TUTORIAL DE ANÁLISIS Y CONTROL DE SISTEMAS USANDO MATLAB

Manuel Vargas Villanueva

Este tutorial está basado en un trabajo original de:
Manuel Berenguel Soria y **Teodoro Álamo Cantarero**

Contenido

Capítulo 1

ANÁLISIS Y CONTROL DE SISTEMAS USANDO MATLAB

1.1 Introducción

En lo que sigue, se va a realizar una introducción a los comandos de MATLAB relacionados con la teoría de control de sistemas. Casi todas las funciones que se describen pertenecen al *Control System Toolbox*. Las funciones principales se van a explicar sobre ejemplos demostrativos, con el fin de que su uso y comprensión sean lo más sencillos posible. Se realizarán ejemplos tanto en el dominio temporal, continuo y discreto, como en el frecuencial. También se mostrará la forma de dar la descripción de un sistema lineal mediante función de transferencia, conjunto de polos y ceros, o variables de estado. Asimismo se comentará la forma de manipular una función de transferencia como un objeto.

1.2 Tratamiento mediante funciones de transferencia. Sistemas continuos

Este apartado muestra el uso de algunas de las herramientas con las que cuenta MATLAB para el diseño y análisis de sistemas de control. Para el ejemplo se va a partir de una descripción de la planta en forma de función de transferencia:

$$H(s) = \frac{.2s^2 + .3s + 1}{(s^2 + .4s + 1)(s + .5)}$$

En MATLAB las funciones de transferencia se introducen dando el par de polinomios numerador-denominador:

```
num = [.2  .3  1];
den1 = [1 .4 1];
den2 = [1 .5];
```

El polinomio del denominador es el producto de dos términos. Para obtener el polinomio resultante se usa el producto de convolución (o de polinomios).

```
den = conv(den1,den2)
```

Para ver los polos (o los ceros) de la función de transferencia, podemos usar: `roots(den)` (`roots(num)`). Una forma más completa de convertir una función de transferencia dada por dos polinomios numerador y denominador, en un conjunto de factores de grado 1, correspondientes a los polos (z_1, z_2, z_3) y ceros (c_1, c_2), de la forma:

$$H(s) = \frac{K\,(1-\frac{s}{c_1})\,(1-\frac{s}{c_2})}{(1-\frac{s}{z_1})\,(1-\frac{s}{z_2})\,(1-\frac{s}{z_3})}$$

es mediante el comando `tf2zp`:

```
[ceros,polos,gan] = tf2zp (N,D);
```

que devuelve un vector conteniendo los ceros de la función de transferencia, un vector conteniendo los polos, y un escalar correspondiente a la ganancia estática. La función complementaria a ésta también existe:

```
[N,D] = zp2tf (ceros,polos,gan);
```

Como curiosidad, cabe mencionar que el nombre de estas funciones es bastante descriptivo: "tf-two-zp" procede de *transfer-function to zero-pole form.*

1.2.1 Dominio Temporal

La respuesta ante un escalón a la entrada se puede analizar en sistemas que tengan una descripción en forma de función de transferencia o una representación en el espacio de estados, generando un vector de tiempos y usando la función `step`:

```
t = [0:.3:15]';
y = step(num,den,t);
plot (t,y);
```

```
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

La respuesta al escalón unitario puede verse en la Fig. 1.1. Seleccionando con el ratón en la curva, pueden modificarse algunos atributos de la misma. También pueden modificarse las propiedades de los ejes de forma análoga. Las figuras guardarse en ficheros *.fig* propios de MATLAB, o exportarse en diferentes formatos gráficos estándar.

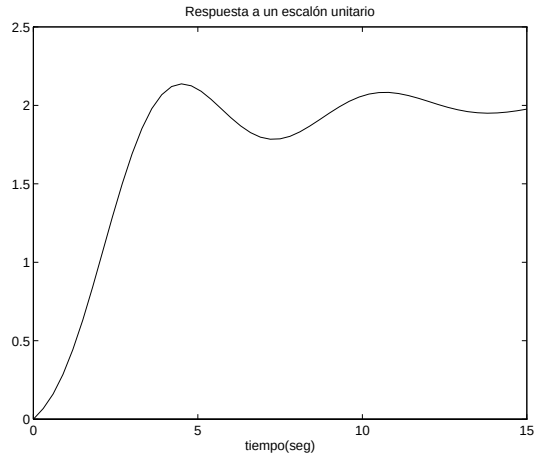

Figura 1.1: Respuesta a escalón unitario

No es necesario recuperar el resultado de la simulación ante escalón que realiza `step` para después representarlo con `plot`, sino que el propio comando `step`, si lo utilizamos sin parámetros de salida, realiza la representación. Es más, en este segundo caso, la representación gráfica es algo más interactiva, puesto que si "pinchamos" con el botón izquierdo en algún punto de la gráfica, nos dice los valores correspondientes al tiempo y al valor de la salida en ese punto.

La respuesta impulsional se puede obtener del mismo modo, pero usando en este caso la función `impulse`, que tiene una sintaxis similar al comando `step`. Nótese que el vector de tiempos ya ha sido definido con anterioridad.

```
impulse (num,den,t);
```

La respuesta al impulso puede verse en la Fig. 1.2.

La respuesta del sistema a cualquier tipo de entrada también puede obtenerse. Para ello es necesario tener la señal de entrada en un vector `u`, que lógicamente deberá tener la misma dimensión que el vector de tiempos. En sistemas multivariables, en vez de un vector de entradas tendremos una matriz. A estos efectos se usa la función `lsim`. Un ejemplo característico es la respuesta a una entrada en rampa:

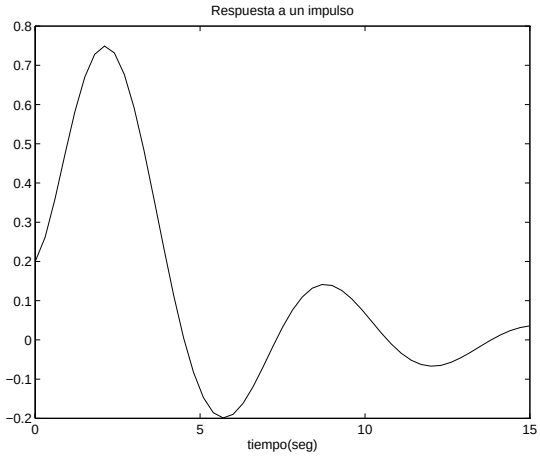

Figura 1.2: Respuesta al impulso

```
ramp = t;
y = lsim (num,den,ramp,t);
plot (t,y,t,ramp);
title ('Respuesta a una rampa');
xlabel ('tiempo(seg)');
```

La respuesta a la rampa puede verse en la Fig. 1.3.

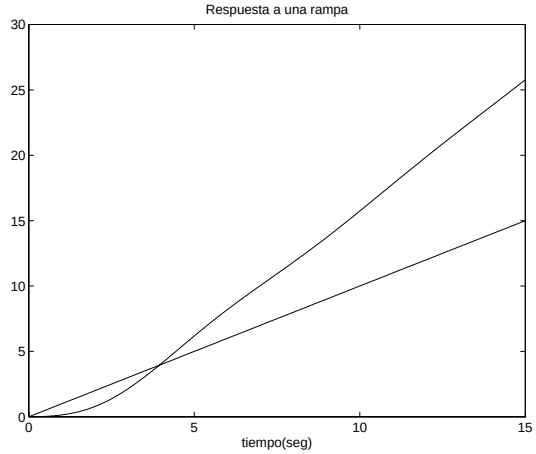

Figura 1.3: Respuesta a la rampa

Otro ejemplo característico es la respuesta a un ruido uniforme aleatorio. Recordamos que `rand(m,n)` genera una matriz $m \times n$ de números aleatorios uniformemente distribuidos entre 0 y 1. Es importante darse cuenta del filtrado que se produce en la señal cuando pasa por el sistema (hace de filtro paso bajas).

```
noise = rand (size(t));
```

```
y = lsim (num,den,noise,t);
plot (t,y,t,noise);
title ('Respuesta a un ruido aleatorio');
xlabel ('tiempo(seg)');
```

La respuesta al ruido puede verse en la Fig. 1.4.

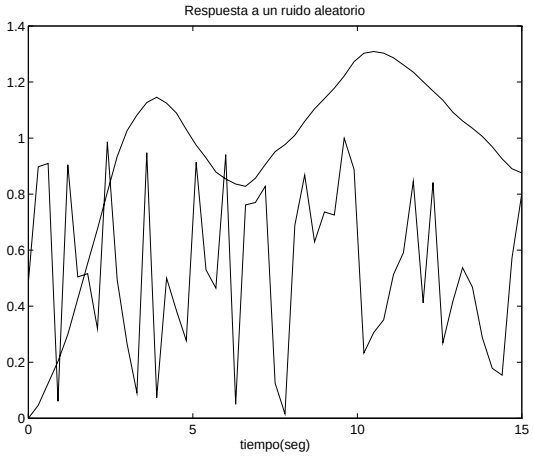

Figura 1.4: Respuesta a ruido aleatorio

1.2.2 Dominio Frecuencial

Respuesta en frecuencia

La respuesta en frecuencia de los sistemas se puede obtener usando las funciones `bode`, `nyquist` y `nichols`. Si no se le ponen argumentos a la izquierda, estas funciones generan las gráficas por sí solas. En caso contrario, vuelcan los datos en los vectores de salida oportunos. A continuación se presentan ejemplos de las tres posibles sintaxis de la función `bode`:

1.- `bode(num,den)`

2.- `[mag,phase,w] = bode (num,den)`

3.- `[mag,phase] = bode (num,den,w)`

La primera de ellas produce un gráfico con la magnitud en decibelios (dB) y la fase en grados. En las otras dos la magnitud se devuelve en el vector *mag* y está expresada en unidades absolutas, no en dB. Por su parte, la fase, devuelta en el vector *phase*, sigue siendo en grados. En esta forma, la representación es más interactiva, en el sentido de que "pinchando" con

el ratón en un punto de la curva, podemos ver los valores correspondientes. La segunda forma automáticamente genera los puntos de frecuencia en el vector w. En la tercera forma es el usuario el que escoge los rangos de frecuencia, y resulta muy adecuado cuando se quieren representar varias gráficas conjuntamente, que habrán de compartir una misma escala frecuencial.

El resultado de los dos últimos comandos se puede representar usando funciones conocidas:

```
subplot(211), loglog(w,mag), title('Magnitud'), xlabel('rad/s');
subplot(212), semilogx(w,phase), title('Fase'), xlabel('rad/s');
```

El resultado para la función de transferencia del ejemplo anterior puede verse en la Fig. 1.5. Cabe comentar que el comando `subplot(n,m,i)` permite dividir una ventana gráfica en una matriz de $n \times m$ sub-gráficas, seleccionando como activa la número i (numeradas consecutivamente de izquierda a derecha y de arriba abajo).

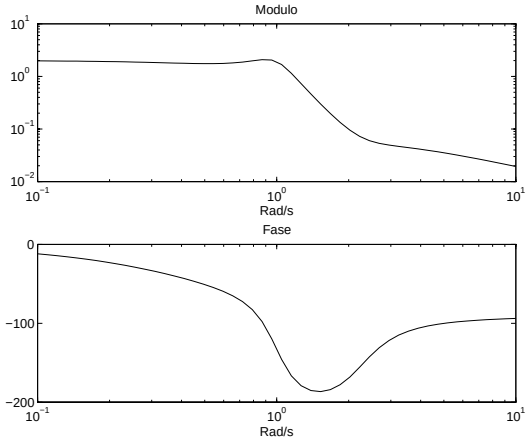

Figura 1.5: Diagrama de Bode

El comando `nyquist` tiene la misma sintaxis:

```
nyquist (num,den,w);
[re,im] = nyquist (num,den,w);
```

Computa las partes real e imaginaria de $G(jw)$ y realiza la representación si no se le ponen parámetros de salida. Para obtener la representación gráfica por nosotros mismos, sólo hay que dibujar la parte real frente a la imaginaria. El resultado obtenido mediante el ejemplo anterior puede verse en la Fig. 1.6.

El comando `nichols` computa el diagrama de Nichols de un sistema a partir de la función de transferencia en bucle abierto. Para verlo basta dibujar la magnitud del bucle abierto en

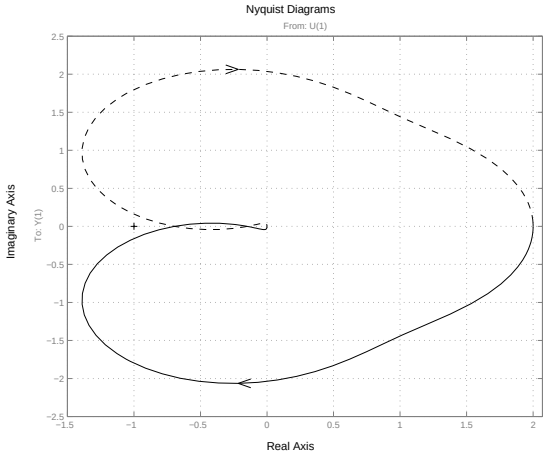

Figura 1.6: Diagrama de Nyquist

dB (en el eje de ordenadas) frente a fase del bucle abierto en grados (en eje de abcisas), o llamar a la función sin argumento de salida. Si se quiere en forma de ábaco, se puede usar el comando `ngrid`.

```
nichols (num,den,w),  ngrid;
[mag,phase] = nichols (num,den,w);
```

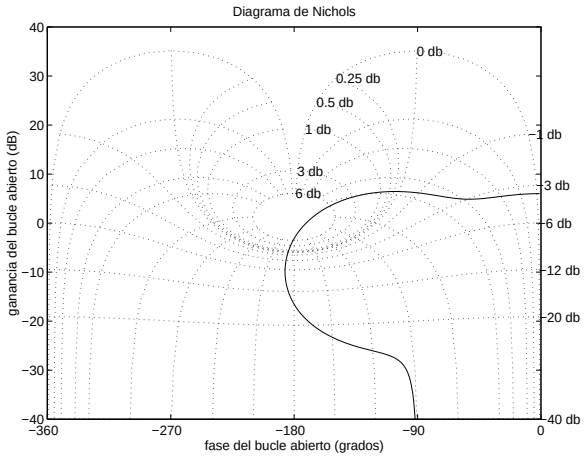

Figura 1.7: Diagrama de Nichols

Márgenes de estabilidad

Como es bien sabido en la teoría clásica del control, los márgenes de estabilidad son el margen de fase y el margen de ganancia. Estos márgenes se calculan usando el comando `margin`.

```
margin (num,den);
```

```
[mg,mf,wmg,wmf] = margin (num,den);
```

Como tercer parámetro de entrada se le puede pasar el rango de frecuencias deseado. En el primer caso indicado, en el que no se le especifican parámetros de salida, se realiza la representación del diagrama de Bode, junto con una indicación, mediante líneas verticales de los puntos donde se mide cada uno de los márgenes y los valores de los mismos. En la segunda variante, como salidas se obtienen los valores de los márgenes de ganancia (no en *dB*), el margen de fase (en grados) y sus correspondientes frecuencias. Si existen varias frecuencias de corte marca los más desfavorables (ver Fig. 1.8).

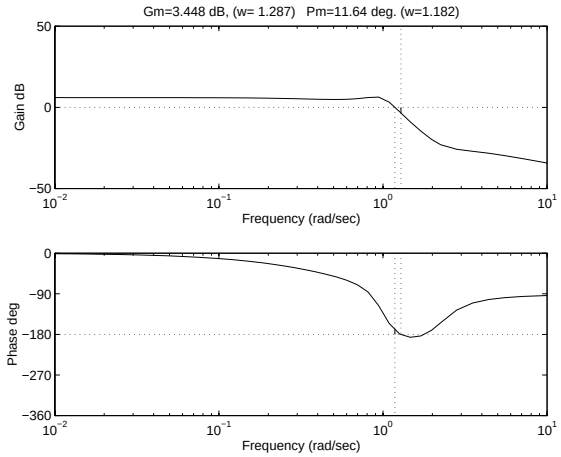

Figura 1.8: Márgenes de fase y ganancia

Efectos de los retardos

Los retardos existen en numerosas aplicaciones de control automático. En sistemas lineales continuos invariantes en el tiempo, el retardo viene representado por e^{-sT}. La forma más sencilla de manipular los retardos en MATLAB es en el dominio de la frecuencia. Nótese que $e^{-jwT} = 1^{\underline{-wT}}$. Por tanto los retardos dejan la magnitud invariable y afectan al desfase, tendiendo a inestabilizar al sistema controlado. Para propósitos de representación mediante el comando `bode`, todo lo que habrá que hacer es restar la fase del retardo a la de la función de transferencia. Por ejemplo:

```
num = [0.2 0.3 1];
den = [1 0.9 1.2 0.5];
T = 1;  % Tiempo de retardo puro
w = logspace (-2,1,100)';
[mag,fase] = bode (num,den,w);
faseDelay = fase - (T*w*180/pi);  % Sustrae la fase tras convertirla a grados
subplot(211);  semilogx (w, 20*log10(mag));  grid;
subplot(212);  semilogx (w, [fase, faseDelay]);  grid;
```

1.2.3 Comandos relacionados con operaciones de bloques

Existen una serie de comandos relacionados con las operaciones típicas en diagramas de bloques:

– `[N12,D12] = series (N1,D1,N2,D2)`: Devuelve la resultante de colocar en serie dos funciones de transferencia (Fig. 1.9). El mismo resultado podría obtenerse llamando dos veces al comando `conv`, que recuérdese permitía multiplicar dos polinomios.

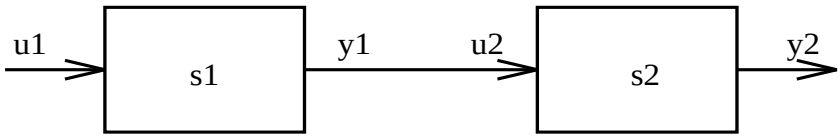

Figura 1.9: Conexión en serie

– `[N12,D12] = parallel (N1,D1,N2,D2)`: Devuelve la resultante de colocar en paralelo dos funciones de transferencia (Fig. 1.10).

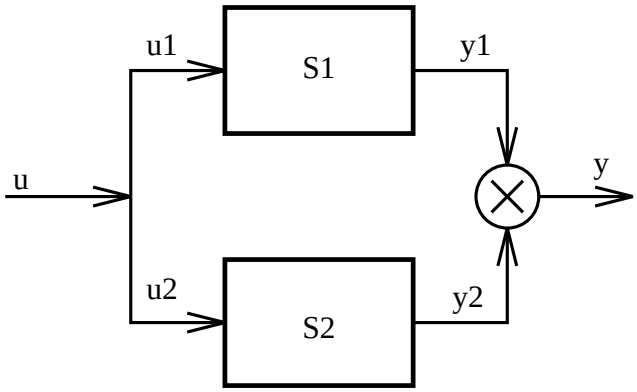

Figura 1.10: Conexión en paralelo

– `[Nbc,Dbc] = feedback (N1,D1,N2,D2,-1)`: A partir de un sistema en bucle abierto, dado por el numerador y denominador $N1$,$D1$, proporciona el correspondiente en bucle cerrado, considerando que en la cadena de realimentación hay otra función de transferencia, dada por $N2$, $D2$ (Fig. 1.11). El último parámetro indica el signo de la realimentación (-1 para realimentación negativa y 1 para positiva).

– `[Nbc,Dbc] = cloop (N1,D1,-1)`: En el caso en que se pretenda obtener la función de transferencia en bucle cerrado con realimentación unitaria, puede emplearse este comando más compacto, en el que se evita tener que especificar una segunda función de transferencia.

Conviene tener claro que para todos estos comandos relacionados con operaciones por bloques, se podría perfectamente estar trabajando con funciones de transferencia discretas, sin ninguna diferencia.

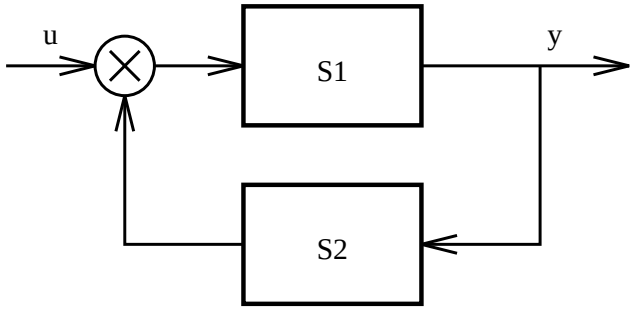

Figura 1.11: Conexión en realimentación

Para operaciones de bloques más complejas, resulta más adecuado usar la herramienta SIMULINK que también se explicará en estas notas.

1.2.4 Lugar de las raíces

El análisis mediante el lugar de las raíces se puede obtener definiendo un vector de ganancias deseadas (que es el parámetro usado habitualmente para ver la evolución de los polos en bucle cerrado). La sintaxis es:

```
r = rlocus (N,D,K);
rlocus (N,D);
```

En la primera forma, calcula el lugar de las raíces de $1+K\frac{N(s)}{D(s)}=0$, para un vector de ganancias especificado, K. `rlocus` devuelve una matriz r con `length(K)` filas y `length(den)` columnas, conteniendo la localización de las raíces complejas. Cada fila de la matriz corresponde a una ganancia del vector K. El lugar de las raíces puede ser dibujado con `plot(r,'x')`.

En la segunda forma, que es la usada habitualmente, la función directamente dibuja el lugar de las raíces. Además, como vemos, no es imprescindible indicar un vector de ganancias. Para la función de transferencia que veníamos utilizando en los ejemplos, se obtendría el lugar de las raíces mostrado en la Fig. 1.12

Un comando muy útil como complemento a `rlocus` es `rlocfind`, cuya sintaxis general es:

```
[K,polos] = rlocfind (num,den)
```

Antes de introducir dicho comando es necesario haber dibujado el lugar de las raíces. Al introducir `rlocfind`, se pide que se seleccione con el ratón un punto determinado del lugar,

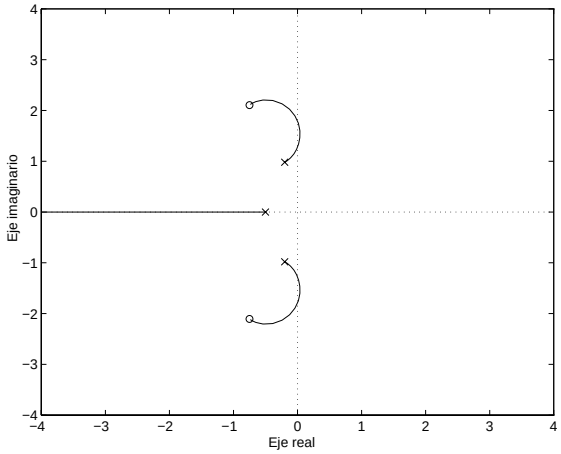

Figura 1.12: Lugar de las raíces

proporcionando como resultado la ganancia K en dicho punto y la localización de los polos correspondientes a esa ganancia.

Para mejorar la precisión, puede realizarse un *zoom* en torno a una zona de interés en la gráfica, antes de ejecutar `rlocfin`. Otra utilidad para ver la sobreoscilación que correspondería a un par de polos complejos conjugados situados en el lugar, sería dibujar los lugares geométricos de factor de amortiguamiento (δ) y frecuencia natural (ω_n) constantes, mediante el comando `sgrid`.

Ejemplo: Los siguientes comandos dibujan el lugar de las raíces de una función de transferencia, realizando una conveniente ampliación y resaltando las líneas de factor de amortiguamiento 0.5, 0.6 , 0.7 y de frecuencia natural $0.5\,rad/s$ (Fig. 1.13).

```
N = 1;
D = [1 3 2 0];
rlocus(N,D);
sgrid ([0.5:0.1:0.7],0.5);
axis([-2.5,1,-3,3]);
```

Las nuevas versiones de MATLAB van más allá y ofrecen una herramienta interactiva para ir viendo en vivo las variaciones que sufre el lugar de las raíces, conforme se añaden, eliminan y mueven los polos y ceros de bucle abierto. A esta herramienta se accede mediante el comando `rltool`. Merece la pena que el alumno dedique unos minutos a esta herramienta de gran valor didáctico.

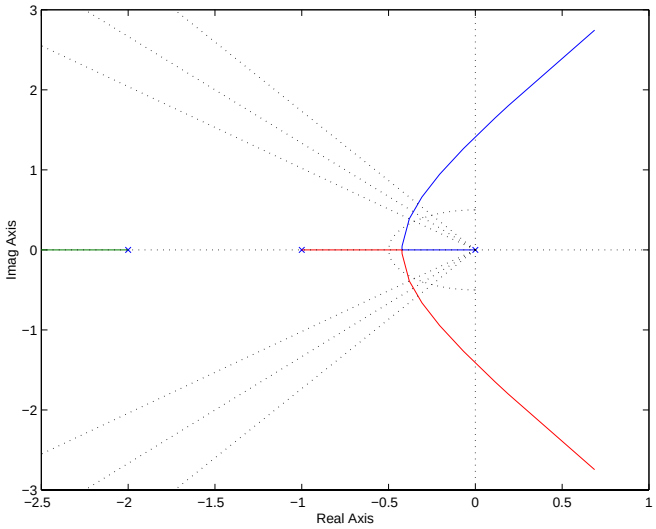

Figura 1.13: Lugar de las raíces con rejilla

1.3 Estudio temporal y frecuencial de sistemas de primer y segundo orden

1.3.1 Sistemas de primer orden

La representación en forma de función de transferencia viene dada por:

$$G(s) = \frac{K}{1 + \tau s}$$

que en notación MATLAB se introduce:

```
K = 1;
tau = 1;
num = K;
den = [tau 1];
```

La respuesta a un escalón unitario de entrada se obtiene con la función `step`. El resultado puede verse en la Fig. 1.14

```
t = [0:0.1:10]';
ye = step(num,den,t);
plot(t,ye);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

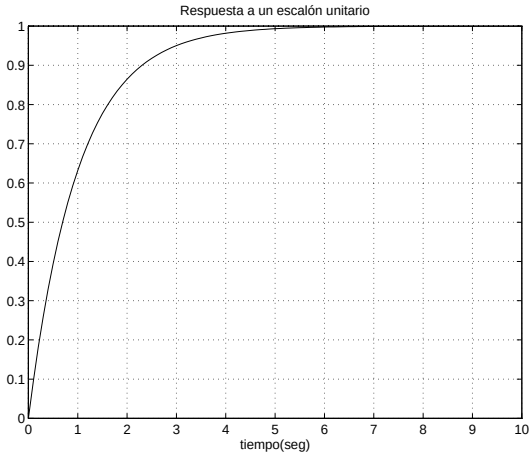

Figura 1.14: Respuesta a escalón unitario del sistema de primer orden

Las dos características fundamentales de un sistema de primer orden son su ganancia estática K y su constante de tiempo τ. La constante de tiempo es el tiempo que tarda en alcanzar el 63% de la salida. La ganancia estática es el cociente entre la amplitud de salida y la de entrada en el régimen permanente. Estos valores se pueden comprobar directamente en la gráfica o analizando el vector de datos resultante:

```
yRP = ye(length(ye));  % Valor en regimen permanente
n = 1;
while ye(n) < 0.63*yRP
    n=n+1;
end
% Constante de tiempo (0.1 es el intervalo transcurrido entre dos medidas,
% se le resta 1, porque los indices empiezan en 1):
tauEstim = 0.1*(n-1);
```

La respuesta a una rampa unitaria de entrada para nuestro sistema de primer orden se puede simular mediante:

```
ramp = t;
yr = lsim (num,den,ramp,t);
plot (t,yr,t,ramp);
title ('Respuesta a una rampa');
xlabel ('tiempo(seg)');
grid;
```

El resultado se muestra en la Fig. 1.15, en la que aparecen, tanto la rampa de entrada como la salida. El error de seguimiento en posición en régimen permanente puede obtenerse, al igual que se ha hecho anteriormente, midiendo directamente en la gráfica o bien a partir del vector de datos resultante.

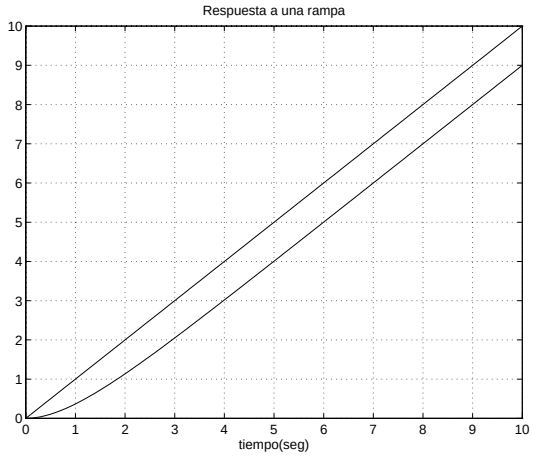

Figura 1.15: Respuesta a rampa unitaria del sistema de primer orden

La respuesta impulsional se puede obtener del mismo modo, pero usando en este caso la función `impulse` (Fig. 1.16), que tiene una sintaxis similar al comando `step`.

```
yi = impulse (num,den,t);
plot (t,yi);
title ('Respuesta a un impulso');
xlabel ('tiempo(seg)');
```

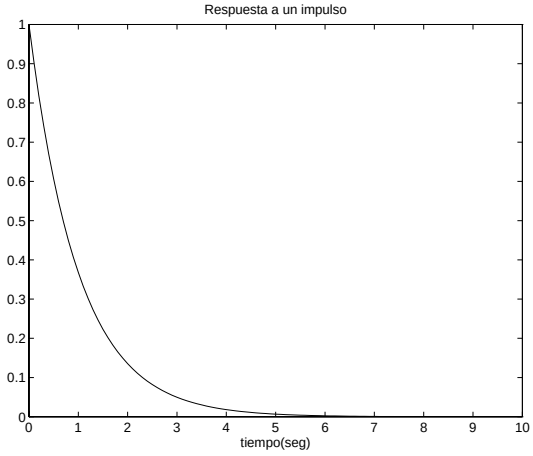

Figura 1.16: Respuesta a impulso del sistema de primer orden

Para finalizar con la sección de sistemas de primer orden, se va a comprobar que los resultados coinciden con los esperados en la teoría. Como es bien sabido, los sistemas lineales de primer orden de ganancia unidad invariantes en el tiempo tienen las siguientes características (nos remitimos a la bibliografía):

