*MSB #2014/1*

# Modelización de Sistemas Biológicos: Prácticas de Simulación con Matlab y Simulink Primer Semestre Curso 2014/2015

## Índice

# 1. MATLAB

## Prefacio

Esta sección pretende ser una introducción a MATLAB[1], muy breve y muy básica, dirigida a aquellos alumnos de la asignatura de Modelización de Sistemas Biológicos impartida en la Escuela Técnica Superior de Ingenieros Agrónomos que no estén familiarizados con esta herramienta, para que puedan comenzar a utilizarla en el plazo más breve posible.

MATLAB es una herramienta informática que integra alrededor de un interfaz de usuario (GUI) un entorno de desarrollo, un lenguaje de programación y una serie de bibliotecas, denominadas toolboxes, que facilita a los usuarios el diseño y simulación de cualquier tipo de sistema.

Dentro de la clasificación de los lenguajes de programación, MATLAB (MATrix LABoratory) se puede considerar dentro del grupo de los lenguajes interpretados. Su característica fundamental es que es un lenguaje especializado en la manipulación de matrices de valores numéricos, aunque las versiones más recientes del lenguaje permitan de una manera restringida la definición de literales y de nuevos tipos abstractos de datos, como por ejemplo datos categóricos, tablas, estructuras, listas o series temporales.

El documento se ha dividido en una serie de secciones con la intención de que se vaya profundizando en el manejo de la herramienta según se avanza en la lectura.

Es recomendable que según van apareciendo los mandatos, los lectores los introduzcan en la herramienta para comprobar el resultado de su ejecución.

El arranque de la herramienta MATLAB depende del sistema donde se encuentre instalado. En la mayoría de los casos bastará con poner `matlab` en el intérprete de mandatos del sistema operativo para comenzar su ejecución, o si se trata de un sistema operativo con interfaz gráfico, hacer un doble *click* con el ratón en el icono correspondiente.

Una vez que se está ejecutando el programa, en la configuración por defecto de la herramienta aparecerán tres ventanas: una con el directorio de trabajo, una segunda que es a la que nos vamos a referir durante el resto de los apuntes, donde aparece el símbolo del *prompt*, >>, que

[1] ©The MathWorks, Inc.

indica que el usuario puede comenzar a introducir mandatos, y una tercera donde están definidos las variables definidas por los usuarios en el `Workspace`.

MATLAB dispone de una ayuda *on-line* muy completa que se puede invocar en cualquier momento mediante el mandato `help`. Si se quiere consultar la ayuda de una determinada función[2] basta con escribir `help 'nombre de función'`. Proporcionará una explicación breve y precisa del ítem consultado.

Si se desea conocer la versión de la herramienta que se está ejecutando se utiliza el mandato `version`.

El mandato `what` obtiene una lista de los ficheros `.M`[3], `.MAT`[4] y `.MEX`[5] del directorio actual.

El mandato `who` presenta las variables de la sesión actual y el mandato `whos` da una información más detallada de las variables.

El mandato `cd` imprime el directorio donde nos encontramos actualmente.

El mandato `diary` habilita o deshabilita respectivamente con `on` y `off` la escritura de los mandatos de la sesión de trabajo en un fichero. Si no se especifica un nombre alternativo, el nombre por defecto del fichero es el mismo que el del mandato.

El mandato `demo` permite ejecutar las demostraciones de todos las bibliotecas que haya instaladas.

## 1.1. Sintaxis básica

MATLAB es un lenguaje que distingue entre mayúsculas y minúsculas. Cuando se escribe `a=1`, `A=1+j`, no se trata de la misma variable. Son dos identificadores distintos. Los nombres de los mandatos se deben escribir en letras minúsculas.

Se puede ejecutar un mandato del sistema operativo utilizando el carácter '`!`' y añadiendo a continuación el mandato que se quiera ejecutar.

[2] Emplearemos indistintamente el término funciones o mandatos.

[3] Un fichero `.M` es un fichero de texto con mandatos de MATLAB.

[4] Un fichero `.MAT` es un fichero binario que contiene datos.

[5] Un fichero `.MEX` es un fichero ejecutable generado a partir de un compilador de un lenguaje de alto nivel.

Los comentarios se introducen poniendo el símbolo `%`. Desde ese punto hasta el final de la línea, todo lo que se escriba es considerado parte del comentario.

Como en todo lenguaje interpretado, cuando se trabaja sobre el *prompt* del intérprete de MATLAB, el computador obtiene inmediatamente la respuesta de cada mandato después de pulsar la tecla `return`. Si al final de la línea se ha introducido el carácter '`;`', el resultado de la ejecución no se visualiza, pero si el mandato se ha escrito sin este carácter, sí se visualiza junto con el nombre de las variables involucradas en el resultado. Si se quiere introducir más de un mandato en una misma línea, basta separarlos con '`,`' o con '`;`'. En el primer caso se visualizará el resultado de los mandatos, mientras que en el segundo no. Si se desea visualizar el contenido de una variable sin que se escriba el identificador asociado a la misma, basta con utilizar el mandato `disp`.

Si es necesario escribir un mandato en varias líneas, basta con escribir '...' al final de una línea y continuar en la siguiente.

Las operaciones aritméticas tradicionales, suma, resta, multiplicación y división se representan con los operadores clásicos `+, -, * y /`. Independientemente del tipo de los operandos involucrados en las operaciones aritméticas, enteros, reales o complejos, los cálculos siempre se realizan con precisión doble. También tiene definida la división inversa, representada con el operador '\'. Esta operación es la recíproca de la división:

```
c = 3\1
c =
    0,3333
```

La asignación se representa con el carácter '`=`'. Las operaciones de exponenciación se definen con el operador '`^`'. La transposición de matrices con `'`.

Con la herramienta MATLAB, no es necesario realizar una definición explícita de las variables, ni en cuanto al tipo, ni respecto a su dimensión o tamaño. Cualquier variable puede ser de tipo `integer, real` o `complex`, y el valor asignado definirá su tipo. Los únicos problemas que pueden surgir al manejar variables es la utilización de un identificador reservado del lenguaje, o que la utilización de un identificador previamente definido por el usuario destruya el significado original del mismo.

Por ejemplo, el usuario puede definir las variables `sin` y `cos`, pero entonces ya no podrá disponer de las funciones trigonométricas seno y coseno hasta que no se eliminen las nuevas variables con el mandato `clear`. En la tabla 1 aparece una relación de los mandatos de MATLAB utilizados más frecuentemente en los problemas de modelización y que los usuarios no deberán reutilizar para definir nuevos mandatos o variables.

