

# Programación en Matlab

## **Introducción a Matlab**

Ing. Eddie Ángel  
Sobrado Malpartida

## Introducción

MATLAB es un lenguaje de programación de alto nivel que permite realizar cálculos complejos, visualizar resultados, análisis de datos y cálculo numérico. y desarrollar algoritmos utilizando notación matemática con bastante naturalidad. El nombre **MATLAB** proviene de **Matrix Laboratory** y fue creado en el año 1984 por la empresa Mathworks. Con MATLAB, podrá resolver problemas de cálculo técnico más rápidamente que con lenguajes de programación tradicionales, tales como C, C++ y FORTRAN.

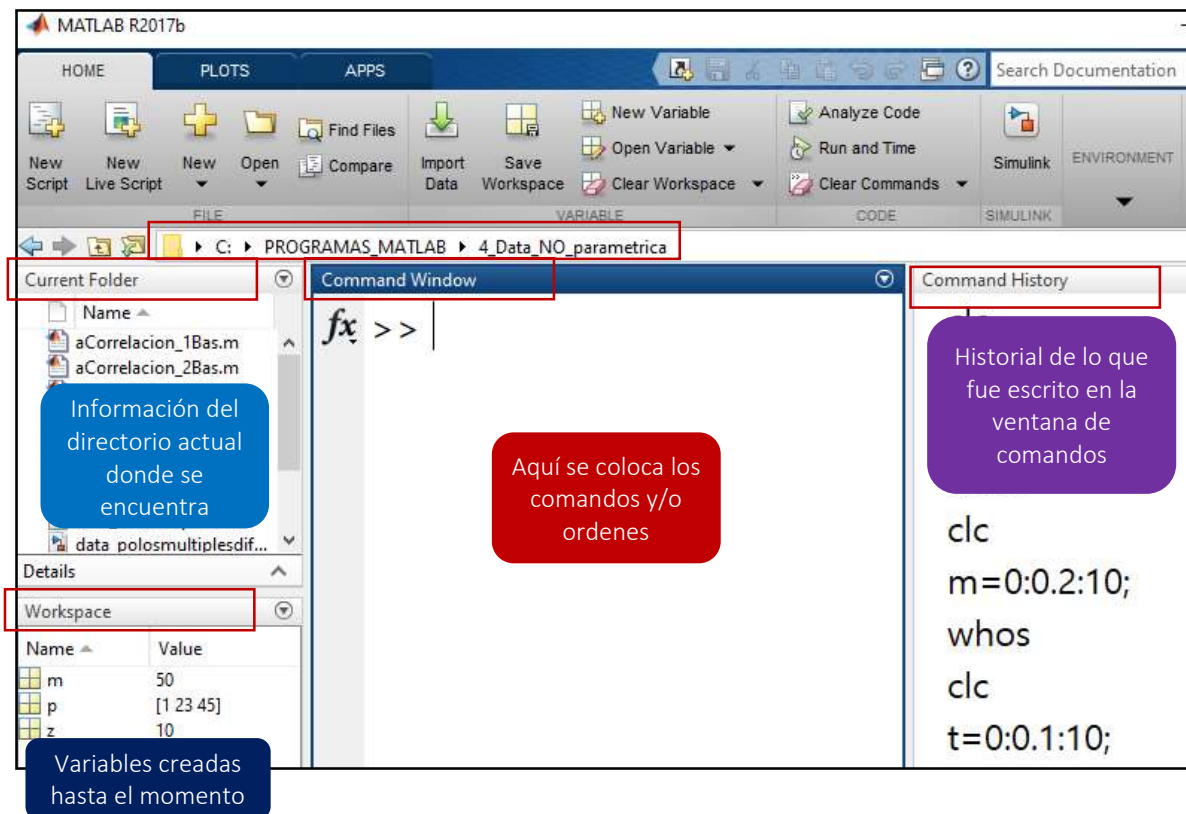
MATLAB trabaja intensivamente con matrices, simplificando de manera notable su uso, se emplea con frecuencia para resolver problemas de computación científica, procesamiento de imágenes comunicaciones, diseño de sistemas de control, sistemas de prueba y medición, modelado y análisis financiero y biología computacional y en control avanzado. En la actualidad MATLAB permite adicionalmente conectarse con otros programas, hacer adquisición de datos, control en tiempo real, procesamiento simbólico, entre otros. En esta sección se va a presentar un breve tutorial con el objeto de aprender los fundamentos de programación con MATLAB y poder utilizarlo para resolver problemas de control clásico.

