



# Señales y Sistemas



## Introducción

Matlab es un sistema interactivo y lenguaje de programación para cómputo científico y técnico en general. Su elemento básico es una matriz. Dado que los comandos de Matlab son similares a la expresión de los pasos de ingeniería en matemáticas, escribir soluciones en computadora con Matlab es mucho más fácil que usar un lenguaje de alto nivel como C o Fortran.

## Los archivos .m en Matlab

Para problemas simples, como algunos de los que hemos tratado introducir cada una de las sentencias necesarias de Matlab puede ser rápido y eficiente. Cuando el problema posee mucha cantidad de variables y necesita una gran cantidad de sentencias, este procedimiento puede ser tedioso e ineficiente.

Para este problema, Matlab representa una solución lógica. Permite colocar sentencias en un simple archivo de texto, pedirle que abra este archivo y ejecute las sentencias escritas en él, exactamente como si hubiesen sido escritas en el espacio de trabajo. Estos archivos se llaman archivos script o archivos .m. El término .m significa que la extensión del archivo debe ser .m para que Matlab lo reconozca y pueda ejecutarlo. La extensión del archivo sirve para que los distintos programas asociados con el archivo lo reconozcan.

**Los pasos para crear un archivo .m son muy simples:** Vaya al menú FILE, seleccione New y luego seleccione .m. Aparecerá una ventana donde Ud. podrá escribir las sentencias de Matlab, una vez escritas grabe el archivo con el nombre de su preferencia sin dejar espacio entre caracteres pero siempre con la extensión .m. Para ejecutar el archivo desde la ventana de ordenes, solamente ponga el nombre del archivo después de ». Así por ejemplo si Ud. creo un archivo con sentencias de Matlab llamado practico\_1.m, vaya a la ventana de ordenes y escriba:

```
» practico_1
```

y el archivo se ejecutará.

Un inconveniente que puede tener es que el directorio donde Matlab está buscando el archivo a ejecutar sea distinto a donde Ud. grabó el archivo .m. Esto se soluciona con la sentencia cd. Por ejemplo si grabó el archivo practico\_1.m en el directorio c:/ señales\_y\_sistemas/, entonces antes de intentar ejecutar practico\_1.m deberá escribir

```
» cd c:/señales_y_sistemas %esta orden cambia el directorio de Matlab a c:/señales_y_sistmas
```

Otra forma de acceder a este directorio, se puede lograr seteando la ruta de acceso. Para esto, vaya al menú File, y seleccione set path, luego seleccione el directorio correspondiente y luego salve el path desde el menú File. También puede ocurrir que Matlab no encuentre el archivo.m debido que el nombre del archivo o el nombre del directorio de trabajo contiene un espacio.



## Introducción de Matrices en Matlab

Primero crear un archivo con nombre tutorial.m en su directorio de trabajo de Matlab. Cada vez que crea un archivo.m en la primera línea coloque las siguientes sentencias:

```
clear all, close all, clc %elimina todas la variables usadas, cierro
                          %todas la ventanas, limpio la ventana de
                          %comando.
```

Los vectores, los cuales son matrices de  $1 \times n$  o  $n \times 1$ , se utilizan de forma normal para guardar señales de datos muestreados en una dimensión, o secuencias. Una cuestión importante es la asociación que realiza MATLAB de los índices del dominio temporal. *Matlab numera los índices de los vectores de 1 a N, siendo N la longitud del vector.* Una manera de introducir una secuencia en Matlab es introducirla mediante una lista explícita de algoritmo de elementos. Obsérvese que los elementos deben estar separados por espacios en blanco o por comas como sigue:

```
x=[1 2 3 4 5]
```

o

```
x=[1,2,3,4,5]
```

La sentencia

```
x=[1 2 3 4 5]
```

crea una única secuencia de cinco elementos reales en un vector fila. La secuencia se puede pasar a vector columna trasponiéndola. Es decir.

```
y=x'
```

y =

```
1
2
3
4
5
```

Anteriormente introducimos los valores de x escribiendo cada elemento individual en x. Aunque esto funciona bien cuando hay solo 5 valores de x, ¿qué sucede si hay 112 valores? Utilizando la notación de dos puntos otras dos formas de introducir x son:

```
»x=(0:1:5)
```

x =

```
0    1    2    3    4    5
```

```
»x=linspace(0,5,5)
```

x =

```
0    1.2500    2.5000    3.7500    5.0000
```

En el primer caso la notación dos puntos comienzo:incremento:fin; (0:1:5) crea un array que comienza en 0, incrementando por 1 y finaliza en 5. En el segundo caso, la función linspace de MATLAB se utiliza para crear x. Los argumentos de esta función se describen mediante

```
linspace(primer_valor,último_valor_números_de_valores)
```

La creación de ambas formas de arrays son comunes en MATLAB. La forma de notación de dos puntos permite especificar directamente el incremento entre datos, pero no el número de ellos. linspace, por otra parte, permite especificar directamente el número de datos, pero no el incremento entre ellos.



## Matemática con arrays

La suma, la resta, multiplicación y división por un escalar simplemente aplica la operación a todos los elementos del array:

```
»x=(0:1:5)*2
x =
    0    2    4    6    8   10
»x=(0:1:5)-2
x =
   -2   -1    0    1    2    3
```

Las operaciones de arrays entre arrays de diferentes longitudes son difíciles de definir e incluso de valor algo más dudoso. Sin embargo, cuando dos arrays tienen la misma longitud, la suma, la resta, la multiplicación y la división se aplican sobre la base de elemento-a-elemento. Por ejemplo:

```
» x= [1  2  3  4  5];
» y= [1  3  5  7  9];
» x+y
ans =
    2    5    8   11   14
» 2*x-y
ans =
    1    1    1    1    1
```

Obsérvese que las matemáticas array-array también usan el mismo orden de precedencia empleando en operaciones escalares para determinar el orden de evaluación.

