UNIVERSIDAD VERACRUZANA
FACULTAD DE INGENIERIA MECANICA ELECTRICA

Manual de Simulacion para Sistemas de Control usando el Software Matlab y Scilab.

TRABAJO DE RECOPILACIÓN.

PARA LA MATERIA DE SISTEMAS DE CONTROL

EN LA CARRERA DE

INGENIERO MECÁNICO ELECTRICISTA.

Que presenta:

César Leyva Rodríguez

H. Boca del Río, Veracruz. **Marzo del 2007.**

Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Prefacio.

En este trabajo se presenta una descripción de las principales herramientas en Matlab y Scilab, para analizar y simular sistemas lineales, vistos en los últimos semestres de la carrera de Ingeniería Mecánica Eléctrica.

La Ingeniería de Control es sin duda una disciplina en la que los alumnos encuentran problemas para asimilar los conceptos teóricos explicados en clase. En parte, esta situación está motivada por la necesidad de introducir un número elevado de conceptos. Esto hace que en ocasiones la motivación del alumno descienda. En esta situación se hace muy importante el disponer de una herramienta de simulación, y cómo ésta constituye uno de los pilares en el aprendizaje de la materia de sistemas de control, este trabajo pretende ser una herramienta de fácil manejo y permitir al estudiante ejercitar los conceptos teóricos vistos en clase. Mediante la simulación se pretende que el alumno ponga en práctica los conceptos teóricos estudiados. El objetivo es poder plantear diferentes experiencias y obtener resultados de forma rápida para poder contrastarlos con lo que predice la teoría.

Para lograr esta simulación se cuenta con el rápido desarrollo que se tiene en las computadoras, lo que ha conducido a que aparezcan en el mercado los más diversos programas adaptados al cálculo y la simulación, ejemplos de estos tenemos a Matlab el cual ha sido utilizado por años cómo una gran herramienta en el análisis matemático, pero con el inconveniente de traer consigo un costo comercial, en la otra parte tenemos el caso de Scilab que por ser un producto de software 'libre' de recién empleo en los sistemas de control, carece de documentación sencilla y en nuestro idioma.
Este trabajo también esta motivado cómo una opción para que el estudiante logre reforzar los conceptos visto en clase durante la carrera de Ingeniería Mecánica Eléctrica y con una base firme en Sistemas de Control pueda tener un buen comienzo en cursar alguna maestría relacionada con la materia.

Este trabajo también es una extensión de los trabajos realizados de:
Tutorial On Line de Matlab y Simulink
Autor : Inmaculada Olivares Olivares
Director : Juan Domingo Aguilar Peña
Fecha : Junio 2003

Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Introducción:

En el Capítulo I de esta manual, llamado 'Introducción a Matlab' se comienza explicando el entorno de Matlab, los comienzos históricos, y se efectúa la presentación de las principales partes de Matlab. Mostrando la forma de utilizar la ayuda con el fin de que el estudiante vallé explorando por cuenta suya cuanto desee aprender de Matlab, desde el comienzo.
Se explica cómo utilizar los comandos para cambiar los formatos, la descripción de algunas variables utilizadas por Matlab y la importancia de la sensibilidad al uso de mayúsculas en Matlab.
Se introduce al álgebra vectorial y se efectúan declaraciones de matrices y operaciones de éstas, como son cambio de valores de columnas y filas, obtención de solución de ecuaciones lineales entre otros.
Se muestra la generación de argumentos, creación de polinomios y obtención de raíces así cómo operaciones realizadas con estos.
En la parte de generación de gráficos, se explica detalladamente la forma de crear gráficas y se da un ejemplo de la grafica obtenida de la función 'seno' a través de un programa escrito en la ventana de Matlab.
En el Capítulo II llamado 'Herramientas para Sistemas de Control en Matlab' se introduce al manejo de las funciones de transferencia dentro del entorno de Matlab, se muestran algunos ejemplos de reducción de diagramas de bloques con retroalimentación, para después pasar mostrar los análisis de las respuestas de las funciones de transferencia ante las entradas impulso, escalón e impulso, el lugar geométrico de las raíces de los polinomios también es mostrado en Matlab y con ejemplos se muestran los lugares de los polos y ceros en las gráficas realizadas.
Una vez que el estudiante ya se encuentra familiarizado con el uso de Matlab se da parte al comienzo del Capítulo III dedicado a 'Simulink' en el describe la creación de diagramas de bloques en Simulink para pasar con la inserción de bloques y manipulación de estos con el fin de poder realizar los ejemplos de sistemas de control que se plantean de manera muy similar a la que usarían los estudiantes cuando resuelven los problemas vistos en clase.

El Capítulo IV corresponde a la 'Introducción a Scilab' se introduce el concepto del término de "Software Libre", mostrando los cuatro principios en que se basa este, explicando de una manera clara las diferencias del término en nuestro idioma y el idioma Ingles. Se muestra la manera en que los estudiantes pueden obtener el software y las principales diferencias con Matlab, de la misma manera que en el 'Capítulo I' se explica el uso de las variables, la declaración y operaciones con matrices, formatos de comandos para graficar, declaración y manejo de los polinomios.
En el Capítulo V llamado 'Herramientas de Sistemas de Control' se muestran la declaración de las de las funciones de transferencia, la obtención del lugar geométrico de las raíces y la obtención de las respuestas ante las entradas impulso, escalón y rampa.

Una vez que el estudiante se encentra familiarizado con Scilab se comienza con la creación de los diagramas de bloques a partir del simulador de Scilab, al cual esta dedicado el Capítulo VI llamado 'Scicos', en este se muestra las diferentes tipos de paletas con que dispone Scicos y se

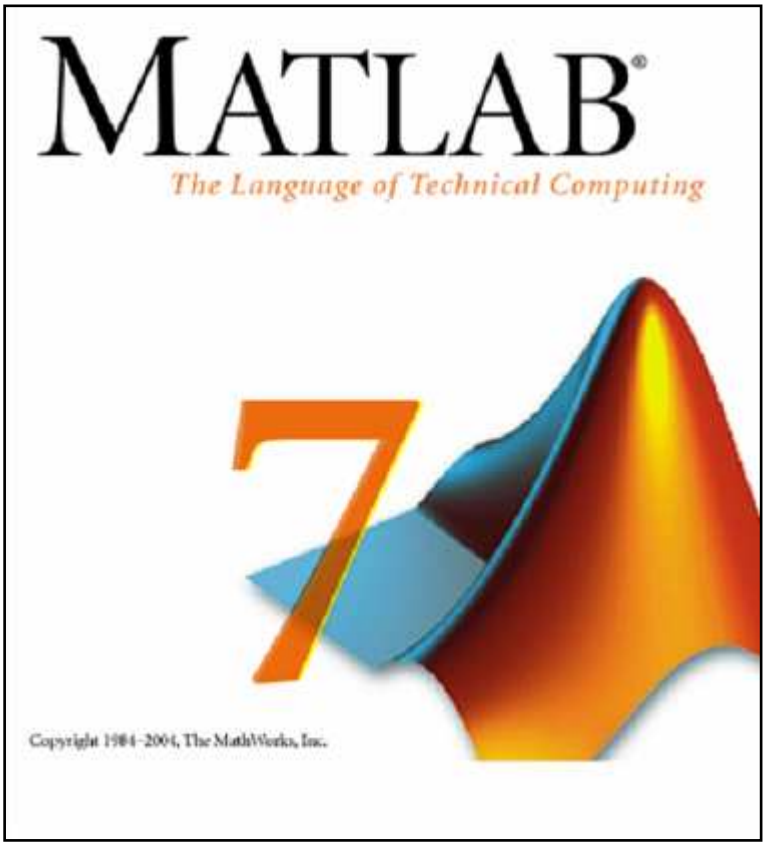
muestra la realización de simulación de un sistema mecánico compuesto por una masa, amortiguador y resorte.

Para finalizar se presentan los Capítulos VII 'Caso de estudio con Matlab' y CapituloVIII 'Caso de estudio con Scilab' correspondiendo ambos, al estudio de un circuito eléctrico de tercer orden integrado por resistencias y capacitores, el cual es analizado y desarrollado por medio del método de Ziegler y Nichols para obtener un controlador PID y mejorar su respuesta ante una entrada escalón, una vez obtenidos los valores de las constantes del controlador PID es simulado en Simulink y Scicos con el objetivo de mostrar que es posible obtener el mismo resultado en cualquiera de los dos simuladores.

Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

ÍNDICE:

Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.



Matlab es un entorno de computación y desarrollo de aplicaciones totalmente integrado orientado para llevar a cabo proyectos en donde se encuentren implicados un gran número de cálculos matemáticos y se requiera la visualización gráfica de los mismos, Matlab integra un análisis numérico, cálculo matricial, proceso de señales y visualización gráfica[1].

Introducción a MATLAB

Él nombre de Matlab proviene de la abreviación inglesa Matrix Laboratory, (Laboratorio de Matrices) y su código en principio fue escrito para proporcionar un fácil acceso al software de matrices desarrollado en los años 70's, dentro de los proyectos Linpack y Eispack, que fueron librerías desarrolladas en Fortran [2].
Lo podemos considerar como un lenguaje de programación interpretado. Éstos lenguajes interpretados o "scripting"no deben ser compilados. Un script es una serie de órdenes que se pasan a un intérprete para que las ejecute. Estas órdenes están escritas en un lenguaje de programación propio del intérprete que ejecuta lo pedido en una línea y no ejecuta la siguiente hasta que haya terminado la anterior, siempre que esta sea correcta. Difiere de un archivo de código fuente por él hecho que no tiene que ser compilado, es decir es ejecutable por sí mismo siempre que se llame con él intérprete adecuado.

De este modo podemos diferenciar los lenguajes basados en código fuente de los lenguajes de scripting. Los primeros son, por ejemplo, C++, Fortran, Cobol, Pascal...
En ellos se genera un código fuente y un programa llamado compilador que a su vez lo convierte en un archivo que él sistema operativo es capaz de ejecutar. Este ejecutable es a su

vez dependiente del sistema operativo, con lo que no distribuimos él ejecutable sino él código fuente sabiendo que la máquina a la que migramos él código dispone de tal compilador. Entre los lenguajes de scripting más conocidos tenemos por ejemplo los lenguajes de uso general, como Java.

Existe una gran variedad en los lenguajes de scripting orientado a fines matemáticos. Como son Matlab, Maple, Mathematica, Scilab, etc.
De éstos los más conocidos son Matlab y Mathematica, en parte debido a que son programas comérciales. No debemos considerar a Matlab como único lenguaje sujeto a un fin. El scripting científico es una gran herramienta que nos puede ser de mucha ayuda en la solución de problemas científicos y la visualización de resultados. Una vez que hayamos aprendido a usar él entorno de Matlab nos será más fácil aprender a utilizar Scilab por la similitud en su ejecución de órdenes.

La sintaxis de Matlab es muy simple y cuenta con una gran biblioteca de funciones matemáticas. Es un lenguaje muy cómodo si deseamos escribir pequeños programas no muy complejos o para guardar instrucciones y después ejecutarlas en secuencia con el fin de mostrar los resultados sobre algún problema complejo. Matlab es ante todo un lenguaje, más que un programa en sí. Y como mencionamos anteriormente casi todo lo que aprendamos en él, lo podemos realizar en Scilab...

Desde sus inicios, Matlab ha estado renovándose. Producto del interés en distintas áreas han surgido gran cantidad de funciones específicas para cada área que vienen agrupadas en paquetes denominados toolbox. Estas “toolboxes” o “cajas de herramientas” consisten en ser colecciones de funciones para Matlab desarrolladas para resolver problemas de un tipo particular. De entre esa área podemos citar, sistemas de control, estadística, procesamiento de imágenes, realidad virtual, economía, adquisión de datos, desarrollo de algoritmos entre otros [4].

Principales partes de Matlab:

Entorno de Desarrollo: Es la parte donde podemos usar los archivos y funciones de Matlab. La mayoría de estas herramientas son interfaces gráficas para él usuario. La de más alto nivel es el denominado Matlab desktop, que incluye él Command Window, Command History, Workspace View y Path Browser.

Librería de Funciones: Es una amplísima colección de algoritmos de cómputo de muy diversa complejidad, desde sumas, restas, declaración de variables y constantes, funciones trigonométricas, operación y solución de matrices, hasta manejo de Funciones de Transferencias etc.

Lenguaje de Matlab: Es un lenguaje de programación cuyo elemento básico es un array ó matriz. Que son regidas por una sintaxis, sentencias, todas orientadas a las principales características de programación.

Sistema de Gráficos: Nos brinda la posibilidad de visualizar los datos en una gran cantidad de representaciones distintas. Gracias a los comandos podemos visualizar gráficos bi-dimensionales y tri-dimensionales, procesamiento de imágenes, animación y presentación de gráficos etc.

Algunas de las aplicaciones de Matlab pueden ser:

Análisis matemáticos.
Simulaciones y Modelado.
Cálculo simbólico.
Desarrollo de algoritmos.
Modelado.
Exploración, visualización y análisis de datos.
Creación de gráficas.

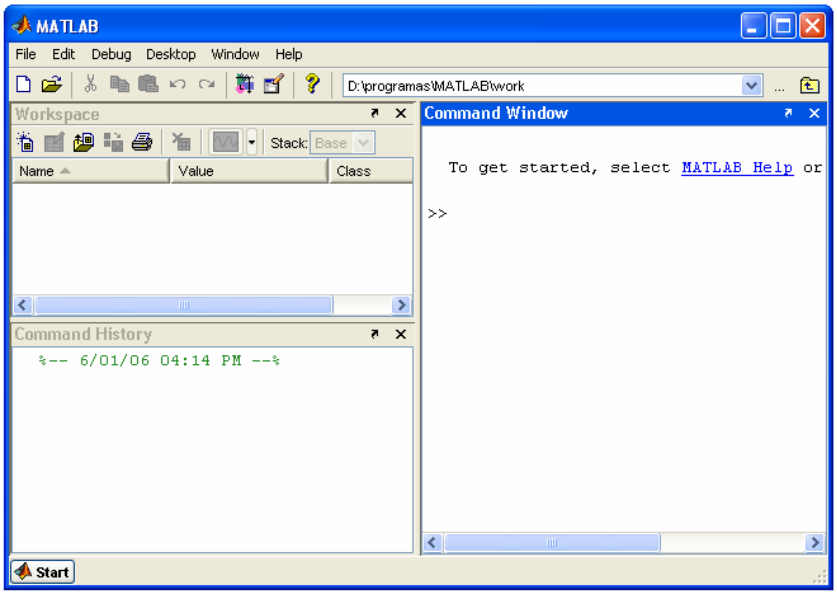
Estas características que hemos comentado hacen de Matlab una herramienta de trabajo muy extendida entre nosotros los estudiantes como entre técnicos e investigadores. [3]

Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Inicio de Matlab

Unas vez que hayamos instalado Matlab en nuestra PC, se creara un icono como el mostrado en la figura, procedemos a localizarlo y hacemos doble- clic en él.

A continuación debe abrirse una ventana como la siguiente figura. En ella se pueden identificar dos columnas de ventanas.



A la izquierda tenemos él "Workspace" el cual nos mostrara cuales son las variables que vamos a declarar y almacenar hasta ese momento.
El Command History nos lleva un historial de los comandos que hemos usado. Y "Current Directory" nos muestra cuál es él directorio de trabajo actual. A la derecha tenemos el "Command Window" o Ventana Principal, que emplearemos para transmitir las órdenes a Matlab. La ayuda en Matlab se muestra tecleando help en la Ventana Principal o pulsando el icono del signo de interrogación. En cualquier caso, si no incluimos ninguna palabra sobre la que solicitamos la ayuda, Matlab devolverá una lista con los temas existentes tal como esta...

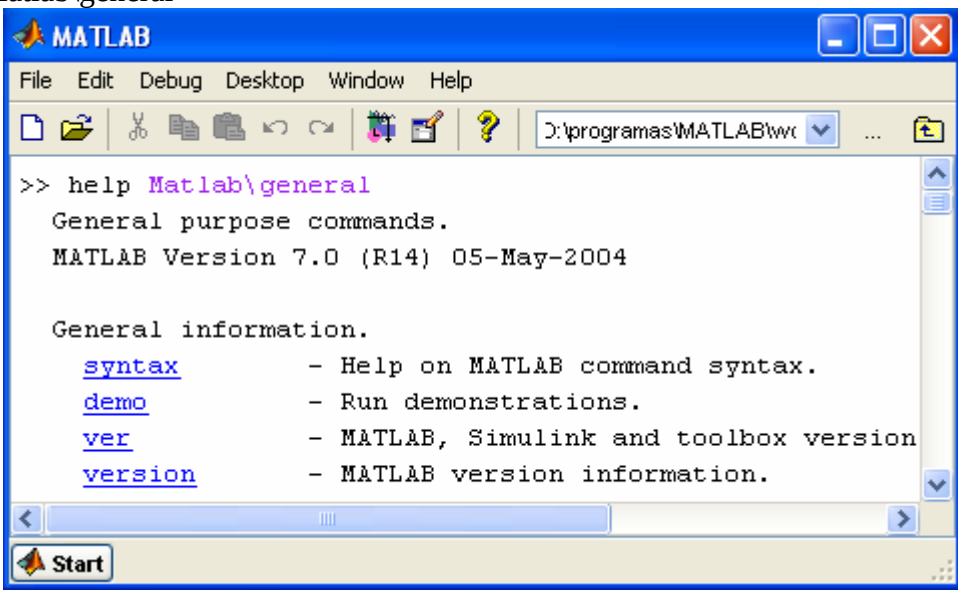
```
help
HELP topics:
Matlab\general -General purpose commands.
Matlab\ops -Operators and special characters.
Matlab\lang -Programming language constructs.
Matlab\elmat -Elementary matrices and matrix manipulation.
Matlab\elfun -Elementary math functions.
Matlab\specfun -Specialized math functions.
```

Si ahora queremos obtener ayuda acerca de algún tópico en particular, por ej. El primero de la lista anterior, tecleamos

```
help Matlab\general
```

Y obtenemos otra lista de ayudas para este tópico:

>> help Matlab\general



Si deseáramos conocer él comando “sin” podríamos escribir.

```
help sin
```

Ahora que sabemos buscar un cierto comando, por ejemplo “sin” en el conjunto de ficheros m-files. Si queremos buscar una cadena completa la especificamos delimitándola entre comillas por Ej.: help ‘matrix arithmetic’.
Una vez que hemos arrancado Matlab, si deseamos terminar podemos bien teclear exit en la ventana de comando, pulsar la secuencia
ctrl.+Q o ir al menú File > Exit Matlab.

Algunas reglas básicas.

En Matlab disponemos de aproximadamente 500 comandos, los cuales tienen ciertos formatos de escritura. Este distingue entre mayúsculas y minúsculas por lo que casi siempre los comandos e instrucciones se escriben con minúsculas.
Por ejemplo, no es lo mismo pi que Pi. Podemos teclear ambas expresiones y descubrir que la primera funciona perfectamente mientras que la segunda causa un error al ser ejecutada. La precisión utilizada de aproximadamente 16 dígitos [4]. La presiciòn con la que deseamos trabajar puede ser modificada por los siguientes comandos:
Para esto utilizaremos la fracción 4/3

Comando	Exhibición	Ejemplo
format short	Por omisión	1.3333
format short e	4 decimales	1.3333e+000
format long	14 decimales	1.33333333333333
format long e	15 decimales	1.333333333333333e+000
format bank	2 decimales	2.33
format hex	Exp. hexadecimal	3ff5555555555555
Format rat	Cociente de entero	4/3

Algunos comandos de nos serán mucha utilidad son:

pwd	Nos indica cual es el directorio de trabajo actual.
helpwin	Despliega una ventana con la descripción específica de un comando y permite ver información sobre otros temas relacionados
lookfor	Busca en la ayuda de todos los comandos la palabra especificada.
helpdesk	Realiza una búsqueda en hipertexto en un buscador Web proporcionando un acceso directo a toda la documentación: PDFs y si se encuentra disponible, información sobre la solución de problemas.
doc	Despliega en un buscador Web la página de referencia para él comando especificado, proporciona una descripción, referencias adicionales y ejemplos del comando especificado
figure	Crea una nueva ventana para una nueva gráfica.
close	Cierra la ventana donde se creo alguna gráfica.
which	Indica la ruta en donde se encuentra él comando especificado.
cd	Cambia la ruta al subdirectorio superior.

El uso de los paréntesis “( )” y los corchetes “[ ]” tienen significados y utilidades distintas. Los primeros los utilizamos para evaluar funciones, ej. (sin(pi/5)) y para cambiar el orden básico de las operaciones, por ejemplo (4*(26+5*(2+2))), mientras que los segundos sirven como delimitadores de vectores por ejemplo [1,2,3,4] o de matrices por ejemplo [5,6,7;8,9,10]. Matlab no permite utilizar las llaves “{ }” con la finalidad de cambiar el orden de las operaciones yen ningún otro caso.

Es habitual cometer un error al escribir una cierta expresión, sobre todo si ésta es larga y complicada, y no darnos cuenta hasta ejecutarla y comprobar él mensaje de error!!!, esto será algo normal si estamos empezando a conocer Matlab, mediante las teclas de los cursores podemos editar las instrucciones escritas anteriormente y así corregirlas.

La siguiente tabla muestra él uso de las teclas del cursor y otros

Teclas del cursor vertical	Permiten él movimiento por todos los comandos que hayamos introducidos anteriormente.
Teclas del cursor horizontal	Nos permite movernos por la línea de los comandos.
Inicio	Para colocar él cursor al comienzo de la línea.
Fin	Para colocarlo al final de la misma.
Blackspace	Borra la línea de edición completa.
Esc	Después de haberlo escrito un comando, esta tecla lo puede borrar.

Determinación de la fecha y hora

Matlab dispone de funciones que nos muestran información sobre la fecha y la hora actual. Las funciones más importantes relacionadas con la fecha y la hora son las siguientes.

Clock: Nos devuelve un vector fila de seis elementos que representan el año, el mes, el día, la hora, los minutos y los segundos, según él reloj interno de nuestra computadora. Los cinco primeros son valores enteros, pero la cifra correspondiente a los segundos contiene información hasta las milésimas de segundo.
Date: Devuelve la fecha actual como cadena de caracteres (por ejemplo: 24-Aug-2007)
Calendar: devuelve una matriz 6×7 con él calendario del mes actual, o del mes y año que se especifique como argumento.

Uso de Variables y Escalares.

Para asignar un valor a una variable, por ejemplo x, usamos el signo "=".
Por ejemplo: x=20

Notemos que este comando se ejecuta en la ventana Command Window, en forma "ruidosa". Para poder ejecutar este en forma silenciosa, utilizamos; ";"después de cada comando. Por ejemplo: x=20; en general, las variables en Matlab serán matrices, pero al asignarle a cualquier variable sólo un número, él programa se da cuenta de que estamos declarando una matriz de 1 por 1. Las variables pueden tener cualquier nombre escrito sean largos o cortos, con letras y números. La excepción son algunos nombres de comandos ya utilizados por Matlab, por ejemplo sin, cos, tf, etc.

Si deseamos trabajar con símbolos, hemos de indicarle al programa que se trata de símbolos. Para esto utilizamos el comando "sym" por ejemplo, si escribimos: 2/3
El programa responde: 0.6667
En cambio, si queremos que 2 y 3 se consideren como símbolos (de forma que sólo se opere de forma exacta con ellos), deberemos escribir:

```
sym(2/(3)
ans =
2/3
```

Si deseamos realizar la siguiente operación aritmética:

$$\frac{2}{3}+\frac{1}{4}$$

Debemos escribir:

```
sym(2/3)+sym(1/4)
ans =
11/12
```

Análogamente se pueden definir variables; por ejemplo, para definir la variable x podríamos escribir:

```
x= sym('x')
```

Una vez definida la variable x, podremos realizar operaciones simbólicas con ella:

```
x+3*x
ans =
4*x
```

Si tenemos que utilizar varios símbolos es recomendable definirlos todos simultáneamente, Esta instrucción va seguida de los símbolos a definir, separados por espacios; Ej.:
syms t s x y

Algunas variables asignadas por Matlab:

Eps: Es él igual a 2.2204*10^-16, que es él número real de valor absoluto más pequeño que se utiliza en Matlab. Todos los cálculos se encuentran realizados con una precisión dada por "eps", es decir cualquiera de dos números o mas que difieran en eps o menos serán iguales.
pi: 3.14159...
inf: infinito (en valor absoluto)
NaN: Not a number. Es el resultado que se proporciona si durante una operación se produce una indeterminación, por ejemplo, $^{0}\!/_{0}$, inf/inf.

Podemos también declarar un valor a la variable x, y asignarla como variable independiente de cierta función, por ejemplo y=2*x

Después de declarar varias variables es común que olvidemos algunas, para esto contamos con el comando "who"que nos mostrara las variables declaradas hasta ese momento o para una lista un poco más detallada sobre las características de cada variable utilizamos "whos" [3] además de la lista proporcionara información acerca del tipo de datos en cada variable y el tamaño que ocupa en memoria.

Si deseamos borrar la variable declarada "x ", podemos utilizar:

```
Clear x %% borramos la variable "x"
Clear %% De esta forma eliminamos todas las varibles declaradas anteriormente
```

Si por el contrario deseamos salvar todas la variables en un archivo llamado por ejemplo "variables", en él directorio c: /Matlab/work/, usamos:

```
save variables
```

Y tiempo después para volver llamarlas usamos:

```
load variables
```

Para guardar un conjunto de variables, por ejemplo x e y, en un archivo por ejemplo "algunas", en el directorio c: /Matlab/work/, usamos:

```
save algunas x y z
```

Para llamar un conjunto de variables de un archivo, por ejemplo x, del archivo "algunas":

```
load algunas x
```

Además podemos agregar variables a un archivo ya existente sin borrar las que ya hemos grabado. Por ejemplo para grabarle la variable "otra" al archivo algunas:

```
save algunas otra -append
```

En el caso de escalares su operación es muy parecida a las que conocemos:

Operación	Comando
Suma	+
Resta	-
Producto	*
Potencia	^
Transposición	'
División	/
Seno	sin(x)
Coseno	cos(x)
Tangente	tan(x)
Arco Seno	asin(x)
Arco Coseno	acos(x)
Arco Tangente	atan(x)
Exponencial	exp(x)
Logaritmo en base e	log(x)
Logaritmo en base 10	log10(x)

Matrices:

Para definir una matriz utilizamos "[ ]" para determinar él principio y fin de la matriz, respectivamente. Para separar valores dentro de la matriz usamos "," para separar valores de la fila utilizamos ";" Por ejemplo:

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

La misma matriz se puede producir usando "," para separar valores dentro de la fila y tecleando enter para separar las filas, por ejemplo:

```
a=[1,2,3
4,5,6
7,8,9]
```

Para poder referirnos a un valor específico dentro de la matriz utilizamos "(,)" por ejemplo el valor de la matriz en la fila 2 y la columna 1, seria:

```
a(2,1) %% Lo cual nos mostrara el valor de 4
ans =
    4
```

Un valor de la matriz puede ser cambiado en forma individual:

```
a(2,1)=10 %% Con esto el valor de 4 es cambiado a 10
a =
     1     2     3
    10     5     6
     7     8     9
```

Nos podemos referir a toda una columna o fila según sea el caso utilizando ":" en la posición de las filas o columnas, indicando que hablamos de todos los valores. Una columna o fila puede ser llamado de la siguiente manera:

```
a(2,:)
ans =
 4  5  6
```

A partir de matrices, se pueden generar nuevas matrices. Algunos ejemplos son:

```
b=[a;1,2,3] %% Generamos una matriz de 4x3
c=[a(1,:);[5,5,5];[6,6,6];a(3,:)] Generamos una matriz de 6x3
d=[a;a] Generamos  una matriz de 6x3
e=[a,a] Generamos  una matriz de 3x6
```

Otro comando de mucha utilidad es “det” él cual nos permite calcular el determinante de una matriz, por ejemplo para la siguiente matriz:

$$a=\begin{vmatrix}3 & 5 & 2\\ 4 & 2 & 3\\ -1 & 2 & 4\end{vmatrix}=3*\begin{vmatrix}2 & 3\\ 2 & 4\end{vmatrix}-5*\begin{vmatrix}4 & 3\\ -1 & 4\end{vmatrix}+2\begin{vmatrix}4 & 2\\ -1 & 2\end{vmatrix}=3*2-5*19+2*10=-69$$

En Matlab la orden para poder obtener el determinante de esta matriz será:

```
a=[3 5 2;4 2 3;-1 2 4]
a =
     3     5     2
     4     2     3
    -1     2     4
det(a)
ans =
   -69
```

Para obtener la diagonal de la matriz A, usamos él comando “diag”, Ej.:

```
a=[3 5 2;4 2 3;-1 2 4]
diag(a)
ans =
    3
    2
    4
```

Para obtener la inversa de la matriz A, usamos él comando “inv”, Ej.:

```
A=[2 4 6;4 5 6;3 1 -2]
inv(A)
ans =

  -2.6667   2.3333  -1.0000
   4.3333  -3.6667   2.0000
  -1.8333   1.6667  -1.0000
```

Nota; A*A’=I

Para obtener el tamaño de la matriz B, usamos el comando “size”, el cual regresa el número de renglones y columnas, Ej.:

```
A=[2 4 6;4 5 6;3 1 -2]
size(A)
ans =
    3    3
```

El comando “bar”, nos muestra los valores de las matrices en forma de barras.

Para obtener la solución de un sistema de ecuaciones lineales, sin necesidad de resolverlo manualmente, en Matlab disponemos del comando “rref” con el cual nos resolverá nuestro sistema: por ejemplo, si tenemos él siguiente sistema:

$$2x_1 + 4x_2 + 6x_3 = 18$$
$$4x_1 + 5x_2 + 6x_3 = 24$$
$$3x_1 + x_2 - 2x_3 = 4$$

La solución de este sistema es:

```
A=[2 4 6 18; 4 5 6 24; 3 1 -2 4];
 rref(A)
```

```
ans =
     1     0     0     4
     0     1     0    -2
     0     0     1     3
```

De donde podemos observar que 4, -2 y son soluciones del sistema, otra forma de resolver este sistema es por medio de si inversa:

```
A=[2 4 6 ; 4 5 6 ; 3 1 -2 ];
b=[18 ;24; 4];
 c=inv(A)*b
c =
    4.0000
   -2.0000
    3.0000
```

Operaciones con matrices:

Las operaciones de las matrices son parecidas a los escalares, desde luego debemos de respetar el rango de las matrices:

Operación	Operador	Ejemplo
Suma	+	b+c
Resta	-	b-c
Transponer	'	c'
Producto	*	b*c'
Potencia	^	c^2

De la misma forma que los escalares, las matrices pueden tener poseer cualquier nombre largo alfanumérico [5].

Otras Maneras de Crear Matrices:

Zeros(n, m) Generamos una matriz compuesta por n filas y m columnas.
ones(n, m) Generamos una matriz compuesta por de n filas y m columnas.
eye(n, m) Generamos una matriz identidad de n filas y m columnas.
rand(n, m) Generamos una matriz con valores aleatorios [0 a 1], de n filas y m columnas.
randn(n, m) Generamos una matriz aleatoria normal, de n filas y m columnas

Generando Argumentos

Podemos generar argumentos de números muy fácilmente, utilizando el operador ":"
Este operador nos genera una serie de números, que comienza en "principio", que avanza de elemento en elemento en, "incremento", hasta que se alcanza valor "final".
Si deseamos una serie que tenga 5 elementos, comience en 1 y termine 10,
La forma de declararla es la siguiente:

```
1:2:10
ans =
    1    3    5    7    9
```

Operaciones con polinomios

Para Matlab un polinomio se puede definir mediante un vector de coeficientes. Por ejemplo, el polinomio, $x^4 - 8x^2 + 6x - 10 = 0$ podemos representarlo mediante él vector [1, 0, -8, 6, -10]. Podemos realizar diversas operaciones sobre el, como por ejemplo evaluarlo para un determinado valor de x (función polyval ()) y calcular las raíces (función roots()):

```
 pol=[1 0 -8 6 -10]
roots(pol)
ans =
-3.2800
2.6748
0.3026 + 1.0238i
0.3026 - 1.0238i
polyval(pol,1)
ans =
-11
```

Para calcular producto de polinomios utilizamos una función llamada conv() (producto de convolución). En el siguiente ejemplo vamos a observar cómo podemos multiplicar un polinomio de segundo grado por otro de tercer grado:

```
 pol1=[1 -2 4]
pol1 =
1 -2 4
 pol2=[1 0 3 -4]
pol2 =
1 0 3 -4
 pol3=conv(pol1,pol2)
pol3 =
1 -2 7 -10 20 -16
```

Para dividir polinomios existe el comando llamado deconv(). Los comandos orientados al cálculo con polinomios son las siguientes:

roots(pol): Nos muestra las raíces del polinomio pol

polyval(pol,x) : Evaluamos el polinomio pol para él valor de x. Si x es un vector, pol se evalúa para cada elemento de x
polyvalm(pol,A): Evaluamos el polinomio pol para la matriz A
conv(p1,p2) producto de convolución de dos polinomios p1 y p2
[c,r]=deconv(p,q) división del polinomio p entre el polinomio q. En c se devuelve el cociente y en r él residuo de la división
residue(p1,p2) descompone él cociente entre p1 y p2 en suma de fracciones simples
polyder(pol) calcula la derivada de un polinomio
polyder(p1,p2) calcula la derivada de producto de los polinomios

Graficando en Matlab:

Las gráficas son producidas por él comando "plot", aunque se pueden generar gráficas de otras maneras, en este caso mostraremos el uso del comando "plot"[7]. Para vamos a crear una serie de incrementos que hemos definido anteriormente, éstos iniciaran con un valor -4 hasta +4, con incrementos de 0.1.
El comando plot nos da la opción de cambiar la representación de los puntos en la gráfica la forma más habitual es:

. Punto	* asterisco	-- segmento
o círculo	- sólido	
x marca	: punteado	
+ más	-. segmento punto	

Pero usaremos '.' (puntos) con lo que crearemos una gráfica de puntos.
Los tipos de color los obtenemos introduciendo la letra de cada color:

m magenta	r rojo	w blanco	b negro
c cyan (azul claro)	g verde	b azul	

El comando "loglog" crea un dibujo usando escalas logarítmicas para ambos ejes, el comando "semilogx" crea un dibujo usando una escala logarítmica para él eje x, y una escala lineal para el eje y "semilogy" crea un dibujo usando una escala logarítmica para el eje y, y una escala lineal para él eje x, además el comando "logspace" crea una escala para un determinado eje
Ej.:

```
x=logspace(-1,3); %Crea un eje x de 10^-1 a 10^3
loglog(x,exp(x)); %Grafica la función e^x
```

El comando "fplot" nos permite dibujar la gráfica de una función con la siguiente sintaxis[6]:
a)fplot(`función',[inicio,final])
b)fplot(`función',[inicio,final],#de puntos)
Ejemplo.

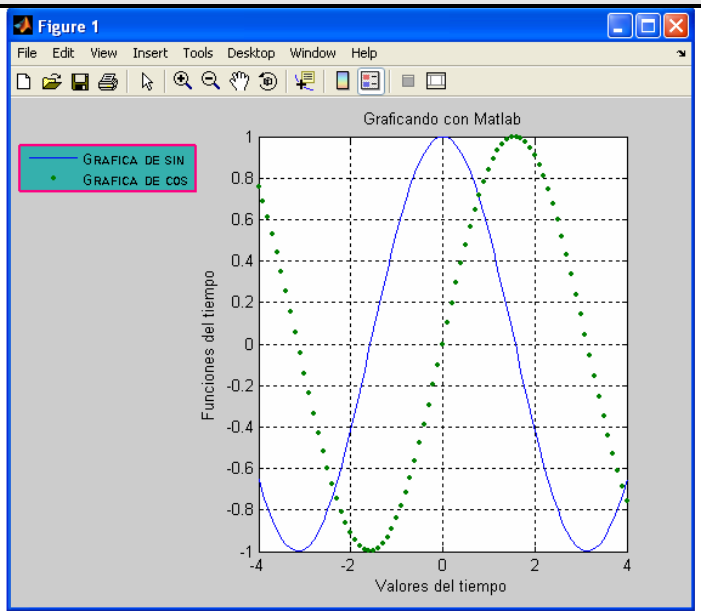
```
fplot('sin',[0,10],20 )
```

El comando "clf" nos borra la gráfica creada.

Si deseamos agregar títulos al gráfico y sus ejes, utilizamos la siguiente sintaxis:
xlabel('eje real') ; ylabel('eje imaginario') ;title('Plano complejo')

Para que la serie tenga una leyenda para distinguir las series gráficadas, ubicada abajo a la derecha del gráfico:
legend('Arg','Bra',4)
Para incluirle al gráfico una rejilla y ejes, usamos:
grid on ;axis
Un comando muy útil también es "[ejes]=ginput(#)" con él que podemos señalar varios puntos (#) en la gráfica y Matlab los desplegara en él Command Windows
Ejemplo :

```
x=-4:0.1:4; %Creamos él argumento
y=sin(x);
z=cos(x);
plot(x,z,'-',x,y,'.')
xlabel('Valores del tiempo');
ylabel('Funciones del tiempo');
title('Graficando con Matlab');
grid on ;axis;
% Ahora crearemos un mensaje en la pantalla;
legend(... %Creamos un mensaje que describa un poco la curva...
   {'Grafica de sin','Grafica de cos'},...
   'Color',[0.2078 0.6902 0.6784],...Definimos él color de fondo combinando rojo,verde,azul
   'LineWidth',2,...%Definimos él ancho del borde
   'XColor',[1 0 0.502],...%Él color del borde con intensidades de 0 a 1
   'YColor',[1 0 0.502],...
   'FontName','Technic',...%Definimos él estilo de las letras
   'Location','NorthWestOutside',...%Definimos la localizacion.
   'EdgeColor',[1 0 0.502]);
[x,y]=ginput(3); [x,y]
```



Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Herramientas para Sistemas de Control en Matlab.

El "Control System Toolbox" es un conjunto de herramientas para él análisis y diseño de Sistemas de Control.
Podemos crear modelos de sistemas SISO (con una señal de entrada y una de salida).
Calcular y representar gráficamente respuestas en el tiempo, respuestas en frecuencia, etc.
Contiene comandos para consultar las características de los modelos como la localizacion de polos, ceros y ganancia [10].

Respuesta Transitoria de la Función de Transferencia

Si deseamos realizar un análisis de un sistema, con una entrada lo primero que debemos de definir es la función transferencia del sistema, esta es la relación de las entradas al sistema y las salidas de este con condiciones nulas, en la transformada de Laplace.

$$H(s)=\frac{Y(s)}{R(s)}=\frac{b_o S^m + b_1 S^{(m-1)} + \ldots b_m}{S^n + a_1 S^{(n-1)} + \ldots a_{(n-1)} S + a_n}$$

Para la cual debemos de definir primero el numerador Y(s) y el denominador U(s) los cuales serán los coeficientes de los polinomios del numerador y del denominador en potencias decrecientes de 's'. Por ejemplo, si deseamos definir la función de transferencia:

$$sistema1=\frac{Y(s)}{U(s)}=\frac{1}{s^2+10s+20}$$

En la ventana de comandos de Matlab (Command Windows), tecleamos las siguientes instrucciones para declarar nuestra primera función transferencia:

```
y=[1];
u=[1      10      20];
sistema1=tf(y,u)
Transfer function:
     1
---------------
s^2 + 10 s + 20
```

Otro ejemplo seria calcular los ceros y los polos de la siguiente función de transferencia:

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

Para esto calculamos las raíces del polinomio del denominador usando la instrucción: roots(num). De manera análoga las raíces o polos del polinomio del denominador se evalúan usando la instrucción: roots (den) [2]

```
num=[1  5];
den=[1 2 3 1];
sistema2=tf(num,den);
roots(num)
roots(den)
ans =
    -5
ans =
  -0.7849 + 1.3071i
  -0.7849 - 1.3071i
  -0.4302
```

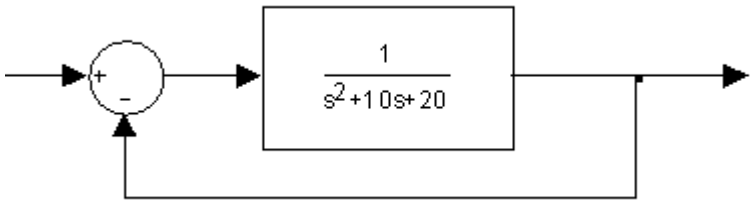
Con lo cual podemos observar que obtenemos como resultado -5,-0.7849 $\pm$ 1.3071i,, -0.4302

Otra manera de visualizar la función de transferencia en forma de polos, zeros y ganancia la podemos realizar con él comando “zpk”, por ejemplo, tomando la función del primer ejemplo:

```
sistema3=zpk(sistema1)
 Zero/pole/gain:
         1
-------------------
(s+7.236) (s+2.764)
```

Feedback

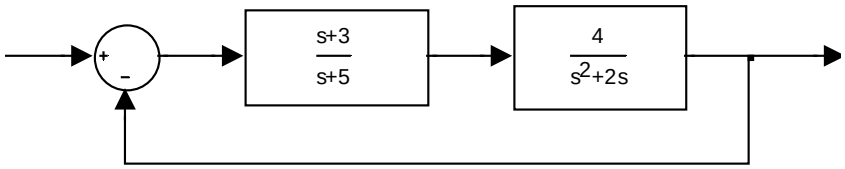
Si deseamos obtener la función de transferencia de un sistema en lazo abierto, es decir, dentro de un lazo con retroalimentación unitaria, debemos emplear el comando “feedback” definiéndola de la siguiente manera.




```
feedback(sistema1,1,-1)
Transfer function:
         1
---------------
s^2 + 10 s + 21
```

Series.

Este comando nos brinda la posibilidad de poder multiplicar dos funciones de trasferencias o ser manipuladas como bloque que se encuentran en serie y que pueden ser unidas originando una sola.




```
 sistema4=tf([1  3],[1  5]);
sistema5=tf([5],[1  2  0]);
series(sistema4,sistema5)
Transfer function:
     4 s + 12
------------------
s^3 + 7 s^2 + 10 s
```

También lo podríamos haber realizado usando dos veces él comando “conv” que permite la multiplicación de polinomios.

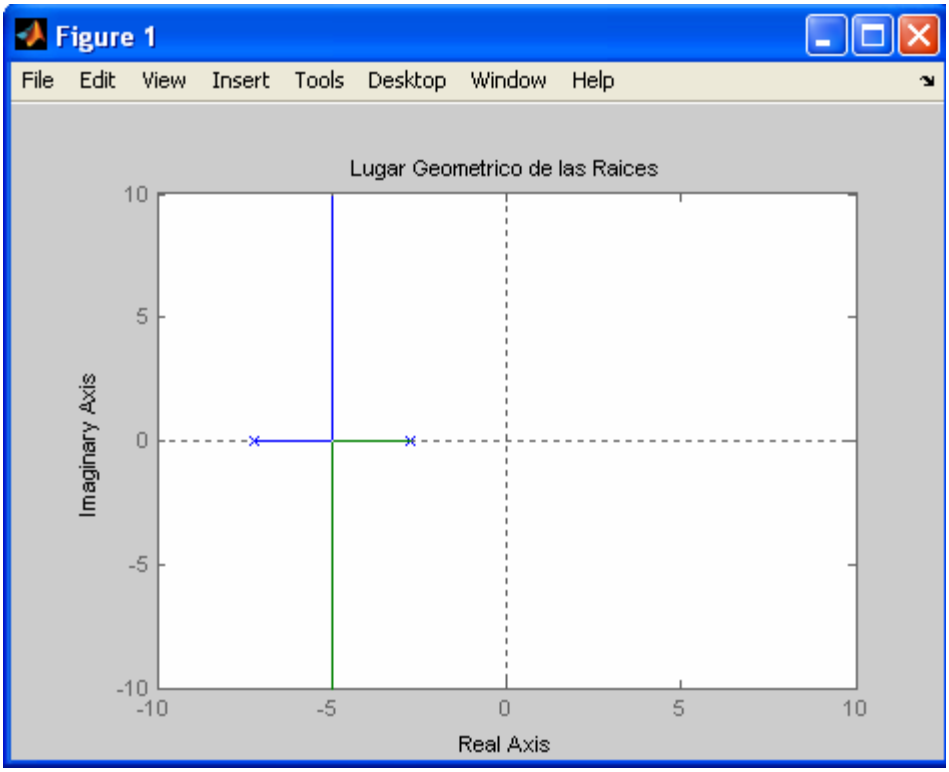
Lugar Geométrico de las Raíces

Para obtener el lugar de las raíces, Matlab dispone del comando “rlocus”. Las diferentes sintaxis para utilizar este comando son[10]:

rlocus(num,den): calcula y dibuja el lugar de las raíces donde ‘num’ y ‘den’ son los vectores de los coeficientes en potencia decrecientes de ‘s’ de los polinomios del numerador y denominador de la función de transferencia.

rlocus(num,den,k): calcula y dibuja el lugar de las raíces cuando se trabaja con la función de transferencia y ha sido previamente definido el rango de valores de K, ganancia del sistema. Por ejemplo de 0 a 100 con incrementos de 10: k=0:10:100

```
num=[1];
den=[1 10 20];
sistema1=tf(num,den)
rlocus(sistema1);
v=[-10 10 -10 10];
axis(v);
title('Lugar Geometrico de las Raices')
```



En Matlab disponemos del comando rlocfind el cual permite determinar los polos del sistema para una valor determinado de k. Su sintaxis es: [k,poles] = rlocfind(num,den) que permite determinar los polos para un valor determinado de k.
En el plano complejo seleccionamos una localización y Matlab retornará el valor de k para esta localización y los polos asociados a esta ganancia [11].

