



PROCESADO DIGITAL DE SEÑAL

PROYECTO ESPECÍFICO - LABORATORIO 1 (Mat)

MATLAB

INTRODUCCIÓN A MATLAB PARA EL PROCESADO DE SEÑALES. GENERACIÓN DE SEÑALES Y SUS CARACTERÍSTICAS.

OBJETIVOS

Familiarizarse con la utilización de MATLAB para procesado de señal:

1. Manejo básico de MATLAB (matrices, funciones, programación, ...)
2. Uso de matrices para almacenar y generar señales.
3. Uso de las funciones de MATLAB para generar señales.
4. Programación en MATLAB
5. Carga de señales externas o generadas con MATLAB en sesiones anteriores.
6. Visualización de señales de diferentes tipos.

INTRODUCCIÓN

La palabra **MATLAB** proviene de las palabras inglesas "**MATrix LABoratory**", y da nombre a un programa de cálculo matemático interactivo basado en matrices y ampliamente utilizado en ciencias e ingeniería. Se desarrolló pensando en facilitar la utilización de las bibliotecas *Linpack* y *Eispack*, desarrolladas con anterioridad para cálculo matricial (resolución de sistemas de ecuaciones lineales, y, problemas de valores propios). El objetivo era resolver problemas numéricos complejos sin tener que escribir programas en lenguajes como el C.

A pesar de su relativamente simple interface, es posible extenderlo con facilidad creando nuevos comandos y funciones. A lo largo de los años, y aprovechando esta capacidad, se han desarrollado numerosos conjuntos de funciones (**toolboxes**) que facilitan el empleo de MATLAB en distintas áreas (procesamiento de señal, control, ...). Nosotros emplearemos los toolboxes para procesado de señal, procesado de imagen y adquisición de datos.

MATLAB no es un lenguaje que compila código fuente como el C, sino que se basa en un intérprete de comandos. Lo que sucede es que el número de comandos es grande y las "toolboxes" amplían este número con funciones especializadas. Además, es muy fácil para el usuario crear nuevas funciones a su medida.

El objetivo de esta práctica es familiarizarse con los comandos básicos para empezar a trabajar y, al mismo tiempo, comprobar la facilidad con que pueden manejarse vectores de datos lo que lo convierte en la herramienta ideal para el procesado de señales.

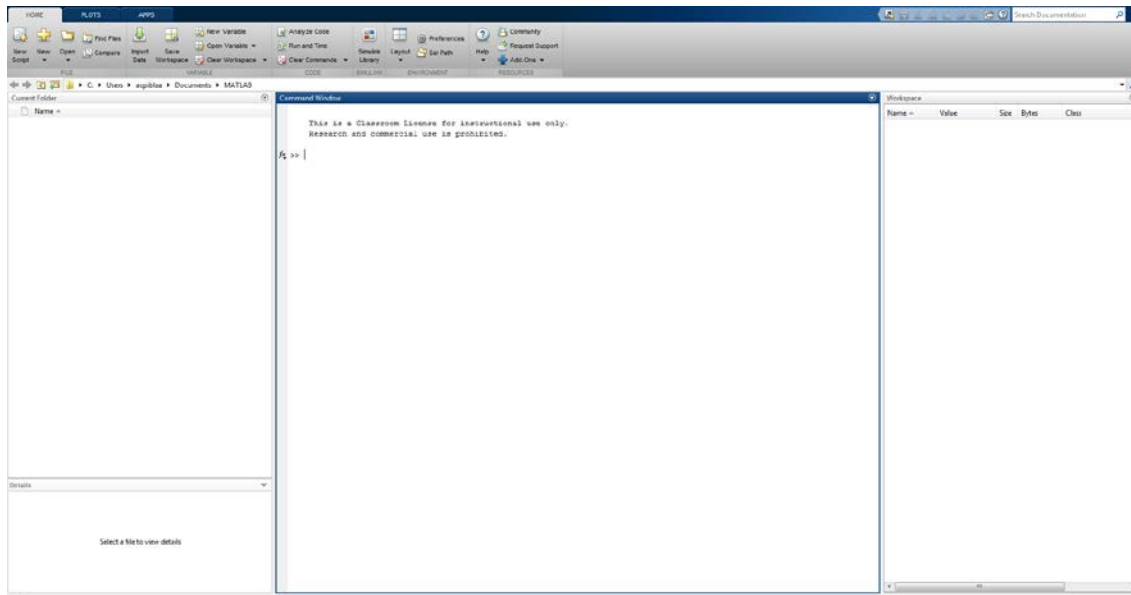
Las funciones de MATLAB se ejecutan mediante comandos, por ello conviene conocer la variedad de funciones disponibles. De todos modos, el programa dispone de **ayuda "on line"** y la sintaxis de la mayoría de las funciones es muy similar, por lo que no es necesario memorizarla.

Si desarrollas tu habilidad en el uso de MATLAB podrás, mediante sus potentes funciones, enfrentarte a numerosos problemas de cálculo numérico con rapidez y eficacia.

COMO SE EJECUTA MATLAB

MATLAB existe en muchas plataformas y el arranque varía de unas a otras. En nuestro caso, vamos a utilizarlo sobre PCs con sistema operativo Windows, y el arranque se realizará desde el icono de acceso directo que se encuentra en el escritorio

Al arrancarlo se abrirá la ventana de MATLAB



y dentro de ella la ventana de comandos. El símbolo » indica que el programa está esperando que introduzcamos un comando. Para comprobar lo que es capaz de hacer prueba a teclear

```
» 2+2  
ans =  
    4
```

Como puedes ver MATLAB ¡sabe sumar!. El comando introducido es muy simple y consiste en una expresión matemática. En este caso MATLAB evalúa la expresión y almacena el resultado en una variable llamada **"ans" (de answer)**, mostrándonos en pantalla el resultado.

