

# La Teoría de Control con

# MATLAB®

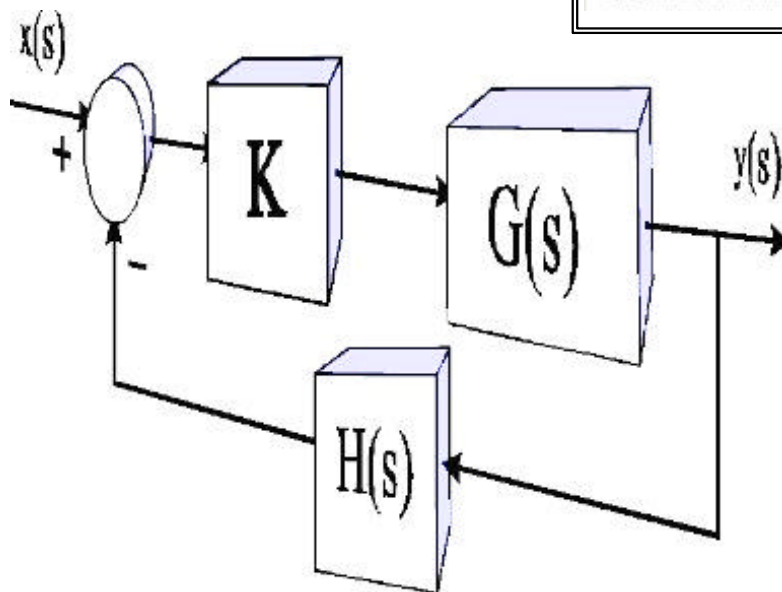
*The Language of Technical Computing*

Version 6.5.0.180913a Release 13  
June 18, 2002



Copyright 1984-2002, The MathWorks, Inc.

Guía para el uso de  
la *toolbox* de control  
y *simulink*



Enrique Pinto  
Septiembre 2004



▪ INTRODUCCIÓN A MATLAB.....	4
▪ OPERACIONES BÁSICAS EN MATLAB.....	5
▪ DEFINICIÓN DE VARIABLES.....	7
▪ FUNCIONES MATEMÁTICAS.....	9
▪ MANEJO DE VECTORES o ARRAYS .....	10
▪ OPERACIONES CON VECTORES Y MATRICES.....	12
▪ CREACIÓN DE GRÁFICOS .....	14
▪ FICHEROS <i>SCRIPT</i> .....	17
▪ FUNCIONES DE USUARIO .....	18
▪ TRABAJANDO CON POLINOMIOS.....	20
▪ TRANSFORMADA DE LAPLACE. <i>Definición de funciones simbólicas</i> .....	23
▪ FUNCIONES DE TRANSFERENCIA EN MATLAB.....	26
▪ CONVERSIÓN ENTRE FUNCIONES DE TRANSFERENCIA .....	28
▪ TRABAJANDO CON BLOQUES EN MATLAB .....	32
▪ OBTENCIÓN DE RESPUESTAS CON MATLAB.....	35
▪ COMANDOS PARA EL MANEJO DE GRÁFICOS EN MATLAB.....	42
▪ ANÁLISIS DE LOS SISTEMAS EN MATLAB.....	43
▪ INTERFASE GRÁFICA <i>RLTOOL</i> .....	50
▪ SIMULINK.....	53
Creación de un modelo.....	54

*MATLAB* es hoy en día una de las principales herramientas *software* existentes en el mercado para el cálculo matemático, análisis de datos y visualización de resultados. Desde su aparición en los años 70 ha ido introduciéndose con fuerza en el ámbito científico y en la universidad. En la actualidad es algo más que una herramienta de ayuda al cálculo avanzado, es también un lenguaje de programación con una elevada potencia de cálculo matricial.

Todas las operaciones que realiza *MATLAB* se basan en una estructura de datos matricial. Dentro del entorno de trabajo de *MATLAB* se pueden definir nuevos comandos o funciones, programadas por el propio usuario, a través de los ficheros-M. Este tipo de ficheros da paso a las *toolbox* de *MATLAB* que son una importante colección de funciones ya programadas y disponibles para el usuario en diferentes campos como son el análisis de datos, procesado de imágenes y de señales, diseño de sistemas de control, procesado digital de señal (DSP), comunicaciones, aplicaciones embebidas y también herramientas para finanzas y economía

Una importante ventaja que presenta *MATLAB* es el entorno gráfico de trabajo, la claridad en la presentación de los resultados y la versatilidad que presenta en la creación y modificación de los mismos.

Dentro del campo del control automático, *MATLAB* ya tiene desarrolladas un gran número de funciones para el análisis de los sistemas de regulación. Estas se encuentran dentro del *toolbox* de control, y permiten el análisis en el dominio de la frecuencia (*bode*, *nyquist*), en el dominio del tiempo (*step*, *impulse*), conversión de sistemas continuos a discretos; representación de los sistemas (funciones de transferencia, polo-cero-K), análisis de sistemas lineales (observabilidad y controlabilidad de sistemas, lugar de las raíces). Es en este campo en el que se va a centrar la presente guía para el uso de la *toolbox* de control y *simulink*.

Las operaciones con datos numéricos se evalúan de izquierda a derecha, con órdenes de prioridad distintos en función de la operación a realizar; es conveniente utilizar paréntesis para evitar resultados que no coinciden con el esperado (también por claridad):

\* SUMA: mínima prioridad.

```
>>12+4      % pulsar tecla de Aceptar o Enter  
ans= 16      % ans es la variable resultado por defecto
```

\* RESTA: mínima prioridad.

```
>>12-4      % pulsar tecla de Aceptar o Enter  
ans= 8       % ans es la variable resultado por defecto
```

**Nota:** suma y resta se evalúan con la misma prioridad.

\* MULTIPLICACIÓN: media prioridad.

```
>>12*4  
ans= 48
```

\* DIVISIÓN: media prioridad.

```
>>12/4  
ans= 3
```

**Nota:** la multiplicación y división se evalúan con la misma prioridad y es mayor que la suma y la resta.

\* POTENCIA: máxima prioridad.

```
>>12^4  
ans= 20736
```

Observar las diferencias en los siguientes ejemplos:

```
>>12/2^2-18/6-1  
ans= -1
```

Empieza a evaluar de izquierda a derecha; la potencia se evalúa primero por tener mayor prioridad que la división; a continuación calcula la segunda división para por último realizar las restas. Para evitar confusiones se podría haber escrito:

```
>>12/(2^2)-(18/6)-1
```

```
ans= -1
```

Si se desea calcular primero la división y después la potencia:

```
>>(12/2)^2-18/6-1
```

```
ans=32
```

dando un resultado claramente distinto.

El formato de los datos numéricos puede ser de varios tipos entre los que destacan:

- \* **short** ⇒ número con 4 dígitos (es la opción por defecto)
- \* **long** ⇒ número con 16 dígitos
- \* **short e** ⇒ número con 4 dígitos y exponente
- \* **long e** ⇒ número con 16 dígitos y exponente
- \* **hex** ⇒ número en hexadecimal

**Nota:** el formato se cambia pinchando en el menú *File* → *Preferences*

```
>>4/12
```

```
ans= 0.3333 (short)
```

```
0.3333333333333333 (long)
```

```
3.3333e-001 (short e)
```

```
4040000000000000 (hex)
```

## DEFINICIÓN DE VARIABLES

---

Para facilitar la manipulación de datos en *MATLAB*, se pueden definir variables con nombres para su rápida identificación. Las reglas a seguir son:

- \* longitud máxima 63 caracteres SIN ESPACIOS.
- \* sensible a mayúsculas y minúsculas.
- \* el primer carácter ha de ser siempre una letra.

Definición de tres variables “x1”, “x2” y “división”:

```
>>x1=12;x2=4;      % el “;” anula la impresión por pantalla
>>division=x1/x2
>>division= 3
```

La consulta de variables se puede realizar con las órdenes **who** y **whos**:

```
>>who
Your variables are: division    x1    x2

>>whos

Name      Size      Bytes Class
division  1x1          8 double array
x1         1x1          8 double array
x2         1x1          8 double array
Grand total is 3 elements using 24 bytes
```

El valor de la variable se consulta tecleando el nombre y pulsando Aceptar o *Enter*.

```
>>división (pulsar aceptar)
división=3
```

El borrado de variables se realiza con la función *clear* seguido del nombre de la variable. IMPORTANTE: ejecutando solamente el comando *clear* se borran TODAS las variables definidas:

```
>>clear x1      %borra la variable x1 del entorno de trabajo
>>clear         %borra todas las variables del entorno de trabajo
```

Todas las variables definidas en una sesión de *MATLAB* se pueden archivar para su uso en posteriores sesiones. Para ello, pinchar en el menú *File* → *Save Workspace as* e introducir un nombre para el fichero con la extensión “.mat” , p.e, “misvar.mat”

Para recuperar el fichero con las variables almacenadas en una anterior sesión de trabajo se ha de pinchar en el menú *File* → *Import Data*, activándose la interfase de importar datos. También es posible recuperarlos directamente desde el editor de comandos mediante la función ***load***:

```
>>load misvar.mat
```

```
>>who
```

```
Your variables are: division x1 x2
```



## FUNCIONES MATEMÁTICAS

---

*Matlab* incluye una amplia librería de funciones matemáticas. Todas las operaciones de *Matlab* se realizan en radianes.

### Funciones Trigonométricas:

<code>sin(x)</code>	<code>sinh(x)</code>
<code>cos(x)</code>	<code>cosh(x)</code>
<code>tan(x)</code>	<code>tanh(x)</code>
<code>asin(x)</code>	<code>asinh(x)</code>
<code>acos(x)</code>	<code>acosh(x)</code>
<code>atan(x)</code>	<code>atanh(x)</code>

```
>>angulo=cos(45*pi/180)           %calcular el coseno de 45°  
angulo= 7.0711e-001
```

### Raíz cuadrada y exponencial:

<code>sqrt(x)</code>	%raíz cuadrada
<code>exp(x)</code>	%exponencial $e^x$

```
>>exp(1)  
ans= 2.7183e+000    %base neperiana
```

### Operaciones con complejos:

<code>real(x)</code>	<code>imag(x)</code>
<code>abs(x)</code>	<code>angle(x)</code>

```
>>comple1=1+i; comple2=1-i;       %definición de 2 variables complejas  
>>real(comple1)  
ans= 1  
>>imag(comple2)  
ans= -1  
>>modulo1=abs(comple1)           %obtención del módulo de la variable comple1  
modulo1= 1.4142e+000
```

```
>>angulo1=angle(comple1)      %obtención del ángulo de la variable comple1
angulo1= 7.8540e-001
>>angulo1=angulo1*180/pi      %paso de radianes a grados
angulo1= 45
```

## MANEJO DE VECTORES o ARRAYS

---

Los *arrays* o cadenas numéricas son, junto a las matrices, el tipo de datos más usual dentro del entorno de *Matlab*. Un *array* es un vector con un determinado número de elementos y que se puede ordenar como:

- \* vector fila: elementos agrupados en 1 fila y varias columnas.
- \* vector columna: elementos agrupados en 1 columna y varias filas.

