**Universidad Nacional de La Plata**
**Facultad de Ingeniería**
**Departamento de Electrotecnia**
**Cátedra de Control Moderno**

# Introducción al Control System Toolbox

**Fernando D. Bianchi**

**Año 2001**

# 1. Introducción

El *Control Systems Toolbox* es un conjunto de rutinas para MATLAB dedicadas a aplicar las distintas herramientas de la teoría de control clásico para sistemas lineales. El objetivo de este apunte es dar las nociones básicas para la utilización de este paquete, dejando que el lector interesado en mayores detalles consulte la ayuda y los manuales en línea.

Podemos dividir este toolbox en cuatro subgrupos,

- ▷ funciones para definir modelos
- ▷ funciones para obtener la respuesta temporal
- ▷ funciones para obtener la respuesta en frecuencia
- ▷ funciones para el diseño de controladores

Dado el sistema que deseamos analizar, deberemos obtener primeramente las ecuaciones dinámicas que describen su comportamiento. En caso que estas ecuaciones sean no lineales, las linealizaremos en la cercanía de un punto de operación. De este proceso tendremos una descripción del sistema ya sea como transferencia o como variables de estado, la cual será nuestro punto de partida.

Comenzaremos primero con los distintas formas de ingresar modelos. Luego, pasaremos a las herramientas de análisis, respuesta en tiempo y frecuencia, para terminar con algunas funciones para el diseño de controladores.

# 2. Modelos

## 2.1. Construcción de modelos

El primer paso para utilizar el *Control Systems Toolbox* es ingresar el modelo del sistema. Esto lo podemos hacer de tres formas, como modelo de estados, como transferencias o como polos y ceros.

---

*Nota:* Las versiones del *Control Systems Toolbox* para MATLAB 5.x almacenan los sistemas en objetos. Esto permite tener en una sola variable toda la información necesaria del sistema, facilitando considerablemente el trabajo con las funciones del toolbox. En versiones anteriores, las funciones tomaban como parámetros ya sea los polinomios numerador y denominador, en caso de transferencias, o las matrices $A$, $B$, $C$ y $D$, en caso de variables de estado.

---

### Modelos de estados

Si el modelo del sistema lo tenemos como variables de estado, es decir,

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

Utilizaremos la función **ss** para obtener un objeto denominado *ss*, el cual contiene toda la información del sistema. Por ejemplo,

```
» A = [0 1 0;0 0 1;-18 -27 -10];
» B = [0;0;1];
» C = [1 0 0];
» D = 0;
» S = ss(A,B,C,D)
```

```
a =
                    x1           x2           x3
          x1         0            1            0
          x2         0            0            1
          x3       -18          -27          -10

b =
                    u1
          x1         0
          x2         0
          x3         1

c =
                    x1           x2           x3
          y1         1            0            0

d =
                    u1
          y1         0

Continuous-time model.
```

El objeto `S` contiene entre otras las *propiedades* `a`, `b`, `c` y `d`. Luego, para acceder a la matriz $A$ del sistema haremos:

```
» S.a
ans =
     0     1     0
     0     0     1
   -18   -27   -10
```

Es decir, el nombre del sistema seguido de `.a`. También es posible obtener la matrices del sistema con la función **ssdata**,

```
» [A,B,C,D] = ssdata(S)
A =
     0     1     0
     0     0     1
   -18   -27   -10
B =
     0
     0
     1
C =
     1     0     0
D =
     0
```

Para los sistemas de tiempo discreto se utilizan las mismas funciones, solo que hay que agregar el tiempo de muestreo. Si tomamos el ejemplo anterior y suponemos que es discreto con un tiempo de muestreo de $0{,}01seg$,

```
» S = ss(A,B,C,D,0.01)

a =
```

```
                    x1          x2          x3
          x1         0           1           0
          x2         0           0           1
          x3       -18         -27         -10

b =
                    u1
          x1         0
          x2         0
          x3         1

c =
                    x1          x2          x3
          y1         1           0           0

d =
                    u1
          y1         0

Sampling time: 0.01
Discrete-time model.
```

Observemos que MATLAB indica que el modelo es discreto.

### Transferencias

Si el modelo de nuestro sistema lo tenemos expresado como una función de transferencia, podemos utilizar la función **tf**. Por ejemplo, dada:

$$G(s) = \frac{1}{s^3 + 10s^2 + 27s + 18} ,$$

para crear el objeto *tf* haremos:

```
» num = 1;
» den = [1 10 27 18];
» G = tf(num,den)

Transfer function:
           1
------------------------
s^3 + 10 s^2 + 27 s + 18
```

donde `num` y `den` son vectores filas que representan los polinomios del numerador y denominador, respectivamente (Recordemos que MATLAB representa a los polinomios como vectores filas, cuyo primer elemento corresponde al coeficiente de mayor grado).

Para los sistemas discretos utilizaremos la misma función, con el argumento adicional del tiempo de muestreo. Así la misma función de transferencia anterior pero en caso discreto la ingresaremos de la siguiente manera:

```
» G = tf(num,den,0.01)

Transfer function:
           1
------------------------
z^3 + 10 z^2 + 27 z + 18
```

```
Sampling time: 0.01
```

En caso de tener un sistema MIMO (*multiple input multiple output*) deberemos armar una *cell* con los vectores correspondientes a los polinomios de los numeradores y denominadores. Por ejemplo, dada la matriz de transferencias:

$$G(s) = \begin{bmatrix} \dfrac{5s+7}{s^3+10s^2+27s+18} \\ \dfrac{8}{s^3+10s^2+27s+18} \end{bmatrix},$$

construiremos el objeto *tf* de la siguiente manera

```
» num1 = [5 7];
» num2 = 8;
» den1 = [1 10 27 18];
» den2 = [1 10 27 18];
» G = tf({num1;num2},{den1;den2})

Transfer function from input to output...
             5 s + 7
 #1:  ------------------------
      s^3 + 10 s^2 + 27 s + 18

                 8
 #2:  ------------------------
      s^3 + 10 s^2 + 27 s + 18
```