## Conociendo el entorno de Matlab

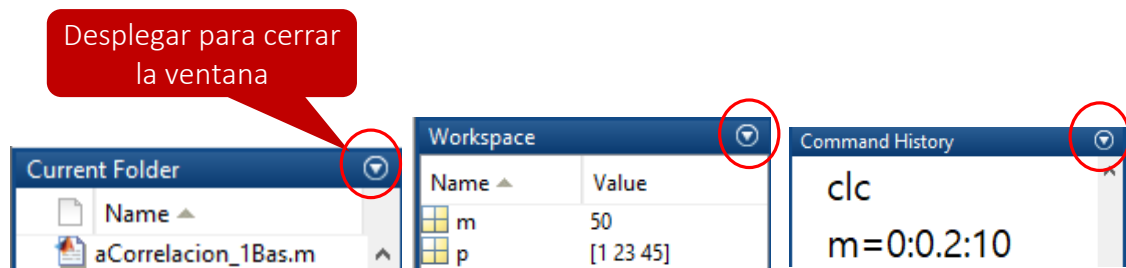
Al iniciar MATLAB, se le presenta al usuario un entorno de trabajo como el que se muestra en la Figura; la presentación por defecto es la que se muestra, salvo que previamente haya realizado alguna configuración para que no aparezcan todas las ventanas.

En este entorno se destaca la ventana indicada **Command Window**, en la cual se pueden ingresar **comandos** y/o **funciones** para obtener resultados de manera directa. Además se observa la ventana **Workspace** donde se pueden encontrar todas las variables definidas por el usuario.

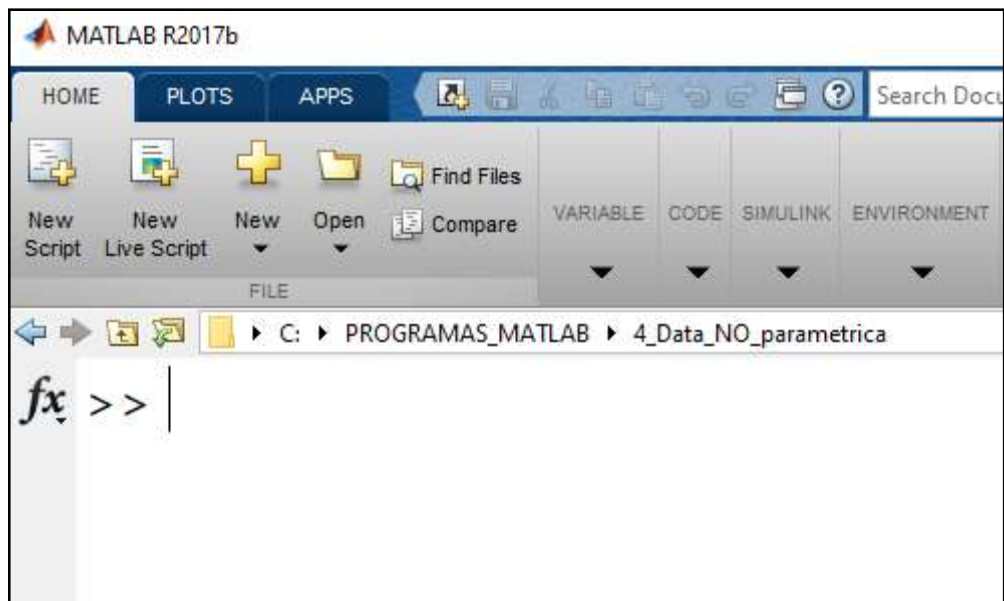
También se muestra una ventana **Command History** donde se va registrando como un historial de todas las ordenes y/o comandos que se escriben en la **Command Window**. Por último, se observa la ventana donde indica el directorio actual en el que se encuentra ubicado usted como usuario.