Al ejecutar él comando 'rlocfind' con la función de transferencia anterior, se activara la ventana de figuras en espera de seleccionar un punto del plano complejo mediante el cursor. En este caso el punto seleccionado fue -5.00 en la parte real y +7.9193 en la parte imaginaria.

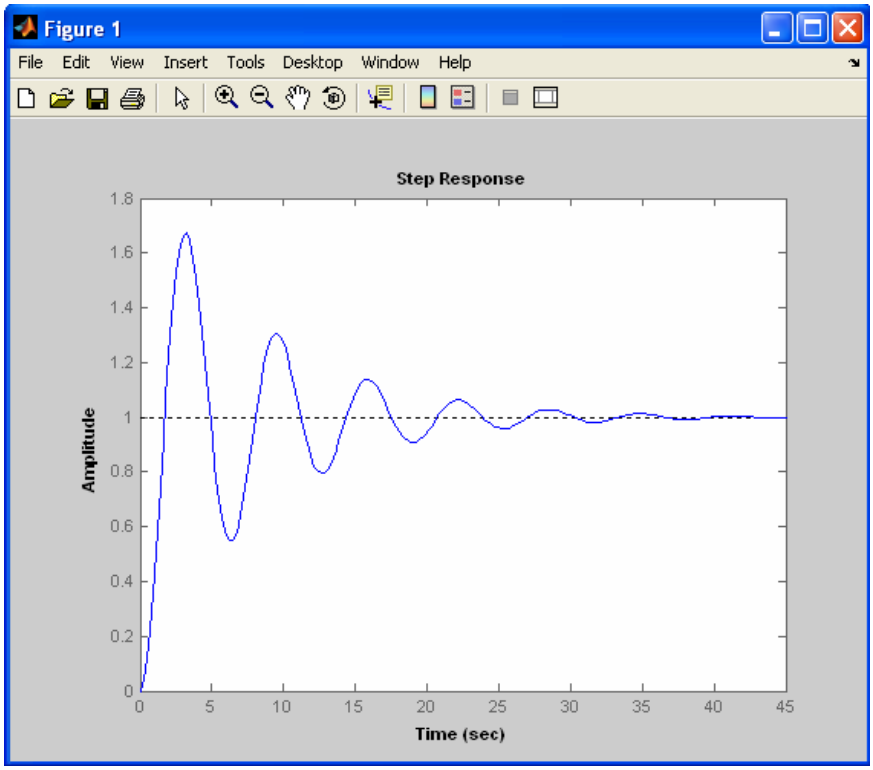
```
[Ganancia,Polos]=rlocfind(sistema1)
Select a point in the graphics window
selected_point =
  -5.0000 + 7.9193i
Ganancia =
   67.7146
Polos =
  -5.0000 + 7.9193i
  -5.0000 - 7.9193i
```

Entrada Escalón.

Para determinar la respuesta en el tiempo ante una entrada escalón unitario de algún sistema usamos él comando 'step' indicando el vector del numerador y del denominador con sus respectivos paréntesis. step(num,den) o en haciendo uso de la función transferencia, previamente declarada step(nombre de la función transferencia)[12].

```
num=[1];
den=[1 0.25 1];
sistema1=tf(num,den);
step(sistema1)
```

Matlab despliega la respuesta en función el tiempo en la siguiente ventana:

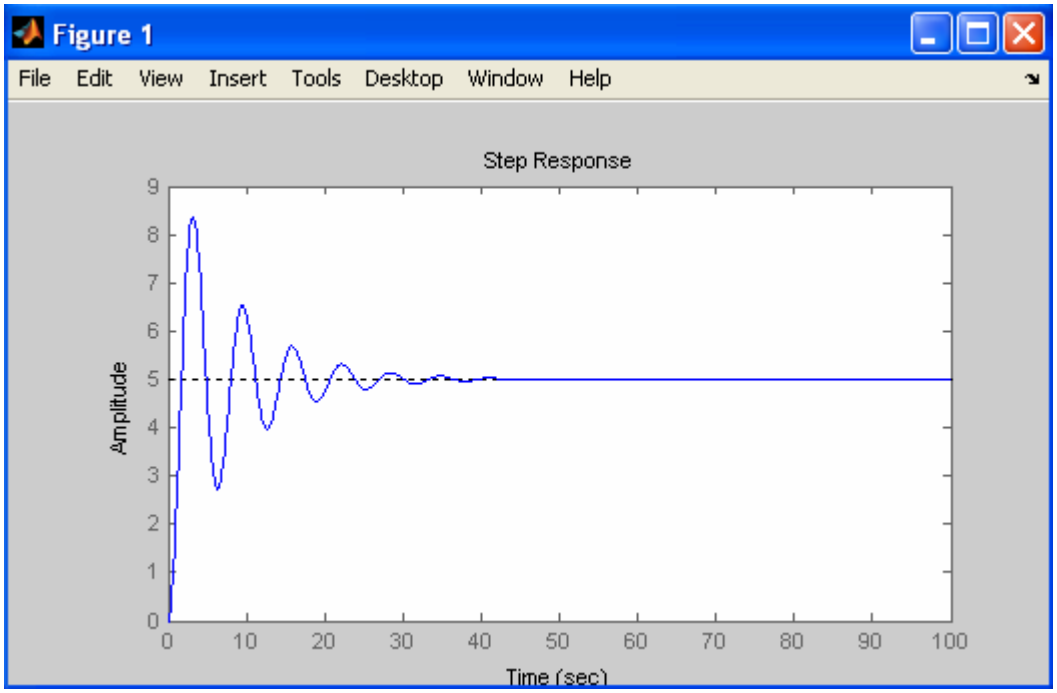


Si la entrada escalón es de un valor de 5 y se desea ver la salida hasta un tiempo de 100 segundos, la instrucción será la siguiente:

```
step(5*sistema1,0:0.1:100)
```

Si deseamos tener una apreciación más amplia sobre cómo se comporta nuestro sistema en una visión más panorámica, lo podemos hacer introduciendo la letra “t” en la sintaxis del comando la cual nos mostrara la gráfica extendida según sean los segundos asignados por t[14].

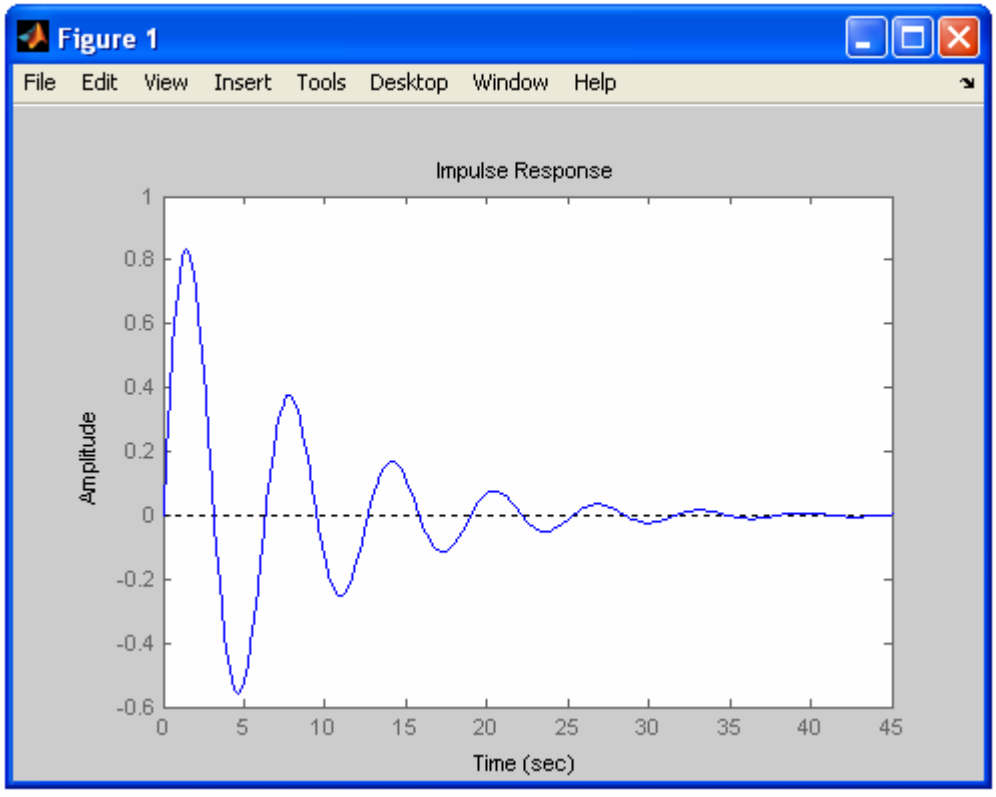
Definimos a “t” como un vector con argumentos, con elemento inicial 0, su elemento final es 100 existiendo elementos que van incrementándose de 0.1 en 0.1.
Nótese que se usamos “;” en la línea de “t”, para evitar el eco de la pantalla.
Al ejecutar el comando ‘step’ para el sistema1 se obtiene en la ventana de figuras la respuesta escalón para los primeros 100 segundos.



Entrada Impulso.

Si deseamos obtener la respuesta en el tiempo para una entrada impulso unitario se usa el comando 'impulse', con la sintaxis idéntica a la que anteriormente habíamos utilizado con el comando 'step'[11], Definimos en Matlab los polinomios del numerador y denominador de la función de transferencia, por ejemplo:

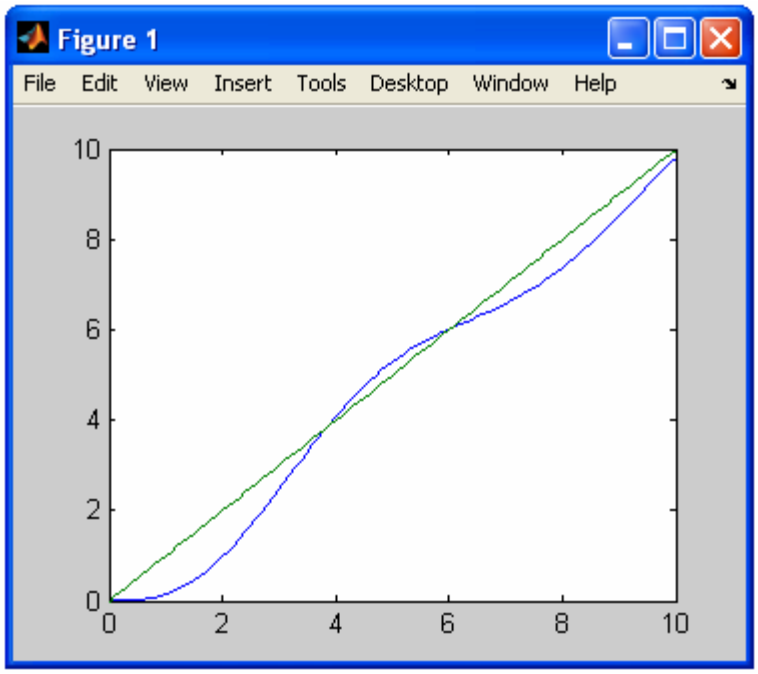
```
num=[1];
den=[1 0.25 1];
 sistema1=tf(y,u);
impulse(sistema1)
```



Como hemos visto en Matlab podemos obtener la respuesta en el tiempo para una entrada escalón o impulso, también podemos obtener respuesta para otras entradas como rampas o sinusoides. El comando "lsim" permite obtener la respuesta en el tiempo para un sistema con una entrada 'u', donde 'u' se define como una función del tiempo. La sintaxis de este comando es: lsim(num,den,u,t) donde 'num' y 'den' son correspondientes a la función de transferencia. Para obtener la respuesta en el tiempo para una función rampa, definimos 'u' de la siguiente manera:

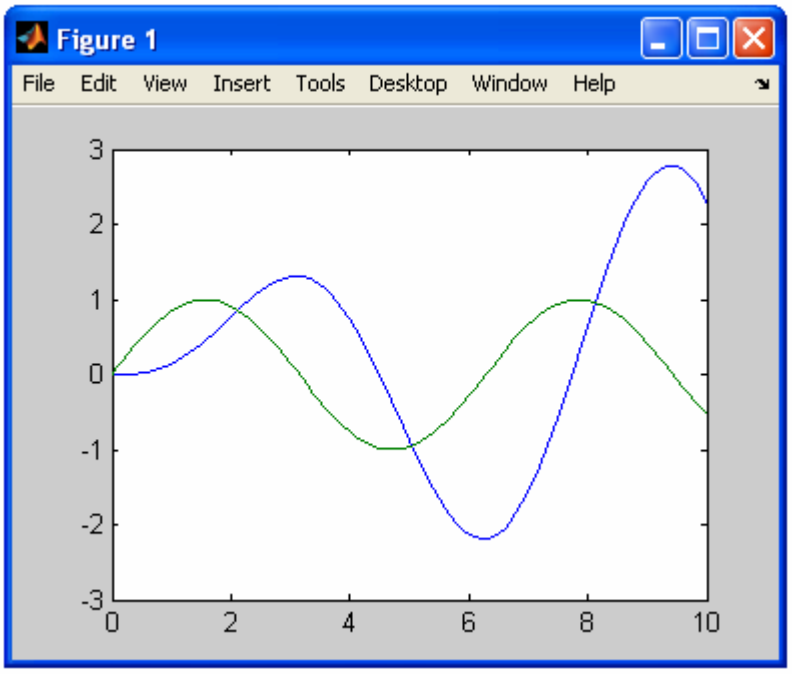
```
t=0:0.1:10;
u=t;
num=[1];
den=[1 0.25 1];
y=lsim(num,den,u,t);
plot(t,y,t,u);
```

Al hacer u=t se está definiendo la función rampa. t es el vector de tiempo variando desde 0 hasta 10 seg. num y den son los vectores de los coeficientes decrecientes en potencia de 's' de los polinomios del numerador y del denominador respectivamente. En la variable "y" se almacena la salida del sistema en función del tiempo "t". El comando plot permite desplegar en la ventana de figuras la variable y (salida) y la entrada u (rampa) en función del tiempo, en donde la gráfica de color verde representa la entrada y la azul la salida, obteniéndose:



Otro ejemplo seria:

```
t=0:0.1:10;
u=sin(t);
num=[1];
den=[1 0.25 1];
y=lsim(num,den,u,t);
plot(t,y,t,u);
```



Respuesta en Función de la frecuencia

Si deseamos realizar un análisis de un sistema en función de la frecuencia existen herramientas disponibles en Matlab como son, los diagramas de Bode.

Diagrama de Bode.

Para obtener el diagrama de Bode, Matlab dispone del comando Bode. La sintaxis para utilizar este comando son: [magnitud,fase,w]=bode(num,den,w) en la que Bode no muestra ninguna gráfica en pantalla, sino que devuelve la respuesta en frecuencia del sistema en tres matrices magnitud, fase y w. Estas matrices contienen las magnitudes y los ángulos de fase de la respuesta en frecuencia del sistema, evaluados en los puntos de frecuencia que hayamos especificado. El ángulo de fase se devuelve en grados.

También podemos usar la sintaxis bode(num,den) con el argumento adicional "w" para especificar manualmente los rangos de frecuencia, de la forma: Bode(num,den,w)

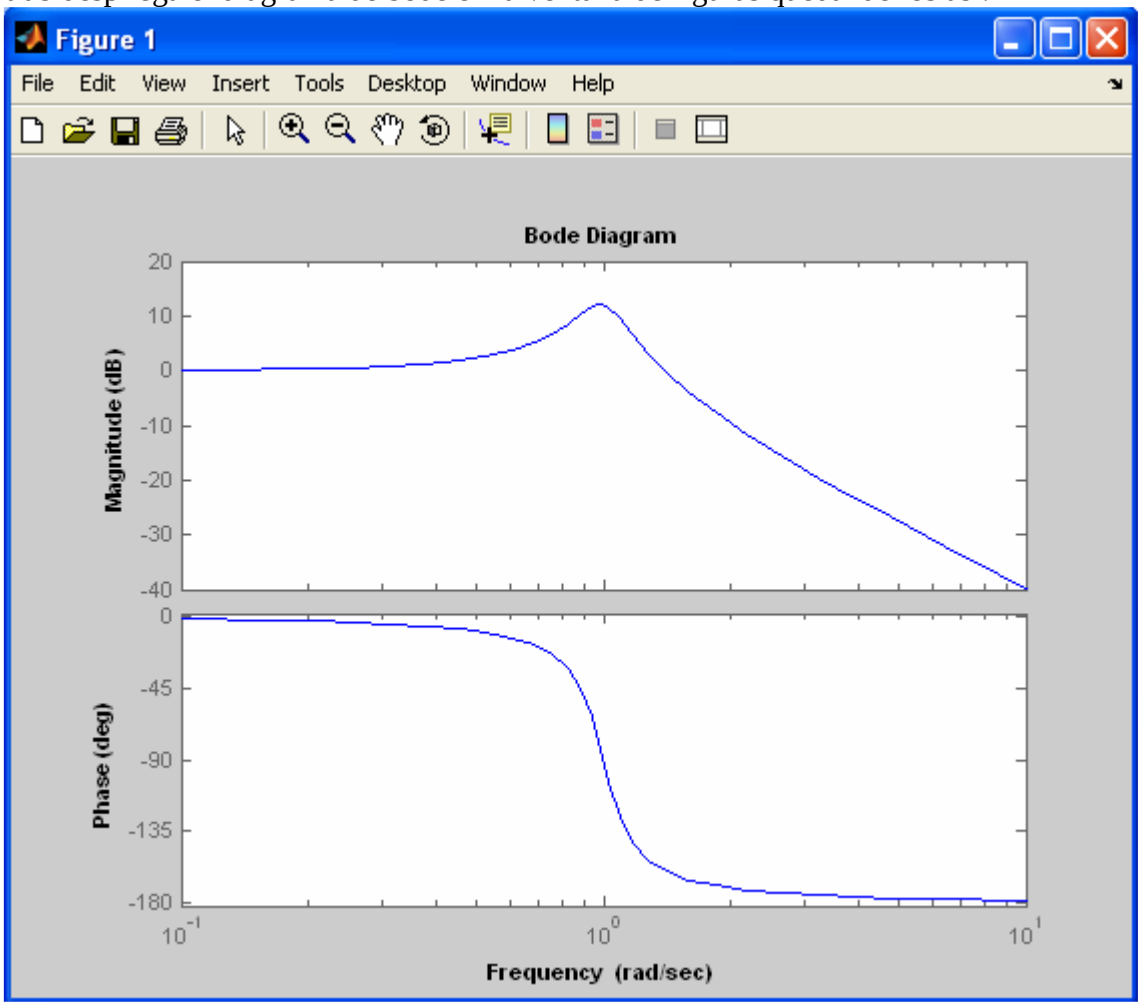
Definimos la función de transferencia:

$$H(s)=\frac{Y(s)}{U(s)}=\frac{1}{S^2+0.25S+1.0}$$

Introducimos en el símbolo del prompt., la siguiente expresión

```
num=[1];
den=[1      0.25      1];
sistema3=tf(num,den);
bode(sistema3)
```

Matlab despliega él diagrama de bode en la ventana de figuras quedándonos así:



Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Simulink.

Simulink es una extensión de Matlab que nos permite realizar simulaciones y análisis de sistemas dinámicos. Nos permite construir diagramas de bloques de manera gráfica, evaluar el desempeño del sistema y hacer modificaciones en su diseño. En un nuestro caso estudiaremos el paquete de Simulink para el análisis de Sistemas de Control, por lo tanto entendamos que Simulink no es completamente un programa separado de Matlab, sino una herramienta de el. La ventana de Matlab estará siempre disponible mientras ejecutemos una simulación en Simulink.

Tiene un entorno parecido a las demás herramientas de Matlab, en la forma que se ejecuta de una manera independiente de la ventana principal de comandos, pero sigue siendo la mejor herramienta para aprovechar toda la potencia de Matlab [2].
Existen dos fases importantes en la simulación, la construcción del sistema y su análisis de este.

La construcción de nuestro sistema, la podemos realizar a partir de elementos básicos que previamente se encuentran en el menú de Simulink, tales como integradores, bloques de ganancia, señales de entrada, sumadores, etc.
El análisis del modelo significa realizar la simulación, y determinar el punto de equilibrio del sistema que previamente hemos definido.
Simulink posee una interfaz gráfica de usuario (GUI), con diagramas de bloques para construir los modelos esto lo realizamos utilizando operaciones con el ratón del tipo pulsar y arrastrar. De esta manera, vamos a poder dibujar los modelos de la misma forma que lo haríamos con lápiz y papel. Una vez construidos los diagramas de bloques, podemos ejecutar las simulaciones para después analizar los resultados, también de forma gráfica.

Simulink posee una amplia gama de bloques, fuentes, componentes lineales y no lineales y conectores. Nos da la opción de poder personalizar y crear nuestros propios bloques. Podemos crear bloques de una manera jerárquica en donde un solo bloque (superbloks o subsistemas) pueda contener dentro de si, más de un block de esta forma podemos ver un sistema desde un nivel superior y entrando en los bloques podemos ir descendiendo a través de los niveles para ver con más detalle.

Los principales bloques de Simulink son:

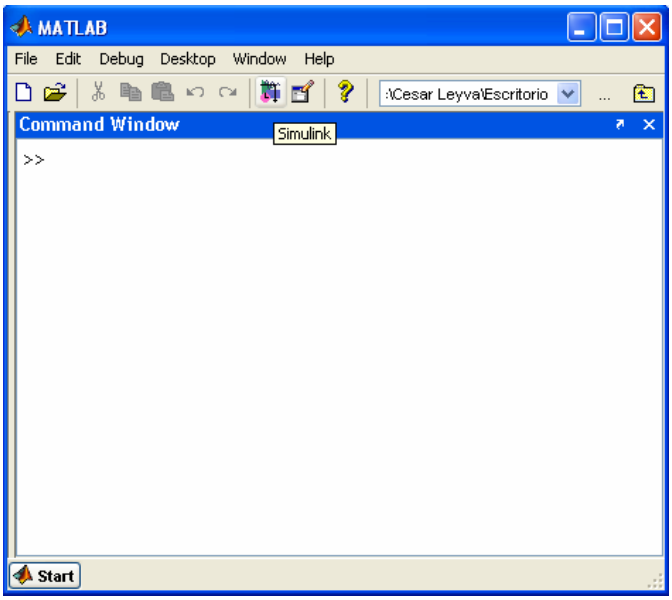
Sources: Entradas o fuentes de señales Step (escalón) Ramp (ramp) Sine Wave (senoidal) Pulse generador (tren de pulsos) From workspace (lectura de datos desde Matlab)
Sinks: Salidas o dispositivos de visualización/almacenamiento de variables del sistema Scope (osciloscopio)

Display (indicador numérico) To workspace (envío de datos a Matlab)
Continuous: Representan sistemas continuos por su relación entrada-salida Derivative(bloque derivador: la salida es la derivada de la entrada) Integrator (bloque integrador: la salida es la integral de la entrada) Transfer Fcn (función de transferencia en s expresada como cociente de polinomios) Zero Pole (función de transferencia en s expresada en forma factorizada)
Math operators: Operaciones matemáticas sobre señales Sum (sumador de señales) Gain (ganancia o multiplicación de una señal por una constante)
Signal Routing: Permite realizar conexiones especiales entre señales Mux: (multiplexor: agrupa distintas señales en un vector o bus)

Además, existen los Demos (demostraciones en el menú Help) y funciones complementarias. Después de definir un modelo, lo podemos Simular, desde el menú de órdenes de Simulink introduciendo órdenes directamente desde la línea de comando de Matlab. Con los bloques de visualización podemos ver los resultados de la simulación mientras se está ejecutando. Además, los resultados de la simulación se pueden transferir al espacio de trabajo de Matlab para su posterior tratamiento. También podemos usar las utilidades de los Toolbox's de aplicación de Matlab [3].
Para iniciar Simulink simplemente tecleamos en la ventana de comando de Matlab:

Simulink

En la línea de comandos de Matlab, o también podemos acceder a través del icono de acceso directo situado en la barra de herramientas:

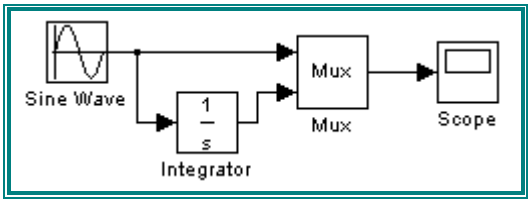


Una vez en Simulink, nos encontramos con la ventana de la librería principal de bloques. Para que podamos realizar un diagrama de simulación debemos primero de conocer algunos pasos básicos como, por ejemplo:

Inserción de bloques

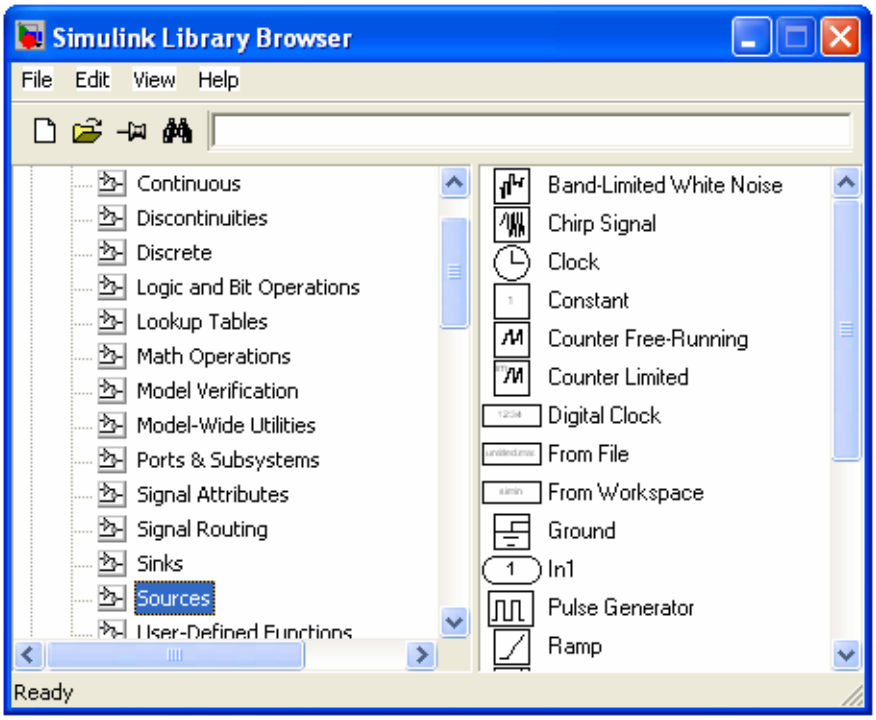
Un modelo en Simulnik es una colección de bloques interconectados que representan un sistema. Los bloques sirven para generar (ej. sine wave), modificar (ej. integrator), combinar (ej.mux) y visualizar señales (ej.scope). Las líneas sirven para transmitir las señales entre los bloques en la dirección de la flecha.

El modelo que vamos a construir es él siguiente:

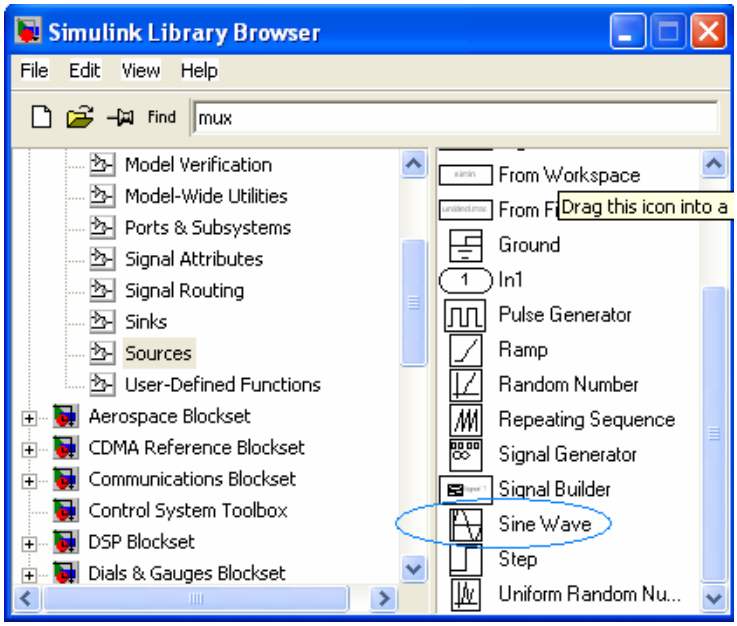


Se trata un sistema en donde la señal de entrada es una onda senoidal, para después ser integrada. El bloque multiplexor forma una señal vectorizada (uniendo las dos señales) que visualizaremos en él bloque Scope (Osciloscopio).
Para construir el modelo, comenzaremos por llamar los bloques que lo integran. Los bloques los podemos encontrar en las siguientes librerías:

Bloque	Librería
Sine Wave (Onda Senoidal)	Sources
Integrador	Continuous
Mux (Multiplexor)	Signal Routing
Scope (Osciloscopio)	Sinks



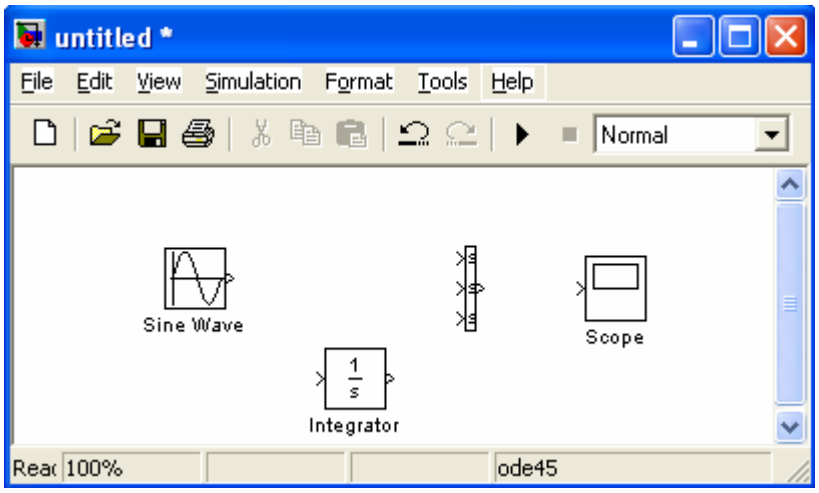
Dentro de ‘Sources’ elegimos una onda senoidal:



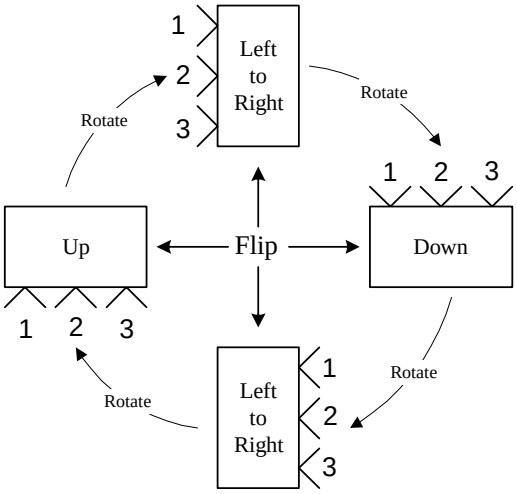
En la librería principal elegimos el bloque ‘Sources’, donde encontramos las señales que podemos elegir como fuentes para nuestro modelo, y realizamos lo mismo para los demás blocas[26].

Para situar el bloque en él documento nuevo simplemente pinchamos con el ratón y lo arrastramos. Al soltarlo quedara situado, y la librería de fuentes abierta, podemos minimizarla o cerrarla. Si deseamos ver al mismo tiempo las librerías de bloques y él modelo que estamos

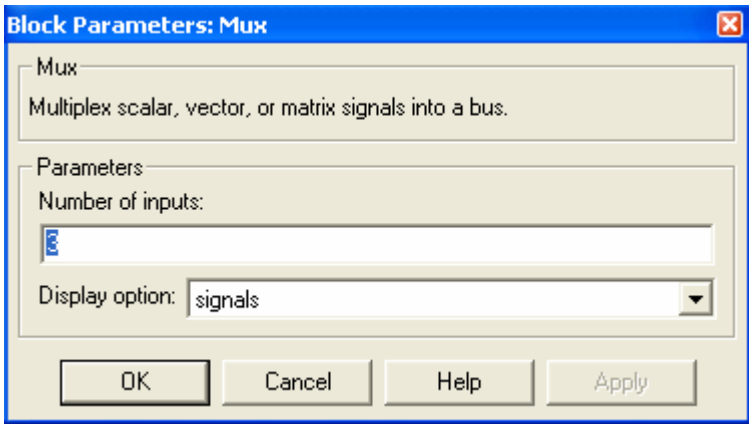
construyendo podemos ajustar el tamaño de las ventanas de forma que queden bien distribuidas en la pantalla y nos facilite el trabajo. Después que hayamos situado todos los bloques del sistema, él aspecto de la pantalla será él siguiente:



En general, las entradas para un bloque se encuentran en el lado izquierdo, con un símbolo > apuntando al bloque. De la misma forma, las salidas se encuentran normalmente en el lado derecho, con él símbolo > apuntando hacia afuera del bloque. En ocasiones por comodidad se requiere que la dirección de las entradas sea invertido para esto lo que debemos realizar es, en el menú 'Format → Rotate Block'. Con lo que la estaremos girando 90 grados También podemos girar 180 grados de una vez con la opción 'Format → Flip Block').



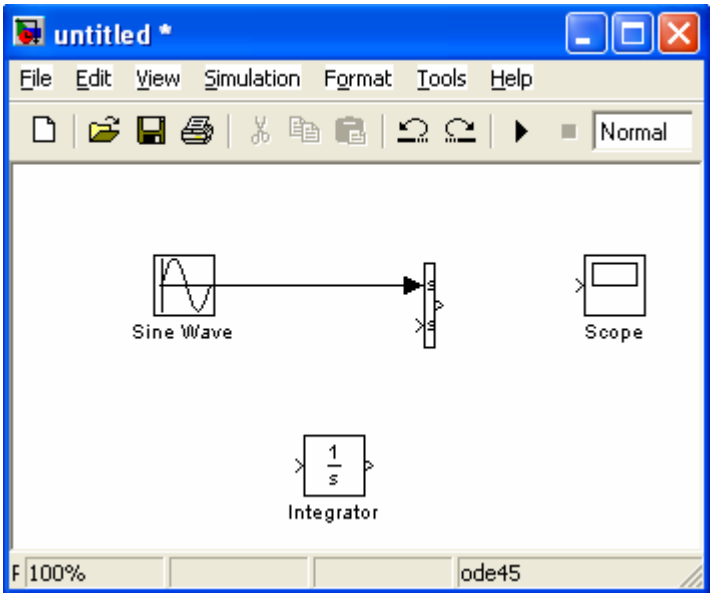
Podemos observar que el bloque multiplexor esta compuesto por tres entradas, de las cuales sólo utilizaremos dos. Para modificar el número de entradas situamos el puntero del ratón sobre el bloque y hacemos doble clic[25]. Nos mostrara el cuadro de diálogo siguiente:



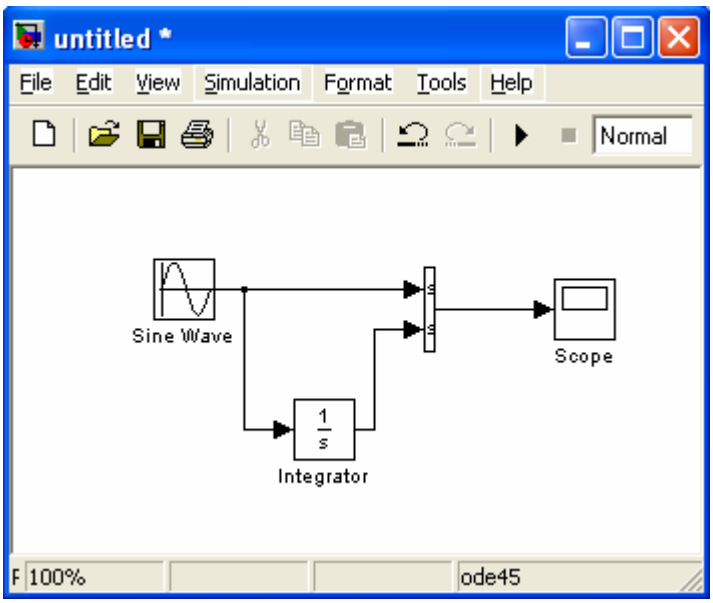
Solo tenemos que cambiar el número de entradas de 3 a 2, y aplicar los cambios. Este es el método que usaremos para cambiar los parámetros de cualquier bloque.
Para los demás bloques del modelo, no es necesario definir ningún parámetro.

Conexiones entre Bloques

Ahora que ya tenemos los bloques situados, ha llegado el momento de conectarlos. Vamos a empezar conectando la salida del bloque de Onda Senoidal a la primera entrada del multiplexor. Si situamos el puntero del ratón sobre la salida de la onda senoidal podremos observar que el cursor del ratón pasa de ser la típica flecha a convertirse en una cruz. Esta es la forma en que Simulink nos manifiesta que vamos a realizar una conexión. Cuando nos aparezca este símbolo presionamos el botón izquierdo del ratón y lo arrastramos hasta el puerto de entrada del multiplexor. Veremos como va apareciendo la línea de conexión, en el momento que estemos sobre el símbolo de entrada volverá cambiar de aspecto, y apareciendo ahora como una cruz doble, esto nos indicara que podemos terminar la conexión. Si soltamos el botón del ratón observamos que el extremo de la conexión que acabamos de realizar cambia para convertirse en una flecha.



Si por error o prueba soltamos la conexión antes del llegar al bloque de destino, la línea se quedará flotando sin llegar a ningún sitio, podemos volver a presionarla sobre ella y alargarla hasta donde queramos.
Haremos lo mismo para conectar el bloque multiplexor al bloque 'Scope'
También podemos realizar líneas que partan de otras líneas y lleven su misma información, pero debemos de tener en cuenta que no podemos insertar la información de una línea en otra línea.
Para hacer que una línea parta de otra línea picamos con el ratón en la línea mientras mantenemos presionada la tecla 'ctrl.'. Si nos lo hacemos así, sucederá que la línea simplemente se moverá.
Para completar las conexiones, unimos la salida del integrador a la entrada libre del multiplexor. Y nuestro diagrama de simulación nos quedara así:

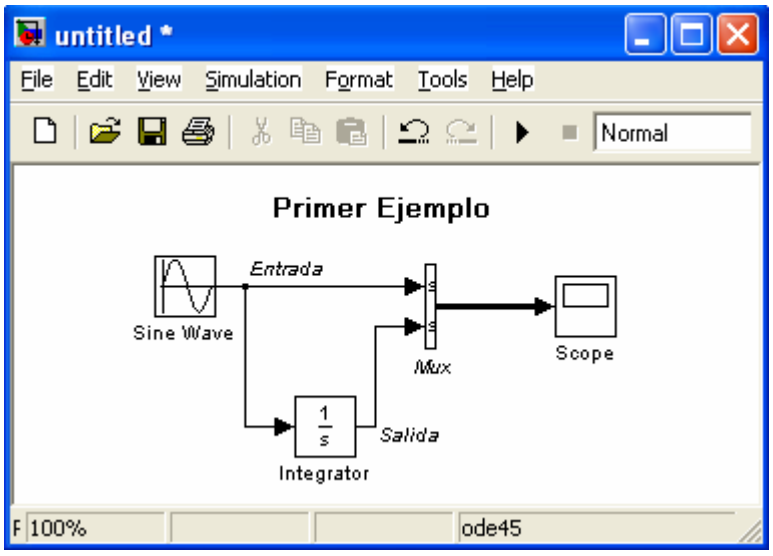


Añadir Etiquetas al sistema

En Simulink podemos escribir anotaciones en cualquier parte del diagrama de bloques. Para escribir una anotación sólo hay que dar doble clic con el ratón en el lugar donde deseemos que aparezca. A continuación se nos mostrara un pequeño rectángulo y él cursor se transformará en una pequeña barra vertical. Podemos escribir varias líneas presionando la tecla 'enter'. Cada línea que escribamos es automáticamente centrada en el rectángulo de escritura. Para terminar la anotación, picamos con el ratón cualquier otra parte de la ventana
Las etiquetas, podemos moverlas a cualquier parte del dibujo arrastrándolas con el ratón. Para cambiar el tipo y tamaño de la letra, seleccionamos la anotación y en el menú "Format – Font" haciendo los cambios necesarios.

Etiquetas de línea

Podemos incluir etiquetas a las líneas del diagrama para hacerlo más sencillo de interpretar. Para esto hacemos doble click sobre la línea que deseamos e insertamos el comentario.



Otras operaciones con bloques y líneas

Existen otras operaciones a realizar con los bloques y líneas de Simulink, que no hemos usado en este primer ejemplo debido a su sencillez. Algunas de estas operaciones son:

1.- Manipulación de bloques:

Selección de objetos: Cuando seleccionamos un objeto, aparecen unos pequeños círculos negros en las esquinas (si se trata de un bloque), o en sus extremos (si hablamos de una línea) esto indica que el objeto ha sido seleccionado. Para seleccionar otro objeto basta con pulsar

sobre el con el ratón, y cualquier objeto que estuviera seleccionado anteriormente dejará de estarlo [2].
Si queremos seleccionar más de un objeto podemos hacerlo de dos formas. Una es haciendo una ventana con el ratón que comprenda todos los objetos que queremos seleccionar. La otra es ir seleccionándolos de uno en uno mientras mantenemos pulsada la tecla 'shift'.
Si queremos seleccionar todos los elementos de la ventana activa podemos hacerlo desde el menú 'Edit' con la opción: 'Edit – Select All'.

Copiar y mover bloques de una ventana a otra: Para copiar y mover bloques de una ventana a otra se hace de la misma forma que si los tomamos de las librerías. Para mover objetos, seleccionamos y arrastramos con el ratón. Para copiarlos, córtalos pegarlos o borrarlos, los seleccionamos y operamos desde el menú 'Edit'. Además, siempre podemos utilizar las teclas de de acceso rápido ('ctrl.+C'=copiar, 'ctrl.+V'=pegar, 'ctrl.+X'=cortar, 'delete'=borrar...). Podemos duplicar objetos pulsando y arrastrando con el botón derecho del ratón.

Especificación de los parámetros del bloque: Para ver o modificar los parámetros de cada bloque, hacemos doble click con él ratón sobre el bloque y nos aparece un cuadro de diálogo. Este cuadro será distinto según cual sea el tipo de bloque que estemos tratando

Desconectar bloques: Para desconectar un bloque del modelo sin suprimirlo, mantenemos pulsada la tecla 'shift' y seleccionamos y arrastramos el bloque desde su posición original en él modelo. Esta técnica es útil, por ejemplo, para desconectar bloques.

Redimensionar los bloques: Podemos cambiar él tamaño de los bloques seleccionándolos y arrastrando de las esquinas. También podemos resaltar bloques añadiéndoles sombras con la opción del menú: 'Format → Show Drop Shadow'.

Manipulación de los nombres de los bloques: Los nombres de bloques en Simulink deben ser únicos y contener al menos un carácter. Si picamos con el ratón en el nombre del bloque podemos editarlo, y si lo arrastramos podemos cambiarlo de posición. Para ocultar el nombre de un bloque vamos al menú 'Format → Hide name', y para situarlo en la posición opuesta a la que está (si está arriba ponerlo abajo, o cambiarlo de izquierda a derecha) lo hacemos desde el menú 'Format → Flip name'.

2.- Manipulación de líneas:

Mover segmentos de línea: Si deseamos mover un segmento de línea pulsamos con él ratón el segmento y lo arrastramos hasta la nueva posición. No se pueden mover los segmentos de líneas que están en contacto directo con los puertos de entrada y salida de los bloques.

Dividir una línea en segmentos: Podemos dividir una línea en dos segmentos dejando fijos los extremos en sus posiciones originales. Para ello seleccionamos la línea y picamos con él ratón, al mismo tiempo que mantenemos pulsada la tecla 'shift' (también podemos hacerlo pulsando al mismo tiempo los dos botones del ratón).

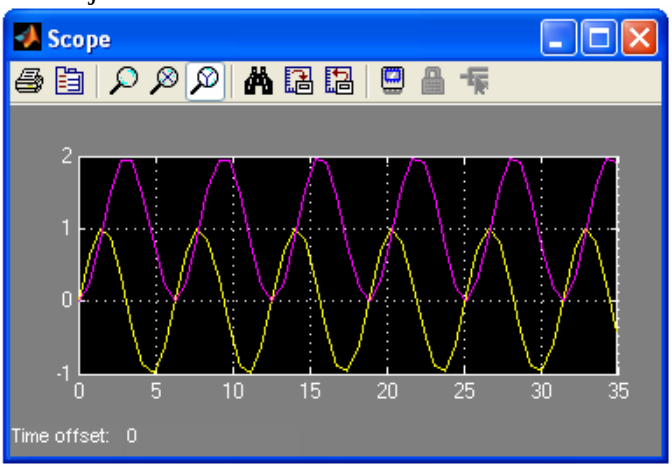
Mover vértices: Podemos mover los vértices de línea en cualquier dirección pulsando y arrastrando con el ratón, excepto los vértices que están en los extremos de línea.

Visualizar señales

Simulink dispone de varios bloques distintos para la visualización de señales. Para nuestro ejemplo arrancaremos nuestro diagrama desde el menú Simulation → Start, y comenzara a ser simulado, para obtener las señales de salida bastara con hacer doble clic en él block Scope para que se nos muestre la ventana de visualización [20].

