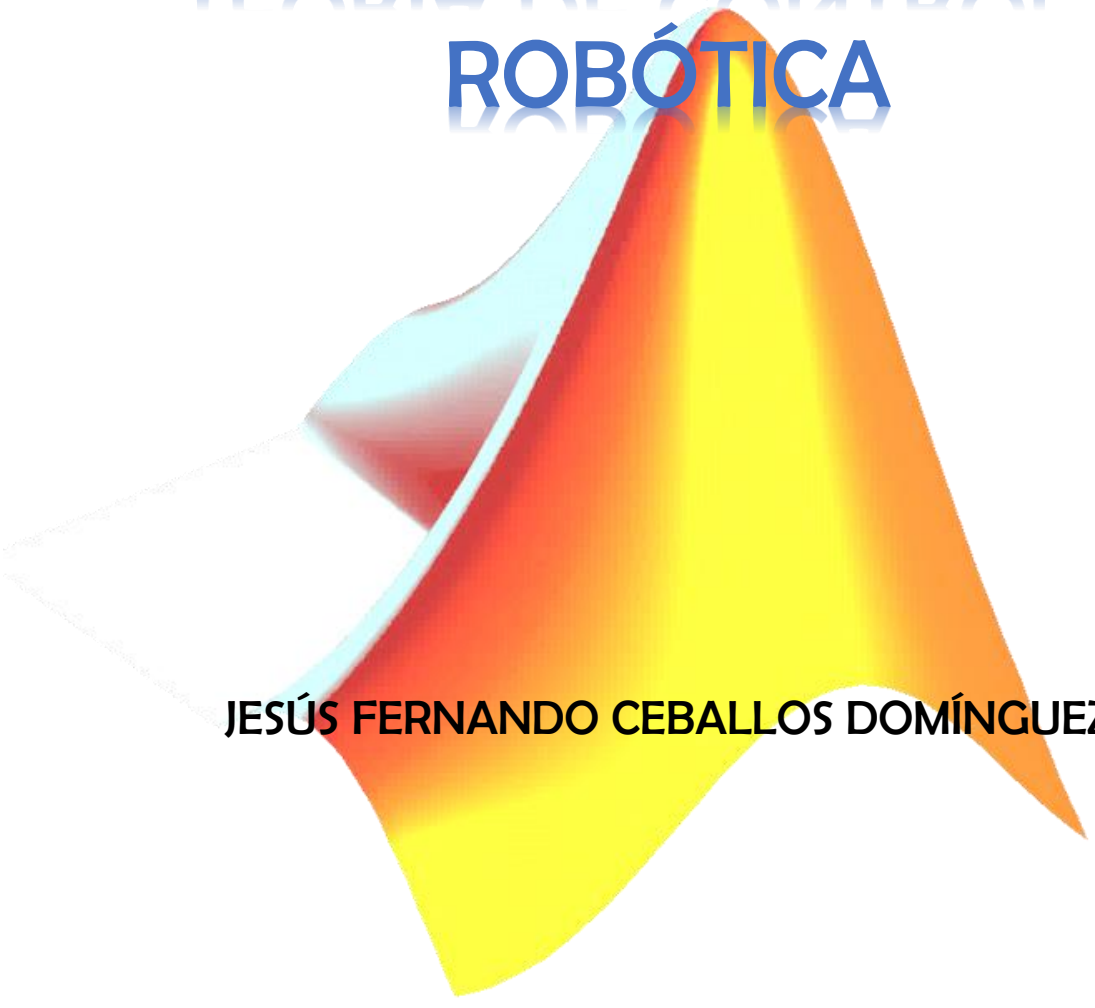
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

MANUAL DE MATLAB

TEORÍA DE CONTROL Y ROBÓTICA



JESÚS FERNANDO CEBALLOS DOMÍNGUEZ

ÍNDICE

INTRODUCCIÓN

El siguiente manual de Matlab está realizado con la ayuda de la herramienta *Help* del propio programa, en su versión 2017.

Me abstuve lo más posible de consultar otra fuente para realizar dicho manual, con la intención de no sugestionar mi criterio para resolver cada uno de los problemas según los temas que hemos visto en clase de Teoría de control y robótica.

El manual está escrito y ejemplificado (mediante capturas de pantalla y ejercicios) con la intención de que incluso personas ajenas al uso del programa pueden entenderlo en poco tiempo. Los pasos para resolver cada una de las situaciones (temas del curso vistos hasta ahora) están desglosados en la mayor cantidad posible, y con cada una de las indicaciones necesarias para trabajar en el Editor de Matlab y en Simulink, si así se requiere.

En cualquier momento se puede regresar al índice del manual, para ir a otro capítulo del manual, sin necesidad de estar desplazándose por todo el documento. Basta con dar clic al vínculo **ÍNDICE** en la esquina inferior izquierda de cada página para regresar.

En cada página del manual, se inserta un retorno rápido al índice del documento: “Índice”, esto con la finalidad de hacer más rápida la navegación dentro del mismo.

Espero que el Manual sea lo más entendible para cualquier persona que lo consulte en un futuro.

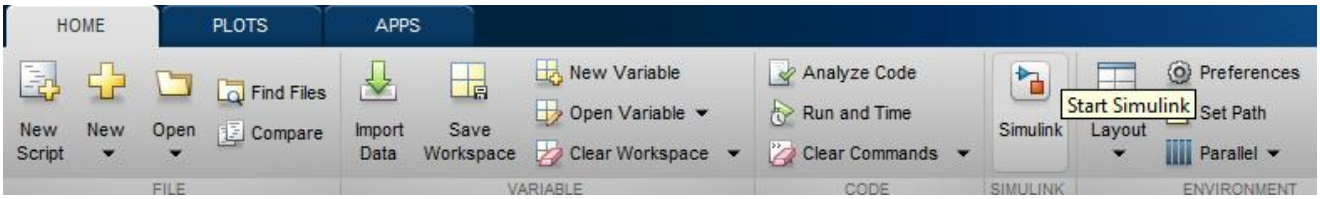
Jesús Fernando Ceballos Domínguez

DIAGRAMAS DE BLOQUES Y MODELO DE SISTEMAS EN MATLAB SIMULINK

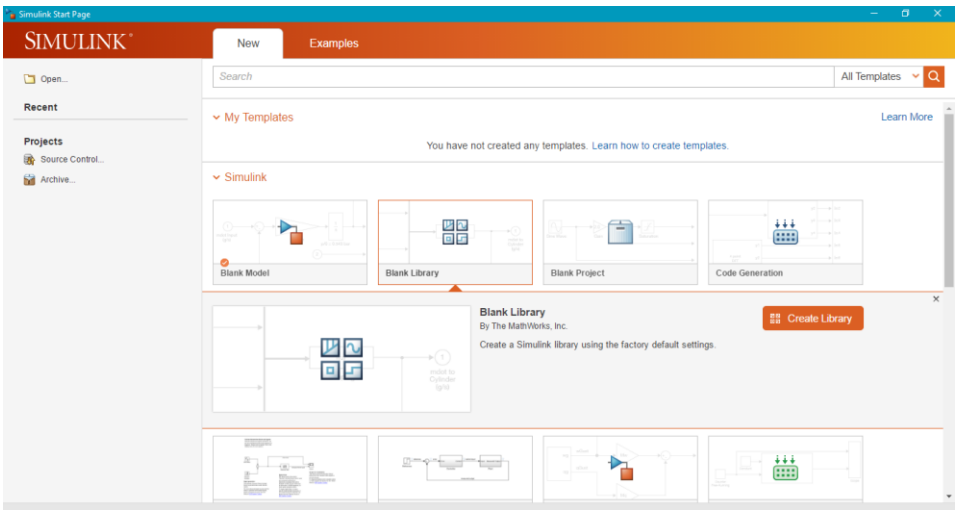
Los Diagramas de bloques son representaciones que permiten desarrollar esquemas para comprender más fácilmente las operaciones de control en el sistema, representando pictóricamente la función de cada elemento físico de dicho sistema.

Simulink es utilizado para diseñar y modelar sistemas que pueden ser físicos o Diagramas de bloques, con el fin de representarlos de manera gráfica y no solo matemática, aunque con Matlab los modelos que creemos en Simulink pueden ser resueltos de manera matemática.

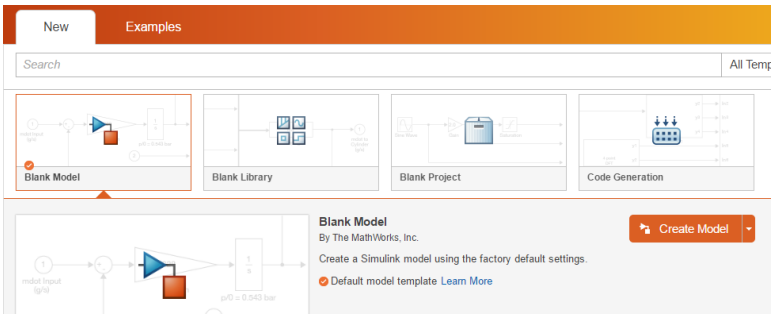
1.Para realizar un diagrama de bloques lo primero que hay que hacer es abrir **Simulink**, presionando el botón **Start Simulink** de la pestaña Home, en la barra de herramientas de Matlab.



Cuando **Simulink** abra, nos arrojará la ventana que se muestra a continuación:

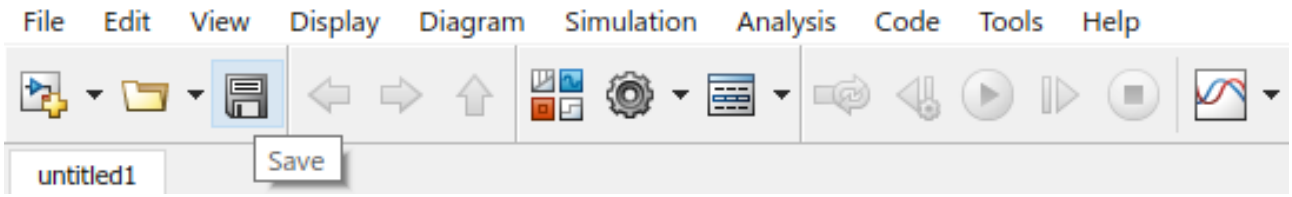


2.Lo que tenemos que hacer es seleccionar la opción **Blank model,** para crear nuestro modelo desde cero.

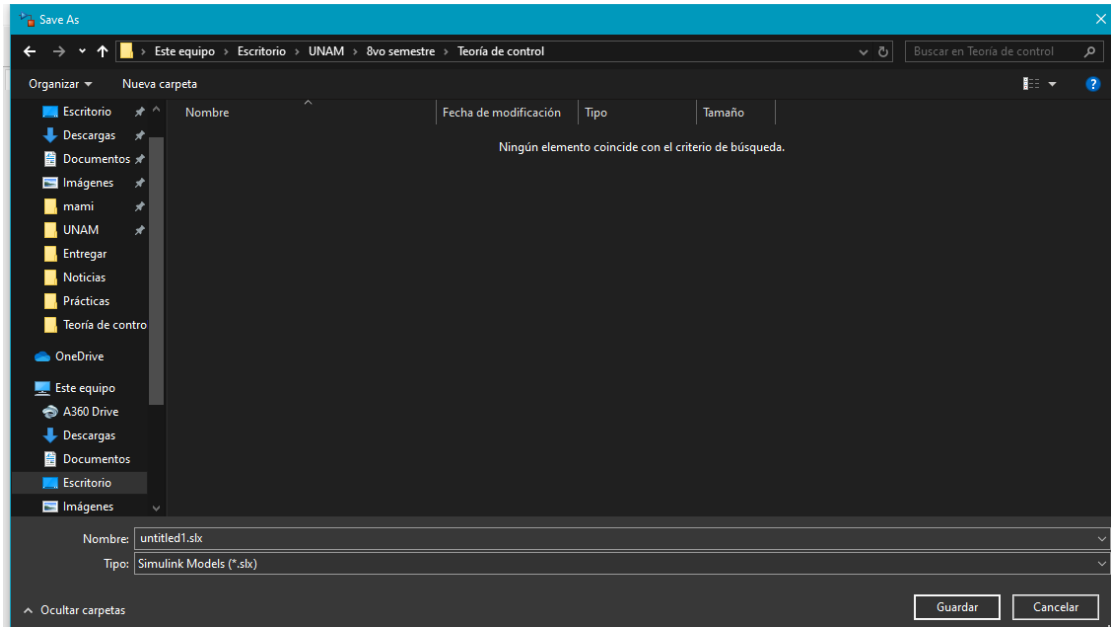


Al presionar **Blank Model** nos abrirá una ventana con un archivo nuevo de Simulink, el cual debemos guardar en nuestra ubicación de preferencia, yo recomiendo hacerlo antes que otra cosa.

Para guardar el archivo, tenemos que dirigirnos a la barra de herramientas y presionar el botón **Save:**



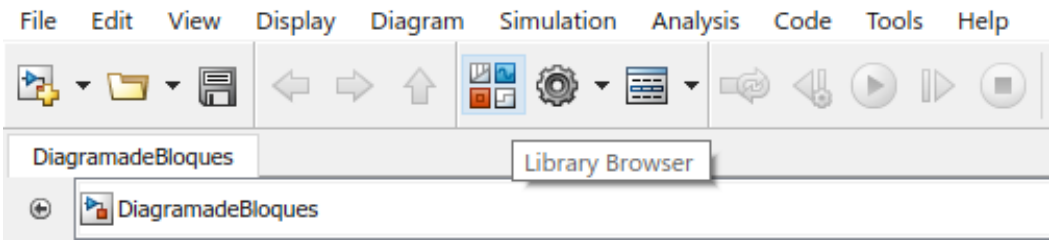
Al dar clic, nos abrirá una ventana que nos permitirá escoger la ubicación donde queramos guardar nuestro archivo. Podemos modificar el nombre del mismo, pero no la extensión (**.slx**).



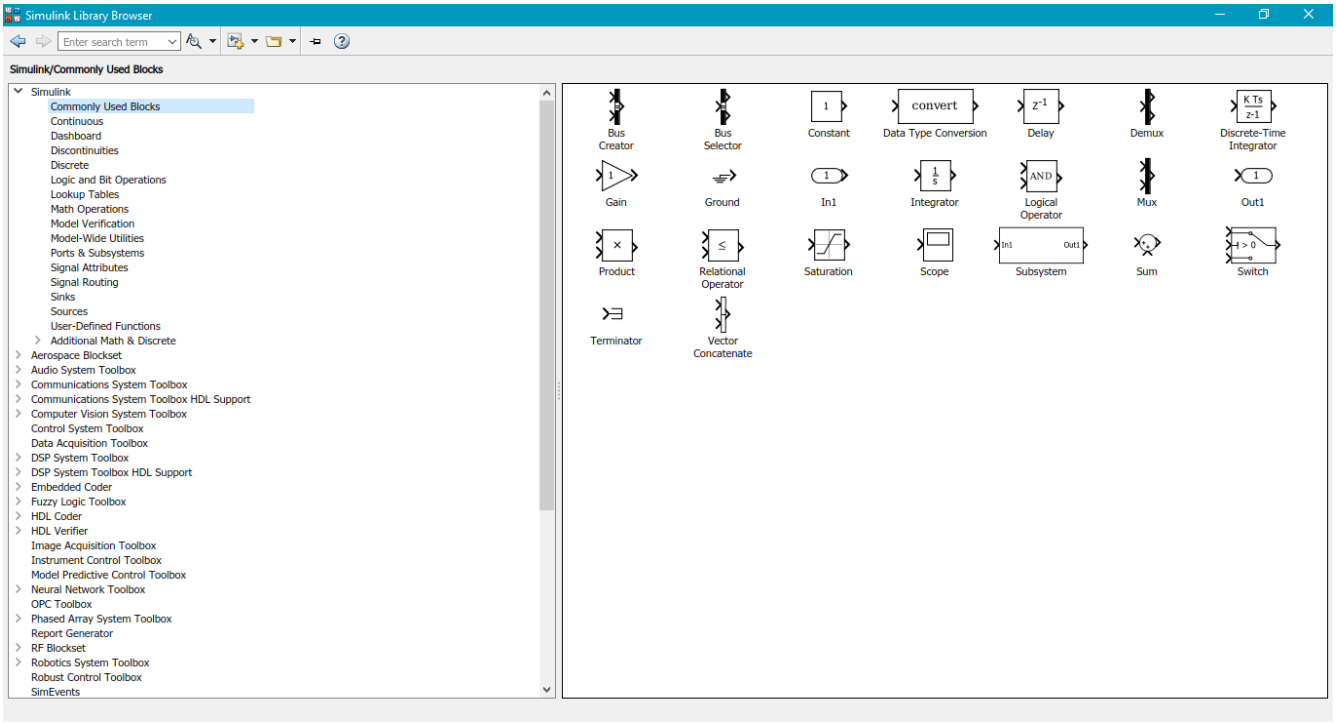
Es importante señalar, que Matlab solo permite nombres de archivo que no contengan espacios, pero si te permite utilizar caracteres especiales como el guion bajo.

3.Una vez asignado el nombre del archivo y una ubicación, procederemos a crear nuestro modelo.

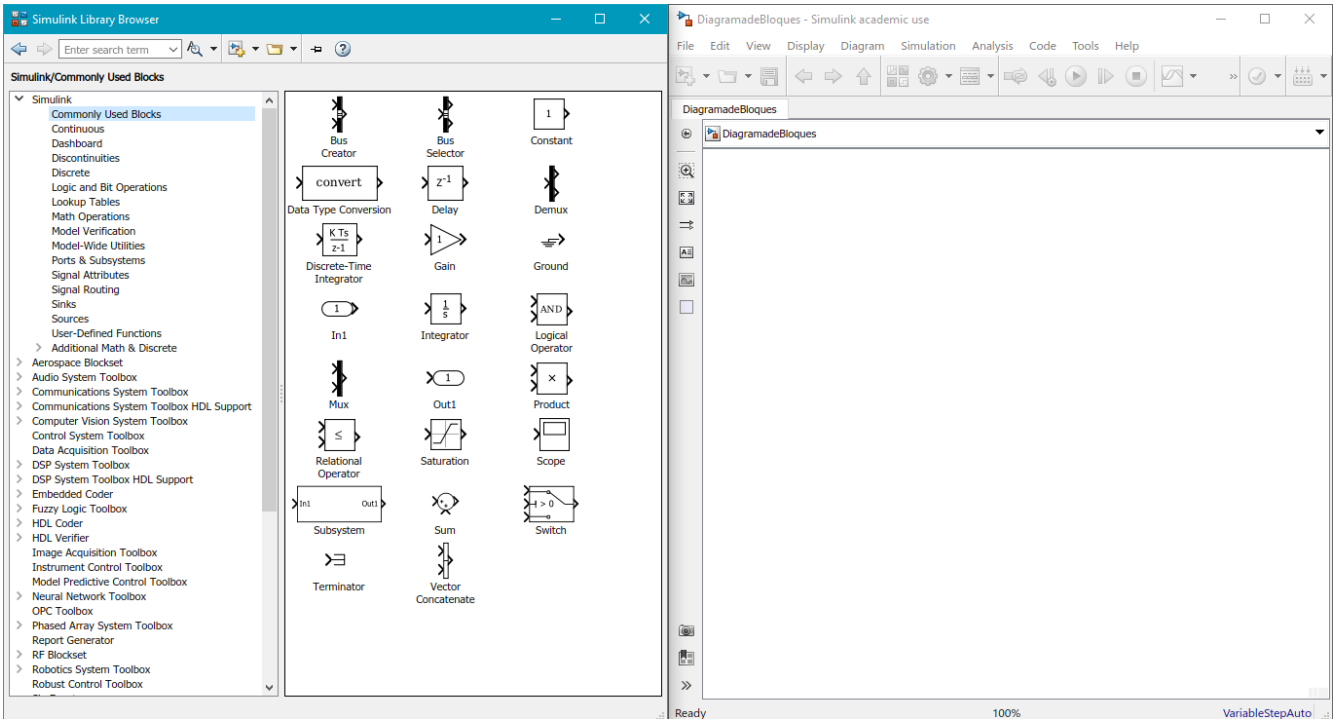
Para tener elementos que añadir a nuestro espacio de trabajo, hay que abrir la librería de Simulink, esto se hace, dando clic en el botón **Library Browser** que se encuentra en la barra de herramientas.



Al abrir la librería, nos arroja una ventana que muestra el explorador todos los comandos que podemos agregar a nuestro proyecto:



A partir de aquí, yo recomiendo dividir la pantalla de la computadora entre la ventana de la librería y la ventana de trabajo de Simulink, ya que la manera de agregar los elementos es arrastrándolos de la primera ventana a la segunda.

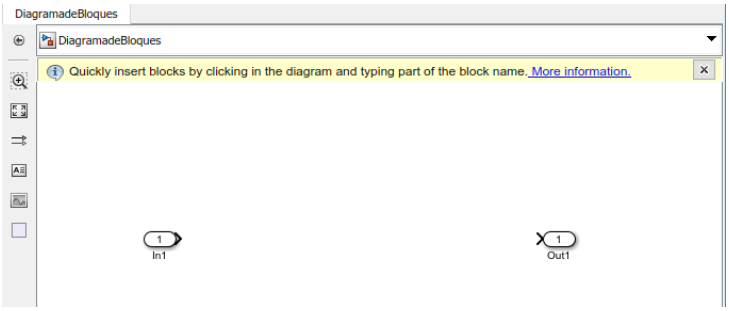


A la izquierda la ventana de la librería y a la derecha, la ventana de trabajo de SImulink.

A partir de aquí el diagrama de bloques dependerá de las necesidades que presentemos. Realizaremos un ejemplo sencillo para mostrar como se realiza.

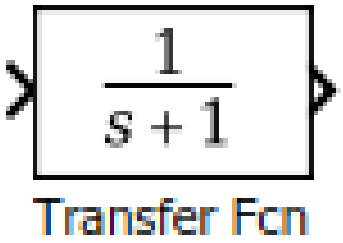
Utilizaremos las librerías **Commonly Used Blocks** y **Continuous**, para realizarlo.

4.Lo primero que se debe agregar es la entrada y la salida del diagrama, para tener un control sobre nuestro espacio. Estos se encuentran en la Librería **Commonly Used Blocks** con el nombre **In 1** para la entrada y **Out1** para la salida.



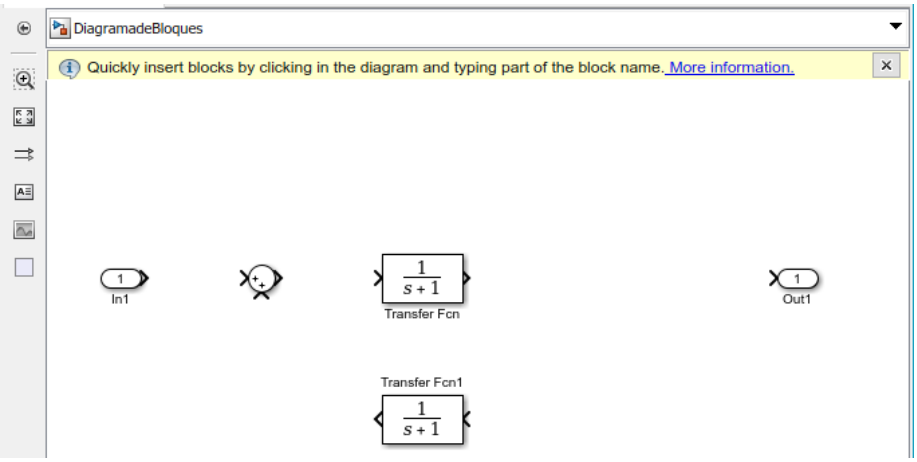
5.Luego se agregará un elemento de suma. Este se encuentra en la librería **Commonly Used Blocks**, con el nombre **Sum**

6.Para continuar con nuestro diagrama, se agregarán dos funciones de transferencia, de las cuales una será de retroalimentación (en clase las vimos como H(s)). Las funciones de transferencia se encuentran en la librería **Continuous** con el nombre de **Transfere Fcn.**

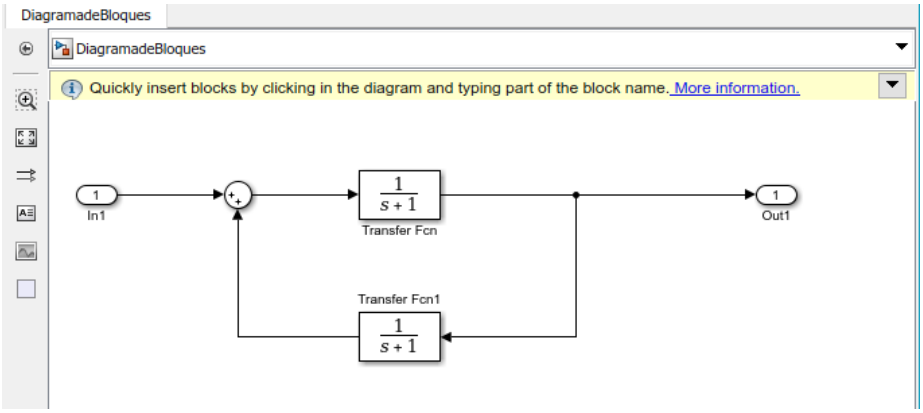


Para hacer que una de las funciones de transferencia sea de retroalimentación, se tiene que rotar hasta que el sentido de la función sea el contrario a la otra, tal como se muestra en la figura siguiente. Esto se logra seleccionando el elemento y posteriormente utilizar el comando **Ctrl + R** para rotarlo.

El comando **Ctrl + R** nos permite rotar los elementos en 4 direcciones, que tienen 90° entre cada una de ellas, hay que oprimirlas cuantas veces sean necesarias para girar el elemento según nuestras necesidades.

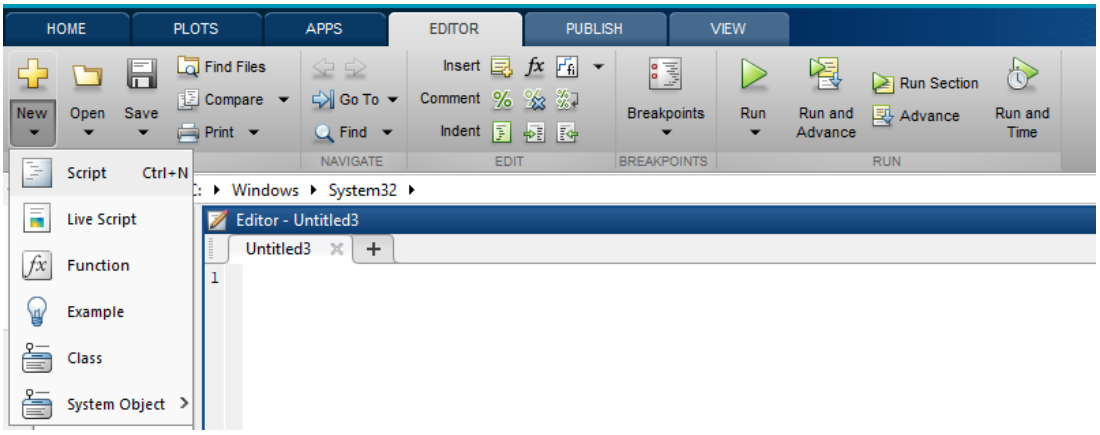


7.Ahora solo falta unir el diagrama mediante líneas. Esto se consigue al posicionar el cursor sobre una de las flechitas que contiene cada elemento y arrastrándolas hasta que toque al elemento siguiente al que lo deseamos unir.



Una vez que nuestro diagrama está listo, guardamos los cambios, presionando en el botón **Save** y abrimos nuevamente Matlab, para proceder a obtener la función transferencia total de nuestro diagrama de bloques.

8.Creamos un archivo nuevo en Matlab: con el botón **New** que se encuentra en la pestaña **EDITOR** de la barra de herramientas. Esta nos arroja una lista de los archivos que podemos crear, a continuación, elegimos la opción **Script.**

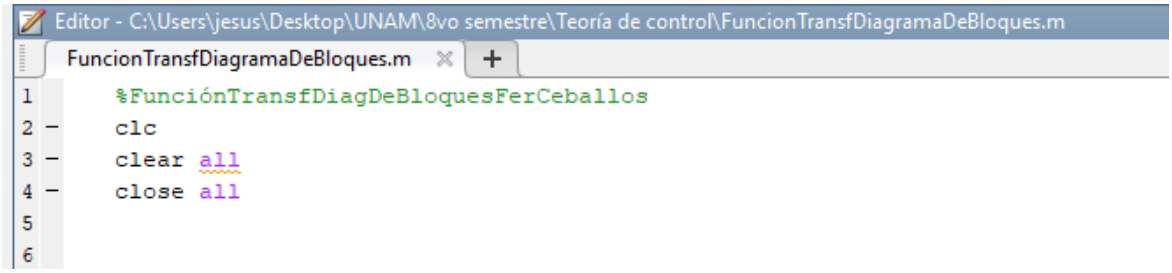


Igual que en el caso de Simulink, recomiendo guardar el archivo, antes que nada, para ir guardando nuestro progreso y no correr el riesgo de perder nuestro avance.

Antes de continuar, me gustaría definir que es lo que se conoce como función transferencia, ya que los usamos en nuestro diagrama de bloques, y vamos camino a encontrar la función transferencia total de nuestro sistema:

Una función transferencia es un modelo matemático que a través de un cociente relaciona la respuesta de un sistema de salida (modelado) con una señal de entrada (también modelada).

9.Continuando con nuestro trabajo en Matlab, procederemos a limpiar el programa de otros proyectos, mediante los comandos clc, clear all y close all:

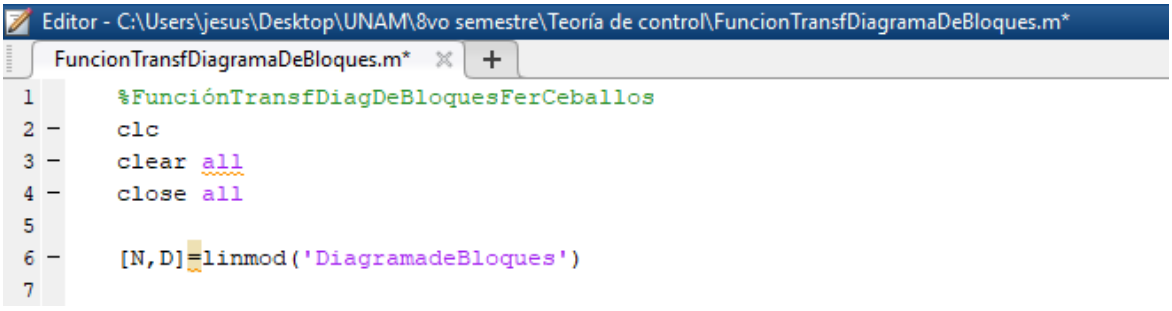


10.Tomando en cuenta la definición de una función transferencia, está claro que para que haya un cociente se necesita de un numerador y un denominador. Por esa razón utilizaremos el comando **numden**, que se encarga de extraer el numerador y el denominador de una función y cuya sintaxis es la siguiente: **[N,D]**.

N es el numerador y **D** es el denominador.

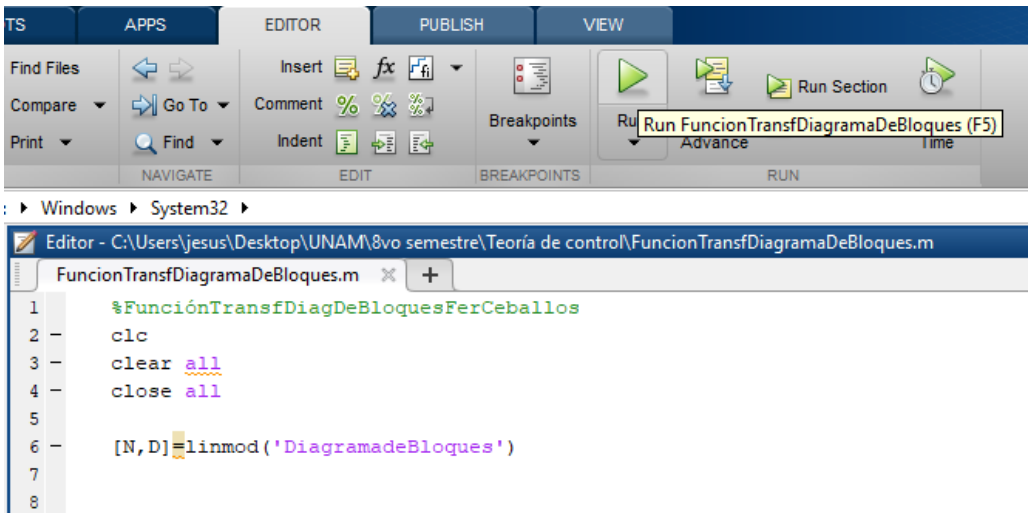
Seguido de este comando se escribe un signo igual (=) para posterior a éste escribir el comando **linmod** que se encarga de enlazar a nuestro programa con el archivo que elijamos, definido por (‘Nombre del archivo’).