Si desea puede cerrar todas las ventanas y dejar solamente la vista de la **Command Window** en donde se podrá iniciar algunas pruebas



Finalmente, la ventana quedaría como la que se muestra



### Comandos y operaciones en Matlab

En lo sucesivo, todo lo escribiremos en la ventana de comando (**Command Window**)

Como en otros lenguajes de programación, MATLAB provee expresiones matemáticas, pero a diferencia de otros lenguajes, estas **expresiones se relacionan con matrices**. Los grupos a partir de los cuales se pueden crear expresiones son los siguientes:

#### *Definición de variables numéricas escalares*

MATLAB permite trabajar con números de tipo entero con y sin signo, así como punto flotante de doble precisión, el cual es la **condición por defecto** al crear una variable. Para crear una variable escalar, sólo se le debe asignar un valor a un nombre:

```
>> var1 = 3.14  
  
var1 =  
  
    3.1400
```

Al elegir un nombre para una variable se debe tener en cuenta que debe **comenzar** con una **letra** y **que se distinguen mayúsculas de minúsculas**. Una vez que ha sido asignada con un valor, ésta se muestra en el **Workspace** y puede ser utilizada para realizar operaciones:

```
>> var2 = 1.3*45+2*var1  
  
var2 =  
  
    64.7800
```

Si desea puede colocar comandos como: **whos**, que permite ver las variables que existen en memoria de Matlab (**workspace**):

```
>> whos  
Name      Size      Bytes  Class  Attributes  
var1      1x1         8  double  
var2      1x1         8  double
```

También puede limpiar la pantalla con:

```
>>clc
```

Si desea eliminar las variables de memoria de Matlab (**workspace**) use **clear all** y luego consulte nuevamente con **whos** y verá que ya no existen variables en memoria:

```
>>clear all  
>>whos
```

Como se observa, se han utilizado operaciones aritméticas como sumas y multiplicaciones de manera bastante sencilla para definir la variable **var2**.

Si **no** se desea **mostrar** algún **resultado o confirmación de orden**, se debe finalizar la línea respectiva con un **punto y coma**:

```
>> var1 = 3.14;  
>> var2 = 1.3*45+2*var1;  
>>numero_estudiantes = 7;  
>>volumen_tanque16 = 135;
```

### *Definición de variables numéricas vectorial y matricial*

Para definir vectores, se debe tener en cuenta si se trata de un vector fila o columna, dado que MATLAB los trata de manera diferente. Para definir **vectores fila** se escriben valores separados por espacios o comas encerrados entre corchetes:

```
>> filaA = [1 2 5.4 -6.6]  
  
filaA =  
  
    1.0000    2.0000    5.4000   -6.6000
```

Mientras que para definir un **vector columna**, los valores deben estar separados por punto y coma encerrados siempre entre corchetes:

```
>> columnaA = [4 ; 2 ; 7 ; 4]  
  
columnaA =  
  
     4  
     2  
     7  
     4
```

Las matrices se pueden definir usando las reglas para los vectores, esto es, especificando cada elemento de manera individual y utilizando el **punto y coma** para escribir **filas distintas** (tener siempre en consideración las dimensiones de los vectores).

```
N = [3 -99 0.001; 9.639 1.60210e-2 6.02252e3; 2 -5 3.4]  
N =  
  
    1.0e+03 *  
  
    0.0030    -0.0990    0.0000  
    0.0096     0.0000    6.0225  
    0.0020    -0.0050    0.0034
```

### Operadores y Funciones

Matlab tiene una gran cantidad de [funciones matemáticas](#) ya definidas que pueden ser empleadas por el usuario, por ejemplo la función [sin\(\)](#) y [cos\(\)](#). Además tiene una serie de [operadores](#) además de la [+](#), [-](#), [\\*](#), por ejemplo [^](#), [:](#), [\](#), etc.

```
ang = 60;  
Y = (sin(ang*pi/180))^2 + (cos(ang*pi/180))^2  
Y =  
  
1
```

### Operador ':'

Usemos ahora un operador bastante empleado: el operador [dos puntos](#) ':'

Por ejemplo, para crear un vector con incrementos de 1, podemos colocar el formato:

*Variable = valor\_inicial : valor\_final*

```
>> t=0:10  
  
t =  
  
0    1    2    3    4    5    6    7    8    9   10  
  
>>
```

Si deseamos tener un incremento diferente a uno usaremos el formato:

*Variable = valor\_inicial : incremento : valor\_final*

```
>> t=0:0.1:0.5  
  
t =  
  
0    0.1000    0.2000    0.3000    0.4000    0.5000
```

Luego podemos aplicar la función seno a todos los componentes del vector t:

```
>> amplitud=sin(t)  
  
amplitud =  
  
0    0.0998    0.1987    0.2955    0.3894    0.4794
```