Hay que hacer una mención especial al tratamiento que da MATLAB a los números complejos. Si se define un vector o una matriz de números complejos, hay que evitar los espacios en blanco entre la parte real e imaginaria de un elemento, porque si no, MATLAB los considera como dos números distintos. Por ejemplo, sean los números complejos $3 + 5j$ y $1 - 8j$. Si se quiere formar un vector a partir de estos 2 números complejos, habría que teclear lo siguiente: `a=[3+5j 1-8j]`. Si se separan en ambos casos la parte real e imaginaria con espacios en blanco, estaríamos definiendo realmente un vector de cuatro elementos.

## 1.2. Mandatos de control de flujo

MATLAB dispone de los mandatos `for/end` y `while/end` para definir bucles en las ejecuciones, y de los mandatos `if/else/elseif/end` y `switch/case/otherwise/end` para establecer ejecuciones alternativas de mandatos.

Nos remitimos a la ayuda *on-line* de la herramienta para definir de una manera más precisa la sintaxis de estos mandatos.

Si se desea calcular el volumen de las esferas cuyos radios varían entre 5 y 1, bastaría con escribir los siguientes mandatos:

```
for r=5:-1:1
   vol = (4/3)*pi*r^3;
   disp([r,vol])
end
```

El índice del bucle se va decrementando en cada iteración en una unidad, desde 5 hasta 1. El cálculo de los resultados no comienza hasta que no se introduce el mandato `end`. En este caso no es necesario introducir el carácter '`;`' detrás de cada `for` o `end`.

El mandato `break` finaliza la ejecución de un bucle `for` o `while`. Cuando se utiliza en bucles anidados, sólo se interrumpe la ejecución del bucle

donde se encuentre el mandato `break`. En el siguiente ejemplo, `break` finaliza la ejecución del bucle más interno cuando se cumple la condición `l>2*k`, pero el bucle más externo continúa hasta que `k=6`:

```
for k=1:6
   for l=1:20
      if l>2*k break; else disp(l); end
   end
end
```

## 1.3. Manipulación de matrices

MATLAB fue un lenguaje concebido en sus orígenes para manejar casi exclusivamente datos de tipo vectorial o matricial. Los vectores y las matrices $n$-dimensionales se pueden definir enumerando cada uno de sus elementos, o bien mediante el constructor ':'. La definición de `x = [0, 0.1, 0.2, 0.3, 0.4, 0.5]` y `x = 0:0.1:0.5` son equivalentes, produciendo un vector fila de dimensión 1x6. El transpuesto se puede obtener bien definiéndolo explícitamente, `y=[0; 0.1; 0.2; 0.3; 0.4; 0.5]` o bien con el operador de transposición a partir del vector `x`, `y = x'`, generándose un vector columna 6x1. Si se desea calcular el transpuesto conjugado de un vector o una matriz de números complejos, basta con poner `y=x'`. El constructor ';' se utiliza para delimitar las filas. Se puede acceder al contenido de un elemento del vector indexando el vector sobre el elemento de interés, por ejemplo `x(3)`, o bien acceder a un rango de valores con el modificador ':', `x(2:4)`. También se pueden utilizar los mandatos `linspace` o `logspace` para generar un vector con una serie de valores dentro de un intervalo, espaciados lineal o logarítmicamente.

Cuando las variables `x` e `y` son del mismo tipo, vector fila o columna, y tienen la misma dimensión, se pueden aplicar los operadores aritméticos.

Si no se recuerda el tamaño de una variable vector o matriz, se pueden utilizar los mandatos `length` y `size` para consultar respectivamente el número de columnas y las dimensiones de la variable.

MATLAB considera las variables de tipo `string` como si fueran vectores fila: `v='helado'` equivale a `v=['h', 'e', 'l', 'a', 'd', 'o']`.

Las matrices bidimensionales se definen de manera similar a los vectores. Una matriz 3x3 se puede definir como `m=[0.1, 0.2, 0.3; 0.4, 0.5, 0.6; 0.7, 0.8, 0.9];`, o bien `m(1,1)=0.1; m(1,2)=0.2;`

`...m(3,3)=0.9`. Se puede acceder al contenido de una determinada fila o columna con el modificador ':'. Por ejemplo, `m(1,:)` y `m(:,3)` son respectivamente la primera fila y la tercera columna de `m`, y se manejan como vectores. El tamaño de un vector o de una matriz se puede aumentar dinámicamente añadiendo un elemento o un vector. Así, para añadir un nuevo elemento al vector `x` basta con poner `x = [x, 0.6]`. Según se van ejecutando los mandatos, la herramienta guarda en memoria todas las variables que se van utilizando hasta que se abandona MATLAB o hasta que se borran las variables. Para borrar las variables, existe el mandato `clear`.

Las operaciones de multiplicación, división y exponenciación sobre los elementos de las matrices están definidas anteponiendo el carácter '.' a los respectivos operadores: '`.*`', '`./`', '`.\`' '`.^`'. El mandato `g=m.^3` es equivalente a

```
for k=1:3
   for l=1:3
      g(k,l)=m(k,l)^3;
   end
end
```

El mismo criterio se aplica a los operadores '`.*`', '`./`' y '`.\`'.

Suponiendo que `m` y `n` son dos matrices del mismo tamaño y dimensión, las variables `m` y `n` se pueden utilizar directamente en expresiones relacionales y lógicas. Por ejemplo:

(a) `m==n` es cierto si `m(i,j)=n(i,j)` para todos los elementos.

(b) `m>=n` es cierto si `m(i,j)>=n(i,j)` para todos los elementos.

(c) `m~=n` es cierto si `m(i,j)~=n(i,j)` para al menos uno de los elementos, siendo `~=` el operador diferencia.

## 1.4. Entrada/Salida

El trasvase de datos hacia o desde MATLAB se puede realizar de varias formas:

**Mediante teclado o ratón:** Se pueden introducir datos a través del teclado con el mandato `input`. Para leer un número, bastaría con teclear `z=input('Teclee el valor del radio:')`. La salida a

pantalla se puede realizar con el mandato `fprintf`. Por ejemplo: `fprintf('El volumen de la esfera%12.5f.\n',vol)`. El formato es muy similar al utilizado en el lenguaje de programación C: se imprime en una línea la cadena de caracteres entre apóstrofes y a continuación el valor de la variable `vol` con el formato especificado con el modificador que sigue al carácter '`%`'. Si aparecen dos mandatos `fprintf` de manera consecutiva y en el primero de ellos no aparece '`\n`', entonces la salida de ambos se imprimirá en una sola línea. Si se desea escribir un valor entero, bastaría con poner un 0 a continuación del punto correspondiente al modificador de formato, `%12.0f`, o bien utilizar el sufijo correspondiente a un valor decimal '`d`'.

