REGULACIÓN AUTOMÁTICA
2^o CURSO
INGENIERÍA ELECTRÓNICA INDUSTRIAL

LIBRO DE PRÁCTICAS

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Manuel López Martínez
José Ángel Acosta Rodríguez
Iván Maza Alcañiz
Fernando Dorado Navas

Carlos Vivas Venegas
Ismael Alcalá Torrego
Ángel Rodríguez Castaño
Luis Merino Cabañas
Daniel Jiménez Jiménez
Mercedes Pérez de la Parte

Índice

PRÁCTICAS 1 y 2 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Manuel López Martínez
José Ángel Acosta Rodríguez

Agradecimientos a Manuel Berenguel Soria

Índice general

Capítulo 1

Introducción a MatLab. Parte I

1.1. Introducción

En estas notas se pretende realizar una introducción muy básica a MATLAB, orientada fundamentalmente al estudio de sistemas de control. En líneas generales, MATLAB es un sistema interactivo basado en matrices para cálculos científicos y de ingeniería. Desde el punto de vista del control, MATLAB se puede considerar un entorno matemático de simulación que puede utilizarse para modelar y analizar sistemas. Sirve para estudiar sistemas continuos, discretos, lineales y no lineales.

MATLAB constituye un entorno abierto, para el cual numerosas paquetes específicos adicionales (toolboxes) han sido desarrollados. En el caso que nos ocupa se utilizará fundamentalmente la 'Control System Toolbox' . Estos paquetes específicos adicionales están constituidos por un conjunto de funciones que pueden ser llamadas desde el programa y mediante las cuales se pueden realizar multitud de análisis.

Las notas se centrarán fundamentalmente en aquellos aspectos y funciones que más interés tengan desde el punto de vista de control, instando al lector a que busque en el manual de usuario cualquier información adicional que desee. Para el desarrollo de las mismas se ha utilizado tanto la experiencia programando en MATLAB de los autores, como una serie de referencias básicas.

El núcleo fundamental de MATLAB se encuentra en los subdirectorios BIN y MATLAB. En BIN se encuentran los programas ejecutables. El subdirectorio MATLAB contiene los ficheros .m (aunque serán explicados posteriormente, comentamos brevemente que consisten en ficheros escritos a base de comandos de MATLAB y que realizan una función determinada), que contienen las funciones básicas para el funcionamiento de MATLAB. En este

sentido, es necesario comentar que MATLAB cuenta con dos tipos básicos de funciones:

Las llamadas ***built-in functions***: Son funciones que MATLAB tiene incorporadas internamente y por tanto no son accesibles al usuario.

Funciones ***m functions***: Son funciones cuyo código es accesible. Las que se encuentran en el subdirectorio MATLAB son las básicas para el funcionamiento del sistema.

Las toolboxes se suelen instalar en forma de subdirectorios en el disco duro, colgando del subdirectorio TOOLBOX(en la versión WINDOWS). En ellos se encuentran también funciones .m orientadas al control de sistemas. Además, se pueden incorporar otros toolboxes (SIGNAL PROCESSING, ROBUST CONTROL, etc), e incluso funciones propias del usuario.

En el caso de las versiones para WINDOWS, el arranque del programa se realiza 'pinchando' con el ratón en el icono correspondiente. Para obtener información adicional se aconseja mirar el manual de usuario.

1.2. Entorno

Una vez arrancado MATLAB, aparece el prompt o línea de comandos del sistema (≫). Este es el momento de comentar la existencia del comando más famoso de cualquier aplicación: **help**. Introduciendo este comando aparecerán todas las citadas ***built-in functions***, las contenidas en el subdirectorio MATLAB y todas aquellas contenidas en los subdirectorios incluidos en el PATH(ver cuadro1.1).

Para obtener información sobre cualquiera de las funciones se introduce **help nombre-función**. Ejemplo: help cos (cos es una función que calcula el coseno de un número). Una cuestión importante a tener en cuenta es que MATLAB distingue entre **mayúsculas** y **minúsculas**. En este sentido, los nombres de función se introducirán en minúsculas. El comando **demo** permite obtener una demostración de las 'capacidades' del sistema.

1.2.1. Funciones y símbolos

- Si se quiere guardar toda la sesión en un archivo (comandos introducidos y resultados), basta usar el comando **diary** nombre-archivo y se guardará la sesión en un archivo llamado diary. Cuando no se quiera seguir almacenando la información se introducirá **diary off** .

- El símbolo % sirve para poner comentarios en los programas (todo lo escrito desde ese símbolo hasta el final de la línea no se ejecutará).

c:\matlab	Establece los parámetros de la sesión MATLAB
matlab\general	Comandos de propósito general
matlab\ops	Operadores y caracteres especiales
matlab\lang	Construcción del lenguaje y debugging
matlab\elmat	Matrices elementales y manipulación de matrices
matlab\specmat	Matrices especiales
matlab\elfun	Funciones matemáticas elementales
matlab\specfun	Funciones matemáticas especiales
matlab\matfun	Funciones matriciales - álgebra lineal numérica
matlab\datafun	Análisis de datos y funciones de transformada Fourier
matlab\polyfun	Funciones polinomiales y de interpolación
matlab\funfun	Funciones de funciones - métodos numéricos no lineales
matlab\sparfun	Funciones para matrices dispersas
matlab\plotxy	Gráficos en dos dimensiones
matlab\plotxyz	Gráficos en tres dimensiones
matlab\graphics	Funciones gráficas de propósito general
matlab\color	Funciones para control de color, brillo y contraste
matlab\sounds	Funciones para procesamiento de sonido
matlab\strfun	Funciones de cadenas de caracteres
matlab\iofun	Funciones de Entrada-Salida de bajo nivel
matlab\demos	La Expo de MATLAB y otras demostraciones
simulink\simulink	Análisis de modelos en SIMULINK y funciones de construcción.
simulink\blocks	Librería de Bloques de SIMULINK
simulink\simdemos	Demostraciones y ejemplos de SIMULINK
toolbox\control	Control System Toolbox
toolbox\local	Librería de funciones locales

Cuadro 1.1: Listado del comando help

- Si lo que se desea es almacenar todas las variables de memoria (y sus valores actuales) en un fichero, se usa el comando **save** nombre-fichero. Esto crea un fichero con el nombre introducido y con extensión .MAT. Si no se pone nombre del fichero crea uno llamado MATLAB.MAT. En caso que se desee guardar en un fichero con formato ASCII, se introducirá en el comando un modificador **save -ascii** nombre fichero ascii. Si sólo se quieren guardar una serie de variables se introducirá **save** nombre-fichero nombre-variables separadas por espacios.
- Para recuperar los ficheros generados con el comando **save** se utilizará **load** nombre-fichero.
- El comando **what** muestra los ficheros **.m** que se encuentran en el disco duro en el subdirectorio desde el cual se haya invocado a MATLAB.
- **dir** muestra todos los ficheros contenidos en el subdirectorio actual.
- Con el comando **delete** se puede borrar cualquier archivo del disco duro.
- **chdir** permite cambiar de directorio.
- El comando **type** permite ver el contenido de cualquier archivo en formato ASCII.
- Para borrar alguna variable de memoria se utiliza **clear** nombre-variables separadas por espacios.
- Para parar la ejecución de un comando se usa **Ctrl c**.
- Para finalizar la ejecución de MatLab se escribe **quit** o **exit**.

1.3. Variables y operadores

Los operadores básicos que usa Matlab son:

- **Aritméticos**:
  - Suma: +
  - Resta: -
  - Multiplicación: *
  - División : /

- Potencia: $\wedge$

- **Lógicos y Relacionales**: Permiten la comparación de escalares (o de matrices elemento a elemento). Si el resultado de la comparación es verdadero, devuelven un 1, en caso contrario devuelven un 0. Los operadores elementales son:

  - $<$ menor que
  - $<=$ menor o igual
  - $==$ igual
  - $>$ mayor que
  - $>=$ mayor o igual
  - $\sim=$ no igual

  Es importante no dejar espacios entre los operadores formados por dos símbolos. Para datos complejos se compara ($==$ y $\sim=$) tanto la parte real como la imaginaria.

  Por otro lado, se pueden usar variables de tipo carácter, cadena de caracteres, booleanas, bytes, enteros y flotantes.

  Para asignar un valor a una variable se escribe el nombre de la variable, el símbolo $=$, y el valor de la misma, o bien el nombre de otra variable previamente inicializada.

  Ejemplo:

```
>> a=100;
>> b=2;
>> c=a
c =
   100
```

  Si al final de la introducción del comando no se pone punto y coma (**;**), aparece el resultado explícitamente en pantalla. En caso contrario se ejecuta pero no muestra el resultado, almacenándolo en la variable a la que se asigna o si no se asigna se guarda en una variable de entorno llamada **ans**.

  De igual modo podemos realizar operaciones entre variables, del ejemplo anterior vamos a multiplicar a y b.

  Ejemplo:

```
>> d=a*b
d =
    200
```

MatLab tiene predefinidas una serie de **variables y constantes especiales**

- ans : respuesta cuando no se asigna expresión.
- eps : precisión mínima de la máquina.
- pi : π
- i, j : $\sqrt{-1}$
- inf: ∞
- NaN: Not a number.
- clock: Reloj.
- date : Fecha.
- flops: Número de operaciones en coma flotante.

Las variables a las que se asignan resultados, así como las variables de entorno, se almacenan en el 'espacio de trabajo'(workspace).

El comando **who** muestra las variables existentes en el entorno generadas por el usuario (pero no las variables especiales). El formato de salida puede modificarse usando **format** (short, long etc).

1.4. Vectores y Polinomios

Los vectores se introducen entre corchetes, y sus elementos están separados por espacios o comas.

Ejemplo:

```
>>v=[77 69 11 88]
v =
    77     69     11     88
```

Los elementos de los vectores se referencian usando índices entre paréntesis. Los índices en MatLab empiezan en 1.

Ejemplo: Para el elemento 2 del vector v

```
>>v(2)
ans =
    69
```

Se pueden referenciar varios elementos a la vez usando el operador **:**.
Ejemplo:

```
>>v(2:3)
ans =
    69 11
```

Los **polinomios** se representan por **vectores**, conteniendo los coeficientes del polinomio en orden descendente. Por ejemplo, el polinomio s^3+2s^2+3s+4 se representa:

```
p=[ 1 2 3 4] ;
```

Mediante la función **roots** se pueden encontrar las raíces de esa ecuación.

```
roots(p)
```

Del mismo modo, se puede calcular un polinomio a partir de sus raíces usando la función **poly**.

```
p2=poly([-1 -2]);
```

Si el argumento de entrada a **poly** es una matriz, devuelve el polinomio característico de la matriz ($det|\lambda I - A|$) como un vector fila.

Un polinomio puede ser evaluado en un punto determinado usando **polyval**.

```
ps=polyval(p,s)
```

donde **p** es el polinomio y **s** es el punto donde va a ser evaluado. Por ejemplo:

```
p2=[ 1 3 2] ; a=[ 1 2; 3 4] ; polyval(p2,a)
```

si se introduce en vez de un valor un vector o una matriz, la evaluación se hace elemento a elemento.

Los polinomios se pueden multiplicar y dividir usando las funciones **conv** y **deconv** respectivamente.

Ejemplo:

```
>> A=[1 -1]; % x-1
>> B=[1  1]; % x+1
>> C= conv(A,B)  %  x^2-1
C =
     1     0    -1
>> polyval(C,1)
ans =
     0
```

1.5. Matrices

El elemento básico en MATLAB es una matriz compleja de doble precisión, de forma que abarca realmente todo tipo de datos (desde números reales hasta complejos) y de estructuras de datos (escalares, vectores y matrices). Así por ejemplo, se pueden introducir:

A=[1 0 2; 2 2 0; 0 0 1]

A partir de esta representación se pueden comentar varias cosas:

- Para separar filas se usa **;** o bien al introducirlas se pulsa **return**.
- Para transponer matrices se usa el apóstrofe **'**.
- Los elementos de vectores y matrices pueden ser reales, complejos e incluso expresiones.
- Si se está introduciendo un comando o conjunto de ellos cuya sintaxis es muy larga, se puede continuar en la siguiente línea introduciendo al final de la actual tres puntos seguidos (**. . .**).
- Otras formas de introducir matrices:
  - Lista explícita de elementos.
  - Generándola mediante funciones y declaraciones.
  - Creándola en un archivo **.m** (matrices .m).
  - Cargándola de un archivo de datos externo (ficheros de datos ASCII y ficheros con formato **.mat**).

El comando **size** devuelve el número de filas y columnas de una matriz y **length** la mayor dimensión.

Ejemplo:

```
>> A=[ 1 0 2; 2 2 0; 0 0 1]

A =

     1     0     2
     2     2     0
     0     0     1

>> size(A)

ans =

     3     3
```

Los elementos de una matriz se referencian de la forma **A(i,j)** donde **i** y **j** son los índices del elemento correspondiente. En este punto es importante comentar uno de los elementos más potentes de MATLAB, que es el símbolo **:** , que permite referenciar varios elementos de una matriz, así por ejemplo:

A(1, 2: 3) daría como resultado los elementos de las columnas 2 y 3 pertenecientes a la primera fila.

A(: , 2) daría como resultado todos los elementos pertenecientes a la segunda columna.

1.5.1. Operaciones con matrices

Las operaciones comunes con matrices son:

- Suma: +
- Resta: -
- Multiplicación: *
- División derecha: / (x=b/A es la solución de x*A=b).
- División izquierda: \ (x=A\b es la solución de A*x=b).
- Potencia: $\wedge$
- Traspuesta: '

Las mismas operaciones se pueden realizar elemento por elemento anteponiendo un punto . a cualquiera de los operandos anteriores (ejemplo: Para hacer el producto de los elementos (i,j) de las matrices A y B, se haría A.*B).

Además de las operaciones anteriores existen las trigonométricas estándar (sin, cos , tan, asin, acos , atan, atan2), funciones hiperbólicas (sinh, cosh, tanh, asinh, acosh, atanh) , funciones trascendentales (log, log10, exp, sqrt) y funciones normales de manipulación matricial:

- det : determinante.
- inv: inversa.
- eig: Obtención de autovalores.
- rank: rango de la matriz.
- norm: norma.
- trace: traza de la matriz.
- real : parte real.
- imag: parte imaginaria.
- abs : valor absoluto.
- conj: conjugada.

Ejemplo:

```
>> A

A =

     1     0     2
     2     2     0
     0     0     1

>> det(A)   % Determinante de la matriz A

ans =

     2
```

```
>>  trace(A) % Traza de la matriz A

ans =

     4

>> inv(A) % Inversa de la matriz A

ans =

    1.0000         0   -2.0000
   -1.0000    0.5000    2.0000
         0         0    1.0000

>> B=rand(3) % Matriz 3X3 de elementos aleatorios entre 0 y 1

B =

    0.4447    0.9218    0.4057
    0.6154    0.7382    0.9355
    0.7919    0.1763    0.9169

>> D=A*B

D =

    2.0286    1.2743    2.2395
    2.1203    3.3200    2.6824
    0.7919    0.1763    0.9169
```

1.6. Funciones Avanzadas

En esta sección simplemente comentaremos que existen una serie de funciones, muy útiles en problemas de integración numérica (**quad**, **quad8**), solución de ecuaciones diferenciales, importantes cuando se estudian los sistemas dinámicos (**ode23**, **ode45**), ecuaciones no lineales e interpolación (**fmin**, **fsolve** etc.), interpolación (**spline**), funciones orientadas al análisis de datos, **min**, **max**, **mean**, **median**, **std**,**sum**, **prod**, **cumsum**, **cumprod** etc.

1.7. Ficheros Scripts

MATLAB puede ejecutar programas que se encuentren almacenados en ficheros ASCII que se encuentren en alguno de los subdirectorios indicados en el PATH o bien en el subdirectorio de trabajo actual y tengan además extensión .m.

Los Scripts son ficheros .m en los que se ponen secuencialmente comandos de MATLAB que se ejecutan en ese orden al introducir el nombre del fichero .m (sin extensión). Operan globalmente con los datos que se encuentran en la memoria, es decir, las variables usadas son variables globales, un cambio en el valor de la variable en el Script actúa sobre la variable en memoria del mismo nombre.

A continuación se va a mostrar un ejemplo de Script. Se muestra el código del fichero .m y se presentan los resultados obtenidos en MatLab tras ejecutar el script. Para ello basta escribir en línea de comando el nombre del fichero excluyendo la extensión.

Ejemplo: Fichero .m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%   Ejemplo de Script:  prueba.m   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Operaciones con Matrices
%
%

A=[1 2 3; 4 5 6] B=[1 2; 3 4 ; 5 6]
C=A*B       % Producto de A por B
T=inv(C)^2  % Cuadrado de la inversa de C
Tt=T'       % Traspuesta de T
```

Ejemplo: Ejecución del Script

```
>> prueba

A =

     1     2     3
     4     5     6
```

```
B =

     1     2
     3     4
     5     6


C =

    22    28
    49    64


T =

    4.2191   -1.8580
   -3.2515    1.4321


Tt =

    4.2191   -3.2515
   -1.8580    1.4321
```

Capítulo 2

Introducción a MatLab.Parte II

En esta segunda práctica se van a tratar más herramientas de Matlab. Entre ellas se verán generación de gráficos y funciones en Matlab para los que será necesario estudiar el control de flujo de programas.

2.1. Gráficos

Para dibujar gráficos es preciso generar la tabla de valores correspondiente. Para ello MatLab dispone de dos funciones, linspace y logspace, que permiten generar vectores de puntos espaciados de forma lineal o logarítmica respectivamente.

- **x=linspace(a,b,n)** Genera un vector de **n** puntos desde **a** hasta **b**, cuyos componentes están espaciados linealmente.
- **x=logspace(a,b,n)** Genera un vector de **n** puntos desde **a** hasta **b**, cuyos componentes están espaciados logarítmicamente.

Para hacer gráficos en dos dimensiones (2D) se utiliza la función **plot** cuya sintaxis básica es:

- **plot(X,Y)** dibuja el vector Y frente al vector X. Se permite dibujar varios gráficos en una misma figura. Para ello la sintaxis es plot(X1,Y1,X2,Y2,...). Si se desea diferenciar las distintas gráficas, se pueden cambiar las propiedades de representación de las mismas, es decir, se puede especificar el color y tipo de línea. Esto se puede ver en la figura 2.1

Para poner título tanto a la figura como a los ejes coordenados existen una serie de funciones:

- **title('Título de la figura')**.
- **xlabel('Título del eje x')**.
- **ylabel('Título del eje y')**.
- **legend('gráfica1','gráfica2')**: Escribe una leyenda asociando un nombre a cada gráfica.
- **grid**: genera una rejilla sobre la gráfica para facilitar la interpretación de la misma.

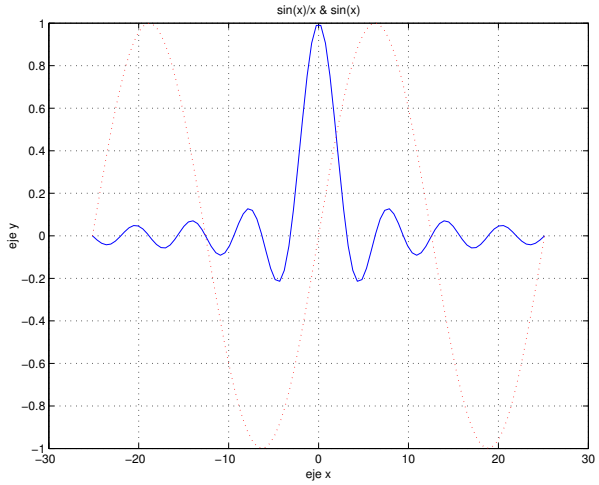


Figura 2.1: Ejemplo de función plot

Ejemplo: Script para generar una figura 2D

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                                                     %
%%         Funcion que genera un grafico de 2D                         %
%%                                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x = linspace(-8*pi,8*pi,100);
y = sin(x)./x;      % ./ representa division elemento
                    % a elemento de dos vectores

figure; plot(x,y,'b',x,sin(x/4),'r:'), title('sin(x)/x & sin(x)'),
xlabel('eje x'),ylabel('eje y'), grid;
```

Por otro lado, Matlab permite realizar gráficas en tres dimensiones (3D). Las gráficas en 3D se definen mediante vectores o matrices de datos en función de que se dibuje una línea o una superficie.

Usaremos los siguientes comandos, además de los previamente comentados para gráficas 2D:

- **plot3(X,Y,Z)** Permite dibujar curvas en 3D. Dibuja el vector Z frente a los vectores X e Y. Se permite dibujar varios gráficos en una misma figura. Para ello la sintaxis es plot3(X1,Y1,Z1,X2,Y2,Z2...). Si se desea diferenciar las distintas gráficas, se pueden cambiar las propiedades de representación de las mismas, es decir, se puede especificar el color y tipo de línea. Esto se puede ver en la figura 2.2

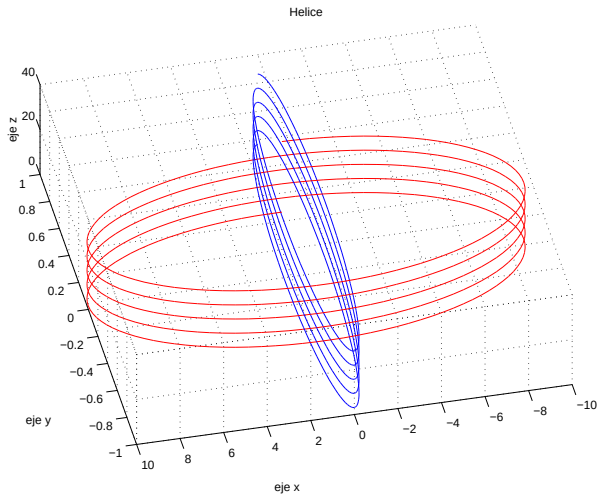


Figura 2.2: Ejemplo de función plot3

Ejemplo: Script para generar una curva 3D

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                                                     %
%%         Funcion que genera una curva de 3D                          %
%%                                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

t = 0:pi/50:10*pi; % : alternativa al comando linspace

figure; plot3(sin(t),cos(t),t,'b',10*sin(t),cos(t)/2,t,'r');
```

```
title('Helice'), xlabel('eje x'),
 ylabel('eje y'), zlabel('ejez'),grid;
```

- **[X,Y]=meshgrid(x,y)**: Genera una rejilla de puntos a partir de los vectores X e Y.
- **mesh(x,y,z), surf(x,y,z)** Para dibujar superficies en 3D. z es el valor que toma la función z=f(X,Y) en el punto de la rejilla X,Y.

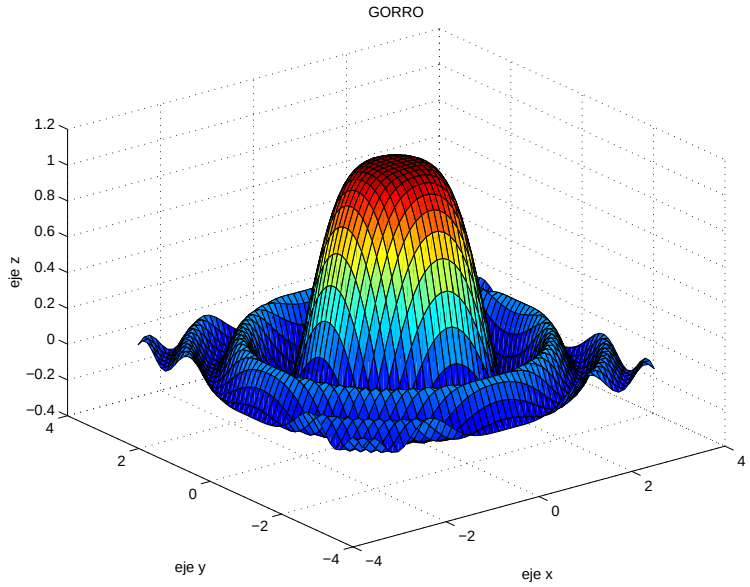


Figura 2.3: Ejemplo de función surf

Ejemplo: Script para generar una superficie 3D

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                                                      %
%%      Funcion que genera una superficie de 3D                         %
%%                                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x=linspace(-pi,pi,50);
y=linspace(-pi,pi,50);
[X,Y]=meshgrid(x,y);

z=sin(1.1*(X.^2+Y.^2))./(X.^2+Y.^2);
```

```
figure; surf(x,y,z);

title('GORRO'), xlabel('eje x'), ylabel('eje y'), zlabel('eje z'),
grid;
```

2.2. Programando en MatLab

MATLAB permite programar una serie de elementos controladores de flujo. La sintaxis es muy parecida a la de cualquier lenguaje de programación. Todos estos operadores se pueden usar en un fichero **.m**.

2.2.1. Bucles y estructuras condicionales

Veremos algunos de los comandos de control de flujo de programas en MATLAB: **for**, **while**, **if-else**.

for:

Sintaxis:

```
for variable = expresion
 hacer algo;
end
```

La expresión es un vector, una matriz o cualquier comando de MATLAB que produzca como salida un vector o una matriz. La ejecución se realiza una vez por cada elemento del vector o de una columna de la matriz.