A continuación, se ilustra la manera de introducir estas líneas de comando en Matlab:

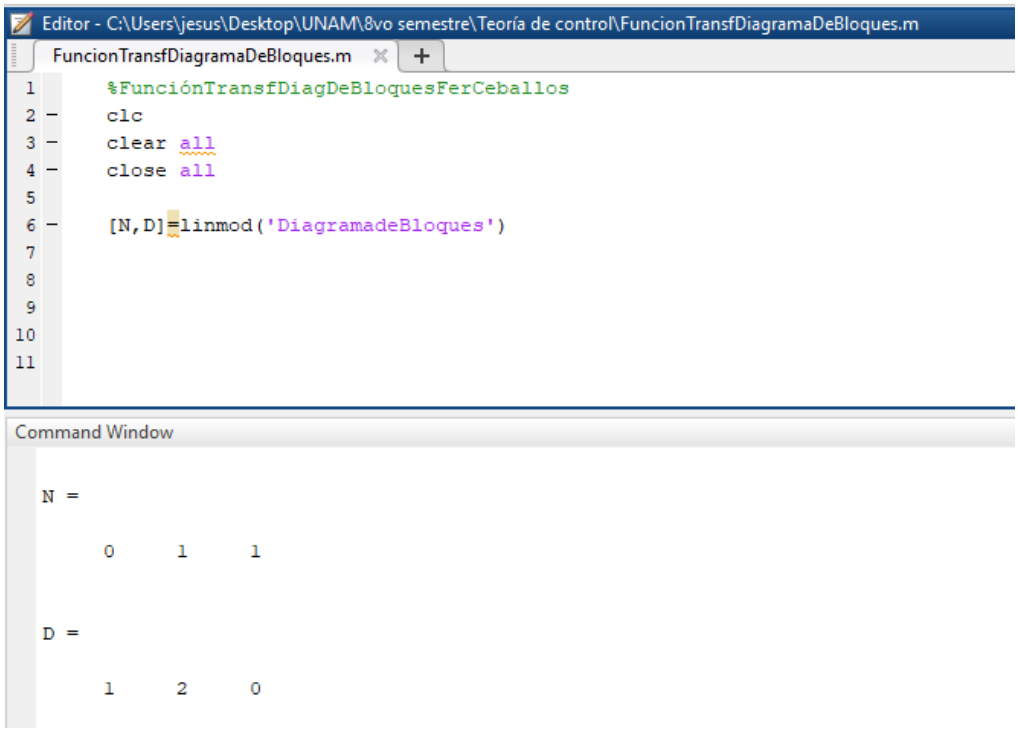


En este caso **DiagramadeBloques** es el nombre que yo le asigné a mi archivo de Simulink, en ese espacio entre los apóstrofos (‘ ‘) debes colocar el nombre que tú le hayas asignado.

11.Por último, para ejecutar nuestro programa, presionamos en el botón **Run** que se encuentra en la pestaña **EDITOR** de la barra de herramientas.



El resultado que nos arroja nuestro programa es el siguiente:



Como podemos darnos cuenta, la presentación de los resultados no es muy similar a un cociente, para mejorar esto, añadiremos una línea más de comando: **FT=tf(N,D)**

FT lo utilizaremos para señalar que es una Función Transferencia

tf es el comando de función transferencia (en inglés *transfer function*)

N es el numerador que se extrajo del diagrama de bloques.

D es el denominador que se extrajo del diagrama de bloques.

Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control\FuncionTransfDiagramaDeBloques.m

FuncionTransfDiagramaDeBloques.m

```
%FunciónTransfDiagDeBloquesFerCeballos
clc
clear all
close all

[N,D]=linmod('DiagramadeBloques')
FT=tf(N,D)
```

Command Window

```
N =

     0     1     1


D =

     1     2     0


FT =

    s + 1
  ---------
  s^2 + 2 s

Continuous-time transfer function.
```

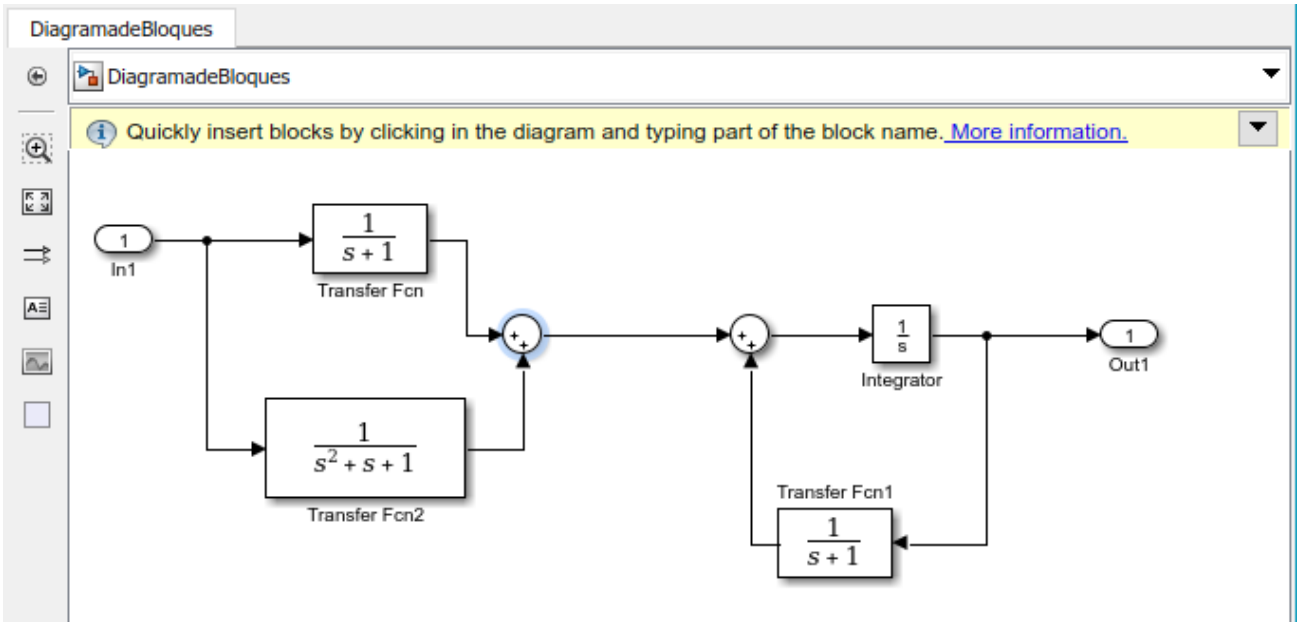
El resultado se aprecia mejor presentado como un cociente.

De esta manera, cada que realicemos un cambio en el archivo donde diseñamos nuestro diagrama de bloques con Simulink, afectará al resultado de la Función Transferencia que obtengamos con el programa hecho en Matlab.

De aquí en más, el diseño de múltiples diagramas de bloques es posible, solo se trata de explorar la librería que nos ofrece Simulink, para combinar los elementos y obtener distintos resultados.

A continuación, dejo otros dos ejercicios realizados con este programa:

Diagrama de bloques del Ejemplo 2



Función Transferencia del Diagrama de bloques del Ejemplo 2

Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control\FuncionTransfDiagramaD...

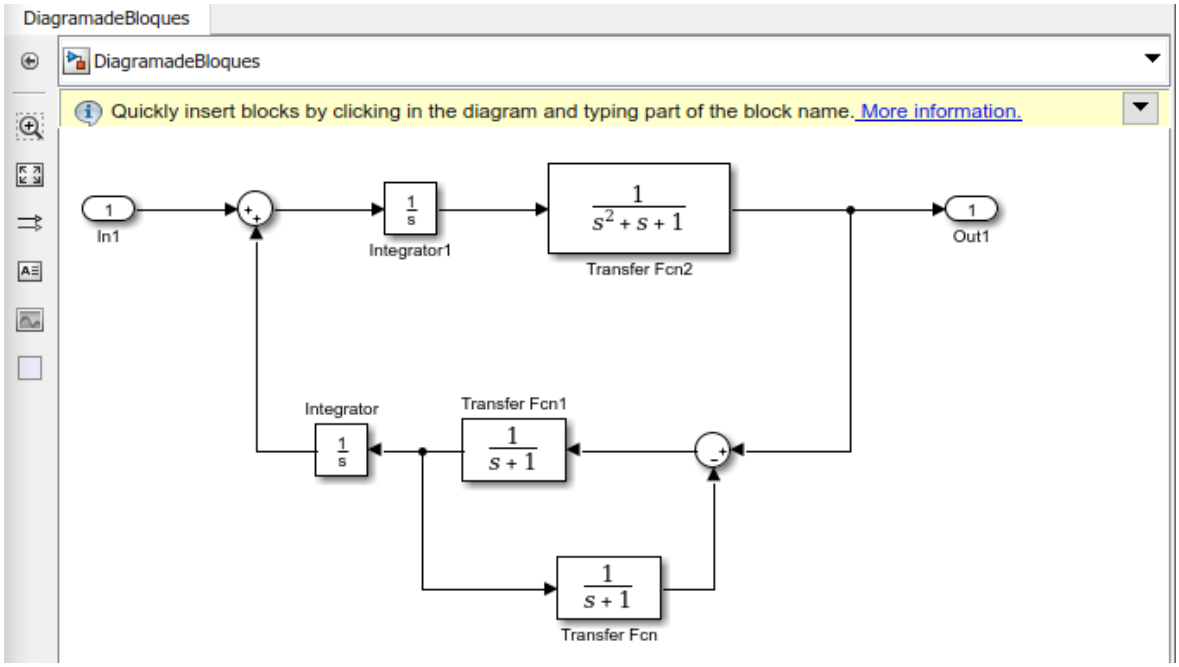
FuncionTransfDiagramaDeBloques.m

```
4 -     close all
5
6 -     [N,D]=linmod('DiagramadeBloques')
7 -     FT=tf(N,D)
8
9
10
```

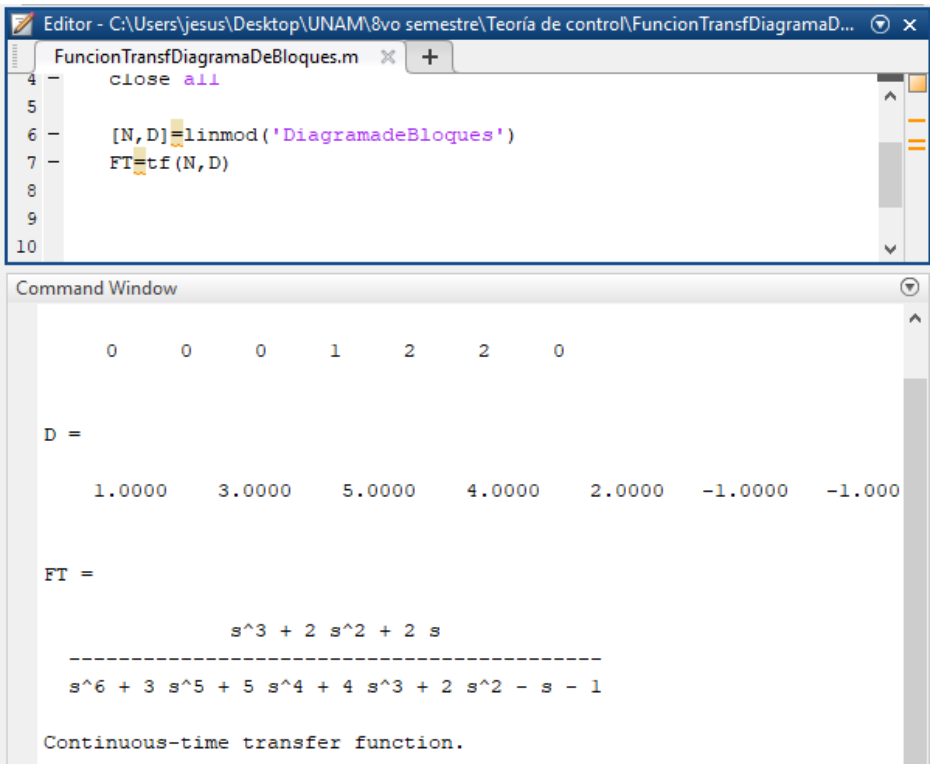
Command Window

```
N =

         0         0    1.0000    3.0000    4.0000    2.0000


D =

    1.0000    3.0000    3.0000    1.0000   -1.0000   -1.0000


FT =

        s^3 + 3 s^2 + 4 s + 2
  ---------------------------------
  s^5 + 3 s^4 + 3 s^3 + s^2 - s - 1

Continuous-time transfer function.
```

Diagrama de bloques del Ejemplo 3



Función Transferencia

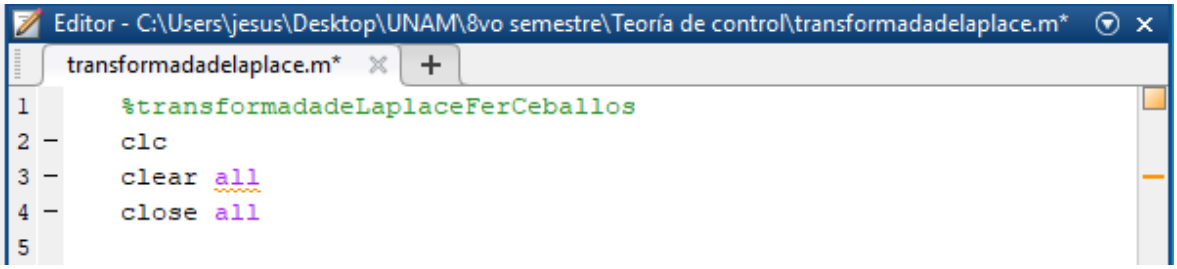


TRANSFORMADA DE LAPLACE

La transformada de Laplace es una herramienta matemática de gran alcance, ya que nos permite hallar soluciones de problemas complejos, de manera más simple. El procedimiento consiste en transformar las ecuaciones diferenciales complicadas en simples problemas algebraicos, de manera que las soluciones puedan ser obtenidas fácilmente.

Para resolver una Transformada de Laplace en Matlab, se deben seguir los siguientes pasos:

1.Primero que nada (después de crear nuestro script y guardarlo en nuestra ubicación de preferencia), limpiar la ventana de comandos mediante las instrucciones clc, clear all y close all



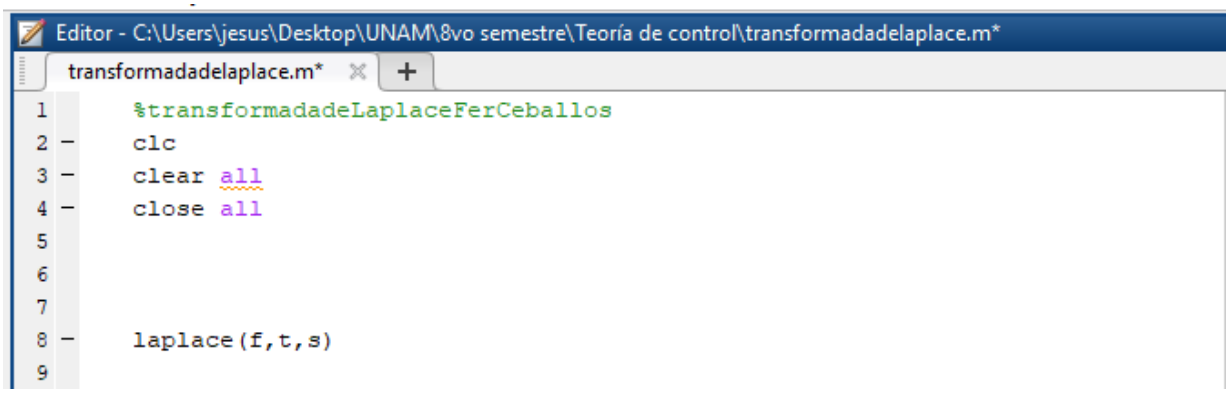
2.La sintaxis para ingresar los datos a una Transformada de Laplace es la siguiente: **laplace(f,var,transvar).** Los parámetros se describen a continuación:

f representa a la función que deseamos transformar.

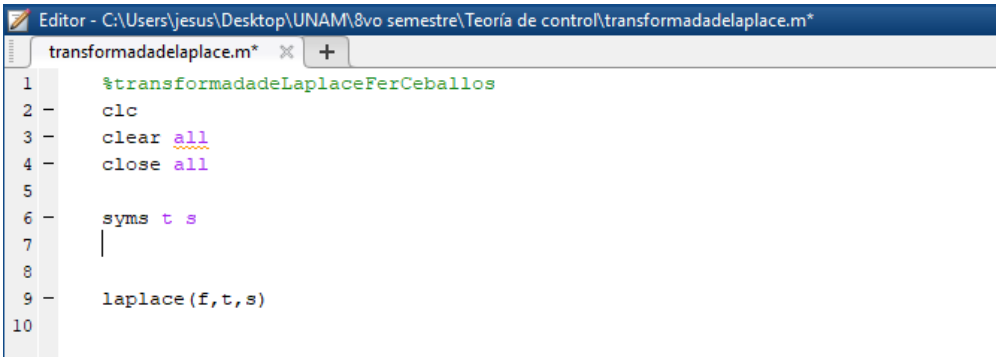
var es la variable a transformar. (En este caso será *t*)

transvar la variable de salida. (En este caso será *s*).

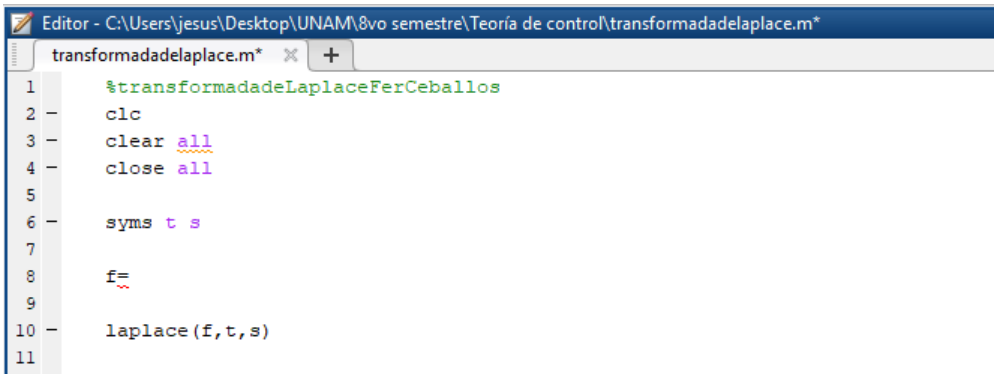
Se escribe esta sintaxis en nuestro editor **laplace(f,t,s)**, dejando un espacio considerable donde se declararán las variables y escribiremos nuestra función.



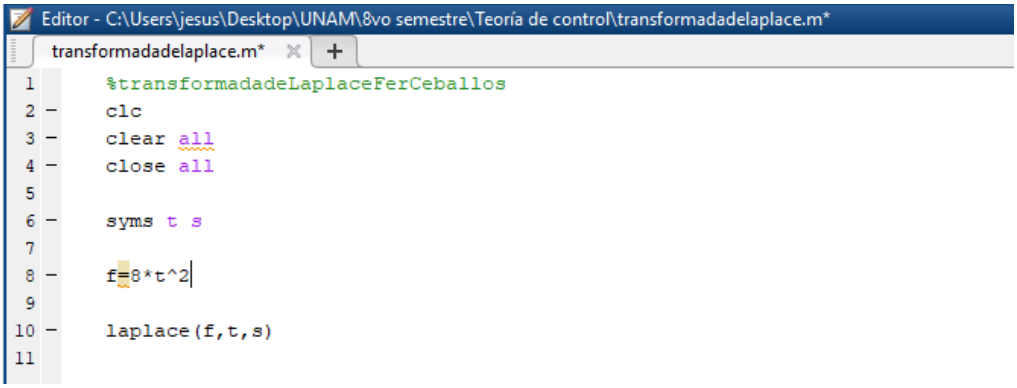
3.Declarar las variables simbólicas (*t* y *s*) para que el programa nos arroje resultados basados en estas variables, y no resultados numéricos.



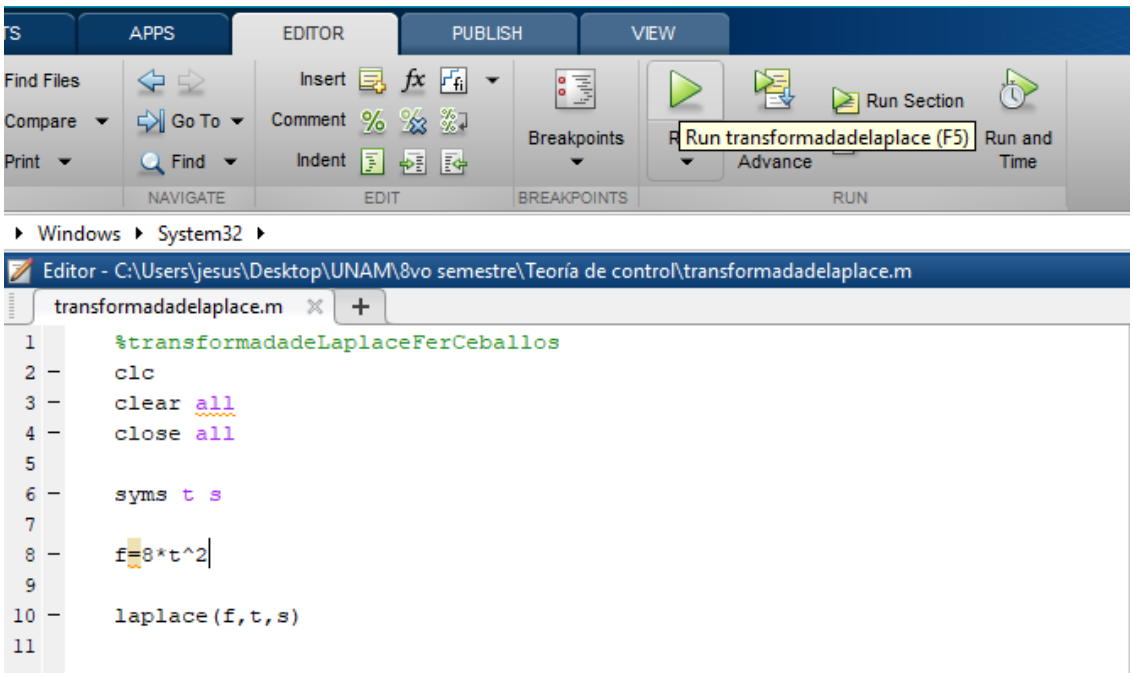
4.Ingresamos la función a evaluar. Al colocar "**f=**" podremos modificar nuestra función y de esta manera utilizar el mismo programa para distintas funciones.



En este caso, ingresaré una función sencilla, compuesta por una constante que multiplica a la variable elevada al cuadrado: **$8n^2$.**

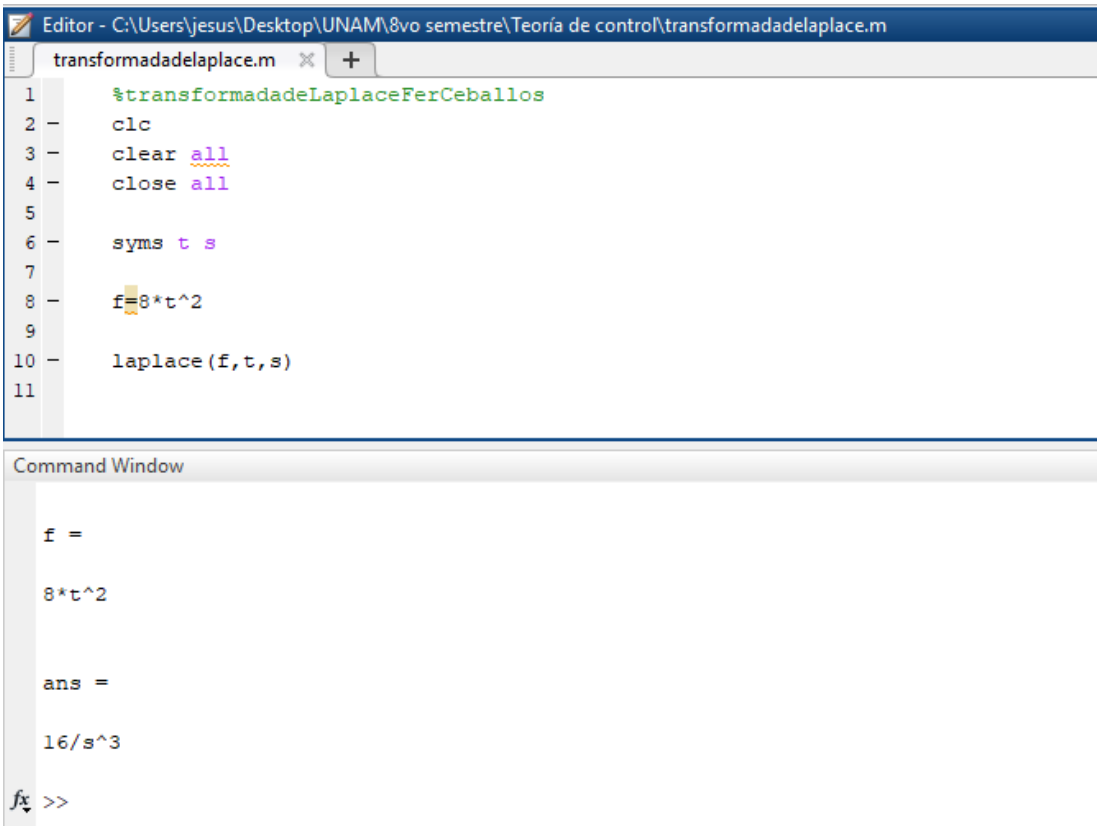


5.Por último, damos clic en el botón **Run** de la pestaña Editor, para correr el programa escrito:



Al darle clic en **Run** me envió un cuadro de diálogo que me solicitó cambiar la ubicación de mi archivo o agregar la carpeta al directorio de Matlab (esto segundo fue lo que hice).

Es cuestión de esperar a que nuestro programa corra y nos arroje el resultado de nuestra Transformada de Laplace:



Comprobamos el resultado utilizando la siguiente fórmula:

$$L\{t^n\} = \frac{n!}{s^{n+1}}$$

$$8L\{t^2\} = \frac{2!}{s^{2+1}} = 8\left(\frac{2}{s^3}\right) = \frac{16}{s^3}$$

Como era de esperarse, el resultado fue correcto.

Añadiré ejemplos más de funciones un poco más complejas, a fin de mostrar que las posibilidades con Matlab son muy grandes:

Ejemplo 2

Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control\transformadadelaplace.m

transformadadelaplace.m

```
1   %transformadadeLaplaceFerCeballos
2   clc
3   clear all
4   close all
5
6   syms t s
7
8   f=(t^2)/(4)
9
10  laplace(f,t,s)
11
```

Command Window

```
f =

t^2/4


ans =

1/(2*s^3)
```

Ejemplo 3

Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control\transformadadelaplace.m

transformadadelaplace.m

```
%transformadadeLaplaceFerCeballos
clc
clear all
close all

syms t s

f=exp(16*t^2)

laplace(f,t,s)
```

Command Window

```
f =

exp(16*t^2)


ans =

-(pi^(1/2)*erfc(-(s*1i)/8)*exp(-s^2/64)*1i)/8
```

Ejemplo 4

Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control\transformadadelaplace.m

transformadadelaplace.m | AntitransformadadeLaplace.m

```
%transformadadeLaplaceFerCeballos
clc
clear all
close all

syms w t s

f=sin(w*t)

laplace(f,t,s)
```

Command Window

```
f =

sin(t*w)


ans =

w/(s^2 + w^2)
```

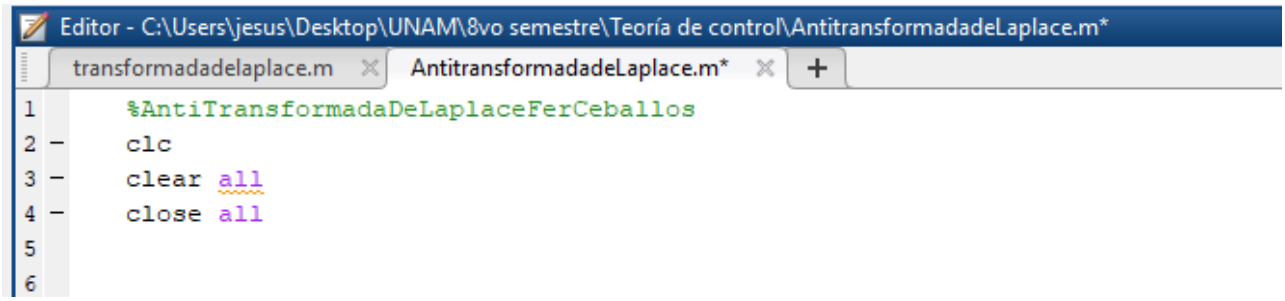
ANTI TRANSFORMADA DE LAPLACE

Básicamente, la anti transformada de Laplace, también llamada *transformada inversa de Laplace* es el procedimiento inverso de la transformada de Laplace.

El encanto de ésta, radica en que con ella podemos recuperar las soluciones de los problemas originales, que convertimos previamente en problemas algebraicos sencillos, con la transformada de Laplace.

Para realizar una anti transformada de Laplace en Matlab se deben seguir los siguientes pasos.

1.Primero que nada (después de crear nuestro Script y guardarlo en nuestra ubicación de preferencia), limpiar la ventana de comandos mediante las instrucciones clc, clear all y close all



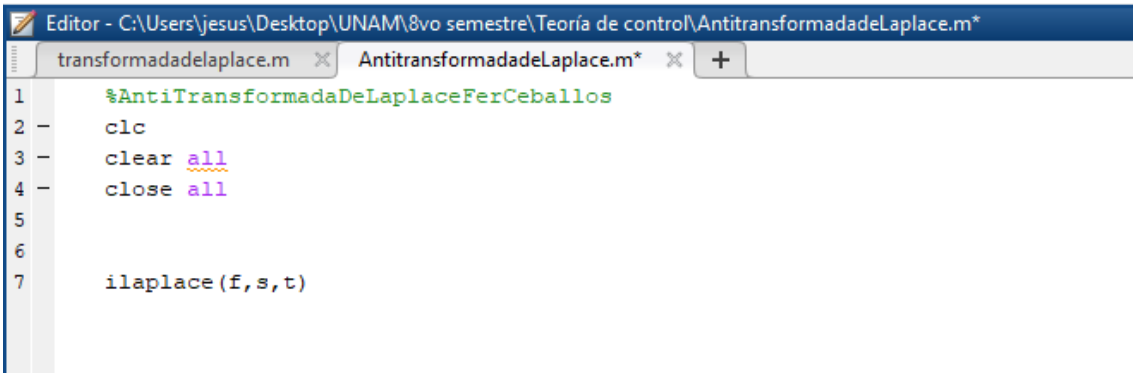
2.La sintaxis para ingresar los datos a una Anti Transformada de Laplace es la siguiente: **ilaplace(f,var,transvar).** Los parámetros se describen a continuación:

f representa a la función que deseamos transformar nuevamente a una función de tiempo.

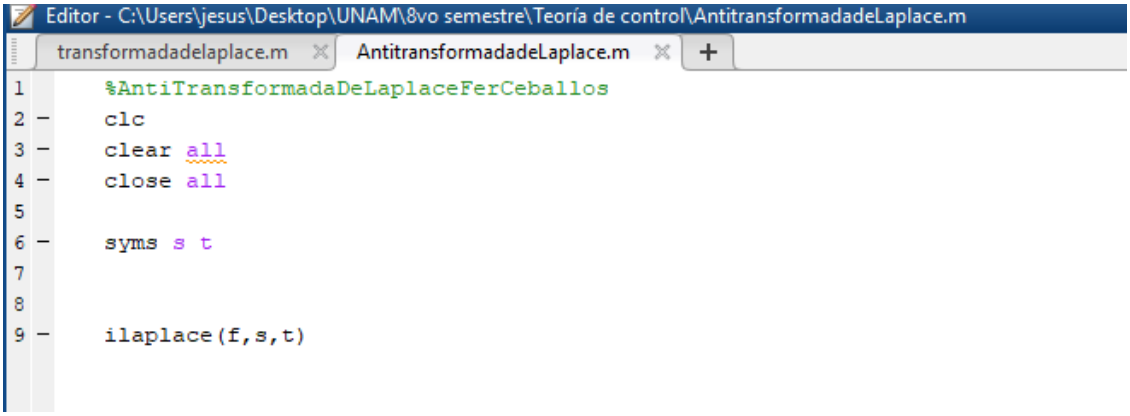
var es la variable de salida que regresará a ser función de tiempo, es decir, antes de habérsele aplicado la transformada de Laplace. (En este caso será *s*)

transvar es la variable de tiempo. (En este caso será *t*).

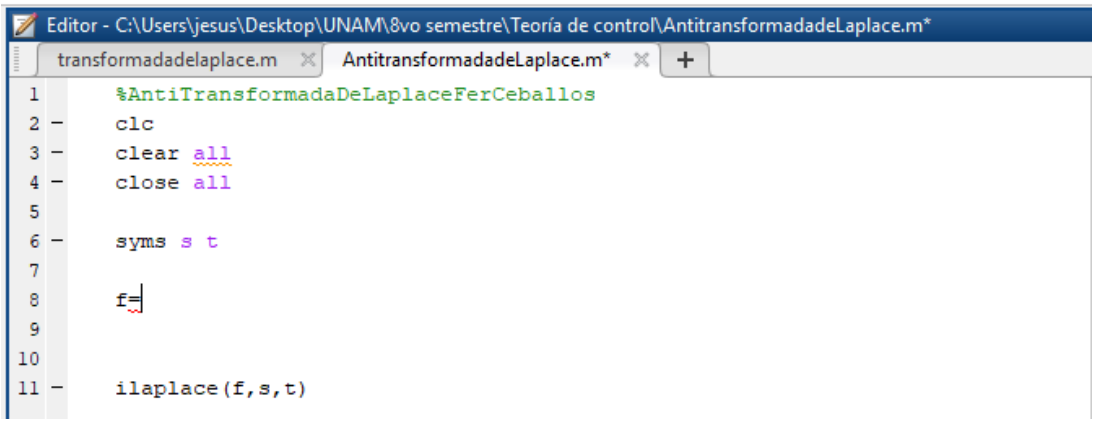
Se escribe esta sintaxis en nuestro editor **ilaplace(f,s,t)**, dejando un espacio considerable donde se declararán las variables y escribiremos nuestra función.



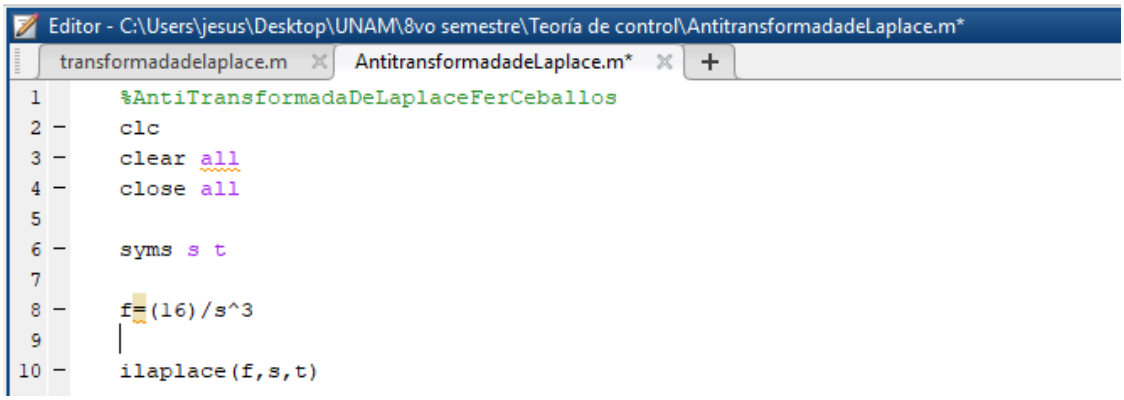
3.Declarar las variables simbólicas (*s* y *t*) para que el programa nos arroje resultados basados en estas variables, y no resultados numéricos.



