Introducción a Matlab y Simulink

Regulación Automática
Ingeniero en Electrónica. Curso 2006/2007.
Javier Aracil y Fabio Gómez–Estern

1. Introducción

Matlab es un programa de gran aceptación en ingeniería destinado realizar cálculos técnicos científicos y de propósito general. En él se integran operaciones de cálculo, visualización y programación, donde la interacción con el usuario emplea una notación matemática clásica.

Los **usos y aplicaciones** típicos de Matlab son:

- Matemáticas y cálculo.
- Desarrollo de algoritmos.
- Adquisición de datos.
- Modelado, simulación y prototipado.
- Análisis y procesado de datos.
- Gráficos científicos y de ingeniería.
- Desarrollo de aplicaciones.

El tipo básico de variable con el que trabaja Matlab es una **matriz** que no requiere ser dimensionada previamente en la declaración. Una de las características más interesantes consiste en que el álgebra vectorial y matricial se expresa con la misma sintaxis que las operaciones aritméticas escalares. Por ejemplo, en lenguaje C, para realizar la suma de dos variables enteras o reales `b` y `c`, escribiremos:

```
a=b+c;
```

Mientras que en Matlab, emplearemos la misma sentencia tanto si `b` y `c` son enteros, reales, vectores o matrices.

2. Componentes de Matlab

Matlab consta de cinco partes fundamentales:

1. **Entorno de desarrollo.** Se trata de un conjunto de utilidades que permiten el uso de funciones Matlab y ficheros en general. Muchas de estas utilidades son interfaces gráficas de usuario. Incluye el espacio de trabajo Matlab y la ventana de comandos.
2. **La librería de funciones matemáticas Matlab.** Se trata de un amplio conjunto de algoritmos de cálculo, comprendiendo las funciones más elementales como la suma, senos y cosenos o la aritmética compleja, hasta funciones más sofisticadas como la inversión de matrices, el cálculo de autovalores, funciones de Bessel y transformadas rápidas de Fourier.
3. **Gráficos.** Matlab dispone de un conjunto de utilidades destinadas a visualizar vectores y matrices en forma de gráficos. Existe una gran cantidad de posibilidades para ajustar el aspecto de los gráficos, destacando la visualización tridimensional con opciones de iluminación y sombreado, y la posibilidad de crear animaciones.
4. **El interfaz de aplicación de Matlab (API).** Consiste en una librería que permite escribir programas ejecutables independientes en C y otros lenguajes, accediendo, mediante DLLs, a las utilidades de cálculo matricial de Matlab.

De estos cuatro puntos, en este capítulo trataremos, de forma somera, los dos primeros.

Los ejemplos que se presentan en este texto, se han desarrollado para la versión de Matlab 7.0. ellos no impide que puedan funcionar con otras versiones del programa. Concretamente, para la versión 6.5 y posteriores está prácticamente garantizado el funcionamiento.

Sin embargo, hay que señalar que algunos complementos de Matlab no aparecen incluidos en la instalación básica del mismo, por tanto un programa que funciona en un ordenador con la versión 7.0 instalada, puede fallar en otro ordenador con la misma versión.

La gestión de complementos de Matlab se realiza mediante lo que se denominan toolboxes (paquetes de herramientas). Un **Toolbox** de Matlab es un conjunto de funciones y algoritmos de cálculo especializados en un área de conocimiento: finanzas, tratamiento de señales, teoría de sistemas, etc. Para el desarrollo del curso es necesario

tener instalado, aparte del sistema básico de Matlab, el denominado **Control System Toolbox**.

3. Simulink

Simulink es una aplicación que permite construir y simular modelos de sistemas físicos y sistemas de control mediante diagramas de bloques. El comportamiento de dichos sistemas se define mediante funciones de transferencia, operaciones matemáticas, elementos de Matlab y señales predefinidas de todo tipo.

Simulink dispone de una serie de utilidades que facilitan la visualización, análisis y guardado de los resultados de simulación. Simulink se emplea profusamente en ingeniería de control.

En el presente curso trabajaremos con la versión 6.0, que viene incluida en el paquete de Matlab 7.0. Su instalación es opcional, por tanto debemos seleccionar la opción correspondiente al instalar el programa

4. El entorno de trabajo de Matlab

4.1. Ayuda en línea

Si se ha seleccionado la la opción correspondiente en la instalación de Matlab, podremos acceder a la ayuda en línea en todo momento pulsando la tecla F1. Dicha documentación está organizada con un índice en forma de árbol y mediante herramientas de navegación como hipervínculos. Es sumamente recomendable su uso, tanto a modo de introducción como de referencia para temas específicos. Si se desea conocer la documentación específica asociada a un comando de Matlab, entonces se tecleará

```
>> doc nombre_comando
```

en la línea de comandos de Matlab.

4.2. Organización de ventanas

La figura 1 muestra la organización por defecto de ventanas que nos encontramos cuando arrancamos Matlab por primera vez. Las ventanas que en ella aparecen, de arriba a abajo son: en la parte izquierda, la estructura del directorio donde nos encontramos, y debajo de ella la historia de los comandos que se han tecleado en el pasado; en la mitad derecha nos encontramos, arriba, la ventana de edición de programas Matlab (que se escriben en un lenguaje propio de Matlab y se almacenan en ficheros **.m**), y debajo la línea de comandos, donde se sitúa el cursor para teclear comandos de Matlab.