**LOS VECTORES (así como las matrices) SE ESCRIBEN ENTRE CORCHETES, CON SUS ELEMENTOS SEPARADOS POR ESPACIOS O COMAS.**

Definición y manipulación de vectores:

```
>>vector1=[1 2 3 4]; vector2=[4 3 2 1];
>>vector1(2)      %consulta del 2º elemento de la variable vector1
ans= 2
>>vector1(2)=1     %modifica el valor del elemento 2 de la variable vector1
vector1= 1 1 3 4
>>vector1(2:4)     %consultar los elementos 2 a 4
ans= 1 3 4
>>vector1(1:2:4)   %consulta elementos impares desde elemento 1º hasta el 4º
ans= 1 3
```

Construcción de vectores:

\* Indicando incremento entre elementos del vector:

```
>>vector3=(1:2:15)  %comienza en 1, acaba en 15 con un incremento de 2
```

```
vector3= 1 3 5 7 9 11 13 15
```

\* Indicando nº de elementos del vector:

```
>>vector4=linspace(1,10,5) %vector de 5 elementos, siendo el primero 1 y el  
                             %último 10
```

```
>>vector4= 1 3.25 5.50 7.75 10
```

El incremento entre los valores de cada elemento del vector lo calcula internamente *Matlab*, promediando entre el valor del elemento inicial y el final de la siguiente forma:

$$(\text{valor final} - \text{valor inicial}) / (\text{nº elementos} - 1)$$

```
>>vector4=linspace(2,10,5)
```

```
vector4= 2 4 6 8 10
```

\* Vector columna:

```
>>vector5=[6;5;4;3;2;1] %se construye separando elementos con “;”
```

```
>>vector5= 6
```

```
5
```

```
4
```

```
3
```

```
2
```

```
1
```

También se puede construir un vector columna trasponiendo un vector fila:

```
>>vector6=vector4'
```

```
>>vector6= 2
```

```
4
```

```
6
```

```
8
```

```
10
```

Construcción de una matriz:

```
>>matriz1=[1 2 3;4 5 6;7 8 9]
```

```
matriz1= 1 2 3
```

```
4 5 6
```

```
7 8 9
```

## OPERACIONES CON VECTORES Y MATRICES

---

Las operaciones entre vectores se realizan elemento a elemento y deben tener el mismo número de elementos:

\* SUMA            **>>vector1+vector2**  
\* RESTA           **>>vector1-vector2**  
\* PRODUCTO      **>>vector1.\*vector2**  
\* DIVISIÓN        **>>vector1./vector2**  
\* POTENCIA       **>>vector1.^vector2**

Nota: en el producto, división y potencia el “.” indica a *Matlab* que ha de operar elemento a elemento, si no dará error.

Las operaciones con matrices deben seguir las reglas del álgebra matricial en cuanto a la dimensión de las matrices con las que se va a operar:

\* SUMA                      **>>matriz1+matriz2**  
\* RESTA                    **>>matriz1-matriz2**  
\* PRODUCTO                **>>matriz1\*matriz2**  
\* TRASPUESTA              **>>matriz3=matriz2'**  
\* DETERMINANTE           **>>det(matriz1)**  
\* INVERSA                  **>>inv(matriz1)**

Desde el comienzo de la sesión de *Matlab*, se han ido introduciendo una gran variedad de variables en el espacio de trabajo; es conveniente echar un vistazo a los nombres y tipos de variables incluidas hasta el momento, y observar el tamaño y tipo de datos de las últimas variables creadas.

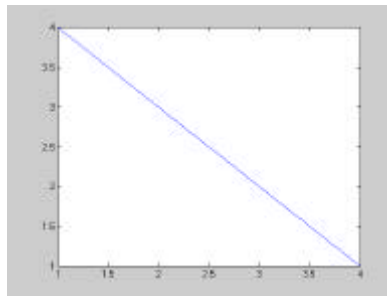
```
>> whos
```

Name	Size	Bytes Class
angulo	1x1	8 double array
angulo1	1x1	8 double array
ans	1x1	8 double array
comple1	1x1	16 double array (complex)
comple2	1x1	16 double array (complex)
division	1x1	8 double array
matriz1	3x3	72 double array
matriz2	3x3	72 double array
modulo1	1x1	8 double array
productomatriz	3x3	72 double array
restamatriz	3x3	72 double array
sumatriz	3x3	72 double array
vector1	1x4	32 double array
vector2	1x4	32 double array
vector3	1x8	64 double array
vector4	1x5	40 double array
vector5	6x1	48 double array
vector6	5x1	40 double array
x1	1x1	8 double array
x2	1x1	8 double array

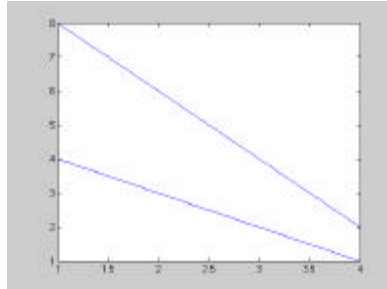
Grand total is 86 elements using 704 bytes

Una de las características más notables de *Matlab* es la interfase gráfica que permite la generación de gráficos de gran calidad y de fácil manipulación. El comando principal para la generación de gráficos es ***plot*** que permite representar parejas de puntos; a continuación se describen algunas de sus posibilidades:

**>>plot(vector1,vector2)**                    %dibuja en abscisa y ordenada los elementos del  
%vector1 y del vector2.



**>>hold on**                                    %congela la gráfica activa para sucesivas curvas.  
**>>plot(vector1,2\*vector2)**                %representa nueva curva sobre la gráfica anterior.



**>>hold off**                                    %descongela la actual gráfica activa; sucesivas  
%curvas se representan en una nueva gráfica.

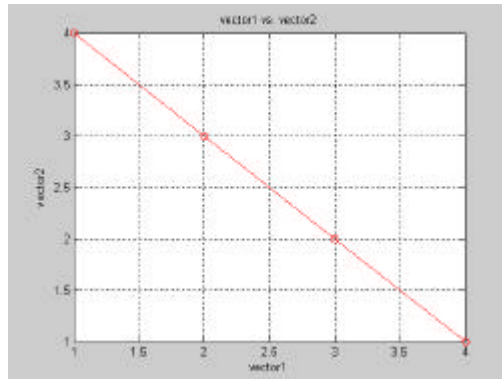
### Otras opciones gráficas

Las gráficas se pueden dibujar especificando un color, tipo de línea y aspecto; basta añadir al comando ***plot*** una cadena de caracteres (entre comillas simples) indicando la opción elegida (ver ayuda ***help plot***):

```
>> plot(vector1,vector2,'ro-')    %representa los puntos en rojo, con círculos y
                                   %unidos mediante una línea continua.
```

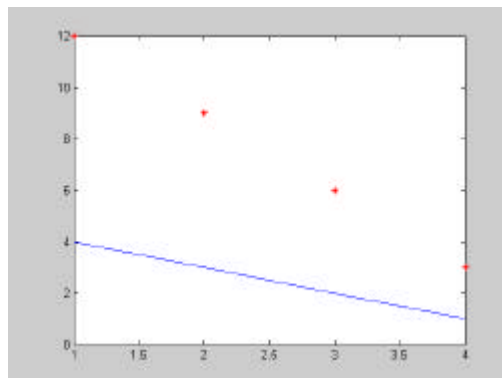
El siguiente grupo de comandos añade a la gráfica una rejilla, etiquetas a los ejes de abscisa y ordenada, y un título:

```
>>grid
>>xlabel('vector1')
>>ylabel('vector2')
>>title('vector1 vs. vector2')
```



La orden *plot* permite representar dos gráficas simultáneamente con una única llamada a la función:

```
>>plot(vector1,vector2,vector1,3*vector2,'c*')
```



Gráficos en 3D se representan con el comando **plot3**:

```
>>plot3(vector1,vector2,vector7)    %vectores de la misma dimensión.
```

Se puede añadir texto en la gráfica con los comandos **gtext** y **text**:

```
>>gtext('texto a insertar')        %inserta el texto pinchando con el ratón.
>>text(x,y,'texto a insertar')      %idem especificando coordenadas de inserción.
```

La modificación de los ejes de la gráfica se realiza con la orden **axis**, válida también para gráficos 3D añadiendo una tercera pareja de valores:

```
<<axis([minX maxX mínY maxY])
```

Nota: ver otras opciones de este comando tecleando **help axis**.

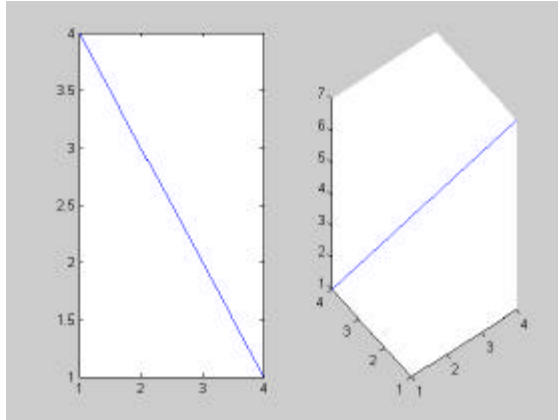
Una gráfica se puede subdividir en zonas para sucesivas representaciones con la orden **subplot**. Esta orden va acompañada de tres parámetros que le indican a *Matlab* el número de divisiones de la gráfica (con notación matricial fila-columna), y la división activa en cada momento para representar los datos correspondientes:

```
>>subplot(1,2,1)    %gráfica dividida en 2 zonas (1 fila, 2 columnas), activando la  
                    %zona de la izquierda para dibujar.
```

```
>>plot(vector1,vector2)
```

```
>>subplot(1,2,2)    % gráfica dividida en 2 zonas (1 fila, 2 columnas), activando la  
                    %zona de la derecha para dibujar.
```

```
>>plot3(vector1,vector2,vector7)
```



Para hacer un **zoom** sobre una gráfica basta con activar dicho comando y pinchar sobre la gráfica con el ratón:

```
>>zoom on
```

(En las últimas versiones de *matlab* aparecen en cada gráfica generada los iconos de *zoom in* y *zoom out*, sin que sea necesario activarlo desde el editor de comandos).



