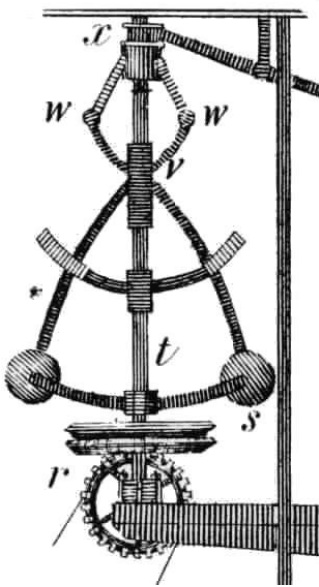


Introducción a MATLAB y SIMULINK para Control

Virginia Mazzone



Regulador centrífugo de Watt

Control Automático 1
<http://iaci.unq.edu.ar/caut1>
Automatización y Control Industrial
Universidad Nacional de Quilmes
Marzo 2002

Índice General

1	Introducción a MATLAB	1
1.1	Conversión de una función transferencia	1
1.2	Raíces de un polinomio	4
1.3	Desarrollo en fracciones simples	4
1.4	Función transferencia a lazo cerrado	6
1.5	Respuesta al impulso	7
1.6	Respuesta al escalón	8
1.7	Gráficos	9
2	Introducción a SIMULINK	14
2.1	Acceso y descripción	14
2.2	Función transferencia a lazo cerrado	16
2.3	Respuesta al Escalón	17
2.4	Utilización de parámetros ya definidos	19

1 Introducción a MATLAB

Este apunte es una introducción elemental a MATLAB, destinado a conocer y practicar algunas de las operaciones básicas con funciones de transferencia. Los comandos que utilizaremos son los que figuran en la Tabla 1. Para mayor información sobre un comando en particular puede ejecutarse `help topic` o simplemente `help 'comando'`, desde la ventana de comando de MATLAB.

1.1 Conversión de una función transferencia

Una función transferencia puede describirse en MATLAB utilizando dos vectores filas: uno para los coeficientes del numerador y otro para los coeficientes del denominador. A menudo se requiere para analizar o diseñar un sistema conocer la ubicación de sus polos y ceros; dicha información está contenida en la función transferencia del sistema. Cuando la función de transferencia está especificada como razón de polinomios, podemos conocer sus polos, ceros y ganancia, o viceversa. Los comandos que nos permiten ésto son: `tf2zp`, que de un cociente de polinomios nos devuelve los ceros, polos y una ganancia, y `zp2tf`, que de conocer los polos, ceros y la ganancia de un sistema nos da el numerador y denominador de su función de transferencia.

Ejemplo 1. Supongamos la función transferencia

$$G(s) = \frac{5s + 20}{s^2 + 4s + 20}$$

si sacamos el 5 factor común del numerador y factorizamos el denominador utilizando sus raíces, nos queda de la forma

$$G(s) = \frac{5(s + 4)}{(s + 2 - 4j)(s + 2 + 4j)}$$

Comando	Breve explicación
exp	Exponencial.
sin	Seno.
cos	Coseno.
sinh	Seno Hiperbólico.
cosh	Coseno Hiperbólico.
clf	Elimina la figura actual.
plot	Crea gráficas.
subplot	Crea múltiples gráficos en la misma figura.
hold	Mantiene la gráfica anterior.
title	Agrega título del gráfico.
xlabel	Agrega nombre del eje-X.
ylabel	Agrega nombre del eje-Y.
text	Agrega texto al gráfico.
print	Imprime el gráfico o lo guarda en un archivo
figure	Crea figuras (ventana para gráficos).
impulse	Respuesta al Impulso.
step	Respuesta al escalón unitario.
tf	Crea un modelo en función de transferencia.
zpk	Crea un modelo de cero-polo-ganancia.
ss2tf	Conversión de modelo en espacio de estados a función de transferencia.
tf2zp	Conversión de modelo en función de transferencia a polos y ceros.
ss2zp	Conversión de modelo en espacio de estados a polos y ceros.
zp2tf	Conversión de modelo en polos y ceros a función de transferencia.
tf2ss	Conversión de modelo en función de transferencia a espacio de estados.
zp2ss	Conversión de modelo en polos y ceros a espacio de estados.

Tabla 1: Comandos que utilizaremos

Para llevar a cabo lo mismo con MATLAB, debemos ingresar los polinomios numerador y denominador, en forma de vectores de la siguiente manera:

```
num=[ 5 20 ];
den=[ 1 4 20 ];
```

Observemos que para definir el vector lo hacemos colocando entre corchetes los coeficientes de cada término, ordenados de mayor orden al menor. Para separar las columnas del vector lo hacemos con un espacio, o también podríamos utilizar coma. El punto y coma final es para que el resultado de lo ejecutado por MATLAB no salga por pantalla.

Si ahora ingresamos:

```
[ z , p , k ] = tf2zp ( num , den )
```

Obtenemos:

```
z=-4
p=[ -2+4j -2-4j ]
k=5
```

Dado que toda función transferencia dada por un cociente de polinomios se puede escribir de la forma

$$G(s) = k \frac{\prod_{i=1}^m (s - z_i)}{\prod_{i=1}^n (s - p_i)} \quad \text{con } m \leq n,$$

podemos armar fácilmente nuestra función transferencia, haciendo

$$G(s) = \frac{5(s + 4)}{(s + 2 + 4j)(s + 2 - 4j)}.$$

Si queremos realizar el procedimiento inverso, necesitamos ingresar las variables k , z y p . Con la instrucción:

```
[ num , den ] = zp2tf ( z , p , k ) ;
```

obtenemos el numerador y denominador de la función transferencia:

```
num=[ 5 20 ]
den=[ 1 4 20 ]
```

El hecho de tener el numerador y el denominador de la función de transferencia en dos variables, no significa que MATLAB la identifique como tal. Para ello se utiliza el comando `tf`, que describe en una sola variable la transferencia dada por su numerador y al denominador. Lo utilizamos de la siguiente forma:

```
G=tf ( num , den ) ;
```

Si queremos que MATLAB arme la función transferencia como cociente de productos de los ceros y los polos, para ello utilizamos `zpk`, de la siguiente forma:

```
G=zpk ( z , p , k ) ;
```

1.2 Raíces de un polinomio

En el Ejemplo 1 vimos que el polinomio denominador de la función transferencia venía dado por: $s^2 + 4s + 20$, y pudimos hallar sus raíces dado que se trata de una ecuación de segundo orden.

