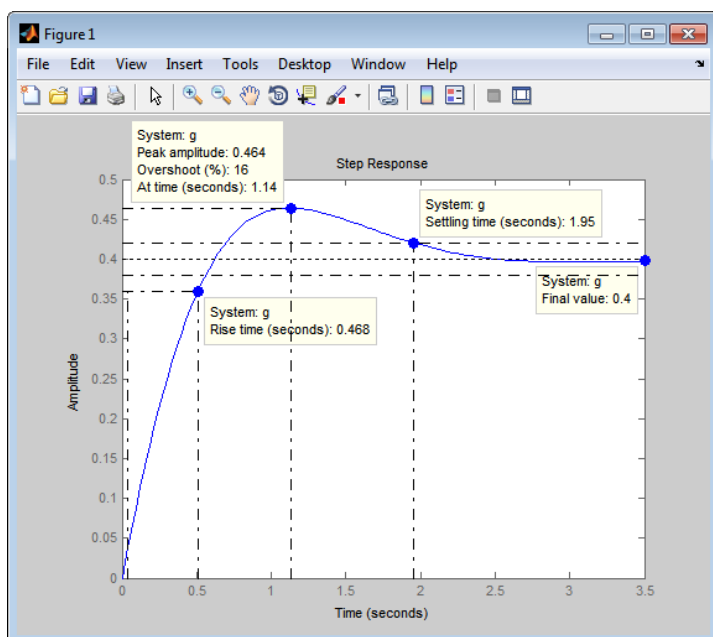


# Introducción a MATLAB

```
g =  
  
      s + 2  
      -----  
      s^2 + 3 s + 5  
  
Continuous-time transfer function.  
  
>> step(g)
```



## 0 Contenido

0	Contenido .....	2
1	MATLAB® y SIMULINK® .....	3
2	Comandos básicos de MATLAB para las prácticas de Regulación .....	3
2.1	Comandos básicos. Variables, vectores y matrices.....	3
2.2	Valores complejos .....	5
2.3	Polinomios.....	6
2.4	Representaciones gráficas 2D y 3D.....	8
2.5	Funciones de Transferencia.....	11
2.6	Respuesta en el tiempo .....	16
2.7	Análisis .....	18
2.8	Respuesta en frecuencia.....	21
2.9	Lugar de las raíces. ....	25

## 1 MATLAB® y SIMULINK®

MATLAB (MATrix LABoratory) es un lenguaje de alto nivel y un entorno interactivo para cálculo numérico, visualización y programación. Permite al ingeniero analizar datos, desarrollar algoritmos y crear modelos y aplicaciones propias. Sus puntos fundamentales son:

- Entorno interactivo para la exploración, diseño y solución de problemas.
- Amplia librería de funciones para cálculo científico que permiten trabajar de forma natural con problemas de álgebra lineal, estadística, análisis de Fourier o cálculo numérico.
- Biblioteca de funciones gráficas para la visualización en 2D y 3D de datos. Incluye herramientas que permiten la generación de gráficos personalizados.
- Lenguaje de programación de alto nivel.
- Una gran variedad de librerías adicionales (toolboxes) especializadas en campos específicos de la ingeniería y la ciencia.

SIMULINK es una herramienta incluida en MATLAB para el modelado, simulación y análisis de sistemas dinámicos. El interfaz principal es una herramienta gráfica para la descripción del sistema mediante diagramas de bloques. Además, permite su extensión mediante la creación de librerías con nuevos tipos de bloques. Está altamente integrado con el resto de MATLAB. Es una herramienta muy usada en ingeniería de control y procesamiento de señal para la simulación de sistemas y el diseño basado en modelos.

## 2 Comandos básicos de MATLAB para las prácticas de Regulación

### 2.1 Comandos básicos. Variables, vectores y matrices.

Matlab es un programa de cálculo matemático muy flexible y potente, con posibilidades gráficas para la presentación de los datos, por lo que se utiliza en muchos campos de la ciencia y la investigación como herramienta de cálculo matemático. Los comandos y funciones que se describen a continuación funcionan generalmente en todas las versiones de MATLAB, aunque algunos sólo lo hacen en las versiones más modernas y otros han quedado obsoletos. En este último caso las versiones más modernas disponen de otros comandos o funciones que los sustituyen. Algunos de ellos están disponibles bajo botones o menús en las nuevas versiones.

**NOTA:** En este documento se recogen unos pocos ejemplos de entre las inmensas posibilidades que ofrece Matlab. Se deben consultar las ayudas de Matlab u otros documentos si se desea un conocimiento más amplio. Se deben utilizar y modificar los ejemplos aquí incluidos para adquirir las destrezas básicas que permitirán utilizar el programa como herramienta para la asignatura. Téngase en cuenta que algunos ejemplos están encadenados, es decir, es necesario ejecutar determinados comandos anteriores para que los siguientes funcionen correctamente.

Realizar operaciones algebraicas es muy sencillo,

```
>> a = 2
>> b = 3
>> suma = a + b
suma =
    5
```

o simplemente

```
>> 2 + 3
ans =
    5
```

La variable "ans" contiene el resultado de la última operación realizada y puede consultarse en cualquier momento así como el resto de las variables que se vayan creando:

```
>> suma          >> ans
suma =           ans =
    5             5
```

Si se quiere ver el nombre de todas las variables que se están utilizando se pueden usar los comandos **who** y **whos**. Con **save** y **load** se pueden guardar las variables que se están usando y recuperarlas posteriormente. El comando **help** proporciona ayuda y combinado con otro comando, ayuda sobre ese comando en concreto (p.ej. **help who**). Para abandonar el programa se pueden usar los comandos **quit** o **exit**. (En versiones bajo Windows se dispone de los correspondientes menús y botones para estas funciones).

MATLAB maneja matrices, y como casos particulares de las mismas, vectores fila y vectores columna:

Función	Ejemplo
Introducción de un vector fila	<b>vectorfila = [1 2 3]</b> vectorfila = 1   2   3
Introducción de un vector fila	<b>vectorfila = [1,2,3]</b> vectorfila = 1   2   3
Introducción de un vector columna como 'transposición de un vector fila	<b>vectorcolumna = [1,2,3]'</b> vectorcolumna = 1 2 3
Introducción de un vector columna	<b>vectorcolumna = [1;2;3]</b> vectorcolumna = 1 2 3
Introducción de una matriz	<b>matriz = [1 2 3 4 5 6 7 8 9]</b> matriz = 1   2   3 4   5   6 7   8   9
Introducción de una matriz	<b>matriz = [1,2,3;4,5,6;7,8,9]</b> matriz = 1   2   3 4   5   6 7   8   9

## 2.2 Valores complejos

Función	Ejemplo
<b>abs</b> se obtiene el módulo de un vector complejo  se utilizan "j" o "i" como valores imaginarios	<b>c = 2 + 3*i</b> c = 2.0000 + 3.0000i  <b>modulo = abs(c)</b> modulo = 3.6056
<b>angle</b> se obtiene el argumento en radianes	<b>argumento = angle(c)</b> argumento = 0.9828
pero se convierte fácilmente a grados	<b>grados = argumento * 180 / pi</b> grados = 56.3099

**NOTA:** obviamente "i", "j", "ans", "pi", "help", "sin", etc. son nombres que ya están definidos para variables, constantes, comandos o funciones de Matlab y no se deben usar para nombrar nuevas funciones o variables del usuario. ¡Matlab distingue entre mayúsculas y minúsculas!