## Creando un programa en Script

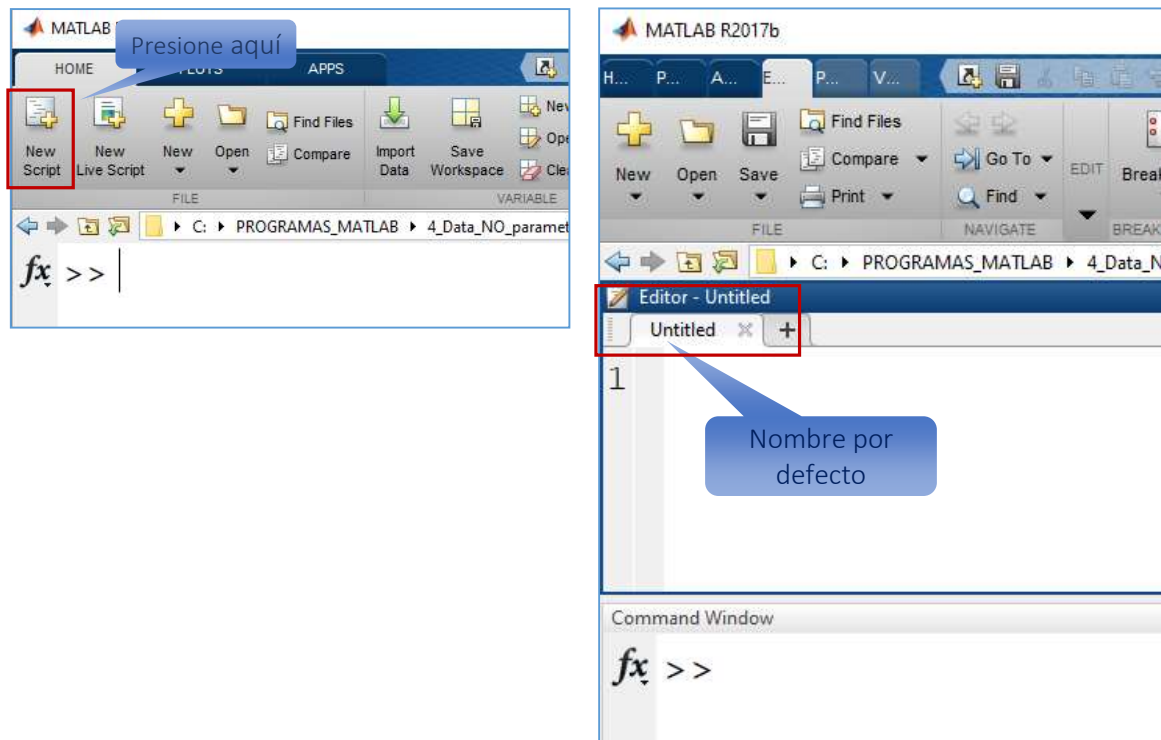
MATLAB es un potente lenguaje de programación, así como también un entorno computacional interactivo. Los archivos que contienen código en lenguaje MATLAB son llamados archivos M (en inglés M-files). Para crear un archivo-M se puede hacer uso de cualquier editor de texto, luego será posible usarlos como cualquier función o comando de MATLAB.

Hay dos tipos de archivos-M

- **Scripts**, estos archivos-M no aceptan argumentos de entrada o retornan argumentos de salida. Estos operan con datos del espacio de trabajo (Workspace).
- **Functions**, este tipo de archivos si aceptan argumentos de entrada y retornan argumentos de salida. Operan con variables internas de la función.