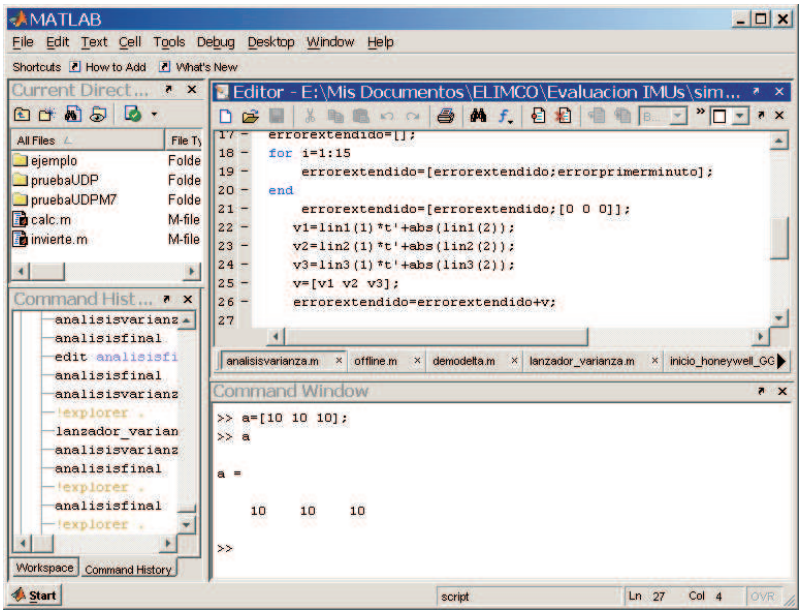

Figura 1: Entorno de trabajo Matlab.

Inicialmente trabajaremos con la línea de comandos de Matlab.

4.3. Operaciones básicas en línea de comandos

Como se ha dicho previamente, en Matlab todos los objetos son matrices. Un escalar no es más que una matriz 1×1. En la línea de comandos, podemos asignar un nombre simbólico para identificar una matriz.

```
>> a=[10; 20; -15];  % Asignacion
```

Esto es una asignación de un vector de columna que llevará el nombre a. A su derecha aparece un comentario, que tiene su utilidad cuando redactemos programas en

Matlab.

Los objetos pueden crearse en cualquier momento. Para ello basta con asignarles un valor mediante una asignación, como en el ejemplo previo. Los identificadores empleados para designar matrices son de libre elección, con la salvedad de que no pueden comenzar por un número, ni contener espacios en blanco.

Una vez creado un objeto de Matlab, éste pasa a formar parte del **espacio de trabajo** ocupando una porción la memoria. Por tanto, a veces, tras horas de trabajo con Matlab, necesitaremos eliminar los objetos que ya no se utilicen. Para ello se emplea el comando `clear`.

```
>> clear a;  % Borra a de la memoria
>> clear;  %  Borra todos los objetos del espacio de trabajo
```

En las sentencias previas, aparece el signo '`;`' al final de cada comando. Este símbolo sirve para separar unos comandos de otros. Por ejemplo, cuando escribimos varios comandos en una sola línea, estos deben aparecer separados por punto y coma. Además, si escribimos un comando aislado (sin '`;`') y pulsamos ENTER, Matlab proporcionará siempre una salida en respuesta al comando:

```
>> a=[10;20;-15]
a =
    10
    20
   -15
```

Sin embargo, si escribimos el comando seguido de '`;`', no se mostrará en pantalla la respuesta. Cuando los comandos forman parte de un programa es conveniente emplear '`;`' para evitar desbordar la pantalla con información innecesaria.

4.4. Sintaxis de vectores y matrices

Las matrices (y vectores como caso particular de las mismas) se expresan en Matlab empleando corchetes (`[ ]`); separando las **filas** por espacios o comas (`,`) y las columnas por ';'. De este modo, se puede crear un objeto matriz del siguiente modo:

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

```
mat =
     1     2     3
     4     5     6
     7     8     9
```

Cuando se trata de un escalar, podemos prescindir de los corchetes

```
>> rad=3.1415;
```

Los elementos de las matrices pueden ser reales o complejos. En este último caso se emplea la letra `i` para representar el valor $\sqrt{-1}$. Como ejemplo crearemos el vector fila $v = [2 + 3i,\ -5i,\ \ 3]$.

```
>> v=[2+3i, -5i, 3]
v =
   2.0000 + 3.0000i        0 - 5.0000i   3.0000
```

El acceso a elementos de una matriz previamente definida puede realizarse especificando la fila y columna del elemento que nos interesa entre paréntesis

```
>> mat(2,3)
ans =
     6
```

Además, dentro de estos paréntesis podemos indicar variables u operaciones más complejas, lo que da una gran potencia al desarrollo de algoritmos.

Una vez definidos los objetos con sus identificadores, podemos realizar operaciones aritméticas entre ello con total simplicidad. Para las operaciones vectoriales y matriciales, Matlab verificará la coherencia de las dimensiones de los operandos y si no hay producirá error producirá un resultado con las dimensiones adecuadas.

```
>> v1=[2+3i, -5i, 3];
>> v2=[0, 1, 7];
>> v3=v1+2*v2+[1, 1, 1]
v3=
  3.0000 + 3.0000i   3.0000 - 5.0000i  18.0000
```

4.5. Operaciones básicas con Matlab

La siguiente tabla ilustra las básicas aritméticas y lógicas que podemos realizar con Matlab.

Expresión en Matlab	Operación
+	Suma aritmética
-	Resta aritmética o cambio de signo
*	Multiplicación aritmética
/	División
^	Elevar un número a una potencia
<	Relación "menor que"
>	Relación "mayor que"
<=	Relación "menor o igual que"
>=	Relación "mayor o igual que"
==	Relación "igual que"
~=	Relación "distinto que"
&	producto lógico (operación ‘‘y’’)
|	suma lógica (operación .º")
~	negación (operación "no")

Cuadro 1: Operaciones aritméticas y lógicas de en Matlab.

