SISTEMAS ELECTRÓNICOS DE CONTROL
curso 2000-2001

PRÁCTICA 1 : Introducción a Matlab y a Simulink

Objetivos

- Conocer las características básicas de los paquetes informáticos Matlab y Simulink.
- Aprender los fundamentos de utilización de Matlab. Representar gráficas y construir funciones.
- Utilizar Simulink como herramienta de simulación sistemas dinámicos. Crear sistemas sencillos y simular su comportamiento.

1. Introducción a Matlab.

Matlab es un entorno y un lenguaje de programación altamente potente para la realización de cálculos técnicos y científicos.

1.1. Características básicas de Matlab.

- Funcionalidades básicas:
  - cálculo matricial
  - representaciones gráficas

- Librerías específicas: *toolboxes*. Un *toolbox* es una colección especializada de archivos de Matlab para trabajar en clases particulares de problemas.
  - **Simulink**: Es una extensión a Matlab que añade un entorno gráfico para modelar, simular y analizar sistemas dinámicos.
  - **Control System**: Contiene funciones para modelado, análisis y diseño de sistemas de control automático.
  - **Matemática simbólica**: opera con variables simbólicas.

1.2. Cómo encontrar ayuda en Matlab.

Existen distintas formas de localizar ayuda en el entorno de Matlab:

- **Ayuda en línea**

Se accede a través de la ventana de comandos tecleando *help nombrefunción*. La ayuda se obtiene en modo texto.

- **Ventana de ayuda (Help → Help Window)**

Constituye una manera más sencilla de localizar la misma información: las funciones están agrupadas en bloques y se proporciona un pequeño interfaz para navegar.

- **Ayuda HTML (Help → Help Desk)**

Con una estructuración similar a la de la ventana de ayuda, es fácil acceder a la información y, además, ésta se presenta en modo gráfico y con ejemplos de utilización.

- **Ejemplos (Help → Examples and Demos)**

Matlab proporciona ejemplos y demostraciones de sus principales funcionalidades. Siempre es accesible el código fuente, con lo que puede ser directamente reutilizado.

- **Comando *lookfor* (búsqueda de palabras clave)**

Aunque más complicado de utilizar, proporciona en ocasiones información extra. El comando *lookfor* permite buscar entre las descripciones de todas las funciones de Matlab, aquellas que contienen la palabra clave que indiquemos.

1.3. Variables y matrices en Matlab.

Matlab soporta nombres de variable de hasta 19 caracteres, y distingue entre mayúsculas y minúsculas.

El tipo de las variables puede ser:

- Entero
- Real
- Complejo
- Carácter

... y es asignado automáticamente.

Una sentencia de creación de variable es, por ejemplo:

```
>> x = 7

x =
      7
```

Esta sentencia crea la variable entera **x** y le asigna el valor **7**. Matlab muestra en pantalla el resultado de cada operación. Para evitarlo basta poner un punto y coma después de cada sentencia:

```
>> x = 7;
```

Todas las variables en Matlab son consideradas matrices. Las posibilidades que utilizaremos son:

- Matriz **n x m**: matriz bidimensional
- Matriz **n x 1** ó **1 x n**: vector (se maneja exactamente igual que una matriz)
- Matriz **1 x 1**: escalar (también se maneja exactamente igual que una matriz).

La forma de definir una matriz en Matlab es elemento a elemento:

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

A =
     1     2     3
     4     5     6
     7     8     9
```

Como puede apreciarse en el ejemplo, los distintos elementos de una fila se separan mediante espacios (o comas) y las distintas filas se separan mediante puntos y coma.

Algunas posibilidades de manejo de variables que ofrece Matlab:

- comprobar el contenido de alguna variable: basta con teclear su nombre en la ventana de comandos

```
>> x

x =
     7
```

- listar todas las variables existentes en un determinado momento: comando **who**.

```
>> who

Your variables are:

A         x
```

- eliminar alguna variable de memoria: comando **clear**.

```
» clear x
» who

Your variables are:

A
```

  Podemos observar cómo la variable **x** ha desaparecido de la memoria.

1.4. Manejo de matrices.

Matlab ofrece bastantes facilidades para el manejo de matrices. Volviendo al ejemplo anterior:

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

```
A =
     1     2     3
     4     5     6
     7     8     9
```

- Podemos acceder a cualquier elemento de la matriz especificando la fila y la columna:

```
» A (1,3)

ans =
     3
```

Nota: **ans** es la variable por defecto donde Matlab guarda cualquier resultado; si hubiéramos deseado utilizar otra variable deberíamos haberlo especificado:

```
» k = A(1,3)

k =
     3
```

- También se puede acceder a toda una fila o toda una columna, utilizando el operador dos puntos:

```
»  A(2,:)

ans =
     4     5     6

» A (:,3)

ans =
     3
     6
     9
```

- O bien a grupos de filas y/o columnas:

```
» A(1:2,2:3)

ans =

     2     3
     5     6
```

- También es posible modificar cualquier elemento de una matriz:

```
» A(1,1) = 9

A =
```

```
     9     2     3
     4     5     6
     7     8     9
```

- E incluso añadir elementos a una matriz dada:

```
» A(4,4) = 1

A =

     9     2     3     0
     4     5     6     0
     7     8     9     0
     0     0     0     1
```

Podemos ver cómo los elementos no especificados se rellenan con ceros.

1.5. Principales operadores aritméticos.

Matlab ofrece una serie de operadores aritméticos válidos tanto para cálculo matricial como para cálculo escalar:

- Suma: +
- Resta: -
- Producto: *
- División: /
- Traspuesta: ’
- Potencia: ^

En algunas ocasiones podrán presentarse ambigüedades. Por ejemplo, al multiplicar dos matrices caben dos posibilidades: producto matricial o producto elemento a elemento. Veamos cómo se resuelven:

```
» A = [1 2;3 4]

A =

     1     2
     3     4

» B = [2 4; 6 8]

B =

     2     4
     6     8

» C = A*B       % producto matricial
```

```
C =

    14    20
    30    44

»  D = A.*B % el punto indica operación elemento a elemento

D =

     2     8
    18    32
```

Localización de ayuda: la información sobre operadores puede ser encontrada en: Help Window → Operators and special characters

Además de los operadores comentados, existen una serie de funciones muy útiles en cálculo matricial:

- obtención de la matriz inversa: función **inv**:

```
» A = [1 2;3 4]

A =

     1     2
     3     4

» B = inv(A)

B =

   -2.0000    1.0000
    1.5000   -0.5000
```

- creación de una matriz de ceros o unos: funciones **zeros** y **ones**:

```
» A = zeros(1,4)

A =

     0     0     0     0

» B = ones(2,3)

B =

     1     1     1
     1     1     1
```

- creación de un vector de términos crecientes o decrecientes:

```
» a = [0:1:5]   % inicio 0, fin 5, incremento 1

a =

     0     1     2     3     4     5

» a = [5:-1:0]  % inicio 5, fin 0, incremento -1

a =

     5     4     3     2     1     0

» a = [0:.2:1]  % inicio 0, fin 1, incremento .2

a =

         0       0.2000       0.4000       0.6000       0.8000
1.0000
```

  Podemos crear cualquier vector creciente o decreciente que deseemos. Esta operación será bastante útil para formar bases de tiempo sobre las que evaluar el valor de funciones.

Localización de ayuda: la información sobre operaciones matriciales puede ser encontrada en:
Help Window → Elementary matrices and matrix manipulation.

1.6. Modos de trabajo.

Matlab permite trabajar de dos maneras distintas:

- **Mediante la introducción directa de comandos:**

Tecleando comandos desde la ventana principal de Matlab podemos realizar operaciones paso a paso. Será el método de trabajo a emplear para hacer pruebas o bien para operaciones sencillas no repetitivas.

- **Mediante creación de programas (*.m)**

La misma secuencia de comandos que podríamos introducir desde la ventana principal puede archivarse en un fichero (que debe tener terminación '.m') y ser ejecutado posteriormente desde la ventana de comandos sencillamente tecleando el nombre del fichero.

Los programas creados pueden ser de dos tipos:
- **macros**: programas que no tienen argumentos de entrada ni de salida.
- **funciones**: programas con parámetros: ofrecen más versatilidad.

1.7. Facilidades para la programación en Matlab.

En comparación con otros lenguajes de programación, Matlab ofrece muchas facilidades para el usuario. Básicamente, cabe destacar:

- Elección automática del tipo de las variables
- Dimensionamiento automático de las matrices
- Posibilidad de manejar números complejos de modo intuitivo
- Posibilidad de funcionamiento en modo interpretado (chequeo de sentencias)
- Entorno de depuración integrado (últimas versiones)

1.8. Sentencias de control: bucles, comparaciones, ... en Matlab.

Se muestra a continuación la sintaxis de las principales sentencias de control de Matlab:

Bucles:

```
for variable = expresion
     sentencias
end

while expresión
     sentencias
end
```

Sentencia condicional if/else/elseif:

```
if expresión
     sentencias

elseif expresión
     sentencias

elseif expresión
     sentencias

else
     sentencias
end
```

Nota: las claúsulas *else* y *elseif* no son necesarias.

Ejemplo:

Deseamos crear una función Matlab que, a partir de una matriz dada, genere una matriz cuadrada añadiendo filas o columnas de ceros, según sea necesario. La función se llamará **cuadra** y se guardará en el fichero **cuadra.m**, en el directorio de cada usuario.

Paso 1: creación del fichero 'cuadra.m'

Con la opción **File** → **New** → **M-file** o bien con el botón ***New*** se lanza el editor/depurador de código Matlab, donde crearemos nuestra función.

El código de nuestra función tendrá el siguiente aspecto:

```
% convierte una matriz en cuadrada añadiendo ceros
function b = cuadra(a)

b = a;                % copia matriz entrada

[x,y] = size(b);      % obtiene dimensiones

if x>y
   b(:,y+1:x) = 0;    % añade columnas

elseif y>x
   b(x+1:y,:) = 0;    % añade filas
end

return
```

Si analizamos un poco en detalle este código, encontraremos elementos que necesariamente deberemos incluir en cualquier función que deseemos crear:

- Línea de comentario: es importante que la primera línea de una función contenga un texto explicativo, será la línea que se muestre al solicitar ayuda. Debe comenzar con el símbolo **%**.

- Declaración de la función: es obligatoria en cualquier **función**: especifica los parámetros de entrada y salida. En **macros** no existe esta línea.

- Cuerpo de la función: contiene todas las operaciones que deseemos realizar.

- **return**: sentencia de finalización de función. Se devolverá el valor que tenga asignada la variable que se especificó como salida (en este caso, la variable **b**).

Paso 2: Selección del directorio o carpeta donde guardar el programa

El directorio donde se archivan por defecto las funciones de usuario puede no ser accesible dependiendo de los sistemas. En nuestro caso deberemos utilizar el directorio de invitado correspondiente –lo ideal es crear un subdirectorio de funciones Matlab-. El fichero se guardará con nombre **cuadra.m** (es importante, éste será el nombre con el que accedamos a la función).

Paso 3: Modificación del path de Matlab

Matlab necesita conocer en qué directorios existen programas de usuario. Para ello dispone de la variable **path**, que debemos modificar adecuadamente. La sentencia a emplear será:

```
>> path (path, 'nuevodirectorio')
```

donde *nuevodirectorio* representa el directorio o carpeta donde se ha guardado el programa. Por ejemplo:

```
>> path (path, 'c:\invitado\alumno\matlab')
```

Otra forma de modificar el **path** es utilizando la opción del menú **File** → **Set Path...**. Esta opción abre una caja de diálogo que permite añadir los directorios que deseemos al **path**.

Paso 4: Comprobación usando la función help

Si hemos incluido la primera línea de comentario en nuestra función y hemos modificado la variable path adecuadamente, debemos obtener un resultado como éste:

```
>> help cuadra

convierte una matriz en cuadrada añadiendo ceros
```

Paso 5: Utilización de la función con una matriz ejemplo

Probaremos la función con una matriz cualquiera:

```
» a = [1 2; 3 4; 5 6]

a =

     1     2
     3     4
     5     6

» b = cuadra(a)

b =

     1     2     0
     3     4     0
     5     6     0
```

Vemos cómo se obtiene el resultado que esperábamos.

1.9. Representaciones gráficas en matlab.

Matlab ofrece facilidades para la creación de gráficos 2D y 3D. Estudiaremos en primer lugar la función **plot**, el medio más sencillo para realizar representaciones bidimensionales.

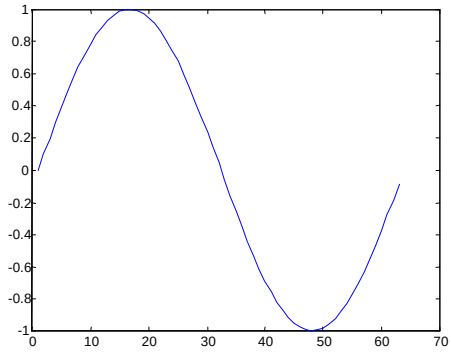
Existen diferentes sintaxis para la función **plot**. Intentaremos mostrar su funcionamiento con un ejemplo:

Supongamos que partimos de los siguientes datos iniciales:

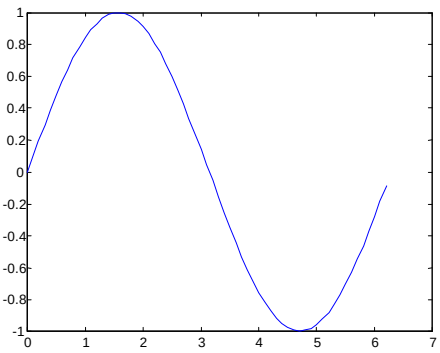
```
» a = [0:0.1:2*pi];   % base de ángulos entre 0 y 2π
» b = sin(a);         % función seno
» c = cos(a);         % función coseno
```

Comparemos dos formas de representar la función seno:

```
>> plot (b);
```



```
>> plot (a,b);
```



El resultado es aparentemente el mismo, pero existe una gran diferencia que es posible observar comparando los ejes *x* de ambas gráficas:

- **plot (b)** representa el vector **b** en el eje *y* frente a los índices de ese vector en el eje *x*.
- **plot (a,b)** representa el vector **b** en el eje *y* frente a los valores correspondientes del vector **a** en el eje *x*.

Normalmente nos interesará más la segunda opción y el vector **a** representará la escala de tiempos.

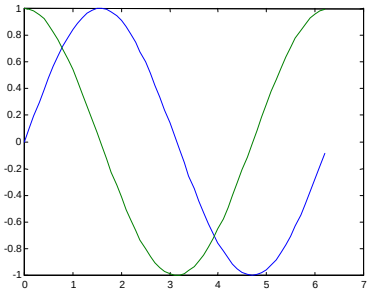
Veamos, a continuación, distintas formas de representar varias series en un mismo gráfico.

