Introducción a Matlab y su aplicación al Análisis y Control de Sistemas

Prácticas de Control Automático

3^o Ingeniería Industrial

Manuel Vargas, Manuel Berenguel
Escuela Superior de Ingenieros
Universidad de Sevilla

28 de Octubre de 2003

Contenido

Capítulo 1

INTRODUCCIÓN A MATLAB

1.1 Introducción

En estas notas se pretende realizar una introducción muy básica a MATLAB, orientándola en el siguiente capítulo al estudio de sistemas de control. En líneas generales, MATLAB es una herramienta interactiva basada en matrices para cálculos científicos y de ingeniería (de hecho, el término MATLAB procede de *matrix laboratory*). Desde el punto de vista del control, MATLAB se puede considerar un entorno matemático de simulación que puede utilizarse para modelar y analizar sistemas. Permitirá el estudio de sistemas continuos, discretos, lineales y no lineales, mediante descripción interna y externa, en el dominio temporal y frecuencial.

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 el *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 operaciones.

Las referencias al *Control System Toolbox* se realizarán directamente en los ejemplos que acompañan a estas notas.

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 ([5], [4], [3]). Para el desarrollo de las mismas se ha utilizado asimismo, una serie de referencias básicas en control: [1], [6], [7], [8], etc.

1.2 INSTALACIÓN

La forma normal en la que se encuentra el sistema una vez instalado es la siguiente (versión 3.5.1):

```
\matlabr11\bin
          \extern
          \help
          \notebook
          \simulink
          \sys
          \toolbox
             \control
             \local
             \matlab
             \simulink
          \work
```

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:

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

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

Como se desprende del árbol de directorios, los *toolboxes* se suelen instalar en forma de subdirectorios en el disco duro, colgando del subdirectorio **TOOLBOX**. En ellos se encuentran también funciones **.m** orientadas al control de sistemas. Además, se pueden incorporar otros *toolboxes* (SIGNAL PROCESSING, IMAGE PROCESSING, ROBUST CONTROL, NON-LINEAR CONTROL, SYSTEM IDENTIFICATION, etc), e incluso funciones propias del usuario.

`matlab\general`	-	Comandos de propósito general
`matlab\ops`	-	Operadores y caracteres especiales
`matlab\lang`	-	Constructores del lenguaje de programación
`matlab\elmat`	-	Matrices elementales y manipulación matricial
`matlab\elfun`	-	Funciones matemáticas elementales
`matlab\specfun`	-	Funciones matemáticas especiales
`matlab\matfun`	-	Funciones matriciales - álgebra lineal numérica
`matlab\datafun`	-	Análisis de datos y transformada de Fourier
`matlab\polyfun`	-	Interpolación y polinomios
`matlab\funfun`	-	Funciones de funciones y métodos para ODE
`matlab\sparfun`	-	Funciones para matrices dispersas
`matlab\graph2d`	-	Gráficos en dos dimensiones
`matlab\graph3d`	-	Gráficos en tres dimensiones
`matlab\specgraph`	-	Gráficos especializados
`matlab\graphics`	-	Manipulación de gráficos
`matlab\uitools`	-	Herramientas de interfaz gráfica de usuario (GUI)
`matlab\strfun`	-	Cadenas de caracteres
`matlab\iofun`	-	Funciones para entrada/salida de ficheros
`matlab\timefun`	-	Hora y fecha
`matlab\datatypes`	-	Tipos de datos y estructuras
`matlab\winfun`	-	Ficheros de interfaz con Windows (DDE/ActiveX)
`matlab\demos`	-	Ejemplos y demostraciones
`simulink\simulink`	-	Simulink
`simulink\blocks`	-	Librería de bloques de Simulink
`simulink\simdemos`	-	Ejemplos y demostraciones de Simulink
`toolbox\control`	-	Paquete de Control de Sistemas
`toolbox\local`	-	Librería de funciones locales

Tabla 1.1: Listado del comando `help`

1.3 PRIMEROS PASOS

Una vez arrancado MATLAB , se abre la ventana de comandos en la que aparece el *prompt* o línea de comandos (representado con el símbolo ≫). 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*, tanto las contenidas en el subdirectorio **MATLAB**, como otras contenidas en subdirectorios eventualmente añadidos por el usuario (ver Tabla 1.1).

Para obtener información sobre cualquiera de las funciones se introduce `help` *nombre-función*.

Ejemplo: *help impulse* (*impulse* es una función que calcula la respuesta impulsional de un sistema y que se encuentra en el CONTROL SYSTEM TOOLBOX).

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 "posibilidades" de MATLAB.

1.4 FUNCIONES Y SÍMBOLOS RELACIONADOS CON EL ENTORNO

– Con el comando `path` puede comprobarse cuáles son las localizaciones de los ficheros y programas con los que va a trabajar MATLAB, pudiendo añadirse nuevos subdirectorios (incluso personales) a conveniencia. La forma más cómoda de interactuar con dichas localizaciones es mediante la opción *File/Set-Path...* en el menú de la ventana de comandos. Para poder usar cualquier función **.m**, como por ejemplo las contenidas en el paquete de control, bastará con que el camino **\matlabr11\toolbox\control** esté incluido en el *path* de MATLAB (cosa que ocurrirá si el paquete se instaló adecuadamente).

– Por otro lado, MATLAB comienza trabajando, por defecto, en el subdirectorio **matlabr11\work**. Si queremos cambiar de *directorio de trabajo* en cualquier momento, podemos hacerlo con el comando `cd` *camino*. Puede utilizarse en nombre completo del comando si se desea: `chdir`. Cabe decir que todas las funciones **.m** que existan en el directorio de trabajo serán localizadas sin necesidad de tener que incluir dicho directorio en el *path* de MATLAB.

– El comando `pwd` nos indica cuál es el directorio de trabajo actual.

– Para mostrar el contenido del directorio de trabajo, se pueden emplear los comandos `dir` ó `ls`. El comando `delete` *nombre-fichero* puede emplearse para eliminar un archivo del directorio de trabajo. Asimismo, se pueden realizar operaciones típicas de línea de comandos del sistema operativo DOS, introduciendo el comando correspondiente precedido por el símbolo "`!`".

– Resulta interesante tener en cuenta que la línea de comandos de MATLAB posee "memoria" y podemos recuperar comandos introducidos previamente, haciendo uso de las teclas de movimiento de cursor arriba y abajo. Para una localización más eficaz de algún comando introducido previamente, podemos teclear los primeros caracteres del mismo antes de usar el cursor arriba y sólo buscará entre los comandos ya introducidos aquéllos cuyos primeros caracteres coincidan con los introducidos.

– Otra posibilidad que se ofrece es la de introducir varios comandos en una misma línea de la ventana de comandos, separados por coma o punto y coma.

– Puede "limpiarse" el contenido de la ventana de comandos mediante la instrucción `clc`.

– El símbolo `%` sirve para introducir comentarios. Todo lo escrito desde ese símbolo hasta el final de la línea será ignorado por el intérprete de MATLAB. El uso de comentarios puede no resultar demasiado interesante en la línea de comandos, aunque sí lo será cuando se estén escribiendo programas, como se verá más adelante.

– Si se quiere guardar toda la sesión en un archivo, basta usar el comando `diary` *nombre-archivo*. Dicho archivo contendrá los comandos introducidos y los correspondientes resultados. Cuando no se quiera seguir almacenando la información se introducirá `diary off`.

– Si se desean almacenar todas las variables de memoria en un fichero, junto con sus valores actuales, se usa el comando `save` *nombre-fichero*. Esto crea un fichero binario en el directorio de trabajo actual con el nombre introducido y con extensión **.mat**. Si no se da el nombre del fichero, se 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*. 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 formato de visualización en la ventana de comandos puede modificarse usando `format`:

- `format long`: Presentará mayor número de decimales en pantalla al presentar los resultados en punto flotante.
- `format short`: Es el modo por defecto, presenta un número de decimales menor. Este formato no afecta para nada a la precisión de los cálculos, es sencillamente una cuestión de visualización.
- `format compact`: Deja menor número de líneas en blanco en la visualización de los resultados, permitiendo dar cabida a más información previa en la ventana de comandos sin necesidad de hacer *scroll*.
- `format loose`: Es el modo por defecto, se dejan más líneas de separación durante la visualización.

También puede modificarse el formato de visualización a través de las opciones de menú: *File/Preferences/General*

– Para detener la ejecución de un comando, se usa `Ctrl-C`.

– La salida del sistema se efectúa al introducir `quit` ó `exit`, o simplemente cerrando la ventana de comandos.

1.5 INTRODUCCIÓN DE DATOS. USO DE LA VENTANA DE COMANDOS

El elemento básico en MATLAB es **la matriz compleja de doble precisión**, estructura que abarca realmente todo tipo de datos, desde escalares tales como números reales o complejos, hasta vectores o matrices de tamaños arbitrarios. Implícitamente se usa la notación matricial para introducir polinomios y funciones de transferencia, de la forma que se explicará más adelante. Por otro lado, si se dispone de una representación de un sistema lineal en el espacio de estados de la forma:

$$\begin{aligned} \dot{x} &= A\,x + B\,u \\ y &= C\,x + D\,u \end{aligned}$$

bastaría con introducir los valores de los elementos de las matrices A, B, C y D, para tener descrito al sistema. Estos elementos se podrían introducir de la siguiente forma:

```
A=[1 0 2;2 2 0;0 0 1]
B=[1, 0,0]'
C=[1 1 sqrt(2)]
D=0;
```

A la vista de esta serie de comandos se pueden comentar varias cosas:

- Si al final de la introducción de un comando cualquiera no se pone punto y coma (`;`), aparecerá explícitamente en pantalla el resultado de dicho comando. En caso contrario, el comando se ejecutará pero no se mostrará su resultado. Dicho resultado se habrá almacenado en la variable a la que se asigna o, si no se realiza asignación, se guardará en una variable de entorno llamada `ans`. En caso de que se asigne a una variable, ésta se creará automáticamente, sin necesidad de una declaración previa.
- Los elementos de cada fila de una matriz se pueden introducir separados por espacios o por comas, indistintamente.
- Para separar filas de una matriz se usa `;` o un simple retorno de carro. Esta última opción puede facilitar muchas veces la visualización de la matriz que se está introduciendo.
- Para transponer matrices se usa el apóstrofe.
- Los elementos de vectores y matrices pueden ser reales, complejos e incluso expresiones, como vemos en el caso del último elemento del vector C.
- Si se está introduciendo un comando o conjunto de ellos cuya sintaxis sea muy larga, se puede continuar en la siguiente línea introduciendo al final de la actual tres puntos seguidos (`...`).
- Las variables a las que se asignan resultados, así como las variables de entorno, se almacenan en lo que se denomina *el espacio de trabajo* de MATLAB (*workspace*).