## 2.3 Polinomios.

Los polinomios se introducen en forma de vectores fila que contienen los coeficientes del polinomio.

Función	Ejemplo
introducción de un polinomio p.e. $s^3 - 6s^2 - 27$	<b>p = [1 -6 0 -27]</b> p = 1 -6 0 -27
<b>roots (p)</b>  da las raíces del polinomio en un vector columna	<b>&gt;&gt; raices = roots(p)</b> raices = 6.6167 -0.3084 + 1.9964i -0.3084 - 1.9964i
<b>poly (raices)</b>  se vuelve a obtener el polinomio original	<b>polinomio = poly (raices)</b> polinomio = 1.0000 -6.0000 -0.0000 -27.0000

Se pueden multiplicar y dividir polinomios:  $c(s) = a(s) * b(s)$ ;  $c(s)/a(s) = q(s) + r(s)/a(s)$

Función	Ejemplo
<b>conv (a,b)</b>  multiplicación de dos polinomios p.e. $a = s^2 + 2s + 3$ $b = 4s^2 + 5s + 6$ $c = 4s^4 + 13s^3 + 28s^2 + 27s + 18$	<b>a = [1 2 3];</b> <b>b = [4 5 6];</b> <b>c = conv(a,b)</b> c = 4 13 28 27 18
<b>[q,r] = deconv (c,a)</b>  división de dos polinomios p.e. $c(s) / a(s)$  q(s) es el cociente de la división r(s) es el resto de la división	<b>&gt;&gt; [q,r] = deconv(c,a)</b>  q = 4 5 6 r = 0 0 0 0

<b>[r,p,k] = residue (b,a)</b> <b>[b, a] = residue (r, p,k)</b>  Considérese una función de transferencia G(s)  $\frac{b(s)}{a(s)} = \frac{b_1 s^m + b_2 s^{m-1} + \dots + b_m s^1 + b_{m+1}}{a_1 s^n + a_2 s^{n-1} + \dots + a_n s^1 + a_{n+1}}$  donde algunos $a_i$ y $b_i$ pueden ser ceros.  El comando <code>[r,p,k] = residue (b, a)</code> encuentra los residuos (r), los polos (p), y los términos directos (k) de un desarrollo en fracciones simples del cociente de dos polinomios y viceversa.	<b>b = [2 5 3 6]</b> <b>a = [1 6 11 6]</b> <b>[r,p,k] = residue(b, a)</b>  r = -6.0000 -4.0000 3.0000 p = -3.0000 -2.0000 -1.0000 k = 2
---	--

Si no hay raíces múltiples, entonces:

$$\frac{b(s)}{a(s)} = \frac{r(1)}{s - p(1)} + \frac{r(2)}{s - p(2)} + \dots + \frac{r(n)}{s - p(n)} + k(s)$$

El número de polos n es

$$n = \text{length}(a) - 1 = \text{length}(r) = \text{length}(p)$$

El vector del coeficiente del término directo está vacío si

**length(b) < length(a)**, en los demás casos se tendrá

$$\text{length}(k) = \text{length}(b) - \text{length}(a) + 1$$

Si **p(j) = ... = p(j+m-1)** es de orden de multiplicidad m, entonces la expansión incluye términos de la forma

$$\frac{r_j}{s - p_j} + \frac{r_{j+1}}{(s - p_j)^2} + \dots + \frac{r_{j+m-1}}{(s - p_j)^m}$$

Que se corresponde con

$$G(s) = \frac{-6}{s+3} + \frac{-4}{s+2} + \frac{3}{s+1} + 2$$

**[b, a] = residue(r, p, k)**

b =

2.0000 5.0000 3.0000 6.0000

a =

1.0000 6.0000 11.0000 6.0000

Como se puede observar, a veces el resultado de una función, pueden ser varias variables y sus argumentos también pueden ser varios y de diferentes tipos. Se debe consultar mediante el comando **help** (p.ej. **help deconv**) cuales son los argumentos que admite cada función y que variables va a devolver como resultado.

## 2.4 Representaciones gráficas 2D y 3D.

### Función

**t = [0:1:100]**

creación de un vector tiempo de 0 a 100 con valores equidistantes de 1 segundo

**y = sin (0.1\*t)**

creación de un vector con los valores de seno (0,1t)

### Ejemplo

**t = [0:1:100]**

t =

Columns 1 through 18

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17						

Columns 19 through 36 ...

**y = sin(0.1\*t)**

y =

Columns 1 through 11

0	0.0998	0.1987	0.2955
0.3894	0.4794	0.5646	0.6442
0.7174	0.7833	0.8415	

Columns 12 through 22...