En color rosa podemos ver la señal que genera él bloque integrador, mientras que en color amarillo vemos la onda senoidal de la que partíamos.

Con él botón podemos ajustar la escala del visualizador de forma automática.



Además, mediante el botón con el dibujo de la lupa podemos acercar o alejar la escala de la gráfica según pulsemos con él botón derecho del ratón (acercar), o con él izquierdo (alejar).

Guardar un modelo

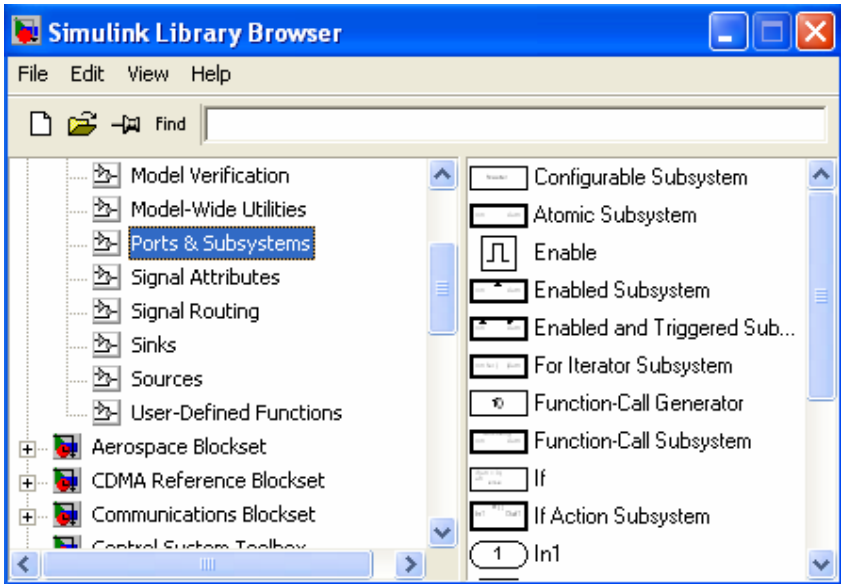
Para guardar un modelo vamos al menú 'File – Save', si él modelo ya tiene un nombre asignado, o 'File – Save As...', cuando es la primera vez que vamos a guardarlo.
El modelo se guardará con extensión .mdl.

Subsistemas

Como mencionamos al principio de este capitulo la creación de subsistemas puede simplificar bastante nuestro diagrama de bloques, ya que consiste en agrupar varios bloques como si fueran uno solo. Para acceder a los elementos que forman el subsistema basta con hacer doble clic con el ratón sobre el bloque subsistema.

Existen dos formas de crear Subsistemas:

Una es seleccionando los bloques que queremos que lo formen y eligiendo en el menú 'Edit – Create Subsystem'. Otra es escogiendo directamente de la librería 'Ports & subsystem'.

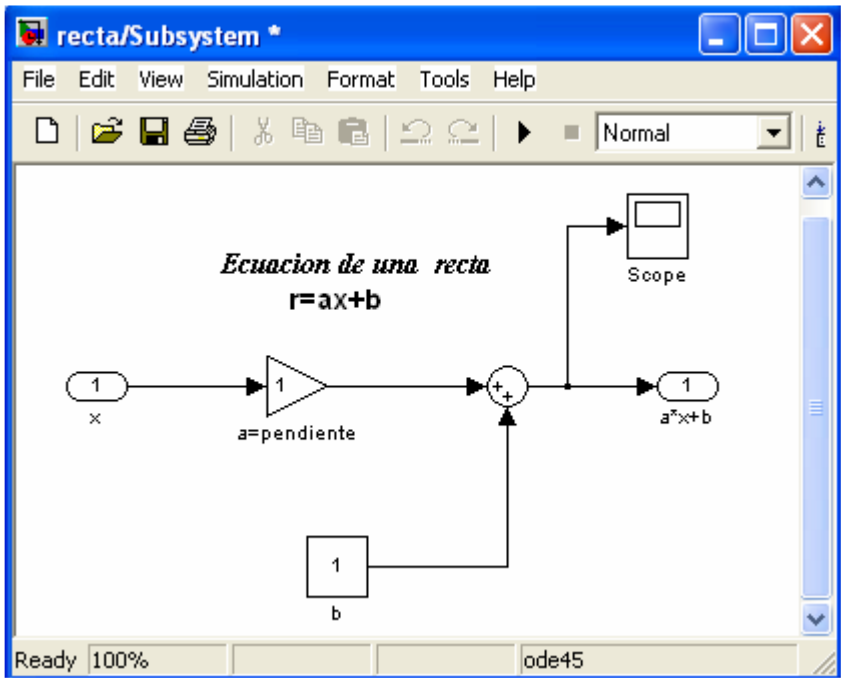


Uso de subsistemas.

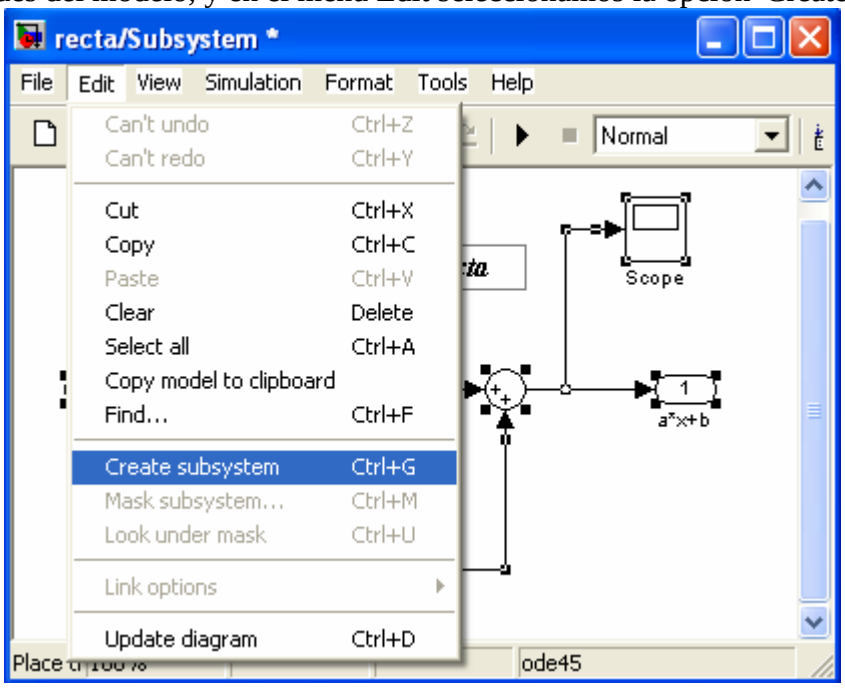
El uso de subsistemas es una poderosa herramienta de Simulink que permite personalizar el cuadro de diálogo y el icono de un subsistema.
Con esto conseguimos:
1.-Simplificar el manejo del modelo, reemplazando el cuadro de diálogo de cada bloque del subsistema por uno sólo. En lugar de abrir cada bloque del modelo para introducir el valor de los parámetros, estos parámetros se introducen en el cuadro de diálogo de la máscara y pasan directamente a los bloques del sistema enmascarado.
2.-Usar una interfaz más fácil y descriptiva, ya que incluimos nuestra propia definición del sistema, texto de ayuda, e incluso podemos usar nuestras propias etiquetas de campo.
3.-Usar los valores de los parámetros de los bloques para definir nuevas variables.
4.-Crear un icono para el bloque que represente el propósito del subsistema
5.-Evitar modificaciones accidentales del subsistema, al ocultar su contenido bajo la máscara.

Ejemplo de subsistema enmascarado

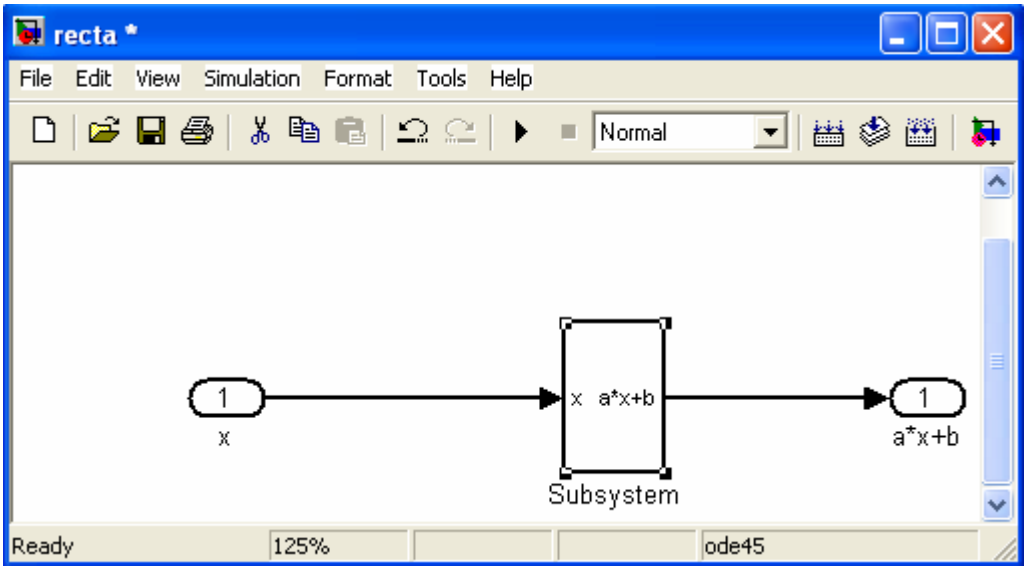
Para hacer un ejemplo de cómo enmascarar un subsistema, primero vamos a crear el subsistema. Para ello, partimos de un modelo sencillo, que representará la ecuación de una recta.
La ecuación de la recta es: r = ax + b. Para representarla, usamos una señal "x", la cual multiplicamos por una pendiente "a". Esto lo llevamos a cabo con un bloque ‘gain’.
Finalmente, a ese producto "ax" le sumamos una constante "b". El resultado final es: ax+b.



Una vez que tenemos creado el modelo, para crear el subsistema seleccionamos con el ratón todos los bloques del modelo, y en el menú Edit seleccionamos la opción 'Create Subsystem':



Vemos que él subsistema se representa como un bloque en el que solo vemos las entradas y las salidas:



Si queremos ver otra vez el contenido del subsistema y acceder a sus bloques, hacemos doble clic con el puntero del ratón y veremos lo que hay en su interior.

Ejemplos de Simulink

Ejemplo 1

Respuesta transitoria a una entrada escalón

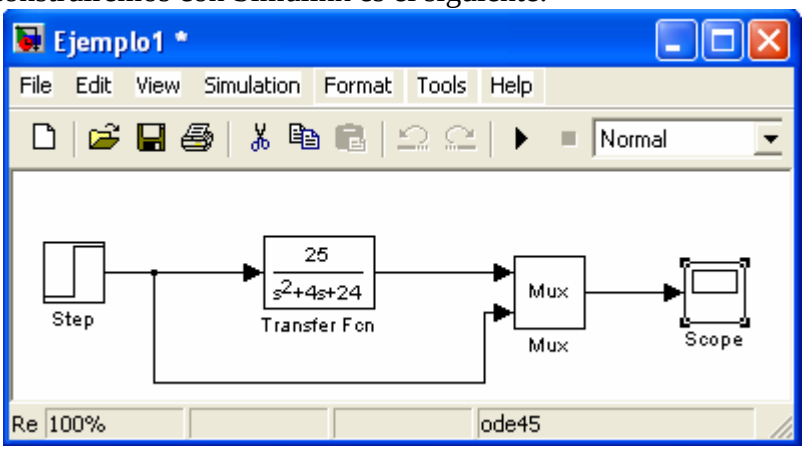
Si deseamos visualizar la repuesta del sistema G(s) ante una entrada escalón, del siguiente sistema [26]:

$$G(s) = \frac{25}{s^2 + 4s + 25}$$

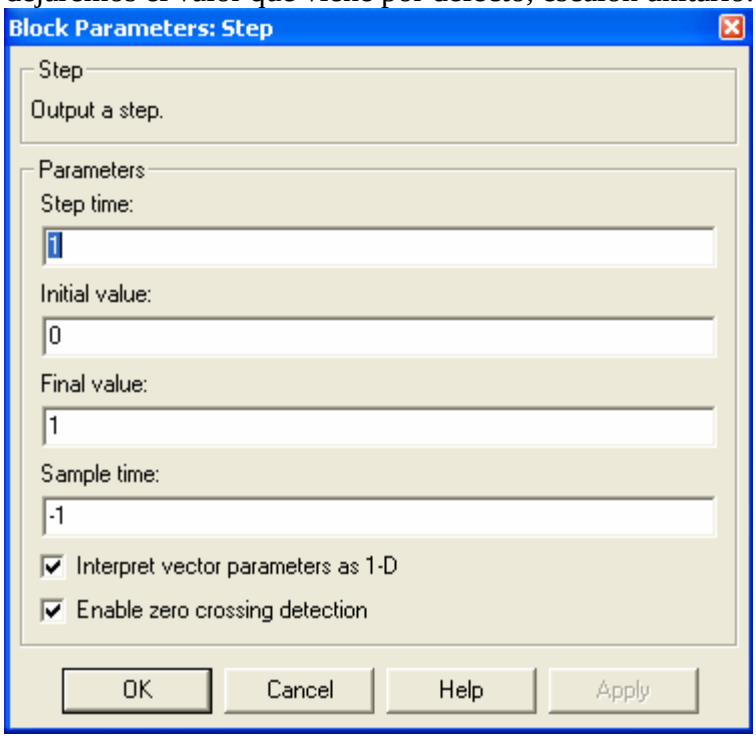
Para ello, los bloques los obtendremos de las siguientes librerías:

BLOQUE	LIBRERÍA
Step	Sources
Transfer Fcn	Continuos
Mux	Signal Routing
Scope	Sinks

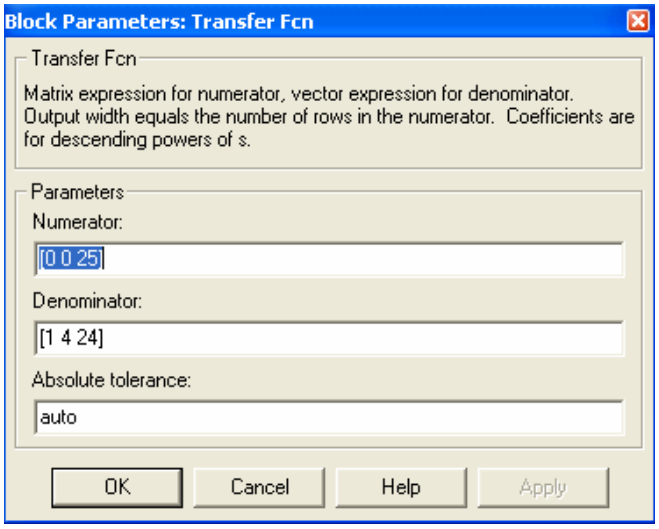
El modelo que construiremos con Simulink es el siguiente:



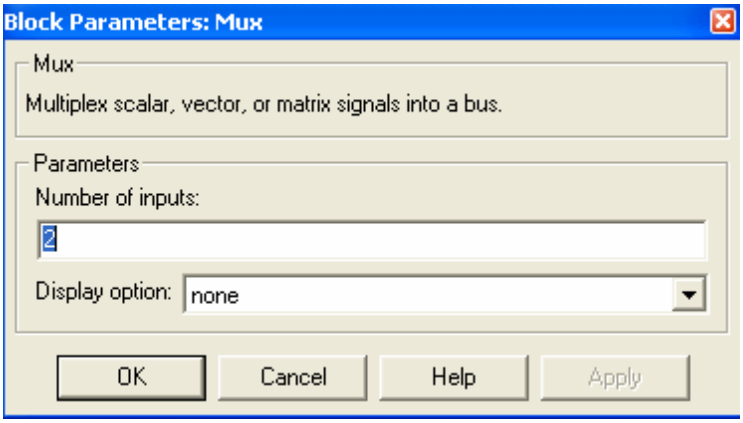
El valor de los bloques lo definiremos como sigue:
En el bloque Step dejaremos el valor que viene por defecto, escalón unitario:



En él bloque 'Transfer Func' pondremos como numerador [0 0 25], y como denominador [1 4 24]

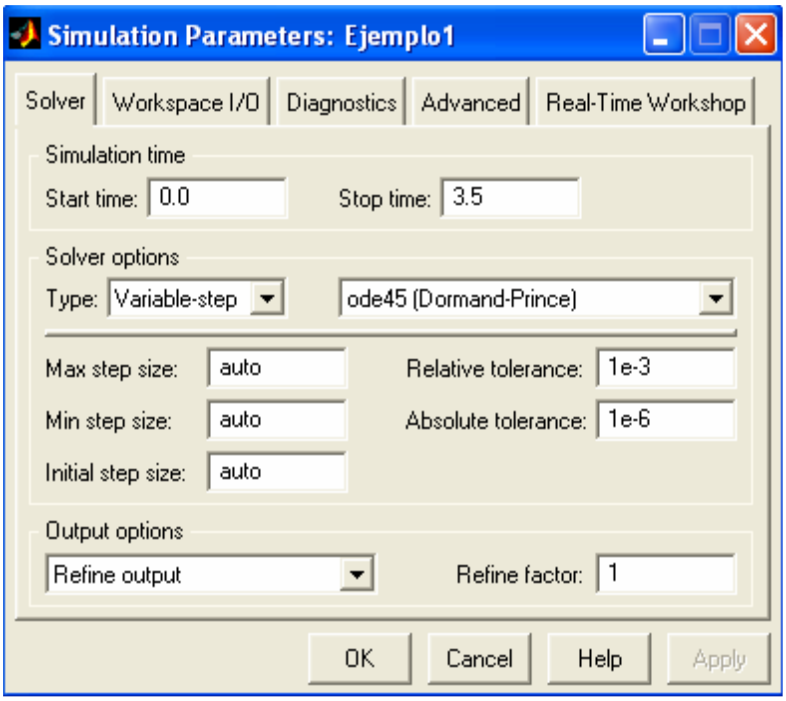


En el bloque Mux pondremos el número de entradas a 2

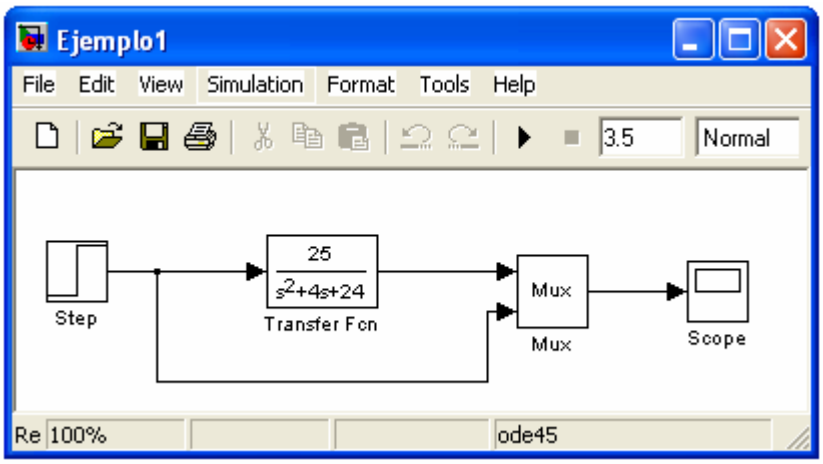


Respuesta del sistema

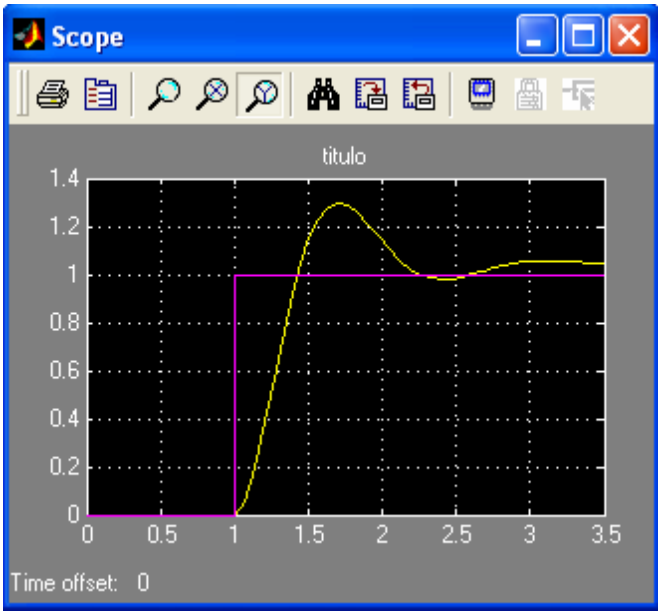
Para ver la respuesta del sistema a la entrada en escalón comenzamos la simulación, con la opción 'Start' del menú Simulación. Antes ajustamos él tiempo de simulación. Un valor pequeño de tiempo será suficiente para ver la respuesta (3.5 sg)



Ahora comenzamos la simulación:



Si abrimos el bloque scope, vemos la repuesta del sistema ante la entrada escalón (pulsar para ampliar)

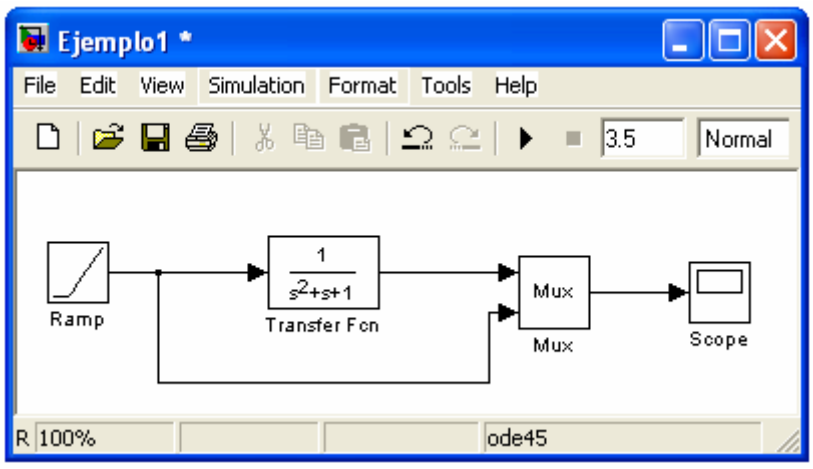


Ejemplo 2

Respuesta transitoria a una entrada rampa [26].
Igual que el ejercicio anterior, pero queremos visualizar la repuesta del sistema G(s) ante una entrada rampa:

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

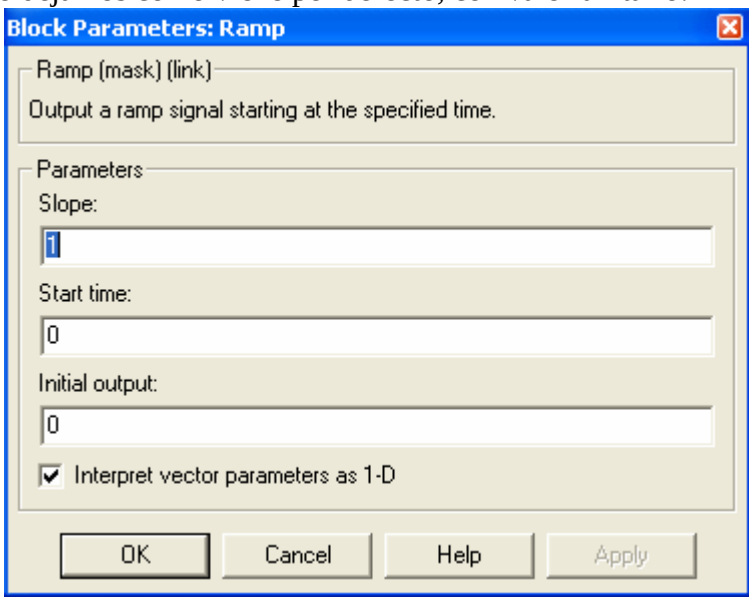
El modelo que construiremos con Simulink es él siguiente:



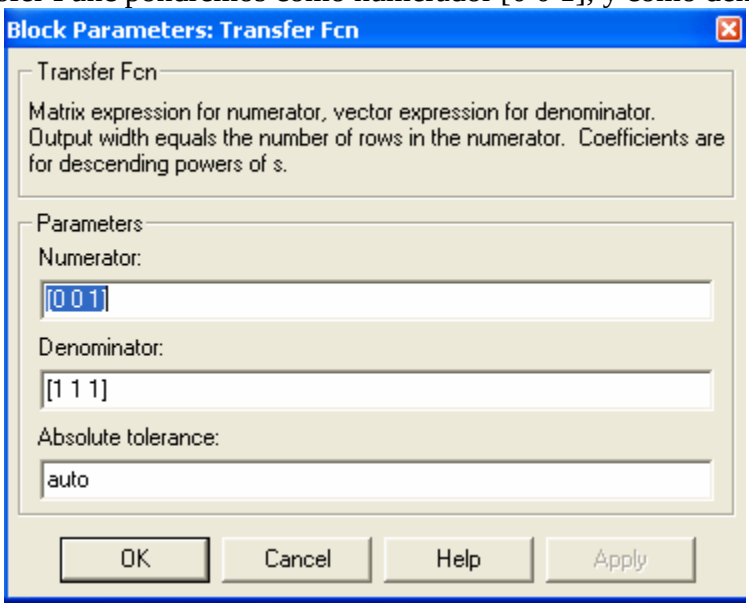
Para esto, los bloques los obtendremos de las siguientes librerías:

BLOQUE	LIBRERÍA
Ramp	Sources
Transfer Fcn	Continuos
Mux	Signal Routing
Scope	Sinks

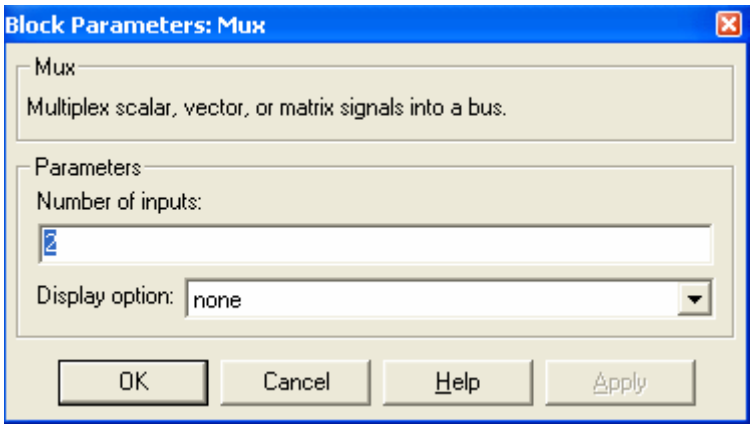
Los valores de los bloques serán los siguientes:
El bloque rampa lo dejamos como viene por defecto, con valor unitario:



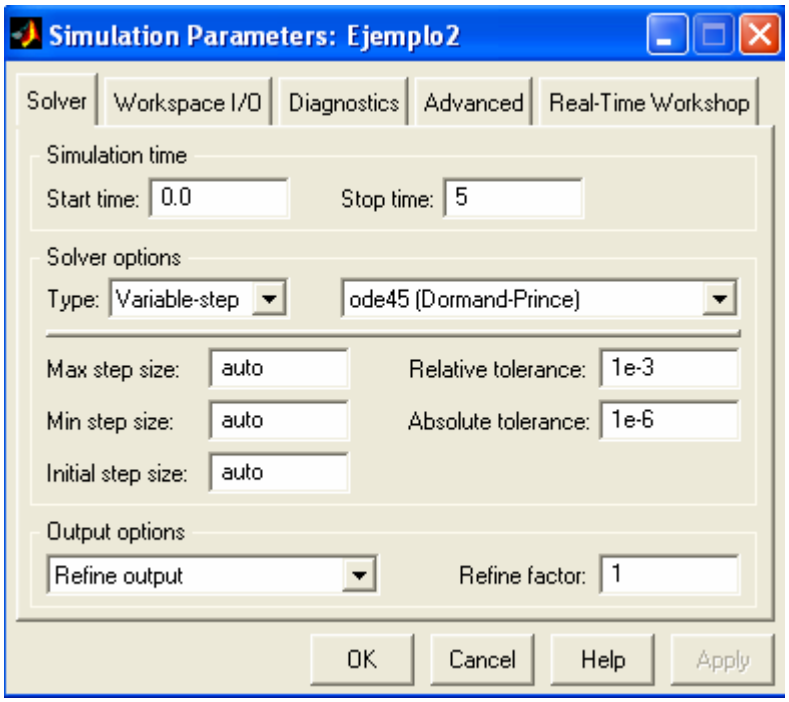
En el bloque Transfer Func pondremos como numerador [0 0 1], y como denominador [1 1 1]



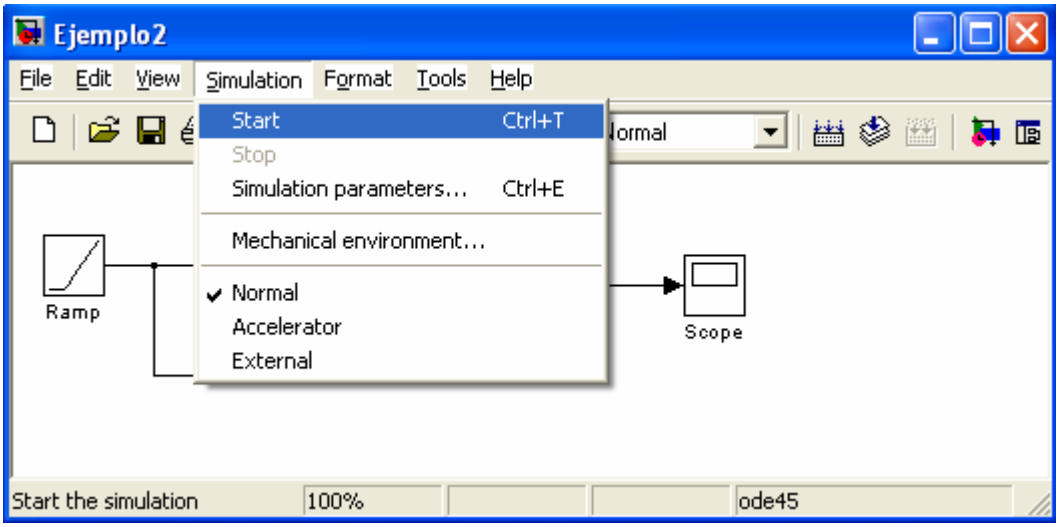
En el bloque Mux pondremos el número de entradas a 2



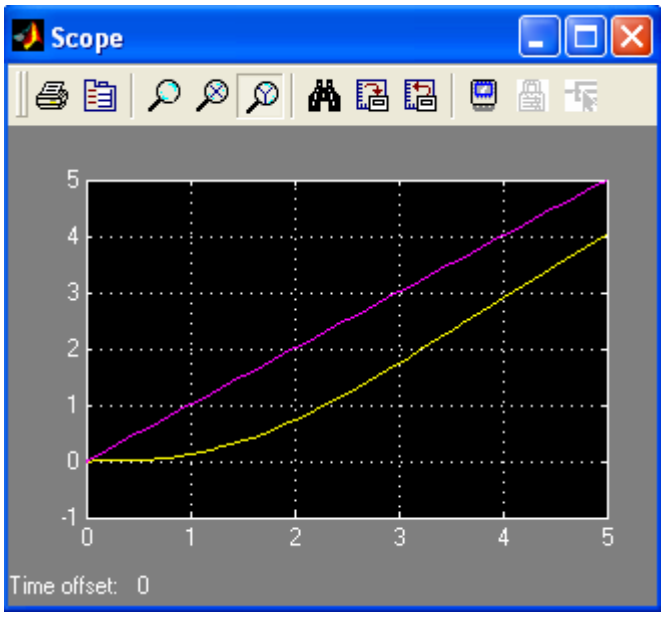
Pondremos un tiempo de simulación pequeño, unos 5 segundos:



Para ver la respuesta del sistema a la entrada en escalón comenzamos la simulación, con la opción ‘Start’ del menú Simulación:



Si abrimos él bloque 'scope', la repuesta del sistema ante una entrada rampa (pulsar para ampliar)



Ejemplo 3

Curvas de Respuesta ante una entrada escalón de un sistema de segundo orden [26]
Consideramos un sistema de segundo orden:

$$\frac{C(s)}{R(s)} = \frac{\omega_n{}^2 s}{s^2 + 2\varsigma\omega_n + 1}$$

Para una entrada impulso unitario, R(s)=1, por tanto:

$$\frac{C(s)}{R(s)} = \frac{\omega_n{}^2 s}{s^2 + 2\varsigma\omega_n + 1} * \frac{1}{s}$$

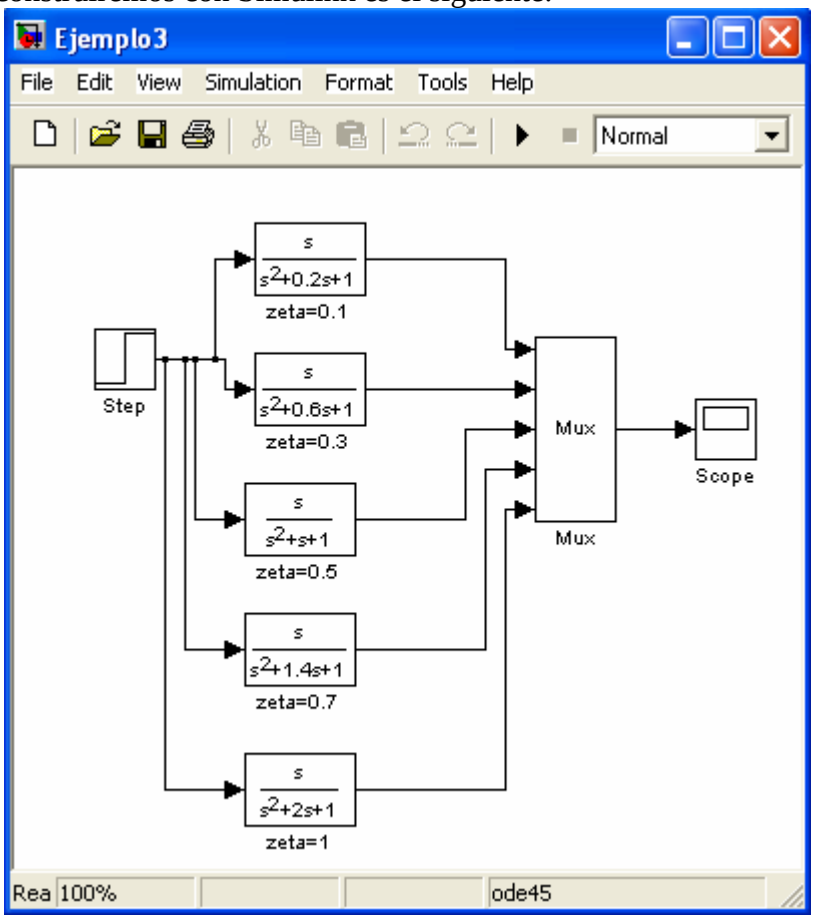
Consideremos el sistema normalizado, donde ωn=1, entonces:

$$\frac{C(s)}{R(s)} = \frac{s}{s^2 + 2\varsigma\omega_n + 1} * \frac{1}{s}$$

Vamos a considerar cinco valores diferentes de zeta: ς=0.1, 0.3, 0.5, 0.7, 1.0, y vamos a obtener las curvas de respuesta al escalón unitario para cada zeta con Simulink. Para construir él modelo, vamos a utilizar 5 funciones de transferencia, una para cada uno de los denominadores que obtendremos para cada valor de zeta. Por ejemplo, la primera FDT será:

$$FDT1 = \frac{s}{1s^2 + 2 * 0.1 * s + 1} = \frac{s}{1s^2 + 0.2s + 1}$$

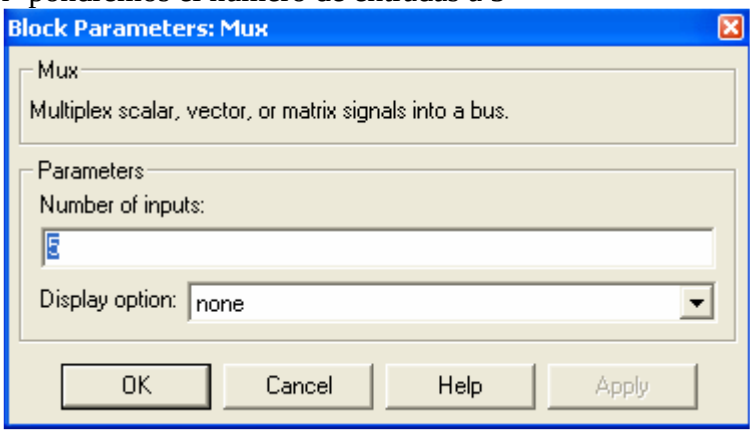
El modelo que construiremos con Simulink es el siguiente:



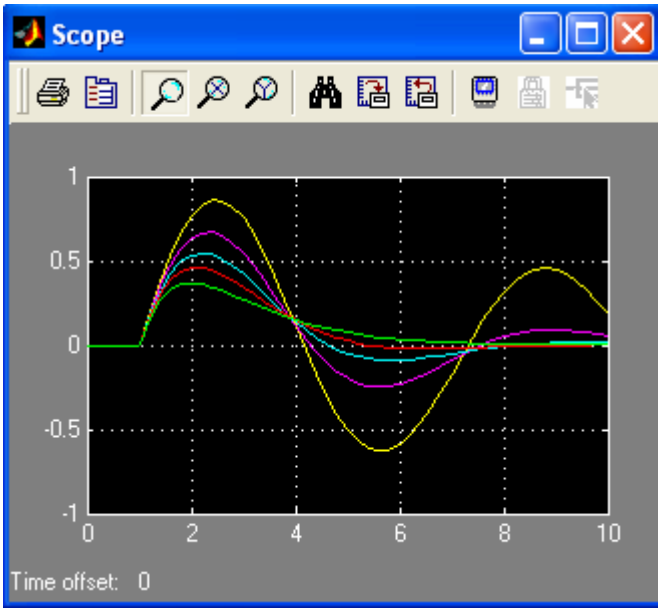
Para ello, los bloques los obtendremos de las siguientes librerías:

BLOQUE	LIBRERÍA
Step	Sources
Transfer Fcn	Continuos
Mux	Signal Routing
Scope	Sinks

Los únicos bloques en que hay que definir parámetros son el bloque ‘Mux’ y el bloque ‘Transfer Func’.
En el bloque ‘Mux’ pondremos el número de entradas a 5



En cada bloque Transfer Func pondremos cada uno de las cinco funciones de transferencia, una para cada valor de zeta
Para ver la respuesta del sistema a la entrada en escalón comenzamos la simulación, con la opción ‘Start’ del menú Simulación:
En el bloque scope podemos ver las cinco curvas que se forman, según el valor de zeta: (pulsar para ampliar)



Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Scilab es un software libre con aplicaciones en la computación para el análisis matemático. Posee una extraordinaria versatilidad y capacidad para resolver problemas aplicados a la Física, Matemáticas, Ingeniería y otras aplicaciones, con un enorme parecido en su sintaxis a Matlab [8].

Introducción a SCILAB.

Si tú tienes una manzana, yo tengo una manzana y las intercambiamos, seguiremos teniendo una manzana cada uno. Pero si tú tienes una idea, yo tengo una idea, y las intercambiamos, cada uno de nosotros tendrá dos ideas [15].

Bernard Shaw (atribuido)

Scilab está concebido como un software libre, como usuarios podemos modificarlo y distribuirlo. Emplea un entorno de ventanas amigable que recuerda mucho a Matlab, Maple, Mathematica, etc.
Se encuentra registrado y avalado ante la “Fundación para el Software Libre”, organización creada por Richard Stallman con el propósito de difundir el software libre, ésta enfocada a eliminar las restricciones sobre el copiado, redistribución, entendimiento y modificación de programas de computadoras, esto con el fin de que se propicie el desarrollo y uso del software libre en todas las áreas de la computación.

Según la enciclopedia libre Wikipedia. Software libre es el software que, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente. El software libre suele estar disponible gratuitamente en Internet, o a precio del costo de la distribución a través de otros medios; sin embargo no es obligatorio que sea así y, aunque conserve su carácter de libre, puede ser vendido comercialmente. Análogamente, el software gratuito (denominado usualmente Freeware) incluye en algunas ocasiones el código fuente; sin embargo, este tipo de software no es libre en el mismo sentido que el software libre, a excepción que se permitan modificaciones y redistribuciones de dichas versiones modificadas del programa [17].

Libertades del Software Libre

A continuación se enuncian las 4 libertades básicas y pilares fundamentales de todo “software libre":

"libertad 0" Ejecutar el programa con cualquier propósito (privado, educativo, público, comercial, etc.)

"libertad 1", Estudiar y Modificar el programa (para lo cuál es necesario poder acceder al código fuente)

"libertad 2", Copiar el programa de manera que se pueda distribuir a cualquiera persona.

"libertad 3", Mejorar el programa, y hacer Públicas las mejoras, de forma que nos beneficiemos toda la comunidad.

Es importante señalar que las libertades 1 y 3 obligan a que se tenga acceso al código fuente. La "libertad 2" hace referencia a la libertad de redistribuir el software libremente bajo algún tipo de licencia de software libre que beneficie a la comunidad a la cual pertenecemos todos los usuarios [18]...

El término software no libre se emplea para referirse al software distribuido bajo una licencia de software más restrictiva que no garantiza estas cuatro libertades. Las leyes de la propiedad intelectual reservan la mayoría de los derechos de modificación, duplicación y redistribución para el dueño del copyright; el software dispuesto bajo una licencia de software libre permite específicamente la mayoría de estos derechos reservados.
La definición de software libre contempla el asunto del precio; un eslogan frecuentemente usado es "libre como en libertad de expresión no como en cerveza gratis" aludiendo a la ambigüedad del término inglés "free", es habitual ver a la venta CD’s de software libre como distribuciones. Sin embargo, en esta situación, como compradores del CD tenemos el derecho de copiarlo y redistribuirlo.
Para evitar la confusión, algunas personas utilizan los términos "libre" (Libre software) y "gratis" (Gratis software) para evitar la ambigüedad de la palabra inglesa "free". Sin embargo, estos términos alternativos son usados únicamente dentro del movimiento del software libre, aunque están extendiéndose lentamente hacia el resto del mundo [15].

Ejemplos y evolución

Existe una gran cantidad de software, cada vez mayor, disponible bajo licencias de software libre. Los observadores (y adeptos) a menudo interpretan este fenómeno como el movimiento del software libre., el servidor de transporte de correo Sendmail, el Servidor web Apache, , el sistema X Window, la suite de ofimática OpenOffice.org, el navegador Mozilla son algunos ejemplos.

Significación Política

Una vez que un producto de software libre ha empezado a circular, rápidamente está disponible a un costo muy bajo o sin costo. Al mismo tiempo, su utilidad no decrece. Esto significa que el software libre se puede caracterizar como un bien público en lugar de un bien privado. Aunque realmente no lo es en ningún momento.

Puesto que el software libre permite el libre uso, modificación y redistribución, a menudo encuentra un hogar en los países del tercer mundo para los cuales el costo del software no libre es a veces prohibitivo. También es sencillo modificarlo localmente, lo que permite que sean posibles los esfuerzos de traducción a idiomas que no son necesariamente rentables comercialmente. La mayoría del software libre se produce por equipos internacionales que cooperan a través de la libre asociación. Los equipos están típicamente compuestos por individuos con una amplia variedad de motivaciones. Existen muchas posturas acerca de la relación entre el software libre y el actual sistema económico capitalista.
Algunos consideran el software libre como un competidor del capitalismo, una forma de anarquismo práctico. Algunos más consideran el software libre como otra forma de competición en el mercado libre, y que el copyright es una restricción gubernamental sobre el mercado.
Otros comparan el software libre a una economía del regalo, donde el valor de una persona está basado en lo que ésta da a los demás.
Grupos como Oekonux e Hipatia consideran que todo debería producirse de ésta forma y que este modelo de producción no se limita a reemplazar el modelo no libre de desarrollo del software. La cooperación basada en la libre asociación puede usarse y se usa para otros propósitos (tales como escribir enciclopedias, por ejemplo).

Seguridad Relativa

Existe una cierta controversia sobre la seguridad del software libre frente al software comercial. Un método usado de forma habitual para determinar la seguridad relativa de los productos es determinar cuántos fallos de seguridad no corregidos existen en cada uno de los productos involucrados. Por lo general los usuarios de este método recomiendan que cuando un producto no proporcione un método de corregir los fallos de seguridad, no se use dicho producto, al menos hasta que no esté disponible un arreglo.
A fecha de Diciembre de 2005 el sitio de seguridad “Secunia” cuenta cero fallos de seguridad no parcheados (no arreglados aún) para los productos software libre más usados para navegación de Internet, productividad de oficina y e-mail -Mozilla Firefox, OpenOffice.org y Mozilla Thunderbird-, en comparación con los varios fallos de seguridad aún no corregidos para cada uno de los tres principales productos no libres equivalentes (hechos por Microsoft) -Internet Explorer, Microsoft Office y Outlook Express.

Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Introducción a Scilab.

Fue desarrollado en el INRIA, “Institut National de Recherche en Informatique et Automatique”, un excelente instituto francés de investigación. Posteriormente colaboro la escuela de ingenieros aeronáuticos ENPC, “Ecole Nationale de Ponts et Chaussées”. Actualmente hay un gran consorcio con empresas como Renault, Peugeot-Citroen, CEA “Commissariat á l’Energie Atomique”, CNES “Centre National d'Etudes spatiales”, Dassault Aviation, EDF “Electricité de France”, [8]...

