Tutorial de Matlab.

Antonio Souto Iglesias.

Presentamos un tutorial de Matlab porque en Cálculo Numérico es importante poder realizar rápidamente y con exactitud los cálculos elementales que son necesarios para los diferentes ejercicios. Matlab es una herramienta potentísima, casi estándar para cálculos en muchas ramas de la Ingeniería, y de uso razonablemente simple. Haremos una descripción de los elementos básicos de Matlab y remitimos al estudiante interesado a cualquier edición del manual de referencia del programa[2] para un mejor conocimiento del mismo. También es interesante el libro sobre Matlab de Higham y Higham[1] y una buena referencia en castellano con ejemplos procedentes de problemas en Cálculo Numérico es el de Quintela[3].

1 Conceptos básicos.

Para arrancar Matlab, se procede como con cuaquier programa Windows, o sea, Inicio, Programas, Matlab o Student Matlab caso de que utilicemos la versión educacional. Una vez arrancado aparece el cursor con el símbolo (>>) o (EDU >>), indicando que puedes introducir órdenes. De hecho, en este tutorial, cuando veas este símbolo, es que tienes que introducir por teclado la orden que aparece escrita a la derecha del mismo.

La utilización más básica de Matlab es como calculadora [1]. Así por ejemplo, puedes calcular $\cos(5) \cdot 2^{7.3}$, para lo cual debes introducir[2]:

```
>>cos(5)*2^7.3
ans =
   44.7013
```

Matlab mantiene en memoria el último resultado. Caso de que ese cálculo no se asigne a ninguna variable, lo hace a una variable por defecto de nombre *ans*. Si quieres referirte a ese resultado, hazlo a través de la variable *ans*, y si no asignas ese nuevo cálculo a ninguna variable, volverá a ser asignado a *ans*.

```
>>log(ans)
ans =
    3.8000
```

En este momento os podríais preguntar si este ha sido un logaritmo decimal o neperiano (natural). Para saberlo, debéis pedir ayuda sobre el comando *log* utilizando:

```
>>help log
 LOG    Natural logarithm.
    LOG(X) is the natural logarithm of the elements of X.
    Complex results are produced if X is not positive.

    See also LOG2, LOG10, EXP, LOGM.
```

[1] Funcionando de este modo, es similar a una calculadora programable, aunque bastante más versátil.

[2] Los argumentos de las funciones trigonométricas siempre están en radianes.

Por defecto, los resultados aparecen con 4 cifras decimales. Si necesitares más precisión en los resultados, puedes utilizar la orden *format long* y repite los cálculos:

```
>>format long
```

Para recuperar una orden y ejecutarla otra vez o modificarla se usan la flechas arriba y abajo del cursor ⇑, ⇓. Presionemos ⇑ hasta recuperar la orden:

```
>>cos(5)*2^7.3
ans =
  44.70132670851334
```

Ejercicio 1.1 *Realizar la siguiente operación:* $2.7^{2.1} + \log_{10} 108.2$.

Ejercicio 1.2 *Realizar la siguiente operación:* $e^{2.7^{2.1}+\log_{10} 108.2}$.

Caso de que necesitéis referiros a determinados cálculos podéis asignarlos a variables y así recuperarlos después mediante esas variables. Por ejemplo, podéis con ⇑ recuperar la orden $\cos(5) \cdot 2^{7.3}$ y asignar su valor a la variable x. Luego podéis utilizarla para otros cálculos.

```
>>x=cos(5)*2^7.3
x =
  44.70132670851334
>>y=log(x)
y =
   3.80000318145901
```

Ejercicio 1.3 *Realizar la siguiente operación:* $2.7^{2.1}+\log_{10} 108.2$ *y asignarla a la variable* x.

Ejercicio 1.4 *Realizar la siguiente operación:* $e^{2.7^{2.1}+\log_{10} 108.2}$ *y asignarla a la variable* t.