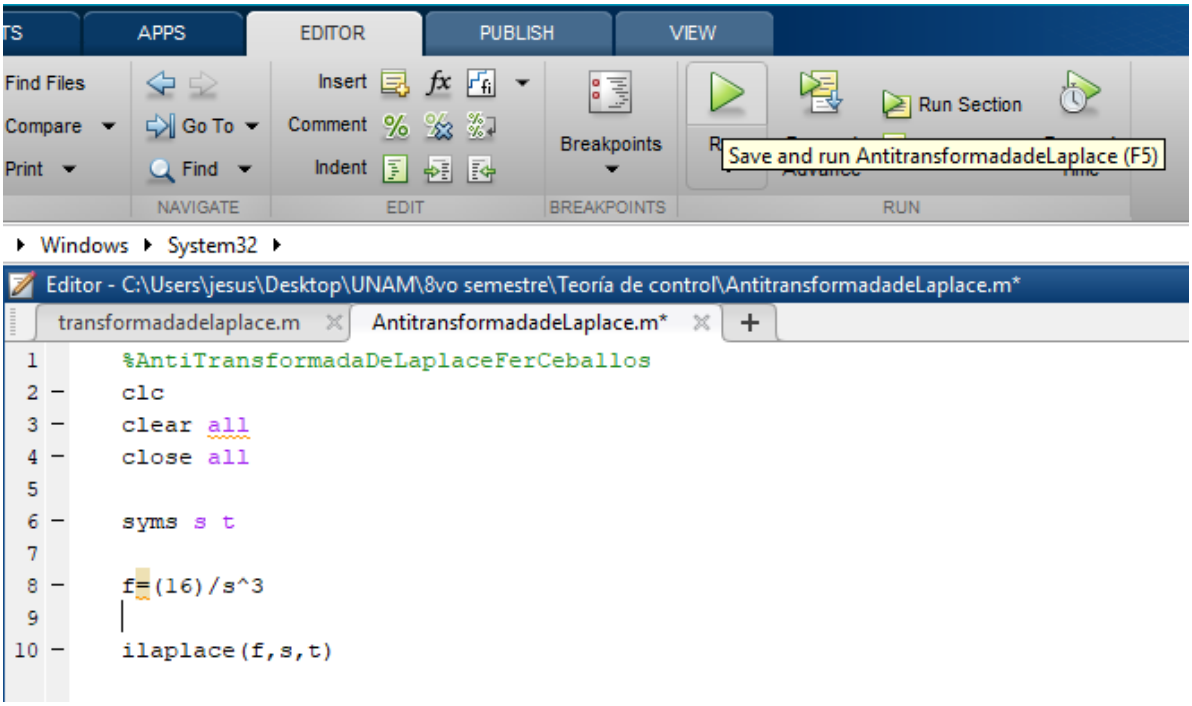
4.Ingresamos la función a evaluar. Al colocar "**f=**" podremos modificar nuestra función y de esta manera utilizar el mismo programa para distintas funciones.



Ingresaré el resultado de la transformada realizada en el manual de **Transformada de Laplace**, en el capítulo anterior: $\frac{16}{s^3}$ para comprobar que debería arrojarnos un resultado de $8t^2$.

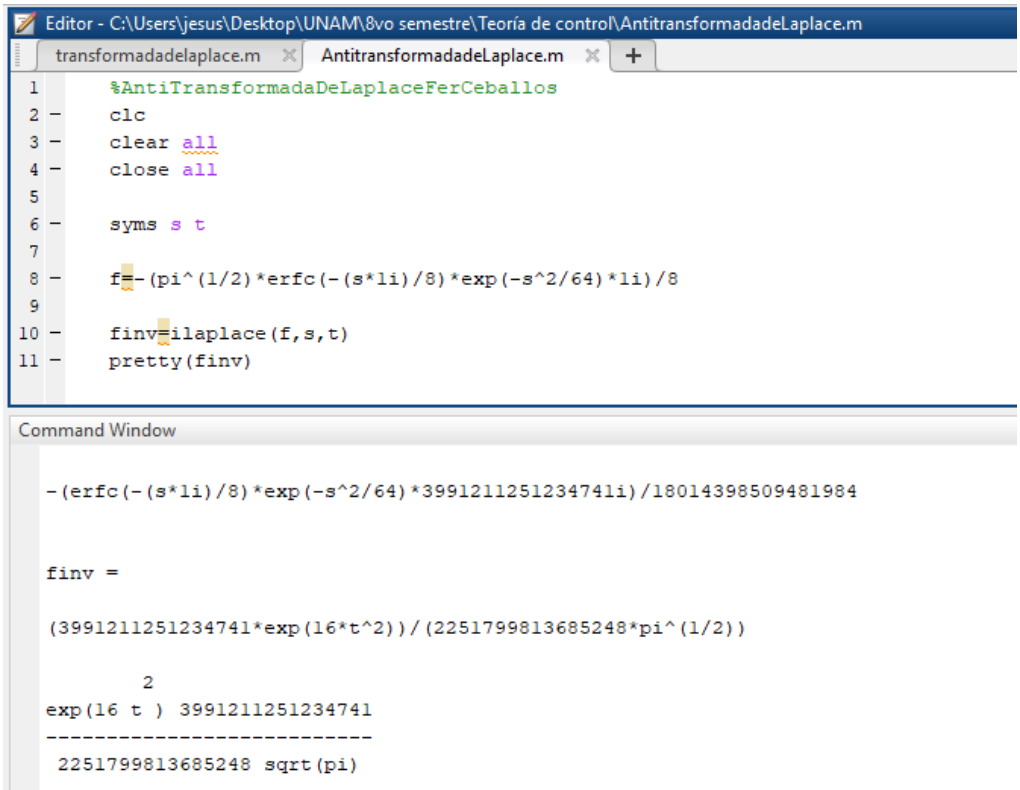


5.Por último, damos clic en el botón **Run** de la pestaña Editor, para correr el programa escrito:



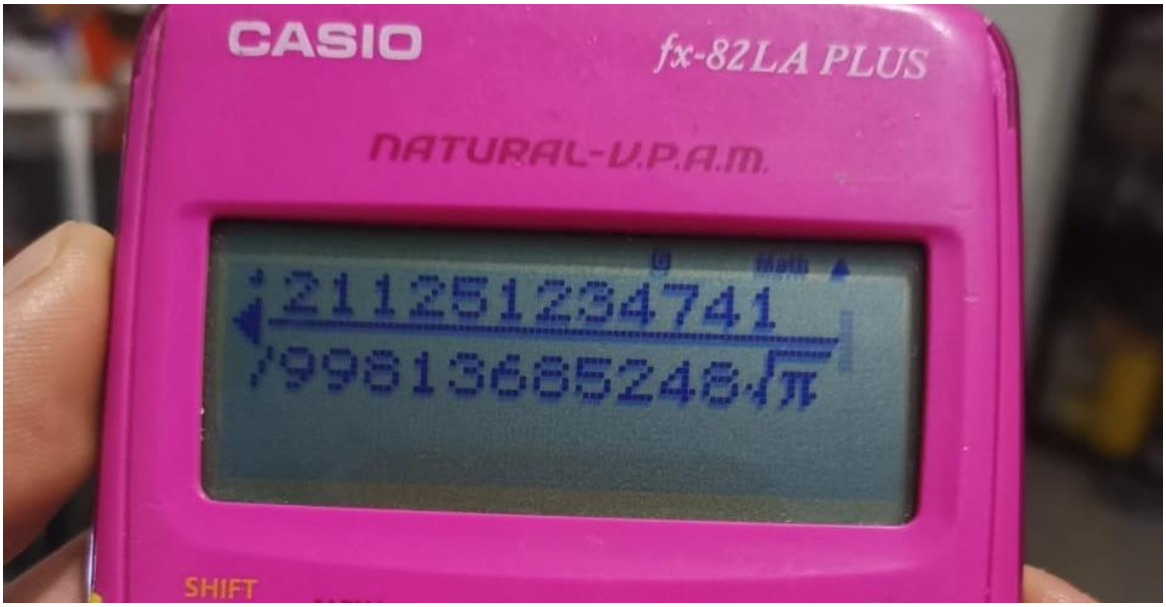
Efectivamente, el resultado fue el esperado: **$8t^2$.** De esta manera comprobamos que tanto nuestro programa de Transformada como el de Anti transformada de Laplace, funcionan adecuadamente.

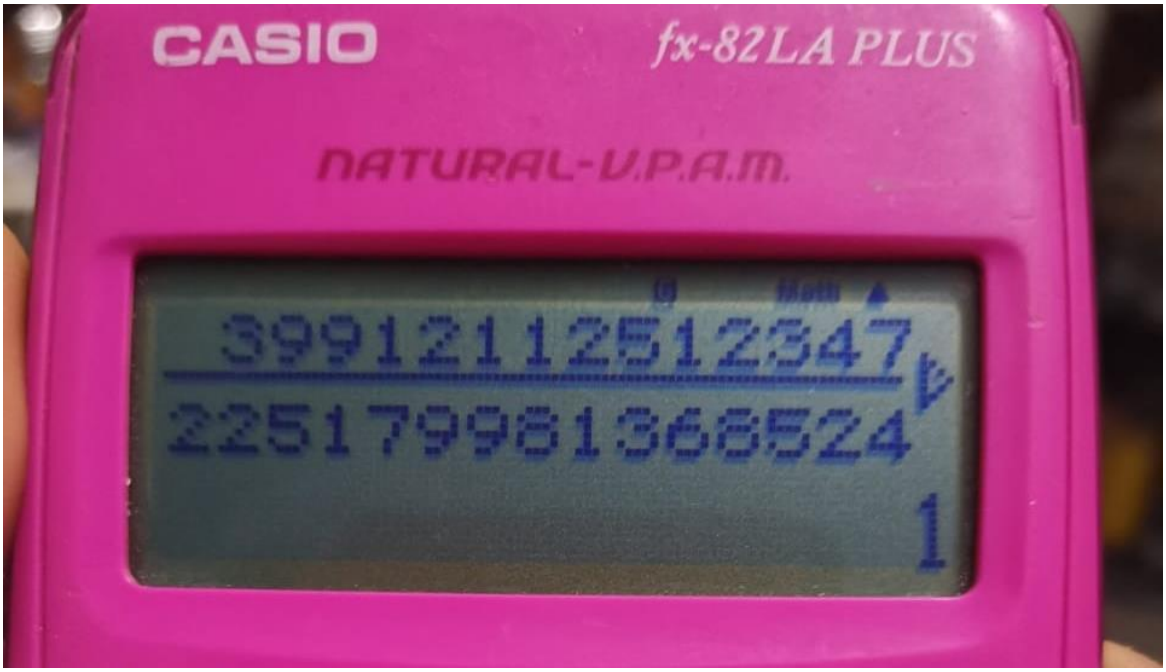
Añadiré las inversas de la transformada de Laplace utilizados en el capítulo anterior, para corroborar también sus resultados:

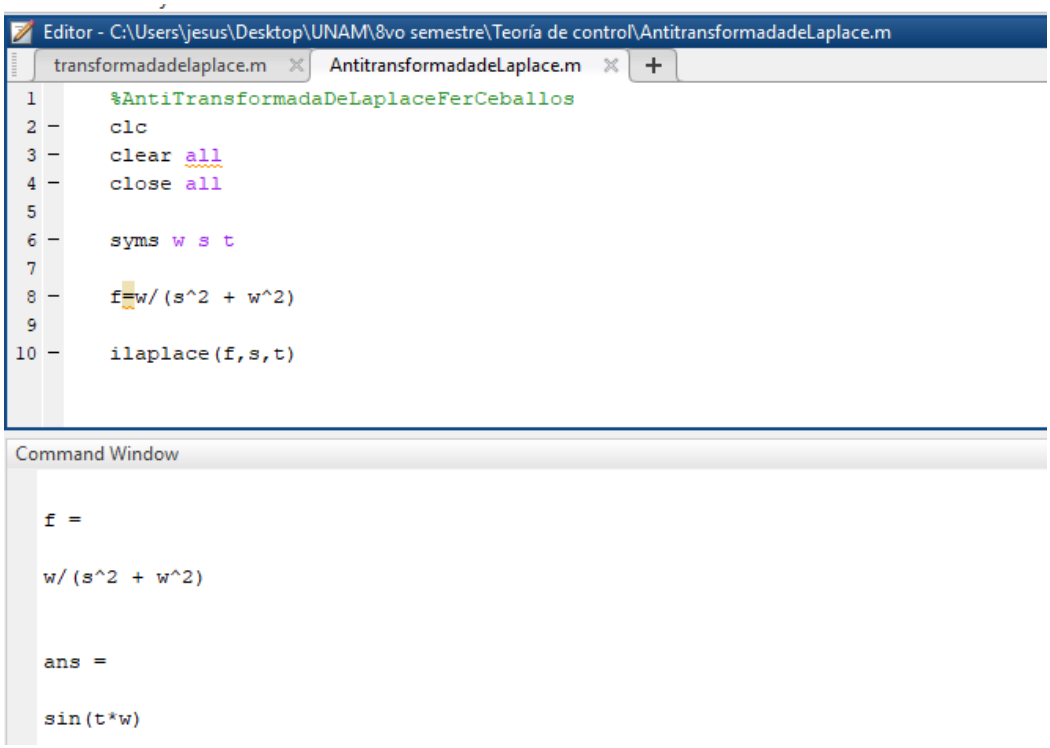


Evidentemente se puede notar que en esta ocasión nuestro programa consta de una línea más: **pretty(finv)**, el comando pretty ayuda a que nuestros resultados se visualicen de mejor manera, y en este caso, decidí utilizarlo para mostrar de una manera más clara lo que explico a continuación. También definí a **finv** como ilaplace, para poder usar el comando.

Este ejemplo parecería estar incorrecto, ya que debería arrojarnos como resultado e^{16t^2}, sin embargo, al ingresar los valores numéricos que nos arroja Matlab en una calculadora científica, podemos constatar que el resultado de esa división es **1**, por lo tanto, el resultado es correcto, aunque la presentación no sea la más adecuada.







Comprobamos nuevamente, que el resultado es el correcto.

FRACCIONES PARCIALES

Las fracciones parciales son fracciones formadas por polinomios, en las que el denominador puede ser un polinomio lineal o cuadrático y, además, puede estar elevado a alguna potencia. A veces, cuando tenemos funciones racionales resulta de gran utilidad reescribir dicha función como una suma de fracciones parciales o fracciones simples para manipular las funciones de mejor manera.

Para realizar la descomposición de una función polinómica racional en fracciones parciales, Matlab nos ofrece la función **residue** para llevarla a cabo. Esta función retoma los coeficientes de los residuos, la localización de los polos y los coeficientes del mínimo directo.

La sintaxis de la función **residue** es la siguiente: **[r,p,k]=residue(N,D)** donde:

r son los coeficientes de los residuos.

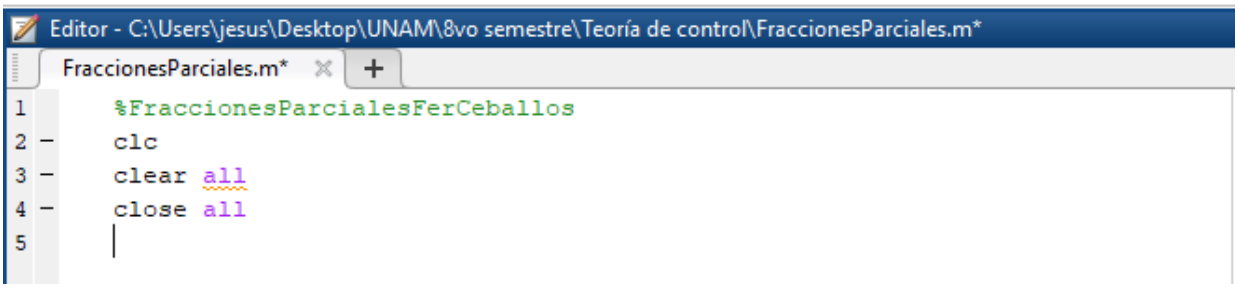
p la localización de los polos.

k los coeficientes del término directo.

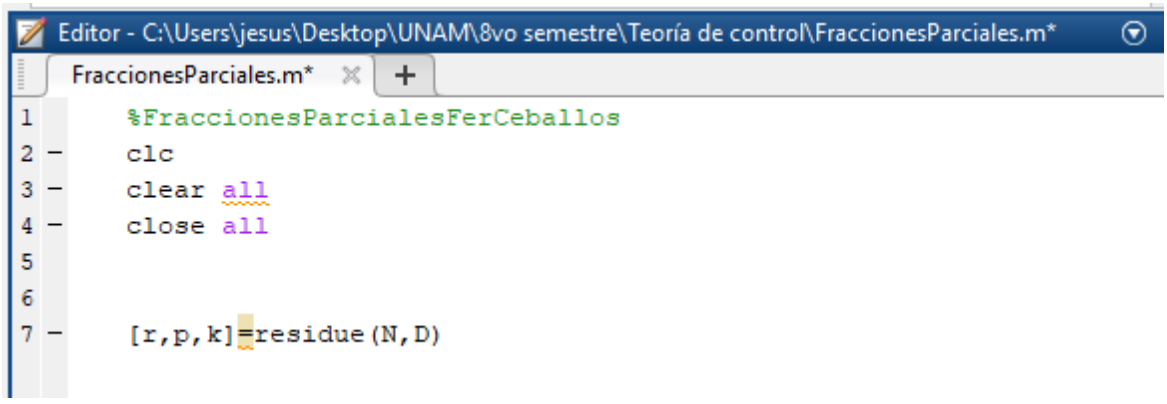
N son los valores del polinomio que se encuentra en el numerador.

D son los valores del polinomio que se encuentra en el denominador.

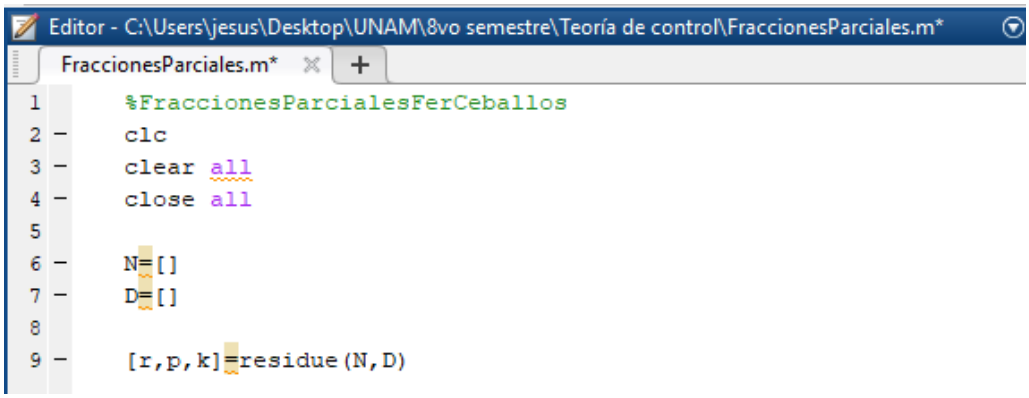
1.Primero que nada (después de crear nuestro Script y guardarlo en nuestra ubicación de preferencia), limpiar la ventana de comandos mediante las instrucciones clc, clear all y close all.



2. Añadir la sintaxis de la función **residue**



3.Encima de la línea donde se escribió la función, se colocarán las letras **N** para definir a los numeradores, y **D** para definir a los denominadores de nuestra función polinómica, seguidas de un signo igual y dos corchetes cerrados en blanco.



Es muy importante recalcar el uso de los corchetes, ya que, si se utilizan paréntesis, Matlab no reconoce la definición de los elementos que se coloquen dentro.

Los corchetes se dejan en blanco para dejar este programa como una especie de plantilla para poder descomponer en fracciones parciales cualquier función polinómica que se introduzca.

La definición de los elementos del numerador y denominador, se darán dentro de los corchetes y cada uno separado por una coma. Se debe respetar la jerarquía de las potencias, de izquierda a derecha y no dejando ningún elemento sin definir, es decir, si el polinomio del numerador es de grado 4, entonces el numerador va a estar constituido por 5 elementos (cuarto, tercer, segundo y primer grado, más un valor constante), obligatoriamente.

Esto se visualiza mejor en el siguiente ejemplo:

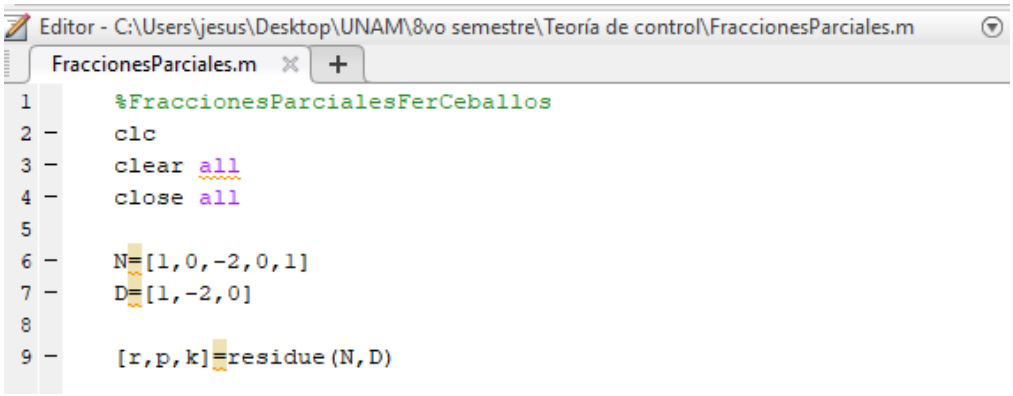
Se descompondrá en Fracciones Parciales la siguiente función polinómica. Nótese que el numerador es de mayor grado que el denominador.

$$\frac{s^4 - 2s^2 + 1}{s^2 - 2s}$$

Como se puede ver, en el numerador el polinomio es de cuarto grado, sin embargo, no posee elementos de tercer y primer grado. Por su parte, en el denominador, el polinomio es de segundo grado, pero no posee un elemento constante.

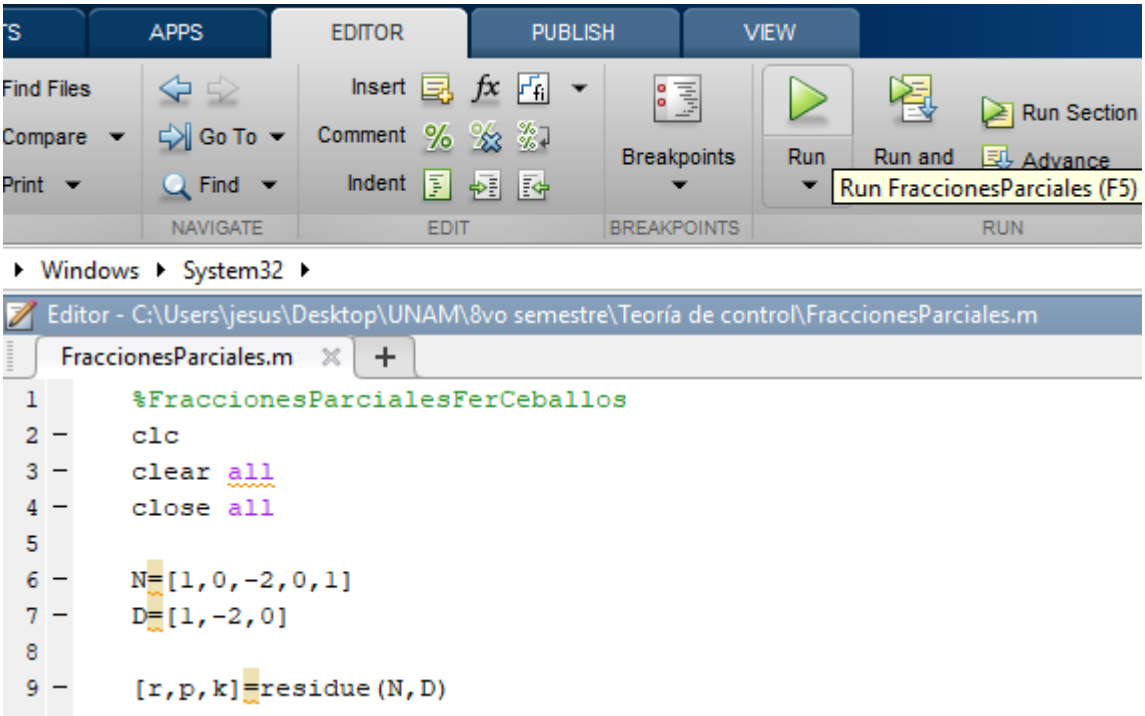
Eso no quiere decir que no se vayan a definir en nuestro programa. La definición de esos elementos "vacíos" que no están dentro del polinomio, se rellenarán con el valor 0 (cero).

De tal manera que la definición de nuestro numerador y denominador, en Matlab, queda de la siguiente manera:

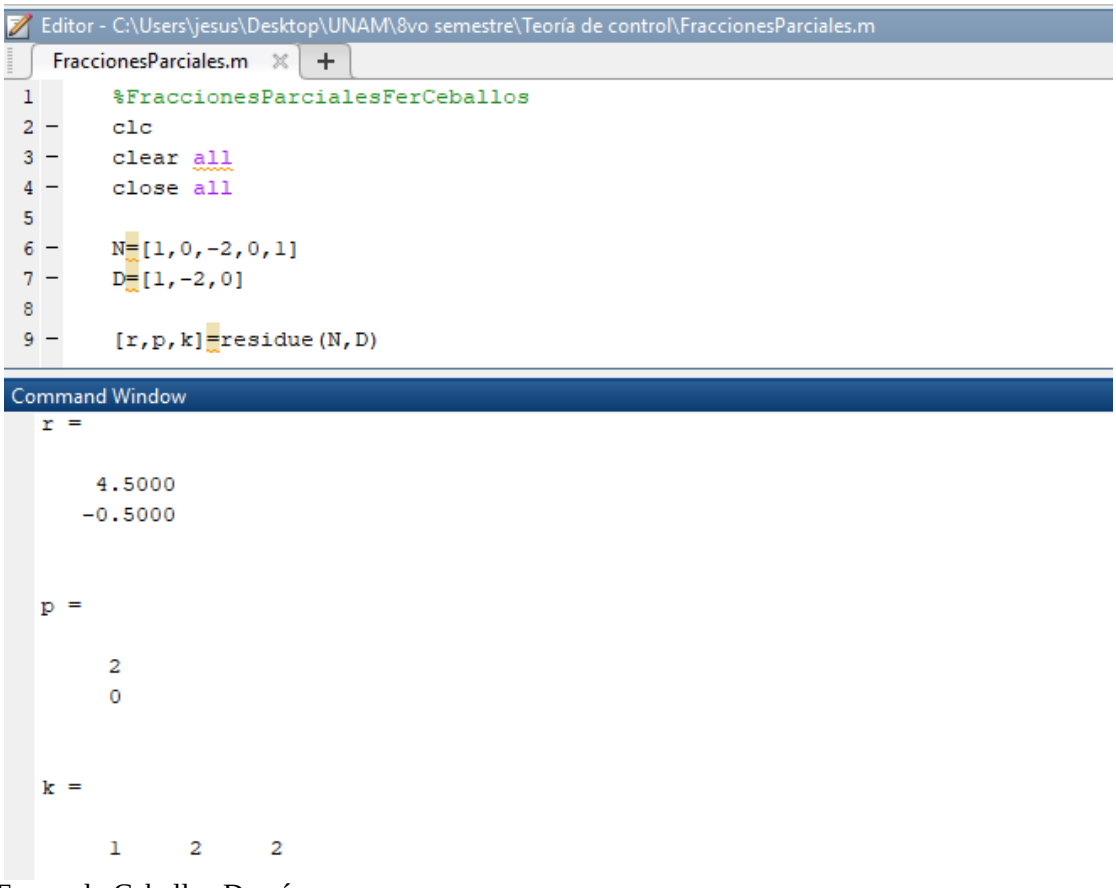


Obsérvese que los elementos vacíos, se rellenaron con 0´s como se indicó.

4.Para obtener los resultados de la descomposición en fracciones parciales de la función, se presiona el botón **Run** ubicado en la pestaña **EDITOR** de la barra de herramientas.



El resultado de la descomposición en fracciones parciales fue la siguiente:



Los valores de **r** corresponden a cada uno de los numeradores de cada fracción parcial.

Los valores de **p** corresponden a cada constante que completarán la parte del denominador en cada fracción parcial, antecedido por una variable de primer grado.

Los valores de **k** corresponden a las constantes que multiplicarán a la variable para sumar al conjunto de fracciones parciales.

Para ilustrar estos últimos 3 puntos, se tomó del buscador de ayuda de Matlab el siguiente modelo de construcción para las fracciones parciales:

$$\frac{b(s)}{a(s)} = \frac{b_m s^m + b_{m-1}s^{m-1} + \ldots + b_1 s + b_0}{a_n s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_0} = \frac{r_n}{s - p_n} + \ldots + \frac{r_2}{s - p_2} + \frac{r_1}{s - p_1} + k(s).$$

Los valores de k, se colocarán en el siguiente orden: el primer término a extrema derecha de los valores encontrados, será el término independiente, y conforme se avance hacia la izquierda, cada valor irá siendo del grado mayor que le sigue.

Por tanto, las fracciones parciales de nuestra función polinómica quedan de la siguiente manera:

$$\frac{s^4 - 2s^2 + 1}{s^2 - 2s} = \frac{4.5}{s - (2)} - \frac{0.5}{s - 0} + s^2 + 2s + 2$$

Se puede ver cómo el 2 más hacia la derecha es el término independiente, el siguiente 2 hacia la izquierda es el coeficiente del término de primer grado, y, por último, el 1 es el coeficiente del término de segundo grado.

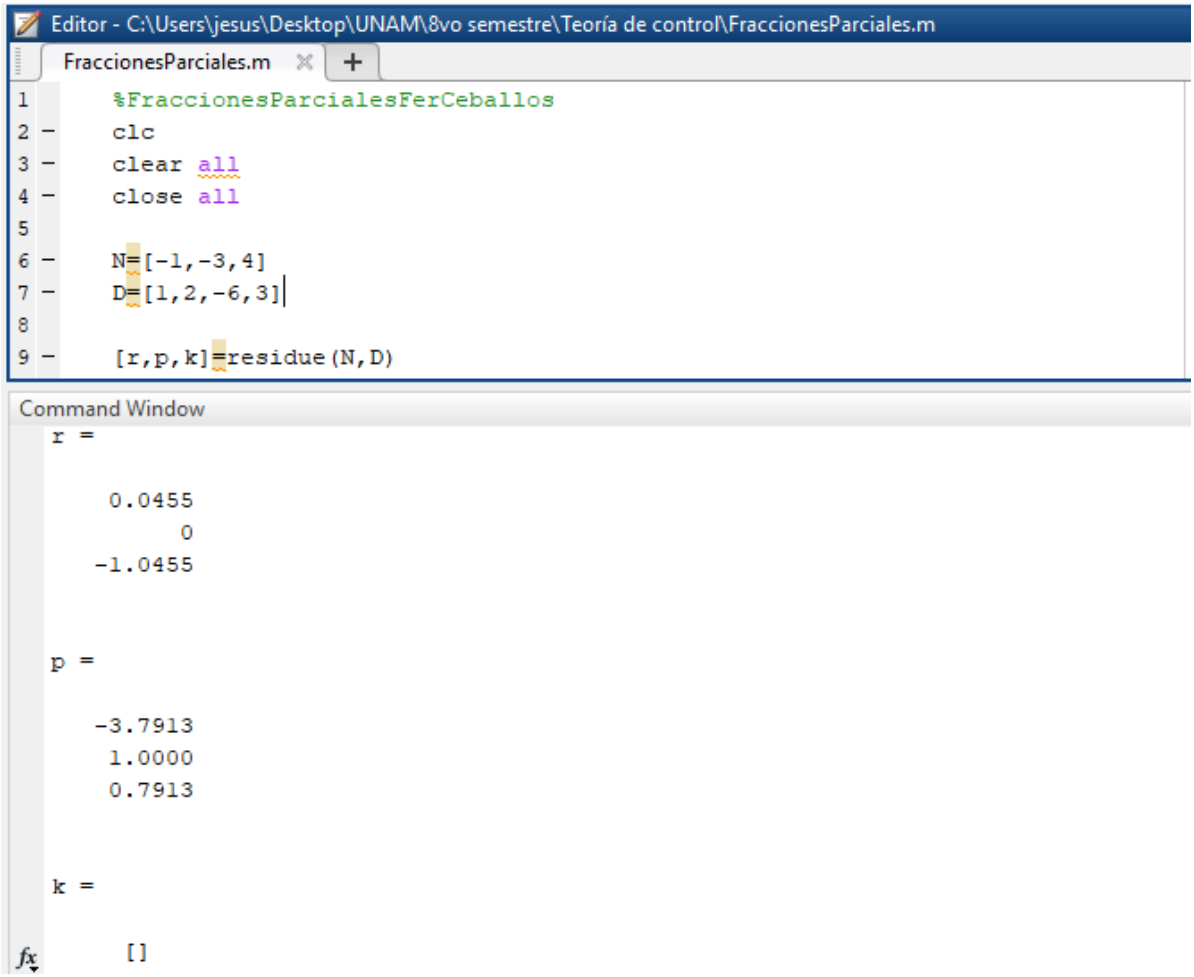
Lo más complicado de realizar fracciones parciales en Matlab es interpretar los resultados que nos arroja, para poder utilizarlos en nuestros problemas, sin embargo, una vez que aprendemos a hacerlo con el modelo de construcción que nos ofrece el mismo Matlab, es sencillo de leer los resultados.