Es un software libre (free), similar a Matlab y a otros paquetes comerciales existentes, Utiliza un ambiente gráfico interactivo combinado con un lenguaje de programación de alto nivel.

Sus principales características son:

- software para cálculo científico.
- Interactivo.
- Programable.
- De libre uso, con la condición de hacer referencia a los autores.
- Disponible para Windows y Linux.

En el sistema Scilab están disponibles las siguientes herramientas:
• Gráficas 2D y 3D.
• Álgebra lineal.
• Polinomios y funciones.
• Scicos: Simulador por diagrama por bloques.
• Estadística
etc.

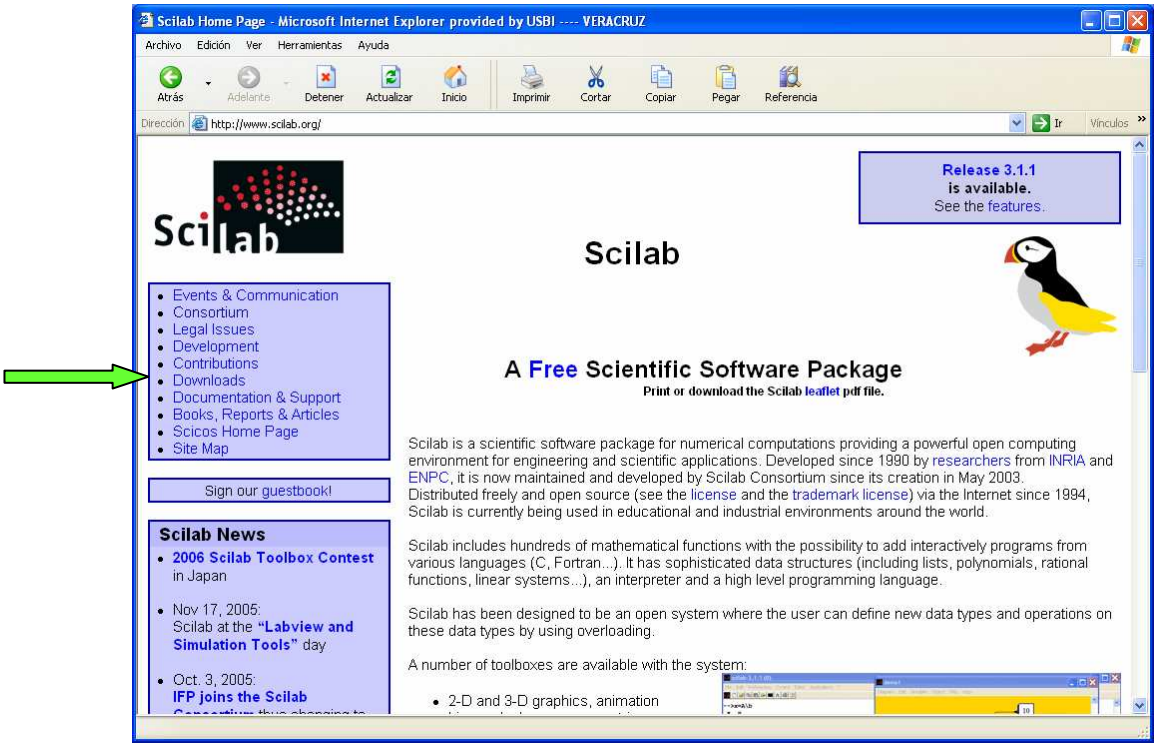
Scilab está disponible gratuitamente en la página Web de WWW.SCILAB .ORG
Allí se encuentra información general, manuales, FAQs (frequent asked questions), referencias sobre reportes, diferencias con Matlab, lista de errores, etc.

Una vez descargado e instalado, ocupará alrededor de 120 MB en el disco duro de nuestras computadoras. Un paquete como Scilab puede satisfacer nuestras necesidades de estudiantes y futuros profesionales de ingeniería y ciencias cuando no se dispone de software comercial, como por ejemplo Matlab, dado el alto costo de las licencias comerciales académicas y profesionales y la dificultad de adquirir versiones estudiantiles del software comercial en algunos países.

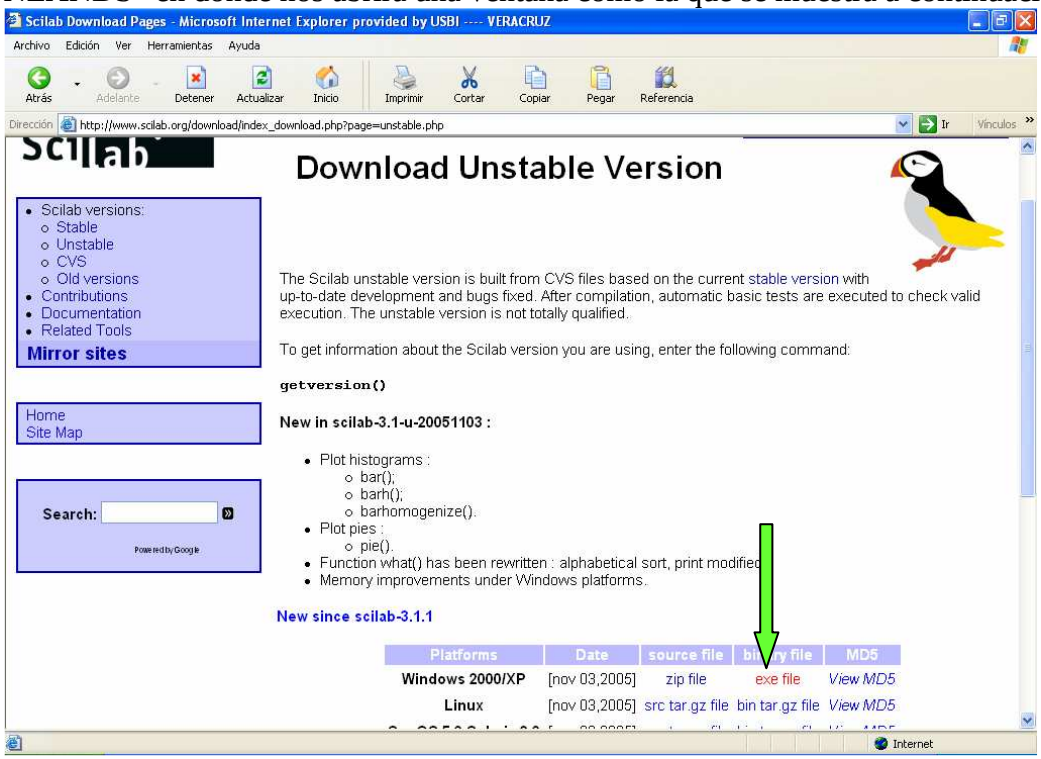
Como obtener Scilab e instalarlo.

Scilab está disponible para el entorno Windows y Linux. Se encuentra disponible en "http://www.Scilab.org/" en formato .exe, para los sistemas más conocidos en nuestro caso usaremos la versión WINDOWS XP.

Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.



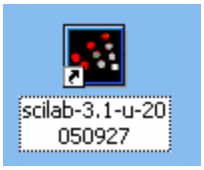
Una vez ubicados en la página de Scilab, nos dirigimos al recuadro de la izquierda donde se aprecia una columna y en la sexta fila de arriba hacia abajo hacemos un doble clic en "DOWNLANDS" en donde nos abrirá una ventana como la que se muestra a continuación:



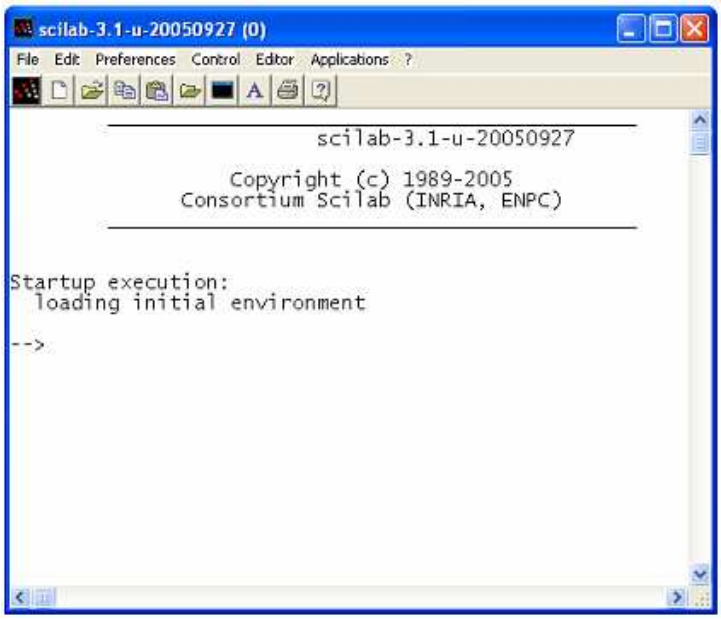
Una vez mostrada la ventana de arriba, hacemos doble clic en el recuadro de abajo en donde se lee "NEW SINCE SCILAB 3.1.1 y se elige por el sistema operativo que se deseé como la mayoría de nosotros tiene instalada en sus computadoras un sistema operativo Windows elegimos el archivo exe.file, nuevamente hacemos doble clic y nos aparecerá el cuadro de descarga de Windows donde nos indicara si deseamos abrir, guardar o cancelar, en guardar damos doble clic y empezara la descarga, ésta toma aproximadamente 5 a 15 minutos dependiendo de la velocidad de conexión que se tenga, una vez que se haya descargado, nos aparecerá el lugar donde lo hayamos descargado el archivo ejecutable. A continuación el archivo se ejecutara como cualquier programa que hayamos instalado en Windows.
Es importante decir que **www.Scilab.org** va actualizando con frecuencia la versión, corrigiendo errores e implementando nuevas funciones [12].

Inicio de Scilab.

Una vez instalado Scilab correctamente en nuestra PC, se creara una icono con su logo en nuestro escritorio en la siguiente figura se puede apreciar como aparecerá, a continuación procedemos a hacer doble clic para ejecutarlo, o también podemos dirigirnos al menú INICIO dentro del menú programas y buscar la carpeta de Scilab. Procedemos hacer doble clic en Scilab 3.1.1



Una vez arrancado nos aparecerá la primera ventana de Scilab con el siguiente encabezado y el signo del Prompt -- >, el signo del Prompt indica que Scilab ésta listo para recibir y ejecutar los comandos que le indiquemos.



PRINCIPALES DIFERENCIAS CON MATLAB.

Antes de empezar con nuestra primera experiencia con Scilab, debemos conocer que Scilab y Matlab tienen muchas similitudes. Pero existen algunas diferencias que se deben conocer antes de empezar a introducir comandos, de tal forma que podamos ir asimilando según vayamos avanzando.

Sci. Files

Los Sci.Files son programas que podemos crear en el editor de Scilab, llamado SciPad, podemos acceder a el desde el menú Editor para después ser guardados en C:\Documents and Settings\Mis Documentos podemos cambiar ésta ubicación a nuestro modo más sencillo de trabajo, para ejecutar el programa deseado debemos de hacerlo con el comando "exec", de la siguiente manera exec (`nombre_del_archivo.sci')

Líneas de comentario.

El entorno de Scilab es muy parecido a los demás programas es por eso que para mantener el orden de las líneas de código es recomendable usar comentarios en Scilab empiezan con //, mientras que en Matlab empiezan con %.

Variables Booleanas.

Las variables booleanas en Scilab son %T y %F, mientras que en Matlab son 0 y 1.

Polinomios.

En Scilab los polinomios y matrices polinómicas son definidas por la función poly, mientras que en Matlab son considerados como vectores de coeficientes [12].
La forma de declarar el orden de los polinomios es **inversa** a la manera de hacerlo con Matlab.

Gráficos.

Son muy similares, con el comando plot.

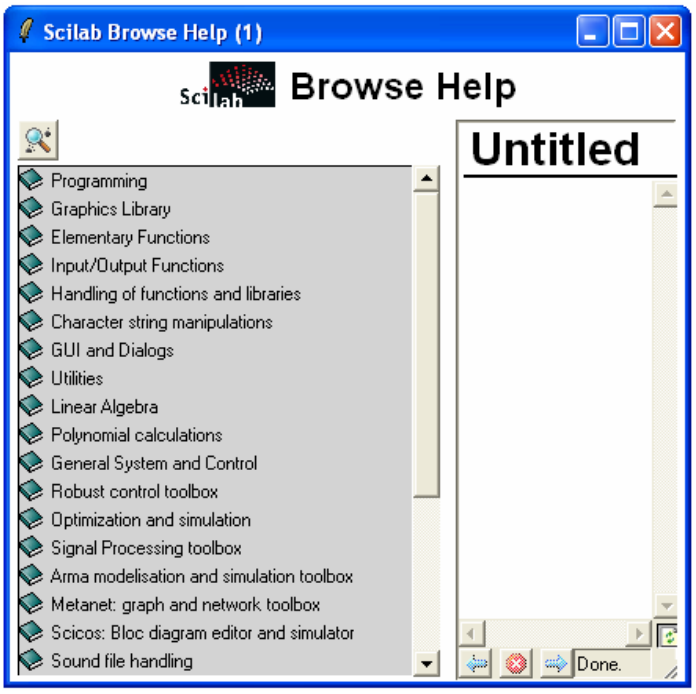
Scicos (Scilab) y Simulink (Matlab).

Ninguno de los dos simuladores son compatibles.

Los modos de trabajar en Scilab son:

a) Modo interactivo: Ejecución de cualquier expresión, programa o función Scilab dentro del entorno.
b) Modo Bach: Ejecución de un programa, previamente escrito y evaluado.

La ayuda en Scilab se muestra tecleando help en la Ventana Principal o pulsando el icono del signo de interrogación. En cualquier caso, si no incluimos ninguna palabra sobre la que solicitamos la ayuda, Scilab devolverá una lista con los temas existentes tal como ésta...



Si deseamos tener información mas detallada sobre alguna función basta con escribir ‘help’ y a continuación el nombre de la función o de la orden. Por ejemplo
‘help evans’. Para esto se requiere que la función “evans” exista.
Al usar la orden who, Scilab muestra las variables que estamos usando en ese momento.

Algunas reglas básicas.

Scilab dispone también de aproximadamente 500 comandos [29], los cuales tienen ciertos formatos de escritura. En primer lugar, vamos a descubrir que también Scilab distingue entre mayúsculas y minúsculas por lo que casi siempre los comandos e instrucciones básicas se escriben con minúsculas.
Por ejemplo, no es lo mismo %pi que %Pi. Podemos teclear ambas expresiones y descubrir que la primera funciona perfectamente mientras que la segunda causa un error al ser ejecutada. La precisión utilizada es aproximadamente 10 dígitos. La presición con que deseamos trabajar puede ser modificada por la siguiente sintaxis:
format ([type], [long]) de donde

Type es para tipo de caracteres
"v": Para el formato de la variable (default).
"e": Para el formato exponencial.

Long: El numero máximo de dígitos que vamos a utilizar.

La presición con que deseamos trabajar puede ser modificada por los siguientes comandos:

Comando	Exhibición	Ejemplo
format('v',7)	6 dígitos	0.6667
format('e',5);	4 dígitos	6.7D-01
format('v',14)	13 dígitos	0.66666666667
format('e',15)	15 dígitos	6.66666667D-01
format('v',3)	4 dígitos	0.67

Para regresar al formato inicial, el formato “variable" (el predefinido por Scilab) se usa format ('v'), por ejemplo:

```
format('v', 10)
```

Para los Números decimales Scilab solo realiza conversiones entre estos por medio del comando “dec2hex” [27]. Ejemplo:

```
dec2hex([10 11 12 13 14 15 16])
 ans  =
!A  B  C  D  E  F  10  !
```

Algunos comandos de mucha utilidad son:

pwd	Nos indica cual es el directorio de trabajo actual.
apropos	Despliega una ventana con la descripción específica de un comando y permite ver información sobre otros temas relacionados
helpdesk	Realiza una búsqueda en hipertexto en un buscador Web proporcionando un acceso directo a toda la documentación: PDFs, información sobre la solución de problemas.
figure	Crea una nueva ventana para crear una gráfica.
close	Cierra la ventana donde se creo alguna gráfica.
cd	Cambia la ruta al subdirectorio superior.

Los paréntesis “( )” y los corchetes “[ ]” tienen significados y utilidades distintas. Los primeros se utilizan para evaluar funciones, ej. (sin(pi/5)) y para cambiar el orden básico de las operaciones, por ejemplo (4*(26+5*(2+2))), mientras que los segundos sirven como delimitadores de vectores por ejemplo [1,2,3,4] o de matrices por ejemplo [5,6,7;8,9,10]. Scilab si permite utilizar las llaves “{ }” con la finalidad de cambiarlas por los corchetes “[ ]”. Scilab dispone de comandos que dan información sobre la fecha y la hora actual. Los comandos más importantes relacionados con la fecha y la hora son las siguientes.

date () devuelve un vector fila que representa año, el mes, el día.
calendar () devuelve una matriz 6×7 con el calendario del mes actual, o del mes y año

Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Uso de Variables y Escalares.

Para asignar un valor a una variable, que llamaremos por ejemplo x, se usa el signo "=". Por ejemplo:
>> x=20

Notemos que este comando se ejecuta, en forma "ruidosa". Para que un comando se ejecute forma silenciosa, utilizamos; ";"después de cada comando. Por ejemplo: x=20; en general, las variables en Scilab serán matrices, pero al asignarle a cualquier variable sólo un número, el programa se da cuenta de que estamos hablando de una matriz de 1x1, es decir, un escalar. Las variables pueden tener cualquier nombre escrito sean largos o cortos, con letras y números. La excepción son algunos nombres ya utilizados por el lenguaje, por ejemplo sin, cos, evans etc.

Para Scilab, como se ha indicado anteriormente, todos los objetos son matrices numéricas. Si deseamos trabajar con símbolos, hemos de indicarle al programa que se trata de símbolos. Para ello se utiliza la función ldivf; por ejemplo, si escribimos: 2/3
El programa responde: 0.6667
En cambio, si queremos que 2 y 3 se consideren como símbolos (de forma que sólo se opere de forma exacta con ellos), deberemos escribir:

```
ldivf('5','4')
 ans  =
 5\4
```

Análogamente se pueden definir variables; por ejemplo, para definir la variable x podríamos escribir:

```
x=poly(0,'x')
```

Una vez que hallamos definida la variable x, podemos realizar operaciones simbólicas con ella:

```
x+3*x
 ans  =
   4x
```

Algunas variables asignadas por Scilab:

Scilab ya tiene asignada algunas variables que solo son necesarias llamarlas seguidas de el signo de %, algunas variables predefinidas por Scilab son, %i, raíz cuadrada de -1 ($\sqrt{-1}$), %pi que representa 3.1416 y %e que represénta 2.7182…. Ej.

```
%pi
%pi  =
3.1415927
```

```
%e
%e  =
2.7182818
```

Otras constantes especiales usadas por Scilab son:

%eps, que representa un número muy pequeño ej.

```
%eps
 %eps  =
   2.220D-16
```

```
r=%eps+1
 r  =
   1.
```

inf, que representa una cantidad muy grande Ej.:

```
%inf
 %inf  =
   Inf
```

```
1/%inf
ans  =
  0.
```

nan, ésta singular constante significa que no es un número, lo podemos comprobar asiendo la siguiente operación, donde la operación que se efectúa tendrá siempre como resultado un nan, Ej.:

```
1+%nan
 ans  =
   Nan
```

Expresiones y funciones.

Como en todos los programas las expresiones matemáticas elementales se construyen empleando los símbolos = (igual), + (suma), - (sustracción),* (multiplicación) / (división), ^ (potenciación) el orden jerárquico de operación es el mismo. Los paréntesis tienen la misma jerarquía que en Matlab la cual sigue siendo para organizar adecuadamente expresiones complejas [24]. Una forma en que podemos decirle al programa que una instrucción demasiado larga no ha terminado y que continúa en la siguiente línea es situar tres puntos al final de la línea.

Manejo de Escalares:

Si deseamos declarar una variable de tipo escalar introducimos el nombre de la variable que va a interpretarla en este caso el nombre es “c” y se iguala con el valor que tendrá dándole a continuación enter, Las sentencias pueden acabar en punto y coma o no, según queramos que aparezca o no reflejado el resultado de comando. Esto es útil en muchos casos, para eliminar el innecesario "ruido" en la pantalla de sentencias.

```
// c tendrá un valor de 20
c=20
 c  =
   20.
```

Las variables en Scilab pueden tener cualquier nombre largos escritos con letras. La excepción son algunas variables ya asignadas automáticamente por Scilab.

El cual tiene la característica de ser un programa Caps-sensitive igual que Matlab, es decir, para el programa Scilab, la variable es distinta si tiene mayúscula o no. Por ejemplo: C es distinto de c, o A es distinto de a. Esto lo podemos comprobar asignándole un valor a la variable C y después volviéndola a llamar para comprobar que su valor es diferente a la variable que anteriormente habíamos declarado (sistema), Ej.:

```
C=3
C  =
   3.
```

Operaciones Básicas:

```
C+c
 ans  =
   23.
```

```
c-C
 ans  =
   17
```

```
C*c
 ans  =
   60.
```

```
C/c
 ans  =
   0.15
```

Como todos sabemos la división por cero ésta indefinida por lo tanto si intentamos realizar ésta operación Scilab nos devolverá un mensaje así:

```
c/0
   !--error 27
división by zero...
```

Potencia:
Para elevar a una determinada potencia basta con usar el signo de "^" o en su defecto el doble signo de la multiplicación
"**" Ej.:

```
2**3
ans  =
  8.
```

```
2^3
 ans  =
   8.
```

Raíces:
Para obtener la raíz cuadrada podemos utilizar el comando "sqrt" seguido de un par de paréntesis con los que encerramos la variable a la cual le queremos substraer la raíz. Para esto crearemos la variable "l" con el resultado de elevar al cuadrado c

```
l=20;,c=2;
l=c^2
l  = 400.
```

```
sqrt(l)
 ans  =
   20.
```

Función Exponencial ex
Para obtener ésta función usamos el comando "exp"

```
exp(C)
 ans  =
   20.085537
```

Logaritmo:

Para obtener el logaritmo usamos el comando log para hallar el logaritmo natural y log10 para hallar el logaritmo en base 10, aunque también podemos obtener el logaritmo en base 5 o 2 usando el comando "log5" o "log2"

```
og(2.7182818)
 ans  =
   1.0000000
```

```
log10(10000)
 ans  =
   4.
```

Eliminación de Variables:

Si deseamos eliminar alguna variable usamos el comando "clear", por ejemplo

```
r=3;
clear r
```

Para seccionarnos que ésta ha quedado eliminado la podemos llamar nuevamente y observar lo que Scilab nos muestra:

```
r
  !--error 4
undefined variable : r
```

Funciones Trigonométricas:

En Scilab podemos obtener las funciones trigonometrícas a las variables declaradas, aunque debemos de tener precaución por que Scilab trabaja en RADIANES y no en grados como normalmente lo hacemos así que para introducir un valor y esperar obtener el valor en grados previamente habremos de convertir de grados a radianes[29].
Por ejemplo para obtener el seno de un ángulo de 30º debemos de convertir a radianes:

$$a = \frac{30^{o} * \pi}{180^{o}}$$

```
l=(30*%pi)/180;
sin(l)
 ans  =
   0.5
```

Algunas de las funciones trigonometrìcas más usuales en Scilab son:

Función	Comando	Ejemplo
Seno	sin	Variab_seno=sin(variable)
Arcoseno	asin	Variab_aseno=asin(variable)
Coseno	cos	Variab_cos=cos(variable)
Arcoseno	acos	Variab_acos=acos(variable)
Tangente	tan	Variab_tan=sin(variable)
Arcotangente	atan	Variab_atan=atan(variable)
Cotangente	cotg	Variab_ctg=cotg(variable)

Matrices:

Para crear una matriz basta dar sus valores entre corchetes separando por espacios blancos los elementos de una misma fila y por punto y coma las distintas filas Scilab las creara automáticamente, así mismo la sintaxis es la misma que en Matlab por ejemplo declaremos la siguiente matriz:

```
v=[4 2 3 6; 1 0 3 2 ]
 v  =
!  4.   2.   3.   6.   !
!  1.   0    3.   2    !
```

Si deseamos localizar elemento, una fila o una columna particular se especifica con la notación (<fila>,<columna>), v(:,<columna>), v(<fila>,:) por ejemplo

```
 v=[4 2 3 6; 1 0 3 2 ] // Definición de una matriz
v =
! 4. 2. 3. 6. !
! 1. 0 3. 2. !
```

```
 v(1,3) // Elemento (1,3) de la matriz
ans = 3.
```

```
v(:,3) // Columna 3 de la matriz
 ans =
! 3. !
! 3. !
```

```
v(1,:) // Fila 1 de la matriz
 ans =
! 4. 2. 3. 6. !
```

También existen las funciones propias del tratamiento matricial. De la misma manera que en Matlab, tenemos la inversa "inv()", determinante "det()", etc. La operación denominada transposición que consiste en convertir las filas en columnas y las columnas en filas de un vector o matriz (v') se realiza situando el apóstrofe " ' " a la derecha del vector, por ejemplo v' .

```
v' // Inversa de la matriz v
 ans  =
!   4.    1.      !
!   2.    0       !
!   3.    3.      !
!   6.    2. + i  !
```

Operaciones punto a punto:

Para realizar las operaciones básicas como son el producto, potencia, división y división inversa en formato "punto a punto" debemos anteponer "." a la primera variable sujeta a la operación. Esto hace que dicha operación se realice elemento por elemento dentro de la matriz. Podemos darnos cuenta que la suma y resta son operaciones elemento por elemento en matrices y, por lo tanto, no necesitamos usar "." Por ejemplo, declaramos las siguientes matrices de 1x3:

```
v1=[1,2,3]; v2=[4,5,6];
```

Operación

Producto

```
v1.*v2
 ans  =
!   4.    10.    18. !
```

Potencia

```
v1.^2
 ans  =
!   1.    4.    9. !
```

División

```
v1./v2
 ans  =
!   0.25    0.4    0.5 !
```

Multiplicación

```
v1.*v2
 ans  =
!   4.    10.    18. !
```

De la misma forma que los escalares, las matrices pueden tener poseer cualquier nombre largo alfanumérico.

Otras Maneras de Crear Matrices:

zeros(n,m) Generamos una matriz compuesta de ceros por n filas y m columnas.
ones(n,m) Generamos una matriz compuesta de unos por de n filas y m columnas.
eye(n,m) Generamos una matriz identidad de n filas y m columnas.
rand(n,m) Generamos una matriz con valores aleatorios [0 a 1], de n filas y m columnas.

Generando Argumentos

En Scilab podemos generar argumentos de números muy fácilmente, utilizando el operador ":". Se utiliza la siguiente sintaxis:
nombre = principio:incremento:fin
Con este comando podemos generar una serie de números reales, que comienza en "principio", que avanza de elemento en elemento en "incremento", hasta que la serie alcanza (o supera)el valor de "fin". Si queremos que una serie tenga n elementos, comience en principio y termine en fin, debemos utilizar un el comando genérico: serie = principio:(fin-principio).

```
v=5:-0.5:3
 v  =
!  5.   4.5   4.   3.5   3. !
```

Existen también otros vectores útiles como el vector nulo formado por N ceros y el vector constante igual a 1 formado por N unos, esto se consigue con las sentencias: zeros (1: N), por ejemplo para una matriz de ceros, que se valle incrementado por ceros a partir de 1 hasta llegar a 5 columnas nos quedaría de la siguiente forma.

```
zeros(1:5)
 ans  =
!  0.   0.   0.   0.   0. !
```

Para crear una fila de unos que se incremente de uno hasta,

```
ones(1:5)
 ans  =
!  1.   1.   1.   1.   1. !
```

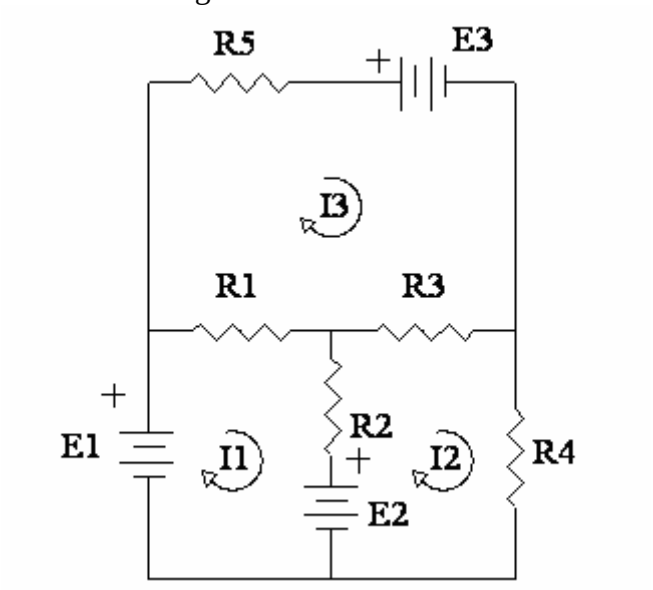
De la misma forma para incrementar una fila desde 4 hasta 5:

```
ones(4:5)
 ans  =
!  1.   1. !
```

Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Ejemplo de Aplicación Usando Matrices.

Hallar el valor de las corrientes del siguiente circuito eléctrico:



Usando la ley de voltajes de Kirchhoff (LVK), hallamos la matriz de voltaje:

$$-E1+E2+(R1+R2)*I1-R2*I2-R1*I3=0$$
$$-E2-R2*I1+(R2+R3+R4)*I2-R3*I3=0$$
$$E3-R1*I1-R3*I2+(R1+R3+R5)*I3=0$$

Ordenando los productos:

$$\begin{array}{lllcl} (R1+R2)*I1 & -R2*I2 & *R1*I3 & = & E1-E2 \\ -R2*I1 & +(R2+R3+R4)*I'' & -R3*I3 & = & E2 \\ -R1*I1 & -R3*I2 & +(R1+R3+R5)*I3 & = & -E3 \end{array}$$

Suponiendo que los vales de las resistencias y las fuentes de voltaje son:

$$R1=4\Omega \quad R2=8\Omega \quad R3=7\Omega \quad R4=9\Omega \quad R5=2\Omega$$
$$E1=9V \quad E2=6V \quad E3=8V$$

Organizándola en la forma de Ax=B

$$\begin{bmatrix} 12 & -8 & -4 \\ -8 & 24 & 7 \\ 4 & -7 & 13 \end{bmatrix} \begin{pmatrix} I1 \\ I2 \\ I3 \end{pmatrix} = \begin{bmatrix} 3 \\ 6 \\ -8 \end{bmatrix}$$

Ahora nos queda simplemente, resolver este sistemas de ecuaciones lineales en Scilab para la cual vamos a declarar la matriz de las resistencias y la matriz de los voltajes, procedemos a resolverla y obtendremos los valores de las corrientes de nuestro circuito.

```
A=[12 -8 -4; -8 24 7;4 -7 13]
V=[3;6;-8]
I=inv(A)*V
I  =
!   0.4525611 !
!   0.5366705 !
! - 0.4656577 !
```

Solución:

$I1 = 0.4525611$

$I2 = 0.5366705$

$I3 = -0.4656577$

Operaciones con polinomios.

Algo interesante de Scilab, es la declaración de polinomios la cual es inversa a Matlab. Los polinomios pueden definirse de dos formas diferentes utilizando el comando poly():

a) Especificando las raíces del polinomio compuesto por p = 2 - 3x + x2 , las cuales son 1 y 2. La instrucción es la siguiente:

```
p = poly( [1 2 ], 'x')
 p  =
                2
   2 - 3x + x
```

Para hallar la raíz de un polinomio basta con utilizar el comando roots ( ) el cual nos devolverá las raíces del polinomio por ejemplo para obtener las raíces del polinomio anteriormente creado:

```
roots(p)
 ans  =
!   1. !
!   2. !
```

b)Otra forma de crear polinomios es a través de sus coeficientes en este caso también se usa el comando “poly”, en comillas simples la variable base del polinomio seguida de coma “,” en comillas simples la letra ‘c’ que significa coeficientes y se cierran paréntesis.

```
root(q)
ans = -0.5
```

```
q = poly( [1 2 ], 'x', 'c')
q = 1+ 2x
```

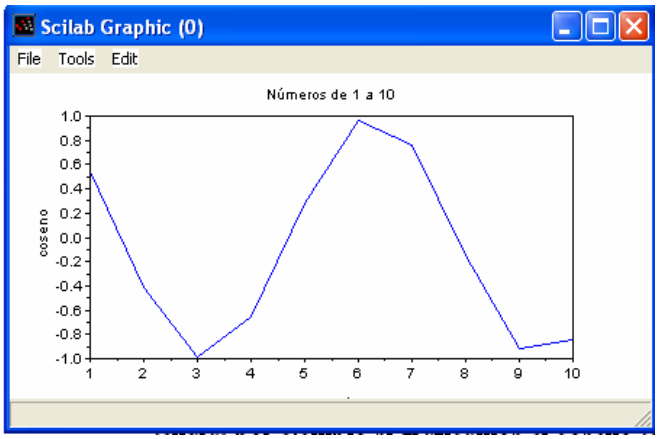
Las operaciones algebraicas elementales suma, resta, producto y cociente las realizamos con los operadores habituales p+q, p-q, p*q, p/q.

Graficando en Scilab.

La ventana de gráficas de Scilab consta de las siguientes herramientas:

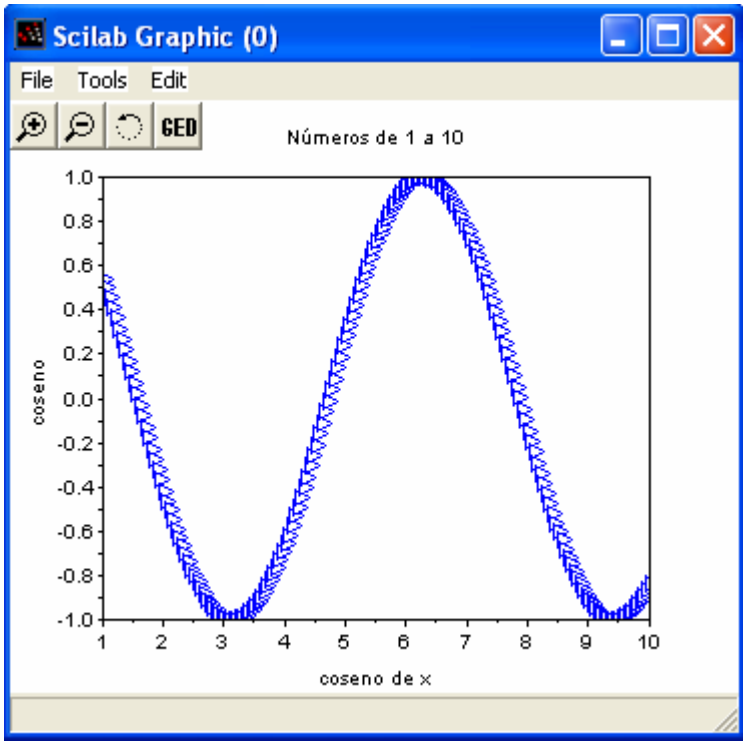
-3D Rot.: Permite aplicar rotaciones a los gráficos tridimensionales empleando el ratón.
-2D Zoom: Amplia un gráfico 2D. Este comando puede ser ejecutado recursivamente.
-UnZoom: Recupera el gráfico inicial.
-File: Abre un panel que permite imprimir el gráfico, exportarlo a un formato específico y salvarlo como gráfico Scilab para después recuperarlo.
El comando plot nos permite elaborar gráficos simples siguiendo la siguiente sintaxis.
Plot(x,y,"titulo del eje x","titulo del eje y", titulo grafico,
En donde x e y son vectores del mismo tamaño y la gráfica a realizar es y con respecto a x.
Cabe mencionar que una gráfica entre más puntos tenga mejor resolución se tendrá, por ejemplo si graficamos el coseno tendríamos:

```
clf;x=[1:1:10];plot(x, cos(x));xtitle("Números de 1 a 10","coseno de x","coseno")
```



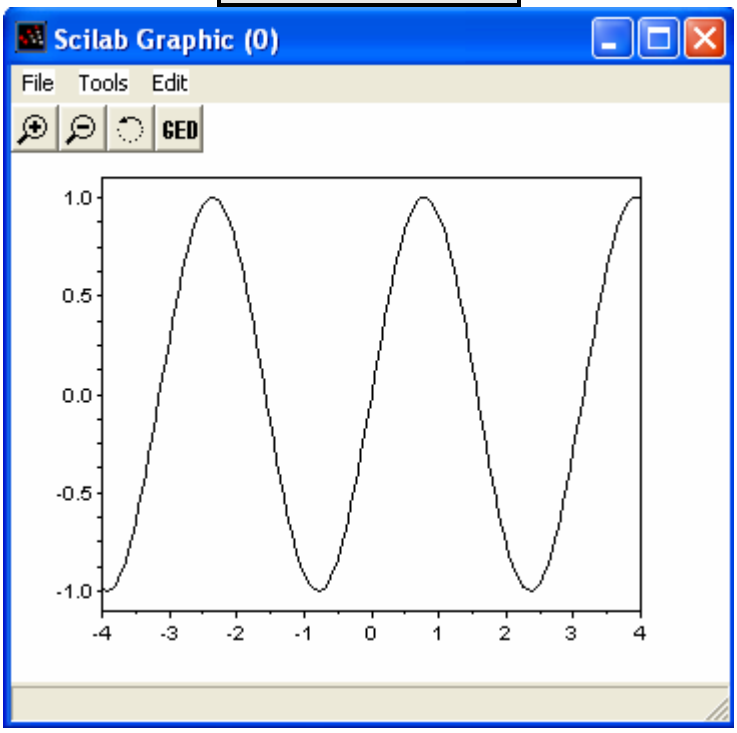
Y si ahora cambiamos el argumento a uno más pequeño de ésta manera

```
Clf
;x=[1:0.05:10];
plot(x, cos(x),'b.>');//la letra "b" indica color azul y el signo  los puntos de la grafica;
xtitle("Números de 1 a 10","coseno de x","coseno")
```



Otro comando que utilizar es el plot2d, el cual permite elaborar gráficos con mejores características de la siguiente manera:
Plot2d(variable independiente, variable dependiente), “argumentos”

```
Clf;
x=[-4:0.05:4];
 plot2d(x,sin(2*x))
```



Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Herramientas para Sistemas de Control en Scilab.

En el software libre Scilab contamos con un conjunto de herramientas para el diseño y análisis de sistemas de control. Podemos crear modelos de sistemas SISO (con una señal de entrada y una salida)[8]. Por ejemplo podemos calcular y representar gráficamente, respuestas en el tiempo, respuestas en frecuencia, lugar de las raíces, diagramas de Bode etc.

Respuesta Transitoria de la Función de Transferencia

Si deseamos realizar un análisis de un sistema, con una entrada, debemos de definir la función transferencia del sistema, esto es, la relación de las entradas al sistema y las salidas de este con condiciones nulas, en la transformada de Laplace.

$$H(s)=\frac{Y(s)}{R(s)}=\frac{b_o S^m+b_1 S^{(m-1)}+...b_m}{S^n+a_1 S^{(n-1)}+...a_{(n-1)}S+a_n}$$

Para la cual debemos definir primero el numerador Y(s) y el denominador U(s) como vectores, cuyos elementos serán son los coeficientes de los polinomios del numerador y del denominador en potencias decrecientes de la variable compleja “S”

Función de Transferencia

El primer paso para crear una función transferencia en Scilab, es conocer el comando “syslin”, cual nos permitirá convertir los polinomios del numerador y denominador en función de transferencia.

Por ejemplo vamos a crear la siguiente, función de transferencia en Scilab:

$$H(s)=\frac{Y(s)}{U(s)}=\frac{1}{S^2+0.25S+1}$$

Lo primero que debemos de realizar es declarar la variable compleja “s”, de la siguiente manera:

```
s=poly(0,'s')
 s  =
   s
```

En la ventana de comandos de Scilab, tecleamos las siguientes instrucciones:

```
num=poly([1],'s','c'),
den=poly([ 1 0.25 1  ],'s','c')
num  =
1
den  =
 2
1 + 0.25s + s
```

Nota: La ‘c’ nos indica que estamos declarando los coeficientes

Recordemos que Scilab trabaja los polinomios contrarios al modo de Matlab, declarándolos de derecha a izquierda. La función transferencia creada por medio del comando “syslin”, que define nuestro sistema lineal, quedándonos de la siguiente manera:

```
sistema=syslin('c',num,den)
 sistema  =
         1
   -------------
                  2
   1 + 0.25s + s
```

Nota: En este caso la 'c' nos indica que el sistema se ésta especificando en un dominio del tiempo continuo

Otro ejemplo sería calcular los ceros y los polos de la siguiente función de transferencia:

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

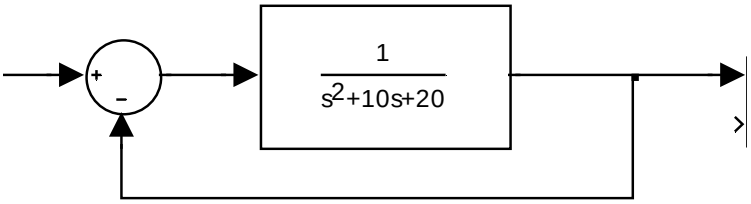
Para calcular las raíces del polinomio del denominador usamos la sintaxis:
roots(num) lo cual produce -5 como resultado. De manera análoga los polos del polinomio se evalúan usando la instrucción: roots(den)

```
num=poly([5 1],'s','c');
den=poly([ 1 3 2 1],'s','c');
sistema=syslin('c',num,den);
roots(num)
roots(den)
 ans  =
  - 5.
 ans  =
! - 0.4301597             !
! - 0.7849201 + 1.3071413i !
! - 0.7849201 - 1.3071413i !
```

Con lo cual podemos observar que tenemos un cero en -5, polos en -0.7849 ± 1.3071i y -0.4302

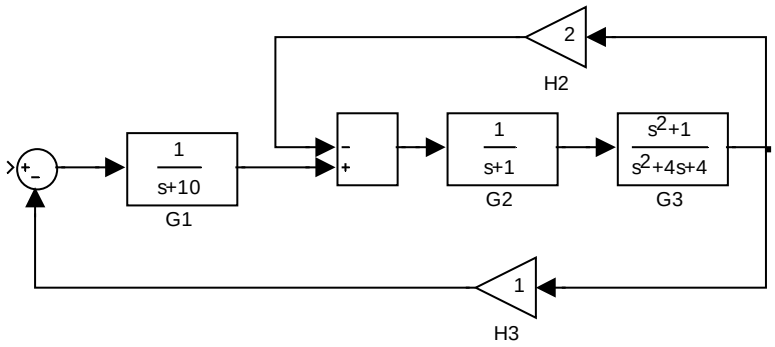
Retroalimentación:

En Scilab el manejo de los diagramas de bloques se realiza de la misma manera que en Matlab con los comandos “Feedback” y “Series”, En un sistema compensado, es decir, dentro de un lazo de retroalimentación negativa, podemos emplear en lugar de los comandos anteriores el operador “/.[-1]” que representa una retroalimentación negativa igual a uno y el operador “*” que representa el comando “Series” el cual multiplica dos o mas funciones de transferencia. Por ejemplo:

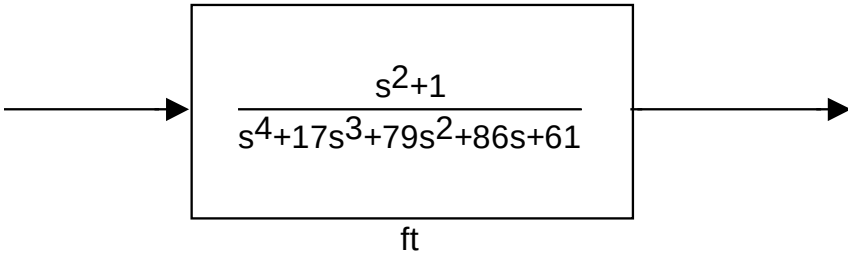



```
s=poly(0,'s');
num=poly([1],'s','c');
den=poly([20 10 1],'s','c');
g=syslin('c',num,den)
ft=g/.[1]
ft  =
          1
   -----------
              2
   21 + 10s + s
```

Un ejemplo más de reducción sería le siguiente diagrama, el cual reduciremos a una sola función de transferencia:



```
s=poly(0,'s');
G1=syslin('c',poly([1],'s','c'),poly([10 1],'s','c'));
G2=syslin('c',poly([1],'s','c'),poly([1 1],'s','c'));
G3=syslin('c',poly([1 0 1],'s','c'),poly([4 4 1],'s','c'));
H2=syslin('c',poly([2],'s','c'),poly([1],'s','c'));
H3=syslin('c',poly([1],'s','c'),poly([1],'s','c'));
w1=G2*G3;
w1=G2*G3 ;
w2=w1/.[H2];
w3=w2*G1;
tf=w3/.[1]
tf  =
                    2
                1 + s
 ---------------------------------
                    2     3     4
  61 + 86s + 79s + 17s + s
```

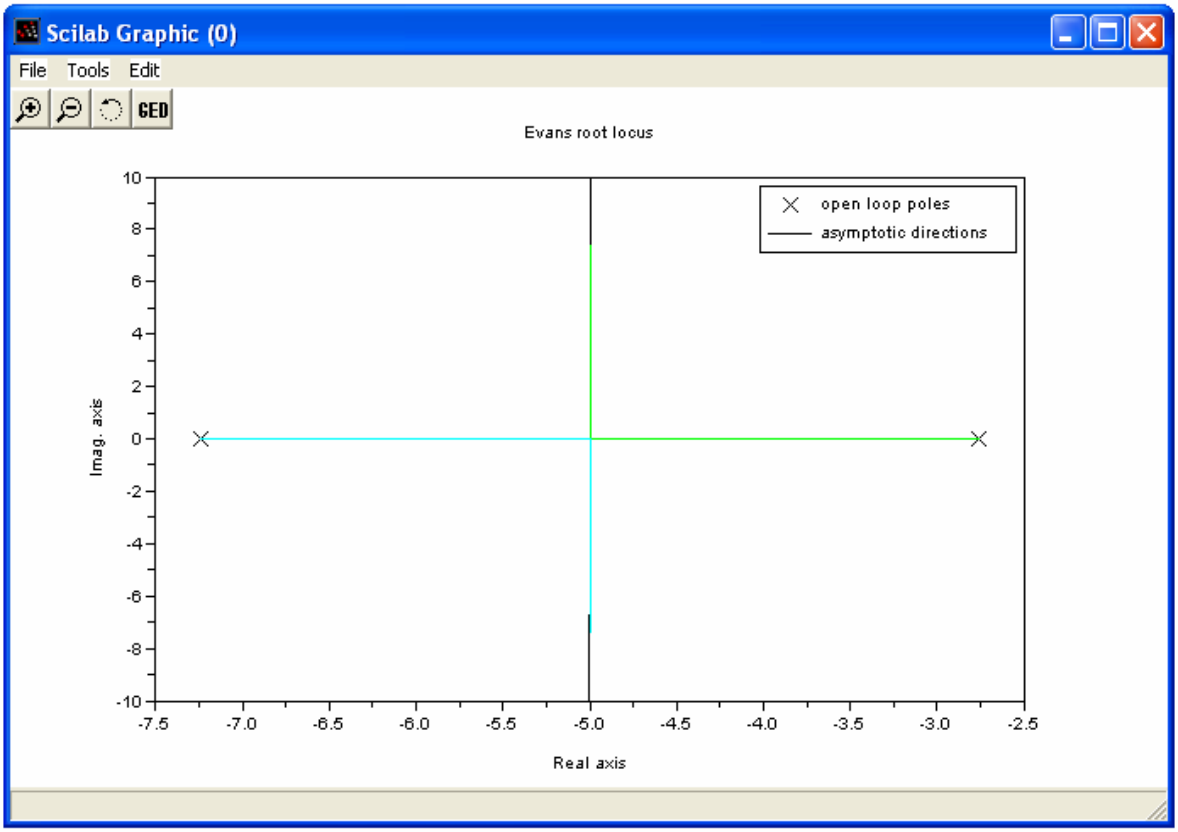


Lugar Geométrico de las Raíces

Para el lugar geométrico de las raíces, Scilab dispone del comando “evans”. El cual es análogo al comando “rlocus, la sintaxis para utilizar el comando “evans” es la siguiente: evans(nombre de la función transferencia, ganancia máxima que se desea). Por ejemplo:

$$D(s) = 1 + k\frac{1}{S^2 + 10S + 20}$$

```
s=poly(0,'s');
num=poly([1],’s’,’c’)
den=poly([20 10 1],’s’,’c’)
sistema5=syslin(‘c’,num,den)
instante=0:0.05:45;
evans(sistema5,60)
```



Donde podemos observar los polos ubicados en -7.25 ,0 y -3.125, 0 y el recorrido que tiene hasta -5,0 en donde se unen y se separan en ese mismo punto hasta finalizar su recorrido.