En este caso, para acceder a las propiedades del objeto *tf* haremos:

```
» G.den{1,1}
ans =
     1    10    27    18
» G.num{1,1}
ans =
     0     0     5     7
```

*Nota:* Los arreglos tipo *cell* han sido introducidos en las versiones 5.x. Básicamente son similares a una matriz pero sus elementos pueden ser cualquier tipo de arreglo. Por ejemplo, una *cell* de $2 \times 1$, puede tener una matriz de $5 \times 2$ en el elemento 1 y un objeto tipo *ss* en el 2. Este tipo de arreglo se crea incluyendo dentro de llaves los elementos, por ejemplo:

```
» c1 = {den,S}
c1 =
    [1x4 double]    [1x1 ss]
» c1 = {den;S}
c1 =
    [1x4 double]
    [1x1 ss    ]
```

Observemos que los símbolos `,` y `;` operan de la misma forma que en la construcción de matrices. Para acceder a uno de los elementos es similar a las matrices pero los paréntesis se reemplazan por las llaves,

```
» c1{1,1}
ans =
     1    10    27    18
```

En este tipo de arreglo no están permitidas las operaciones matemáticas disponibles para las matrices.

Para obtener las *propiedades* del objeto *tf* también podemos utilizar la función **tfdata**,

```
» [NUM,DEN] = tfdata(G)
NUM =
    [1x4 double]
    [1x4 double]
DEN =
    [1x4 double]
    [1x4 double]
```

donde `NUM` y `DEN` son *cell* que contienen los polinomios del numerador y denominador, respectivamente. Por ejemplo, para obtener el numerador de la transferencia $G_1(s)$ deberemos hacer,

```
» NUM{1,1}
ans =
     0     0     5     7
```

Para acceder a uno de los elementos de la matriz de transferencia debemos utilizar una notación similar a las matrices, por ejemplo, para obtener la transferencia $G_{11}$

```
» G(1,1)

Transfer function:
        5 s + 7
------------------------
s^3 + 10 s^2 + 27 s + 18
```

el primer índice indica la salida y el segundo la entrada.

### 2.1.1. Ceros-Polos-Ganancia

Si el modelo se encuentra expresado en función de los ceros, los polos y la ganancia, utilizaremos la función **zpk**, la cual crea un objeto *zpk*. Por ejemplo, si tenemos el siguiente modelo:

$$G(s) = \begin{bmatrix} \dfrac{5(s+1{,}4)}{(s+6)(s+3)(s+4)} \\ \dfrac{8}{(s+6)(s+3)(s+4)} \end{bmatrix},$$

haremos:

```
» Z = {[-1.4];[]};
» P = {[-6;-4;-3];[-6;-4;-3]};
» K = [5;8];
» G = zpk(Z,P,K)

Zero/pole/gain from input to output...
          5 (s+1.4)
 #1:  -----------------
      (s+6) (s+4) (s+3)

              8
 #2:  -----------------
      (s+6) (s+4) (s+3)
```

Donde `Z` y `P` son *cell* que contienen en vectores columna los ceros y los polos, respectivamente, y `K` es una matriz con la ganancia de cada elemento de la matriz de transferencias. Observemos que si una transferencia no tiene ceros finitos, se indica con la matriz vacía (`[]`).

En el caso discreto, nuevamente, se utiliza la misma función solo que se debe agregar el tiempo de muestreo.

Los objetos *zpk* tienen entre otras las *propiedades* `z`, `p` y `k`, en donde se almacena los ceros, los polos y las ganancias de los elementos de la matriz de transferencia. Así, para obtener los ceros del elemento $G_1(s)$ haremos:

```
» G.z{1}
ans =
   -1.4000
```

También podemos utilizar la función **zpkdata**, para obtener la misma información:

```
» [z,p,k] = zpkdata(G)
z =
    [-1.4000]
          []
p =
    [3x1 double]
    [3x1 double]
k =
     5
     8
» z{1}
ans =
   -1.4000
```

## 2.2. Conversión de modelos

Es posible que necesitemos pasar de un modelo en variable de estado a transferencia, y viceversa. Para esto simplemente se utilizan las misma funciones **ss**, **tf** y **zpk**. Por ejemplo, si tenemos un modelo `S`, en variables de estado, para convertirlo a transferencia haremos:

```
» S = ss(A,B,C,D);
» T = tf(S)

Transfer function:
          1
------------------------
s^3 + 10 s^2 + 27 s + 18
```

Lo mismo sucederá con las otras posibles combinaciones. Lo que debemos tener en cuenta, es que si bien en los modelos tipos *tf* y *zpk* es posible tener transferencias con más ceros que polos en variables de estado no. Así si tenemos:

```
» T = zpk([-2;-1],[-5],1)

Zero/pole/gain:
(s+2) (s+1)
-----------
   (s+5)
```

no lo podemos pasar a modelo de estados:

```
» S = ss(T)
??? Error using ==> zpk/ss
Improper system: conversion to state space not possible
```

---

*Nota:* Si estamos utilizando una versión inferior a la 5.0, para estas conversiones se dispone de las funciones **tf2ss**, **tf2zp**, **zp2ss**, **zp2tf**, **ss2tf** y **ss2zp**. Por ejemplo, si tenemos la transferencia:

$$G(s) = \frac{1}{s^3 + 10s^2 + 27s + 18} ,$$

para llevarla a modelo de estados:

```
» num = 1;
» den = [1 10 27 18];
» [A,B,C,D] = tf2ss(num,den)
A =
   -10   -27   -18
     1     0     0
     0     1     0
B =
     1
     0
     0
C =
     0     0     1
D =
     0
```

Observemos que para estas versiones del *Control Systems Toolbox* la transferencia se representa por dos vectores, `num` y `den`, en vez de un único objeto *tf*.

