|  |  |
| --- | --- |
|  | |
| **TALLER** | |
| **CARRERA:** Mecatrónica | **ASIGNATURA**: Sistemas Embebidos de Control Automático |
| **DOCENTE RESPONSABLE:** César Minaya | |
| **TÍTULO**: Matlab para el estudio y análisis de sistemas embebidos de control automático | |

1. **Introducción al software de simulación MATLAB.**

Dada su flexibilidad y amplia adopción, MATLAB emerge como una herramienta robusta en la solución de desafíos en ingeniería de control. Este taller explorará diversas órdenes y funciones dentro del software, esenciales para abordar variadas metodologías y técnicas en el control de sistemas físicos, así como para simular su comportamiento con precisión.

1. **Entorno de trabajo de MATLAB**

Matlab se compone de una interfaz de ventanas dividida en cuatro secciones:

**Command Window:** La ventana de comandos donde se ingresan y ejecutan las instrucciones.

**Current Folder:** la carpeta actual que exhibe el contenido del directorio de trabajo y se puede modificar mediante una barra desplegable.

**Workspace:** el sspacio de trabajo que proporciona detalles sobre las variables y objetos definidos.

**Editor / Command History:** el editor/historial de comandos que posibilita la introducción y ejecución secuencial de comandos, evitando la necesidad de ejecutar cada uno individualmente. Además, esta sección también puede mostrar los comandos previamente ejecutados.

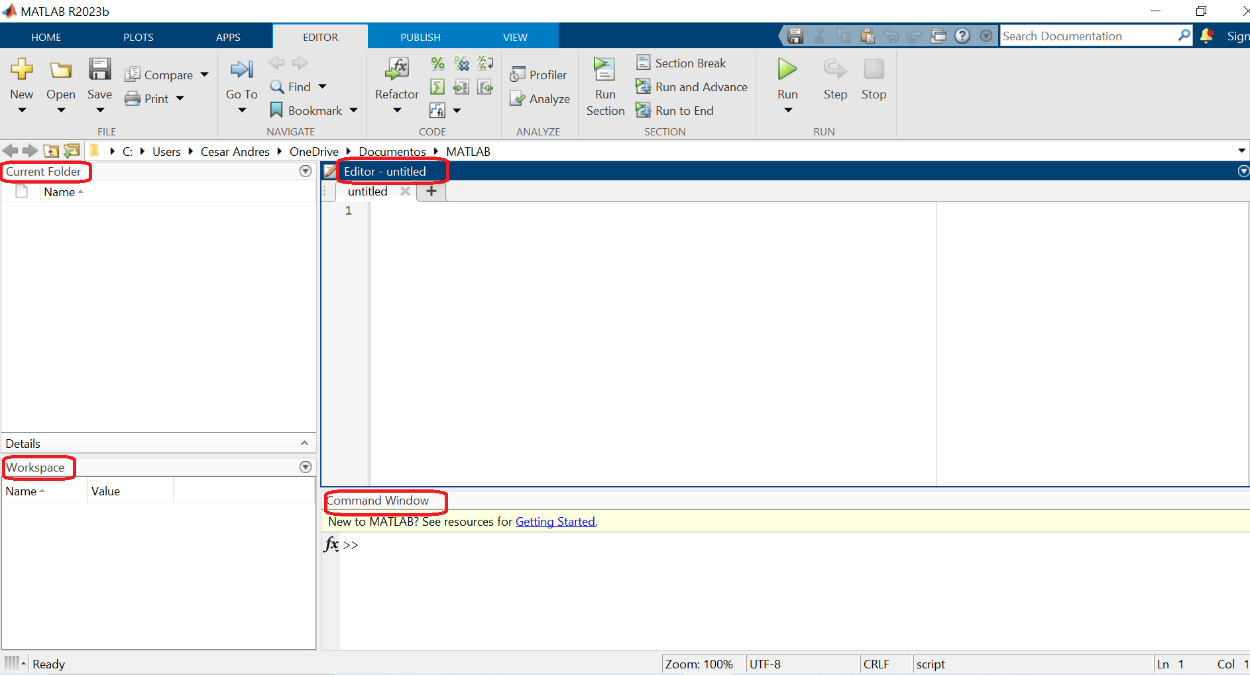
****

Figura 1. Entorno de trabajo de MATLAB.

1. **Variables de Matlab.**

El MATLAB automáticamente crea siempre una variable llamada ans que guarda el resultado de la última operación cuando no se indica ninguna variable donde guardar dicho resultado.

Si se ejecuta:

>> a=2

>> b=3

>> a\*b

>> ans

uno se encuentra que ans guarda el resultado de a\*b.

ans = 6

1. **Introducción de datos.**

**Vectores.**

El modo más sencillo de introducir una secuencia de datos en MATLAB es hacerlo mediante una lista explícita de elementos, donde los elementos deben estar separados por espacios en blanco o por comas, y se encuentran encerrados entre corchetes:

>> x=[1 2 3 4 5]

ó

>> x=[1,2,3,4,5]

x =

1 2 3 4 5

De ahí que el comando anterior creará un vector fila que contiene los elementos 1, 2, 3, 4 y 5. Si lo que queremos es introducir los datos como un vector columna habría que haber puesto:

>> x=[1 2 3 4 5]’

ó

>> x=[1,2,3,4,5]’

x =

1

2

3

4

**Matrices.**

Para representar una matriz hay que tener en cuenta 3 reglas básicas, que son:

* Los elementos deben estar separados por comas o blancos.
* Los elementos deben estar encerrados entre corchetes.
* El final de cada fila se indica con un punto y coma (;) o mediante un retorno de carro.

Así, se puede representar una matriz como sigue:

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

ó

>> A=[ 1 2 3

4 5 6

7 8 9]

A =

1 2 3

4 5 6

7 8 9

Los elementos de una matriz pueden ser cualquier expresión matemática evaluable por MATLAB. Un ejemplo podría ser:

>> B=[1 exp(-0.08); sqrt(2) 2]

B =

1.0000 0.9231

1.4142 2.0000

Cómo ya se vio en el apartado anterior, el apostrofe (‘) indica la traspuesta conjugada de una matriz. Si la matriz es real, la traspuesta conjugada consiste en transponer los elementos de la matriz, es decir cambiar filas por columnas.

>> C=A’

dará como resultado que la matriz C es la matriz traspuesta de la matriz A.

