UNIVERSIDAD NACIONAL DE CORDOBA

Facultad de Ciencias Exactas Físicas y Naturales

Curso: Herramientas en MATLAB y/o OCTAVE

ORIENTADO A ALUMNOS DE SISTEMAS DE CONTROL I.

Introducción y consideraciones.

Este curso tiene como propósito orientar a los estudiantes de sistemas de control en el uso de esta poderosa herramienta. Además tiene la intención de colaborar con la formación de conceptos sobre los temas de control.

Por otro lado se intenta promover el uso del software libre (Linux) , y se intenta hacer notar los beneficios de utilizar este tipo de sistema operativo. Matlab es un programa comercial, pero hay versiones para Linux que están completas y tienen licencia para estudiantes. Además hay programas muy similares a Matlab pero totalmente gratuitos. Uno de estos programas es el OCTAVE, que funciona de manera muy similar a Matlab.

La razón por la cual se intenta promover el uso de software libre es el hecho de que en nuestra realidad actual es impensable la posibilidad de tener programas originales con su respectiva licencia de uso. Esto trae aparejado un marco de ilegalidad en los desarrollos que puedan realizarse con ese software pirateado. Esta ilegalidad impide que nos desarrollemos como universidad, y hace que aumente el nivel de dependencia.

Entonces es bastante ilógico trabajar de esta manera teniendo la posibilidad de utilizar software libre, el cual por ser gratuito no es de inferior calidad, sino todo lo contrario. La dificultad en el uso de Software libre, esta en que no es muy conocido lo cual asusta a usuarios novatos. Es por esto que en este curso se intentara orientar y ayudar a la instalación y uso de Linux.

Cualquier Sugerencia Duda o Comentario enviarlo a:

FRANCOBATROUNI@DATAFULL.COM

INTRODUCCIÓN

MATLAB (matrix laboratory) es, conceptualmente, un lenguaje de programación de alto nivel. Contiene amplias librerías de funciones matemáticas que nos permitirán operar con matrices y obtener representaciones gráficas de datos. Para utilizar bajo Linux (tambien bajo windows) , hay una versión gratuita del MATLAB (un programa muy similar) llamado OCTAVE. La forma de trabajo es muy similar.

Nota:

Se recomienda ***siempre*** *leer las unidades desde* ***Definición de Variables y Matrices, y la información general.***

Si se está interesado en trabajar ***solo en sistemas de control*** *se* ***puede pasar por alto todo lo que sigue hasta la unidad de Graficas de Funciones y luego Funciones de Transferencia – Respuesta Temporal –Respuesta en Frecuencia.***

En el caso de querer tener un manejo mas amplio de Matlab se recomienda leer todo el manual.

DEFINICIÓN DE VARIABLES

Asignamos valores numéricos a las variables simplemente tecleando las expresiones correspondientes:
a = 1+2
lo cual resulta:
a =3
Si colocamos ; al final de la expresión, el resultado se almacena en a pero no aparece en pantalla.
Por ejemplo teclear
a = 1+2;.
MATLAB utiliza los siguientes operadores aritméticos:

+ suma
- resta
*** multiplicación**
/ división
^ potencia
' transposición

Una variable puede ser asignada mediante una fórmula que emplee operadores aritméticos, números o variables previamente definidas. Por ejemplo, como a estaba definida de antemano, la siguiente expresión es válida:
b = 2*a;
Para visualizar o recordar el valor de una variable que ha sido previamente asignada basta con teclearla de nuevo. Si tecleamos:
b
obtenemos:
b =6
Si la expresión no cabe en una línea de la pantalla, podemos utilizar una elipsis, esto es, tres o mas puntos suspensivos:

c = 1+2+3+...
5+6+7;
Existen algunas variables predefinidas en MATLAB, por ejemplo:

i - sqrt(–1)
j - sqrt(–1)
pi - 3.1416...
o sea que, si introducimos:
y= 2*(1+4*j)
tendremos
y=2.0000 + 8.0000i
Existe también una serie de funciones predefinidas que pueden ser empleadas para asignar valores a nuevas variables, por ejemplo:
abs: valor absoluto de un número real o módulo de un número complejo
angle: fase de un número complejo en radianes
cos: función coseno, con el argumento en radianes
sin: función seno, con el argumento en radianes
exp: función exponencial
Por ejemplo, con la y definida anteriormente:
c = abs(y)
resulta en
c =8.2462
Mientras que:
c = angle(y)
resulta en
c =1.3258
y con a=3 como definimos antes:
c = cos(a)
resulta en
c =-0.9900
Mientras que:
c = exp(a)
resulta en
c =20.0855
Nótese que exp puede usarse con números complejos. Por ejemplo, para el y = 2+8i definido anteriormente:
c = exp(y)
resulta en
c =-1.0751 + 7.3104i

DEFINICIÓN DE MATRICES

MATLAB está basado en el álgebra de vectores y matrices, incluso los escalares son considerados matrices de elementos. Así que las operaciones entre vectores y matrices son tan simples como las operaciones de cálculo comunes que ya hemos revisado. Los vectores pueden definirse se dos formas. Por una lado, podemos definirlos explícitamente, introduciendo los valores de los elementos:
v = [1 3 5 7];
este comando crea un vector de dimensiones con los elementos 1, 3, 5 y 7. Podemos usar comas para separar los elementos. Además, podemos añadir elementos al vector, teclear:
v(5) = 8;
resulta en el vector v = [1 3 5 7 8]. Los vectores definidos con anterioridad pueden servirnos para definir nuevos vectores, por ejemplo:
a = [9 10];
b = [v a];
origina el vector b = [1 3 5 7 8 9 10].
El otro método se utiliza para definir vectores con elementos equi-espaciados:
t = 0:.1:10;
origina un vector de dimensiones con los elementos 0, .1, .2, .3,...,10.
Nótese que en la definición de t, el número que aparece en medio define el incremento de un elemento al siguiente. Si sólo tenemos dos números, el incremento por defecto es 1. Así:
k = 0:10;

da lugar a un vector de dimensiones 1x11 con los elementos 0, 1, 2, ..., 10.

Para definir una matriz A=[1 2 3;4 8 2;2 3 5] los elementos de las filas separados por espacios, y las columnas se separan con punto y coma.