Ejemplo, donde la variable i pasa por los valores 10, 9, ..., 1:

```
for i=10:-1:1
 kk(11-i)=i ;
end
```

Como se observa, los bucles (y las estructuras condicionales) se terminan con end. Es importante evitar en la medida de lo posible el uso de bucles en MATLAB, ya que consumen mucho tiempo, pudiéndose en muchos casos realizar las mismas operaciones de una forma más eficiente. Los siguientes ejemplos calculan logaritmos de números desde 1 a 10.000. Se hará de diferentes maneras para comparar. Se utilizan los comandos **clock** y **etime** para calcular el tiempo consumido en las operaciones.

- **clock**: Hora actual.
- **etime**: Devuelve el tiempo en segundos que ha transcurrido entre dos instantes de tiempo.

Método 1:

```
t1=clock;
for i=1: 10000,
    a(i)=log(i);
end;
e1=etime(clock,t1)
```

Método 2:

```
t1=clock; ind=[ 1: 10000];
a=zeros(1,10000);
a=log(ind);
e2=etime(clock,t1)
```

El tiempo de computación para el método 2 es del orden de 50 a 100 veces menor que para el método 1, dependiendo de la máquina.

Las causas de la disminución importante de tiempos es que en el primer método, MATLAB tiene que recalcular la dimensión del vector en cada pasada por el bucle (importancia de las inicializaciones), y además usa bucles for , que como se ha indicado, consumen mucho tiempo. Esto por supuesto no quiere decir que no deban usarse, pues habrá ocasiones en que no haya más remedio, pero siempre que haya una forma alternativa de hacerlo, ésta será preferible al uso de bucles.

while:

Permite bucles condicionales. Su sintaxis es:

```
while expresion,
   hacer algo,
end;
```

La expresión es de la forma X operador Y, donde X e Y son escalares o expresiones que devuelven escalares y los operadores suelen ser operadores relacionales. En el siguiente ejemplo se busca una matriz aleatoria con parte real de autovalores negativa:

```
rand(normal);
a=rand(2);
while max(real(eig(a)))>=0,
    a=rand(2);
end;
eig(a)
```

if, **else**, **elseif**:

La sintaxis es la siguiente:

```
if expresion1,
    hace algo,
    hace otras cosas,
elseif expresion2,
    hace algo,
    hace otras cosas,
else
    hace algo,
end
```

else y elseif son opcionales, no así end que es obligatorio para acabar la instrucción. Se puede usar **break** para salir de un bucle si se cumple la condición incluida en el if.

2.3. Funciones

Además de los script-files, hay otro tipo de ficheros .m: los **ficheros de funciones**.

A diferencia de los scripts anteriores, se le pueden pasar argumentos y pueden devolver resultados. Por tanto utilizan variables que se pasan por valor. La mayoría de los ficheros contenidos en las diferentes 'toolboxes' son funciones. La sintaxis de todas las funciones almacenadas en ficheros .m es la siguiente:

```
function[sal1,sal2,...] =nombre_fichero(ent1,ent2,...)
% Comentarios adicionales para el help
comandos de MATLAB
```

Una función puede tener múltiples entradas y salidas.
Ejemplo:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                                  %
%%          Funcion que calcula la media y          %
%%                                                  %
%%          la varianza de un vector de 3D          %
%%                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  [media,varianza] = funcion(vector)
%
 function [media,varianza] = funcion(x)
           n = length(x);
           media = med(x,n);
           varianza = sum((x-med(x,n)).^2)/n;

           %-------------------------
           function media = med(x,n)
           %subfuncion
           media = sum(x)/n;
```

Para calcular la media y la varianza del vector [6,4] se debe escribir lo siguiente:

```
>>[m,v]=funcion([6,4])

m =

     5


v =

     1
```

PRÁCTICA 3 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Iván Maza Alcañiz
José Ángel Acosta Rodríguez

Agradecimientos a Manuel Berenguel Soria

Capítulo 1

Ejemplo

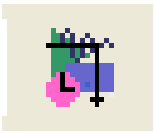
1.1. Modelado de un sistema dinámico

En este ejemplo se realizará el modelado de un sistema dinámico muy sencillo. Se modelará el movimiento de una masa sobre una superficie rugosa, sobre la que se le aplica una fuerza. El sistema a modelar posee una entrada u, que se corresponde con la fuerza aplicada, y una salida x que será la posición de la masa. El modelo del sistema dinámico se puede expresar mediante las ecuaciones de Newton:

$$m\ddot{x} + c\dot{x} = F \tag{1.1}$$

m : Masa del cuerpo (Kg)
c : Coeficiente de fricción del cuerpo sobre la superficie
F : Fuerza aplicada (N)

Queremos hacer un modelo en con la herramienta "Simulink"para el sistema propuesto. Primero ejecutamos la herramienta "Simulink"desde la ventana de comandos de Matlab haciendo 'click' en el icono correspondiente



Saldrá por pantalla una ventana gráfica, como la de la Fig. 1.1, que contiene todas las librerías que el entorno de "Simulink"bajo Matlab soporta.

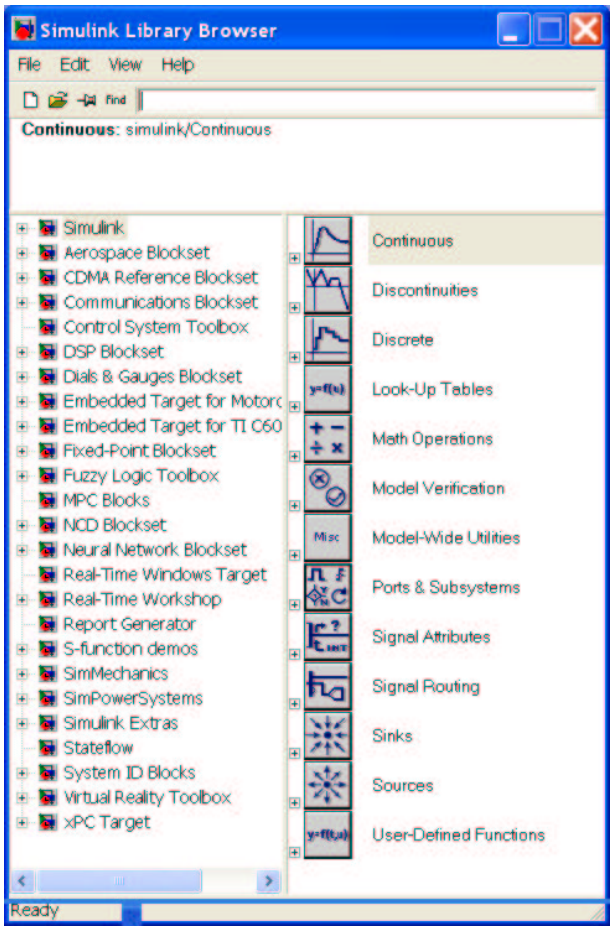


Figura 1.1: Librerías del entorno Simulink

Para este sencillo ejemplo sólo necesitaremos la librería básica de "Simulink", por tanto expandimos el menu simulink en la ventana anterior, quedando como aparece en la Fig. 1.2

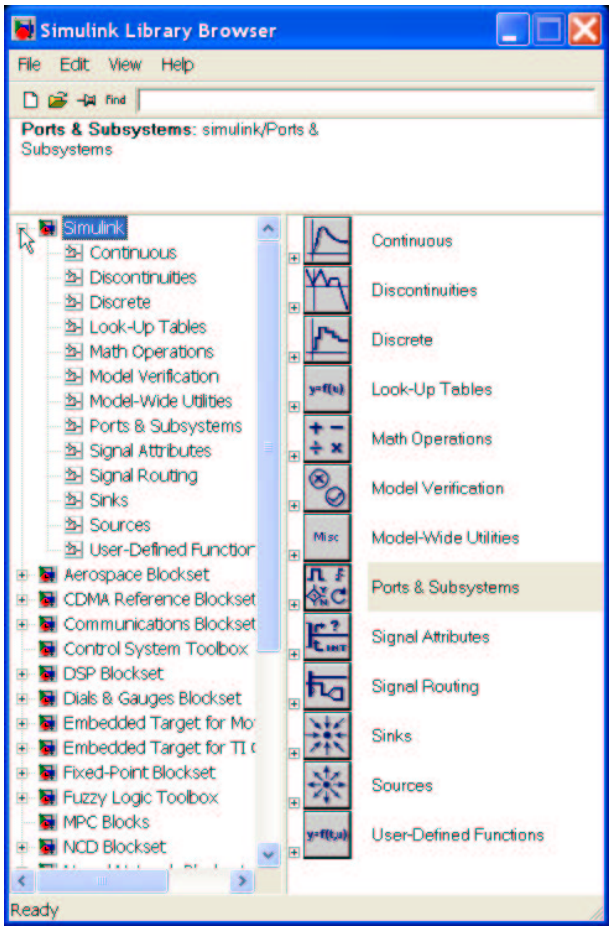


Figura 1.2: Librería base de Simulink

Esta ventana está dividida en dos partes. La de la derecha es la correspondiente a las librerías y la de la derecha es el contenido de la librería seleccionada.

Elegimos un nuevo fichero donde guardaremos el modelo: seleccionamos en el menu File → New → Model. Tendremos la situación de la Fig. 1.3

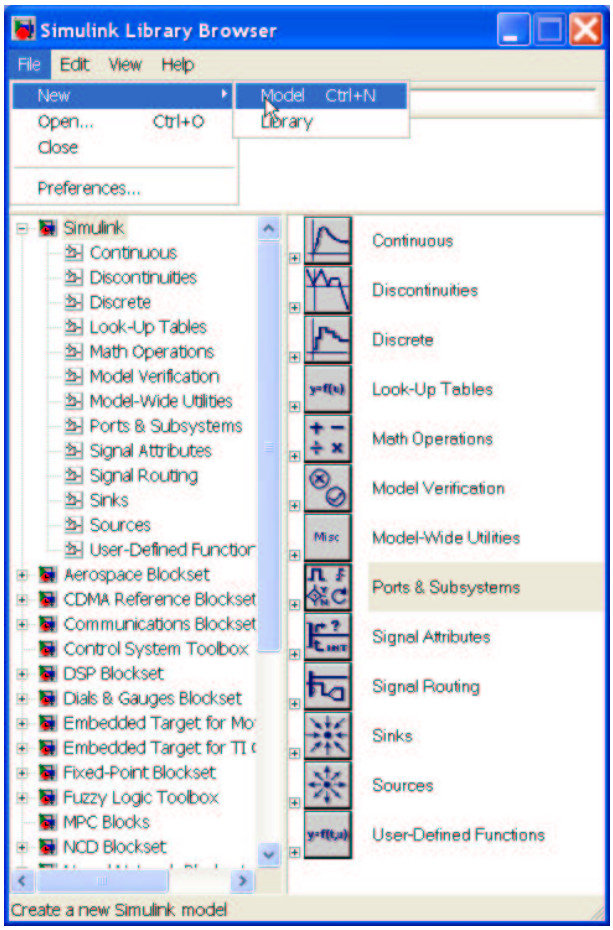


Figura 1.3: Apertura de un nuevo fichero modelo

Se abrirá una ventana en blanco donde crearemos el modelo. La situación debe ser ahora la de la Fig. 1.4

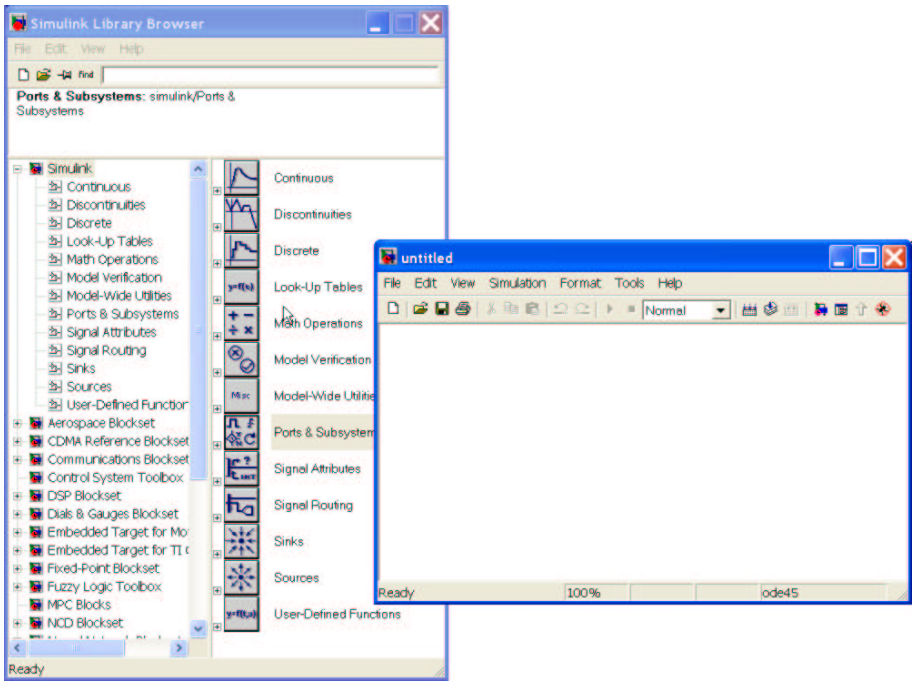


Figura 1.4: Apertura de un nuevo fichero modelo

En esta nueva ventana que aún no hemos dado nombre ('untitled') desarrollaremos el ejemplo. Lo primero que hacemos es darle un nombre adecuado. Para ello, en el menu File de la nueva ventana elegimos File → Save. Nos situamos en el directorio adecuado a través del menu desplegable, ponemos un nombre al archivo, por ejemplo "masaz guardamos el modelo. Ya tenemos un archivo donde crear el modelo. La extensión por defecto de los archivos de modelo es *.mdl.

Empezamos a crear el modelo dado por la ecuación (1.2). Para ello es necesario hacer alguna modificación en la ecuación (1.2). Despejando de la ecuación (1.2) la aceleración del cuerpo se obtiene:

$$\ddot{x} = -\frac{c}{m}\dot{x} + \frac{F}{m} \tag{1.2}$$

Como puede verse necesitaremos varios tipos de bloques. Elegimos estos bloques de la ventana de la derecha de la librería (Fig. 1.2). El primero que seleccionamos el que definirá la fuerza aplicada a la masa, lo haremos mediante una constante. Seleccionamos 'Sources' y en la derecha seleccionamos

el bloque de 'Constant'. Ahora lo arrastramos hacia la ventana de nuestro modelo con el botón izquierdo del ratón pulsado. Hacemos 'click' en el la etiqueta del nombre del bloque de constante y le damos su nombre, por ejemplo F. La situación debe ser la de la Fig. 1.5

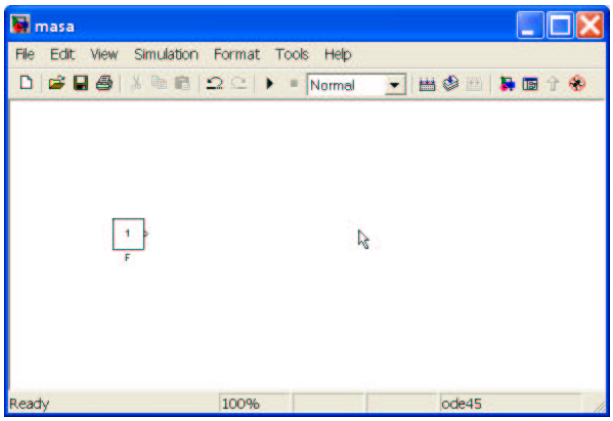


Figura 1.5: Construyendo el modelo

Observando la ecuación (1.2), puede verse que se necesita hacer las operaciones de sumar y dividir. Para ello seleccionamos 'Math Operations' en la ventana de la librería y escogemos del mismo modo que antes los bloques de 'Sum' y 'Gain'. Para describir la ecuación diferencial se necesitará además el bloque integrador 'Integrator' en la librería 'Continuous'. Ya se está en disposición de describir la ecuación (1.2) utilizando bloques. Debemos unir los bloques de forma adecuada para describir dicha ecuación (1.2). Haremos el esquema como describe la Fig. 1.6. Para unir los bloques debemos pinchar con el botón izquierdo del ratón en el bloque de origen y soltar en el bloque de destino.

Como puede verse en la Fig. 1.6, se han editado los nombres de los bloques poniéndoles nombres distintos a los originales. También se ha editado el valor de algunos de los bloques. Daremos valores concretos a las constantes. Supongamos que la masa es de un kg $m = 1$, que la constante de fricción vale $c = 0{,}8$ y que la fuerza aplicada es 0.1 N ($F = 0{,}1$). Así por ejemplo el bloque 'Gain' denominado 'c/m' posee en su interior el valor correspondiente a $\frac{c}{m} = 0{,}8$, y el denominado '1/m' tendrá valor 1. Estos valores se introducen haciendo doble 'click' en los bloques y editando el campo correspondiente.

Por otro lado se ha escrito texto para hacer más fácil la lectura del modelo. Estas cajas de texto se crean simplemente haciendo doble 'click' en el lugar que se desee y editando el recuadro que aparece.

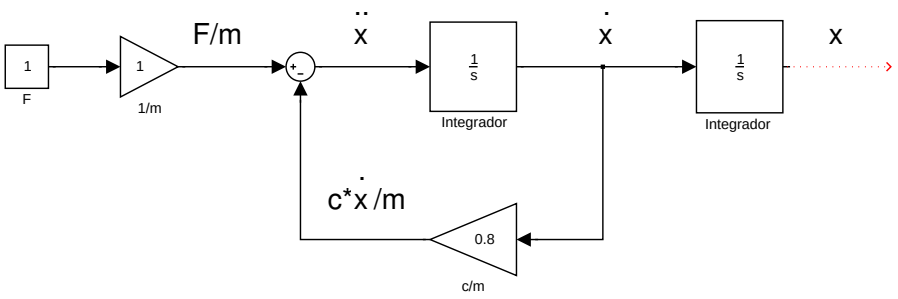


Figura 1.6: Modelo

Para poder ver los resultados ponemos un bloque que nos muestre la posición de la masa frente al tiempo. Seleccionamos dentro de la librería 'Sinks' el bloque 'Scope'. Lo añadimos al modelo de la forma habitual. Ya tenemos el modelo completo. Los bloques deben estar como se muestra en la Fig. 1.7.

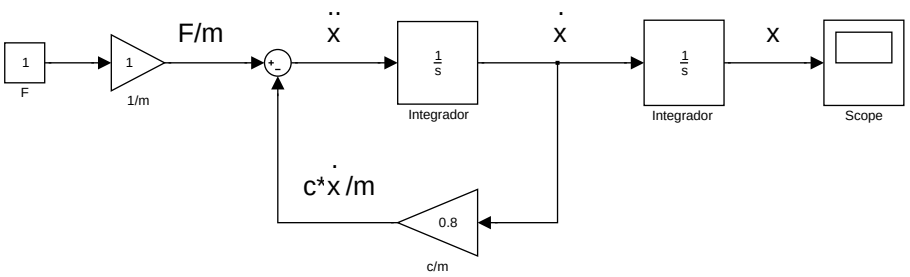


Figura 1.7: Modelo con 'Scope'

En la Fig. 1.7 debe notarse que las variables de estado están perfectamente definidas y accesibles en el diagrama de bloques. Ahora se está en disposición de hacer una simulación del proceso. Para ello debemos definir algunos parámetros esenciales. Los más importantes son las condiciones iniciales de las variables de estado y el tiempo de simulación. Las condiciones iniciales deben ponerse en los bloques integradores. Se hace doble 'click' en ellos y se definen las mismas en la zona de edición correspondiente. Por ejemplo ponemos el valor inicial de la velocidad a -1. En la Fig. 1.8 puede verse dónde se define el parámetro después de haber hecho doble 'click' en el integrador que nos da la velocidad.

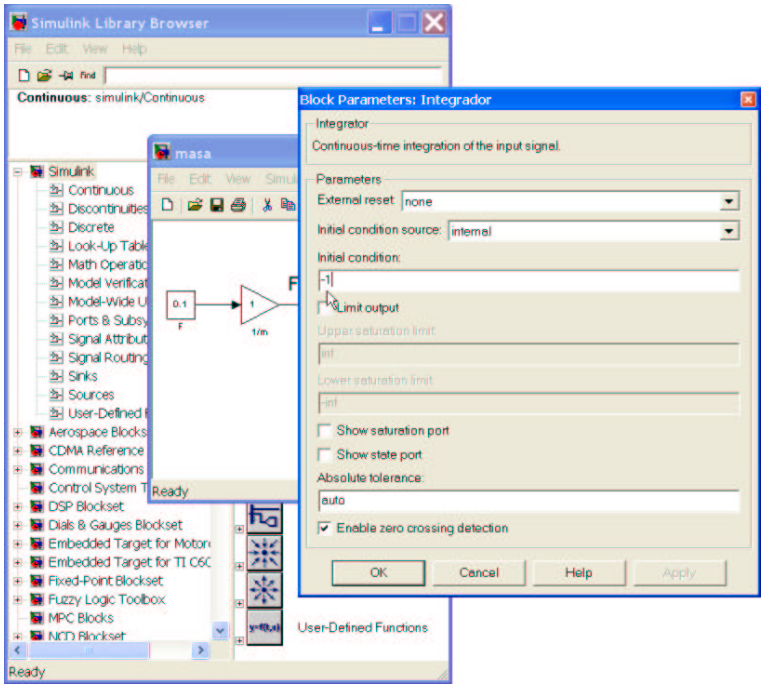


Figura 1.8: Condición inicial en velocidad

Para definir el tiempo de simulación accedemos al menu de la ventana del modelo Simulation → Simulation parameters. Se abre una ventana dónde es posible definir entre otros parámetros el tiempo de simulación, el método de resolución y el paso fijo o variable. Dejamos los dos últimos como están y ponemos el tiempo de simulación a 10 segundos. La situación será como la mostrada en la Fig. 1.9

Por último definimos la fuerza aplicada que deseamos. Hacemos doble 'click' en el bloque donde está definida la fuerza y ponemos el valor deseado que era 0.1.

Para ver el resultado en el Scope debemos hacer doble 'click' sobre el mismo y se abrirá la ventana gráfica que nos dará la posición del cuerpo. Ahora pulsamos el botón de inicio de simulación ▶ . Una vez acabada la simulación tendremos el resultado que puede verse en la Fig. 1.10.

Si deseamos ver también la velocidad tenemos acceso a la variable en el diagrama. Podemos poner otro Scope para la velocidad. El resultado puede verse en la Fig. 1.11

Se podrían ver las dos variables de estado en una sola ventana gráfica Scope. Se necesita para ello el bloque 'Mux' dentro de la librería 'Signal routing'. Este bloque hace las veces de un multiplexor y anida vectores. Se

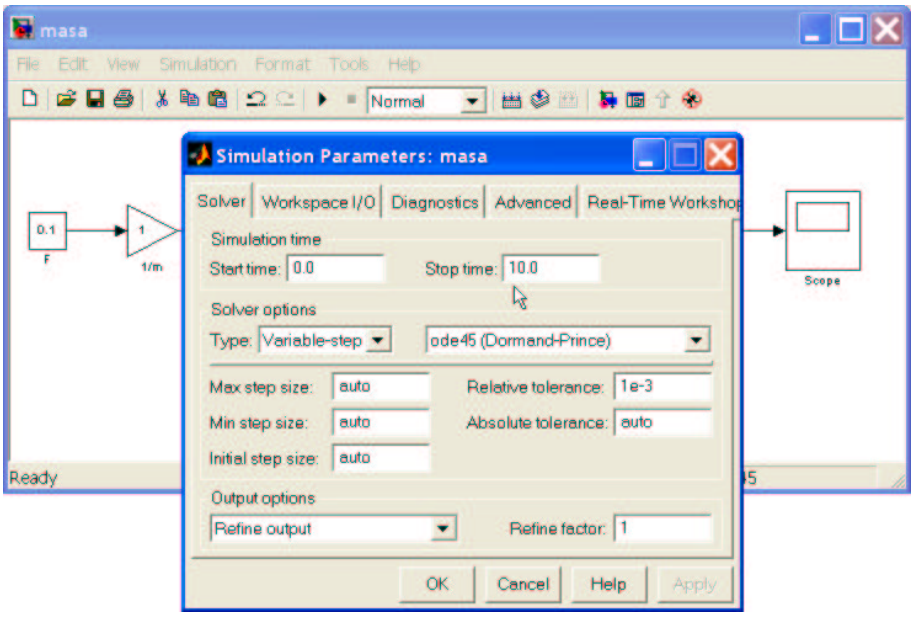


Figura 1.9: Parámetros de simulación

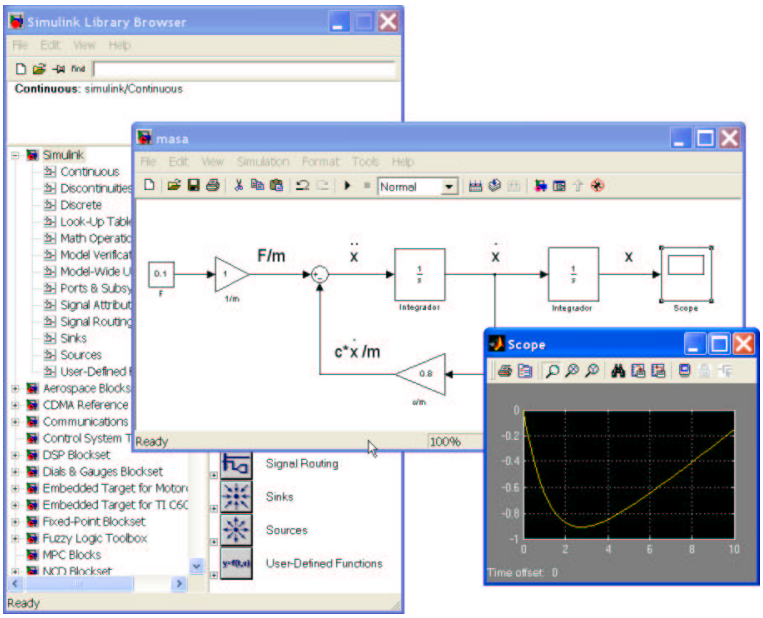


Figura 1.10: Simulación

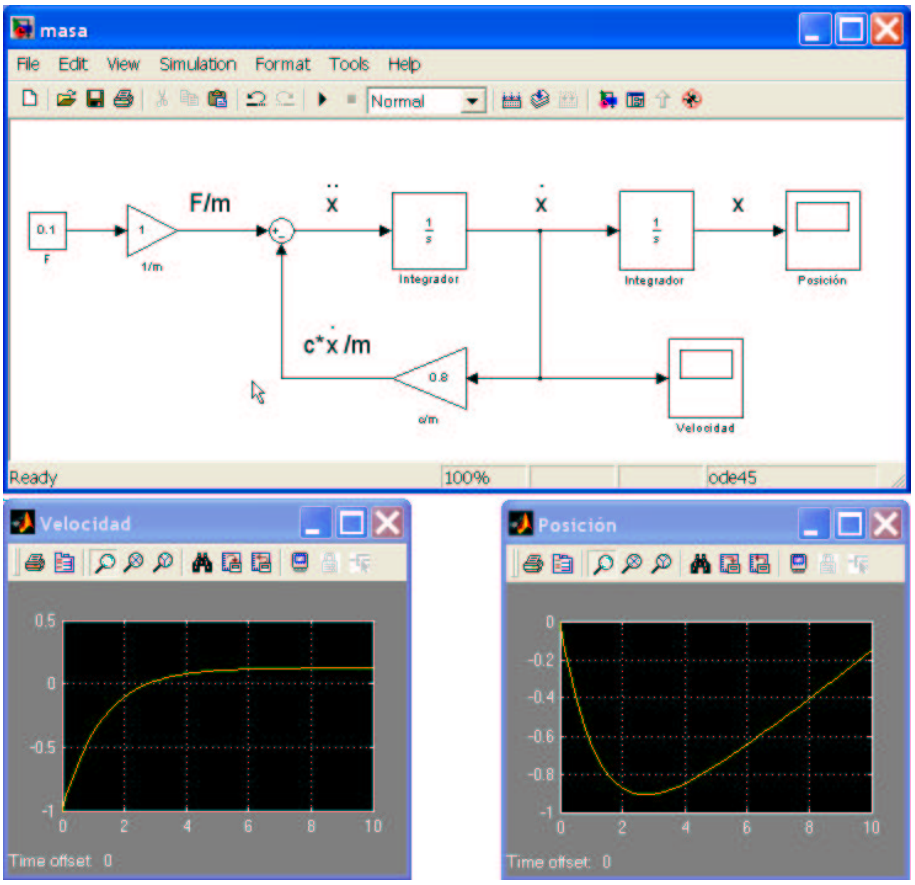


Figura 1.11: Simulación. Utilización del bloque Scope

modifica el diagrama como se ve en la Fig. 1.12 y ya se tienen las dos variables en una sóla ventana gráfica.

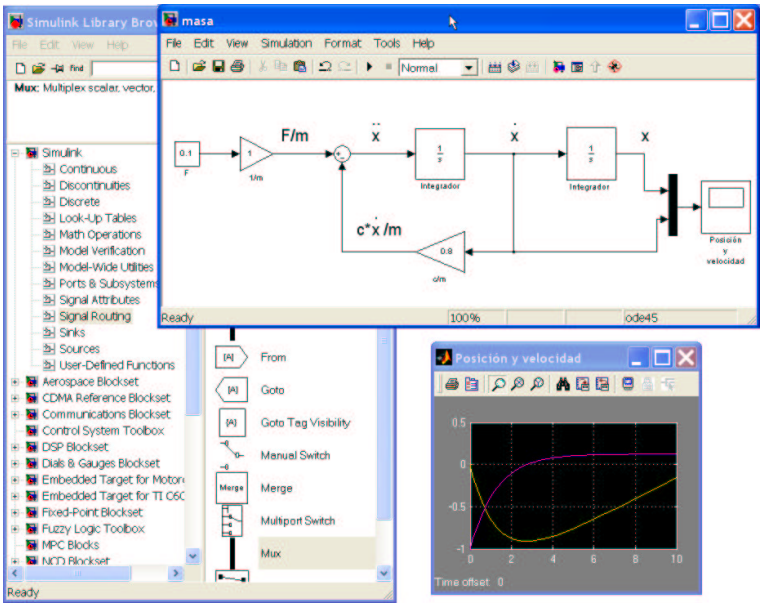


Figura 1.12: Simulación. Utilización del bloque Mux

Supongamos que no sólo queremos ver el resultado sino que también queremos guardar las variables en memoria para poder acceder a ellas. Buscamos dentro de la librería 'Sinks' el bloque 'To Workspace'. Lo añadimos al diagrama anterior y le damos un nombre a la matriz donde queremos guardar el valor de las variables, por ejemplo X. El resultado es el de la Fig. 1.13

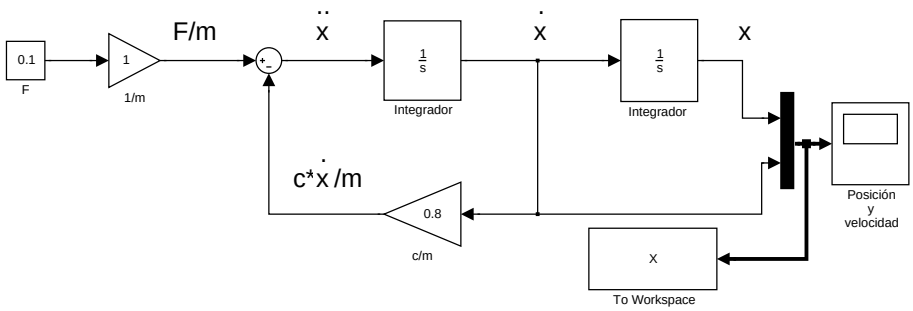


Figura 1.13: Simulación. Utilización del bloque To Workspace

Si ahora queremos ver el valor de las variable desde la linea de comandos de Matlab, hacemos 'plot(tout, X)'. En la matriz X se encuentran la posición

y la velocidad por columnas en el orden que se han puesto en el diagrama de bloques Fig. 1.13. El tiempo de simulación se guarda por defecto en la variable 'tout' dada en el menu Simulation → Simulation parameters → Workspace I/O. El resultado se muestra en la Fig. 1.14.

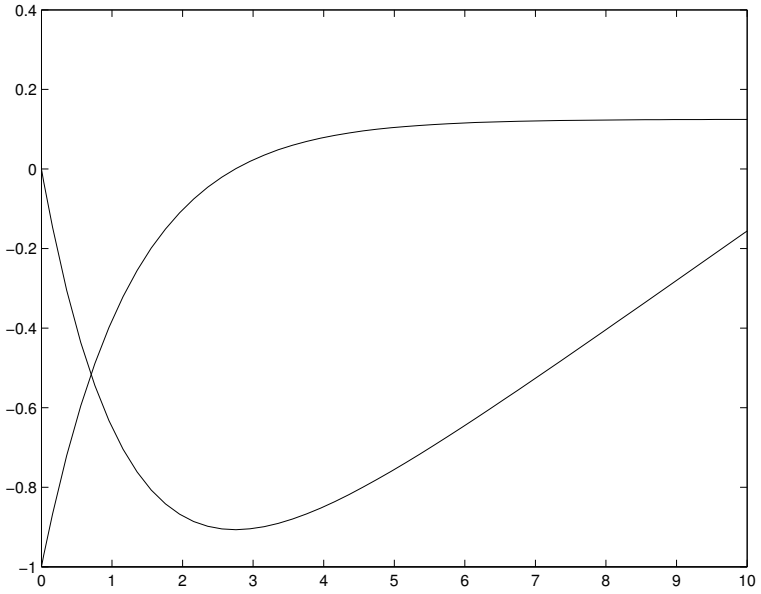


Figura 1.14: Resultado de ejecutar el comando plot(tout,X)

Todo este modelo se ha creado a través de las variables temporales y sus derivadas. Simulink permite hacer los modelos untilizando la transformada de Laplace. Para ello necesitamos transformar la ecuación del modelo (1.2) en el dominio de Laplace. Suponemos las condiciones iniciales iguales a cero. Por tanto, la ecuación (1.2) quedará en el dominio de Laplace

$$ms^2X(s) + csX(s) = F \tag{1.3}$$

Podemos transformarla en una función de transferencia si tomamos como salida la posición ($X(s)$) y como entrada la fuerza aplicada ($F(s)$) del modo siguiente

$$\begin{aligned} \frac{X(s)}{F(s)} &= \frac{\frac{1}{c}}{s(\frac{m}{c}s+1)} \\ &= \frac{K}{s(\tau s+1)}, \end{aligned} \tag{1.4}$$

donde $K = \frac{1}{c}$ y $\tau = \frac{m}{c}$. Para el ejemplo anterior $K = \tau = 1{,}25$.

Ahora ya podemos construir el modelo utilizando Laplace. De la librería 'Continuous' elegimos los bloques 'Integrator' y 'Transfer Fcn'. Editamos este último bloque con los valores de K y τ anteriores haciendo doble 'click'. El modelo quedará como se muestra en la Fig. 1.15

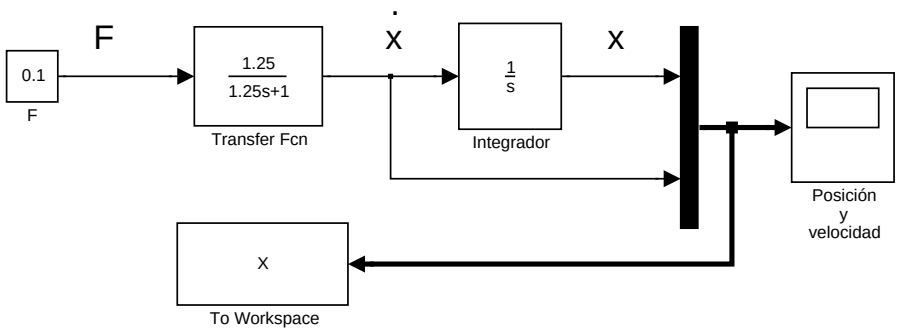


Figura 1.15: Modelo en el dominio de Laplace

Debe notarse que en este último esquema Fig. 1.15, la condición inicial de la velocidad no está accesible. Si se desea tener en cuenta hay que hacerlo a la hora de pasar las ecuaciones al dominio de Laplace.

Ya sabemos hacer un modelo de un sistema dinámico, tanto en el dominio del tiempo como en el de la frecuencia (Laplace). Ahora se describe como hacer subsistemas. La idea de estos subsistemas es agrupar bloques con algún criterio predefinido. Como ejemplo agruparemos los bloques del primer ejemplo como un sólo bloque que sea el modelo del sistema. Tendrá como entrada la fuerza aplicada y como salidas la posición y la velocidad del sistema. Para conseguir esto debemos seleccionar todo aquello que queremos que pertenezca al subsistema. La selección se hace con el botón izquierdo del ratón, como en Windows, haciendo un recuadro con todo aquello que queremos seleccionar. Todo esto con el botón pulsado. Después se suelta y nos vamos al menu 'Edit' → 'Create subsystem', como en la Fig. 1.16

Una vez hecho esto tendremos la situación de la Fig. 1.17, donde todo lo seleccionado anteriormente se ha metido dentro de un bloque.

Haciendo doble 'click' en el bloque se puede ver su contenido en otra ventana como se muestra en la Fig. 1.17.

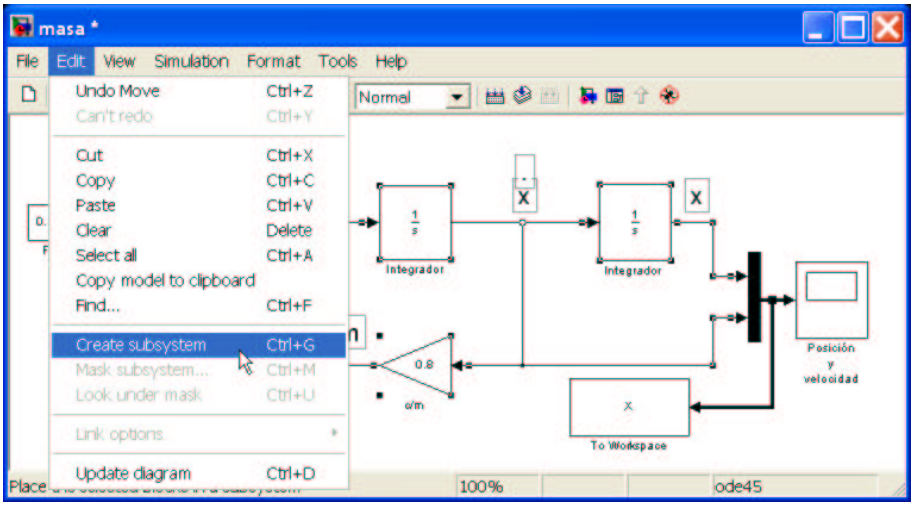


Figura 1.16: Creando subsistemas

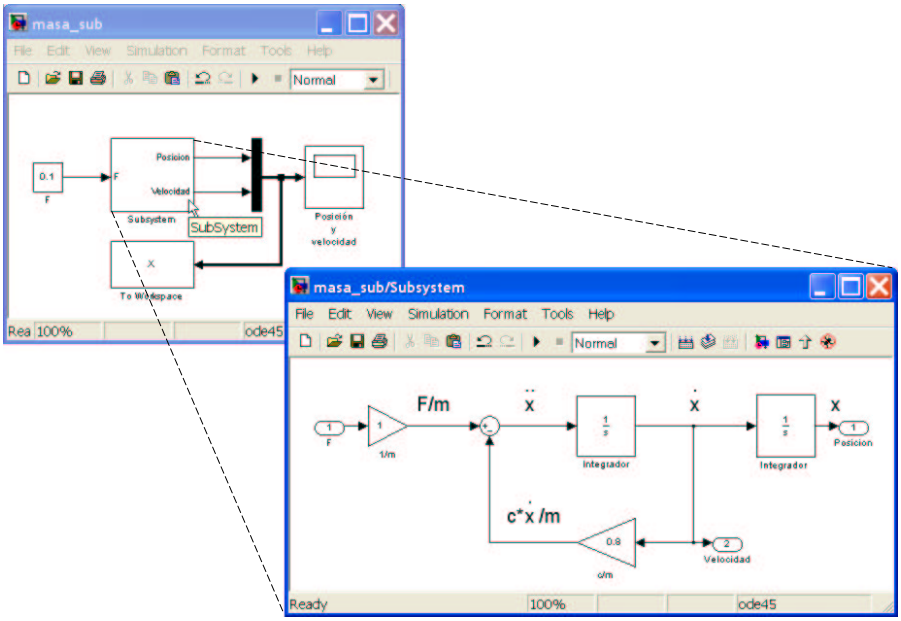


Figura 1.17: Creando subsistemas

PRÁCTICA 4 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Iván Maza Alcañiz

Agradecimientos a Manuel Berenguel Soria

Índice general

Capítulo 1

Introducción a Simulink. Parte II

1.1. Introducción

Al final de la anterior práctica se aborda la creación de subsistemas. En esta práctica se verá cómo crear máscaras para estos subsistemas. Dichas máscaras permiten configurar el cuadro de diálogo asociado a un subsistema, su icono, su documentación y los comandos que se ejecutan al comenzar una simulación.

A continuación se describirá superficialmente cómo arrancar simulaciones de Simulink desde MATLAB y cómo generar funciones MATLAB que permitan modificar los parámetros del modelo y arrancar múltiples simulaciones.

Finalmente, se planteará un modelo que permite simular la eyección de un piloto de un avión de combate, en el que se podrán aplicar los conceptos introducidos en esta práctica. Acerca de este modelo se plantean una serie de ejercicios para que el alumno los resuelva.

1.2. Aplicación de máscaras a subsistemas

Después de crear un subsistema cómo se vio en la anterior práctica, es posible crear una máscara asociada a dicho subsistema. Se va a tomar como ejemplo un subsistema muy sencillo que implementa una línea recta (ver Figura 1.1).

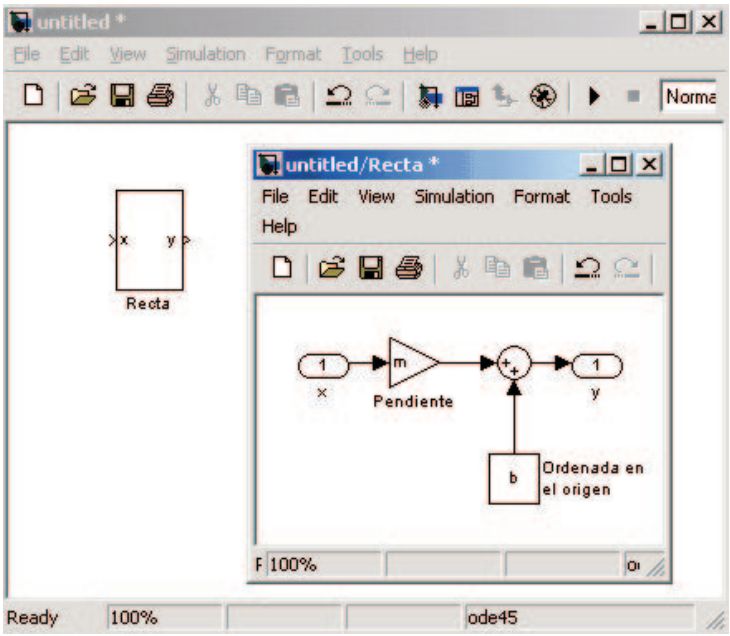


Figura 1.1: Diagrama de bloques del subsistema de ejemplo.

En dicha figura se puede observar que la pendiente se ha implementado mediante un bloque **Gain** y que el valor que se ha dado ha dicho bloque ha sido la variable `m` (posteriormente se verá por qué es necesario hacer esto). La ordenada en el origen de la recta se ha implementado con un bloque **Constant**, al que se ha asignado la variable `b`.

Los nombres que se han asignado a la entrada y la salida del subsistema han sido `x` e `y` respectivamente. La operación que realiza el subsistema es: $y = mx + b$.

Para aplicar una máscara sobre un subsistema hay que seleccionarlo previamente con el botón izquierdo del ratón. A continuación dentro de la opción **Edit** del menú, hay que seleccionar **Edit mask...** Entonces se despliega un cuadro de diálogo similar al que se muestra en la Figura 1.2.

Existen tres pestañas en dicho cuadro de diálogo que se describen brevemente en las siguientes secciones.

1.2.1. Configuración del icono

Es posible crear un icono específico para cada subsistema. Para ello se utiliza la pestaña **Icon** dentro del cuadro de diálogo de la máscara del sub-

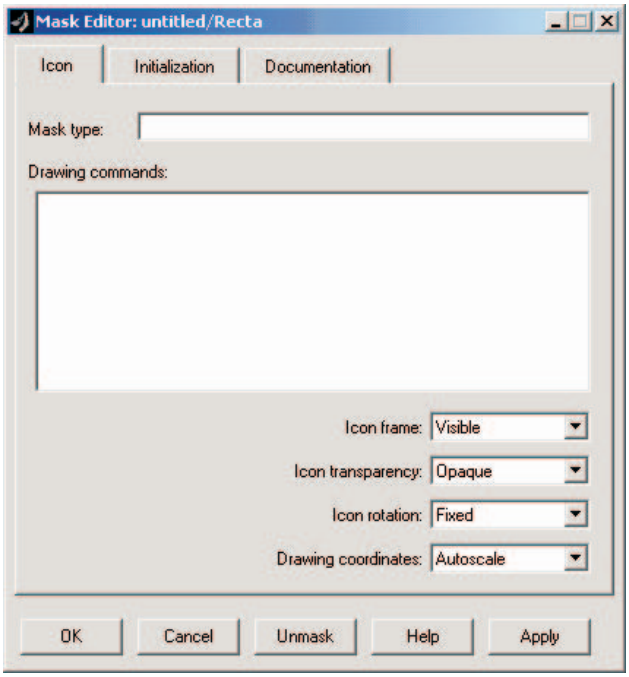


Figura 1.2: Cuadro de diálogo asociado a la máscara de un subsistema.

sistema (ver Figura 1.2). Dentro de la ventana **Drawing commands** es posible escribir un código en MATLAB para dibujar el icono que se desea para el subsistema. Siguiendo con el ejemplo de la recta, si se desea que el icono sea una línea bastará con escribir (ver Figura 1.3):