C =

1 4 7

2 5 8

3 6 9

1. **Operaciones con datos.**

**Suma y resta de matrices.**

Para sumar o restar con matrices hay que tener en cuenta que cada una de las matrices operando deben tener las mismas dimensiones. Dadas dos matrices A y B.

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

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

Se puede obtener una matriz C que resulta de sumar A y B como:

>> C=A+B

C =

30 24 18

84 69 54

138 114 90

Y la matriz D que resulta de restar A y B como:

>> D=A-B

D =

-8 -6 -4

-2 0 2

4 6 8

**Multiplicación de matrices.**

Para multiplicar matrices hay que tener en cuenta que las columnas de una de las matrices coinciden con el número de filas de la otra matriz. Dadas dos matrices A y X, donde el número de columnas de A es igual al número de filas de X.

>> X=[ 1; 2; 3]

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

Se puede obtener una matriz Y que resulta de multiplicar A por X como:

>>Y=A\*X

Y =

46

28

10

**Otras operaciones con matrices y vectores.**

Si se desea obtener una matriz B donde cada uno de los elementos sea el cuadrado de los elementos de una matriz A se utilizaría la instrucción siguiente:

B=A.^2

Si se desea multiplicar, uno a uno, los elementos correspondientes de dos matrices A y B (que ocupan la misma posición de fila y columna) se ejecutaría la siguiente instrucción:

C=A.\*B

Si se desea dividir, uno a uno, los elementos correspondientes de dos matrices A y B (que ocupan la misma posición de fila y columna) se ejecutaría la siguiente instrucción:

C=A./B

Si lo que se quiere es dividir los elementos B con sus correspondientes en A, entonces podríamos poner:

C=B./A ó C=A.\B

**Matrices especiales.**

La matriz identidad I es una matriz cuadrada de dimensiones nxn cuyos elementos son cero a excepción de su diagonal cuyos valores son 1. Para definir una matriz identidad en MATLAB se usa la sentencia eye(n), donde n indica las dimensiones de la matriz:

>> I=eye(5)

I =

1 0 0 0 0

0 1 0 0 0

0 0 1 0 0

0 0 0 1 0

0 0 0 0 1

dará una matriz 5x5 con unos en la diagonal y el resto ceros.

Una matriz diagonal es una matriz cuadrada cuyos elementos son cero a excepción de su diagonal cuyos valores son un número determinado m. Si x es un vector, la orden diag(x) produce una matriz diagonal con x sobre la diagonal. Por ejemplo:

>> x=[1 2 3]

>> D=diag(x)

D =

1 0 0

0 2 0

0 0 3

**Nombre de la función Cometario**

* ones(n) Crea una matriz nxn de unos.
* ones(m,n) Crea una matriz mxn de unos.
* zeros(n) Crea una matriz nxn de ceros
* zeros(m,n) Crea una matriz mxn de ceros
* diag(A) Obtiene la diagonal de una matriz A

**ACTIVIDAD**

Considere la siguiente matriz:

𝐴=[5 17 61

32 80 −44

−1 −11 −13]

Indicar el resultado de las siguientes operaciones (utilice Matlab) y comente que significa cada una de ellas:

* A(:,1)
* det(A)
* A(:,2:3)
* poly(eig(A))
* B=[A,[ones(1,2);eye(2)]]
* A(:,:)
* diag(A)
* ones(4,3)
* eye(3)
* A(2:4)=[]
* zeros(size(A))
* rand(size(A))
* magic(length(A))
* rank(A)

1. **Representación gráfica de curvas.**

MATLAB dispone de un conjunto de rutinas para obtener diferentes tipos de salidas gráficas. El modo de utilización de cada una de ellas es muy similar, siendo la única diferencia entre todas ellas el modo en el que se escalan los ejes y el modo de visualización los datos.

Las rutinas básicas para construir gráficas se muestran en la siguiente tabla:

|  |  |
| --- | --- |
| **Nombre de la función** | **Comentario** |
| plot | Crea un gráfico a partir de dos vectores de datos o columnas de matrices |
| subplot | Permite crear múltiples gráficos en una misma figura |
| loglog | Crea un gráfico con escala logarítmica para ambos ejes |
| semilogx | Crea un gráfico con escala logarítmica para el eje x, y escala lineal para el eje y |
| semilogy | Crea un gráfico con escala logarítmica para el eje y, y escala lineal |
| title | Añade título al gráfico |
| xlabel | Añade título al eje x |
| ylabel | Añade título al eje y |
| text | Añade texto en una localización determinada |
| gtext | Añade texto a la gráfica |
| grid | Crea líneas de rejilla |
| polar(theta,ro) | Crea un gráfico en coordenadas polares de ángulo theta frente a radio |

Tabla 1. Funciones de construcción y manejo de gráficas.

**Representación gráfica de curvas.**

Para dibujar una gráfica a partir de dos vectores x e y de la misma longitud se procede de la siguiente manera: plot(x,y)dibuja los valores de y frente a los valores de x. Por ejemplo:

>> x=[1 2 3 4 5 6 7]

>> y=[2 4 9 16 25 36 49]

>>plot(x,y)

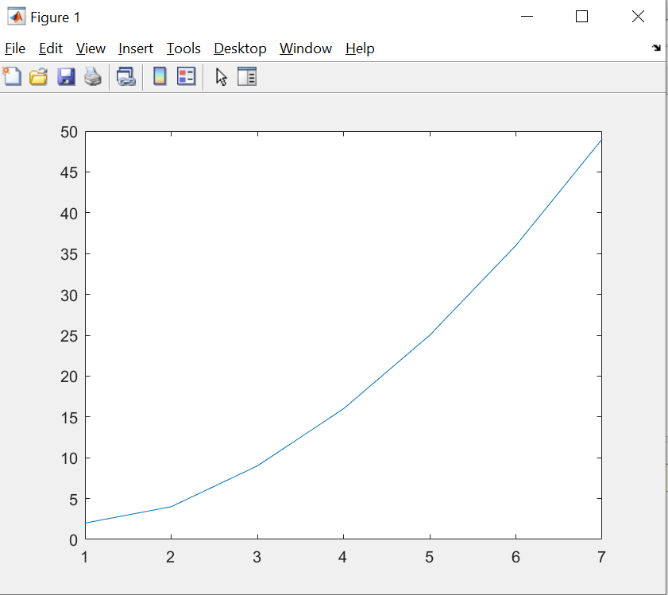


Figura 2. Ejemplo de construcción de una gráfica sencilla.