---

Otra posibilidad es que deseemos convertir un modelo continuo a uno discreto. Para ello se dispone de las funciones **c2d** y **d2c**. Por ejemplo, si tenemos el siguiente sistema continuo:

```
» G = tf(1,[1 10 27 18])

Transfer function:
           1
-----------------------
s^3 + 10 s^2 + 27 s + 18
```

para convertirlo a uno discreto con tiempo de muestreo de 0,01$seg$:

```
» Gd = c2d(G,0.01)

Transfer function:
1.626e-007 z^2 + 6.342e-007 z + 1.546e-007
-----------------------------------------
    z^3 - 2.902 z^2 + 2.807 z - 0.9048

Sampling time: 0.01
```

Puede agregarse un tercer argumento que indica el método de discretización, por defecto se utiliza *Zero order hold*. (ver `help c2d`). Para la conversión opuesta,

```
» Gc = d2c(Gd,'zho')

Transfer function:
           1
-----------------------
s^3 + 10 s^2 + 27 s + 18
```

También es posible modificar el tiempo de muestreo de un sistema discreto, por ejemplo:

```
» Gd1 = d2d(Gd,0.1)

Transfer function:
0.0001305 z^2 + 0.0004086 z + 7.915e-005
----------------------------------------
   z^3 - 2.194 z^2 + 1.573 z - 0.3679

Sampling time: 0.1
```

Esto produce el mismo resultado que `c2d(d2c(Gd),0.1)`.

## 2.3. Distintas realizaciones de modelos de estados

El modelo de estados de un sistema no es único. Partiendo de las ecuaciones físicas que describen el comportamiento del sistema, se llega a un modelo que se suele denominar de variables físicas. Es común aplicarle una transformación para llevarlo a otro donde ciertas características se pueden analizar más fácilmente. Esto es, dado el modelo del sistema en variable de estado:

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

A través de una matriz de transformación podemos cambiar los estados $x$ por los $\overline{x} = Px$, entonces el nuevo modelo resulta:

$$\begin{aligned} \dot{\overline{x}} &= PAP^{-1}\overline{x} + PBu, \\ y &= CP^{-1}\overline{x} + Du. \end{aligned}$$

El *Control Systems Toolbox* cuenta con dos funciones para estas transformaciones, una **canon** y la otra **ss2ss**. La primera permite llevar un modelo a la forma canónica diagonal. Por ejemplo, dado el siguiente sistema:

```
» A = [-9 -2.5 -1.5;8 0 0;0 1 0];
» B = [2;0;0];
» C = [1 0.4375 0.0625];
» D = 0;
» S = ss(A,B,C,D);
```

la forma diagonal se obtiene de hacer:

```
» Sd = canon(S)

a =
                      x1           x2           x3
           x1         -6            0            0
           x2          0           -2            0
           x3          0            0           -1

b =
                      u1
           x1    -6.0531
           x2    -9.1652
           x3    -4.5431

c =
                      x1           x2           x3
           y1   -0.25607    -0.13639     0.17609

d =
                      u1
           y1          0

Continuous-time model.
```

Donde el sistema `S` debe tener autovalores distintos.

Por otro lado, si disponemos de la matriz de transformación $P$ podemos utilizar la función **ss2ss** para obtener el nuevo modelo. Por ejemplo, es posible aplicar la siguiente matriz de transformación, para pasar el modelo anterior a la forma canónica controlable,

$$P = Q \begin{bmatrix} a_1 & a_2 & 1 \\ a_2 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

donde $Q$ es la matriz de controlabilidad y los coeficientes $a_i$ corresponde al polinomio característico, $\lambda^3 + a_2\lambda^2 + a_1\lambda + a_0$. Primero calculamos $P$ con las funciones disponibles de MATLAB:

```
» p = poly(A);
» W = [p(3) p(2) 1;p(2) 1 0;1 0 0]
W =
   20.0000    9.0000    1.0000
    9.0000    1.0000         0
    1.0000         0         0
» Q = ctrb(S)
Q =
```

```
     2   -18   122
     0    16  -144
     0     0    16
» P = Q*W
P =
   -0.0000    0.0000    2.0000
    0.0000   16.0000         0
   16.0000         0         0
```

y luego, aplicamos **ss2ss**:

```
» Sd = ss2ss(S,inv(P))

a =
                        x1           x2           x3
           x1  3.5527e-015            1            0
           x2 -2.8422e-014            0            1
           x3          -12          -20           -9

b =
                        u1
           x1            0
           x2            0
           x3            1

c =
                        x1           x2           x3
           y1            1            7            2

d =
                        u1
           y1            0

Continuous-time model.
```

También podríamos haber utilizado las operaciones matriciales:

```
» Ac = P\A*P;
» Bc = P\B;
» Cc = C*P;
» Dc = D;
» Sc = ss(Ac,Bc,Cc,Dc)

a =
                        x1           x2           x3
           x1  3.5527e-015            1            0
           x2 -2.8422e-014            0            1
           x3          -12          -20           -9


b =
                        u1
           x1            0
           x2            0
           x3            1
```

```
c =
                    x1          x2          x3
          y1         1           7           2


d =
                    u1
          y1         0

Continuous-time model.
```

En forma similar, se puede hacer la transformación para llevar este modelo a la forma canónica observable.