Como es muy fácil recuperar órdenes previas podemos utilizar esta idea para simular los términos de una sucesión recurrente. Por ejemplo $x_{n+1} = cos(x_n)$

```
>>x=0.2
x =
   0.20000000000000
>>x=cos(x)
x =
   0.98006657784124
>>x=cos(x)
x =
   0.55696725280964
>>x=cos(x)
x =
```

```
   0.84886216565827
>>x=cos(x)
x =
   0.66083755111662
>>x=cos(x)
x =
   0.78947843776687
>>x=cos(x)
x =
   0.70421571334199
```

Ejercicio 1.5 *Repetir la operación anterior hasta que se estabilice el cuarto decimal de x de un paso al siguiente.*

Ejercicio 1.6 *Cambiar el formato para que otra vez se vean sólo cuatro decimales.*

Ejercicio 1.7 *Empezando por $x = 100$ repetir la operación $x = x - (x^2 - 81)/2/x$ hasta que se converja en el cuarto decimal. ¿Qué relación hay entre el último x y 81?*

Ejercicio 1.8 *Definir A como vuestro DNI. Empezando por $x = 100$ repetir la operación $x = x - (x^2 - A)/2/x$ hasta que se converja en el cuarto decimal. ¿A qué ha convergido la sucesión?*

A veces es bueno apagar y encender la "calculadora"para borrar todo y empezar de nuevo. Esto se hace con la orden *clear*. Hay que tener cuidado al utilizarla pues no solicita confirmación, y sus resultados son definitivos.

```
>>clear
>>x
??? Undefined function or variable 'x'.
```

Ejercicio 1.9 *Preguntar el valor de A igual que acabamos de preguntar x. ¿Tiene sentido el resultado?*

2 Manejo de vectores.

Para crear y almacenar en memoria un vector v que tenga como componentes $v_1 = 0$, $v_2 = 2$, $v_3 = 4$, $v_4 = 6$ y $v_5 = 8$ podemos hacerlo componente a componente:

```
>>v(1)=0
v =
     0
>>v(2)=2
v =
     0     2
>>v(3)=4
```

```
v =
     0     2     4
>>v(4)=6
v =
     0     2     4     6
>>v(5)=8
v =
     0     2     4     6     8
```

O, para simplificar la creación de vectores, se puede definir un vector especificando su primer elemento, un incremento, y el último elemento. Matlab rellenará ese vector. Así podemos definir igualmente el vector v como una secuencia que empieza en 0, avanza de 2 en 2 y que termina en el 8:

```
>> v = [0:2:8]
v =
     0     2     4     6     8
>> v
v =
     0     2     4     6     8
```

Si ponemos ; al final de una línea de comandos, cuando pulsemos la tecla Enter para ejecutarla, se ejecutará pero no mostrará el resultado. Esto es muy útil algunas veces:

```
>> v = [0:2:8];
>> v
v =
     0     2     4     6     8
```

Podemos construir el vector v editando directamente entre los corchetes las componentes del vector v:

```
>>v = [0 2 4 6 8];
>> v
v =
     0     2     4     6     8
```

Puedes acceder fácilmente al contenido de una posición del vector, por ejemplo la primera.

```
>> v(1)
ans =
     0
```

O modificarla:

```
>> v(1)=-3;
>> v
v =
    -3     2     4     6     8
```

O hacer operaciones entre componentes, $v_2 \cdot v_5^3$:

```
>> v(2)*v(5)^3
ans =
        1024
```

Ejercicio 2.1 *Calcular la suma de los elementos de v, elemento a elemento.*

Para trasponer un vector una matriz se usa el apóstrofo que es el acento que está en la misma tecla que el signo de interrogación "?".

```
>> v'
ans =
     -3
     2
     4
     6
     8
```

Como hemos comentado, para recuperar una orden y ejecutarla otra vez o modificarla se usan la flechas arriba y abajo del cursor ⇑, ⇓. Presionemos ⇑ hasta recuperar la orden:

```
>> v(1)=-3;
```

Modifiquémosla para dejar el valor original

```
>> v(1)=0;
```

Al definir ese vector v de 5 componentes, en realidad lo que definimos es una matriz fila de cinco columnas, o sea un matriz de 1x5. Esto lo podemos comprobar si preguntamos el tamaño de v utilizando la sentencia *size*:

```
>>size(v)
ans =
     1     5
```

que nos indica que v tiene una fila y 5 columnas.