Representación de varias curvas en una misma gráfica.

Para dibujar varias curvas en una misma gráfica se utiliza la orden plot con múltiples argumentos. Por ejemplo, si se quiere dibujar n curvas cuyos valores se encuentran en los vectores x e y de cada una de ellas, la rutina sería:

plot(x1,y1,x2,y2,x3,y3,...,xn,yn)

donde x1 e y1 son los vectores de los valores de la primera curva, x2 e y2 los vectores de los valores de la segunda curva y así sucesivamente hasta n curvas.

Por ejemplo:

>> x=[1 2 3 4 5 6 7]

>> y=[1 4 9 16 25 36 49]

>> z=[1 8 27 64 125 216 343]

>>plot(x,y,x,z)

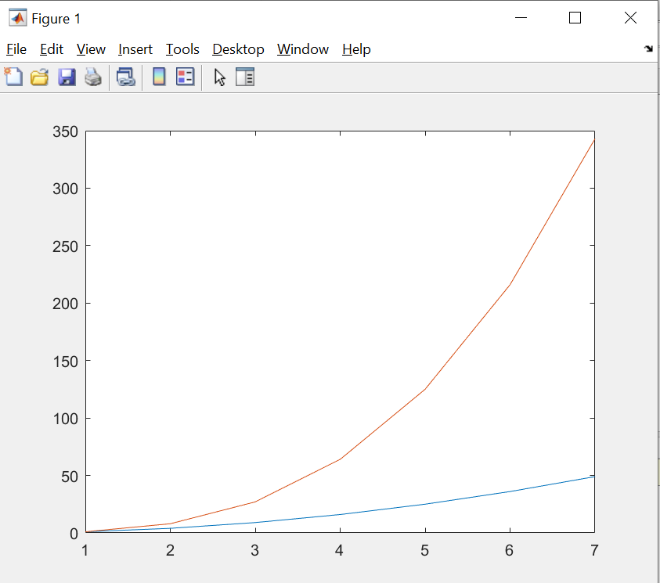


Figura 3. Ejemplo de construcción de varias gráficas en una misma ventana.

Otra forma de dibujar más de una curva en un único gráfico se consigue utilizando la orden hold. Esta orden congela el gráfico actual e inhibe las acciones de borrado y escalado. De modo, que usando esta orden, las curvas que se dibujen después de la ejecución de este comando se dibujarán sobre la curva original, superponiendo unas sobre otras. Si se desea activar de nuevo el redibujado, permitiendo el borrado y escalado, será necesario volver a ejecutar la orden hold.

>> x=[1 2 3 4 5 6 7]

>> y=[1 4 9 16 25 36 49]

>> plot(x,y)

>> hold

>> z=[1 8 27 64 125 216 343]

>> plot(x,z)

Matlab permite representar varias subfiguras en una única figura. Para ello, se emplea el comando subplot(m,n,p) donde el primer parámetro corresponde al número de filas, el segundo parámetro al número de columnas y el tercer parámetro a la posición relativa en la matriz de subfiguras.

**Ejemplo de representación de una gráfica.**

Dibujar la gráfica de la función sin(x) en el intervalo 0 £ x £ 10 con un paso de 0.05.

>>x=0:0.05:10;

>>y=sin(x);

>>plot(x,y);

>>grid;

>>title(‘Gráfica del seno’);

>>xlabel(‘Segundos’);

>>ylabel(‘Seno(x)’);

>>text(3,0.45,’sen(x)’);

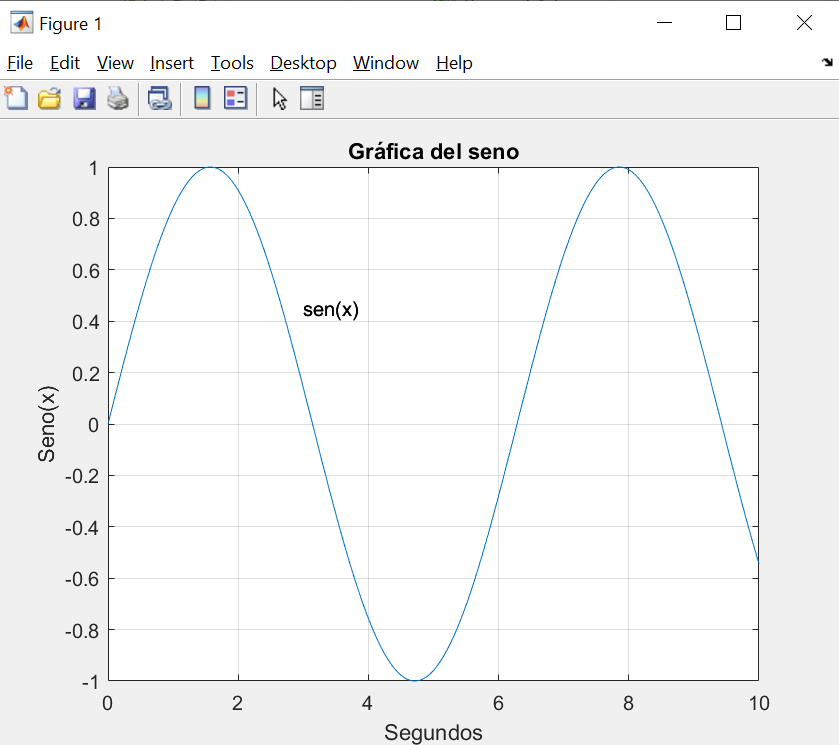


Figura 4. Ejemplo de representación gráfica y etiquetado.

**ACTIVIDAD**

* Representar gráficamente las siguientes funciones en ventanas diferentes, f(x)=sen(x), g(x)=x2+3x en el intervalo [0,2π].
* Representar f(x)=sen(x)cos(x) en [0,2π], con etiquetas en los ejes, título y en color rojo.
* Representar en [0,6] y en la misma gráfica las funciones: a. f(x)=3xex en azul. b. g(x)=sen(x+3) en rojo y con trazo discontinuo. Poner leyendas.