Las funciones se aplican elemento a elemento:
t = 0:10;
x = cos(2*t);
origina un vector x cuyos elementos valen Cos(2t) para t=0,1,2.....10
A veces necesitamos que las operaciones se realicen elemento a elemento. Para ello precedemos el operador correspondiente de un punto ".". Por ejemplo, para obtener un vector x que contenga como elementos los valores de x(t)=t*cos(t) para unos instantes de tiempo determinados, no podemos multiplicar simplemente el vector t por el vector cos(t). Lo que hacemos es:
t = 0:10;
x = t.*cos(t); (nótese el punto después de t)

INFORMACIÓN GENERAL

MATLAB detecta las mayúsculas y minúsculas como diferentes, de modo que "a" y "A" serán dos variables distintas.
Las líneas de comentario en los programas deben estar precedidas de "%"
Mediante el comando help podemos obtener ayuda on-line. Si tecleamos help aparecerá todo un menú de temas sobre los que existe la ayuda y si tecleamos help seguido del nombre de una función recibiremos ayuda específica para dicha función.
El número de dígitos con que MATLAB representa los números en pantalla no está relacionado con la precisión con que estos han sido calculados. Para cambiar el formato de pantalla teclearemos 'format short e' si queremos notación científica con 5 cifras significativas, 'format long e' para notación científica con 15 cifras significativas y 'format bank' para tener sólo dos dígitos decimales.
Los comandos who y whos nos dan los nombres de las variables definidas actualmente en el espacio de trabajo (workspace). Tambien hay un icono para acceder al workspace y observar todas las variables.
El comando length(x) nos da la longitud del vector x y size(x) las dimensiones de la matriz x.

Salvar y recuperar datos desde un fichero
Es muy probable que cuando usemos MATLAB estemos interesados en guardar los vectores y matrices que hemos creado. Para hacer esto sólo tenemos que hacer: save nombre_del_fichero y para recuperar dichos datos en otra sesión :load nombre_del_fichero
Es mas sencillo hacer esto a través de la ventana de comando file, y elegir la opcion save workspace as y luego para recuperar load workspace.
Para mas información escribir help save.

RAICES DE ECUACIONES DE GRADO N **(se puede pasar por alto)**

MATLAB, permite trabajar ya sea utilizando métodos numéricos ó paralelamente se puede trabajar con variables simbólicas.

Si queremos encontrar las raices de una ecuación por ejemplo: X^2+ 2*X +1

r=**roots**([1 2 1]) ------ Los coeficientes se ponen en orden decreciente
Matlab entregara el resultado:
r =
-1
-1
donde r será un vector de 2x1 cuyos elementos son las raíces del polinomio con coeficientes [1 2 1]

Si queremos encontrar un polinomio a partir de sus raíces por ejemplo tenemos las raices [-1,-1]
Escribimos;

```
p=poly([-1 –1])
p =
    1     2     1
```

donde p será un vector 1x3 cuyos valores serán los coeficientes en orden decreciente del polinomio cuyas raices son [-1 –1].

Nótese que **roots** y **poly** son funciones inversas.

Ahora veamos la forma **simbólica** de realizar lo mismo:

Primero hay que declarar las variables a traves de la funcion **syms**

```
syms x
syms a b c real
```

De esta manera se han creado 4 variables: x que será la variable incógnita y a b y c que seran constantes reales (tambien se podrían haber utilizado como constantes complejas). El nombre de las variables puede ser cualquier otro.
Y utilizamos luego la funcion **solve**('expresión matematica',variable a despejar)

```
solve('a*x^2+b*x+c=0',x)
ans =
[ 1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[ 1/2/a*(-b-(b^2-4*a*c)^(1/2))]
```

Nótese que si utilizamos una función y no asignamos una variable al resultado MATLAB siempre guarda el ultimo resultado obtenido en una variable ans

Vemos también que la forma en que **ans** esta escrito de una forma un poco difícil de leer por lo que es muy útil una función que se llama **pretty** hace lo siguiente:

```
pretty(ans)
```

```
[                2                1/2]
[    -b~ + (b~  - 4 a~ c~)   ]
[1/2 ------------------------    ]
[             a~                 ]
[                                ]
[                   2         1/2]
[    -b~ - (b~  - 4 a~ c~)   ]
[1/2 ------------------------    ]
[             a~                 ]
```

ahora el resultado es mas legible.

Si luego se desea reemplazar el valor de a, b y c se utiliza la funcion
subs(ecuación,{variable1, variable2, variablen},{valor1,valor2,valorn})

en nuestrocaso

```
subs(ans,{a,b,c},{1,2,1})

ans =
   -1
   -1
```

entrega el valor numérico de las raices.

Hay dos funciones que nos permiten pasar del la forma numérica a la simbólica y viceversa.

Son **sym2poly** (de simbólico a numérico) y **poly2sym** (de numérico a simbólico)

EJ:
sym2poly(x^2+2*x+1)
ans =
1 2 1
poly2sym([1 2 1])
ans =
x^2+2*x+1

SOLUCION DE SISTEMAS DE ECUACIONES (se puede pasar por alto)

Hay muchísimas formas de resolver un sistema de de ecuaciones. Nombraremos algunas.

Si tenemos una matriz AB=[A B] donde A es la matriz de coeficientes y B es el vector de términos independientes. Podemos reducir la matriz por filas con la funcion **rref**(AB) y asi obtener la solucion del sistema de ecuaciones. A y B pueden contener elementos tanto numéricos como simbólicos.

Otra forma seria con determinantes utilizando la regla de Cramer. La función determinante es **det(A).** Donde A es la matriz cuadrada de coeficientes. Vale la pena repetir que la matriz puede contener elementos simbólicos, y el determinante va a estar en función de esos elementos.

La forma **simbólica** es:

solve('eqn1','eqn2',...,'eqnN','var1,var2,...,varN')

donde eqnn son ecuaciones simbólicas o texto entre comas indicando ecuaciones, y varn son las variables a conocer.

Ejemplo:

[x,y] = **solve**('x^2 + x*y + y = 3','x^2 - 4*x + 3 = 0') entrega

x = [1]

[3]

y = [1]

[-3/2]

el resultado también puede ser simbólico:

[x,y,z]=**solve**('a*x+b*y+c*z=58','b*a*x+c*b*y+c*z=5','c*x+a*y+b*z=0')

x =

-(53*a*c+5*b^2-58*b^2*c)/(-a^2*c+b^2*a*c-b^3*a+c^2*b+c*b*a^2-c^3*b)

y =

-(-5*b*a+58*b^2*a-53*c^2)/(-a^2*c+b^2*a*c-b^3*a+c^2*b+c*b*a^2-c^3*b)

z =

(-5*a^2+5*c*b+58*b*a^2-58*c^2*b)/(-a^2*c+b^2*a*c-b^3*a+c^2*b+c*b*a^2-c^3*b)

y luego si a,b y c fueron declarados como reales (**syms** a b c real) podrán ser reemplazados con la función **subs,** para obtener resultados numéricos

COMANDOS ALGEBRAICOS-LIMITES DERIVADAS E INTEGRALES (se puede pasar por alto)

En algunos casos puede ser de utilidad si tenemos una expresión simbólica complicada simplificarla, o sacar factor común algún termino. Bueno, esta necesidad también está cubierta por Matlab.

collect(S,v) es una función que rescribe la ecuación simbólica S en términos de la variable v.