## 2.4. Características del modelo

Dado un modelo del sistema, existen una serie de funciones dedicadas a determinar las características dinámicas del sistema. Por ejemplo, considerando el siguiente sistema:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -3 & 5 \\ -3 & -1 & -6 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 10 \end{bmatrix} \quad (1)$$

$$y = \begin{bmatrix} 1 & 0 & 5 \end{bmatrix}. \quad (2)$$

Podemos determinar los autovalores con las funciones **pole** y **eig**:

```
» eig(S) ans =
   -1.0000
   -3.0000
   -5.0000
```

(Observemos que la función **eig** reconoce cualquiera de los objetos del *Control Systems Toolbox* devolviendo los autovalores de la matriz $A$). Y los ceros con la función **zero**:

```
» zero(S) ans =
   -2.6180
   -0.3820
```

También, podemos determinar la frecuencia y el amortiguamiento de cada autovalor con la función **damp**:

```
» damp(S)

 Eigenvalue      Damping     Freq. (rad/s)

 -1.00e+000     1.00e+000      1.00e+000
 -3.00e+000     1.00e+000      3.00e+000
 -5.00e+000     1.00e+000      5.00e+000
```

Por último, las funciones **ctrb** y **obvs** calcula las matrices de controlabidad y observabilidad, respectivamente. Esta junto con la función **rank** permite determinar si un sistema es controlable u observable. Por ejemplo, para el sistema anterior,

```
» Q = ctrb(S);
» rank(Q)
ans =
     3
```

Como el rango coincide con el orden del sistema, `S` es controlable.

## 2.5. Conexión de modelos

En la Tabla 1 se listan algunas de la funciones para conectar distintos sistemas. Además de estas funciones se dispone de los operadores dados en la Tabla 2.

| Función | Descripción |
|---|---|
| `series` | Conexión en series de varios sistemas |
| `parallel` | Conexión en paralelo de varios sistemas |
| `feedback` | Realiza la conexión realimentada de dos sistemas |
| `augment` | Crea un sistema aumentado |
| `append` | Agrega sistemas a otro |

Tabla 1: Funciones para la conexión de sistemas

| Operador | Descripción |
|---|---|
| `*` | Conexión en series de varios sistemas |
| `+` o `-` | Conexión en paralelo de varios sistemas |
| `/` o `\` | División de sistemas |

Tabla 2: Operadores para la conexión de sistemas

La función **series** realiza la conexión en cascada. Por ejemplo, dados los siguientes sistemas:

```
» T1 = tf(1,[1 5])

Transfer function:
  1
-----
s + 5

» T2 = tf([1 3],[1 10])

Transfer function:
s + 3
------
s + 10
```

La conexión en cascada de ambos será:

```
» T3 = series(T1,T2)

Transfer function:
     s + 3
---------------
s^2 + 15 s + 50
```

o utilizando el operador `*`,

```
» T3 = T1*T2

Transfer function:
     s + 3
---------------
s^2 + 15 s + 50
```

Para los casos MIMO, la función **series** resulta un tanto más compleja. Si nuestra intención es conectar en serie dos sistemas MIMO, estos deben tener las dimensiones adecuada (se rigen por las mismas reglas que la multiplicación matricial). Por ejemplo, dado los siguientes sistemas:

```
» T1 = tf({1;[1 2]},[1 5])

Transfer function from input to output...
        1
 #1:  -----
      s + 5

      s + 2
 #2:  -----
      s + 5

» T2 = zpk({[],[-4]},{[-6],[-1;7]},[1,8])

Zero/pole/gain from input 1 to output:
  1
-----
(s+6)

Zero/pole/gain from input 2 to output:
  8 (s+4)
-----------
(s+1) (s-7)
```

La conexión en serie de ambos,

```
» T3 = T1*T2

Zero/pole/gain from input 1 to output...
           1
 #1:  -----------
      (s+5) (s+6)

         (s+2)
 #2:  -----------
      (s+5) (s+6)

Zero/pole/gain from input 2 to output...
           8 (s+4)
 #1:  -----------------
      (s+5) (s-7) (s+1)

        8 (s+2) (s+4)
 #2:  -----------------
      (s+5) (s-7) (s+1)
```

resulta en un sistema de dos entradas y dos salidas. Observemos que en sistemas MIMO el orden en que se conectan en serie es importante (calcule el sistema que resulta de hacer `T2*T1`).

La función **series** tiene otros dos argumentos, que en el caso de sistemas MIMO la hacen más versátil. Por ejemplo, si queremos hacer la conexión de los sistemas $T_1$ y $T_2$ como se muestra en la Fig. 1 utilizamos:

```
» T4 = series(T1,T2,[1 2],[2 1])

Zero/pole/gain:
(s+5) (s+3.062) (s^2 + 0.9378s + 58.13)
---------------------------------------
       (s+1) (s+5)^2 (s+6) (s-7)
```

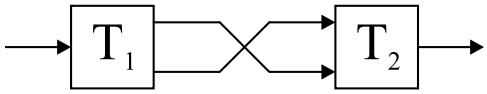


Figura 1: Conexión de dos sistemas utilizando la función **series**.

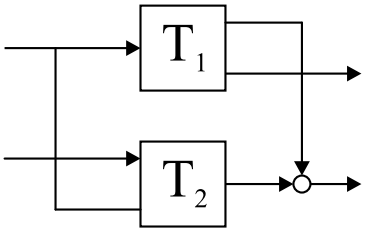


Figura 2: Conexión de dos sistemas utilizando la función **parallel**.

donde el vector `[1 2]` y el `[2 1]` indica que la salida 1 de $T_1$ debe conectarse a la entrada 2 de $T_2$ y que la salida 2 de $T_1$ debe unirse a la entrada 1 de $T_2$.