1. **Transformadas y transformadas inversas.**

La transformada de Laplace, la transformada de Fourier y la transformada z son tres técnicas de transformación que proporcionan una conversión de variables. Todas ellas convierten modelos de ecuaciones diferenciales lineales a modelos algebraicos. La transforma de Laplace se usa para obtener una función de transferencia, que modeliza el comportamiento de un sistema continuo, mientras que la transformada z modeliza el comportamiento de sistemas discretos.

|  |  |
| --- | --- |
| **Nombre de función Comentario** | **Nombre de función Comentario** |
| fourier | Transformada de Fourier |
| ifourier | Transformada inversa de Fourier |
| laplace | Transformada de Laplace |
| ilaplace | Transformade inversa de Laplace |
| ztrans | Transformada z |

Si se desea calcular por ejemplo la transformada de Laplace de una función

en primer lugar, se necesita declarar una variable de tipo simbólica con el uso de la instrucción syms y después se ejecuta el comando de la transformada que se desea calcular pasando como parámetro la función de la cual se quiere calcular su transformada. Por ejemplo, para este caso:

>> syms t

>> TL= laplace(sin(t)-2\*exp(-2\*t))

El resultado que se obtiene es:

TL =

1/(s^2+1)-2/(s+2)

Si ahora se calcula la transformada inversa de Laplace se obtiene la función f(t) de partida:

>> syms s

>> f=ilaplace(1/(s^2+1)-2/(s+2))

En el caso de un sistema discreto, se puede obtener la transformada z de un modo muy similar:

>> syms n

>> TZ= ztrans((-9\*(0.9)^n+10)

El resultado que se obtiene es :

TZ =

(10\*z)/(z - 1) - (9\*z)/(z - 9/10)

Si ahora se calcula la transformada z inversa se obtiene la función f(n) de partida:

>> syms z

>> f=iztrans((10\*z)/(z - 1) - (9\*z)/(z - 9/10))

f =

10 - 9\*(9/10)^n

**ACTIVIDAD**

**Ejercicio 1**

Realizar un script de Matlab llamado **orbita.m** que represente una señal de las funciones (x vs. y) en función de los siguientes parámetros de entrada: tiempo (t),

Las ecuaciones que se muestran a continuación representan la órbita de Mercurio (x vs. y) alrededor de la tierra. Generar una función para ingresar las ecuaciones y mediante un script graficar dicha orbita.

Generar tanto x(t) e y(t) como funciones de Matlab y la variable independiente t desde 0 hasta 44𝜋/3 en intervalos de 𝜋/360. Incluir etiquetas en los ejes, título de los gráficos, etc.

**Ejercicio 2**

La transformada de Laplace es una herramienta matemática más usada en control. Ésta permite obtener una solución simbólica que simplifica las ecuaciones físicas que modelan el comportamiento del sistema continuo que se quiere someter a estudio. Aplicando transformaciones como la de Laplace se consiguen funciones donde las relaciones causa-efecto en un sistema son más fáciles de entender. Así, en definitiva, con la transformada de Laplace lo que se consigue es simplificar las funciones que modelan el sistema convirtiendo ecuaciones diferenciales de variable real en modelos algebraicos de variable compleja. En el caso de control de sistemas discretos, se utiliza la transformada Z.

1. **Modelización de sistemas usando MATLAB**

MATLAB ofrece un conjunto de herramientas valiosas para convertir un modelo matemático específico de un sistema lineal a otro modelo. A continuación, se describe el enfoque utilizado para aplicar estas transformaciones de manera efectiva en la resolución de problemas en ingeniería de control.

**Funciones de transferencia. Sistemas continuos.**

La representación de una función de transferencia implica la relación de dos polinomios como un cociente. Una forma de ingresar esta función en MATLAB implica la definición de dos vectores fila, donde cada uno contendrá los coeficientes del numerador y del denominador respectivamente.

Por ejemplo, para definir la función de transferencia:

se realiza en MATLAB la siguiente secuencia de comandos:

>> n=[0 5 20]

>> d=[1 4 20]

En control, existen diversas técnicas de análisis y diseño que necesitan obtener las raíces de los polinomios numerador y denominador de la función de transferencia para conocer la localización de sus polos y ceros, así como una constante denominada factor de ganancia. Los polos se pueden definir matemáticamente como las raíces del polinomio del denominador y los ceros como las raíces del numerador.

MATLAB proporciona una orden de conversión para transformar la función de transferencia a una notación que permita trabajar con los polos y ceros y ganancia por separado. Por ejemplo, si se quiere pasar la función de transferencia anterior a polos-ceros bastará con ejecutar el comando:

>> [z p k]=tf2zp(n,d)

z =

-4

p =

-2.0000 + 4.0000i

-2.0000 - 4.0000i

k =

5

donde z es un vector que contiene los valores de los ceros, p los valores de los polos y k el factor

de ganancia.

Además, también se puede realizar la operación contraria. Así, dados los valores de polos ceros-

ganancia se puede obtener la función de transferencia equivalente ejecutando el comando zp2tf. Por ejemplo, se conocen los ceros (s + 4) y los polos (s + 2 - 4 j)(s + 2 + 4 j) y se quiere

obtener la función de transferencia equivalente.

>> k=5

>> z=-4

>> p=[-2+4\*j –2-4\*j]’

>> [n,d]=zp2tf(z,p,k)

n =

0 5 20

d =

1 4 20

Otra manera de introducir la función de transferencia en dominio de Laplace es haciendo uso de la orden tf.

>> G=tf(n,d)

Otro ejemplo:

>> G=tf([1 1],[1 2 1])

Transfer function:

s + 1

-------------

s^2 + 2 s + 1

De este modo la variable G representará una vez ejecutada la orden tf el sistema continuo identificado por su función de transferencia.

**Descomposición de la función de transferencia.**

La descomposición en fracciones simples es una metodología matemática muy útil para obtener los polos y ceros cuando la función de transferencia está constituida por polinomios de un orden elevado.

Por ejemplo, dada la función de transferencia se quiere obtener su descomposición en fracciones simples.

El primer paso es introducir la función de transferencia:

>> n=[2 5 3 6]

>> d=[1 6 11 6]

>> printsys(n,d)

num/den =

2 s^3 + 5 s^2 + 3 s + 6

-----------------------

s^3 + 6 s^2 + 11 s + 6

Una vez se ha introducido ésta en forma de dos vectores fila, se utiliza la orden residue de la siguiente manera:

>> [r p k]=residue(n,d)

r =

-6.0000

-4.0000

3.0000

p =

-3.0000

-2.0000

-1.0000

k =

2

La salida p es un vector columna que contiene los polos, la salida r es otro vector columna que contiene los residuos de los polos correspondientes y la salida k es un vector fila que contiene los términos constantes.