```
plot([0 1], [0 m]+(m<0))
```

En la parte inferior del cuadro de diálogo hay diversas opciones que permiten configurar el modo en que se muestra el icono diseñado.

1.2.2. Inicialización y parámetros del subsistema

Seleccionando la pestaña **Initialization** se muestra un cuadro de diálogo como el de la Figura 1.4.

Este cuadro de diálogo permite configurar los parámetros que nos solicitará el subsistema cuando se haga doble 'click' sobre él. Es posible añadir nuevos parámetros mediante el botón **Add**. Una vez que se ha pulsado este botón es posible configurar el texto que se nos mostrará para solicitarnos los

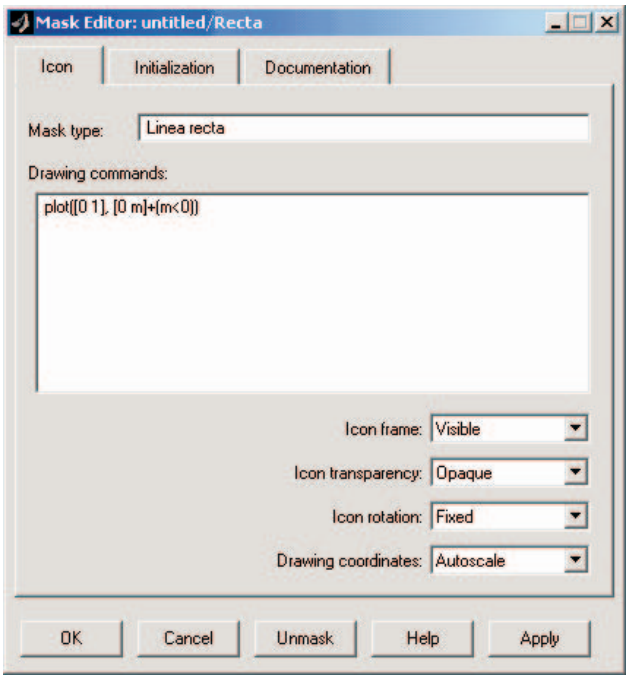


Figura 1.3: Cuadro de diálogo asociado a la creación de un icono.

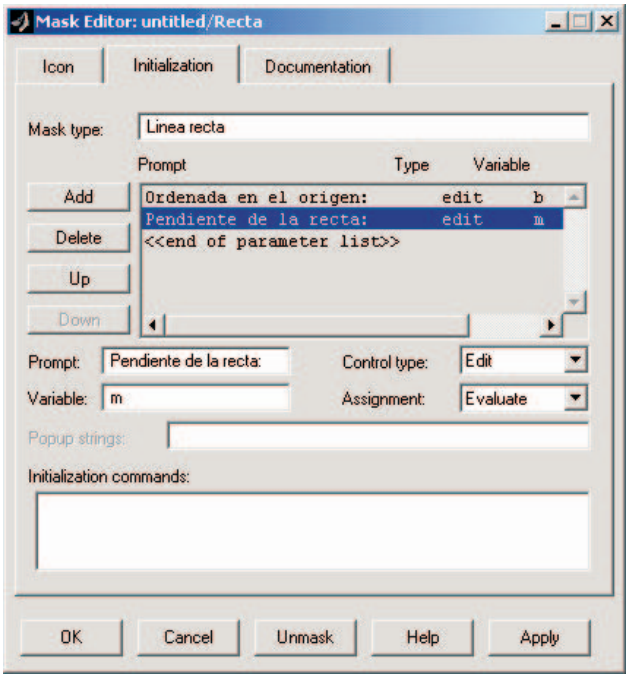


Figura 1.4: Cuadro de diálogo asociado a los parámetros e inicialización de un subsistema.

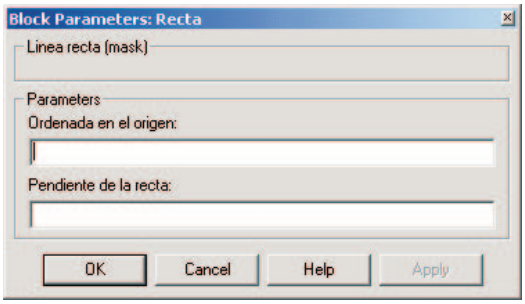


Figura 1.5: Cuadro de diálogo asociado al subsistema una vez que se configuran sus parámetros.

parámetros (en la casilla **Prompt:**) y la variable a la que se asignará el valor que introduzcamos (en la casilla **Variable:**). También es posible configurar el tipo de control que se utiliza para la variable y el modo de asignación (mediante **Control type:** y **Assignment:**). Los parámetros que se vayan configurando aparecerán en la lista que hay en la parte superior del cuadro de diálogo. En la Figura 1.4 se puede observar que se han configurado dos parámetros, de tal modo que cuando se pulse sobre el subsistema en cuestión, aparecerá el cuadro de diálogo que se muestra en la Figura 1.5.

Cuando el usuario introduzca dos valores, estos serán asignados automáticamente a las variables `m` y `b`. Hay que recordar que éstas variables están presentes en el esquema del subsistema y son respectivamente la pendiente y la ordenada en el origen de la recta.

Respecto al cuadro de diálogo **Initialization** comentar finalmente que en la parte inferior aparece una ventana titulada **Initialization commands:**. En esa ventana es posible escribir cualquier código MATLAB que se desee ejecutar justo antes de empezar la simulación.

1.2.3. Documentación del subsistema

Pulsando sobre la pestaña **Documentation** se muestra el cuadro de diálogo de la Figura 1.6.

En la ventana titulada **Block description:** es posible introducir el texto que se desea que aparezca en la parte superior del cuadro de diálogo del subsistema (ver Figura 1.7).

Finalmente en la ventana **Block help:** se puede introducir un texto que aparecerá cuando se pulse sobre la tecla **Help** del cuadro de diálogo del

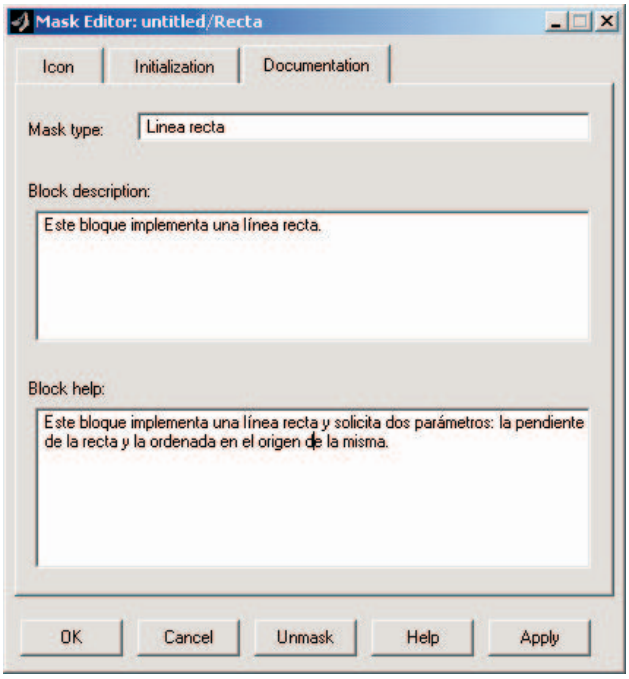


Figura 1.6: Cuadro de diálogo que permite configurar la documentación de un subsistema.

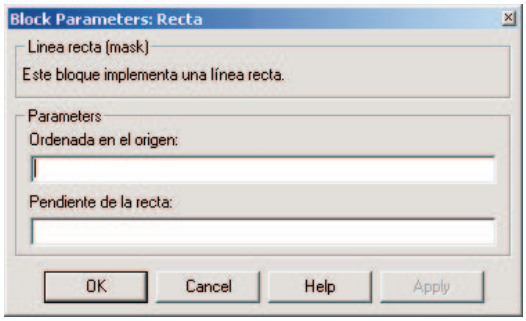


Figura 1.7: Se puede observar el texto de ayuda insertado en la Figura 1.6.

subsistema.

1.3. Simulación de modelos desde la línea de comandos

En muchas ocasiones resulta de utilidad escribir ficheros `.m` desde los que sea posible arrancar simulaciones. Para ello, basta escribir el siguiente comando MATLAB:

```
sim('nombre')
```

donde `nombre` es el fichero `.mdl` que se pretende simular.

Asimismo, también es posible utilizar variables del espacio de trabajo de MATLAB en los modelos Simulink sin necesidad de emplear el bloque **From Workspace**. Para ello basta con utilizar el nombre de la variable en cuestión en el bloque correspondiente. En el ejemplo anterior, se había usado un valor `m` para el bloque **Gain**. Si en el espacio de trabajo de MATLAB existiera una variable denominada `m`, se adoptaría el valor de dicha variable en el bloque de Simulink.

A continuación, se presenta un ejercicio en el cual se podrán aplicar los conceptos introducidos hasta ahora.

1.4. Ejercicio: Eyección de un piloto

Se trata de estudiar y simular el sistema de eyección de la combinación piloto/asiento en un avión de combate. El sistema que se va a analizar se presenta en la Figura 1.8.

Cuando el piloto se ve obligado a saltar, el conjunto asiento/piloto es guiado por un conjunto de railes para quedar separados convenientemente del avión. La velocidad de eyección v_E es constante a lo largo de la dirección que forma un ángulo θ_E con el eje y asociado al avión. La eyección tiene lugar cuando el conjunto asiento/piloto ha recorrido una distancia vertical y_1.

Después de la eyección, el piloto y el asiento siguen una trayectoria balística sujeta a la fuerza de resistencia aerodinámica sobre el conjunto y su propio

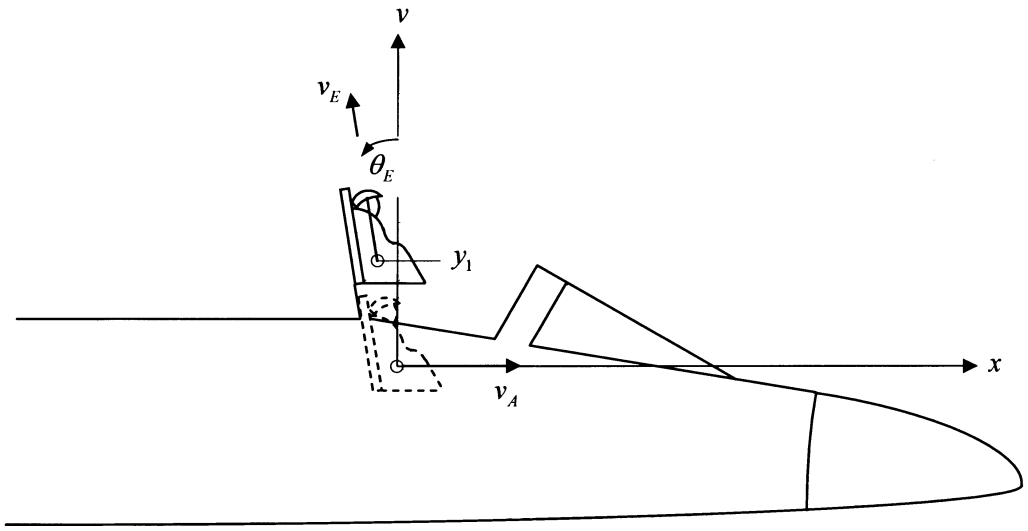


Figura 1.8: Diagrama de la eyección del piloto.

peso. Las ecuaciones del movimiento pueden ser deducidas en el sistema de coordenadas $x - y$ o en el $n - t$, donde n y t corresponden a las direcciones normal y tangencial a la trayectoria que siguen el piloto y el asiento (ver Figura 1.9). Sumando las fuerzas en las direcciones n y t,

$$\Sigma F_t = ma_t \Rightarrow -F_D - W \operatorname{sen}\theta = m\dot{v} \tag{1.1}$$

$$\Sigma F_n = ma_n \Rightarrow -W\cos\theta = m\frac{v^2}{R} \tag{1.2}$$

donde R es el radio de curvatura instantáneo de la trayectoria del conjunto asiento/piloto. Se supone que el avión viaja en horizontal a velocidad constante v_A.

La velocidad hacia adelante v y la velocidad angular $\dot{\theta}$ están relacionadas por la siguiente expresión:

$$v = R\dot{\theta} \tag{1.3}$$

Despejando R de la ecuación 1.3 y sustituyendo en la ecuación 1.2 se llega a:

$$-W\cos\theta = mv\dot{\theta} \tag{1.4}$$

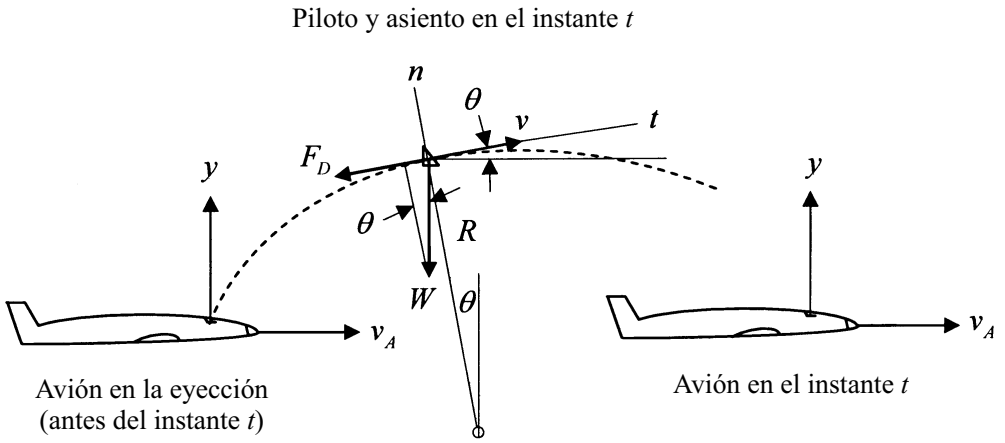


Figura 1.9: Trayectoria del piloto y el asiento tras la eyección.

Dado que $W = mg$ y siendo v y θ las variables de estado, se pueden expresar las derivadas de las mismas como:

$$\dot{v} = \begin{cases} 0 & 0 \leq y < y_1 \\ -\frac{F_D}{m} - g \operatorname{sen} \theta & y \geq y_1 \end{cases} \tag{1.5}$$

$$\dot{\theta} = \begin{cases} 0 & 0 \leq y < y_1 \\ -\frac{g \cos \theta}{v} & y \geq y_1 \end{cases} \tag{1.6}$$

donde los intervalos $0 \leq y < y_1$ y $y \geq y_1$ corresponden a los instantes antes y después de la eyección respectivamente.

Se necesitan adicionalmente las variables de estado x e y (coordenadas relativas del asiento/piloto respecto al avión en movimiento) para visualizar la trayectoria del asiento/piloto respecto al avión y comprobar si supera con seguridad el estabilizador vertical trasero del avión. Las derivadas de estas variables de estado vienen dadas por:

$$\dot{x} = v \cos \theta - v_A \tag{1.7}$$

$$\dot{y} = v \operatorname{sen} \theta \tag{1.8}$$

Es conveniente para comenzar la simulación, integrar por ejemplo las

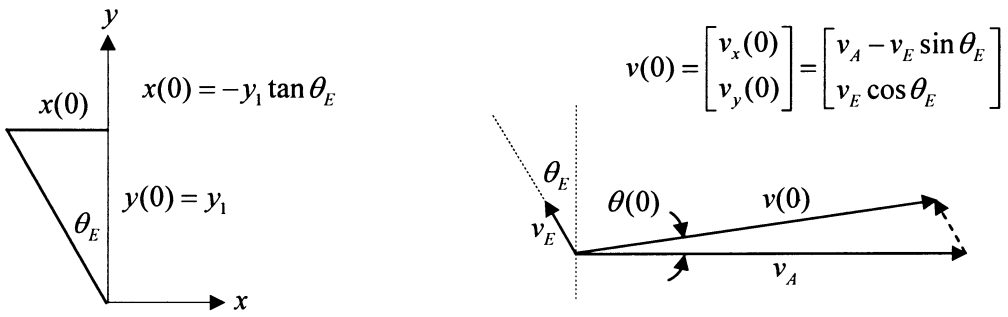


Figura 1.10: Estados iniciales $x(0)$, $y(0)$, $v(0)$ y $\theta(0)$ ($t = 0$ corresponde al instante de la eyección).

derivadas de las variables de estado en el momento de la eyección. Las condiciones iniciales se pueden obtener con la ayuda de la Figura 1.10.

Los estados iniciales $v(0)$ y $\theta(0)$ se pueden obtener a partir de las siguientes expresiones:

$$v(0) = \sqrt{v_x^2(0) + v_y^2(0)} \Rightarrow v(0) = \sqrt{(v_A - v_E \operatorname{sen}\theta_E)^2 + (v_E \cos\theta_E)^2} \quad (1.9)$$

$$\theta(0) = \arctan \frac{v_y(0)}{v_x(0)} \Rightarrow \theta(0) = \arctan \frac{v_E \cos\theta_E}{v_A - v_E \operatorname{sen}\theta_E} \quad (1.10)$$

Finalmente, la fuerza de resistencia aerodinámica F_D viene dada por:

$$F_D = \frac{1}{2} C_D \rho A v^2 \quad (1.11)$$

donde C_D es el coeficiente de penetración aerodinámica, ρ es la densidad del aire y A es el área frontal del conjunto asiento/piloto normal al vector velocidad.

Se va a realizar un estudio mediante simulación para investigar las combinaciones de velocidad del avión v_A y altitud h que permiten una eyección segura, es decir que el conjunto asiento/piloto pasen por encima del estabilizador vertical de cola con una holgura predeterminada.

En primer lugar, se simulará un único caso en el que se tomará $v_A = 500$ *pies/seg* y $h = 0$ (a nivel del mar). Para ello se utilizará el diagrama

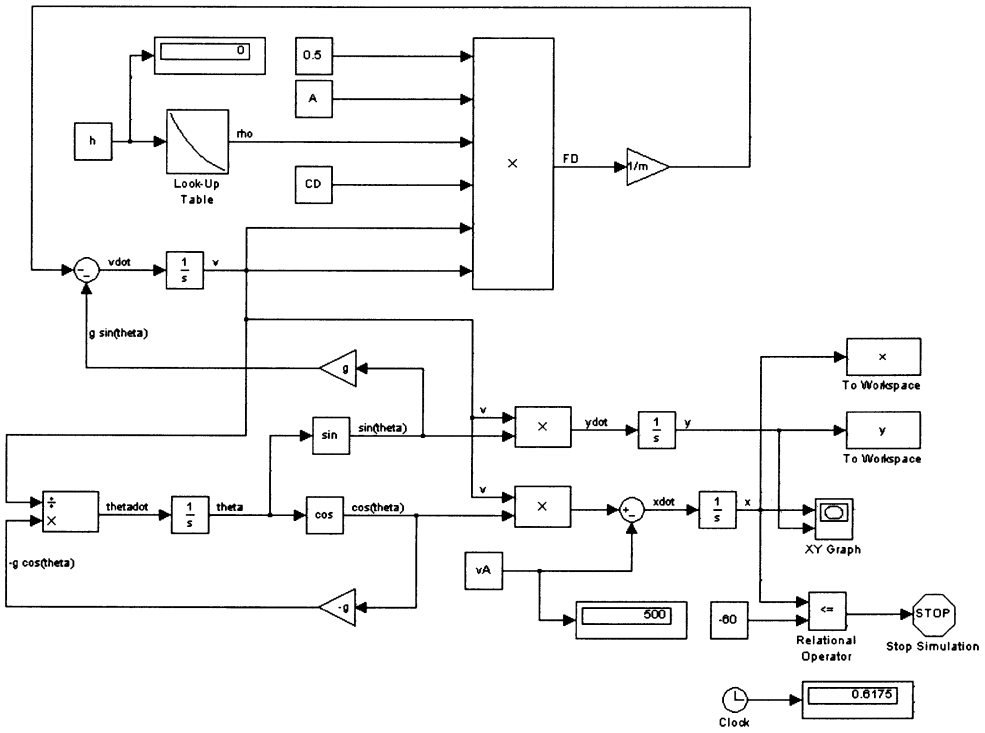


Figura 1.11: Modelo Simulink correspondiente al sistema de eyección.

Simulink de la Figura 1.11. Es necesario utilizar un bloque **Look-Up Table** para calcular la variación de la densidad del aire con la altura. En dicha tabla se deberán colocar los valores empíricos correspondientes a la densidad del aire ρ en $Kg/pies^3$ frente a la altitud $h(pies)$ desde el nivel del mar hasta los 50,000 $pies$ de altitud.

Los valores a introducir en el bloque **Look-Up Table** se pueden obtener fácilmente a partir de la información del cuadro 1.1 teniendo en cuenta que $\rho_0 = 0{,}0347\ Kg/pies^3$.

Altitud (pies)	Dens rel ρ/ρ_0
0	1.000
2000	.943

4000	.888
6000	.836
8000	.786
10000	.738
12000	.693
14000	.650
16000	.609
18000	.570
20000	.533
22000	.498
24000	.464
26000	.432
28000	.403
30000	.374
32000	.347
34000	.322
36000	.298
38000	.271
40000	.246
42000	.224
44000	.203
46000	.185
48000	.168
50000	.152

Cuadro 1.1: Valores de presión relativa del aire en función de la altitud.

Por otra parte, se puede tomar como área frontal un valor de 12,9 $pies^2$ y como coeficiente aerodinámico un valor igual a 1.5. El valor inicial de la velocidad de eyección es $v_E = 10\ ft/seg$, mientras que el ángulo de eyección es aproximadamente $\theta_E = 0{,}17\ rad$. El perfil del avión y la posición inicial del asiento/piloto (a 4 pies de altura sobre el perfil del avión) se muestran esquemáticamente en la Figura 1.12. Finalmente, se supone que la masa del conjunto asiento/piloto es igual a 120 Kg.

Se pide:

1. Crear un modelo Simulink con objeto de simular el sistema de eyección.

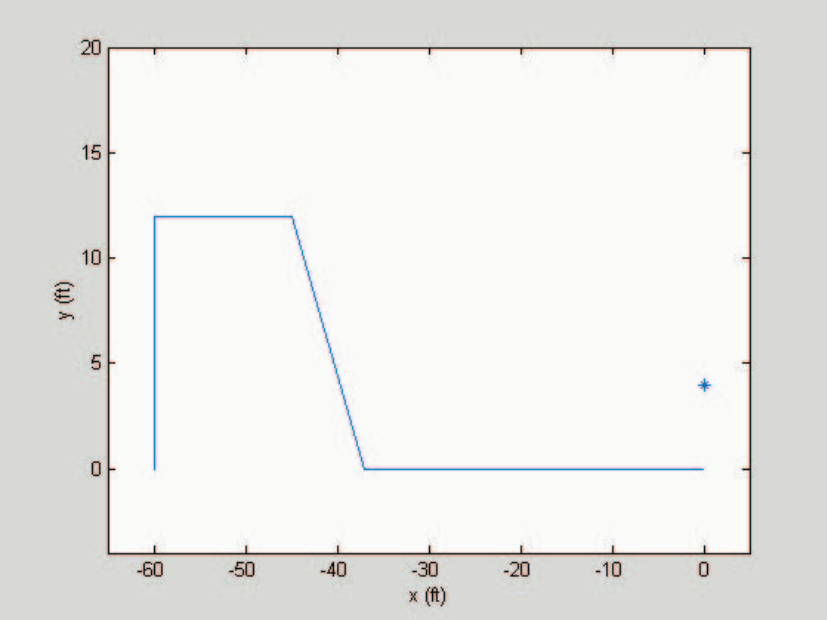


Figura 1.12: Perfil del estabilizador y posición inicial del asiento/piloto.

2. Dentro de dicho sistema, emplear un subsistema para el cálculo de la fuerza de resistencia aerodinámica. Dicho subsistema deberá tener como entradas:

   - La altura del avión.
   - La velocidad instantánea del avión.

   y devolverá a la salida la fuerza de resistencia aerodinámica. Los parámetros del mismo serán:

   - El área frontal del asiento/piloto.
   - El coeficiente aerodinámico.

3. Obtener una gráfica de la trayectoria del asiento/piloto relativa al avión (en el sistema de coordenadas $x - y$).

4. Obtener gráficas de la posición y frente al tiempo y del ángulo instantáneo θ frente al tiempo también.

5. A una cierta altitud h, la trayectoria del conjunto asiento/piloto superará convenientemente el estabilizador de cola siempre y cuando la velocidad del avión v_A esté dentro de un cierto rango. Así, a velocidades bajas, la velocidad de salida es insuficiente para lanzar al asiento/piloto

por encima del estabilizador convenientemente, mientras que a velocidades altas la excesiva fuerza de resistencia aerodinámica producirá un efecto similar. Suponiendo que es necesario superar el estabilizador con una holgura de 5 pies, se pide calcular para un conjunto de altitudes desde 0 a 50000 pies (con incrementos de 5000 en 5000 pies) cuál es el rango de velocidades del avión seguras para la eyección. Se recomienda representar todos los resultados en una gráfica de velocidades frente a altitudes, que muestre para cada altura dos puntos que corresponderán a la velocidad mínima y máximas que permiten un salto seguro.

PRÁCTICA 5 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Mercedes Pérez de la Parte

Agradecimientos a Manuel Berenguel Soria

Práctica 5

PID. Descripción y reglas heurísticas de Sintonización

1. Introducción

El objetivo de esta práctica es que el alumno se familiarice y profundice en el conocimiento de la estructura de control PID, profusamente usada en el mundo industrial.

Para ello se empleara el software de simulación de sistemas dinámicos SIMULINK® asociado al paquete de computación técnica MATLAB ®.

La descripción de la práctica y los puntos a tratar en la misma se recogen en los siguientes apartados.

2. El sistema a controlar

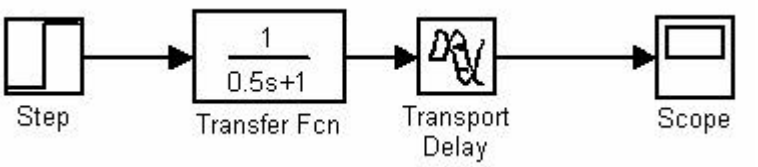
Para comenzar abordaremos el problema del control de un sistema simple de primer orden con retardo, definido por la función de transferencia.

$$G(s) = \frac{k}{1+ts} e^{-sL}$$

donde k representa la ganancia estática del sistema, t es su constante de tiempo y L es el retardo del mismo.

Este tipo de sistemas, a pesar de su sencillez, modelan bastante bien una amplia clase de sistemas dinámicos que involucran generalmente fenómenos de transporte de materia como sucede en muchos procesos químicos, térmicos y muchos otros muy comunes en la industria de procesos.

Comenzaremos viendo la respuesta de este sistema a lazo abierto ante una entrada en escalón, y para ello construiremos el siguiente sistema en SIMULINK



para un sistema con los siguiente parámetros: $k = 2$, $t = 0.5\,\text{s}$ y $L=0.8\ s$.

En la simulación observaremos la respuesta esperada, es decir la respuesta de un sistema de primer orden con un retardo de 0.8 segundos respecto a la entrada en escalón marcada.

3. Controlador PID

El controlador PID es una estructura de control en la que la señal de control del proceso se expresa en función del error, $e(t)=y_{ref}(t) - y(t)$, según la expresión estandar:

$$u(t) = K_p e(t) + \int_0^t K_i e(t)dt + K_d \frac{de(t)}{dt}$$

donde K_p, K_i y K_d corresponden respectivamente a las constantes Proporcional, Integral y Derivativa del controlador.

La expresión anterior puede igualmente expresarse como la siguiente función de transferencia del controlador PID

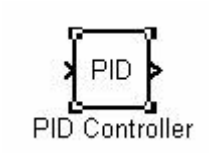
$$K(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + K_d s$$

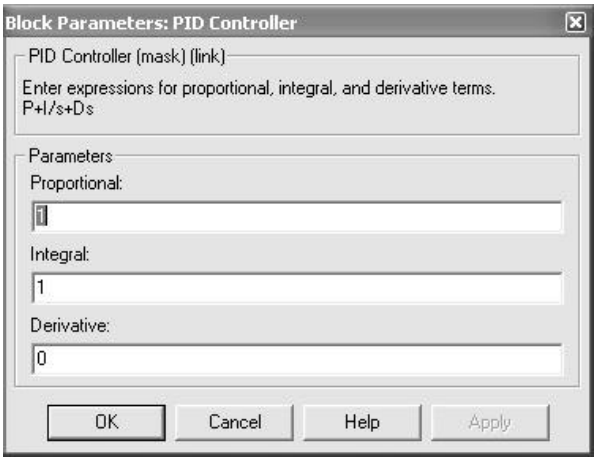
Esta función de transferencia puede implementarse en SIMULINK de dos modos distintos:

1) Empleando el bloque PID que proporciona el software para este controlador

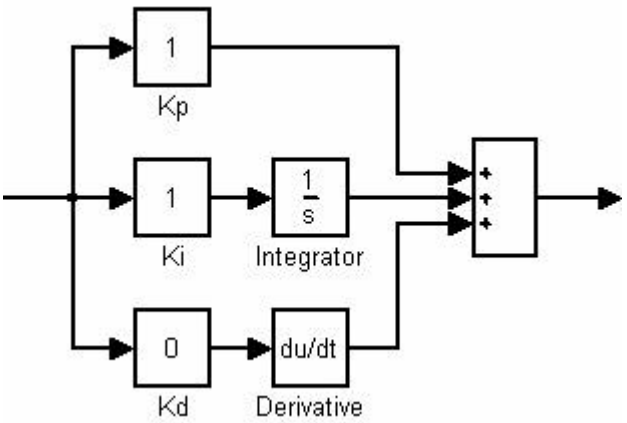
   que puede encontrarse en *Simulink Extras -> Additional Linear*

   Pulsando dos veces sobre este bloque obtenemos la ventana de diálogo donde podemos introducir los parámetros del controlador arriba indicados





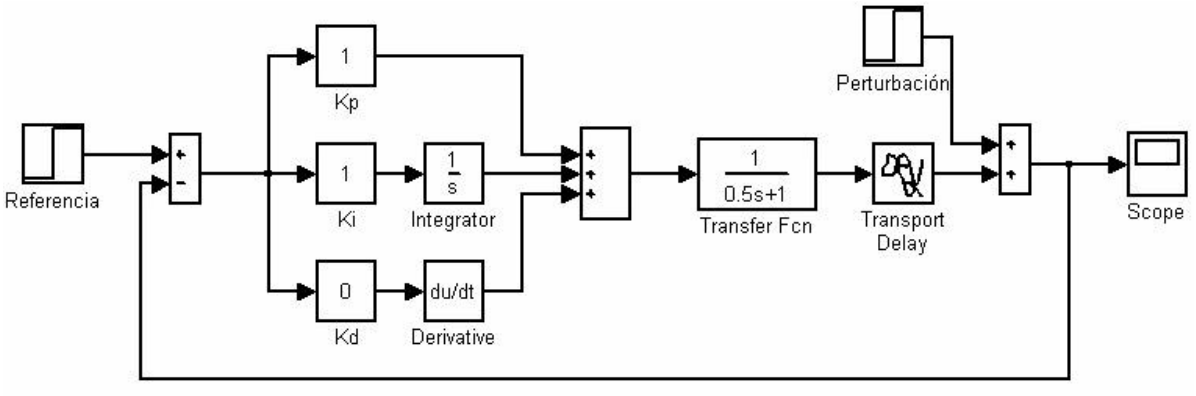
2) También es posible construir la estructura del PID partiendo de bloques elementales de SIMULINK del siguiente modo



Esta segunda estructura será la que emplearemos en lo sucesivo en la práctica.

4. Control a lazo cerrado.

Para comprobar la influencia del controlador PID en el sistema propuesto construiremos la siguiente estructura de control realimentada



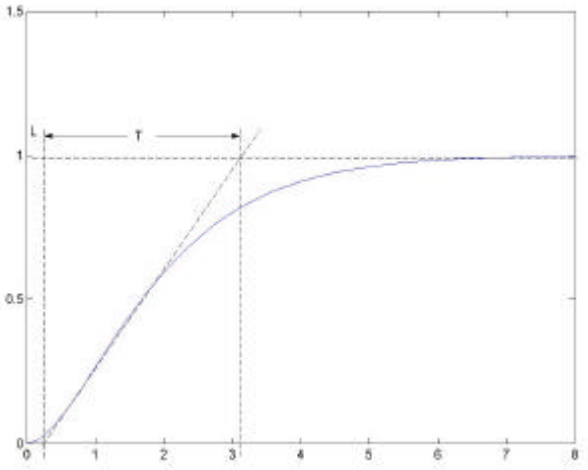
Esta estructura representa un control PID clásico que incluye el controlador en la cadena directa del sistema.

5. Ajuste del PID. Reglas del Ziegler-Nichols.

Para un ajuste inicial del controlador anterior, emplearemos las conocidas reglas de Ziegler-Nichols.

a) Primer método de Ziegler-Nichols

Las características del sistema estudiado permite emplear el método de respuesta a escalón de Ziegler-Nichols que caracteriza un sistema mediante dos parámetros, L y T, obtenidos a partir de la respuesta a lazo abierto del mismo como representa la figura siguiente



Según este procedimiento de sintonización los parámetros del controlador pueden obtenerse de acuerdo con las expresiones de la siguiente tabla.

Controlador	K_p	K_i	K_d
P	$\frac{T}{L}$	0	0
PI	$0.9\frac{T}{L}$	$\frac{0.3}{L}$	0
PID	$1.2\frac{T}{L}$	$\frac{1}{2L}$	$0.5L$

Tabla 1: Parámetros del PID según el método de respuesta a Escalón de Ziegler-Nichols

De este modo a partir de la respuesta a lazo abierto del sistema, calcularemos los controladores P, PI y PID apropiados para nuestro sistema.

Para cada uno de los tres controladores anteriores se pide:

1) Calcular las respuesta en el dominio temporal y caracterizar la respuesta según la ganancia estática a lazo cerrado (K_0), sobreoscilación (SO), tiempo de subida, (t_s), tiempo de establecimiento (t_e) y ratio de decaimiento (r_d).

2) Modificar los parámetros de cada controlador para un ajuste fino de la respuesta anotando la influencia del aumento o disminución de cada parámetro en la respuesta temporal.

b) Segundo método de Ziegler-Nichols

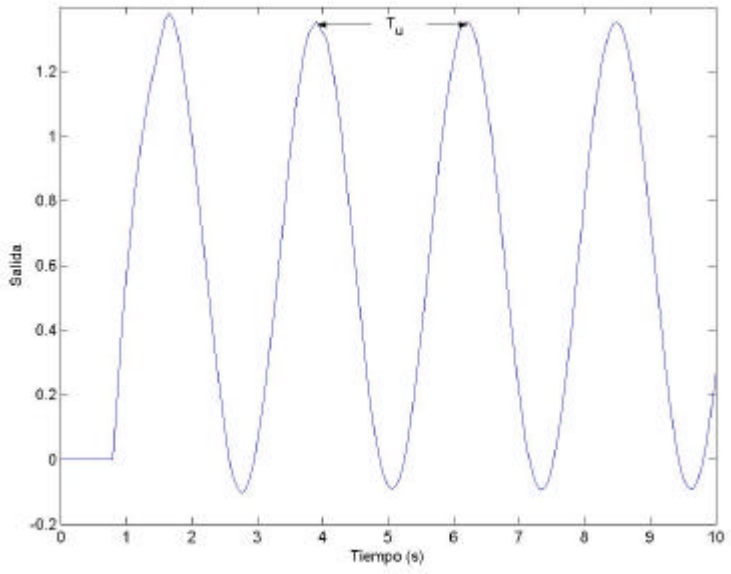
El segundo método de Ziegler-Nichols, o método de respuesta en frecuencia es un método alternativo de sintonización de PIDs que puede describirse como sigue:

En primer lugar es necesario ajustar las ganacias integral y derivativa a cero, esto es $K_i = 0$ y $K_d = 0$.
A continuación, partiendo de un valor bajo de la ganancia proporcional, K_p, vamos aumentando ésta gradualmente hasta conseguir un comportamiento oscilatorio mantenido en la respuesta del sistema tal como muestra la gráfica. A esta ganancia la llamaremos K_U.
El otro parámetro que nos hace falta es el periodo de oscilación del sistema para esta ganancia, que llamaremos T_U, y que se calcula como muestra la gráfica.

Con los valores de K_U y T_U entramos en la tabla 2 de Ziegler-Nichols y calculamos los parámetros correspondientes.

Controlador	K_p	K_i	K_d
P	$0.5K_U$	0	0
PI	$0.45K_U$	$\frac{1.2}{T_U}$	0
PID	$0.6K_U$	$\frac{2}{T_U}$	$0.125T_U$

Tabla 2: Parámetros del PID según el método de respuesta en frecuencia de Ziegler-Nichols



Para cada uno de los tres controladores, P, Pi y PID, se pide:

1) Calcular las respuesta en el dominio temporal y caracterizar la respuesta según la ganancia estática a lazo cerrado (K_0), sobreoscilación (SO), tiempo de subida, (t_s), tiempo de establecimiento (t_e) y ratio de decaimiento (r_d).

PRÁCTICA 6 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Daniel Jiménez Jiménez
Luis Merino Cabañas

Agradecimientos a Manuel Berenguel Soria

Respuesta temporal de sistemas LTI

Introducción

En la práctica anterior se han descrito comandos básicos en MATLAB para el análisis y control de sistemas. En la presente práctica, se emplearán dichos comandos para estudiar más profundamente y de forma práctica, mediante simulación, la respuesta temporal de sistemas.

A la hora de diseñar sistemas de control automático, una aproximación consiste en estudiar la respuesta del sistema ante entradas características, denominadas entradas de test o prueba (algunas de ellas pueden verse en la figura 1); en general, el objetivo será el diseño de un sistema de control de modo que optimice la respuesta del sistema controlado ante este tipo de entradas. Para ello, también habrá que determinar criterios que permitan comparar las respuestas de los sistemas ante entradas y controladores distintos.

En esta práctica nos centraremos en la respuesta ante entradas en escalón y entradas sinusoidales, y la relación entre los resultados y los modelos considerados. También se mostrará como el análisis de la respuesta temporal puede emplearse para el diseño de un controlador.

La práctica se estructura de la siguiente forma. En primer lugar se recordará y profundizará en los comandos que permiten la definición y ejecución de operaciones con sistemas lineales e invariantes en el tiempo (LTI) en MATLAB. A continuación se mostrarán los comandos relacionados con la respuesta temporal de sistemas, analizando la respuesta de algunos sistemas ante entradas en escalón y entradas sinusoidales. Finalmente se presentará el denominado Visor LTI, que permite obtener en MATLAB toda la información de interés (respuesta temporal y análisis frecuencial) para un sistema dado.

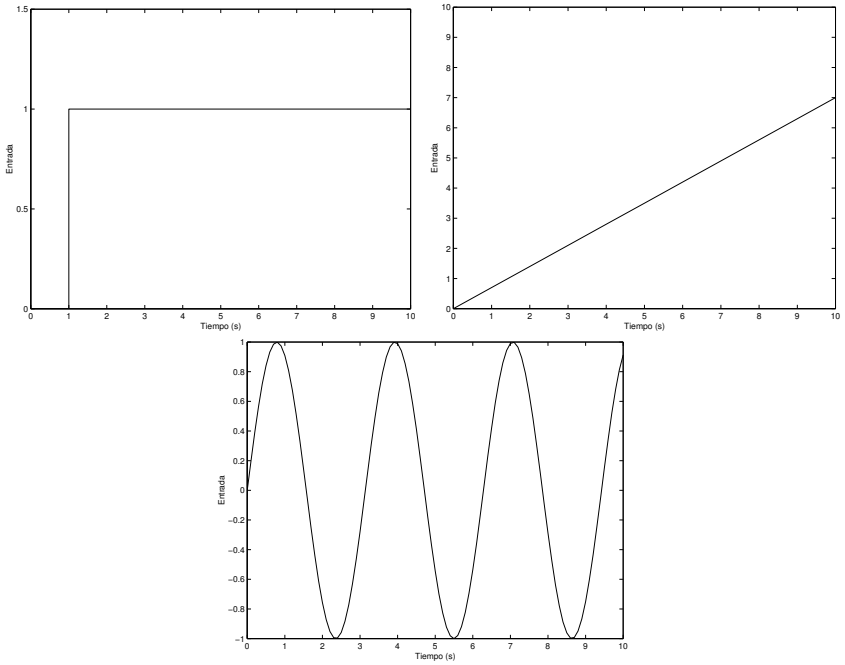


Figura 1: Entradas características: escalón, rampa y seno

Modelos LTI en MATLAB

En la práctica anterior, se ha visto como trabajar con un sistema en MATLAB en base al numerador y denominador de su función de transferencia. Las funciones y comandos para el estudio de sistemas necesitaban como parámetros el numerador y denominador. Sin embargo, MATLAB, a partir de la versión 5, tiene un nuevo tipo de variable que permite almacenar un sistema lineal e invariante en el tiempo (LTI).

Para la definición de un sistema LTI en MATLAB se tienen varios comandos. En principio, el comando más interesante es el que permite definir un sistema LTI a partir de su función de transferencia, el comando **tf**. Dicho comando recibe como parámetros los vectores con los coeficientes que definen el numerador y el denominador de la función de transferencia. Así