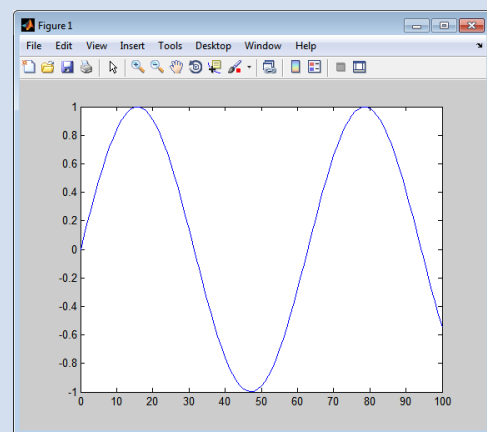
**plot (t,y)**

representación en 2D de la función seno

eje de abscisas el tiempo

eje de ordenadas  $\sin(wt) = \sin(0,1t)$

**>> plot(t,y)**



**plot3 (t,y,z)**

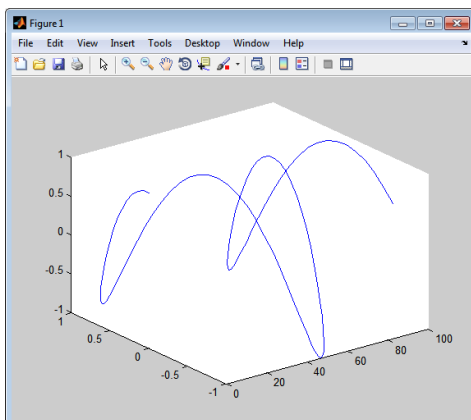
representación en 3D de la función seno

eje x el tiempo

eje y  $\sin(0,1\cdot t)$

eje z  $\cos(0,2\cdot t)$

**t = [0:1:100] ;  
y = sin(0.1\*t);  
z = cos(0.2\*t);  
plot3(t,y,z)**



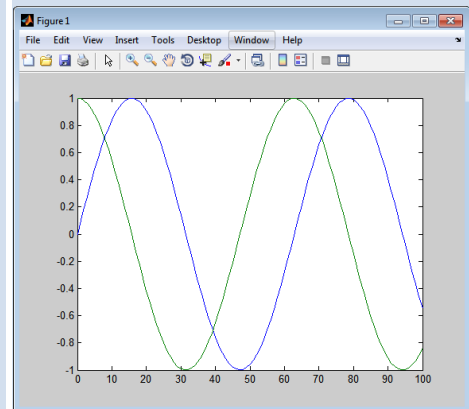


## Función

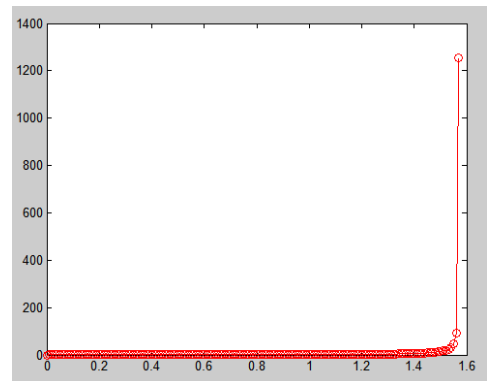
representación de dos funciones

## Ejemplo

```
t = [0:1:100]
y = [sin(0.1*t);cos(0.1*t)]
plot(t,y)
```



```
x = 0: .01: pi/2;
plot(x,tan(x),'-ro')
```



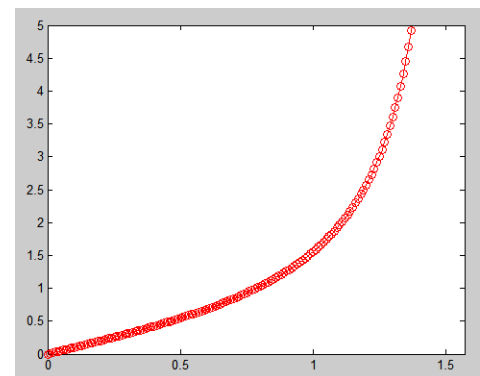
**axis ([xmin xmax ymin ymax])**

Permite cambiar la escala de la figura actual.

axis ([xmin xmax ymin ymax]) configura los límites de los ejes x e y actuales.

cambia la escala para ver mejor entre 0 y 5 del eje y

```
>> axis([0 pi/2 0 5])
```



**clf**

Borra el contenido de una figura

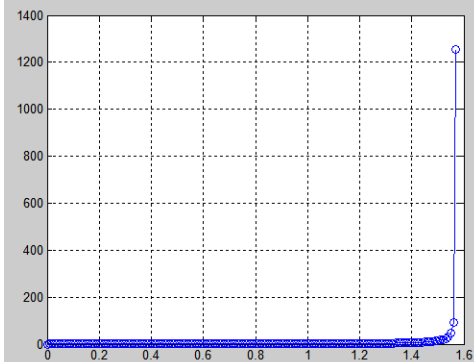
**figure**

Crea un nuevo objeto gráfico figura

## grid

Dibuja el grid en los ejes seleccionados en la figura actual

```
x = 0: .01: pi/2;  
plot(x,tan(x),'-bo')  
grid
```

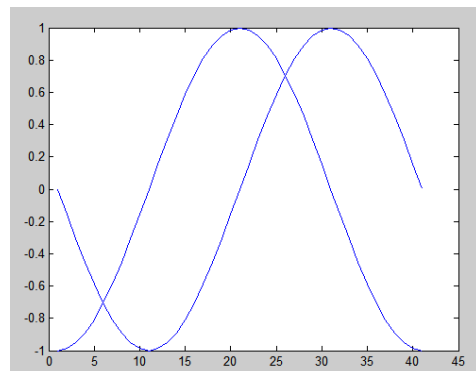


## hold on

## hold off

Activa/Desactiva la persistencia de los gráficos de una figura. Esto es, el segundo gráfico se representa en la misma figura.

```
x = -pi: pi/20: pi;  
plot(sin(x))  
hold on  
plot(cos(x))  
hold off
```



## title ('string')

Añade un título a la figura actual

## xlabel ('string')

añade texto en el eje x

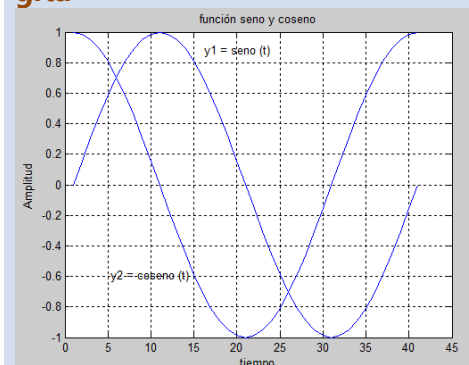
## ylabel ('string')

añade texto en el eje y

## gtext ('string')

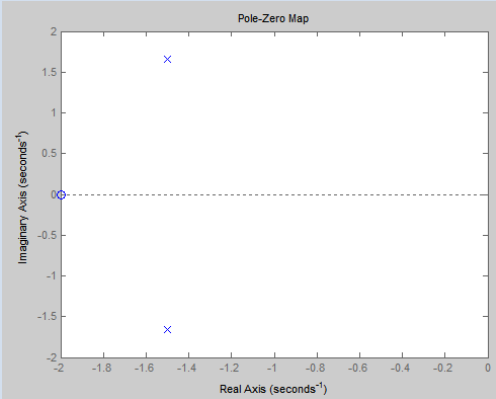
inserta texto con el ratón

```
x = 0: pi/20: 2*pi;  
plot(sin(x))  
hold on  
plot(cos(x))  
hold off  
title ('función seno y coseno')  
ylabel ('Amplitud')  
xlabel ('Tiempo')  
gtext ('y1 = seno (t)')  
gtext ('y2 = coseno (t)')  
grid
```



## 2.5 Funciones de Transferencia.

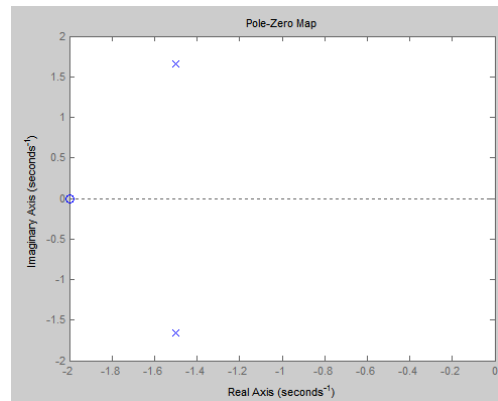
Para trabajar con funciones de transferencia, se introducen por separado dos vectores, uno con los coeficientes del polinomio del numerador y otro con los del denominador:

Función	Ejemplo
<b><code>s = tf('s')</code></b> crea el objeto "s" variable de Laplace	<b><code>s=tf('s')</code></b> $s = \frac{1}{s}$ Continuous-time transfer function.
<b><code>sys = tf(num,den)</code></b> crea una función de transferencia continua en el tiempo con numerador NUM y denominador DEN. SYS es un objeto de tipo <b>tf</b> cuando NUM,DEN son vectores numéricos.	<b><code>g = tf([1 2], [1 3 5])</code></b> $g = \frac{s + 2}{s^2 + 3s + 5}$
<b><code>printsys(num,den)</code></b> muestra la función de transferencia asociada a un cociente de polinomios <code>num = [1 2]</code> $\rightarrow s+2$ <code>den = [1 3 5]</code> $\rightarrow s^2+3s+5$	<b><code>num = [1 2];</code></b> <b><code>den = [1 3 5];</code></b> <b><code>printsys(num,den)</code></b> $\frac{\text{num/den}}{s + 2} = \frac{s^2 + 3s + 5}{s^2 + 3s + 5}$
<b><code>[p,z] = pzmap(num,den)</code></b> obtiene los vectores columna "p" y "z" de los polos y ceros respectivamente del sistema  Ver también: <b><code>[p,z] = pzmap(g)</code></b>	<b><code>[p,z] = pzmap(num,den)</code></b> $p = \begin{matrix} -1.5000 + 1.6583i \\ -1.5000 - 1.6583i \end{matrix}$ $z = -2$
<b><code>pzmap(num,den)</code></b> muestra el mapa de ceros y polos del sistema gráficamente	<b><code>pzmap(num, den)</code></b> 

## pzmap (p,z)

muestra el mapa de ceros y polos del sistema a partir de estos

## pzmap (p, z)



## sys = zpk (z,p,k)

crea o convierte un modelo LTI a partir de las posiciones de los ceros y los polos del sistema y de la ganancia escalar K

$$h(s) = k \frac{(s - z(1))(s - z(2)) \dots (s - z(m))}{(s - p(1))(s - p(2)) \dots (s - p(n))}$$

## g=zpk (1, [-1+3\*i -1-3\*i], 1.5)

$$g = \frac{1.5 (s-1)}{(s^2 + 2s + 10)}$$

## g2=zpk (0, [-1 -2], 3)

$$g2 = \frac{3 s}{(s+1)(s+2)}$$

## [z,p,k] = zpkdata (sys)

Devuelve los ceros, los polos y la ganancia k del sistema

## [z,p,k] = zpkdata(g)

$$\begin{aligned} z &= 1 \\ p &= -1.0000 + 3.0000i \\ &= -1.0000 - 3.0000i \\ k &= 1 \end{aligned}$$

## [n, d] = tfdata (sys,'v')

Devuelve el numerador y el denominador de la función de transferencia de un modelo LTI.  
El parámetro 'v' fuerza a dar los valores de n y d.

## g=zpk (1, [-1+3\*i , -1-3\*i], 1) [n,d]=tfdata(g,'v')

$$\begin{aligned} n &= \begin{bmatrix} 0 & 1 & -1 \end{bmatrix} \\ d &= \begin{bmatrix} 1 & 2 & 10 \end{bmatrix} \end{aligned}$$

**[n2, d2] = ord2 (wn, xi)**

función de transferencia de segundo orden con  $\omega_n$  y  $\xi$

**wn = 30**  
**xi = 0.3**  
**[n2, d2] = ord2 (wn, xi)**

n2 =  
1  
d2 =  
1 18 900

la función de transferencia de segundo orden

**printsys (n2, d2)**

num/den =  
1  
-----  
 $s^2 + 18s + 900$

**sys = tf (num,den)**

crea una función de transferencia continua en el tiempo con numerador NUM y denominador DEN. SYS es un objeto de tipo tf cuando NUM,DEN son vectores numéricos.

**g = tf ([1 2], [1 3 5])**

g =  
 $s + 2$   
-----  
 $s^2 + 3s + 5$

**isstable (g)**

Devuelve cierto si un sistema es estable ans = 1

**g = (s+2)/(s^2+3\*s+5)**

g =  
 $s + 2$   
-----  
 $s^2 + 3s + 5$

**isstable (g)**

ans =  
1

**sistemas con tiempo muerto**

se usa la propiedad 'Input delay' al crear el modelo

**g=tf(1,[1 10],'ioDelay',2.1)**

g =  
 $\exp(-2.1s) * \frac{1}{s + 10}$

Continuous-time transfer function.

### minreal (sys)

Mínima realización de un sistema por cancelación de polos y ceros

```
g = zpk ([], 1,1);  
h = tf([2 1],[1 0])  
cloop = inv (1+g*h)*g  
cloop =  
      s (s-1)  
-----  
(s-1) (s^2 + s + 1)
```

Continuous-time zero/pole/gain model.

### minreal (cloop)

```
ans =  
      s  
-----  
(s^2 + s + 1)
```

Continuous-time zero/pole/gain model.

Estos objetos Función de Transferencia se pueden sumar, multiplicar, etc. y aplicarles las funciones que aquí se describen, válidas también para un par de polinomios numerador/denominador:

### Función

### Ejemplo

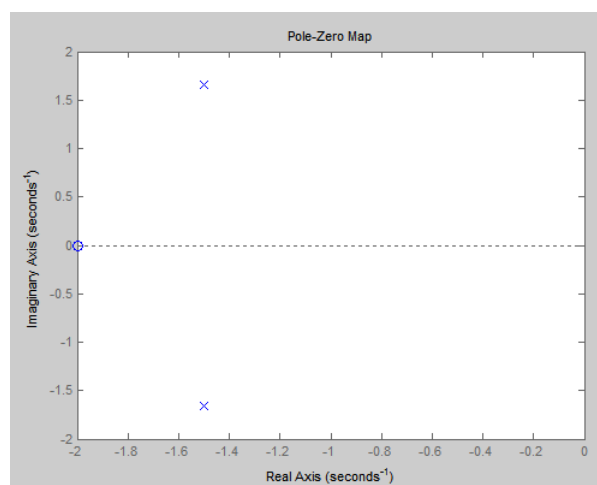
multiplicación de g por si misma

### g2 = g\*g

```
g2 =  
      s^2 + 4 s + 4  
-----  
s^4 + 6 s^3 + 19 s^2 + 30 s + 25
```

Equivale a:

```
num = [1 2];  
den = [1 3 5];  
pzmap (num,den)
```



```
g = tf ([1 2], [1 3 5]);  
pzmap( g)
```

para pasar de "num/den" a "g"

**g = tf (num,den)**

$$g = \frac{s + 2}{s^2 + 3s + 5}$$

y de "g" a "num/den"

