TEORÍA DE SISTEMAS

PRÁCTICA 1: INTRODUCCIÓN A MATLAB

1. CARACTERÍSTICAS BÁSICAS DE MATLAB

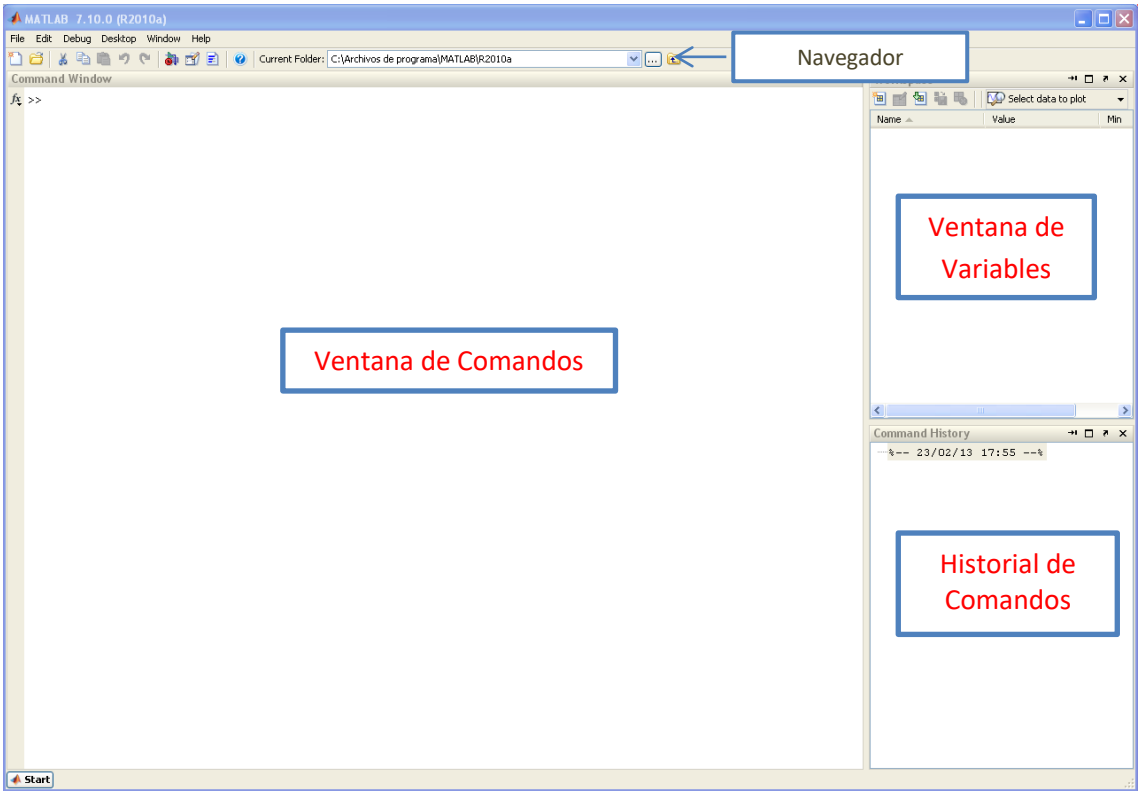
Matlab es una aplicación informática que facilita la el uso de funciones matemática de cálculo, algebra lineal y análisis numérico, incorporando representaciones gráficas y un lenguaje de programación sencillo que permite simular sistemas dinámicos.

Matlab dispone de librerías específicas denominadas ***toolboxes***. A lo largo del curso se utilizarán las siguientes:

- **Simulink**: simulación de sistemas dinámicos.
- **Control**: análisis de sistemas y ajuste de bucles de control.
- **Matemática simbólica**: opera con variables simbólicas.

2. EL ASPECTO DE MATLAB 6.5/2010

En el laboratorio está disponible dos versiones de Matlab (6.5 o 2010) que ofrecen un aspecto configurable en función de la información que se desee mostrar en la pantalla. En la figura siguiente se muestra un posible aspecto:



Existen cuatro elementos fundamentales, tal y como puede apreciarse en la imagen:

- Ventana de comandos ***(Command window)***: es un interfaz en modo texto sobre el que se introducen por teclado instrucciones Matlab. Será el elemento fundamental a utilizar en esta práctica.
- Ventana de variables ***(Workspace)***: en ella aparece un listado de todas las variables que se han empleado durante la sesión, de modo que pueden visualizarse su tamaño y su tipo.
- Historial de comandos ***(Command history)***: ofrece un listado de todas las instrucciones tecleadas durante la sesión.

- Navegador o ***(Current directory)***: indica los contenidos del directorio actual y permite navegar por la estructura de directorios del PC.

Es posible seleccionar cuáles de las ventanas se desean visualizar desde el menú ***Desktop***. Para la presente práctica se desactivará la visualización de todas las ventanas excepto la ventana de comandos.