En polinomios de orden superior, la tarea de encontrar sus raíces no siempre es tan fácil. Con la función de MATLAB `roots` podemos calcular las raíces de cualquier polinomio. Para ejecutar dicha función tenemos que ingresar el polinomio, como vector, recordando que los polinomios se ingresan en la primer componente el término de mayor orden y luego en forma descendente separados por coma o un espacio.

Ejemplo 2. Consideremos el siguiente polinomio:

$$P = s^4 + 4s^3 + 4s^2 + s + 20$$

Ingresamos el polinomio `p=[1 4 4 1 20]` y luego:

```
r=roots(p);
```

En lugar de hacer la operación en dos pasos, podemos hacerlo solo en uno; si tipeamos `r=roots([1 4 4 1 20])` obtenemos el mismo resultado. Las cuatro raíces del polinomio anterior que surgen de MATLAB son: $-2.6445 \pm 1.2595j$ y $0.6545 \pm 1.3742j$.

Si el caso es al revés, es decir, tengo las raíces y quiero conocer el polinomio, el comando `poly` es el que se utilizaremos. Siguiendo con el mismo ejemplo, supongamos que lo que tenemos son las raíces $p_{1,2} = -2.6445 \pm 1.2595j$ y $p_{3,4} = 0.6545 \pm 1.3742j$. Entonces el polinomio al que le corresponden esas raíces es:

```
P=poly([p1,p2,p3,p4]);
```

Notemos que el polinomio P que obtuvimos es *mónico*; si quisiéramos cualquier otro, deberíamos multiplicar a P por el coeficiente principal. Otra cosa a tener en cuenta es que siempre que pongamos una raíz compleja debemos poner su conjugada.

1.3 Desarrollo en fracciones simples

Cuando analizamos un sistema de control, por lo general disponemos de su función transferencia a lazo cerrado $G(s)$, donde $G(s) = \frac{Y(s)}{R(s)}$. Con lo que podemos escribir la salida en función de la transferencia y la entrada: $Y(s) = G(s) \times R(s)$.

Si deseáramos conocer la respuesta temporal $g(t)$ del sistema cuando lo excitamos con una señal de entrada $r(t)$, debemos calcular la transformada inversa de Laplace, es decir $g(t) = \mathcal{L}^{-1}\{Y(s)\} = \mathcal{L}^{-1}\{G(s) \times R(s)\}$. Como sabemos, es más sencillo de antitransformar cuando se trata de un cociente de polinomios, dado que si lo expresamos en fracciones simples podemos utilizar una tabla de transformadas de Laplace.

Ejemplo 3. Supongamos que tenemos la siguiente función transferencia:

$$G(s) = \frac{16s + 16}{(s + 2)(s + 4)} \quad \text{y que} \quad R(s) = \frac{1}{s}$$

Como las raíces del denominador son reales y distintas, el método de desarrollo en fracciones simples nos permite escribir a $G(s) \times R(s)$ de la siguiente manera:

$$\frac{16s + 16}{s(s + 2)(s + 4)} = \frac{A}{s} + \frac{B}{s + 2} + \frac{C}{s + 4}$$

Ahora podemos calcular $c(t)$ se la siguiente forma:

$$\begin{aligned} c(t) &= \mathcal{L}^{-1} \left\{ \frac{A}{s} + \frac{B}{s + 2} + \frac{C}{s + 4} \right\} \\ &= \mathcal{L}^{-1} \left\{ \frac{A}{s} \right\} + \mathcal{L}^{-1} \left\{ \frac{B}{s + 2} \right\} + \mathcal{L}^{-1} \left\{ \frac{C}{s + 4} \right\} \\ &= A + Be^{-2t} + Ce^{-4t} \end{aligned}$$

Para calcular los valores de A , B y C lo hacemos mediante la fórmula de residuos, dado que en este ejemplo los polos son de primer orden, resulta que

$$\text{Res}\{p\} = \lim_{s \rightarrow p} (s - p)F(s)$$

donde p es el polo para el cual se está calculado el residuo. Veamos como sería en este ejemplo:

$$\begin{aligned} A &= \lim_{s \rightarrow 0} (s) \frac{16s + 16}{s(s + 2)(s + 4)} = \frac{16(0) + 16}{(0 + 2)(0 + 4)} = 2 \\ B &= \lim_{s \rightarrow -2} (s + 2) \frac{16s + 16}{s(s + 2)(s + 4)} = \frac{16(-2) + 16}{(-2)(-2 + 4)} = 4 \\ C &= \lim_{s \rightarrow -4} (s + 4) \frac{16s + 16}{s(s + 2)(s + 4)} = \frac{16(-4) + 16}{(-4)(-4 + 2)} = -6 \end{aligned}$$

Con estos residuos, queda determinada la salida como: $c(t) = 2 + 4e^{-2t} - 6e^{-4t}$

En general, estos cálculos pueden tornarse muy complicados de realizar 'a mano'. Veamos como se simplifican utilizando la función MATLAB `residue`. Ingreseemos nuevamente los polinomios numerador y denominador de la misma forma como lo venimos haciendo hasta ahora. Ingreseemos ahora la sentencia:

```
[res,p]=residue(num,den);
```

Esta función nos devuelve dos parámetros vectoriales: en la variable `res` aparecen los residuos correspondientes a los polos que figuran en la variable `p`, es decir, el primer residuo corresponde al primer polo y así sucesivamente.

Si la función transferencia resulta ser *propia*, es decir que el grado del numerador es igual al del denominador, podemos añadir una parámetro más al argumento del lado izquierdo, que lo podemos llamar k . Veamos como sería esto mediante otro ejemplo:

Ejemplo 4. Supongamos que queremos hallar $f(t)$ siendo:

$$F(s) = \frac{2s^3 + 5s^2 + 3s + 6}{s^3 + 6s^2 + 11s + 6} \quad \Rightarrow \quad f(t) = \mathcal{L}^{-1}\{F(s)\}$$

Si aplicamos el comando:

```
[res,p,k]=residue(num,den);
```

y si armamos, como lo hicimos anteriormente, la función desarrollada en fracciones simples, el término independiente es el que aparece en la variable k . Por lo tanto $F(s) = \frac{-6}{s+3} + \frac{-4}{s+2} + \frac{3}{s+1} + 2$, de donde ahora calcular la $f(t)$ resulta muy sencillo.