A continuación, agrego 2 ejercicios más de Fracciones Parciales.

Ejemplo 2. En este ejemplo el numerador es de menor grado que el denominador.

$$\frac{-s^2-3+4}{s^3+2s^2-6s+3}$$

La descomposición en fracciones que nos arroja Matlab es la siguiente:



Acomodando los datos tenemos:

$$\frac{-s^2-3+4}{s^3+2s^2-6s+3}=\frac{0.0455}{s-(-3.7913)}+\frac{0}{s-1}-\frac{1.0455}{s-0.7913}=\frac{0.0455}{s+3.7913}-\frac{1.0455}{s-0.7913}$$

En este caso, no hay valores de k, por lo tanto, se omite esa parte del modelo de construcción.

Ejemplo 3. En este ejemplo el numerador y el denominador son del mismo grado.

$$\frac{s^4 + 3s^3 - 2s^2 + 1}{2s^4 - s^3 - 5s^2 + s}$$

La descomposición en Fracciones Parciales que nos arroja Matlab es la siguiente:

Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control\FraccionesParciales.m

FraccionesParciales.m

```
1     %FraccionesParcialesFerCeballos
2 -   clc
3 -   clear all
4 -   close all
5
6 -   N=[1,3,-2,0,1]
7 -   D=[2,-1,-5,1,0]
8
9 -   [r,p,k]=residue(N,D)
```

Command Window

```
r =

    1.1719
    0.5178
   -0.9397
    1.0000


p =

    1.7594
   -1.4548
    0.1953
         0


k =

    0.5000
```

Acomodando los datos obtenemos:

$$\frac{s^4 + 3s^3 - 2s^2 + 1}{2s^4 - s^3 - 5s^2 + s} = \frac{1.1719}{s - 1.7594} + \frac{0.5178}{s - (-1.4548)} - \frac{0.9397}{s - 0.1953} + \frac{1}{s - 0} + 0.5$$

$$= \frac{1.1719}{s - 1.7594} + \frac{0.5178}{s + 1.4548} - \frac{0.9397}{s - 0.1953} + \frac{1}{s} + \frac{1}{2}$$

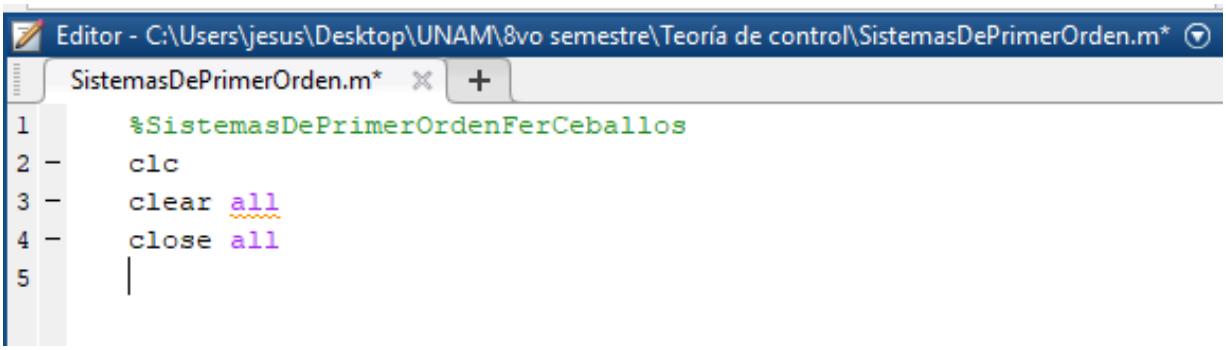
En este caso, solo obtuvimos un valor de k, por lo que se agrega como un valor constante.

SISTEMAS DE PRIMER ORDEN

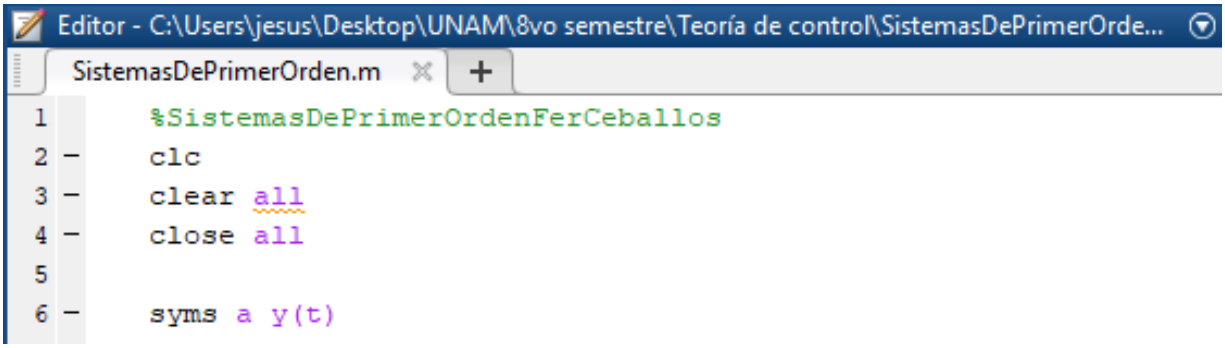
Un sistema de primer orden se caracteriza por tener solamente un elemento capaz de almacenar energía. Se representa por ecuaciones diferenciales ordinarias de primer orden.

Los sistemas de primer orden pueden ser de tipo *eléctrico, mecánico (traslacional y rotacional), hidráulicos* y *térmicos* e incluso *híbridos.*

1.Primero que nada (después de crear nuestro Script y guardarlo en nuestra ubicación de preferencia), limpiar la ventana de comandos mediante las instrucciones clc, clear all y close all



2.Definir las variables que utilizaremos en nuestro programa, en este caso, **a** y **y(t)** (y en función de t).



3.Matlab nos ofrece una función para resolver ecuaciones diferenciales, esta es **dsolve** y su sintaxis es la siguiente: **S=dsolve(eqn)** para ecuaciones diferenciales y **S=dsolve(eqn,cond)** para ecuaciones diferenciales con condiciones iniciales. Donde:

eqn es la ecuación y de especifica con la sintaxis **diff(y,t)==.**

diff nos indica que se trata de una ecuación diferencial

y representa la variable de la función.

t representa la variable respecto a la cual funciona el sistema.

== se utiliza para definir la ecuación diferencial.

cond es la condición inicial del sistema de primer orden y e introduce al programa mediante la siguiente sintaxis: **cond=y(0)==.**

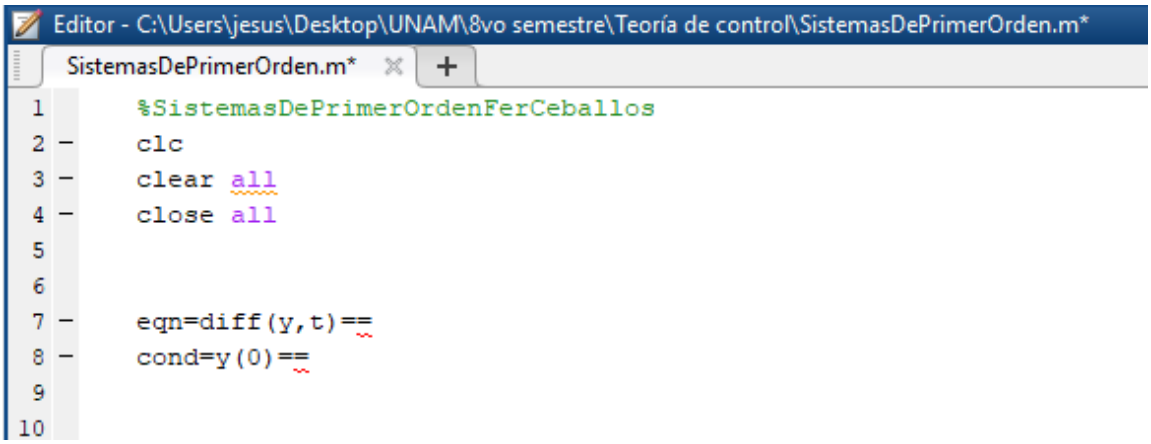
y(0) es la condición inicial. El cero entre paréntesis **(0)** representa el dominio de la condición inicial.

== define el valor de la condición inicial

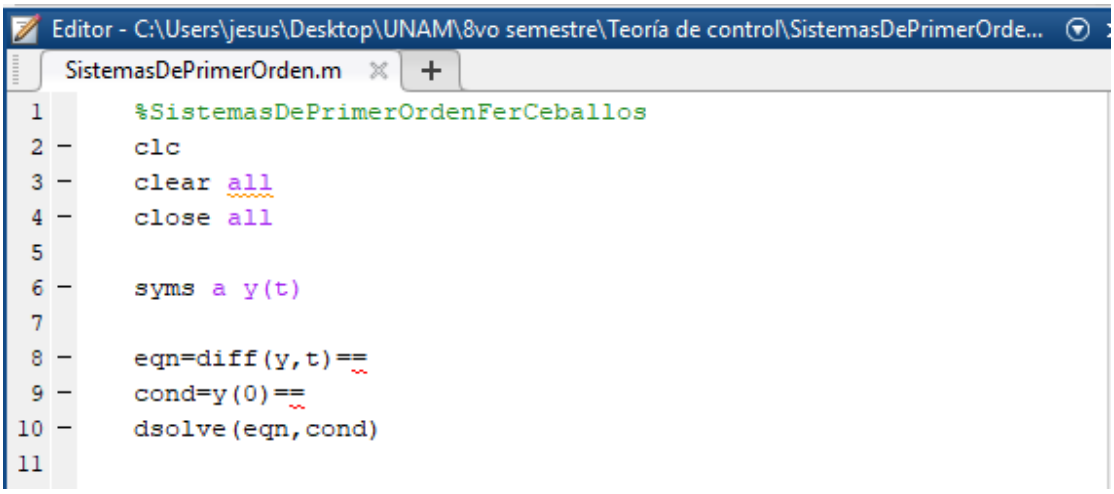
Seguido de == se escribe el valor de la condición inicial.

Se pueden definir cuantas condiciones sean necesarias para satisfacer el sistema.

Introducimos esta sintaxis, de la siguiente manera:



3.Una vez introducido esto, procedemos a escribir la función de Matlab que nos permitirá obtener el resultado del sistema de primer orden: **dsolve(eqn,cond)**

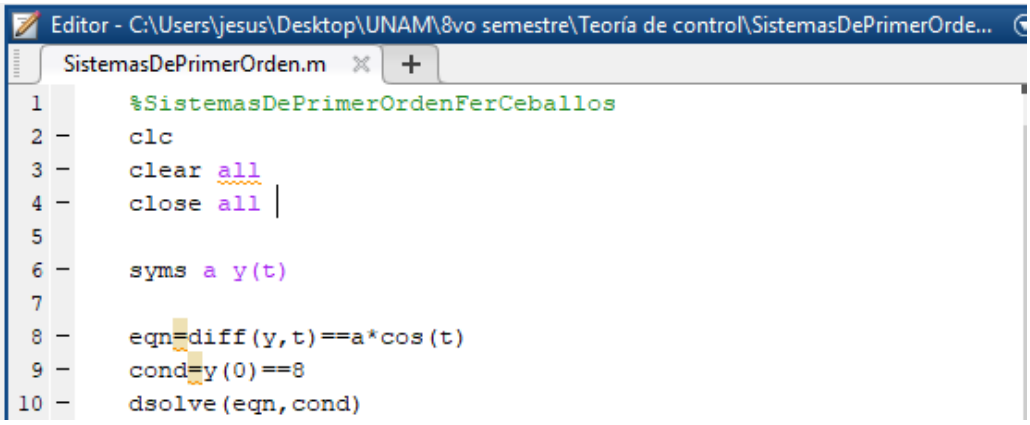


4.Añadimos nuestra ecuación diferencial en la línea **eqn.** Para ejemplificar como funciona, el sistema de primer orden a realizar es el siguiente:

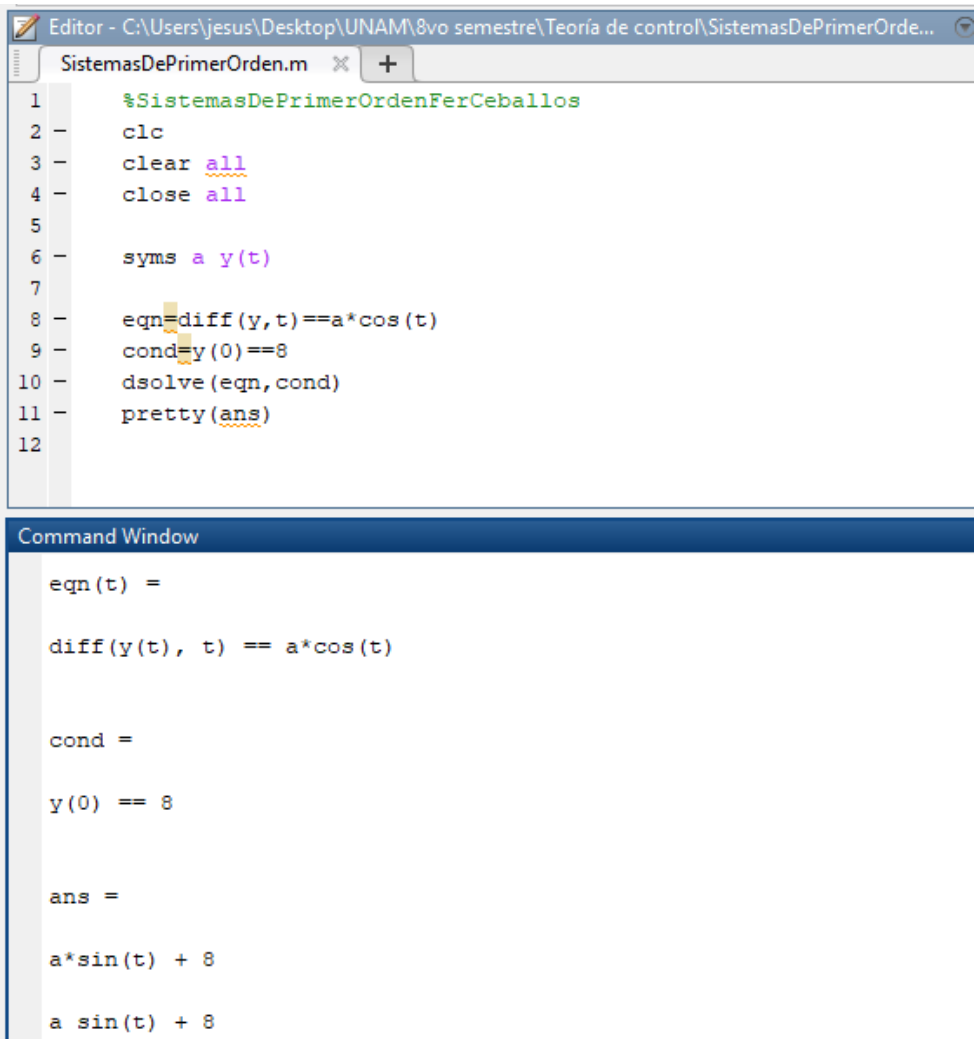
$$\frac{dy}{dt} - a\cos(t) = 0 \;; \qquad y(0) = 8$$

Para ingresarla al programa tenemos que separar el diferencial dy/dt de un solo lado de la función:

$$\frac{dy}{dt} = \text{acos}(t)$$



5.Por último, corremos el programa, presionando el botón **Run** en la pestaña **EDITOR** de la barra de herramientas.



Se puede añadir el comando **pretty(ans)** para que la presentación del resultado final sea mejor y más entendible (en caso de requerirlo)

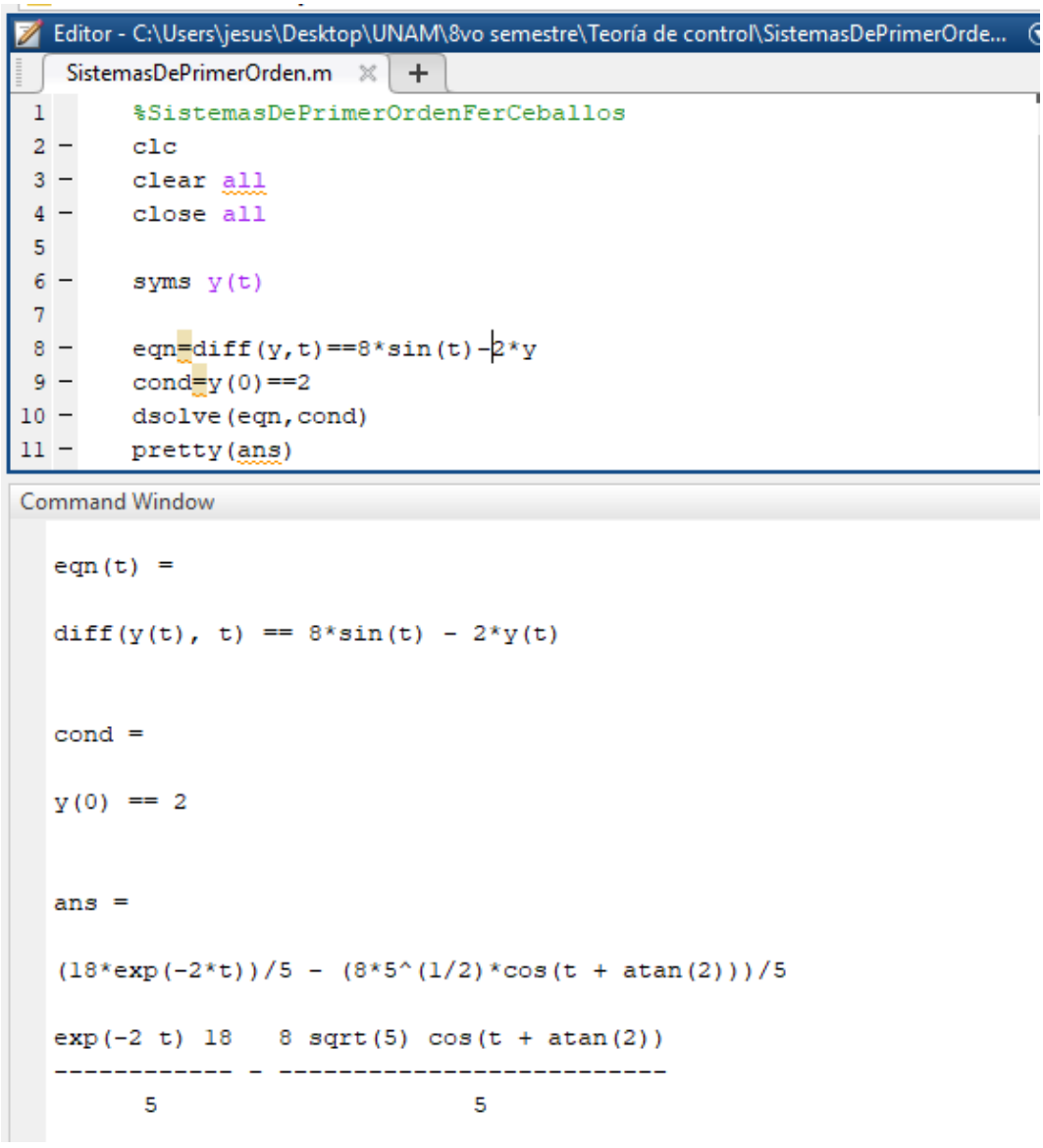
A continuación, se realizan dos ejemplos más de Sistemas de Primer Orden

Ejemplo 2. El sistema de primer orden a realizar es el siguiente:

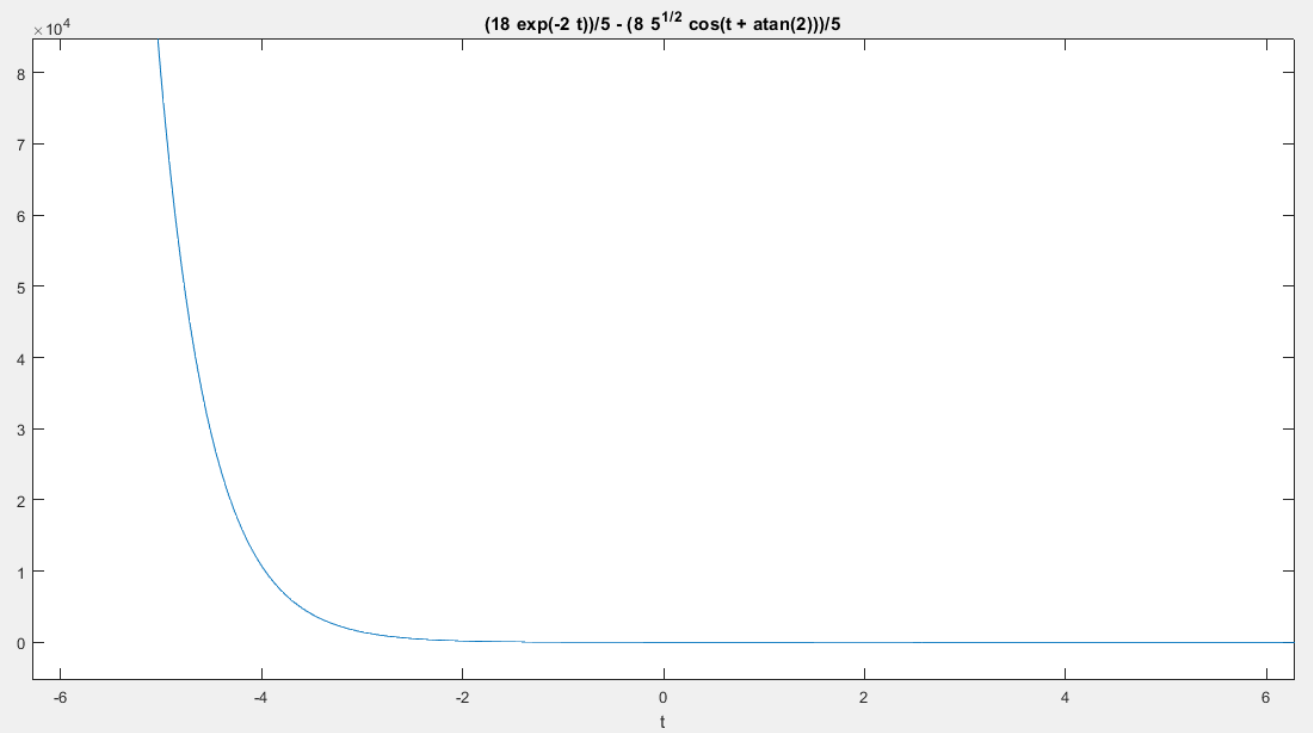
$$\frac{dy}{dt} - 8\sin(t) + 2y = 0 \ ; \qquad y(0) = 2$$

Recordemos que, para ingresarla al programa, tenemos que separar dy/dt e igualarla al resto de la función:

$$\frac{dy}{dt} = 8\sin(t) - 2y$$



Con el comando **ezplot(ans)** obtenemos la gráfica de nuestro sistema:



En este ejemplo no fue necesario declarar otra variable más que **y(t).** Las **a** que aparecen multiplicando a las funciones trigonométricas indican que la función es inversa. Es decir, **atan** es la *inversa de la tangente*.

Ejemplo 3. El sistema de primer orden a realizar es el siguiente:

$$\frac{dy}{dt} + \sin(t) + 5 = 0 \ ; \qquad y(0) = 1$$

Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control\SistemasDePrimerOrden.m

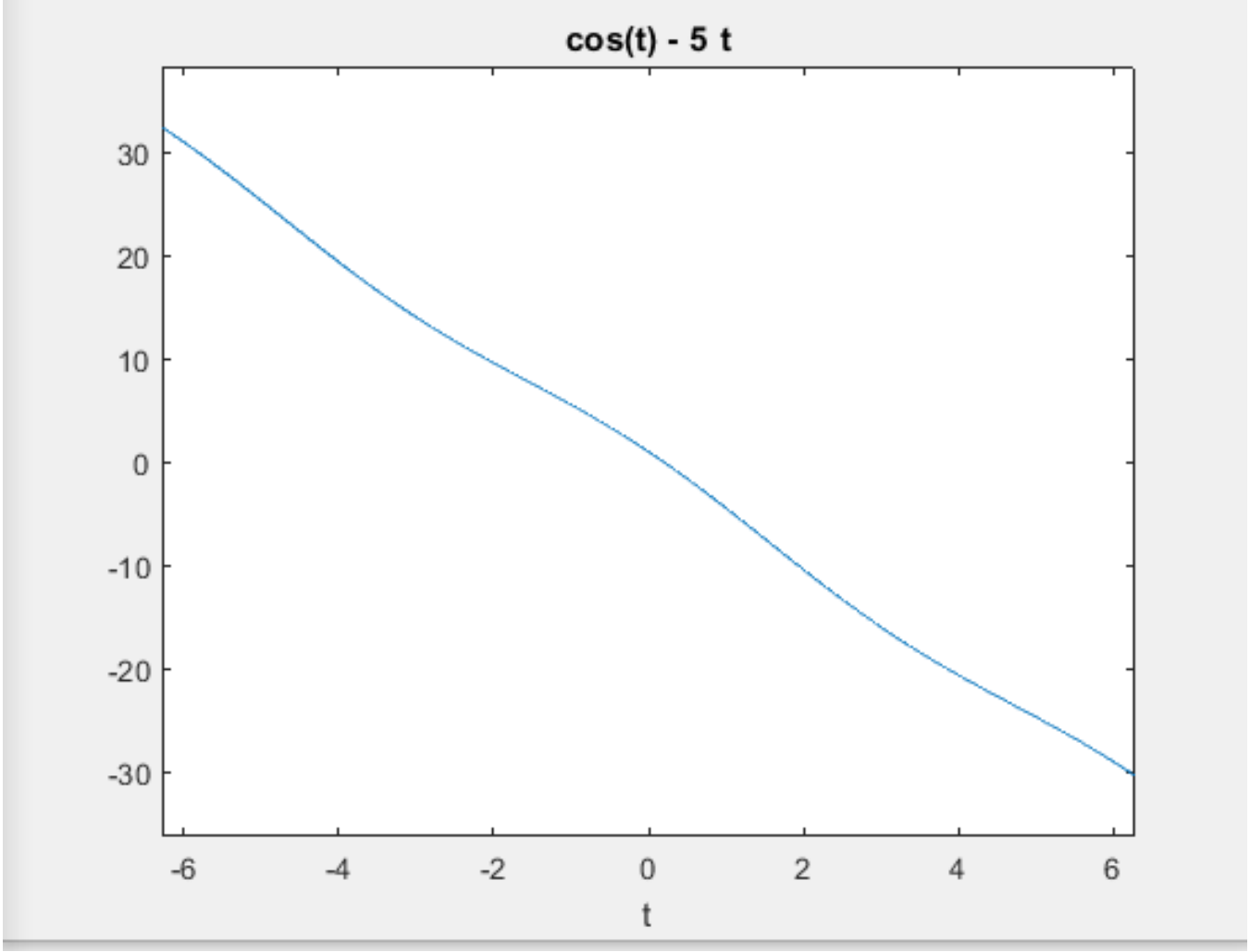
SistemasDePrimerOrden.m

```
1     %SistemasDePrimerOrdenFerCeballos
2 -   clc
3 -   clear all
4 -   close all
5
6 -   syms t y(t)
7
8 -   eqn=diff(y,t)==-sin(t)-5
9 -   cond=y(0)==1
10
11 -  dsolve(eqn,cond)
12 -  pretty(ans)
13 -  ezplot (ans)
```

Command Window

```
eqn(t) =

diff(y(t), t) == - sin(t) - 5


cond =

y(0) == 1


ans =

cos(t) - 5*t

cos(t) - 5 t
```

Con el comando **ezplot(ans)** obtenemos la gráfica de nuestro sistema:

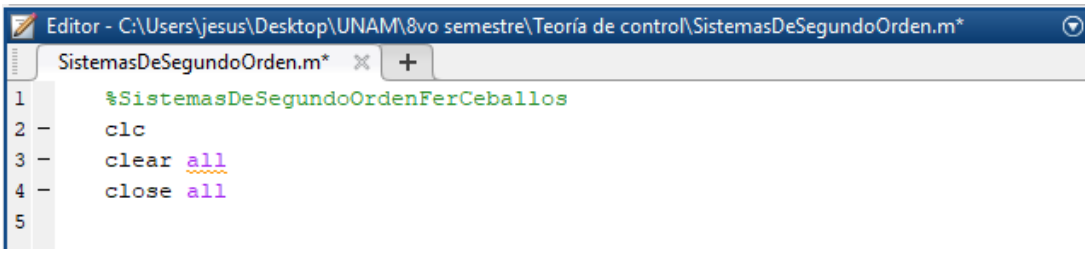


SISTEMAS DE SEGUNDO ORDEN

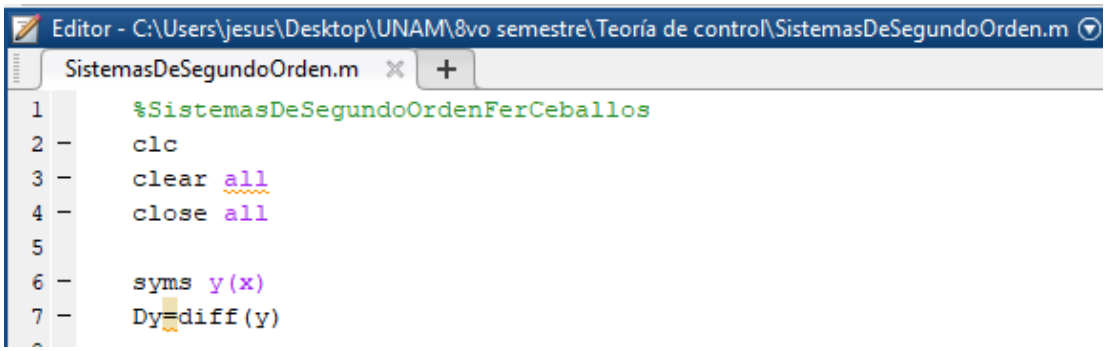
Los sistemas de segundo orden se caracterizan por tener dos polos y están representados por ecuaciones diferenciales de grado 2.

La manera de resolverlas es muy similar a como se resuelven los sistemas de primer orden, solo que se indica el grado del sistema, en este caso 2.

1.Primero que nada (después de crear nuestro Script y guardarlo en nuestra ubicación de preferencia), limpiar la ventana de comandos mediante las instrucciones clc, clear all y close all



2.Definir las variables que utilizaremos en nuestro programa, en este caso, y**(x)** (y en función de x). También **Dy** que representará la derivada. **Dy=diff(y),** Dy es igual a la derivada de y.



Ese **Dy** también es útil a la hora de colocar las condiciones iniciales del sistema.

3.Continuamos definiendo el orden de nuestro sistema, en este caso es de segundo, con la función **ode** que nos ofrece Matlab. Su sintaxis es la siguiente: **ode=diff(y,x,2)==.**

ode Order Differential Equatioin

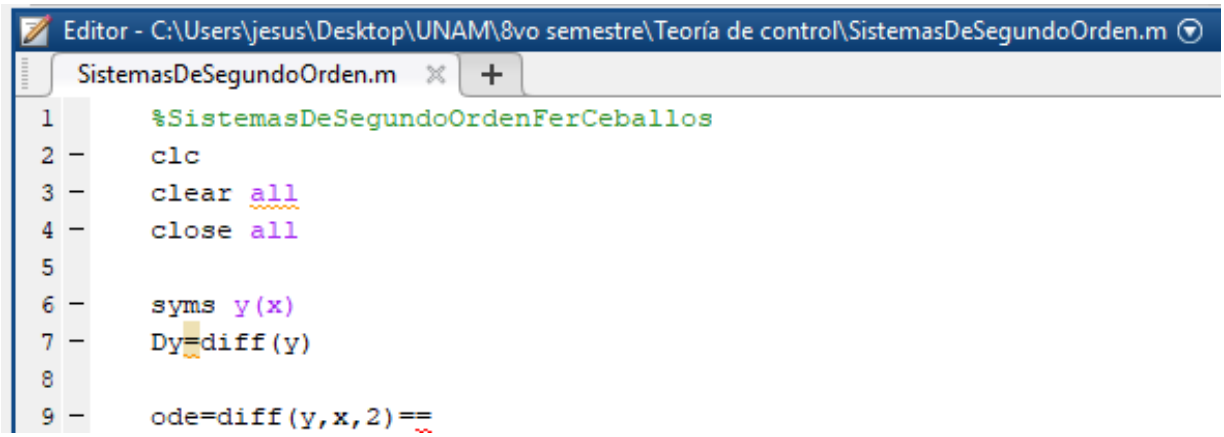
diff define el diferencial de la ecuación, en este caso y respecto a x)

y es la variable de la ecuación.

x define la variable respecto a la cual se deriva.

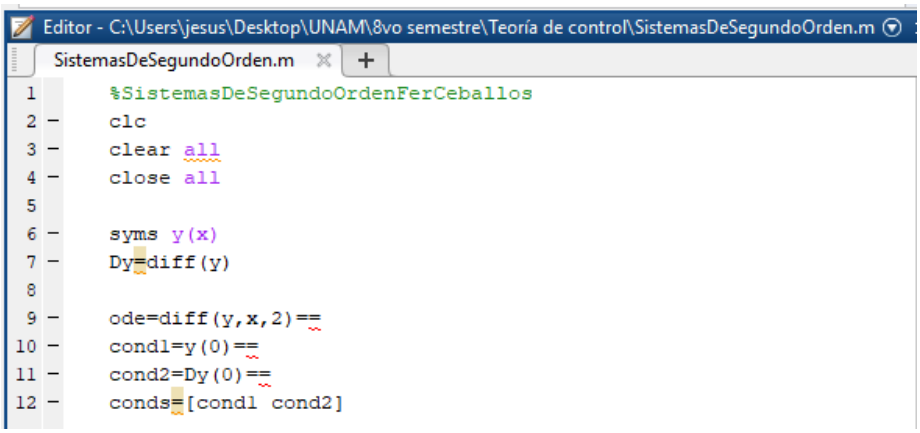
2 es el grado de la ecuación.

== se encargan de declararle al programa la ecuación diferencial a resolver.



4.El siguiente paso es añadir las condiciones iniciales de nuestro sistema de segundo orden. Al ser de segundo orden, son dos las condiciones iniciales.