3. CÓMO ENCONTRAR AYUDA EN MATLAB

Existen distintas formas de localizar ayuda en el entorno de Matlab:

- **Ayuda en línea**

Se accede a través de la ventana de comandos tecleando *help nombrefunción*. La ayuda se obtiene en modo texto. Como ejemplo, se visualizará la ayuda de la función que permite invertir matrices tecleando:

```
>> help inv
```

- **Navegador de ayuda**

Se accede desde el menú ***Help***, seleccionando la opción ***Product Help***. Constituye una manera más sencilla de localizar la misma información: las funciones están agrupadas en bloques y se proporciona un interfaz para navegar. Además ofrece información adicional como ejemplos e instrucciones de uso.

- **Ejemplos**

Matlab proporciona ejemplos y demostraciones de sus principales funcionalidades. Siempre es accesible el código fuente, con lo que puede ser directamente reutilizado. Se accede a ellos a través del menú ***Help***, seleccionando la opción ***Demos***.

- **Comando *lookfor* (búsqueda de palabras clave)**

Aunque más complicado de utilizar, proporciona en ocasiones información extra. El comando *lookfor* permite buscar entre las descripciones de todas las funciones de Matlab, aquellas que contienen la palabra clave que indiquemos. Como ejemplo, buscaremos todas las funciones de Matlab relacionadas con la transformada de Fourier tecleando:

```
>> lookfor Fourier
```

4. VARIABLES Y MATRICES EN MATLAB

Matlab soporta nombres de variable de hasta 19 caracteres, y distingue entre mayúsculas y minúsculas.
El tipo de las variables puede ser:

- Entero
- Real
- Complejo
- Carácter
- □..

... y es asignado automáticamente.

Una sentencia de creación de variable es, por ejemplo:

```
>> a = 7
a =
      7
```

Esta sentencia crea la variable entera **a** y le asigna el valor **7**. Matlab muestra en pantalla el resultado de cada operación. Para evitarlo basta poner un punto y coma después de cada sentencia:

```
>> a = 7;
```

Todas las variables en Matlab son consideradas matrices. Las posibilidades que utilizaremos son:

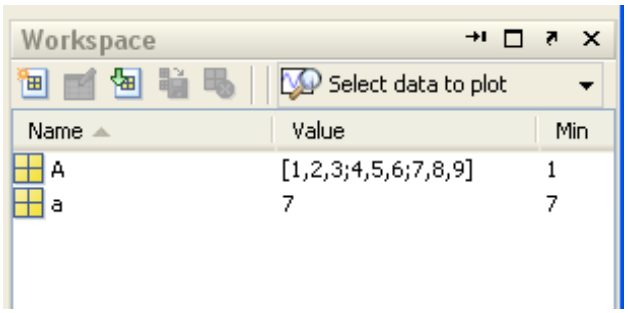
- Matriz **n x m**: matriz bidimensional
- Matriz **n x 1** ó **1 x n**: vector (se maneja exactamente igual que una matriz)
- Matriz **1 x 1**: escalar (también se maneja exactamente igual que una matriz).

La forma de definir una matriz en Matlab es elemento a elemento:

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

Como puede apreciarse en el ejemplo, los distintos elementos de una fila se separan mediante espacios (o comas) y las distintas filas se separan mediante puntos y coma.

NOTA: Es importante que te fijes que las variables **a** y **A** definidas anteriormente son distintas ya que Matlab distingue entre mayúsculas y minúsculas. En la ventana ***Workspace*** se muestran las variables definidas.



Algunas posibilidades de manejo de variables que ofrece Matlab:

- Comprobar el contenido de alguna variable: basta con teclear su nombre en la ventana de comandos

```
>> a
a =
     7
```

- Listar todas las variables existentes en un determinado momento: comando **who**.

```
>> who
Your variables are:
A a
```

- Eliminar alguna variable de memoria: comando **clear**.

```
» clear a
» who
Your variables are:
A
```

Podemos observar cómo la variable **a** ha desaparecido de la memoria.

5. MANEJO DE MATRICES

Matlab ofrece bastantes facilidades para el manejo de matrices. Volviendo al ejemplo anterior:

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

- Podemos acceder a cualquier elemento de la matriz especificando fila y columna :

```
» A (1,3)
ans =
      3
```

Nota: **ans** *es la variable por defecto donde Matlab guarda cualquier resultado; si hubiéramos deseado utilizar otra variable deberíamos haberlo especificado:*

```
» k = A(1,3)
k =
      3
```

Nota: *el primer índice en un vector o en una matriz es el **1** no el **0***

- También se puede acceder a toda una fila o toda una columna, utilizando el operador dos puntos.

Este primer comando muestra todos los elementos de la fila 2:

```
» A(2,:)
ans =
      4 5 6
```

Este segundo comando muestra todos los elementos de la columna 3:

```
» A (:,3)
ans =
      3
      6
      9
```

- bien a grupos de filas y/o columnas: Este comando muestra los elementos de las filas 1 hasta la 2 y de las columnas 2 hasta la 3:

```
» A(1:2,2:3)
ans =
2 3
5 6
```

- También es posible modificar cualquier elemento de una matriz:

```
» A(1,1) = 9
A =
9 2 3
4 5 6
7 8 9
```

- E incluso añadir elementos a una matriz dada:

```
» A(4,4) = 1
A =
9 2 3 0
4 5 6 0
7 8 9 0
0 0 0 1
```

Podemos ver cómo los elementos no especificados se rellenan con ceros.