Al finalizar el trabajo y para abandonar el programa hay que cerrar la ventana o teclear el comando

```
» quit
```

USO BÁSICO DE MATLAB PARA REALIZAR CÁLCULOS

VARIABLES Y ESPACIO DE TRABAJO MATLAB

La mayoría de los comandos producen un resultado numérico, y éste se almacena en una variable para su posible utilización posterior. Si no se indica explícitamente, el nombre de la variable es por defecto "ans", pero en la siguiente operación su contenido se perderá al tomar el valor del nuevo resultado. Para almacenar el valor de forma permanente debemos indicar el nombre que deseamos dar a la variable y utilizar el operador de asignación "=".

```
>> x = 2+2
x =
    4
```

Los nombres de las variables en MATLAB, deben comenzar con una letra y sólo pueden contener letras, dígitos y el carácter "_" (subrayado). Además, MATLAB distingue entre mayúsculas y minúsculas. A medida que trabajamos las variables creadas se almacenan formando el espacio de trabajo de MATLAB, y pueden ser usadas en nuevos comandos. El listado de variables, su formato y valor se muestra en la ventana "workspace" que habitualmente se sitúa en la parte derecha.

```
>> x
x =
    4
>> y = 2
y =
    2
>> a = 34;
>> b = 27;
>> total = a*x + b*y
total =
   190
```

Habrás observado que el carácter ";" al final del comando hace que no se muestre el resultado en pantalla. Esto puede ser importante cuando utilicemos vectores o matrices con muchos elementos.

El valor de una variable puede modificarse en cualquier momento, sin más que asignarle un nuevo valor.

```
>> x = 3; total = a*x + b*y
total =
   156
```

Fíjate en que se puede teclear más de un comando en la misma línea, separándolos con ";" o con ",". Intenta ver la diferencia al usar uno u otro.

Para facilitar la introducción de comandos el programa recuerda los comandos ya tecleados. Para verlos y reutilizarlos pueden utilizarse las flechas ↑ y ↓, y con ← y → podemos movernos por el texto y modificarlo. Si escribimos alguna letra y tecleamos ↑ aparecerán sólo los comandos que empiecen con esas letras.

El comando "who" muestra el nombre de las variables almacenadas en el espacio de trabajo o "workspace".

```
>> who
Your variables are:
ans                a                b
total              x                y
```

"whos" muestra además su tamaño.

El espacio de trabajo puede guardarse en un fichero y recuperarse en una sesión posterior, como veremos más adelante, mediante los comandos "save" y "load".

Al arrancar, MATLAB tiene definidos varios nombres de **variables especiales**:

ans	Nombre de la variable resultado por defecto.
pi	3.1416
eps	Número mínimo que sumado a 1 produce un número mayor que 1.
inf	Infinito. MATLAB lo emplea como resultado de operaciones como 1/0
NaN	"Not a Number". Como resultado de operaciones como 0/0.
i o j	Unidad imaginaria: $\text{Sqrt}(-1)$.
realmin	El número real positivo más pequeño utilizable.
realmax	El número real positivo más grande utilizable.

MATRICES

MATLAB usa, por defecto, un único tipo de datos, **matrices de números complejos en coma flotante**. Un vector no es más que una matriz con una sola fila (vector fila) o con una sola columna (vector columna). Un número o escalar es una matriz 1x1. Un número real es un número complejo sin parte imaginaria.

Una de las mayores comodidades de MATLAB es que **no es necesario dimensionar las variables**, el propio MATLAB asigna el tamaño necesario a la matriz cada vez que se le asigna valores. Tampoco hace falta tratar de forma diferente los números complejos.

A la hora de asignar valores a una matriz, se usan **corchetes** para indicar el comienzo y el final de la matriz. Los elementos se separan por **comas o espacios** en blanco, si están en la misma fila, y para separar filas se usa el **punto y coma o el <Return>**. Si se desea inicializar un vector o matriz sin especificar su dimensión se recurre a:

```
» VectorF = []
```

Por ejemplo, el siguiente comando, crea una variable llamada VectorF y le asigna valores numéricos. El resultado se muestra en pantalla y es un vector fila (matriz 1x3).

```
» VectorF = [1 2 3]
VectorF =
     1     2     3
```

El siguiente comando genera un vector columna (matriz 3x1) con datos complejos.

```
» VectorC = [1; sqrt(-2); 3*j]
VectorC =
 1.0000
 0 + 1.4142i
 0 + 3.0000i
```

Como habrás apreciado, has introducido un número complejo utilizando la variable j que MATLAB interpreta como la unidad imaginaria (también vale i), pero recuerda que estos nombres no están reservados, si les asignas otro valor se comportarán como variables.

Por otra parte, puedes ver que cuando los resultados son complejos MATLAB los muestra en forma binómica.