Si ahora nuestro interés es el inverso, es decir que tenemos una función escrita en fracciones simples y quisiéramos obtener la función como cociente de polinomios, analíticamente deberíamos sacar común denominador y hacer todas las cuentas correspondientes. Esto resulta inmediato con el comando de MATLAB:

```
[num,den]=residue(res,p,k);
```

1.4 Función transferencia a lazo cerrado

Ejemplo 5. Supongamos que disponemos del sistema de la Figura 1 donde $G_1(s) = 0.4$; $G_2(s) = \frac{100}{s(s+2)}$; $H_2(s) = \frac{s}{s+20}$ y $H_1(s) = 1$; y pretendemos hallar la función transferencia a lazo cerrado $G(s) = \frac{Y(s)}{R(s)}$. Si aplicamos reducción de bloques, o resolviendo el diagrama de

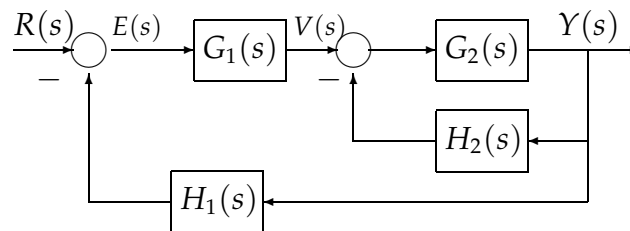


Figura 1: Diagrama de bloques

flujo y aplicando Mason, obtenemos:

$$G(s) = \frac{40s + 800}{s^3 + 22s^2 + 180s + 800}$$

En MATLAB la función transferencia a lazo cerrado se puede calcular de dos formas:

- Utilizando SIMULINK (lo veremos más adelante).
- Utilizando las funciones de MATLAB `series`, `parallel`, `feedback` y `cloop`.

Para calcular la función transferencia a lazo cerrado $G(s)$ sigamos los siguientes pasos:

1. Definimos los numeradores y denominadores de las funciones transferencia de cada bloque de la siguiente forma:

```
numg1=0.4;          deng1=1;
numg2=100;         deng2=[1 2 0];
numh2=[1 0];      denh2=[1 20];
```

2. Calculamos la función transferencia de $V(s)$ a $Y(s)$:

```
[numvc , denvc ]=feedback ( numg2 , deng2 , numh2 , denh2 , -1 ) ;
```

3. Ahora calculamos la función transferencia de $E(s)$ a $Y(s)$ con:

```
[numec , denec ]=series ( numg1 , deng1 , numvc , denvc ) ;
```

4. Por último calculamos el lazo cerrado:

```
[num , den]=cloop ( numec , denec , -1 ) ;
```

Lo que obtuvimos son los vectores numerador y denominador de la función transferencia por separado. Recordemos que para ingresarla como función de transferencia a MATLAB, debemos utilizar `tf`.

1.5 Respuesta al impulso

Ahora que ya sabemos como pasar de la respuesta temporal a Laplace, verifiquemos que la respuesta al impulso de la transformada de Laplace coincide con la respuesta temporal. Para ello utilizaremos el comando de MATLAB `impulse`.

Ejemplo 6. Supongamos que tenemos una función transferencia de la siguiente forma:

$$Y(s) = \frac{1}{(s+a)(s+b)}; \quad \text{donde } a = 1, b = 2$$

Si calculamos ahora la antitransformada, desarrollando en fracciones simples como en la sección 1.3, resulta que $y(t) = e^{-t} - e^{-2t}$. Ingreseemos los vectores numerador y denominador y luego ejecutemos el comando:

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

Veremos que este comando devuelve el gráfico de la Figura 2. Como podemos ver, solo nos muestra los primeros 6 segundos de la respuesta. Si quisiéramos que nos mostrara 12 segundos, debemos definir un vector de tiempo. Para ello ingresemos, por ejemplo,

```
t=0:0.1:12;
```

El vector t tendrá como primer elemento el 0 y como último al 12. Cada elemento estará a una distancia de 0.1 de su consecutivo. Si ahora introducimos este parámetro en el comando `impulse (num , den , t)`, el gráfico mostrará los primeros 12 segundos de la respuesta al impulso.

Notemos que este comando no fue asignado a ninguna variable; podríamos asignarle un vector, es decir `y=impulse (num , den , t)`, y así tendríamos los valores de la salida de la respuesta al impulso en dicho vector. Podríamos también graficar este vector con el comando `plot (t , y)`, comando que veremos en la sección 1.7.

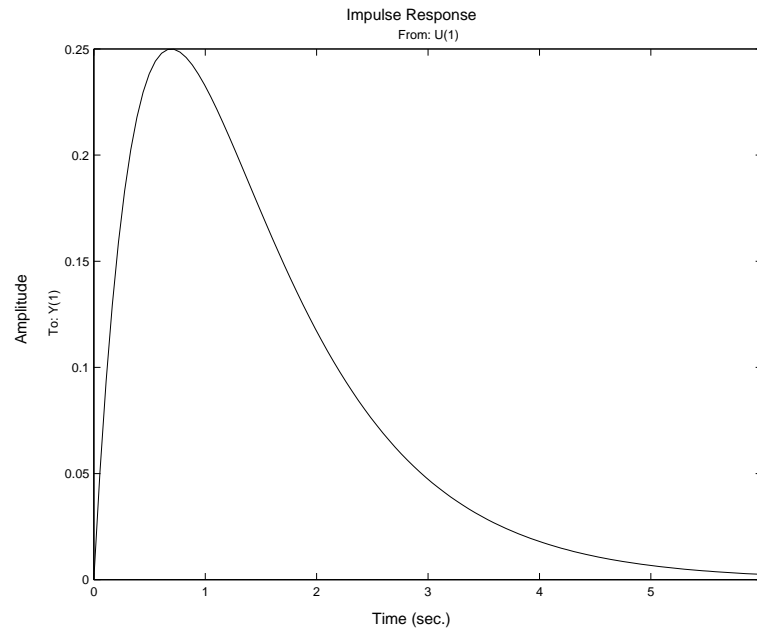


Figura 2: Respuesta al impulso

Dado que la función transferencia de un sistema lineal invariante en el tiempo se define como la transformada de Laplace de la respuesta al impulso cuando todas las condiciones iniciales son nulas, comparemos el resultado obtenido con el que resultaría si calculáramos la respuesta temporal. Para ello utilizaremos el mismo vector temporal t , y la instrucción $f = \exp(-t) + \exp(-2*t)$. Ahora podemos comparar los valores obtenidos desde la respuesta al impulso con los obtenidos desde la respuesta temporal (por ejemplo, restándolos).