Todas estas operaciones se aplican indistintamente a escalares, vectores y matrices, y se pueden emplear de dos modos diferentes: en primer lugar, Matlab funciona directamente como una calculadora, para lo cual tecleamos expresiones en línea de comandos para obtener inmediatamente el resultado de las mismas:

```
>> 12*24.8
ans =
  297.6000
```

Así mismo se pueden emplear las operaciones dentro de otras expresiones más amplias, logrando así escribir expresiones matemáticas de cualquier complejidad.

```
>> x1=-b+sqrt(b^2-4*a*c)/(2*a);
```

4.6. Funciones en Matlab

Buena parte de las operaciones que se realizan con Matlab, son llamadas a funciones. Las funciones procesan información, por lo que poseen datos de entrada y de salida,

que pueden ser matriciales. Los datos de entrada se especifican entre paréntesis, y si son varios separados por comas. Por ejemplo, la siguiente función calcula la raíz cuadrada de su **único** valor de entrada, que es el vector fila $[4, 2]$.

```
>> sqrt([4 2])
ans =
    2.0000    1.4142
```

Las funciones son programas escritos por el usuario o incorporados en el paquete básico de Matlab. Entre estas últimas destacan las siguientes funciones:

Nombre	**Función**
`sin`	Seno
`sinh`	Seno hiperbólico
`cos`	Coseno
`cosh`	Coseno hiperbólico
`tan`	Tangente
`tanh`	Tangente hiperbólica
`cot`	Cotangente
`coth`	Cotangente hiperbólica
`sec`	Secante
`sech`	Secante hiperbólica
`csc`	Cosecante
`csch`	Cosecante hiperbólica
`asin`	Arcoseno (inversa del seno)
`asinh`	Inversa del seno hiperbólico
`acos`	Arcocoseno (inversa del coseno)
`acosh`	Inversa del coseno hiperbólico
`atan`	Arcotangente (inversa de la tangente)
`atan2`	Arcotangente de cuatro cuadrantes

Cuadro 2: Funciones elementales de Matlab: Trigonometría.

Nombre	**Función**
`exp`	Exponencial
`log`	Logaritmo natural (base e)
`log2`	Logaritmo en base 2
`log10`	Logaritmo en base 10
`sqrt`	Raíz cuadrada

Cuadro 3: Funciones elementales de Matlab: Exponenciales.

Nombre	Función
`fix`	Redondear hacia cero
`floor`	Redondear hacia menos infinito
`ceil`	Redondear hacia más infinito
`round`	Redondear hacia el entero más cercano
`mod`	Módulo de la división entera
`rem`	Resto de la división entera

Cuadro 4: Funciones elementales de Matlab: Ajuste y redondeo.

Nombre	Función
`inv`	Matriz inversa
`det`	Determinante
`eig`	Autovalores
`'`	Matriz traspuesta
`eye`	Crear una matriz identidad dado el número de filas/columnas
`zeros`	Crear una matriz de ceros dado el número de filas/columnas
`ones`	Crear una matriz de unos dado el número de filas/columnas
`length`	Longitud de un vector
`size`	Dimensiones de una matriz

Cuadro 5: Funciones elementales de Matlab: Operaciones matriciales.

4.7. Operaciones lógicas

Algunas de las operaciones y funciones presentadas no devuelven un valor numérico o matricial como resultado. En su lugar, evalúan si cierta condición es verdadera o falsa. En estos casos, el valor devuelto por la función equivaldrá a 1 si la condición se cumple, y 0 si no.

A modo de ejemplo comprobaremos si una variable x se encuentra en un intervalo determinado

```
>> x=5
>> (x>=0)&(x<=10) % Intervalo [0,10]
ans =
     1
>> (x>7)&(x<10) % Intervalo (7,10)
ans =
     0
```

Las operaciones lógicas se emplearán sobre todo para implementar bifuraciones y bucles en los programas Matlab.

Nombre	Función
`clear`	Elimina todas las variables del espacio de trabajo
`clear x`	Elimina la variable `x` del espacio de trabajo
`who`	Lista las variables del espacio de trabajo

Cuadro 6: Funciones elementales de Matlab: Espacio de trabajo.

4.8. Operaciones de rango

En Matlab existe un operador de gran utilidad que no tiene parangón en otros lenguajes de programación: el operador de rango (`:`). Para ilustrar su utilidad, baste indicar que si se desea crear un vector con todos los números enteros entre 1 y 10, podemos emplear la expresión `1:10`.

```
>> a=1:10
a =
     1     2     3     4     5     6     7     8     9    10
```

En general, para secuencias no enteras o no crecientes la sintaxis del operador de rango es

```
valor_minimo : incremento : valor_maximo
```

Por ejemplo, para generar todos los números entre 1 y 2 en incrementos de 0.2 escribiremos

```
>> a=1:0.2:2
a =
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
```

Una segunda aplicación del operador de rango es el acceso a submatrices o subvectores. Supongamos que hemos definido la matriz `mat` anteriormente mencionada:

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

Para acceder a la submatriz comprendida entre los elementos $(2, 1)$ y $(3, 2)$ bastará con escribir

```
>> mat(2:3,1:2)
ans =
     4     5
     7     8
```

Además, se puede prescindir de alguno de los extremos de este operador cuando se emplea en el acceso a vectores y matrices. Por ejemplo, si se desea mostrar todos los elementos menos los 3 primeros de un vector:

```
>> a(4:)
>> a(4:end)
```

Por otro lado, si lo que deseamos es obtener los 3 últimos elementos del vector `a`, escribiremos

```
>> a((length(a)-2):end)
```

4.9. Almacenamiento en archivos

Matlab permite almacenar en el disco las variables del espacio de trabajo. De este modo es posible parar una sesión de trabajo y continuar en otro momento sin volver a repetir cálculos. La orden más común para almacenar datos es `save`, que puede usarse de varias maneras. En la tabla siguiente se presenta un resumen.