Así, el resultado de descomponer la función de transferencia anterior en fracciones simples es:

También, es posible pasar de un conjunto de fracciones parciales simples al cociente polinómico

que determina una función de transferencia, con la función residue, como sigue:

>> [n,d]=residue(r,p,k);

n =

2.0000 5.0000 3.0000 6.0000

d =

1.0000 6.0000 11.0000 6.0000

**Descomposición de la función de transferencia.**

La función de transferencia de un sistema describe la relación algebraica entre las variables de salida y las de entrada. Un diagrama de bloques proporciona una representación visual de estas relaciones algebraicas. MATLAB facilita el cálculo de la función de transferencia en bucle cerrado de un sistema representado mediante un diagrama de bloques. Para simplificar un diagrama de bloques a su función de transferencia equivalente, se pueden emplear las funciones feedback, series o parallel.

A continuación, se presenta la secuencia de comandos que permiten calcular la función de transferencia en bucle cerrado del sistema representado en la Figura 5.

>> G=tf([0 0 100],[1 2 0]) %Funcion G(s)

>> H=tf([0 1],[1 20]) %Funcion H(s)

%Calcula función de transferencia de la realimentación

>> M=feedback(G,H,-1) % Funcion M(s)

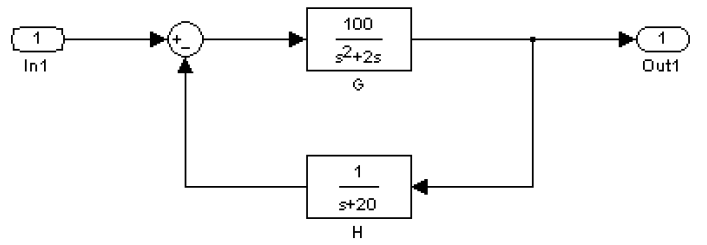


Figura 5. Diagrama de bloques equivalente.

Si se aumenta la complejidad del sistema hasta el representado por la Figura 6. La secuencia de ordenes se podría poner como:

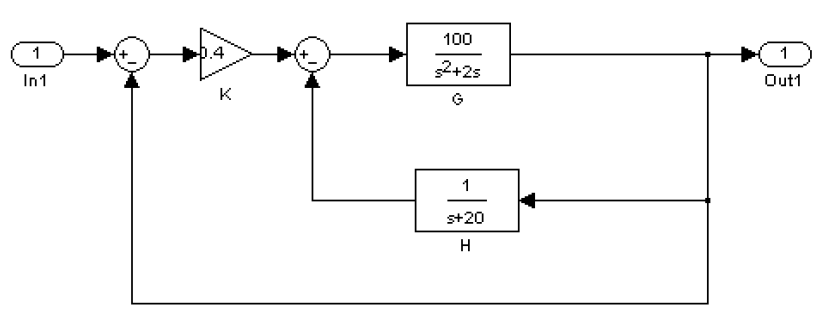


Figura 6. Diagrama de bloques

>> K=tf(0.4,1) % Ganancia K

>> G=tf([0 0 100],[1 2 0]) %Funcion G(s)

>> H=tf([0 1],[1 20]) % Funcion H(s)

%Calcula función de transferencia de la realimentación G y H

>> R1=feedback(G,H,-1)

%Calcula función de transferencia en serie con el factor K

>> S1=series(K,R1)

%Calcula función de transferencia de la realimentación general

>> M=feedback(S1,tf(1,1),-1) % Funcion M(s)

**Visualización de la respuesta de un sistema**

Para mostrar cómo se comporta un sistema continuo, es útil analizar cómo responde a una entrada específica, como un escalón. En sesiones posteriores, se examinará más detalladamente la respuesta temporal de los sistemas. Por ejemplo, si tenemos un sistema continuo con una función de transferencia G, podemos obtener su respuesta al escalón utilizando la función step de la siguiente manera:

n=[0 5 20];

>> d=[1 4 20];

>> G=tf(n,d);

>> t=10 % tiempo de representación

>> [Y,T] = step(G,t); % respuesta ante escalón

>> plot(T,Y)

>> grid

>> title('Respuesta ante entrada escalón')

>> xlabel('Tiempo(en segundos)')

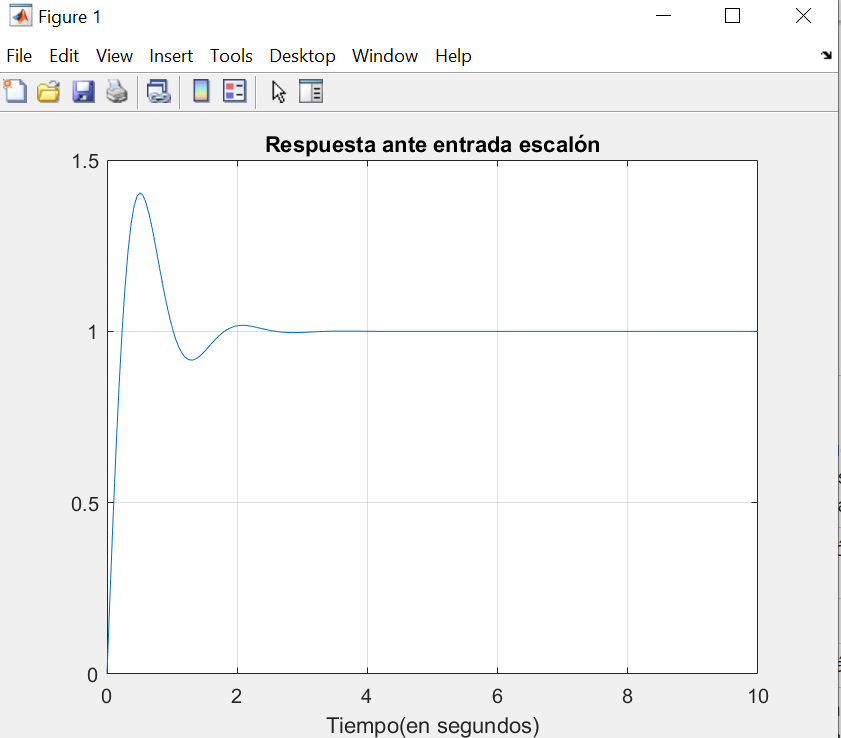


Figura 7. Respuesta ante entrada escalón de un sistema continuo

**ACTIVIDAD**

Realice las siguientes actividades utilizando la función de transferencia G(s):

* Usando alguno de los métodos mencionados anteriormente declare G(s).
* Determine el valor de los polos, los ceros de G(s), además encuentre el valor de la constante “k”.