Ejemplo:

collect(x^2*y + y*x - x^2 - 2*x,x)

devuelve

(y-1)*x^2+(y-2)

f = -1/4*x*exp(-2*x)+3/16*exp(-2*x)

collect(f,exp(-2*x))

devuelve

(-1/4*x+3/16)*exp(-2*x)

la funcion inversa a collect es:

expand(s)

Ejemplo

expand((y-1)*x^2+(y-2)*x)

ans =

x^2*y+y*x-x^2-2*x

NOTA IMPORTANTE: **Si no se especifica la variable MATLAB elige la letra alfabéticamente mas cercana a la 'x'. Esto funciona así para todas las funciones simbólicas.**

La función para simplificar es:

[R,HOW] = **simple**(S) donde S es una expresión simbólica. Esta función va a probar varios métodos de simplificación, y va a devolver el resultado mas simple que encuentre en la variable R y en la variable HOW guardara un texto con el método utilizado.

Ejemplo

S R How

cos(x)^2+sin(x)^2	1	combine(trig)
2*cos(x)^2-sin(x)^2	3*cos(x)^2-1	simplify
cos(x)^2-sin(x)^2	cos(2*x)	combine(trig)
cos(x)+(-sin(x)^2)^(1/2)	cos(x)+i*sin(x)	radsimp
cos(x)+i*sin(x)	exp(i*x)	convert(exp)
(x+1)*x*(x-1)	x^3-x	collect(x)
x^3+3*x^2+3*x+1	(x+1)^3	factor
cos(3*acos(x))	4*x^3-3*x	expand

syms x y positive

log(x) + log(y)	log(x*y)	combine

Matlab tiene funciones especificas para calcular **Limites, Derivadas e Integrales.**

Limites:

limit(F,x,a) limite de la expresión simbólica F cuando **x -> a**

limit(F,x,a,'right') o **limit**(F,x,a,'left') especifica si es limite por la derecha o por la izquierda.

limit(F) toma el limite cuando a=0

Ejemplos

syms x a t h;

limit(sin(x)/x)	1
limit((x-2)/(x^2-4),2)	1/4
limit((1+2*t/x)^(3*x),x,inf)	exp(6*t)
limit(1/x,x,0,'right')	inf
limit(1/x,x,0,'left')	-inf
limit((sin(x+h)-sin(x))/h,h,0)	cos(x)
v = [(1 + a/x)^x, exp(-x)];	
limit(v,x,inf,'left')	[exp(a), 0]

nota: inf es infinito.

Para derivar:

diff(S,'v',n) donde S es una expresión simbólica a derivar, v es la variable con respecto a la cual se quiere derivar y n es el orden de la derivada.

Ejemplo:

diff(sin(x^2)) resulta 2*cos(x^2)*x

diff(t^6,6) resulta 720.

Para Integrar:

INT(S,v) Integra indefinidamente la expresión simbólica S con respecto a la variable v.

INT(S,v,a,b) Calcula la integral definida de la expresión simbólica S con respecto a v entre los limites a y b.

Ejemplo:

syms x x1 alpha u t;

A = [cos(x*t),sin(x*t);-sin(x*t),cos(x*t)];

int(1/(1+x^2)) atan(x)

int(sin(alpha*u),alpha) -cos(alpha*u)/u

int(x1*log(1+x1),0,1) 1/4

int(4*x*t,x,2,sin(t)) 2*sin(t)^2*t-8*t

int([exp(t),exp(alpha*t)]) [exp(t), 1/alpha*exp(alpha*t)]

int(A,t) [sin(x*t)/x, -cos(x*t)/x]

[cos(x*t)/x, sin(x*t)/x] Es la integral de los elementos de la matriz A

TRANSFORMADA Y ANTITRANSFORMADA DE LAPLACE (se puede pasar por alto)

Matlab calcula con exactitud la transformada y la anti-transformada de Laplace de cualquier expresión simbólica por compleja que sea. En realidad para el uso de este curso la anti-transformada será de mayor utilidad ya que por lo general se trabaja con los circuitos operacionales equivalentes y luego interesa anti-transformar la respuesta para ver su comportamiento en el tiempo.

Para transformar:

L **= laplace**(F) De esta manera guardaremos en la variable L la transformada de Laplace de la expresión simbólica F. Por defecto L va a estar en función de la variable 's' y F tiene que estar en función de 't'.

Ejemplo:

syms a s t w x

laplace(t^5) devuelve 120/s^6

Para anti-transformar:

F=**ilaplace**(L)

Ejemplo:

ilaplace(1/(s-1)) devuelve exp(t)

RESIDUOS EN LOS POLOS PARA EXPANSIÓN EN FRACCIONES SIMPLES. (se puede pasar por alto)

[r,p,k] = **residue**(b,a) ; entrega en el vector r los residuos en los polos correspondientes al vector p los cuales corresponden a la fracción b/a donde b y a son vectores cuyos elementos corresponden a los coeficientes en orden descendiente de 's' del numerador b y el denominador a. En k se guardan los términos directos.

Si no hay polos múltiples:

$$\frac{B(s)}{A(s)} = \frac{R(1)}{s - P(1)} + \frac{R(2)}{s - P(2)} + \ldots + \frac{R(n)}{s - P(n)} + K(s)$$

Si hay polos multiples:

$$\frac{R(j)}{s - P(j)} + \frac{R(j+1)}{(s - P(j))^2} + \ldots + \frac{R(j+m-1)}{(s - P(j))^m}$$

[b,a] = **residue**(r,p,k) utilizando la funcion de esta manera obtenemos los coeficientes b y a á partir de los residuos en los polos.

GRAFICAS DE FUNCIONES

Vamos a ver ahora como graficar funciones en 2 dimensiones.

Ezplot:

La funcion **ezplot** es una funcion de graficas simplificada. Se usa de las siguientes maneras:

ezplot(f,[a,b]) grafica la funcion f=f(x) entre los limites a < x < b. Si no se especifican a y b Matlab utiliza por defecto a=-2*pi b=2*pi.

ezplot(f, [xmin,xmax,ymin,ymax]) siendo f=f(x,y) una funcion definida implicitamente, grafica f(x,y)=0 entre los limites xmin < x < xmax y ymin < y < ymax. Si esto ultimo no es especificado Matlab usa -2*pi < x < 2*pi y -2*pi <y < 2*pi

ezplot(x,y,[tmin,tmax]) Grafica la curva definida parametricamente por x=x(t) e y=y(t) entre

tmin < t < tmax. Si tmin,tmax no son especificados Matlab usa 0 < t < 2*pi.