Son ficheros de texto con la extensión “.m” que ejecutan línea a línea las órdenes y comandos que aparecen en él. Se pueden construir con cualquier editor de texto. Para crear un fichero *script* desde *Matlab*, pinchar en *File* → *New* → *M-file*; se abrirá el editor de ficheros .m donde se escribirán los comandos y funciones de *Matlab* requeridos para su ejecución:

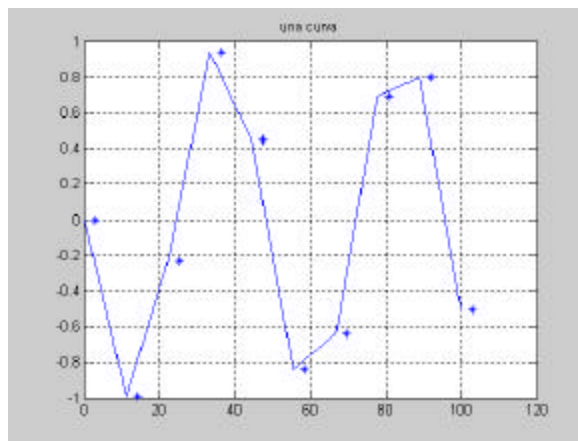
### Fichero “curva.m”:

```
w=linspace(0,100,10)
y=sin(w)
plot(w,y)
grid
title('una curva')
hold on
plot(w+3,y,'*')
```

Este fichero se ha de guardar dentro de la estructura de directorios (*path*) de *Matlab*. Para ejecutarlo basta con teclear desde el editor de funciones el nombre del fichero:

```
>>curva
```

%genera una gráfica con la función seno (línea continua) y la misma desplazada tres instantes (sólo puntos), ambas definidas en el intervalo 0 a 100.



Las variables definidas en el fichero *curva.m* pasan a formar parte de la colección de variables existentes en la sesión de *Matlab* desde la que se ha ejecutado el fichero *script*.

De forma parecida a los ficheros *script*, el usuario puede definir sus propias funciones, con paso de parámetros, devolviendo resultados, pintando gráficas, etcétera. A diferencia de los ficheros *script* anteriormente comentados, las variables empleadas en las funciones creadas por el usuario no son visibles en la sesión de *Matlab* desde donde se ha ejecutado dicha función. Es regla primordial que el nombre de la función coincida con el nombre del fichero y que, al igual que los ficheros de tipo *script*, también ha de tener la extensión “.m”. Sólo serán variables del entorno de trabajo aquellas que se comunican con la función:

### Fichero “cuadrado.m”:

```
1. function y=cuadrado
2. % Calcula y representa los cuadrados de los elementos del vector
   introducido por el usuario.
3. % Devuelve en una variable el vector "cuadrado".
4. min=input('Introduce el primer elemento del vector= ');
5. max=input('Introduce el último elemento del vector= ');
6. t=input('Incremento entre elementos del vector= ');
7. if t<=0
       fprintf('Incremento no valido');
8. else
       x=[min:t:max];
       y=x.^2;
       plot(x,y);
       grid;
       fprintf('Los cuadrados son: \n',y);
9. end
```

### Comentarios a las líneas de programa anteriores

- Línea 1: definición de la función “cuadrado”; devuelve como resultado la variable interna “y”.
- Líneas 2 y 3: comentarios a la función; aparecerán si se invoca el comando help seguido del nombre de la función (“cuadrado” en este caso).
- Línea 4: variable interna “min” donde se almacenará el 1<sup>er</sup> elemento de un vector.
- Línea 5: variable interna “max” donde se almacenará el último elemento del vector.
- Línea 6: variable interna “t” donde se almacenará el incremento entre elementos.
- Línea 7: bucle condicional para asegurar incrementos mayores de cero.

Línea 8: creación del vector, obtención de los cuadrados de los elementos, dibujar los puntos, añadir rejilla a la gráfica y sacar por pantalla los valores de la variable “y”.

Para ejecutar la función en el entorno de trabajo:

» **a=cuadrado**

**Introduce el primer elemento del vector= 1**

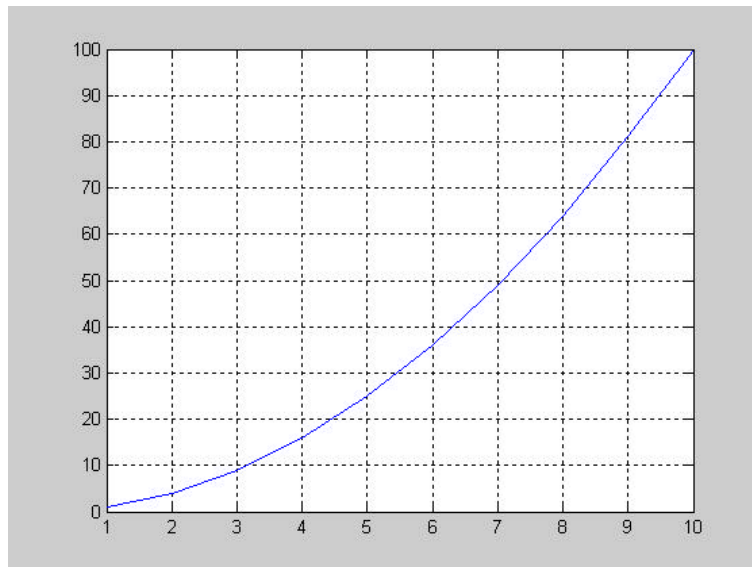
**Introduce el último elemento del vector= 10**

**Incremento entre elementos del vector= 1**

**Los cuadrados son:**

**a =**

**1   4   9   16   25   36   49   64   81   100**



Si se consultan las variables del entorno de *Matlab* tras la ejecución de la función *cuadrado*, se podrá observar que sólo aparece visible la variable *a* (las variables *x*, *t* e *y* son locales a la función *cuadrado*, y no forman parte de la sesión de *matlab*). Si se ejecuta el fichero *script curva*, aparecerán como nuevas variables del entorno de trabajo *w* e *y* empleadas en este último fichero.

En *Matlab* los polinomios se definen como vectores fila, introduciendo los coeficientes de sus elementos en orden descendiente. Observar como se introducen los siguientes polinomios:

$$x^3 + 6x^2 + 5x - 3$$

```
>>p1=[1 6 5 -3]
```

$$2x^4 + x^2 - x + 1$$

```
>>p2=[2 0 1 -1 1]    % si falta alguna potencia se introduce el coeficiente 0.
```

A continuación se muestran algunas operaciones entre polinomios

Obtención de las raíces de un polinomio:

```
>>r1=roots(p1)
```

```
r1= -4.8385
```

```
    -1.5592
```

```
     0.3977
```

Nota: las raíces son siempre vectores columna.

Obtención de polinomios a través de sus raíces:

```
>>r3=[-1;0.5+i;0.5-i]    % vector columna con las raíces
```

```
>>p3=poly(r3)
```

```
p3=1.0000 0 0.2500 1.2500    % p3 = x3 + 0.25x + 1.25
```

Operaciones entre polinomios:

Las operaciones entre polinomios requiere que estos tengan la misma dimensión; se completará con ceros los términos de potencias que no existen para mantener la misma dimensión.

\* SUMA:

```
>>psuma=p1+p2
```

??? Error using  $\rightarrow \pm$

Matrix dimensions must agree.

```
>>p1=[0 p1] % añade un cero en la potencia  $x^4$  para tener la misma dimensión de “p2”
```

```
>>psuma=p1+p2
```

```
psuma= 2 1 7 4 -2 %  $2x^4 + x^3 + 7x^2 + 4x - 2$ 
```

\* MULTIPLICACIÓN:

```
>>pmult=conv(p1,p2)
```

```
pmult= 0 2 12 11 ..... -3
```

\* COCIENTE:

```
>>[cociente,resto]=deconv(pmult,p2)
```

```
cociente= 0 1 6 5 -3
```

```
resto= 0 0 0 0 0 0 0 0 0
```

\* DERIVAR:

```
>>p1deriv=polyder(p1) % derivar el polinomio  $x^3 + 6x^2 + 5x - 3$ 
```

```
p1deriv= 3 12 5 % resultado  $3x^2 + 12x + 5$ 
```

### Resolución de polinomios

\* Especificando un valor para la variable independiente:

```
>>polyval(p1,1) % evalúa p1 para  $x=1$ 
```

```
ans= 9
```

\* Especificando varios valores para la variable independiente:

```
>>x=polyval(p1,[1 2 3]) % evalúa para  $x=1,2,3$ ; resultado en variable x
```

```
>>x= 9 39 93
```

### Descomposición en fracciones simples

En ocasiones interesa obtener los residuos de un cociente polinomial o, lo que es lo mismo, descomponerlo en fracciones simples de la forma:

$$g(s) = \frac{n}{d} = \frac{s^2 - s}{s^2 + 3s + 2} = \frac{r_1}{s - p_1} + \frac{r_2}{s - p_2} + k$$

```
>>n=[1 -1 0];d=[1 3 2];
```

```
>>[r p k]=residue(n,d)
```

```
r= -6
```

```
2
```

```
p= -2
```

```
-1
```

```
k= 1
```

cuyo resultado es,

$$g(s) = \frac{-6}{s+2} + \frac{2}{s+1} + 1$$

La descomposición en fracciones simples de expresiones definidas en el dominio  $z$  se obtendrán mediante el comando **residuez**, dando como resultado

$$g(z) = \frac{n}{d} = \frac{r_1}{1 - p_1 z^{-1}} + \frac{r_2}{1 - p_2 z^{-1}} + \dots + k(1) + k(2)z^{-1} + \dots$$

*MATLAB* permite realizar operaciones matemáticas de manera simbólica — sin realización de cálculo numérico— si previamente se definen como símbolos las variables y funciones con las que se va a operar. La *toolbox* de matemática simbólica define un nuevo tipo de dato: el objeto simbólico, que debe crearse mediante el comando **sym** (o **syms**). Este nuevo objeto es, internamente, una estructura de datos que permite ampliar el concepto de *símbolo* no solo a variables sino también a funciones, matrices y expresiones.

Con variables y funciones simbólicas, *MATLAB* puede realizar cálculo diferencial, integral, algebraico, matricial; permite la resolución de ecuaciones simbólicas y diferenciales; la aplicación de distintas transformadas (*Fourier*, *Laplace*, *Z*).

El comando **sym** permite la creación de funciones simbólicas; el comando **syms** facilita la definición conjunta de varias variables simbólicas:

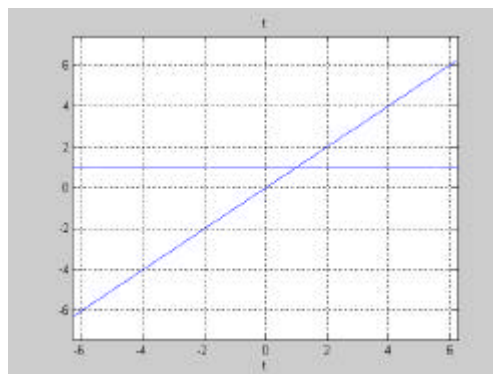
```
>>y1=sym('1')      %Definición de la función y1=1

>>syms t a w s      %Definición de variables t, a, w y s como símbolos

>>y2 = t
>>y4 = exp(-a*t)      %Definición de distintas funciones simbólicas
>>y5 = sin(w*t)
```

La representación gráfica de funciones simbólicas se realiza con el comando **ezplot**.

```
>> ezplot(y1)
>> grid on
>> hold on
>> ezplot(y2)
```



Para obtener la transformada de *Laplace* de funciones simbólicas:

```
>> laplace(y1)
```

```
ans =
```

$$1/s$$

```
>> laplace(y2)
```

```
ans =
```

$$1/s^2$$

```
>> pretty(ans)
```

$$\frac{1}{s^2}$$

```
>> laplace(y4)
```

```
ans =
```

$$1/(s+a)$$

```
>> laplace(y5)
```

```
ans =
```

$$w/(s^2+w^2)$$

```
>> pretty(ans)
```

$$\frac{w}{s^2 + w^2}$$

Para obtener la transformada inversa de *Laplace* de funciones simbólicas:

```
>> F1=(s^2+2*s+3)/(s+1)^3
```

```
>> pretty(F1)
```



$$\frac{s^2 + 2s + 3}{(s + 1)^3}$$

```
>> f1=ilaplace(F1)
```

```
f1 = t^2*exp(-t)+exp(-t)
```

```
>> pretty(f1)
```

$$t^2 \exp(-t) + \exp(-t)$$

Como se comentó al principio, *MATLAB* tiene como estructura básica de datos la matriz; es por esto que la forma de introducir las funciones de transferencia se realiza definiendo dos polinomios: un vector numerador y un vector denominador, que incluye los coeficientes de los términos en  $s$ , en sentido decreciente, de la correspondiente función de transferencia. Una forma más compacta y práctica de definir las funciones de transferencia se obtiene a través del comando **tf** o **zpk**.