Realice las siguientes actividades utilizando la función de transferencia M(s):

* Declare M(s) utilizando la notación de polos, ceros y la constante “k”.
* Transforme M(s) a su forma de polinomio de manera que se compruebe que:

Calcular la función de transferencia en bucle cerrado del sistema representado por G(s) “planta” y M(s) “retroalimentación” feedback negativa.

Ilustrar el comportamiento de un sistema continuo para un tiempo de 20s.

1. **Modelización de sistemas usando SIMULINK.**

MATLAB cuenta con una interfaz gráfica llamada SIMULINK, que ofrece la ventaja de permitir el modelado de sistemas de manera más intuitiva a través de representaciones gráficas. Esto elimina la necesidad de depender únicamente de la ventana de comandos. La potencia y versatilidad de SIMULINK se hacen evidentes especialmente al describir un sistema mediante la construcción directa de su diagrama de bloques, sin la necesidad de utilizar funciones como feedback, series o parallel que anteriormente requerían comandos específicos para modelar el mismo diagrama.

**Iniciar SIMULINK.**

Para iniciar SIMULINK se puede ejecutar la orden simulink en la ventana de comandos de MATLAB o bien pinchar en el botón  . Se abrirá una biblioteca con todos los elementos (bloques de funciones de transferencia, tipos de entradas, sumadores, bloques de factores de ganancia, etc) necesarios para dibujar diagramas de bloques. Todos estos bloques se organizan en diferentes grupos de acuerdo a cuál es su funcionalidad y cometido. Algunos de los grupos más empleados son:

* **Continuous:** dispone de bloques relacionados con el análisis de sistemas continuos, como funciones de transferencia en el dominio s.
* **Discrete:** dispone de bloques relacionados con el análisis de sistemas discretos, como funciones de transferencia en el dominio z.
* **Sources:** dispone de los bloques de generación de señales, por ejemplo, para generar una función escalón o rampa.
* **Sinks:** dispone de bloques que sólo reciben señales, sin producir una salida, por ejemplo, para poder visualizar la salida de un sistema.
* **Math operations:** dispone de bloques para realizar operaciones matemáticas en el flujo de señales, por ejemplo, un sumador.

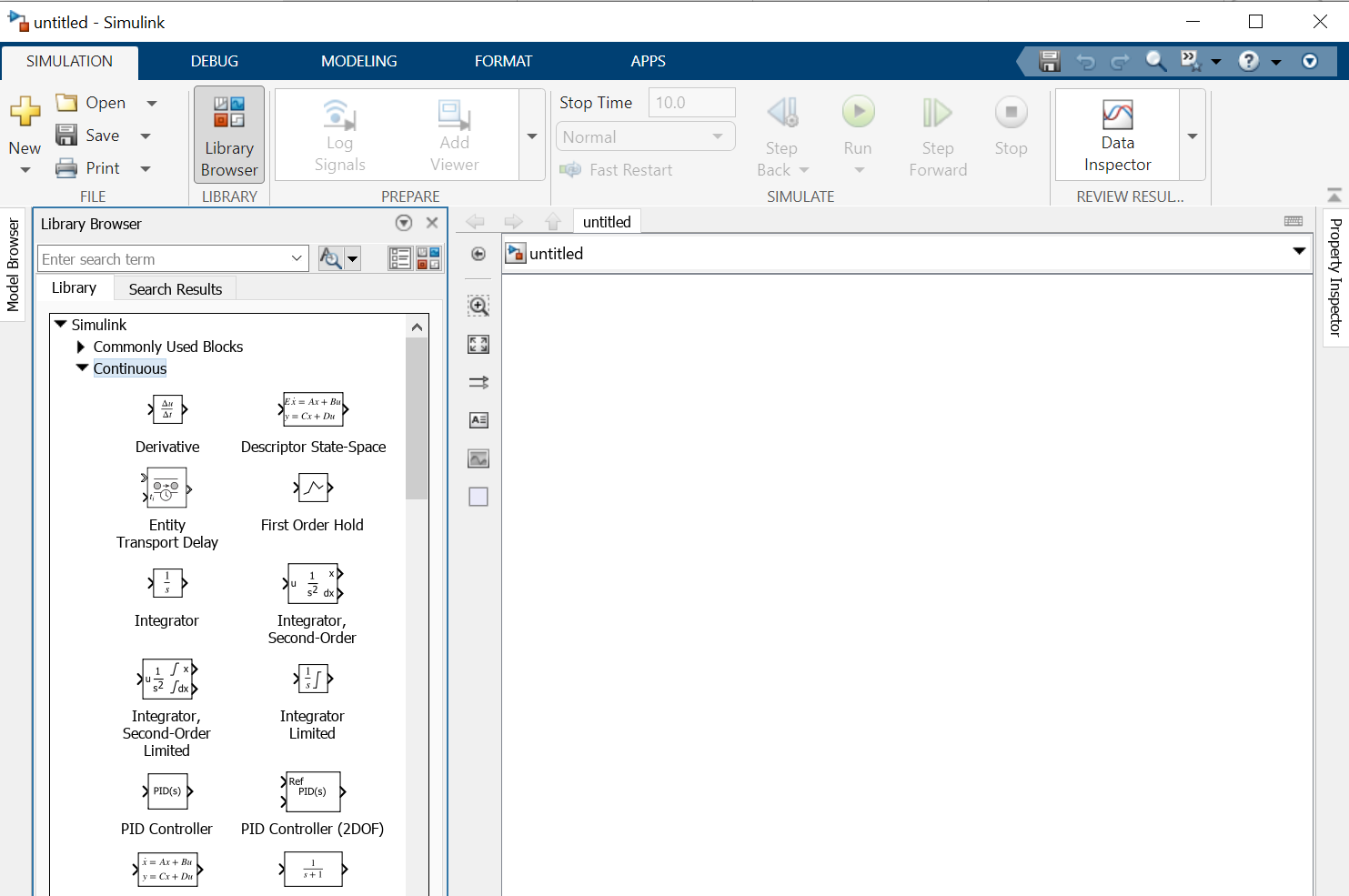


Figura 8. Biblioteca Simulink

**Construir un diagrama de bloques.**

Para dibujar un diagrama de bloques primero hay que crear un nuevo modelo: File -> New -> Blank\_Model. A partir de ahí sólo se requiere la utilización del ratón con el que se pueden arrastrar y pegar los bloques de las distintas bibliotecas. Cuando el elemento se encuentre sobre la ventana de trabajo soltando el botón del ratón éste queda fijo. Mediante este procedimiento de pinchar, arrastrar y soltar se pueden seleccionar los elementos deseados para la construcción de un determinado diagrama de bloques.