```
>>sysP=tf([1 3],[1 1 2 4]);
```

define el sistema *sysP* cuya función de transferencia es:

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

Nótese que *sysP* define el sistema completo, y ya no es necesario arrastrar el numerador y denominador para realizar operaciones, como se verá. También es posible definir un sistema LTI a partir de sus polos y ceros, mediante el comando **zpk**, al que se le pasan como parámetros los polos y ceros del sistema. Así:

```
>>%sys=zpk(ceros,polos,ganancia);
>>sysZ=zpk(-1,[-0.1 -3],2);
```

crea un modelo LTI con un cero en $s = -1$ y polos en $s = -0{,}1$ y $s = -3$, con ganancia 2.

$$\frac{2(s+1)}{(s+0{,}1)(s+3)}$$

También es posible especificar un modelo LTI según el espacio de estados,mediante el comando **ss**:

```
>>sysS=ss(A,B,C,D);
```

crea un modelo LTI cuyas matrices A, B, C y D son las especificadas en la llamada de la función.

Además, es posible convertir de unos modelos a otros empleando las mismas funciones. Así, si *sysZ* es un modelo especificado según sus polos y ceros, la instrucción

```
>>sysP2=tf(sysZ);
```

lo transforma al formato de función de transferencia.

Recuérdese que mediante el comando **help** de MATLAB es posible acceder a una descripción detallada de las distintas instrucciones.

Operaciones con modelos LTI.

Es posible realizar operaciones con modelos LTI en MATLAB, como interconexiones, obtención de parámetros, etc.

Así, si *sys1* y *sys2* son dos modelos LTI, la instrucción

```
>>sys3=sys1*sys2;
```

almacena en *sys3* la conexión serie de ambos modelos (ver figura 2).

La instrucción

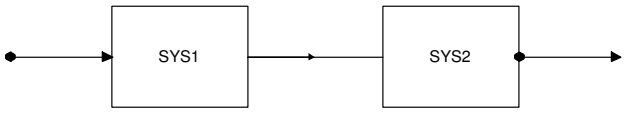


Figura 2: Conexión serie de dos sistemas.

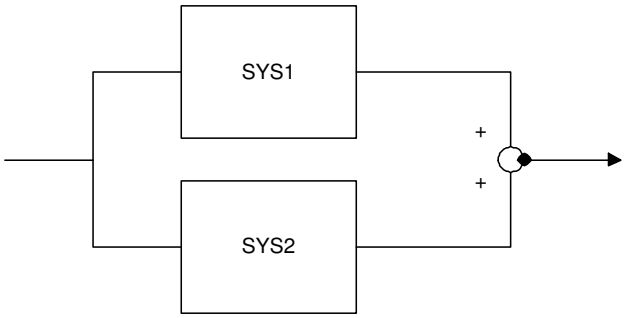


Figura 3: Conexión paralelo de dos sistemas.

```
>>sys4=sys1+sys2;
```

almacena en *sys4* la conexión paralelo de ambos modelos (ver figura 3).

Finalmente, la instrucción

```
>>sys5=feedback(sys1,sys2);
```

almacena en *sys5* la conexión mediante realimentación (negativa) de ambos modelos, según aparece en la figura 4. Para el caso de realimentación unitaria, *sys2* es igual a la unidad.

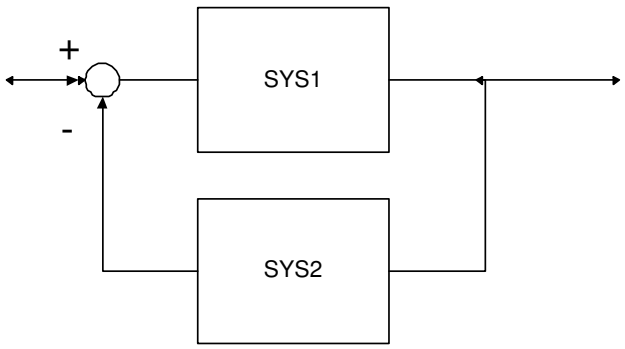


Figura 4: Conexión mediante realimentación.

Como ejercicio, el alumno debe obtener, empleando MATLAB, la función de transferencia del sistema correspondiente a *sys3* en la figura 5, siendo *sys1* y *sys2* los sistemas cuyos modelos se definen a continuación (en este caso, *sys3* corresponde al sistema controlado en bucle cerrado, si consideramos *sys2* como la planta a controlar y *sys1* como el controlador):

$$sys1(s) = \frac{3}{s} \tag{1}$$

$$sys2(s) = \frac{s+1}{s^2+2s+1} \tag{2}$$

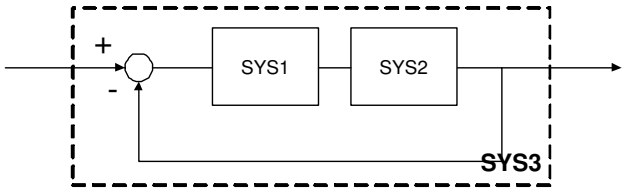


Figura 5: Determinar la función de transferencia correspondiente a *sys3* (sistema controlado en bucle cerrado).

Análisis de Modelos LTI

Acceso a propiedades del Modelo

Mediante el comando **get** se puede acceder a las propiedades de un modelo de tipo LTI. Suponiendo que previemente se hubiera creado uno de estos modelos (como función de transferencia),llamado *sysP* se escribiría:

```
>>sysP=tf(1,[1 2 1]);
>>get(sysP);
```

lo cual desplegaría una serie de características para el modelo en cuestion, entre las que destacan, el numerador, denominador, retraso a la entrada, retraso a la salida (en el caso de sistemas monovariables estos dos últimos serían lo mismo),nombre elegido para las variables de entrada y salida...etc. Para cambiar cualquiera de estas variables se utiliza el operando "punto", de la siguiente manera:

```
>>sysP.InputDelay=2;\%Retraso en segundos
>>sysP.InputName='Tension';\%Nombre de la variable de Entrada.
>>sysP.OutputName='Par Motor';\%Nombre de la variable de Salida.
```

Se obtendría entonces:

```
Transfer function from input "Tension" to output "Par Motor":
                  1
exp(-2*s) * -------------
            s^2 + 2 s + 1
```

donde, como puede observarse, ya aparece el retraso en forma de función exponenecial.

Obtención de Parámetros

Mediante los comandos **damp** y **dcgain** se pueden obtener las características del modelo, relacionadas con la respuesta temporal del mismo: El factor de amortiguamiento,la frecuencia natural y la ganancia en régimen permanente. Para el mismo sistema anterior, se haría lo siguiente:

```
>>dcgain(sysP)
```

obteniéndose

```
ans =

     1
```

es decir, una ganancia unitaria en régimen permanente (facilmente comprobable, sin más que dar un escalón al sistema). Para el caso de la frecuencia natural y la ganancia, se tendría:

```
>>damp(sysP)
```

produciendo como salida:

```
Eigenvalue      Damping     Freq. (rad/s)

 -1.00e+000     1.00e+000      1.00e+000
 -1.00e+000     1.00e+000      1.00e+000
```

es decir, una frecuencia natural y un factor de amortiguamiento de valor unidad, y unas raices del polinomio del denominador iguales ambas a -1. Puede también comprobarse a partir de la respuesta en escalón, que el sistema no tiene oscilaciones de ningún tipo. Será más o menos rápido, dependiendo de la frecuencia natural. Se invita al alumno a que realice la misma operación pero con otros sistemas, incluyendo alguno de orden superior.

Análisis de la respuesta temporal de sistemas.

Una vez revisado como es posible en MATLAB introducir sistemas y las operaciones que podemos realizar con ellos, se va a proceder al estudio mediante MATLAB de la respuesta temporal de dichos sistemas, lo que constituye la parte fundamental de la práctica.

Comandos MATLAB para el análisis temporal.

MATLAB incorpora ciertos comandos para la obtención de la respuesta de un sistema LTI ante ciertas entradas características. Dichos comandos ya fueron descritos en la práctica anterior, pero serán brevemente repasados aquí. Recuérdese que si *sysP* es un sistema LTI (obtenido por ejemplo como función de transferencia):

```
>>sysP=tf([4],[1 2 4]);
```

la respuesta del sistema ante un impulso unitario se obtiene con el comando:

```
>>impulse(sysP);
```

Para la respuesta ante escalón se tiene el comando **step** (probar **help step** para obtener más información sobre el funcionamiento del comando):

```
>>step(sysP);
```

Los dos comandos anteriores admiten también como parámetro el vector de tiempos sobre el que queremos que se represente la respuesta, y permiten almacenar la respuesta:

```
>>t=0:.1:10; %Definimos un vector de tiempos
>>y=step(sysP,t); %Representa la salida ante escalón entre 0 y 10
segundos. La respuesta se almacena en y.
```

Por último, la respuesta ante una entrada cualquiera definida por el usuario se puede obtener con el comando **lsim**.