1.6 Respuesta al escalón

De la misma forma que en la sección anterior, podríamos querer graficar la respuesta al escalón unitario. MATLAB posee un comando, llamado `step`, para calcular la salida temporal cuando la entrada se trata de un escalón unitario. Lo único que necesita este comando es el numerador y el denominador de la función transferencia.

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

Si utilizamos el comando sin asignarle la salida a ninguna variable, MATLAB abre una ventana gráfica mostrando el gráfico de la salida a la excitación escalón unitario, de la misma forma que antes. Sin embargo, al igual que vimos en el comando `impz`, cuando éste es asignado a una variable, los valores obtenidos se guardan en el vector y .

Ejemplo 7. Calculemos la respuesta al escalón unitario de la función transferencia:

$$G(s) = \frac{Y(s)}{R(s)} = \frac{4}{s^2 + 0.8s + 4}$$

Si ingresamos el comando `step(num,den)`, veremos un gráfico similar al que podemos observar en la Figura 3.

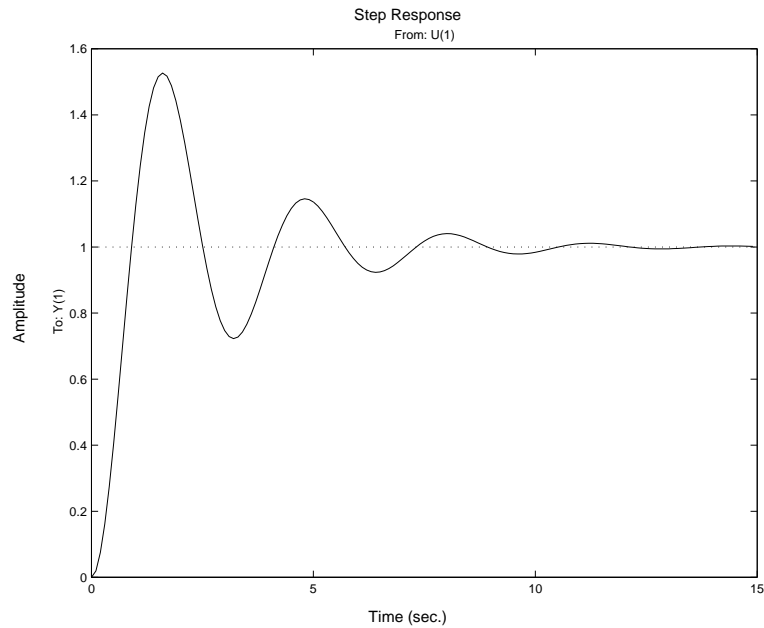


Figura 3: Respuesta al escalón unitario

Si ahora queremos la respuesta a una entrada rampa unitaria, MATLAB no posee ningún comando que lo resuelva. Por lo que veremos cómo con el comando `step` podemos obtener una rampa. Si seguimos con el ejemplo anterior y excitamos al sistema con $r(t) = t$, es decir que $R(s) = \frac{1}{s^2}$, tenemos lo siguiente:

$$Y(s) = \left(\frac{4}{s^2 + 0.8s + 4} \right) \frac{1}{s^2} \Rightarrow Y(s) = \left(\frac{4}{s^3 + 0.8s^2 + 4s} \right) \frac{1}{s}$$

Por lo que utilizando como denominador de la función transferencia al polinomio $s^3 + 0.8s^2 + 4s$, es decir `den=[1 0.8 4 0]`, y calculando la respuesta al escalón unitario con `step(num,den)`, obtenemos la respuesta a la rampa unitaria que se muestra en la Figura 4.

1.7 Gráficos

Como vimos en secciones anteriores los comandos `step` e `impz` grafican las respuestas al escalón y al impulso respectivamente, pero ahora vamos a introducir algo más general. Para graficar cualquier función en general utilizaremos el comando `plot`, que sólo necesita definir el vector a graficar en la forma básica

```
plot(vector);
```

Se obtiene un gráfico donde el eje de abscisas será la posición del elemento del vector y la ordenada el valor que tiene el vector en dicha posición. En el ejemplo 6, guardamos en el vector y los valores de la salida de la respuesta al impulso. Si ahora ingresamos `plot(y)`, en lugar de tener segundos en el eje de abscisas tendremos la cantidad de elementos de ese vector. Si ingresamos `plot(t,y)`, ahora el eje de abscisas corresponderá al vector temporal ya definido e irá desde $t = 0$ a $t = 12$, que es como lo teníamos definido. Grafiquemos

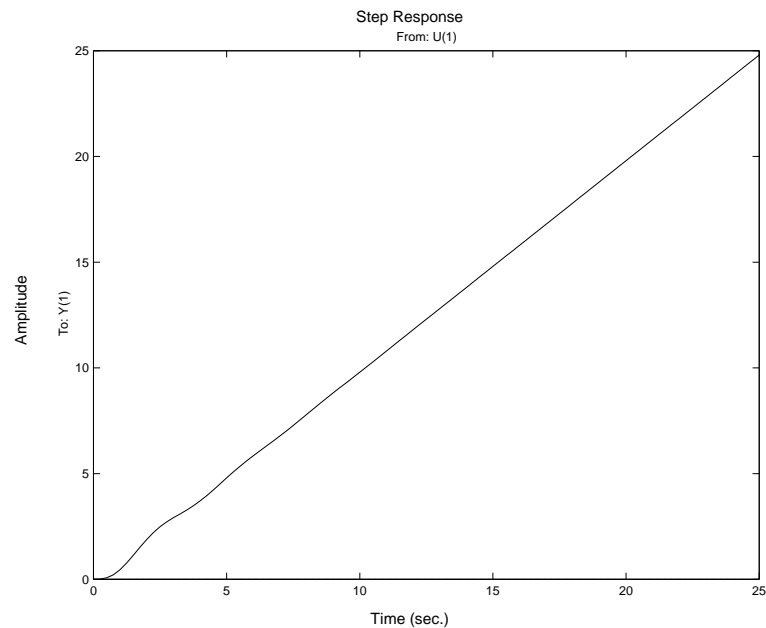


Figura 4: Respuesta a la rampa unitaria

entonces los valores guardados en el vector f . Estos valores corresponden a la respuesta temporal, por lo que el gráfico deberá ser el mismo. Si ingresamos `plot(t,f)`, obtendremos el gráfico de la Figura 5