Respuesta a escalón unitario: $y_{e2}(t) = 1 - e^{(-t/\tau)},\ (t \geq 0)$

Respuesta a rampa unitaria: $y_{r2}(t) = t - \tau + \tau e^{(-t/\tau)},\ (t \geq 0)$

Respuesta a impulso: $y_{i2}(t) = (1/\tau)e^{(-t/\tau)},\ (t \geq 0)$

Si se dibujan estas funciones con el vector de tiempos definido anteriormente, y se superponen con las gráficas vistas anteriormente, puede apreciarse que coinciden perfectamente (Fig. 1.17).

```
ye2 = 1 - exp(-t/tau);
yr2 = t - tau + tau * exp(-t/tau);
yi2 = (1/tau) * exp(-t/tau);
plot (t,ye,t,ye2,'o');
title('Respuesta teorica a escalon unitario');
grid;
pause;
plot (t,yr,t,ramp,t,yr2,'o');
title('Respuesta teorica a rampa unitaria');
grid;
pause;
plot (t,yi,t,yi2,'o');
title ('Respuesta teorica a impulso');
grid;
```

1.3.2 Sistemas de segundo orden

La representación normal de un sistema de segundo orden en forma de función de transferencia viene dada por:

$$G(s) = \frac{Kw_n^2}{s^2 + 2\delta w_n s + w_n^2}$$

donde:

K: ganancia estática del sistema. Se va a suponer en el análisis siguiente, sin pérdida de generalidad, que $K = 1$.

δ: Coeficiente de amortiguamiento.

w_n: Frecuencia natural no amortiguada del sistema.

Del polinomio característico se tiene que las dos raíces son $s_{1,2} = -\delta w_n \pm w_n\sqrt{\delta^2 - 1}$, pudiendo distinguirse los siguientes casos:

Caso 1: Si $\delta > 1 \rightarrow$ 2 raíces reales distintas en SPI (sobreamortiguado).

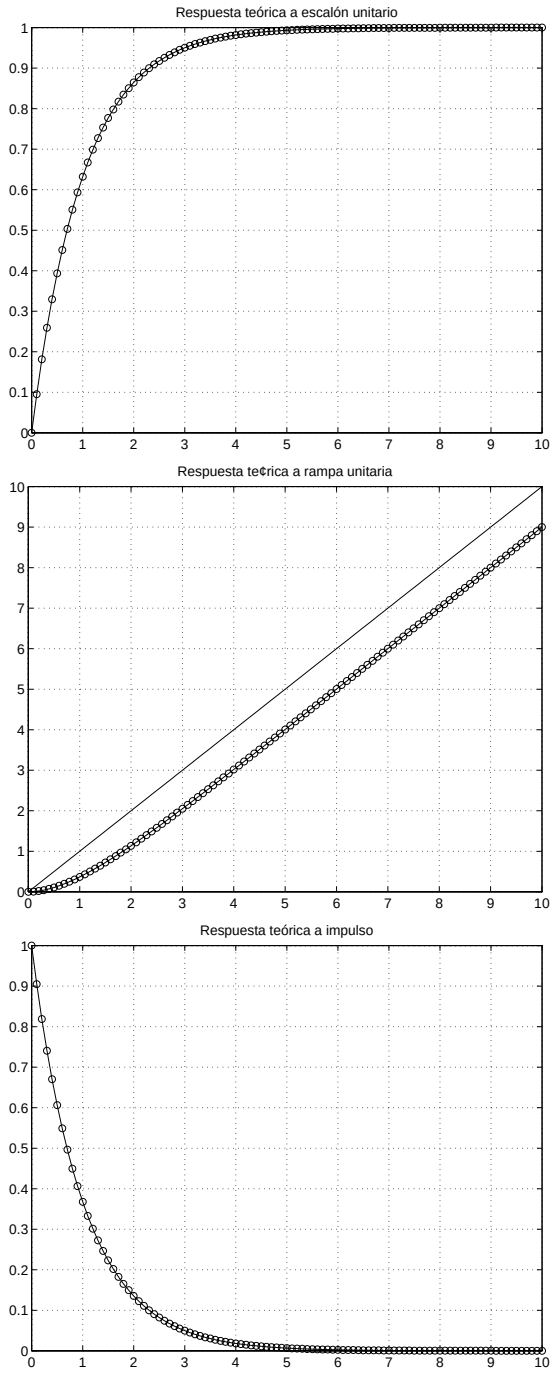

Figura 1.17: Respuesta a escalón, rampa e impulso del sistema de primer orden

Caso 2: Si $\delta = 1 \rightarrow$ 2 raíces reales iguales en SPI (límite sobre-sub), sistema críticamente amortiguado.

Caso 3: Si $0 < \delta < 1 \rightarrow$ raíces complejas conjugadas en SPI (subamortiguado)

Caso 4: Si $\delta = 0 \rightarrow$ Respuesta oscilatoria. Sistema críticamente estable. Raíces en eje imaginario.

Caso 5: Si $\delta < 0 \rightarrow$ Sistema inestable, raíces en SPD.

Se va a analizar el comportamiento para el conjunto de valores de δ en la respuesta a un escalón unitario. Se supone $w_n = 1$. Dado que el caso 3 se verá con más detalle, dado su mayor interés, se presentará en último lugar.

Primer caso: 2 raíces reales distintas (Fig. 1.18).

```
t = [0:0.2:20]';
wn = 1;
d = 2;
num = [wn^2];
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
plot (t,ye);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

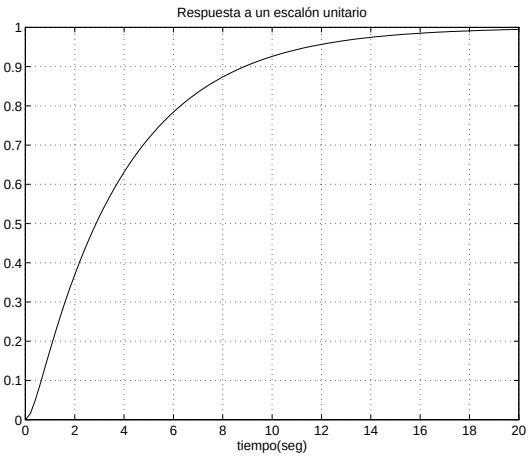

Figura 1.18: Respuesta a escalón del sistema de segundo orden con dos raíces reales distintas

Segundo caso: 2 raíces reales iguales (críticamente amortiguado) (Fig. 1.19). El sistema, en este caso el lo más rápido posible, antes de hacerse subamortiguado.

```
d = 1;
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
plot (t,ye);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

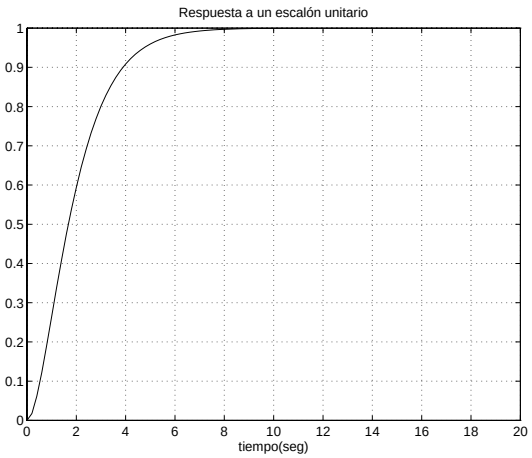

Figura 1.19: Respuesta a escalón de un sistema segundo orden con dos raíces reales iguales

Cuarto caso: Sistema en punto crítico de oscilación (Fig. 1.20).

```
d = 0;
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
plot (t,ye);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

Quinto caso: Sistema inestable (Fig. 1.21).

```
d = -0.1;
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
plot(t,ye);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

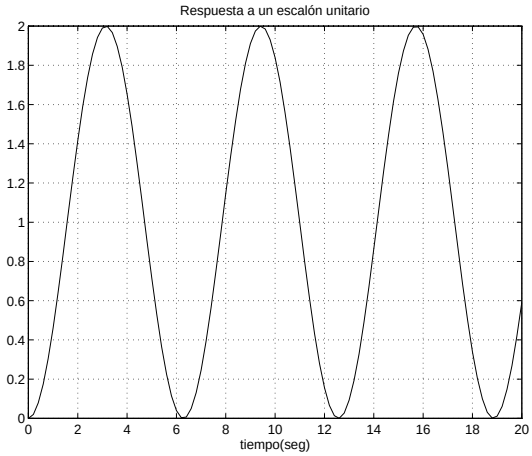

Figura 1.20: Respuesta a escalón de un sistema de segundo orden críticamente estable

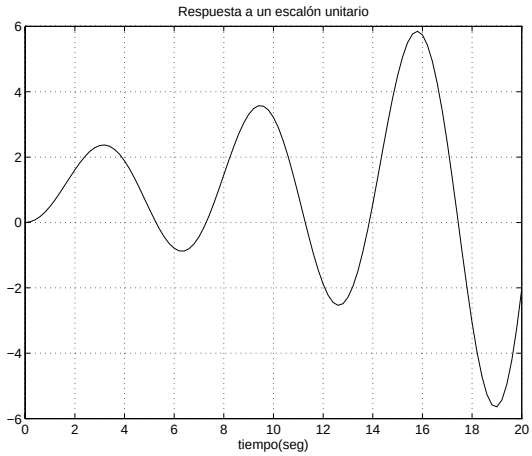

Figura 1.21: Respuesta a escalón de un sistema de segundo orden inestable

Tercer caso: Dos raíces complejas conjugadas (Fig. 1.22).

```
d = 0.5;
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
plot (t,ye);
title ('Respuesta a un escalo'n unitario');
xlabel ('tiempo(seg)');
grid;
```