Para la conexión en paralelo de sistemas tenemos la función **parallel** o los operadores `+` y `-`. Su aplicación es bastante similar a la conexión en serie, solo hay algunas diferencias en los sistemas MIMO. Por ejemplo, considerando nuevamente los sistemas $T_1$ y $T_2$, la conexión indicada en la Fig. 2 la haremos con la siguiente instrucción:

```
» T5 = parallel(T1,T2,1,2,1,1)

Zero/pole/gain from input 1 to output...
      (s+2)
 #1:  -----
      (s+5)

      9 (s^2 + 7.333s + 17)
 #2:  ---------------------
        (s+5) (s-7) (s+1)

Zero/pole/gain from input 2 to output...
 #1:  0

        1
 #2:  -----
      (s+6)
```

Donde el tercer y cuarto argumento indica cuales entradas se deben unir y el quinto y sexto indica cuales salidas se deben sumar.

Terminamos esta sección con la función **feedback**, las restantes se dejan para que el lector investigue por su cuenta. La función **feedback** realiza la conexión de dos sistemas en un lazo de realimentación. Por ejemplo, dados

```
» P = tf({1;[1 2]},[1 5])

Transfer function from input to output...
        1
 #1:  -----
      s + 5
```

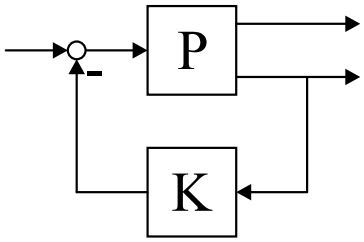


Figura 3: Conexión de dos sistemas utilizando la función **feedback**.

```
      s + 2
 #2:  -----
      s + 5

» K = zpk(-1,-10,1)

Zero/pole/gain: (s+1)
------
(s+10)
```

para realizar la conexión indicada en la Fig. 3 usaremos:

```
» T = feedback(P,K,1,2,-1)

Zero/pole/gain from input to output...
        0.5 (s+10)
 #1:  ---------------
      (s^2 + 9s + 26)

      0.5 (s+10) (s+2)
 #2:  ----------------
      (s^2 + 9s + 26)
```

Donde tercer argumento indica cual entrada de `P` está involucrada en la realimentación, el cuarto cual salida de `P` se realimenta y el último indica el signo de la realimentación. Observemos que el caso de sistemas SISO el tercer y cuarto argumento no son necesarios.

# 3. Respuesta en el tiempo

Una vez que conocemos como ingresar y conectar los sistemas podemos pasar a ver las funciones dedicadas al análisis. Para el cálculo de la respuesta temporal de un sistema LTI el *Control Systems Toolbox* cuenta con las funciones listadas en la Tabla 3.

| **Función** | **Descripción** |
|---|---|
| `step` | Respuesta al escalón |
| `impulse` | Respuesta a un impulso |
| `initial` | Respuesta libre del sistema |
| `lsim` | Respuesta a una dada señal de entrada |
| `gensig` | Genera señales de entrada |

Tabla 3: Funciones para el cálculo de la respuesta temporal

Las funciones **step** e **impulse** se aplican de la misma forma. Por ejemplo, dado el sistema:

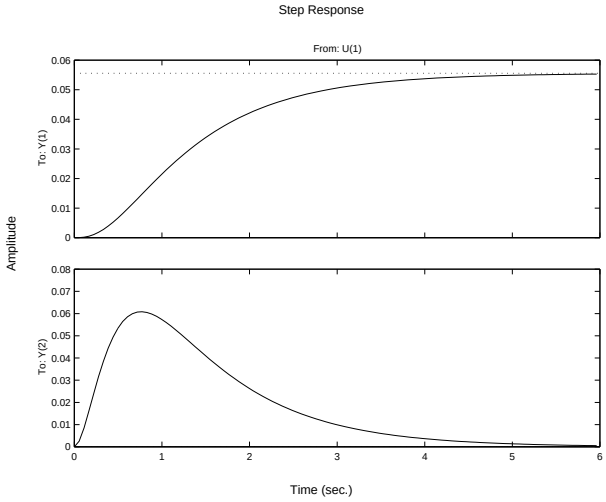


Figura 4: Respuesta al escalón unitario obtenida con la función **step**.

```
» A = [0 1 0;0 0 1;-18 -27 -10];
» B = [0;0;1];
» C = [1 0 0;0 2 0];
» D = 0;
» S = ss(A,B,C,D);
```

para obtener la respuesta al escalón unitario haremos:

```
» step(S)
```

El resultado lo podemos ver en la Fig. 4. Observemos que se hace un subplot en función de la cantidad de entradas y salidas. Con el agregado de un segundo argumento es posible controlar el intervalo de tiempo. Si este argumento es un escalar lo considera como el tiempo final, en cambio si es un vector calcula la respuesta temporal en cada elemento de este. Por ejemplo, para que calcule la respuesta entre 0 y $15seg$ dividiendo este intervalo en 100 puntos utilizaremos las siguientes intrucciones:

```
» t = linspace(0,15,100);
» step(S,t)
```

También acepta argumentos para indicar el tipo de línea usada en el gráfico, el cual son iguales a los utilizados en la función **plot**. Por ejemplo,

```
» step(sys1,'k-',sys2,'r:');
```

grafica la respuesta del sistema `sys1` con línea continua de color negro y la del sistema `sys2` con línea punteada de color rojo.