```
>>t=0:.1:10
>>entrada=sin(t);
>>y=lsim(sysP,entrada,t);
>>plot(t,y,'r',t,entrada,'b') % Representamos la salida y la
entrada al mismo tiempo.
```

En este caso, y almacenará la respuesta de $sysP$ ante la entrada definida por los vectores **entrada** y **t**, es decir, en este caso, ante un seno de frecuencia unidad.

Caracterización de la respuesta temporal

Tal como se ha dicho, es interesante caracterizar la respuesta de un sistema ante entradas tipo. Vamos a limitarnos en este caso a la caracterización de la respuesta ante escalón unitario.

Respuesta ante escalón unitario

Para caracterizar la respuesta ante escalón de un sistema, se definen los siguientes parámetros.

- **Sobreoscilación:** Se define como la diferencia entre el máximo valor de pico de la respuesta y el valor de régimen permanente, relativa a dicho valor de régimen permanente (en tantos por ciento).
- **Tiempo de subida:**Tiempo que transcurre entre que la salida alcanza el 10 % del régimen permanente y el 90 % del mismo (es posible definir otros intervalos, como entre el 0 % y el 100 %).
- **Tiempo de pico:**Tiempo que tarda la respuesta en alcanzar su valor de pico (el máximo).
- **Tiempo de retardo:**Tiempo que tarda la salida en alcanzar el 50 % del valor de régimen permanente.
- **Tiempo de establecimiento (5 %):**Tiempo que tarda la respuesta en situarse establemente a menos del 5 % del valor de régimen permanente.

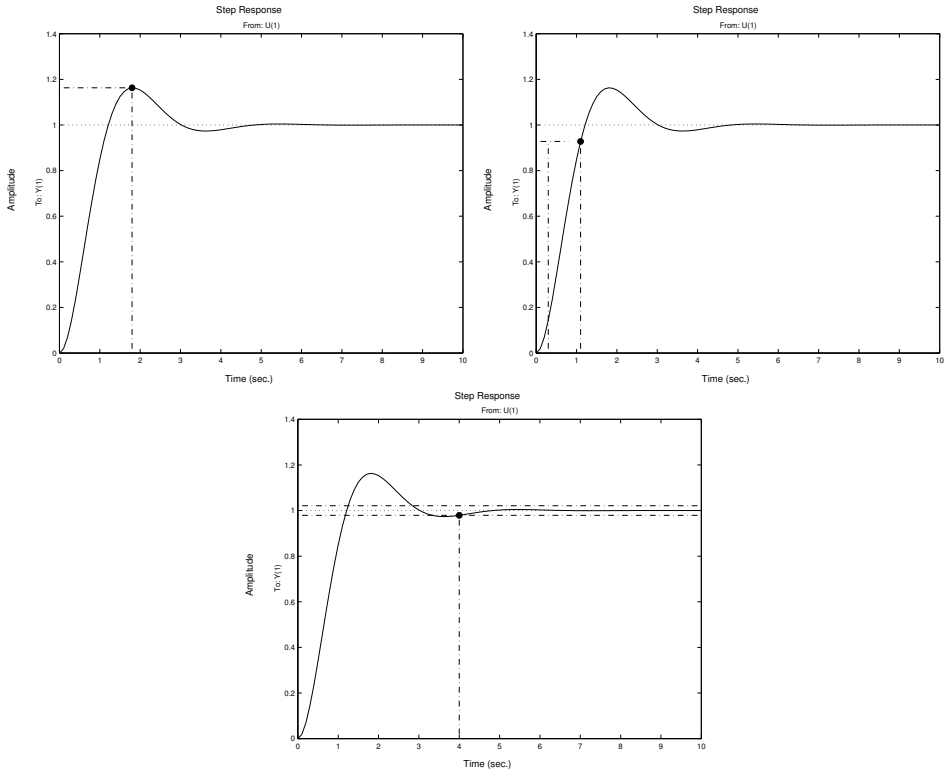


Figura 6: Parámetros relativos a la respuesta ante escalón. Arriba izquierda: sobreoscilación; arriba derecha: tiempo de subida; abajo: tiempo de establecimiento.

La figura 6 muestra de forma gráfica alguno de estos parámetros.

Estos parámetros permiten comparar respuestas de diferentes sistemas, o establecer especificaciones (un ejemplo típico de especificaciones sería por ejemplo el diseño de un sistema de control de modo que el sistema controlado presente una sobreoscilación menor del 10 % y un tiempo de subida de menos de 3 segundos).

Sistemas de segundo orden

Se va a estudiar de forma breve la respuesta ante escalón de sistemas de segundo orden.

La función de transferencia correspondiente a un sistema de segundo orden se puede parametrizar de la siguiente forma:

$$P(s) = \frac{K\omega_n^2}{s^2 + 2\delta\omega_n s + \omega_n^2}$$

donde K es la ganancia, ω_n es la frecuencia natural y δ es el denominado factor de amortiguamiento. La respuesta temporal de un sistema de segundo orden está caracterizada por el valor de dichos parámetros. Analizando la respuesta temporal ante entradas en escalón, podemos ver los siguiente casos (ver figura 7):

- $\delta > 1$ Sistema sobreamortiguado.
- $\delta = 1$ Sistema críticamente amortiguado.
- $0 < \delta < 1$ Sistema subamortiguado.
- $\delta = 0$ Sistema críticamente estable.
- $\delta < 0$ Sistema inestable.

Dado el sistema:

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

determinar su frecuencia natural y factor de amortiguamiento. Generar el modelo en MATLAB y comprobar el tipo de respuesta ante escalón mediante el comando **step**. Variar a continuación el parámetro δ para comprobar los distintas respuestas ante escalón.

También se ha comprobado en teoría la relación existente entre los parámetros de la función de transferencia y los valores característicos de la respuesta

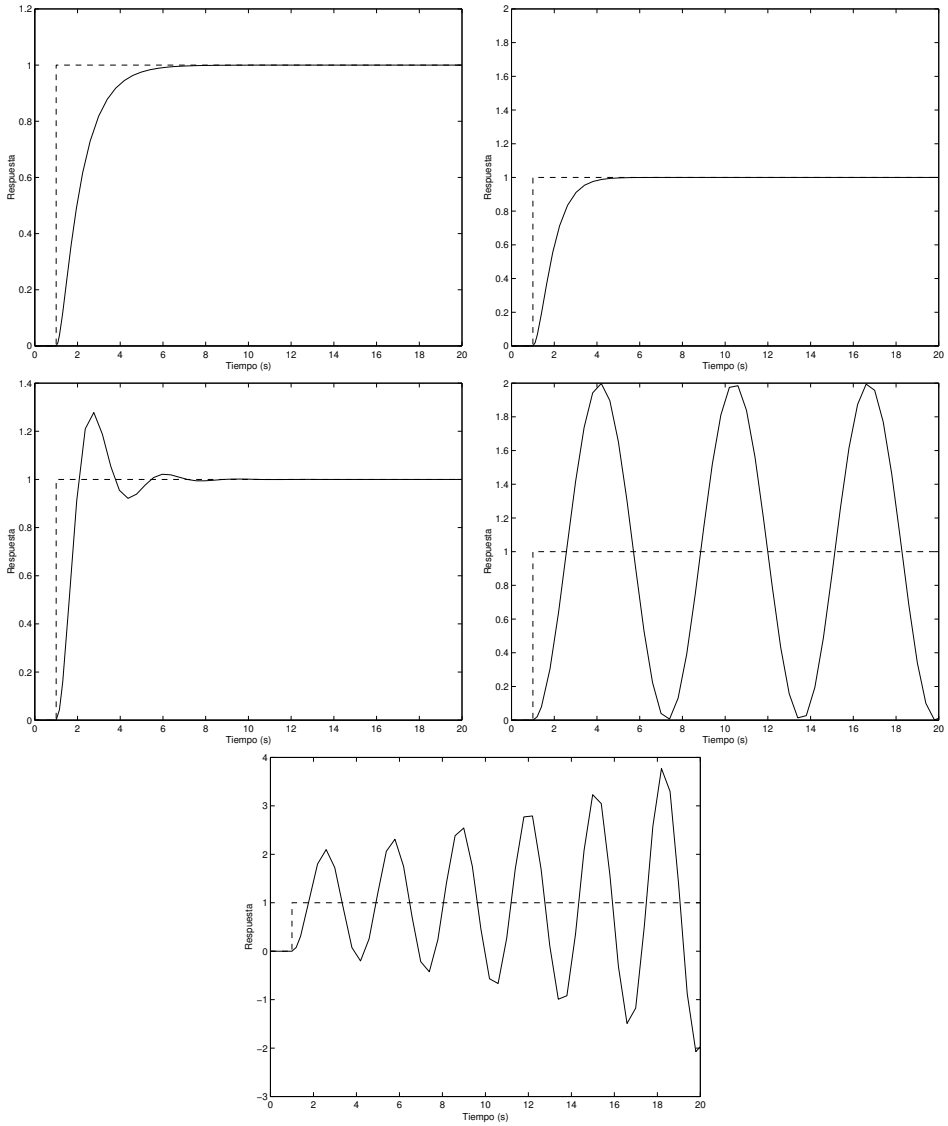


Figura 7: Caracterización de la respuesta ante escalón para un sistema de segundo orden. De izquierda a derecha y de arriba a abajo: Sistema sobreamortiguado, sistema críticamente amortiguado, sistema subamortiguado, sistema críticamente estable y sistema inestable.

ante escalón para sistemas de segundo orden (tiempo de pico, tiempo de subida, sobreoscilación).

$$SO = 100e^{\frac{-\delta\pi}{\sqrt{1-\delta^2}}} \tag{3}$$

$$t_p = \frac{\pi}{\omega_n\sqrt{1-\delta^2}} \tag{4}$$

$$t_{s(0-100)} = \frac{\pi-\varphi}{\omega_n\sqrt{1-\delta^2}} \tag{5}$$

Para el sistema subamortiguado dado por la ecuación 6:

$$G(s) = \frac{9}{s^2+3s+9} \tag{6}$$

determinar empleando MATLAB el tiempo de pico, tiempo de subida y la sobreoscilación, y comparar con los valores teóricos esperados.

Dado el sistema en bucle abierto (ecuación 7)

$$G(s) = \frac{k}{s(s+p)} \tag{7}$$

determinar los valores de k y p para que el sistema en bucle cerrado (realimentación unitaria) presente una sobreoscilación del 20 % y un tiempo de pico de 1 seg. Comprobar los resultados.

También suele resultar de interés la respuesta de un sistema ante una rampa. Determinar la respuesta ante rampa para el sistema anterior (empleando **lsim**). Comprobar que el error en régimen permanente es igual a:

$$\varepsilon = \frac{2\delta}{\omega_n} \tag{8}$$

El análisis temporal de sistemas se puede realizar además con la herramienta **ltiview** que será descrita más adelante.

Respuesta ante entrada sinusoidal

Por último, se va a estudiar la respuesta de sistemas ante entradas sinusoidales, poniendo de manifiesto la relación entre ésta y el análisis frecuencial de sistemas (diagramas de Bode).

Para sistemas lineales e invariantes en el tiempo, ante una entrada sinusoidal como:

$$x(t) = A \cdot sin(\omega t) \tag{9}$$

la respuesta es siempre otra señal sinusoidal de la misma frecuencia, pero que sufre una amplificación (o atenuación) en su amplitud y un cierto desfase. Es decir, la salida $y(t)$ será de la forma:

$$y(t) = \|G(j\omega)\| \cdot A \cdot sin(\omega t + \arg[G(j\omega)]) \tag{10}$$

donde $\|G(j\omega)\|$ no es más que la magnitud en el diagrama de Bode del sistema $G(s)$ para la frecuencia ω, y $\arg[G(j\omega)]$ la fase.

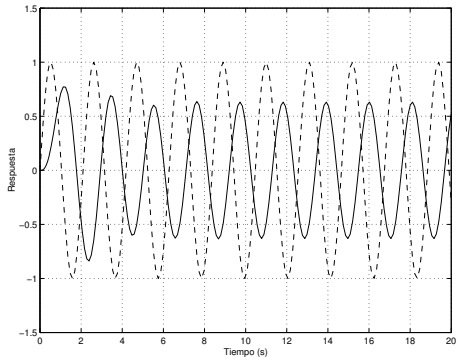


Figura 8: Respuesta ante entrada sinusoidal. Obsérvese que la salida aparece desfasada y atenuada respecto a la entrada (gráfica punteada).

Por tanto, el análisis de la respuesta de un sistema ante entradas sinusoidales de diferentes frecuencias, permite de forma empírica construir el diagrama de Bode de un sistema. Para ello, se anotan la ganancia y el desfase que sufre la entrada frente a la frecuencia de la misma. La ganancia se calcula comprobando la amplitud pico a pico de la entrada y la salida:

$$\|G(j\omega)\| = 20 \log \frac{y_{picopico}}{x_{picopico}} \tag{11}$$

El logaritmo se aplica para convertir la ganancia a decibelios.

El desfase se puede calcular comprobando la diferencia de tiempos entre 2 cruces por cero de la entrada y la salida. Para calcular el desfase correspondiente a dicha diferencia de tiempos no hay más que multiplicar por la frecuencia del seno correspondiente:

$$\arg[G(j\omega)] = \omega \Delta t(radianes) \tag{12}$$

Como ejercicio, el alumno debe calcular el diagrama de bode correspondiente al sistema cuya función de transferencia se define mediante la ecuación:

$$G(s) = \frac{9}{s^2 + 1{,}5s + 9} \tag{13}$$

Para ello, se simulará la respuesta del sistema ante entradas sinusoidales de frecuencias 0.5, 1, 2, 2.5, 3, 5 y 10 rad/s, anotando en cada caso la ganancia en decibelios y el desfase en grados.

Frecuencia, ω (rad/s)	Ganancia (dB)	Desfase (grados)
0.5		
1		
2		
2.5		
3		
5		
10		

El Visor LTI

El visor LTI es un interfaz gráfico utilizado para visulizar y manipular respuestas (tanto temporales como frecuenciales)de modelos de tipo LTI.

En este apartado se va a describir como mostrar y manipular respuestas de un sistema LTI a entradas en escalón e impulso. Las distintas operaciones que se pueden realizar sobre estas representaciones son:

- Cambiar el tipo de representación para cada sistema.
- Hacer aparecer y desaparecer las respuestas de algunos sistemas (muy útil cuando se están representando varios sistemas a la vez)
- Mostrar características importantes de la evolución temporal de cada uno de los sistemas:
  - tiempo de subida (rise time)
  - tiempo de establecimiento(settling time)
  - Valor en Régimen Permanente (Steady State)
  - Valor Máximo (Peak Response)
- Controlar características gráficas como tipos de línea, estilo, ..etc

A continuación, a través de un simple ejemplo se va a mostrar el funcionamiento y las posibilidades de esta herramienta de Matlab.

Supóngase el sistema descrito por la función de transferencia

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

cuyas características son

$$\delta = 0{,}5 \qquad (Factor \quad de \quad Amortiguamiento)$$

$$\omega_n = 1 rad/seg \qquad (Frecuencia \quad Natural)$$

Comience por definir el sistema de la forma

```
>>P=tf(1,[1 1 1])
```

y arrancando posteriormente el visor LTI

```
>>ltiview
```

Para importar el sistema recien definido al visor siga los siguientes pasos:

- Abra el menú **Files**
- Seleccione la acción **Import...**

Aparecerá un recuadro como el que se muestra en la figura 9. En el se muestran todos los sistemas de tipo LTI definidos previamente en el espacio de trabajo, en particular el introducido por nosotros. Seleccionando este, y pulsando 'OK' aparecerá por defecto la respuesta temporal del sistema en lazo abierto ante entrada en escalón.

En el menú 'Tools', se pueden configurar una serie de preferencias tanto para las respuestas como para el estilo de líneas representadas. Asímismo, se podrá elegir la forma de representación de varias gráficas (de haberlas) con la opción 'Configuración del Visor'.

Preferencias en la Respuesta

El cuadro de diálogo correspondiente se muestra en la figura 10, de la cual, para esta práctica, nos interesa la parte izquierda, correspondiente al dominio del tiempo. En ella, se pueden configurar los siguientes parámetros:

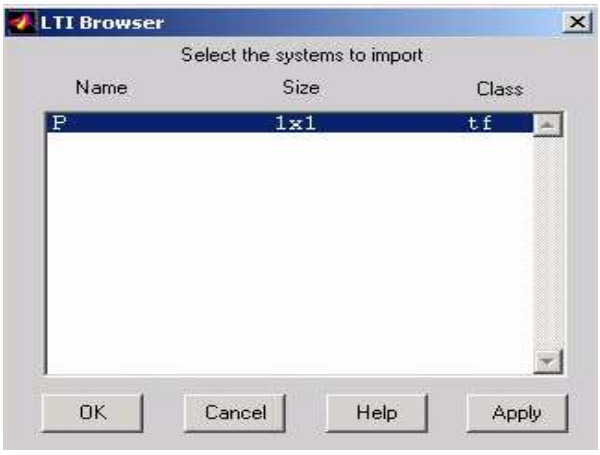


Figura 9: Menú para Importar Sistemas

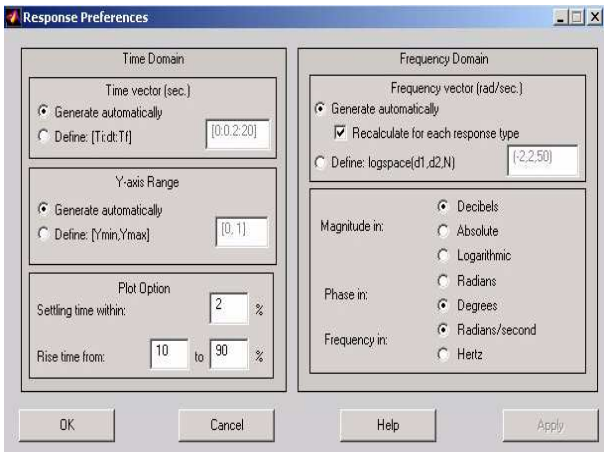


Figura 10: Preferencias en la Respuesta

- **Vector de tiempos:** Hace referencia al rango temporal en el que se representará la figura correspondiente. Puede ser automático (opción por defecto) o introducido manualmente por el usuario en la forma [Instante inicial:Paso:Instante final] en segundos

- **Rango de la Respuesta:** Análogo al caso anterior pero para el eje de ordenadas.

- **Opciones de Respuesta Temporal (Plot Options):**Se definen aquí las tolerancias que utilizará el sistema para calcular los parámetros de la respuesta temporal, tiempo de establecimiento y tiempo de subida

Preferencias en el estilo de Línea

Son una serie de opciones que gestionan el visionado de distintos sistemas simultáneamente encargándose de diferenciarlos en cuanto a color, tipo de línea, etc.

Configuración del Visor

Se puede elegir en esta ventana de diálogo, ver figura 11, la disposición de las distintas gráficas, resultado del análisis de uno o varios sistemas. Se pueden visualizar hasta seis simultáneamente, eligiendo en la parte derecha de la ventana que tipo de respuesta se quiere representar. *Ejercicio* Definir otro sistema distinto y mostrar los dos simultáneamente cambiando las opciones de representación.

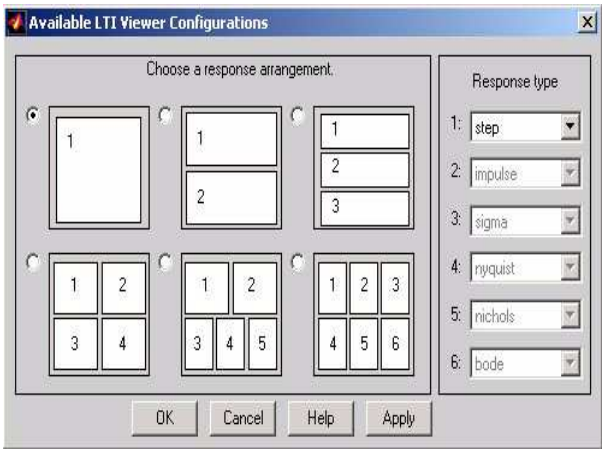


Figura 11: Configuración del Visor

Por su parte, las otras opciones del menú **files** son las siguientes:

- **New Viewer:** Se encarga de abrir un nuevo visor.
- **Import..:**Ya explicado anteriormente. Sirve para trabajar en el visor con sistemas definidos en el espacio de trabajo del MATLAB
- **Export..:**Realiza la operación inversa, es decir, manda un sistema desde el visor al espacio de trabajo o a un disco.
- **Delete System:**Borra un sistema del visor y del espacio de trabajo.
- **Refresh System:**Sirve para actualizar en el visor un sistema que previamente haya sido modificado en el espacio de trabajo.

- **Print:** Manda la pantalla del Visor a la impresora
- **Print to Figure:**Genera una figura de MATLAB ($*.fig$) con lo que haya representado en el visor.
- **Close Viewer:**Cierra el Visor.

Adicionalmente, pulsando con el botón derecho del ratón sobre la pantalla de representación, se abre un recuadro de diálogo con las siguientes opciones:

- **Plot Type:** Permite elegir que tipo de respuesta del sistema se quiere representar (En el ámbito de esta práctica solo interesaran 'step' e 'impulse'). Se puede realizar lo mismo desde el menú **Tools ¿Configuración del Visor**
- **Systems:** Permite activar/desactivar la representación de los sitemas mostardos en cada momento
- **Characteristics:**Activa/Desactiva las características de la respuesta temporal del sistema
  - Respuesta Máxima (Peak Response)
  - Tiempo de Establecimiento (Settling Time)
  - Tiempo de Subida (Rise Time)
  - Valor en Régimen Permanente (Steady state)

  Téngase en cuenta que los tiempos de subida y el de establecimiento vienen condicionados por los parámetros con que se configuraron en el menú de preferencias de la respuesta.

  En la figura 12 puede verse la representación de la respuesta ante escalón del sistema, en la que se han marcado todas las características anteriores. Haciendo click con el botón izquierdo del ratón en cada uno de estos puntos aparecerá el correspondiente valor numérico.
- **Zoom:**Como su propio nombre indica, permite realizar un acercamiento por recuadro (X-Y) o por eje (In-X, In-Y)

Ejemplo de Aplicación

Para la siguiente planta que representa un sistema de segundo orden en lazo abierto, se pide:

$$P(s) = \frac{2}{s^2 + 0{,}8s + 4}$$

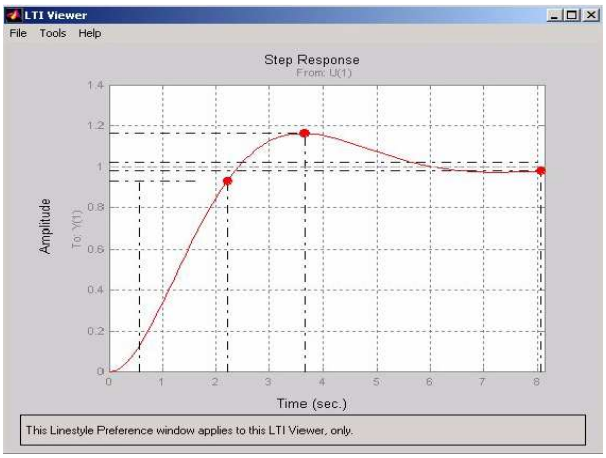


Figura 12: Características Temporales

- ANÁLISIS DEL SISTEMA SIN CONTROLAR

  - Representar en una misma ventana el sistema en lazo abierto y con realimentación unitaria (sin controlador) cuando se proporciona a la entrada un escalón unitario.
  - Comprobar analíticamente los valores obtenidos gráficamente para la sobreoscilación,tiempo de subida, y valor en régimen permanente de los sistemas en lazo abierto y lazo cerrado.
  - A partir de las características obtenidas en lazo cerrado, ¿Cúal es el error en régimen permanente del sistema?

- DISEÑO DEL CONTROL

  Se pretende diseñar un controlador de forma que el sistema en lazo cerrado tenga un tiempo de subida comprendido entre 1 y 2 segundos y una sobreoscilación entre 0 y 10 %. Asímismo, el error en régimen permanente permitido es de ± 5 %.

  - Utilizar el visor LTI para representar 2 curvas de segundo orden que representen los límites inferior y superior propuestos.
  - Sintonizar un controlador que permita cumplir las especificaciones anteriores y representar la curva resultante con todas sus características.

PRÁCTICA 7 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Manuel López Martínez

Agradecimientos a Manuel Berenguel Soria

Análisis y Control de Sistemas usando MATLAB

1.1. Introducción

En lo que sigue, se va a realizar una introducción a los comandos de MATLAB relacionados con la teoría de control de sistemas. Casi todas las funciones que se describen pertenecen al Control System Toolbox. Dichas funciones se van a describir en la forma en que vienen dadas en la versión 4.0 de MATLAB o superior. Para versiones inferiores, la sintaxis es la misma, cambiando sólo la forma de representación gráfica y las posibilidades que ofrece cada comando. Las funciones principales se van a explicar sobre ejemplos demostrativos, con el fin de que su uso y comprensión sean lo más sencillas posibles. Se realizarán ejemplos tanto en dominio continuo como en tiempo discreto y finalmente se dispondrá de tablas donde consultar los comandos (normalmente los referentes a tiempo discreto son los mismos de tiempo continuo con una d delante). En la mayoría de los casos se harán los ejemplos para tiempo continuo, haciendo alguna alusión a los comandos equivalentes para tiempo discreto cuando sea necesario.

1.2. Comandos relacionados con la teoría clásica de Control

Este apartado muestra el uso de algunas de las herramientas con las que cuenta MATLAB para el diseño y análisis de sistemas de control. Para el ejemplo se va a partir de una descripción de la planta en forma de función de transferencia:

$$G(s) = \frac{0{,}2s^2 + 0{,}3s^2 + 1}{(s^2 + 0{,}4s + 1)(s + 0{,}5)}$$

En MATLAB las funciones de transferencia se introducen en la forma numerador-denominador:

```
num=[.2 .3 1];
den1=[1 .4 1];
den2=[1 .5];
```

El polinomio del denominador es el producto de dos términos. Para obtener el polinomio resultante se usa el producto de convolución (o de polinomios).