Ejercicio 2.2 *Definir un nuevo vector que sea el traspuesto de v y aplicar a ese vector el comando size. ¿Es coherente el resultado?*

3 Introducción al tratamiento de matrices.

Haremos una introducción a la definición y manipulación de matrices. Se supone que has seguido la sección anterior y dispones de los conocimientos básicos sobre la definición y manipulación de vectores usando Matlab. Definir una matriz es muy similar a la definición de un vector. Para definir una matriz, puedes hacerlo dando sus filas separadas por un punto y coma (tener cuidado de poner los espacios en blanco!):

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

o definirla directamente fila a fila[3]:

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

Se puede modificar alguno de los elementos de la matriz A, accediendo a cualquiera de sus posiciones, por ejemplo:

```
>> A(2,2)=-9
A =
     1     2     3
     3    -9     5
     6     7     8
```

Dejemos su valor original:

```
>> A(2,2)=4;
```

De igual modo, puedes considerarla como una fila de vectores columna:

```
>> B = [ [1 2 3]' [2 4 7]' [3 5 8]']
B =
     1     2     3
     2     4     5
     3     7     8
```

(Otra vez, es importante colocar los espacios en blanco.)

Ejercicio 3.1 *Sumar los elementos diagonales de la matriz A, refiriéndonos a ellos, elemento a elemento.*

Podemos sumar o restar matrices para tener otras matrices.

```
>>C=A+B
C =
     2     4     6
     5     8    10
     9    14    16
```

[3]Esto es lo que yo suelo hacer.

Ejercicio 3.2 *Definir la matriz* $D = 2B - A$.

También podemos multiplicarlas.

```
>>C=A*B
C =
    14    31    37
    26    57    69
    44    96   117
```

Ejercicio 3.3 *Definir la matriz* $D = B - A \cdot B$.

Ejercicio 3.4 *Definir la matriz* $C = AA^t$.

Podemos definir algunos tipos especiales de matrices, como por ejemplo una matriz de 3x3 que tenga todos sus elementos nulos.

```
>>I=zeros(3)
I =
     0     0     0
     0     0     0
     0     0     0
```

Podemos modificar sus elementos diagonales para tener la matriz identidad, para lo cual debes usar ⇑ para modificar la definición del primer elemento, cambiando los índices para definir los demás.

```
>>I(1,1)=1;
>>I(2,2)=1;
>>I(3,3)=1
I =
     1     0     0
     0     1     0
     0     0     1
```

Ejercicio 3.5 *Definir la matriz* $D = B - A \cdot B$ *como* $D = B(I - A)$.

Otra forma de definir la matriz identidad es a través de la función *diag*, que recibe un vector que convierte en diagonal de una matriz cuyos otros elementos son nulos.

```
>>J=diag([1 1 1])
J =
     1     0     0
     0     1     0
     0     0     1
```

Ejercicio 3.6 *Definir una matriz* D *diagonal cuyos elementos sean* -2, 1, 0.2 *y* -0.7.

Ejercicio 3.7 *Pedir ayuda de la función eye, y definir la matriz diagonal de 10x10.*

3.1 Definición de submatrices.

Se pueden definir "subvectores" o submatrices muy facilmente. Si definimos v como:

```
>> v = [0:2:8]
v =
     0     2     4     6     8
```

Podemos definir un vector e que sean las tres primeras componentes del vector v como:

```
>> e=v(1:1:3)
e =
     0     2     4
```

Esta orden es equivalente a la siguiente, pues cuando el incremento es la unidad:

```
>> e=v(1:3)
e =
     0     2     4
```

Como comentamos al principio, la notación usada por Matlab sigue en lo posible la notación estándar de Álgebra Lineal que ya conocéis. Es muy sencillo multiplicar matrices y vectores, teniendo cuidado de que las dimensiones sean las adecuadas.

```
>> A*v(1:3)
??? Error using == *
Inner matrix dimensions must agree.
>> A*v(1:3)'
ans =
    16
    28
    46
```

Acostúmbrate a ver ese mensaje de error! Una vez que empiezas a trabajar con vectores y matrices, es sencillo olvidar los tamaños de los objetos que has creado.