6. PRINCIPALES OPERADORES ARITMÉTICOS

Matlab ofrece una serie de operadores aritméticos válidos tanto para cálculo matricial como para cálculo escalar:

- Suma: +
- Resta: -
- Producto: *
- División: /
- Traspuesta: ‘
- Potencia: ^

En algunas ocasiones podrán presentarse ambigüedades. Por ejemplo, al multiplicar dos matrices caben dos posibilidades: producto matricial o producto elemento a elemento. Veamos cómo se resuelven:

```
» A = [1 2;3 4]
A =
      1 2
      3 4
» B = [2 4; 6 8]
B =
      2 4
      6 8
» C = A*B % producto matricial
C =
      14 20
      30 44
» D = A.*B % el punto indica operación elemento a elemento
D =
      2 8
      18 32

» D’ % transpuesta de la matriz D
ans =
     2    18
     8    32

» [f c]=size(D) % obtiene el tamaño de la matriz
f =
     2
c =
     2
```

Además de los operadores comentados, existen una serie de funciones muy útiles en cálculo matricial:

- obtención de la matriz inversa: función **inv**:

```
» A = [1 2; 3 4]
A =
1 2
3 4
» B = inv(A)
B =
      -2.0000 1.0000
      1.5000 -0.5000
```

- creación de una matriz de ceros o unos: funciones **zeros** y **ones**:

```
» A = zeros(1,4)
A =
      0 0 0 0
» B = ones(2,3)
B =
      1 1 1
      1 1 1
```

- creación de un vector de términos crecientes o decrecientes:

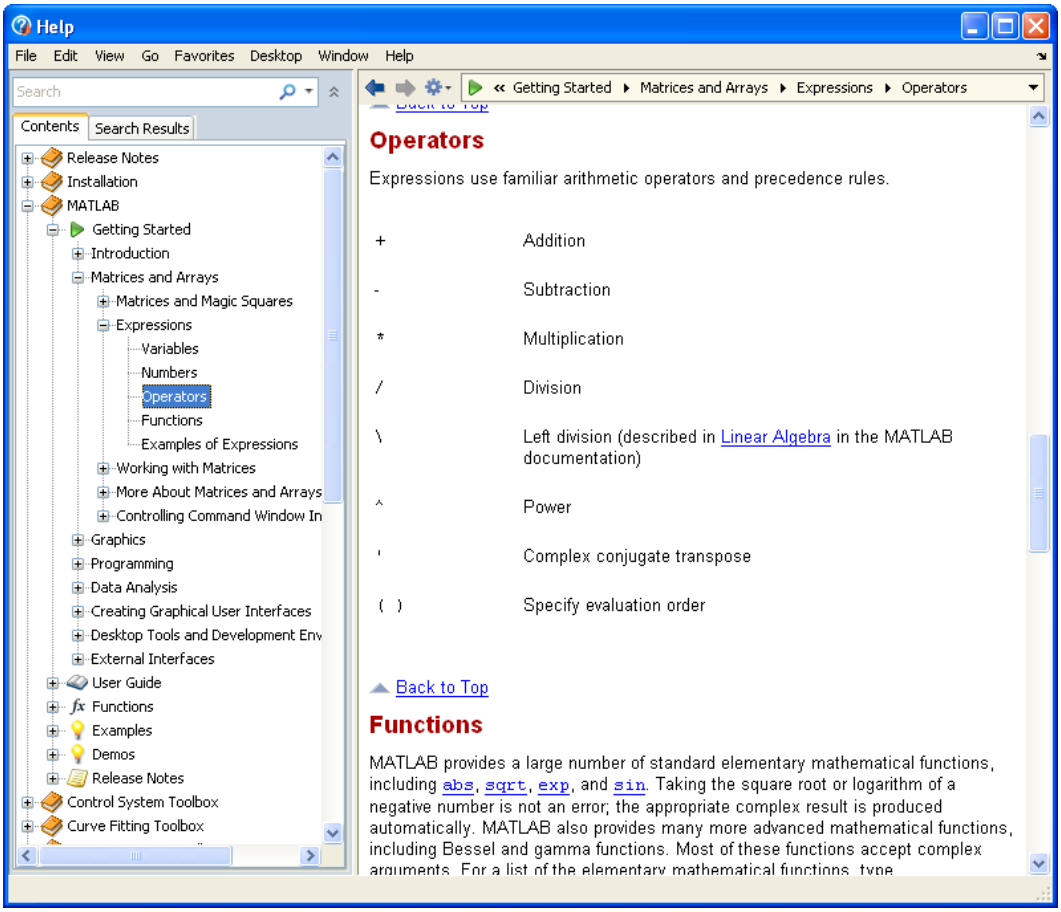
```
» a = [0:1:5] % inicio 0, fin 5, salto 1
a =
     0 1 2 3 4 5
» a = [5:-1:0] % inicio 5, fin 0, salto -1
a =
     5 4 3 2 1 0

» a = [0:.2:1] % inicio 0, fin 1, salto .2
a =
     0 0.2000 0.4000 0.6000 0.8000 1.0000
```

Podemos crear cualquier vector creciente o decreciente que deseemos. Esta operación será bastante útil para formar bases de tiempo sobre las que evaluar el valor de funciones.