**[p,z] = pzmap (g)**

$$p = \begin{matrix} -1.5000 + 1.6583i \\ -1.5000 - 1.6583i \end{matrix}$$

$$z = -2$$

**num = dcgain (g\*tf(poly(p), poly(z)))\*poly(z)**

igual a poly(z)

$$\text{num} = \begin{matrix} 1.0000 & 2.0000 \end{matrix}$$

**den = poly(p)**

$$\text{den} = \begin{matrix} 1.0000 & 3.0000 & 5.0000 \end{matrix}$$

**pole (g)**

nos muestra los polos de un sistema

**g = tf ([1 2], [1 3 5])**

**pole (g)**

$$\text{ans} = \begin{matrix} -1.5000 + 1.6583i \\ -1.5000 - 1.6583i \end{matrix}$$

**zero (g)**

nos muestra los ceros de un sistema

**g = tf ([1 2], [1 3 5])**

**zero (g)**

$$\text{ans} = -2$$

## 2.6 Respuesta en el tiempo

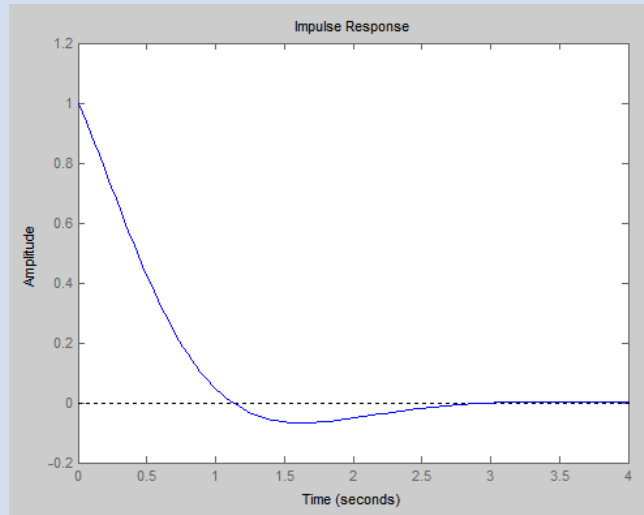
### Función

#### **impulse (g)**

representa gráficamente la respuesta en el tiempo del sistema ante una entrada impulso

### Ejemplo

```
g = tf ([1 2], [1 3 5]);  
impulse (g)
```



#### **[y,t] = impulse (g)**

Nos da el vector **y** de la amplitud de la señal de salida y el vector **t** de los tiempos.

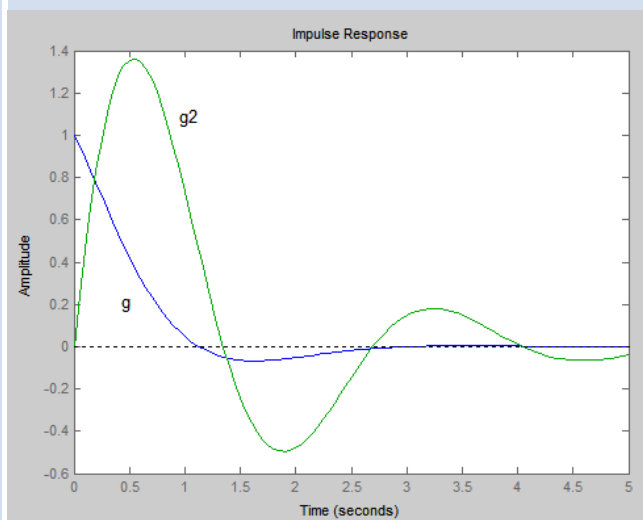
```
[y,t] = impulse (g)
```

```
y =  
1.0000  
0.9684  
0.9352 ...  
t =  
0  
0.0307  
0.0614 ...
```

#### **impulse (g, g2, 5)**

respuesta impulsional de g y g1 durante un tiempo de simulación de 5 segundos

```
g = tf ([1 2], [1 3 5]);  
g2 = tf ([5], [1 1.5 6]);  
impulse (g, g2, 5)  
gtext ('g')  
gtext ('g2')
```





## step (g)

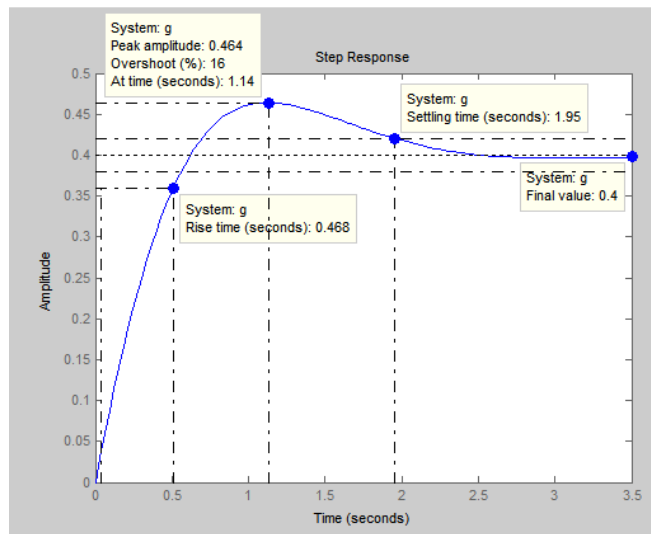
### step (g)

representa gráficamente la respuesta en el tiempo del sistema ante una entrada escalón unitario.

Un clic con el ratón sobre las gráficas proporciona información sobre sus valores.

Un clic con el botón derecho:

- permite mostrar las características y otros elementos interesantes.
- configurar los tiempos de establecimiento  $t_s$  y tiempo de subida  $t_r$  en propiedades /opciones



### [y,x,t] = step (g)

almacena en "**y**" la respuesta del sistema, en "**x**" la evolución de las variables de estado y en "**t**" el vector de tiempo

### [y,x,t] = step (g)

y =  
0  
0.0302  
0.0594 ....

### maximo = max (y)

da el valor máximo que alcanza la respuesta del sistema

### maximo = max(y)

maximo =  
0.4642

## 2.7 Análisis

### Función

#### ltiview (sys)

Visor de la respuesta de un sistema LTI.

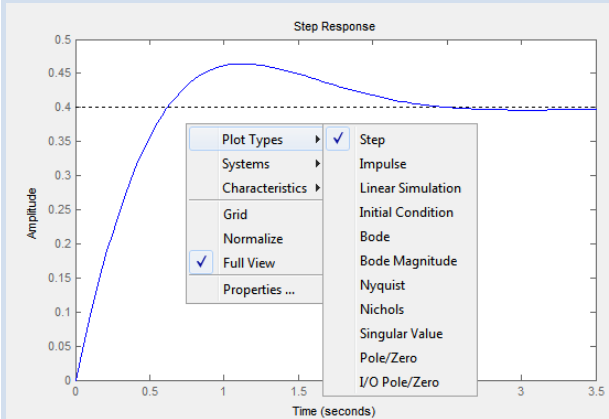
En la opción del menu Edit /Plot configurations.. podemos elegir si queremos ver más respuestas y de que forma.