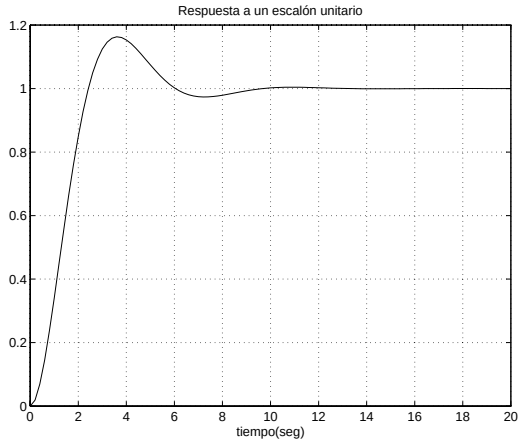

Figura 1.22: Respuesta a escalón de un sistema segundo orden subamortiguado

Se puede también analizar para este tercer caso de dos raíces complejas el efecto de modificar el factor de amortiguamiento. Se muestra para valores $\delta = \{0.1, 0.2, ...0.9\}$ (Fig. 1.23).

```
t = [0:0.2:20]';
wn = 1;
vectDelta = [0.1:0.1:0.9];
num = wn^2;
Y = [];
for ind = 1:length(vectDelta)
   d = vectDelta(ind);
   den = [1,2*d*wn,wn^2];
   y = step (num,den,t);
   Y = [Y, y];
end
plot (t,Y);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

Se mantiene el valor de $\delta = 0.2$ para el siguiente análisis:

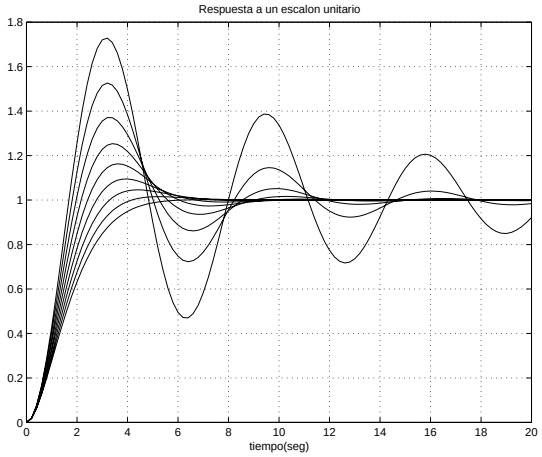

Figura 1.23: Familia de respuestas escalón de un sistema segundo orden subamortiguado

```
d = 0.2;
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
```

La salida del sistema viene dada por la ecuación:

$$y_e(t) = 1 - \frac{e^{-(\delta w_n t)}}{\sqrt{(1-\delta^2)}} sen(w_d t + \phi) \quad , \qquad \text{con :} \quad \phi = \text{arctg}\frac{\sqrt{(1-\delta^2)}}{\delta}$$

y donde $w_d = w_n\sqrt{(1-\delta^2)}$, siendo w_d la frecuencia natural amortiguada.

Las envolventes de la respuesta anterior vendrán dadas por:

$$1 + \frac{e^{-(\delta w_n t)}}{\sqrt{(1-\delta^2)}} \qquad y \qquad 1 - \frac{e^{-(\delta w_n t)}}{\sqrt{(1-\delta^2)}}$$

```
ev1 = 1 + ((exp(-d*wn*t)/(sqrt(1-d^2))));
ev2 = 1 - ((exp(-d*wn*t)/(sqrt(1-d^2))));
plot (t,ye,t,ev1,t,ev2);
title ('Respuesta de sist. segundo orden')
xlabel ('tiempo (s)');
ylabel ('salida');
grid;
```

Los parámetros característicos del transitorio vienen dados por:

Tiempo de pico: $t_p = \frac{\pi}{w_d} = \frac{\pi}{w_n\sqrt{(1-\delta^2)}}$

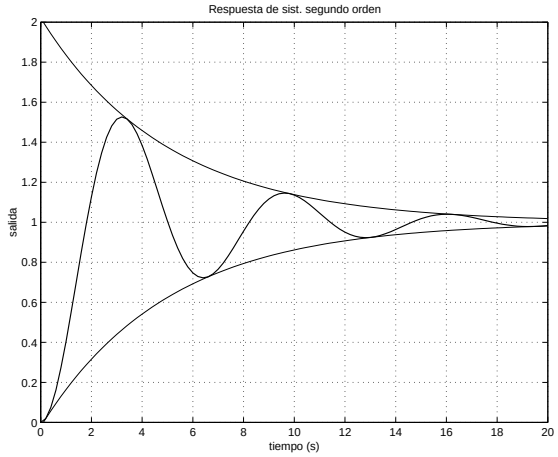

Figura 1.24: Envolventes de la respuesta de un sistema de segundo orden subamortiguado

Tiempo de subida: $t_s = \frac{\pi - \phi}{w_d}$

Sobreoscilación (%): $S0 = e^{-\frac{\delta\pi}{\sqrt{(1-\delta^2)}}}$

Estos valores se pueden calcular analíticamente y compararlos con los obtenidos directamente de la gráfica o analizando el vector de resultados.

```
% Valor de la salida en reg. perm:
yeRP = 1;
% Calculo SO teorica:
SOana = exp (-(d*pi)/(sqrt(1-d^2)));
% A partir de la respuesta obtenida:
SO = max(ye) - yeRP;

% Tiempo de pico teorico:
Tpana = pi/(Wn*sqrt(1-d^2));
% A partir de la respuesta. Aunque existen muchas formas de calcularlo,
% una trivial es la siguiente (la precision viene determinada por la
% primera cifra decimal, debido al modo de definir el vector de tiempos):
for i = 1:length(ye)
   if ye(i) == max(ye)
      Tp = t(i);
      break;
   end
end

% Tiempo de subida teorico:
fi = atan (sqrt(1-d^2)/d);
Tsana = (pi-fi) / (wn*sqrt(1-d^2));
```

```
% A partir de la respuesta. Por definicion, el tiempo de subida es aquel
% para el cual la salida iguala por primera vez al valor en reg. perm.
for i = 1:length(t)
   if (ye(i) <= yeRP  &  ye(i+1) >= yeRP)
      Ts = t(i);
      break;
   end
end

disp 'Sobreoscilaciones'; [SOana SO]
disp 'Tiempos de pico';   [Tpana Tp]
disp 'Tiempos de subida'; [Tsana Ts]
```

1.3.3 Análisis del efecto de un cero en la respuesta temporal de un sistema de segundo orden

Los ceros afectan al valor de la ganancia y a la forma de respuesta transitoria, pero no a la estabilidad. Supongamos la función de transferencia siguiente:

$$G(s) = \frac{K\frac{w_n^2}{z}(s+z)}{s^2 + 2\delta w_n s + w_n^2}$$

Según la posición de los polos y los ceros se tendrán comportamientos diferentes.

Si el valor absoluto del cero es mucho mayor que el valor absoluto de la parte real de los polos, apenas varía la forma de la respuesta típica (polos dominantes). Ver Fig. 1.25.

```
t = [0:0.2:20]';
K = 1;
wn = 1;
d = 0.5;
c = 10;
% Numeradores sin cero y con cero:
num  = K * wn^2;
num2 = K * (wn^2/c) * [1 c];
den  = [1 2*d*wn wn^2];
% Comparamos la salida con la correspondiente al mismo sistema sin cero:
y  = step (num,den,t);
y2 = step (num2,den,t);
plot (t,y2,t,y,'o');
title ('Influencia de un cero lejos del eje imaginario');
polos = roots(den);
disp 'Magnitud de la parte real polos:';
abs(real(polos(1)))
```

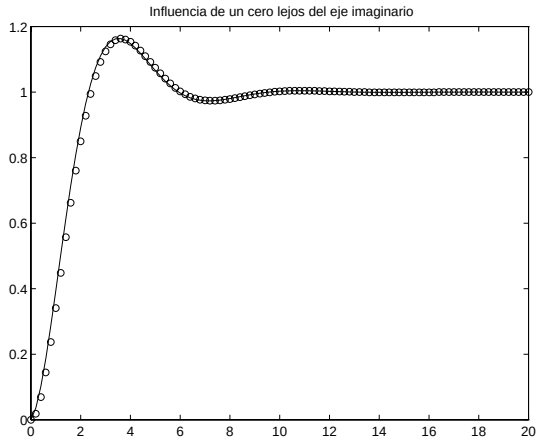

Figura 1.25: Influencia de un cero lejano en la respuesta temporal de un sistema de segundo orden

Si el cero está entre dos raíces reales no varía la forma de la respuesta, pero varía su rapidez (Fig. 1.26), pues se introduce acción derivativa.

```
t = [0:0.2:20]';
K=1;  wn=1;  d=2;  c=0.5;
num  = K*wn^2;
num2 = K*(wn^2/c)*[1 c];
den  = [1 2*d*wn wn^2];

y  = step (num,den,t);
y2 = step (num2,den,t);
plot (t,y2,t,y,'o');
title('Influencia del cero entre polos reales');
disp 'Magnitud de los polos:';
abs(roots(den))
```

Si el cero está más cerca del eje imaginario que los polos reales, aumenta la rapidez de la respuesta (Fig. 1.27), pudiendo la salida rebasar ampliamente su valor de régimen permanente. Como el efecto derivativo es muy grande, aunque el sistema sin el cero no sobrepasara el valor de régimen permanente, el efecto del cero hace que sí sobrepase dicho valor.