**Leer o escribir los datos en un fichero:** La sintaxis y los mandatos son muy parecidos a las correspondientes sentencias de entrada/salida del lenguaje C: `fopen, fclose, fscanf, sscanf, sprintf, fread` o `fwrite`. Para una descripción detallada de la sintaxis de los mismos nos remitimos a la ayuda *on-line* proporcionada por la herramienta.

**Los mandatos save o load:** Se utilizan para almacenar y recuperar las variables en ficheros binarios de datos con extensión `.MAT`. La sintaxis del mandato es `save mx m x`, almacenándose en el fichero `mx.mat` las variables `m` y `x`. Si posteriormente se quisiera recuperar el contenido de la variable `x`, bastaría con escribir `load mx x`. También es posible exportar e importar los datos a ficheros de texto con la opción `-ascii`.

## 1.5. Representación de gráficos

La mayoría de las ecuaciones matemáticas expresan complejas relaciones en una, dos, tres o más dimensiones. En ocasiones, tratar de comprenderlas sin su representación gráfica puede convertirse en una tarea imposible. En MATLAB, la representación gráfica de funciones se puede realizar fácilmente y con muy pocos mandatos. No obstante, hay que advertir que algunos de los mandatos continúan activos después de que se ha dibujado el gráfico y pueden intereferir posteriormente con nuevos gráficos si no se pone algo de cuidado.

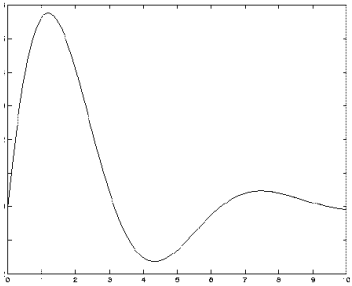


Figura 1: Ejemplo de representación gráfica con MATLAB.

El mandato básico para representar gráficas es `plot`. Supongamos que deseamos representar gráficamente la función $y = \sin(x)e^{-0,4x}, \quad 0 \leq x \leq 10$. Bastaría ejecutar el siguiente código:

```
x=0:0.05:10;
y=sin(x).*exp(-0.4*x);
plot(x,y)
xlabel('x');ylabel('y')
```

Este código produce la figura 1. Las etiquetas de los ejes se imprimen con los mandatos `xlabel` y `ylabel`. Se pueden añadir nuevas opciones al mandato `plot` para cambiar el trazo o el color de la línea que se dibuja. También se pueden representar funciones gráficamente con el mandato `fplot`. El mismo ejemplo que el visto con el mandato `plot` quedaría así:

```
x=0:0.05:10;
fplot('sin(x).*exp(-0.4*x)',[0,10])
xlabel('x');ylabel('y')
```

En este caso el primer argumento es la función que se desea representar, y el segundo el rango de valores para los que se quiere representar la función. La función a representar debe ir entre comillas. El mandato `clf` borra todo el contenido de la figura, mientras que el mandato `cla` borra las curvas representadas y redibuja los ejes. Se puede añadir una

rejilla con el mandato `grid`. Si la magnitud representada en alguno de los ejes de coordenadas necesitara de una escala logarítmica, sería posible su representación con el mandato `loglog`, `semilogx` o `semilogy` en lugar del `plot`. También es posible hacer una representación de la función en coordenadas polares con el mandato `polar`. Para representar más de una curva en un sólo gráfico, basta con poner sucesivamente todas las funciones que se quieren representar. Por ejemplo, para representar el seno y el coseno en una misma gráfica:

```
x = 0:0.05:5;
y = sin(x);
z = cos(x);
plot(x,y,x,z)
```

Hay que señalar que las unidades de las funciones trigonométricas vienen expresadas en radianes, pero los gráficos manejan grados.

El mandato `hold on` permite el trazado de nuevas curvas manteniendo el contenido del gráfico. Este mandato tiene un efecto global, y todo nuevo gráfico que se cree se dibujará sobre la figura ya existente. Para desactivar esta característica y posibilitar la creación de nuevas figuras, se utiliza el mandato `hold off`. El siguiente ejemplo utiliza este mandato para añadir una nueva curva al gráfico. El resultado aparece en la figura 2.

```
x = 0:0.05:5;
y = sin(x);
plot(x,y);
hold on
z = cos(x);
plot(x,z,'--')
xlabel('x');ylabel('y(-) y z(--)');
```

Otros mandatos que permiten información adicional en el gráfico son el mandato `title`, que añade una etiqueta en la parte superior del gráfico, el mandato `text`, que escribe un rótulo en una determinada posición del gráfico, y el mandato `gtext`, que permite la misma acción que el `text`, pero situándolo manualmente con la ayuda del ratón.

Si fuera necesario utilizar más de un gráfico, el mandato `figure` abre una ventana por cada aparición del mismo. Si se quiere volver a dibujar

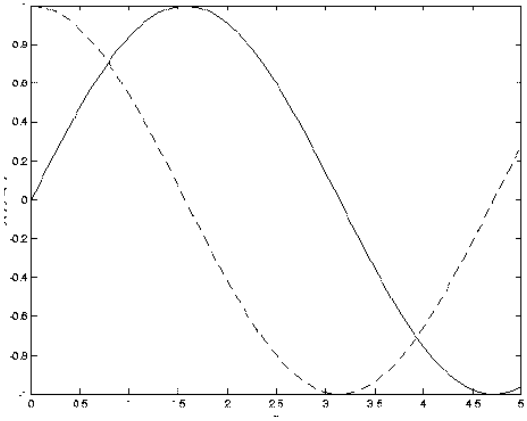


Figura 2: Ejemplo de representación de dos curvas en una sola figura.

en alguno de los gráficos definidos previamente, por ejemplo en el $n$, basta con activarlo con escribir `figure(n)`. Con el mandato `close all` se eliminan todos los gráficos.