El siguiente ejemplo genera una matriz 2x2.

```
» Matriz2x2 = [1 2; 3 4]
Matriz2x2 =
     1     2
     3     4
```

Un error común para los programadores de C, por el parecido en la notación, consiste en indexar los elementos de las matrices empezando por 0. **Sin embargo, para MATLAB el primer elemento es el 1.**

```
» x = [1 3 5 7 9]
x =
     1     3     5     7     9
```

```

» x(0)
??? Index exceeds matrix dimensions.
» x(1)
ans =
     1
» A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
» A(2,3)
ans =
     6
» A(3,2) = 10
A =
     1     2     3
     3     5     6
     7    10     9

```

OPERACIONES CON MATRICES

MATLAB permite realizar las siguientes **operaciones con matrices**

+	suma	-	resta
*	multiplicación	^	potencia
\	división por la izquierda	/	división por la derecha
'	transposición		

Para comprobarlo realiza las siguientes operaciones

```

» A = [1 2 3 ; 4 5 6 ; 7 8 9], b = [2; 4; 6]
» B = A'
» 2*A
» A/3
» A + [b,b,b]

```

MATLAB admite, también, algunas **operaciones que no existen en el Álgebra tradicional**, por ejemplo, la **suma de una matriz y un escalar**. Lo que hace MATLAB, cuando se encuentra una expresión de este tipo, es sumar el escalar a todos los elementos de la matriz

```

» A+1
ans =
     2     3     4
     5     6     7
     8     9    10

```

La **multiplicación de matrices** se realiza según las reglas del álgebra y MATLAB produce un mensaje de error si las dimensiones no son correctas (nº de columnas del primer factor igual al nº de filas del segundo).

```

» A*b
» b'*A
» A*A', A'*A
» b'*b, b*b'

```

MATLAB permite también realizar **operaciones entre matrices de las mismas dimensiones elemento a elemento**, escribiendo un punto delante del operador. Por ejemplo, **X.*Y** representa el producto de dos matrices X e Y. Sin embargo **X.*Y** representa una matriz en la que cada elemento se obtiene multiplicando los elementos de X e Y que están en la misma posición (X e Y deben tener las mismas dimensiones).

```

» A^2, A.^2
» A.*A, b.*b
» 1./A

```

```
» 1./A.^2
```

Con lo visto hasta ahora, ya podemos ver que al menos como calculadora, MATLAB es muy eficaz para resolver problemas con matrices. Si la matriz A contiene los coeficientes de un sistema de ecuaciones lineales y b es el vector columna con los términos independientes,

$$A x = b$$

podemos obtener la solución al problema ejecutando simplemente:

```
» x = A \ b
```

ya que dividir por la izquierda equivale a multiplicar por la izquierda la inversa.

OPERADORES RELACIONALES Y LÓGICOS

Los operadores relacionales permiten la comparación de escalares (o matrices, elemento a elemento). El resultado de una operación de este tipo es un 0 o un 1 (o una matriz del mismo tamaño que los argumentos formada por ceros y unos) según el resultado de la comparación sea falso o cierto. Se pueden utilizar los siguientes operadores:

<	- menor	<=	- menor o igual
>	- mayor	>=	- mayor o igual
==	- igual	~=	- distinto

Además las expresiones relacionales pueden combinarse mediante los operadores lógicos:

& - and	- or	~ - not (ctl alt 4)
---------	------	---------------------

Prueba las siguientes operaciones

```
» 3 > 5
» 3 < 5
» 3 == 5
» 3 == 3
» A == 5
```

CONSTRUCCIÓN DE MATRICES

Para sacar todo el partido al manejo de matrices en MATLAB, sólo nos falta ver como construir matrices de forma cómoda. Para facilitar estas tareas, MATLAB incorpora el operador ":" que permite generar vectores y referenciar submatrices de forma sencilla. El uso de este operador permitirá reducir el número de bucles en nuestros programas y hará el código mucho más simple y legible.

La expresión 1:5 representa un vector formado por los números 1 al 5.

```
» 1:5
ans =
     1     2     3     4     5
```

Los números no tienen por que ser enteros, ni el incremento 1

```
» 0.1:0.2:0.9
ans =
    0.1000    0.3000    0.5000    0.7000    0.9000
» 6:-1:1
ans =
     6     5     4     3     2     1
```

Los siguientes comandos generan una tabla de cosenos

```
» x = [0.0:0.1:2.0]';
```

```
» y = cos(x);
» [x y]
```

¿Sencillo no? Fijaos en que la función cos es una función escalar y que en MATLAB **cuando se aplica una función escalar a un vector, se aplica a cada elemento del vector**. Fijaos también en que los corchetes permiten combinar matrices para formar otra matriz mayor. En este caso formamos una matriz de dos columnas mediante dos vectores columnas.

Otro ejemplo útil para generar **impulsos**, es combinar los dos puntos con un operador lógico:

```
» v = 1:10
» d = (v == 3)
```

se genera un vector v de 10 elementos de valor 0 salvo en la 3 posición que vale 1.

Otra propiedad que a veces es útil es definir un vector y sumarle una constante escalar (se sumará a cada elemento):

```
» v = 1:8
» v = v+2
```

Los dos puntos se pueden usar para referenciar **submatrices**. Por ejemplo

```
» A(1:3,2)
```

es el vector columna formado por los tres primeros elementos de la segunda columna de A. **Los dos puntos por sí solos representan una fila o columna entera.**

```
» A(:,2)
```

es la segunda columna de A y

```
» A(1:2:5,:)
```

es la matriz formada por las filas 1, 3 y 5 de A

En realidad, cualquier vector de elementos enteros puede utilizarse para referenciar elementos de una matriz. Por ejemplo

```
» A(:, [1 4])
```

contiene **las columnas 1 y 4 de A**.

Esta forma de referenciar los elementos de una matriz es también válida a la izquierda del operador “=” de asignación

```
» A(:, [2 4 5]) = B(:, 1:3)
```

reemplaza las columnas 2, 4 y 5 de A con las tres primeras columnas de B

Si x es un vector de n elementos ¿Qué efecto produce el siguiente comando?

```
» x = x(n:-1:1)
```

Cuando os familiaricéis con este operador os resultará más sencillo que programar en C ¿no?