Nota: f,x e y pueden ser tanto texto entre ' comas' como expresiones simbólicas utilizando variables ya declaradas.

Ejemplos

ezplot('cos(x)')

ezplot('cos(x)', [0, pi])

ezplot('1/y-log(y)+log(-1+y)+x - 1')

ezplot('x^2 - y^2 - 1')

ezplot('x^2 + y^2 - 1',[-1.25,1.25]); axis equal (esta especificación de axis equal hace que se utilizen los mismos rangos de valores para x y para y , en este caso para que se note que es una circunferencia.

ezplot('x^3 + y^3 - 5*x*y + 1/5',[-3,3])

ezplot('x^3 + 2*x^2 - 3*x + 5 - y^2')

ezplot('sin(t)','cos(t)')

ezplot('sin(3*t)*cos(t)','sin(3*t)*sin(t)',[0,pi])

ezplot('t*cos(t)','t*sin(t)',[0,4*pi])

Nota1: si se hubiesen declarado las variables x,y,t haciendo* syms x y t *no hubiese hecho falta poner las 'comas' en las expresiones.

Nota2: si se quiere graficar una función en el mismo eje coordenado que la anterior se debe escribir HOLD luego de haber graficado la primera función. Para desactivar esta opción escribir nuevamente HOLD.

Si quiere graficar en otra ventana, debe escribir FIGURE para que se habilite una nueva ventana de graficación.

Si desea graficar en la misma ventana puede usar el comando subplot(n,m,x) con el cual formara una ventana que tendra n filas con m columnas de ejes coordenados para graficar. La x es un vector que apunta a la fila / columna que desea utilizar para la próxima grafica.

Nota3: si se quiere conocer el valor de la función en un punto especifico de la grafica, hay que utilizar la función GINPUT, que nos permite señalar en la grafica n puntos y nos devuelve un a matriz nx2 con los pares ordenados correspondientes a los puntos que seleccionamos. Escribir GINPUT, seleccionar los puntos y luego tocar enter.

Nota4: Existe una aplicación para Matlab llamada FUNTOOL , que nos permite graficar funciones de

forma muy sencilla y realizar operaciones a modo de calculadora. Tiene ciertas desventajas, pero puede servir en algunos casos. Para ejecutarla escribir FUNTOOL.

Nota5: Si se desea se puede investigar la funcion PLOT(X,Y), la cual grafica los puntos indicados por los pares ordenados correspondientes al vector X versus el vector Y. A veces puede ser de utilidad por que tiene mas opciones en cuanto a el tipo de línea y le permite graficar funciones mas complicadas o de alta frecuencia con mayor exactitud ya que es totalmente configurable.

FUNCIONES DE TRANSFERENCIA- RESPUESTA TEMPORAL -ANALISIS EN FRECUENCIA

Hay varias formas de definir funciones de transferencia. Hay que aclarar que lo que veremos aquí no es compatible con los métodos simbólicos que vimos anteriormente. Estos métodos se utilizan mucho en Sistemas de Control, para los LTI que son sistemas lineales invariantes en el tiempo.

Definición de función de transferencia a través de coeficientes:

sys=**tf**(num,den) genera una función de transferencia con un numerador y un denominador cuyos términos en potencias de 's' tienen coeficientes correspondientes a los vectores num y den respectivamente.num y den denotan los coeficientes en orden descendiente de las potencias de s.

Ejemplo

tf([1 2 3],[5 8 9])

Transfer function:

```
 s^2 + 2 s + 3

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

5 s^2 + 8 s + 9
```

Otra forma de usar la función **tf** es asi:

```
s = tf('s');

H = (s+1)/(s^2+3*s+1)
```

De esta manera disponemos de s como una variable simbólica (aunque no es compatible con los métodos simbólicos vistos anteriormente) y podemos definir funciones de transferencia como la anterior con total facilidad.

Definición de función de transferencia a través de sus ceros, polos y ganancia:

zpk(z,p,k) define una funcion de transferencia cuyos ceros estan dados por el vector z, sus polos por el vector p y su ganancia por el escalar k.

Ejemplo:

zpk([1 9],[-2 0],100)

Zero/pole/gain:

100 (s-1) (s-9)

---

s (s+2)

Nota: si es necesario convertir de **tf** a **zpk** es muy sencillo hacerlo:

Si se quiere pasar de **zpk** a **tf**

F=**zpk**([1 9],[-2 0],100)

Zero/pole/gain:

100 (s-1) (s-9)

---

s (s+2)

D=**tf**(F)

Transfer function:

100 s^2 - 1000 s + 900

---

s^2 + 2 s

y si quiero pasar de **tf** a **zpk**

zpk(D) Zero/pole/gain:

100 (s-9) (s-1)

---

s (s+2)

POLOS Y CEROS DE LAS FUNCIONES DE TRANSFERENCIA:

pole(sys) entrega un vector cuyos elementos son los polos de la f.t. sys

zero(sys) entrega un vector cuyos elementos son los ceros de la f.t. sys

pzmap(sys) entrega un mapa de los ceros y los polos de la función en el plano complejo.

Respuesta temporal

step(sys) : Grafica la respuesta al escalon unitario de el sistema sys, el que debe crearse ya sea con la funcion **tf** o con **zpk**.

impulse(sys) : Grafica la respuesta al impulso de el sistema sys, el que debe crearse ya sea con la funcion **tf** o con **zpk**.

lsim(sys,u,t) : Grafica la respuesta del sistema sys a la entrada u valuando en el vector equi-espaciado t. (nos evita antiransformar)

Ejemplo:

t = 0:0.01:5; u = sin(t); lsim(sys,u,t)

Esto va a graficar la salida del sistema sys cuando en su entrada tiene una función senoidal durante 5 segundos.

Respuesta en frecuencia

bode(sys) : Grafica el diagrama de Bode del sistema sys.

nyquist(sys) : Grafica el diagrama de Nyquist del sistema sys. (tener en cuenta que si hay polos en el origen la

funcion NO CIERRA el diagrama

Nota: Una vez que las respuestas están graficadas, pulsando el boton derecho del mouse sobre la grafica aparecen opciones.

Zoom: para acercarse y/o alejar la vista.

Peak Response: Marca el pico de la grafica.

Grid: Genera una grilla sobre la grafica.

Closedloop response: Da el diagrama de Bode o de Nyquist del sistema a lazo cerrado . (sys/(1+sys))

Recordemos la función **ginput** que nos permitira saber el valor de la respuesta en cualquier punto.