El mandato `subplot(m,n,k)` crea una matriz de $m$ por $n$ curvas en un sólo gráfico y dibuja en el situado en la posición $k$-ésima. Para calcular la posición, se empieza a contar por filas y después por columnas desde la esquina superior izquierda de la figura. El siguiente ejemplo produce el gráfico de la figura 3.

```
clear;clf
t=0:.3:30;
subplot(2,2,1), plot(t,sin(t)),title('SUBPLOT 2,2,1')
              xlabel('t'); ylabel('sin(t)')
subplot(2,2,2), plot(t,t.*sin(t)),title('SUBPLOT 2,2,2')
              xlabel('t'); ylabel('t.*sin(t)')
subplot(2,2,3), plot(t,t.*sin(t).^2),title('SUBPLOT 2,2,3')
              xlabel('t'); ylabel('t.*sin(t).^2')
subplot(2,2,4), plot(t,t.^2.*sin(t).^2),title('SUBPLOT 2,2,4')
              xlabel('t'); ylabel('t.^2.*sin(t).^2')
```

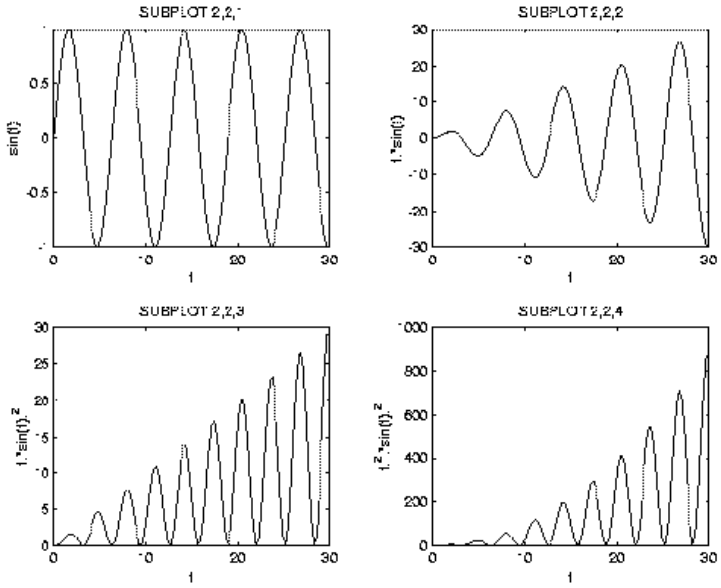



Figura 3: Ejemplo de la utilización del mandato `subplot`.

Finalmente, mencionar que para dibujar gráficos con 3 ejes de coordenadas existe el mandato `plot3`.

## 1.6. Definición de nuevos mandatos

La ejecución de los mandatos en una ventana es adecuado solamente si la cantidad de texto a teclear es pequeña o si se quiere hacer alguna prueba de manera interactiva. Cuando la extensión de los mandatos involucrados abarca unas cuantas líneas, es más recomendable escribir un fichero `.M`, también denominado *script*.

Cuando se ejecuta un script, los mandatos del fichero no se visualizan en

pantalla. Si se desea activar la visualización de los mandatos ejecutados, basta con incluir el mandato `echo on`. Para desactivarlo nuevamente habría que incluir `echo off`.

El formato de estos ficheros puede ser simplemente un conjunto de mandatos que se ejecutan según una determinada secuencia, o bien seguir un planteamiento idéntico a la definición de funciones y subprogramas de los lenguajes de programación clásicos. En este caso, habría que introducir una cabecera donde se utiliza la palabra reservada `function` junto con el identificador correspondiente a la nueva función que se está definiendo y los argumentos de entrada y de salida de la función. Veamos un ejemplo para ilustrar esta idea.

Se va a definir una función que devuelve un solo valor:

$$f(x) = \frac{2x^3 + 7x^2 + 3x - 1}{x^2 - 3x + 5e^{-x}}$$

Si el fichero lo denominamos `ejemf.m`, el contenido del mismo sería:

```
function y = ejemf(x)
y = (2*x.^3+7*x.^2+3*x-1)./(x.^2-3*x+5*exp(-x));
```

Hay que advertir que el nombre del fichero `.M` es el mismo que el identificador de la función que aparece en el lado derecho de la cabecera. El operador de potenciación utilizado es el `.^`, por lo que se puede pasar indistintamente como argumento un escalar o un vector. Una vez que se ha creado el fichero `ejemf.m`, ya se puede utilizar en la definición de nuevos mandatos. El mandato `ejemf(3)` produce como resultado

```
y =
    502.1384
```

Si el argumento fuera una matriz, por ejemplo `ejemf([3,1;0,-1])` el resultado sería también una matriz gracias a la utilización del operador `.^`:

```
ans =
   502.1384     -68.4920
    -0.2000       0.0568
```

Para definir más de un argumento de salida hay que definir un vector en el lado izquierdo de la cabecera de la función. Por ejemplo:

```
function [media,devstd] = media_devstd(x)
n=length(x);
media = sum(x)/n;
devstd = sqrt(sum(x.^2)/n - media.^2);
```

Si estos mandatos se han escrito en el fichero `media_devstd.m`, al invocar el mandato `media_devstd` con el argumento

```
x = [1 5 3 4 6 5 8 9 2 4];
[m, s] = media_devstd(x)
```

genera

```
m =
    4.70000
s =
    2.3685
```

También se puede utilizar como argumento otro mandato ya existente gracias al mandato `feval`. Por ejemplo, sea la función

$$f_{av} = \frac{f(a) + 2f(b) + f(c)}{4}$$

donde $f(x)$ es la función que se pasa como argumento. El script `f_av.m` define la función anterior:

```
function wa = f_av(f_name, a, b, c)
wa = (feval(f_name,a) + 2*feval(f_name,b) ...
                      + feval(f_name,c))/4;
```

`f_name` es una variable de tipo string que representa el nombre de la función $f(x)$. Si $f(x)$ es la función `ejemf`, definida anteriormente, `y=feval('ejemf',x)` equivale a `y=ejemf(x)`. La llamada a la función `z=f_av('ejemf', 1, 2, 3)` daría como resultado `89.8976`.

MATLAB incorpora un editor de mandatos, `edit`, que realiza también tareas de depuración mediante la definición de puntos de control en la secuencia de ejecución del mandato que se esté editando.