En este caso, se han creado una serie de variables (en particular, matrices) mediante la introducción explícita de sus elementos en línea de comandos. Otras formas de producir variables podrían ser: generándolas mediante funciones y declaraciones, creándolas en un archivo `.m`, cargándolas desde un archivo de datos externo mediante el comando `load` (bien se trate de ficheros de datos ASCII o bien de ficheros binarios con formato de datos de MATLAB `.mat`).

Además de variables numéricas, escalares o matriciales, en MATLAB pueden usarse cadenas de caracteres. Para ello se delimita una secuencia de caracteres mediante apóstrofes:

```
cadena = 'ejemplo de cadena de caracteres'
```

Para hacer referencia a cualquiera de los caracteres que componen una cadena, podemos hacerlo como si de un vector se tratara (la forma de indexar vectores y matrices se verá más adelante).

1.6 Variables de entorno y variables especiales

Existen una serie de variables predefinidas en MATLAB, son las siguientes:

- `ans`: Contiene la respuesta (*answer*) del último comando ejecutado, cuando el resultado de dicho comando no se asigna explícitamente a ninguna variable.
- `eps`: Da el valor de la precisión con la que la máquina realiza las operaciones en punto flotante. Típicamente, esta precisión es del orden de 10^{-17}.
- `pi`: π.
- `i`, `j`: $\sqrt{-1}$. Constante imaginaria.
- `inf`: ∞. Se trata de un valor excesivamente grande para ser almacenado.
- `NaN`: *Not a number*. Es el resultado que se proporciona si durante una operación se produce una indeterminación, del tipo $0 \cdot \infty$, $\frac{0}{0}$, $\frac{\infty}{\infty}$, etc.
- `clock`: Reloj.
- `date`: Fecha.
- `flops`: Número de operaciones en punto flotante realizadas hasta el momento.

El comando `who` muestra las variables existentes en el espacio de trabajo generadas por el usuario, pero no las variables especiales.

Para borrar alguna variable de memoria se utiliza `clear` *nombre-variables* separadas por espacios. Pueden borrarse todas las variables a la vez si no se especifica ningún nombre a continuación del nombre del comando.

1.7 ELEMENTOS DE LAS MATRICES

En este punto es importante comentar uno de los elementos más potentes de MATLAB, que es el símbolo `:`, que permite generar una secuencia, y en particular permitirá referenciar varios elementos de una matriz. Veamos algunos ejemplos en los que se usa este operador:

`1:0.1:10` Generará una secuencia comenzando por 1 hasta 10, cada elemento de la secuencia estará separado del anterior en 0.1.

`1:10` Si se obvia el valor central, la separación entre cada dos elementos de la secuencia será 1.

`[1:0.1:10]` Si lo ponemos entre corchetes, estaremos generando un vector con los elementos de la secuencia.

En la forma más directa, los elementos de una matriz se referencian mediante $A(i, j)$, donde i y j son los índices del elemento correspondiente. Podemos usar una secuencia que facilitar la indexación de múltiples elementos, como en los siguientes ejemplos:

`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.

Lógicamente, en estos casos, los elementos especificados como inicio, final e incremento para producir la secuencia deben ser enteros.

Otra forma de generar datos secuencialmente es usando los comandos `linspace` y `logspace`, su formato es:

```
t = linspace(n1,n2,n);
w = logspace(n1,n2,n);
```

El comando `linspace` genera un vector desde $n1$ a $n2$ de longitud n, cuyos componentes poseen valores espaciados linealmente. Por su parte, `logspace` produce también un vector de n elementos, pero sus valores están espaciados logarítmicamente desde 10^{n1} a 10^{n2}. Este último comando resultará útil para la generación de escalas frecuenciales para el análisis de sistemas mediante diagramas de Bode, Nyquist, etc.

1.8 Operaciones con matrices

Las operaciones comunes con matrices son:

- Suma: `+`
- Resta: `-`
- Multiplicación: `*`
- División derecha `/` ($x = b/A$ es la solución del sistema de ecuaciones $x * A = b$. Es decir calcula la inversa de la matriz A y multiplica b por la derecha por dicha inversa)
- División izquierda \ ($x = A \setminus b$ es la solución de $A * x = b$. Es decir, igual que en el caso anterior, pero realiza la multiplicación de la inversa con b por la izquierda)
- Potenciación `^`. Este operador permite, en particular, implementar otra forma de realizar la inversión de una matriz: `A^(-1)`.
- Conjugada traspuesta `'`

Cabe mencionar la potencia de los operadores `/`, \, y `^`, puesto que si la matriz A no es cuadrada, automáticamente se realiza el cálculo de su pseudoinversa, lo que equivaldría a resolver el sistema de ecuaciones correspondiente por mínimos cuadrados.

Las mismas operaciones que se han enumerado se pueden realizar elemento a elemento, anteponiendo un punto a cualquiera de los operandos anteriores. Como ejemplo, el siguiente comando realizaría el producto de cada elemento de la matriz A con su correspondiente de la matriz B (para que dicho producto sea realizable, obviamente, dichas matrices deben tener las mismas dimensiones):

```
A .* B
```

Además de los operadores anteriores, existen funciones tales como:

- Trigonométricas estándar: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `atan2`
- Trigonométricas hiperbólicas: `sinh`, `cosh`, `tanh`, `asinh`, `acosh`, `atanh`
- Trascendentales: `log`, `log10`, `exp`, `sqrt`
- Manipulación de números complejos:
  - `real`: parte real de un escalar o de los elementos de una matriz.
  - `imag`: parte imaginaria.
  - `conj`: proporciona el conjugado de un escalar o la matriz conjugada a una dada.

– Cálculo del módulo: `abs` permite calcular tanto el valor absoluto de un escalar real como el módulo de un escalar complejo o el módulo de un vector.

– Funciones típicas de matrices:

- `det`: determinante de una matriz
- `inv`, `pinv`: inversa y pseudoinversa
- `eig`: obtención de autovalores
- `rank`: rango de la matriz
- `norm`: norma de una matriz (norma 2, norma 1, norma infinito, norma de Frobenius)
- `trace`: traza de la matriz
- `diag`: produce un vector conteniendo los elementos de la diagonal de una matriz, o si recibe un vector como parámetro, genera una matriz diagonal.
- `tril`: devuelve la matriz triangular inferior de una matriz dada
- `triu`: devuelve la matriz triangular superior de una matriz dada

– funciones para generar matrices:

- `eye(n)`: produce una matriz identidad de dimensión $n \times n$
- `zeros(n,m)`: genera una matriz de ceros de dimensión $n \times m$
- `ones(n,m)`: genera una matriz de unos de dimensión $n \times m$
- `rand(n,m)`: permite generar una matriz de valores aleatorios, entre 0 y 1, de dimensión $n \times m$
- `A = [A11,A12;A21,A22]`: podemos producir una nueva matriz por bloques, mediante su composición a partir de submatrices ya existentes.

1.9 FUNCIONES ORIENTADAS AL ANÁLISIS DE DATOS

Se trata de funciones que operan con vectores. Si se aplican a matrices operan columna a columna. Permiten realizar análisis sobre el conjunto de datos contenido en los vectores correspondientes, tales como calcular su valor mínimo, máximo, media, mediana, desviación típica, suma de los elementos de dicho vector, etc. `min`, `max`, `mean`, `median`, `std`, `sum`, `prod`, etc.

1.10 POLINOMIOS

Esta es una sección importante, dado que las funciones de transferencia de los sistemas se introducirán habitualmente en la forma *numerador-denominador*, los cuales serán tratados

como polinomios por MATLAB. En las demos que acompañan a estas notas se podrán analizar numerosos ejemplos.

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

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

que muy bien podría ser el denominador de una función de transferencia.

Mediante la función `roots` se pueden encontrar las raíces de esa ecuación:
`roots(p)`

De modo complementario, 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(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, como en este caso, un vector o una matriz, en lugar de un valor individual, la evaluación se hace elemento a elemento.

Podemos realizar cómodamente operaciones de multiplicación y división de polinomios mediante las funciones `conv` y `deconv`, respectivamente:

```
conv([1,2],[2,0])
```

1.11 Otras funciones de interés

En esta sección simplemente comentaremos, de forma rápida, la existencia de una serie de funciones muy útiles en problemas de integración numérica (`quad`, `quad8`), solución de ecuaciones diferenciales (`ode23`, `ode45` y muchos otros), importantes cuando se estudian los sistemas dinámicos, ecuaciones no lineales (`fmin`, `fsolve`, etc.), interpolación (`spline`, etc.)...

1.12 GRÁFICOS

MATLAB es muy potente a la hora de generar gráficos (sobre todo en sus últimas versiones), no sólo por la variedad de comandos que ofrece para ello, sino también por la versatilidad de dichos comandos. En las demostraciones aparecerán varios tipos de gráficos. De momento, comentaremos los comandos fundamentales para la realización de los mismos. En primer lugar, comandos genéricos y comandos orientados a gráficos bidimensionales:

- `figure(n)`: Las representaciones de gráficos en MATLAB se realizan en ventanas gráficas. En un momento dado puede haber varias ventanas gráficas abiertas. La función `figure` se utiliza para abrir una nueva ventana gráfica que será numerada de acuerdo con el parámetro, o bien, si ya existe una ventana con ese número, se convertirá en la ventana gráfica activa, donde se realizará la próxima representación gráfica.
- `clf`: Limpia la ventana gráfica activa.
- `close(n)`: Para cerrar una ventana gráfica. `close all` cierra todas las ventanas gráficas.
- `plot`: es la función básica de representación gráfica de datos en dos dimensiones. La representación se realiza en la ventana gráfica que esté activa en un momento dado. En caso de no haber ninguna, se crea una ventana gráfica nueva. Ejemplos de uso:
  - `plot(v)`: representa en el eje vertical los valores contenidos en el vector v, frente a los valores del índice en el eje horizontal.
  - `plot(t,v)`: representa los valores del vector v frente a los del vector t.
  - `plot(t,A)`, `plot(t,[v1,v2])`: presentará varias gráficas, puesto que cada columna de la matriz A es considerada como un vector a representar frente al vector t. En la segunda variante indicada, se consigue lo mismo mediante la agrupación de los vectores $v1$, $v2$ en una matriz.
  - `plot(t1,v1,t2,v2)`: En este caso también se obtendrán dos gráficas, pero cada una de ellas tiene un conjunto de valores diferente para el eje horizontal.
- `loglog`: representación en escala logarítmica en ambos ejes.
- `semilogx`: representación en escala semilogarítmica, el eje vertical aparecerá en escala lineal.
- `semilogy`: representación en escala semilogarítmica, el eje horizontal aparecerá en escala lineal.
- `polar`: representación de datos dados en forma polar, es decir en lugar de dar un par de vectores de componentes horizontales y verticales, se dan los vectores conteniendo el vector de ángulo y módulo.