Ejercicio 3.8 *Utilizando el comando size, razona sobre los problemas en lo que se refiere a dimensiones en la multiplicación anterior.*

Se pueden extraer columnas o filas de una matriz. Por ejemplo, quiero que C sea la tercera fila de la matriz A:

```
>> C=A(3,:)
C =
     6     7     8
```

O que C sea la segunda columna de la matriz B

```
>>C=B(:,2)
C =
     2
     4
     7
```

O que D sea la submatriz cuadrada de orden dos inferior derecha de la matriz A.

```
>> D=A(2:3,2:3)
D =
     4     5
     7     8
```

Una vez que eres capaz de crear y manipular una matriz, puedes realizar muchas operaciones estándar. Por ejemplo, puedes calcular su inversa. Tienes que tener cuidado, dado que las operaciones son cálculos numéricos realizados por ordenador. En el ejemplo, A no es una matriz regular, y sin embargo Matlab devolverá su inversa, pues los errores de redondeo durante su cálculo convierten en *invertible* a dicha matriz.

```
>> inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 4.565062e-18
ans =
   1.0e+15 *
   -2.7022    4.5036   -1.8014
    5.4043   -9.0072    3.6029
   -2.7022    4.5036   -1.8014
```

Con la matriz B sí que es posible calcular su inversa:

```
>>inv(B)
ans =
   -3.0000    5.0000   -2.0000
   -1.0000   -1.0000    1.0000
    2.0000   -1.0000         0
```

Ejercicio 3.9 *Definir una matriz de nombre $B1$ como la inversa de B. Multiplicar B por $B1$ y razonar la coherencia del resultado.*

Hay que comentar que Matlab distingue entre mayúsculas y minúsculas. Este puede ser el origen de algunas confusiones si manejas algoritmos complejos.

```
>> inv(a)
??? Undefined function or variable a.
```

4 Cálculo de los autovalores.

Otras operaciones se refieren a encontrar aproximaciones a los autovalores de una matriz. Hay dos versiones de este comando, una que encuentra sólamente los autovalores, mientras que la otra encuentra además sus autovectores correspondientes. Si olvidas cómo utilizarla, puedes obtener esta información pidiendo ayuda sobre esta orden en la línea de comandos de Matlab.

```
>> eig(A)
ans =
   14.0664
   -1.0664
    0.0000
>> [V,e] = eig(A)
V =
   -0.2656    0.7444   -0.4082
   -0.4912    0.1907    0.8165
   -0.8295   -0.6399   -0.4082
e =
   14.0664         0         0
         0   -1.0664         0
         0         0    0.0000
>> diag(e)
ans =
   14.0664
   -1.0664
    0.0000
```

Ejercicio 4.1 *Definir un vector w como la primera columna de la matriz V.*

Ejercicio 4.2 *Calcular el producto Aw.*

Ejercicio 4.3 *Calcular el producto $14.0664w$. ¿Son parecidos? ¿Por qué sí o por qué no?.*

5 Resolución de sistemas lineales.

También hay funciones para resolver sistemas lineales. Por ejemplo, si $Ax = b$ y queremos encontrar x, el modo más directo es simplemente invertir A, y luego multiplicar por la inversa ambos lados. Sin embargo hay medios mucho más eficientes y estables para resolver sistemas lineales (descomposición LU con pivote o eliminación gaussiana, por ejemplo). Matlab dispone de comandos especiales que permiten realizar estas operaciones. Por ejemplo si quiero resolver el sistema lineal $Bx = v$ con:

```
>>v = [1 3 5]'
v =
```

```
     1
     3
     5
>>B = [ [1 2 3]' [2 4 7]' [3 5 8]'];
```

lo haré por eliminación gaussiana con retrosustitución como:

```
>> x = B\v
x =
     2
     1
    -1
>> B*x
ans =
     1
     3
     5
```

Ejercicio 5.1 *Definir una matriz* $B2 = BB^t$.