Orden	Operación realizada.
`save`	Crea el archivo de nombre *matlab.mat* en la carpeta actual. Dicho archivo contiene todas las variables que existen en ese momento en entorno Matlab.
`save nombrearchivo`	Crea el archivo de nombre en *nombrearchivo.mat* en la carpeta actual. Dicho archivo contiene todas las variables que existen en ese momento en el entorno Matlab.
`save nombrearchivo x y z`	Crea el archivo de nombre *nombrearchivo.mat* en la carpeta actual. Dicho archivo contiene únicamente las variables `x`, `y` y `z`.

Para recuperar las variables almacenadas en un fichero previamente creado emplearemos principalmente la función `load`. La siguiente tabla ilustra tres operaciones típicas de recuperación de datos.

Orden	Operación realizada.
`load`	Lee toda las variables del archivo de nombre *matlab.mat* de la carpeta actual. Si alguna de las variables del disco tiene nombre coincidente con otra que previamente existe en Matlab se producirá la destrucción de la variable existente para dejar su sitio a la variable del disco.
`load nombrearchivo`	Igual que en el caso anterior, pero leyendo del archivo *nombrearchivo.mat* de la carpeta actual.
`load nombrearchivo x y z`	Igual que el anterior pero leyendo únicamente las variables `x`, `y` y `z`.

4.10. Gráficas en Matlab

Las posibilidades de Matlab a la hora de crear gráficos de todo tipo son vastísimas. Para tener una visión general de ellas se recomienda al lector un recorrido por la ayuda en línea partir del comando

```
>> doc plot
```

En este punto veremos los pasos básicos para crear una gráfica a partir de una tabla de valores (x, y). Concretamente, trazaremos la parábola de ecuación

$$y = 2x^2 + 3x - 1$$

en el intervalo $[-3, 3]$.

Toda gráfica de Matlab ha de ser creada a partir de una nube de puntos, que en el caso bidimensional consiste en una serie de valores de las coordenadas x y otra serie del mismo tamaño de valores de y. Cada pareja (x,y) formada a partir de ambas series será un punto de la gráfica. Para ello crearemos dos vectores de igual tamaño n. El primer vector será $\mathbf{x}$, para las coordenadas de los puntos, a partir de una división suficientemente fina del eje de abcisas:

```
>> x=-3:0.1:3;
```

y a continuación creamos el vector $\mathbf{y}$, sabiendo que en el gráfico el elemento i-ésimo del dicho vector formará un punto (x, y) con el elemento i-ésimo del vector $\mathbf{x}$. Por tanto,

se ha de crear un vector **y** de n componentes, según la fórmula

$$y_i = 2x_i^2 + 3x_i - 1 \qquad i = 1 \ldots n$$

Esto se obtiene en Matlab con un sólo comando, sin necesidad de bucles:

```
>> y=2x.^2+3x-1;
```

Obsérvese el '.' antes de la exponenciación. Esto evita que el término `x^2`, al ser `x` un vector, se calcule como el producto escalar de **x** por sí mismo. Finalmente, creados los vectores, creamos la gráfica y la etiquetamos con los siguientes comandos:

```
>> plot(x,y); % El orden de los parametros es importante
>> grid; % Visualizar una malla
>> xlabel('Eje x'); % Etiqueta eje x
>> ylabel('Eje y'); % Etiqueta eje y
```

Obteniendo el gráfico de la figura:

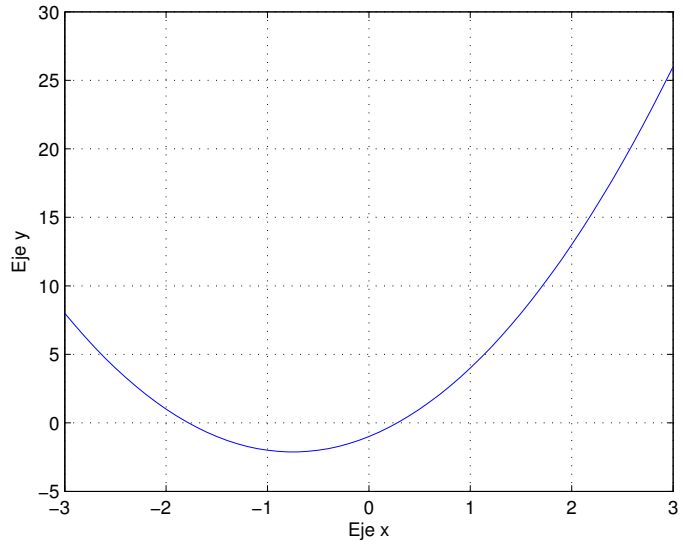

Figura 2: Gráfico resultante.

Es muy recomendable consultar la ayuda para conocer las opciones en cuanto a tipos y colores de línea, tratamiento de ejes (comando `axis`), etiquetado (comandos `xlabel`, `legend`, `text`), etc.

5. Control System Toolbox

El Control System Toolbox es un componente opcional en la instalación de Matlab que consta de una serie de funciones, objetos, bloques Simulink y herramientas destinados a la asistencia en el análisis y diseño de sistemas de control. El objeto fundamental con el que trabajaremos es la **función de transferencia**. Para ilustrar sus propiedades y el álgebra asociada, estudiaremos un ejemplo sencillo de control.

Considérese el sistema realimentado de la figura 3. Dicho sistema está formado por tres bloques independientes: $G_1(s)$, que representa el controlador, $G_2(s)$, que corresponde a la planta a controlar, y $G_3(s)$, la función de transferencia del sensor con el que se mide la salida del sistema. Los valores de las tres funciones son:

$$\begin{aligned} G_1(s) &= \frac{1}{s+0{,}5} \\ G_2(s) &= \frac{3}{s^2+2s+1} \\ G_3(s) &= \frac{40}{s^2+5s+40} \end{aligned}$$

Supongamos que deseamos calcular la función de transferencia en bucle cerrado de dicho sistema, y a continuación trazar su diagrama de Bode. Lo primero que debemos conocer es cómo definir una función de transferencia en el entorno Matlab.