```
>> plot (a,b,a,c)
```

o bien

```
>> plot (a,[b;c])
```

Cualquiera de las instrucciones anteriores ofrece el siguiente resultado:



Vemos como se representan los vectores **b** y **c** en el eje *y* frente a un mismo vector **a** en el eje *x*.

Existe una forma más de representar varias series en un mismo gráfico: utilizando el comando **hold on**. El comando **hold on** sirve para añadir en el gráfico existente todos los comandos gráficos que se ejecuten a continuación. Para desactivar esta opción hay que ejecutar **hold off**. De esta forma si quisiéramos dibujar las dos series sobre el mismo gráfico ejecutaríamos:

```
>> plot(a,b);
>> hold on
>> plot(a,c);
```

Notas:

- Existen diferentes parámetros de la función **plot** para personalizar colores y tipos de líneas. Por ejemplo, para dibujar las series anteriores, una en rojo (**'r'**) y otra en azul (**'b'**), se ejecutaría:

  ```
  » plot(a,b,'r',a,c,'b')
  ```

  Si se deseara representar una serie mediante una línea continua (**'-'**) y la otra mediante una línea discontinua (**':'**) habría que ejecutar:

  ```
  » plot(a,b,'-',a,c,':')
  ```

  Para ver todas las opciones del comando **plot** consultar la ayuda de Matlab.

- Matlab posee funciones para asignar nombres a los ejes de un gráfico (**xlabel** e **ylabel**), así como para establecer el título de un gráfico (**title**). Consultar la ayuda de estas funciones para obtener más información

2. Ejecicios Matlab.

Ejercicio 1. Crear una función de Matlab que, a partir de dos matrices cuadradas, calcule el determinante de la suma. La declaración de la función debe ser la siguiente:

```
function res = detsuma(matriz1,matriz2)
```

La función debe almacenarse en un fichero con el nombre **detsuma.m**.

Ejercicio 2. Crear una función que representa la función *f(x) = a cos(bx)* en el intervalo de tiempo *[tmin,tmax]*. La función será declarada de la siguiente manera:

```
function dibujacos(a,b,tmin,tmax)
```

La función debe almacenarse en un fichero con el nombre **dibujacos.m**.

3. Introducción a Simulink.

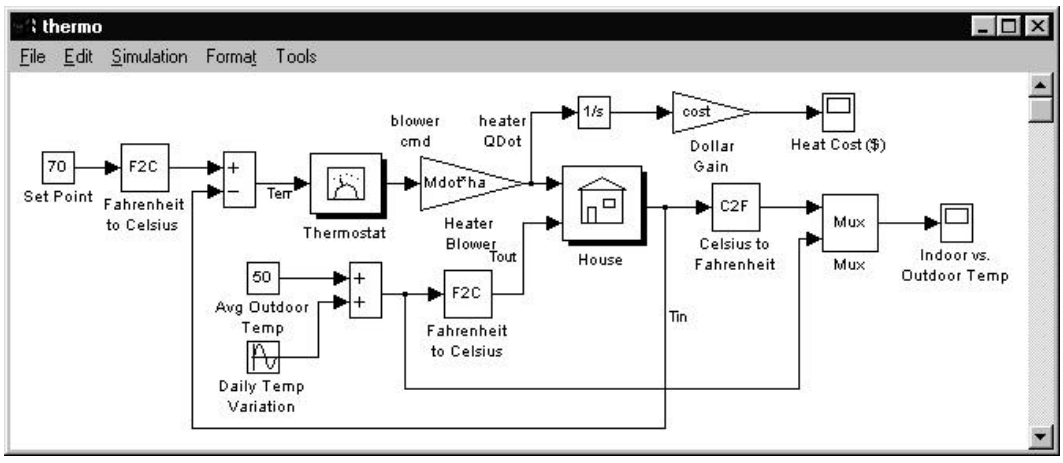
Simulink es un *toolbox* de Matlab que proporciona un entorno gráfico para modelar, simular y analizar sistemas dinámicos.

3.1. Características básicas de Simulink.

- Funcionalidad básica: simulación de sistemas dinámicos.

- Características principales:
  - Se trata de un entorno gráfico.
  - El sistema se representa como una interconexión de bloques elementales.
  - Cada bloque lleva asociado un modelo matemático que representa su relación entrada/salida.

Un ejemplo de representación de un sistema mediante Simulink sería el siguiente:



El esquema anterior representa un sistema de calefacción para una vivienda. Se supone conocida la variación de la temperatura en el exterior de la casa y en función de ello se puede observar en qué instantes se conecta y se desconecta la calefacción, cuáles son las variaciones de la temperatura en el interior de la casa y cuál es el coste de calefacción. Este ejemplo proviene de una *demo* de Simulink y será analizado más adelante.

3.2. Cómo encontrar ayuda en Simulink.

Como ha sido comentado anteriormente, Simulink es un toolbox de Matlab. Por tanto, los distintos tipos de ayuda de Matlab serán también válidos para Simulink:

- **Ayuda en línea**

Se accede a través de la ventana de comandos de Matlab tecleando *help nombrefunción*. La ayuda se obtiene en modo texto.

- **Ventana de ayuda (Help → Help Window)**

Exactamente igual que en Matlab.

- **Ayuda HTML (Help → Help Desk)**

Buscar en *'Other products: Simulink blocks'*.

- **Ejemplos (Help → Examples and Demos)**

El comando *'demo simulink'* tecleado en la ventana de Matlab nos ofrece todas las demo disponibles.

3.3. Cómo ejecutar Simulink.

- Desde la ventana de comandos de Matlab teclear 'simulink'
- Desde la ventana de comandos de Matlab, pinchar sobre el botón 'Nuevo modelo de Simulink'

3.4. Ejecución de un sistema ejemplo.

Como primer contacto con Simulink, ejecutaremos el ejemplo del sistema de calefacción de una vivienda mostrado anteriormente.

Formas de abrir el sistema ejemplo:

- Desde el menú **'File'** de la ventana de Simulink, seleccionar la opción **'Open'** y elegir el fichero **'C:\Matlab\toolbox\simulink\simdemos\thermo.mdl'**
- Desde la ventana de comandos de Matlab, teclear **'thermo'**.
- Desde la ventana de bloques de Simulink, a través del botón **'Demos'** y eligiendo, dentro de **'Complex Models'**, el modelo **'Thermodynamic Model of a House'**.

Forma de lanzar una simulación:

Lanzar una simulación equivale a poner el sistema a funcionar durante un periodo de tiempo determinado y observar los resultados que se obtienen. Desde el menú ***'Simulation'*** deberemos seleccionar la opción ***'Start'***.

El sistema indica con un aviso acústico cuando se ha alcanzado el final de la simulación; si deseáramos parar la ejecución antes de llegar al final, bastaría con seleccionar, dentro del menú ***'Simulation'***, la opción ***'Stop'***.

Es interesante comprobar como en las ventanas gráficas se han representado los datos que se deseaba obtener: la evolución de la temperatura en el interior de la casa (representada conjuntamente con la temperatura exterior) y los costes de calefacción.

Algunos detalles importantes sobre la simulación:

- Si no se han modificado los parámetros iniciales, Simulink habrá simulado el comportamiento del sistema durante las primeras 12 horas (43200 segundos).

- Los instantes de comienzo y fin de simulación se pueden cambiar dentro del menú ***'Simulation'*** eligiendo la opción ***'Parameters'***. Probaremos a relanzar la simulación con distintos tiempos de comienzo ***(Start time)*** y fin ***(Stop Time)***. El resto de los parámetros no se modificarán por ahora.

Las ventanas de representación gráfica de resultados:

- Estas ventanas funcionan de un modo similar a un registrador de datos o un osciloscopio digital. Serán bastante utilizadas en nuestras simulaciones y, por tanto, conviene familiarizarse con su comportamiento. En el diagrama aparecen con el nombre **'Scope'**.

- Los tres botones de la izquierda, cada uno de los cuales representa una lupa, sirven para hacer zoom sobre un área de la gráfica. El primero de ellos permite definir un rectángulo sobre el gráfico y ampliar exactamente esa área. Los otros dos permiten definir una recta horizontal o vertical respectivamente y hacen zoom sólo en una dimensión. Por último, el cuarto botón, que presenta un dibujo de unos prismáticos, sirve para ajustar el tamaño del gráfico a los datos disponibles.