NOTA: La información sobre operadores puede ser encontrada en la ayuda de Matlab. La ventana de ayuda se puede acceder desde el menú de Matlab ***Help -> Product Help***. En al ventana de ayuda aparecerá un árbol con todas las Toolboxes instaladas. En la pestaña MATLAB disponemos de la ayuda general. Por ejemplo para obtener ayuda acerca de las expresiones matriciales seleccionaremos

Matlab -> Getting started -> Matrices and Arrays -> Expressions -> Operators
Matlab Help -> Getting started ->Matrices and Arrays -> Working with matrices
Matlab Help -> Getting started -> Matrices and Arrays -> More about matrices and arrays



7. MODOS DE TRABAJO

Matlab permite trabajar de dos maneras distintas:

- **Mediante la introducción directa de comandos:**

Tecleando comandos desde la ventana principal de Matlab podemos realizar operaciones paso a paso. Será el método de trabajo a emplear para hacer pruebas o bien para operaciones sencillas no repetitivas.

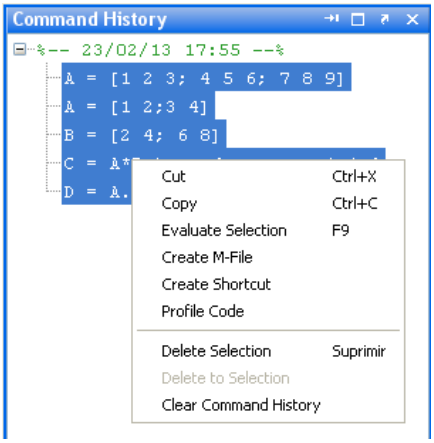
- **Mediante creación de programas (*.m)**

La misma secuencia de comandos que podríamos introducir desde la ventana principal puede archivarse en un fichero (que debe tener terminación '.m') y ser ejecutado posteriormente desde la ventana de comandos sencillamente tecleando el nombre del fichero.

Adicionalmente podemos almacenar las variables y las expresiones introducidos en la ventana de comandos, en ficheros para utilizarlos de nuevo posteriormente.

- Para guardar las variables creadas seleccionaremos del menú principal de Matlab ***File -> Save workspace as,*** indicando el nombre del fichero y el directorio donde guardarlo. La extensión será siempre '.***mat***'.

- Para guardar los comandos en un fichero M-File seleccionaremos en la ventana '***Command History***' el conjunto de comandos que nos interesen y pulsaremos el botón derecho del ratón. Esto nos mostrará un menú contextual en el que pulsaremos la opción '***Create M-File***'. También podemos borrar manualmente comandos no deseados.



8. PROGRAMACIÓN EN MATLAB: SENTENCIAS DE CONTROL: BUCLES, COMPARACIONES,

En comparación con otros lenguajes de programación, Matlab ofrece muchas facilidades para el usuario. Básicamente, cabe destacar:

- Elección automática del tipo de las variables
- Dimensionamiento automático de las matrices
- Posibilidad de manejar números complejos de modo intuitivo
- Posibilidad de funcionamiento en modo interpretado (chequeo de sentencias)
- Entorno de depuración integrado

Se muestra a continuación la sintaxis de las principales sentencias de control de Matlab:

Bucles:

```
for variable = expresion
      sentencias
end

while expresión
      sentencias
end
```

Nota: la expresión en el bucle **`for`** debe ser un rango de valores para la variable , utilizando para ello el operador **`:`** visto anteriormente. Ejemplo: **`for i = 1:n`**

Sentencia condicional *if/else/elseif*:

```
if expresión
      sentencias
elseif expresión
      sentencias
elseif expresión
      sentencias
else
      sentencias
end
```

Nota: las claúsulas else y elseif son opcionales.

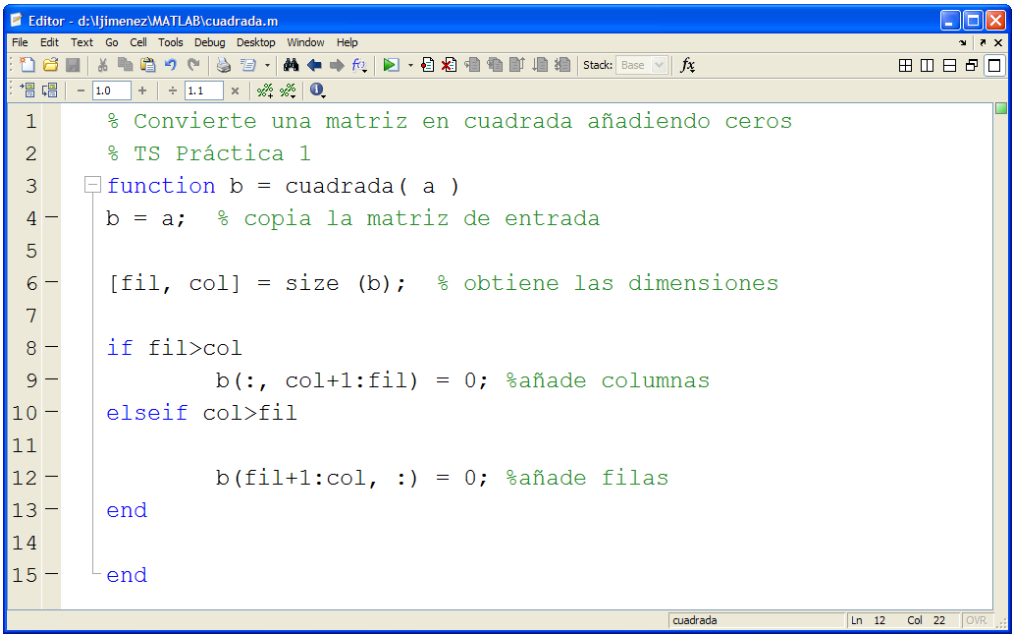
EJEMPLO:

Deseamos crear una función Matlab que, a partir de una matriz dada, genere una matriz cuadrada añadiendo filas o columnas de ceros, según sea necesario. La función se llamará **`cuadrada`** y se guardará en el fichero **`cuadrada.m`**, en el directorio de cada usuario.

Paso 1: creación del fichero cuadrada.m

Con la opción '***File->New->M-file'***, o bien '***File->New->Script***' según la versión de Matlab, se lanza el editor/depurador de código Matlab, donde crearemos nuestra función.

El código de nuestra función tendrá el siguiente aspecto una vez tecleado:




```
% Convierte una matriz en cuadrada añadiendo ceros
% TS Práctica 1
function [b] = cuadrada( a )

b = a;  % copia la matriz de entrada

[fil, col] = size (b);  % obtiene las dimensiones

if fil>col
     b(:, col+1:fil) = 0; %añade columnas
elseif col>fil
     b(fil+1:col, :) = 0; %añade filas
end

end
```

Si analizamos un poco en detalle este código, encontraremos elementos que necesariamente deberemos incluir en cualquier función que deseemos crear:

- **Línea de comentario**: es importante que la primera línea de una función contenga un texto explicativo, será la línea que se muestre al solicitar ayuda. Debe comenzar con el símbolo **%**.

- **Declaración de la función**: es obligatoria en cualquier **función**: especifica los parámetros de entrada y salida. En el ejemplo:

Si hubiera habido más de un parámetro de entrada, se habrían separado por comas en la declaración; por ejemplo:

```
function b = convolucion (a1, a2)
```

Y si hubiera habido más de un parámetro de salida, se habrían introducido entre corchetes en la declaración; por ejemplo:

```
function [b1 b2] = convolucion (a1, a2)
```

- **Cuerpo de la función**: contiene todas las operaciones que deseemos realizar.

- **End** o **Return**: sentencia de finalización de función. Se devolverá el valor que tenga asignada la variable que se especificó como salida (en este caso, la variable **b**).

Paso 2: Selección del directorio o carpeta donde guardar el programa

El directorio donde se archivan por defecto las funciones de usuario no es adecuado en muchos casos, particularmente cuando el mismo ordenador lo utilizan varias personas. En este caso será preferible crear un directorio propio por alumno donde guardar los datos. Se creará un directorio personal colgando de la carpeta "invitado", que en algunos ordenadores estará en la unidad C y en otros estará en la unidad D:

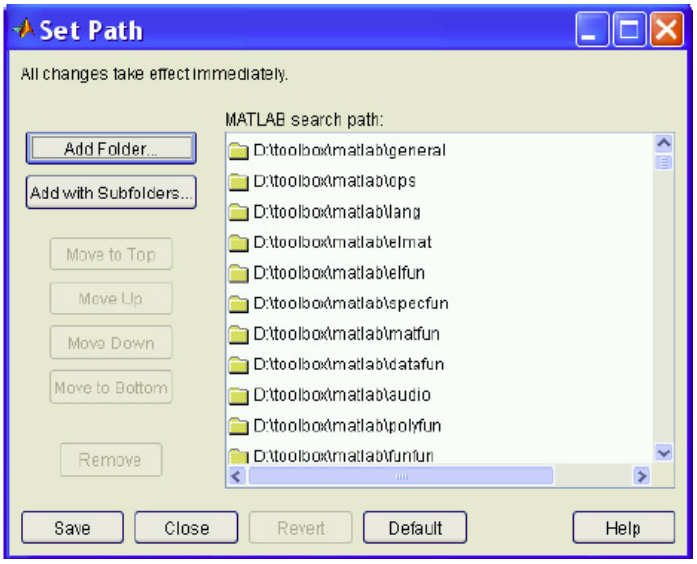
c:\invitado\<nombrealumno> **d:\invitado\<nombrealumno>**

En el directorio anterior, **<nombrealumno>** será sustituido por el nombre y apellidos del alumno. Para crear el directorio puede utilizarse el explorador de Windows o bien la ventana de navegación ***(Current directory)*** de Matlab

Una vez creado el directorio, el fichero que se tecleó previamente se guardará en él con la opción ***Save*** del menú ***File*** y con nombre **cuadrada.m** (es importante, éste será el nombre con el que accedamos a la función).