LUGAR DE LAS RAICES

Una de las herramientas mas completas para trabajar en sistemas de control es **RLTOOL.** Esta herramienta nos permite ver el lugar de las raíces de cualquier función de transferencia que tengamos definida en el workspace. Pero además tiene centralizadas todas las herramientas de control como los diagramas de Bode –Nyquist –Nichols-Respuesta al Escalón y al Impulso, etc. Otra gran ventaja es que nos permite importar el modelo con el que estamos trabajando al **simulink .**

Esta herramienta es bastante intuitiva, cuando uno toca con el mouse en algun punto de la grafica, un compensador modificara la ganancia de la función de transferencia, lo que también modificará todas las graficas que se puedan estar ejecutando (ya sea un Bode, o una respuesta temporal). Esto es muy interesante ya que se puede ver la influencia en tiempo y en frecuencia de modificar la ganancia en un sistema, lo cual es muy didáctico para afianzar los conceptos de control. Si se desea ver el efecto de añadir polos y ceros a una función de transferencia se puede hacer gráficamente.

Para comenzar a probarla basta con escribir **rltool** en la pantalla de comandos y se abrirá una nueva ventana donde podrá comenzar a trabajar.

Para importar una función de transferencia selccionar ***file – import model .*** Luego elegir el nombre de la variable que representa a la TF a utilizar.

En lo que sigue se resolverán algunos ejercicios de la GTP de Sistemas de Control I .

RESOLUCIÓN DE EJERCICIOS DE LA GTP

Se aconseja siempre resolver los ejercicios manualmente y luego controlar sus resultados con Matlab.

KUO 3.7 (pagina 3 de la gtp)

Este ejercicio pide encontrar la ganancia según masón de una FT en un grafo de fluencia.

Si lo hacemos debemos obtener:

$M_{15} = Y_5 / Y_1 = 0.938$

$M_{12} = Y_2 / Y_1 = 0.0620$.

¿Cómo hacemos para controlar este resultado?

Mason es simplemente un metodo grafico para resolver un sistema de n ecuaciones con n incógnitas. Por lo tanto lo que se hace es armar el sistema y resolverlo con Matlab.

Tomamos la entrada $Y_1 = 1$ y leyendo el grafo deducimos que:

$Y_2 = Y_1 - Y_5$
$Y_3 = 5*Y_2 - Y_4$
$Y_4 = -2*Y_5 + 10* Y_3$
$Y_5 = Y_4 + 2*Y_6$
$Y_6 = 10*Y_2 - 0.5*Y_6$

Acomodándolo como un sistemas de ecuaciones y escribiendo la matriz ampliada AB tomando 5 ecuaciones con 5 incognitas y los terminos independientes se forma:

```
AB=[-1 0 0 -1 0 -1;1 0 0 1 0 1;5 -1 -1 0 0 0;0 10 -1 -2 0 0;0 0 1 -1 2 0;10 0 0 0 -1.5 0]
```

Para obtener Y_2 - Y_6 calculo la reducida por filas del sistema

```
» rref(AB)
ans =
   1.0000         0         0         0         0    0.0620
        0    1.0000         0         0         0    0.1987
        0         0    1.0000         0         0    0.1113
        0         0         0    1.0000         0    0.9380
        0         0         0         0    1.0000    0.4134
        0         0         0         0         0         0
```

Como vemos Y_2 e Y_5 tienen los valores que nos había dado la resolución por mason.
De la misma manera se pueden resolver con ecuaciones simbólicas ejercicios donde se pida una FT en función de la variable de Laplace S.

KUO 6.2 (pagina 23 de la gtp)

a)

Se pide determinar las constantes de error de un sistema. Calcular las constantes es muy sencillo y no hace falta mayor calculo . Pero lo interesante es ver las gráficas de las respuestas a los sistemas a las distintas entradas y ver realmente el error.

Por comodidad defino la FT de la siguiente manera
s=tf('s')

Transfer function:
s

» G=50/((1+0.1*s)*(1+2*s))

Transfer function:
50

0.2 s^2 + 2.1 s + 1

Error de posición

El $e_{ss}=1/(1+K_p)$ para el caso de un Escalon unitario .

$K_p = 50$

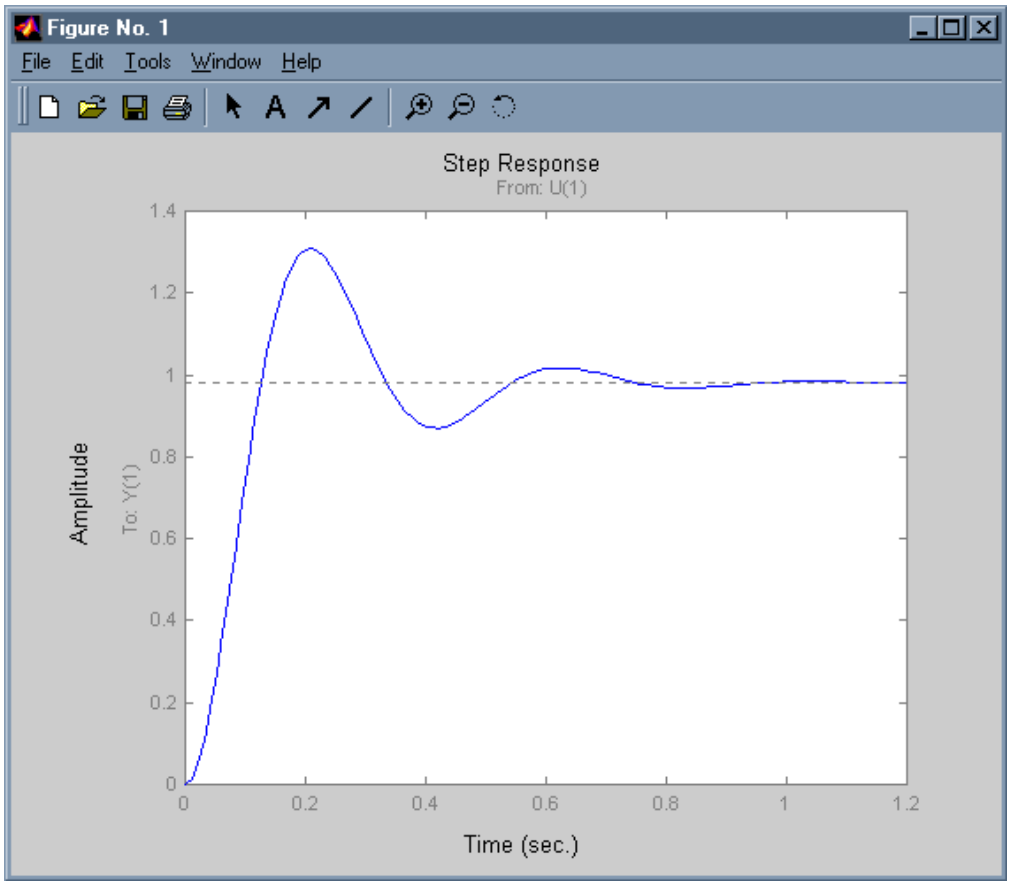
$e_{ss} =0.0167$

Para ver el error de posición, graficaremos la respuesta al escalón del sistema a lazo cerrado.

step(G/(1+G))

Esto nos da la grafica de la respuesta al Escalon del sistema.