Tabla 1: Relación de mandatos más habituales en las aplicaciones de modelización de sistemas con MATLAB.

| Nombre del mandato | Descripción |
|---|---|
| `abs` | Valor absoluto, magnitud compleja |
| `angle` | Fase o ángulo en radianes |
| `ans` | Respuesta cuando no se asigna expresión |
| `atan` | Arco tangente |
| `axis` | Controla la apariencia y la escala de los ejes de coordenadas en un gráfico |
| `bode` | Representación con el diagrama de Bode |
| `clear` | Libera memoria eliminando variables y mandatos |
| `cla` | Borra el contenido de una figura y redibuja los ejes |
| `clf` | Borra el contenido de una figura |
| `close` | Cierra ventanas donde se han representado gráficas |
| `conj` | Complejo conjugado |
| `conv` | Convolución |
| `corrcoef` | Coeficientes de correlación |
| `cos` | Coseno |
| `cosh` | Coseno hiperbólico |
| `cov` | Matriz de covarianza |
| `deconv` | Deconvolución |
| `det` | Determinante |
| `diag` | Matriz diagonal |
| `echo` | Visualización de los mandatos de un script |
| `edit` | Invoca el editor y el depurador de mandatos de MATLAB |
| `eig` | Valores y vectores propios |
| `exit` | Salir de MATLAB |
| `exp` | Función exponencial |
| `expm` | Matriz exponencial |
| `eye` | Matriz identidad |

Tabla 1: Relación de mandatos más habituales en las aplicaciones de modelización de sistemas con MATLAB (cont.).

| Nombre del mandato | Descripción |
|---|---|
| `feval` | Ejecuta el mandato representado por la cadena de caracteres que se pasa como argumento |
| `figure` | Genera una ventana para dibujar un gráfico |
| `filter` | Operación de filtrado |
| `format` | Formato de salida de los cálculos |
| `fplot` | Representación gráfica de funciones |
| `grid` | Líneas de rejilla |
| `gtext` | Colocación de una etiqueta en un gráfico con ayuda del ratón |
| `hold` | Conserva el contenido del gráfico actual |
| `i` | $\sqrt{-1}$ |
| `imag` | Parte imaginaria de un número complejo |
| `inf` | Infinito ($\infty$) |
| `inv` | Matriz inversa |
| `j` | $\sqrt{-1}$ |
| `length` | Longitud del vector |
| `linspace` | Generación de un vector espaciado linealmente |
| `log` | Logaritmo natural |
| `loglog` | Gráfica con ambos ejes representados en escala logarítmica |
| `logm` | Logaritmo matricial |
| `logspace` | Generación de un vector espaciado logarítmicamente |
| `log10` | Logaritmo en base 10 |
| `max` | Valor máximo |
| `mean` | Valor medio |
| `median` | Mediana |
| `min` | Valor mínimo |

Tabla 1: Relación de mandatos más habituales en las aplicaciones de modelización de sistemas con MATLAB (cont.).

| Nombre del mandato | Descripción |
|---|---|
| `NaN` | No es un valor numérico |
| `nyquist` | Respuesta en frecuencia en el diagrama de Nyquist |
| `obsv` | Matriz de observabilidad |
| `obsvf` | Descomposición de un sistema en espacios observable y no observable |
| `ones` | Genera un vector o una matriz constante con unos |
| `pi` | Número $\pi$ |
| `plot` | Dibuja un gráfico con los ejes en escala lineal |
| `plot3` | Dibuja un gráfico con 3 ejes de coordenadas en escala lineal |
| `polar` | Ídem en coordenadas polares |
| `poly` | Polinomio característico |
| `polyfit` | Ajuste polinomial |
| `polyval` | Evalúa el valor de un polinomio en un punto |
| `polyvalm` | Ídem respecto de una matriz |
| `prod` | Producto de los elementos de un vector |
| `pzmap` | Calcula y representa gráficamente los polos y los ceros de un sistema lineal |
| `quit` | Abandonar el programa |
| `rand` | Generación de números aleatorios |
| `rank` | Calcula el rango de una matriz |
| `real` | Parte real de un número complejo |
| `rem` | Resto o módulo |
| `residue` | Cálculo de los residuos de una división entre polinomios |
| `roots` | Calcula las raíces de un polinomio |

Tabla 1: Relación de mandatos más habituales en las aplicaciones de modelización de sistemas con MATLAB (cont.).

| Nombre del mandato | Descripción |
|---|---|
| `semilogx` | Gráfica con el eje x representado en escala logarítmica |
| `semilogy` | Gráfica con el eje y representado en escala logarítmica |
| `sign` | Función signo |
| `sin` | Función seno |
| `sinh` | Seno hiperbólico |
| `size` | Dimensión de una matriz |
| `sqrt` | Raíz cuadrada |
| `sqrtm` | Raíz cuadrada matricial |
| `std` | Desviación estándar |
| `step` | Función escalón |
| `subplot` | Representación de varias gráficas dentro de una misma ventana |
| `sum` | Suma de los elementos de un vector |
| `tan` | Tangente |
| `tanh` | Tangente hiperbólica |
| `text` | Etiqueta textual en un gráfico |
| `title` | Título de un gráfico |
| `trace` | Traza de una matriz |
| `who` | Visualiza las variables definidas |
| `xlabel` | Etiqueta del eje x |
| `ylabel` | Etiqueta del eje y |
| `zeros` | Genera un vector o una matriz nula |

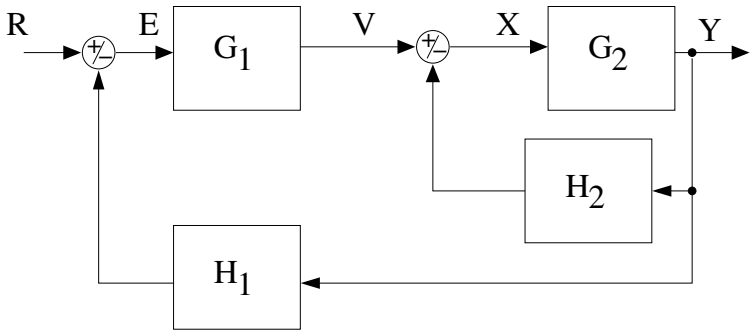



Figura 4: Ejemplo de sistema en lazo cerrado.

## 2. Simulación utilizando MATLAB

`MATLAB` proporciona un conjunto de funciones que permiten realizar cálculos numéricos sobre vectores y matrices. Con el *Control System Toolbox* se pueden manejar funciones específicas que facilitarán la labor de diseño y análisis de los modelos.

Para la asociación de módulos con `MATLAB`, se dispone de las siguientes funciones:

- `series`
- `parallel`
- `feedback`
- `append`
- `connect`

Por ejemplo, sea el sistema de la Figura 4, donde $G_1(s) = 0'4$, $G_2(s) = \frac{100}{s(s+2)}$, $H_2(s) = \frac{s}{s+20}$, y $H_1(s) = 1$. La función de transferencia en lazo cerrado se puede obtener en MATLAB con los mandatos del siguiente ejemplo:

```
% ej1.m
ng1=0.4;dg1=1;
```

```
ng2=[0 0 100];dg2=[1 2 0];
nh2=[1 0]; dh2=[1 20];
% Determinar la función de transferencia de V a Y
[ngvy,dgvy]=feedback(ng2,dg2,nh2,dh2,-1)
% Determinar la función de transferencia de E a Y
[ngey,dgey]=series(ng1,dg1,ngvy,dgvy)
% Determinar la función de transferencia en lazo cerrado de R a Y
[n,d]=feedback(ngey,dgey,1,1,-1)
```

Resultando: $\frac{40s + 800}{s^3 + 22s^2 + 180s + 800}$.