En los sistemas de segundo orden solo necesitábamos definir una condición inicial, entonces no había necesidad de nombrar a las condiciones, sin embargo, como en este caso ocuparemos más, se enumeran comenzando por 1, posteriormente se agruparán con el comando **conds=[cond1 cond2].**



5.Una vez definidas las condiciones iniciales, lo siguiente es escribir el comando que nos permita resolver nuestra ecuación diferencial, esto es con una función de Matlab ya conocida: **dsolve.**

En este caso, para resolver *y*, la sintaxis de la función dsolve varía un poco: **dsolve(ode,conds).**

Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control\SistemasDeSegundoOrden.m

SistemasDeSegundoOrden.m

```
%SistemasDeSegundoOrdenFerCeballos
clc
clear all
close all

syms y(x)
Dy=diff(y)

ode=diff(y,x,2)==
cond1=y(0)==
cond2=Dy(0)==
conds=[cond1 cond2]

Sol=dsolve(ode,conds)
simplify(Sol)
```

Añadí una definición a la operación **Sol=dsolve(ode,conds)** y el comando **simplify(Sol)** para simplificar el resultado final de la solución del sistema de segundo orden.

Ahora sí, está listo nuestro programa para resolver cualquier tipo de sistema de segundo orden.

Se resuelven a continuación 3 ejemplos de sistemas de segundo orden con el programa codificado.

Ejemplo 1. El sistema de segundo orden a resolver es el siguiente:

$$\frac{d^2y}{dx^2} + \cos(x) - y + 8 = 0 \ ; \qquad y(0) = 2 \ ; \qquad y'(0) = 1;$$

Para introducir los datos a nuestro programa, es necesario separar $\frac{d^2y}{dx^2}$ del resto de la ecuación e igualarla.

$$\frac{d^2y}{dx^2} = -\cos(x) + y - 8 = 0$$

Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control\SistemasDeSegundoO

SistemasDeSegundoOrden.m

```
%SistemasDeSegundoOrdenFerCeballos
clc
clear all
close all

syms y(x)
Dy=diff(y)

ode=diff(y,x,2)==-cos(x)+y-8
cond1=y(0)== 2
cond2=Dy(0)==1
conds=[cond1 cond2]

Sol=dsolve(ode,conds)
Y=simplify(Sol)
pretty(Y)
```

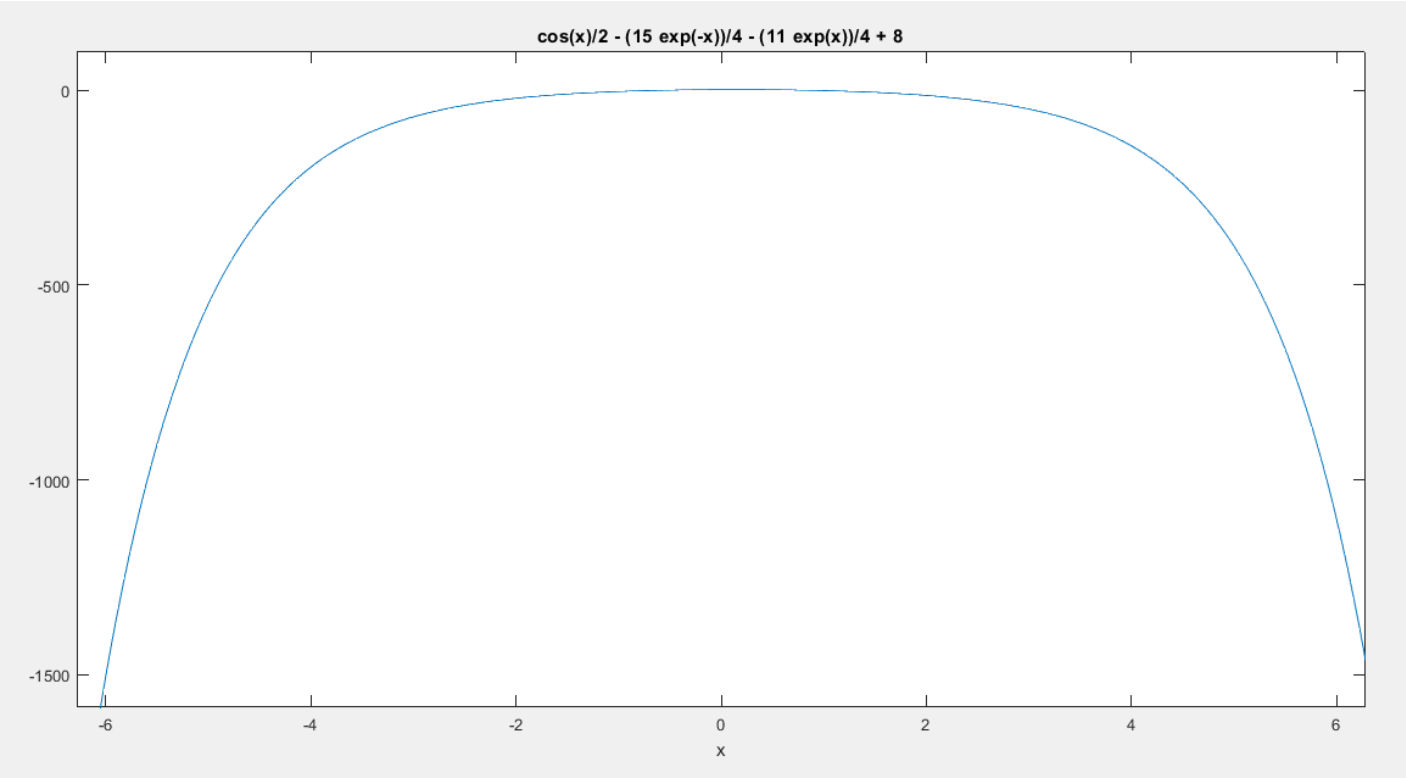
La resolución del sistema es la siguiente:

Command Window

```
diff(y(x), x, x) == y(x) - cos(x) - 8


cond1 =

y(0) == 2


cond2 =

subs(diff(y(x), x), x, 0) == 1


conds =

[ y(0) == 2, subs(diff(y(x), x), x, 0) == 1]


Sol =

cos(x)/2 - (15*exp(-x))/4 - (11*exp(x))/4 + 8


Y =

cos(x)/2 - (15*exp(-x))/4 - (11*exp(x))/4 + 8

cos(x)   15 exp(-x)   11 exp(x)
------ - ---------- - --------- + 8
   2          4           4
```

Con el comando **pretty(Y)j se mejora la presentación del resultado final.**

La gráfica que muestra el comportamiento del sistema es la siguiente:



Ejemplo 2. El sistema de segundo orden a resolver es el siguiente:

$$\frac{d^2y}{dx^2} - e^{2x} + 3 = 0 \ ; \qquad y(0) = 1 \ ; \qquad y'(0) = 0;$$

$$\frac{d^2y}{dx^2} = e^{2x} - 3$$

SistemasDeSegundoOrden.m

```
1     %SistemasDeSegundoOrdenFerCeballos
2 -   clc
3 -   clear all
4 -   close all
5 -   syms y(x)
6 -   Dy=diff(y)
7
8 -   ode=diff(y,x,2)==exp(2*x)-3
9 -   cond1=y(0)== 1
10 -  cond2=Dy(0)==0
11 -  conds=[cond1 cond2]
12
13 -  Sol=dsolve(ode,conds)
14 -  Y=simplify(Sol)
15 -  pretty(Y)
16 -  ezplot(Sol)
```

La resolución del sistema es la siguiente:

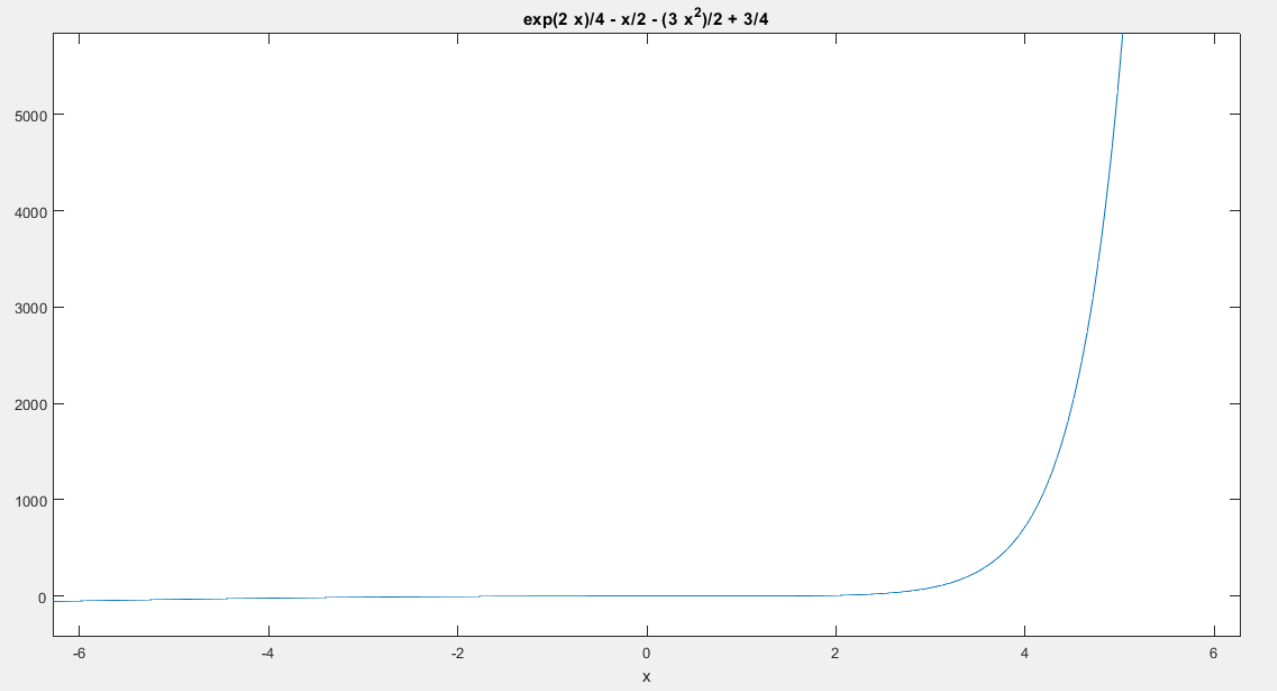
Command Window

```
diff(y(x), x, x) == exp(2*x) - 3


cond1 =

y(0) == 1


cond2 =

subs(diff(y(x), x), x, 0) == 0


conds =

[ y(0) == 1, subs(diff(y(x), x), x, 0) == 0]


Sol =

exp(2*x)/4 - x/2 - (3*x^2)/2 + 3/4


Y =

exp(2*x)/4 - x/2 - (3*x^2)/2 + 3/4

                    2
exp(2 x)    x    3 x     3
-------- - - - ---- + -
    4       2     2     4
```

fx

La gráfica del sistema es la siguiente:

La gráfica del comportamiento del sistema es la que se muestra a continuación:



Ejemplo 3. El sistema de segundo orden a resolver es el siguiente:

$$\frac{d^2y}{dx^2} - 8x + e^x - 1 = 0 \; ; \qquad y(0) = 1 \; ; \qquad y'(0) = 0;$$

$$\frac{d^2y}{dx^2} = 8x - e^x + 1$$

Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control\SistemasDeSegundoOrde...

SistemasDeSegundoOrden.m

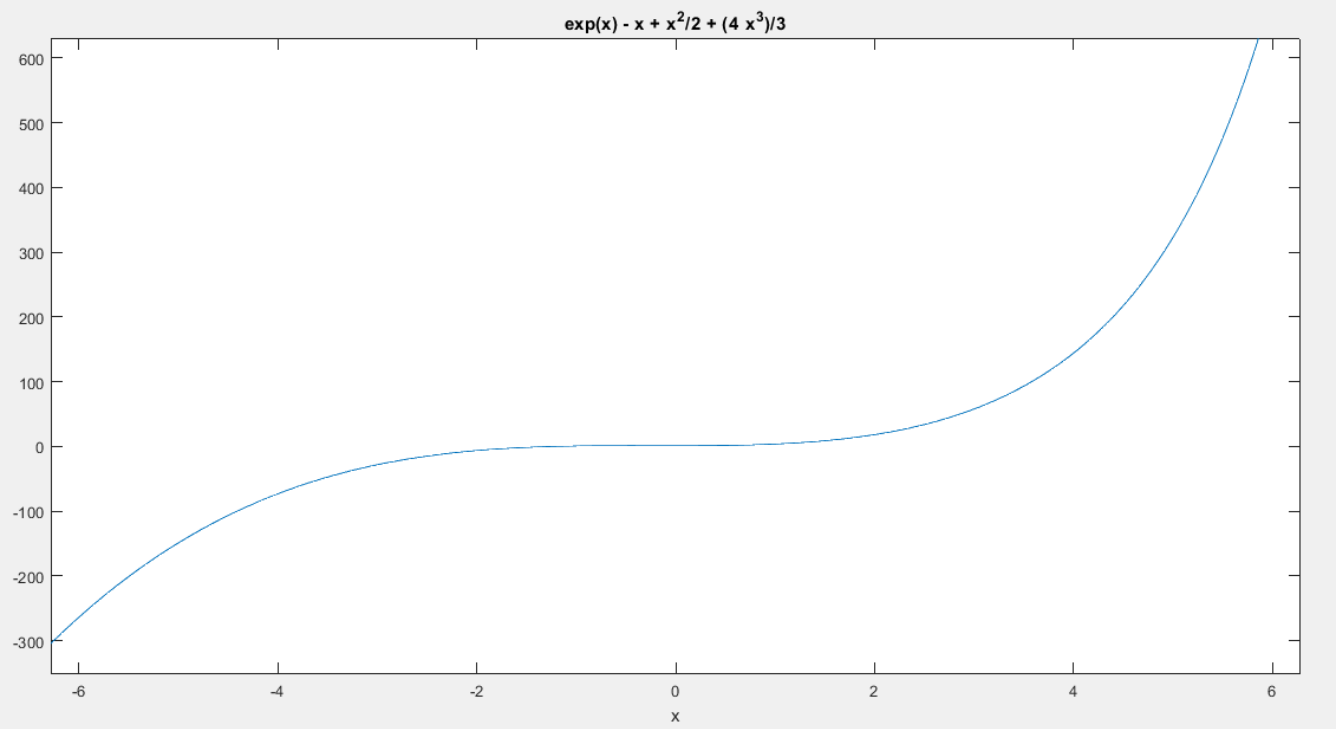
```
%SistemasDeSegundoOrdenFerCeballos
clc
clear all
close all
syms y(x)
Dy=diff(y)

ode=diff(y,x,2)==8*x+exp(x)+1
cond1=y(0)== 1
cond2=Dy(0)==0
conds=[cond1 cond2]

Sol=dsolve(ode,conds)
Y=simplify(Sol)
pretty(Y)
ezplot(Sol)
```

La resolución del sistema es la siguiente:

Command Window

```
diff(y(x), x, x) == 8*x + exp(x) + 1


cond1 =

y(0) == 1


cond2 =

subs(diff(y(x), x), x, 0) == 0


conds =

[ y(0) == 1, subs(diff(y(x), x), x, 0) == 0]


Sol =

exp(x) - x + x^2/2 + (4*x^3)/3


Y =

exp(x) - x + x^2/2 + (4*x^3)/3

                2       3
               x     4 x
exp(x) - x + -- + ----
               2      3
```

La gráfica del sistema es la siguiente:



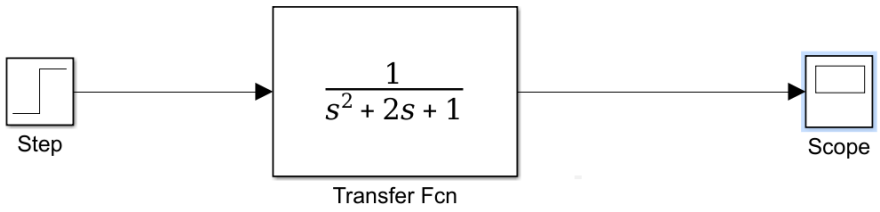
SISTEMAS DE CONTROL EN MATLAB Y SIMULINK

Para realizar el diseño de un sistema de control en Simulink, se recomienda leer el capítulo ***Diagramas de bloques y modelo de sistemas en Matlab Simulink*** que se encuentra en la primera parte de este manual. Ahí se encontrarán los pasos para realizar el diagrama de bloques correspondiente a cualquier sistema de control que se requiera, ya que solo se ocupa localizar la librería.

En este capítulo si se mencionarán los elementos nuevos que lleguen a utilizarse, específicamente los controladores P, PI, PID.

Abrimos Simulink. Para ejemplificar el proceso, se hará uso de un ejemplo, la función de transferencia siguiente:

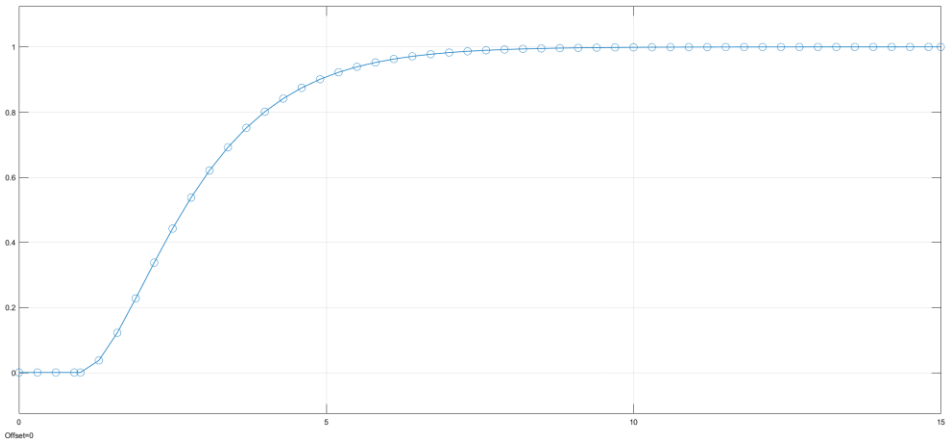
$$\frac{1}{s^2 + 2s + 1}$$



El diagrama de bloques consta de tres elementos. Una función escalón como entrada, una función transferencia y un *scope* que es una especie de osciloscopio que nos permite graficar a la función transferencia dada

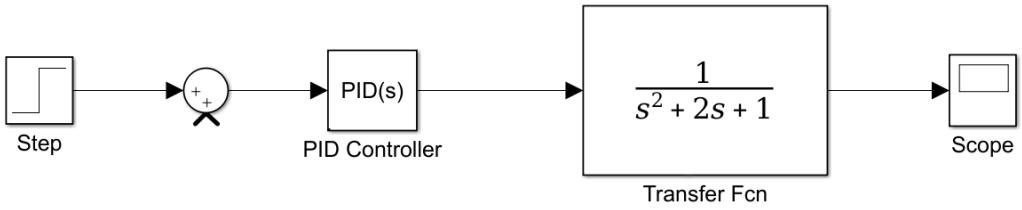
La entrada de función escalón (*step*) se encuentra en la librería ***Sources***. Esta será la señal de entrada de nuestro sistema. La función transferencia se encuentra en la librería ***Continuous.*** *Scope* se encuentra en la librería ***Sinks***.

Después de tener el diagrama de bloques de nuestro sistema de control, presionamos **Run**. Para ejecutar el programa. Después de que se realice esta acción, se da doble clic sobre el ícono de *Scope*, esto nos arrojará una ventana con la gráfica proporcionada, en este caso:



En la gráfica se puede observar como se estabiliza el sistema después de un tiempo, no se perciben oscilaciones.

A continuación, se procederá a agregar un controlador de tipo P al sistema, para ver de que manera afecta al desempeño de la función dada.

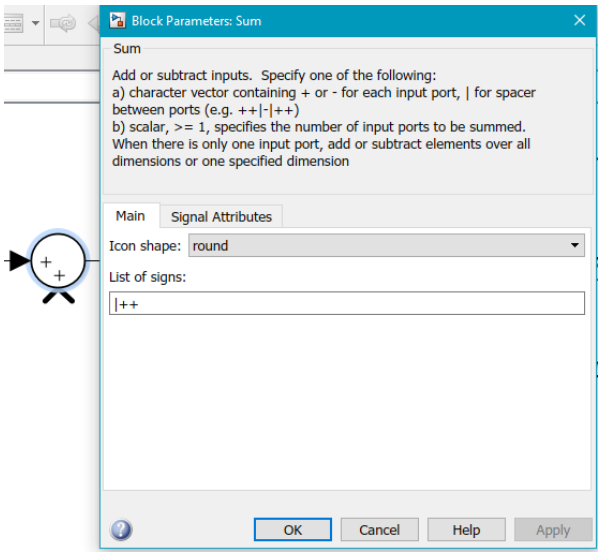


Nótese, que el controlador se colocó justo antes de la función transferencia, y también, que enseguida de la señal de entrada se añadió un sumador.

El sumador se encuentra en la librería ***Math Operations*** con el nombre *Sum***.** El controlador PID se halla en la librería ***Continuous*** con el nombre *PID Controller*. Este nos permitirá hacer uso de los controladores de tipo P, PI y PID modificando sus parámetros como se mencionará más adelante.

Para que nuestro controlador tenga efecto sobre el sistema, se necesita un regreso de la salida al sumador. Se requiere cambiar el signo del regreso en el sumador a negativo, ya que por default es positivo.

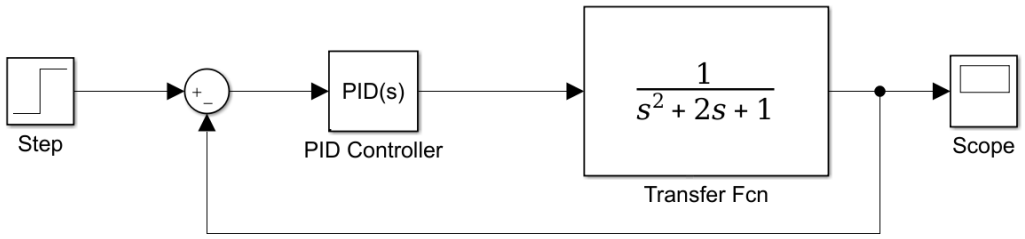
Para cambiar el signo, basta con dar doble clic sobre el icono del sumador, se abrirá un cuadro como se muestra a continuación:



De izquierda a derecha, el primer signo corresponde a la parte de la entrada y el segundo al que utilizaremos como retorno. En este caso, se requiere cambiar a negativo el signo del retorno, basta con eliminar el positivo y escribir el negativo con nuestro teclado.

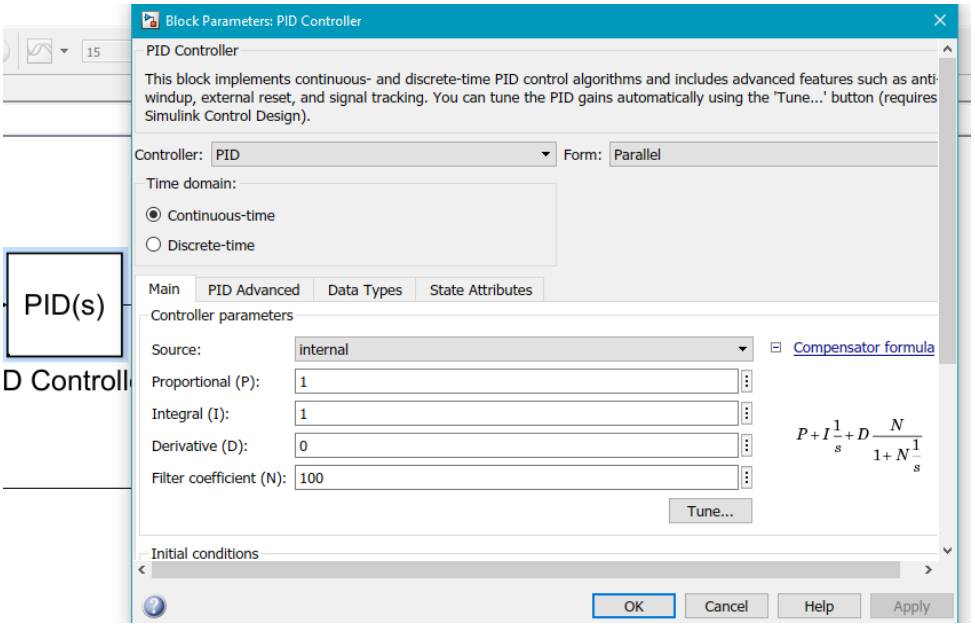
Una vez cambiado, se presiona *Apply* y enseguida *Ok* para validar el cambio.

El nuevo diseño de nuestro sistema luce de la siguiente manera (ya con el retorno):

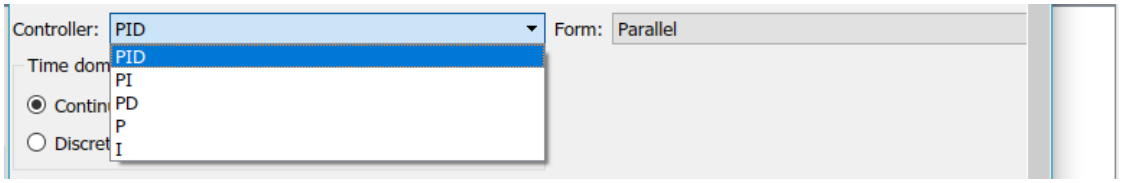


Ahora, lo que queda es modificar nuestro controlador PID para que funcione como **P.**

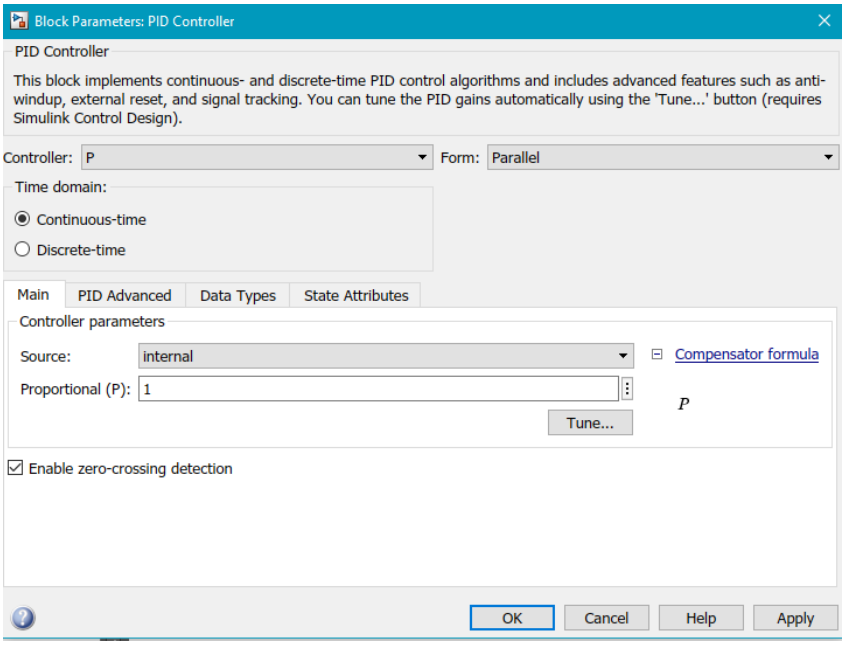
Para modificar la configuración de nuestro controlador, basta con dar doble clic sobre el ícono. Se desplegará una ventana como se muestra a continuación:



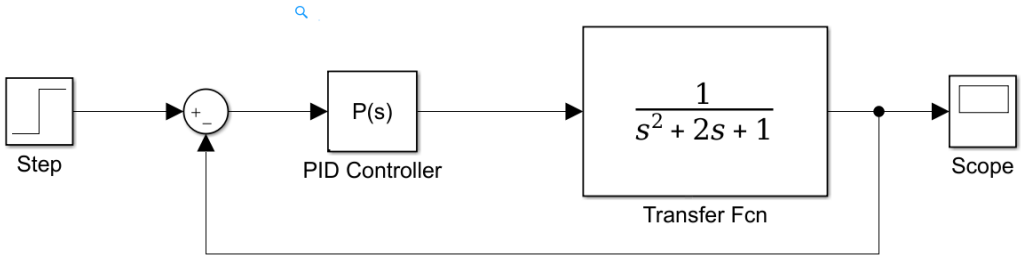
En el apartado **Controller** está seleccionado por default PID. Al dar clic sobre este, se desplegarán las opciones de tipo de controlador con las que se cuenta.



Se seleccionará la opción P, y se notará como el menú de los parámetros modificables cambia respecto a como se mostraron en un principio, esto debido a que el tipo de controlador se modificó.

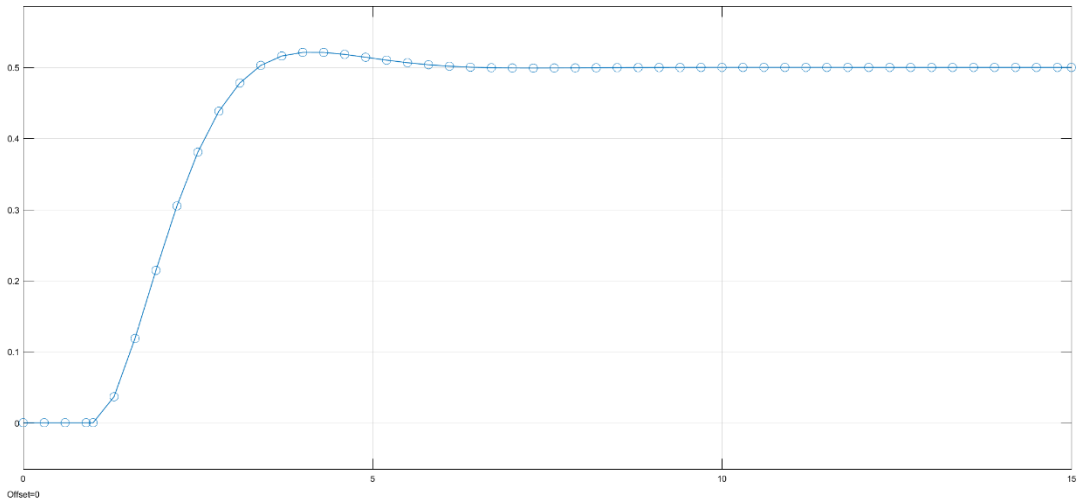


Por el momento no se modificará la constante de proporcionalidad, así que presionamos *Apply* y seguidamente *Ok*, para validar las modificaciones.



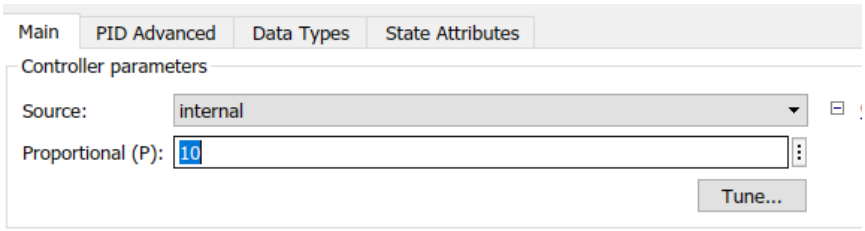
Puede observarse como nuestro controlador pasó de PID(S) a P(s).

Se procede a correr nuevamente el programa, presionando **Run** y enseguida se ejecuta el *Scope* como se indicó anteriormente.



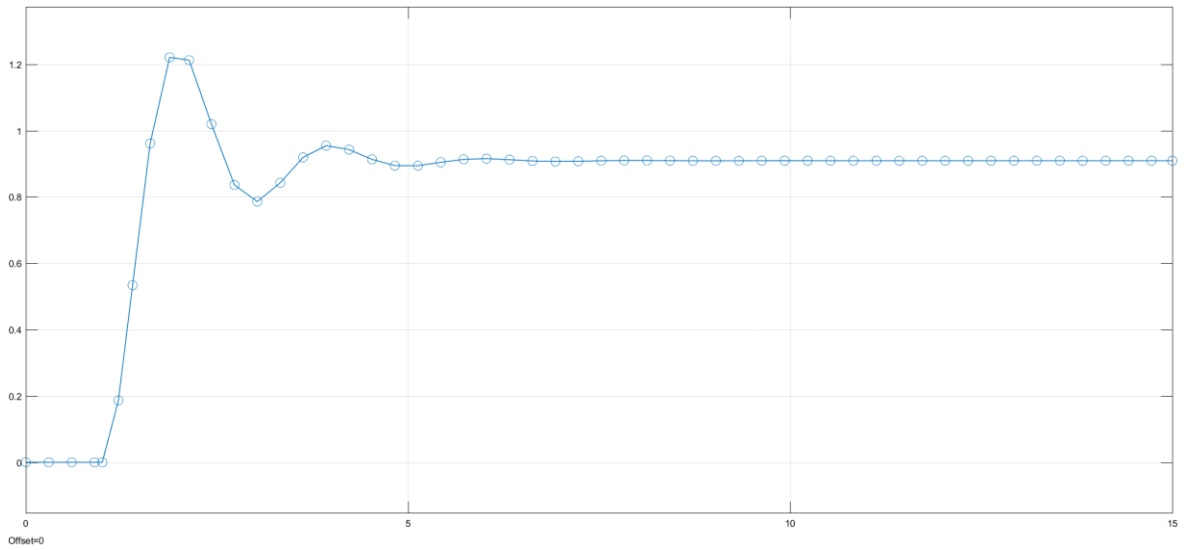
La gráfica arrojada nos muestra como el controlador añade una leve oscilación que no se notaba antes añadir el controlador. Sin embargo, es claro que la modificación de la señal no es tan drástica, esto se debe a que mantuvimos la constante de proporcionalidad en 1 como se mostró.