Para los sistemas discretos, las funciones **step** e **impulse**, se aplican en forma similar, solo que el espaciado del vector `t` debe coincidir con el tiempo de muestreo. Por ejemplo, considerando el siguiente sistema discreto, con un tiempo de muestreo $0{,}01seg$:

```
» G=zpk(-0.5,[-0.9;0.85],2,0.01)

Zero/pole/gain:
   2 (z+0.5)
----------------
(z+0.9) (z-0.85)

Sampling time: 0.01
```

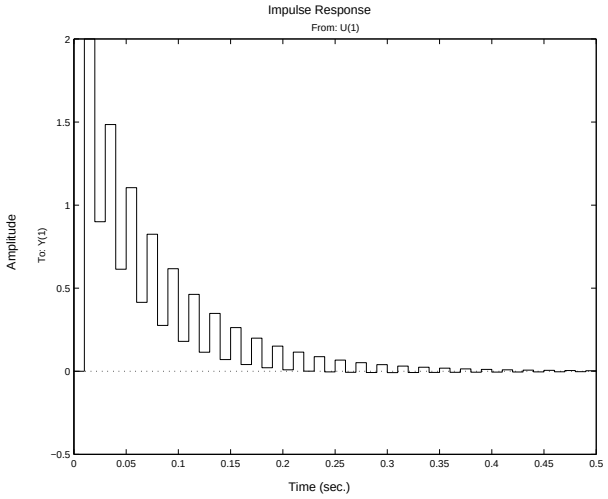


Figura 5: Respuesta al impulso unitario obtenida con la función **impulse**.

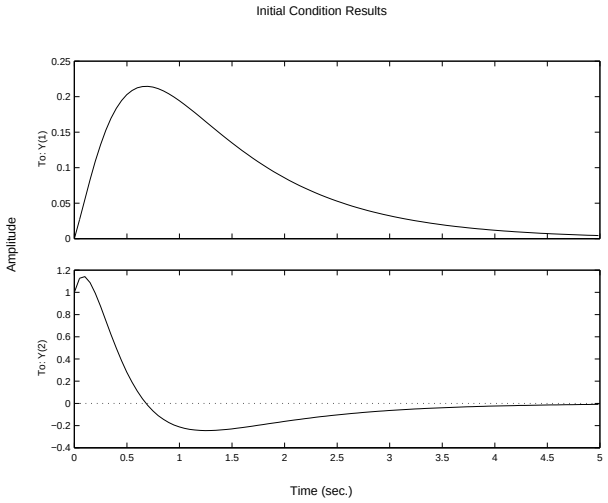


Figura 6: Respuesta al libre obtenida con la función **initial**.

podemos obtener la respuesta al impulso unitario, en el intervalo $[0, 0{,}5]$, de la siguiente manera:

```
» t = 0:0.01:0.5;
» impulse(G,t)
```

En la Fig. 5 se tiene el gráfico resultante.

Para los sistemas descritos en variables de estados, la función **initial** determina la respuesta libre del sistema para cierta condición inicial. Considerando, nuevamente, el sistema `S` que definimos anteriormente,

```
» xo = [0;0.5;2];
» initial(S,xo)
```

donde el vector `xo` son las condiciones iniciales, el cual debe ser un vector columna con tantas filas como estados tenga el sistema. La Fig. 6 muestra la respuesta obtenida. Esta función, al igual que las anteriores, acepta argumentos para especificar el intervalo de tiempo y el tipo de línea utilizada.

En los casos que necesitemos la respuesta temporal ante otras entradas podemos utilizar la función **lsim**. El llamado a esta función tiene la siguiente forma:

```
» lsim(sys,u,t,xo);
```

donde `sys` es el sistema, `u` el vector de entrada, `t` el vector de tiempo y `xo` las condiciones iniciales (solo para sistemas en variables de estado). El vector `u` debe tener tantas filas como elementos tiene `t` y tantas columnas con entradas tenga el sistema.

Es posible usar la función **gensig** para generar el vector de entradas, `u`. Esta tiene la siguiente sintaxis:

```
» [u,t] = gensig(tipo,tau,Tf,Ts)
```

donde `tipo` puede ser "square", para onda cuadrada, "sin", para onda senoidal y "pulse", para onda de pulsos. El argumento `tau` indica el período, `Tf` el tiempo final y `Ts` el tiempo de muestreo.

Consideremos el siguiente sistema:

```
» G = tf({[1 0.5],2},[1 4 3])

Transfer function from input 1 to output:
   s + 0.5
-------------
s^2 + 4 s + 3

Transfer function from input 2 to output:
      2
-------------
s^2 + 4 s + 3
```

La entrada $u_1$ será una señal senoidal de amplitud 0,1 y la $u_2$ un escalón unitario. Construyamos el vector de entrada `u`,

```
» [u1,t] = gensig('sin',1,10,0.01);
» u2 = [zeros(100,1);ones(901,1)];
» u = [0.1*u1,u2];
```

Observemos que hemos generador un escalón unitario, para la entrada $u_2$, simplemente concatenando dos vectores, uno de todos 0 y otro de 1. Luego el vector de entrada `u` lo construimos concatenando los vectores `u1` y `u2`, notemos que el `u1` lo hemos escalado pues tenía amplitud 1. Ahora ya podemos utilizar la función **lsim** para calcular la respuesta del sistema a esta entrada,

```
» lsim(G,u,t)
```

El resultado lo podemos ver en la Fig. 7.

# 4. Respuesta en frecuencia

En la Tabla 4 se enumeran algunas de las funciones que dispone el *Control Systems Toolbox* para obtener la respuesta en frecuencia de un sistema.

Considerando:

```
» G = zpk([-5],[0;-1;-3],1)

Zero/pole/gain:
    (s+5)
-------------
s (s+1) (s+3)
```

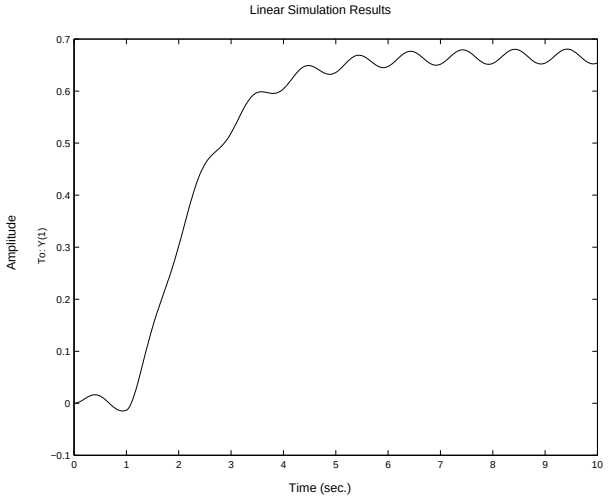


Figura 7: Respuesta a una entrada especificada obtenida con la función **lsim**.