P.e. la respuesta al escalón, al impulso, el mapa de polos y ceros y el Bode

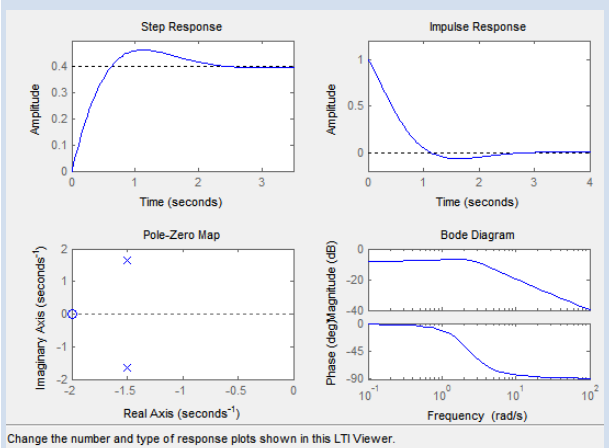
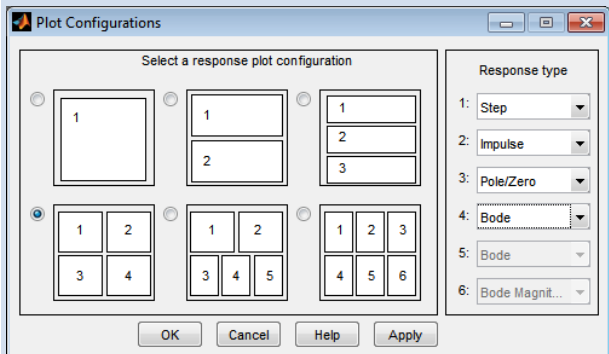
- \* Mirar los diferentes menus y sus opciones.
- \* Ver la ayuda para este comando.

### Ejemplo

$g = \text{tf}([1 \ 2], [1 \ 3 \ 5])$   
`ltiview(g)`



Change the number and type of response plots shown in this LTI Viewer.



Change the number and type of response plots shown in this LTI Viewer.

## Función

### **lsim (sys)**

Nos da la respuesta de un sistema ante una entrada arbitraria

**[u,t] = gensig (type,tau)**

**[u,t] = gensig (type,tau,Tf,Ts)**

genera señales de entrada para la función lsim

genera un escalar "**u**" del tipo "**sin**" onda senoidal, "**square**" onda cuadrada o "**pulse**" pulso periódico con periodo "**tau**".

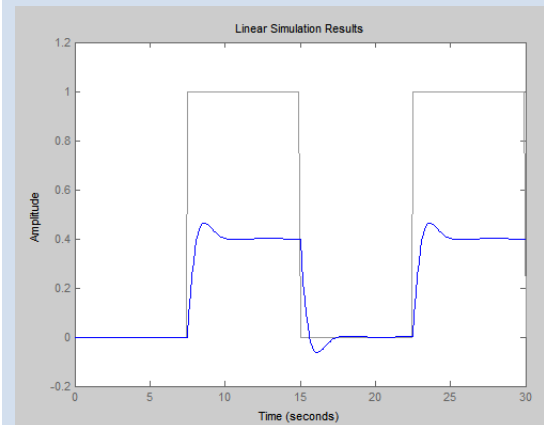
**Gensig** devuelve un vector "**t**" tiempos de muestreo del tiempo y un vector "**u**" de los valores de la señal para ese muestreo. Todas las señales generadas tienen la unidad de amplitud.

**[u,t] = gensig(type,tau,Tf,Ts)** también especifica la duración de la señal **Tf** y el tiempo de espaciado entre las muestras **Ts**.

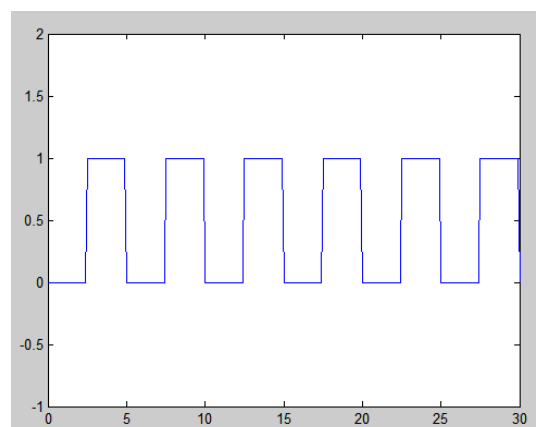
respuesta del sistema ante entrada parábola

## Ejemplo

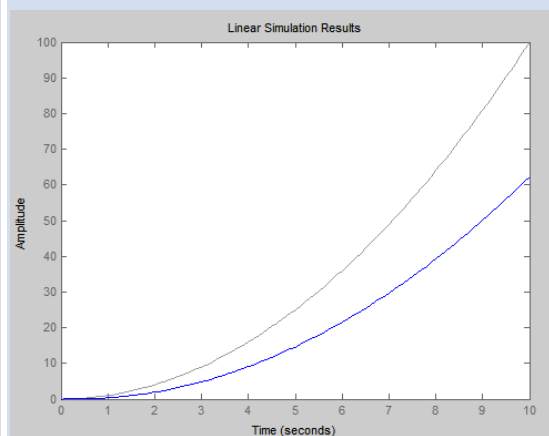
```
g = tf ([1 2], [1 3 5])  
[u, t] = gensig ('square', 15, 30, 0.1);  
lsim (g, u, t)
```



```
[u,t]= gensig('square',5,30,0.1);  
plot (t,u)  
axis ([0 30 -1 2])
```



```
g = tf ([2], [1 3]);  
t=0:0.1:10;  
u=t.^2;  
lsim (g,u,t)
```



---

**S = stepinfo (sys , 'RiseTimeLimits', RT)**

Devuelve las características de la respuesta escalón.

Lets you specify the lower and upper thresholds used in the rise time calculation. By default, the rise time is the time the response takes to rise from 10 to 90% of the steady-state value (RT=[0.1 0.9]). Note that RT(2) is also used to calculate SettlingMin and SettlingMax.