Ejercicio 5.2 *Calcular los autovalores de* $B2$. *¿Qué tienen de especial estos autovalores?*

Ejercicio 5.3 *Encontrar la solución del sistema lineal* $BB^t x = v$ *asignándo esa solución al vector* x.

Ejercicio 5.4 *Comprobar la solución obtenida realizando el cálculo* $BB^t x - v$.

Podemos crear una matriz aumentada a partir de B y del término independiente y reducirla hasta convertir el sistema en uno equivalente triangular:

```
>>BA=[B v]
BA =
     1     2     3     1
     2     4     5     3
     3     7     8     5
>>BA(2,:)=BA(2,:)-2*BA(1,:)
BA =
     1     2     3     1
     0     0    -1     1
     3     7     8     5
>>BA(3,:)=BA(3,:)-3*BA(1,:)
BA =
     1     2     3     1
     0     0    -1     1
     0     1    -1     2
```

La segunda fila tiene el elemento diagonal nulo, así que hay que realizar una permutación de filas, premultiplicando por la identidad permutada:

```
>>IP=[1 0 0;0 0 1;0 1 0];
>>BA=IP*BA
BA =

     1     2     3     1
     0     1    -1     2
     0     0    -1     1
```

Ahora es fácil resolver este sistema por sustitución hacia atrás:

Ejercicio 5.5 *Definir una matriz H de 3x3 a partir de las tres primeras columnas de la matriz BA.*

Ejercicio 5.6 *Definir un vector h utilizando la ultima columna de BA.*

Ejercicio 5.7 *Definir el vector z tal que $Hz = h$. ¿Es coherente el resultado?*

Ejercicio 5.8 *Pedir ayuda de la función det.*

Ejercicio 5.9 *Calcular el determinante de la matriz H.*

6 Vectorización de operaciones.

Ejercicio 6.1 *Borrar la memoria porque vamos a empezar con cosas nuevas reutilizando nombres de variables que ya hemos usado.*

Con Matlab es sencillo crear vectores y matrices. La potencia de Matlab nace de la facilidad con la que se pueden manipular estos vectores y matrices. Primero mostraremos cómo realizar operaciones sencilas como sumar, restar y multiplicar. Luego las combinaremos para mostrar que se pueden realizar operaciones complejar a partir de estas operaciones simples sin mucho esfuerzo. Primero definiremos dos vectores, los cuales sumaremos y restaremos:

```
>> v = [1 2 3]'
v =
     1
     2
     3
>> b = [2 4 6]'
b =
     2
     4
     6
>> v+b
ans =
     3
```

```
     6
     9
>> v-b
ans =
    -1
    -2
    -3
```

La multiplicación de vectores y matrices sigue reglas estrictas, igual que su suma. En el ejemplo anterior, los vectores son ambos vectores columna con tres elementos. Por supuesto que no se puede sumar un vector fila con un vector columna. La multiplicación puede inducir a otros errores. El número de columnas del operando de la izquierda debe ser igual número de filas del de la derecha (esto es equivalente a un producto escalar de estos dos vectores si la matriz de Gramm es la identidad).

```
>> v*b
Error using == *
Inner matrix dimensions must agree.
>> v*b'
ans =
     2     4     6
     4     8    12
     6    12    18
>> v'*b
ans =
    28
```

Muchas veces queremos realizar una operación con cada uno de los elementos de un vector o matriz. Matlab permite realizar estas operaciones a nivel elemento de modo muy sencillo. Por ejemplo, supongamos que queremos multiplicar cada elemento del vector v con su correspondiente elemento en el vector b. En otras palabras, supongamos que quieres conocer $v(1) * b(1)$, $v(2) * b(2)$, y $v(3) * b(3)$. Sería estupendo poder usar el símbolo "$*$"pues en realidad estamos haciendo una especie de multiplicación, pero como esta multiplicación tiene un sentido diferente, tenemos que pensar algo diferente. Los programadores que crearon Matlab decidieron usar el símbolo "$\cdot*$"para realizar estas operaciones. De hecho, puedes poner un punto delante de cualquier símbolo para significar que queremos que las operaciones se realicen elemento a elemento.

```
>> v.*b
ans =
     2
     8
    18
>> v./b
ans =
    0.5000
```

```
    0.5000
    0.5000
```