A continuación, se modificará el valor de la constante de proporcionalidad para observar que sucede si este se aumenta en 10 veces su valor:



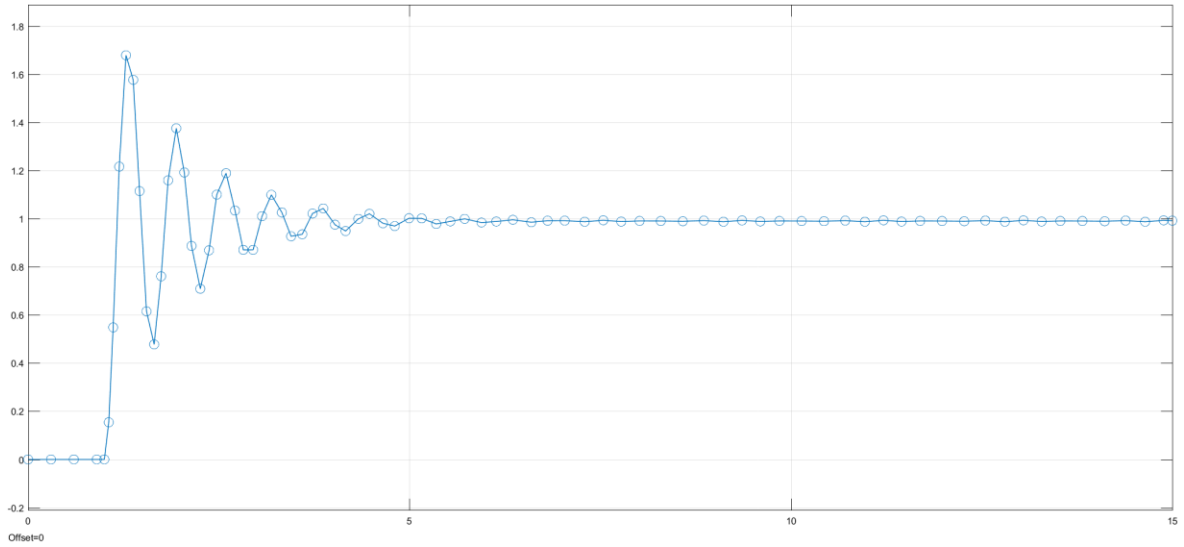
Después de esto, se aplica el cambio tal como se ha indicado anteriormente. Y se procede a correr el programa.

La gráfica que nos arroja el sistema es la siguiente:



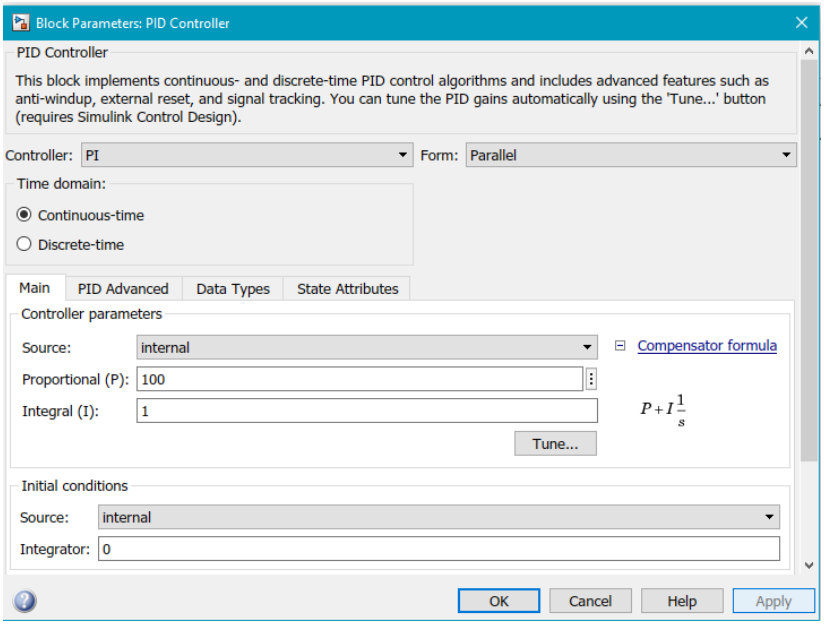
En esta ocasión se puede notar un mayor efecto del controlador en el sistema. Se observa como hay una mayor oscilación para estabilizar el sistema en menor tiempo.

Ahora veremos que pasa si se utiliza una constante de proporcionalidad de 100.

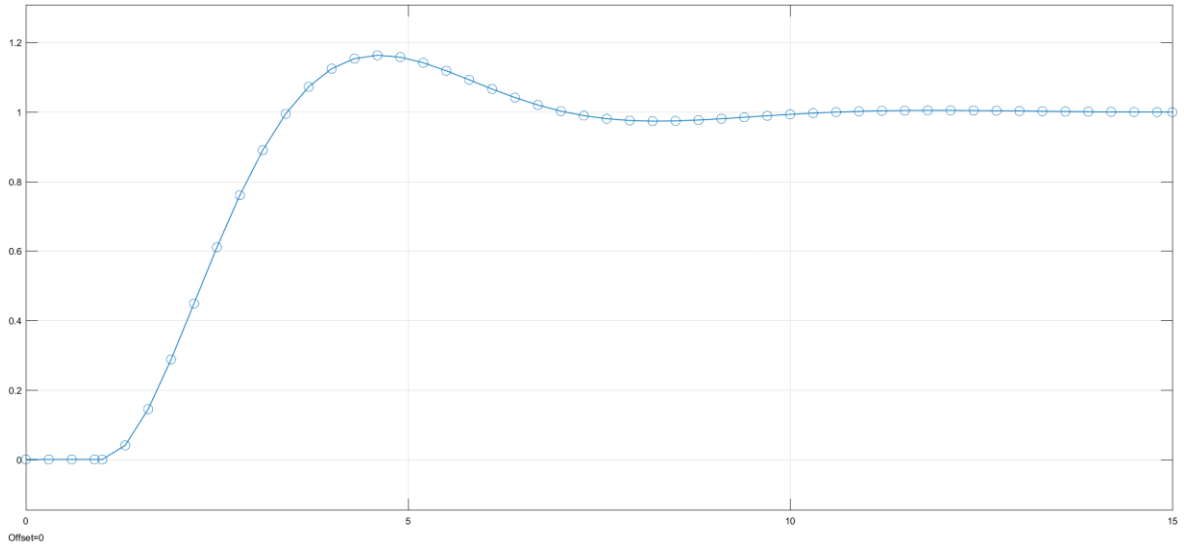


Las oscilaciones son más notorias, el sistema se vuelve estable en menor tiempo. Esto nos deja claro que al aumentar la constante de proporcionalidad se reduce el tiempo de estabilización del sistema a través de oscilaciones dadas por el controlador P.

Ahora toca el turno de observar que sucede si se añade un controlador de tipo PI. Se modificará igual que en el caso del controlador P. Solo se apuntará la modificación de los parámetros del controlador tipo PI.

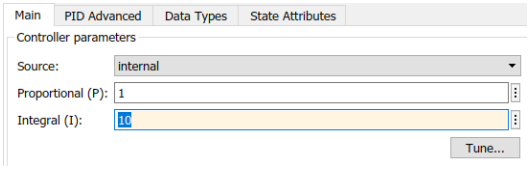


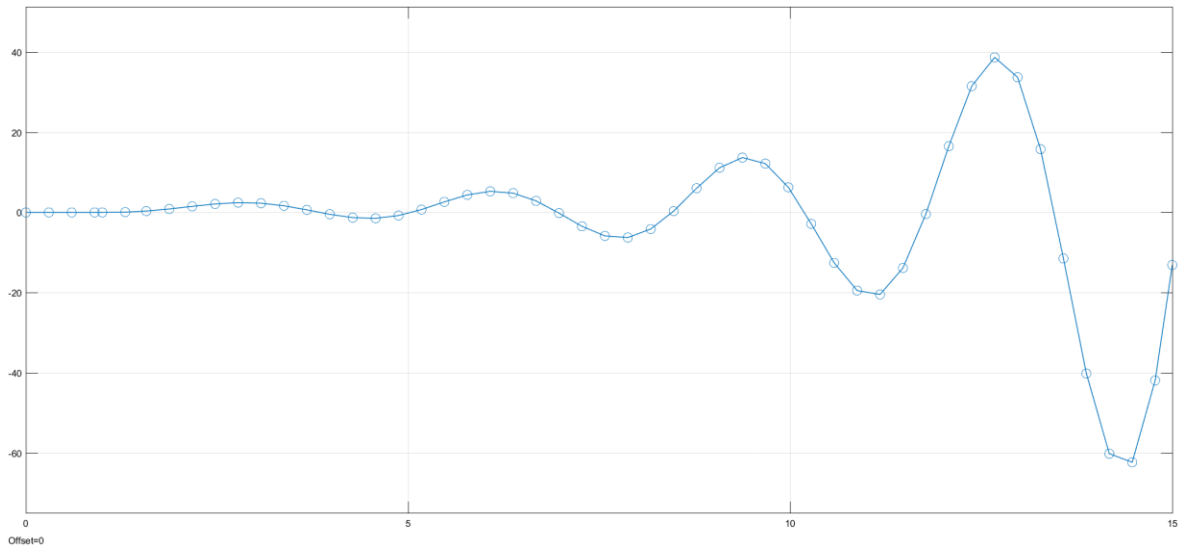
En los parámetros del controlador se observa que la constante de proporcionalidad está en un valor de 100, ya que fue el último que utilizamos en el ejemplo anterior. Éste se cambiará nuevamente a 1, para observar que sucede cuando ambos parámetros son igual a la unidad.



La gráfica con I=1y P=1 se deja ver como a diferencia de cuando solo se contaba con el controlador tipo P, con la parte integral se suaviza levemente la estabilización del sistema enseguida de la oscilación dada por la constante de proporcionalidad.

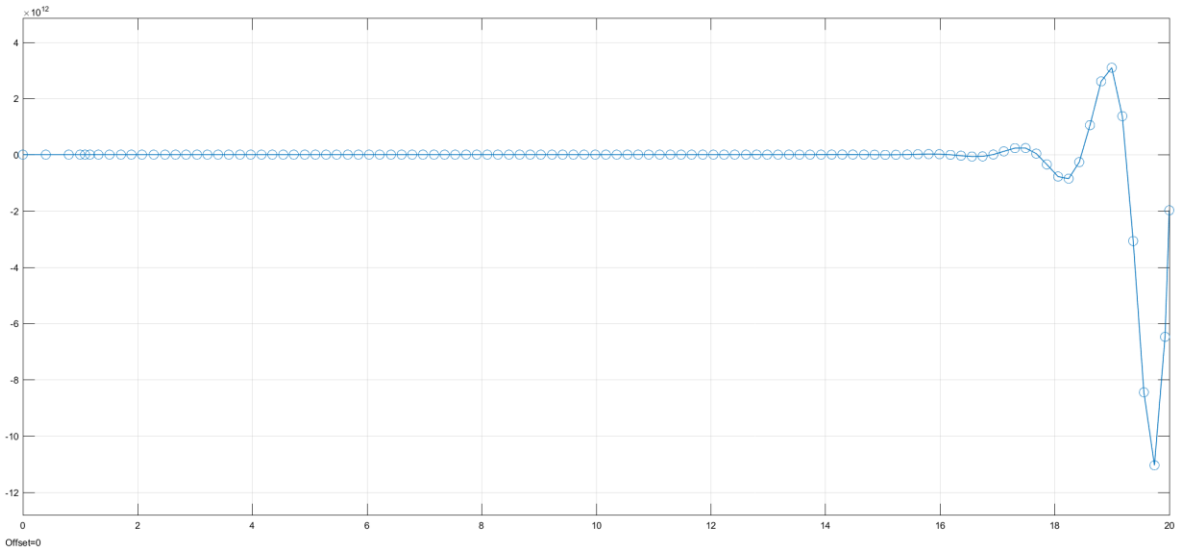
Ahora bien, se modificará la constante integral para ver que sucede cuando se aumenta en 10 veces su valor:





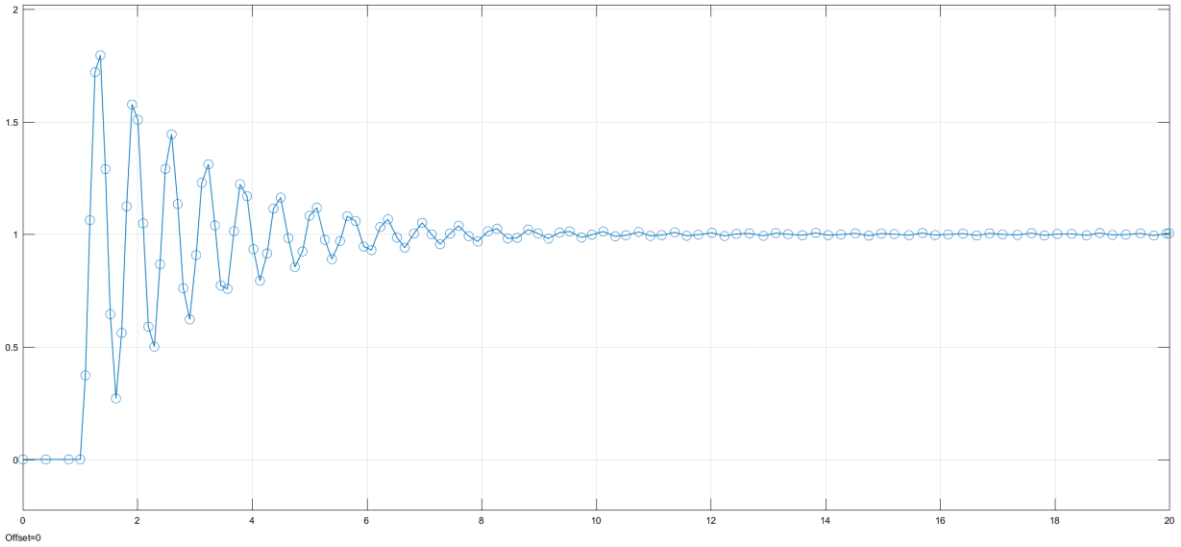
Con la gráfica podemos observar como si se mantiene la constante de proporcionalidad en 1 y se aumenta la constante integral en 10 veces su valor, el sistema parece tomar oscilaciones paulatinas hasta desestabilizarse.

La gráfica con la constante integral en 100 es la siguiente:



En esta ocasión la estabilidad del sistema es por más tiempo, pero se desestabiliza, esta vez con oscilaciones más drásticas.

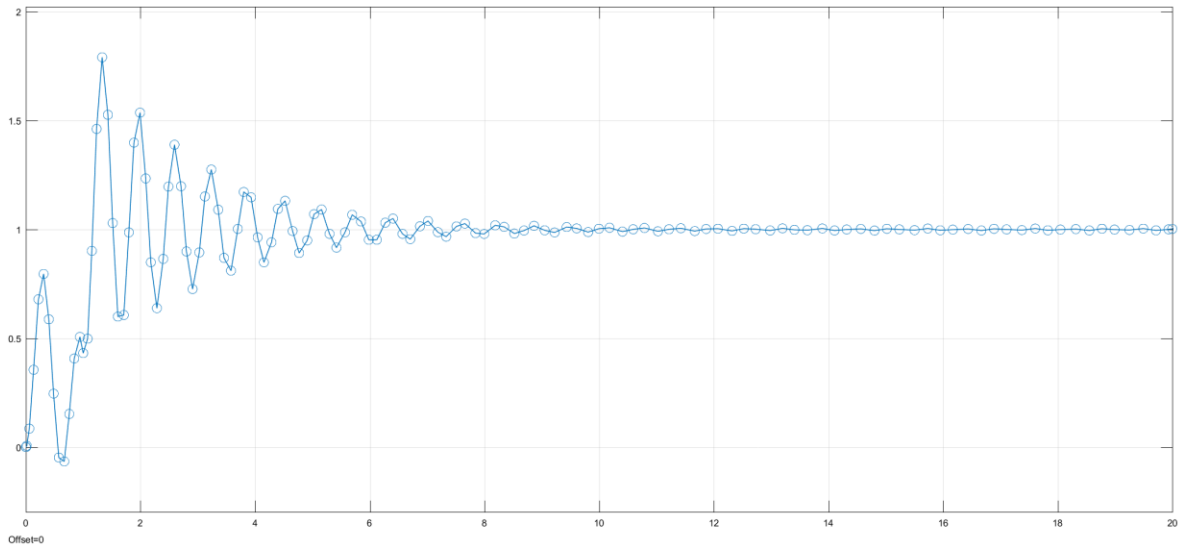
Veremos que pasa si se iguala el valor de la constante de proporcionalidad a la de la integral, en este caso de 100.



Se puede notar como el sistema se estabiliza en un tiempo relativamente corto, sin embargo, lo peculiar es la forma en la que las oscilaciones se van reduciendo de manera uniforme, lo que se traduce en una estabilización más suavizada.

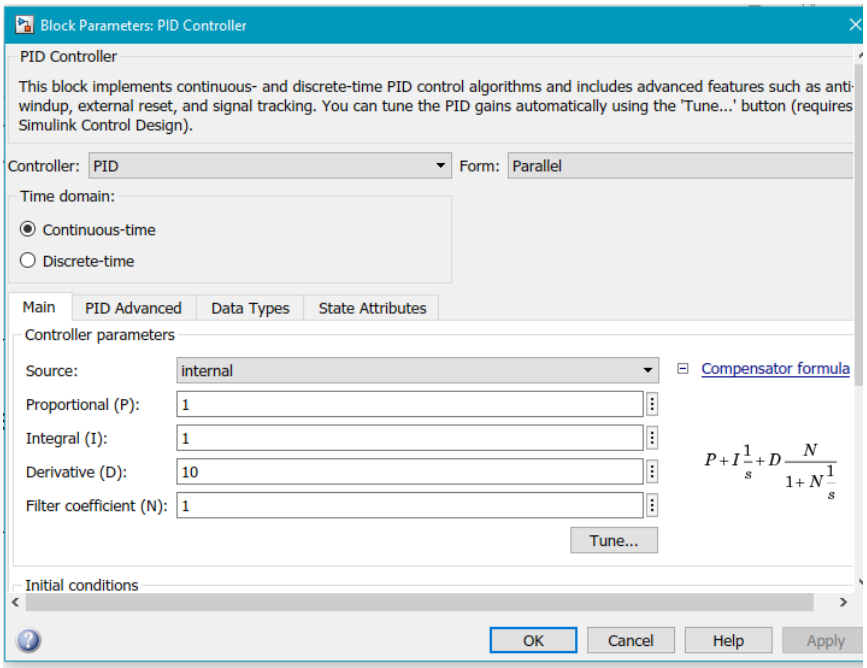
Cabe recalcar, que no se añadió una condición inicial de integración, con el fin de apreciar el efecto de modificar los parámetros básicos de los controladores. Aun así, al añadir la condición inicial de integración, el sistema se desplaza hacia arriba y abajo en el inicio, de manera que se aprecia una especie de impulso, pero luego el comportamiento para la estabilización del sistema no sufre cambios.

A continuación, se muestra un ejemplo con una condición inicial de integración con valor 50 para ejemplificar de mejor manera lo descrito.

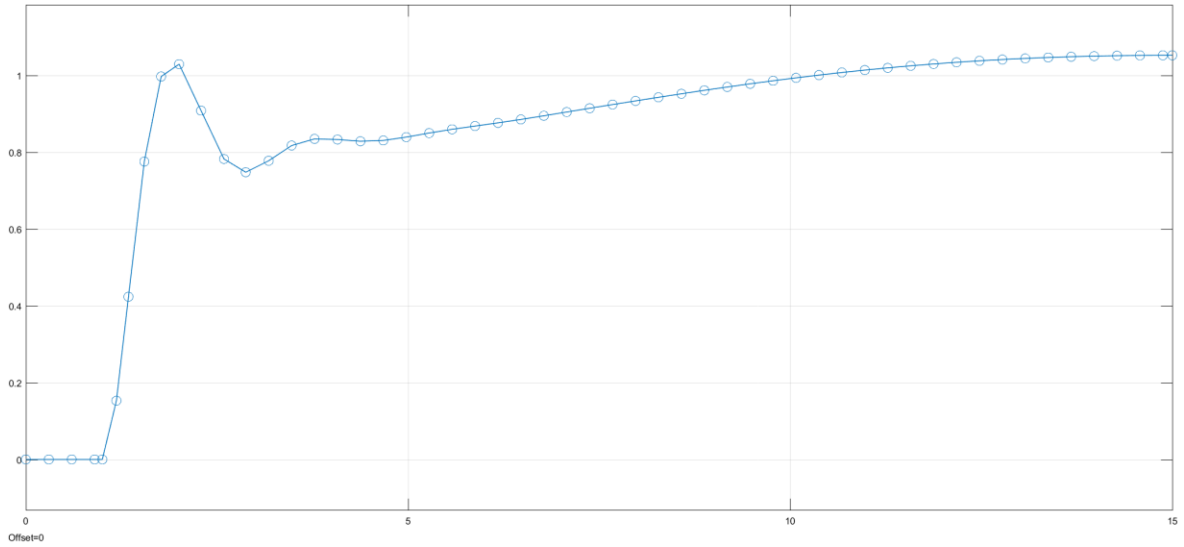


Los valores en las constantes de Proporcionalidad e Integral se mantuvieron en 100. Se nota el impulso que da la constante de integración, y aunque la gráfica se modifica precisamente al principio, véase como la forma de la estabilización se mantiene.

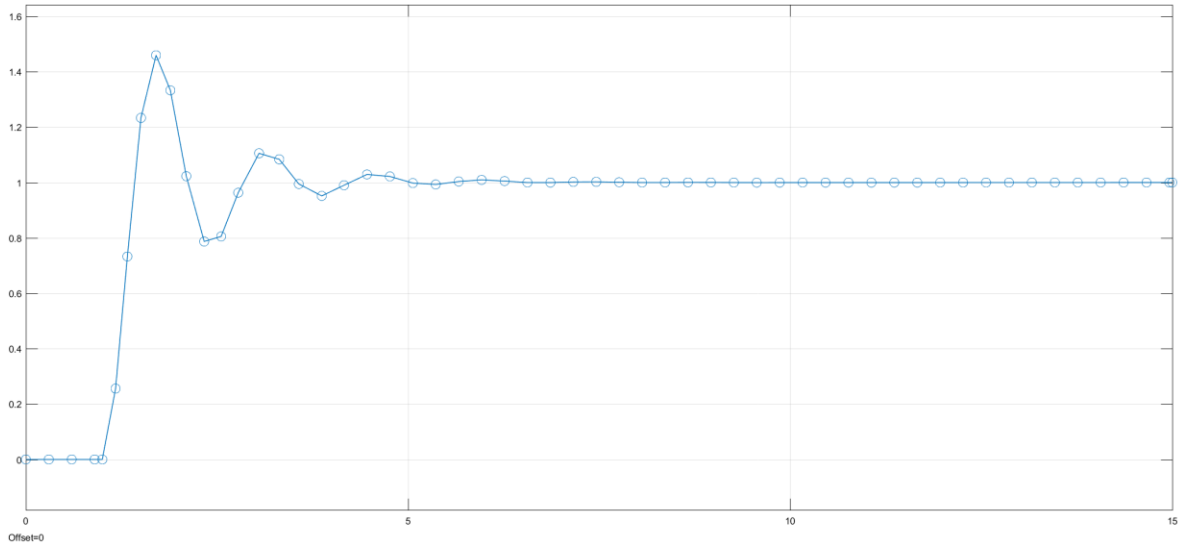
Por último, se cambiará el tipo de controlador por uno PID.



Se colocan todos los parámetros del controlador en 1, a excepción de la constante de derivación, esta se coloca en un valor de 10, para observar cómo influye en el sistema una vez que esta se agrega, y es mayor a las otras dos constantes.

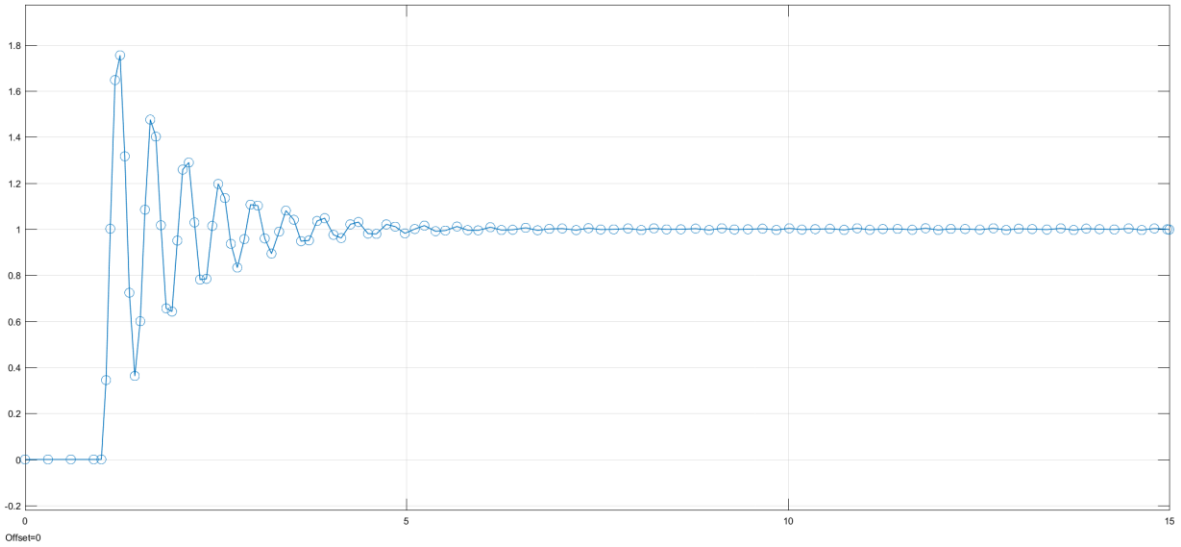


La gráfica es similar a la obtenida con el controlador P, sin embargo, nótese como la estabilización no se da tan pronto como en ese caso. Ahora bien, procederemos a ver que sucede cuando las tres constantes se encuentran en 10.



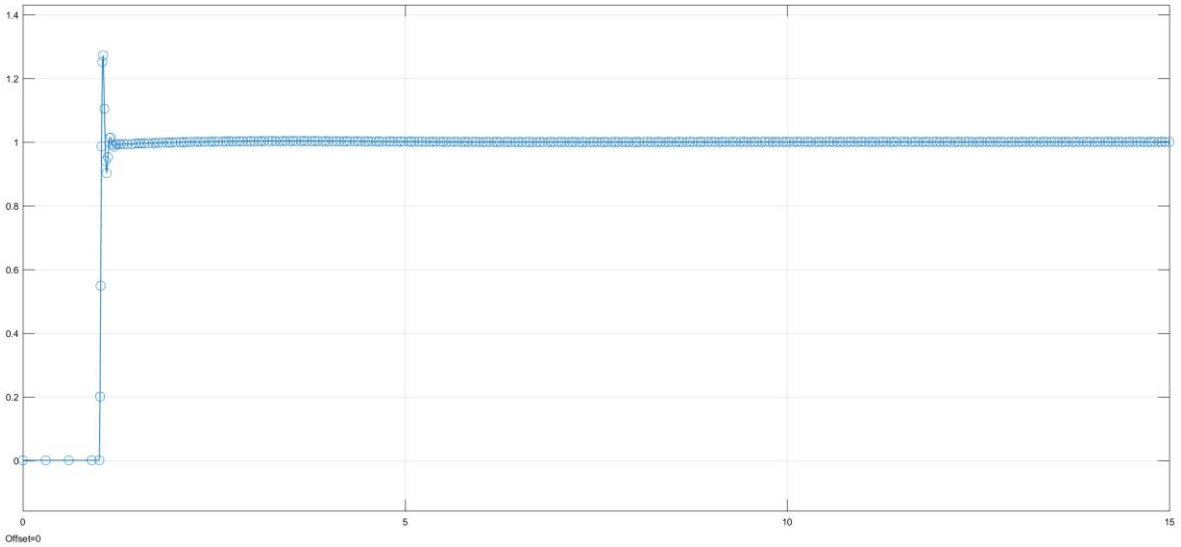
La estabilización se logra de manera más rápida, pero no de forma uniforme, por lo que se sacrifica suavidad en el proceso.

Al aumentar el valor de las tres constantes (P, I, D) en 100, se obtiene la siguiente gráfica:



Vemos como al aumentar el valor de las constantes de manera proporcional, la estabilización es más uniforme, y la rapidez se mantiene.

Al modificar el coeficiente de filtrado con un valor de 50, se obtiene la siguiente gráfica:



Vemos como la estabilización se realiza de manera más rápida, sin embargo, esa uniformidad que se pierde, hace que se vea una gráfica muy brusca.

Se anexan ejemplos para reafirmar lo aprendido. Cada ejemplo contiene un diagrama de Bode que sirve para analizar de manera gráfica el comportamiento en magnitud y fases de cada sistema.

DIAGRAMAS DE BODE

A partir de aquí utilizaré la función que nos ofrece Matlab para graficar los diagramas de Bode, esto con el fin de favorecer al análisis de los ejemplos que se darán, así como los ejercicios encargados.

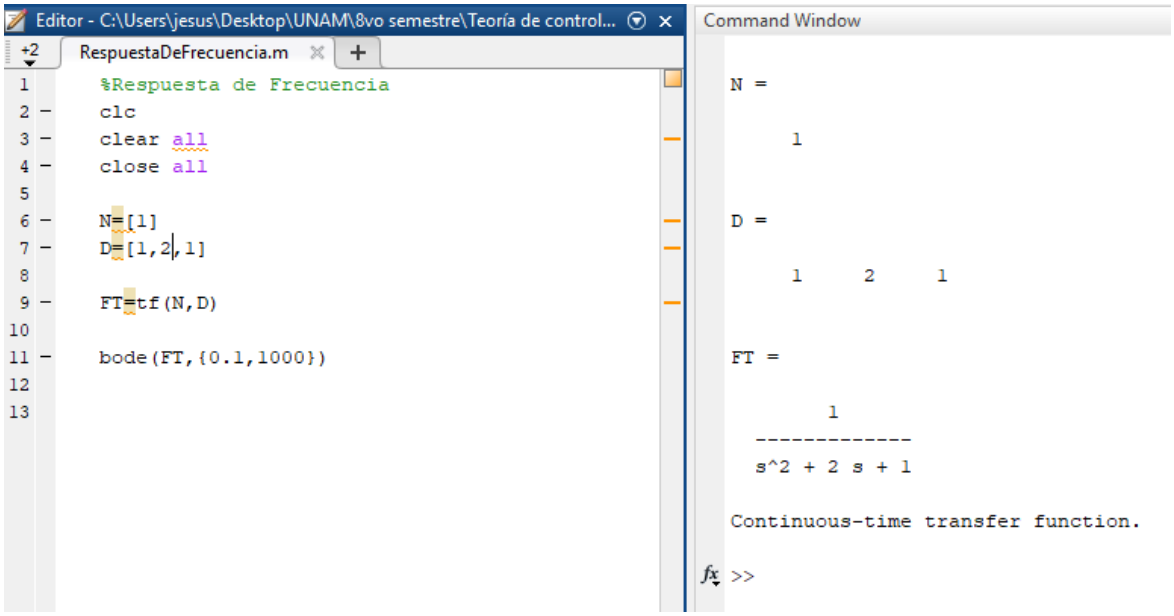
La sintaxis de la función Bode es muy sencilla

bode(FT,{0.1,1000})

FT Representa la función transferencia proporcionada.

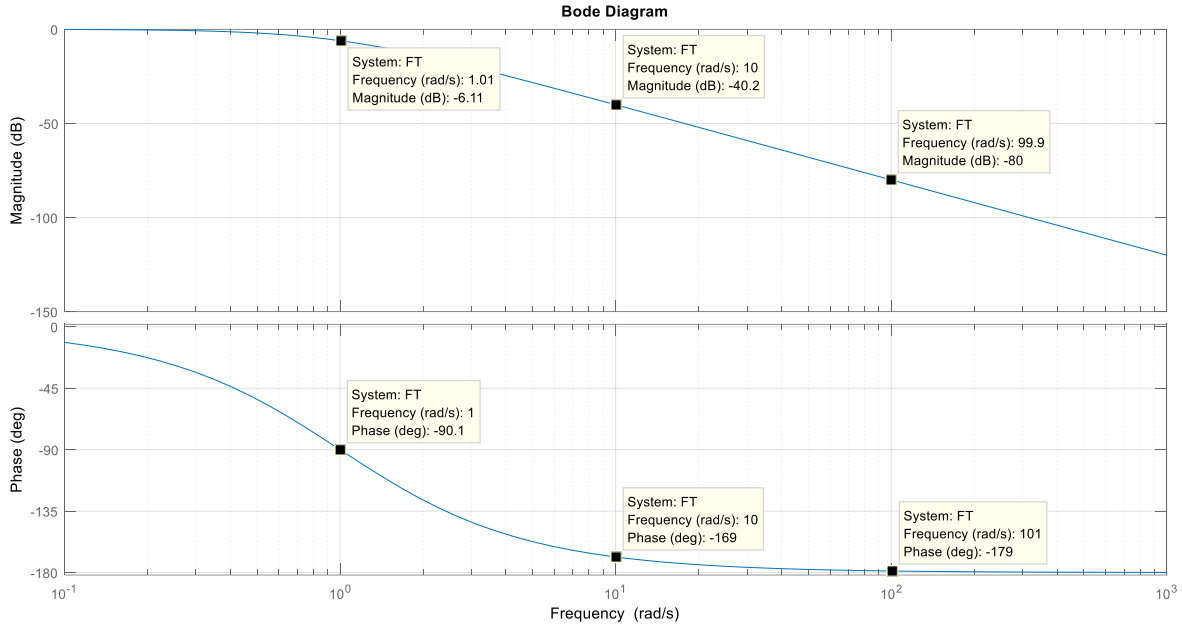
{0.1,1000}) Entre corchetes se escribe el rango de frecuencias deseado. En este caso utilizaremos el rango de 0.1 a 1000.

Se utilizará la función transferencia del ejemplo que se utilizó para desarrollar el manual.



Esto es lo que obtenemos en nuestra ventana de comando al escribir el programa. Se puede ver que la función transferencia es la obtenida. No ahondo en como se ingresa la función transferencia mediante los numeradores y denominadores, ya que ese punto se tocó anteriormente en otro capítulo del manual.

La función bode nos arroja la gráfica, que se muestra a continuación.