Como podemos ver, no hay casi diferencias con la figura 2, excepto por el título y los nombres de los ejes que el comando `impz(num,den)` pone automáticamente. Veamos que no es difícil si se lo queremos agregar a un gráfico, para ello utilizaremos las sentencias `title`, `xlabel` y `ylabel`. Estos comandos se utilizan luego de ingresar el comando `plot`, ya que tanto el título, como los nombres de los ejes, se escribirán en la figura que se encuentre abierta:

```
title('Respuesta al Impulso');
xlabel('Tiempo(seg.)');
ylabel('Salida c(t)');
```

Notemos que el texto que queremos que aparezca está escrito entre comillas simples. Los comandos anteriores, pueden ser también utilizados con `step` y `impz`, aunque cuando son utilizados en estos comandos, el título y el nombre de los ejes que trae la función por defecto también aparecen. Otros comandos que pueden ser útiles a la hora de trabajar con gráficos son `grid` y `text`, que se utilizan para agregar una grilla y agregar texto respectivamente. El comando `text` se utiliza de la misma forma que `title`, es decir, el texto que aparecerá será el que se encuentra escrito entre las comillas simples, pero antes debemos ingresar las coordenadas (x,y) donde queremos que aparezca el texto. El comando `grid`, se usa sin parámetros. Veamos el siguiente ejemplo:

Ejemplo 8. Supongamos que tenemos la función transferencia de la Figura 1, que ya la calculamos con MATLAB en el Ejemplo 2.2. Abramos un archivo nuevo e ingresemos lo siguiente:

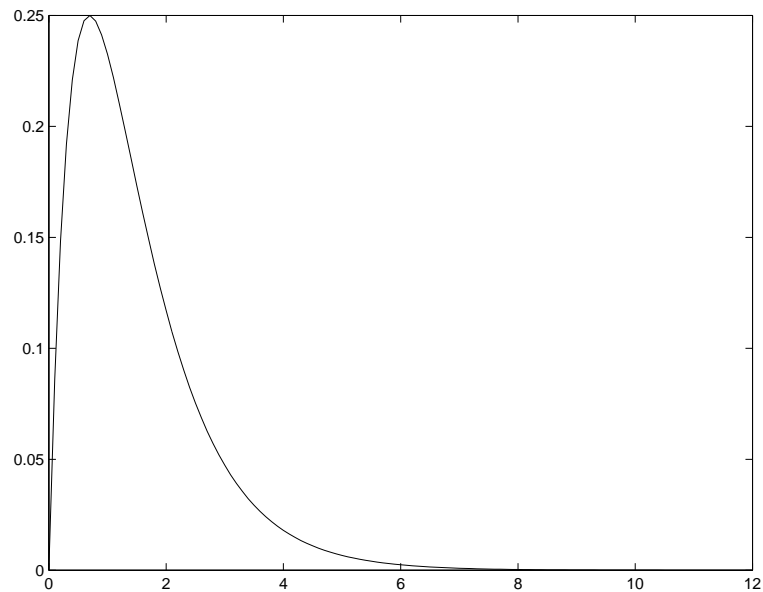


Figura 5: Respuesta temporal del ejemplo 6

```

num=[40 800];
den[1 22 180 800];
t=0:0.01:2;
y=step(num,den,t);
plot(t,y);
title('Respuesta al escalon unitario');
xlabel('Tiempo (seg.)');
ylabel('Salida del sistema');
text(0.5,1.1,'maximo valor');
grid;

```

Si ejecutamos el programa vamos a obtener el gráfico de la Figura 6.

Supongamos ahora que queremos graficar en la misma figura dos o más gráficos para poder compararlas. Esto es posible utilizando el comando `hold on` - `hold off`, que mantiene la figura y superpone el siguiente gráfico sobre la misma figura, como veremos en el siguiente ejemplo.

Ejemplo 9. Supongamos que queremos graficar tres sinusoides con frecuencias diferentes, ingresemos en un archivo nuevo:

```

t=0:pi/20:2*pi;
y1=sin(t);
y2=sin(t-pi/2);
y3=sin(t-pi);
plot(t,y1);
hold on;
plot(t,y2);
plot(t,y3);
hold off;

```

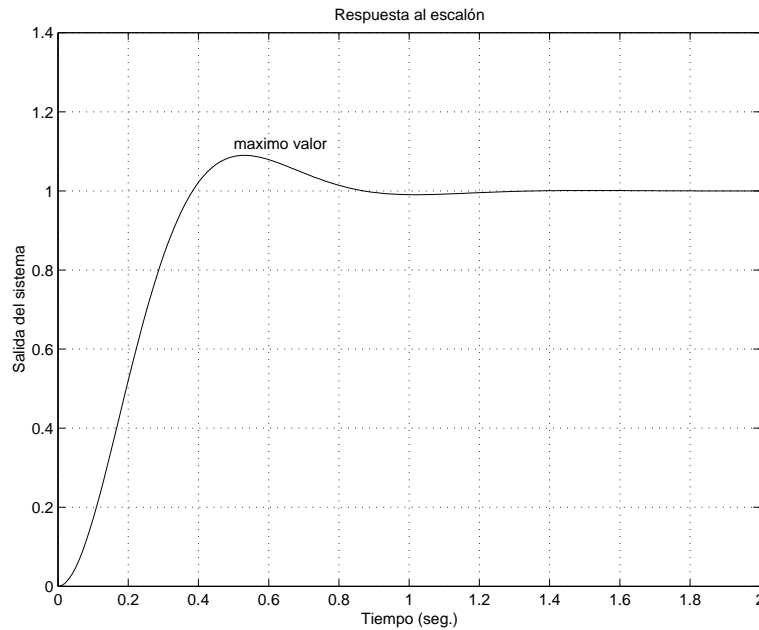


Figura 6: Respuesta al escalón del ejemplo

Luego de ejecutar estas líneas veremos que en la figura que resulta, aparecen 3 gráficas en el mismo color, e imposible de identificar cual corresponde a cada una porque ambas se encuentran graficadas con el mismo tipo de línea y el mismo color. Para ello veamos un parámetro del tipo *string* que podemos agregar al comando `plot` para especificaciones del estilo del gráfico. Los parámetros que figuran en la Tabla 2 son para elegir el color de la línea, los que se encuentran en la Tabla 3 son para elegir el estilo de la línea y los que se encuentran en la Tabla 4 son para elegir el tipo de marca que aparecerá sobre los puntos del vector graficado.