Las siguientes funciones de transferencia se pueden definir de la siguiente manera:

$$G_1(s) = \frac{2}{s + 2} \qquad G_2(s) = \frac{9}{s^2 + 1.5s + 9}$$

```
>> n1=2
n1 =
     2
>> d1=[1 2]
d1 =
     1     2
>> n2=9 ; d2=[1 1.5 9] ;
```

siendo  $n_1$ ,  $d_1$  y  $n_2$ ,  $d_2$  los vectores correspondientes a los polinomios numerador y denominador de las funciones de transferencia anteriores.

Nota: en cada línea del editor de comandos (`>>` ) se pueden definir distintas variables separando éstas por punto y coma. Un punto y coma al final de la línea de comandos hace que *MATLAB* acepte esa variable sin mostrarla por pantalla; si se omite el punto y coma, *MATLAB* muestra la variable que se acaba de definir.

Haciendo uso del comando **tf** se obtiene:

```
>> g1=tf(2,[1 2])
```

Transfer function:

$$\frac{2}{s+2}$$

Existe otra posibilidad para definir funciones de transferencia definiendo la variable compleja  $s$  como variable simbólica:

```
>>s=tf('s')
>>g1=2/(s+2)
```

La inclusión de un tercer parámetro en el comando  $tf$ , el tiempo de muestreo  $T$ , permite la definición de funciones de transferencia discreta (cambia la variable  $s$  por  $z$ ):

```
» g1z=tf(2,[1 2],1)
```

Transfer function:

$$\frac{2}{z+2}$$

Sampling time: 1

Si se define como variable simbólica la  $z$ , la construcción de funciones de transferencia discretas se simplifica:

```
>>z=tf('z',1)
>>g1z=2/(z+2)
```

Para definir una función de transferencia a través de sus ceros, polos y ganancia se empleará el comando **zpk**. Un sistema con un cero doble en 1, un polo en 0 y en 1, y una ganancia de 10 se definirá de la siguiente manera:

```
» g3=zpk([1;1],[1;0],10)
```

Zero/pole/gain:

10 (s-1)^2

-----

s (s-1)

Si se especifica el tiempo de muestreo T, el comando *zpk* devuelve la función de transferencia discreta:

» g3z=**zpk**([1;1],[1;0],10, 1)

Zero/pole/gain:

10 (z-1)^2

-----

z (z-1)

Sampling time: 1

---

## CONVERSIÓN ENTRE FUNCIONES DE TRANSFERENCIA

---

La conversión de funciones de transferencia, bien de la forma de cociente de polinomios, bien a través de la ganancia (K), polos (p) y ceros (z) del tipo:

$$G(s) = K \cdot \frac{\prod_m (s - z_i)}{\prod_n (s - p_i)} \quad m \leq n$$

se hace mediante los comandos **tf2zp** y **zp2tf**. También los comandos **tf** y **zpk** permiten la conversión de funciones expresadas previamente de forma compacta.

>> [z2,p2,k2] = **tf2zp** (n2, d2)

z2 =

Empty matrix: 0-by-1

p2 =

-0.7500 + 2.9047i

-0.7500 - 2.9047i

$$k_2 = \frac{9}{9}$$

que se corresponde con la función,

$$G(s) = 9 \cdot \frac{1}{(s + 0.75 - 2.9i) \cdot (s + 0.75 + 2.9i)}$$

Por el contrario, el aspecto que tendría la función de transferencia de un sistema con una ganancia igual a 18, un cero en -2, y tres polos en 0, -1 y -5 se obtendría fácilmente mediante el comando **zp2tf**.

```
>>[n3,d3] = zp2tf (-2, [0;-1;-5], 18)
```

n3 =

```
0 0 18 36
```

d3 =

```
1 6 5 0
```

Se observa que las nuevas variables n3 y d3 contienen los polinomios del numerador y denominador de la nueva G(s). No es necesario definir nuevas variables para estos valores de K, z y p; se pueden incluir sus valores directamente en el argumento, a la derecha del comando *MATLAB*, como se ha hecho en el último ejemplo. Recordar que los ceros y polos se definen como vectores columna (elementos separados por punto y coma); si no hay ceros se introduce el vector vacío [].

Para convertir la función g3 compacta, definida en su forma cero-polo-k, a una función polinómica g4:

```
» g3
```

Zero/pole/gain:

```
10 (s-1)^2
```

```
-----
```

```
s (s-1)
```

```
>> g4=tf(g3)
```

Transfer function:

$$10 s^2 - 20 s + 10$$

-----

$$s^2 - s$$

Para convertir la función g5 compacta, definida de forma polinómica, a una función g6 de la forma cero-polo-k:

```
» g5=tf(n,d)
```

Transfer function:

$$s^2 - s$$

-----

$$s^2 + 3 s + 2$$

```
» g6=zpk(g5)
```

Zero/pole/gain:

$$s (s-1)$$

-----

$$(s+2) (s+1)$$

---

*Matlab* ofrece la posibilidad de realizar conversiones entre funciones de transferencia continuas (plano  $s$ ) y discretas (plano  $z$ ) a través de varios comandos: **c2d**, **d2c**, **d2d**.

La conversión de sistemas continuos a discretos se realiza con el comando **c2d**. Al comando **c2d** se le pasan los correspondientes valores del sistema continuo, el tiempo de muestreo **T** y el método de conversión. Los métodos de conversión implementados en este comando son con bloqueador de orden cero, bloqueador de orden 1, aproximación bilineal y otros.

Dada la función de transferencia de un sistema continuo,

$$G_2(s) = \frac{9}{s^2 + 1.5s + 9}$$

su función de transferencia discreta equivalente  $BG(z)$ , empleando un bloqueador de orden cero y un tiempo de muestreo de  $T=0.2$  seg. tiene el siguiente aspecto:

```
>> g2z= c2d (g2, 0.2, 'zoh' )
```

Transfer function:

$$0.1585 z + 0.1433$$

-----

$$z^2 - 1.439 z + 0.7408$$

Sampling time: 0.2

Opciones conversión: 'zoh' para bloqueador de orden cero.

'foh' para bloqueador de orden uno.

'tustin' para aproximación bilineal.

(consultar *help c2d* para otras opciones de conversión)

La conversión de sistemas discretos a continuos se realiza con el comando **d2c**. De manera similar al ejemplo anterior:

```
» g2=d2c(g2z,'zoh')
```

Transfer function:

$$9$$

-----

$$s^2 + 1.5 s + 9$$

Para obtener la función de transferencia discreta, de un sistema discreto previamente dado, con un nuevo tiempo de muestreo  $T$  se empleará el comando **d2d**. A partir de la  $g2z$  del ejemplo anterior, se procede a obtener la nueva función como resultado de muestrear ahora con un tiempo  $T=1s$ .

```
» g4z=d2d(g2z,1)
```

Transfer function:

```

1.431 z + 0.7109
-----
z^2 + 0.9184 z + 0.2231
Sampling time: 1

```

Este resultado es el mismo que se hubiera obtenido discretizando el sistema continuo original con el nuevo  $T=1s$  usando el comando *c2d*:

```

» g4z=c2d(g2,1,'zoh')
Transfer function:
1.431 z + 0.7109
-----
z^2 + 0.9184 z + 0.2231
Sampling time: 1

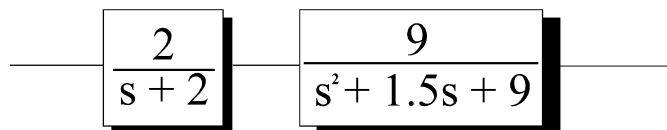
```

---

## TRABAJANDO CON BLOQUES EN MATLAB

---

Las operaciones más comunes del álgebra de bloques se resuelven sencillamente con tres comandos en *MATLAB*: **series**, **parallel** y **feedback**. El primero obtiene la función de transferencia del conjunto formado por sólo dos bloques en serie; el segundo obtiene la función de transferencia del conjunto formado por dos bloques en paralelo; por último, el comando *feedback* da la función de transferencia de un sistema realimentado. El sistema equivalente de la figura adjunta es,



```

>> g7 = series (g1, g2)           % ver también el comando conv

```

```

Transfer function:
18
-----
s^3 + 3.5 s^2 + 12 s + 18

```

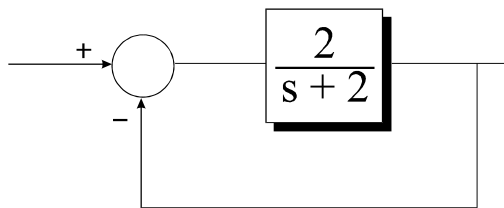


Otra posibilidad es realizar directamente la operación algebraica,

```
>> g7 = g1*g2
```

Nota: tambien se puede emplear la forma no compacta de las funciones de transferencia, esto es, mediante el uso de variables vectoriales num y den para definir las funciones de transferencia.

La obtención de la función de transferencia equivalente del sistema de la figura realimentado unitariamente será:



```
» g8=feedback(g1,1)
```

Transfer function:

$$\frac{2}{s + 4}$$

La realimentación por defecto es negativa; con realimentación positiva añadir 1 al comando *feedback*.