A continuación, se muestra un ejemplo de un diagrama de bloques sencillo con un integrador y una señal de entrada escalón unitario.

**Elementos:**

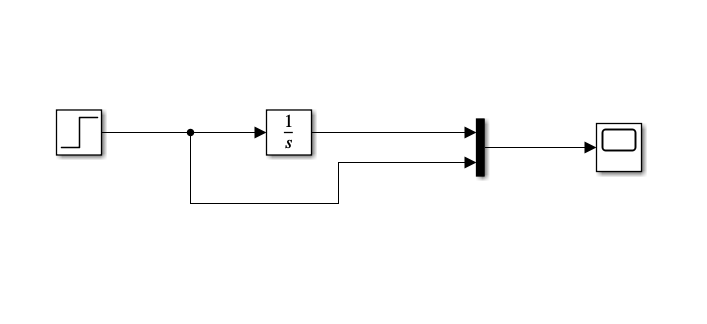
Entrada escalón (*Sources -> Step*)

Visor (*Sinks -> Scope*)

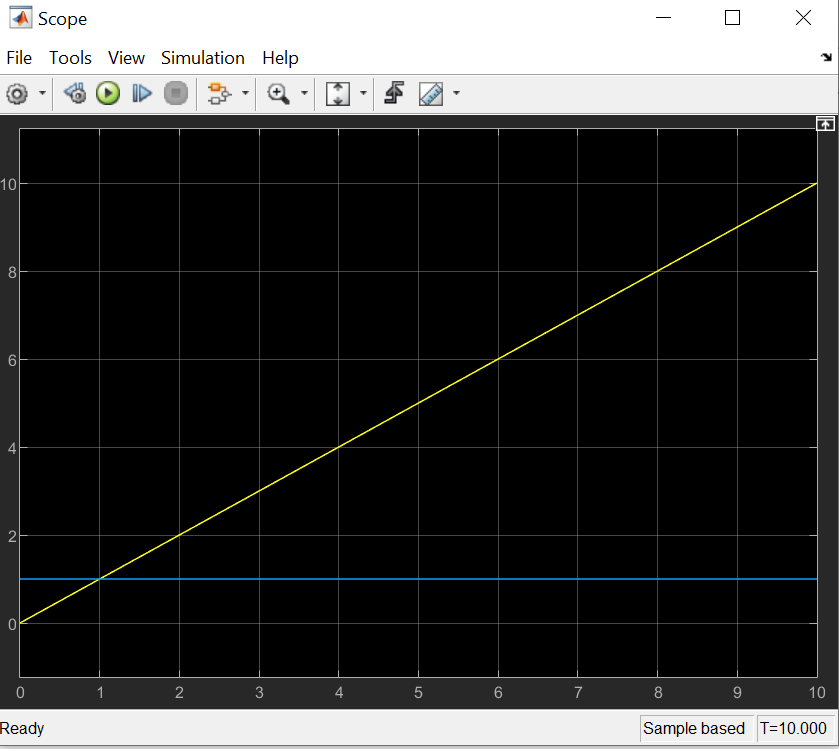
Integrador (*Continuous –> Integrator*)

Multiplexor (*Signal Routing -> Mux)*

**Esquema:**

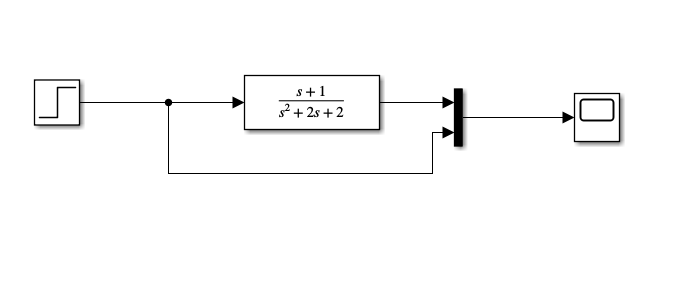


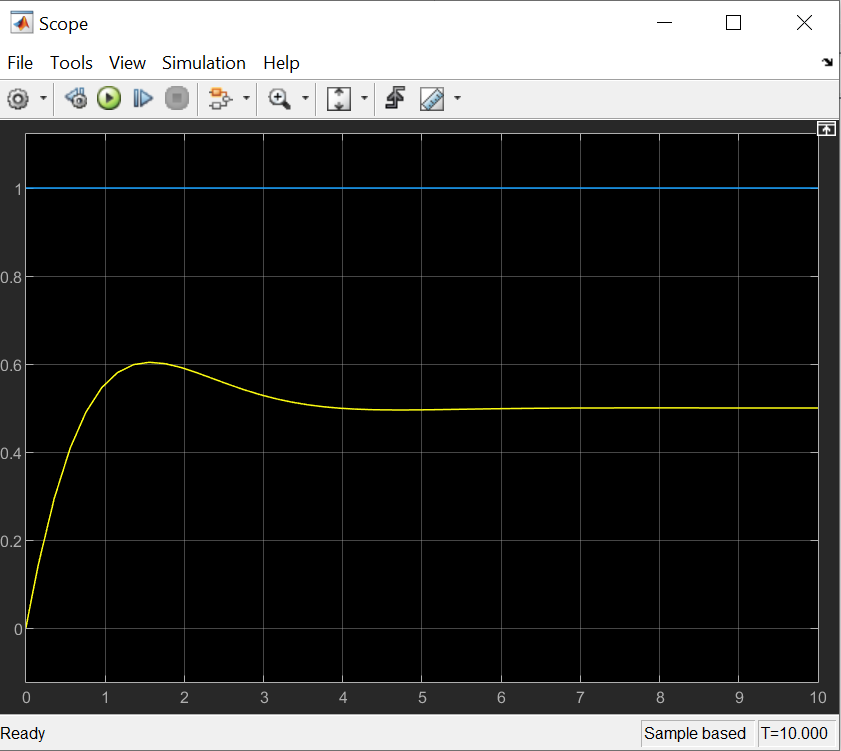
Después de haber escogido los elementos deseados, el siguiente paso es introducir los valores de esos elementos, para que empiecen a ser parte activa del diagrama de bloques que modela el sistema a implementar. Así, de acuerdo al tipo de elemento que representa cada bloque se podrá introducir un determinado valor.