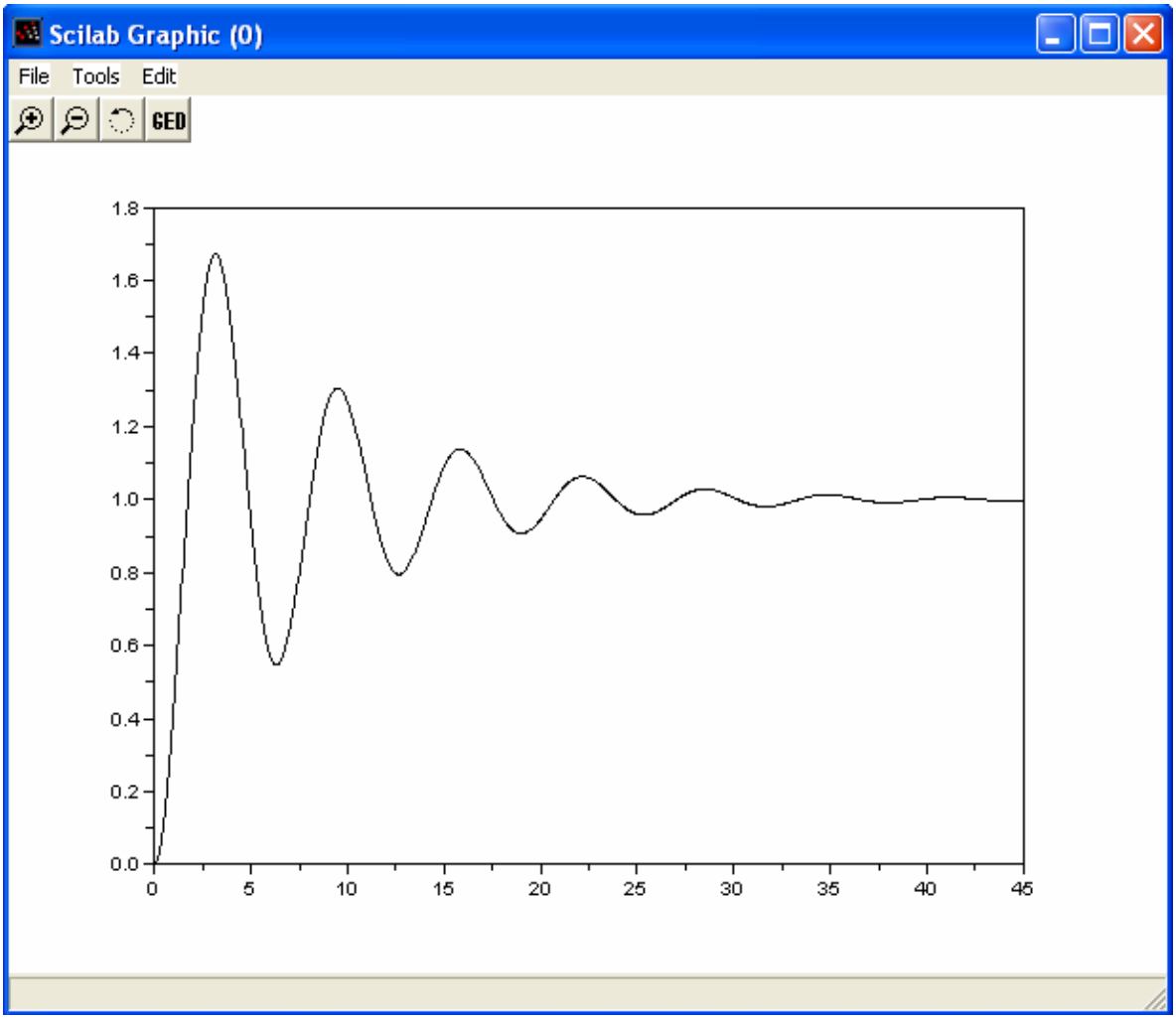
Entrada Escalón.

En Scilab el comando “step” no tiene el mismo procedimiento que en Matlab, en este comando debemos de definir el tiempo que deseamos graficar, definir la función de transferencia, para después con el comando”csim” dar una entrada escalón[9].

$$G(s) = \frac{1}{s^2 + 0.25 * s + 1}$$

```
s=poly(0,'s');
num=poly([1],’s’,’c’)
den=poly([1 0.25 1],’s’,’c’)
sistema6=syslin(‘c’,num,den)
instante=0:0.05:45;
y=csim('step',instante,sistema6);
plot2d(instante',y')
```

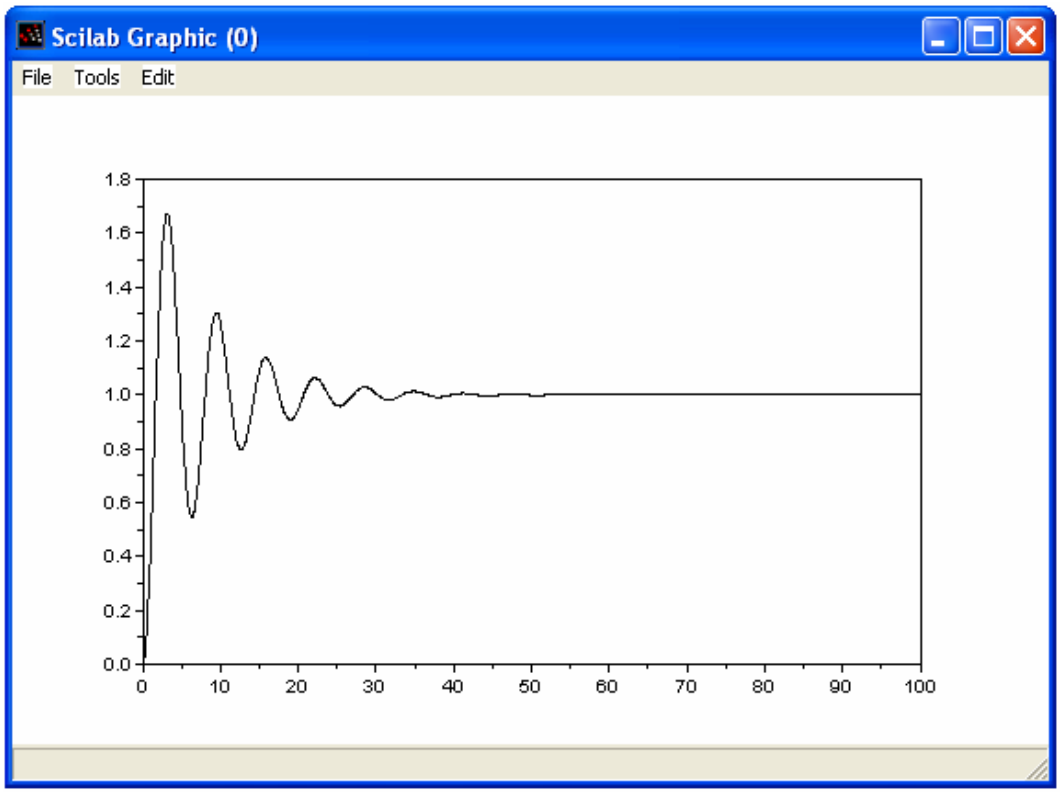
A continuación, Scilab despliega la respuesta para una entrada escalón para la función de transferencia:



Si deseamos tener una apreciación amplia de cómo se comporta nuestro sistema en una mejor vista, lo podemos hacer incrementando el valor final de "instante" en la sintaxis del comando la cual nos dará una ampliación en la gráfica extendiéndola según sean los segundos asignados;

```
instante=0:0.05:100;
y=csim('step',instante,sistema6);
plot2d(instante',y')
```

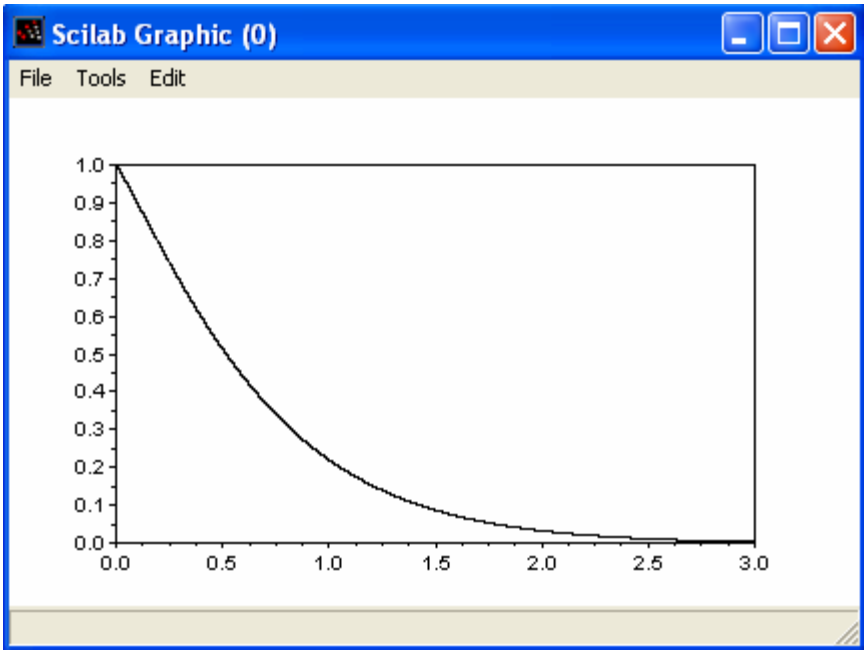
Nótese que se usamos ";" en la línea de "t" para evitar el eco de la pantalla.



Entrada Impulso.

Si deseamos obtener la respuesta en el tiempo para una entrada impulso la sintaxis es igual a la que anteriormente habíamos utilizado con el comando "step"[12]. Definimos en Scilab los polinomios del numerador y denominador de la función de transferencia, por ejemplo:

```
s=poly(0,'s');
sistema7=syslin('c',(s^2+5*s+4),(s**3+6*s**2+11*s+6))
 sistema7  =
                    2
      4 + 5s + s
    ----------------
                   2   3
    6 + 11s + 6s + s
instante=0:0.0001:3;
y=csim('imp',instante,sistema2);
plot2d(instante',y')
```



Respuesta en función de la frecuencia.

Si deseamos realizar un análisis en de la frecuencia podemos hacer uso de los diagramas de Bode.

Diagrama de Bode.

El comando Bode calcula las magnitudes y los ángulos de fase de la respuesta en frecuencia de sistemas continuos, lineales e invariantes en el tiempo. Los diagramas de Bode se utilizan frecuentemente para analizar y diseñar sistemas de control. Estos diagramas indican el margen de ganancia, al margen de fase.
Para obtener el diagrama de Bode de una función de transferencia, debemos de declarar dos vectores cuyos elementos son los coeficientes de los polinomios del numerador y del denominador en potencias decrecientes de S. Estos vectores son usados en el comando bode con la siguiente sintaxis: bode(num,den) o bode(función transferencia)

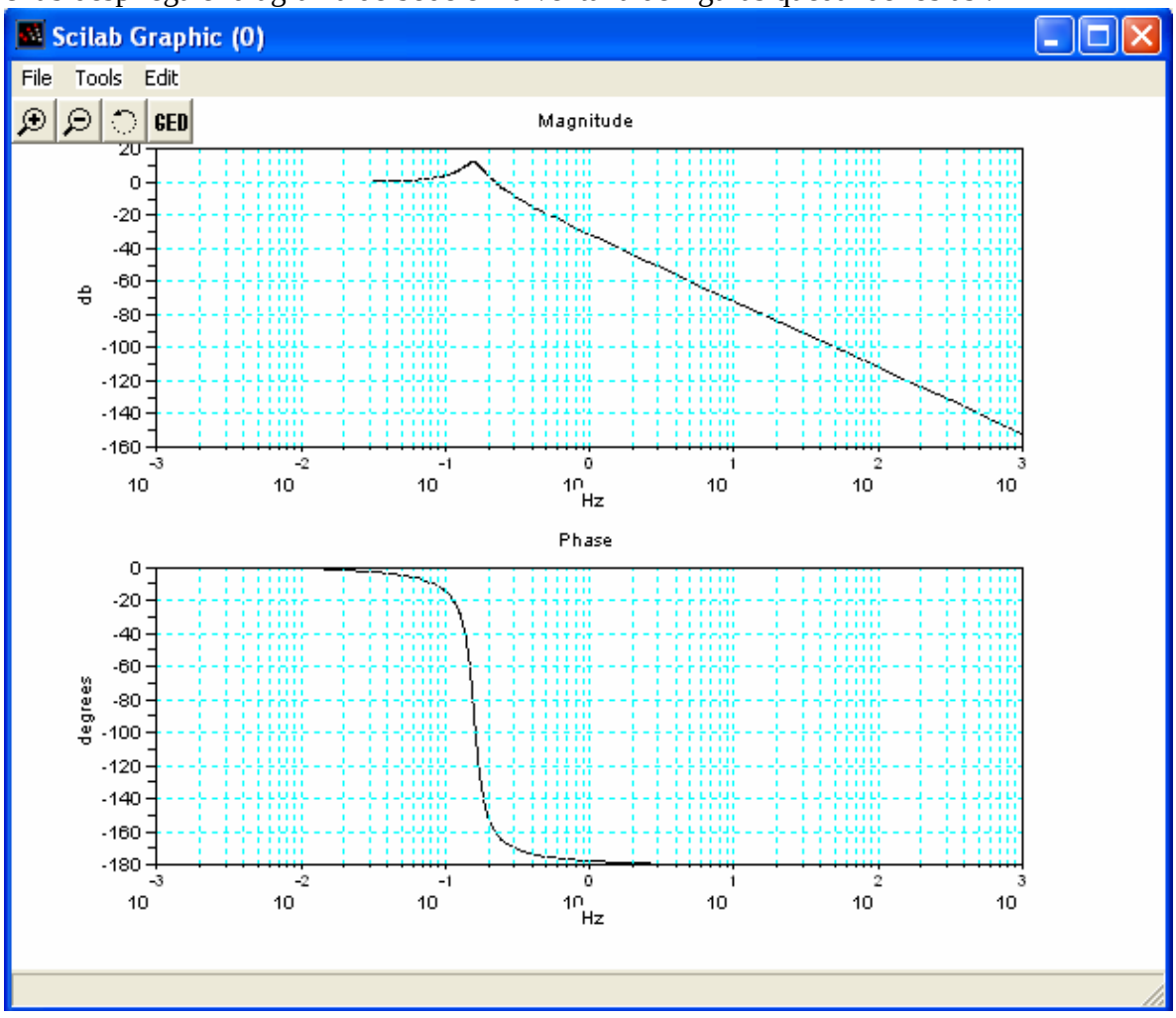
Para obtener el diagrama de Bode de una función de transferencia, debemos de declarar dos vectores cuyos elementos son los coeficientes de los polinomios del numerador y del denominador en potencias decrecientes de S. Estos vectores son usados en el comando bode con la siguiente sintaxis: bode (num, den). Definimos la función de transferencia:

$$H(s) = \frac{Y(s)}{U(s)} = \frac{1}{s^2 + 0.25s + 1.0}$$

Introducimos en el símbolo del Prompt., la siguiente expresión

```
sistema8=syslin('c',(1),(s**2+0.25*s+1))
 sistema8  =
         1
   -------------
                2
    1 + 0.25s + s
bode(sistema8);
```

Scilab despliega el diagrama de bode en la ventana de figuras quedándonos así:



Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Scicos

Scicos es una herramienta para el modelado y simulación de los sistemas de control del software Scilab desarrollado en INRIA (Intitute National de Recherche en Informatique et en Automatique), al igual que Scilab

Scicos (Connected Object Simulator) es un "Toolkit" de Scilab para modelar y simular sistemas dinámicos, tiene una familiaridad con los GUI (interfaz gráfica con el usuario) para editar modelos e interconectar los blocks que representan las funciones fundamentales predefinidas por Scilab, el generador de ondas, la ganancia, el osciloscopio(scope) etc. Incluye un editor grafico para la construcción de modelos mediante su interconexión, los blocks pueden ser encontrados en Scicos por medio de un menú de blocks (paletas).

Con la interfaz gráfica de usuario (GUI), podemos usar los diagramas de bloques para construir los modelos de los sistemas, los blocks pueden ser obtenidos desde varias paletas, además maneja un ambiente similar a Windows y se pueden grabar, cargar, salvar, compilar y simular usando las interfaces gráficas de Scicos.

Antes de tener nuestra primera experiencia con Scicos vamos a tomar en cuenta algunas sugerencias para realizar nuestros modelos:

1.-Dibujamos el modelo en un papel, lo analizamos y después lo construimos en Scicos: Para esto localizamos primero los bloques que vamos a utilizar y luego realizamos las conexiones, esto último reduce el tiempo de diseño a través de sucesivas aperturas de las bibliotecas de bloques.

2.-Utilicemos sistemas jerárquicos, siempre que sea posible procuremos estandarizar y realizar modelos sencillos de comprender, esto con el fin de que la persona que los interprete pueda realizar mejor su estudio y nos facilite nuestra tarea, los modelos complejos los podemos incorporar de jerarquías haciendo uso de los subsistemas o superblocks.

3.- Realicemos modelos claros, bien organizados y documentados que serán fáciles de entender. Para lo anterior, vamos a utilizar etiquetas para las señales y anotaciones sobre el modelo.

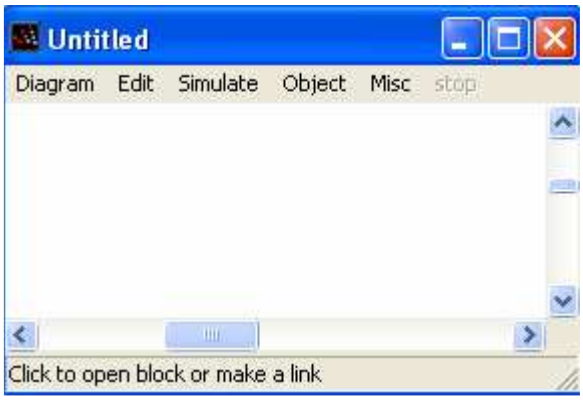
La construcción de un diagrama típico de simulación en Scicos, consiste de los siguientes pasos [13]:

Iniciar en la ventana principal de Scicos y que ésta se encuentre vacía sin ningún archivo de Scicos cargado.
Abrir una o varias paletas
Copiar los blocks desde las paletas para después pegarlos en nuestro diagrama
Cambiar el parámetro de los blocks de acuerdo a nuestro sistema.
Conectar las entradas y las salidas de los blocks.

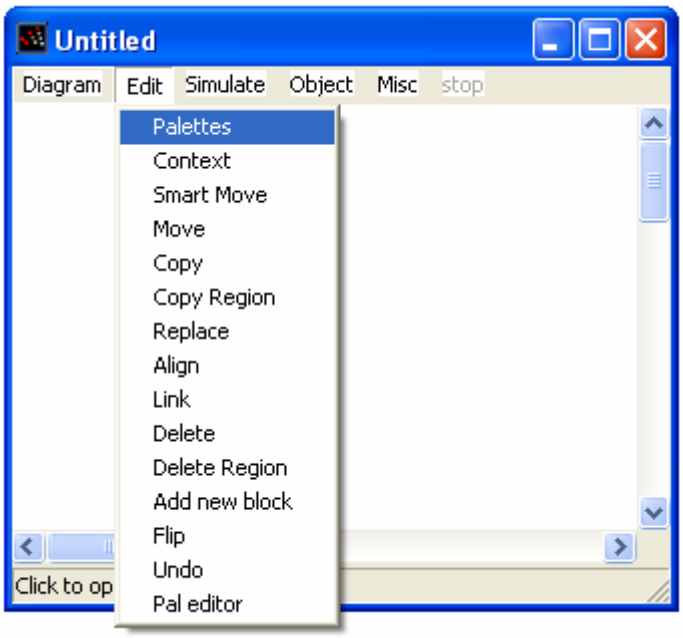
Compilar y Simular unl Diagrama.

Para empezar con Scicos desde la ventana principal de Scilab debemos escribir el comando Scicos; con lo que se nos abrirá la ventana del editor de Scicos:

En donde podremos construir nuestros diagramas para que sean modelados mediante la interconexión de bloques o subsistemas; los subsistemas son también llamados súper bloques (superblocks) que forman un único bloque partiendo de un conjunto de bloques simples de la misma manera que en Matlab.
Para abrir la ventana principal de Scicos debo teclear Scicos () en Scilab, Una vez ejecutado este comando se nos mostrara una ventana como la siguiente:



El primer paso para construir un nuevo modelo es abrir una o más paletas de bloques, usando el botón ‘Palettes’ que se encuentra dentro del menú ‘Edit’ se mostrara un cuadro de dialogo en donde están las paletas que se tienen para construir un diagrama de simulación



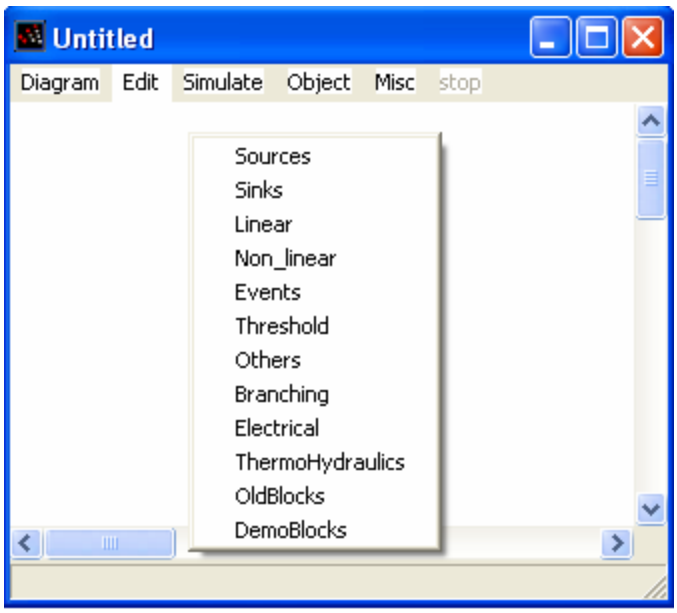
Las opciones hacen referencia a las distintas formas en como están agrupadas los blocks, las principales son:

Sources: Entradas o fuentes de señales

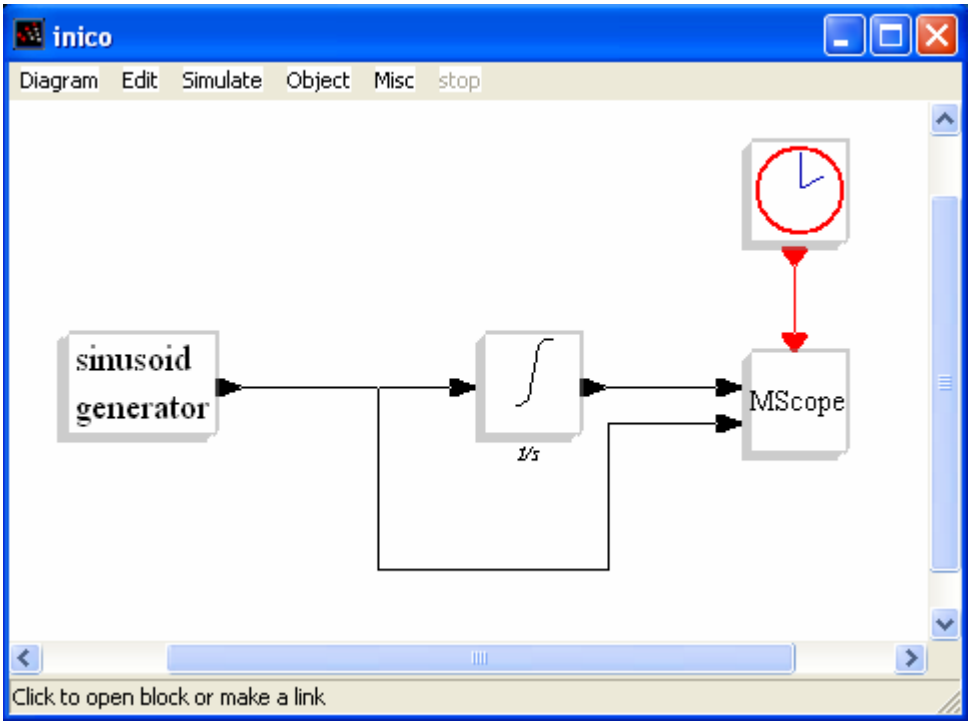
Sinks: Salidas o dispositivos de visualización/almacenamiento de variables Del sistema

Linear: Características de los sistemas lineales.

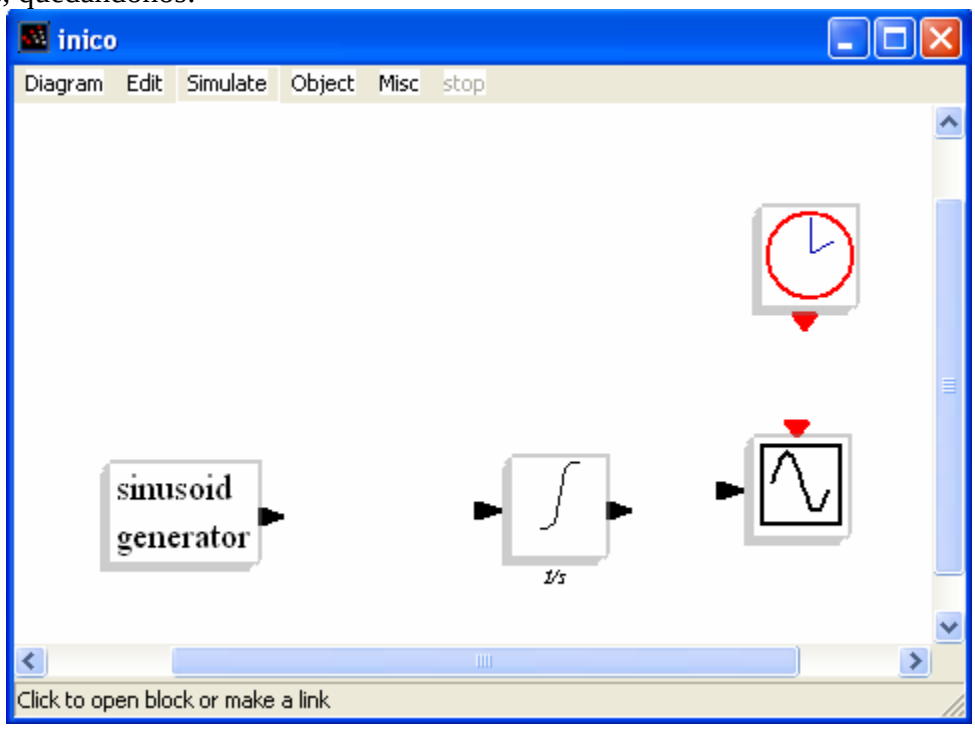
Eventos: Álgebra Booleana.



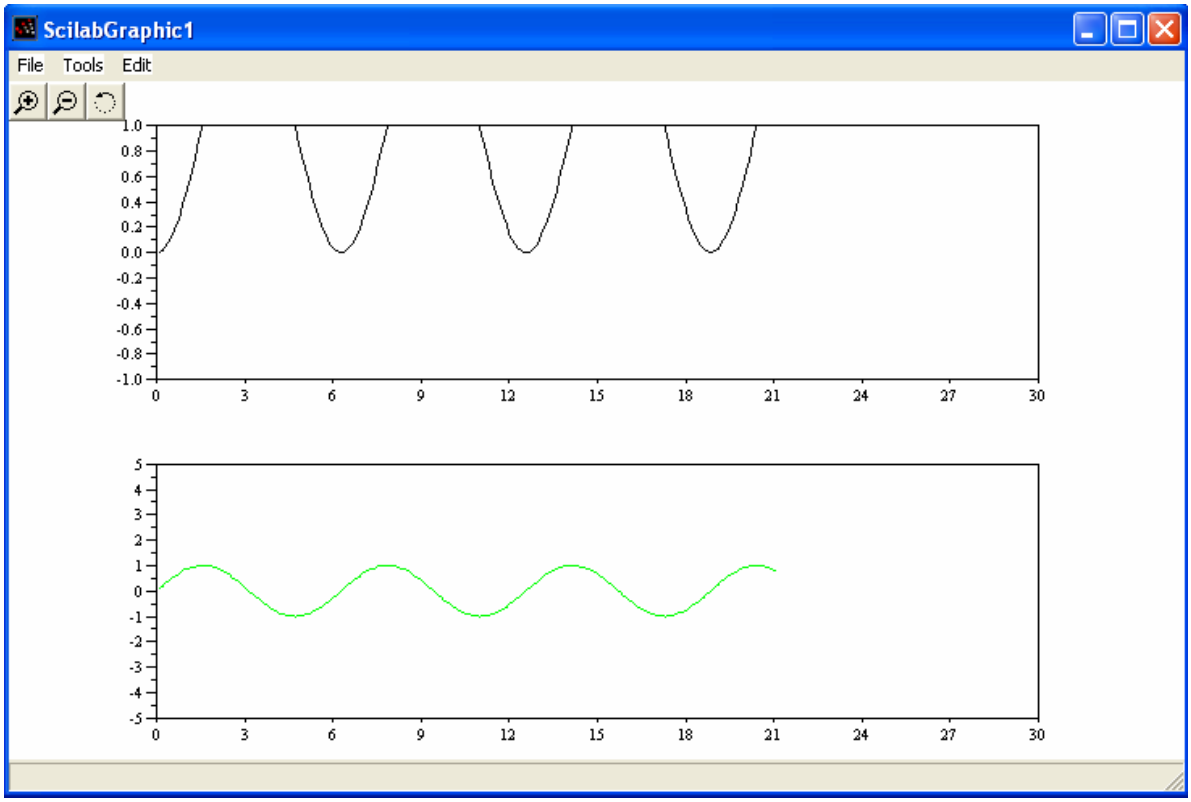
La copia de bloques en la ventana principal de Scicos desde las paletas, se realiza haciendo click en el botón copy del menú Edit. Vamos a construir un modelo como ejemplo, para copiar hacemos click con el botón izquierdo del ratón y arrastramos hasta la ventana principal de Scicos; volviendo a hacer click con el botón izquierdo.



Para esto el primer paso será abrir el menú Edit y seleccionar Palettes, como se hizo anteriormente, a continuación encontramos el block del generador de ondas (sinusoid generador) y el reloj de activación.
El block integrador lo encontramos en la paleta lineal, y el block “mscope” en la paleta oldblock. Los copiamos con el botón derecho del Mouse y después los pegamos en la ventana de Scicos, quedándonos:

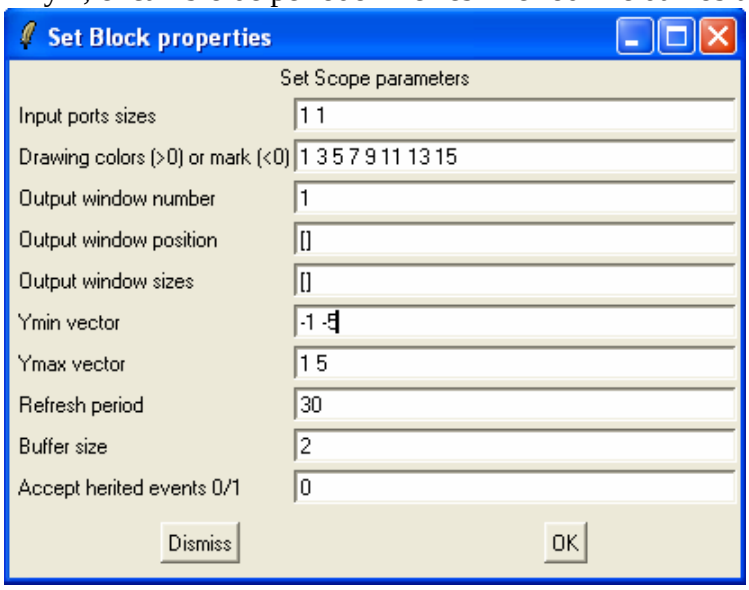


Procedemos a unir los blocks, conectando las entradas y salidas de los bloques, seleccionando primero el botón Link del menú Edit.
Una vez hecho click en el botón Link, hacemos click sobre la salida de un bloque y luego sobre la entrada del siguiente, quedándonos ambos conectados. También podemos partir o terminar en puntos intermedios en los enlaces, o sea hacemos click sobre el botón Link y pinchamos en un punto de un enlace a partir del cual podemos obtener las bifurcaciones que deseemos [12].
El proceso de enlazado puede ser detenido cuando deseemos simplemente haciendo click en el botón de derecho del ratón. El arreglo que nos deberá de quedar será el mostrado al principio, a continuación una vez completo el modelo podemos realizar la simulación usando Run del menú Simulate. Una vez seleccionado el botón Run se realiza la compilación del modelo (si no ha sido ya compilado) y después la simulación. La simulación puede ser detenida en cualquier momento simplemente haciendo click sobre el botón Stop en la ventana principal de Scicos.
Si la compilación resulta correcta, se simulara en muy poco tiempo, mostrando una gráfica como la siguiente:

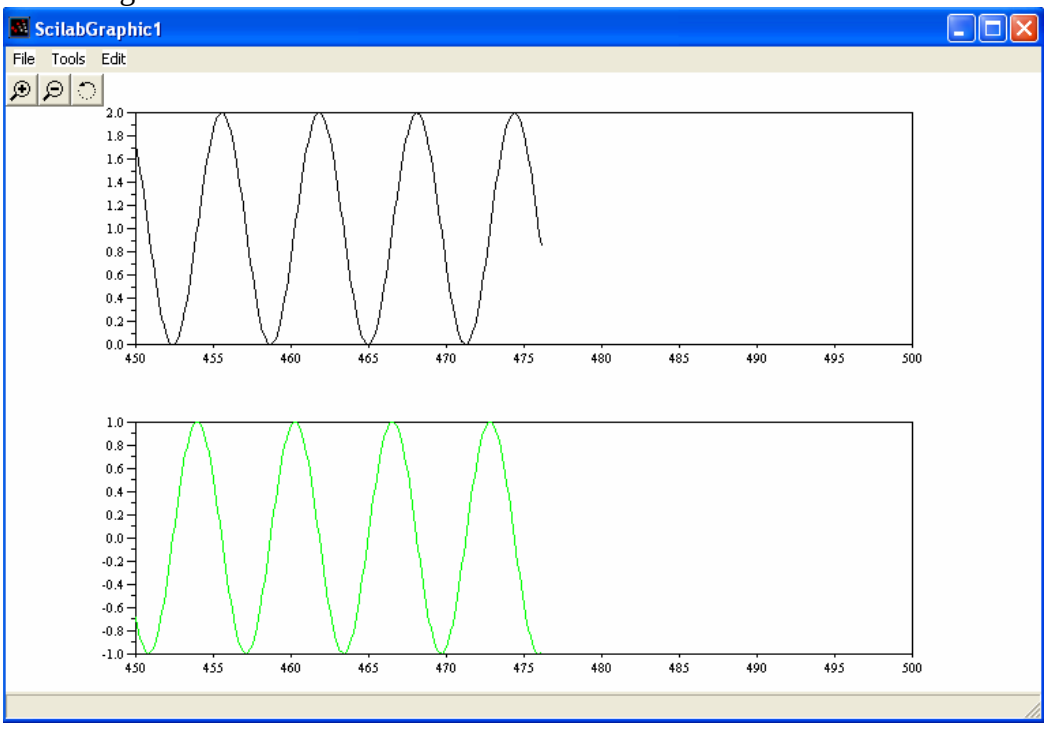


Como podemos ver en las gráficas anteriores hemos obtenido una simulación con los parámetros que incluían los bloques por defecto. Con una simple visualizaciones podemos observar que las escalas no están acordes con las señales, por lo que los parámetros de los bloques pueden ser modificados abriendo los menús de dialogo, simplemente seleccionando el botón Open/set del menú Object. Ahora vamos a modificar los parámetros de los bloques de nuestro ejemplo, con el único fin de obtener una salida gráfica en buenas condiciones.

Como decíamos la gráfica no ésta adecuada para los valores de la onda senoidal mostrada, dado que el eje "Y" solo llega hasta el valor de 1, esto lo podemos modificar haciendo doble clic en el block del MScope, y se nos mostrara una ventana como la siguiente donde modificaremos los valores mínimos de "Y", vamos a cambiar los de valores de 0 y -1, por valores de "Y" de 1 y 2, el cambio de periodo "Refresh Period" le damos un valor de 50.



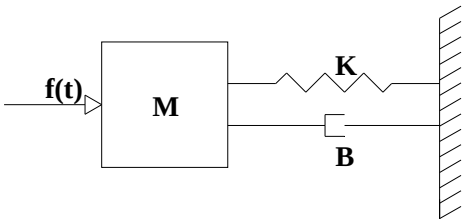
Ahora la nueva gráfica será mostrada como:



Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Simulación de un Sistema Descrito por una Ecuación diferencial.

En este caso vamos a representar un sistema físico formado por una masa M unida a un muelle de constante elástica K, y con un rozamiento viscoso B, representado como un amortiguador tal y como se describe en la figura:



El objetivo será ver cómo afecta la fuerza aplicada f (t) al movimiento de la masa, descrito por x (t). La ecuación diferencial que rige el comportamiento de este sistema es:

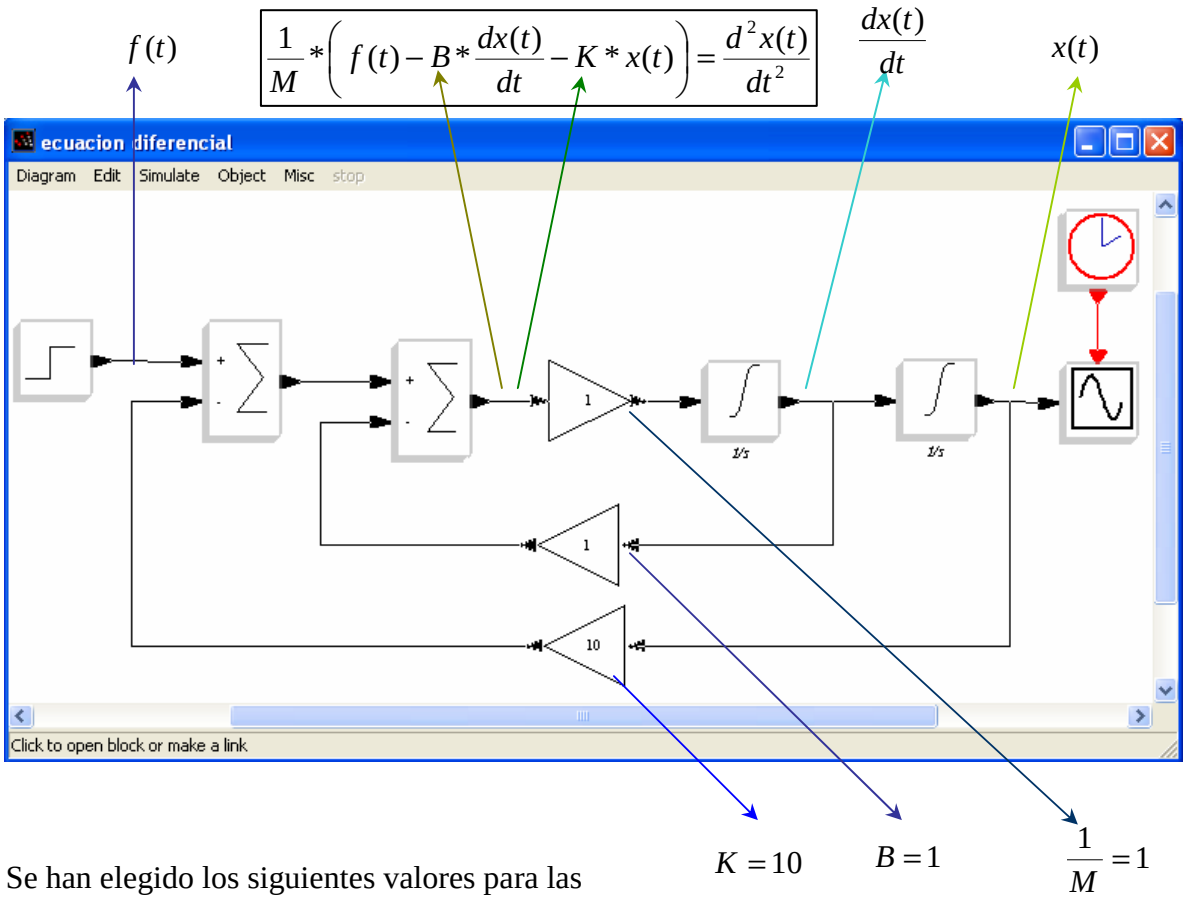
$$f(t) = M * \frac{d^2 x(t)}{dt^2} + B\frac{dx(t)}{dt} + K * x(t)$$

Para representar este sistema en Scicos, veremos a continuación unos ejemplos de utilización de los bloques integrador, sumador y multiplicador por una constante [11].