| **Función** | **Descripción** |
|---|---|
| `bode` | Diagrama de Bode |
| `nyquist` | Diagrama de Nyquist |
| `rlocus` | Lugar de raíces |
| `pzmap` | Gráfico de polos y ceros |
| `margin` | Margen de fase y ganancia |

Tabla 4: Funciones para el cálculo de la respuesta en frecuencia

podemos obtener el diagrama de Bode con

```
» bode(G)
```

El gráfico resultante lo podemos ver en la Fig. 8. En forma similar a las funciones para respuesta temporal, podemos controlar el intervalo de frecuencia y el tipo de línea utilizada. Por ejemplo,

```
» w = logspace(-1,2,100);
» bode(G,w)
```

Una función cercana al diagrama de Bode es **margin** que calcula el margen de ganancia y de fase. Utilizándola en el sistema anterior,

```
» margin(G)
```

resulta en el gráfico mostrado por la Fig. 9. Si en cambio, la llamada a la función se realiza con argumentos de salida,

```
» [Gm,Pm,Wcg,Wcp] = margin(G)
Gm =
   12.0000
Pm =
   34.9872
Wcg =
    3.8730
Wcp =
    1.0862
```

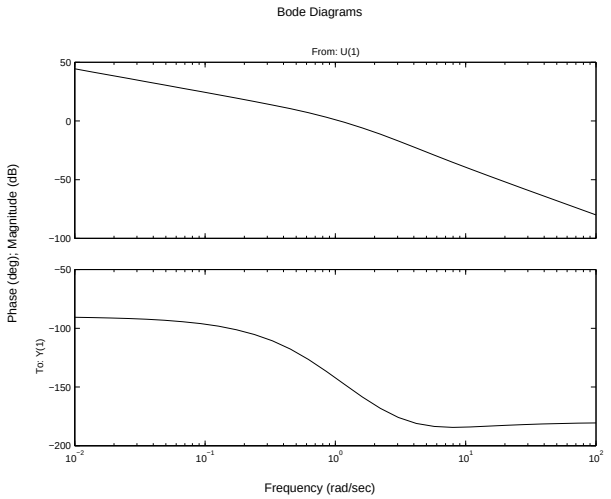


Figura 8: Diagrama de Bode obtenido con la función **bode**.

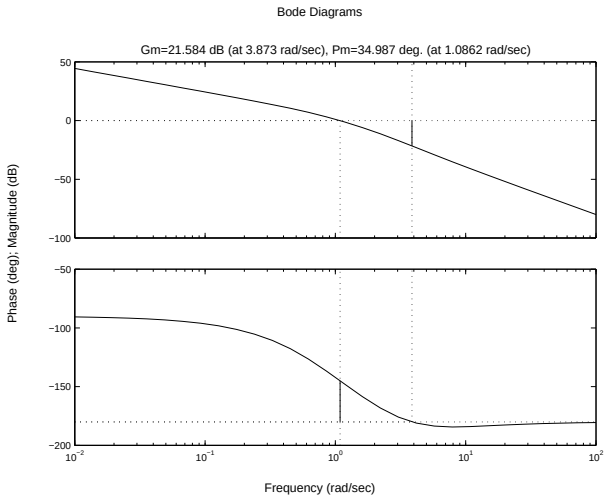


Figura 9: Margen de ganancia y de fase obtenido con la función **margin**.

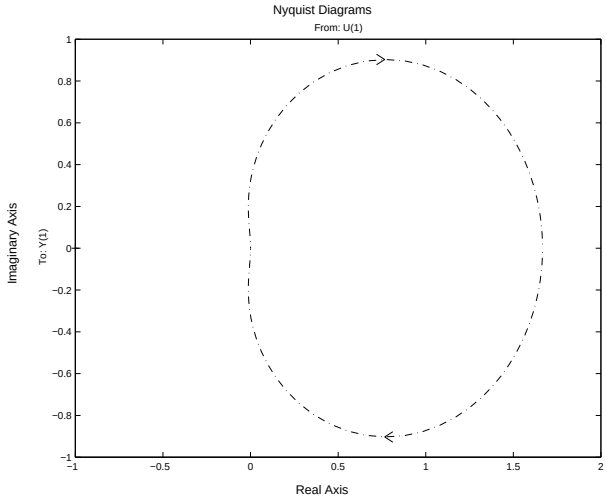


Figura 10: Diagrama de Nyquist obtenido con la función **nyquist**.

tendremos en `Gm` el margen de ganancia, en `Pm` el margen de fase, en `Wcg` la frecuencia en donde se mide el margen de ganancia y en `Wcp` la frecuencia correspondiente al margen de fase.

Por su parte, el diagrama de Nyquist lo obtenemos con la función **nyquist**. Por ejemplo, considerando el siguiente sistema,

```
» G = zpk([-5],[-1;-3],5)

Zero/pole/gain:
  5 (s+5)
-----------
(s+1) (s+3)
```

utilizaremos la función **nyquist** de la siguiente manera:

```
» w = logspace(-2,3,100);
» nyquist(G,w,'k-.')
```

Resultando en el gráfico mostrado en la Fig. 10. Donde hemos utilizado el vector `w` para controlar el intervalo de frecuencia y `k-.` para definir el tipo de la línea utilizada.

Por último, para el diagrama de lugar de raíces tenemos la función **rlocus**. Considerando el sistema anterior con

```
» rlocus(G)
```

obtenemos el diagrama de lugar de raíces mostrado en la Fig. 11.