Otra posibilidad es realizar directamente la operación algebraica,

```
» g8=g1/(1+g1)
```

Transfer function:

$$\frac{2s + 4}{s^2 + 6s + 8}$$

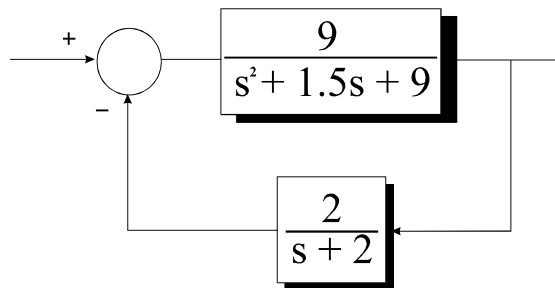
En su resultado no hay cancelación de polos y ceros, aspecto éste que se resuelve con el comando **minreal**, dando el mismo resultado que el obtenido con el comando **feedback**:

» g8=**minreal**(g8)

Transfer function:

$$\frac{2}{s + 4}$$

En el sistema de la figura, su función de transferencia equivalente (g9) se obtiene de la siguiente manera:



» g9=**feedback**(g2,g1)

Transfer function:

$$\frac{9s + 18}{s^3 + 3.5s^2 + 12s + 36}$$

O también,

» g9=g2/(1+(g2\*g1))

La función de transferencia equivalente del conjunto de la figura, formado por g1 en paralelo con g2 será:

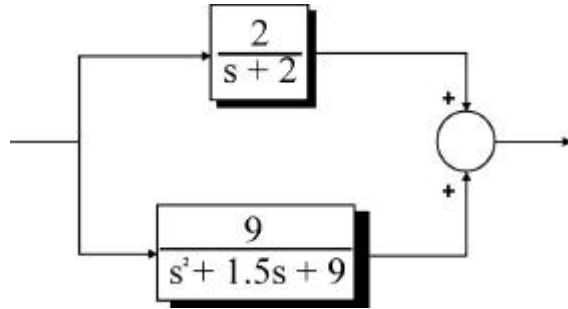
» g10=parallel(g1,g2)

Transfer function:

$$2s^2 + 12s + 36$$

-----

$$s^3 + 3.5s^2 + 12s + 18$$




---

### OBTENCIÓN DE RESPUESTAS CON MATLAB.

---

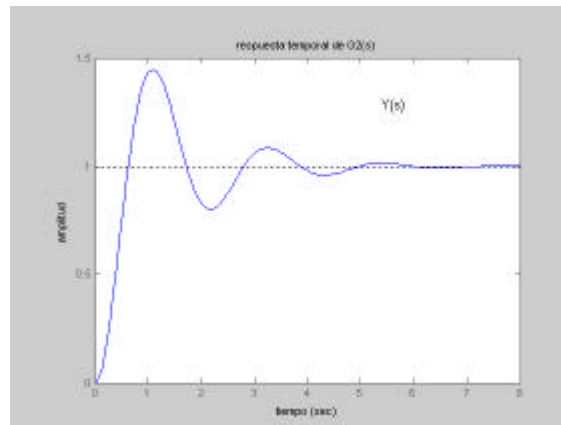
Definidas previamente las funciones de transferencia de distintos sistemas, se pasa a continuación a estudiar las respuestas temporales de los mismos con *MATLAB*, lo cual va a permitir analizar el comportamiento transitorio y permanente de la respuesta. Conviene recordar que, para este tipo de análisis, las dos señales típicas son el escalón y el impulso unitario. Para ello, *MATLAB* proporciona los comandos **step** e **impulse**. Mediante el comando **step** se obtiene la respuesta temporal de un sistema lineal ante entrada escalón unitario; con el comando **impulse** se obtiene la respuesta ante entrada impulso.

Hay dos formas de trabajar con estos comandos: sin argumentos o con argumentos a la izquierda del comando. En el primer caso *MATLAB* genera directamente una gráfica donde representa la respuesta temporal; con argumentos a la izquierda del comando, los valores de la respuesta no se representan gráficamente sino que quedan grabados en las nuevas variables definidas en dicho argumento. A continuación se muestra la respuesta temporal de G2(s) en ambos casos.

$$G2(s) = \frac{9}{s^2 + 1.5s + 9}$$

>> **step** (g2)

dando como resultado la gráfica de abajo.



Observar que la gráfica aparece con un título de cabecera, un texto dentro de la gráfica y unos textos para los ejes de abscisa y ordenada. Esto se consigue añadiendo los siguientes comandos de personalización de gráficos:

```
» title('respuesta temporal de G2(s)')  
» ylabel('Amplitud')  
» xlabel('Tiempo')  
» gtext('Y(s)')
```

La respuesta al impulso del sistema  $G2(s)$  se obtiene de la siguiente manera:

```
>> impulse (g2)
```

Disponiendo argumentos a la izquierda del comando se obtienen los siguientes resultados, donde las nuevas variables  $y2$ ,  $t2$  representan los vectores de salida y tiempo respectivamente:

```
» [y2,t2]= step(g2)    % (no dibuja gráfica alguna)  
y2 =  
    0  
    0.0194  
    0.0745
```

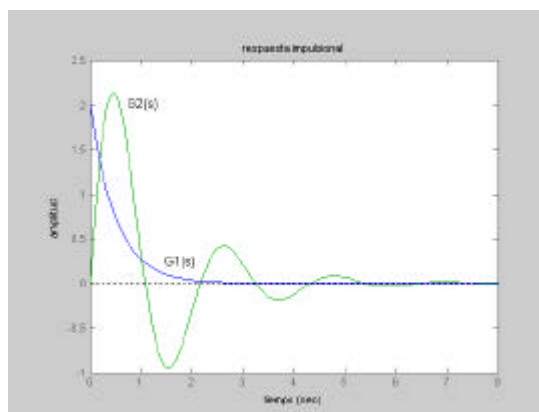
0.1597  
 0.2688  
 0.3955, etc.  
 t2 =  
 0  
 0.0669  
 0.1339  
 0.2008  
 0.2677  
 0.3346, etc.

Se puede variar el tiempo en la gráfica de la respuesta si se le especifica un tercer parámetro con el nuevo valor de tiempo:

`>>step(n2,d2,5)`

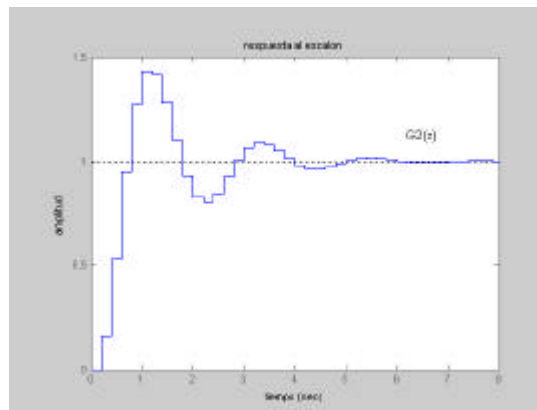
Se pueden representar varias respuestas sobre una misma grafica de la siguiente manera:

`» impulse(g1,g2)`



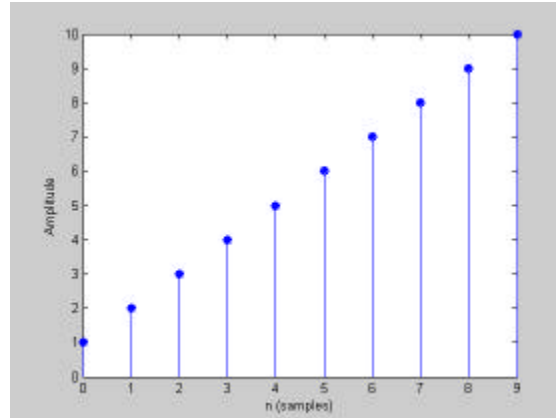
La forma de representar la respuesta temporal ante entrada escalón, impulso u otra, dependerá del tipo de función de transferencia, esto es, continua  $G(s)$  o discreta  $G(z)$ :

» **step(g2z)** %respuesta ante entrada escalón de un sistema (g2z) discreto.



## OTRAS RESPUESTAS

- Respuesta de un filtro o sistema discreto a la secuencia impulso.  
Obtiene la salida de un sistema discreto (o filtro), expresado como cociente de polinomios en z, cuando la entrada es la secuencia impulso {1,0,0,...0}  
» **impz(1,[1 -2 1])**



La gráfica anterior corresponde a la respuesta de un sistema definido por la ecuación,

$$G(z) = \frac{1}{1 - 2z^{-1} + z^{-2}}$$

Es posible especificar el nº de muestras a representar,

» **impz**(1,[1 -2 1],5)                      %genera 5 muestras

Se puede especificar la frecuencia de muestreo entre muestras f1,

» **impz**(1,[1 -2 1],5,f1)                      %genera 5 muestras espaciadas un tiempo T=1/f1

Se pueden almacenar los valores de la gráfica en variables definidas con argumentos a la izquierda del comando,

» [amplitud, muestra]=**impz**(1,[1 -2 1])

- Respuesta de un filtro o sistema discreto ante cualquier secuencia de entrada (resolución de la ecuación en diferencias).

Obtiene la respuesta de un sistema ante cualquier tipo de entrada especificada mediante su secuencia; equivale a resolver la ecuación en diferencias de dicho sistema:

$$a_1 y_k + a_2 y_{k-1} + \dots = b_1 x_k + b_2 x_{k-1} + \dots$$

» y=**filter**(1,[1 -2 1],[1 1 1 1])

y =

1    3    6    10

La variable y={ 1 3 6 10} representa la secuencia de salida ante secuencia de entrada x={1 1 1 1} del sistema dado por la ecuación,

$$\frac{Y(z)}{X(z)} = \frac{1}{1 - 2z^{-1} + z^{-2}}$$

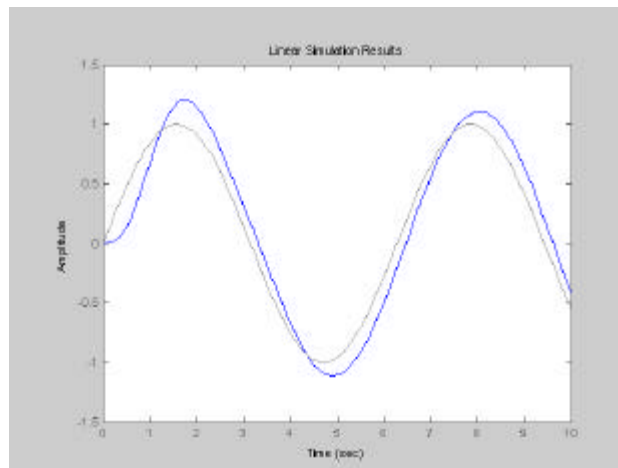
- Respuesta de un sistema continuo ante cualquier tipo de entrada.  
Permite la obtención de respuestas temporales de sistemas continuos especificando el tipo de señal de entrada y el tiempo de duración.

» t=0:0.1:10;

» u=sin(t);

» **lsim**(g2,u,t)

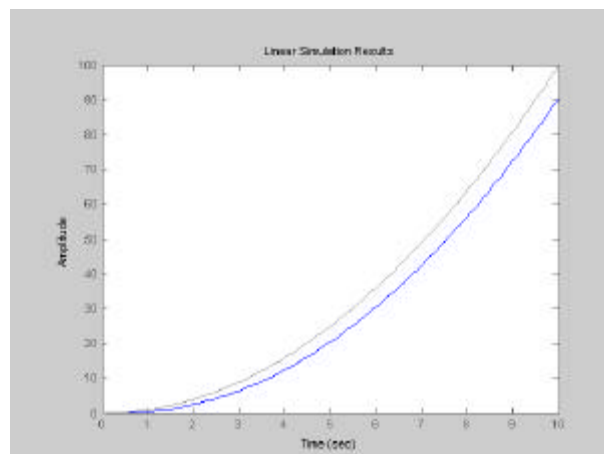
La anterior secuencia de comandos obtiene la respuesta del sistema g2 cuando se excita con una señal senoidal, en el intervalo de tiempo 0 a 10s con un total de 100 valores (ver figura adjunta).