- Si el tiempo de simulación es muy elevado, la gráfica no retiene todos los datos, sino que descarta los más antiguos y conserva sólo los últimos. Para modificar el número de datos a retener, se debe pinchar sobre el botón de la derecha (propiedades) y modificar, dentro de ***'Settings'***, el valor indicado para el campo ***'Limit rows to last...'*** que, por defecto, debe tener un valor de 5000. También es posible desactivar la opción de forma que no se descarte ningún valor.

3.5. Tipos de bloques en Simulink.

En el sistema elegido como ejemplo puede observarse como existen multitud de bloques distintos; todos los bloques de Simulink se estructuran en las siguientes categorías o librerías:

Sources: ***Entradas o fuentes de señales***

- Constantes
- Senoidales
- Cuadradas
- Escalón
- Aleatorias

Sinks: ***Salidas o dispositivos de visualización/almacenamiento de variables del sistema***

- Osciloscopio
- Fichero
- Gráfico

Discrete/linear/nonlinear: ***Representan sistemas sencillos mediante su relación entrada/salida***

- Discrete: sistemas discretos (muestreados). Dominio z.
- Linear: sistemas continuos lineales. Dominio s.
- Nonlinear: sistemas continuos no lineales. Dominio t.

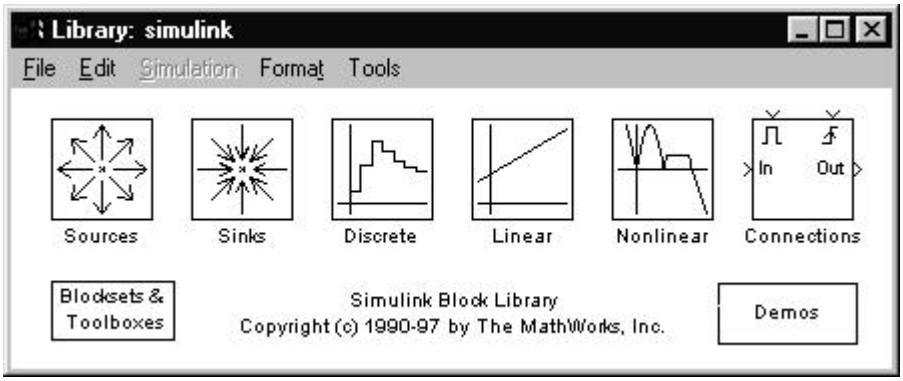
Connections: ***Se utilizan bien para conectar elementos o para estructurar los modelos.***

- Subsistema: permite jerarquizar diseños.
- Multiplexadores/demultiplexadores: agrupan o desagrupan señales.
- Memorias: permiten almacenar valores.

Blocksets/toolboxes: ***Elementos específicos para diferentes aplicaciones:***

- comunicaciones
- redes neuronales
- control
- etc.

Todas estas categorías de bloques se muestran en la ventana inicial de Simulink. El procedimiento de trabajo para la creación de un modelo será seleccionar los elementos adecuados de entre los presentes en estas categorías, colocarlos sobre la ventana de diseño y establecer las conexiones correspondientes entre ellos. La figura siguiente muestra la ventana inicial de Simulink:

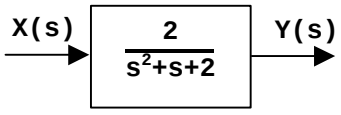


Debemos comprobar como, haciendo doble clic sobre una cualquiera de las categorías, se muestran todos los bloques correspondientes a la misma.

3.6. Creación de un modelo sencillo.

Como ejemplo, crearemos un modelo que nos permita observar las variaciones en el comportamiento de un sistema de segundo orden en función de la posición de sus polos.

En primer lugar, utilizaremos un sistema definido por la siguiente función de transferencia:



Y comprobaremos cuál es su respuesta ante una señal de tipo escalón.

Primer paso: creación del modelo.

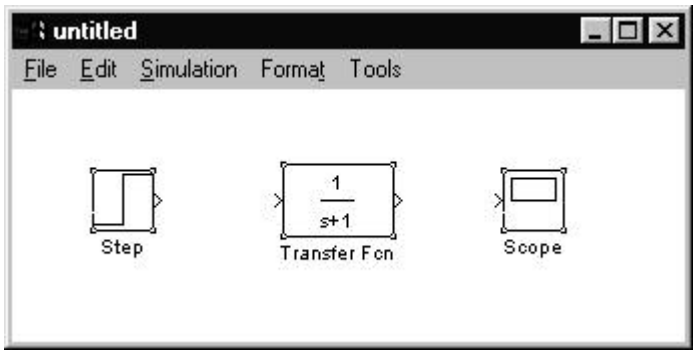
Desde la ventana inicial de Simulink, elegiremos, dentro del menú ***'File'***, la opción ***'New'*** y seleccionaremos ***'Model'***. Automáticamente se abrirá una ventana en blanco que será la ventana de diseño para nuestro modelo.

Segundo paso: introducción de bloques en el modelo.

Los elementos se introducen haciendo un arrastre con el ratón hacia la ventana de diseño. En nuestro caso requeriremos los siguientes bloques:

- Dentro de la librería **'Sources'**, el bloque **'Step'**. Este elemento se corresponde con una señal de tipo escalón, que será la señal de entrada para nuestro sistema.
- Dentro de la librería **'Linear'**, el bloque **'Transfer Fcn'**. Nos permitirá representar cualquier función de transferencia.
- Dentro de la librería **'Sinks'**, el bloque **'Scope'**. Será el que utilicemos para visualizar los resultados.

Una vez introducidos estos modelos, la ventana de diseño debería presentar un aspecto como el siguiente:



Tercer paso: modificación de parámetros en los bloques introducidos.

Todos los bloques de Simulink permiten una cierta configuración. En particular, el bloque de función de transferencia se puede configurar seleccionando los polinomios de numerador y denominador. Para ello se debe hacer doble clic sobre el bloque, con lo que aparecerá una ventana de introducción de parámetros.

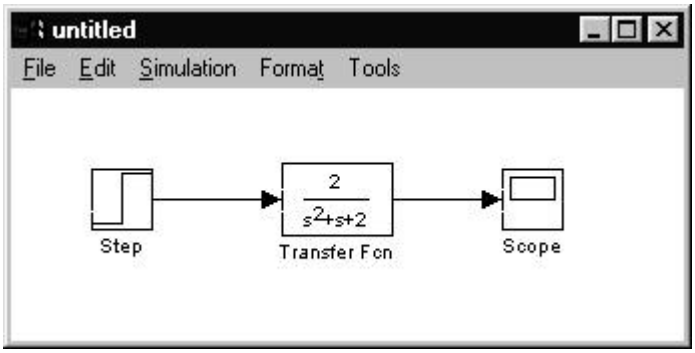
En esta ventana, los polinomios de numerador y denominador se introducen como vectores numéricos en orden de potencias decrecientes. Si recordamos cuál es la función de transferencia que queremos representar, deberemos introducir:

- Para el numerador: [2] (representa 2)