La multiplicación y la división elemento a elemento funciona análogamente pero utiliza una notación ligeramente menos convencional:

```
» x.*y
ans =
    1    6   15   28   45
```

Hemos multiplicado los arrays x e y elemento a elemento usando el símbolo .\* de multiplicación con punto. El punto que procede al asterisco, que es el símbolo estándar de la multiplicación, le indica a MATLAB que efectué la multiplicación de los arrays elemento a elemento. La multiplicación sin el punto significa multiplicación matricial. Para este ejemplo en particular, la multiplicación matricial no está definida:

```
» x*y
??? Error using ==> *
Inner matrix dimensions must agree.
```

La división de arrays, o división con punto, también requiere el uso del símbolo del punto:

```
» x./y
ans =
    1.0000    0.6667    0.6000    0.5714    0.5556
» y./x
ans =
    1.0000    1.5000    1.6667    1.7500    1.8000
```

La potencia de un array se define de varias formas. Como con la multiplicación y la división, ^ se reserva para la potencia de una matriz y .^ se utiliza para denotar potencia elemento a elemento:



```
» x=[1 2 3 4 5]
x =
     1     2     3     4     5
» x.^2
ans =
     1     4     9    16    25
» 2.^x
ans =
     2     4     8    16    32
```

### Como introducir matrices

Una matriz

```
      1 2 3
A=    3 5 2
      4 6 7
```

se puede introducir con un vector fila como sigue:

```
»A=[1 2 3;3 5 2;4 6 7]    %matriz de 3x3 (3 filas x 3 columnas)
```

Como se muestra, los valores deben ser introducidos entre corchetes. Los elementos de cualquier fila deben estar separados por blancos (o por comas). El final de cada fila, excepto la última, se señala con un punto y coma.

Al igual que en el caso de los vectores filas, se puede acceder a los elementos individuales de una matriz de la manera siguiente:

```
»A(3,2)
ans =
     5
```

El primer número entre paréntesis indica la fila y el segundo la indica la columna del elemento de la matriz requerido.

Una matriz grande se puede extender en varias líneas, pulsando la tecla Enter cuando se está introduciendo una matriz también le dice a Matlab que comience una nueva fila. Por ejemplo, introduzcamos la siguiente matriz B de la siguiente manera:

```
» B=[1.3 8.3 92 5 36
     4.2 68 65 32 8
     10 15 23 44 2]
```

```
B =
    1.3000    8.3000   92.0000    5.0000   36.0000
    4.2000   68.0000   65.0000   32.0000    8.0000
   10.0000   15.0000   23.0000   44.0000    2.0000
```

Observe que los retornos de carro sustituyen a los puntos y comas.



## Matrices de utilidad

En Matlab, las funciones

```
ones(n)
ones(m,n)
zeros(n)
zeros(m,n)
```

generan matrices especiales. Es decir, ones(n) produce una matriz de unos de  $n \times n$ . ones(m,n) produce una matriz de unos de  $m \times n$ . Análogamente, zeros(n) produce una matriz de ceros de  $n \times n$ , mientras zeros(m,n) produce una matriz de ceros de  $m \times n$ .

Si queremos formar una matriz de  $3 \times 3$  de ceros, debemos hacer lo siguiente:

```
» zeros(3)
```

```
ans =
```

```
0 0 0
0 0 0
0 0 0
```

Matriz de  $3 \times 1$  de unos:

```
» ones(3,1)
```

```
ans =
```

```
1
1
1
```

Además de especificar el tamaño de una matriz explícitamente, puede también usar la función size para crear una matriz especial del mismo tamaño que otra:

```
» A=[1 2 3;4 5 6];
```

```
» ones(size(A))
```

```
ans =
```

```
1 1 1
1 1 1
```

una matriz de unos del mismo tamaño que A.

## Matriz identidad

A menudo necesitamos introducir una matriz identidad I en los programas de MATLAB. La sentencia eye(n) de una matriz identidad de  $n \times n$ . Es decir,

```
» eye(5)
```

```
ans =
```

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```



### Matriz diagonal

Si  $x$  es un vector, la orden `diag(x)` produce una matriz diagonal con  $x$  sobre la diagonal. Por ejemplo, para un vector

$$x = [\text{ones}(1,n)]$$

`diag([ones(1,n)])` produce una matriz identidad de  $n \times n$  como sigue:

» `x=[ones(1,5)]`

`x =`

1 1 1 1 1

» `diag([ones(1,5)])`

`ans =`

1 0 0 0 0  
0 1 0 0 0  
0 0 1 0 0  
0 0 0 1 0  
0 0 0 0 1

Si  $A$  es una matriz cuadrada, entonces `diag(A)` es un vector formado por la diagonal de  $A$ . Y `diag(diag(A))` es una matriz diagonal con los elementos de `diag(A)` sobre la diagonal.

» `A=[1 2 3;4 5 6;7 8 9]`

`A =`

1 2 3  
4 5 6  
7 8 9

» `diag(A)`

`ans =`

1  
5  
9

### Representación gráfica de curvas

Matlab tiene un conjunto extensivo de rutinas para obtener salidas gráficas. La orden `plot` crea dibujo lineales  $x$ - $y$ . (Los dibujos logarítmicos y polares se crean sustituyendo las palabras `loglog`, `semilogx` o `polar` por `plot`). Todas estas órdenes se utilizan de la misma manera: únicamente se diferencian en como se escalan los ejes y en como se visualizan los datos.

#### Gráficas $x$ - $y$

Si  $x$  e  $y$  son vectores de la misma longitud, la orden `plot(x,y)` dibuja los valores de  $y$  frente a los valores de  $x$ .

### Representación de curvas múltiples

Para dibujar curvas en un solo gráfico, utilice la orden `plot` con múltiples argumentos

$$\text{Plot}(X1,Y1,X2,Y2,\dots,Xn,Yn)$$

Las variables  $X1,Y1,X2,Y2,\dots,Xn,Yn$  son pares de vectores. Se dibuja cada par  $x$ - $y$  y se generan múltiples curvas en el gráfico. Los argumentos múltiples tienen la ventaja de que permiten visualizar vectores de distinta longitud en mismo gráfico. Cada par utiliza un tipo de línea distinto.