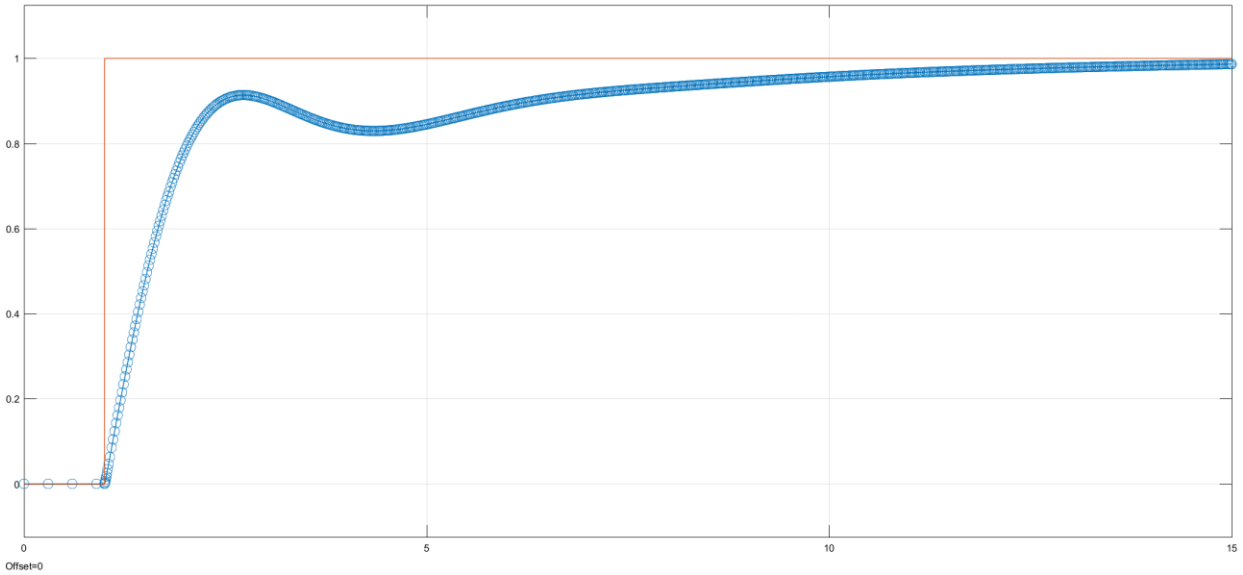


El diagrama de Bode es útil para analizar el comportamiento en magnitud y fase de una función o sistema dado.

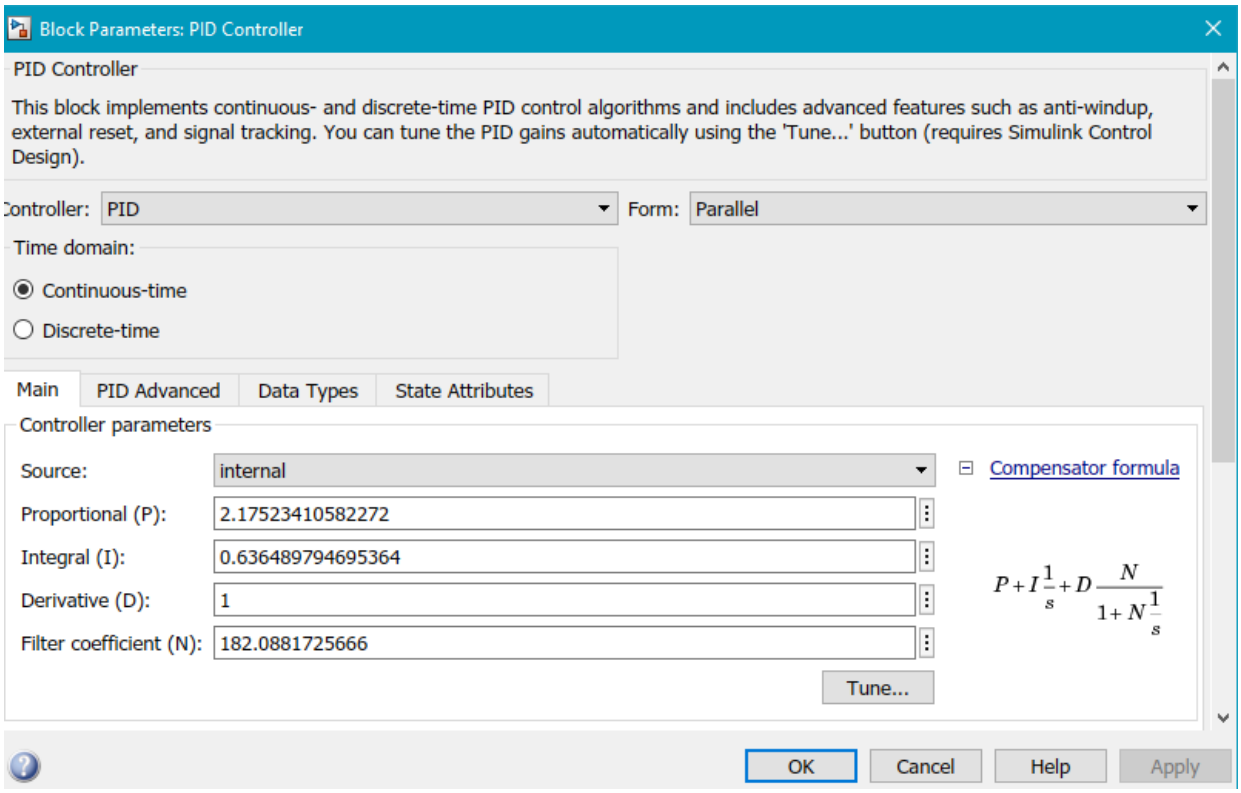
En este caso se observa que cada década, el sistema disminuye 50db. En la primera década hay un desfase de -90°. Parece ser que la señal se estabiliza después de la segunda década.

Ejemplo 2 Sistema con controlador tipo PID

$$G(s) = \frac{s+1}{s^3 + 2s^2 + 2s + 1}$$



La línea color rojo es la función escalón a la entrada.



Los valores de las constantes se obtuvieron al modificar la salida de nuestra señal mediante la herramienta *Tuner* que permite modificar el tiempo de respuesta del sistema, así como el comportamiento transitorio del mismo.

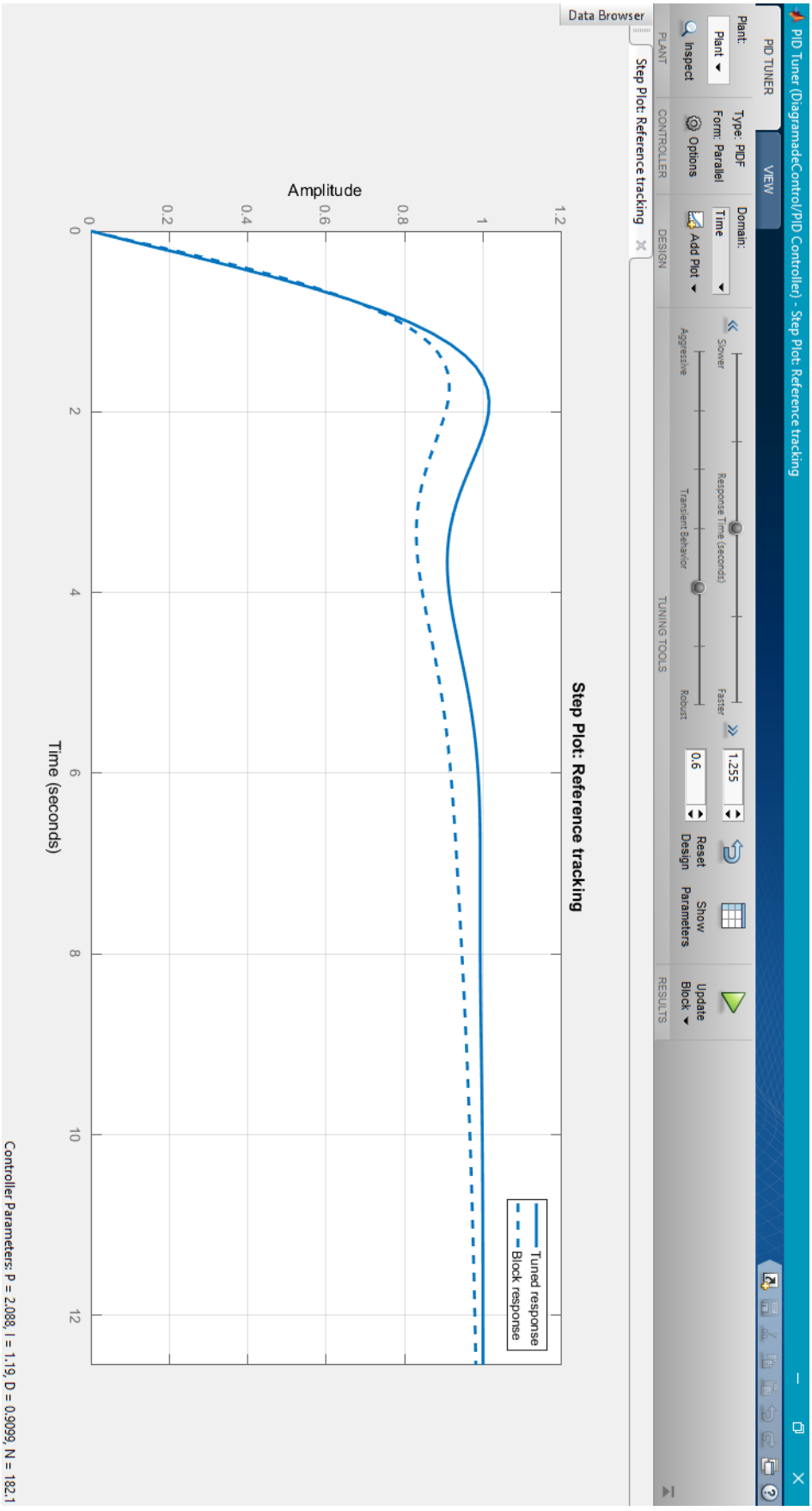


Diagrama de Bode de la función en el Ejemplo 2

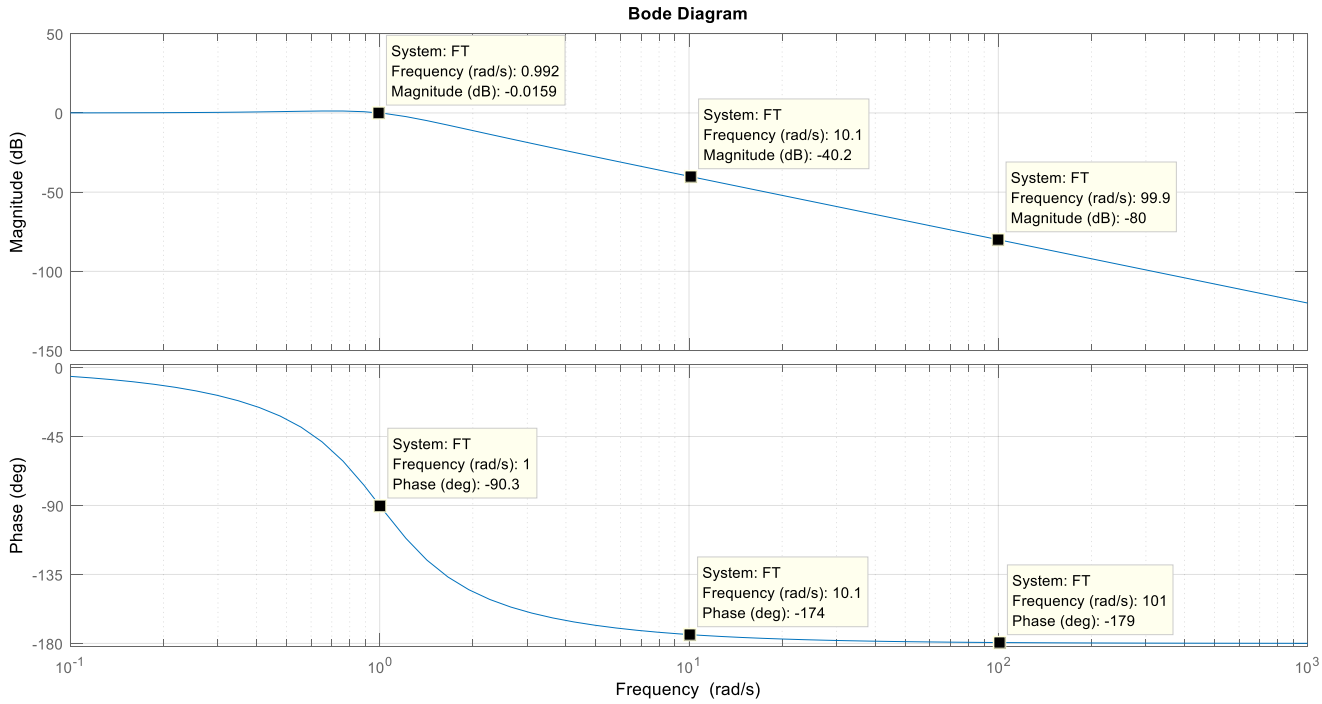
Editor - C:\Users\jesus\Desktop\UNAM\8vo semestre\Teoría de control...

RespuestaDeFrecuencia.m

```
%Respuesta de Frecuencia
clc
clear all
close all

N=[0,1,1]
D=[1,2,2,1]

FT=tf(N,D)

bode(FT,{0.1,1000})
```

Command Window

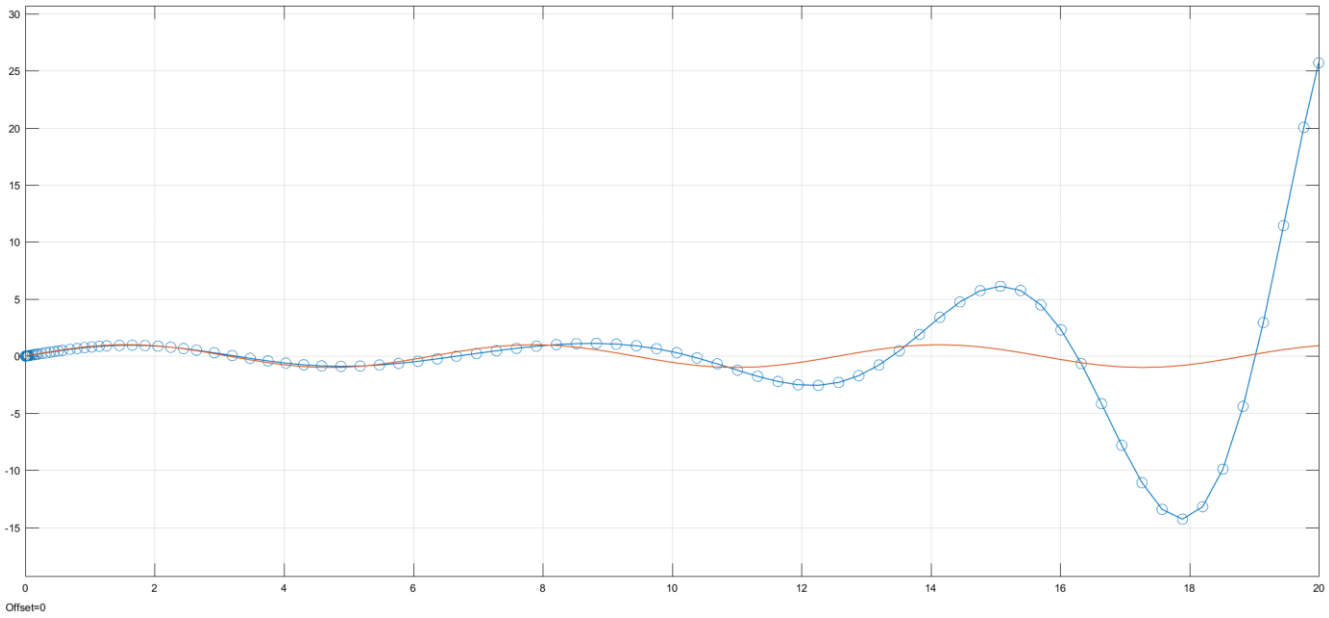
```
N =

     0     1     1


D =

     1     2     2     1


FT =

          s + 1
  ---------------------
  s^3 + 2 s^2 + 2 s + 1

Continuous-time transfer function.
```



Se puede observar cómo cada década el sistema disminuye en magnitud 40 db. En la primera década hay un desfase de 90°. El desfase que se da en cada década se muestra en los cuadros de diálogo en cada punto marcado sobre la gráfica.

Ejemplo 3 Sistema con señal de entrada senoidal y controlador tipo PID

$$G(s) = \frac{s^3 + s + 1}{s^4 + s^3 + 2s^2 + 2s + 1}$$



La línea roja es la señal de entrada. Se puede observar como el sistema parece estable, una señal senoidal atenuada, sin embargo, conforme pasa el tiempo el sistema de desestabiliza.

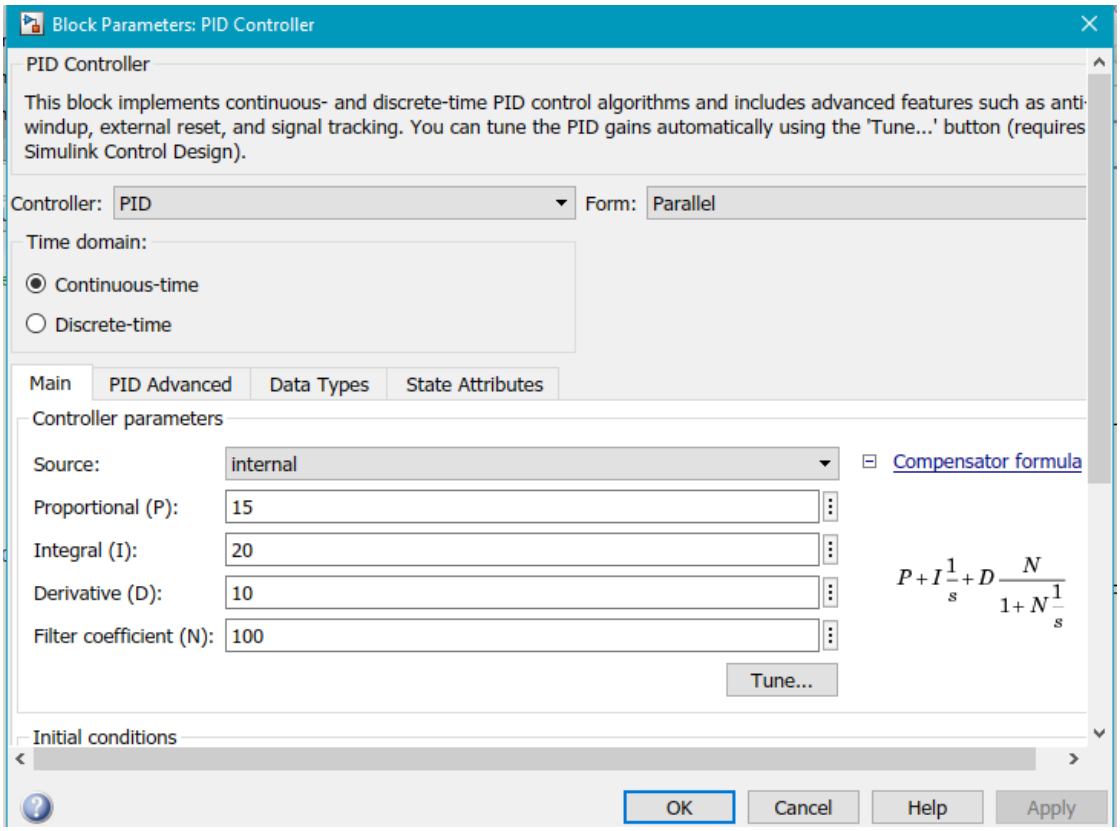
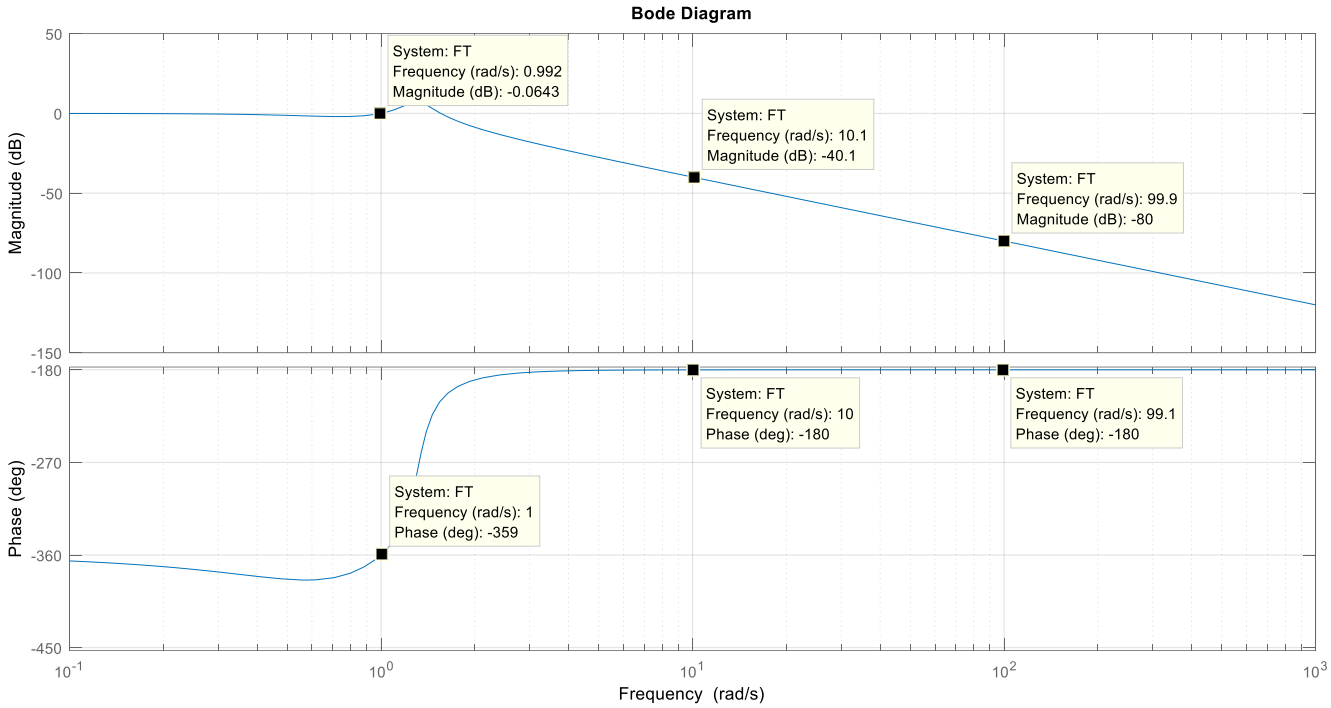


Diagrama de Bode de la función en el ejemplo 3



En cada década el sistema disminuye 40 db. En la primera década hay un desfase de -360, lo cual interpretaría como un giro completo, lo que al final representaría un desfase nulo.

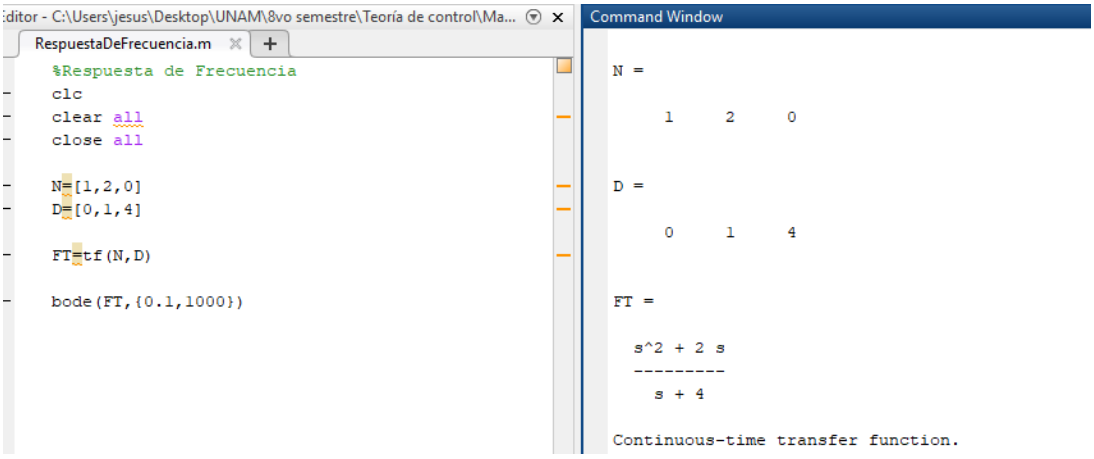
Ejercicio del escrito

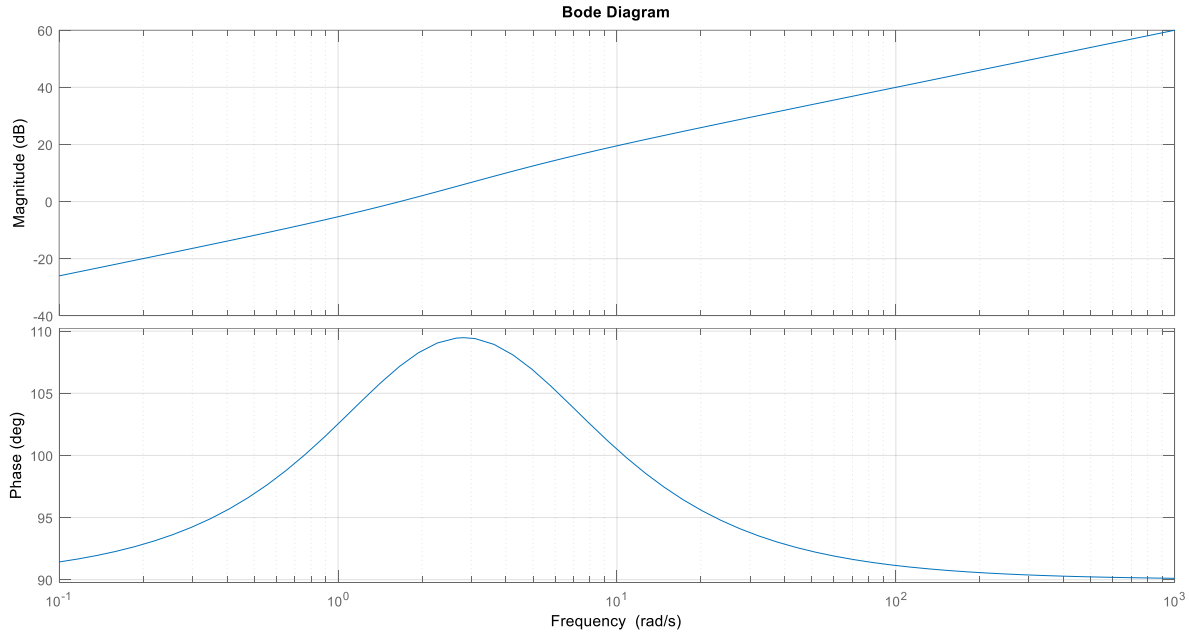
Ingresé a Matlab la función de transferencia proporcionada por el profesor para obtener la gráfica de Bode y compararla con la ya obtenida de forma manual.

$$H(s) = \frac{s(s+2)}{s+4}$$

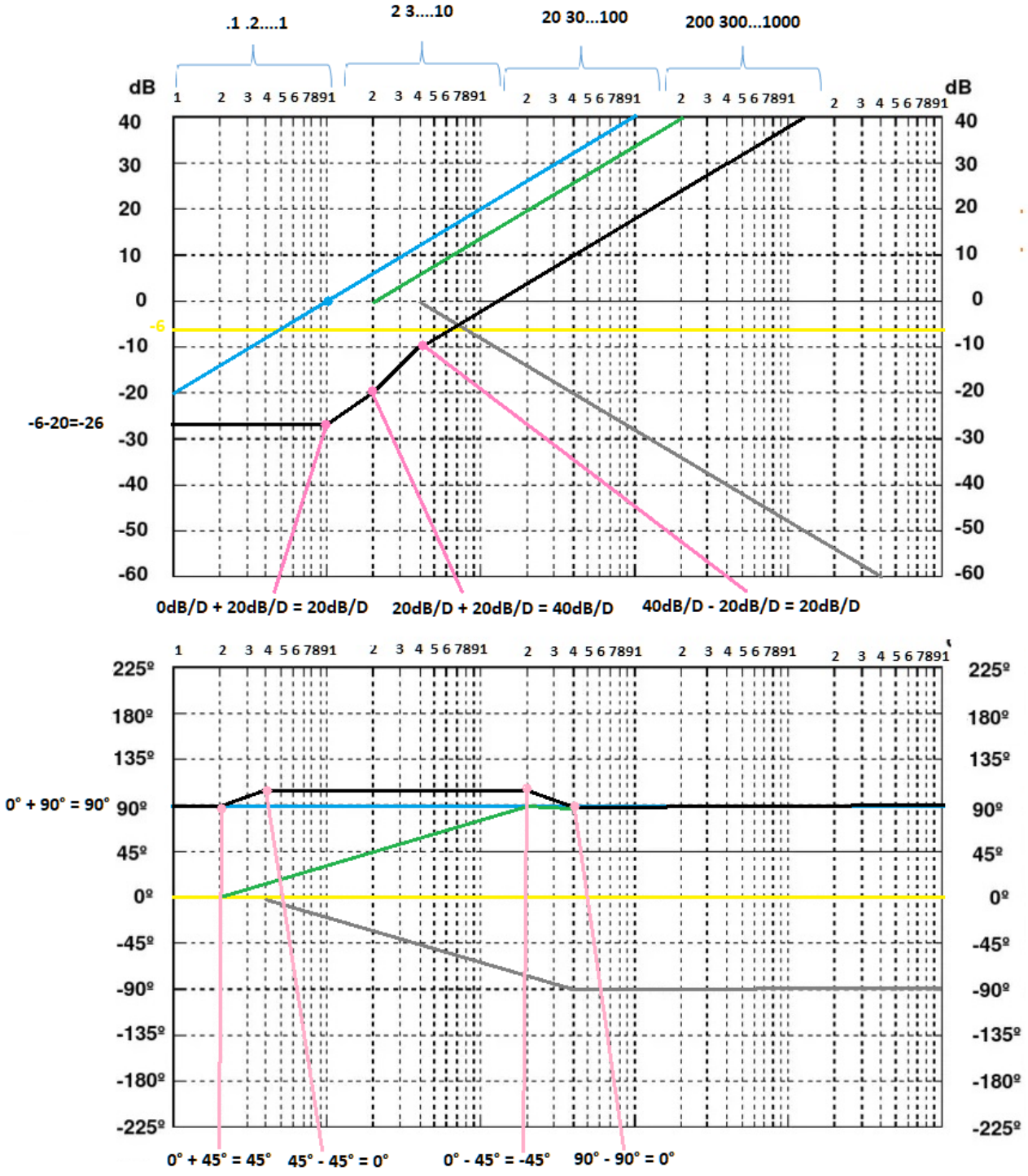
$$H(s) = \frac{s\left(s\frac{2}{2}+2\right)}{\left(s\frac{4}{4}+4\right)} = \frac{2s\left(\frac{s}{2}+1\right)}{4\left(\frac{s}{4}+1\right)} = \frac{0.5s\left(\frac{s}{2}+1\right)}{\left(\frac{s}{4}+1\right)}$$

$$20\log H(s) = 20\log 0.5 + 20\log s + 20\log\left(\frac{s}{2}+1\right) - 20\log\left(\frac{s}{4}+1\right)$$





El diagrama de Bode nos permite observar que, por cada década, el sistema aumenta 20db. En la primera década hay un desfase de 103°. La señal parece atenuarse después de la segunda década.


.1 .2....1
2 3....10
20 30...100
200 300...1000
dB
dB
40
30
20
10
0
-6
-10
-20
-30
-40
-50
-60
-6-20=-26
0dB/D + 20dB/D = 20dB/D
20dB/D + 20dB/D = 40dB/D
40dB/D - 20dB/D = 20dB/D
225º
180º
135º
90º
45º
0º
-45º
-90º
-135º
-180º
-225º
0° + 90° = 90°
0° + 45° = 45°
45° - 45° = 0°
0° - 45° = -45°
90° - 90° = 0°

Ejercicio encargado.

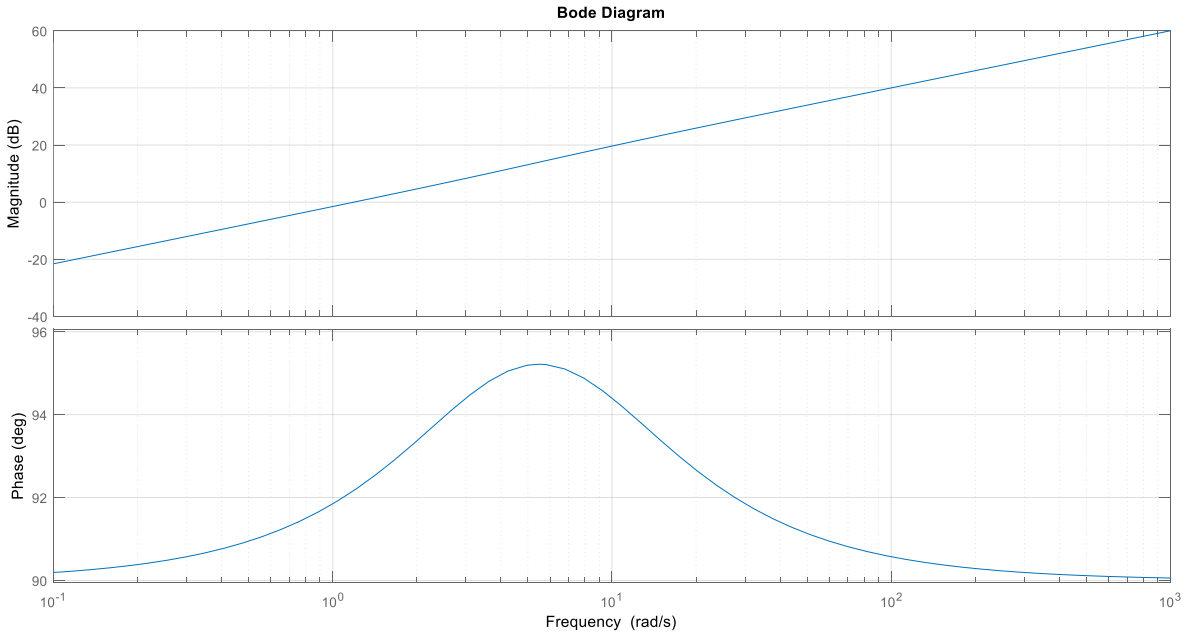
$$H(s) = \frac{s(s+5)}{s+6}$$

$$H(s) = \frac{s\left(s\frac{5}{5}+5\right)}{\left(s\frac{6}{6}+6\right)} = \frac{2s\left(\frac{s}{5}+1\right)}{6\left(\frac{s}{6}+1\right)} = \frac{\frac{1}{3}s\left(\frac{s}{5}+1\right)}{\left(\frac{s}{6}+1\right)}$$

$$20\log H(s) = 20\log\frac{1}{3} + 20\log s + 20\log\left(\frac{s}{5}+1\right) - 20\log\left(\frac{s}{6}+1\right)$$

```
%Respuesta de Frecuencia
clc
clear all
close all

N=[1,5,0]
D=[0,1,6]

FT=tf(N,D)

bode(FT,{0.1,1000})
```

```
N =

     1     5     0


D =

     0     1     6


FT =

  s^2 + 5 s
  ---------
    s + 6

Continuous-time transfer function.
```



El diagrama de Bode nos permite observar que, por cada década, el sistema aumenta 20db. Mientras que hay un desfase de 94° en la primera década. La señal en cada década se desfasa levemente, por ejemplo, en la segunda década, está en más o menos 90.5°, no es mucha la diferencia, pero existe.