Cuando se representan varias curvas simultáneamente en una misma ventana gráfica, se utiliza una secuencia predefinida de colores para aplicar uno diferente a cada una de ellas. Se puede

cambiar manualmente el color que por defecto tendrá una determinada curva con la adición de un parámetro: `plot(t,y,'r')`. En este ejemplo, en lugar de representarse la curva con el color por defecto (azul), aparecerá en color rojo. Para ver los códigos de colores, puede consultarse la ayuda del comando `plot`.

También pueden realizarse gráficos en tres dimensiones:

- `plot3(x,y,z)`: comando análogo a `plot` para dibujar curvas, pero en tres dimensiones.
- `mesh(x,y,Z)`: para dibujar superficies, Z debe ser una matriz con tantas filas como longitud del vector x y tantas columnas como la longitud del vector y. Los puntos que se representan son: $(x(i), y(j), Z(i,j))$.
- `contour`: representa en un plano horizontal las curvas de nivel de una superficie tridimensional.

Por otro lado, existen comandos que permiten añadir determinados complementos a estos gráficos:

- `title`: permite añadir un título a la gráfica
- `xlabel`: añadir una etiqueta al eje horizontal de la gráfica
- `ylabel`: añadir etiqueta al eje vertical
- `grid`: añadir una rejilla
- `axis`: permite modificar los límites de los ejes horizontal y vertical
- `text`: añadir un texto en una posición cualquiera de la gráfica
- `gtext`: igual que `text` pero permite seleccionar la ubicación del texto mediante el ratón.

Por otra parte, muchos de los elementos gráficos pueden manipularse como objetos que tienen una serie de propiedades asociadas. Por ejemplo:

```
handlePlot = plot(x,y);
```

con este comando estamos asignando el objeto de tipo *plot* a una variable. Podemos ver las propiedades asociadas a un objeto mediante la función `get(handlePlot)`, o bien especificar alguna de ellas: `get(handlePlot,'LineStyle')`. Cualquiera de las propiedades de un objeto pueden ser alteradas mediante la función `set(handlePlot,'Color','g')`.

Por otra parte, también se dispone de cierta capacidad de modificación de las gráficas mediante opciones de la propia ventana gráfica, en lugar de usar instrucciones desde la ventana de comandos.

1.13 PROGRAMANDO EN MATLAB

MATLAB permite a la hora de programar una serie de elementos típicos para la modificación del flujo de una secuencia de instrucciones. La sintaxis es muy parecida a la de cualquier lenguaje de programación. Todos estos operadores se pueden usar en la ventana de comandos, en línea, o en un fichero `.m`.

1.13.1 Operadores 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. Si los datos a comparar son matrices, la comparación se hace elemento a elemento, devolviendo una matriz binaria.

1.13.2 Bucles y estructuras condicionales

En esta sección se explica una serie de comandos importantes a la hora de hacer un programa en MATLAB: `for`, `while`, `if-else`.

- `for`

  La sintaxis de este comando es la siguiente:

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

  La *expresion* 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. Tanto los bucles como las estructuras condicionales se terminan con `end`.

  Presentamos un primer ejemplo en el que la variable i toma los valores 10, 9, ..., 1:

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

A continuación otro ejemplo en el que aparecen dos bucles anidados:

```
x = [0:0.1:pi]';
y = x;
for f=1:length(x)
   for c=1:length(y)
      Z(f,c) = sin(x(f)).^2 + cos(y(c)).^2;
   end
end
mesh(x,y,Z);
```

Es importante evitar en 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 y compacta.

Los siguientes ejemplos calculan logaritmos de números desde 1 a 10.000. Se hará de diferentes maneras para comparar. Se utilizan los comandos `clock` (que devuelve la hora actual) y `etime` (que devuelve el tiempo en segundos que ha transcurrido entre dos instantes) para calcular el tiempo consumido en las operaciones.

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

t1=clock; ind=[1:10000]; for i=ind, a(i)=log(i); end;...
e2=etime(clock,t1);

t1=clock; a=zeros(1,10000); ind=[1:10000];...
for i=ind, a(i)=log(i); end; e3=etime(clock,t1);

t1=clock; ind=[1:10000]; a=log(ind); e4=etime(clock,t1);

t1=clock; ind=[1:10000]; a=zeros(1,10000); a=log(ind); ...
e5=etime(clock,t1);
```

Los tiempos de computación para los diferentes métodos son:

```
86.17   86.56   2.42   0.27  0.28
```

Las causas de la disminución importante de tiempos es que en los primeros métodos, MATLAB tiene que recalcular la dimensión del vector 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 implementar 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 estable (parte real de autovalores negativa):

```
A = randn(2); % Genera numeros aleatorios con distribucion normal
while max(real(eig(A))) >= 0
   A=randn(2);
end;
eig(A)
```

  Se puede usar el comando `break` para salir de un bucle en función de una determinada condición.

- `if, else, elseif`

  La sintaxis es la siguiente:

```
if expresion 1
   hace algo
elseif expresion 2
   hace algo
else
   hace algo
end
```

  `else` y `elseif` son opcionales.

1.13.3 Ficheros .m

MATLAB puede ejecutar programas que se encuentren almacenados en ficheros ASCII que pueden encuentrarse en alguno de los subdirectorios indicados en el camino de búsqueda o bien en el subdirectorio de trabajo actual y tengan además extensión `.m`. Hay dos tipos de ficheros `.m`: *script files* y *function files*

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. Los ejemplos que ilustran estas notas son en sí *script-files*, pues llevan un conjunto de comandos MATLAB y comentarios.

funciones

Son también ficheros `.m`, pero a diferencia de los anteriores, se le pueden pasar argumentos y pueden devolver resultados. Por tanto utilizan variables globales que se pasan por valor. La mayoría de los ficheros contenidos en los *toolboxes* son funciones. La sintaxis de todas las funciones almacenadas en ficheros `.m` es la siguiente:

```
function [out1,out2,...] = nombre_fichero (in1,in2,...)
% Comentarios adicionales para el help
comandos de MATLAB
return;
```

Una función puede tener múltiples parámetros de entrada y salida. Numerosos ejemplos de funciones serán utilizados en las demostraciones.

Para finalizar, comentar que existen una serie de utilidades a la hora de programar en MATLAB. Las más comunes son:

- `pause`: Para la ejecución hasta que se pulsa una tecla. Puede usarse para pausar la ejecución durante un número de segundos determinado, en lugar de esperar a que se pulse una tecla: `pause(n)`.

- `disp`: Muestra una cadena de caracteres por pantalla.

- `input`: Muestra una cadena de caracteres por pantalla y espera a que el usuario introduzca un valor, que generalmente será asignado a una variable.

1.14 RESUMEN DE LOS COMANDOS DE MATLAB

	CARACTERES ESPECIALES
`=`	Instrucción de asignación
`[`	Usado para formar vectores y matrices
`]`	Ver [
`(`	Precedencia aritmética
`)`	Ver (
`.`	Punto decimal
`...`	La instrucción continúa en la siguiente línea
`,`	Separa índices y argumentos de función
`;`	Acaba filas, suprime la impresión
`%`	Comentarios
`:`	Indexación, generación de vectores
`!`	Ejecuta instrucción del sistema operativo

	VALORES ESPECIALES
`ans`	Respuesta cuando no se asigna la expresión
`eps`	Precisión
`pi`	π
`i,j`	$\sqrt{-1}$
`inf`	∞
`NaN`	No Número (Not-a -Number)
`clock`	Reloj
`date`	Fecha
`flops`	Número de operaciones
`nargin`	Número de argumentos de entrada de una función
`narout`	Número de argumentos de salida de una función

	ARCHIVOS DE DISCO
`chdir`	Cambiar de directorio
`delete`	Borrar archivo
`diary`	Diario de la sesión
`dir`	Directorio de archivos en el disco
`load`	Cargar variables de un archivo
`save`	Guardar variables en un archivo
`type`	Mostrar función o archivo
`what`	Mostrar archivos .m en el disco
`fprintf`	Escribir en un archivo
`pack`	Compactar memoria vía `save`

	Matrices especiales
`compan`	Compañera
`diag`	Diagonal
`eye`	Identidad
`gallery`	Esotérica
`hadamard`	Hadamard
`hankel`	Hankel
`hilb`	Hilbert
`invhilb`	Inversa de Hilbert
`linspace`	Vectores igualmente espaciados
`logspace`	Vectores logarítmicamente espaciados
`magic`	Mágica cuadrada
`meshdom`	Dominio para puntos de malla
`ones`	Matriz constante de unos
`pascal`	Pascal
`rand`	Elementos aleatorios
`toeplitz`	Toeplitz
`vander`	Vandermonde
`zeros`	Matriz de ceros

	Manipulación de matrices
`rot90`	Rotación
`fliplr`	Invierte el orden de las columnas
`flipud`	Invierte el orden de las filas
`diag`	Diagonal
`tril`	Parte triangular inferior
`triu`	Parte triangular superior
`reshape`	Reordena una matriz en otra
`'`	Traspuesta
`:`	Convierte una matriz en una columna simple

	Funciones lógicas y relacionales
`any`	Condiciones lógicas
`all`	Condiciones lógicas
`find`	Encuentra índices de valores lógicos
`isnan`	Detecta NaNs
`finite`	Detecta infinitos
`isempty`	Detecta matrices vacías
`isstr`	Detecta variables de cadena
`strcomp`	Compara variables de cadena

	CONTROL DE FLUJO
`if`	Ejecuta instrucciones condicionalmente
`elseif`	Usado con `if`
`else`	Usado con `if`
`end`	Termina `if, for, while`
`for`	Repite instrucciones un número de veces
`while`	Repite instrucciones mientras una sentencia lógica sea verdadera
`break`	Sale de los bucles `for` y `while`
`return`	Salida desde funciones
`pause`	Pausa hasta que se pulse una tecla

	TEXTO Y CADENAS
`abs`	Convierte cadena en valores **ASCII**
`eval`	Evalúa texto como instrucciones
`num2str`	Convierte números en cadenas
`int2str`	Convierte enteros en cadenas
`setstr`	Indicador de cadenas
`sprintf`	Convierte números en cadenas
`isstr`	Detecta variables de cadena
`strcomp`	Compara variables de cadena
`hex2num`	Convierte cadenas hexadecimales en números

	PROGRAMACIÓN Y ARCHIVOS **.m**
`input`	Obtiene números desde el teclado
`keyboard`	Llamada al teclado como si fuera un archivo **.m**
`error`	Muestra mensaje de error
`function`	Define función
`eval`	Evalúa texto en variables
`feval`	Evalúa función dada por una cadena
`echo`	Permite mostrar las instrucciones en pantalla
`exist`	Comprueba si las variables existen
`casesen`	Sensibilidad a las mayúsculas
`global`	Define variables globales
`startup`	Archivo de inicialización
`getenv`	Accede a una variable de entorno
`menu`	Genera un menú
`etime`	Tiempo gastado

	VENTANA ALFANUMÉRICA
`clc`	Limpia pantalla
`home`	Mueve cursor al comienzo
`format`	Establece el formato de salida
`disp`	Muestra matriz o texto
`fprintf`	Imprime número formateado
`echo`	Permite la muestra de las instrucciones

Gráficos	
`plot`	Gráfico lineal en el plano XY
`loglog`	Gráfico logarítmico en el plano XY
`semilogx`	Gráfico semilogarítmico
`semilogy`	Gráfico semilogarítmico
`polar`	Gráfico polar
`mesh`	Superficie de malla tridimensional
`contour`	Plano de contornos
`meshdom`	Dominio para gráficos de superficie
`bar`	Gráficos de barras
`stairs`	Gráficos de escaleras
`errorbar`	Añade barras de errores

Anotación Gráfica	
`title`	Título
`xlabel`	Anotación en eje x
`ylabel`	Anotación en eje y
`grid`	Dibuja cuadriculado
`text`	Posiciona un texto arbitrariamente
`gtext`	Posiciona un texto con el ratón
`ginput`	`input` gráfico

Control de la ventana gráfica	
`axis`	Escalado manual de ejes
`hold`	Mantiene gráfico en pantalla
`shg`	Muestra la pantalla gráfica
`clf`	Limpia la pantalla gráfica
`subplot`	Divide la pantalla gráfica

Funciones elementales	
`abs`	Módulo complejo
`angle`	Argumento complejo
`sqrt`	Raíz cuadrada
`real`	Parte real
`imag`	Parte imaginaria
`conj`	Conjugado complejo
`round`	Redondeo al entero más cercano
`fix`	Redondeo hacia cero
`floor`	Redondeo hacia $-\infty$
`ceil`	Redondeo hacia ∞
`sign`	Función signo
`rem`	Resto
`exp`	Exponencial base e
`log`	Logaritmo natural
`log10`	Logaritmo base 10

	FUNCIONES TRIGONOMÉTRICAS
`sin`	Seno
`cos`	Coseno
`tan`	Tangente
`asin`	Arcoseno
`acos`	Arcocoseno
`atan`	Arcotangente
`atan2`	Arcotangente de x/y
`sinh`	Seno hiperbólico
`cosh`	Coseno hiperbólico
`tanh`	Tangente hiperbólica
`asinh`	Arcoseno hiperbólico
`acosh`	Arcocoseno hiperbólico
`atanh`	Arcotangente hiperbólica

	FUNCIONES ESPECIALES
`bessel`	Función de Bessel
`gamma`	Función gamma
`rat`	Aproximación racional
`erf`	Función de error
`inverf`	Inversa de la función de error
`ellipk`	Integral completa elíptica de primera especie
`ellipj`	Integral elíptica de Jacobi

	DESCOMPOSICIONES Y FACTORIZACIONES
`balance`	Forma equilibrada
`backsub`	Sustitución regresiva
`cdf2rdf`	Convierte diagonales complejas en diagonales reales
`chol`	Factorización de Cholesky
`eig`	Autovalores y autovectores
`hess`	Forma de Hessenberg
`inv`	Inversa
`lu`	Factores de la eliminación gaussiana
`nnls`	Mínimos cuadrados con restricciones
`null`	Base ortonormal del núcleo
`orth`	Base ortonormal de la imagen
`pinv`	Pseudoinversa
`qr`	Factorización QR
`qz`	Algoritmo QZ
`rref`	Forma escalonada reducida por filas
`schur`	Descomposición de Schur
`svd`	Descomposición en valores singulares

	Condicionamiento de matrices
`cond`	Número de condición en la norma 2
`norm`	Norma 1, norma 2, norma de Frobenius, norma ∞
`rank`	Rango
`rcond`	Estimación de la condición (inverso)

	Funciones matriciales elementales
`expm`	Matriz exponencial
`logm`	Matriz logaritmo
`sqrtm`	Matriz raíz cuadrada
`funm`	Función arbitraria de matriz
`poly`	Polinomio característico
`det`	Determinante
`trace`	Traza
`kron`	Producto tensorial de Kronecker

	Polinomios
`poly`	Polinomio característico
`roots`	Raíces de polinomios - método de la matriz compañera
`roots1`	Raíces de polinomios - método de Laguerre
`polyval`	Evaluación de polinomios
`polyvalm`	Evaluación de polinomio matricial
`conv`	Multiplicación
`deconv`	División
`residue`	Desarrollo en fracciones parciales
`polyfit`	Ajuste por un polinomio

	Análisis de datos por columnas
`max`	Valor máximo
`min`	Valor mínimo
`mean`	Valor medio
`median`	Mediana
`std`	Desviación típica
`sort`	Ordenación
`sum`	Suma de elementos
`prod`	Producto de elementos
`cumsum`	Suma acumulativa de elementos
`cumprod`	Producto acumulativo de elementos
`diff`	Derivadas aproximadas
`hist`	Histogramas
`corrcoef`	Coeficientes de correlación
`cov`	Matriz de covarianza
`cplxpair`	Reordena en pares complejos

TRATAMIENTO DE SEÑALES	
`abs`	Módulo complejo
`angle`	Argumento complejo
`conv`	Convolución
`corrcoef`	Coeficientes de correlación
`cov`	Covarianza
`deconv`	Deconvolución
`fft`	Transformada rápida de Fourier
`fft2`	FFT 2-dimensional
`ifft`	FFT inversa
`ifft2`	FFT inversa 2-dimensional
`fftshift`	Cambia las dos mitades de un vector

INTEGRACIÓN NUMÉRICA	
`quad`	Función de integración numérica
`quad8`	Función de integración numérica

SOLUCIÓN DE ECUACIONES DIFERENCIALES	
`ode23`	Método Runge-Kutta de orden 2/3
`ode45`	Método Runge-Kutta-Fehlberg de orden 4/5

ECUACIONES NO LINEALES Y OPTIMIZACIÓN	
`fmin`	Mínimo de una función de una variable
`fmins`	Mínimo de una función de varias variables
`fsolve`	Solución de un sistema de ecuaciones no lineales (ceros de una función de varias variables)
`fzero`	Cero de una función de una variable

INTERPOLACIÓN	
`spline`	Spline cúbico
`table1`	Genera tablas 1-D
`table2`	Genera tablas 2-D

Capítulo 2

ANÁLISIS Y CONTROL DE SISTEMAS USANDO MATLAB

2.1 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*. Las funciones principales se van a explicar sobre ejemplos demostrativos, con el fin de que su uso y comprensión sean lo más sencillos posible. Se realizarán ejemplos tanto en el dominio temporal, continuo y discreto, como en el frecuencial. También se mostrará la forma de dar la descripción de un sistema lineal mediante función de transferencia, conjunto de polos y ceros, o variables de estado. Asimismo se comentará la forma de manipular una función de transferencia como un objeto.

2.2 Tratamiento mediante funciones de transferencia. Sistemas continuos

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:

$$H(s) = \frac{.2s^2 + .3s + 1}{(s^2 + .4s + 1)(s + .5)}$$

En MATLAB las funciones de transferencia se introducen dando el par de polinomios 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)
```

Para ver los polos (o los ceros) de la función de transferencia, podemos usar: `roots(den)` (`roots(num)`). Una forma más completa de convertir una función de transferencia dada por dos polinomios numerador y denominador, en un conjunto de factores de grado 1, correspondientes a los polos (z_1, z_2, z_3) y ceros (c_1, c_2), de la forma:

$$H(s) = \frac{K\,(1-\frac{s}{c_1})\,(1-\frac{s}{c_2})}{(1-\frac{s}{z_1})\,(1-\frac{s}{z_2})\,(1-\frac{s}{z_3})}$$

es mediante el comando `tf2zp`:

```
[ceros,polos,gan] = tf2zp (N,D);
```

que devuelve un vector conteniendo los ceros de la función de transferencia, un vector conteniendo los polos, y un escalar correspondiente a la ganancia estática. La función complementaria a ésta también existe:

```
[N,D] = zp2tf (ceros,polos,gan);
```

Como curiosidad, cabe mencionar que el nombre de estas funciones es bastante descriptivo: "tf-two-zp" procede de *transfer-function to zero-pole form.*

2.2.1 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)');
grid;
```

La respuesta al escalón unitario puede verse en la Fig. 2.1. Seleccionando con el ratón en la curva, pueden modificarse algunos atributos de la misma. También pueden modificarse las propiedades de los ejes de forma análoga. Las figuras guardarse en ficheros *.fig* propios de MATLAB, o exportarse en diferentes formatos gráficos estándar.

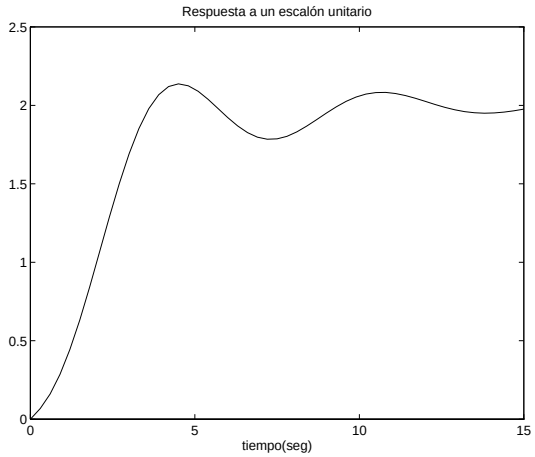

Figura 2.1: Respuesta a escalón unitario

No es necesario recuperar el resultado de la simulación ante escalón que realiza `step` para después representarlo con `plot`, sino que el propio comando `step`, si lo utilizamos sin parámetros de salida, realiza la representación. Es más, en este segundo caso, la representación gráfica es algo más interactiva, puesto que si "pinchamos" con el botón izquierdo en algún punto de la gráfica, nos dice los valores correspondientes al tiempo y al valor de la salida en ese punto.

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.