---

*Nota:* El *Control Systems Toolbox* cuenta con la función **ltiview**, la cual constituye un resumen de las funciones que hemos visto hasta aquí para obtener la respuesta temporal y en frecuencia. Utilizando el comando `ltiview` se abre una *Figure Window* con menus que permiten importar a esta ventana cualquiera de los sistema definidos en el *MATLAB Workspace* y seleccionar el tipo de gráfico.

---

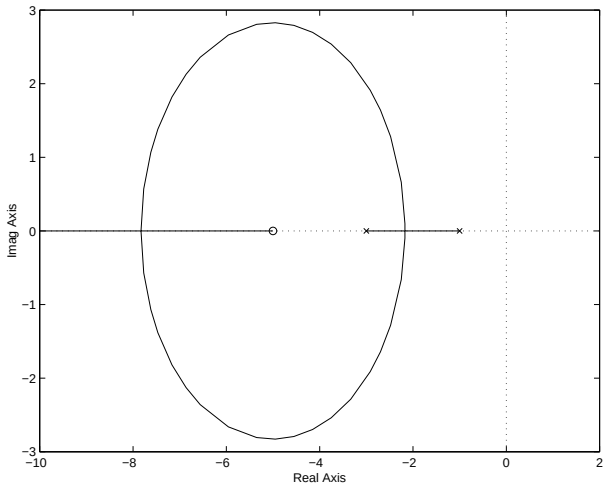


Figura 11: Diagrama de lugar de raíces obtenido con la función **rlocus**.

# 5. Diseño de controladores

Dentro de los métodos de diseño de la teoría clásica de control el *Control Systems Toolbox* contiene las funciones **rlocfind** y **rltool**. Estas permite diseñar en forma interactiva un controlador SISO en el lugar de raíces. La función **rlocfind** determina la ganancia que hace que los polos del sistema de lazo cerrado se encuentren en una ubicación especificada. Por ejemplo, suponiendo que tenemos la siguiente planta:

$$P(s) = \frac{s+7}{s^2+7s+10}$$

con polos en $s = -2$ y $s = -5$. Primeramente hacemos el gráfico del lugar de raíces y luego llamamos a la función **rlocfind**

```
» rlocus(P);
» [k,polos]=rlocfind(P);
```

La última instrucción espera que seleccionemos con el mouse la ubicación de los polos de lazo cerrado deseado, una vez que lo hagamos no devolverá en `k` la ganancia de realimentación y los polos obtenidos. Observemos que los polos de lazo cerrado deben estar sobre el lugar de raíces, si esto no es así la función devolverá la ganancia que ubique los polos lo más cerca posible de lo deseado. La otra función **rltool** ofrece un completo entorno gráfico para el diseño de controladores a partir del diagrama de lugar de raíces, el lector puede llamar a esta función simplemente con `rltool` e investigar como se hace el diseño en este caso.

Continuamos con las funciones para el diseño dentro del contexto de control moderno. El conjunto se puede dividir en las funciones para *ubicación de polos* y en las relacionadas con *control óptimo*. Dentro del primer grupo tenemos **acker** y **place**, ambas calculan la ganancia de realimentación de estados necesaria para que los autovalores de lazo cerrado se ubiquen en la posición fijada (la función **acker** sirve solo para sistemas SISO, en cambio **place** se puede utilizar también en sistemas MIMO. Se recomienda utilizar la última en todos los casos pues numéricamente es mejor). Consideremos la planta del ejemplo anterior, deseamos que el sistema de lazo cerrado tenga $\omega_n = 5rad/s$ y $\xi = 0{,}707$. Comenzamos calculando los polos deseados utilizando la función **roots**, la cual determina las raíces de un polinomio,

```
» polos = roots([1 2*0.707*5 25])
polos =
  -3.5350 + 3.5361i
  -3.5350 - 3.5361i
```

Luego, obtenemos las matrices de un modelo de estado de la planta:

```
» [A,B,C,D]=ssdata(P)
A =
   -7.0000   -2.5000
    4.0000         0
B =
     2
     0
C =
    0.5000    0.8750
D =
     0
```

Con estas podemos ahora aplicar la función **place**

```
» K = place(A,B,polos)
place: ndigits= 15
K =
    0.0350    1.8750
```

El vector `K` es el vector de ganancias, de modo que la matriz de lazo cerrado $A - KB$ tenga los autovalores deseados. Notemos que es posible utilizar esta función para obtener las ganancias de un observador de estados cambiando los argumentos:

```
» L = place(A',C',[-25;-30]).'
place: ndigits= 15
L =
  661.6000
 -323.2000
```

Finalmente, utilizando la función **reg** podemos obtener el controlador correspondiente a la realimentación de estados y al observador,

```
» S = ss(P);
» ksys = reg(S,K,L)

a =
                      x1           x2
           x1      -337.87      -585.15
           x2        165.6        282.8


b =
                      u1
           x1         661.6
           x2       -323.2


c =
                      x1           x2
           y1       -0.035       -1.875


d =
                      u1
           y1            0
```

```
I/O groups:
    Group name      I/O    Channel(s)
    Measurement      I         1
     Controls        O         1

Continuous-time model.
```

Observemos que se requiere primero convertir la planta `P` a modelo de estados. A modo de verificación podemos armar el sistema de lazo cerrado y calcular los autovalores,

```
» Scl = feedback(S,ksys,1);
» eig(Scl)
ans =
 -30.0000
  -3.5350 + 3.5361i
  -3.5350 - 3.5361i
 -25.0000
```

Por último, **K = lqr(A,B,Q,R)** calcula el vector de realimentación de estados que minimiza el siguiente índice de performance:

$$J = \int_0^\infty (x^T Q x + u^T R u) dt$$

donde el par $(A,B)$ debe ser controlable, $N$ y $Q$ definidas positivas.