*Para dibujar más de una curva en un único gráfico se puede utilizar también la orden `hold`.* La orden `hold` congela el gráfico actual e inhibe las acciones de borrado y escalado. Por tanto, las siguientes curvas se dibujarán sobre la curva original. Introduzca nuevamente la orden `hold` para liberar el gráfico actual.



### Inclusión de líneas de rejilla, título de la gráfica, etiqueta en el eje x y etiqueta en el eje y:

Una vez que tiene el gráfico en la pantalla, se pueden dibujar las líneas de rejilla, se puede poner título a la gráfica y los ejes  $x$ - $y$  pueden ser etiquetados. Las órdenes de Matlab para incluir las líneas de rejilla, el título de la gráfica, la etiqueta en el eje  $x$  y la etiqueta en el eje  $y$  son

grid (líneas de rejilla)  
title (título del gráfico)  
xlabel (etiqueta en el eje  $x$ )  
ylabel (etiqueta en el eje  $y$ )

Observe que, una vez que el orden de visualización ha sido ejecutado, las líneas de rejilla, el título del gráfico y las etiquetas en los ejes  $x$  e  $y$  se pueden introducir sucesivamente en el gráfico introduciendo las órdenes.

### Escritura de texto en la pantalla gráfica

Para escribir texto al comienzo del punto (X,Y) sobre la pantalla gráfica, utilice la orden  
text(X,Y,'texto')

Por ejemplo, la declaración

text(3,0.45,'sin t')

escribirá sin  $t$  horizontalmente empezando en el punto (3,0.45). También, las declaraciones

plot(x1,y1,x2,y2),text(x1,y1,'1'),text(x2,y2,'2')

marcarán dos curvas para que se puedan distinguir fácilmente.

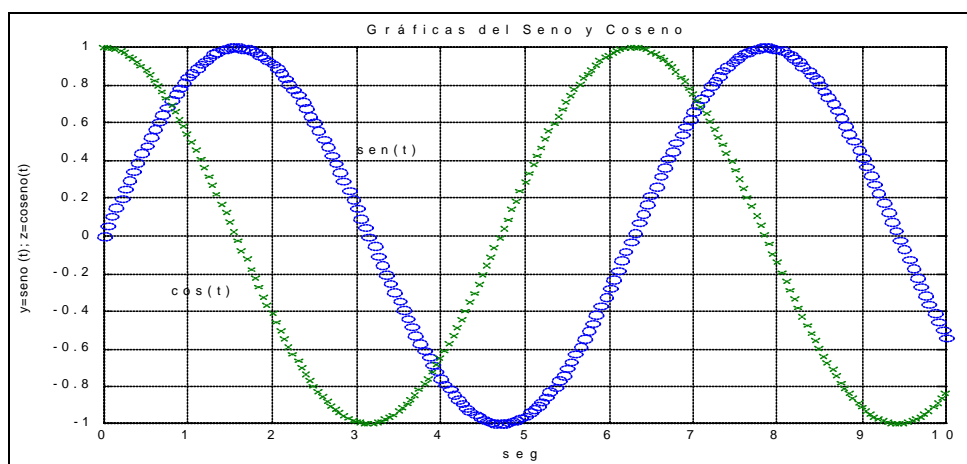
#### **Ejemplo\_1:**

Introduzca el siguiente programa de órdenes de Matlab y muestre el gráfico resultante

```
clear all, close all, clc  
t=0:0.05:10;  
y=sin (t);  
z=cos (t);  
plot(t,y,'o',t,z,'x')  
grid  
title('Gráficas del Seno y Coseno')  
xlabel('seg')  
ylabel('y=seno (t); z=coseno(t)')  
text(3,0.45,'sen(t)')  
text(0.8,-0.3,'cos(t)')
```

Observe que el vector  $t$  es una partición del dominio  $0 \leq t \leq 10$  con paso 0.05, mientras que  $y$  y  $z$  son vectores que dan los valores del seno y del coseno en los puntos de la partición.

Cuando el gráfico se encuentra en la pantalla, observe que pulsando cualquier tecla Matlab mostrará la pantalla de comandos. Con la tecla de *flecha hacia arriba*, introduzca cualquiera de las últimas siete órdenes (plot, grid, xlabel, text, etc). Matlab mostrará los gráficos actuales en pantalla. También, si se introduce la orden shg (mostrar gráfico), Matlab mostrará los gráficos actuales en pantalla.







### Ejemplo\_2

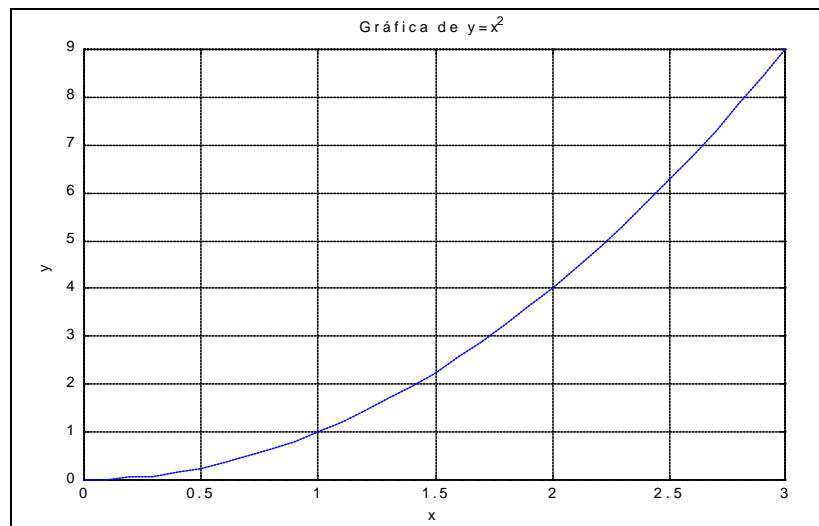
Dibuje la siguiente gráfica

$$y = x^2$$

en el intervalo  $0 \leq x \leq 3$  con incrementos de 0.1. El programa de Matlab para este problema del ejemplo es

```
clear all, close all, clc
x=0:0.1:3;
y=x.^2;
plot(x,y)
grid
title('Gráfica de y=x^2')
xlabel('x')
ylabel('y')
```