```
impulse (num,den,t);
```

La respuesta al impulso puede verse en la Fig. 2.2.

La respuesta del sistema a cualquier tipo de entrada también puede obtenerse. Para ello es necesario tener la señal de entrada en un vector `u`, que lógicamente deberá tener la misma dimensión que el vector de tiempos. 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:

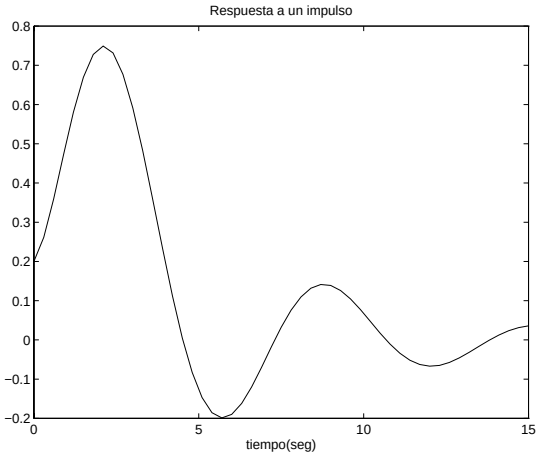

Figura 2.2: Respuesta al impulso

```
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. 2.3.

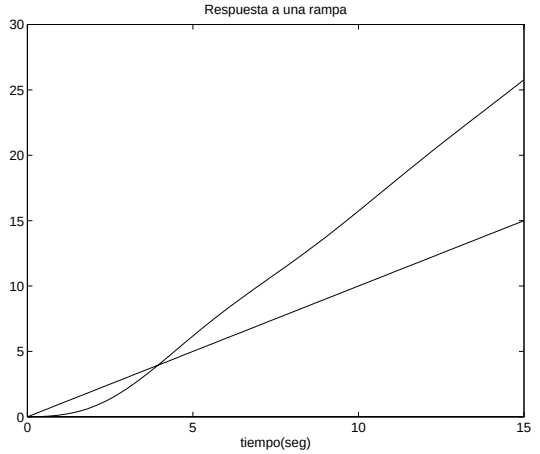

Figura 2.3: Respuesta a la rampa

Otro ejemplo característico es la respuesta a un ruido uniforme aleatorio. Recordamos que `rand(m,n)` genera una matriz $m \times n$ 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 paso bajas).

```
noise = rand (size(t));
```

```
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. 2.4.

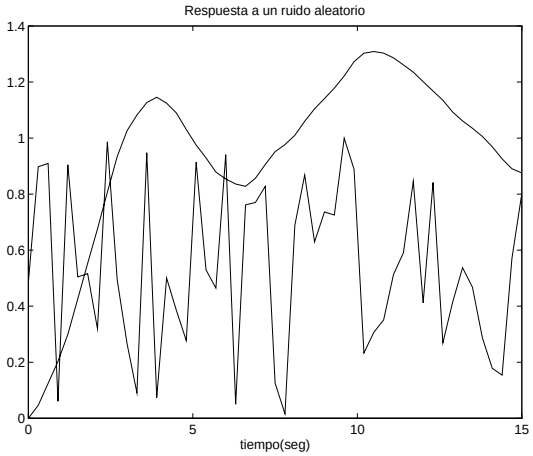

Figura 2.4: Respuesta a ruido aleatorio

2.2.2 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 las gráficas por sí solas. En caso contrario, vuelcan los datos en los vectores de salida oportunos. A continuación se presentan ejemplos de las tres posibles sintaxis de la función `bode`:

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 decibelios (dB) y la fase en grados. En las otras dos la magnitud se devuelve en el vector *mag* y está expresada en unidades absolutas, no en dB. Por su parte, la fase, devuelta en el vector *phase*, sigue siendo en grados. En esta forma, la representación es más interactiva, en el sentido de que "pinchando" con

el ratón en un punto de la curva, podemos ver los valores correspondientes. La segunda forma automáticamente genera los puntos de frecuencia en el vector w. En la tercera forma es el usuario el que escoge los rangos de frecuencia, y resulta muy adecuado cuando se quieren representar varias gráficas conjuntamente, que habrán de compartir una misma escala frecuencial.

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

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

El resultado para la función de transferencia del ejemplo anterior puede verse en la Fig. 2.5. Cabe comentar que el comando `subplot(n,m,i)` permite dividir una ventana gráfica en una matriz de $n \times m$ sub-gráficas, seleccionando como activa la número i (numeradas consecutivamente de izquierda a derecha y de arriba abajo).

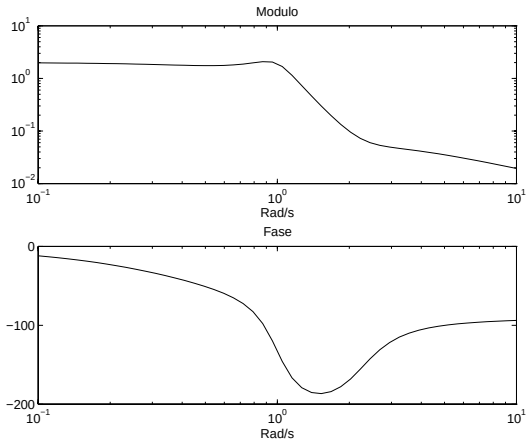

Figura 2.5: Diagrama de Bode

El comando `nyquist` tiene la misma sintaxis:

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

Computa las partes real e imaginaria de $G(jw)$ y realiza la representación si no se le ponen parámetros de salida. Para obtener la representación gráfica por nosotros mismos, sólo hay que dibujar la parte real frente a la imaginaria. El resultado obtenido mediante el ejemplo anterior puede verse en la Fig. 2.6.

El comando `nichols` computa 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

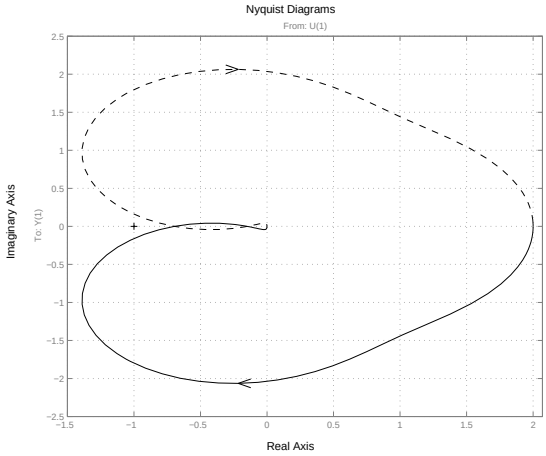

Figura 2.6: Diagrama de Nyquist

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`.

```
nichols (num,den,w),  ngrid;
[mag,phase] = nichols (num,den,w);
```

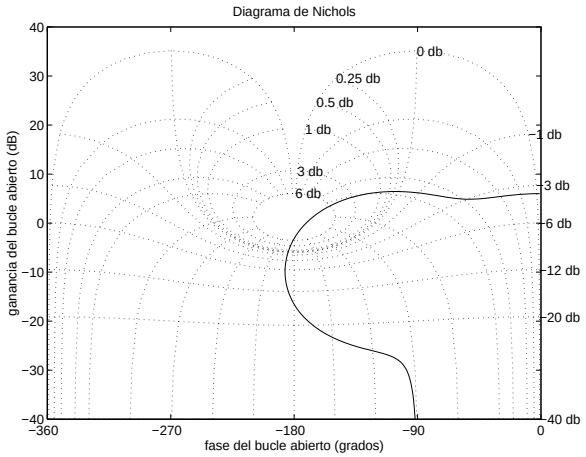

Figura 2.7: Diagrama de Nichols

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`.

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

```
[mg,mf,wmg,wmf] = margin (num,den);
```

Como tercer parámetro de entrada se le puede pasar el rango de frecuencias deseado. En el primer caso indicado, en el que no se le especifican parámetros de salida, se realiza la representación del diagrama de Bode, junto con una indicación, mediante líneas verticales de los puntos donde se mide cada uno de los márgenes y los valores de los mismos. En la segunda variante, como salidas se obtienen los valores de los márgenes de ganancia (no en *dB*), el margen de fase (en grados) y sus correspondientes frecuencias. Si existen varias frecuencias de corte marca los más desfavorables (ver Fig. 2.8).

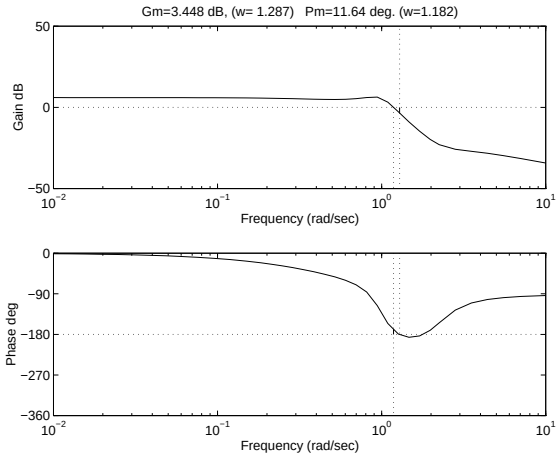

Figura 2.8: Márgenes de fase y ganancia

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^{-sT}. La forma más sencilla de manipular los retardos en MATLAB es en el dominio de la frecuencia. Nótese que $e^{-jwT} = 1^{\underline{|-wT}}$. Por tanto los retardos dejan la magnitud invariable y afectan al desfase, tendiendo a inestabilizar al sistema controlado. Para propósitos de representación mediante el comando `bode`, todo lo que habrá que hacer es restar la fase del retardo a la de la función de transferencia. Por ejemplo:

```
num = [0.2 0.3 1];
den = [1 0.9 1.2 0.5];
T = 1;  % Tiempo de retardo puro
w = logspace (-2,1,100)';
[mag,fase] = bode (num,den,w);
faseDelay = fase - (T*w*180/pi);  % Sustrae la fase tras convertirla a grados
subplot(211);  semilogx (w, 20*log10(mag));  grid;
subplot(212);  semilogx (w, [fase, faseDelay]);  grid;
```

2.2.3 Comandos relacionados con operaciones de bloques

Existen una serie de comandos relacionados con las operaciones típicas en diagramas de bloques:

– `[N12,D12] = series (N1,D1,N2,D2)`: Devuelve la resultante de colocar en serie dos funciones de transferencia (Fig. 2.9). El mismo resultado podría obtenerse llamando dos veces al comando `conv`, que recuérdese permitía multiplicar dos polinomios.

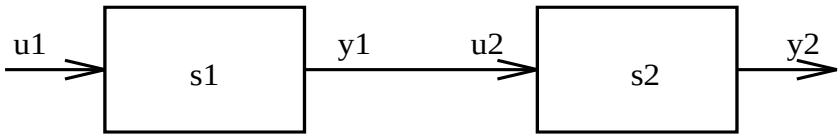

Figura 2.9: Conexión en serie

– `[N12,D12] = parallel (N1,D1,N2,D2)`: Devuelve la resultante de colocar en paralelo dos funciones de transferencia (Fig. 2.10).

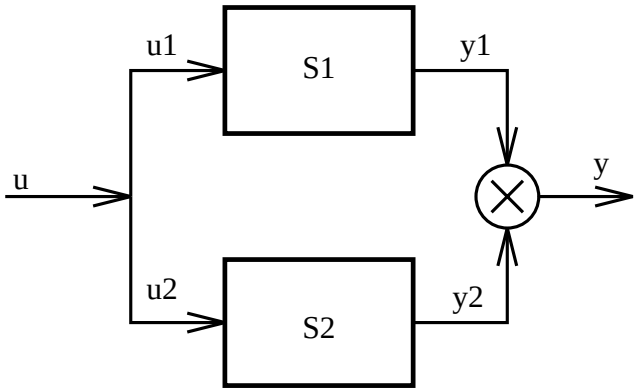

Figura 2.10: Conexión en paralelo

– `[Nbc,Dbc] = feedback (N1,D1,N2,D2,-1)`: A partir de un sistema en bucle abierto, dado por el numerador y denominador $N1$,$D1$, proporciona el correspondiente en bucle cerrado, considerando que en la cadena de realimentación hay otra función de transferencia, dada por $N2$, $D2$ (Fig. 2.11). El último parámetro indica el signo de la realimentación (-1 para realimentación negativa y 1 para positiva).

– `[Nbc,Dbc] = cloop (N1,D1,-1)`: En el caso en que se pretenda obtener la función de transferencia en bucle cerrado con realimentación unitaria, puede emplearse este comando más compacto, en el que se evita tener que especificar una segunda función de transferencia.

Conviene tener claro que para todos estos comandos relacionados con operaciones por bloques, se podría perfectamente estar trabajando con funciones de transferencia discretas, sin ninguna diferencia.

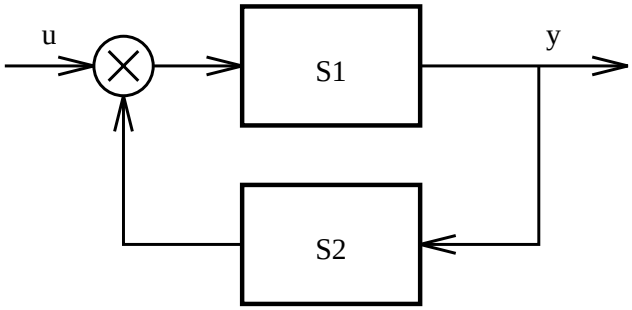

Figura 2.11: Conexión en realimentación

Para operaciones de bloques más complejas, resulta más adecuado usar la herramienta SIMULINK que también se explicará en estas notas.

2.2.4 Lugar de las raíces

El análisis mediante el lugar de las raíces se puede obtener definiendo un vector de ganancias deseadas (que es el parámetro usado habitualmente para ver la evolución de los polos en bucle cerrado). La sintaxis es:

```
r = rlocus (N,D,K);
rlocus (N,D);
```

En la primera forma, calcula el lugar de las raíces de $1+K\frac{N(s)}{D(s)}=0$, para un vector de ganancias especificado, K. `rlocus` devuelve una matriz r con `length(K)` filas y `length(den)` columnas, conteniendo la localización de las raíces complejas. Cada fila de la matriz corresponde a una ganancia del vector K. El lugar de las raíces puede ser dibujado con `plot(r,'x')`.

En la segunda forma, que es la usada habitualmente, la función directamente dibuja el lugar de las raíces. Además, como vemos, no es imprescindible indicar un vector de ganancias. Para la función de transferencia que veníamos utilizando en los ejemplos, se obtendría el lugar de las raíces mostrado en la Fig. 2.12

Un comando muy útil como complemento a `rlocus` es `rlocfind`, cuya sintaxis general es:

```
[K,polos] = rlocfind (num,den)
```

Antes de introducir dicho comando es necesario haber dibujado el lugar de las raíces. Al introducir `rlocfind`, se pide que se seleccione con el ratón un punto determinado del lugar,

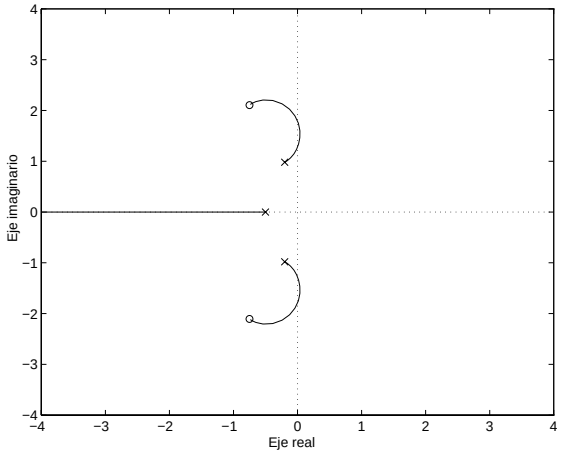

Figura 2.12: Lugar de las raíces

proporcionando como resultado la ganancia K en dicho punto y la localización de los polos correspondientes a esa ganancia.

Para mejorar la precisión, puede realizarse un *zoom* en torno a una zona de interés en la gráfica, antes de ejecutar `rlocfin`. Otra utilidad para ver la sobreoscilación que correspondería a un par de polos complejos conjugados situados en el lugar, sería dibujar los lugares geométricos de factor de amortiguamiento (δ) y frecuencia natural (ω_n) constantes, mediante el comando `sgrid`.

Ejemplo: Los siguientes comandos dibujan el lugar de las raíces de una función de transferencia, realizando una conveniente ampliación y resaltando las líneas de factor de amortiguamiento 0.5, 0.6 , 0.7 y de frecuencia natural $0.5\,rad/s$ (Fig. 2.13).

```
N = 1;
D = [1 3 2 0];
rlocus(N,D);
sgrid ([0.5:0.1:0.7],0.5);
axis([-2.5,1,-3,3]);
```

Las nuevas versiones de MATLAB van más allá y ofrecen una herramienta interactiva para ir viendo en vivo las variaciones que sufre el lugar de las raíces, conforme se añaden, eliminan y mueven los polos y ceros de bucle abierto. A esta herramienta se accede mediante el comando `rltool`. Merece la pena que el alumno dedique unos minutos a esta herramienta de gran valor didáctico.

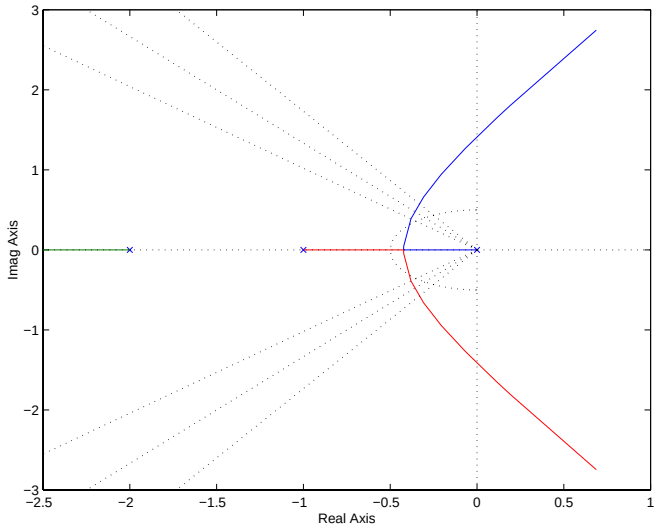

Figura 2.13: Lugar de las raíces con rejilla

2.3 ESTUDIO TEMPORAL Y FRECUENCIAL DE SISTEMAS DE PRIMER Y SEGUNDO ORDEN

2.3.1 Sistemas de primer orden

La representación en forma de función de transferencia viene dada por:

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

que en notación MATLAB se introduce:

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

La respuesta a un escalón unitario de entrada se obtiene con la función `step`. El resultado puede verse en la Fig. 2.14

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

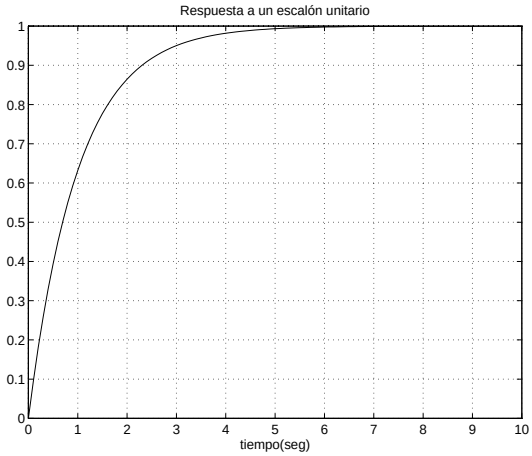

Figura 2.14: Respuesta a escalón unitario del sistema de primer orden

Las dos características fundamentales de un sistema de primer orden son su ganancia estática K y su constante de tiempo τ. 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 vector de datos resultante:

```
yRP = ye(length(ye));  % Valor en regimen permanente
n = 1;
while ye(n) < 0.63*yRP
    n=n+1;
end
% Constante de tiempo (0.1 es el intervalo transcurrido entre dos medidas,
% se le resta 1, porque los indices empiezan en 1):
tauEstim = 0.1*(n-1);
```

La respuesta a una rampa unitaria de entrada para nuestro sistema de primer orden se puede simular mediante:

```
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 Fig. 2.15, en la que aparecen, tanto la rampa de entrada como la salida. 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 vector de datos resultante.