---

```
» t=0:0.1:10;  
» u= t.^2;      %el punto entre t y ^ es necesario para extraer la potencia de t  
                 definida como variable simbólica.  
» lsim(g1,u,t)
```

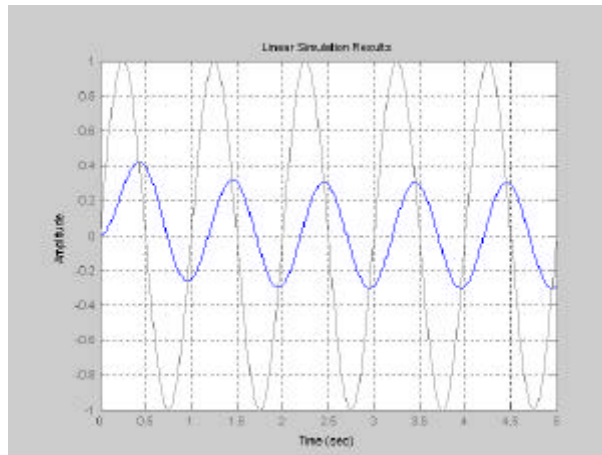
La anterior secuencia de comandos obtiene la respuesta del sistema g1 cuando se excita con una señal de tipo parábola ( $t^2$ ), en el intervalo de tiempo 0 a 10s con un total de 100 valores (ver figura adjunta).



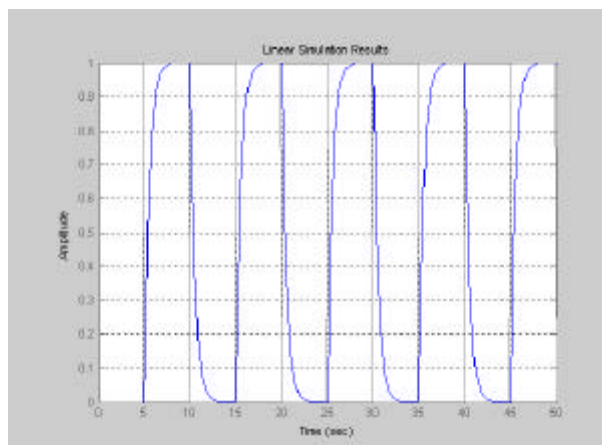


Para obtener la respuesta ante señales senoidales ('sin'), onda cuadrada ('square') o pulso ('pulse') se puede emplear el comando **gensig**; basta especificar el tipo de señal y el periodo de la misma (la amplitud de la señal es siempre unitaria):

```
» [u t]=gensig('sin',1)    %Genera una senoide de periodo 1; u es amplitud y t tiempo
» lsim(g1,u,t)              %Respuesta del sistema g1 ante la entrada senoidal previa.
```



```
» [u t]=gensig('square',10)    %Genera una onda cuadrada de periodo 10
» lsim(g1,u,t)                  %Respuesta del sistema g1 ante la entrada anterior.
```



En el apartado anterior se obtuvieron las gráficas de las respuestas de distintos sistemas. Para modificar sus características, existe una colección de comandos básicos de manejo de gráficos que se pasan a describir a continuación.

- **HOLD ON:** congela la gráfica que esté activa de forma que las sucesivas respuestas se visualizan en la misma.
- **HOLD OFF:** descongela la gráfica previamente activada.
- **FIGURE (4):** activa una nueva gráfica (la número 4) y permanece activa para representaciones futuras.
- **GRID ON/OFF:** activa o desactiva una rejilla sobre la gráfica actual.
- **SGRID:** activa sobre el lugar de las raíces de un sistema continuo una rejilla de líneas donde los parámetros  $\xi$  y  $\omega_n$  son constantes.
- **ZGRID:** activa una rejilla sobre el lugar de las raíces de un sistema discreto una rejilla de líneas donde los parámetros  $\xi$  y  $\omega_n$  son constantes.
- **AXIS ([Xmín,Xmáx,Ymín,Ymáx]):** escalado de ejes definido por el usuario.
- **AXIS (AXIS):** congela el escalado actual de los ejes en sucesivas representaciones.
- **PLOT (Y):** visualiza en una gráfica los valores de la variable Y; se pueden representar parejas de puntos (`plot(X,Y)`) e incluso el tipo de línea para su representación (ver *help plot*).
- **STEM (Y):** visualiza la secuencia discreta Y; al igual que con *plot*, se puede representar la secuencia Y en los puntos indicados por un vector X, así como el tipo de representación (`stem(X,Y,tipolínea)`).

Muchos son los comandos incluidos en la *toolbox* de control para el análisis de los sistemas de control. Los más habituales se pueden agrupar en tres categorías:

1. Extracción de características.
2. Representaciones polares.
3. Representaciones frecuenciales.

### 1. Extracción de características

---

- Obtención de polos de un sistema,

» `g2`

Transfer function:

9

-----

$s^2 + 1.5 s + 9$

» `pole(g2)` % obtención de los polos del sistema `g2`.

`ans = -0.7500 + 2.9047i`

`-0.7500 - 2.9047i`

- Obtención de ceros de un sistema,

» `zero(g2)` % obtención de los ceros del sistema `g2`.

- Obtención de la ganancia estática de un sistema,

» `k=dcgain(g2)`

`k = 1.0000`

- Obtención de  $\xi$  y  $\omega_n$ ,

» `damp(g2)`

Eigenvalue

Damping

Freq. (rad/s)

-7.50e-001 + 2.90e+000i	2.50e-001	3.00e+000
-7.50e-001 - 2.90e+000i	2.50e-001	3.00e+000

Si se disponen argumentos a la izquierda del comando *damp* el resultado se almacena en dos variables:

```
» [W z]=damp(g2)
```

```
W = 3.0000
```

```
3.0000
```

```
z = 0.2500
```

```
0.2500
```

- Cancelación de polos y ceros de un sistema.

Este comando obtiene el sistema simplificado como resultado de eliminar ceros del numerador con polos del denominador.

```
» g4
```

Transfer function:

```
10 s^2 - 20 s + 10
```

```
-----
```

```
s^2 - s
```

```
» g10=minreal(g4)
```

Transfer function:

```
10 s - 10
```

```
-----
```

```
s
```

- Obtención de los ceros y k de un sistema.

```
» [ceros k]=tzero(g4)
```

```
ceros = 1
```

```
1
```

```
k = 10
```

- Obtención de la FDT de un sistema de 2º orden especificando  $\xi$  y  $\omega_n$ .

Devuelve los polinomios numerador y denominador de la FDT.

```

» [num den]=ord2(1,0.7)
num =    1
den =    1.0000    1.4000    1.0000

```

La función de transferencia de un sistema con  $\xi=0.7$  y  $\omega_n=1$  resulta ser,

$$\frac{num}{den} = \frac{1}{s^2 + 1.4s + 1}$$

## 2. Representaciones polares

---

- Situación de polos y ceros en el plano complejo,

```
» g2
```

Transfer function:

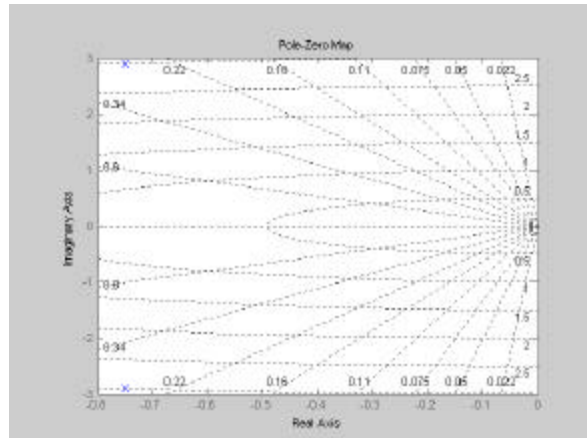
9

-----

$s^2 + 1.5s + 9$

```
» pzmap(g2)
```

```
» sgrid
```



Con parámetros a la izquierda de la función *pzmap*:

```
» [polos ceros]=pzmap(g2)
```

polos = -0.7500 + 2.9047i

-0.7500 - 2.9047i

ceros = Empty matrix: 0-by-1

De la misma manera se puede aplicar a un sistema de naturaleza discreta:

```
» g2z
```

Transfer function:

$$0.1585 z + 0.1433$$

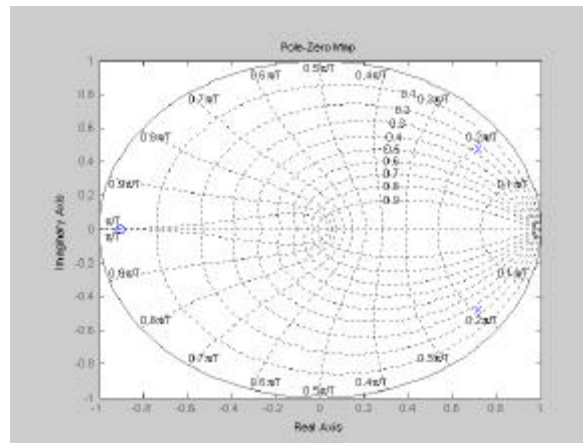
-----

$$z^2 - 1.439 z + 0.7408$$

Sampling time: 0.2

» **pzmap(g2z)**

» **zgrid**

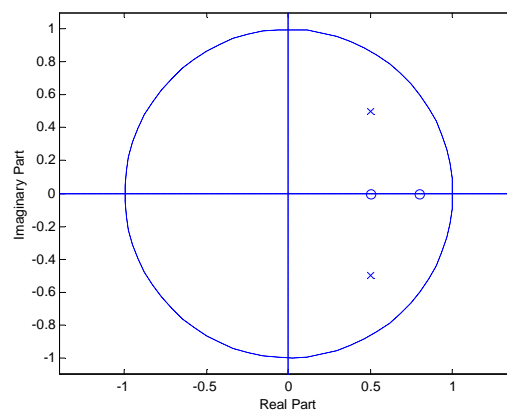


- Localización de polos y ceros con respecto al círculo unidad.

Este comando es útil para determinar la estabilidad de los sistemas discretos puesto que localiza los polos y ceros con respecto al círculo unidad. Los polos y ceros se han de introducir en vectores columna. Si se especifica con vectores fila *Matlab* interpreta que son las expresiones *num* y *den* de la función de transferencia discreta .

Nota: se ha de completar con ceros los elementos que no existan en los vectores *num* y *den* para que el orden de ambos sea igual.

» **zplane([0.5;0.8],[0.5+0.5i;0.5-0.5i])**



El mismo resultado se obtendría especificando los polinomios de la función de transferencia de la siguiente manera:

$$\frac{num}{den} = \frac{z^2 - 1.3z + 0.4}{z^2 - z + 0.5}$$

» **zplane**([1 -1.3 0.4],[1 -1 0.5])

- Obtención del lugar de las raíces.

Dibuja las trayectorias de los polos en cadena cerrada del sistema especificado cuando varia la K.