Espec.	Color
r	rojo
b	azul (por defecto)
w	blanco
g	verde
c	cian
m	magneto
y	amarillos
k	negro

Tabla 2: Especificadores de color

Ahora especifiquemos cada uno de los `plot` con un estilo diferente, por ejemplo, en lugar del comando `plot(t,y)` escribamos:

```
plot(t,y1,'-rx');
plot(t,y2,'--mo');
plot(t,y3,':bs');
```

Espec.	Estilo de línea
-	línea sólida (por defecto)
-	línea rayada
:	línea punteada
-.	línea punto-rama

Tabla 3: Especificadores de línea

Espec.	Estilo de marca
+	signo más
o	círculo
.	punto
*	asterisco
s	cuadrado
d	diamante
x	cruz
p	estrella de 5 puntas
h	estrella de 6 puntas

Tabla 4: Especificadores de marca

Si corremos nuevamente el archivo veremos que hay diferencia entre una función y la otra, pero seguimos sin saber cuál corresponde a qué función. Para ello utilizemos el comando `legend`, que pone la leyenda que queramos a cada gráfico. Es decir, escribamos como última línea:

```
legend('sin(t)', 'sin(t-pi/2)', 'sin(t-pi)');
```

Ahora si observamos el gráfico debería ser como el de la Figura 7.

También podríamos querer cada gráfico en una figura diferente. Para ello debemos ejecutar el comando `figure(2)` antes de graficar por segunda vez y `figure(3)` antes del último gráfico. Estas sentencias se usan sin el comando de `hold on - hold off` y lo que hacen es abrir una nueva figura para cada gráfico. Otra opción válida para MATLAB, por ejemplo, es que las tres funciones aparezcan en una sola figura pero las tres graficadas en forma independiente. Para ello utilizemos `subplot(m,n,p)`, que dividirá a la figura en m filas y n columnas, pero crea una figura en la posición p . Ingreseemos lo siguiente para ver como funciona:

```
clf;           %borra el grafico actual
subplot(3,1,1);
plot(t,y1,'.-r');
title('sin(t)');
subplot(3,1,2);
plot(t,y2,'--m');
title('sin(t-pi/2)');
subplot(3,1,3);
plot(t,y3,':b');
```

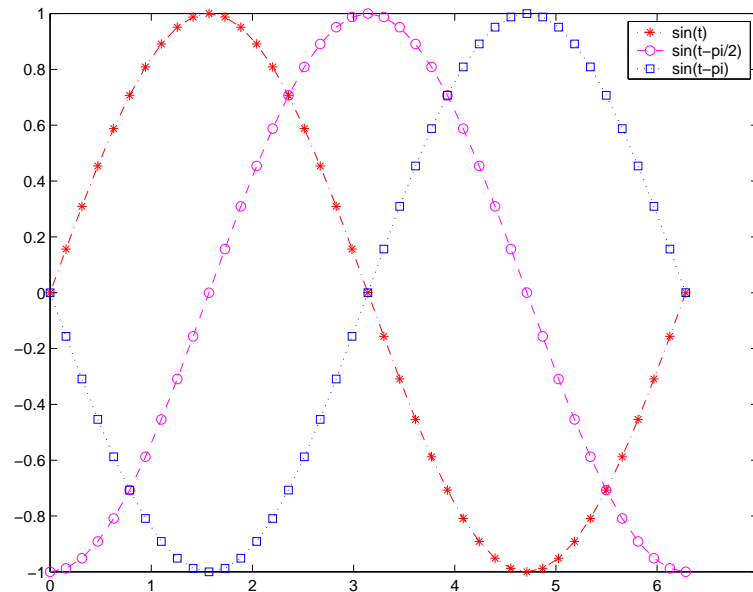


Figura 7: Tres gráficos en una misma figura

```
title('sin(t-pi)');
```

Notemos que con el símbolo %, comentamos texto dentro del archivo. Si ejecutamos nuevamente el programa, obtenemos lo que se observa en la Figura 8

Con respecto a gráficos, MATLAB posee muchas otras opciones, como graficar en escala logarítmica, con `loglog`, `semilogx` y `semilogy`, gráficos en tres dimensiones, `plot3`, gráficos de barras, `bar`, etc. También permite con el comando `print`, guardar el gráfico en un archivo de extensión, por ejemplo `postscript` o `.jpg`, o también lo podemos imprimir con el mismo comando indicando el nombre de la impresora.

No nos olvidemos que MATLAB cuenta con una ayuda a la cual podemos recurrir en caso de no recordar como se utiliza un comando. Si investigamos un poco el `help`, podemos encontrar funciones que resuelven muchas otras cosas interesantes. Invito a que se metan a conocerlas, como así también a que conozcan las distintas demostraciones que pueden encontrar si tipean: `demo`.

2 Introducción a SIMULINK

Hasta ahora vimos que MATLAB dispone de un entorno de programación con líneas de órdenes, ahora veremos como se puede suplementar utilizando un interfaz de usuario gráfica llamada SIMULINK. Este entorno nos permite describir gráficamente un sistema dibujando su diagrama en bloques, que resulta muy conveniente para la simulación y análisis de sistemas dinámicos.