Además de esto, MATLAB dispone de unas cuantas funciones para construir algunos **tipos de matrices**, las más útiles son:

eye	matriz identidad
ones	matriz formada por unos
zeros	matriz formada por ceros
diag	matriz diagonal
triu	parte triangular superior de una matriz

tril parte triangular inferior de una matriz
rand matriz de números aleatorios

Por ejemplo, `zeros(m,n)` produce una matriz $m \times n$ formada por ceros, y `zeros(n)` produce una matriz nula $n \times n$.

```
> zeros(2,3)
ans =
     0     0     0
     0     0     0
```

Si b es un vector, `diag(b)` es una matriz diagonal formada con los elementos de b , y, si A es una matriz cuadrada entonces `diag(A)` es un vector formado por los elementos de la diagonal de A . ¿Qué será entonces `diag(diag(A))`?

Prueba el siguiente comando que construye una matriz B 5×5 a partir de A 3×3

```
> B = [A, zeros(3,2); zeros(2,3), eye(2)]
```

FUNCIONES ESCALARES, VECTORIALES Y MATRICIALES

Una vez que ya sabemos como manejar datos en MATLAB y operar con ellos hay que hablar de las **funciones**. Como podéis imaginar MATLAB dispone de muchas funciones que, como veremos más adelante, podemos ampliar con nuevas funciones escritas por nosotros. Algunas funciones son **escalares**. Por ejemplo, algunas de las funciones más empleadas son las siguientes

sin	cos	tan	asin	acos
atan	exp	sign	rem (resto)	abs
sqrt	log	round	floor	ceil

Otras funciones de MATLAB tienen como **argumento vectores** (matrices fila o columna). Si se usan con una matriz como argumento, **MATLAB aplica la función a cada columna de la matriz**. Algunas de estas funciones son:

max	min	prod	median	mean
std	sum	cumsum	cumprod	

Por ejemplo, para hallar el elemento mayor de la matriz A , podemos aplicar la función `max` en cascada

```
> max(max(A))
ans =
     9
```

El resultado de algunas de estas funciones puede ser múltiple (más de un resultado) dependiendo de cómo se utilicen. Por ejemplo, las funciones `max` y `min` además del valor máximo o mínimo pueden **devolver la posición** de este valor en el vector argumento. Por ejemplo

```
> [Max_b index_b] = max(b)
Max_b =
     6
index_b =
     3
```

Ahora bien, si asignamos el resultado a un escalar, como hemos visto antes, se asigna el primer elemento del vector resultado.

Por último, hay funciones cuyo **argumento** son **matrices**, por ejemplo:

inv	- matriz inversa
det	- determinante
norm	- norma de una matriz o de un vector
size	- tamaño de una matriz
rank	- número de filas o columnas linealmente independientes de una matriz
eig	- valores y vectores propios
poly	- characteristic polynomial.

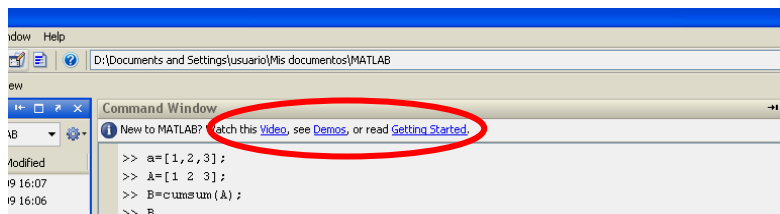
- lu - factorización LU
- qr - descomposición triangular ortogonal

BUSCANDO AYUDA

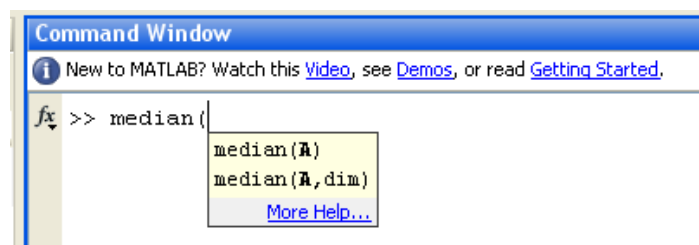
Viendo la gran cantidad de funciones de las que dispone MATLAB, puede parecer difícil acordarse de ellas y de su sintaxis, pero MATLAB incluye ayuda para facilitarnos su uso. Para obtener información sobre los comandos de MATLAB basta seleccionar **help** en la barra superior para que se abra la ventana siguiente:



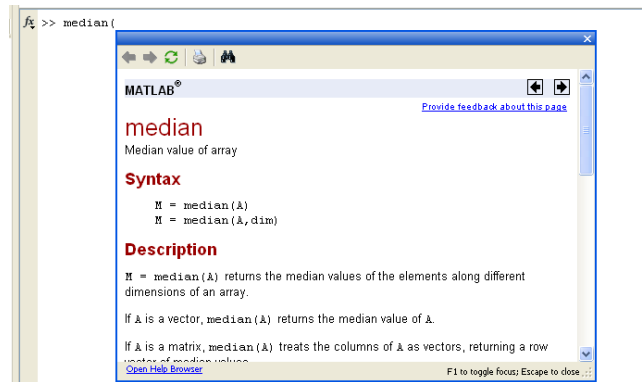
Desde esta ventana podemos acceder a todo tipo de información sobre MATLAB: ejemplos, funciones, demos, etc. También desde la barra superior de la ventana de comandos podemos acceder a videos y demos explicativos sobre este programa.