» **g7**

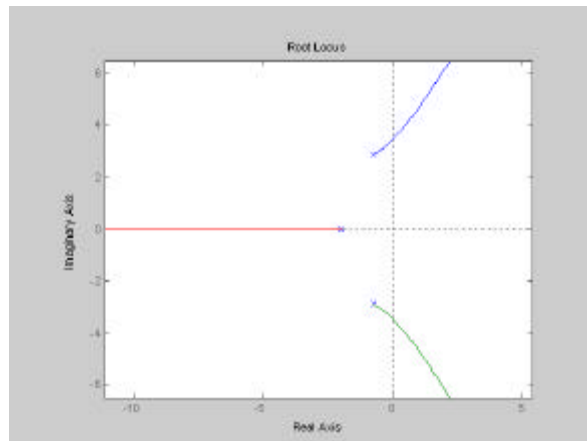
Transfer function:

18

-----

$s^3 + 3.5 s^2 + 12 s + 18$

» **rlocus**(g7)



Especificando un valor de K, representa la posición de los polos en el plano complejo correspondientes a ese valor:

» **rlocus**(g7, K)

Con argumentos a la izquierda, devuelve todas las posiciones de los polos y los valores de K a lo largo de las trayectorias del lugar:

» [polos, k]=**rlocus**(g7)

O sólo la posición del polo para un valor de K especificado:

» polos=**rlocus**(g7,10)                      % polos de g7 con K=10

polos = -6.4267

$$1.4633 + 5.3542i$$

$$1.4633 - 5.3542i$$

Si se desea conocer un determinado valor sobre el lugar de las raíces:

» **rlocfind**(g7)      %(espera a que se seleccione con el ratón un punto sobre el lugar)

Select a point in the graphics window

selected\_point = -0.0161 + 3.4152i

ans = 1.2480

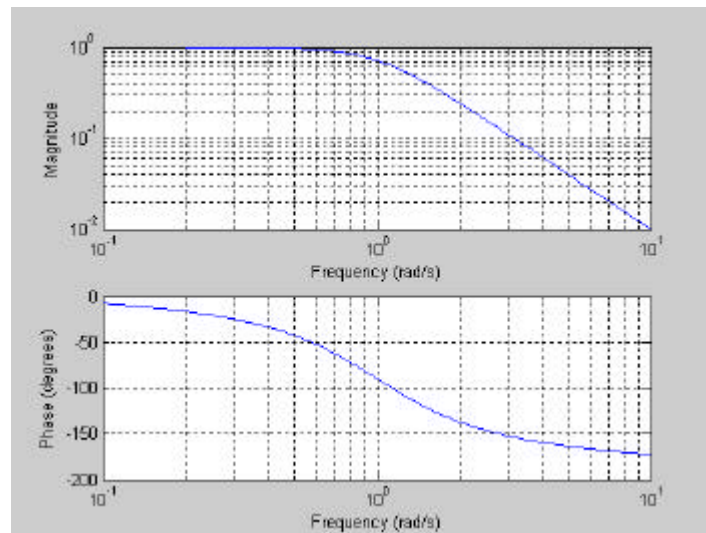
### 3. Representaciones frecuenciales

---

- Obtención de diagramas de amplitud y de fase de un sistema continuo.

Se han de especificar los polinomios *num* y *den* del correspondiente sistema:

» **freqs**(1, [1 1.4 1])



Se puede especificar un determinado margen de frecuencias para la representación mediante un vector fila :

» **freqs**(1, [1 1.4 1],[0.1 1 10 100])



Se pueden almacenar los correspondientes valores complejos de la gráfica en los puntos de frecuencia indicados en el vector fila:

» puntos=**freqs**(1,[1 1.4 1],[0.1 1 10 100])

puntos =

0.9903 - 0.1400i      0 - 0.7143i -0.0099 - 0.0014i -0.0001 - 0.0000i

- Obtención del diagrama de bode

Se puede introducir la función de transferencia definida de manera compacta. Si se desea dibujar el diagrama en un rango determinado de frecuencias estas se definirán entre llaves { }. Asimismo, se pueden almacenar los valores de amplitud y fase disponiendo argumentos a la izquierda del comando *bode*.

» **bode**(g2)

» **bode**(g2,{0.1,100})

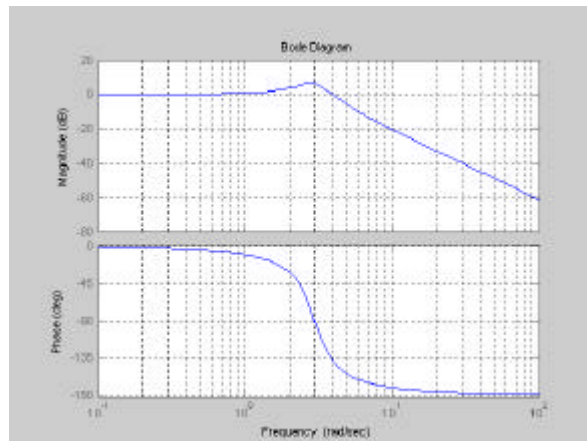
» [ampl fase]=**bode**(g2)

ampl(:,1) = 1.1057

ampl(:,2) = 1.1183

fase(:,1) = -10.6197

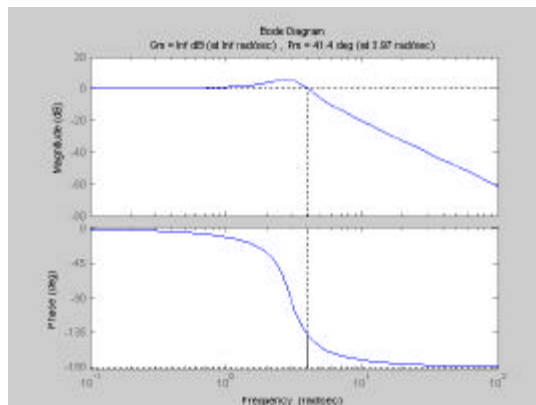
fase(:,2)= -11.3184



- Obtención del margen de ganancia, de fase y frecuencias de cruce.

Sin argumentos a la izquierda del comando *margin*, representa el diagrama de bode especificando los márgenes de ganancia y de fase.

» **margin**(g2)



Con argumentos a la izquierda del comando, se almacenan los valores de margen de ganancia, margen de fase, frecuencia de cruce de ganancia y frecuencia de cruce de fase:

» [Mg Mf Wg Wf]=**margin**(g2)

Mg = Inf        % este dato no se da en dB.

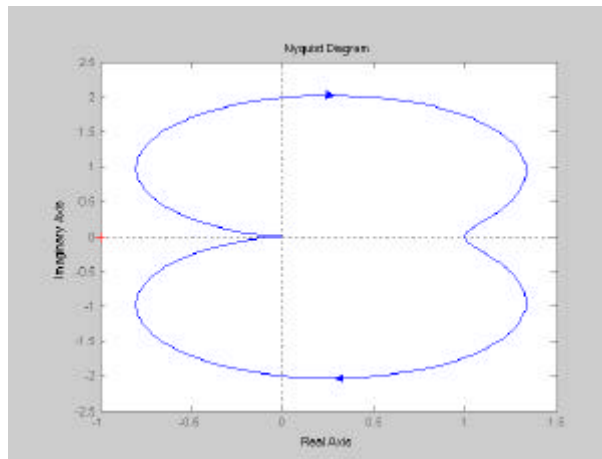
Mf = 41.4096   % son grados

Wg = Inf        % son rad/s

Wf = 3.9686    % son rad/s

- Obtención del diagrama de Nyquist.

» **nyquist**(g2)

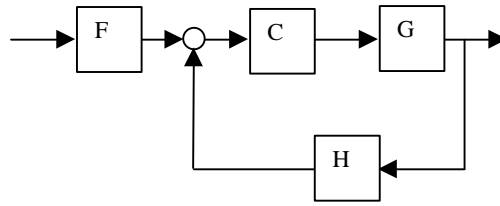


---

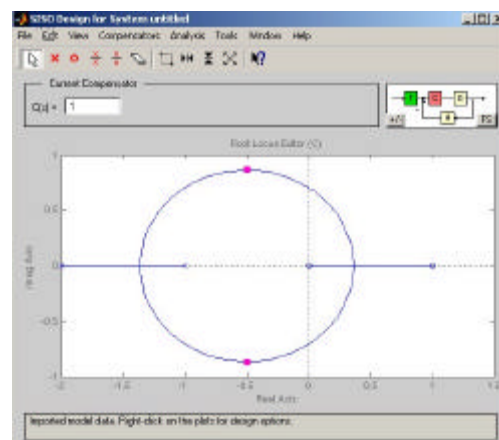
## INTERFASE GRÁFICA *RLTOOL*

---

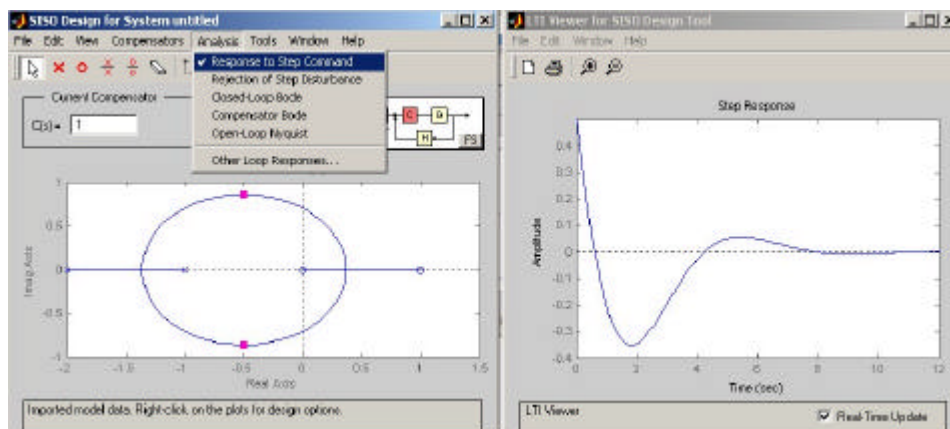
*MATLAB* dispone de una herramienta gráfica, llamada ***rltool***, que permite el análisis de los sistemas de control realimentados mediante el lugar de las raíces. La estructura de control que es posible analizar debe seguir el esquema de la figura adjunta (aunque existen otras tres distintas configuraciones que se pueden seleccionar dentro de la interfase *rltool*). Las funciones de transferencia de los bloques F (prefiltro), C (compensador), G (planta) y H (sensor), deben definirse previamente en el editor de comandos de *matlab*. Hecho esto, solo queda ejecutar la interfase tecleando **>>rltool**.



A continuación, se importarán las funciones de transferencia necesarias (**File®Import**), asignándolas a los correspondientes bloques de la estructura de control de la figura. Una vez hecho esto, y de manera automática, se obtiene el lugar de las raíces del sistema, en el que aparecen indicadas las posiciones de los polos en cadena cerrada.