[Ver ayuda](#)

**g = tf ([1 5], [1 2 5 7 2]);**  
**S = stepinfo (g, 'RiseTimeLimits', [0.05, 0.95])**

S =

RiseTime: 7.4458  
SettlingTime: 13.9387  
SettlingMin: 2.3737  
SettlingMax: 2.5202  
Overshoot: 0.8091  
Undershoot: 0  
Peak: 2.5202  
PeakTime: 15.2118

---

## 2.8 Respuesta en frecuencia.

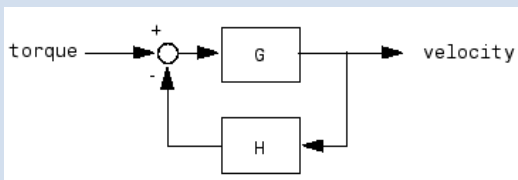
### Función

**clloop = feedback (g , 1, -1)**

proporciona la función de transferencia en bucle cerrado con realimentación unitaria "1" y negativa "-1"

**clloop = feedback (g,h)**

Si la realimentación no es unitaria



### Ejemplo

**g = tf ([2 5 1],[1 2 3]);**  
**clloop = feedback (g , 1, -1)**

clloop =  

$$\frac{s + 2}{s^2 + 4s + 7}$$

**h = zpk (-2,-10,5)**

h =  

$$\frac{5(s+2)}{(s+10)}$$

**clloop = feedback (g,h)**

clloop =  

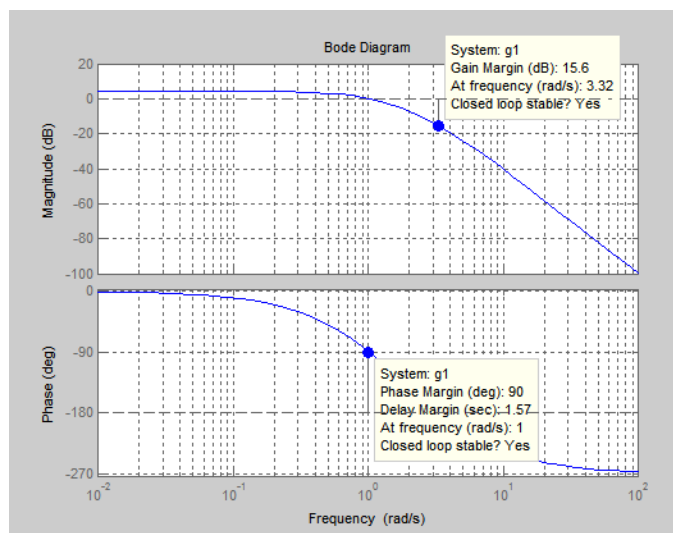
$$\frac{0.18182(s+10)(s+2.281)(s+0.2192)}{(s+3.419)(s^2 + 1.763s + 1.064)}$$

### bode (g)

calcula el diagrama de bode del sistema

con el botón derecho se puede seleccionar que se visualicen los valores característicos, la rejilla, etc..

**g = 10/((s+1)\*(s+2)\*(s+3));**



**[mag,fase] = bode (g,w)**

da los valores de magnitud (no está en decibelios) y de fase (en grados) del sistema para  $\omega = 3$

**g = 10/((s+1)\*(s+2)\*(s+3));**  
**[mag,fase] = bode (g,3)**

mag =  
 0.2067  
 fase =  
 -172.8750

**[mag,fase,w] = bode (g)**

devuelve los valores de magnitud, fase y pulsación, en tres vectores, para utilizarlos posteriormente con otras funciones como por ejemplo:

**maximo = max (mag)**

valor en el pico de resonancia de la relación de amplitudes

**resonancia = 20\*log10(maximo)**

valor en el pico de resonancia expresado en decibelios

**g = 10/((s+1)\*(s+2)\*(s+3))**

**[mag,fase,w] = bode(g)**

**maximo = max (mag)**

maximo =  
1.6666

**resonancia = 20\*log10(maximo)**

resonancia =  
4.4364

**[Gm,Pm,wg,wp] = margin (g)**

**Gm** margen de ganancia (no está en dB)

**Pm** margen de fase en grados

**wg** pulsación de cruce (rad/s) para el margen de ganancia

**wp** pulsación de cruce (rad/s) para el margen de fase

**Gm** = ∞ Indica que el margen de fase y de ganancia son infinito,

**Pm** = ∞ y que no se puede determinar un valor de  $\omega$  para ellos

**wg** = NaN (NaN = Not a Number)

**wp** = NaN

**g = 10/((s+1)\*(s+2)\*(s+3))**

**[Gm,Pm,wg,wp] = margin(g)**

Gm =  
6.0000

Pm =  
90.0000

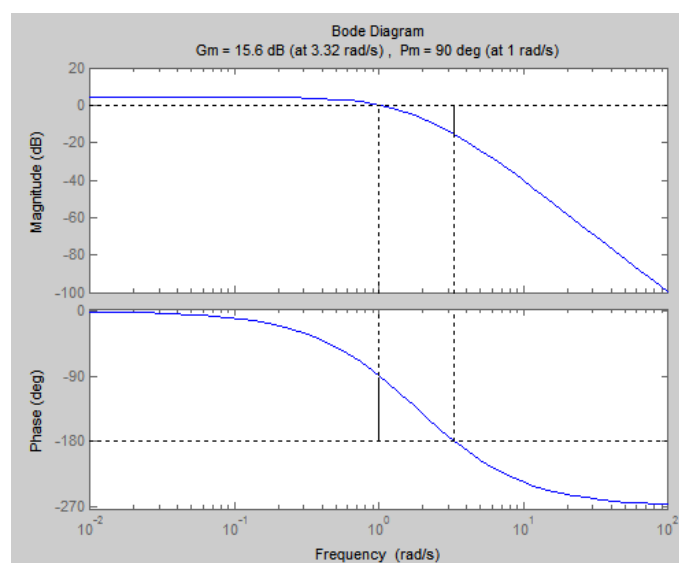
wg =  
3.3166

wp =  
1.0000

**margin(g)**

representa gráficamente el margen de ganancia y de fase

**margin(g)**



**damp (g)**

Nos muestra la frecuencia natural  $w_n$  y el coeficiente de amortiguamiento  $\xi$  de los polos de un sistema

Ver también:

**[frec, amtg] = damp (g)**

**[frec, amtg, polos] = damp (g)**

**g = 10/((s+1)\*(s+2)\*(s+3))**  
**damp (g)**

Eigenvalue	Damping	Frequency
-1.00e+00	1.00e+00	1.00e+00
-2.00e+00	1.00e+00	2.00e+00
-3.00e+00	1.00e+00	3.00e+00

(Frequencies expressed in rad/seconds)

**bandwidth (g)**

calcula el ancho de banda del sistema

**bandwidth (g)**

ans =  
0.7858

**dcgain (g)**

Calcula la ganancia estática del sistema

Ver también:

**Ke = dcgain (g)**

**dcgain (g)**

ans =  
1.6667

salida = **evalfr** (sys,wj)

nos devuelve el valor de  $g(s)$  para una frecuencia dada en valor complejo

**g = tf ([1 2], [1 3 5]);**  
**salida = evalfr(g,5j)**

salida =  
0.0560 - 0.2080i

**modulo = abs (x)**

nos devuelve el valor del módulo del vector x

**modulo = abs (salida)**

modulo =  
0.2154

**fase = angle (x)**

nos devuelve la fase del vector x en radianes

**fase = angle (salida)**

fase =  
-1.3078

### nichols (g)

dibuja el diagrama magnitud-fase

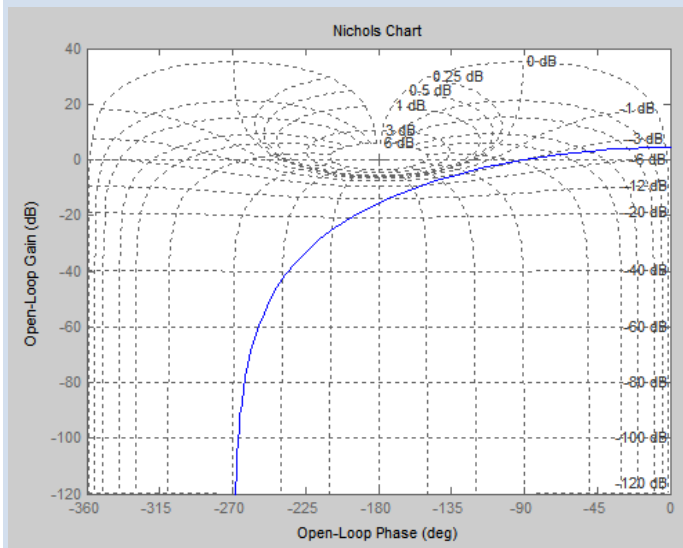
### ngrid

superpone el ábaco de Nichols al diagrama anterior

### shg

muestra la pantalla gráfica

### nichols (g) ngrid

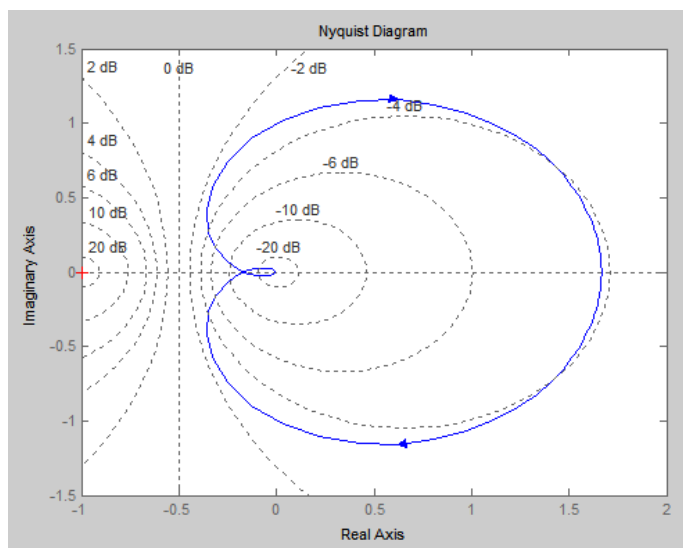


### nyquist (g)

Dibuja el diagrama de Nyquist del sistema.

El tramo con valores de  $\omega > 0$  es el "Diagrama Polar" del sistema

### nyquist (g)





## 2.9 Lugar de las raíces.

### Función

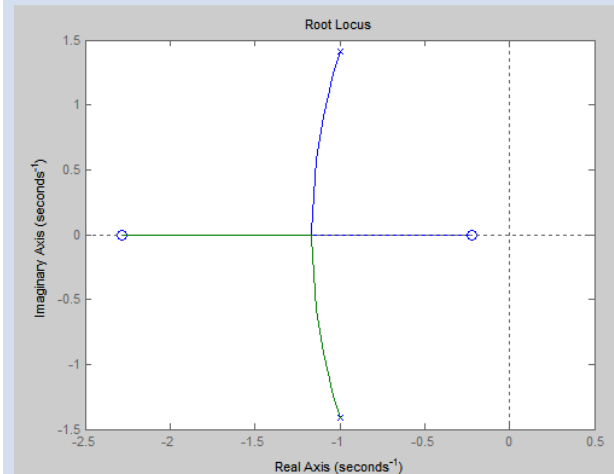
#### **rlocus (sys)**

calcula el lugar de las raíces de una función en bucle cerrado partiendo de la función (sistema SISO) en bucle abierto. Nos permite ver la trayectoria que siguen en función del parámetro  $k$ . Supone realimentación negativa. Nos permite además, ver el comportamiento de las respuesta temporal y de frecuencia.

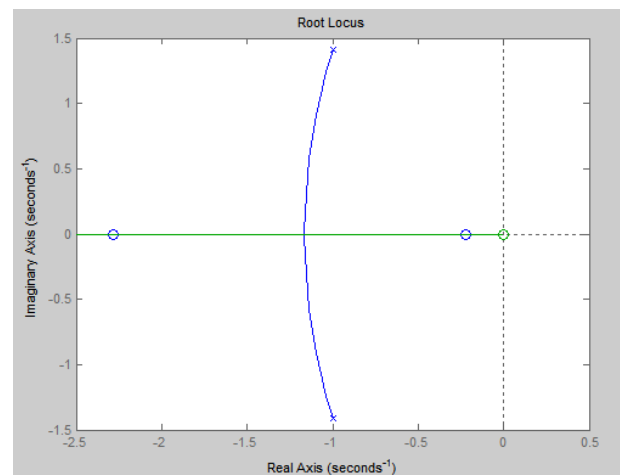
You can use the right-click menu for **rlocus** to add grid lines, zoom in or out, and invoke the Property Editor to customize the plot. Also, click anywhere on the curve to activate a data marker that displays the gain value, pole, damping, overshoot, and frequency at the selected point.

### Ejemplo

**$g = \text{tf}([2 \ 5 \ 1],[1 \ 2 \ 3])$**   
 **$\text{rlocus}(g)$**



#### **rlocus (g,3)**



#### **rlocus (sys,k)**

dibuja los polos del sistema en bucle cerrado para una ganancia de  $K=3$ . (círculos de color azul)

#### **$[r,k]=\text{rlocus}(\text{sys})$**

nos devuelve el vector  $k$  de las ganancias y el vector  $r$  de las localizaciones de las raíces complejas para esas ganancias.

**$g = \text{tf}([2 \ 5 \ 1],[1 \ 2 \ 3])$**   
 **$[r, k] = \text{rlocus}(g)$**

$r =$

Columns 1 through 4  
 $-1.0000 + 1.4142i$   $-1.0372 + 1.2459i$   $-1.0625 + 1.1165i$   $-1.0971 + 0.9087i$  ...

$k =$

Columns 1 through 7  
 0 0.0874 0.1665 0.3173 0.6047  
 0.9596 0.9605 .....

**polos = rlocus (sys,k)**

nos da los polos del sistema en bucle cerrado para una ganancia K.

**polos = rlocus (g,3)**

polos =  
-2.0000  
-0.4286

**sgrid**

dibuja en plano cero-polar o en lugar de las raíces, las líneas de coeficiente de amortiguamiento constante desde cero a uno, en intervalos de 0,1.

Las líneas de la frecuencia natural se dibujan a intervalos de 10 rad/sg con divisiones de un rad/sg.

**sgrid**