2.1 Acceso y descripción

Para acceder a SIMULINK, desde la ventana de comandos de MATLAB, tenemos varias opciones: una es escribiendo el comando `simulink`, de esta forma se abrirá solo una ventana

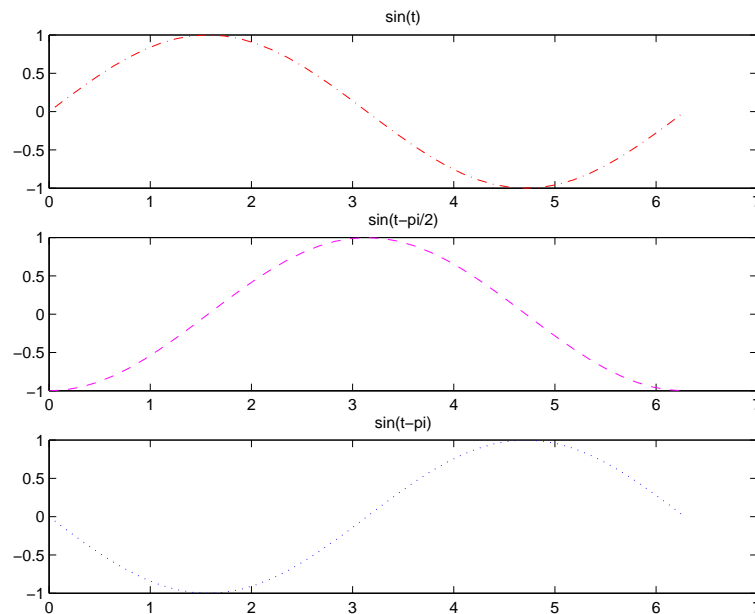


Figura 8: Tres gráficos diferentes en la misma figura

con las librerías disponibles; otra es desde la barra de menú *File* elegir la opción *New Model*, de esta forma se abren no sólo las librerías sino también el entorno donde vamos a trabajar; por último, existe un botón de acceso directo a las librerías tanto en el entorno de trabajo de MATLAB como en el de SIMULINK.

Una vez abiertas las librerías, lo que encontraremos depende de la versión de MATLAB que se encuentre instalada. Nos vamos a referir a la versión 5.3. Dentro de la librería *Simulink* se encuentran los elementos que vamos a utilizar organizados en sublibrerías de acuerdo con su comportamiento. Las sublibrerías que aparecen son:

- Continuous (Bloques para sistemas en tiempo continuo)
- Discrete: (Bloques para sistemas en tiempo discretos)
- Functions & Tables
- Math (Sumadores, Ganancias matriciales o constantes, etc.)
- Nonlinear
- Signals & Systems(multiplexores, demultiplexores, etc.)
- Sinks (Graficadores, etc.)
- Sources (Varias fuentes de entradas)

Con un doble click sobre la librería podemos visualizar los elementos que posee. Por ejemplo si ingresamos a *Continuous*, entre los elementos disponibles utilizaremos los siguientes:

Derivative: bloque derivador, es decir $\frac{du}{dt}$.

Integrator: bloque integrador, función transferencia $\frac{1}{s}$.

State-Space: bloque para expresar al sistema en modelo de estados.

Transfer Fnc: bloque para expresar al sistema como cociente de polinomios.

Zero-pole: bloque para expresar al sistema con ceros, polos y una ganancia.

2.2 Función transferencia a lazo cerrado

En la Sección 2.2 vimos como podemos calcular la función transferencia a lazo cerrado desde la ventana de comandos. Tomemos el mismo ejemplo para ver como lo hacemos con SIMULINK, ingresemos el diagrama en bloques como se puede ver en la Figura 9.

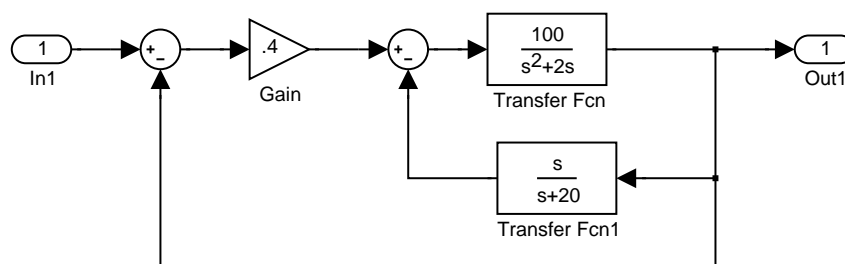


Figura 9: Diagrama en bloques con SIMULINK

Para poder implementar dicho gráfico se procedió de la siguiente forma:

1. Para insertar un bloque de función transferencia, ya sabemos que se encuentra en *Continuous*, lo tomamos y lo arrastramos hasta la ventana de trabajo de SIMULINK. Si hacemos doble click sobre el bloque se despliega una ventana de propiedades del bloque, donde tenemos que ingresar el numerador y el denominador de la misma forma que lo hacemos desde el entorno de trabajo de MATLAB, es decir entre corchetes y separado por espacios. Si en lugar de seleccionar el bloque de función transferencia elegimos el bloque de polos y ceros, los parámetros a definir serán los polos, los ceros y la ganancia.
2. Para insertar otro bloque igual no es necesario realizar el ítem anterior nuevamente, podemos seleccionar el bloque anterior, haciendo un click con el botón derecho del mouse, copiar el bloque y pegarlo donde queramos. Esto mismo se puede hacer simplemente arrastrando el objeto seleccionado con el botón derecho del mouse. De la misma forma que antes, ingresamos los parámetros de esta función de transferencia. Para girar el bloque, para que quede mejor orientado para hacer la realimentación, tenemos que seleccionar el objeto, ir a *Format* de la barra de menú, y seleccionar *Flip Block* o simplemente con las teclas `ctr-f`.
3. El bloque de ganancia lo encontramos en *Math*, lo mismo que los sumadores. Para ingresarlos a la figura procedemos de la misma manera, arrastrando el objeto hasta donde queremos ubicarlo. El sumador, por defecto, viene con dos entradas sumadas, si hacemos doble click sobre él, podemos no solo cambiar el signo que queramos sino también agregarle las entradas que queramos, en este caso solo modificamos ++ por

+-. Otra propiedad que podemos modificar es la forma del icono. Notemos que cuando hacemos un doble click en cualquier objeto, encontramos una pequeña ayuda en la parte superior de la ventana, ésto nos es útil para saber con que datos hay que completar cada campo. Para el bloque de la ganancia, solo ingresamos el valor que corresponde 0.4.