Observaciones:

Los ejercicios dados no se pudieron ingresar a Matlab, al intentar ingresar la función de transferencia, el programa arroja un error que señala el numerador es de un grado mayor al denominador. Al resolver por fracciones parciales, no queda una función transferencia adecuada para ingresar a Simulink.

Busqué en la librería de simulink algún elemento que me pudiera permitir ingresar la función dada que tiene polos y ceros. Si hay un elemento llamado *Zero-Polos* pero su sintaxis contiene los ceros y polos de manera inversa en el numerador y denominador a los dados, por lo cual tampoco pude ingresarlos

Lo que si pude hacer mediante Matlab fue el análisis del sistema mediante el diagrama de Bode. Que por cierto, no me parecieron muy similares a las que se nos proporcionó en el escrito “Respuesta en frecuencia”.

ESTABILIDAD GEOMÉTRICO Y LUGAR GEOMÉTRICO DE LA RAÍZ

La estabilidad relativa y la respuesta transitoria de un sistema de control en lazo cerrado están directamente relacionadas con la localización de los polos de su función de transferencia (o las raíces de la función característica) en el plano complejo, por tal razón es necesario analizar el comportamiento de los polos del sistema en lazo cerrado a la variación de los parámetros, en otras palabras, es importante el análisis del lugar geométrico de las raíces del sistema en lazo cerrado.

La técnica del Lugar Geométrico de las Raíces (LGR) es un método gráfico para dibujar la posición de los polos del sistema en el plano complejo a medida que varía un parámetro, la información que proporciona este método es utilizada para el análisis de la estabilidad y funcionamiento del sistema.

El lugar geométrico de las raíces es una gráfica de los puntos del plano complejo que solo satisfacen la condición de ángulo. Las raíces de la ecuación característica (los polos en lazo cerrado) que corresponden a un valor específico de la ganancia se determinan a partir de la condición de magnitud.

Hay distintas maneras de conocer la estabilidad de un sistema, una de ellas es el criterio de Routh Hurwitz, el cual dice que *"Un sistema es estable si todos los elementos que se encuentran en la primera columna de la estructura ordenada tienen el mismo signo siempre, si existe un cambio de signo en tal columna, el sistema es inestable".* Este sistema es muy fiable, sin embargo, utilizando Matlab podemos corroborar la estabilidad o inestabilidad de un sistema a partir de encontrar el Lugar Geométrico de las Raíces, ya que es fácil la construcción de las gráficas mediante este programa.

El Lugar Geométrico de las Raíces puede permitirnos identificar si un sistema es inestable o estable, ya que *"la estabilidad está directamente ligada a los valores de los polos dentro de la ecuación que representa la función de transferencia de un sistema"*. Y *"para que un sistema sea estable, todos los polos deben de centrarse en el lado izquierdo del plano cartesiano".*

Conociendo la primicia anterior, se puede asociar que, al construir la gráfica del Lugar Geométrico de las raíces, se podrán visualizar donde están situados los polos de la función, y de esta manera identificar si es estable o inestable.

Para encontrar el valor geométrico de las raíces en Matlab, nos ayudaremos de un tema tocado anteriormente en la primera parte de este manual: la función transferencia.

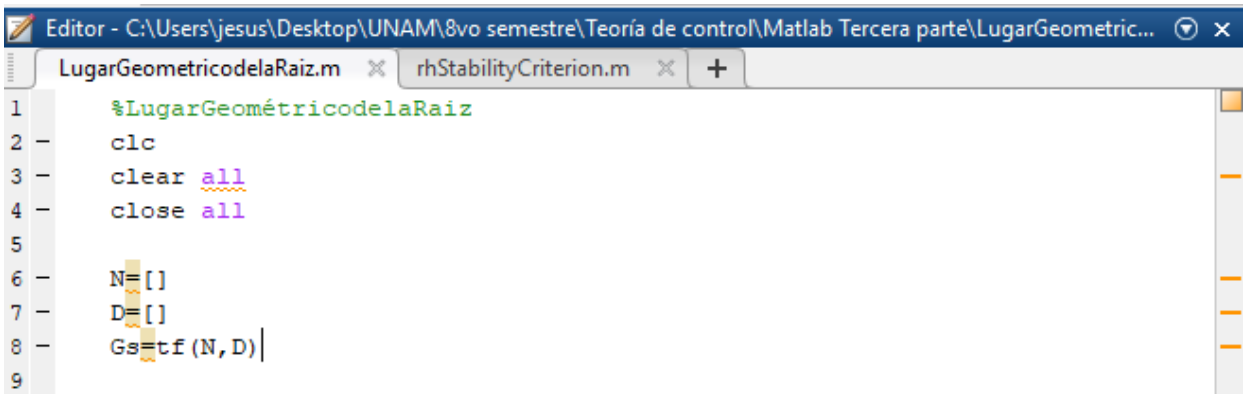
La función transferencia consta de un numerador y de un denominador, ya que el resultado es un cociente.

1. Primero que nada limpiar el editor con los comandos que ya conocemos "clc", "clear all" y "close all".

2. Definir el numerador y el denominador de nuestra función en el programa con **N** y **D.**

3. Utilizar el comando **tf** para definir a nuestra función transferencia. Se puede llamar a la función de cualquier manera, en este caso la señalaremos con **Gs**.

Esto nos ayudará a que la función transferencia sea apreciada de mejor manera al correr nuestro programa.



4. Una vez que se añadieron esos comandos (ya conocidos por haber sido utilizados anteriormente en el manual), se procede a añadir el comando que nos permitirá encontrar el Lugar Geométrico de las Raíces. La sintaxis es la siguiente: **rlocus(Gs)**

rlocus es el comando que arrojará, gráficamente, el lugar geométrico de las raíces.

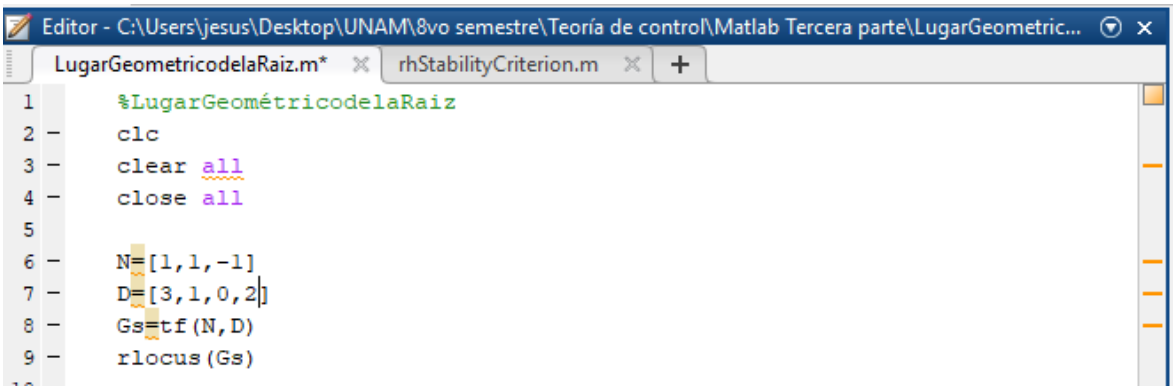
Gs es la función transferencia introducida, compuesta de un numerador y un denominador.

Al ejecutar el programa, se mostrará en una gráfica el Lugar Geométrico de las Raíces que tiene el sistema, además de que identificando donde se encuentran los polos, podremos saber si el sistema es estable o inestable.

Ejemplo 1: Se tiene la siguiente función transferencia:

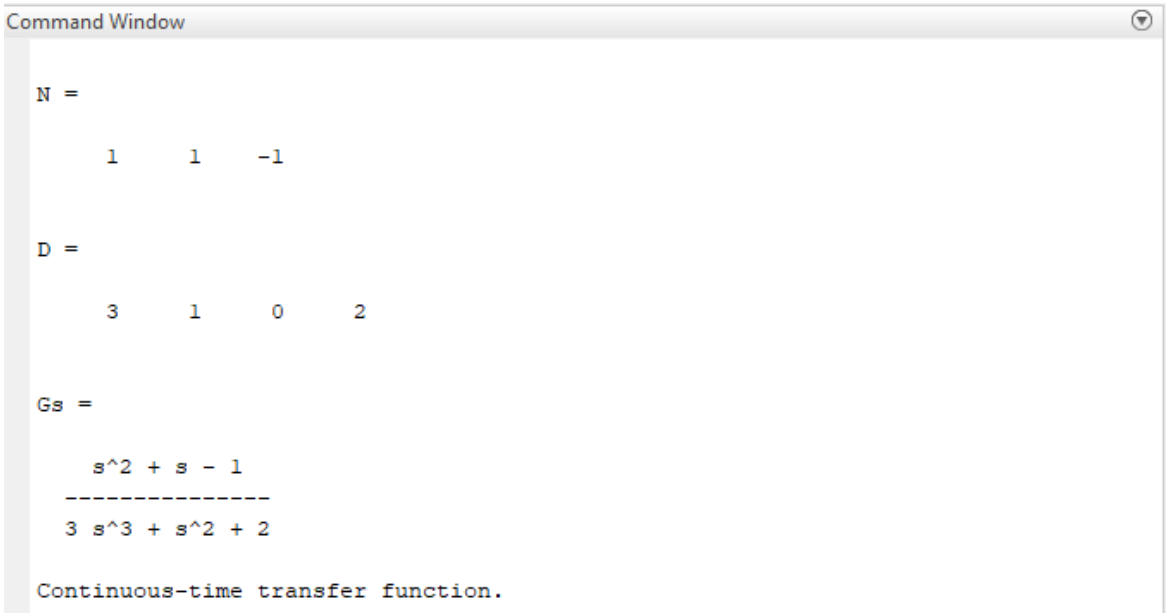
$$\frac{s^2 + s - 1}{3s^3 + s^2 + 2}$$

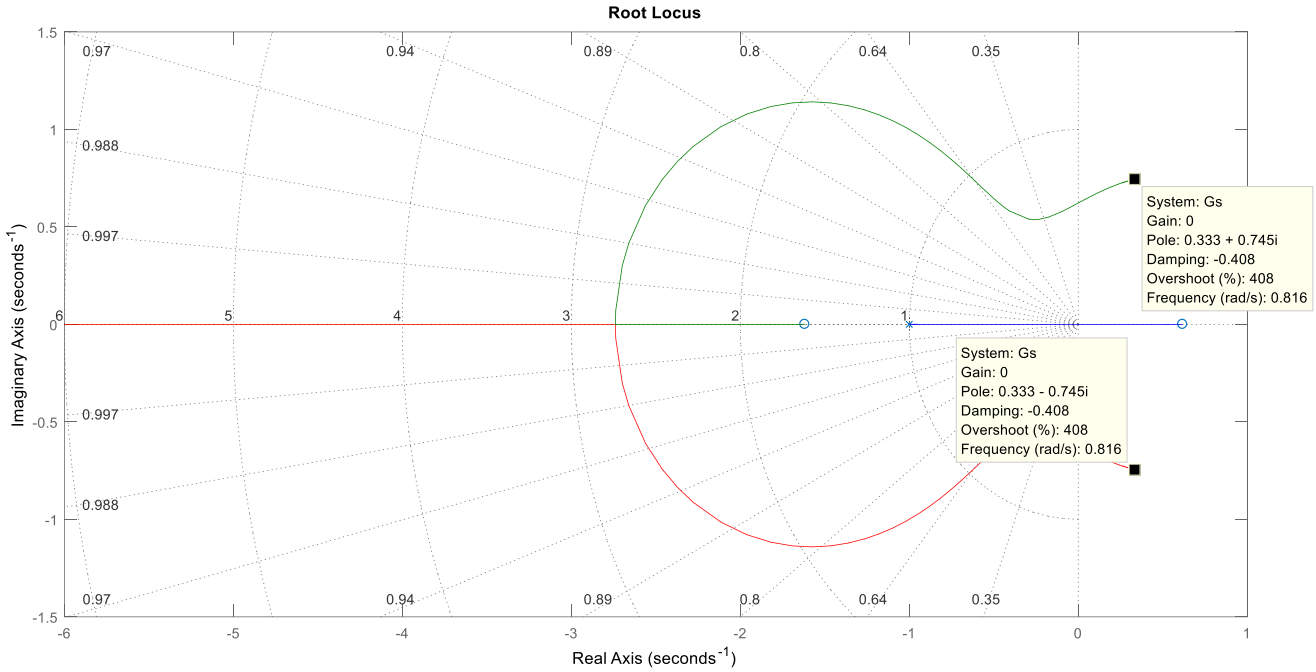
Recordemos que los datos del numerador y denominador, se introducen separados por una coma, y los grados de la función que no tengan coeficientes, se rellenan con cero.



Como se puede observar, en el denominador no se tiene un coeficiente de grado 1, es decir **s**, al introducir la función en Matlab, se colocó un cero, para que el programa lo reconozca.

Una vez introducida la función, se corre el programa y se aguarda por la gráfica que nos mostrará el Lugar Geométrico de las Raíces.





Cómo se puede apreciar, al dar clic sobre las líneas de las asíntotas, éstas nos arrojan información del punto señalado por el puntero, esto se puede hacer con las dos asíntotas al mismo tiempo.

Estos datos que nos arroja en el recuadro color crema se definen de la siguiente manera

System En este apartado va a mencionar el nombre del sistema que se está representado, en este caso es Gs, ya que así llamamos a nuestra función.

Gain Es la ganancia del sistema, ésta aumenta o disminuye según se desplace el punto sobre la asíntota.

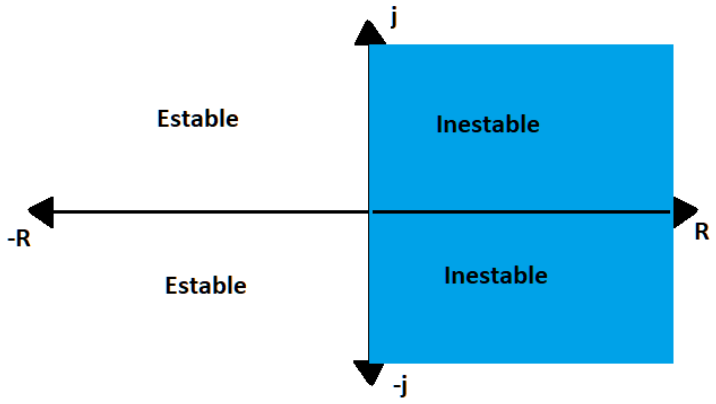
Pole Son los polos de la función. En el ejemplo se puede notar que dos asíntotas comparten el elemento real, mientras que en el caso del imaginario tiene ambos el mismo valor, pero signo contrario, lo que confirma la simetría de las asíntotas.

Damping Es el amortiguamiento del sistema. Este valor no es mayor a 1.

Overshoot Indica la desviación del sistema respecto al eje real R, en porcentaje. Siendo la menor desviación 0%, esto se ubica sobre el eje real R.

Frequency Es la velocidad angular del sistema, en radianes por segundo. Para obtener grados, se multiplica lo obtenido por un factor de 57.29578. Tomando la gráfica del Ejemplo 1, se puede ver que la frecuencia es de 0.816 rad/s. Al multiplicar por el factor de conversión tendríamos que son 46.7533 grados/segundo.

Podemos observar que el sistema es **inestable,** ya que dos de los tres polos en la función se encuentran en el lado derecho del plano cartesiano, lo cual indica que lo es.



Corroboraremos la inestabilidad del sistema aplicando el criterio de Routh Hurwitz con ayuda de Matlab.

```
Command Window
input vector of your system coefficients:
 i.e. [an an-1 an-2 ... a0] = [3 1 0 2]

 Routh-Hurwitz Table:

rhTable =

     3     0
     1     2
    -6     0
     2     0

~~~~~> it is an unstable system! <~~~~~

 Number of right hand side poles = 2
Do you want roots of system be shown? Y/N
```

Obsérvese que en la primera columna hay dos cambios de signo, de positivo a negativo y nuevamente a positivo, por lo cual, el sistema efectivamente es **inestable.**

A continuación, se muestran otros ejemplos.

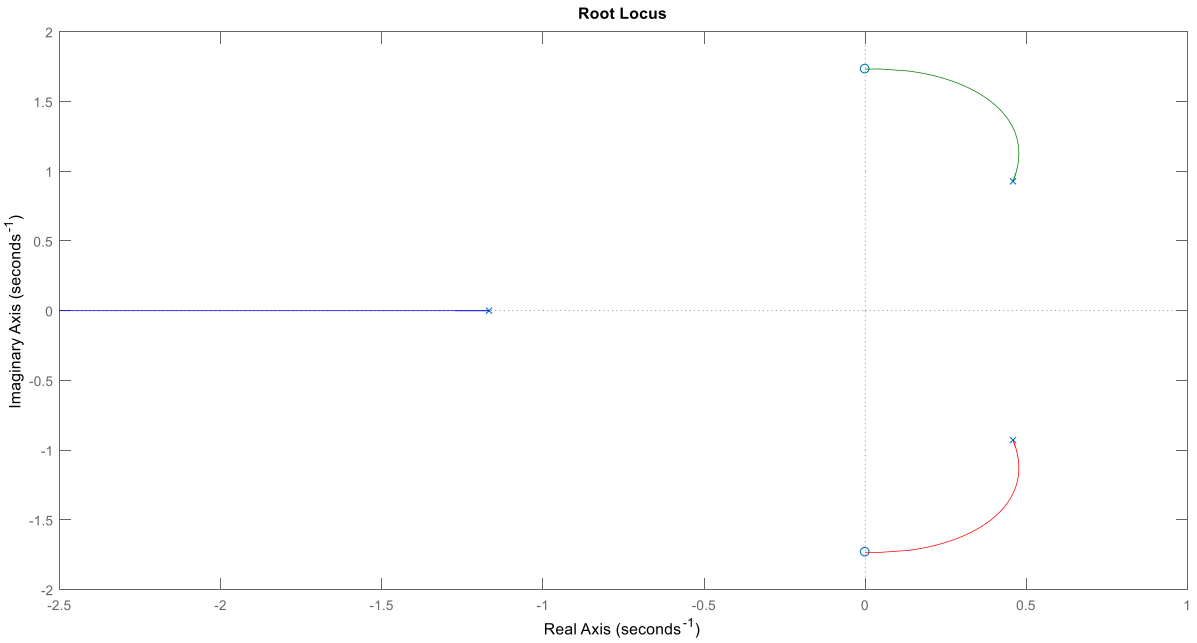
Ejemplo 2: Se tiene la siguiente función transferencia:

$$G(S) = \frac{s^2 + 3}{4s^3 + s^2 + 5}$$

```
Editor - C:\Users\jesus\Desktop\UNAM\...
LugarGeometricodelaRaiz.m
1   %LugarGeométricodelaRaiz
2   clc
3   clear all
4   close all
5
6   N=[1,0,3]
7   D=[4,1,0,5]
8   Gs=tf(N,D)
9   rlocus(Gs)
10
```

```
Command Window

N =

     1     0     3


D =

     4     1     0     5


Gs =

      s^2 + 3
  ---------------
  4 s^3 + s^2 + 5

Continuous-time transfer function.
```

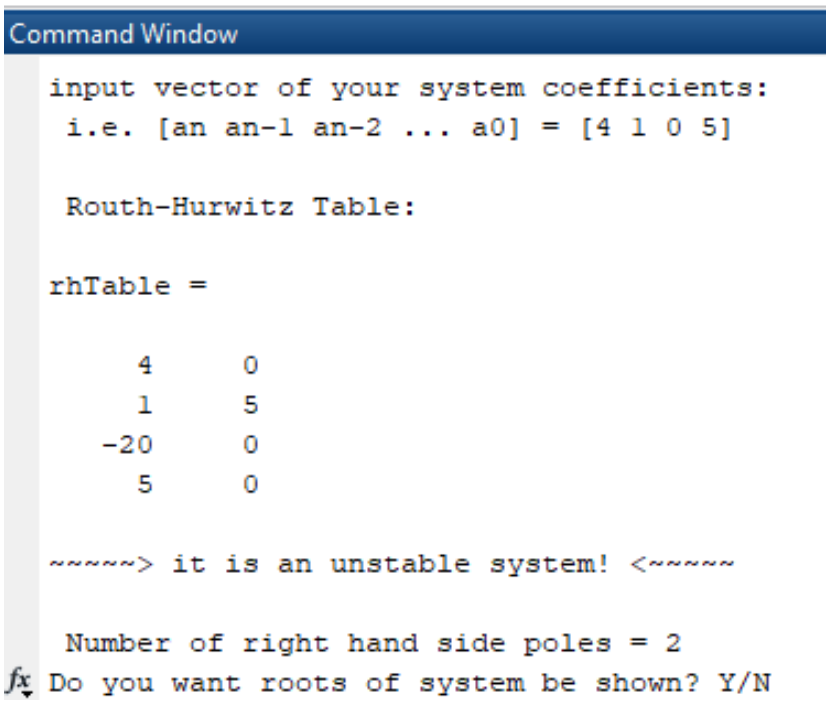
Con la simple función transferencia nos podemos dar cuenta que posee 3 polos y 2 ceros. La gráfica generada por el programa es la siguiente:



Los ceros están representados en la gráfica por los círculos, y los polos son las *x*.

A simple vista se puede observar que el sistema es **inestable,** ya que dos de los tres polos en la función se encuentran en el lado derecho del plano cartesiano.

Se corrobora la información aplicando el criterio de Routh Hurwitz.

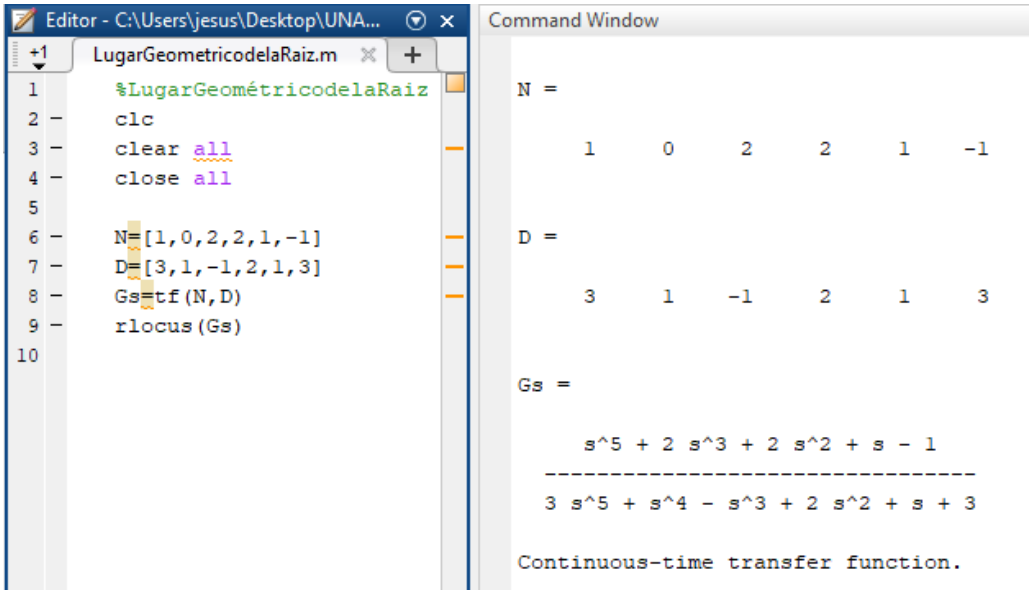


Igual que en el ejemplo anterior, se observa el cambio de signo en la primera columna, de positivo a negativo y nuevamente a positivo.

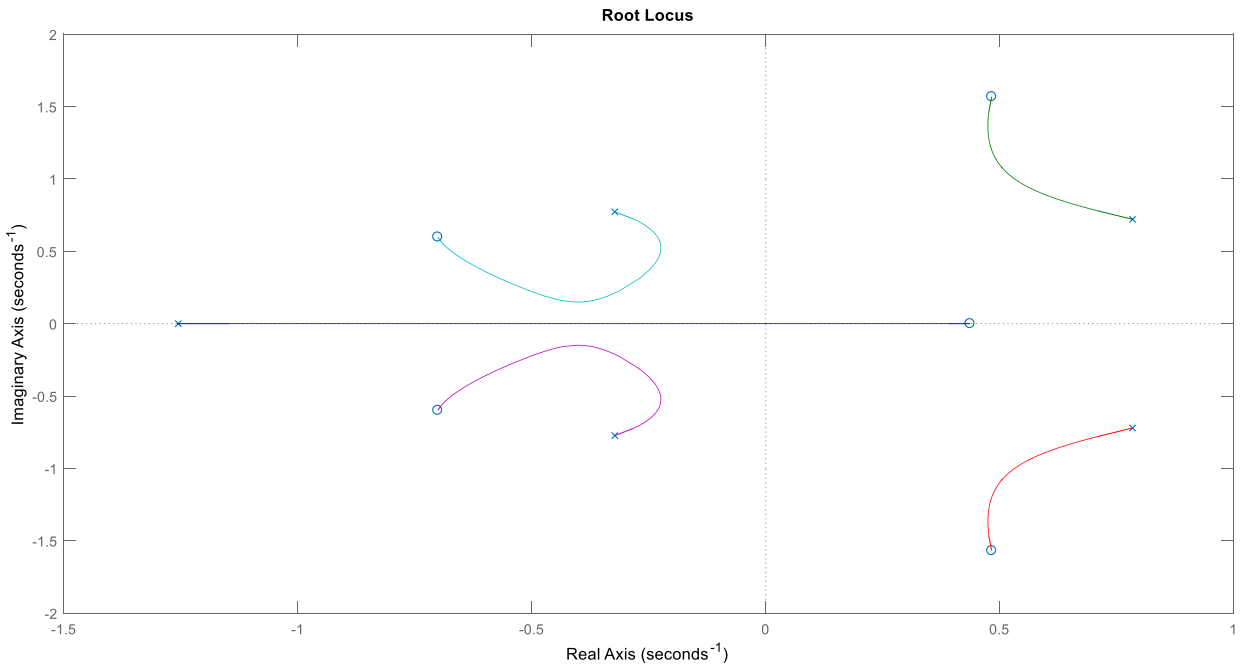
Ejemplo 3: Se tiene la siguiente función de transferencia:

$$G(S) = \frac{s^5 + 2s^3 + 2s^2 + s - 1}{3s^5 + s^4 - s^3 + 2s^2 + s + 3}$$

En este último ejemplo, se analizará la gráfica del Lugar Geométrico de las Raíces para ver de que manera se comportan y que información podemos obtener a partir de ellas, además de la estabilidad o inestabilidad del sistema.



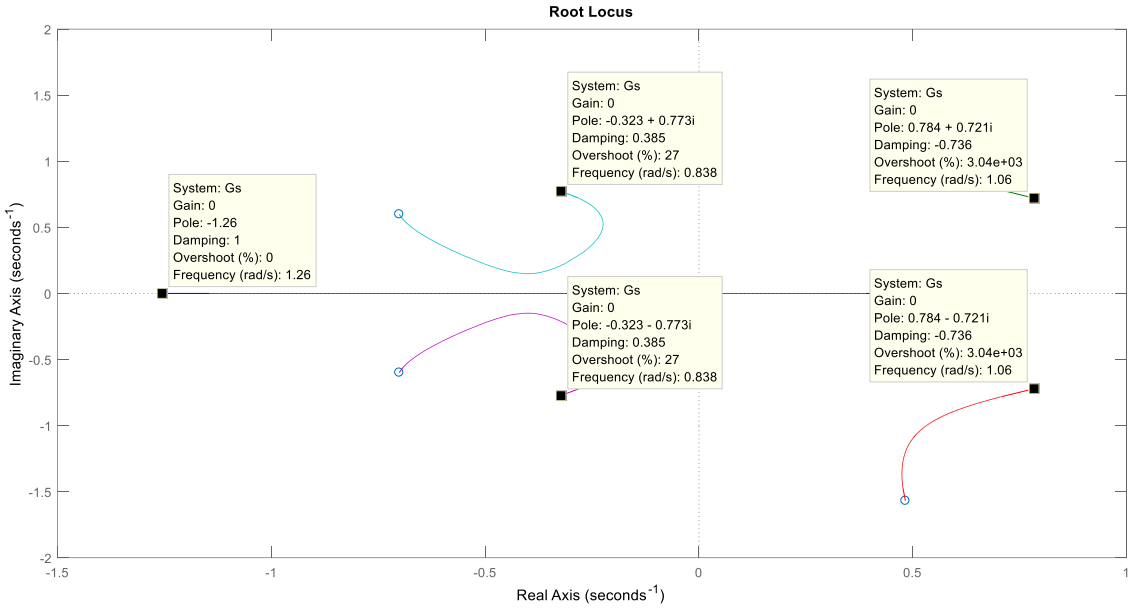
La gráfica de la función es la siguiente:



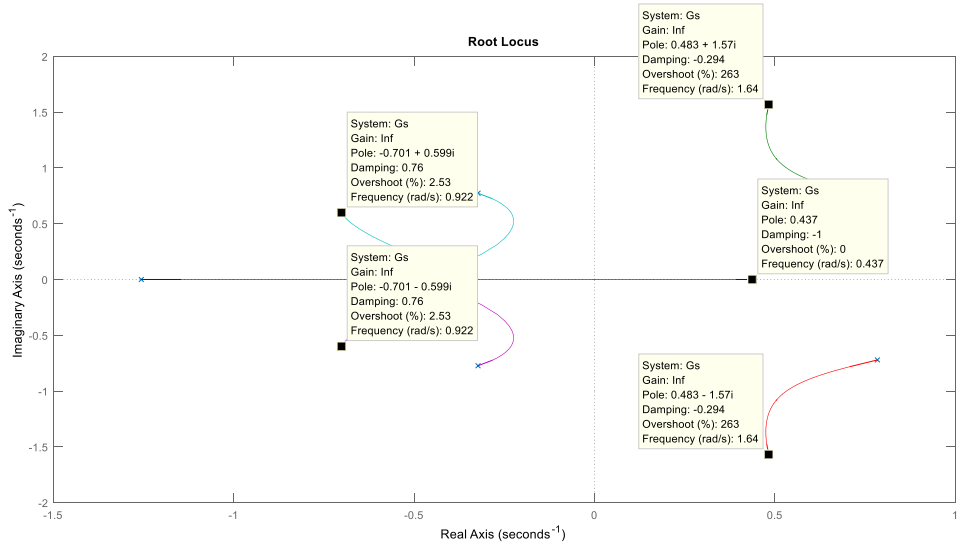
Análisis del Lugar Geométrico de las Raíces

La trayectoria del LGR empiezan en los polos de lazo abierto. Como se puede ver en la gráfica la ganancia en estos puntos es de 0. También se puede observar los polos en las asíntotas son simétricas.

Solamente un polo se encuentra sobre el eje real.



Las trayectorias del LGR sobre el eje real (el de las abscisas) existen entre 0 y $-\infty$.



Todas las trayectorias de LGR tienden a infinito son 5, es decir, todos los ceros de la función, ya que no existen ceros infinitos.

En la imagen anterior están seleccionados los 5 ceros del sistema, se observa que la ganancia tiende a infinito en estos puntos.

Solamente un cero se encuentra sobre el eje real, que está del lado derecho.

Este sistema no tiene punto de quiebre, ya que no existe trayectoria entre dos polos o dos ceros reales.

Al no haber punto de quiebre, el sistema tampoco cuenta con ganancia de quiebre.

Para obtener la ganancia crítica se aplica el criterio de Routh Hurwitz a la ecuación característica. Sin embargo, en este caso la ganancia está dada por la función característica:

$$3s^5 + s^4 - s^3 + 2s^2 + s + 3$$

El término independiente 3 es la ganancia del sistema.

Aplicar el criterio de Routh Hurwitz a la ecuación, nos demostrará la presencia de la ganancia K y también si el sistema es estable o inestable.

```
Command Window
input vector of your system coefficients:
 i.e. [an an-1 an-2 ... a0] = [3 1 -1 2 1 3]

 Routh-Hurwitz Table:

rhTable =

    3.0000   -1.0000    1.0000
    1.0000    2.0000    3.0000
   -7.0000   -8.0000         0
    0.8571    3.0000         0
   16.5000         0         0
    3.0000         0         0

~~~~~> it is an unstable system! <~~~~~

 Number of right hand side poles = 2
fx Do you want roots of system be shown? Y/N
```

Al aplicar el criterio de Routh Hurwitz se observa que el sistema es **inestable**. En la primera columna se observa el cambio de signo de positivo a negativo y nuevamente a positivo.

Obsérvese que hay un elemento subrayado en azul, el 3. Esta es la ganancia crítica del sistema, esta se encuentra en la tercera columna antes de que se haga cero el coeficiente.

Otro dato que nos arrojó, es que hay dos polos que se encuentran en el lado derecho del plano, es decir del lado positivo de los reales. Y si recordamos que la estabilidad está ligada a los valores de los polos en la función, pudimos darnos cuenta desde obtener la gráfica que el sistema sería inestable, ya que se observan los polos que están en el lado derecho “inestable”.

FUENTE(S):

Herramienta *Help* del Programa Matlab 2017.