Bloque integrador: permite obtener a partir de d2x (t)/dt2 sus integrales d.C. (t)/d.C. y x (t):
Bloque sumador: permite sumar/restar señales
Bloque multiplicador o ganancia: permite multiplicar un bloque por una constante:

Una vez vistos esto, pasaremos a representar la ecuación diferencial propuesta. Para ello Despejaremos de la ecuación inicial el término d2x (t)/dt2, obteniendo:

$$\frac{d^2x(t)}{dt^2} = \frac{1}{M} * \left(f(t) - B * \frac{dx(t)}{dt} - K * x(t) \right)$$

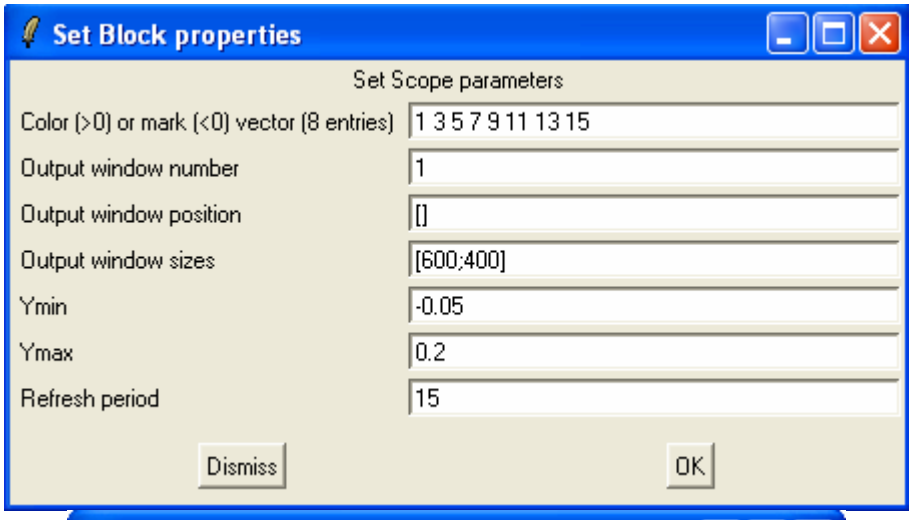


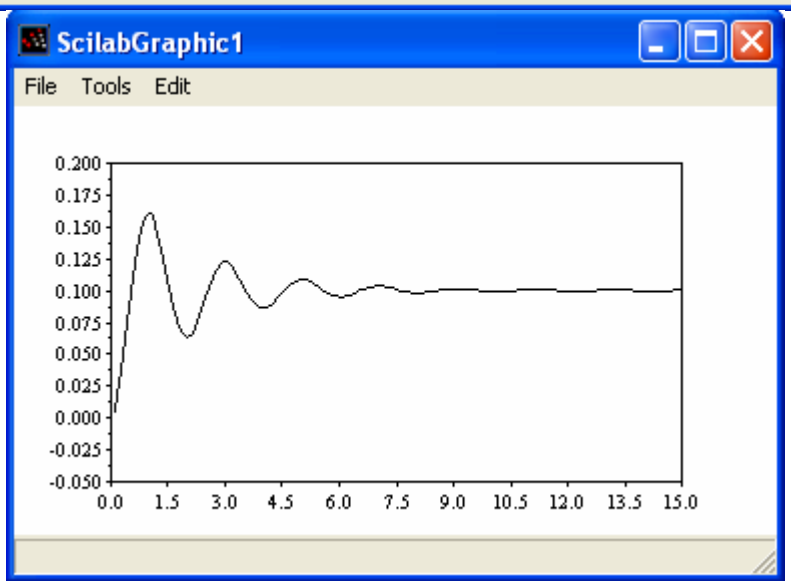
Se han elegido los siguientes valores para las constantes:

K=10	B=1
M=1	F(t)=Función escalón

Nota: en el diagrama que se muestra alguno de los bloques se encuentran girados para que sea más fácil realizar las conexiones.

Una vez comprendido el funcionamiento del esquema propuesto, lo introduciremos en Scicos y observaremos cuál es el resultado obtenido en el elemento 'Scope'. Lo que estamos viendo es el efecto que una fuerza en forma de escalón (señal de entrada) produce sobre la posición del conjunto (señal de salida). Al tratarse de un conjunto muelle-amortiguador, el movimiento de la masa debe presentar oscilaciones que poco a poco deben ir desapareciendo. El aspecto que debe tener la salida debe ser el que se muestra en la figura siguiente (será necesario hacer el zoom para ver correctamente la señal).





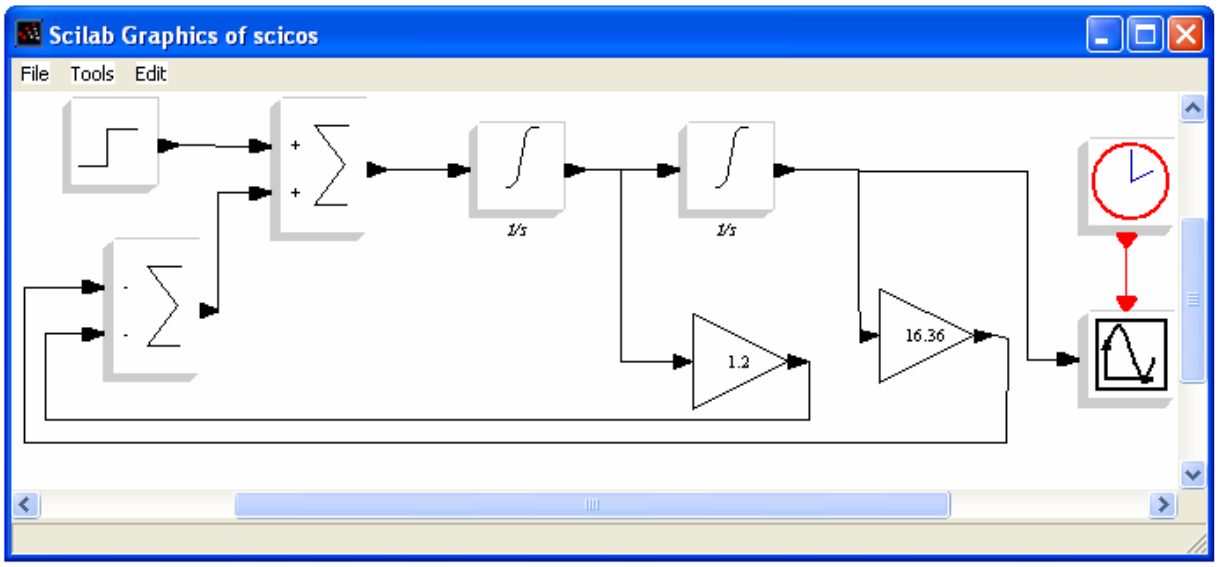
Una vez que hemos creado el esquema que permite simular un determinado sistema físico, podemos comprobar hasta qué punto es útil disponer de una simulación para comprobar el funcionamiento de cualquier sistema.
A continuación vamos a modelar la siguiente ecuación diferencial de segundo orden con coeficientes constantes.

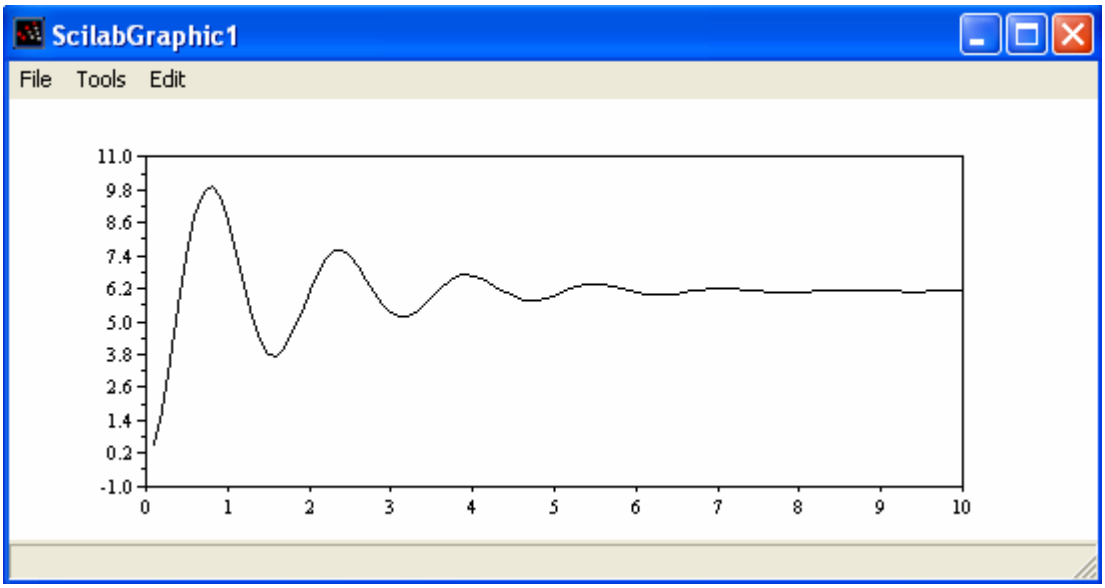
$$y''+1.2y'+16.36y=100$$

La modificamos para poder conectar los bloques de acuerdo a las ecuaciones, para ello hacemos:

$$y''= 100-1.2y'-16.36y$$

Observamos que hay tres sumados en el segundo miembro de la última ecuación, por lo tanto conectamos tres entradas al sumador compuesto, rotulando cada entrada con el sumando correspondiente del segundo miembro de la ecuación, la salida y'' se aplica al integrador obteniéndose y',si se integra nuevamente se obtiene y ,la variable y' se la multiplica por 1.2 y se aplica a una entrada del sumador el cual modifica su signo y a la variable y se la multiplica por 16.36 y se la aplica al sumador,





Subsistemas

La creación de subsistemas puede simplificar bastante nuestros diagramas de bloques, ya que podemos agrupar varios bloques como si fueran uno solo. Para acceder a los elementos que forman el subsistema basta con hacer doble click con el ratón sobre el bloque subsistema[9]. La forma en que podemos hacer subsistemas es a través del menú 'Diagram' de la ventana principal de Scicos y a continuación elegimos la opción de 'Región to Super block' en donde seleccionaremos la parte de nuestro diagrama que deseemos se convierta en un súper block.

Ejemplo 1

Respuesta transitoria a una entrada escalón

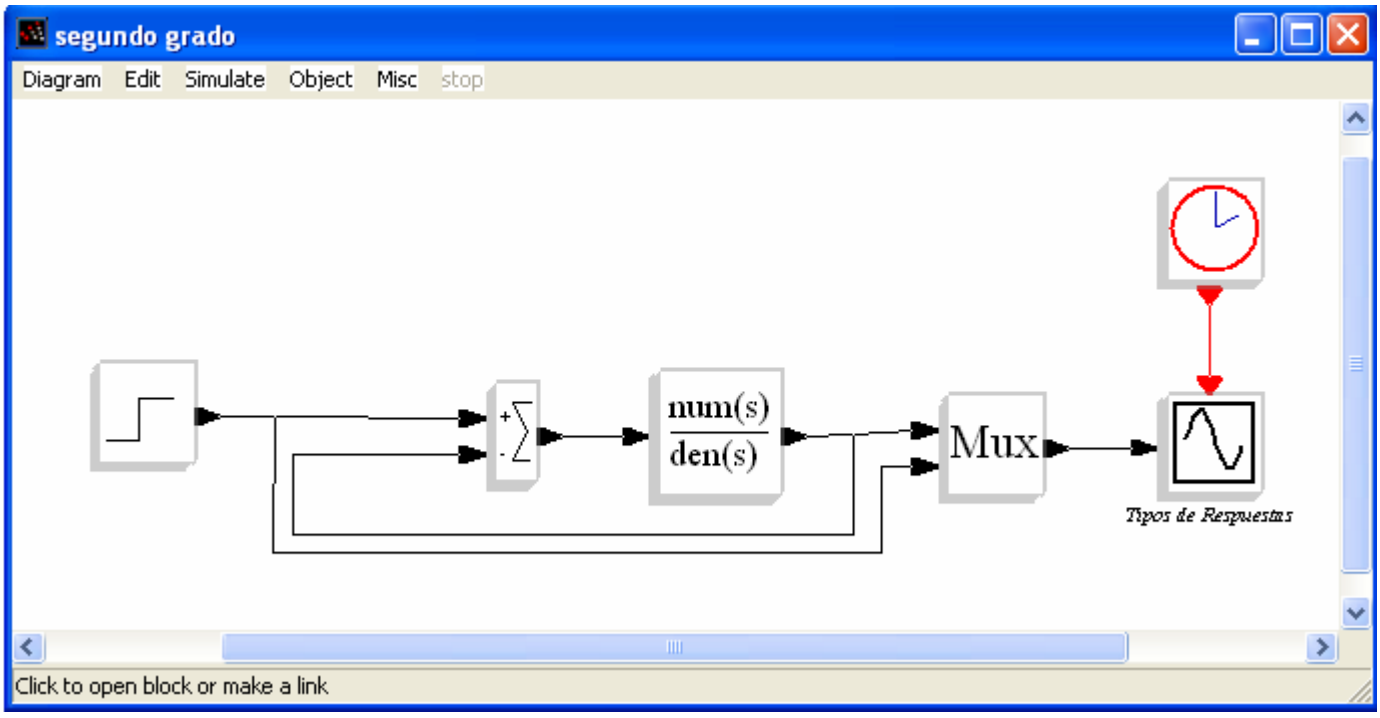
Si deseamos visualizar la repuesta del sistema G(s) ante una entrada escalón, del siguiente sistema:

$$G(s) = \frac{25}{s^2 + 4s + 25}$$

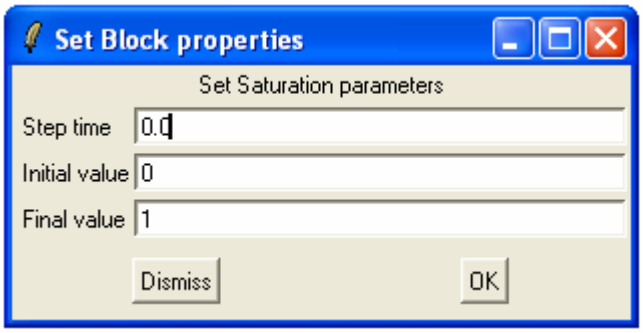
Para ello, los bloques los obtendremos de las siguientes librerías o paletas:

BLOQUE	PALETAS
Step	Sources
Función de Transferencia	Linear
Mux	OldBlocks
Set clock	Sources
Sumador	OldBlocks
Scope	OldBlocks

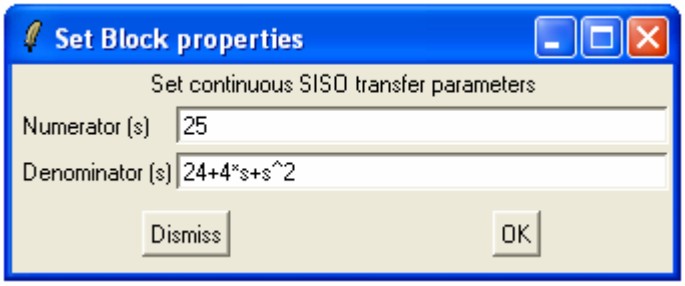
El modelo que construiremos con Scilab es el siguiente:



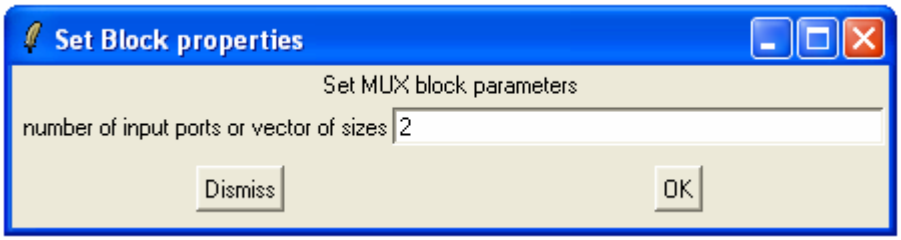
El valor de los bloques lo definiremos como sigue:
En el bloque Step cambiaremos el valor inicial de uno a cero y el valor final lo definiremos en uno.



En el bloque de la Función de Transferencia lo definiremos como numerador 25, y como denominador 24+4*s+s^2, quedándonos así:

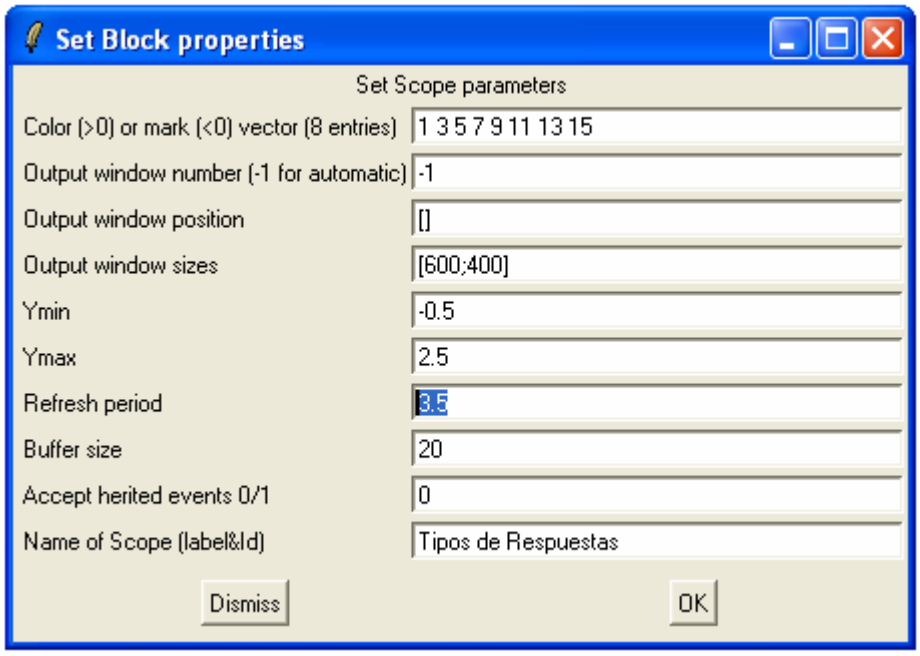


En el bloque Mux ptomaremos el número de entradas a 2

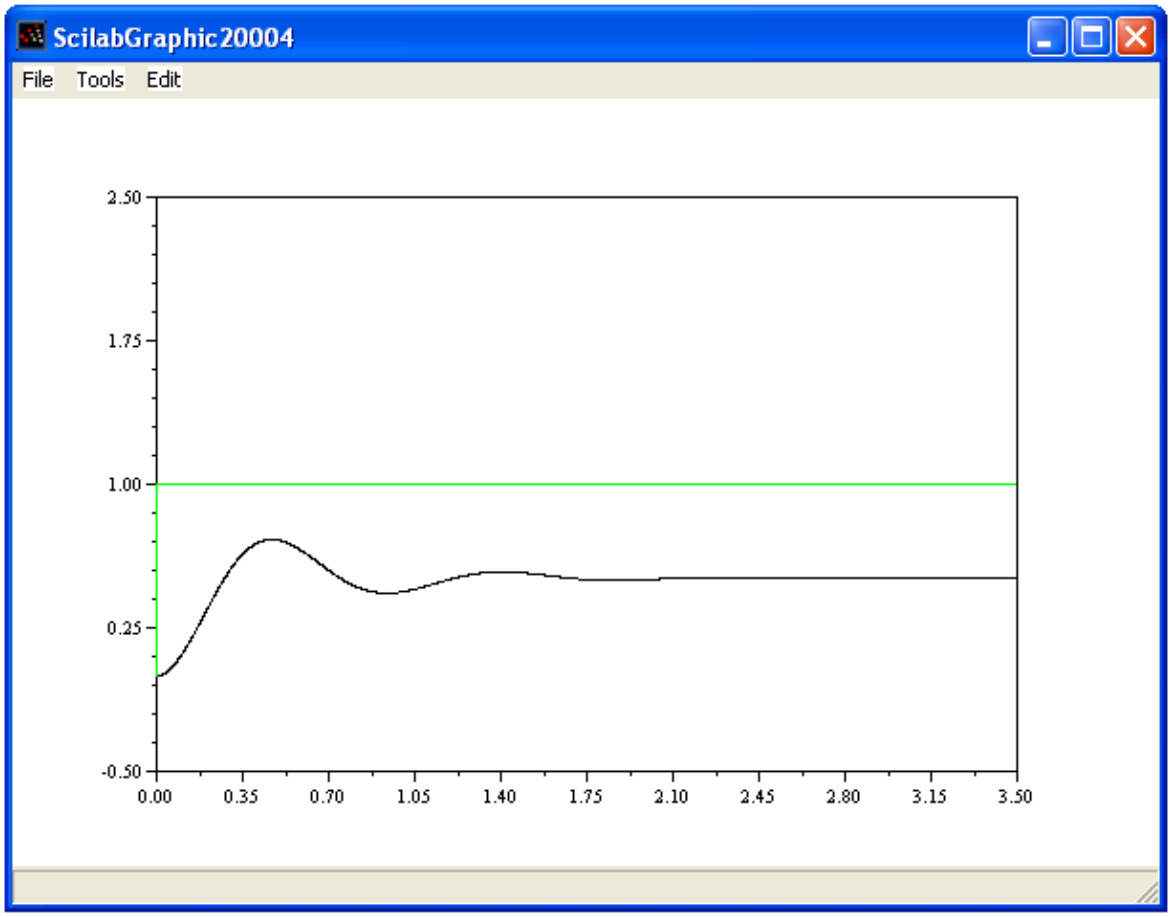


Respuesta del sistema

Para ver la respuesta del sistema a la entrada en escalón comenzamos la simulación, con la opción “Simulate”. Antes ajustamos el tiempo de simulación. Un valor pequeño de tiempo será suficiente para ver la respuesta (3.5 sg)



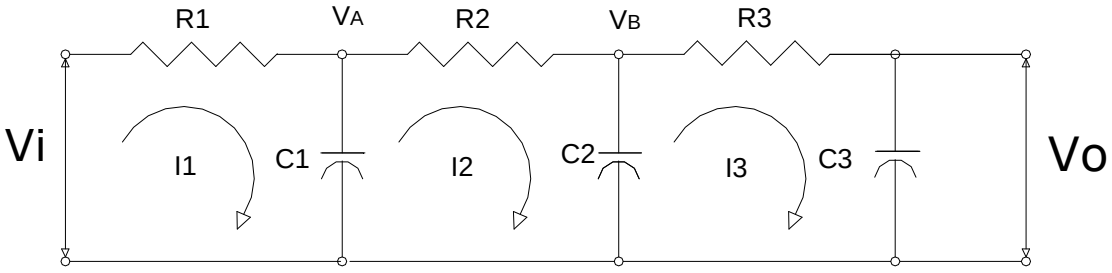
Ahora comenzamos la simulación:



Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Caso de Estudio resuelto con Matlab.

Circuito Eléctrico:



Para el circuito anterior, tomemos los valores para modelo matemático de resistencias y las capacitancías, como R_1, R_2, R_3 = R = 1 mΩ C_1, C_2, C_3 = C = 100 nF

Obteniendo el Modelo Matemático:

$$Z_{numero\_de\_capacitor} = \frac{1}{SC} \qquad Z_{resistencia} = R$$

$Z_1 = R_1$	$Z_2 = C_1$	$Z_3 = R_2$	$Z_4 = C_2$	$Z_5 = R_3$	$Z_6 = C_3$

$iR_1 = iC_1 + iR_2$, $iR_2 = iC_2 + iR_3$, $iR_3 = iC_3$

$$\frac{v_i - v_A}{Z_1} = \frac{v_A}{Z_2} + \frac{v_A - v_B}{Z_3}$$

$$v_i = \frac{z_1}{z_2}v_A + \frac{z_1}{z_3}v_A - \frac{z_1}{z_3}v_B + v_A \quad \text{.....(1)}$$

$$\frac{v_A - v_B}{Z_3} = \frac{v_B - v_C}{Z_5} + \frac{v_B}{Z_4}$$

$$v_A = \frac{z_3}{z_5}v_B - \frac{z_3}{z_5}v_C + \frac{z_3}{z_4}v_B + v_B \quad \text{.....(2)}$$

$$\frac{v_B - v_C}{Z_5} = \frac{v_C}{Z_6}$$

$$v_B = \frac{z_5}{z_6}v_C + v_C \quad \text{.....(3)}$$

Sustituyendo (3) en (2):

$$v_A = \frac{Z_3}{Z_5}\left[\frac{Z_5}{Z_6}v_C + v_C\right] - \frac{Z_3}{Z_5}v_C + \frac{Z_3}{Z_4}\left[\frac{Z_5}{Z_6}v_C + v_C\right] + \left[\frac{Z_5}{Z_6}v_C + v_C\right]$$

$$v_A = \frac{Z_3}{Z_6}v_C + \frac{Z_3}{Z_4}\frac{Z_5}{Z_6}v_C + \frac{Z_3}{Z_4}v_C + \frac{Z_5}{Z_6}v_C + v_C \quad \text{.....(4)}$$

Sustituyendo (4) en (1):

$$v_i = \frac{Z_1}{Z_3}\left[\frac{Z_3}{Z_6}v_C + \frac{Z_3}{Z_4}\frac{Z_5}{Z_6}v_C + \frac{Z_3}{Z_4}v_C + \frac{Z_5}{Z_6}v_C + v_C\right] - \frac{Z_1}{Z_3}\left[\frac{Z_5}{Z_6}v_C + v_C\right]$$

$$+ \frac{Z_1}{Z_2}\left[\frac{Z_3}{Z_6}v_C + \frac{Z_3}{Z_4}\frac{Z_5}{Z_6}v_C + \frac{Z_3}{Z_4}v_C + \frac{Z_5}{Z_6}v_C + v_C\right]$$

$$+ \left[\frac{Z_3}{Z_6}v_C + \frac{Z_3}{Z_4}\frac{Z_5}{Z_6}v_C + \frac{Z_3}{Z_4}v_C + \frac{Z_5}{Z_6}v_C + v_C\right]$$

$$v_i = \frac{Z_1}{Z_6}v_C + \frac{Z_1}{Z_4}\frac{Z_5}{Z_6}v_C + \frac{Z_1}{Z_4}v_C + \frac{Z_3}{Z_6}\frac{Z_1}{Z_2}v_C + \frac{Z_3}{Z_4}\frac{Z_5}{Z_6}\frac{Z_1}{Z_2}v_C$$

$$+ \frac{Z_3}{Z_4}\frac{Z_1}{Z_2}v_C + \frac{Z_5}{Z_6}\frac{Z_1}{Z_2}v_C + \frac{Z_1}{Z_2}v_C + \frac{Z_3}{Z_6}v_C + \frac{Z_3}{Z_4}\frac{Z_5}{Z_6}v_C$$

$$+ \frac{Z_3}{Z_4}v_C + \frac{Z_5}{Z_6}v_C + v_C$$

Obteniendo un factor común:

$$v_i = \left[\frac{Z_1}{Z_6} + \frac{Z_1}{Z_4}\frac{Z_5}{Z_6} + \frac{Z_1}{Z_4} + \frac{Z_3}{Z_6}\frac{Z_1}{Z_2} + \frac{Z_3}{Z_4}\frac{Z_5}{Z_6}\frac{Z_1}{Z_2} + \frac{Z_3}{Z_4}\frac{Z_1}{Z_2} + \frac{Z_5}{Z_6}\frac{Z_1}{Z_2}\right.$$

$$\left. + \frac{Z_1}{Z_2} + \frac{Z_3}{Z_6} + \frac{Z_3}{Z_4}\frac{Z_5}{Z_6} + \frac{Z_3}{Z_4} + \frac{Z_5}{Z_6} + 1\right]v_C \quad \text{.....(5)}$$

A continuación sustituimos los valores de las resistencias y los capacitores con valores que nos permitan obtener una mejor apreciación, los valores de las resistencias y capacitancías serán:

1.- Valores de Resistencias y Capacitancías.

R_1	R_2	R_3	Rn = 1 MΩ
C_1	C_2	C_3	$C_{n,}$ = 100 nF

Ahora sustituimos los valores de las resistencias, pasados al dominio de la variable compleja, para ser tratados como operaciones algebraicas, quedándonos:

$$\frac{Z_1}{Z_6} = \frac{R_1}{1/SC_3} = SC_3R_1 = (1m\Omega)(100nF)S = 0.1S$$

$$\frac{Z_1}{Z_4}\frac{Z_5}{Z_6} = \frac{R_1R_3}{1/S^2C_2C_3} = S^2R_1R_3C_2C_3 = (1m\Omega)^2(100nF)^2S^2 = 0.01S^2$$

$$\frac{Z_1}{Z_4} = \frac{R_1}{1/SC_2} = SC_2R_1 = (1m\Omega)(100nF)S = 0.1S$$

$$\frac{Z_3Z_1}{Z_6Z_2} = \frac{R_1R_2}{1/S^2C_1C_3} = S^2R_1R_2C_1C_3 = (1m\Omega)^2(100nF)^2 = 0.01S^2$$

$$\frac{Z_3Z_5Z_1}{Z_4Z_6Z_2} = \frac{R_1R_2R_3}{1/S^3C_1C_2C_3} = S^3R_1R_2R_3C_1C_2C_3 = (1m\Omega)^3(100nF)^3S^2 = 0.001S^3$$

$$\frac{Z_3Z_1}{Z_4Z_2} = \frac{R_1R_2}{1/S^2C_1C_3} = S^2R_1R_2C_1C_3 = (1m\Omega)^2(100nF)^2S^2 = 0.01S^2$$

$$\frac{Z_5Z_1}{Z_6Z_2} = \frac{R_1R_3}{1/S^2C_1C_3} = S^2R_1R_3C_1C_3 = (1m\Omega)^2(100nF)^2S^2 = 0.01S^2$$

$$\frac{Z_1}{Z_2} = \frac{R_1}{1/SC_1} = SC_1R_1 = (1m\Omega)(100nF)S = 0.1S$$

$$\frac{Z_5}{Z_6} = \frac{R_3}{1/SC_3} = SC_3R_3 = (1m\Omega)(100nF)S = 0.1S$$

$$\frac{Z_3}{Z_6} = \frac{R_2}{1/SC_3} = SC_3R_2 = (1m\Omega)(100nF)S = 0.1S$$

$$\frac{Z_3Z_5}{Z_4Z_6} = \frac{R_2R_3}{1/S^2C_2C_3} = S^2R_2R_3C_2C_3 = (1m\Omega)^2(100nF)^2S^2 = 0.01S^2$$

$$\frac{Z_3}{Z_4} = \frac{R_2}{1/SC_2} = SC_2R_2 = (1m\Omega)(100nF)S = 0.1S$$

$$\frac{Z_5}{Z_6} = \frac{R_3}{1/SC_3} = SC_3R_3 = (1m\Omega)(100nF)S = 0.1S$$

Ordenando los coeficientes:

$$V_i = [0.1S + 0.01S^2 + 0.1S + 0.01S^2 + 0.001S^3 + 0.01S^2 + 0.01S^2 + 0.1S + 0.1S + 0.01S^2 + 0.1S + 0.1S + 1]V_C$$

$$V_i = [0.001S^3 + 0.05S^2 + 0.6S + 1]V_C$$

Obtenemos la siguiente función de transferencia:

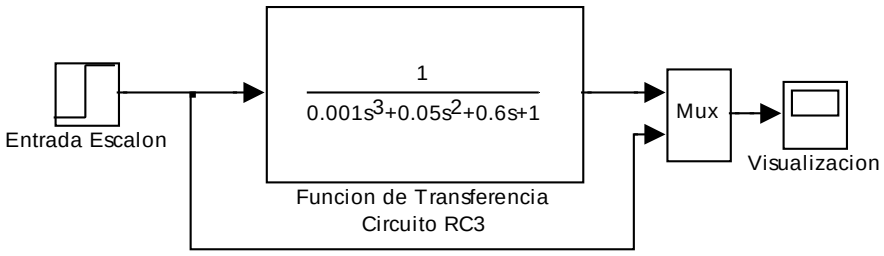
$$\frac{V_o(s)}{V_i(s)} = \frac{1}{0.001S^3 + 0.05S^2 + 0.6S + 1}$$

Todas las operaciones algebraicas y aritméticas realizadas anteriormente las podemos realizar en Matlab declarando los valores de los parámetros del circuito e introduciendo la ecuación (4), quedándonos de la siguiente manera:

```
%Este es un script en Matlab
s=sym('s');
      z1=10^6; %Valores de las Resistencias;
      z3=10^6; %Valores de las Resistencias;
      z5=10^6;%Valores de las Resistencias;
      z2=(1/(s*100*10^(-9)));%Valores del Capacitor;
      z4=1/((s*100*10^(-9)));%Valores del Capacitor;
      z6=(1/(s*100*10^(-9)));%Valores del Capacitor;
%Introducimos la ecuación para ser reducida a su expresion mas simple;
     ft=[(z1/z6)+((z1*z5)/(z4*z6))+(z1/z4)+((z3*z1)/(z6*z2))+...
     ((z3*z5*z1)/(z4*z6*z2))+((z3*z1)/(z4*z2))+((z5*z1)/(z6*z2))+...
     (z1/z2)+(z3/z6)+((z3*z5)/(z4*z6))+(z3/z4)+(z5/z6)+1]^(-1);
pretty(ft)
                              1
                  -------------------------------
                                2           3
                  3/5 s + 1/20 s  + 1/1000 s  + 1
```

Análisis en el dominio del tiempo

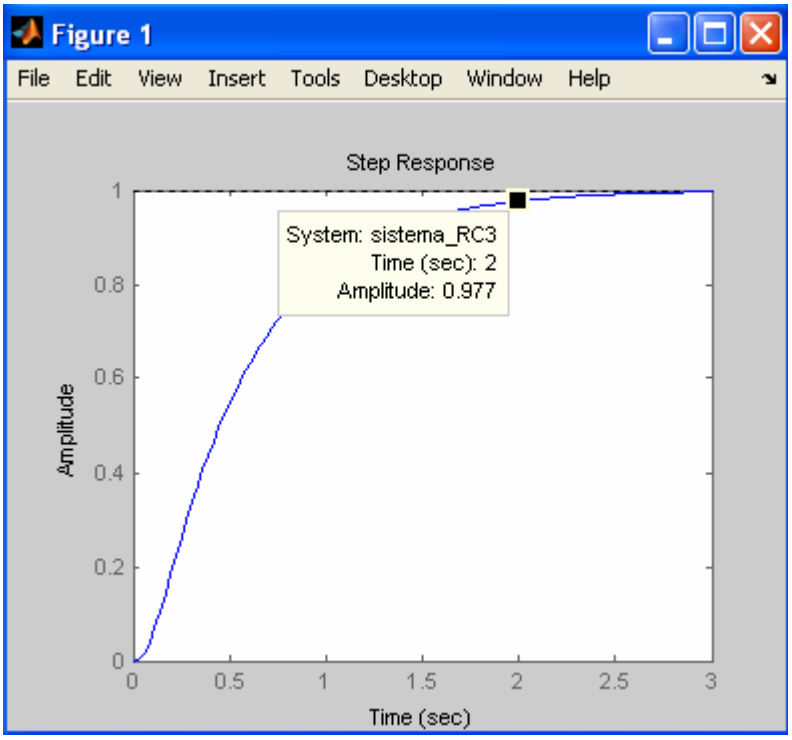
Una vez obtenida nuestra función de transferencia para el sistema resistencias-capacitor de tercer orden. Nuestro diagrama de bloques tomara la siguiente forma:



A continuación simulamos la respuesta de la función de transferencia de nuestro modelo con una entrada escalón a lazo abierto:

Esta respuesta temporal la simulamos en Matlab, introduciendo las siguientes instrucciones:

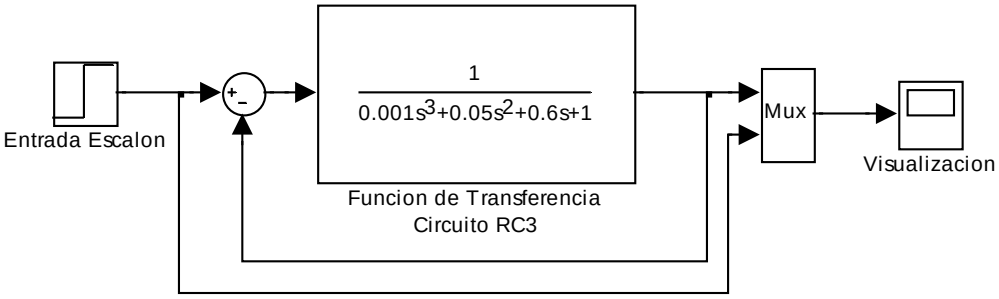
```
sistema_RC3=tf([1],[0.001 0.05 0.6 1])
step(sistema_RC3)
```



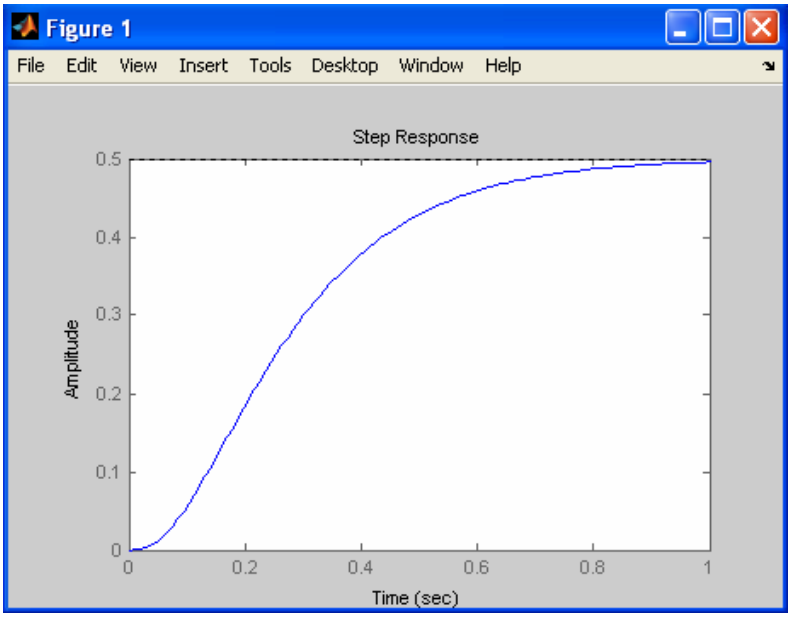
Con la gráfica mostrada arriba observamos que existe un tiempo de asentamiento, de 2 segundos con un respuesta de 0.977 volts del sistema, con lo que podemos decir que se acerca demasiado a la entrada de un valor de1 volts.

En general los sistemas físicos reales que forman parte del sistema de control poseen inercias que le impiden seguir la señal de entrada de manera instantánea, esto implica la existencia de un período transitorio que es necesario que conozcamos el tiempo requerido para llegar al estado estacionario.

Analizaremos el sistema en lazo cerrado con un controlador de ganancia unitaria, introduciendo en Matlab los siguientes comandos:




```
sistema_RC3=tf([1],[0.001 0.05 0.6 1])
Resp_escalón=feedback(sistema_RC3,1,-1)
step(Resp_escalón)
```



La gráfica nos muestra que la respuesta del sistema aumenta si velocidad de respuesta también lo hace, con un tiempo de establecimiento de unos 0.9 segundos En contra, la respuesta del sistema ha disminuido significativamente, aproximadamente 0.5 V.

Realizando un análisis de las respuestas de las gráficas observamos que en lazo abierto y lazo cerrado la respuesta cayo de 1 volt a 0.5 volt, esto sucede debido a que el tiempo tiende a infinito en el dominio de Laplace, esto es lo mismo que la variable compleja tienda a cero.

La función de transferencia en lazo abierto es;

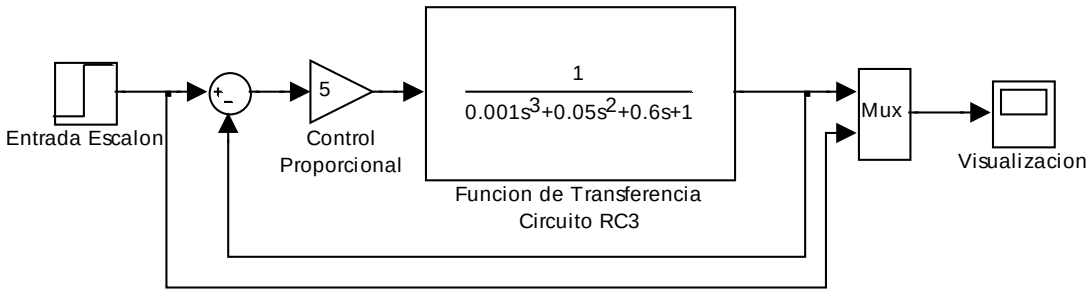
$$sistema\_RC3 = \frac{1}{0.001s^3 + 0.05s^2 + 0.6s + 1}$$

La función de transferencia en lazo cerrado es:

$$sistema\_RC3 = \frac{1}{0.001s^3 + 0.05s^2 + 0.6s + 2}$$

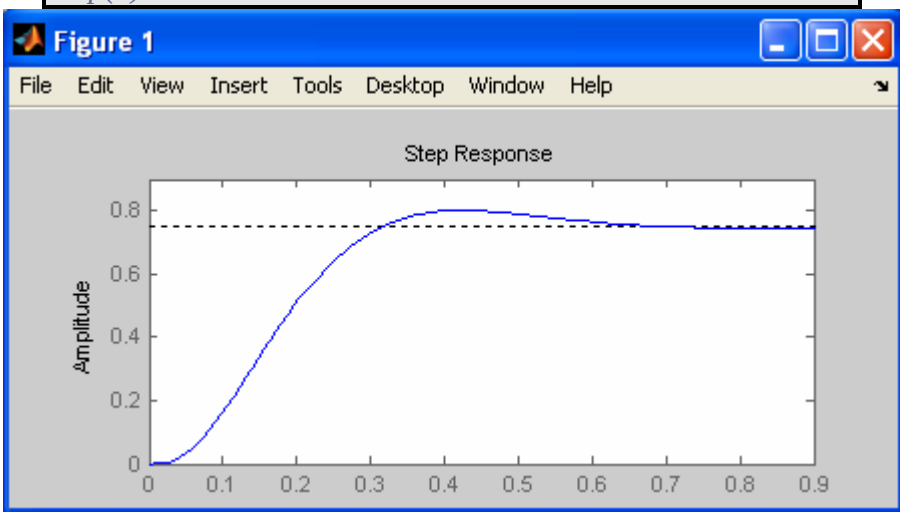
Análisis con una ganancia variable:

A continuación realizamos el análisis de la función de transferencia del circuito resistencia capacitor de tercer grado para poder mover los polos y de esta manera analizar su respuesta para los diversos valores que pueda tomar K, y que mantengan la salida del voltaje del circuito dentro de los márgenes de estabilidad.