- Para el denominador: [1 1 2] (representa $x^2 + x + 2$)

El resto de bloques no se modificarán por ahora.

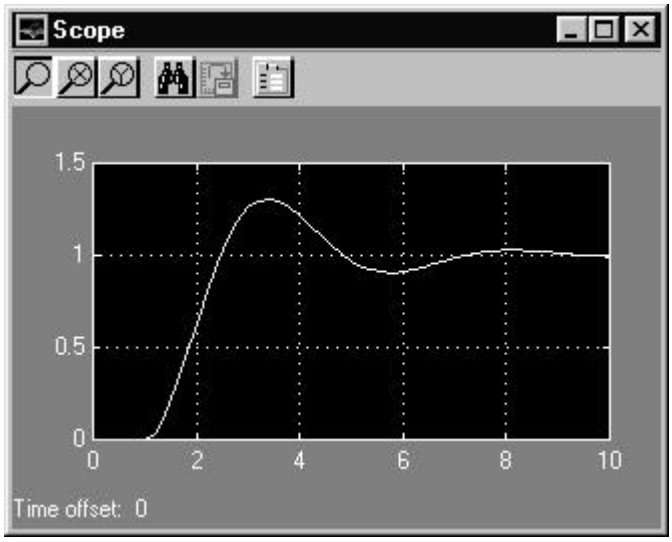
Cuarto paso: interconexión de elementos.

Una vez situados los elementos en la ventana de diseño, es necesario establecer conexiones entre ellos. Para conectar 2 elementos debe hacerse un arrastre con el ratón desde la salida de uno de ellos hasta la entrada del elemento correspondiente. Una vez establecidas las dos conexiones necesarias para nuestro sistema, deberíamos obtener un resultado como el que se muestra:



Quinto paso: lanzamiento de la simulación y comprobación de resultados.

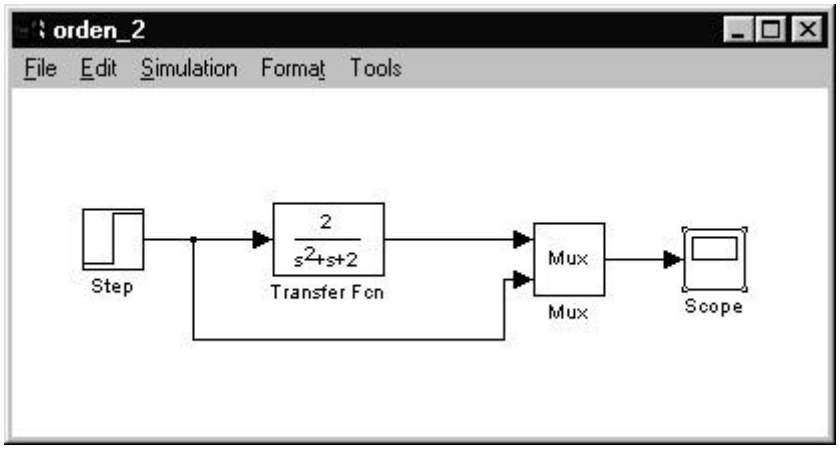
Lanzaremos la simulación con la opción *Start* del menú *Simulation* y comprobaremos los resultados obtenidos haciendo doble clic sobre el bloque *Scope*. Se mostrará una ventana gráfica como la vista en el primer ejemplo sobre la que podremos hacer zoom para ajustar nuestra curva. Deberá aparecer algo similar a lo que se muestra a continuación:



Podemos ver como el sistema presenta oscilaciones amortiguadas sobre la posición de equilibrio. Si lo deseáramos, podríamos aumentar el tiempo de simulación para comprobar cómo la señal termina por estabilizarse.

Terminada esta primera parte del ejercicio, refinaremos un poco más nuestro modelo para mostrar en el gráfico tanto la señal de salida como la señal de entrada (el escalón).

Para visualizar dos señales en el elemento *Scope* es necesario recurrir al bloque multiplexor *Mux* de la librería *Connections*. Este bloque tiene por finalidad agrupar dos o más señales. De este modo, se agruparán la señal de entrada y la señal de salida y el conjunto será lo que se envíe al osciloscopio. Las conexiones se deben realizar tal y como se muestra en la figura siguiente:

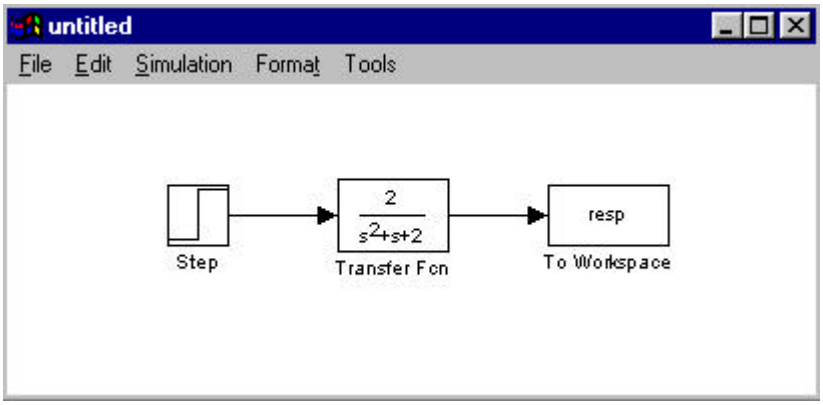


En este esquema hay algo nuevo que aún no sabemos hacer: cómo crear una rama nueva sobre una conexión ya existente (la segunda rama que llega al multiplexor no parte de la salida de un bloque, sino de un punto intermedio de una conexión). La forma de crear este tipo de conexiones es pulsando el botón **'Ctrl'** al mismo tiempo que se empieza a arrastrar con el ratón.

Lanzando una nueva simulación, debemos comprobar que en la ventana del osciloscopio se muestran las dos señales: la entrada y la salida.

3.7. Interacción entre Simulink y Matlab.

Supongamos que en el modelo creado anteriormente, en lugar de visualizar la respuesta del sistema en Simulink deseamos visualizarla en Matlab. Para ello deberíamos ser capaces de almacenar en una variable de Matlab la respuesta del sistema, con la finalidad de generar la gráfica de la respuesta utilizando el comando **plot**. Esto se puede realizar utilizando el bloque *To Workspace* de la librería *Sinks*. Este bloque permite definir variables en el entorno de Matlab en las que se almacenarán valores generados en Simulink. En la siguiente figura se muestra la disposición de los bloques para almacenar la respuesta del sistema en una variable de Matlab:



Como puede observarse se ha añadido un bloque *To Workspace* para almacenar la respuesta del sistema. La variable en la que se almacenará la respuesta del sistema se ha denominado *resp*. El nombre de la variable es un parámetro del bloque *To Workspace*. La variable definida tendrá una columna por cada elemento de entrada al bloque y una fila por cada paso de simulación. Puesto que en este caso la entrada al bloque únicamente es la respuesta del sistema, la variable *resp* será un vector columna.

Una vez realizada la simulación aparecerá en Matlab la variable *resp* (comprobar con el comando **who**). Para obtener la gráfica de la respuesta del modelo simulado únicamente habrá que ejecutar en Matlab:

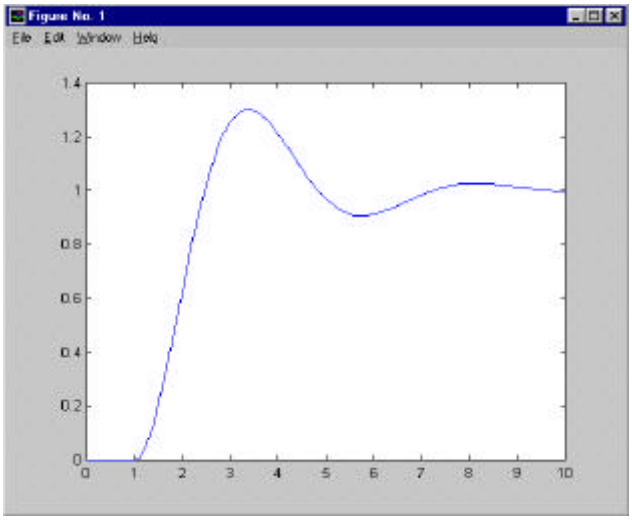
```
» plot(resp)
```

Si observamos la gráfica se puede apreciar que en el eje *x* no aparece el tiempo de simulación. Esto se debe a que en el comando **plot** no hemos especificado el vector que representa el tiempo.

Por defecto, después de realizar una simulación, se genera un vector columna llamado *tout* en el que se almacenan los instantes de tiempo de simulación. (Comprobar en el menú *Simulation* → *Parameters*, pestaña *Workspace I/O*, que está seleccionada la opción de grabar el tiempo al espacio de trabajo de Matlab). Por lo tanto, para representar la gráfica de la respuesta del sistema frente al tiempo habrá que ejecutar:

```
» plot(tout,resp)
```

Ejecutando este comando se obtiene la gráfica:

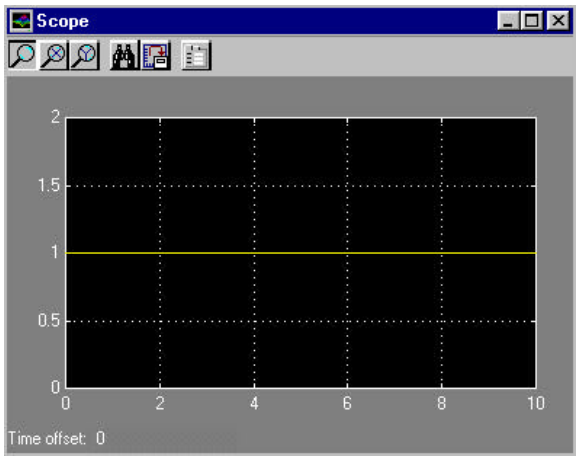


Para finalizar hay que señalar que desde Simulink se tiene acceso a todas las variables definidas en el entorno de Matlab.

4. Ejercicios Simulink.

Ejercicio 3. Representar en Simulink la respuesta a escalón unitario de los siguientes sistemas de 2º orden:

El escalón debe introducirse en el instante inicial de simulación, es decir, debe ser como el que se muestra a continuación:



Ejercicio 4. Sobre cada uno de los esquemas anteriores, y haciendo uso de las posibilidades de zoom de los gráficos, se deberán tomar con precisión las siguientes medidas, para cada respuesta:

- valor en régimen permanente (ganancia)
- sobreoscilación en %
- tiempo de pico de sobreoscilación
- tiempo de establecimiento (entrada en una franja de ±5% valor final)

Ejercicio 5. Visualizar la respuesta de cada sistema junto a la entrada escalón introducida en Simulink.

Ejercicio 6. Representar la respuesta de cada sistema en Matlab utilizando el comando **plot**.

Ejercicio 7. Representar la respuesta de cada sistema junto a su entrada en Matlab utilizando el comando **plot**.

Importante

- Debe entregarse un informe detallando cada uno de los ejercicios realizados en la práctica.

- El plazo de entrega del informe de la primera práctica finaliza el día 22 de enero de 2001.