Ejercicio 6.2 *Definir un vector tal que sus componentes sean las de v al cubo.*

Al haber abierto la puerta a operaciones no lineales, por qué no ir hasta el final? Si pasamos un vector a una función matemática predefinida, devolverá un vector del mismo tamaño en el que cada elemento se obtiene realizando la operación en el elemento correspondiente del vector original:

```
>> sin(v)
ans =
    0.8415
    0.9093
    0.1411
>> log(v)
ans =
         0
    0.6931
    1.0986
```

Saber manejar hábilmente estas funciones vectoriales es una de las ventajas de Matlab. De este modo, se pueden definir operaciones sencillas que se pueden realizar fácilmente y rápidamente. En el siguiente ejemplo, se define un vector muy grande y lo manipulamos de este modo tan sencillo. (Poner ; al final de la línea indica a Matlab que no debe mostrar en pantalla el resultado.)

```
>> x = [0:0.1:100]
x =
  Columns 1 through 7
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
  ......................
  Columns 995 through 1001
   99.4000   99.5000   99.6000   99.7000   99.8000   99.9000  100.0000
>> y = sin(x).*x./(1+cos(x));
```

Además, usando este tratamiento vectorial, Matlab permite generar gráficos de modo muy sencillo. Damos esta muestra pero nos dedicaremos a ello en detalle.

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

Ejercicio 6.3 *Definir un vector t cuya primera componente sea la -4, que tenga un incremento entre componentes de 0.05 y termine en el punto 1.*

Ejercicio 6.4 *Definir un vector y a partir de cada componente del vector t recién definido como:*

$$y = 5e^{-t^2} + sin(10t)$$

Ejercicio 6.5 *Dibujar la curva* t, y.

7 Creación de gráficas.

En esta sección presentamos los comandos básicos para crear representaciones gráficas de funciones. Para mostrar el uso del comando plot, utilizaremos la función seno y su desarrollo en serie en torno al cero, $x - x^3/6$. Seleccionamos el paso del vector de muestreo x para pintar la gráfica y sus valores superiores e inferiores.

```
>>h=0.1
>>xmin=-2;
>>xmax=2;
>>x=xmin:h:xmax;
>>yseno=sin(x);
>>ytaylor=x-x.^3/6;
```

Ahora, tenemos en los vectores *yseno* e *ytaylor* los valores reales y los valores aproximados a través del desarrollo. Para compararlos, dibujamos los valores exactos superpuestos con los aproximados dados por puntos verdes 'o'. El comando plot se utiliza para generar gráficas en Matlab. Admite una gran variedad de argumentos. Aquí sólo utilizaremos el rango y el formato, y la posibilidad de pintar dos curvas en la misma gráfica.

```
>>plot(x,yseno,'go',x,ytaylor);
```

Ejercicio 7.1 *En la ventana en la que aparece la figura, seleccionar Edit, Copy Figure. Abrir un nuevo documento de Word y pegar la figura en ese documento.*

También es buena idea pintar la función error:

```
>>plot(x,abs(yseno-ytaylor),'mx');
```