En MATLAB existen las siguientes funciones que permiten manejar funciones de transferencia asociadas a cada uno de los bloques del modelo de planta:

- `tf`: Crea una variable correspondiente al tipo de datos asociado a una función de transferencia descrita mediante el numerador y del denominador de un cociente de polinomios.
- `zpk`: Crea una variable correspondiente al tipo de datos asociado a una función de transferencia descrita mediante la descomposición en fracciones simples con ceros y polos.
- `ss`: Crea una variable correspondiente al tipo de datos asociado a una función de transferencia descrita mediante un modelo en el espacio de estados.
- `tfdata`: Obtiene el numerador y el denominador de una función de transferencia.
- `zpkdata`: Obtiene la representación basada en ceros y polos de una función de transferencia.
- `ssdata`: Obtiene las matrices del modelo de espacio de estados de una función de transferencia.

La notación de las funciones `tfdata` y de `zpkdata` para un sistema lineal de una entrada y una salida es `función(sys,'v');`. El mismo ejemplo presentado en la Figura 4, pero utilizando en este caso las funciones anteriores, quedaría descrito de la siguiente forma:

```
%ej2.m
ng1=0.4;dg1=1;
ng2=[0 0 100];dg2=[1 2 0];
nh2=[1 0]; dh2=[1 20];
nh1=1;dh1=1;
% Determinar las funciones de transferencia de cada bloque
sysg2=tf(ng2,dg2)
sysh2=tf(nh2,dh2)
sysg1=tf(ng1,dg1)
sysh1=tf(nh1,dh1)
% Determinar la función de transferencia de V a Y
sysvy=feedback(sysg2,sysh2,-1)
% Determinar la función de transferencia de E a Y
sysey=series(sysg1,sysvy)
% Determinar la función de transferencia en lazo cerrado de R a Y
sysry=feedback(sysey,sysh1,-1)
[num,den]=tfdata(sysry,'v')
[z,p,k]=zpkdata(sysry,'v')
[a,b,c,d]=ssdata(sysry)
```

Hay que recordar que existen las siguientes funciones para realizar conversiones entre las distintas técnicas de representación:

- `tf2zp`
- `zp2tf`
- `ss2tf`
- `tf2ss`

Para manejar polinomios, disponemos de las siguientes funciones:

- `roots`: Obtiene las raíces de un polinomio.
- `poly`: Dadas las raíces de un polinomio, obtiene éste.
- `residue`: `[R,P,K]=residue(B,A)`$\Leftrightarrow \frac{B(s)}{A(s)} = \frac{R[1]}{s-P[1]} + \cdots + \frac{R[n]}{s-P[n]} + K(s)$
- `conv`: Convolución de polinomios.

- `deconv`: División de polinomios.
- `polyval`: Obtiene el valor de un polinomio en un punto.
- `polyfit`: Encuentra el polinomio de grado $n$ que mejor se ajusta por mínimos cuadrados a los datos de entrada. `polifit(X,Y,n)`$\equiv$ `P(X(i))`$\simeq$ `Y(i)`
- `eig`: calcula los autovalores del polinomio.

Veamos un ejemplo de utilización de estas funciones:

```
% polinomio.m
function polinomio(x1,x2)
y1=roots(x1)
poly(y1)
[r,p,k]=residue(x1,x2)
[numerador,denominador]=residue(r,p,k)
polimul=conv(x1,x2)
[cociente,resto]=deconv(polimul,x1)
polyval(x1,3)
x3=[0:0.1:20];
y2=sin(x3);
plot(x3,y2,'k');
hold on;
tamx3=length(x3);
colores=['y' 'y' 'y' 'g' 'g' 'g' 'b' 'b' 'b' 'r' 'r' 'r'];
for k=0:10,
   polajust=polyfit(x3,y2,k);
   res=polyval(polajust,x3);
   if mod(k,5)==0
      plot(x3,res,colores(k+1));
   end;
   hold on
end
```

Se dispone de las siguientes funciones para analizar la respuesta de un sistema lineal:

- `step`: Obtiene la respuesta de un sistema ante una entrada escalón.

- `impulse`: Obtiene la respuesta del sistema ante una función impulso. En ambos casos, `MATLAB` convierte internamente los modelos de las plantas a una representación mediante espacio de estados y la simulación numérica se realiza utilizando la técnica de exponencial matricial. P. ej.: `step(sysvy); figure; impulse(sysvy); close all`
  Además, disponemos de las siguientes funciones:
- `initial`: Determina las condiciones iniciales del sistema en una representación basada en el espacio de estados.
- `lsim`: Permite obtener la respuesta de sistemas lineales, pudiéndose introducir entradas definidas por el usuario, entradas múltiples o estados iniciales.
- `rlocus: Calcula y dibuja el lugar de las raíces de un sistema con una entrda y una salida.`
- `rlocfind`: Permite obtener interactivamente los polos del lugar de las raíces de un sistema.
- `bode`: Dibuja el diagrama de Bode de un sistema.
- `nyquist` Dibuja el diagrama de Nyquist de un sistema.
- `nichols` Dibuja el diagrama de Nichols de un sistema.

A continuación se presentan diversos ejemplos de utilización de estas funciones.

```
% rlocusej
n=[0 0 0 1 2]; %Numerador de P(s)
d=[1 3 4 2 0]; %Denominador de P(s)
rlocus(n,d);
[k,poles]=rlocfind(n,d);
```

```
% ejbode.m
close all;
n=[0 0 0 0 6.21e9];
d=[1.62e4 0 9.37e7 0 6.21e9];
w=[0:0.2:100];
```

```
[m,p,w]=bode(n,d,w);
f=w/(2*pi);dbg=20*log10(m);
plot(f,dbg);
```

En el caso de los sistemas no lineales, se puede simular el comportamiento del sistema utilizando las funciones `ode23`, correspondiente al algoritmo de Runge-Kutta de segundo y tercer orden, y la función `ode45`, que implementa el algoritmo de Runge-Kutta-Fehlberg.