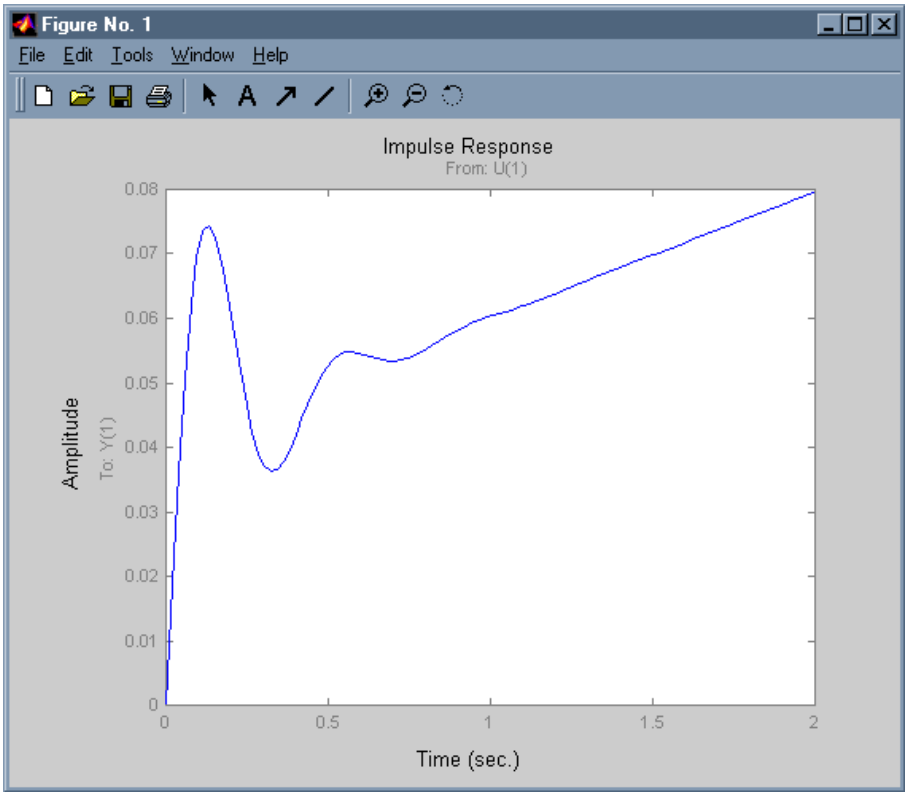
Podemos comprobar con ginput que el valor en estado estacionario esta en el orden de 0.99 y esto demuestra que el error en estado estacionario es del orden del 0.01 como hemos calculado.



Como ya sabemos este es un sitema de tipo 0 por lo tanto el error de posición tiene un valor y los de velocidad y aceleración son infinitos. Esto se puede ver gráficamente asi :

```
» R=1/s^2  % Recta

Transfer function:
 1
---          (transformada de
s^2          una recta)


» E=R/(1+G)

Transfer function:
   0.2 s^2 + 2.1 s + 1
-------------------------
0.2 s^4 + 2.1 s^3 + 51 s^2

» impulse(E,20)
```

Si hacemos esto vemos que el error tiende a infinito en estado estacionario

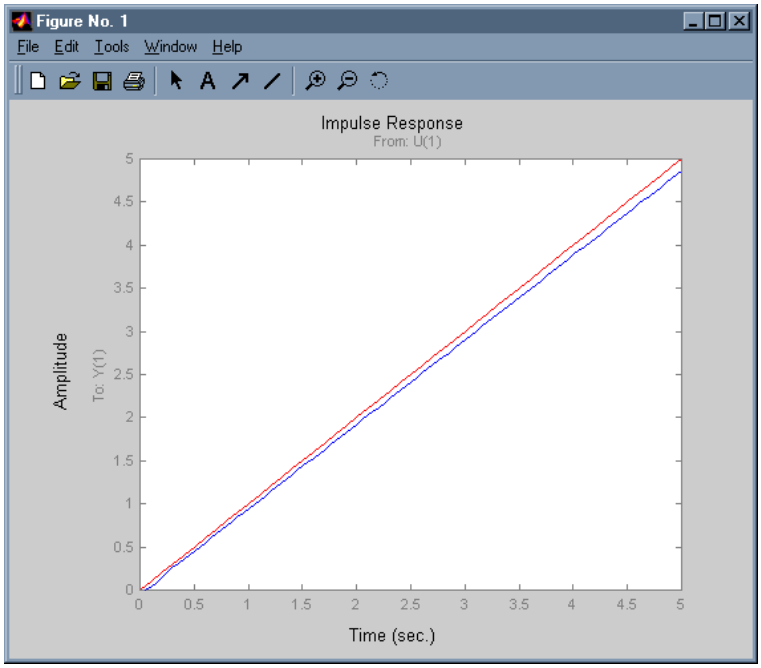


Queda como ejercicio hacer lo mismo para una entrada parabolica.

Mas interesante aun es ver la salida del sistema cuando la entrada es una rampa. Para esto agregamos a lo anterior:

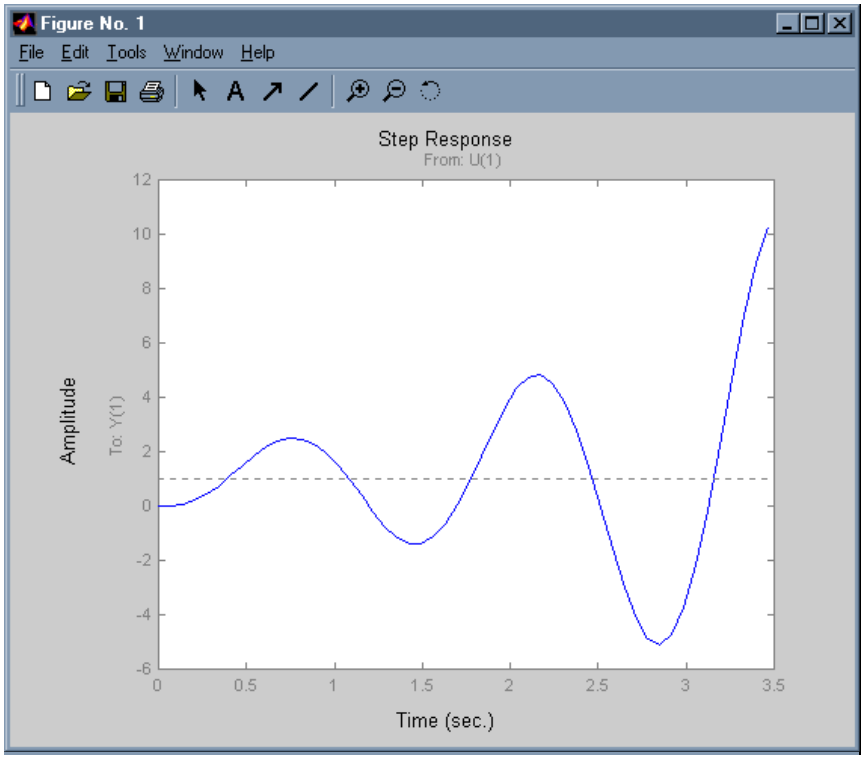
```
C=G*E

Transfer function:
              10 s^2 + 105 s + 50
-----------------------------------------------------
0.04 s^6 + 0.84 s^5 + 14.81 s^4 + 109.2 s^3 + 51 s^2

t=0:0.1:5;
» plot(t,t,'r')  ; % Grafico la entrada en rojo
» hold
Current plot held
» impulse(C,5)  %la salida

otra forma de obtener lo mismo seria con:
»lsim(G/(1+G),t,t) ; %Obtenemos la respuesta de el
                      sistema a lazo cerrado
```

Podemos ver que las rectas tienden a separarse, por Cual el error tiende a crecer.



En la parte b) del ejercicio pide obtener lo mismo pero para es sistema de orden 1. Como ya sabemos el error de posición debe ser 0. El de velocidad cte , y el de aceleración infinito. Pero es muy interesante ver esto gráficamente. Podemos encontrarnos con cosas que no nos esperamos.

```
» G=50/(s*(1+0.1*s)*(1+2*s))

Transfer function:
        50
---------------------
0.2 s^3 + 2.1 s^2 + s

» step(G/(1+G))
```

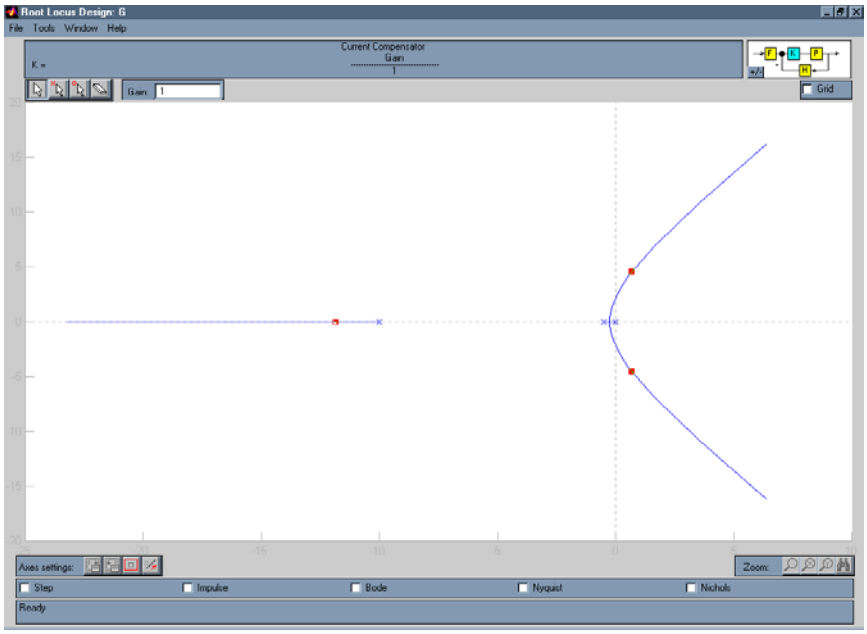


Aqui notamos algo que puede llegar a confundirnos. La respuesta parece inestable, el error tiende a infinito. Esto no era de esperarse ya que el sistema es de orden 1 por lo que el error deberia dar 0.

Lo que sucede es que para esa ganancia el sistema es inestable. Y las definiciones de error valen para sistemas estables.

Veamos el lugar de las raices:

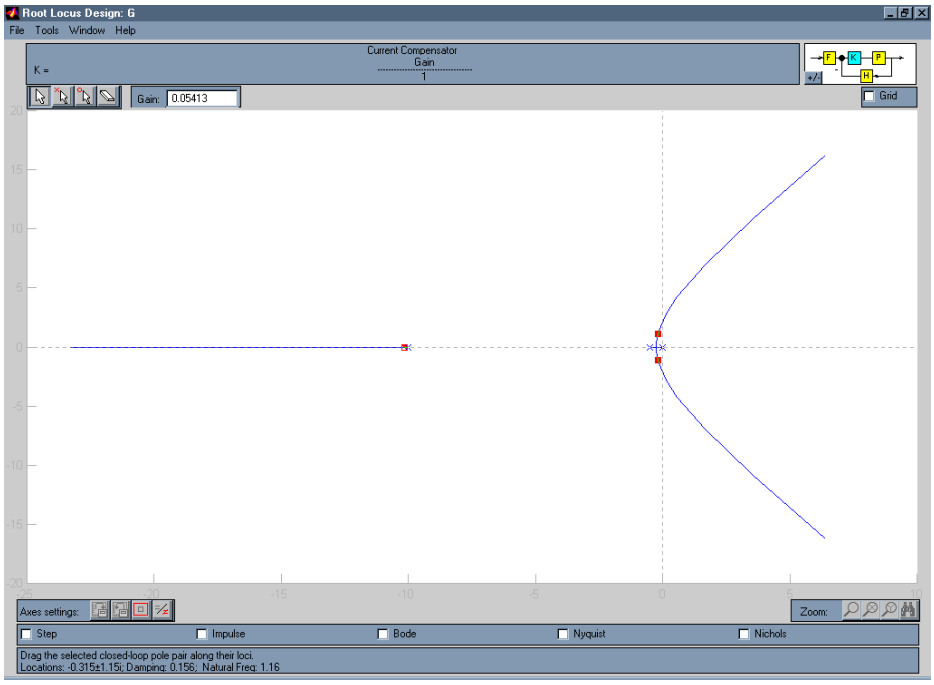
```
» rltool(G)
```



Como podemos ver en los puntos rojos para esta ganancia el sistema tiene polos a parte real positiva siendo esto lo que nos causa inestabilidad, y hace que los error nulo que esperábamos no aparezca.