```
t = [0:0.2:20]';
K=1;  wn=1;  d=2;  c=0.05;
num  = K*wn^2;
num2 = K*(wn^2/c)*[1 c];
den  = [1 2*d*wn wn^2];

y  = step (num,den,t);
```

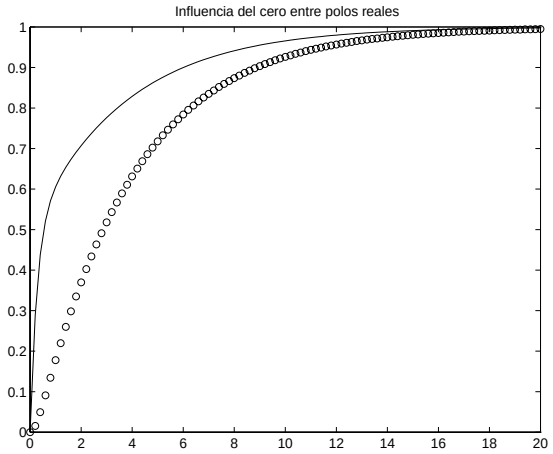

Figura 1.26: Influencia de un cero "dominante" en la respuesta temporal de un sistema de segundo orden

```
y2 = step (num2,den,t);
plot (t,y2,t,y,'o');
title('Influencia del cero cercano al eje imag.');
disp 'Magnitud de los polos:';
abs(roots(den))
```

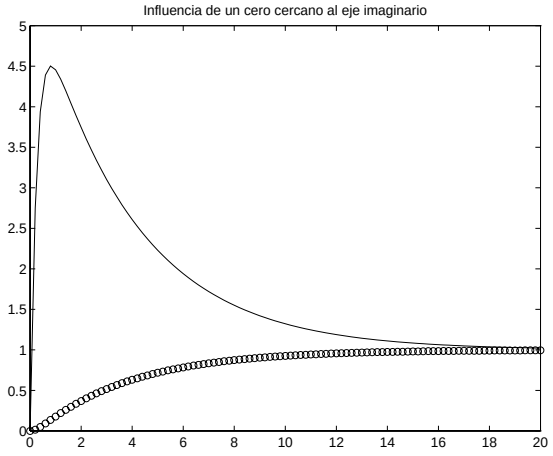

Figura 1.27: Influencia de un cero "muy dominante" en la respuesta temporal de un sistema de segundo orden

Si el cero pasa al semiplano derecho (sistema de fase no mínima) modifica sensiblemente la oscilación (típico por ejemplo en arranque de turbinas). Ver Fig. 1.28. En este caso, la acción derivativa inicial va en sentido contrario a la salida del sistema, por lo que si dicha acción derivativa es grande (cero cercano al eje imaginario), la salida irá inicialmente en sentido contrario a la de régimen permanente.

```
t = [0:0.2:20]';
```

```
K=1;  wn=1;  d=2;  c=-0.5;
num  = K*wn^2;
num2 = K*(wn^2/c)*[1 c];
den  = [1 2*d*wn wn^2];

y  = step (num,den,t);
y2 = step (num2,den,t);
plot (t,y2,t,y,'o');
title('Influencia de un cero positivo');
```

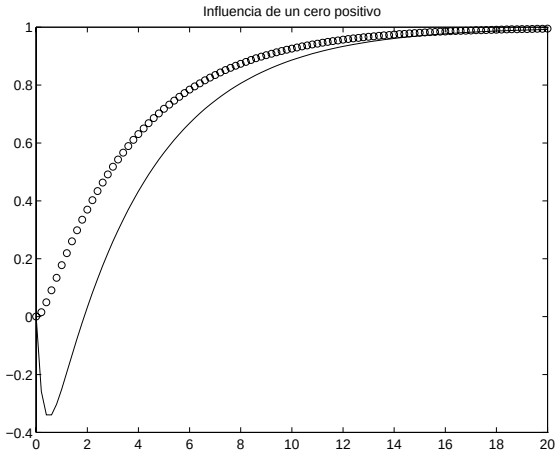

Figura 1.28: Influencia de un cero en el semiplano derecho en la respuesta temporal de un sistema de segundo orden

1.3.4 Influencia de polos adicionales. Polos dominantes

Al régimen transitorio afectan fundamentalmente los polos y ceros cercanos al eje imaginario. El efecto de un cero adicional se ha visto anteriormente. Analizamos ahora el de un polo adicional. Supongamos la función de transferencia siguiente:

$$G(s) = \frac{p\,K\,w_n^2}{(s^2 + 2\delta w_n s + w_n^2)(s+p)}$$

Se va a tomar para el estudio el caso de comportamiento subamortiguado analizado en las secciones anteriores. En dicho caso, la función de transferencia correspondiente tenía los polos con su parte real en -0.5. Si la distancia entre esos polos y el resto es del orden de $5w_n$ (con δ cercano a 1) todos los polos y ceros a partir de esa distancia no afectan al transitorio. De hecho, en el análisis de sistemas, normalmente cuando un sistema tiene un número grande de polos, se trata de simplificar para obtener un sistema aproximado de segundo orden con sus polos localizados en los polos dominantes citados. La influencia de un polo lejano y uno cercano puede verse en las Fig. 1.29 y 1.30. En ambas figuras se muestra la salida del sistema con y sin el polo p adicional comparadas ante una entrada en escalón.

```
t = [0:0.2:20]';
K=1;  wn=1;  d=0.5;  p=10;
num  = K*wn^2;
num2 = wn^2*K*p;
den  = [1 2*d*wn wn^2];
den2 = conv (den,[1 p]);
y  = step (num,den,t);
y2 = step (num2,den2,t);
plot (t,y2,t,y,'o');
title ('Influencia de un polo lejano');
disp 'Situacion de los polos:';
roots(den2)
```

Como se observa en la Fig. 1.29 la influencia es muy pequeña o casi inexistente en este caso.

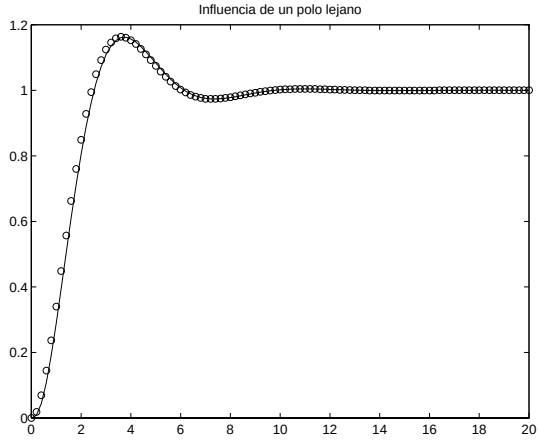

Figura 1.29: Influencia de un polo lejano en la respuesta temporal

Si se analiza el caso de un polo cercano al eje imaginario:

```
t = [0:0.2:20]';
K=1;  wn=1;  d=0.5;  p=0.2;
num  = K*wn^2;
num2 = wn^2*K*p;
den  = [1 2*d*wn wn^2];
den2 = conv (den,[1 p]);
y  = step (num,den,t);
y2 = step (num2,den2,t);
plot (t,y2,t,y,'o');
title ('Influencia de un polo lejano');
disp 'Situacion de los polos:';
roots(den2)
```

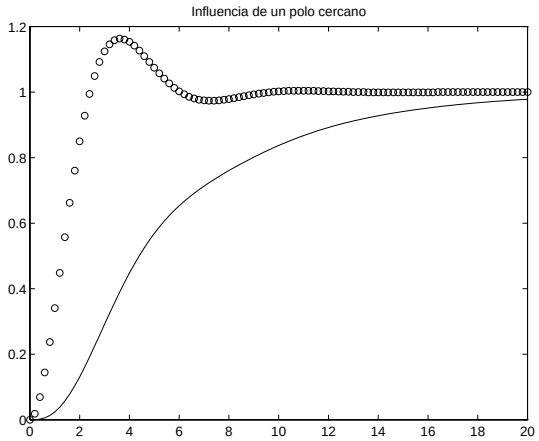

Figura 1.30: Influencia de un polo cercano al eje imaginario

1.4 TRATAMIENTO MEDIANTE FUNCIONES DE TRANSFERENCIA. SISTEMAS DISCRETOS

Se incluyen en esta sección algunas aclaraciones sobre comandos para tiempo discreto, cuya sintaxis suele ser similar, en su caso, a su equivalente continuo, añadiendo una `d` delante. Se van a exponer sólo unas pocas, dejando al lector el análisis por su cuenta del resto.

- `[Nz,Dz] = c2dm (N,D,Ts,'metodo')`: Discretización de un modelo en tiempo continuo, cuya función de transferencia viene dada por los polinomios numerador y denominador. Como tercer parámetro se especifica el periodo de muestreo. El último parámetro proporciona una cadena de caracteres que indica el método con el que se va a hacer la discretización, las posibilidades son:
  - `'zoh'`: Discretización utilizando mantenedor de orden cero (ZOH). Es la opción por defecto.
  - `'foh'`: Discretización utilizando mantenedor de orden uno (FOH).
  - `'tustin'`: Discretización mediante aproximación trapezoidal.
  - `'prewarp'`: Discretización trapezoidal con *prewarping*.
  - `'matched'`: Discretización mediante emparejamiento de polos y ceros (ver [1], pag. 147).

  Se echa de menos en esta función la posibilidad de usar otros métodos de discretización como son el rectangular hacia delante o hacia atrás. También da numerosos problemas cuando se intenta discretizar una función no propia (como pueda ser la función de transferencia de un controlador PID). No es el único comando de MATLAB que tiene esta limitación.

- `[N,D] = d2cm(Nz,Dz,Ts,'metodo')`: Se trata de la función contraria a la anterior. Transforma un sistema discreto en uno continuo, mediante alguno de los métodos citados.

- `dstep(Nz,Dz)`: Calcula la respuesta temporal de un sistema discreto a una secuencia escalón. Esta función dibuja directamente la respuesta. Esta representación no tendrá en el eje horizontal valores temporales absolutos, sino que aparecerán múltiplos del periodo de muestreo. Por otro lado, si indicamos un parámetro de salida en la llamada al comando, `y = dstep (Nz,Dz);`, no se realiza la representación, deberemos hacerlo nosotros mismos. En este caso, sería conveniente pintar la curva con puntos, en lugar de con un trazo continuo (como por defecto hace el comando `plot`). Veámoslo con el siguiente ejemplo, que dará como resultado la figura que aparece a la izquierda en Fig. 1.31:

```
N = [0.2  0.3  1];
D = [1  0.9  1.2  0.5];
[Nz,Dz] = c2dm (N,D,1,'zoh');
y = dstep (Nz,Dz);
plot (y,'.');
title ('Respuesta escal{\'o}n de un sistema discreto');
xlabel ('Periodo de muestreo');
ylabel ('Salida');
grid;
```

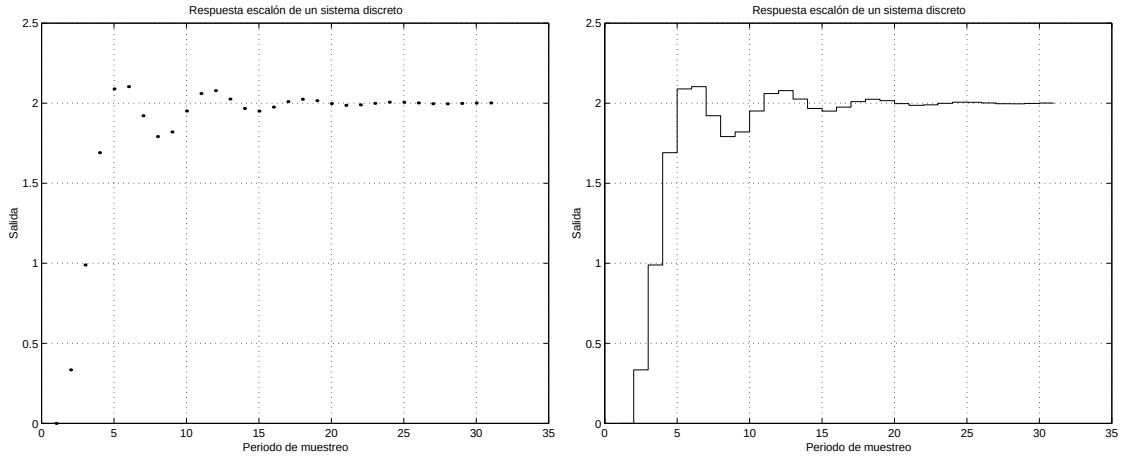

Figura 1.31: Respuesta ante escalón de un sistema discreto, mostrando sólo valores en instantes de muestreo (izq); mostrando salida continua, mantenida entre periodos de muestreo (der)

- `stairs(y)`: Para obtener una respuesta como la de que aparece a la derecha en Fig. 1.31, con la forma escalonada típica de sistemas digitales, simulando que la salida se mantiene constante entre dos periodos de muestreo, bastaría con reemplazar la instrucción `plot(y,'.')` del ejemplo anterior por `stairs(y)`.

- `dimpulse(Nz,Dz)`: Versión discreta de `impulse`.

- `dlsim(Nz,Dz)`: Versión discreto de `lsim`.

- `[mag,phase] = dbode(Nz,Dz,Ts,w)`: Calcula el diagrama de bode de un sistema en tiempo discreto. Siendo w un vector con las frecuencias donde queremos que se evalúen magnitud y fase de la función de transferencia.
- Existen también los correspondientes `dnyquist` y `dnichols`.
- El lugar de las raíces se calcula igual que en dominio s, usando `rlocus`. Sin embargo, se usa una rejilla distinta para que la representación tenga en cuenta el círculo de radio unidad. Por ejemplo, para el sistema discretizado anterior, si hacemos:

```
rlocus (Nz,Dz);
zgrid;
```

  podremos obtener analizar el lugar de las raíces en relación con los lugares geométricos de δ constante y $w_n T_s$ constante, en el plano z (Fig. 1.32).

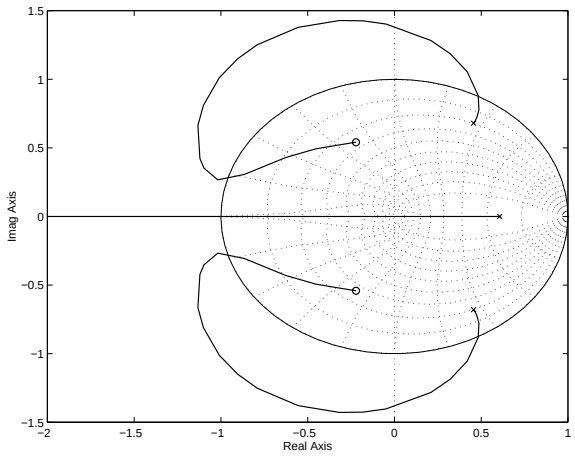

Figura 1.32: Lugar de las raíces del sistema discreto, mostrando rejilla plano Z

1.5 TRATAMIENTO MEDIANTE DESCRIPCIÓN EN EL ESPACIO DE ESTADOS

Se va a utilizar para el análisis la misma función de transferencia que ya apareció anteriormente:

$$H(s) = \frac{0.2s^2 + 0.3s + 1}{(s^2 + 0.4s + 1)(s + 0.5)}$$

El sistema se puede representar en el espacio de estados

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

Existen comandos que permiten obtener una descripción de un sistema en espacio de estados (*ss*) a partir de una función de transferencia, venga esta dada mediante polinomios numerador-denominador (*tf*) o mediante polos-ceros (*zp*), y viceversa:

`[A,B,C,D] = tf2ss(num,den)`	Paso de función de transferencia a espacio de estados
`[A,B,C,D] = zp2ss(z,p,k)`	Paso de descripción polo-cero a espacio de estados
`[num,den] = ss2tf(A,B,C,D)`	Paso de espacio de estados a función de transferencia
`[z,p,k] = ss2zp(A,B,C,D)`	Paso de espacio de estados a descripción polo-cero

Supongamos que disponemos de nuestra función de transferencia dada mediante:

```
num = [.2  .3  1];
den = conv([1 .4 1],[1 .5]);
```

con el comando `tf2ss`, podemos obtener las matrices correspondientes a su descripción en espacio de estados: `[A,B,C,D] = tf2ss (num,den)`.

La mayoría de las funciones que se han comentado en secciones anteriores para la manipulación y simulación de sistemas lineales dados por un par numerador-denominador, tienen su correspondencia para espacio de estados. Valgan los ejemplos siguientes:

```
step (A,B,C,D,1,t);
impulse (A,B,C,D,1,t);
[Abc,Bbc,Cbc,Dbc] = cloop (A,B,C,D,-1);
[A12,B12,C12,D12] = series (A1,B1,C1,D1,A2,B2,C2,D2);
[Ad,Bd] = c2dm (A,B,Ts,'metodo')};
```

y muchas otras.

Como es bien sabido, una propiedad fundamental de los sistemas es el concepto de estabilidad. Si se considera la ecuación no forzada $\dot{x} = Ax, \quad x(0) = x_0$, se dice que el sistema es asintóticamente estable si el estado alcanza el valor cero asintóticamente con el tiempo, es decir, $x(t) \to 0$ con $t \to \infty$. Se puede demostrar que esto ocurre cuando los autovalores de la matriz A tienen partes reales negativas. Por tanto, se puede analizar la estabilidad encontrando los autovalores de la matriz A, usando el comando:

```
evalues = eig (A)
```

1.5.1 Diseño de reguladores en el espacio de estados

Se analizan en esta sección una serie de comandos de MATLAB de gran utilidad para el diseño y simulación de esquemas de control por realimentación lineal del vector de estados:

- `K = place(A,B,P)`: Calcula la matriz o vector K de tal forma que los autovalores de $A - B * K$ (matriz de transición de estados del sistema en bucle cerrado) sean los especificados en el vector P.
- `K = acker(A,B,P)`: Calcula la matriz o vector de ganancias K tal que el sistema de una sola entrada $\dot{x} = Ax + Bu$, con una ley de control $u = -Kx$, tenga los polos en bucle cerrado en los valores especificados en el vector P. Es una implementación de la fórmula de Ackerman).
- `[y,x,t] = initial(A,B,C,D,x0)`: Proporciona la respuesta que describe el comportamiento de un sistema lineal continuo de la forma:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

  ante una cierta condición inicial `x0` de los estados. Devuelve la evolución temporal de la salida y de los estados Existe la correspondiente versión discreta de este comando `dinitial`.
- `Co = ctrb(A,B)`: Devuelve la matriz de controlabilidad del sistema
- `Ob = obsv(A,C)`: Devuelve la matriz de observabilidad.
- `[Ac,Bc,Cc,T,K] = ctrbf(A,B,C)`: Devuelve una descripción en espacio de estados en forma canónica de control, separando los subespacios controlables y no controlables.
- `[Ao,Bo,Co,T,K] = obsvf(A,B,C)`: Devuelve una descripción en espacio de estados en forma canónica de observación, separando los subespacios observables y no observables.

Ejemplo de diseño: Mostramos a continuación un breve ejemplo de un diseño de un controlador por realimentación lineal del vector de estados con un observador de orden completo:

```
% Se obtiene una descripcion continua en espacio de estados:
[A,B,C,D] = tf2ss (num,den);
% Se calcula, por ejemplo, la forma canonica de observacion:
[Ao,Bo,Co,T,J] = obsvf (A,B,C);
% Se calcula la dinamica deseada para el bucle cerrado (3 polos en s=-2):
Pd = poly ([-2,-2,-2]);
% Se calcula el vector de ganancias para realimentacion del vector de estados:
K = acker (Ao,Bo,[-2,-2,-2]);
% O bien se realiza manualmente:
```