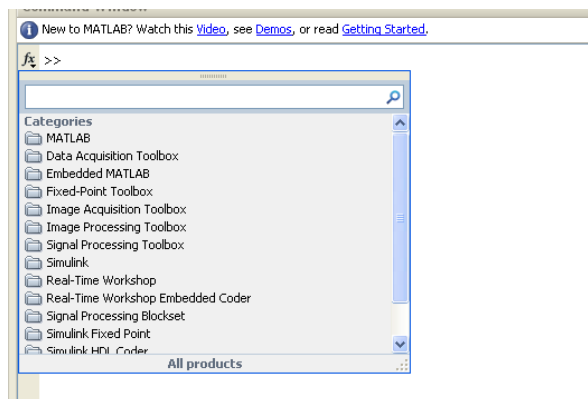
Además, en esta versión del programa, a medida que tecleamos una función la ayuda nos informa de los posibles argumentos de la misma, opciones, etc., como se muestra en la siguiente imagen:



y si tecleamos **More help** obtenemos más información sobre la función en cuestión:



En la barra lateral si **seleccionamos** **fx** obtenemos información sobre las distintas funciones que podemos usar agrupados según el tipo de aplicaciones:



En cualquier caso, siempre existe la opción de teclear **help** desde la ventana de comandos:

```
>> help
```

HELP topics:

```
toolbox\local - Local function library.  
matlab\datafun - Data analysis and Fourier transform functions.  
matlab\elfun - Elementary math functions.
```

```
toolbox\symbolic- Symbolic Math Toolbox.  
toolbox\sigsys - Signals and Systems Toolbox.
```

For more help on directory/topic, type "help topic".

Si se conoce el nombre del comando que se quiere utilizar

```
>> doc comando
```

presenta su sintaxis y una descripción detallada de lo que hace.

Si no se conoce el nombre del comando que realiza una determinada función, es muy útil el comando

```
>> lookfor cadena
```

que nos da una lista de los comandos que contienen una determinada cadena de caracteres en la primera línea del texto de ayuda asociado al mismo

VISUALIZACIÓN DE DATOS

MATLAB dispone de numerosas funciones para visualizar los resultados de nuestros cálculos. Permite representar funciones en dos y tres dimensiones (2-D y 3-D). Para obtener toda la información sobre las diferentes posibilidades lo mejor es acudir al apartado de “graphics” de la ayuda on-line.

En el caso 2-D, utilizaremos principalmente los comandos `stem` y `plot`. Para comprobarlo, prueba a ejecutar los siguientes comandos

```
» x = -10:10;  
» v = x.^2;  
» plot(v);
```

Como habrás visto el comando `plot(v)`, siendo `v` un vector, dibuja en una ventana gráfica los puntos que tienen como **coordenada y** los valores del vector `v`, y como **coordenada x** su posición en el vector `v`, uniéndolos mediante una línea. Si se usa esta función con dos vectores como argumento

```
» figure  
» plot( x, x.^2 )  
» figure  
» plot(v, x )
```

los puntos que se dibujan tienen como coordenada `x` los valores del primer vector y como coordenada `y` los del segundo. El comando `figure` se utiliza para abrir una nueva ventana gráfica, en vez de utilizar la que ya está abierta.

```
» plot( x, x.*sin(x) )  
» plot( x.*cos(x), x.*sin(x) )
```

Para visualizar **señales discretas** se utiliza la función `stem`. En ese caso los puntos no se unen entre sí, sino que cada punto está unido a su coordenada `x` correspondiente.

```
» figure  
» stem(v)
```

Se puede comprobar la importancia del vector de índices, comparando el resultado anterior con:

```
» figure  
» stem(x,v)
```

En la práctica, es más habitual utilizar la función `plot`, que es más potente que `stem`. Sin embargo, no debemos olvidar que esta función, por defecto, une los puntos mediante líneas y que, aunque la señal parece una señal en tiempo continuo, no lo es.

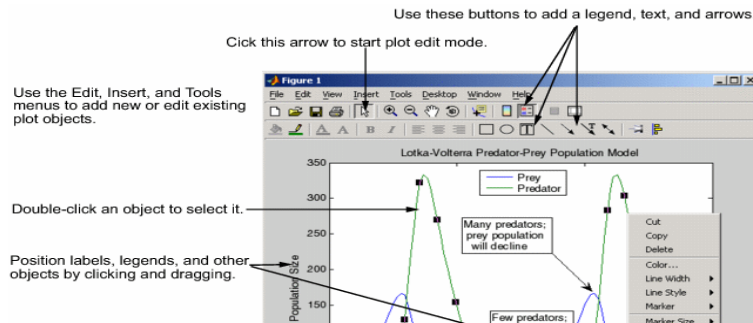
El comando `plot` permite asignar diferentes tipos a la línea trazada, seleccionar el color de la línea, o señalar los puntos dibujados con distintos símbolos. Ejecuta:

```
» figure, plot( x, v, 'r:x' );  
» figure, stem(v, 'g--o');
```

Se pueden rotular los gráficos utilizando los comandos `xlabel`, `ylabel`, `title`:

```
» xlabel('indice'); ylabel('valores de v'); title('grafica de prueba');
```

Estas operaciones se pueden realizar así mismo desde la misma **ventana gráfica**:



También se pueden representar funciones de 2 variables y para ello existen diferentes comandos. Prueba los siguientes:

```
>> [x y] = meshgrid(-3:.2:3, -3:.2:3);
>> z = x.^2 - y.^2;
>> mesh(x,y,z)
>> surf(x,y,z)
>> shading interp, colormap('copper')
>> contour(z)
>> imagesc(z)
```

Las posibilidades son muchas y no vamos a profundizar en ellas. Una información más completa se obtiene se obtiene tecleando

```
>> doc plot
```

Para **imprimir** un gráfico en esta versión de MATLAB para Windows, lo más fácil es usar el **menú** de la barra de menús de la ventana gráfica que queramos imprimir. Si tenemos una impresora conectada y correctamente configurada en Windows, no tendremos ningún problema para imprimir el contenido de la ventana.