El siguiente ejemplo, también muestra la utilización del comando de petición de ayuda, *help* que se aplica a cualquier comando, en particular al comando *plot*. Como podéis comprobar la ayuda es muy completa.

```
>> plot(x,y,'rx')
>> help plot
 PLOT   Plot vectors or matrices.
    PLOT(X,Y) plots vector X versus vector Y. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matrix,
    whichever line up.

    PLOT(Y) plots the columns of Y versus their index.
    If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)).
    In all other uses of PLOT, the imaginary part is ignored.

    Various line types, plot symbols and colors may be obtained with
    PLOT(X,Y,S) where S is a 1, 2 or 3 character string made from
    the following characters:
```

```
y     yellow        .     point
m     magenta       o     circle
c     cyan          x     x-mark
r     red           +     plus
g     green         -     solid
b     blue          *     star
w     white         :     dotted
k     black         -.    dashdot
                    --    dashed
```

```
For example, PLOT(X,Y,'c+') plots a cyan plus at each data point.

PLOT(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) combines the plots defined by
the (X,Y,S) triples, where the X's and Y's are vectors or matrices
and the S's are strings.

For example, PLOT(X,Y,'y-',X,Y,'go') plots the data twice, with a
solid yellow line interpolating green circles at the data points.

The PLOT command, if no color is specified, makes automatic use of
the colors specified by the axes ColorOrder property.  The default
ColorOrder is listed in the table above for color systems where the
default is yellow for one line, and for multiple lines, to cycle
through the first six colors in the table.  For monochrome systems,
PLOT cycles over the axes LineStyleOrder property.

PLOT returns a column vector of handles to LINE objects, one
handle per line.

The X,Y pairs, or X,Y,S triples, can be followed by
parameter/value pairs to specify additional properties
of the lines.

See also SEMILOGX, SEMILOGY, LOGLOG, GRID, CLF, CLC, TITLE,
XLABEL, YLABEL, AXIS, AXES, HOLD, and SUBPLOT.
```

Ejercicio 7.2 *Pedir ayuda del comando grid.*

Ejercicio 7.3 *Dibujar la curva t, y del ejercicio 6.4 con con equis rojas y con una retícula incorporada.*

También se puede copiar desde la ventana del gráfico este gráfico al portapapeles para después pegarlo en un documento word por ejemplo, como ya vimos en un ejercicio anterior.

8 Conjuntos de ordenes.

En esta sección explicaremos cómo reunir órdenes en ficheros ejecutables desde la línea de comandos de Matlab. Ello permite realizar operaciones más complejas, y es más fácil repetir estas operaciones. Por ejemplo, podemos tener un conjunto de sentencias para realizar las aproximaciones y gráficas anteriores, pero podría ser interesante usar esas mismas órdenes para resolver otras ecuaciones. Lo primero que hacemos es ejecutar *clear* para borrar las variables activas.

Como ejemplo damos el fichero correspondiente al dibujo de las gráficas anteriores. Para ejecutar los comandos del fichero se debe especificar el intervalo entre los valores de las abscisas en el muestreo. De este modo, puedes construir infinidad de aproximaciones variando este parámetro. Primero hay que crear el fichero. El editor más conveniente es el que trae incorporado el propio Matlab. Este editor es muy simple y suficiente para este tipo de aplicaciones. A partir de la versión 5, viene incorporado al propio Matlab. Los ficheros ejecutables de Matlab, los M-files, deben tener la extensión ".m". En este ejemplo creamos un fichero de nombre *tutorm.m*. Para que Matlab ejecute los comandos en el fichero sólamente hay que ejecutar el comando *tutorm*. Este fichero lo podéis guardar donde queráis pero tenéis que decirle a Matlab donde está. Esto se hace indicando la ruta del archivo en el *path browser*. Por defecto, si lo guardáis en el directorio ..\matlab\bin, Matlab lo encontrará[4].

Una vez que el editor aparece en la pantalla (*File, New, M-file*)se trata de ir escribiendo y/o copiando-pegando los comandos necesarios. Debéis tener en cuenta que cuando una sentencia comienza por %, entonces es que es un comentario, y no se va a ejecutar. Por tanto, en este ejemplo, no es necesario reproducir esas líneas.

```
% file: tutorm.m
% Seno y desarrollo del seno.
%
% Para ejecutarlo tienes que fijar el paso
%      h       : intervalo entre las x
%
%  La rutina genera tres vectores, x con las abscisas, yseno con
%  el seno evaluado en esas abscisas, e ytaylor con el desarrollo
%  hasta el termino cubico del seno en torno al cero.
%
xmin=-2;
xmax=2;
x=xmin:h:xmax;
yseno=sin(x);
ytaylor=x-x.^3/6;
```