Obsérvese que es necesario que '^2' esté precedido por un punto para asegurarse que se realiza la operación deseada. La siguiente figura muestra la gráfica resultante



### Ejercicio 1:

Dibuje la siguiente gráfica

$$y = 5x^3$$

en el intervalo  $-3 \leq x \leq 3$  con incrementos de 0.01

### Datos imaginarios y complejos

Si  $z$  es un vector complejo, entonces  $\text{plot}(z)$  es equivalente a  $\text{plot}(\text{real}(z), \text{imag}(z))$ .

### Diagramas polares

La función *polar* (*teta*, *ro*) dará un gráfico en coordenadas polares de ángulo "teta" (en radianes) frente al radio "ro". Utilice la orden `grid` para dibujar las líneas de rejilla del diagrama polar.



## Diagramas logarítmicos

log-log : un gráfico utilizando escalas  $\log_{10} - \log_{10}$

semilogx: un gráfico utilizando escala semilogarítmica; el eje x es  $\log_{10}$  y el eje y es lineal.

semilogy: un gráfico utilizando escala semilogarítmica; el eje y es  $\log_{10}$  y el eje x es lineal.

## Algoritmos automáticos de representación

En Matlab el gráfico es escalado automáticamente. Este gráfico permanece como actual hasta que se dibuja otro, en tal caso el gráfico antiguo se elimina y los ejes se reescalan automáticamente. Los algoritmos automáticos de representación para la respuesta transitoria de curvas, lugar de las raíces, diagramas de bode, diagramas de Nyquist, etc... son diseñados para trabajar con una gran variedad de sistemas, pero no siempre son perfectos. Así, en algunas ocasiones, puede llegar a ser deseable no hacer caso a la característica de escalado automático de la orden plot y seleccionar manualmente los límites del gráfico.

## Escalado manual de ejes

Para cambiar la apariencia de los ejes horizontal y vertical de la gráfica, Matlab utiliza la orden axis. Esta sentencia tiene muchas variantes, por ejemplo si desea cambiar el valor entre los cuales se grafican las coordenadas x e y.

» axis([0 6 0 2]);

Esta sentencia corresponde a axis([Xmin Xmax Ymin Ymax]). La sentencia modifica el valor de los ejes en las figuras anteriores, graficando ambas funciones en los intervalos  $0 \leq x \leq 6$  para la coordenada x y  $0 \leq y \leq 2$  para la coordenada y.

## Tipos de gráficas

plot(X,Y, 'x')

Dibuja un punto en el gráfico utilizando la marca 'x', mientras

plot(X1,Y1, 'X2,Y2, '+')

utiliza una línea punteada para la primera curva y el símbolo (+) para la segunda curva. Los tipos de líneas y puntos disponibles son los siguientes:

### Tipos de líneas                      Tipos de puntos

Sólida	-	punto	.
Discontinua	--	signo de sumar	+
Punteada	:	círculo	o
Discontinua-punteada	-.	marca-x	x
		estrella	*

### Color

Las declaraciones

plot(X,Y, 'r')

plot(X,Y, '+g')

indican la utilización de una línea roja en el primer gráfico y marcas + de color verde en el segundo. Los colores disponibles son

rojo	r
verde	g
azul	b
blanco	w
invisible	i



### Subgráficas

El comando `subplot` permite dividir la ventana de gráficos en subventanas. Las posibles divisiones pueden ser dos subventanas o cuatro subventanas. Dos subventanas pueden quedar arriba y abajo o a la izquierda y la derecha. Una división de cuatro ventanas tiene dos subventanas arriba y dos abajo. Los argumentos del comando `subplot` son tres:  $m$ ,  $n$ ,  $p$ . Los dígitos  $m$  y  $n$  especifican que la ventana de gráficos se divida en una retícula de  $m$  por  $n$  ventanas más pequeñas, y el dígito  $p$  especifica la  $p$ -ésima ventana para la gráfica actual. Las ventanas se enumeran de izquierda a derecha y de arriba abajo. Por tanto, los siguientes comandos especifican que la ventana de gráficos se divida en una gráfica superior y una inferior, y que la gráfica actual se coloque en la subventana superior:

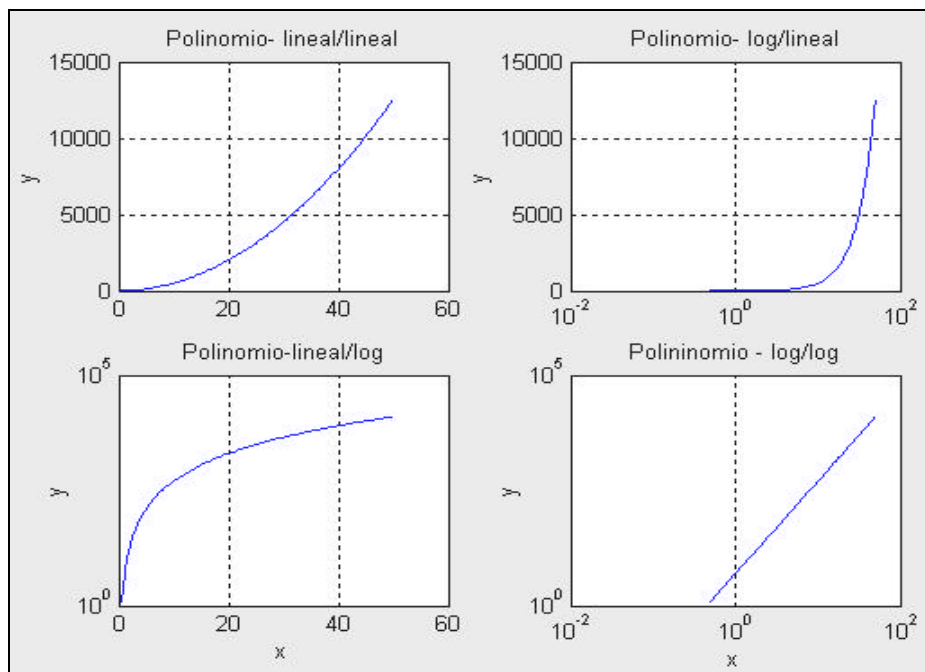
```
subplot(2,1,1), plot(x,y)
```

la siguiente figura contiene cuatro gráficas que ilustran el comando `subplot` empleando escalas lineales y logarítmicas. Esta figura se generó usando las siguientes instrucciones:

### Ejemplo\_3

%Generar curvas de un polinomio

```
clear all, close all, clc
x=0:0.5:50;
y=5*x.^2;
subplot(2,2,1), plot(x,y)
title('Polinomio- lineal/lineal')
ylabel('y'), grid
subplot(2,2,2), semilogx(x,y)
title('Polinomio- log/lineal')
ylabel('y'), grid
subplot(2,2,3), semilogy(x,y)
title('Polinomio-lineal/log')
xlabel('x'), ylabel('y'), grid
subplot(2,2,4), loglog(x,y)
title('Polinomio - log/log')
xlabel('x'), ylabel('y'), grid
```



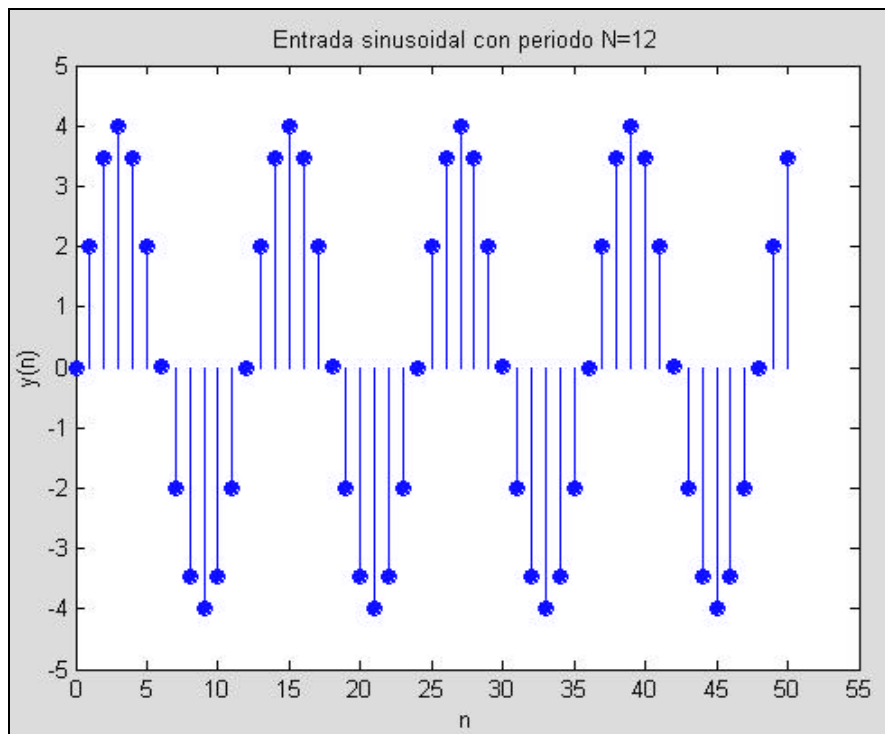


## Representación grafica de señales en tiempo discreto

Al igual que el comando plot visto anteriormente para graficar señales, Matlab utiliza el comando *stem* para graficar señales en tiempo discreto. Veamos algunos ejemplos

**Ejemplo\_4.** Graficar una señal sinusoidal con un periodo igual a 12.

```
clc, clear, close all
xmin=0; xmax=55;
ymin=-5; ymax=5;
ts=1; %tiempo de muestreo
nn=0:1:50; %vector de indice temporales
y=4*sin(2*pi/12*nn*ts);
stem(nn,y,'fill'); %grafico la señal xlabel('n')
ylabel('y(n)')
title('Entrada sinusoidal con periodo N=12')
```



### **Ejercicio 2**

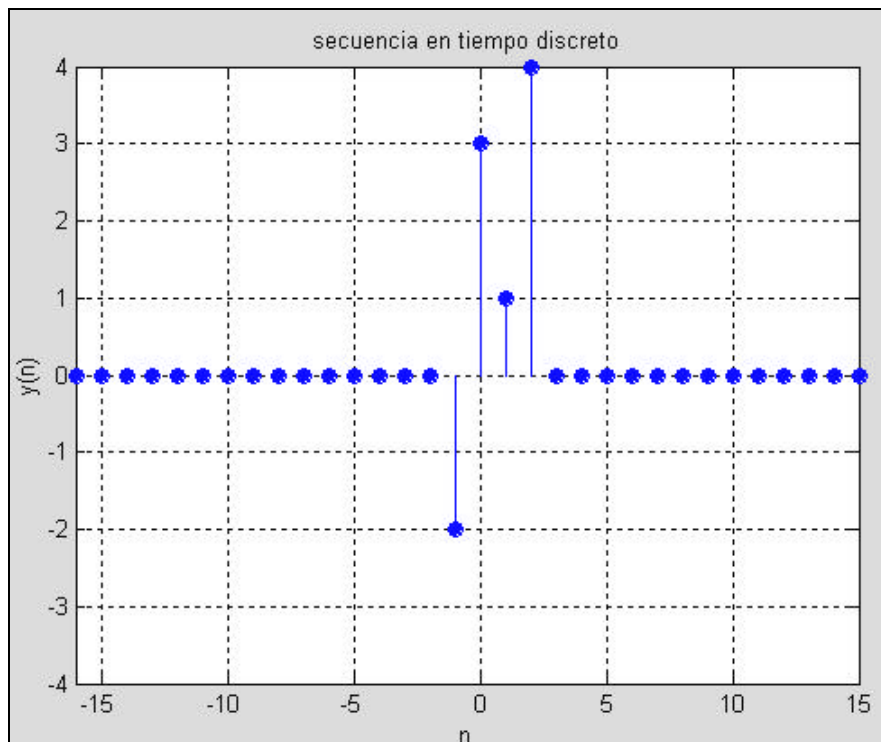
Grafique una señal cosenoidal con periodo igual a 8 y un tiempo de muestreo igual a 1.