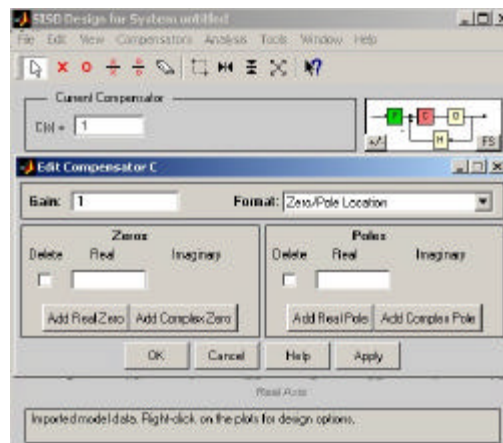


Bajo el menú **Analysis** se encuentra la forma de obtener distintas respuestas del sistema realimentado, así como los diagramas de *bode* y *nyquist*.



El menú **View** permite consultar la posición de los polos en cadena cerrada, los datos del sistema, así como distintas opciones de carácter informativo.

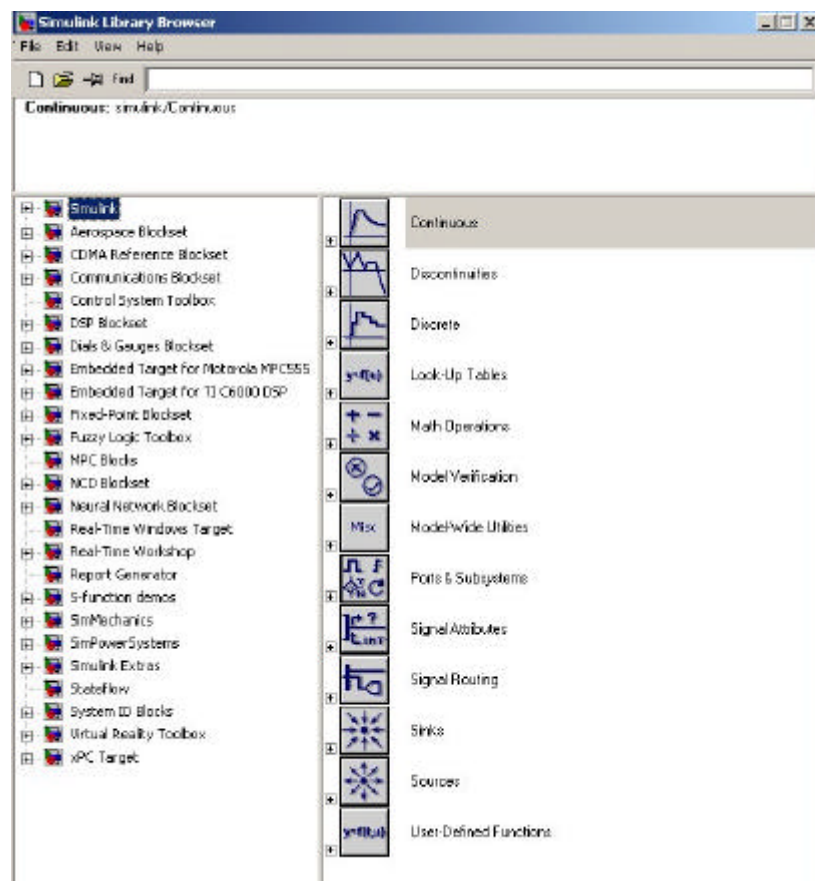
Pinchando sobre la ventana **Current Compensator**, se puede definir el regulador (*compensator*) que se desee mediante la adición de sus polos, ceros y el valor de la ganancia K. Los polos y ceros se pueden insertar directamente sobre el lugar de las raíces pinchando en los iconos **X**, **O**, los cuales pueden ser desplazados con el ratón; el cuadrado rojo que aparece sobre el lugar indica el punto de funcionamiento del sistema (posición de los polos) para los valores introducidos. Cada modificación que se realice sobre el lugar, la interfase actualiza todos sus datos (polos, ganancia, respuestas, etc.).



En todas las gráficas generadas por la interfase *rltool*, es posible activar rejillas (grid), obtener distintas características del sistema (*characteristics* :  $t_p$ ,  $t_s$ ,  $t_r$ ,  $y_\infty$ ), generar respuestas al escalón, impulso, bode (*plot types*). Todas estas utilidades se activan, sobre la gráfica correspondiente, con el botón derecho del ratón.

*Simulink* es la interfase gráfica de simulación de *MATLAB*. Permite el análisis y estudio de sistemas (de distintas disciplinas de la técnica) mediante la simulación de los modelos construidos en *simulink*. La creación de estos modelos es sencilla e intuitiva, ya que se forman mediante la interconexión gráfica de distintos bloques. Dentro del editor de modelos de *simulink* se insertan bloques, se conectan y se parametrizan para su posterior simulación.


Dentro de *simulink* es posible crear y simular modelos mecánicos, eléctricos, electrónicos, aeronáuticos, etc. gracias a la gran variedad de bloques (*blocksets*) de los que dispone. Estos conjuntos de bloques se encuentran agrupados en la *simulink library browser*, que se despliega al ejecutar *simulink*, y que presenta el aspecto de la figura



La librería principal de bloques se encuentra bajo la carpeta llamada *simulink*, y en ella aparecen los bloques agrupados en las siguientes categorías: continuos, no lineales (*Discontinuities*), discretos, tablas, operaciones matemáticas, verificación de modelos, puertos y subsistemas, señales, dispositivos de salida (*Sinks*), generadores (*Sources*).

### Creación de un modelo

---

Para ejecutar *simulink* pulsar sobre el botón  en el menú principal, o bien teclear **simulink** desde el editor de comandos de *matlab* (aparecerá una nueva ventana con todas las librerías disponibles). A continuación, y desde el menú principal, se seleccionará *File®New®Model* que abrirá la ventana de edición donde se creará el modelo *simulink* para su posterior simulación. Los distintos bloques del modelo a crear se han de seleccionar primero en las correspondientes librerías, después arrastrar y soltar en la ventana de edición; por último, interconectar entre si.

Pulsando dos veces sobre cada bloque se despliega la ventana de parámetros correspondiente a dicho bloque; cada campo que aparece en ella se rellenará con los datos requeridos para el modelo que se va a simular.

Una vez creado el modelo y parametrizados todos sus bloques se procede a la simulación seleccionando en el menú *Simulation®Start*. Para detener la simulación seleccionar *Simulation®Stop*. Los tiempos de la simulación del modelo, tiempo de inicio (*start time*) y tiempo de parada (*stop time*), se especifican en *Simulation parameters*.

### Ejemplo

---

Se desea simular un sistema de control en cadena cerrada con la incorporación de un regulador PID. La señal de referencia será de tipo escalón. Se visualizarán en una misma gráfica la evolución de las señales de salida y el error del sistema; los valores de estas dos señales se pasarán, en formato vector, al editor de *matlab*, como dos nuevas variables con el nombre "salida" y "error".

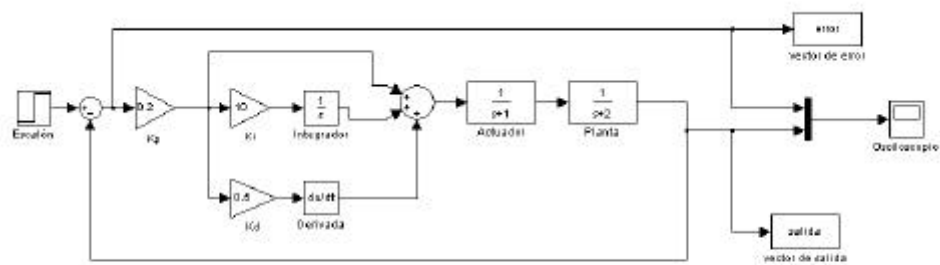
### Datos del sistema:

- Actuador  $\rightarrow G_1(s) = \frac{1}{s+1}$
- Planta  $\rightarrow G_2(s) = \frac{1}{s+2}$
- PID  $\rightarrow R(s) = 0.2 \cdot \left[ 1 + \frac{10}{s} + 0.5 \cdot s \right]$
- Señal de referencia  $\rightarrow$  escalón unitario

### Datos de la simulación:

- Start time: 0 segundos
- Stop time: 20 segundos

### Diagrama de bloques en simulink:

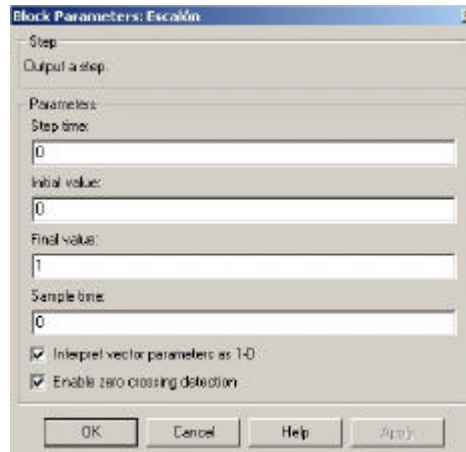


### Ubicación de bloques en la Simulink Library Browser:

Nomenclatura	Librería	Nombre bloque
Escalón	Sources	Step
+-, +++	Math Operations	Sum
Kp, Ki, Kd	Math Operations	Gain
Integrador	Continuous	Integrator
Derivada	Continuous	Derivative
Actuador	Continuous	Transfer Fcn
Planta	Continuous	Transfer Fcn
Vector de salida	Sinks	To Workspace
Vector de error	Sinks	To Workspace
Osciloscopio	Sinks	Scope

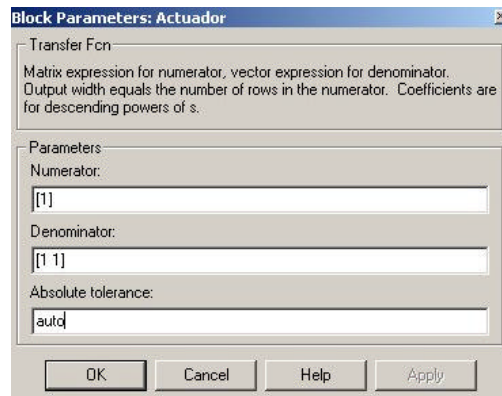
### Parametrización de los bloques del modelo:

- Escalón (*step*). Se ajustará un valor inicial de 0 y un valor final de 1. El tiempo de transición (*step time*) igual a 0 segundos.



- Comparador y sumador (*sum*). Para el primero se especificará la lista de signos  $\pm$ , mientras que para el segundo será  $++$ .
- $K_p$ ,  $K_i$ ,  $K_d$  (*gain*). Se ajustará en el parámetro *gain* el valor de 0.2, 10 y 0.5 respectivamente.
- Los bloques integrador y derivada no llevan ningún ajuste de parámetros.
- Actuador y planta (*transfer fcn*). Se especificarán los vectores numerador y denominador (con los coeficientes en potencias de  $s$  en sentido decreciente) del dispositivo actuador (1,[1 1]), y de la planta (1,[1 2]).





### Resultado de la simulación:

En la figura aparecen las señales correspondientes a la salida del sistema y al error, representadas durante los primeros 20 segundos de su evolución.