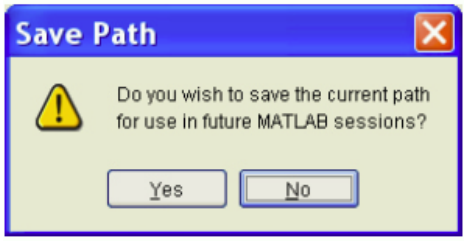
Paso 3: Modificación del path de Matlab

Matlab necesita conocer en qué directorios existen programas de usuario. Para ello dispone de la variable ***path***, que debemos modificar adecuadamente. La forma más sencilla de hacer esto en Matlab será a través de la opción ***Set Path*** del menú ***File***. Al seleccionar esa opción aparecerá una ventana como la que se muestra a continuación:



Para añadir un nuevo directorio al path de Matlab, se deberá hacer clic sobre el botón ***Add Folder***. Aparecerá un navegador que nos permitirá seleccionar el directorio recién creado para añadirlo al path.

Una vez seleccionado se pulsará el botón ***Close*** para que los cambios tengan efecto y se responderá **NO** cuando el sistema pregunte si se desean archivar las modificaciones en el path para la próxima vez que se ejecute Matlab, como muestra la figura siguiente:



Paso 4: Comprobación usando la función **help**

Si hemos incluido la primera línea de comentario en nuestra función y hemos modificado el path adecuadamente, al solicitar la ayuda de la función que acabamos de crear debemos obtener un resultado como éste:

```
>> help cuadrada
convierte una matriz en cuadrada añadiendo ceros
```

Paso 5: Utilización de la función con una matriz ejemplo

Probaremos la función con una matriz cualquiera:

```
» a = [1 2; 3 4; 5 6]
a =
      1 2
      3 4
      5 6
» b = cuadrada(a)
b =
      1 2 0
      3 4 0
      5 6 0
```

Vemos cómo se obtiene el resultado que esperábamos.

9. REPRESENTACIONES GRÁFICAS EN MATLAB

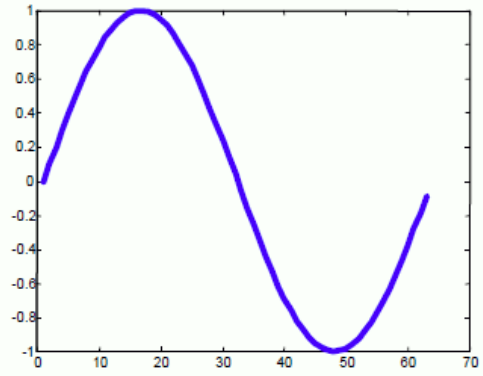
Matlab ofrece facilidades para la creación de gráficos 2D y 3D. Estudiaremos en primer lugar la función ***plot***, el medio más sencillo para realizar representaciones bidimensionales.

Existen diferentes sintaxis para la función ***plot***. Intentaremos mostrar su funcionamiento con un ejemplo: Supongamos que partimos de los siguientes datos iniciales:

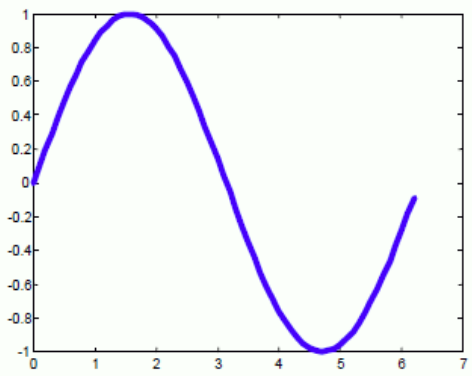
```
» a = [0:0.1:2*pi]; % a: contiene 63 ángulos entre 0 y 2pi
» b = sin(a); % b: contiene los valores del seno de a
» c = cos(a); % c: contiene los valores del coseno de a
```

Comparemos dos formas de representar la función seno:

```
>> plot (b);
```



```
>> plot (a,b);
```



El resultado es aparentemente el mismo, pero existe una gran diferencia que es posible observar comparando los ejes x de ambas gráficas:

plot (b) representa los valores del vector b en el eje y frente a los índices (números de orden) de ese vector en el eje x; por eso el eje x toma valores que van desde 1 hasta 63.

plot (a,b) representa los valores del vector b en el eje y frente a los valores correspondientes del vector a en el eje x; por eso el eje x toma valores entre 0 y 2≺.

Normalmente nos interesará más la segunda opción y la magnitud **a** representará la escala de tiempos.

Veamos ahora de qué forma podríamos representar a la vez el seno y el coseno, bien sobre un gráfico o sobre 2 gráficos distintos.

Si llamamos repetidamente a la función plot, el segundo gráfico borrará el primero, con lo cual no lograremos nuestro objetivo:

```
>> plot (a,b);
>> plot (a,c);
```