**Ejemplo\_5.** Graficar la siguiente secuencia en tiempo discreto.

$$y[n] = -2d[n+1] + 3d[n] + d[n-1] + 4d[n-2] \quad -16 \leq n \leq 15$$

```
clc, clear, close all
N=32;
nn=-N/2:N/2-1; % vector de indice temporal (-16 a 15)
xmin=-N/2; xmax=N/2-1;
ymin=-4; ymax=4;
%senal de entrada
y=zeros(N,1); %genero una matriz de ceros de Nx1
y(N/2)=-2;
y(N/2+1)=3;
y(N/2+2)=1;
y(N/2+3)=4;
%Representacion grafica
stem(nn,y,'fill');
grid
title('secuencia en tiempo discreto');
xlabel('n'); ylabel('y(n)');
axis([xmin xmax ymin ymax])
```



### Ejercicio 3

Grafique la siguiente señal en tiempo discreto

$$y[n] = 4d[n] + 3d[n-1] + 2d[n-2] + d[n-3] \quad -8 \leq n \leq 7$$



## Representación en Matlab de expresiones simbólicas:

Las expresiones simbólicas son expresiones que poseen números y letras, en las cuales las variables no necesitan tener valores predefinidos. Por ejemplo, la siguiente es una ecuación simbólica

$$x^2 + 2x + 1 = 0$$

Normalmente podríamos resolver esta ecuación sin necesidad de tener que evaluar la expresión para cada valor de  $x$ , hasta obtener valor de las raíces de la ecuación. Lo mismo puede realizar Matlab, aún con expresiones mucho más complejas o con sistemas de ecuaciones.

La forma de introducir en Matlab una expresión simbólica no difiere mucho de la manera que vimos anteriormente. Veamos dos ejemplos

```
» f='x^3+1'
```

```
f =
```

$$x^3+1$$

Hemos definido la función  $f$  como  $x^3 + 1$ . Si quisiéramos introducir en Matlab una ecuación de segundo grado, escribimos

```
» ecu='x^2+2x+1=0'
```

```
ecu =
```

$$x^2+2x+1=0$$

Observe que la forma de escribir la función y la ecuación es prácticamente la misma que cuando trabajamos con números pero ambas expresiones están encerradas entre signos apóstrofe ' '. Otro ejemplo

```
» f1='x^n'
```

```
f1 =
```

$$x^n$$

Aquí hemos definido la función  $f1$  como  $x$  elevado a la  $n$ . En este caso la expresión simbólica contiene más de una variable.

En todos los casos utilizaremos el signo ' ' para indicarle a Matlab que estamos definiendo una expresión simbólica. Matlab posee muchos operadores para trabajar con expresiones simbólicas, solo concentraremos nuestra atención sobre unos pocos, en particular veremos como resolver ecuaciones y como graficar funciones.

## Resolviendo ecuaciones con Matlab

Veamos como resolver ecuaciones simbólicas con Matlab. Por ejemplo, intentemos resolver la ecuación de segundo grado  $t^2 - 2t + 1 = 0$ .

```
» eq='t^2-2*t+1=0'
```

```
eq =
```

$$t^2-2*t+1=0$$

Hemos definido la ecuación como  $eq$ . Ahora utilizaremos la función *solve* para obtener las raíces de la ecuación.

```
» tsol=solve(eq,'t') %solve(nombre_ecuación,'variable')
```

```
tsol =
```

```
[ 1]
```

```
[ 1]
```

La función *solve* da como respuesta que ambas raíces son iguales a 1. Esta función puede visualizarse para resolver muchos tipos de ecuaciones. Veamos otro ejemplo del uso *solve*

```
» eq='(3*x^2+2*x+1)=(5*x+12)';
```

```
» xsol=solve(eq,'x')
```

```
xsol =
```

$$[ 1/2+1/6*141^{(1/2)}]$$

$$[ 1/2-1/6*141^{(1/2)}]$$

En este ejemplo hemos utilizado *solve* para resolver la ecuación

$$3x^2 + 2x + 1 = 5x + 12$$

Como respuesta *solve* nos dice que existen dos valores de  $x$  que verifican la ecuación. Si queremos que la respuesta esté escrita como un número que sea más fácil de leer en lugar de una expresión, podemos utilizar el comando *numeric* de la siguiente manera



```
» res=numeric(xsol)
res =
    2.4791
   -1.4791
```

Ahora veamos como resolver un sistema de ecuaciones, para ello veamos como resolver el sistema

$$\begin{cases} 2x - y = 1 \\ -x + 2y = 7 \end{cases}$$

Introducimos ambas ecuaciones en Matlab llamándolas eq1 y eq2

```
» eq1='2*x-y=1';
» eq2='-x+2*y=7';
```

Para resolver este sistema de ecuaciones utilizaremos el mismo comando *solve* pero en este caso tiene una estructura un poco diferente a la vista anteriormente