Un polinomio en s se representa en Matlab como un vector cuyos elementos son los coeficientes del polinomio por orden de exponente descendente: por ejemplo, s^2-2s+1 se define en Matlab como el vector $[1\ \ -2\ 1]$. Por tanto, para definir una función de transferencia en Matlab necesitamos dos vectores. A partir de ellos, con la función `tf` construiremos las función de transferencia del ejemplo:

```
>> G1=tf([1],[1 0.5]);
>> G2=tf([3],[1 2 1]);
>> G3=tf([40],[1 5 40]);
```

Lo más interesante de esos objetos es la posibilidad de realizar operaciones matemáticas entre ellos. Para ilustrar esto, calcularemos la función de transferencia del sistema realimentado en bucle cerrado, desde la referencia hasta la salida. Sabiendo que dicha función tiene la forma

$$G_{bc}(s) = \frac{Y(s)}{R(s)} = \frac{G_1(s)G_2(s)}{1+G_1(s)G_2(s)G_3(s)},$$

teclearemos en Matlab simplemente

```
>> Gbc=G1*G2/(1+G1*G2*G3)
Transfer function:
      3s^5+22.5s^4+163.5s^3+331.5s^2+247.5s+60
-----------------------------------------------------------------
s^8+10s^7+75.25s^6+262.3s^5+471.5s^4+594.5s^3+570.3s^2+321.3s+70
```

lo que nos muestra la estructura de la función de transferencia en bucle cerrado $G_{bc}(s)$, que podrá ser utilizada a partir de ahora en llamadas a funciones, como las que trazan los diagramas de Bode (función `bode`) y Nyquist (función `nyquist`).

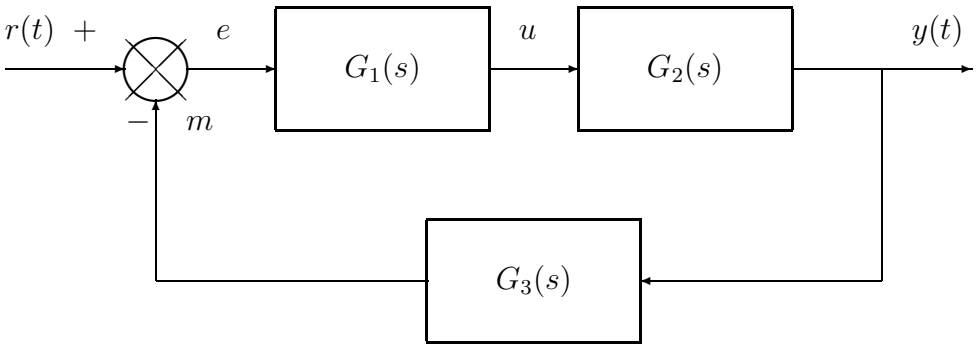

Figura 3: Sistema de Control realimentado

5.1. Operaciones con polinomios

El Control System Toolbox permite, además de lo explicado, realizar ciertas operaciones con polinomios almacenados en forma de vector, que son muy interesantes dentro de la teoría del control automático. Por ejemplo, podemos calcular el producto de dos polinomios en s mediante la función `conv` y a partir de ellos calcular el producto (cascada) de dos funciones de transferencia:

```
>> num1=[1]; den1=[1 0 0.5];
>> num2=[3]; den2=[1 2 1];
>> num_producto=conv(num1,num2);
>> den_producto=conv(den1,den2);
>> G12=tf(num_producto,den_producto)
```

en este caso el resultado del cálculo sería igual al producto de las funciones $G_1(s)$ y $G_2(s)$, que como sabemos, también se obtendría, a partir de las definiciones anteriores,

escribiendo

```
>> G12=G1*G2
```

Por otra parte, para obtener las raíces de un polinomio definido en Matlab como un vector, se emplea la función `roots`:

```
>> roots([1 2 -1 ])
ans =
   -2.4142
    0.4142
```

5.2. Herramientas numéricas y gráficas

Dada una función de transferencia, ya sea de bucle abierto o cerrado, existen en el Control System Toolbox operaciones numéricas y gráficas de gran utilidad a la hora de analizar la estabilidad y otras propiedades. Algunas de ellas aparecen en la siguiente tabla

Comando	Operación realizada.
`evalfr`	Evalúa la magnitud y fase de una función de transferencia en la frecuencia especificada.
`bode`	Traza el diagrama logarítmico de Bode de una función de transferencia dada. Además presenta interesantes opciones de visualización como son los márgenes de ganancia y fase.
`nyquist`	Traza el diagrama de Nyquist de una función de transferencia dada.
`rlocus`	Traza el lugar de las raíces al realimentar negativamente con una ganancia K (variable) la función de transferencia dada.
`margin`	Calcula, sobre el diagram de Bode, los márgenes de fase y ganancia de una función de transferencia y las frecuencias de corte.
`pzmap`	Muestra en una gráfica del plano complejo la ubicación de los polos y los ceros de una función de transferencia dada.

Como ejemplo, se obtendrá el diagrama de Bode de la función de transferencia (estable)

$$G(s) = \frac{1}{s^2 + 0{,}1s + 0{,}5}$$

y se calcularán los márgenes de fase y ganancia. Para ello tecleamos

```
>> G1=tf([1],[1 0.1 0.5])
>> bode(G1);
>> margin(G1);
```

y el resultado aparece representado en la figura 4.