```
den = conv( den1, den2)
```

Dominio Temporal

La respuesta ante un escalón a la entrada se puede analizar en sistemas que tengan una descripción en forma de función de transferencia o una representación en el espacio de estados, generando un vector de tiempos y usando la función **step**:

```
t = 0:.3:15; y=step( num, den, t );

plot(t,y) , title ('Respuesta a un escalon unitario' )
xlabel('tiempo(seg)' )
```

La respuesta al escalón unitario puede verse en la Fig.1.1

En las nuevas versiones para WINDOWS no es necesario generar el vector de tiempos y MATLAB automáticamente escala la gráfica. Tanto para este comando como para todos, en dichas versiones, si no se pone argumento de salida (en este caso y), el resultado es una gráfica con la respuesta, por lo que no sería necesario hacer después un plot. Los títulos, anotaciones en ejes, etc. aparecen por defecto en inglés.

La respuesta impulsional se puede obtener del mismo modo, pero usando en este caso la función **impulse**, que tiene una sintaxis similar al comando **step**. Nótese que el vector de tiempos ya ha sido definido con anterioridad y no es necesario volver a definirlo pues queda como variable de entorno.

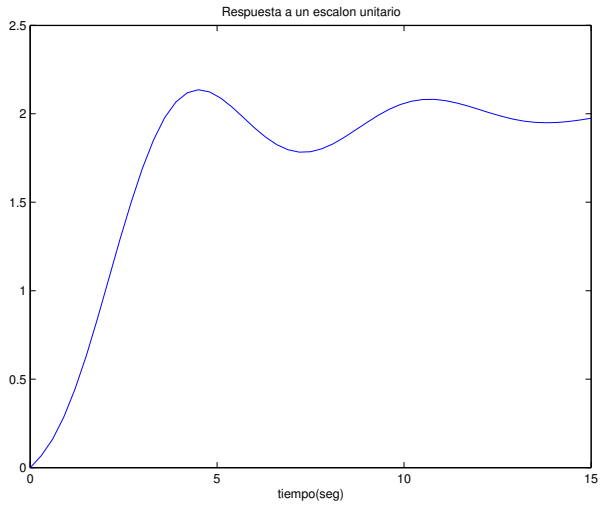


Figura 1.1: Respuesta al escalón unitario

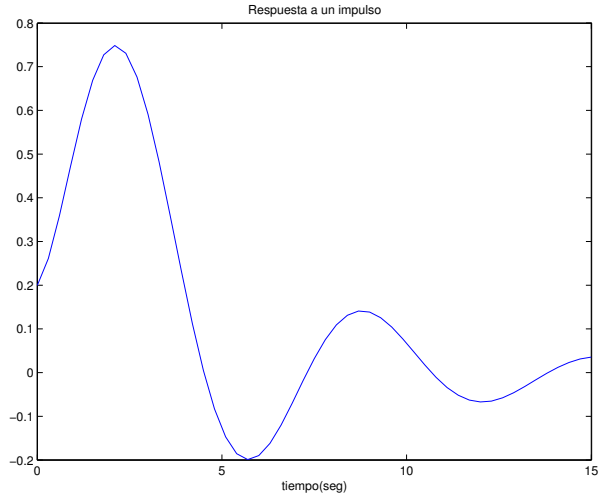


Figura 1.2: Respuesta a un impulso

```
y=impulse(num,den,t) ; plot(t,y), title('Respuesta a un impulso'),
xlabel('tiempo(seg)')
```

La respuesta al impulso puede verse en la Fig.1.2

La respuesta del sistema a cualquier tipo de entrada también puede obtenerse. Para ello es necesario tener el conjunto de entradas en un vector u, que lógicamente debería tener la misma dimensión que el vector de tiempos. Si el tamaño de este vector es menor, sólo se computan las salidas correspondientes. En sistemas multivariables, en vez de un vector de entradas tendremos una matriz. A estos efectos se usa la función **lsim**. Un ejemplo característico es la respuesta a una entrada en rampa:

```
ramp=t; y=lsim(num,den,ramp,t); plot(t,y,t,ramp),

title('Respuesta a una rampa')
xlabel('tiempo(seg)')
```

La respuesta a la rampa puede verse en la Fig.1.3

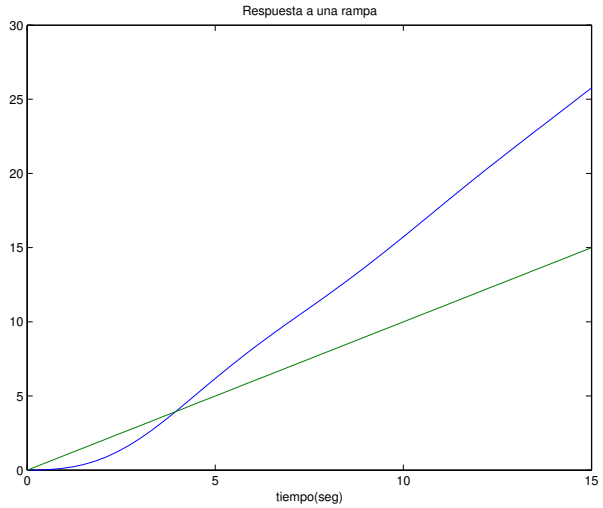
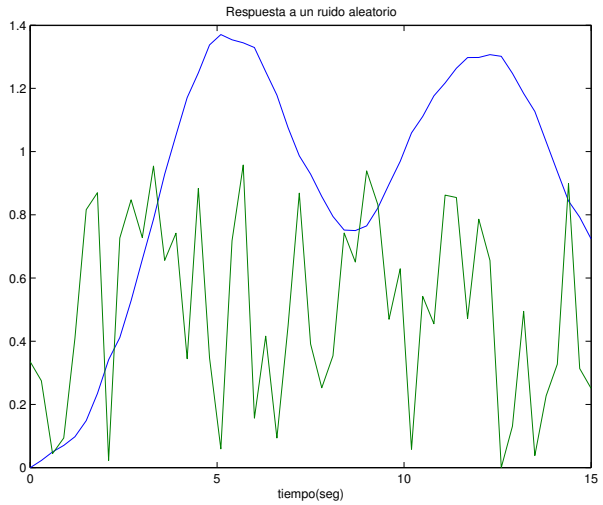


Figura 1.3: Respuesta a una rampa

Otro ejemplo característico es la respuesta a un ruido uniforme aleatorio. Recordamos que r and(m, n) genera una matriz m x n de números aleatorios uniformemente distribuidos entre 0 y 1. Es importante darse cuenta del filtrado que se produce en la señal cuando pasa por el sistema (hace de filtro de altas frecuencias).

```
noise=rand(length(t),1);
y=lsim(num,den,noise,t);
plot(t,y,t,noise),title('Respuesta a un ruido aleatorio'),...
xlabel('tiempo(seg)');
```

La respuesta al ruido puede verse en la Fig.1.4



Comandos relacionados con operaciones de bloques

Otro comando muy útil es el que proporciona la realimentación unitaria para un sistema en bucle abierto, dando su representación en bucle cerrado con realimentación unitaria:

```
[nbc,dbc]=cloop(num,den,-1)
```

donde **nbc**, **dbc** son respectivamente el numerador y denominador de la función de transferencia en bucle cerrado, **num**, **den** los de bucle abierto y -1 indica que la realimentación es negativa (signo de la realimentación).

1.3. Estudio Temporal de Sistemas de Primer Orden

La representación en forma de función de transferencia viene dada por

$$G(s) = \frac{K}{1 + \tau \cdot s}$$

que en notación MATLAB se introduce

```
K=1;
T=1;
num=K;
den=[T 1];
```

La respuesta a un escalón unitario de entrada se obtiene con la función **step**. El resultado puede verse en la Fig.1.13

```
t=0:0.1:10;
ye=step(num,den,t);
plot(t,ye),title('Respuesta a un escalon unitario'),
xlabel('tiempo(seg)'), grid;
```

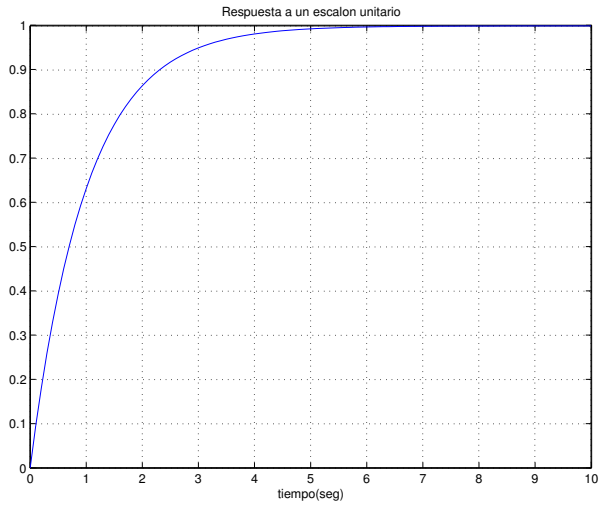


Figura 1.13: Respuesta a escalón unitario del sistema de primer orden

Las dos características fundamentales de un sistema de primer orden son su ganancia estática **K** y su constante de tiempo $\boldsymbol{\tau}$. La constante de tiempo

es el tiempo que tarda en alcanzar el 63 % de la salida. La ganancia estática es el cociente entre la amplitud de salida y la de entrada en el régimen permanente. Estos valores se pueden comprobar directamente en la gráfica o analizando el fichero de datos resultante.

```
y_rp=ye(length(ye))/1. % Regimen Permanente
n=1
while ye(n)<0.63
   n=n+1;
end
ctet_aprox= 0.1*n; % Constante de tiempo
```

La respuesta a una rampa unitaria de entrada en los sistemas de primer orden se puede calcular:

```
ramp=t;
yr=lsim(num,den,ramp,t);
plot(t,yr,t,ramp),
title('Respuesta a una rampa'),
xlabel('tiempo(seg)'),grid;
```

El resultado se muestra en la figura 1.14

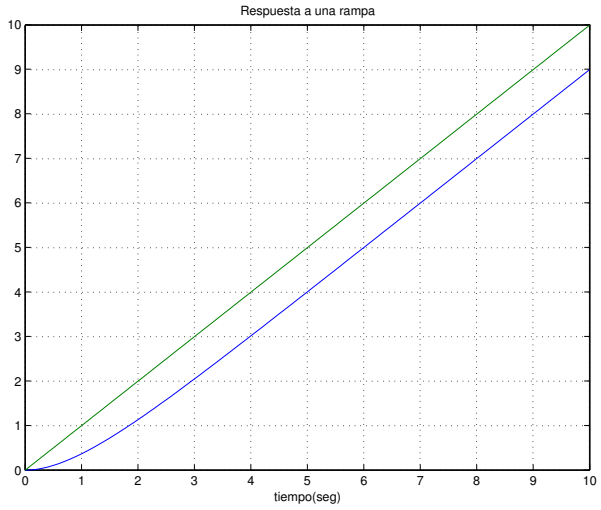


Figura 1.14: Respuesta a rampa unitaria del sistema de primer orden

El error de seguimiento en posición en régimen permanente puede obtenerse, al igual que se ha hecho anteriormente, midiendo directamente en la gráfica o bien a partir del fichero de datos resultante.

La respuesta impulsional se puede obtener del mismo modo, pero usando en este caso la función **impulse** (Fig.1.15), que tiene una sintaxis similar al comando **step**.

```
yi=impulse(num,den,t);
plot(t,yi),title('Respuesta a un impulso'),xlabel('tiempo(seg)')
```

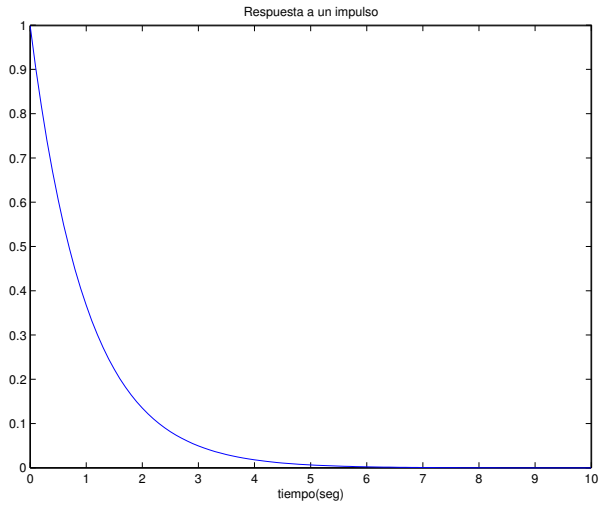


Figura 1.15: Respuesta a impulso del sistema de primer orden

Para finalizar con la sección de sistemas de primer orden, se va a comprobar que los resultados coinciden con los esperados en la teoría. Como es bien sabido, los sistemas lineales de primer orden de ganancia unidad invariantes en el tiempo tienen las siguientes características (nos remitimos a la bibliografía):

Respuesta a escalón unitario: $C_e(t) = 1 - e^{(-t/\tau)}, (t \geq 0)$

Respuesta a rampa unitaria: $C_r(t) = t - \tau + \tau \cdot e^{(-t/\tau)}, (t \geq 0)$

Respuesta a impulso: $C_i(t) = (1/\tau)e^{(-t/\tau)}, (t \geq 0)$

Si se dibujan estas funciones con el vector de tiempos definido anteriormente, se puede observar que se obtiene el mismo resultado (Fig.1.16

```
Ce=1-exp(-t/T);
Cr=t-T+T*exp(-t/T);
Ci=(1/T)*exp(-t/T);
plot(t,Ce,t,ye,'o'),
```

```
title('Respuesta teorica a escalon unitario'),grid,
figure, plot(t,Cr,t,ramp,t,yr,'o'),
title('Respuesta teorica a rampa unitaria'),grid,
figure, plot(t,Ci,t,yi,'o'),
title('Respuesta teorica a impulso'),grid;
```

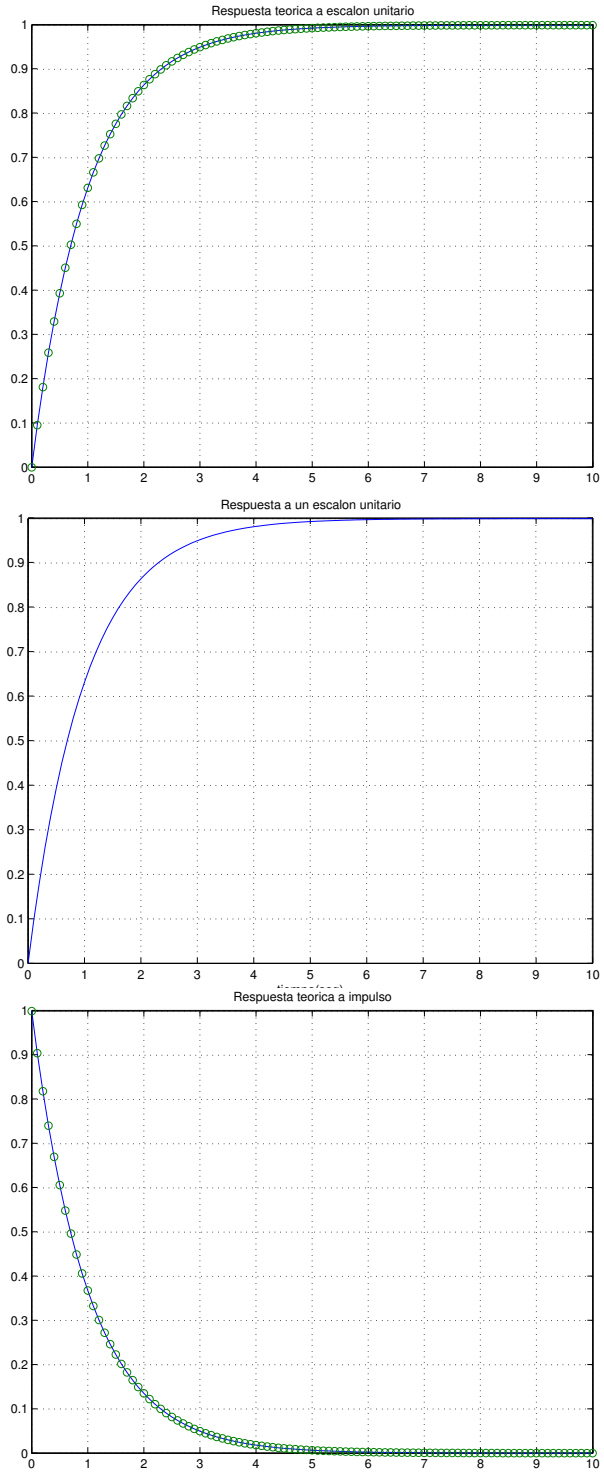


Figura 1.16: Respuesta a escalón, rampa e impulso del sistema de primer orden

PRÁCTICA 8 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Angel Rodríguez Castaño
Ismael Alcalá Torrego

Respuesta temporal de un servomecanismo de posición

Material necesario

Para la realización de la práctica será necesario el enunciado de la práctica anterior y los apuntes de la asignatura correspondientes a la respuesta temporal de sistemas.

Introducción

La finalidad de la práctica consiste en estudiar la respuesta de un servomecanismo de posición en el dominio del tiempo. Mediante dicho estudio, se identificará el sistema dinámico que tendrá especial relevancia para las fases posteriores de diseño de controladores.

El modelo del servomecanismo

El servomecanismo de posicionamiento objeto de estudio, viene expresado como un bloque de SIMULINK con una entrada (la tensión aplicada, u_e) y dos salidas (la posición y velocidad angulares, θ_c y ω_c).

Para abrir el modelo se procederá siguiendo los siguientes pasos:

1. Una vez ejecutada la aplicación Matlab, se cambiará al directorio donde se encuentra el modelo escribiendo en la línea de comandos

   ```
   >> cd nombre_del_directorio
   ```

   o bien, en la ventana para cambiar de directorio que aparece en la figura 1.

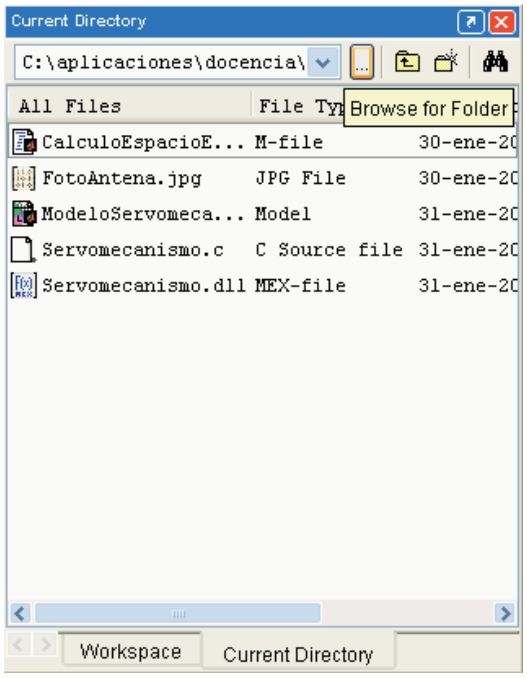


Figura 1: Ventana para cambiar el directorio de trabajo

2. a continuación, se ejecutará el comando de Matlab `dir` y se comprobará que nos encontramos efectivamente en dicho directorio si aparecen los ficheros `ModeloServomecanismo.mdl` y `Servomecanismo.dll`

3. Escribir en la línea de comandos de Matlab el nombre del fichero `ModeloServomecanismo.mdl`, de modo que aparezca en pantalla el sistema en bucle abierto de la figura 2.

4. Pulsar dos veces sobre el bloque del servomecanismo e introducir los cuatro últimos números del DNI (ver figura 3).

5. Cerrar la ventana de configuración y comenzar a estudiar la respuesta temporal.

Objetivos de la práctica

En la práctica anterior se han descrito comandos y fórmulas[1] para estudiar mediante simulación, la respuesta temporal de sistemas. Dichos conceptos, serán necesarios para el desarrollo de los objetivos de la práctica:

[1] $SO = 100e^{\frac{-\delta\pi}{\sqrt{1-\delta^2}}}$ $t_p = \frac{\pi}{\omega_n\sqrt{1-\delta^2}}$ $t_{s(0-100)} = \frac{\pi-\varphi}{\omega_n\sqrt{1-\delta^2}}$

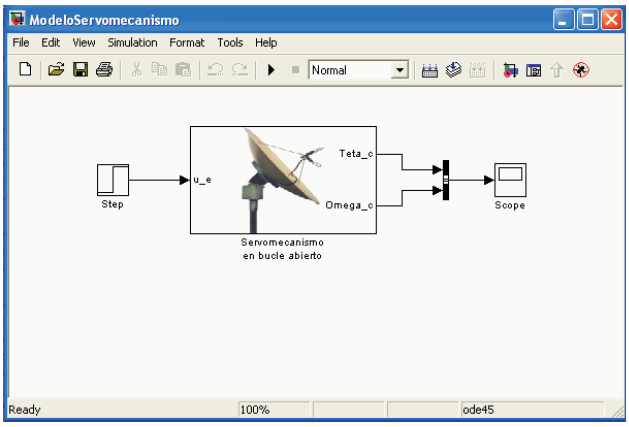


Figura 2: Diagrama de bloques del servomecanismo en SIMULINK.

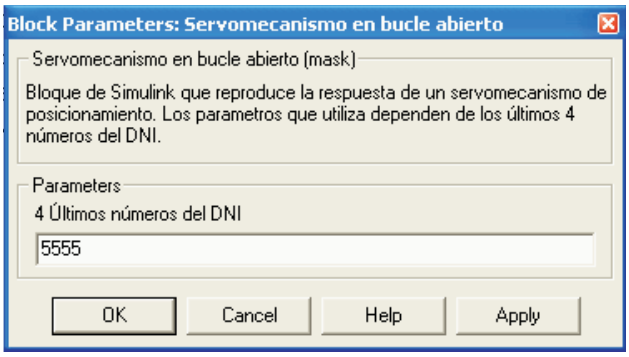


Figura 3: Configuración del bloque que representa al servomecanismo de posicionamiento.

1. Buscar los parámetros de la función de transferencia $\frac{\omega_c(s)}{u_e(s)}$. Para ello, estudiar la respuesta de la velocidad angular en la carga ω_c ante un escalón en la tensión de entrada u_e.

2. Función de transferencia del servomecanismo $G(s) = \frac{\theta_c}{u_e}$.

3. ¿Por qué no se intenta identificar directamente la respuesta de la posición angular en la carga θ_c ante un escalón en la tensión de entrada u_e?

4. Error en régimen permanente ante rampa para el sistema con realimentación unitaria de la posición. Obtenerlo respecto a cada una de las salidas del servomecanismo por simulación y de forma analítica (utilizando la función de transferencia identificada).

PRÁCTICA 9 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Angel Rodríguez Castaño
Ismael Alcalá Torrego

Respuesta frecuencial de un servomecanismo de posición

Material necesario

Para la realización de la práctica será necesario el enunciado de la práctica anterior y los apuntes de la asignatura correspondientes a la respuesta frecuencial de sistemas.

Introducción

La finalidad de la práctica consiste en estudiar la respuesta frecuencial del servomecanismo de posición. Mediante dicho estudio, se identificará el sistema dinámico que tendrá especial relevancia para las fases posteriores de diseño de controladores con técnicas frecuenciales.

El modelo del servomecanismo

El servomecanismo de posicionamiento objeto de estudio es el utilizado en la práctica anterior. Está construido con un bloque de SIMULINK con una entrada (la tensión aplicada, u_e) y dos salidas (la posición y velocidad angulares, θ_c y ω_c).

Objetivos de la práctica

En las clases teóricas se han descrito los conceptos básicos del análisis de sistemas en el dominio de la frecuencia. En esta práctica se van a aplicar dicha teoría a la construcción del diagrama de Bode de un sistema servomecanismo con función de transferencia desconocida. El análisis se va a efectuar

por simulación, aplicando diferentes entradas senoidales al servomecanismo y observando la salida en posición del sistema.

Se pide:

- Dibujar el diagrama de Bode del servomecanismo $\frac{\theta_c(s)}{u_e(s)}$. Para ello, estudiar la respuesta de la posición angular θ_c ante una senoide de distintas frecuencias en la tensión de entrada u_e. Anotar la ganancia y el desfase para poder representar el diagrama de Bode del servomecanismo.

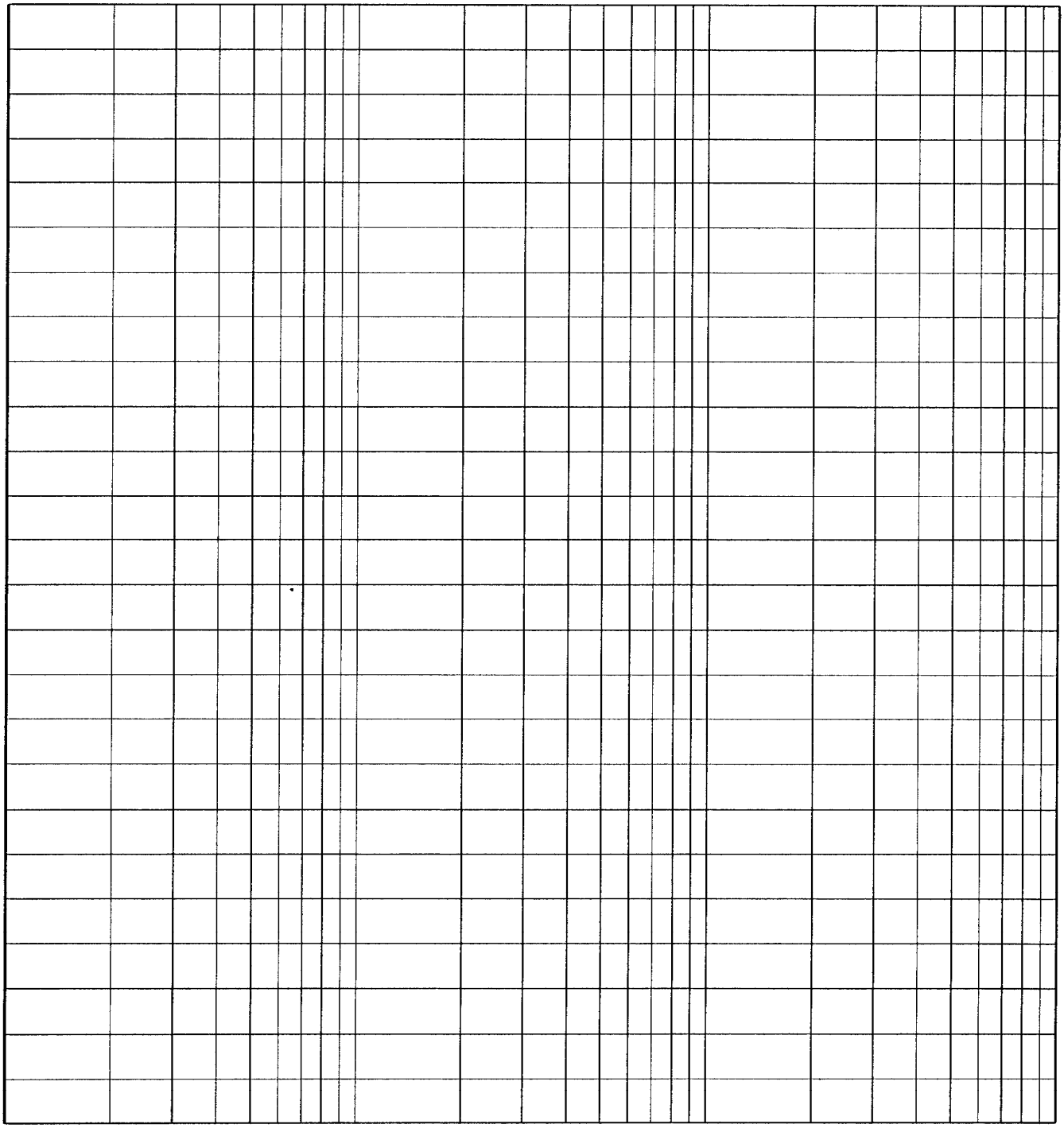


Diagrama de Bode del servomecanismo

PRÁCTICA 10 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Manuel López Martínez

Agradecimientos a Manuel Berenguel Soria

Análisis y Control de Sistemas usando MATLAB

1.1. Introducción

En lo que sigue, se va a realizar una introducción a los comandos de MATLAB relacionados con la teoría de control de sistemas. Casi todas las funciones que se describen pertenecen al Control System Toolbox. Dichas funciones se van a describir en la forma en que vienen dadas en la versión 4.0 de MATLAB o superior. Para versiones inferiores, la sintaxis es la misma, cambiando sólo la forma de representación gráfica y las posibilidades que ofrece cada comando. Las funciones principales se van a explicar sobre ejemplos demostrativos, con el fin de que su uso y comprensión sean lo más sencillas posibles. Se realizarán ejemplos tanto en dominio continuo como en tiempo discreto y finalmente se dispondrá de tablas donde consultar los comandos (normalmente los referentes a tiempo discreto son los mismos de tiempo continuo con una d delante). En la mayoría de los casos se harán los ejemplos para tiempo continuo, haciendo alguna alusión a los comandos equivalentes para tiempo discreto cuando sea necesario.

1.2. Comandos relacionados con la teoría clásica de Control

Este apartado muestra el uso de algunas de las herramientas con las que cuenta MATLAB para el diseño y análisis de sistemas de control. Para el ejemplo se va a partir de una descripción de la planta en forma de función de transferencia:

$$G(s) = \frac{0{,}2s^2 + 0{,}3s^2 + 1}{(s^2 + 0{,}4s + 1)(s + 0{,}5)}$$

En MATLAB las funciones de transferencia se introducen en la forma numerador-denominador:

```
num=[.2 .3 1];
den1=[1 .4 1];
den2=[1 .5];
```

El polinomio del denominador es el producto de dos términos. Para obtener el polinomio resultante se usa el producto de convolución (o de polinomios).