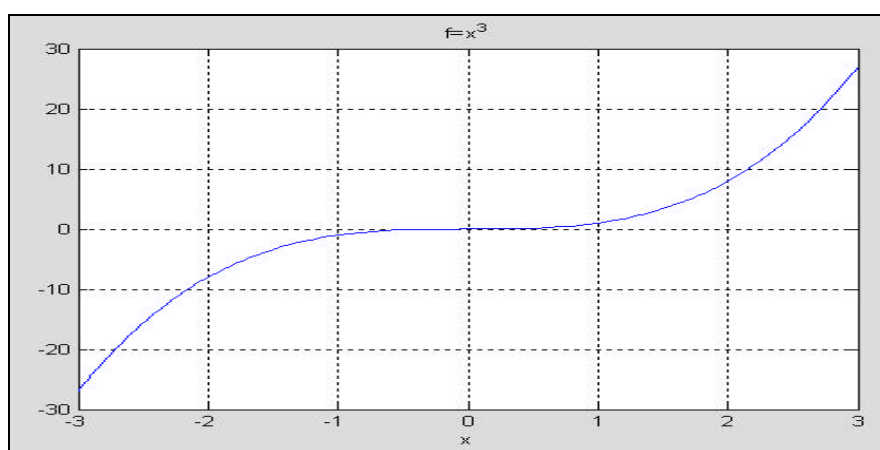
```
» [xsol,ysol]=solve(eq1,eq2,'x,y')
xsol =
    3
ysol =
    5
```

Hemos indicado a solve que debe resolver de manera simultánea eq1 y eq2, con respecto a las variables x e y. El resultado lo hemos asignado a los números *xsol* y *ysol*.

### Graficando expresiones simbólicas

Para graficar una función definida como una expresión simbólica, Matlab utiliza la sentencia *fplot*. Veamos como se utiliza para graficar una función como por ejemplo

```
f(x) = x^3
» f='x^3';
» fplot(f,[-3,3])    %[-3,3] indica [xmin,xmax] para el eje x
» gris
» title('f=x^3')
» xlabel('x')
```



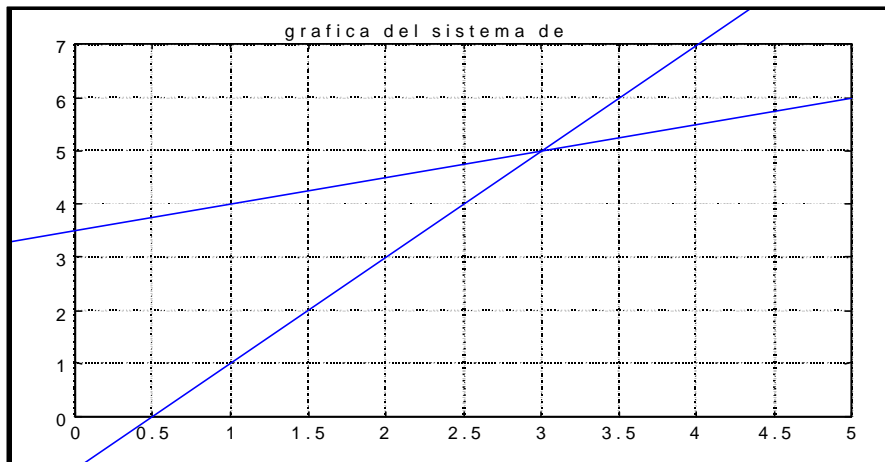


Veamos otro ejemplo del uso *fplot*, en este caso dos rectas superpuestas.

```
» f='x^3';  
» fplot(f,[-3,3]) %[-3,3] indica [xmin,xmax] para el ejex  
» y1='2*x-1';  
» y2='x/2+3.5';  
» fplot(y1,[-5 5]) %grafica y1 entre x=[-5,5]  
» hold on
```

La sentencia *hold on* mantiene la figura activa, a fin de que la segunda gráfica se superponga en misma figura

```
» y1='2*x-1';  
» y2='x/2+3.5';  
» fplot(y1,[-5 5]) %grafica y1 entre x=[-5,5]  
» hold on  
» fplot(y2,[-5 5]) %grafica y2  
» axis([0 5 0 7]) %fija nuevos valores mínimos y máximos de los ejes [Xmin Xmax Ymin Ymax]  
» grid on %adiciona una grilla de líneas punteadas a la figura  
» title('grafica del sistema de ecuaciones')
```







## **Anexo**

### **Operador dos puntos (:)**

Este operador es muy importante en MATLAB y puede usarse de varias formas. Se sugiere al lector que practique mucho sobre los ejemplos contenidos en este apartado, introduciendo todas las modificaciones que se le ocurran y haciendo pruebas abundantes (¡Probar es la mejor forma de aprender!).

Para empezar, defínase un vector  $x$  con el siguiente comando:

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

En cierta forma se podría decir que el operador  $(:)$  representa un rango: en este caso, los números enteros entre el 1 y el 10. Por defecto el incremento es 1, pero este operador puede también utilizarse con otros valores enteros y reales, positivos o negativos. En este caso el incremento va entre el valor inferior y el superior, en las formas que se muestran a continuación:

```
>> x=1:2:10  
x =  
1 3 5 7 9  
>> x=1:1.5:10  
x =  
1.0000 2.5000 4.0000 5.5000 7.0000 8.5000 10.0000  
>> x=10:-1:1  
x =  
10 9 8 7 6 5 4 3 2 1
```

Puede verse que, por defecto, este operador produce vectores fila. Si se desea obtener un vector columna basta trasponer el resultado. El siguiente ejemplo genera una tabla de funciones seno y coseno. Ejecútese y obsérvese el resultado

```
>> x=[0.0:pi/50:2*pi]';  
>> y=sin(x); z=cos(x);  
>> [x y z]
```

El operador dos puntos  $(:)$  es aún más útil y potente –y también más complicado– con matrices.