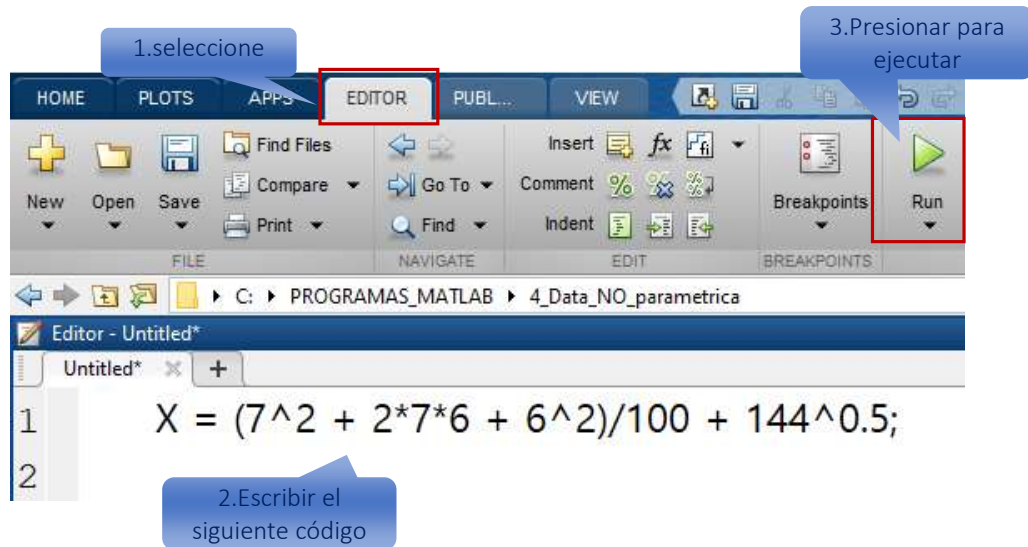
**Scripts:** Cuando se invoca un *script*, MATLAB simplemente ejecuta los comandos que se encuentran dentro del archivo. Los *scripts* pueden operar con los datos que existen en el espacio de trabajo (workspace) o también pueden crear nuevos datos durante su ejecución. Sin embargo, no retornan ningún argumento. Todas las variables que son creadas se mantendrán en el espacio de trabajo y estarán disponibles para cualquier cálculo siguiente.

Crearemos un programa que incluirá lo visto previamente, es decir **variables**, **operadores**, **matrices** o **vectores**, **funciones**, etc. Para ello crearemos un archivo **script** como se muestra a continuación:



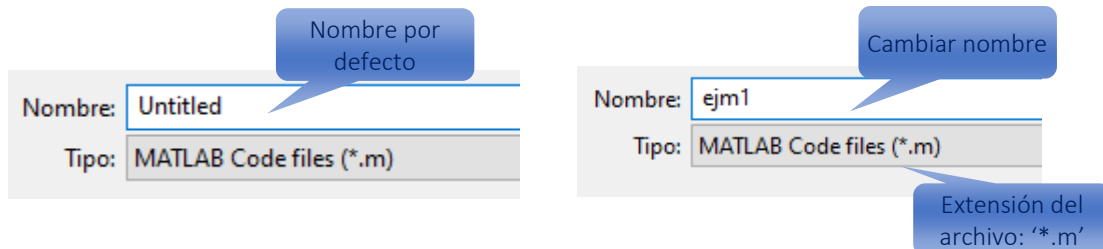


Editamos el siguiente programa y lo ejecutamos:

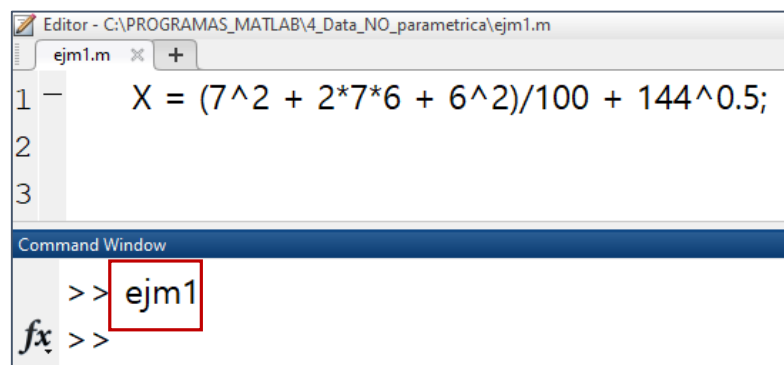


Si aun no hemos asignado el nombre al archivo y lo ejecutamos la primera vez, nos pedirá ingresar un nombre del archivo. Evitar colocar un nombre que inicio con un número y no debe dejar espacios en blanco en el nombre del archivo.

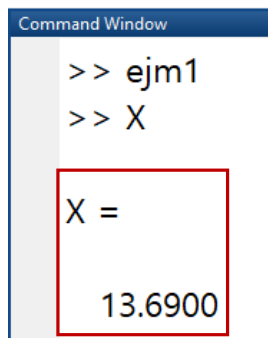
Cambiar por ejemplo el nombre a `ejm1`, la extensión por defecto es `*.m`



Al ejecutar el programa se mostrará en la **ventana de comandos** el nombre del archivo ejecutado:



Si desea ver el valor de la variable solo debo colocar el nombre de la variable en la línea de comandos:



```
Command Window
>> ejm1
>> X

X =

    13.6900
```

### Uso de comentarios (%)

Para [insertar comentarios](#) se emplea el símbolo '%'. En el siguiente ejemplo podríamos colocar comentarios que nos ayude recordar.