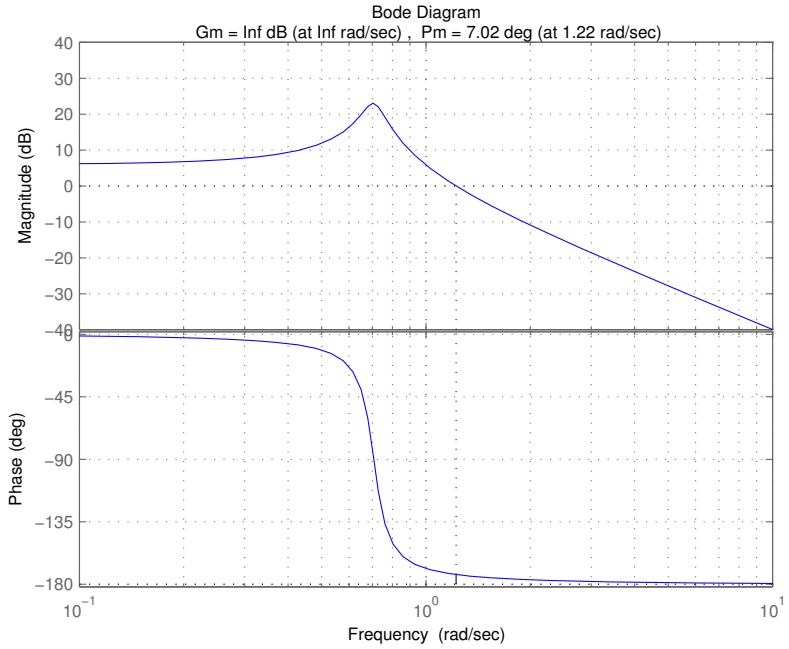

Figura 4: Diagrama logarítmico de Bode.

Por último, volveremos a la función $G_bc(s)$ calculada para analizar su estabilidad. Para ello extraemos su denominador tecleando

```
>> pol=Gbc.den{1}
pol =
1.0000 10.0000 75.2500 262.2500 471.5000 594.5000 570.2500 321.2500 70.0000
```

y a partir de ahí evaluamos sus raíces mediante

```
>> roots(pol)
ans =
  -2.5301 + 5.8437i
  -2.5301 - 5.8437i
  -2.3763
```

```
-0.0317 + 1.2049i
-0.0317 - 1.2049i
-1.0000 + 0.0000i
-1.0000 - 0.0000i
-0.5000
```

Al estar todas las raíces en el semiplano izquierdo, deducimos que el sistema en bucle cerrado es estable. Otro modo de verificar esto es trazando el diagrama polo–cero de G_{bc}, mediante la instrucción `pzmap(Gbc)`. El resultado se muestra en la figura 5.

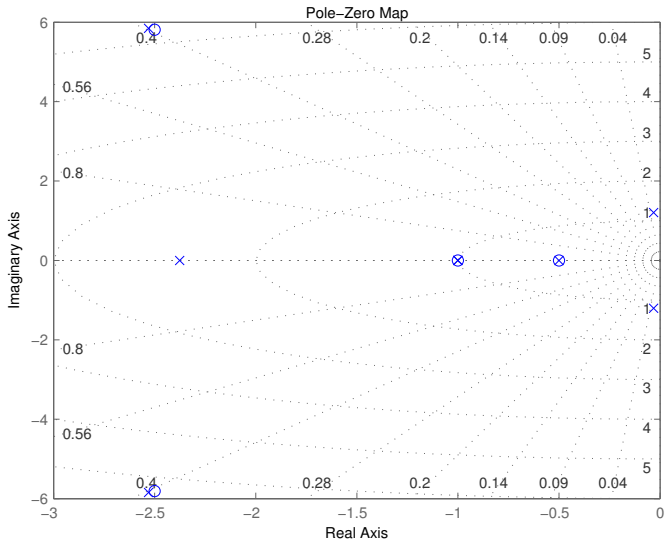

Figura 5: Diagrama polo–cero de la función de transferencia en bucle cerrado $G_{bc}(s)$.

6. El entorno de trabajo de Simulink

Simulink es una herramienta de gran utilidad para la simulación de sistemas dinámicos. Principalmente, se trata de un entorno de trabajo gráfico, en el que se especifican las partes de un sistema y su interconexión en forma de diagrama de bloques. De nuevo, se trata de una herramienta amplísima que además se complementa con numerosos elementos opcionales. Por tanto, nos limitaremos a dar unas pinceladas de los elementos más útiles en Regulación Automática.

Además de las capacidades de simulación de las que está dotado Simulink, conviene destacar que contiene cómodas utilidades de visualización y almacenamiento de resultados de simulación.

6.1. Uso de Simulink

En primer lugar, lanzaremos la aplicación escribiendo `simulink` en la línea de comandos de Matlab, o abriendo desde el Explorador de Windows cualquier fichero con extensión *.mdl*. En el primero de los casos se abrirá la ventana de la figura 6. Esta

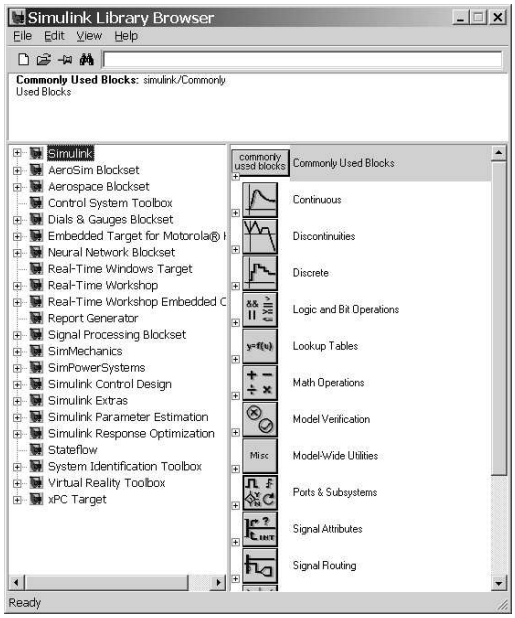

Figura 6: Ventana navegación de bloques de Simulink (Simulink Library Browser).

ventana inicial no está destinada a crear modelos de simulación; su función principal consiste en navegar por la enorme librería de bloques disponibles para el modelado.