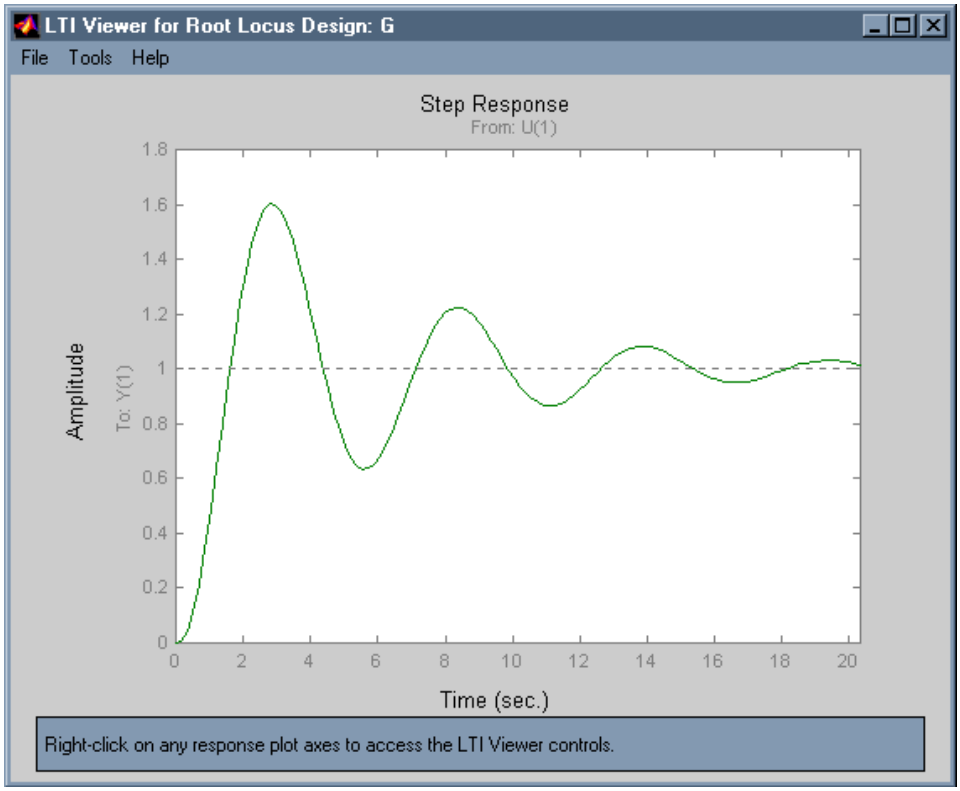
Tocando con el mouse sobre una zona estable esta herramienta compensara la ganancia del sistema y podremos ver la respuesta para el sistema estable.

Al tocar con el mouse donde los puntos rojos, el compensador de ganancia en serie reduce la ganancia multiplicando por un factor de 0.05 con el cual el sistema pasa a ser estable.



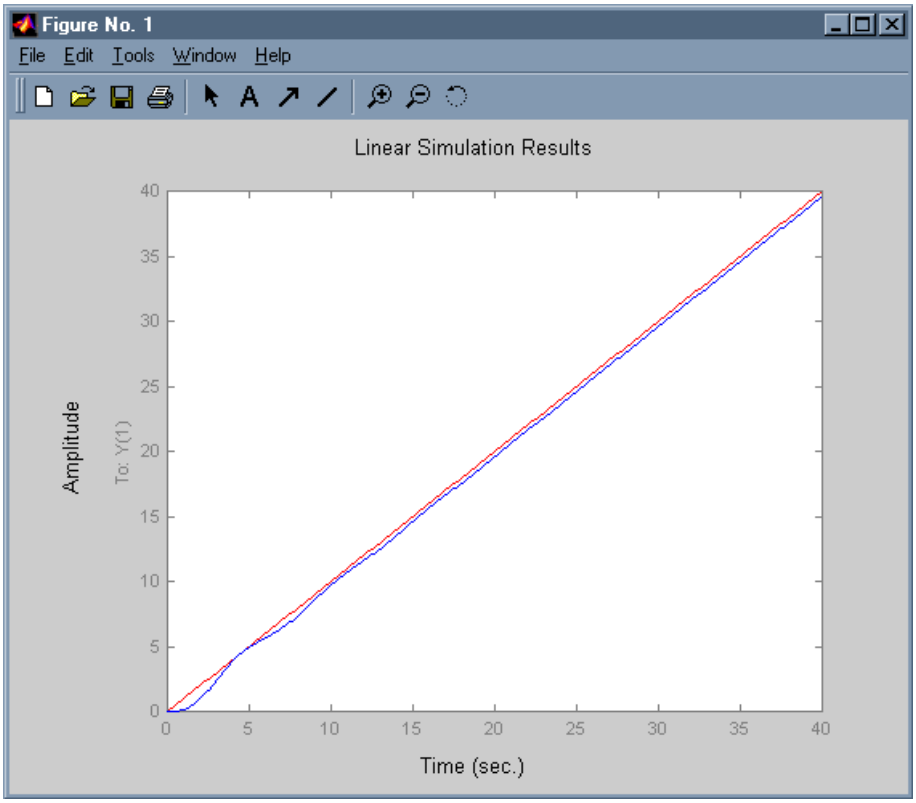
Poniendo el tick con el mouse donde dice sep, veremors ahora la respuesta para el sistema estable.

Podemos ver ahora que la respuesta del sistema es como esperábamos. Con **error nulo para el escalon.**



Bueno ahora veamos la respuesta a una rampa, con el sistema compensado.

```
»G=2.5/(s^2*(1+0.1*s)
*(1+2*s))

»t=0:0.1:40 ;
» plot(t,t,'r') ;
» hold
Current plot held
» lsim((G/(1+G)),t,t) ;
```



Aqui podemos ver que la salida se parece mucho a la entrada y que las rectas tienden a mantenerse juntas, lo que nos indica un **error constante en estado estacionario.**

Si queremos corroborar el valor exacto

Queda como ejercicio ver la respuesta para una parábola. Tener en cta que si se quiere hacer graficar t^2 se debe poner el punto antes(t.^2) para que eleve al cuadrado elemento a elemento el vector t .

Hacer la parte c) del ejercicio .

Con este simple ejercicio se han utilizado las herramientas mas importantes. Se recomienda probar la parte de análisis en frecuencia, relacionándola con el lugar de las raices. Identificar los sistemas de fase minima y no mininma por los diagramas de Bode y Nyquist. Es muy interesante ver las graficas de nichols. Todo esto se hace con solo clicks desde la pantalla del rltool .

Si a alguien le interesa colaborar:
Seria interesante completar este manual con un ejerecicio de compensación por lugar de las raices (uno en adelanto y otro en atraso – los del dorf son los mejorcitos) . Y un ejercicio de compensación por Bode con su respectiva relacion con el diagrama de nyquist . El broche de oro seria hacer un ejercicio que incluya leer una carta de nichols.

Enviar mejoras a **francobatrouni@datafull.com**