```
den = conv( den1, den2)
```

Dominio Frecuencial

Respuesta en frecuencia

La respuesta en frecuencia de los sistemas se puede obtener usando las funciones **bode**, **nyquist** y **nichols**. Si no se le ponen argumentos a la izquierda, estas funciones generan gráficas; en caso contrario vuelcan los datos en los vectores oportunos. Ejemplos: Las tres posibles sintaxis de la función **bode** son:

```
1.- bode(num,den)
2.- [mag,phase,w]=bode(num,den)
3.- [mag,phase]=bode(num,den,w)
```

La primera de ellas produce un gráfico con la magnitud en dB y la fase en grados. En las otras dos la magnitud y la fase se devuelven en vectores (la magnitud en este caso no va en dB). La segunda forma automáticamente

genera los puntos de frecuencia en un vector. En la tercera forma es el usuario el que escoge los rangos de frecuencia (muy adecuado cuando se quieren representar varias gráficas conjuntamente).

El resultado de los dos últimos comandos se puede representar usando funciones conocidas:

```
subplot(211),loglog(w,mag),title('Modulo'),xlabel('rad/s'),
subplot(212),semilogx(w,phase), title('Fase'),xlabel('rad/s'),
subplot
```

(también se puede hacer directamente llamando sin argumentos de salida). El resultado para la función de transferencia del ejemplo anterior puede verse en la Fig.1.5

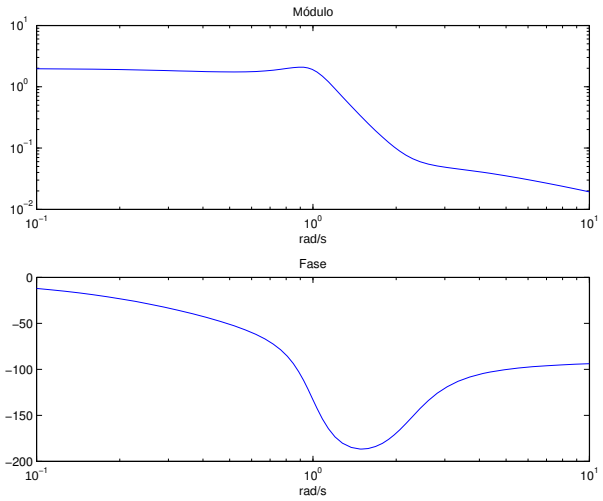


Figura 1.5: Diagrama de Bode

El comando **nyquist** tiene la misma sintaxis:

```
[re,im]=nyquist(num,den,w);
```

El comando nyquist calcula las partes real e imaginaria de $G(jw)$. Para obtener la representación gráfica sólo hay que dibujar la parte real frente a la imaginaria o no poner argumentos de salida a la función.

```
nyquist(num,den,w);
title('Nyquist'),xlabel('Re(G(jw))'),...
ylabel('Im(G(jw))'),grid;
```

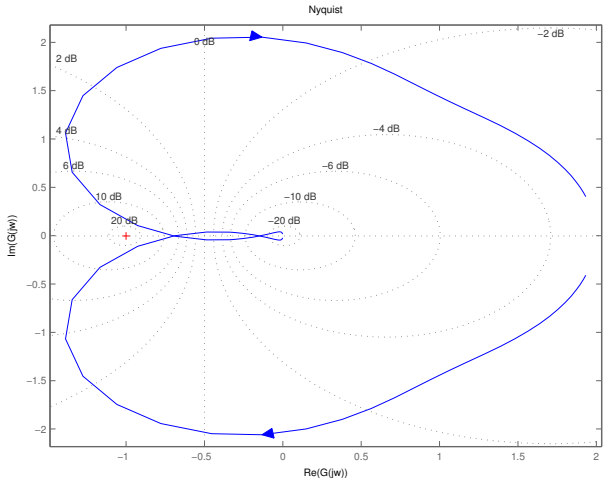


Figura 1.6: Diagrama de Nyquist

El resultado para el ejemplo citado puede verse en la Fig1.6.

El comando **nichols** calcula el diagrama de Nichols de un sistema a partir de la función de transferencia en bucle abierto. Para verlo basta dibujar la magnitud del bucle abierto en dB (en el eje de ordenadas) frente a fase del bucle abierto en grados (en eje de abcisas) o llamar a la función sin argumento de salida. Si se quiere en forma de ábaco, se puede usar el comando ngrid (ver Fig.1.7)

```
ngrid, nichols(num,den,w);
[mag,phase]=nichols(num,den,w);
 magdb=20*log10(mag);
```

Márgenes de estabilidad

Como es bien sabido en la teoría clásica del control, los márgenes de estabilidad son el margen de fase y el margen de ganancia. Estos márgenes se calculan usando el comando **margin**, cuyos argumentos son la magnitud (no en dB), fase y vector de frecuencias obtenidos con los comandos anteriores.

```
[gm,pm,wpc,wgc]=margin(mag,phase,w);
```

Como salidas se obtienen el margen de ganancia (no en dB), el margen de fase (en grados) y sus correspondientes frecuencias. Si se usa otra sintaxis:

```
margin(mag,phase,w);
```

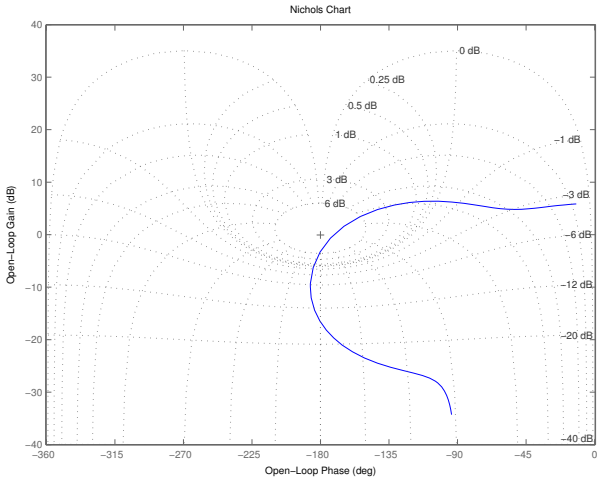


Figura 1.7: Diagrama de Nichols

genera un diagrama de Bode con los márgenes marcados en forma de líneas verticales. Si existen varias frecuencias de corte marca los más desfavorables(ver Fig.1.8).

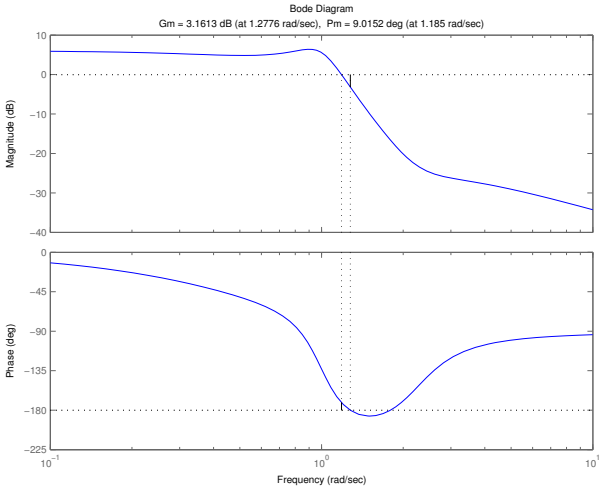


Figura 1.8: Márgenes de Fase y Ganancia

Efectos de los retardos

Los retardos existen en numerosas aplicaciones de control automático. En sistemas lineales continuos invariantes en el tiempo, el retardo viene representado por $e^{-T \cdot s}$. Los sistemas con retardo pueden ser analizados convenien-

temente en el dominio de la frecuencia. Nótese que $e^{-T \cdot s} = 1|\underline{-wt}$. Por tanto los retardos dejan la magnitud invariable y afectan al desfase, tendiendo a inestabilizar al sistema controlado.

Para propósitos de simulación, si se utiliza el comando **bode**, todo lo que habrá que hacer es restar la fase del retardo a la de la función de transferencia (ver Fig.1.9). Por ejemplo:

```
num2=2; den2=[1 1]; Td=1;
w=logspace(-2,0.7,100)';
[m,p]=bode(num2,den2,w);
pd=p-(Td*w*180/pi); %fase en grados
subplot(211), semilogx(w,20*log10(m)),grid,title('M\'{o}dulo (dB)');
subplot(212), semilogx(w,[p pd]),grid,title('Fase (grados)');
subplot
```

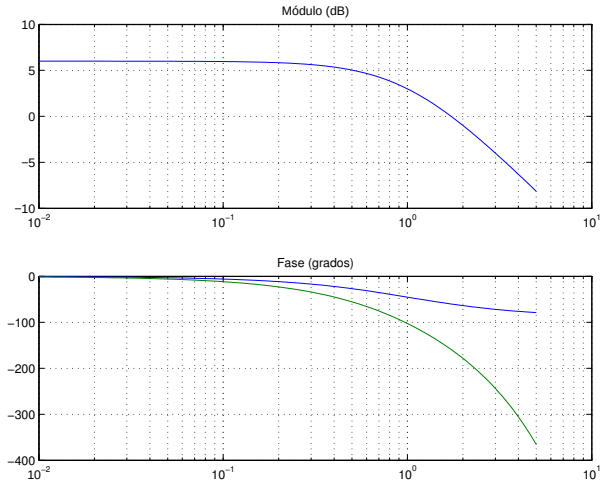


Figura 1.9: Efecto del retardo

Comandos relacionados con operaciones de bloques

Otro comando muy útil es el que proporciona la realimentación unitaria para un sistema en bucle abierto, dando su representación en bucle cerrado con realimentación unitaria:

```
[nbc,dbc]=cloop(num,den,-1)
```

donde **nbc**, **dbc** son respectivamente el numerador y denominador de la función de transferencia en bucle cerrado, **num**, **den** los de bucle abierto y -1 indica que la realimentación es negativa (signo de la realimentación).

1.3. Estudio Frecuencial de sistemas de primer orden

La representación en forma de función de transferencia viene dada por

$$G(s) = \frac{K}{1 + \tau \cdot s}$$

que en notación MATLAB se introduce

```
K=1;
T=1;
num=K;
den=[T 1];
```

A continuación se van a dibujar los diagramas de Bode y Nyquist para el sistema de primer orden (ver figura).

```
figure, margin(num,den);
figure, nyquist(num,den);
```

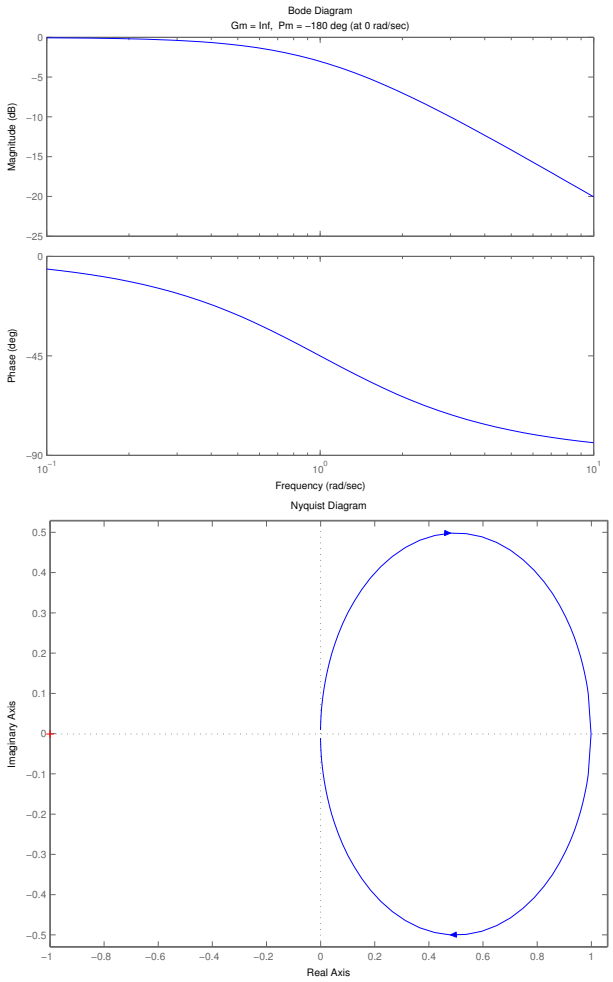


Figura : Diagramas de Bode y Nyquist

PRÁCTICA 11 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Daniel Jiménez
Luis Merino Cabañas

Agradecimientos a Manuel Berenguel Soria

Control de un servomecanismo de posición basado en la respuesta frecuencial

Descripción del servomecanismo y especificaciones

En las prácticas anteriores se han determinado las respuestas temporal y frecuencial de un servomecanismo de posicionamiento. Dicho análisis nos ha permitido construir un modelo matemático (función de transferencia) del sistema, modelo sobre el que vamos a trabajar en esta práctica. El objetivo es el diseño de controladores mediante técnicas frecuenciales, empleando el modelo obtenido. Los resultados los comprobaremos realizando simulaciones con el modelo Simulink

Para el desarrollo de la práctica será necesario aplicar los resultados teóricos estudiados sobre diseño de controladores empleando métodos frecuenciales. Recuérdese que el objetivo de estas técnicas es modificar el diagrama de Bode del sistema controlado, de modo que se cumplan las especificaciones de margen de fase, margen de ganancia y otras dadas. Si las especificaciones vienen dadas en el dominio del tiempo (tiempo de subida, sobreoscilación, etc) habrán, por tanto, de ser convertidas previamente a especificaciones en el dominio de la frecuencia.

Descripción del sistema

El sistema bajo estudio corresponde al sistema descrito en las prácticas anteriores

Asimismo, el sistema corresponde con el descrito en el trabajo de curso, por lo que la práctica puede aprovecharse para el desarrollo de alguna de las tareas requeridas en el mismo.

Especificaciones

A lo largo de la práctica, trataremos de diseñar controladores de modo que el sistema con realimentación de posición presente las siguientes características:

1. Error de seguimiento en velocidad menor o igual al 0,1 %
2. Sobreoscilación menor o igual al 30 %
3. Tiempo de subida menor o igual a 0,1 segundos.

Breve repaso a la caracterización temporal de controladores PD, PI y PID.

Brevemente se va a describir la respuesta en frecuencia de los controladores PD, PI y PID, que aportará algunas claves para el diseño en el dominio de la frecuencia con este tipo de controladores.

Respuesta frecuencial de una red PD

Ya se ha visto que en el caso de compensación PD, la señal de control es proporcional al error y su derivada. La función de transferencia de una red PD se puede escribir como:

$$G(s) = K(1 + \tau s) \tag{1}$$

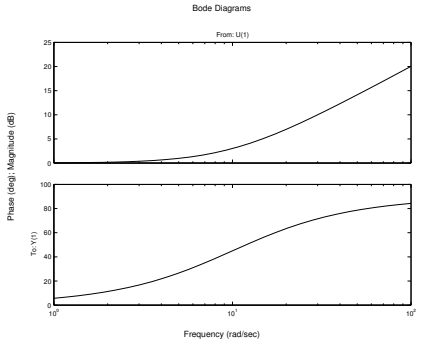


Figura 1: Respuesta frecuencial de una red PD.

En la figura 1 se puede ver la respuesta frecuencial para $\tau = 1$. Se puede observar que para valores superiores a $\frac{1}{\tau}$ la ganancia crece (a unos $20\frac{dB}{dec}$ para

frecuencias altas), así como la fase; para frecuencias bajas, su efecto llega a hacerse despreciable.

En dicho diagrama de Bode pueden observarse dos de los efectos interesantes para una red PD:

- Aumento del ancho de banda, es decir, una disminución del tiempo de subida (el sistema se hace más rápido).
- Aumento del margen de fase.

Respuesta frecuencial de una red PI

Para un controlador PI, la señal de control es proporcional al error y la integral del mismo. La función de transferencia de una red PI se puede escribir como:

$$G(s) = K(1 + \frac{1}{\tau s}) \tag{2}$$

En la figura 2 se puede ver la respuesta frecuencial para $\tau = 1$. Para frecuencias altas el efecto es pequeño, mientras que para frecuencias bajas, aumenta la ganancia del sistema completo y disminuye la fase.

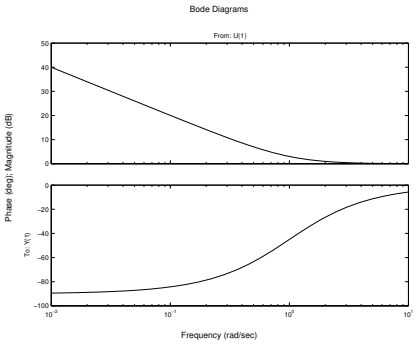


Figura 2: Respuesta frecuencial de una red PI.

Los efectos de una red PI son:

- Cambia el tipo de un sistema (añade un polo en el origen).
- Aumenta la sobreoscilación y disminuye el tiempo de subida.
- Disminuye el error en régimen permanente (al aumentar la ganancia a bajas frecuencias).

Respuesta frecuencial de una red PID

Para un controlador PID, la señal de control es proporcional al error, a la derivada del error y la integral del mismo. La función de transferencia de una red PID se puede escribir como:

$$G(s) = K(1 + \frac{1}{\tau_1 s} + \tau_2 s) \tag{3}$$

En la figura 3 se puede ver la respuesta frecuencial para $\tau_1 = 1$ y $\tau_2 = 0{,}2$. Para frecuencias altas el predomina el efecto derivativo, mientras que para frecuencias bajas, predomina el efecto integral.

El controlador PID permite aprovechar las ventajas de las redes PI y PD.

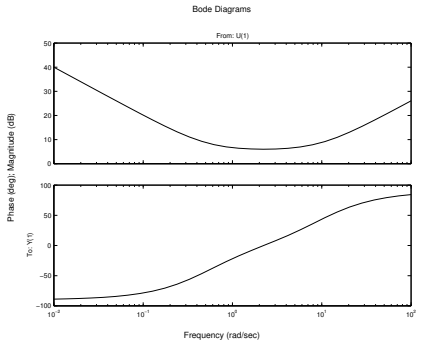


Figura 3: Respuesta frecuencial de una red PID.

Realización de la práctica

Las tareas a realizar en la práctica son:

1. Diseñe controladores de tipo PD, PI y PID tratando de cumplir las especificaciones dadas en el apartado Especificaciones

PRÁCTICA 12 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Manuel López Martínez
José Ángel Acosta Rodríguez

Agradecimientos a Manuel Berenguel Soria

Práctica 12: Redes de Compensación

En esta práctica se realizarán dos redes de compensación: Red de Avance y Red de Retraso.

Ambas para controlar dos sistemas inestables en bucle abierto.

1.1. Red de Avance

Dado el sistema

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

calcular una red de avance de fase de manera que el sistema en bucle cerrado tenga:

- Un error en régimen permanente para seguimiento en aceleración menor o igual que 0.1,
- sobreoscilación menor o igual que el 20 %

1.2. Red de Retraso

Diseñar una red de atraso de fase para que el sistema

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

satisfaga que:

- el coeficiente de error en velocidad sea 5 s^{-1},
- el margen de fase sea menor o igual que 40^o y
- el margen de ganancia sea mayor o igual que 10 dB.

PRÁCTICA 14 DE REGULACIÓN AUTOMÁTICA

Dpto. Ing. Sistemas y Automática
Universidad de Sevilla

Mercedes Pérez de la Parte
Fernando Dorado Navas

Agradecimientos a Manuel Berenguel Soria

Práctica 14

Control por computador de un servomecanismo de posición.

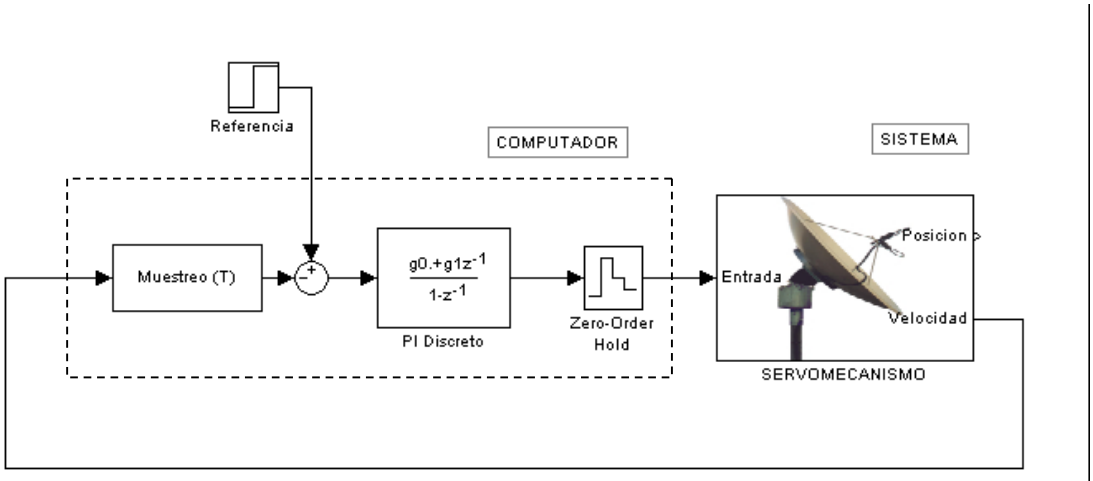
Profesores: Mercedes Pérez de la Parte, Fernando Dorado Navas
Asignatura: Regulación Automática, Curso 2002-2003
Dpto de Ingeniería de Sistemas y Automática

1. Objetivos

Se pretende revisar los conceptos de control por computador, así como la identificación de sistemas en tiempo discreto. Para ello se programará el control PI en velocidad (y en posición) de un servomecanismo utilizando Simulink y Matlab.

2. Elementos del control por computador.

Muestreo de la señal continua, periodo de muestreo: T.
Programación de la ley de control discreta (PI, para el caso indicado).
Mantenedor de orden cero.



3. Modelado del sistema

Se sabe que el servomecanismo se puede modelar por un sistema de segundo orden de la forma:

$$G_p(s) = \frac{K}{1+\tau\, s} \cdot \frac{1}{s}$$

si se pretende controlar la velocidad del mismo, la función de transferencia que describe el sistema se reduce al modelo de primer orden de la expresión anterior:

$$G_v(s) = \frac{K}{1 + \tau\, s}$$

Para la práctica se tomará:

$$K = 0.5$$

$$\tau = 10$$

La función de transferencia en el dominio discreto que describe el proceso real, incluidos el mantenedor de orden cero y el proceso de muestreo es:

$$G_v(z) = \frac{b\, z^{-1}}{1 + a z^{-1}}$$

y los nuevos parámetros son:

$$a = -e^{-T/\tau}$$

$$b = K(1 - e^{-T/\tau})$$

Un criterio para elegir el periodo de muestreo es $T = \tau/10$

4. Programación de la ley de control discreta

Se desea programar un PI, que en tiempo discreto, con la aproximación de Euler hacia delante tiene la forma:

$$G_{PI}(z) = \frac{g_0 + g_1 z^{-1}}{1 - z^{-1}}$$

tal que la constante de tiempo en bucle cerrado sea la mitad de la del sistema en bucle abierto.

Notas

Se simularán los experimentos con Simulink, empleando un bloque “Matlab function” para el cálculo de la ley de control y el mantenimiento de la señal durante un periodo de muestreo. El PI se programará en función ***PI.m***.