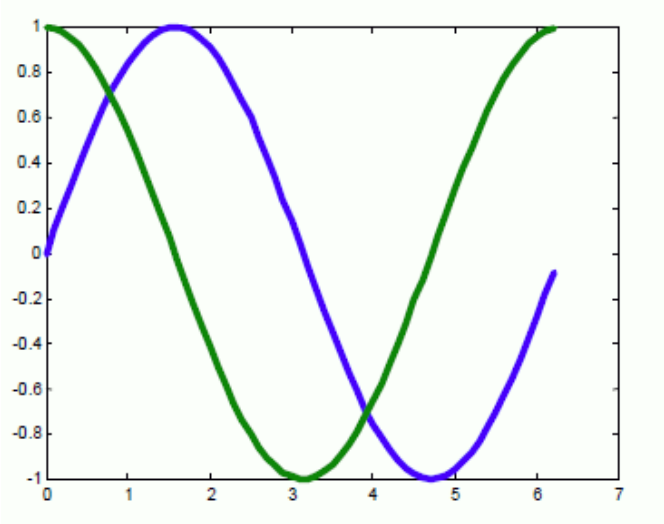
Si deseamos que el segundo gráfico se muestre sobre una ventana distinta, debemos intercalar la instrucción **'figure'**. Esta instrucción crea una nueva ventana de dibujo sobre la que se mostrarán todos los gráficos que se pidan a continuación:

```
>> plot (a,b);
>> figure;
>> plot (a,c);
```

En el caso de que deseemos representar ambas funciones sobre un mismo gráfico, será necesario intercalar la instrucción **'hold on'**. Esta instrucción permite dibujar nuevos datos sobre los datos anteriores, sin borrarlos:

```
>> plot (a,b);
>> hold on;
>> plot (a,c);
```

El resultado de las instrucciones anteriores debería ser similar al siguiente:



Cuando superponemos varias curvas en la misma ventana conviene elegir diferentes colores, para ello utilizaremos un parámetro adicional en el que indicaremos el color y el tipo de línea como una cadena de texto.

```
>> plot (a,b, 'g');
>> hold on;
>> plot (a,c, 'b');
```

A continuación se muestran todas las combinaciones disponibles de color y tipo de línea:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	white	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

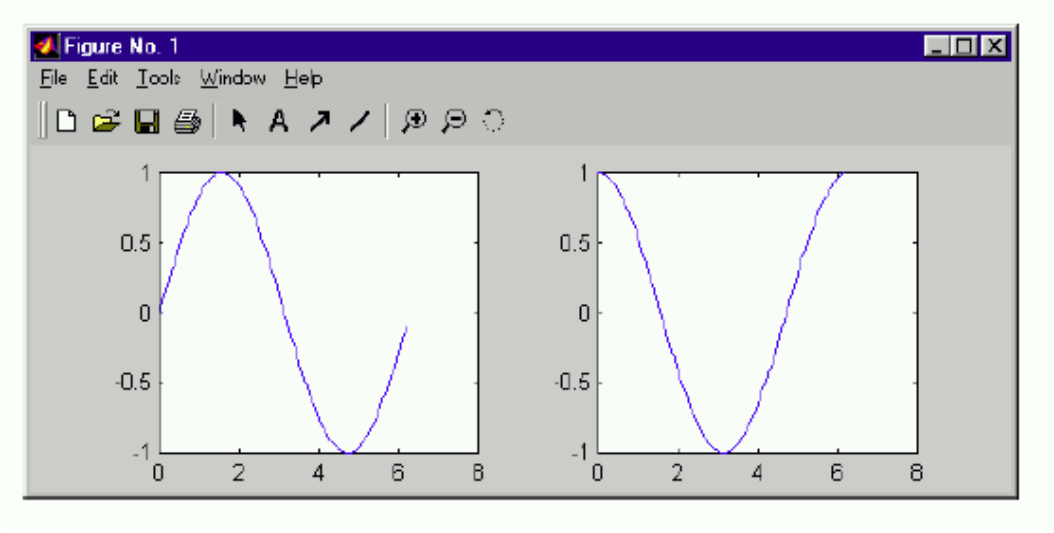
Si, por último, deseamos representar ambas funciones sobre una misma ventana pero en gráficos separados, deberemos utilizar la instrucción **'subplot'**. El formato de esta instrucción es el siguiente:

```
subplot(filas, columnas, activar)
```

Donde filas y columnas representa el número de filas y columnas de gráficos que queremos que aparezcan en nuestra ventana, y activar indica el gráfico sobre el que queremos dibujar. Si, por ejemplo, deseamos mostrar a la izquierda el gráfico del seno y a la derecha el gráfico del coseno, se deberá crear una ventana con dos columnas y una fila de gráficos, de acuerdo con las instrucciones siguientes:

```
» subplot(1,2,1) % crea ventana 1x2 y selecciona primer gráfico
» plot(a,b) % representa el seno
» subplot(1,2,2) % selecciona el segundo gráfico
» plot(a,c) % representa el coseno
```