4. Para unir los bloques solo tenemos que hacer un click en la salida de un bloque y arrastra el mouse hasta la entrada de otro, cuando soltamos, si la conexión está bien hecha, marcará una flecha negra, en caso de estar uniendo con un nodo, deberemos ver un cuadradito negro. Para borrar cualquier elemento, simplemente lo seleccionamos y con la tecla DEL se elimina.
5. Por último nos falta solo indicar cuál es la entrada y cual es la salida, esto lo hacemos para poder sacar la transferencia a lazo cerrado, de otra forma no lo pondríamos. La necesidad de marcar la entrada y la salida es para que MATLAB sepa desde donde hasta donde vamos a querer la transferencia. Estos bloques los encontramos en *Signals & Systems*, se llaman "In1" e "Out1".
6. Salvemos el archivo, por ejemplo con el nombre "FuncTrans", si ingresamos desde la ventana de comando de MATLAB las sentencias

```
[A,B,C,D]=linmod('Functtrans');
[num,den]=ss2tf(A,B,C,D);
```

obtenemos la función transferencia. La primer orden produce un modelo de estado del sistema de bloques, tomando la entrada y la salida que seleccionamos; y la segunda sentencia convierte ese modelo de estados en el numerador y el denominador de la función transferencia. En este caso el resultado es

```
num=[40 800]
den=[1 22 180 800]
```

que si los comparamos con los obtenidos antes son idénticos.

2.3 Respuesta al Escalón

Siguiendo con el sistema de la Figura 9, nos interesa saber ahora cómo responde a una entrada escalón unitario. Como tenemos la función transferencia a lazo cerrado, podríamos utilizar el comando `step(num,den)` desde la ventana de comandos para obtener la salida. Pero veamos cómo lo podemos hacer desde SIMULINK. Para ello cambiemos el bloque de entrada por un bloque de entrada escalón, que lo encontramos en la librería *Sources* bajo el nombre "Step". A dicho bloque podemos modificarle algunos parámetros como el tiempo en que se realizará el escalón, el valor inicial y final de escalón y en caso de que lo necesitamos discreto, el tiempo de muestreo. Para nuestro ejemplo, elegimos como valor inicial 0, valor final 1 y tiempo de realización del escalón 0seg.. Para poder visualizar la salida, debemos conectar a la salida un osciloscopio. Este bloque lo encontramos en *Sinks* bajo el nombre "Scope". Luego de agregados estos bloques, el sistema resultante es el que observamos en la Figura 10.

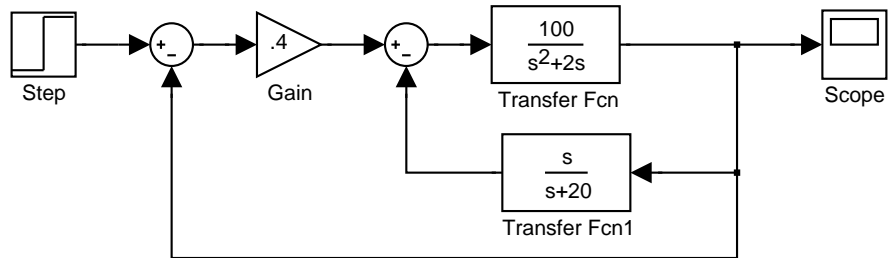


Figura 10: Sistema excitado con un escalón

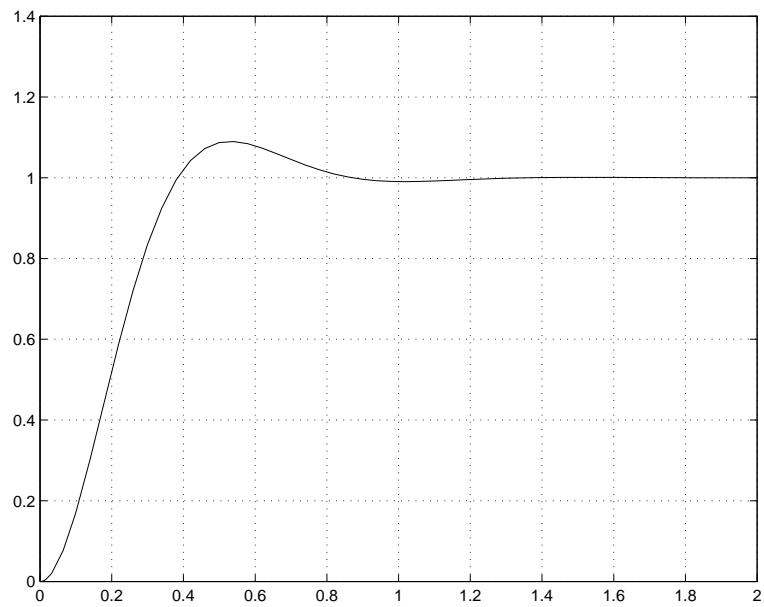


Figura 11: Señal que muestra el osciloscopio

Si hacemos doble click sobre el bloque "Scope", veremos la salida del sistema como en la Figura 11. Este bloque también posee algunas propiedades que podemos modificar, entre ellas están los valores de los ejes, el título y la escala. Cuenta también con *Zoom* para visualizar alguna zona en detalle. Otra propiedad es la posibilidad de asignarle los datos que posee a una variable. Luego desde la ventana de comandos podemos visualizar los valores o graficarlos dado que SIMULINK guarda también en una variable el vector temporal que utiliza en la simulación, dicha variable se llama *tout*.

Si en lugar de la respuesta al escalón unitario queremos la respuesta al impulso, dado que SIMULINK no posee un bloque generador de impulsos, debemos generarlo nosotros como resta de dos escalones.

2.4 Utilización de parámetros ya definidos

SIMULINK nos permite utilizar variables definidas ya sea en la ventana de comando de MATLAB, como también en archivos del editor. Para ello debemos definir las variables con anticipación y luego utilizarlas dentro de los bloques con el mismo nombre. De esta forma, SIMULINK identifica el valor de dicho parámetro y es el que utiliza en los cálculos. Esto resulta apropiado cuando queremos utilizar un mismo diseño para distintos valores de parámetros, o nos permitirá utilizar el mismo sistema cada vez que nos encontremos con problemas similares.