```
K_ = [0 0 1] * inv([Bo, Ao*Bo, Ao^2*Bo]) * polyvalm(Pd,Ao);
% Se calcula un observador de orden completo, con una din{\'a}mica deseada:
L = place (Ao',Co',[-5,-5+j,-5-j]);
L = L';
% Formamos el sistema en bucle cerrado:
Abc = [Ao-Bo*K, -Bo*K; zeros(size(Ao)), Ao-L*Co];
Bbc = 0 * [Bo;Bo];
Cbc = [Co Co];
Dbc = 0;
x0 = [1;1;1;-1;-1;-1];
[y,x,t] = initial (Abc,Bbc,Cbc,Dbc,x0);
plot (t, [y, x]);
title ('Control por realim. vector estados con observador');
xlabel ('tiempo (s)');
ylabel ('Salida y estados');
```

El resultado del programa indicado aparece en la Fig. 1.33, donde se muestran tanto la evolución temporal de los estados reales partiendo de condiciones iniciales no nulas como los errores de estimación (suponiendo también un valor no nulo inicialmente en dicho error de estimación).

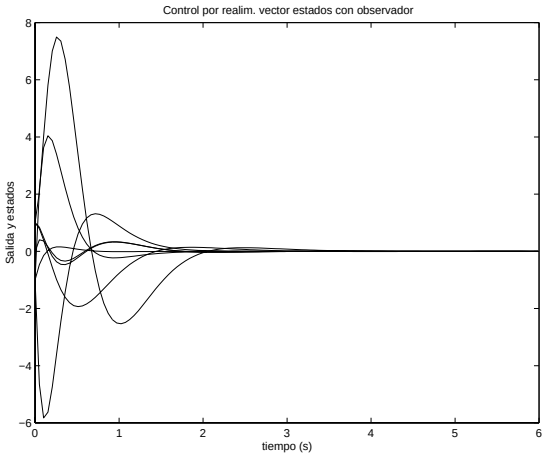

Figura 1.33: Simulación de la evolución de la salida y los estados reales

1.6 Manipulación mediante objetos

En las nuevas versiones del paquete de control (en consonancia con la nueva filosofía de MATLAB), se ofrece la posibilidad de manipular los sistemas mediante unas estructuras de datos específicas para modelos de sistemas lineales. Estos "objetos" harán más fácil la manipulación de los sistemas, pudiendo ser tratados de forma más abstracta, independientemente de que vinieran descritos mediante funciones de transferencia, conjuntos de polos y ceros o matrices correspondientes a una descripción en espacio de estados.

Para empezar, veamos justamente las tres formas de crear un "objeto de tipo sistema", en función de cómo proporcionemos la descripción de dicho sistema:

- `sys = tf(num,den)`
- `sys = zpk(ceros,polos,k)`
- `sys = ss(A,B,C,D)`

En cualquiera de los casos, se puede añadir un parámetro adicional que sería indicativo de un periodo de muestreo, y por tanto, el sistema sería discreto.

Para cada una de las tres posibilidades el objeto que se crea tiene una estructura diferente, puesto que tiene que almacenar datos de distinto tipo.

Supongamos que creamos dos sistemas mediante sendas funciones de transferencia:

```
sys1 = tf (1,[1 3 1]);
sys2 = tf ([1,2],1);
sys1

    Transfer function:
             1
       -------------
       s^2 + 3 s + 1
```

Como vemos, al pedirle el valor de *sys1* nos lo da de una forma monolítica, que hará muy cómoda su manipulación. Por ejemplo, podríamos sumar o poner en serie ambos sistemas sin más que:

```
sys1+sys2
       Transfer function:
       s^3 + 5 s^2 + 7 s + 3
       ---------------------
           s^2 + 3 s + 1
sys12 = sys1*sys2
       Transfer function:
           s + 2
       -------------
       s^2 + 3 s + 1
```

Por otra parte, las propiedades que están asociadas a estos objetos pueden verse mediante:

```
get(sys1)
              num: {[0 0 1]}
              den: {[1 3 1]}
         Variable: 's'
               Ts: 0
       InputDelay: 0
      OutputDelay: 0
    ioDelayMatrix: 0
        InputName: {''}
       OutputName: {''}
       InputGroup: {0x2 cell}
      OutputGroup: {0x2 cell}
            Notes: {}
         UserData: []
```

Nos informa del numerador y denominador de la función de transferencia en s. Al ser continua, su periodo de muestreo es $T_s = 0$. Se pueden asociar retardos a las entradas o salidas, asociar nombres a entradas o salidas, etc. Si en un momento dado, quisiéramos cambiar alguna propiedad podríamos hacerlo mediante el operador `.`:

```
sys1.Ts = 1;
sys1.num{1} = [0 0 2];
sys1
      Transfer function:
            2
      -------------
      z^2 + 3 z + 1
```

el nuevo sistema es discreto, con periodo de muestreo $T_s = 1$ y con un polinomio ganancia 2 en el numerador (no confundir esta manipulación con una discretización, en este caso el sistema continuo original y el discreto no tienen porqué guardar ninguna relación).

Estos objetos también están pensados para poder manipular cómodamente sistemas de múltiples entradas y múltiples salidas. De hecho, podríamos haber creado un sistema de la siguiente forma:

```
num1 = 0.5;   num2 = [1 1];
den1 = [1 2 1]; den2 = [1 0];
sys = tf ({num1,num2},{den1,den2})

            Transfer function from input 1 to output:
                 0.5
            -------------
            s^2 + 2 s + 1
```

```
            Transfer function from input 2 to output:
            s + 1
            -----
              s
% O lo que hubiera sido igual:
sys1 = tf (num1,den1);
sys2 = tf (num2,den2);
sys = [sys1, sys2];
```

1.7 Resumen de los comandos más importantes del Control System Toolbox

Normalmente, en la nomenclatura usada en este paquete de control, los comandos referentes a tiempo discreto que posean un equivalente para tiempo continuo, se denominan igual que éstos, pero precedidos de una `d`. Por ejemplo: `step` y `dstep`.

	Construcción de modelos
`append`	Agrupa dinámica de varios sistemas
`blkbuild`	Construye un sistema en representación en espacio de estados a partir del diagrama de bloques
`cloop`	Calcula el bucle cerrado de un sistema
`connect`	Modelado con diagrama de bloques
`feedback`	Conexión de sistemas realimentados
`ord2`	Genera las matrices A, B, C y D para un sistema de segundo orden
`pade`	Aproximación de Padé a un retardo
`parallel`	Conexión de sistemas en paralelo
`series`	Conexión de sistemas en serie

	Conversión de modelos
`c2d`	Conversión de sistema continuo a tiempo discreto
`c2dm`	Conversión de sistema continuo a tiempo discreto por varios métodos
`c2dt`	Conversión de sistema continuo a discreto con retardo
`d2c`	Conversión de sistema en tiempo discreto a continuo
`d2cm`	Conversión de sistema en tiempo discreto a continuo
`ss2tf`	Paso de representación en espacio de estados a función de transferencia
`ss2zp`	Conversión de espacio de estados a representación polo-cero
`tf2ss`	Paso de representación externa (función de transferencia) a interna (espacio de estados)
`tf2zp`	Conversión de función de transferencia a representación polo-cero
`zp2tf`	Paso de representación polo-cero a función de transferencia
`zp2ss`	Paso de representación polo-cero a espacio de estados

	Reducción de modelos
`minreal`	Realización mínima y cancelación polo-cero
`modred`	Reducción del orden del modelo

	Realización de modelos
`canon`	Conversión de un sistema a forma canónica
`ctrbf`	Matriz de controlabilidad en escalera
`obsvf`	Matriz de observabilidad en escalera

	Propiedades de los modelos
`ctrb`	Matriz de controlabilidad
`damp`	Factores de amortiguamiento y frecuencias naturales
`dcgain`	Ganancia en régimen permanente
`ddamp`	Factores de amortiguamiento y frecuencias naturales discretas
`ddcgain`	Ganancia discreta en régimen permanente
`eig`	Autovalores del sistema
`esort`	Clasifica los autovalores según su parte real
`obsv`	Matriz de observabilidad
`roots`	Raíces del polinomio

	Solución de ecuaciones
`are`	Solución algebraica de la ecuación de Riccati
`dlyap`	Solución de la ecuación de Lyapunov discreta
`lyap`	Solución de la ecuación de Lyapunov continua

	Respuesta temporal
`dimpulse`	Respuesta a impulso unitario en tiempo discreto
`dinitial`	Condiciones iniciales para la respuesta en tiempo discreto
`dlsim`	Simulación para entradas arbitrarias en tiempo discreto
`dstep`	Respuesta a escalón unitario en tiempo discreto
`filter`	Simulación de un sistema SISO en tiempo discreto
`impulse`	Respuesta impulsional continua
`initial`	Condiciones iniciales para la respuesta temporal
`lsim`	Simulación continua para entradas arbitrarias
`step`	Respuesta continua a escalón unitario

	Respuesta frecuencial
`bode`	Diagrama de Bode
`dbode`	Diagrama de Bode para tiempo discreto
`dnichols`	Ábaco de Nichols para tiempo discreto
`dnyquist`	Diagrama de Nyquist para tiempo discreto
`freqs`	Transformada de Laplace
`freqz`	Transformada Z
`margin`	Márgenes de fase y ganancia
`nichols`	Ábaco de Nichols
`ngrid`	Líneas de relleno para el ábaco de Nichols
`nyquist`	Diagrama de Nyquist

	Lugar de las raíces
`pzmap`	Mapeo polo-cero
`rlocfind`	Determinación interactiva de la ganancia en el lugar de las raíces
`rlocus`	Lugar de las raíces
`sgrid`	Relleno del lugar con líneas que indican w_n, δ constantes
`zgrid`	Relleno del lugar para tiempo discreto con líneas de w_n y δ constantes

	Selección de la ganancia
`acker`	Situación de los polos del bucle cerrado en sistemas SISO
`dlqe`	Diseño de un estimador lineal-cuadrático para tiempo discreto
`dlqew`	Diseño de un estimador general lineal-cuadrático para tiempo discreto
`dlqr`	Diseño de un regulador lineal-cuadrático para tiempo discreto
`dlqry`	Diseño de un regulador lineal-cuadrático para tiempo discreto con pesos en las salidas
`lqe`	Estimador lineal-cuadrático
`lqed`	Diseño de un estimador para tiempo discreto partiendo de una función de coste continua
`lqew`	Estimador lineal-cuadrático general
`lqr`	Regulador lineal-cuadrático
`lqrd`	Diseño de un regulador en tiempo discreto a partir de una función de coste continua
`lqry`	Regulador con pesos en las salidas
`place`	Situación de los polos dominantes

	Utilidades
`abcdchk`	Analiza la consistencia del grupo (A, B, C, D)
`dfrqint`	Selector automático de rango para diagramas de Bode en tiempo discreto
`dfrqint2`	Selector automático de rango para diagramas de Nyquist en tiempo discreto
`freqint`	Selector automático de rango para diagramas de Bode
`freqint2`	Selector automático de rango para diagramas de Nyquist

Bibliografía

[1] G.F. Franklin and J.D. Powell. *Digital Control of Dynamic Systems*. Addison-Wesley, 1980.