)
```

Además se puede especificar el color de cada curva, el tipo de línea y el marcador, según la tabla:

COLOR	*TIPO DE LÍNEA*	*MARCADOR*
'r' rojo	'-' línea continua	'+' signo más
'k' negro	'--' línea discontinua	'-' signo menos
'w' blanco	':' línea a puntos	'o' círculo
'b' azul	'-.' línea a puntos y rayas	's' cuadrado
'y' amarillo		'd' diamante
'g' verde		'p' estrella de 5 puntas
'm' magenta		'h' estrella de 6 puntas
'c' azul celeste		'*' asterisco

Otros comandos muy útiles a la hora de realizar gráficas son:

`title`: le ponemos un título a la gráfica

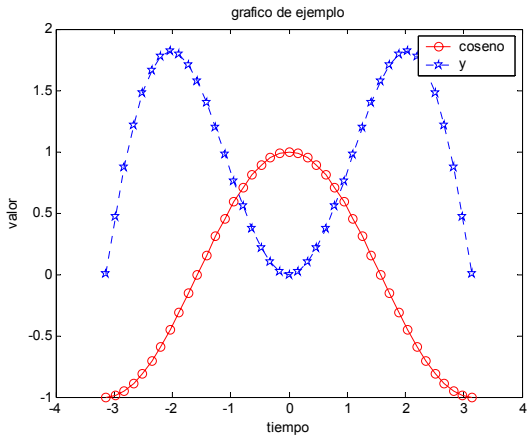
`xlabel`: le ponemos una etiqueta al eje x

`ylabel`: le ponemos una etiqueta al eje y

`legend`: incluimos una legenda en la gráfica

Por ejemplo

```
>> plot(x,cos(x),'ro-',x,y,'bp:')
>> xlabel('tiempo')
>> ylabel('valor')
>> legend('coseno','y')
>> title('grafico de ejemplo')
```



También se pueden realizar gráficas logarítmicas con las instrucciones: `semilogx` (eje x logarítmico), `semilogy` (eje x logarítmico) y `loglog` (ejes x e y logarítmicos). Además de gráficas en 3D con las instrucciones: `mesh`, `surf`, `plot3`...

EJERCICIO vii.

Representar simultáneamente dos funciones sinusoidales de frecuencia 50 Hz desfasadas 90 º ($\pi/2$ rad) de amplitud 10 y 20, respectivamente. Siendo la primera de ellas en color verde a puntos y la segunda en azul continua y con asteriscos. Incorporar además:

nombre eje x: "tiempo" nombre eje y: "señal"

título gráfica: "onda seno" legenda: "onda1" y "onda 2"

9. Programación con MATLAB

Para escribir un programa o fichero de comandos con MATLAB habrá que crear un fichero que tenga extensión `.m` y contenga las instrucciones. Esto se puede hacer con cualquier editor de textos, pero tiene algunas ventajas usar el editor propio de MATLAB llamándolo desde la barra de herramientas con los menús **FILE > NEW > M-FILE**. Una vez estamos en el editor de funciones se puede ejecutar el código escrito con el menú **DEBUG>RUN** o pulsando la tecla **F5,** o bien **escribiendo el la Command Windows el nombre de la función**.

MATLAB trabaja con memoria dinámica, por lo que no es necesario declarar las variables que se van a usar. Por esta misma razón, habrá que tener especial cuidado y cerciorarse de que entre las variables del espacio de trabajo no hay ninguna que se llame igual que las de nuestro programa (proveniente, por ejemplo, de un programa previamente ejecutado en la misma sesión), porque esto podría provocar conflictos.

Un programa escrito en MATLAB admite la mayoría de las estructuras de programación al uso y su sintaxis es bastante estándar. En los siguientes apartados se muestra la sintaxis de algunas de estas estructuras (`if`, `for`, `while`,...).

9.1. La condiciones "if" y "switch"

IF

Una de las operaciones más habituales en programación es la ejecución de unos determinados comandos si se cumple una determinada condición lógica, es lo que se conoce como bifurcaciones. Para ello se emplea la orden "`if...end`" de la siguiente forma:

```
if condición
    comandos
end
```

Si la bifurcación es múltiple entonces podemos combinar con los comandos `else` y `elseif`

```
if condicion
    comandos
elseif condicion
    comandos
elseif condicion
    comandos
else
    comandos
end
```

Por ejemplo, crear el fichero "prueba_condicion.m" y ejecutar el siguiente código:

```
A = 1; B = 2;
if A + B == 3
   disp('el resultado es 3')
elseif A+B > 5
      disp('el resultado es mayor que 5 ')
else
      disp('el resultado ni es 3 ni mayor que 5 ')
end
```

(la orden "`disp`" se utiliza para visualizar texto en la orden de comandos)

Para ejecutar la función realizada se pulsa F5 se estamos en el editor de funciones, o se escribe su nombre en la Command Window:

```
>>prueba_condicion

el resultado es 3
```

También se puede llamar la función recién descrita desde otra función sin más que escribir su nombre.

EJERCICIO viii.

Probar el código de ejemplo mostrado arriba para distintos valores de A y B.

SWITCH

La sentencia `switch` realiza una función análoga a un conjunto de if...elseif concatenados. Su forma general es la siguiente:

```
switch expresion
      case case_expr1
            comandos
      case case_expr2
            comandos
      ...
      otherwise % opción por defecto
            comandos
end
```

Por ejemplo si queremos realizar acciones distintas en función del valor de una variable entera podemos escribir:

```
A = 1; B = 2;
switch A + B
case 3
   disp('el resultado es 3')
case 5
      disp('el resultado es 5 ')
otherwise
      disp('el resultado ni es tres ni cinco')
end
```

EJERCICIO ix.

Probar el código de ejemplo mostrado arriba para distintos valores de A y B.

9.2. Los bucles for y while

FOR...END

Cuando queremos ejecutar un conjunto de comandos un número predeterminado de veces utilizamos la función "`for...end`", cuya forma general es:

```
for contador = [vector de valores]
    comandos
end
```

A modo de ejemplo si queremos sumar los cuadrados de los números enteros impares del 1 al 11 podemos escribir:

```
suma = 0
for k = 1:2:11
    suma = suma + k^2;
end
suma
```

Aunque produciría el mismo resultado la instrucción:

```
>> suma = sum([1:2:11].^2)
```

EJERCICIO x.

Crear un fichero llamado "primos.m" (FILE>NEW>M-FILE) para calcular la suma de los 100 primeros términos de la sucesión 1, 2x, $3x^2$, $4x^3$, ...

WHILE...END

A diferencia del comando `for`, el comando `while` se utiliza cuando el número de veces que se ejecuta un determinado conjunto de sentencias depende de una determinada condición lógica. Su utilización genérica es la que sigue:

```
while condicion
    comandos
end
```

Por ejemplo si queremos sumar el cuadrado de números pares mientras no se supere un determinado valor, por ejemplo, 500, entonces:

```
suma = 0; num = 2;
while suma < 500
    suma = suma + num^2;
    num = num + 2;
end
suma
```

Una instrucción muy útil es el comando "`break`" que se utiliza para salir de un bucle. Por ejemplo, si desea salir del bucle en el caso de que alguna suma parcial sea exactamente igual a 100, entonces:

```
suma = 0; num = 2;
while suma < 500
    suma = suma + num^2;
    num = num + 2;
    if suma == 100
        break
    end
end
suma
```

EJERCICIO xi.

Dada la función $y = e^{-x}$ calcular el valor x que haga que se cumpla la igualdad $x = e^{-x}$ con un error de 10^{-6}. Tomar como valor inicial 0.5. Poner una condición que limite a 10 el número de iteraciones para llegar a la solución (utilizar `break`).

Además, probar el programa realizado suponiendo como condición inicial x=10.

9.3. Funciones

Por último, también pueden programarse funciones, esto es, programas que devuelven una variable o conjunto de variables. La primera instrucción de un fichero que contenga una función de nombre `mifuncion` debe ser:

```
function [argumentos de salida]=mifuncion(argumentos de entrada)
```

Es conveniente que el fichero que contenga la función se llame como ella; así, la función anterior debería guardarse en el fichero **mifuncion.m**; por ejemplo, si se desea programar una función que calcule, mediante el algoritmo de Euclides, el máximo común divisor de dos números naturales, basta escribir un fichero euclides.m cuyo contenido sea:

```
function m=euclides(a,b)
% Cálculo del máximo común divisor de dos números naturales
% mediante el algoritmo de Euclides
if a<b
   c=b;
```

```
    b=a;
    a=c;
end
while b>0
    c=rem(a,b);
    a=b;
    b=c;
end
m=a;
```

Si, una vez escrito el fichero anterior, en el espacio de trabajo o en un programa se escribe la instrucción

```
mcd=euclides(33,121)
```

en la variable `mcd` se almacenará el valor `11`.

Las variables de una función son siempre locales. Por tanto, aunque en el seno de la función se modifiquen los argumentos de entrada, el valor de las variables correspondientes queda inalterado. Por ejemplo, en la función euclides.m se modifica el valor de los argumentos de entrada, pero, sin embargo:

```
>>x=15;
>>mcd=euclides(x,3);
>>x
x =
15
```

Si se pretende que las modificaciones de un argumento de entrada afecten a la variable correspondiente, deberá situarse dicho argumento, además, en la lista de argumentos de salida.

EJERCICIO xii.

Crear una función llamada "miexp" que dado un valor de x devuelva como los dos resultados de las ecuaciones: $y_1 = \sqrt{x} + e^{-x}$ e $y_2 = e^{-x}$.

Utilizar dicha función para resolver la ecuación no lineal $x = \sqrt{x} + e^{-x}$ mediante Newton-Raphson con un error de 10^{-6},y limitando el número de iteraciones a 10.

10. Referencias

- "Tutorial de MATLAB", Ender José López Méndez, (web: http://www.monografias.com/trabajos13/tumatlab/tumatlab.shtml)
- "Aprenda Matlab 6.1 como si estuviera en primero" Javier García de Jalón, José Ignacio Rodríguez y Alfonso Brazález, Escuela Técnica Superior de Ingenieros Industriales, Universidad Politécnica de Madrid, 2001 (web: http://mat21.etsii.upm.es/ayudainf/aprendainf/varios.htm)
- "Introducción a MATLAB", Juan-Antonio Infante y José María Rey, (web: http://www.mat.ucm.es/~infante/matlab/notas.htm)

- Ayuda del MATLAB
- "MATLAB edición de estudiante", The MathWorks Inc, Prentice Hall

11. Apéndice A. Principales funciones de MATLAB.

Trigonometría

acos Inversa del coseno	**acosh** Inversa del coseno hiperbólico
acot Inversa de la cotangente	**acoth** Inversa de la cotangente hiperbólica
acsc Inversa de la cosecante	**acsch** Inversa de la cosecante hiperbólica
asec Inversa de la secante	**asech** Inversa de la secante hiperbólica
asin Inversa del seno	**asinh** Inversa del seno hiperbólico
atan Inversa de la tangente	**atanh** Inversa de la tangente hiperbólica
atan2 Inversa de la tangente en 4 cuadrantes	**cos** Cosino
cosh Cosino hiperbólico	**cot** Cotangente
coth cotangente hiperbólica	**csc** Cosecante
csch Cosecante hiperbólica	**sec** Secante
sech Secante hiperbólica	**sin** Seno
sinh Seno hiperbólico	**tan** Tangente
tanh Tangente hiperbólica	

Exponencial

exp Exponencial	**log** Logaritmo natural
log2 Logaritmo en base 2	**log10** Logaritmo en base 10
nextpow2 Potencia de 2 mayor que un número dado	**pow2** Potencia en base 2
reallog Logaritmo natural para matrices reales no-negativas	**realpow** Potencia de matrices reales
realsqrt Raíz cuadrada de matrices no-negativas	**sqrt** Raíz cuadrada

Complejos

abs Valor absoluto	**angle** Ángulo de fase
complex Construimos un número complejo a partir de su parte s real e imaginaria	**conj** Conjugado de un número complejo
cplxpair Ordena número en complejos conjugado	**i** Unidad imaginaria [sqrt(-1)]
imag Parte imaginaria de un complejo	**isreal** Devuelve 1 si la entrada es un número real
j Unidad imaginaria [sqrt(-1)]	**real** Parte real de un complejo
unwrap Corrige vectores con fases para mejorar su representación	

Redondeo y resto

fix Redondea hacia cero	**floor** Redondea hacia menos inifinito
ceil Redondea hacia más inifinito	**round** Redondea hacia el entero más cercano
mod Módulo de un múmero con respecto a otro	**rem** Resto de una división
sign Signo	

Matemática Discreta

factor Factores primos	**factorial** Factorial
gcd Máximo común divisor	**isprime** Devuelve 1 si dos números son primos
lcm Mínimo común múltiplo	**nchoosek** Combinatoria
perms Permutaciones	**primes** Genera una lista de números primos
rat, rats Aproximación a fracciones racionales	

Operadores

+ Suma	**-** Resta
***** Multiplicación	**^** Potencia
**** División	**/** División por la derecha
' Transpuesta y conjugada	**.'** Transpuesta no conjugada
.* Multiplicación matricial elemento a elemento	**.^** Potencia matricial elemento a elemento
. División elemento a elemento	**./** División por la derecha elemento a elemento