Sea el sistema no lineal definido por el sistema de ecuaciones:

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= \tan^{-1}(u^2 - x_1^2) - 2x_2 \end{aligned} \tag{1}$$

donde $u = 2$, $x_1(0) = -1$ y $x_2(0) = 0$, entonces:

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= \tan^{-1}(4 - x_1^2) - 2x_2 \end{aligned} \tag{2}$$

Hay que definir una función de usuario que implemente este sistema:

```
% nlsys.m
function xdot=nlsys(t,x)
xauxdot(1)=x(2);
xauxdot(2)=atan(4-x(1)^2)-2*x(2);
xdot=xauxdot';
```

El programa que permite realizar la simulación es el siguiente:

```
% simnolineal.m
t0=0; tf=10; %Definición del instante inicial y final
x0=[-1;0];
[t,x]=ode23('nlsys',[t0 tf],x0);
plot(t,x);
xlabel('Tiempo (s)'),ylabel('Amplitud'),grid
```

## 3. Simulación utilizando SIMULINK

SIMULINK es un paquete de software para modelar, simular y analizar sistemas dinámicos gráficamente. Soporta sistemas lineales y no lineales,

modelos definidos con la variable temporal en su forma continua, discreta o versiones híbridas. También puede simular sistemas multifrecuencia. La definición de los modelos se puede realizar utilizando técnicas jerárquicas de abstracción, permitiendo crear nuevos modelos más complejos a partir de subsistemas más sencillos.

## 3.1. Introducción al manejo de la herramienta diseñando un sistema

**3.1.1. Estructura de SIMULINK basada en bibliotecas:** `SIMULINK` permite la utilización de funciones y bloques de los *toolboxes* disponibles en `MATLAB`.

Se va a ilustrar con el ejemplo de la Figura 4 las operaciones básicas que se pueden realizar con `SIMULINK`.

**3.1.2. Definición de entradas y salidas:** Las entradas a los modelos se definen a partir del grupo `Sources`. Las salidas, a partir del grupo `Sinks`.

**3.1.3. Simulación de un modelo:** A modo de introducción, se va a realizar una simulación a partir de la planta definida en el fichero `modej1.mdl`. Va a ser necesario definir las siguientes variables:

```
% entrada2.m
% [u,t]=entrada2(f)
% f: parámetro que determina la amplitud de la entrada
% u: vector con las muestras de la entrada
% t: vector de tiempos
%
function [u,t]=entrada2(f)
clear t u;
t(1:301)=0:0.1:30;
for i=1:20,
   u(i)=f*0.4*(i-1)/(20-1+1);
end;
u(21:40)=0.4*f;
u(41:60)=0.55*f;
for i =61:70,
   u(i)=f*(0.55+(0.8-0.55)*(i-61)/(70-61+1));
```

```
end;
u(71:301)=0.8*f;
t=t';
u=u';
```

**3.1.4. Edición:** Sobre el modelo de la demo, practicar las siguientes operaciones:

- Selección de objetos: de uno en uno y por grupos.
- Copiado y movimiento de objetos.
- Borrado.
- Parametrización.
- Orientación de los bloques.
- Redimensionado de bloques.
- Adición de líneas entre bloques.
- Adición de líneas a partir de otra línea: `Ctrl`+botón izqdo. del ratón.

**3.1.5. Creación de subsistemas:** Bloques dentro del grupo `Ports & Systems`.

**3.1.6. Impresión de modelos:** El mandato `print` permite imprimir los diagramas de los modelos y los gráficos tanto en un fichero como directamente en la impresora.

### 3.2. Simulación y análisis

Los parámetros de la simulación incluyen:

- Instantes inicial y final de la simulación.
- Tamaño del paso mínimo: en cada paso de la integración.
- Tamaño del paso máximo: en cada paso de la integración.

- Tolerancia o error relativo: en cada paso de la integración.
- Variables de retorno.

SIMULINK dispone de diferentes métodos de simulación numérica. Los métodos disponibles son tanto de paso variable como de paso fijo. En los del paso variable, el tamaño de paso se ajusta continuamente para cumplir el criterio de error relativo máximo permitido. El tamaño de paso indica la mínima distancia entre puntos generada por las trayectorias de salida.

Todos los algoritmos de simulación, excepto `euler`, reducen el tamaño del paso cuando el error predicho que se calcula es mayor que el error relativo. El tamaño del paso nunca se reduce por debajo del valor definido como paso mínimo. Por lo tanto, es posible obtener resultados poco precisos si el error relativo o el tamaño de paso mínimo es demasiado grande.

## 3.3. Interfaz entre SIMULINK y MATLAB

Existen las siguientes posibilidades para intercambiar información entre SIMULINK y MATLAB:

- Bloque `From/To Workspace`
- `linmod`: `[a,b,c,d]=linmod('modej1')` Extrae el modelo lineal en el espacio de estados de un sistema alrededor de un punto de operación de un sistema descrito con un modelo creado con SIMULINK.
- `dlinmod`: esta función puede linealizar sistemas discretos, multifrecuencia e híbridos continuos y discretos en cualquier momento de muestreo.
- `linmod2`: proporciona una forma avanzada de linealización. Es más costosa computacionalmente que `linmod` pero puede producir resultados más precisos.
- `trim`: encuentra los parámetros en estado estacionario que satisfacen condiciones de entrada, salida y estado. Es posible que no exista una solución factible a los valores que se introducen como argumento, en cuyo caso `trim` minimiza el caso más desfavorable de la desviación de cero de las dervidadas del vector de estado.

- `sim`: simulación de un modelo desde `MATLAB`.
- `simset`: definición de los parámetros para realizar una simulación con la función `sim`. Desde `MATLAB` se pueden realizar simulaciones de modelos definidos con `SIMULINK` utilizando la función `sim`.

## 4. Soluciones para la Biotecnología

`MATLAB` dispone de un toolbox específico para facilitar el procesamiento y análisis de datos en el dominio de aplicación de la Bioinformática. Se trata del toolbox `Bioinformatics`
http://www.mathworks.es/products/bioinfo/.

Hay disponible un vídeo de introducción a este toolbox en

http://www.mathworks.es/videos/bioinformatics-toolbox-overview-61196.html

También hay disponible un vídeo de introducción al análisis de micro-arrays

http://www.mathworks.es/videos/matlab-for-microarray-analysis-81589.html.

En este caso, la información útil se puede encontrar a partir del minuto 9.

También existe un módulo específico de `Simulink` para la simulación de sistemas biológicos

http://www.mathworks.es/products/simbiology/.

Hay un vídeo ilustrando la utilización de este módulo en

http://www.mathworks.es/videos/teaching-pkpd-and-mechanistic-modeling-with-matlab-and-simbiology-89577.html?s_iid=main_rw_SB_cta1.

## 5. Enunciado

Diseñar un sistema de regulación de temperatura de un invernadero.

Como parámetros de entrada, el sistema recibe información sobre la temperatura interior deseada como referencia, la temperatura en el exterior del invernadero y la fluctuación esta magnitud a lo largo del día.