USO DE MATLAB PARA GENERAR Y ALMACENAR SEÑALES

El tipo de datos (matrices) utilizado por MATLAB se presta de forma natural a la representación de señales discretas (simples, multicanales o multidimensionales).

Ahora bien, no hay que olvidar que:

- Sólo podemos almacenar señales de **longitud finita**.
- En MATLAB, los elementos de un vector de dimensión N se indican mediante un índice que va de **1 a N**. Mientras que una señal está formada por una serie de valores que corresponden a una variable independiente en un dominio que no tiene por que ir de 1 a N. Si conocer el índice es necesario en nuestra aplicación, habrá que almacenar la información correspondiente a la variable independiente en otro vector.

Además, la posibilidad que ofrece MATLAB de aplicar funciones escalares a vectores permite generar señales básicas sin necesidad de aplicar bucles **for**.

Ejemplo:

Generar la señal $x(n) = (n/2)^2 + 1$ para $-10 \leq n \leq 20$

```
nn = -10:20; %Generación del vector de índices
x = (nn/2).^2 +1; %Generación de la señal
```

Para visualizar señales discretas se utiliza la función **stem** como ya se ha señalado en el apartado anterior. Se puede comprobar la importancia del vector de índices, comparando el resultado de ejecutar:

```

stem(x)
y
stem(nn, x)

```

En la práctica, es más habitual utilizar la función `plot`, que es más potente que `stem`, aunque ya se han señalado las diferencias en el apartado anterior.

Por otra parte, si la señal que queremos almacenar está formada por muestras de una señal en tiempo continuo, el vector de índices se puede utilizar para guardar los valores del tiempo en que se han tomado las muestras.

Ej: Visualizar la señal $x(t) = \sin(2\pi ft + \pi/4)$ para $f=1\text{Hz}$ y $-1\text{s} \leq t \leq 1\text{s}$ muestreada a 50 Hz

```

tt = -1:1/50:1;
x = sin(2*pi*tt + pi/4);
plot(tt, x);
xlabel('t(s)'), ylabel('sin'), title('funcion sin muestreada a fs=50Hz');

```

Además, como ya hemos comentado, el comando `plot` permite asignar diferentes símbolos y colores a los puntos a representar, así como diferentes tipos de líneas para unirlos. Por ejemplo, ejecuta:

```

figure,
plot(nn, x, 'r:o');

```

Usa los comandos de la ventana para etiquetar los ejes, colorear, etc., y observa la diferencia.

GENERACIÓN DE SEÑALES BASICAS

Con las ideas anteriores trata de generar y visualizar las siguientes señales, y responde a las preguntas de la hoja de respuestas.

a) Impulsos

$$x_1(n) = 0.8\delta(n-5) \quad -10 \leq n \leq 20$$

$$x_2(n) = 5.4\delta(n+7) \quad -10 \leq n \leq 0$$

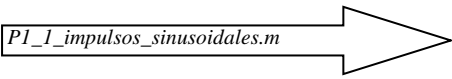
b) Sinusoidales

$$s_1(n) = \cos\left(\frac{30\pi}{23}n\right) \quad 0 \leq n \leq 50$$

$$s_2(n) = \sin\left(\frac{\pi}{17}n\right) \quad 0 \leq n \leq 50$$

$$s_3(n) = 0.8\cos\left(3\pi n + \frac{\pi}{2}\right) \quad -10 \leq n \leq 10$$

Pl_1_impulsos_sinusoidales.m



c) Exponenciales complejas

En MATLAB el manejo de expresiones complejas no presenta ninguna dificultad. Si al aplicar una función a un dato el resultado es complejo, MATLAB lo almacena como un valor complejo (parte real e imaginaria). Sólo hay que tener en cuenta lo siguiente:

- Para extraer las partes real e imaginaria de un n° complejo, MATLAB ofrece las funciones `real` e `imag`.

- Una función puede comportarse de manera diferente según su argumento sea real o complejo. Por ejemplo, las funciones `stem(n,z)` y `plot(n,z)` aplicadas a una señal z compleja con vector de índices n , visualizan sólo la parte real. Sin embargo, las funciones `stem(z)` y `plot(z)` dibujan la parte imaginaria en función de la real, es decir visualizan puntos en el plano complejo.

Ej: Generar y visualizar la señal:

$$x(n) = e^{j\frac{\pi}{4}n} \quad -2 \leq n \leq 16$$

```
nn = -2:16;
x = exp(j*pi/4*nn);
plot(nn, x)
figure, plot(nn,real(x), 'g',nn,imag(x), 'r')
figure, plot(x)
axis equal
figure, plot(x,'o'), axis equal
```

Con estas ideas, dada las señales asociadas al conjunto de armónicos de periodo 15:

$$s_k(n) = e^{j\frac{2\pi k}{15}n} \quad k = 0, \dots, 14$$

genera y visualiza los armónicos de orden 4, 5 y 11 (sus partes real e imaginaria en gráficas por separado). Y responde a las preguntas de la hoja de respuestas.