A continuación se va a definir una matriz  $A$  de tamaño  $6 \times 6$  y después se realizarán diversas operaciones sobre ella con el operador  $(:)$ .

```
>> A=magic(6)      % magic(6) crea una matriz (6x6), con la propiedad de que todas las filas y columnas  
                  %suman lo mismo  
A =  
35 1 6 26 19 24  
3 32 7 21 23 25  
31 9 2 22 27 20  
8 28 33 17 10 15  
30 5 34 12 14 16  
4 36 29 13 18 11
```

Recuérdese que MATLAB accede a los elementos de una matriz por medio de los índices de fila y de columna encerrados entre paréntesis y separados por una coma. Por ejemplo:

```
>> A(2,3)  
ans =  
7
```

El siguiente comando extrae los 4 primeros elementos de la 6ª fila:

```
>> A(6,1:4)  
ans =  
4 36 29 13
```



Los dos puntos aislados representan "todos los elementos". Por ejemplo, el siguiente comando extrae todos los elementos de la 3ª fila:

```
>> A(3, :)
```

```
ans =
```

```
31 9 2 22 27 20
```

Para acceder a la última fila o columna puede utilizarse la palabra end, en lugar del número correspondiente.

Por ejemplo, para extraer la sexta fila (la última) de la matriz:

```
>> A(end, :)
```

```
ans =
```

```
4 36 29 13 18 11
```

El siguiente comando extrae todos los elementos de las filas 3, 4 y 5:

```
>> A(3:5,:)
```

```
ans =
```

```
31 9 2 22 27 20
```

```
8 28 33 17 10 15
```

```
30 5 34 12 14 16
```

Se pueden extraer conjuntos disjuntos de filas utilizando corchetes [ ]. Por ejemplo, el siguiente comando extrae las filas 1, 2 y 5:

```
>> A([1 2 5],:)
```

```
ans =
```

```
35 1 6 26 19 24
```

```
3 32 7 21 23 25
```

```
30 5 34 12 14 16
```

En los ejemplos anteriores se han extraído filas y no columnas por motivos del espacio ocupado por el resultado en la hoja de papel. Es evidente que todo lo que se dice para filas vale para columnas y viceversa: basta cambiar el orden de los índices.

El operador dos puntos (:) puede utilizarse en ambos lados del operador (=). Por ejemplo, a continuación se va a definir una matriz identidad B de tamaño 6x6 y se van a reemplazar filas de B por filas de A. Obsérvese que la siguiente secuencia de comandos sustituye las filas 2, 4 y 5 de B por las filas 1, 2 y 3 de A,

```
>> B=eye(size(A));
```

```
>> B([2 4 5],:)=A(1:3,:)
```

```
B =
```

```
1 0 0 0 0 0
35 1 6 26 19 24
0 0 1 0 0 0
3 32 7 21 23 25
31 9 2 22 27 20
0 0 0 0 0 1
```

Para invertir el orden de las columnas de una matriz se puede hacer lo siguiente:

```
>> A=magic(3)
```

```
A =
```

```
8 1 6
```

```
3 5 7
```

```
4 9 2
```

```
>> A(:,3:-1:1)
```

```
ans =
```

```
6 1 8
```

```
7 5 3
```

```
2 9 4
```

aunque hubiera sido más fácil utilizar la función fliplr(A), que es específica para ello.

Finalmente, hay que decir que A(:) representa un vector columna con las columnas de A una detrás de otra.



## Hipermatrices (arrays de más de dos dimensiones)

MATLAB permite trabajar con hipermatrices, es decir con matrices de más de dos dimensiones ver figura. Una posible aplicación es almacenar con un único nombre distintas matrices del mismo tamaño (resulta una hipermatriz de 3 dimensiones). Los elementos de una hipermatriz pueden ser números, caracteres, estructuras, y vectores o matrices de celdas. El tercer subíndice representa la tercera dimensión: la "profundidad" de la hipermatriz. Las funciones para trabajar con estas hipermatrices están en el sub-directorio `toolbox\matlab\datatypes`.

Las funciones que operan con matrices de más de dos dimensiones son análogas a las funciones vistas previamente, aunque con algunas diferencias. Por ejemplo, las siguientes sentencias generan, en dos pasos, una matriz de  $2 \times 3 \times 2$ :

```
>> AA(:,:,1)=[1 2 3; 4 5 6]
```

```
AA =
```

```
1 2 3
4 5 6
```

```
>> AA(:,:,2)=[2 3 4; 5 6 7]
```

```
AA(:,:,1) =
```

```
1 2 3
4 5 6
```

```
AA(:,:,2) =
```

```
2 3 4
5 6 7
```

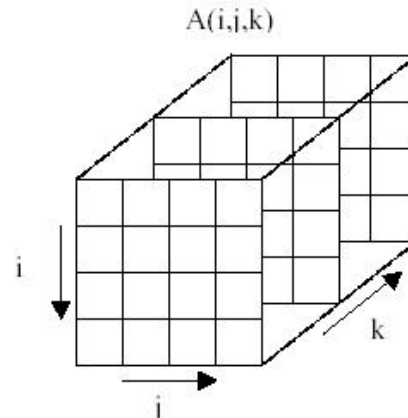


Figura 23. Hipermatriz de tres dimensiones.

Las siguientes funciones de MATLAB se pueden emplear también con hipermatrices:

<code>size()</code>	devuelve tres o más valores (el nº de elementos en cada dimensión)
<code>ndims()</code>	devuelve el número de dimensiones
<code>squeeze()</code>	elimina las dimensiones que son igual a uno
<code>reshape()</code>	distribuye el mismo número de elementos en una matriz con distinta forma o con distintas dimensiones
<code>permute(A,v)</code>	permuta las dimensiones de A según los índices del vector v
<code>ipermute(A,v)</code>	realiza la permutación inversa

### Algunos ejemplos

```
>> size (AA)
```

```
ans =
```

```
2 3 3
```

```
>> ndims(AA)
```

```
ans =
```

```
3
```

```
>> permute(AA,[2 3 1])
```

```
ans(:,:,1) =
```

```
1 2
2 3
3 4
```

```
ans(:,:,2) =
```

```
4 5
5 6
6 7
```