Podemos agregar lo siguiente y experimentar que realiza cada uno de ellos:

```
X = (7^2 + 2*7*6 + 6^2)/100 + 144^0.5;

t1 = 4^2                %Potencia: 4^2 = 16
t2 = (3+4*j)^0.1        %Ordena con paréntesis
t3 = sqrt(2)            %Raíz cuadrada: 2 = 2^0.5
t4 = [log(2) log10(0.23)] %Logaritmo: [ln(2) log10(0.23)]
t5 = [cos(pi/6) ; atan(-0.8)] % cos() en radianes, pi = 3.1416...
t6 = exp(2+4*i)          %Exponencial: e(2+4i)
t7 = [round(1.4), floor(3.3), ceil(4.23)] %Redondeos
t8 = [angle(i) abs(1+i)] %Operaciones sobre complejos:
                        %argumento y norma
```

Como [no se ha colocado](#) punto y coma al final de cada operación, los resultados se verán en la ventana de comandos cuando se ejecute el archivo.

Al final, se ha hecho uso de la variable especial **"i"**, la cual [está definida](#) en MATLAB como la base de los **números imaginarios**.

### Generación de matrices con funciones de Matlab

Existen funciones de Matlab que generan matrices cuyos elementos son cero, aleatorios, la unidad, etc. Para ello probemos lo siguiente

```
C = zeros(3,4)      %Todos ceros
D = 3*ones(4,4)     %Todos unos multiplicado por 3
E = rand(2,2)       %Elementos aleatorios uniformemente distribuidos
F = randn(3,3)      %Elementos aleatorios normalmente distribuidos
```

### Graficas en Matlab

MATLAB ofrece muchas facilidades para mostrar vectores y matrices como gráficas. En esta sección se verán algunas de las más importantes funciones para generar graficar a partir de matrices.

Una de las funciones más usadas para crear un grafica es [plot](#). Esta función tiene diferentes formas, dependiendo de los parámetros de entrada.

Crearemos otro archivo script: [ejm2.m](#) y realizamos el siguiente programa:

```
clc;
close all;
clear all
t=0:0.1:10
amplitud1=sin(t);
amplitud2=cos(t);
plot(t,amplitud1)
grid
figure
plot(t,amplitud2)
grid
```

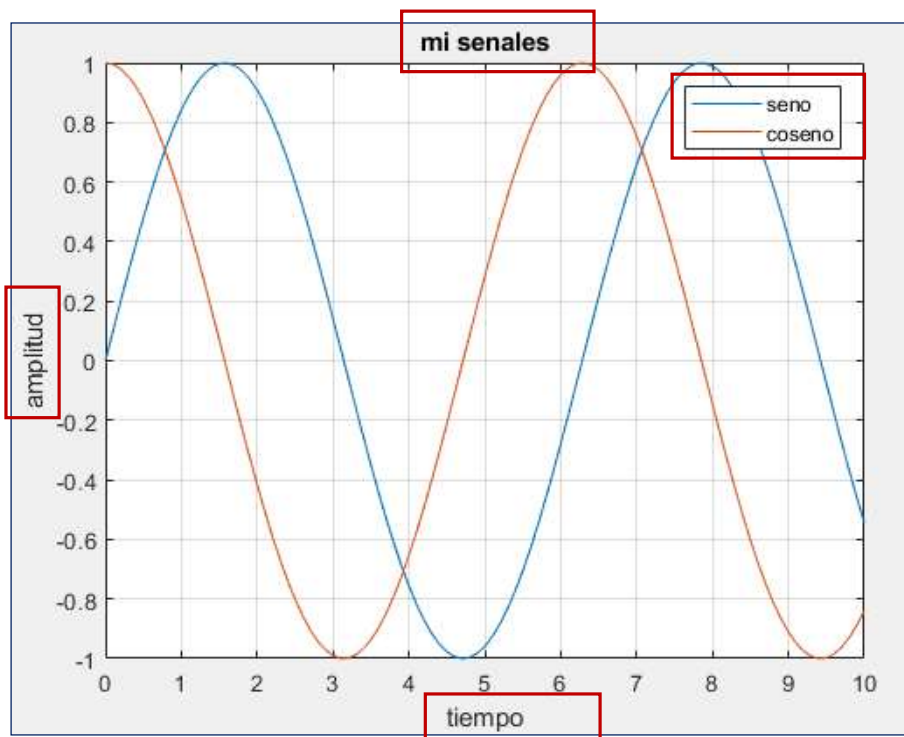
Otra forma de obtener las gráficas seria:

```
clc;
close all;
clear all
t=0:0.1:10
amplitud1=sin(t);
amplitud2=cos(t);
plot(t,amplitud1, t,amplitud2)
grid
```