CARGA DE SEÑALES

Además de generarlas tecleando, o mediante funciones MATLAB, las señales se pueden cargar del exterior desde un fichero. Para ello se utiliza el comando `load` que puede cargar datos desde un fichero:

- en formato `.mat` (previamente salvado desde MATLAB usando el comando `save`)
- en formato `ASCII` generado por un editor o un programa no MATLAB.

Existe también la posibilidad de utilizar las funciones de entrada-salida de bajo nivel de que dispone MATLAB para leer los datos de un fichero. Estas funciones (`fopen`, `fread`, `fscanf`, ...) recuerdan a las del lenguaje C, pero no las utilizaremos en esta asignatura.

Como ejemplo, puedes cargar desde el directorio de trabajo dos señales sonoras que se encuentran almacenadas en dos ficheros.

La primera de ellas se encuentra en un fichero llamado `whale.dat` que contiene en formato `ascii` (puedes leerlo con `notepad`) un fragmento de sonido emitido por una ballena (11542 muestras a 22050Hz). Carga los datos con el comando

```
load whale.dat
```

y trata de visualizar la señal y de escucharla mediante el comando `soundsc` (lee primero la ayuda de este comando para ver la influencia de la frecuencia de muestreo).

La segunda se encuentra en un fichero `e101.wav` que es un formato estándar de Windows. Puedes leerlo mediante la función `wavread` o `audioread` de MATLAB que está escrita mediante funciones de bajo nivel para poder acceder al fichero en el formato adecuado. Lee la ayuda para leer dicha señal y escúchala mediante el comando `soundsc`.

```
[frase,fs] = audioread('e101.wav');  
soundsc (frase,fs);
```

MATLAB permite también el manejo de señales multicanal y bidimensionales de forma sencilla, utilizando matrices.

Señales multicanal

En realidad, no son más que un conjunto de señales individuales. Se suelen almacenar en una matriz de forma que **cada señal ocupe una columna** (conviene que sea así y no por filas porque las funciones de MATLAB se generalizan al caso multicanal siguiendo este criterio).

Como ejemplo de una señal de este tipo puedes cargar los datos contenidos en el fichero `sunspot.dat` que contiene datos del número medio mensual de manchas solares desde el año 1600. Al hacerlo obtendrás una matriz de 13 columnas. La primera indica el año y sirve de índice. Las otras 12 corresponden cada una a un mes del año.

Señales multidimensionales

El ejemplo más habitual de señal multidimensional es el de las imágenes, que en MATLAB se almacenan en forma de matriz y se visualizan mediante el comando `image`.

Prueba a ejecutar los siguientes comandos y entender lo que hacen:

```
load earth  
image(X)  
colormap(map)  
axis image
```

PROGRAMACIÓN EN MATLAB

Lo que hemos visto hasta ahora convierte a MATLAB en una potente calculadora, pero lo que la convierte en una herramienta verdaderamente útil es la posibilidad de crear nuevas funciones o programas, y poder almacenarlas para su posterior utilización. Para ello MATLAB permite, por un lado, **almacenar conjuntos de comandos en ficheros** para su posterior utilización y, por otro, disponer de algunos comandos básicos que permiten ejecutar **bucles y ejecuciones condicionales con expresiones lógicas y relacionales**. La sintaxis utilizada en estas estructuras es muy similar a la utilizada en C. En la ayuda on-line se encuentra un apartado dedicado a esta programación denominado “*programming*”. Consúltala.

A continuación, hacemos un pequeño resumen de los comandos más importantes:

FICHEROS .M: SCRIPTS Y FUNCIONES

Una serie de comandos MATLAB se pueden escribir en un fichero con cualquier editor (por ejemplo, el **notepad** de Windows). La única condición para su posterior uso desde MATLAB es que el nombre del fichero tenga la extensión “**.m**”. A estos conjuntos de comandos se les denomina “scripts” o ficheros .m. Para ejecutarlos desde MATLAB, basta teclear el nombre del fichero (sin extensión) como si fuera un comando. Es necesario que MATLAB pueda encontrar el fichero .m y para ello hay que tener en cuenta que MATLAB busca los ficheros en su **directorio de trabajo** (que se puede modificar con el comando **cd** o en la ventana superior de **Current Directory**) y en una lista de directorios (que se puede modificar con el comando **path**). En las sesiones de laboratorio **deberías empezar siempre por seleccionar el directorio deseado**.

La versión de MATLAB que estamos utilizando dispone de un **editor/debugger** propio que utilizaremos porque incluye ayudas a la edición y depurado de programas. Para probarlo arráncalo desde la barra de menús de la ventana de comandos **File/New/M-file**, y edita los comandos siguientes:

```
[x y] = meshgrid(-3:.2:3, -3:.2:3);  
z = x.^2 - y.^2;  
mesh(x,y,z);
```

guarda el fichero con el nombre **prueba.m** en la carpeta de trabajo, y desde la ventana de comandos ejecuta

```
> prueba
```

El resultado es el mismo que hubieras obtenido si hubieras tecleado directamente los tres comandos desde MATLAB.

Las **funciones** son **como los “scripts”, pero permiten el paso de argumentos** y además son compiladas la primera vez que se usan en una sesión dada para mejorar la velocidad.