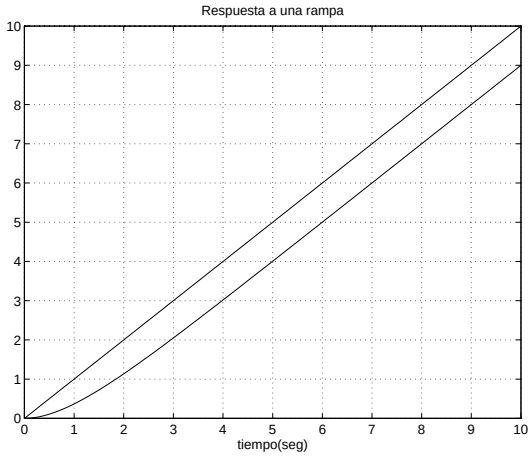

Figura 2.15: Respuesta a rampa unitaria del sistema de primer orden

La respuesta impulsional se puede obtener del mismo modo, pero usando en este caso la función `impulse` (Fig. 2.16), que tiene una sintaxis similar al comando `step`.

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

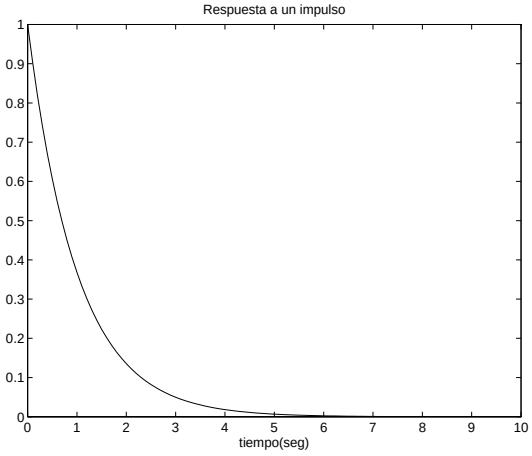

Figura 2.16: Respuesta a impulso del sistema de primer orden

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: $y_{e2}(t) = 1 - e^{(-t/\tau)},\ (t \geq 0)$

Respuesta a rampa unitaria: $y_{r2}(t) = t - \tau + \tau e^{(-t/\tau)},\ (t \geq 0)$

Respuesta a impulso: $y_{i2}(t) = (1/\tau)e^{(-t/\tau)},\ (t \geq 0)$

Si se dibujan estas funciones con el vector de tiempos definido anteriormente, y se superponen con las gráficas vistas anteriormente, puede apreciarse que coinciden perfectamente (Fig. 2.17).

```
ye2 = 1 - exp(-t/tau);
yr2 = t - tau + tau * exp(-t/tau);
yi2 = (1/tau) * exp(-t/tau);
plot (t,ye,t,ye2,'o');
title('Respuesta teorica a escalon unitario');
grid;
pause;
plot (t,yr,t,ramp,t,yr2,'o');
title('Respuesta teorica a rampa unitaria');
grid;
pause;
plot (t,yi,t,yi2,'o');
title ('Respuesta teorica a impulso');
grid;
```

2.3.2 Sistemas de segundo orden

La representación normal de un sistema de segundo orden en forma de función de transferencia viene dada por:

$$G(s) = \frac{Kw_n^2}{s^2 + 2\delta w_n s + w_n^2}$$

donde:

K: ganancia estática del sistema. Se va a suponer en el análisis siguiente, sin pérdida de generalidad, que $K = 1$.

δ: Coeficiente de amortiguamiento.

w_n: Frecuencia natural no amortiguada del sistema.

Del polinomio característico se tiene que las dos raíces son $s_{1,2} = -\delta w_n \pm w_n\sqrt{\delta^2 - 1}$, pudiendo distinguirse los siguientes casos:

Caso 1: Si $\delta > 1 \rightarrow$ 2 raíces reales distintas en SPI (sobreamortiguado).

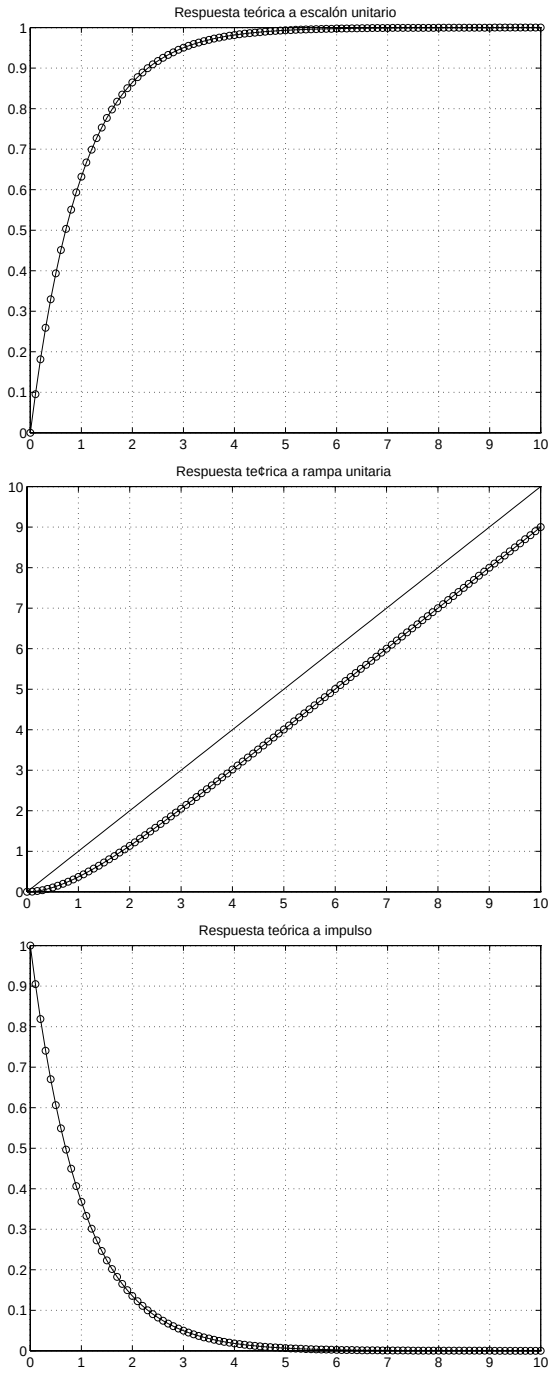

Figura 2.17: Respuesta a escalón, rampa e impulso del sistema de primer orden

Caso 2: Si $\delta = 1 \rightarrow$ 2 raíces reales iguales en SPI (límite sobre-sub), sistema críticamente amortiguado.

Caso 3: Si $0 < \delta < 1 \rightarrow$ raíces complejas conjugadas en SPI (subamortiguado)

Caso 4: Si $\delta = 0 \rightarrow$ Respuesta oscilatoria. Sistema críticamente estable. Raíces en eje imaginario.

Caso 5: Si $\delta < 0 \rightarrow$ Sistema inestable, raíces en SPD.

Se va a analizar el comportamiento para el conjunto de valores de δ en la respuesta a un escalón unitario. Se supone $w_n = 1$. Dado que el caso 3 se verá con más detalle, dado su mayor interés, se presentará en último lugar.

Primer caso: 2 raíces reales distintas (Fig. 2.18).

```
t = [0:0.2:20]';
wn = 1;
d = 2;
num = [wn^2];
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
plot (t,ye);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

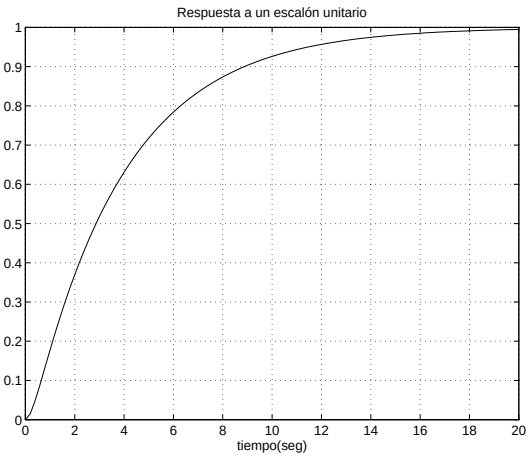

Figura 2.18: Respuesta a escalón del sistema de segundo orden con dos raíces reales distintas

Segundo caso: 2 raíces reales iguales (críticamente amortiguado) (Fig. 2.19). El sistema, en este caso el lo más rápido posible, antes de hacerse subamortiguado.

```
d = 1;
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
plot (t,ye);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

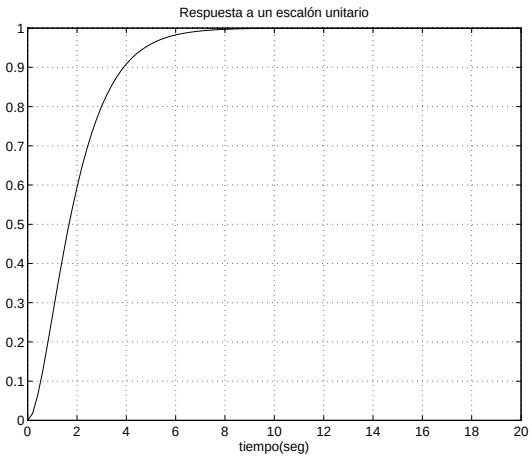

Figura 2.19: Respuesta a escalón de un sistema segundo orden con dos raíces reales iguales

Cuarto caso: Sistema en punto crítico de oscilación (Fig. 2.20).

```
d = 0;
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
plot (t,ye);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

Quinto caso: Sistema inestable (Fig. 2.21).