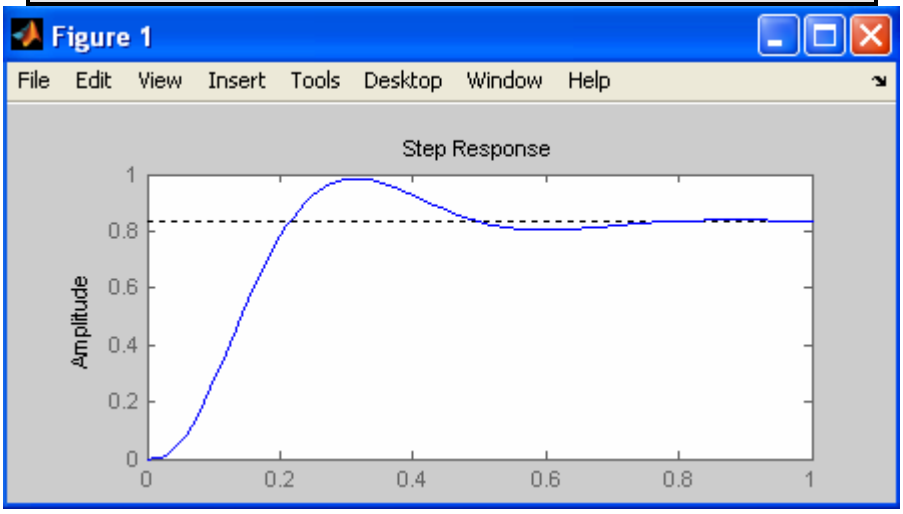


Variamos experimentalmente los valores de la ganancia desde cero hasta lograr que los polos traspasen al semiplano derecho y con esto se consiga la inestabilidad del circuito o sistema pudiendo observar los diferentes tipos de respuestas que se obtiene,

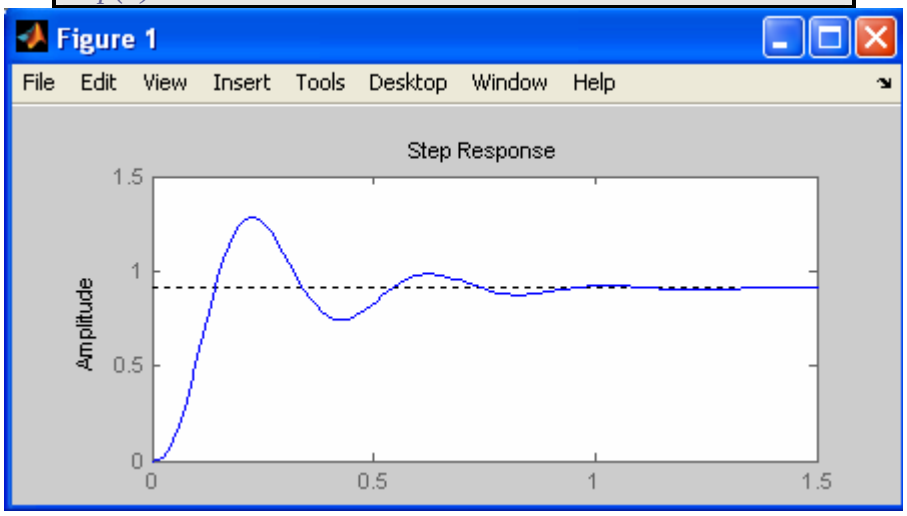
```
k=3
sistema=tf([1],[0.001 0.05 0.6 1]);%FT del circuito
ganancia=tf([k],[1]);%Control Proporcional, valores de K
a=series(sistema,ganancia);%Multiplicacion de los bloques
b=feedback(a,1,-1);%Lazo Abierto
step(b)%Analsis con entrada escalón
```



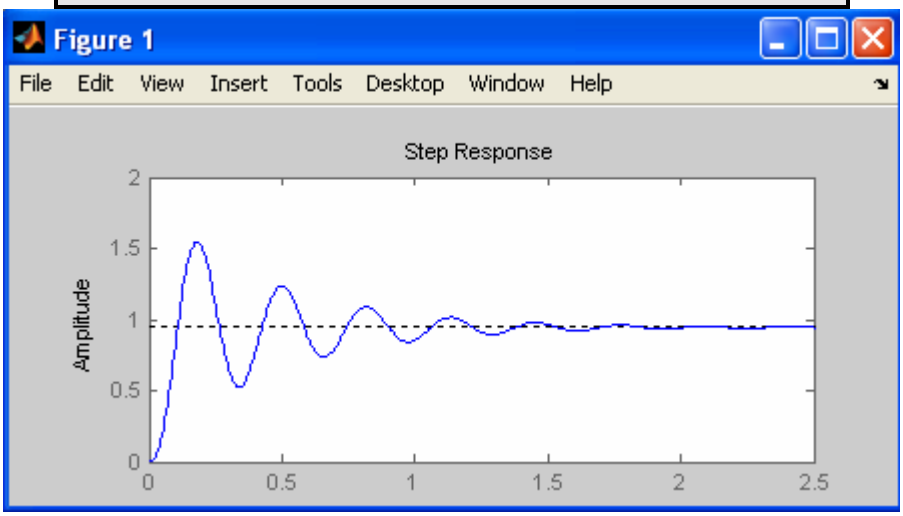

```
k=5
sistema=tf([1],[0.001 0.05 0.6 1]);%FT del circuito
ganancia=tf([k],[1]);%Control Proporcional, valores de K
a=series(sistema,ganancia);%Multiplicacion de los bloques
b=feedback(a,1,-1);%Lazo Abierto
step(b)%Analsis con entrada escalón
```



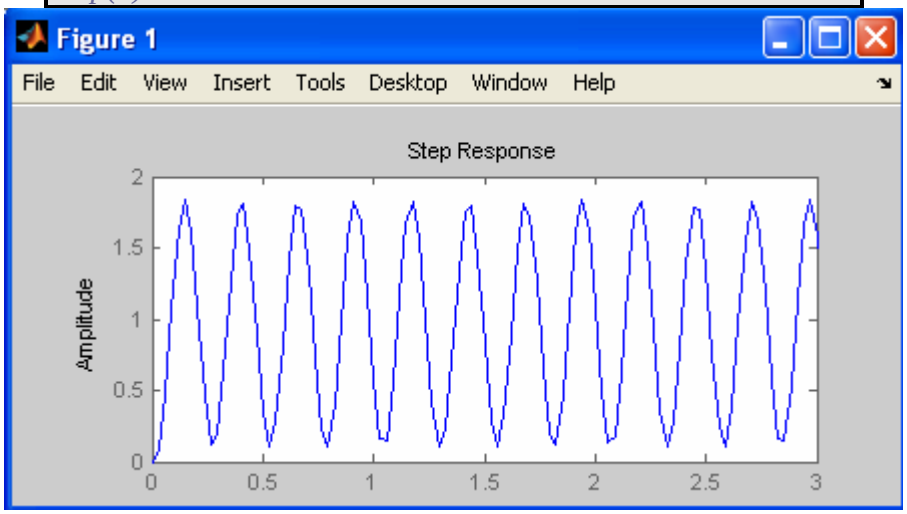
```
k=10
sistema=tf([1],[0.001 0.05 0.6 1]);%FT del circuito
ganancia=tf([k],[1]);%Control Proporcional, valores de K
a=series(sistema,ganancia);%Multiplicacion de los bloques
b=feedback(a,1,-1);%Lazo Abierto
step(b)%Analsis con entrada escalón
```



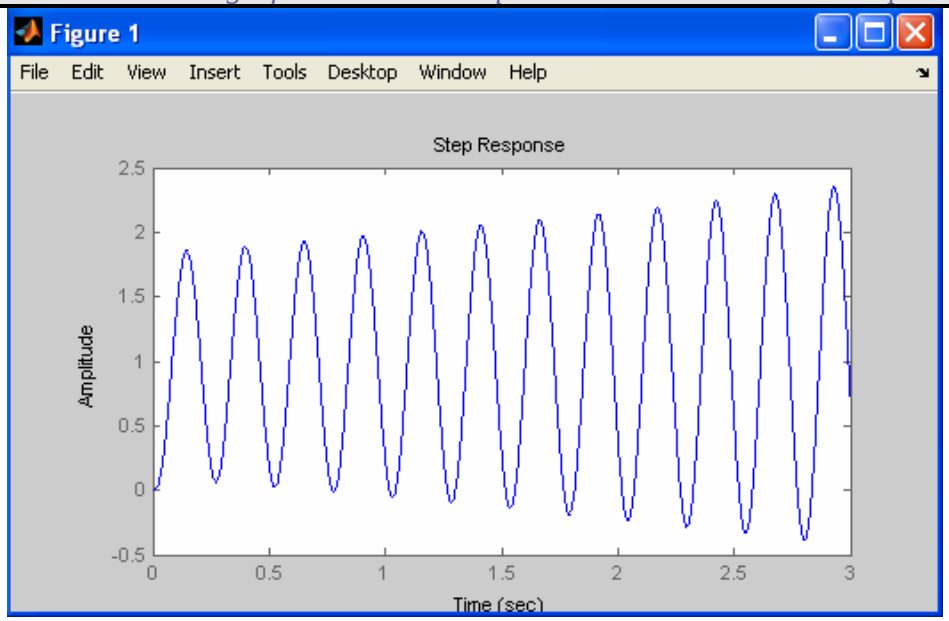

```
k=17
sistema=tf([1],[0.001 0.05 0.6 1]);%FT del circuito
ganancia=tf([k],[1]);%Control Proporcional, valores de K
a=series(sistema,ganancia);%Multiplicacion de los bloques
b=feedback(a,1,-1);%Lazo Abierto
step(b)%Analsis con entrada escalón
```



```
k=29
sistema=tf([1],[0.001 0.05 0.6 1]);%FT del circuito
ganancia=tf([k],[1]);%Control Proporcional, valores de K
a=series(sistema,ganancia);%Multiplicacion de los bloques
b=feedback(a,1,-1);%Lazo Abierto
step(b)%Analsis con entrada escalón
```

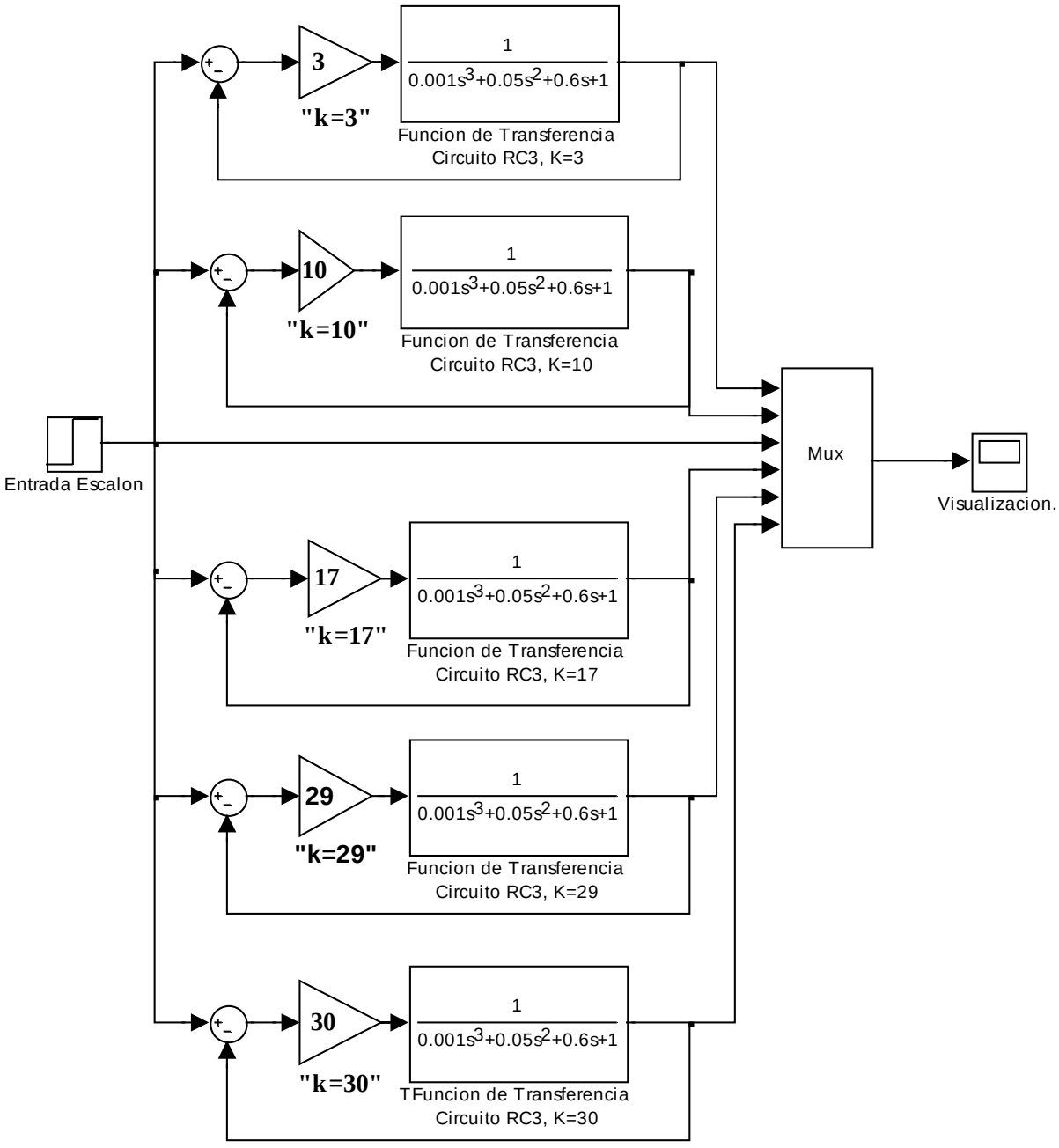



```
k=30
sistema=tf([1],[0.001 0.05 0.6 1]);%FT del circuito
ganancia=tf([k],[1]);%Control Proporcional, valores de K
a=series(sistema,ganancia);%Multiplicacion de los bloques
b=feedback(a,1,-1);%Lazo Abierto
step(b)%Analsis con entrada escalón
%%Analizando la grafica observamos que el sistema se vuelve mas rapido
```

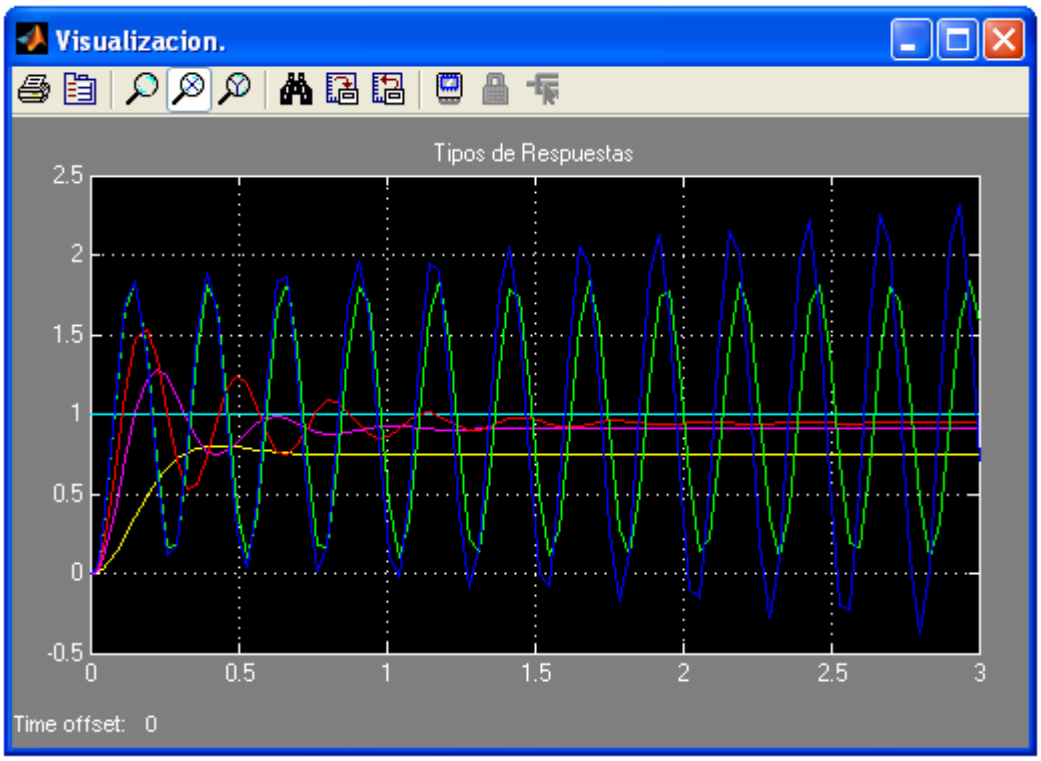


Analizando cada una de las graficas observamos como cambia su respuesta al variar la ganancia con valores de 3 a 30.

A la vista de las gráficas anteriores podemos concretar unos rangos de ganancias a los que les corresponden distintas respuestas.
Obtenemos respuestas sobre amortiguadas con ganancias inferiores a 3.
La respuesta es críticamente estable con una ganancia de 29.
La respuesta es subamortiguada. En el rango de valores de la ganancia entre 3 y 29
La respuesta es inestable con valores mayores a 29



De donde obtendremos la siguiente grafica:


Visualizacion.
Tipos de Respuestas
2.5
2
1.5
1
0.5
0
-0.5
0
0.5
1
1.5
2
2.5
3
Time offset: 0

Controlador PID.

Un controlador PID dispone de un componente proporcional (Kp), un componente integrativo (Ti) y un componente derivativo (Td), de tal manera que produce una señal de control igual a

$$u \approx Kp\left(1+\frac{1}{sTi}+sTd\right)e$$

Donde la acción integrativa del controlador tiene su mayor efecto sobre la respuesta estacionaria del sistema (tratando de minimizar el valor de ess) y la acción derivativa tiene su mayor efecto sobre la parte transitoria de la respuesta.

Para obtener el control PID vamos a utilizar el método de la ganancia crítica, desarrollado por Ziegler y Nichols quienes propusieron reglas para determinar los valores de la ganancia proporcional K_p, del tiempo integral T_i y del tiempo derivativo T_d, basándose en las características de la respuesta transitoria de un sistema dado. Existen dos métodos desarrollado el primero se basa en obtener la respuesta del sistema a una entrada escalón unitario de manera experimental en donde si el sistema no contiene integradores ni polos dominantes complejos conjugados la curva de respuesta tendrá la forma de S, a partir de estas curvas podemos obtener parámetros como el tiempo de retardo y la constante del tiempo del sistema en donde aplicando los valores de la tabla de Ziegler y Nichols se obtiene el controlador PID.

El segundo método es el que utilizaremos para modelar nuestro sistema RC de tercer orden, se basa en dar un valor infinito al tiempo de integración y un valor nulo al tiempo de derivación. Usando solo la acción del control proporcional e ir incrementando los valores de la ganancia proporcional desde cero hasta un valor crítico, donde la respuesta del sistema se nos vuelva inestable, pasando por oscilaciones sostenidas cabe mencionar que si para cualquier valor que pueda tomar k no se presentan las oscilaciones sostenidas este método no aplica, en nuestro caso esto sí existen los periodos sostenidos y sí podremos obtener el la ganancia critica y el periodo critico correspondiente.

Para comenzar vamos a determinar primero el valor de *K* hasta que el sistema se encuentre en la estabilidad marginal. Es decir cuando la salida muestre oscilaciones estables.
En este punto vamos a medir el valor de la ganancia crítica el cual ya hemos obtenido previamente y corresponde a un valor de 29 y el periodo de oscilación al cual llamaremos periodo crítico, este lo obtenemos aplicando el método de Routh Hurwitz el cual representa un procedimiento para evaluar la estabilidad absoluta de un sistema lineal e invariante en el tiempo donde por medio de la inspección del polinomio formado por la ecuación característica.

$$\frac{V_o(s)}{V_i(s)}=\frac{1}{0.001S^3+0.05S^2+0.6S+1}$$

De donde la ecuación característica tiene la forma de:

$$0.001S^3 + 0.05S^2 + 0.6S + 1$$

S^3	1	600
S^2	50	1000+1000K
S^1	580-20K	
S^0	1000+1000K	

$$\frac{50*600-1000-1000k}{50} = 580-20k \quad ; \quad \frac{580}{29} = k_U \Rightarrow 29$$

$$-1\langle k \rangle 29$$

De donde podemos suponer que $29 \approx 30$

Usando la ganancia crítica en el polinomio auxiliar:

$$F_u(s) = 50s^2 + 1000 + 1000k \Rightarrow 50s^2 + 1000(30) \Rightarrow 50s^2 + 3000 = 0$$

$$s_{1-2} = \pm j24.5 \qquad w = 2\pi f \qquad f = \frac{24.5}{2*\pi} \qquad f = 3.9hz$$

$$T_{cr} = \frac{1}{f} = \frac{1}{3.9} = 0.256seg.$$

Con estos valores obtenidos usaremos la tabla desarrollada por Zieglers y Nichols:

Tipo de controlador	Kp	$\frac{1}{T_1}$	Td
P	$0.5^* K_c$		
PI	$0.45^* K_c$	$\frac{T_u}{1.2}$	
PID	$0.6 * K_c$	$T_{cr}/2$	$T_{cr}/2$

Aplicando el método de la ganancia limite por la tabla de Zieglers y Nichols:

	K_p	$1/T_i$	T_d
PID =	$0.6 * K_u$	$T_{cr}/2$	$T_{cr}/2$

Podemos observar que el controlador PID sintonizado mediante el segundo método de Ziegler y Nichols nos produce la siguiente ecuación:

$$G_c(S) = K_p\left(1+\frac{1}{T_iS}+T_dS\right) \quad \Rightarrow \quad 0.6K_{cr}\left(1+\frac{1}{0.5P_{cr}S}+0.125P_{cr}S\right)$$

$$\Rightarrow \quad 0.6K_{cr}\left(1+\frac{2}{P_{cr}S}+0.125P_{cr}S\right)$$ Sacando a S como factor común:

$$\Rightarrow \quad \frac{0.6K_{cr}\left(0.125P_{cr}S^2+S+\frac{2}{P_{cr}}\right)}{S} \quad \Rightarrow \quad \frac{0.075K_{cr}P_{cr}S^2+0.6K_{cr}S+1.2\frac{K_{cr}}{P_{cr}}}{S}$$

Sacando como factor común a: $0.075K_{cr}P_{cr}$

$$\Rightarrow \quad \frac{0.075K_{cr}P_{cr}(S^2+8\frac{S}{P_{cr}}+\frac{16}{P^2{}_{cr}})}{S}$$

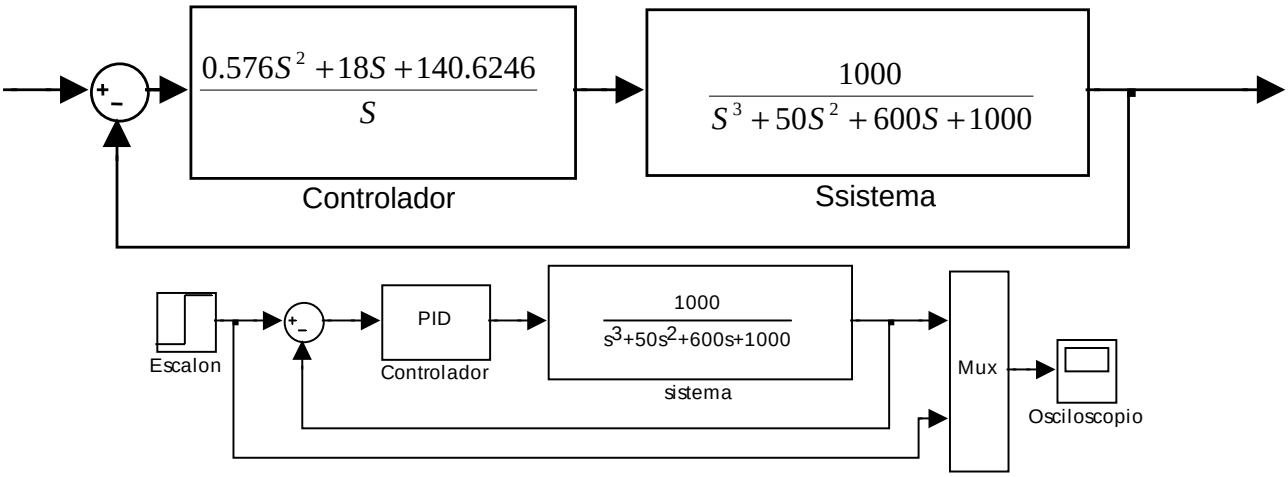
Obtenemos:

$$\Rightarrow \quad \frac{0.075K_{cr}P_{cr}\left(S+\frac{4}{P_{cr}}\right)^2}{S}$$

Podemos observar que el controlador contiene un polo en el origen y un cero doble en $-4/P_{cr}$
Ahora con los valores que anteriormente habíamos obtenido de la ganancia y el periodo crítico los sustituimos en el controlador PID de Ziegler y Nichols, quedándonos de la siguiente forma:

$$\frac{0.075K_{cr}P_{cr}\left(S+\frac{4}{P_{cr}}\right)^2}{S} \quad \Rightarrow \quad \frac{0.075*30*0.256\left(S+\frac{4}{0.256}\right)^2}{S}$$

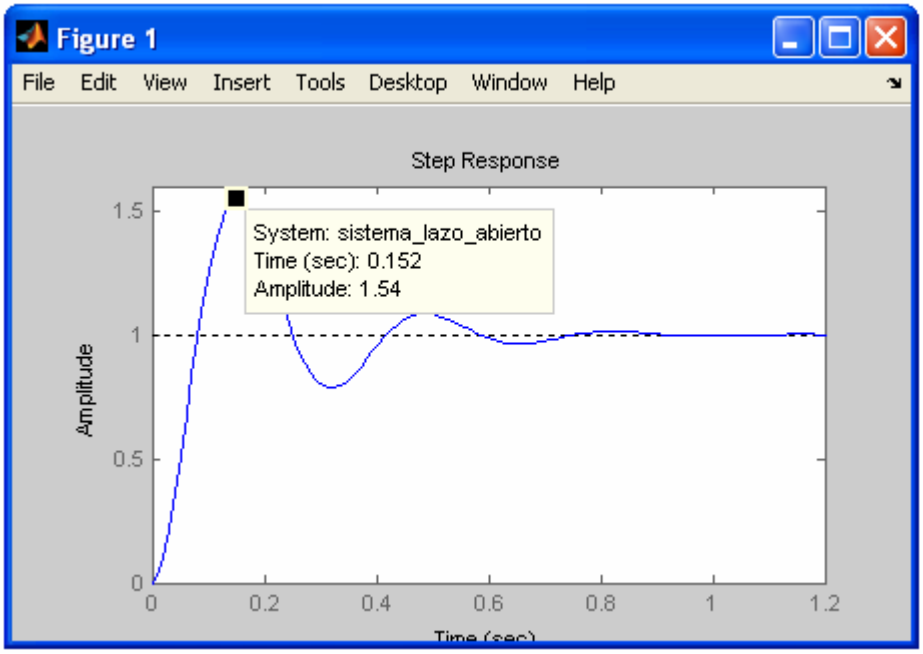
$$\Rightarrow \quad \frac{0.576(S^2+31.2518S+244.14)}{S} \quad \Rightarrow \quad \frac{0.576S^2+18S+140.6246}{S}$$



Una vez obtenido nuestro diagrama de bloques, procedemos a pasarlo a Matlab para simular la respuesta ante una entrada escalón unitario, quedándonos de la siguiente manera:

```
sistema=tf([1000],[1 50 600 1000])
      num_cont = [0.576 18 140.6];
     den_cont = [1 0];
controlador_PID =tf(num_cont,den_cont)
sistema_lazo_cerrado=series(controlador_PID,sistema)
sistema_lazo_abierto=feedback(sistema_lazo_cerrado,1,-1)
step(sistema_lazo_abierto)
```

La siguiente figura nos muestra la curva de la respuesta escalón unitario resultante, el sobre impulso es de aproximadamente un 54%. Este sobre impulso es excesivo y se puede reducir mediante una sintonización muy fina a los parámetros del controlador que hemos diseñado



Para lograr una mejor respuesta del controlador la podemos obtener haciendo que el efecto del doble cero moviéndolo de -15.625 a -7.8125 y mantenido la ganancia K_p obtenida por la tabla de Ziegler y Nichols con el mismo valor de 18,

$$G_c(S) = K_p\left(1+\frac{1}{T_i S}+T_d S\right) \quad = 0.6K_{cr}\left(1+\frac{1}{0.5P_{cr}S}+0.125P_{cr}S\right)$$

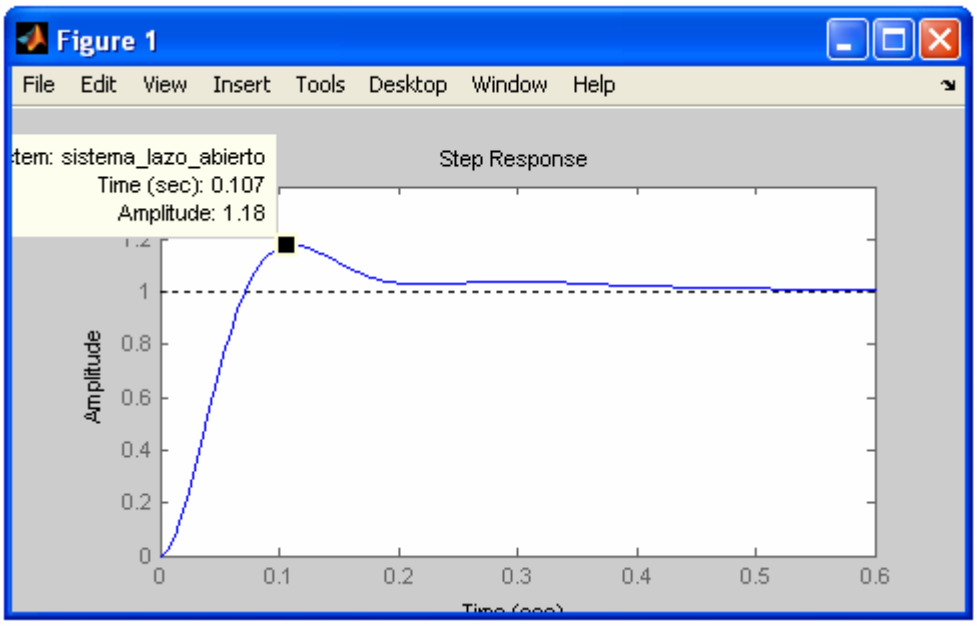
$$\frac{0.075K_{cr}P_{cr}\left(S+\frac{4}{P_{cr}}\right)^2}{S} \qquad \text{de donde} \qquad \frac{4}{P_{cr}} = 7.8125 \Rightarrow P_{cr} = 0.512$$

Y por lo tanto: $\dfrac{0.075*30*0.512(S+7.8125)^2}{S} \Rightarrow \dfrac{1.152S^2+18S+70.31}{S}$

Quedándonos nuestro script de Matlab así:

```
sistema=tf([1000],[1 50 600 1000])
      num_cont = [1.152 18 70.31]
      den_cont = [1 0]
controlador_PID =tf(num_cont,den_cont)
  sistema_lazo_cerrado=series(controlador_PID,sistema)
  sistema_lazo_abierto=feedback(sistema_lazo_cerrado,1)
step(sistema_lazo_abierto)
```

Obtenemos la siguiente figura en donde podemos observar que el sobre impulso se ha disminuido hasta un 18%, lo cual esta por debajo del criterio de Ziegler y Nichols que establece un sobre impulso de un 25% aprox., nuestra figura para el controlador anterior es la siguiente:



Si incrementamos la ganancia proporcional al doble sin modificar la localización del doble cero (S= -7.8125), así como incrementando el valor de la ganancia critica: $Kp = 0.6 * Kc$ entonces tendremos que para un doble valor Kp= 36

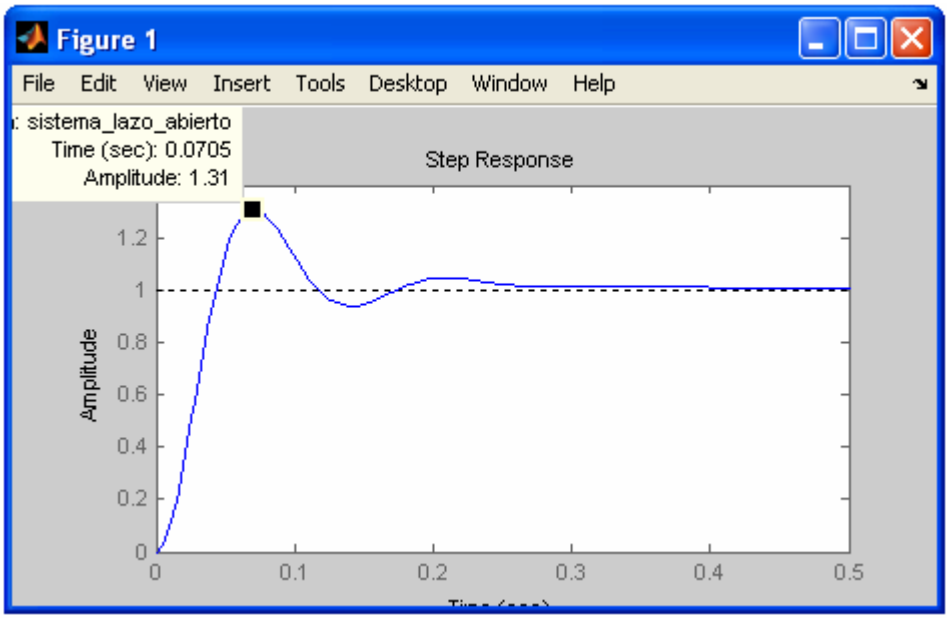
$$\frac{0.075\frac{K_p}{0.6}P_{cr}\left(S+\frac{4}{P_{cr}}\right)^2}{S} \quad \Rightarrow \quad \frac{0.075*\frac{36}{0.6}*0.512\left(S+\frac{4}{0.512}\right)^2}{S}$$

$$\frac{2.304(S+7.8125)^2}{S} \quad \Rightarrow \quad \frac{2.304(S^2+15.6250S+61.0352}{S} \quad \Rightarrow \quad \frac{2.304S^2+36S+140.6251}{S}$$

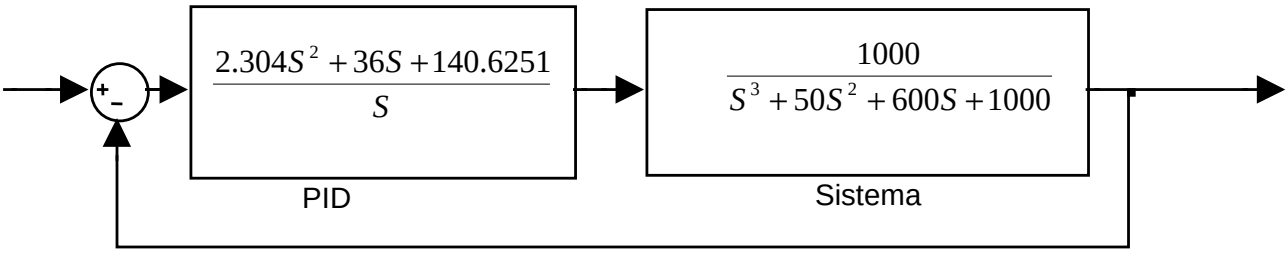
Quedándonos nuestro script de Matlab así:

```
sistema=tf([1000],[1 50 600 1000])
      num_cont = [2.304 36 140.6251  ]
      den_cont = [1 0]
controlador_PID =tf(num_cont,den_cont)
sistema_lazo_cerrado=series(controlador_PID,sistema)
sistema_lazo_abierto=feedback(sistema_lazo_cerrado,1)
step(sistema_lazo_abierto)
```

Con lo que obtenemos una sobre impulso bastante cercana a un 25% y la respuesta del sistema es aun más rápida:



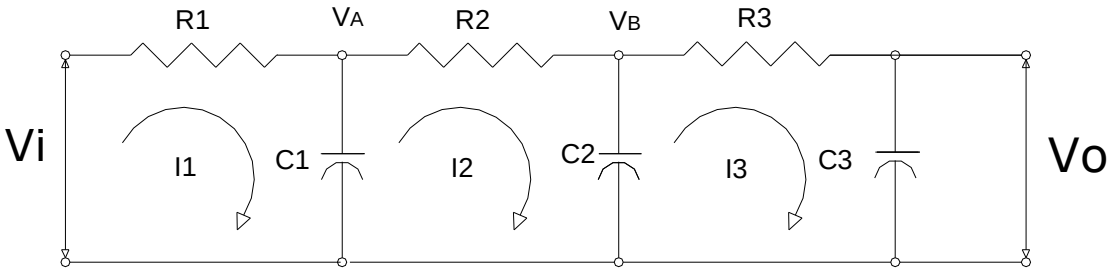
Podemos observar que la velocidad de la respuesta se incrementa pero el sobre impulso también aumenta aproximadamente un 31% como se observa en la figura, como en este caso el sobre impulso es bastante cerca del 25% y la respuesta es más rápida que en el controlador anterior, lo podemos considerar aceptable el controlador quedándonos de la siguiente manera:



Autor: César Leyva Rodríguez email: celero7@yahoo.com.mx
FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD VERACRUZANA.

Caso de Estudio resuelto con Scilab.

Circuito Eléctrico:



Del capítulo anterior tenemos que la siguiente función de transferencia es:

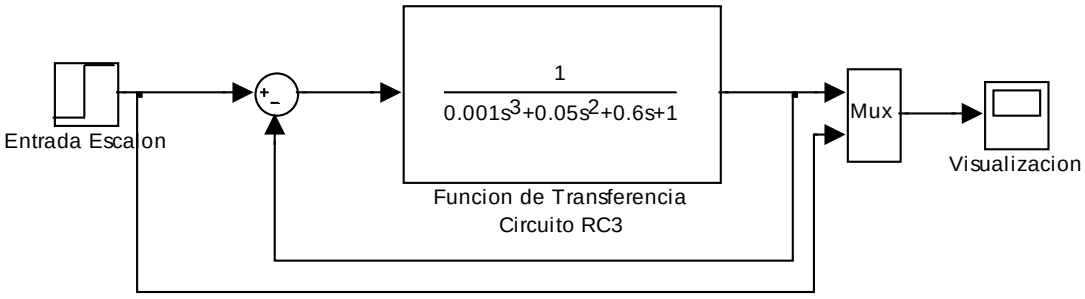
$$\frac{V_o(s)}{V_i(s)} = \frac{1}{0.001S^3 + 0.05S^2 + 0.6S + 1}$$

Todas las operaciones algebraicas y aritméticas realizadas anteriormente las podemos realizar en Scilab declarando los valores de los parámetros del circuito e introduciendo la ecuación (4), quedándonos de la siguiente manera:

```
//Este es un script en Scilab
s=poly(0,'s');
z1=10^6;z3=10^6;z5=10^6;//Valores de las Resistencias;
z2=(1/(s*100*10^(-9)));//Valores del Capacitor;
z4=1/((s*100*10^(-9)));//Valores del Capacitor;
z6=(1/(s*100*10^(-9)));//Valores del Capacitor;
//Introducimos la ecuación para ser reducida a su expresión mas simple;
ft=[(z1/z6)+((z1*z5)/(z4*z6))+(z1/z4)+((z3*z1)/(z6*z2))...
+((z3*z5*z1)/(z4*z6*z2))+((z3*z1)/(z4*z2))+...
((z5*z1)/(z6*z2))+(z1/z2)+(z3/z6)+((z3*z5)/(z4*z6))+(z3/z4)+(z5/z6)+1]^(-1)
//Y obtenemos la función de transferencia del circuito RC3
ft  =
                    1
          -------------------------
                        2         3
      1 + 0.6s + 0.05s + 0.001s
```

Análisis en el dominio del tiempo

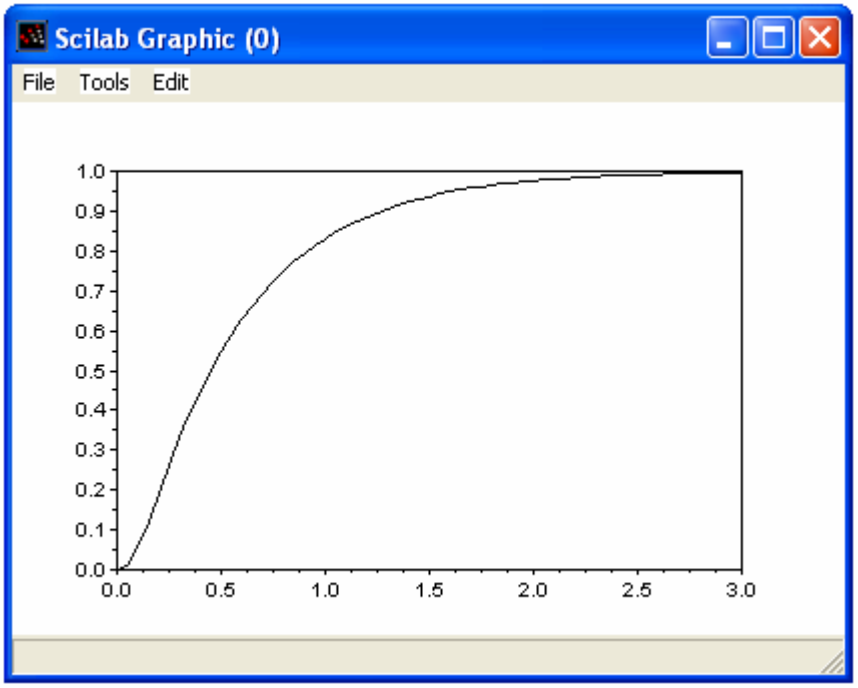
Una vez obtenida nuestra función de transferencia para el sistema resistencias-capacitor de tercer orden. Nuestro diagrama de bloques tomara la siguiente forma:



A continuación simulamos la respuesta de la función de transferencia de nuestro modelo con una entrada escalón a lazo abierto:

Esta respuesta temporal la simulamos en Scilab, introduciendo las siguientes instrucciones:

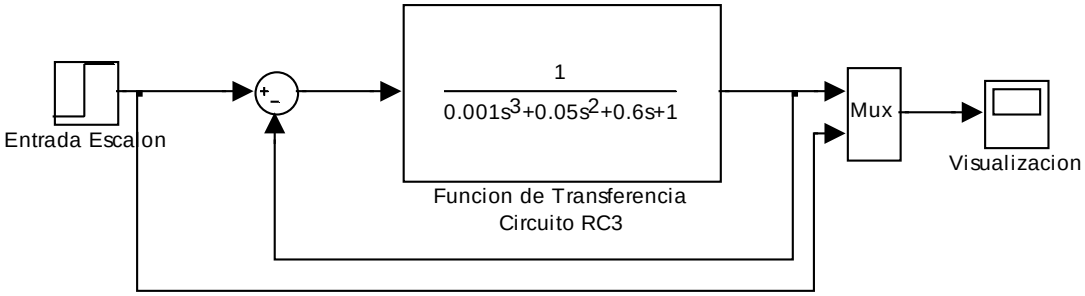
```
s=poly(0,'s'); instante=0:0.05:3;
sistema_RC3=syslin('c',1/(0.001*s^3+0.05*s^2+0.6*s+1))
y=csim('step',instante,sistema_RC3);
plot2d(instante',y')
```



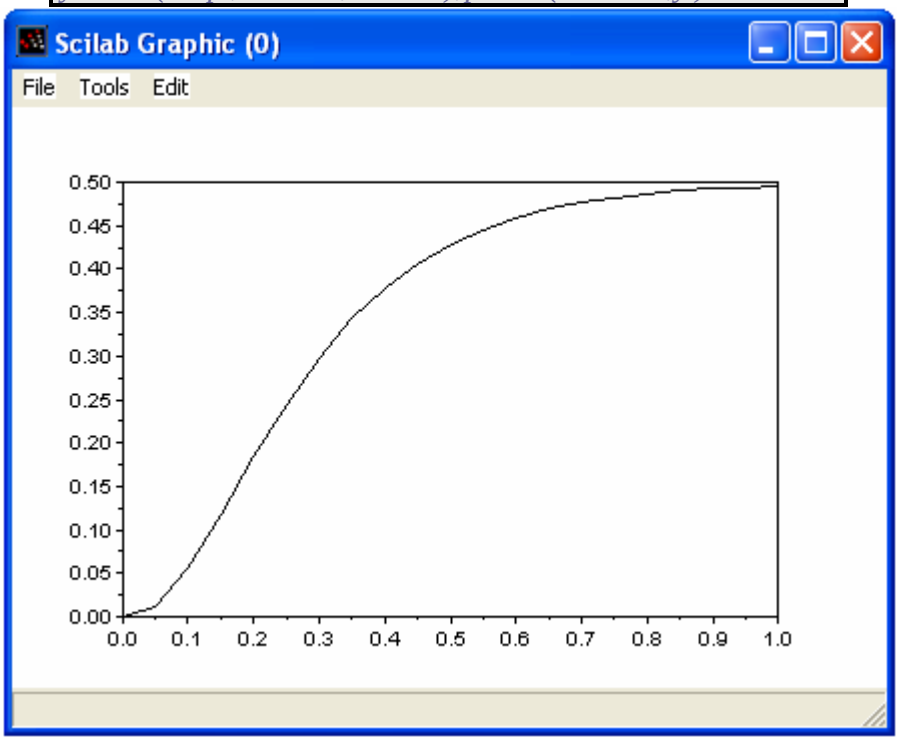
Con la grafica mostrada arriba observamos que existe un tiempo de asentamiento, de 2 segundos con una respuesta de 0.977 volts del sistema, con lo que podemos decir que se acerca demasiado a la entrada de un valor de 1 volts.

En general los sistemas físicos reales que poseen inercias que le impiden seguir la señal de entrada de manera instantánea, esto implica la existencia de un período transitorio que es necesario que conozcamos el tiempo requerido para llegar al estado estacionario.

Analizando el sistema en lazo cerrado con un controlador de ganancia unitaria, introducimos en Scilab los siguientes comandos:




```
s=poly(0,'s'); ;instante=0:0.05:1;
sistema_RC3=syslin('c',1/(0.001*s^3+0.05*s^2+0.6*s+1))
sistema=sistema_RC3/.[1]
y=csim('step',instante,sistema);plot2d(instante',y')
```



La grafica nos muestra que la respuesta del sistema aumenta si velocidad de respuesta también lo hace, con un tiempo de establecimiento de unos 0.9 segundos En contra, la respuesta del sistema ha disminuido significativamente, aproximadamente 0.5 V.

Realizando un análisis de las respuestas de las graficas observamos que en lazo abierto y lazo cerrado la respuesta cayo de 1 volt a 0.5 volt, esto sucede debido a que el tiempo tiende a infinito en el dominio de Laplace, esto es lo mismo que la variable compleja tienda a cero.

La función de transferencia en lazo abierto es:

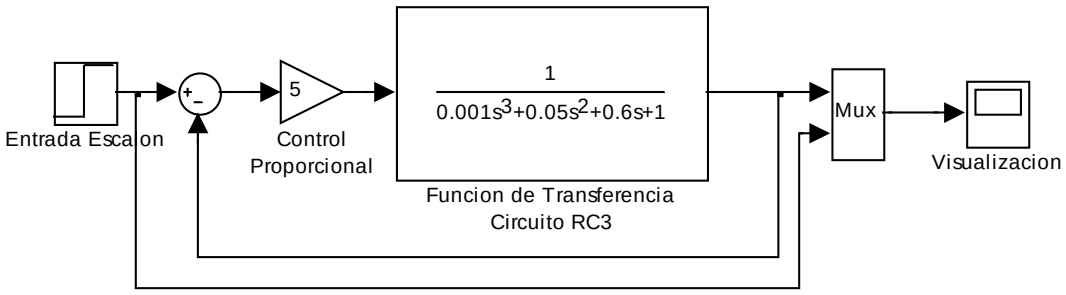
$$sistema\_RC3 = \frac{1}{0.001s^3 + 0.05s^2 + 0.6s + 1}$$

La función de transferencia en lazo cerrado es:

$$sistema\_RC3 = \frac{1}{0.001s^3 + 0.05s^2 + 0.6s + 2}$$

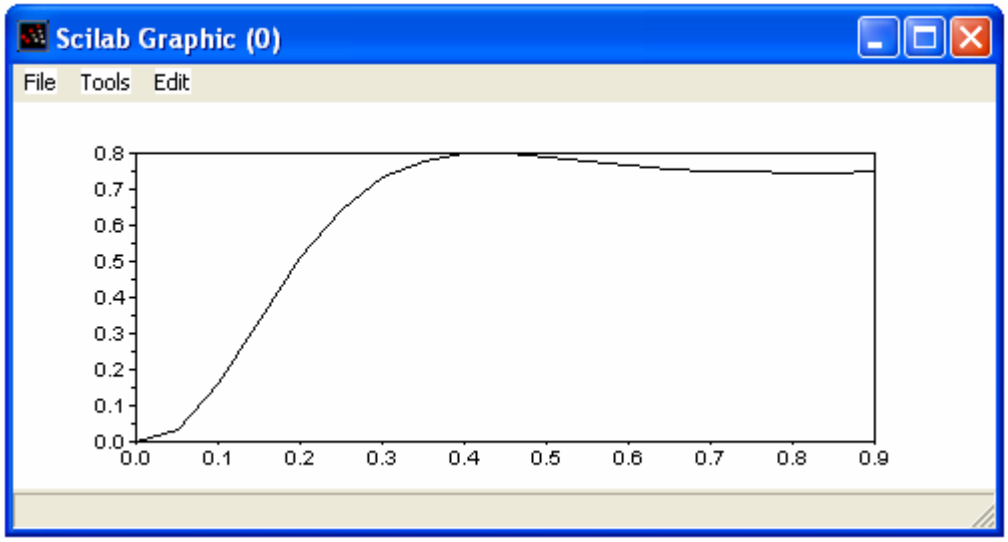
Análisis con Ganancia Variable:

A continuación realizamos el análisis de la función de transferencia del circuito resistencia capacitor de tercer grado para poder mover los polos y de esta manera analizar su respuesta para los diversos valores que pueda tomar K, manteniendo la salida del voltaje del circuito dentro de los márgenes de estabilidad.