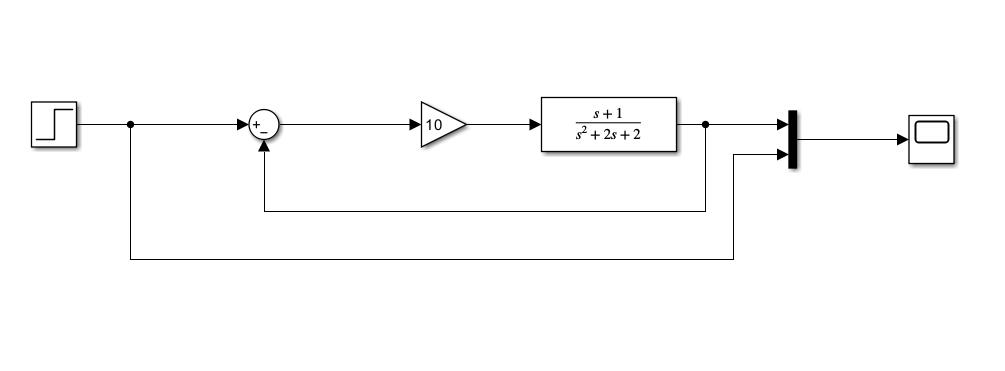
**Funciones de transferencia (lazo abierto y lazo cerrado)**

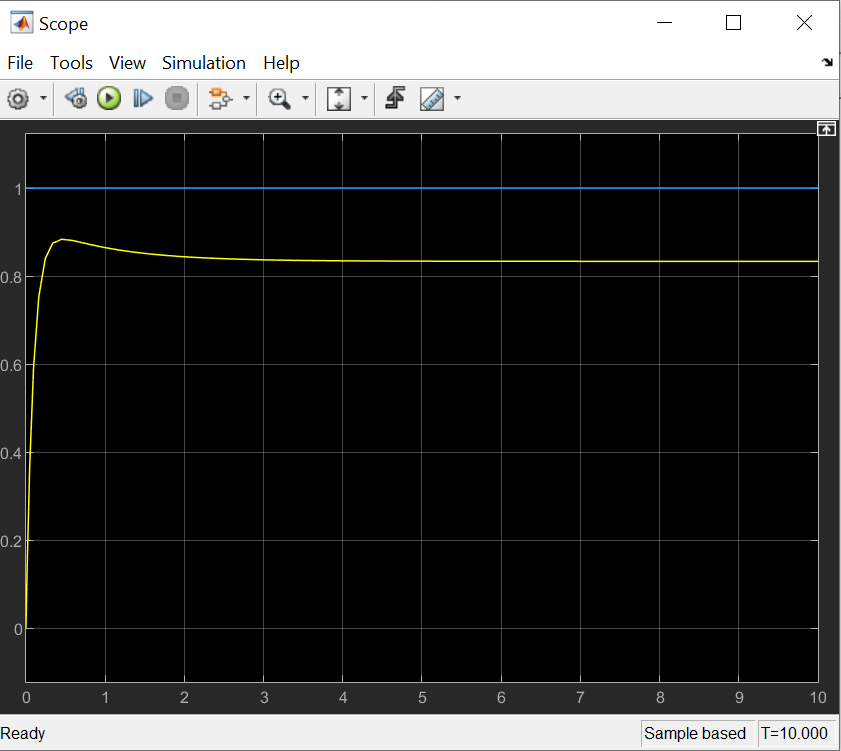
Para utilizar funciones de transferencia en SIMULINK se debe incluir un bloque Continuous -> Transfer Function. Como antes, los parámetros de dicho bloque se modifican pulsando dos veces sobre el bloque. En el cuadro de diálogo que aparece, se introducen los polinomios numerador y denominador, siguiendo la sintaxis seguida en la Sesión 1 de esta práctica. En el siguiente esquema se puede ver un ejemplo:



****

El ejemplo anterior, muestra un sistema en lazo abierto, es decir, no existe una realimentación de la salida a la entrada del sistema. Para introducir una realimentación se debe usar el bloque *Math Operations -> Sum*. Este bloque puede variar su signo si se trata de una realimentación positiva y negativa. Adicionalmente, se pueden introducir ganancias mediante el bloque *Math Operations -> Gain*. En el siguiente ejemplo se muestra un sistema con todos estos elementos:

****

****

**ACTIVIDAD**

Ejercicio 1

Calcula, utilizando los comandos de Matlab *feedback*, *series* y *parallel*, la función de transferencia equivalente de los siguientes diagramas de bloques:

*Nota: en alguno de los ejercicios se deberá realizar algún cambio previo para poder resolverlo*

*G1=3*

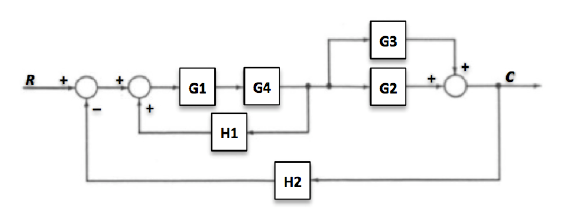
*G2=1/s*

*G3=1/(s+3)*

*G4=2*

*H1=1*

*H2=1/2*

**

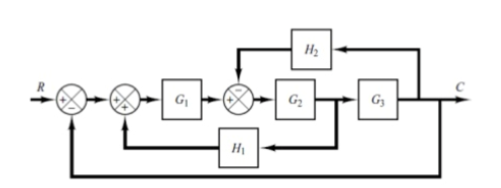
G1=2

G2=1/(s+1)

G3=1/s

H1=1/2

H2=1/2



**Ejercicio 2**

A partir de los diagramas propuestos en el ejercicio anterior. Se pide:

**Apartado a**

Simular su comportamiento en Simulink y obtener la salida se los sistemas ante una entrada escalón unitario.

**Apartado b**

Simular el comportamiento del sistema reducido en Simulink (función de transferencia obtenida previamente) y comprobar si la reducción del sistema es correcta mediante la visualización de su salida ante entrada escalón.