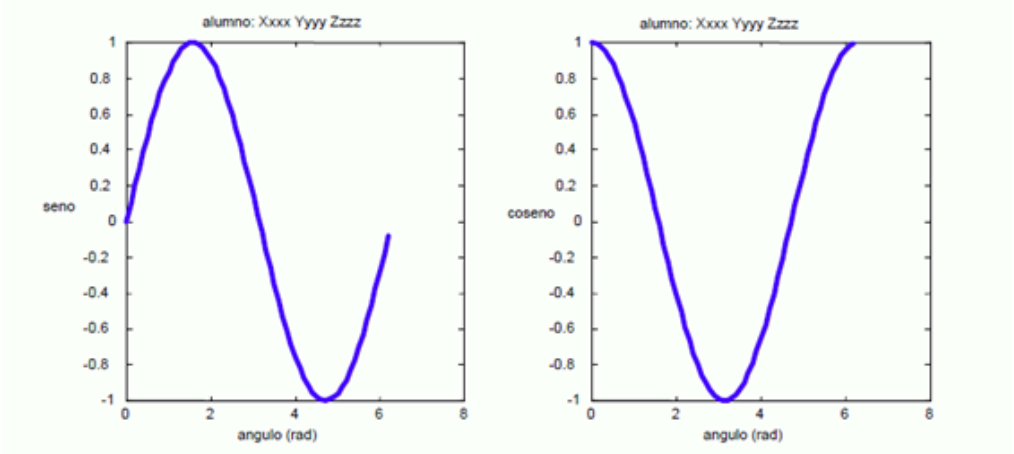
El resultado debería ser similar al que se muestra en la figura siguiente:



Un último aspecto que será importante a la hora de obtener representaciones gráficas será la forma de incluir textos sobre estas representaciones. Las principales instrucciones a utilizar son:

- **`title 'texto':`** escribe un título para el gráfico (en la parte superior).
- **`xlabel 'texto ':`** da un nombre al eje x del gráfico.
- **`ylabel 'texto':`** da un nombre al eje y del gráfico.
- **`legend 'texto':`** es útil cuando superponemos varios gráficos, la leyenda aparecerá superpuesta al gráfico e indica lo que representa cada trazo del gráfico.

La forma de utilizar estas instrucciones se puede encontrar en la ayuda de Matlab. Como vemos el parámetro es una cadena de texto que en Matlab se indica enmarcada en comillas simple**`'.`** Utilizaremos fundamentalmente las tres primeras instrucciones. Con ellas, y sobre el último gráfico realizado, se puede obtener un resultado como el siguiente:



De ahora en adelante, todos los gráficos que obtengáis durante las prácticas deberán incluir un título en el que se indiquen asignatura, curso y nombre de alumno similar al mostrado en este ejemplo.

*La información necesaria sobre la función **plot** puede ser encontrada en la ayuda de Matlab en:*
Matlab help -> Graphics -> Basic plotting
Matlab help -> Graphics -> Formatting graphs

EJERCICIO MATLAB

1. Crear una función Matlab que sea capaz de multiplicar dos matrices y obtener la matriz inversa del resultado

La declaración de la función debe ser como la siguiente:

```
function resultado = calcula (matriz1, matriz2)
```

Nota: la función debe guardarse como ***calcula.m.***

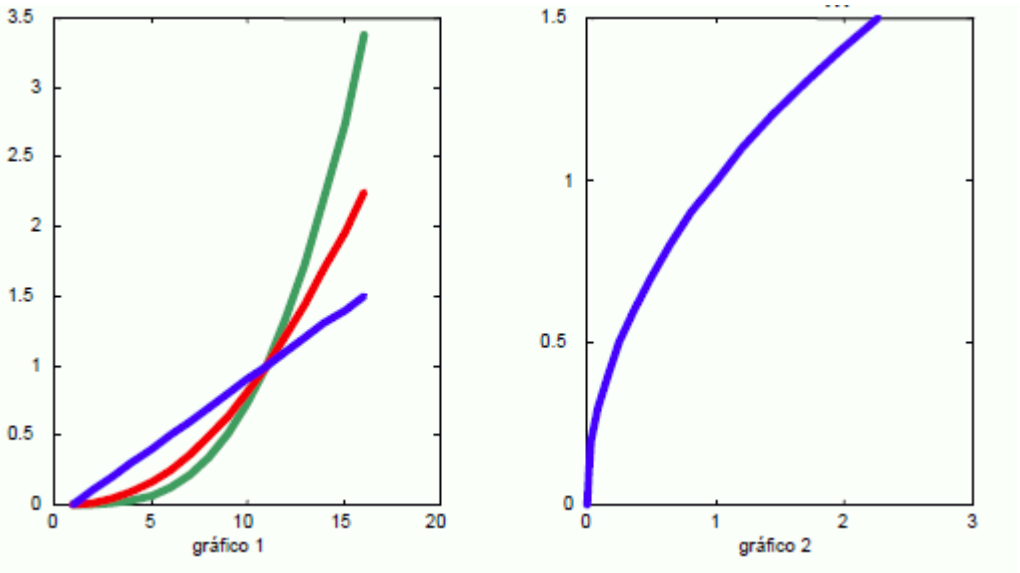
2. Crear una función Matlab que, tomando como entrada un vector V cualquiera, represente en una misma ventana (subplot) los dos gráficos siguientes:

- Gráfico 1: valores de cada elemento de V, de cada elemento de V al cuadrado y de cada elemento de V al cubo (eje y) con respecto al número de orden (eje x).

- Gráfico 2: valores de V (eje y) con respecto a los valores de V^2 (eje x).

Se muestra el aspecto que deben tener los gráficos para un vector de entrada como el siguiente:

```
» v = [0:0.1:1.5] % desde 0 hasta 1.5 en incrementos de 0.1
```



La declaración de la función debe tener este aspecto:

```
function dibuja (v)
```

Nota: la función debe guardarse en el fichero ***dibuja.m***