Crea un fichero llamado **gencos.m**, con las siguientes líneas:

```
function y = gencos (A, F, fi, ni, nf)  
%GENCOS genera una sinusoidal: A*cos(2*pi*F*n + fi)  
% uso: y = gencos (A, F, fi, ni, nf)  
% A amplitud  
% F frecuencia en ciclos por muestra  
% fi desfase  
% ni valor inicial del índice  
% nf valor final del índice  
% y señal de salida  
  
nn = ni:nf;  
y = A*cos(2*pi*F*nn + fi);
```

y después de salvarlo, ejecuta desde MATLAB los siguientes comandos


```

» y1= gencos(1.5, 0.1, pi/6, -10, 50);
» figure, plot(y1)
» y2 = gencos(0.5, 0.01, 0, -100, 500);
» figure, plot(y2)

```

Como puedes ver la función, no es mas que un fichero .m, que comienza con el comando **function** que sirve para especificar los argumentos de entrada y salida. Las líneas que empiezan con % son comentarios

Prueba ahora el comando

```

» help gencos

```

¿Qué ocurre? Es como si la función que acabamos de crear, fuera un comando más de MATLAB. ...déjame pensar ... ¿y las demás funciones? ¿Serán también ficheros .m? Prueba a editar alguno, por ejemplo C:\Program Files\MATLAB\R2017a\toolbox\matlab\datafun\mean.m.

BUCLES Y ESTRUCTURAS CONDICIONALES

Hay tres comandos en MATLAB que permiten realizar bucles y sentencias condicionadas. Son los comandos **for**, **while** y **if-else**. Son muy similares a los de cualquier otro lenguaje de programación y los vamos a ver con ejemplos.

for

Por ejemplo, los comandos:

```

n = 10;
x = [];
y = [];
for i = 1:n
    x = [x,i^2]
    y = [y,i^3]
end

```

Guarda el fichero con el nombre **prueba2.m** en la carpeta de trabajo, y desde la ventana de comandos ejecuta

```

» prueba2

```

genera un vector **x** con el cuadrado de los 10 primeros enteros y un vector **y** con los cubos. Fíjate que un vector puede estar inicialmente vacío (**x=[]**). Lo que hace el comando es ejecutar los dos comandos que hay en el bucle, para todos los valores de la variable **i** calculados en la expresión 1:10.

Prueba ahora el **debugger**, ejecutando desde la ventana del editor los comandos **set/clear** **breakpoints**, **step**, ...

Otro ejemplo con doble bucle anidado

```

for i = 1:5
    for j = 1:4
        H(i, j) = 1/(i+j-1);
    end
end
H

```

produce una matriz de Hilbert.

En el siguiente ejemplo se ejecuta el comando (**n = n + 1**) repetidamente hasta que deja de cumplirse la expresión (**2^n < a**) y luego se escribe **n**. Es decir, para una variable **a** definida anteriormente se calcula el número natural mínimo que cumple **2^n >= a**.

```

while

```

```
n = 0;
while 2^n < a
    n = n + 1;
end
n
```

En el siguiente ejemplo para un valor de `n` dado, se asignará a la variable `parity`, 0 si `n` es negativa, 2 si es par y 1 si es impar.

```
if ~ else
    if n < 0
        parity = 0;
    elseif rem(n,2) == 0
        parity = 2;
    else
        parity = 1;
    end
end
```

Por supuesto, los comandos `elseif` y `else` son optativos.

Cuando en una condición utilizada en un bloque `while` o `if` el resultado es una matriz, la expresión se considera cierta sólo si todos los elementos de la matriz son distintos de 0. Así para ejecutar un comando cuando dos matrices `A` y `B` son iguales basta escribir:

```
if A == B
    statement
end
```

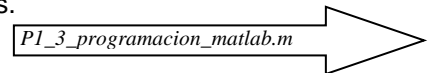
pero si lo que queremos es ejecutar una sentencia cuando `A` y `B` son distintas, habrá que escribir

```
if A == B else
    statement
end
```

De todos modos, es habitual que los principiantes utilicen más bucles de los necesarios, y esto ralentiza el funcionamiento de MATLAB. Como hemos dicho antes, el uso adecuado de el operador ":" puede evitar en muchas ocasiones gran cantidad de código.

Realiza los ejercicios propuestos en la parte final del documento de resultados.

PI_3_programacion_matlab.m



AYUDAS A LA PROGRAMACIÓN

Con lo que hemos visto es suficiente para que sigas avanzando. Como ya te habrás dado cuenta, es muy fácil crear funciones nuevas y probarlas, pero además del debugger que anteriormente hemos comentado, MATLAB incluye algunos comandos enfocados a la depuración de tus programas. Estos comandos permiten parar la ejecución en puntos concretos, examinar el espacio de trabajo y avanzar paso a paso. Vamos a introducirlos brevemente.

El primer comando es

```
> pause
```

que detiene la ejecución de un fichero `.m` hasta que se pulsa una tecla cualquiera. `pause(n)` detiene la ejecución `n` segundos y `pause(-2)` hace que el programa ignore los siguientes pauses.

Para ver o no los comandos que se ejecutan se utiliza el comando

```
> echo
> echo on
> echo off
```

Para parar la ejecución y pasar el control al teclado se usa **Ctrl+C**

Por último, para introducir parámetros durante la ejecución y desde el teclado, puedes usar el comando:

```
» x=input('prompt')
```

que presenta en pantalla el texto utilizado como argumento y carga en la variable x el valor tecleado.

De todos modos, aunque estos comandos son útiles y existen desde las primeras versiones de windows, en las versiones actuales puede ser más práctico utilizar el `debugger` ya mencionado que incorpora el editor. Con él podemos insertar “breakpoints” en el programa y examinar el valor de todas las variables en tiempo de ejecución.

En esta práctica hemos presentado un pequeño y rápido resumen de las posibilidades que brinda MATLAB. Habéis aprendido el mínimo necesario para empezar a trabajar.