Como salida, se proporcionarán las gráficas del coste que supone mantener encendido el sistema y la variación de las temperaturas exterior e interior del invernadero. Las unidades se expresarán en grados Celsius.

En el modelado del sistema se deberá incorporar la siguiente información:

- Geometría acristalada del invernadero: ancho, largo, alto, y superficie de la cubierta.
- Propiedades térmicas de la superficie acristalada.
- Características del sistema de calefacción: temperatura y cantidad del flujo de aire generado.
- Características del sistema de refrigeración: temperatura y cantidad del flujo de aire generado.
- Hay que señalar que cada aparato dispondrá de su propio termostato y que solo uno de los dos sitemas funcionará en un momento determinado, bien para refrigerar el invernadero o bien para calentarlo. También puede suceder que los dos aparatos estén apagados porque la temperatura del invernadero no requiera su funcionamiento.
- Cada termostato dispondrá de su propio rango de funconamiento. En el caso de la calefacción, el termostato saltará cuando se detecte una temperatura inferior a 17 grados y se desactivará cuando la temperatura del invernadero alcance 23 grados. En el caso del refrigerador, saltará cuando se detecte una temperatura superior a 35 grados y se desconectará con una temperatura inferior a 29 grados.
- El coste del consumo total del sistema (Kw/hora). El coste de la refrigeración será distinto del de la calefacción.

- La temperatura exterior al invernadero será la suma de dos patrones, uno semanal y otro diario. En el semanal, la entrada al sistema se modelizará como una senoide centrada en 15 grados y una amplitud de 10 grados, mientras que el diario se modelizará como una senoide que centrada en 0 y una amplitud de 15 grados.

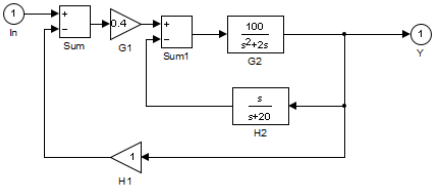
Se deberá entregar por correo electrónico al profesor encargado de la práctica el modelo del invernadero (fichero .slx), los ficheros auxiliares de inicialización de variables (ficheros .m), la documentación detallada del mismo (fichero .pdf) y las gráficas correspondientes al resultado de alguna simulación, recogiendo en una gráfica el coste del funcionamiento del sistema de calefacción y refrigeración del invernadero y en otra la temperatura tanto del invernadero como de su exterior.

La fecha límite para la entrega será el día 1 de diciembre (8:00 am).

# A. Ejemplo de documentación en Simulink



## modej1




**arodri**

22-Sep-2014 10:35:48

---

**Table of Contents**



**List of Tables**



## Model - modej1

**Full Model Hierarchy**

1. modej1

| Simulation Parameter | Value |
|---|---|
| Solver | ode45 |
| RelTol | 1e-3 |
| AbsTol | 1e-6 |
| Refine | 1 |
| MaxOrder | 5 |
| ZeroCross | on |

*[more info]*

# System - modej1

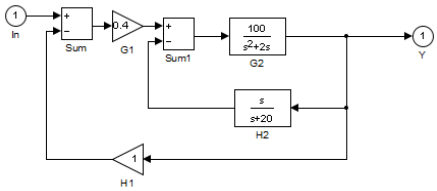



**Table 1. Gain Block Properties**

| Name | Gain | Multiplication | Param Data Type Str | Out Data Type Str | Lock Scale | Rnd Meth | Saturate On Integer Overflow |
|---|---|---|---|---|---|---|---|
| G1 | 0.4 | Element-wise(K.*u) | Inherit: Same as input | Inherit: Same as input | off | Floor | on |
| H1 | 1 | Element-wise(K.*u) | Inherit: Same as input | Inherit: Same as input | off | Floor | on |

**Table 2. Inport Block Properties**

| Name | Port | Defined In Blk |
|---|---|---|
| In | 1 | modej1 (model) |

**Table 3. Outport Block Properties**

| Name | Port | Icon Display | Lock Scale | Var Size Sig | Signal Type | Sampling Mode | Source Of Initial Output Value | Output When Disabled | Initial Output | Used By Blk |
|---|---|---|---|---|---|---|---|---|---|---|
| Y | 1 | Port number | off | Inherit | auto | auto | Dialog | held | 0 | H1, H2, modej1 (model) |

**Table 4. Sum Block Properties**

| Name | Icon Shape | Inputs | Collapse Mode | Collapse Dim | Input Same DT | Accum Data Type Str | Out Data Type Str | Lock Scale | Rnd Meth | Saturate On Integer Overflow |
|---|---|---|---|---|---|---|---|---|---|---|
| Sum | rectangular | +- | All dimensions | 1 | on | Inherit: Inherit via internal rule | Inherit: Same as first input | off | Floor | on |
| Sum1 | rectangular | +- | All dimensions | 1 | on | Inherit: Inherit via internal rule | Inherit: Same as first input | off | Floor | on |

**Table 5. TransferFcn Block Properties**

| Name | Numerator | Denominator | Absolute Tolerance | Continuous State Attributes |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| G2 | [100] | [1 2 0] | auto | " |
| H2 | [1 0] | [1 20] | auto | " |

# Appendix

**Table 6. Block Type Count**

| BlockType | Count | Block Names |
|---|---|---|
| TransferFcn | 2 | G2, H2 |
| Sum | 2 | Sum, Sum1 |
| Gain | 2 | G1, H1 |
| Outport | 1 | Y |
| Inport | 1 | In |

## Registro de Revisiones

Este documento ha sido publicado como informe docente número 2014/1 por el *Departamento de Arquitectura y Tecnología de Sistemas Informáticos* de la *Escuela Técnica Superior de Ingenieros Informáticos* de la *Universidad Politécnica de Madrid* como parte de la asignatura *Sistemas Operativos*. El histórico de revisiones realizadas a este documento son:

- Revisión 1.0 (11/09/2014).
- Versión inicial (24/06/2014).



## Referencias

[1] S. Nakamura: *Numerical Analysis and Graphic Visualization with MATLAB*, Prentice Hall, 1996.

[2] B. C. Kuo y D. C. Hanselman: *MATLAB Tools for Control Systems Analysis and Design*, Prentice Hall, 1994

[3] The MathWorks Inc.: *La edición de estudiante de SIMULINK. Software de simulación de sistemas dinámicos.* Prentice Hall, 1996.

[4] Libros de MATLAB y Simulink en el campo Bio: http://www.mathworks.es/support/books/index_by_categorytitle.html?category=15&sortby=title