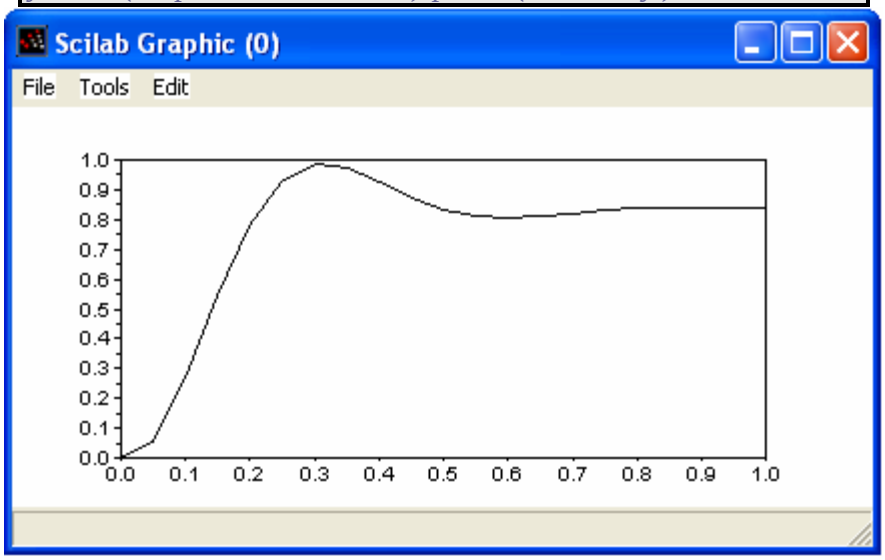


Variamos experimentalmente los valores de la ganancia desde cero hasta lograr que los polos traspasen al semiplano derecho y con esto se consiga la inestabilidad del circuito o sistema pudiendo observar los diferentes tipos de respuestas que se obtiene,

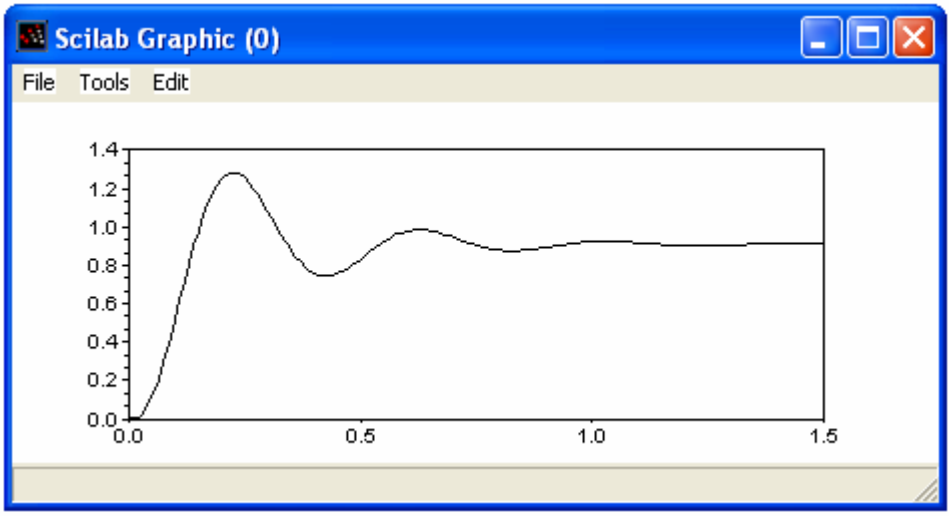
```
k=3;
s=poly(0,'s'); ;instante=0:0.05:0.9;
sistema_RC3=syslin('c',1/(0.001*s^3+0.05*s^2+0.6*s+1))
ganancia=k
a=sistema_RC3*ganancia;//multiplicacion de los bloques
sistema=a/.[1]
y=csim('step',instante,sistema);plot2d(instante',y')
```



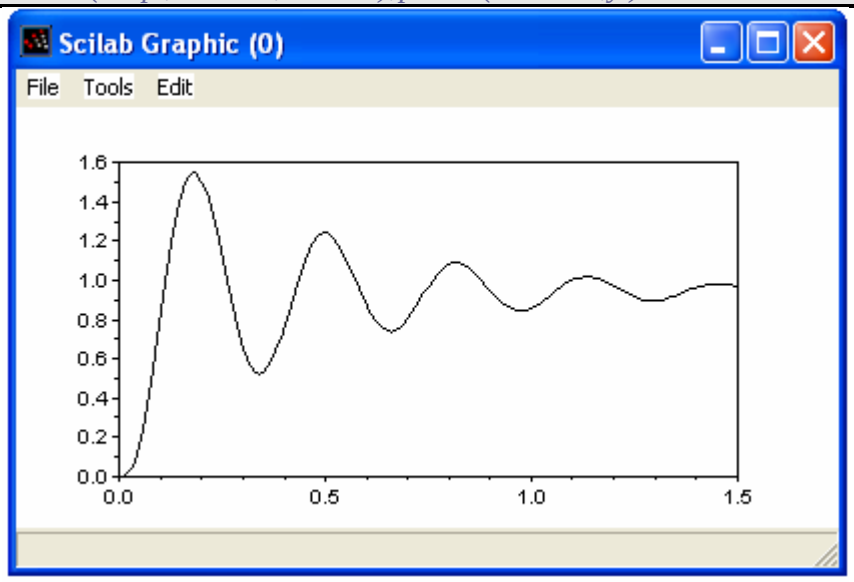

```
k=5;
s=poly(0,'s'); ;instante=0:0.05:1;
sistema_RC3=syslin('c',1/(0.001*s^3+0.05*s^2+0.6*s+1))
ganancia=k
a=sistema_RC3*ganancia;//multiplicacion de los bloques
sistema=a/.[1]
y=csim('step',instante,sistema);plot2d(instante',y')
```



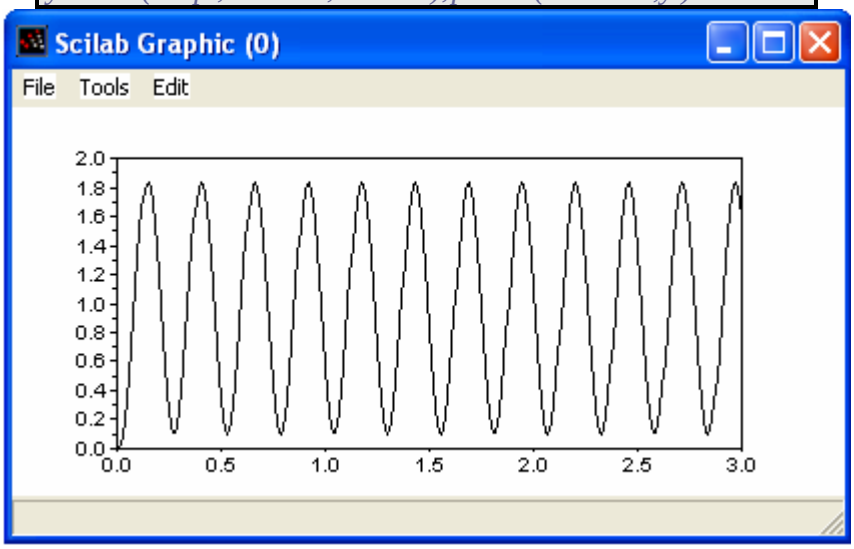
```
k=10;
s=poly(0,'s'); ;instante=0:0.01:1.5;
sistema_RC3=syslin('c',1/(0.001*s^3+0.05*s^2+0.6*s+1))
ganancia=k
a=sistema_RC3*ganancia;//multiplicacion de los bloques
sistema=a/.[1]
y=csim('step',instante,sistema);plot2d(instante',y')
```



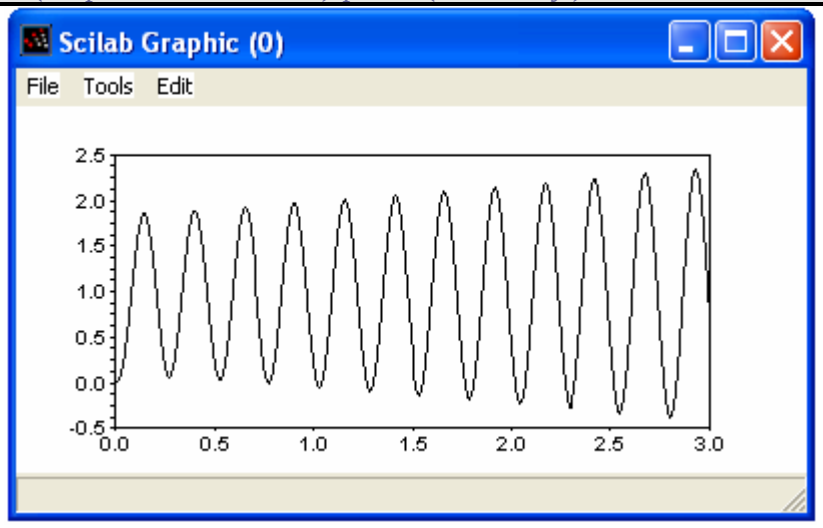

```
k=17;
s=poly(0,'s'); ;instante=0:0.01:1.5;
sistema_RC3=syslin('c',1/(0.001*s^3+0.05*s^2+0.6*s+1))
ganancia=k
a=sistema_RC3*ganancia;//multiplicacion de los bloques
sistema=a/.[1]
y=csim('step',instante,sistema);plot2d(instante',y')
```



```
k=29;
s=poly(0,'s'); ;instante=0:0.01:3;
sistema_RC3=syslin('c',1/(0.001*s^3+0.05*s^2+0.6*s+1))
ganancia=k
a=sistema_RC3*ganancia;//multiplicacion de los bloques
sistema=a/.[1]
y=csim('step',instante,sistema);plot2d(instante',y')
```



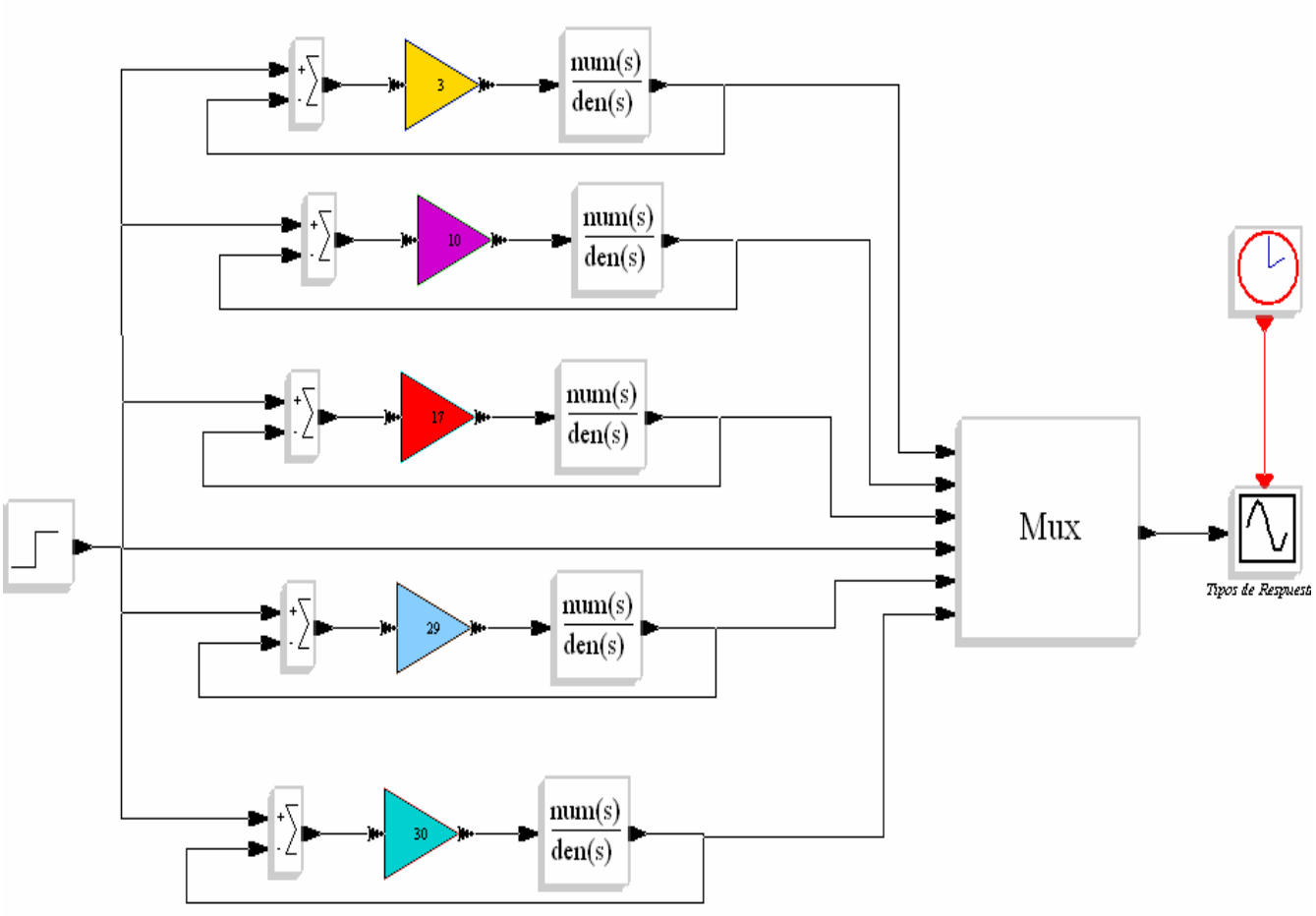

```
k=30;
s=poly(0,'s'); ;instante=0:0.01:3;
sistema_RC3=syslin('c',1/(0.001*s^3+0.05*s^2+0.6*s+1))
ganancia=k
a=sistema_RC3*ganancia;//multiplicacion de los bloques
sistema=a/.[1]
y=csim('step',instante,sistema);plot2d(instante',y')
```



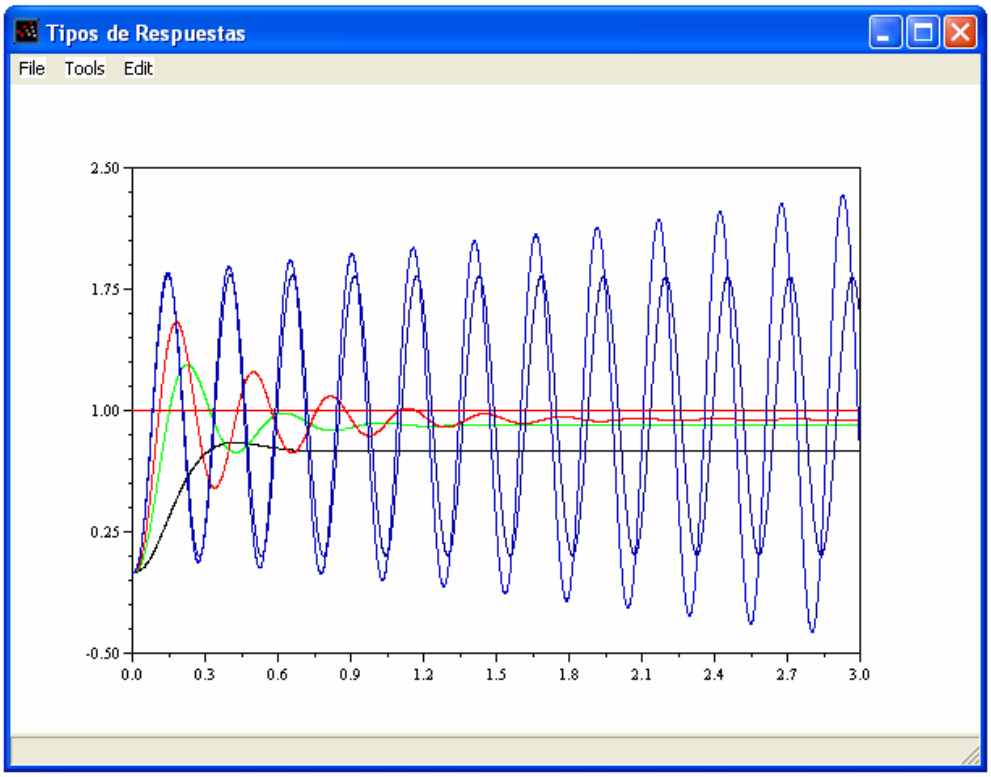
Analizando cada una de las graficas observamos como cambia su respuesta al variar su ganancia con valores de 3 a 30.

A la vista de las gráficas anteriores podemos concretar unos rangos de ganancias a los que les corresponden distintas respuestas.

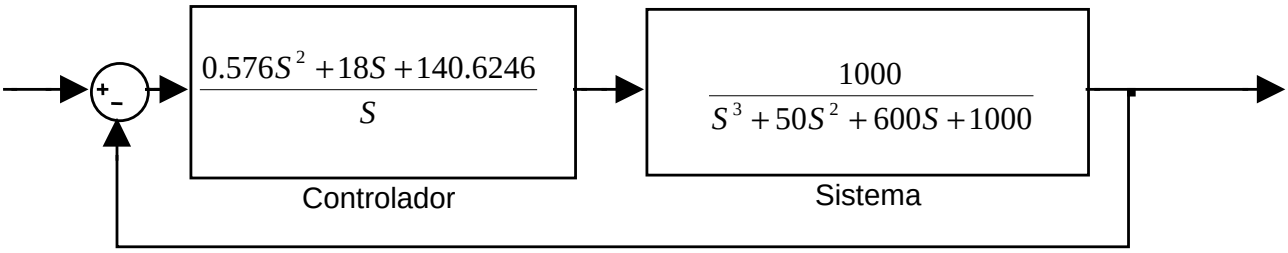
Obtenemos respuestas sobre amortiguadas con ganancias inferiores a 3.
La respuesta es críticamente estable con una ganancia de 29.
La respuesta es subamortiguada. En el rango de valores de la ganancia entre 3 y 29
La respuesta es inestable con valores mayores a 29
Todo lo anterior que hemos realizado desde Scilab lo podemos hacer en Scicos, veamos como nos quedaría nuestro diagrama:



De donde obtendremos la siguiente grafica:

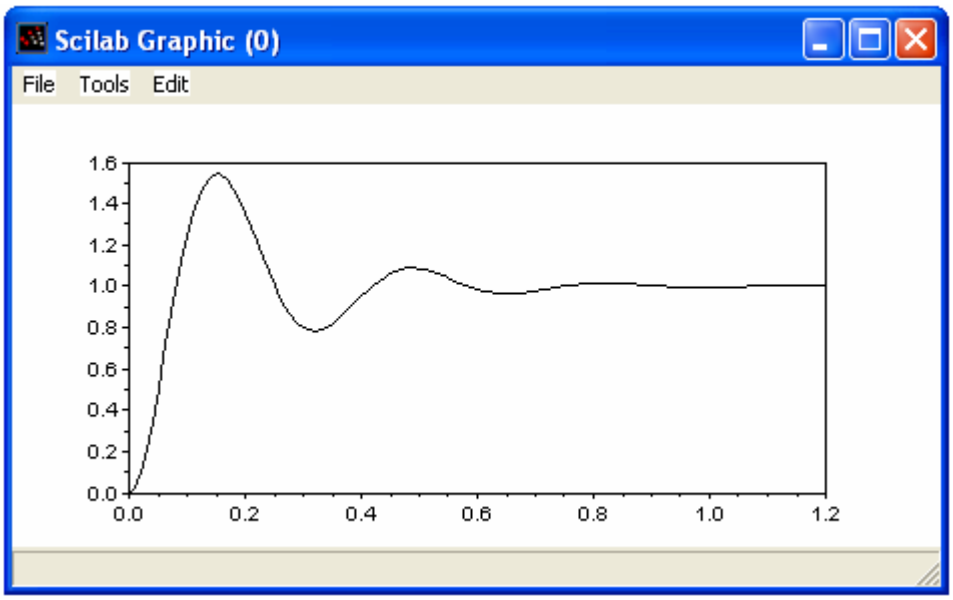


De los parámetros que anteriormente obtuvimos con la tabla de Ziegler y Nichols, obtuvimos el siguiente controlador:

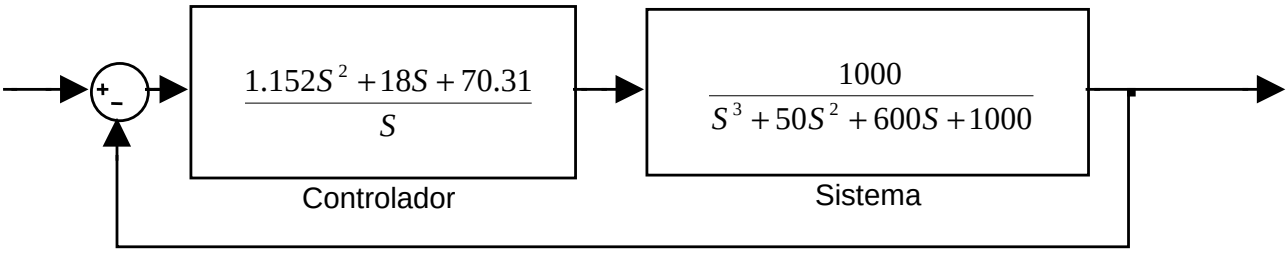


Una vez obtenido nuestro diagrama de bloques, procedemos a pasarlo a Scilab para simular la respuesta ante una entrada escalón unitario, quedándonos de la siguiente manera:

```
s=poly(0,'s'); ;instante=0:0.01:1.2;
num=poly([1000],'s','c'),
den=poly([ 1000 600 50 1],'s','c')
sistema=syslin('c',num,den)
num_con=poly([140.6246 18 0.576],'s','c'),
den_con=poly([0 1],'s','c')
controlador_PID =syslin('c',num_con,den_con)
sistema_lazo_cerrado=controlador_PID*sistema
sistema_lazo_abierto=sistema_lazo_cerrado/.[1]
y=csim('step',instante,sistema_lazo_abierto);
plot2d(instante',y')
```



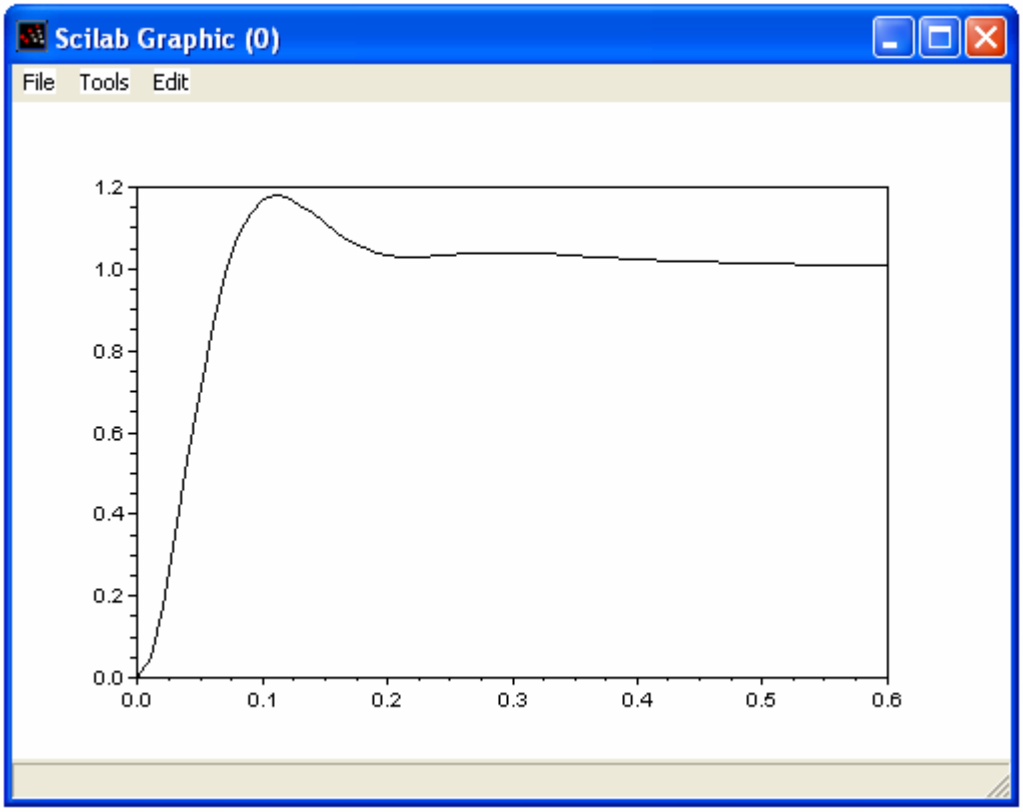
Del segundo intento de mejorar el controlador y haciendo una sintonización, obtuvimos el siguiente diagrama:



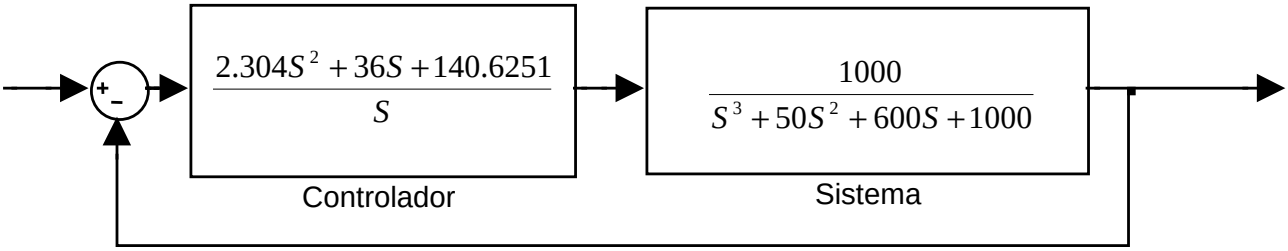
Quedándonos nuestro srcipt de Scilab así:

```
s=poly(0,'s'); ;instante=0:0.01:0.6;
num=poly([1000],'s','c'); den=poly([ 1000 600 50 1],'s','c');
sistema=syslin('c',num,den)
num_con=poly([7.8125^2 7.8125*2*1 1 ],'s','c'); den_con=poly([0 1],'s','c');
controlador=syslin('c',num_con,den_con);
controlador_PID=controlador*0.075*30*0.512
sistema_lazo_cerrado=controlador_PID*sistema
sistema_lazo_abierto=sistema_lazo_cerrado/.[1]
y=csim('step',instante,sistema_lazo_abierto);  plot2d(instante',y')
```

De donde obtenemos la respuesta ante una entrada escalon:



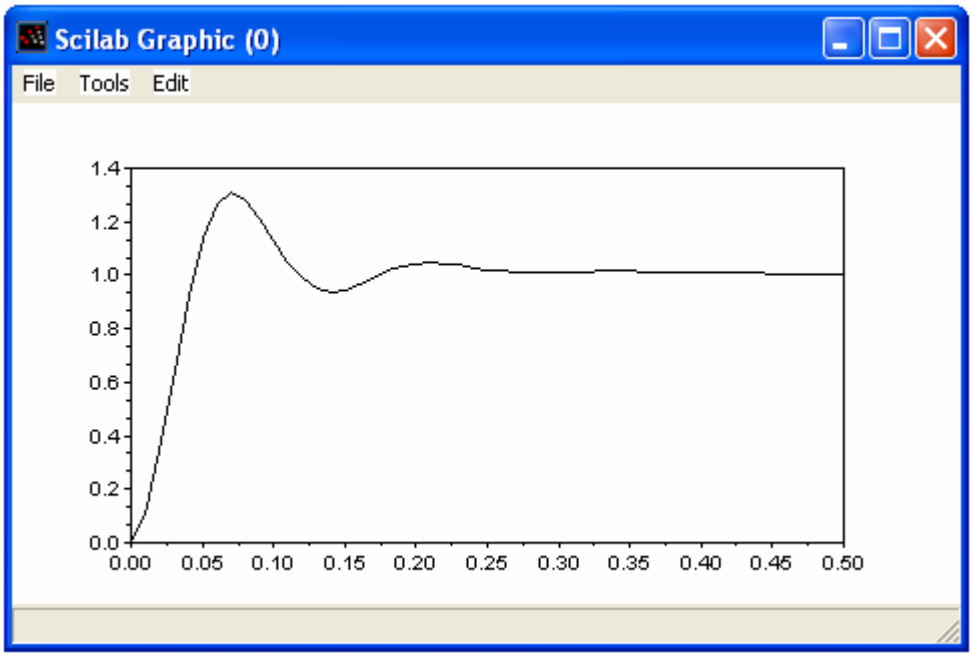
Del tercer intento de mejorar el controlador y volviendo a hacer una haciendo una sintonización, obtuvimos el siguiente diagrama:



Quedándonos nuestro srcipt de Scilab así:

```
s=poly(0,'s'); ;instante=0:0.01:0.5;num=poly([1000],'s','c');
den=poly([ 1000 600 50 1],'s','c');sistema=syslin('c',num,den)
num_con=poly([7.8125^2 7.8125*2*1 1 ],'s','c');den_con=poly([0 1],'s','c');
controlador=syslin('c',num_con,den_con);
controlador_PID=controlador*0.075*(36/0.6)*0.512
sistema_lazo_cerrado=controlador_PID*sistema
sistema_lazo_abierto=sistema_lazo_cerrado/.[1]
y=csim('step',instante,sistema_lazo_abierto);
plot2d(instante',y')
```

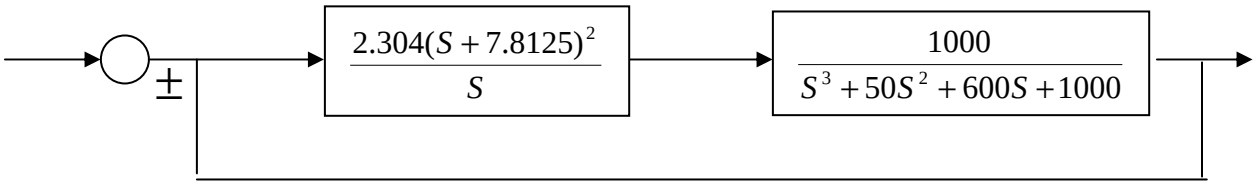
De donde obtenemos la respuesta ante una entrada escalón:



Si incrementamos la ganancia proporcional al doble sin modificar la localización del doble cero (S= -7.8125), también incrementaremos el valor de la ganancia critica: $Kp = 0.6 * Kc$ entonces tendremos que para un doble valor Kp= 36

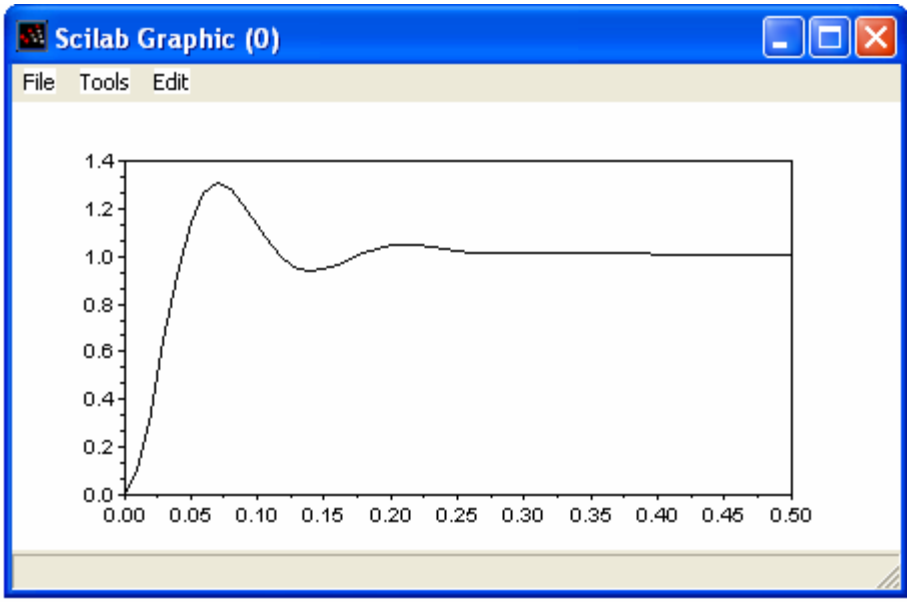
$$\frac{0.075\frac{K_p}{0.6}P_{cr}\left(S+\frac{4}{P_{cr}}\right)^2}{S} = \frac{0.075*\frac{36}{0.6}*0.512\left(S+\frac{4}{0.512}\right)^2}{S}$$

$$= \frac{2.304(S+7.8125)^2}{S}$$




```
s=poly(0,'s'); ;instante=0:0.01:0.5;
num=poly([1000],'s','c');
den=poly([ 1000 600 50 1],'s','c');
sistema=syslin('c',num,den)
num_con=poly([7.8125^2 7.8125*2*1 1 ],'s','c');
den_con=poly([0 1],'s','c');
controlador=syslin('c',num_con,den_con);
controlador_PID=controlador*0.075*(36/0.6)*0.512
sistema_lazo_cerrado=controlador_PID*sistema
sistema_lazo_abierto=sistema_lazo_cerrado/.[1]
y=csim('step',instante,sistema_lazo_abierto);
plot2d(instante',y')
```

Con lo que obtenemos una sobreelogación bastante cercana a un 25% y la respuesta del sistema es aun más rápida.



Podemos observar que la velocidad de la respuesta se incrementa pero la sobreelogación también aumenta aproximadamente un 31% como se observa en la figura, como en este caso la sobreelogación es bastante cerca del 25% y la respuesta es más rápida que en el controlador anterior, lo podemos considerar aceptable el controlador obteniendo los siguientes valores de sintonización:

Por lo tanto $K_p = 0.6K_c \therefore K_c = K_p / 0.6 \rightarrow K_c = 60$

El periodo critico es $P_{cr} = 0.512 = T_u$

	Kp	$\frac{1}{T_1}$	Td
PID =	$0.6 * k_c$	$\frac{T_{cr}}{2}$	$\frac{T_{cr}}{8}$

$0.6 * 60 = 36 \rightarrow Kp$

$\frac{0.512}{2} = 0.256 \rightarrow 3.90625 \rightarrow T_i$

$\frac{0.512}{8} = 0.064 \rightarrow T_d$

Y las constantes del regulador son:

Kp=36		Td=0.064		Ti=3.906025

RESUMEN.

Diseñar y simular un controlador PID de acuerdo a los criterios de Ziegler y Nichols para el siguiente sistema.

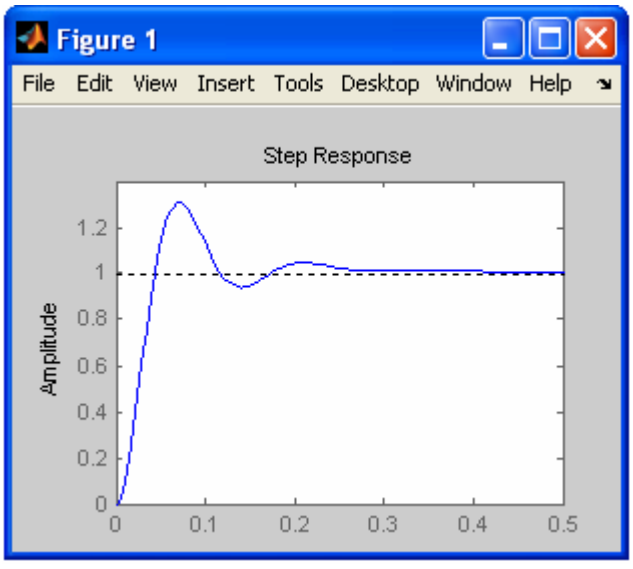
$$\frac{V_o(s)}{V_i(s)} = \frac{1}{0.001\,S^3 + 0.05\,S^2 + 0.6S + 1}$$

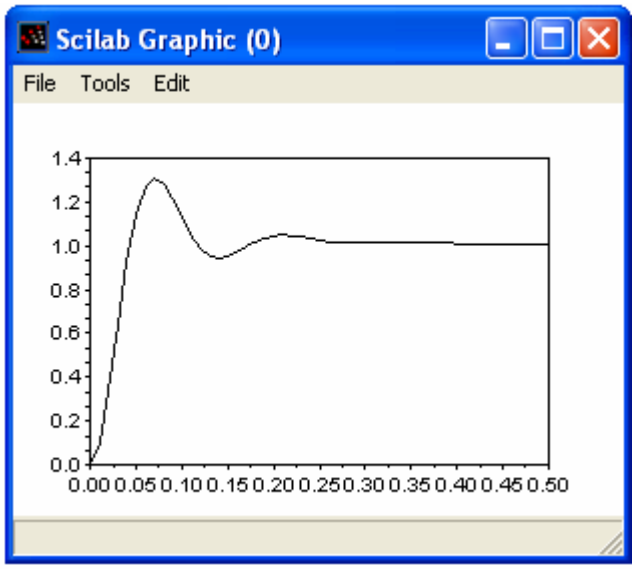
El controlador diseñado anteriormente es:

$$\frac{2.304S^2 + 36S + 140.6251}{S}$$

```
sistema=tf([1000],[1 50 600 1000])
      num_cont = [2.304 36 140.6251  ]
     den_cont = [1 0]
controlador_PID =tf(num_cont,den_cont)
sistema_lazo_cerrado=series(controlador_PID,sistema)
sistema_lazo_abierto=feedback(sistema_lazo_cerrado,1)
step(sistema_lazo_abierto)
```

```
s=poly(0,'s'); ;instante=0:0.01:0.5;
num=poly([1000],'s','c');
den=poly([ 1000 600 50 1],'s','c');
sistema=syslin('c',num,den)
num_con=poly([7.8125^2 7.8125*2*1 1 ],'s','c');
den_con=poly([0 1],'s','c');
controlador=syslin('c',num_con,den_con);
controlador_PID=controlador*0.075*(36/0.6)*0.512
sistema_lazo_cerrado=controlador_PID*sistema
sistema_lazo_abierto=sistema_lazo_cerrado/.[1]
y=csim('step',instante,sistema_lazo_abierto);
plot2d(instante',y')
```





Bibliografía

[1] The Mathworks, Inc. Matlab,
Edición del estudiante. Guía del usuario.
Editorial: Prentice Hall,
Madrid. 2004

[2] The Mathworks, Inc.
La edición del estudiante de Simulink.
Editorial: Prentice Hall.
Madrid, 2004

[3] Apuntes de Sistemas de Control
Dr. Rafael Campillo Rodríguez
Universidad Veracruzana, 2004
Xálapa, Ver. www.uv.mx

[4] A prenda Matlab 7.0 cómo si estuviera en primero
Javier García de Jalón, José Ignacio Rodríguez
Escuela Técnica Superior de Ingenieros Industriales
Universidad Politécnica de Madrid, Edición 2006
www.tayuda.com

[5] Introducción a Matlab, Toolbox de Control y Simulink
Departamento de Electrónica
Especialidad Sistemas Electrónicos
Ingeniería Técnica de Telecomunicación
Universidad de Alcalá, España 2005

[6] Introducción a Matlab y su aplicación al Análisis y Control de Sistemas
Laboratorio de Control Automático de la Carrera de Ingeniería en Telecomunicaciones
Manuel Vargas y Manuel Berenguel
Escuela Superior de Ingenieros
Universidad de Sevilla, España 2005

[7] Introducción informal a Matlab y Octave
Guillem Borrel Nogueras
Escuela Técnica Superior de Ingenieros Aeronáuticos
Universidad Politécnica de Madrid
27 de octubre de 2005 guillemborrell@gmail.com
http://torroja.dmt.upm.es:9673/Guillem_Site/

[8] Página Web de Scilab.
http://scilabsoft.inria.fr/.

[9] Introducción a Scilab.
Héctor Manuel Mora Escobar
Departamento de Matemáticas
Universidad Nacional de Colombia
Bogota, Colombia. Mayo del 2006

[10] Manual de Introducción al Tratamiento de Señales con Scilab para usuarios de Matlab.
Autor: Bernardo A. Delicado (delicadob@inta.es).
Programa de Doctorado en Automática e Informática Industrial.
Escuela Técnica Superior de Ingenieros Industriales
Universidad Politécnica de Madrid edición 2006

[11] Modelación y Estudio de las ecuaciones diferenciales en el dominio de Laplace utilizando Matlab y Simulink.
Ing. Francisco Palomera Palacios
Departamento de Mecatrónica y Automatización,
Instituto Tecnologico de Estudios superiores de Monterrey. Campus Monterrey
fpalomera@itesm.mx http://148.216.5.25/~rrusiles

[12] Introducción a Señales, Sistemas y Control (Versión Preliminar).
De Wikilibros, la colección de libros de texto de contenido libre.
Autor: Cristian Segura.
http://es.wikibooks.org/wiki/Usuario:Csegura26

[13] Técnicas de Proyecto y Compensación
Rubén Alejo Palomares.
Departamento de Ing. Electrónica
UDLAP. México 2002.

[14] Análisis y Diseño de Sistemas de Control en Matlab.
Dr. José Luís Vázquez. 2004.
Departamento de Ing. Electrónica – UDLAP. México.

[15] Evolución, darwinismo y software libre.
Guillermo Movía y Esteban Magnani.
http://pagina12.feedback.net.ar/suplementos/futuro/vernota.php?id_nota=517&sec=13

[16] Foro de Software libre por Richard Stallman
http://www.gnu.org/philosophy/shouldbefree.es.html

[17] Los nuevos cercamientos en el dominio público contra la privatización del mundo.
Daniel Bensaïd.
http://www.ft.org.ar/Notasft.asp?ID=1185

[18] ¿"Crisis del trabajo" o crisis del capitalismo?
Juan Chingo.
http://www.ft.org.ar/estrategia/ei1112/trabajo.htm

[19] Ecuaciones Diferenciales con Aplicaciones a Transitorios de Circuitos
Prof. Carlos A. Garbarello. cgarbarello@yahoo.com.ar, garby@speedy.com.ar.
Profesor Titular de la cátedra de Laboratorio de Mediciones Eléctricas II
Escuela Técnica Nº 9 "Ing. Luís A. Huergo"
Secretaría de Educación del Gobierno Autónomo de la Ciudad de Buenos Aires

[20] Respuesta en el Tiempo de los Sistemas de Control.
Prof. Carlos A. Garbarello
Profesor Titular de la cátedra de Laboratorio de Mediciones Eléctricas II
Escuela Técnica Nº 9 "Ing. Luís A. Huergo"
Secretaría de Educación del Gobierno Autónomo de la Ciudad de Buenos Aires

[21] Investigación del Toolbox de Matlab.
Alumno: Miguel Ángel Joya Chagollan.
Profesor: José Rodrigo Rodríguez Godinez.
Materia: Teoría de Control II
Centro Universitario de Ciencias Exactas e Ingenierías.
Universidad de Guadalajara. Abril del 2002

[22] Control Automático de Procesos. Teoría y Práctica.
Smith, Carlos A. Corripio.
Limusa Noriega Editores. Edición 1996

[23] Control de Sistemas Dinámicos con Retroalimentación.
Franklin, Gene. Powell y David. Emami-Naeine,
Abbas Addison-Wesley Iberoamericana. Edición 1991.

[24] Control System Design
Chen, Chi-Tsong.
Saunders College Publishing.
Hartcourt Brace Jovanovich College Publishers. Edición 2005

[25] Introducción al uso de Matlab.
Jorge Arturo Pardiñas Mir
Departamento de Electrónica, Sistemas e Informática
Instituto Tecnológico y de Estudios Superiores de Occidente.
Bogota Colombia.
Febrero de 1997

[26] Proyecto Fin de Carrera
Tutorial On Line de Matlab y Simulink
Autor: Inmaculada Olivares Olivares
Director: Dr. Juan Domingo Aguilar Peña
Escuela Superior de Ingenieros Industriales de Navarra.
España junio 2004

[27] Introducción a los Números Complejos.
Patrici Molinàs Mata (pmolinas@uoc.edu)
José Francisco Martínez Boscá (jmartinezb@uoc.edu)
Proyecto e-Math, Financiado por la Secretaria de Estado de Educación y Universidades.
Bogota Colombia.

[28] Tutorial de Controladores PID.
Desarrollado por Javier Portillo Berasaluze y Naxo Martín Díaz de Cerio.
Directora de Proyecto. Marga Marcos Muñoz.
Departamento de Ingeniería en sistemas y Automática.
Escuela Técnica Superior de Ingenieros Industriales.
Bilboa, España.

[29] Signal Processing with Scilab.
Scilab Group
INRIA Meta2 Project/ENPC Cergrene and Unité de Recherche de Rocquencourt
Domaine de Voluceau-Rocquencourt-B.P 105-78153 Le Chesnay Cedex (France)
E-mail : scilab@inria.fr