[4]Si utilizas Matlab en el centro de cálculo o laboratorio de tu facultad, probablemente no tengas permiso de escritura sobre ese directorio y no puedas guardar ahí tus ficheros. En este caso lo que tienes que hacer es guardarlos en la carpeta que creas oportuno e incorporar esa carpeta a la ruta de búsqueda (path), bien con el comando *path* o con el icono correspondiente

Una vez que hayas acabado de introducir las sentencias, salva el fichero. Vuelve a la ventana con la línea de comando y teclea en esta línea el nombre del fichero quitando *.m*, en este caso *tutorm*.

```
>>tutorm
??? Undefined function or variable 'h'.
Error in ==> C:\MATLAB\bin\tut.m
On line 13  ==> x=xmin:h:xmax;
```

Si tratas de llamar al fichero sin haber definido primero la variable h, aparecerá un mensaje de error. Debes definir todas las variables que no se definen en el propio fichero y que éste utiliza.

```
>>h = 0.01;
>>tutorm
>>plot(x,yseno,'rx',x,ytaylor)
```

Una vez que hayas definido las variables necesarias y que teclees *tutorm* en la línea de comandos, Matlab buscará en los directorios indicados en el path un fichero llamado *tutorm.m*. Una vez que lo encuentre leerá el fichero y ejecutará los comandos como si los hubieses tecleado uno detrás de otro en la línea de comandos. Si quieres ejecutar el programa otra vez pero con un paso diferente, tienes que tener cuidado. El programa sobreescribirá los vectores x, *yseno* e *yatylor*. Si quieres guardar estos vectores tienes que especificarlo.

```
>>x1 = x;
>>yseno1 = yseno;
>>ytaylor1 = ytaylor;
>>h = h/2;
>>tutorm
>>plot(x1,abs(yseno1-ytaylor1),'gx',x,abs(yseno-ytaylor),'yo');
```

Ahora tenemos dos aproximaciones. La primera es un paso de 0.1 y se almacena en los vectores $x1$ e $y1$. La segunda aproximación es para un paso de 0.05 y se guarda en los vectores x e y.

Ejercicio 8.1 *Crear y ejecutar desde Matlab un fichero que se llame BAIP.m con la secuencia de comandos siguiente*

```
v = [1 3 5]';
B = [ [1 2 3]' [2 4 7]' [3 5 8]'];
BA=[B v]
BA(2,:)=BA(2,:)-2*BA(1,:)
BA(3,:)=BA(3,:)-3*BA(1,:)
IP=[1 0 0;0 0 1;0 1 0];
BA=IP*BA
```

Ejercicio 8.2 *Pedir ayuda del comando pause e incorporarlo al ejercicio anterior para ver todos los pasos de la secuencia de comandos.*

Ejercicio 8.3 *Crear y ejecutar desde Matlab un fichero que se llame CURVATY.m con una secuencia de comandos que realicen las operaciones siguientes:*

1. *Borrar todas las variables activas de la memoria.*
2. *Definir un vector t cuya primera componente sea la -4, que tenga un incremento entre componentes de 0.05 y termine en el punto 1.*
3. *Definir un vector y a partir de cada componente del vector t recién definido como:*

$$y = 5e^{-t^2} + sin(10t)$$

4. *Dibujar la curva t, y con con equis rojas y con una retícula incorporada.*

Referencias

[1] Higham, D.J., Higham, N.J., MATLAB guide. Society for Industrial and Applied Mathematics, 2000.

Un libro sobre Matlab exhaustivo y con buenos ejemplos.

[2] The Math Works Inc, MATLAB, edición de estudiante: versión 4. Prentice Hall, cop. 1996.

Es el manual del programa en una versión más antigua, pero más que suficiente para el nivel de complejidad de las operaciones que vamos a realizar.

[3] Quintela Estévez, P., Matemáticas en ingeniería con MATLAB. Servicio de Publicacións da Universidade de Santiago de Compostela, 2000.

Tiene ejemplos hechos en Matlab, y además está en castellano.