El etiquetado de los ejes y título de la gráfica se hace usando las funciones [xlabel](#), [ylabel](#) y [title](#) respectivamente.

```
clc;
close all;
clear all
t=0:0.1:10
amplitud1=sin(t);
amplitud2=cos(t);
plot(t,amplitud1, t,amplitud2)
grid
%-----
title('mi senales')
xlabel('tiempo')
ylabel('amplitud')
legend('seno', 'coseno')
```

El resultado en este caso será:



Otra forma de 'plotear' más de una gráfica en la misma figura es usando el comando **hold**

```
clc;
close all;
clear all
t=0:0.1:10
amplitud1=sin(t);
amplitud2=cos(t);
plot(t,amplitud1)
hold on
plot(t,amplitud2)
grid
```

Finalmente realice un programa más:

```
y = 4*exp(-t/3).*sin(2*t); % Define funciones a graficar
z = 2*exp(-t/5).*cos(t);
plot (t, y, 'o', t, z, 'x') % plotea con opciones de formato
grid                        % Habilita líneas de grilla
title('Probando Plot...', 'FontSize', 24) % Título de la gráfica, tamaño
                                         de letra 24
xlabel('Segundos') % Etiqueta para el eje X
ylabel('Amplitud') % Etiqueta para el eje Y
legend('y=4*exp(-t/3)*sin(2*t)', '2*exp(-t/5)*cos(t)'); % Leyenda
```

## Comandos más usados en Ingeniería de Control

Comando	Descripción
acker	Compute the K matrix to place the poles of A-BK, see also place
bode	Draw the Bode plot, see also logspace.
c2dm	Continuous system to discrete system
conv	Convolution (useful for multiplying polynomials), see also deconv
ctrb	The controllability matrix
det	Find the determinant of a matrix
eig	Compute the eigenvalues of a matrix
feedback	Feedback connection of two systems.
impulse	Impulse response of continuous-time linear systems, see also step, lsim, dlsim
lqr	Linear quadratic regulator design for continuous systems, see also dlqr
lsim	Simulate a linear system, see also step, impulse, dlsim.
margin	Returns the gain margin, phase margin, and crossover frequencies, see also bode
obsv	The observability matrix, see also ctrb
ones	Returns a vector or matrix of ones, see also zeros
place	Compute the K matrix to place the poles of A-BK, see also acker
poly	Returns the characteristic polynomial
pzmap	Pole-zero map of linear systems
Rank	Find the number of linearly independent rows or columns of a matrix
Real	Returns the real part of a complex number, see also imag
rlocfind	Find the value of k and the poles at the selected point
rlocus	Draw the root locus
roots	Find the roots of a polynomial
rscale	Find the scale factor for a full-state feedback system
series	Series interconnection of Linear time-independent systems
sgrid	Generate grid lines of constant damping ratio (zeta) and natural frequency (Wn), see also jgrid, sigrid, zgrid
sigrid	Generate grid lines of constant settling time (sigma), see also jgrid, sgrid, zgrid
size	Gives the dimension of a vector or matrix, see also length
sqrt	Square root
ss	Create state-space models or convert LTI model to state space, see also tf
ss2tf	State-space to transfer function representation, see also tf2ss
ss2zp	State-space to pole-zero representation, see also zp2ss
step	Plot the step response, see also impulse, lsim, dlsim.
tf	Creation of transfer functions or conversion to transfer function, see also ss
tf2ss	Transfer function to state-space representation, see also ss2tf
tf2zp	Transfer function to Pole-zero representation, see also zp2tf
zgrid	Generates grid lines of constant damping ratio (zeta) and natural frequency (Wn), see also sgrid, jgrid, sigrid
zp2ss	Pole-zero to state-space representation, see also ss2zp
zp2tf	Pole-zero to transfer function representation, see also tf2zp