En ella distinguimos dos partes: la izquierda contiene una visión en forma de árbol de todos los Toolboxes instalados que contienen bloques Simulink. La amplitud de este árbol dependerá de las opciones que hayamos activado al seleccionar Matlab. De todos los nodos del árbol nos interesan, de momento, los denominados Simulink y Control System Toolbox. Cabe mencionar además, por su interés, los bloques Real Time Workshop destinados a generar automáticamente código de control para determinadas plataformas Hardware comerciales.

La parte derecha de la ventana de la figura 6 muestra los bloques Simulink contenidos en el Toolbox o nodo de la parte izquierda de la ventana. Estos bloques se deben arrastrar sobre el espacio de trabajo de Simulink para la creación de modelo a simular.

Por último, cabe indicar que en la parte superior de la ventana de inicio de Simulink hay varias herramientas como la búsqueda de un bloque determinado a partir de su nombre, que nos pueden resultar bastante útiles.

6.2. El espacio de trabajo de Simulink

Si pulsamos en el icono superior izquierdo de la ventana de la figura 6 (página en blanco), se abre una ventana blanca sobre la que iniciaremos la creación de un modelo de simulación. Dicha ventana se muestra en la figura 7.

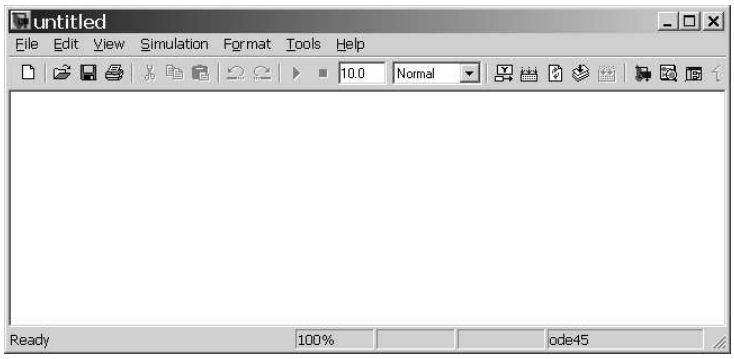

Figura 7: Espacio de trabajo de Simulink.

En el espacio de trabajo de Simulink crearemos un modelo insertando los bloques correspondientes. Concretamente realizaremos la simulación del sistema de control representado en la figura 3. En lugar de emplear las definiciones en Matlab de las funciones de transferencia presentadas en el apartado anterior (empleando la función `tf`), crearemos las funciones de transferencia directamente sobre el diagrama de bloques.

En primer lugar, hemos de insertar tres bloques de tipo *Función de Transferencia* en el modelo. Para ello tecleamos la palabra *transfer* en el campo de búsquedas en la parte superior de la ventana de navegación y el buscador localizará el bloque llamado *Transfer Fcn*, que cuelga del nodo *Simulink*, como se muestra en la figura 8.

Una vez localizado el bloque *Transfer Fcn* arrastraremos dicho bloque hacia el espacio de trabajo de Simulink. El arrastre de bloques se realiza seleccionando el icono del bloque con el botón izquierdo del ratón, y manteniendo éste pulsado se desplazará el cursor hasta la ventana del modelo.

Repetiremos la operación tres veces, para reproducir la estructura de la figura 3, dando lugar a la ventana mostrada en la figura 9.

Una vez insertados los bloques de las funciones de transferencia, les asignamos nombres específicos (G1,G2 y G3) editando el texto al pie de cada icono, y les damos valores a dichas funciones, para que coincidan con los parámetros de las funciones $G_1(s)$, $G_2(s)$ y $G_3(s)$ definidas anteriormente.

Con este fin, haremos doble click sobre cada bloque de función de transferencia, y

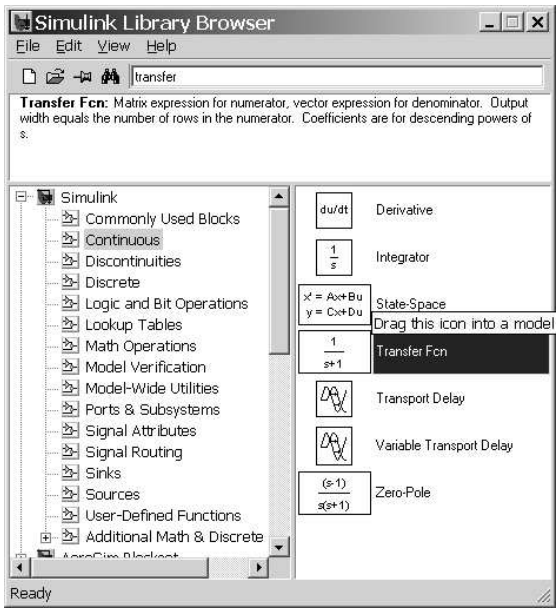

Figura 8: Ubicación del bloque *Transfer Fcn*.

en la ventana que se abre en cada caso, introduciremos los vectores de coeficientes de los polinomios numerador y denominador de cada función de transferencia. La figura 10 muestra la ventana donde se introducen los parámetros de $G_1(s)$.

Una vez configuradas las tres funciones de transferencia las conectaremos entre sí con arreglo a la estructura de interconexión de bloques de la figura 3. Para ello empleamos las siguientes operaciones:

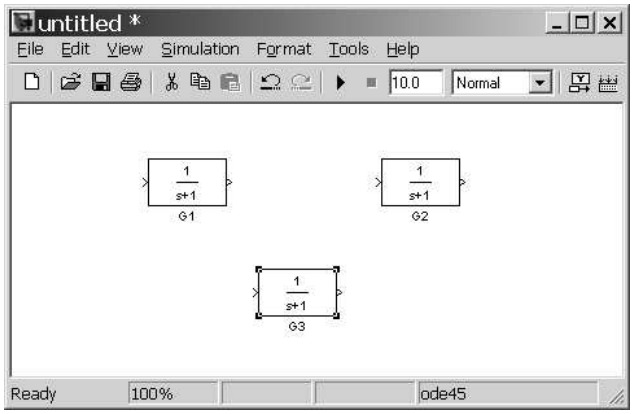

Figura 9: Bloques de función de transferencia en Simulink.

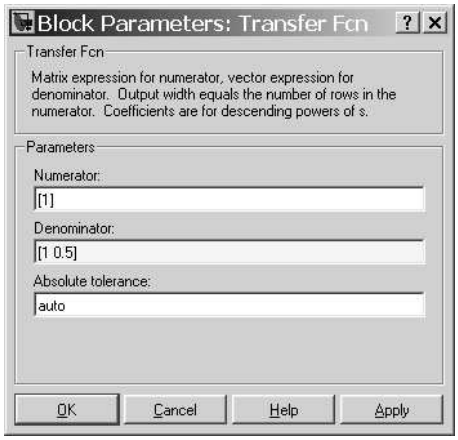

Figura 10: Introducción de los parámetros de $G_1(s) = 1/(s + 0{,}5)$.

Operación	Procedimiento.
Conectar bloques (I)	Para conectar las salidas de un bloque a la entrada de otro, hacer click con el botón izqdo. del ratón en el bloque origen. Pulsar y mantener la tecla CTRL y hacer de nuevo click sobre el bloque destino.
Conectar bloques (II)	También se puede extraer un cable de señal haciendo click en el saliente derecho del bloque origen y prolongar la señal (pulsando y manteniendo el botón izquierdo del ratón) hasta llegar a la parte izquierda del bloque destino.
Bifurcar cables	Un cable de señal (que lleva la salida de un bloque hacia otro bloque), puede bifurcarse para distribuir la señal a varios bloques pulsando con el botón derecho en cualquier punto del cable.
Sumar o restar señales	Las señales procedentes de salidas de los bloques se pueden sumar o restar entre sí mediante el bloque sumador, que se ubica fácilmente tecleando *Sum* en la ventana de navegación de Simuink.

Tras una serie de operaciones de los tipos indicados en la tabla anterior, logramos construir la estructura de realimentación de la figura 11. En esta figura hemos añadido dos bloques nuevos: *Step* y *Scope*. Ambos pertenecen, respectivamente, a los nodos *Simulink/Sources* y *Simulink/Sinks* que serán comentados en el siguiente apartado.

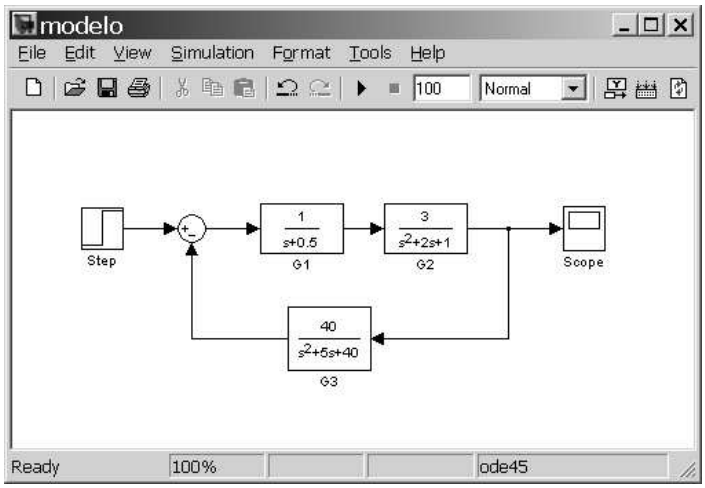

Figura 11: Modelo completo.

6.3. Fuentes y sumideros de señal

Los bloques de suma y resta de señales y los de funciones de transferencia, funcionan como procesadores de señal. Sin embargo, en las simulaciones han de existir fuentes de señal externas, pues lo que se pretende en general es ver cómo responden determinados sistemas a estímulos exteriores.

En nuestro ejemplo necesitamos una señal externa para generar una referencia a seguir por el sistema controlado. Esta referencia debe ser, lógicamente, cambiante con el tiempo. En nuestro caso emplearemos una señal de tipo escalón, que se implementa, con sus parámetros específicos, mediante el bloque *Step*. Bloques como éste, que sólo tienen salidas y ninguna entrada, se localizan en el árbol de navegación de Simulink en el nodo *Simulink/Sources*.

Por otro lado, existen bloques con entradas y sin ninguna salida: nodos sumidero. Uno de ellos es el empleado en nuestro modelo para visualizar la salida del sistema: *Scope*. Los bloques de este tipo se ubican en el árbol de navegación de Simulink en el nodo *Simulink/Sinks*.

A modo de referencia, la tabla 7 muestra algunas fuentes de señal de uso común (nodo *Simulink/Sources*), mientras que la tabla 8 muestra algunos de los bloques sumidero (*Simulink/Sinks*) más comunes.

Elemento	**Función**
Clock	Marcas de tiempo de la simulación. Útil para trazar gráficas con los resultados.
Sin	Señal senoidal parametrizable.
Step	Señal en escalón
Constant	Señal de valor constante.
Signal generator	Permite elegir entre un conjunto de señales predefinidas.
Random Number	Generación de ruido blanco configurable.
From Workspace	Señal generada a partir de una variable del espacio de trabajo de Matlab.

Cuadro 7: Fuentes de señal en Simulink.

Elemento	**Función**
Scope	Gráfica 2D para visualizar las señales frente al tiempo durante la simulación.
XY Graph	Gráfica 2D para visualizar un gráfico X-Y creado a partir de dos señales de entrada.
To Workspace	Almacena las muestras de la señal de entrada en una variable (vector) del espacio de trabajo de Matlab.

Cuadro 8: Sumideros de señal en Simulink.