```
d = -0.1;
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
plot(t,ye);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

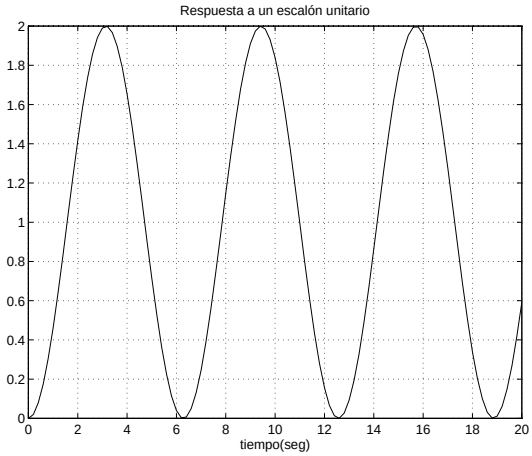

Figura 2.20: Respuesta a escalón de un sistema de segundo orden críticamente estable

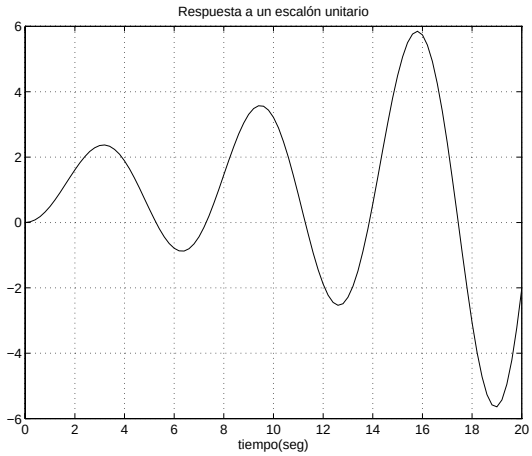

Figura 2.21: Respuesta a escalón de un sistema de segundo orden inestable

Tercer caso: Dos raíces complejas conjugadas (Fig. 2.22).

```
d = 0.5;
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
plot (t,ye);
title ('Respuesta a un escalo'n unitario');
xlabel ('tiempo(seg)');
grid;
```

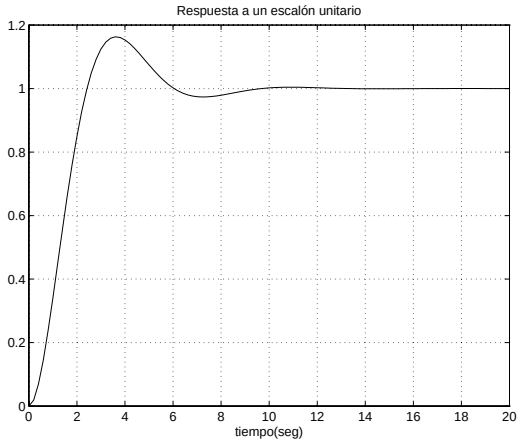

Figura 2.22: Respuesta a escalón de un sistema segundo orden subamortiguado

Se puede también analizar para este tercer caso de dos raíces complejas el efecto de modificar el factor de amortiguamiento. Se muestra para valores $\delta = \{0.1, 0.2, ...0.9\}$ (Fig. 2.23).

```
t = [0:0.2:20]';
wn = 1;
vectDelta = [0.1:0.1:0.9];
num = wn^2;
Y = [];
for ind = 1:length(vectDelta)
   d = vectDelta(ind);
   den = [1,2*d*wn,wn^2];
   y = step (num,den,t);
   Y = [Y, y];
end
plot (t,Y);
title ('Respuesta a un escalon unitario');
xlabel ('tiempo(seg)');
grid;
```

Se mantiene el valor de $\delta = 0.2$ para el siguiente análisis:

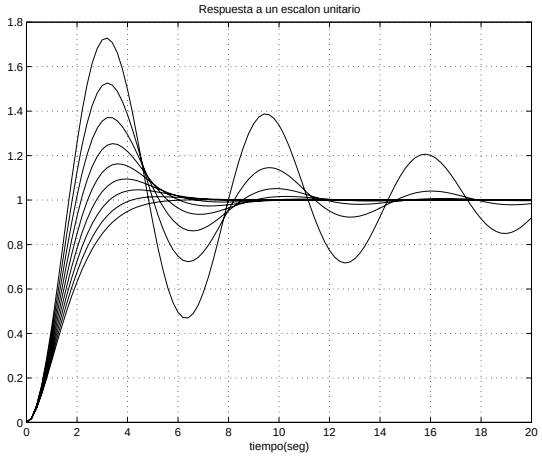

Figura 2.23: Familia de respuestas escalón de un sistema segundo orden subamortiguado

```
d = 0.2;
den = [1,2*d*wn,wn^2];
ye = step (num,den,t);
```

La salida del sistema viene dada por la ecuación:

$$y_e(t) = 1 - \frac{e^{-(\delta w_n t)}}{\sqrt{(1-\delta^2)}} sen(w_d t + \phi) \quad , \qquad \text{con :} \quad \phi = \text{arctg}\frac{\sqrt{(1-\delta^2)}}{\delta}$$

y donde $w_d = w_n\sqrt{(1-\delta^2)}$, siendo w_d la frecuencia natural amortiguada.

Las envolventes de la respuesta anterior vendrán dadas por:

$$1 + \frac{e^{-(\delta w_n t)}}{\sqrt{(1-\delta^2)}} \qquad y \qquad 1 - \frac{e^{-(\delta w_n t)}}{\sqrt{(1-\delta^2)}}$$

```
ev1 = 1 + ((exp(-d*wn*t)/(sqrt(1-d^2))));
ev2 = 1 - ((exp(-d*wn*t)/(sqrt(1-d^2))));
plot (t,ye,t,ev1,t,ev2);
title ('Respuesta de sist. segundo orden')
xlabel ('tiempo (s)');
ylabel ('salida');
grid;
```

Los parámetros característicos del transitorio vienen dados por:

Tiempo de pico: $t_p = \frac{\pi}{w_d} = \frac{\pi}{w_n\sqrt{(1-\delta^2)}}$

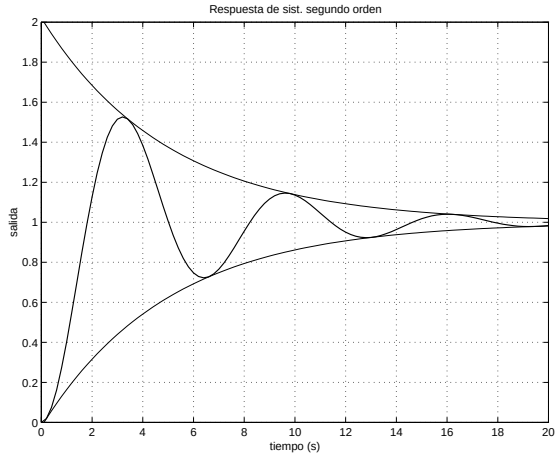

Figura 2.24: Envolventes de la respuesta de un sistema de segundo orden subamortiguado

Tiempo de subida: $t_s = \frac{\pi - \phi}{w_d}$

Sobreoscilación (%): $S0 = e^{-\frac{\delta\pi}{\sqrt{(1-\delta^2)}}}$

Estos valores se pueden calcular analíticamente y compararlos con los obtenidos directamente de la gráfica o analizando el vector de resultados.

```
% Valor de la salida en reg. perm:
yeRP = 1;
% Calculo SO teorica:
SOana = exp (-(d*pi)/(sqrt(1-d^2)));
% A partir de la respuesta obtenida:
SO = max(ye) - yeRP;

% Tiempo de pico teorico:
Tpana = pi/(Wn*sqrt(1-d^2));
% A partir de la respuesta. Aunque existen muchas formas de calcularlo,
% una trivial es la siguiente (la precision viene determinada por la
% primera cifra decimal, debido al modo de definir el vector de tiempos):
for i = 1:length(ye)
   if ye(i) == max(ye)
      Tp = t(i);
      break;
   end
end

% Tiempo de subida teorico:
fi = atan (sqrt(1-d^2)/d);
Tsana = (pi-fi) / (wn*sqrt(1-d^2));
```

```
% A partir de la respuesta. Por definicion, el tiempo de subida es aquel
% para el cual la salida iguala por primera vez al valor en reg. perm.
for i = 1:length(t)
   if (ye(i) <= yeRP  &  ye(i+1) >= yeRP)
      Ts = t(i);
      break;
   end
end

disp 'Sobreoscilaciones'; [SOana SO]
disp 'Tiempos de pico';   [Tpana Tp]
disp 'Tiempos de subida'; [Tsana Ts]
```

2.3.3 Análisis del efecto de un cero en la respuesta temporal de un sistema de segundo orden

Los ceros afectan al valor de la ganancia y a la forma de respuesta transitoria, pero no a la estabilidad. Supongamos la función de transferencia siguiente:

$$G(s) = \frac{K\frac{w_n^2}{z}(s+z)}{s^2 + 2\delta w_n s + w_n^2}$$

Según la posición de los polos y los ceros se tendrán comportamientos diferentes.

Si el valor absoluto del cero es mucho mayor que el valor absoluto de la parte real de los polos, apenas varía la forma de la respuesta típica (polos dominantes). Ver Fig. 2.25.

```
t = [0:0.2:20]';
K = 1;
wn = 1;
d = 0.5;
c = 10;
% Numeradores sin cero y con cero:
num  = K * wn^2;
num2 = K * (wn^2/c) * [1 c];
den  = [1 2*d*wn wn^2];
% Comparamos la salida con la correspondiente al mismo sistema sin cero:
y  = step (num,den,t);
y2 = step (num2,den,t);
plot (t,y2,t,y,'o');
title ('Influencia de un cero lejos del eje imaginario');
polos = roots(den);
disp 'Magnitud de la parte real polos:';
abs(real(polos(1)))
```

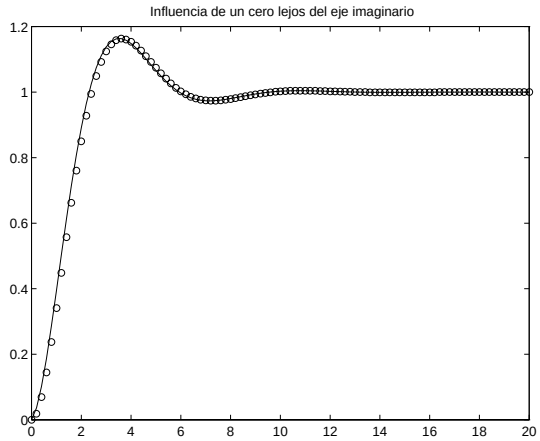

Figura 2.25: Influencia de un cero lejano en la respuesta temporal de un sistema de segundo orden

Si el cero está entre dos raíces reales no varía la forma de la respuesta, pero varía su rapidez (Fig. 2.26), pues se introduce acción derivativa.

```
t = [0:0.2:20]';
K=1;  wn=1;  d=2;  c=0.5;
num  = K*wn^2;
num2 = K*(wn^2/c)*[1 c];
den  = [1 2*d*wn wn^2];

y  = step (num,den,t);
y2 = step (num2,den,t);
plot (t,y2,t,y,'o');
title('Influencia del cero entre polos reales');
disp 'Magnitud de los polos:';
abs(roots(den))
```

Si el cero está más cerca del eje imaginario que los polos reales, aumenta la rapidez de la respuesta (Fig. 2.27), pudiendo la salida rebasar ampliamente su valor de régimen permanente. Como el efecto derivativo es muy grande, aunque el sistema sin el cero no sobrepasara el valor de régimen permanente, el efecto del cero hace que sí sobrepase dicho valor.

```
t = [0:0.2:20]';
K=1;  wn=1;  d=2;  c=0.05;
num  = K*wn^2;
num2 = K*(wn^2/c)*[1 c];
den  = [1 2*d*wn wn^2];

y  = step (num,den,t);
```

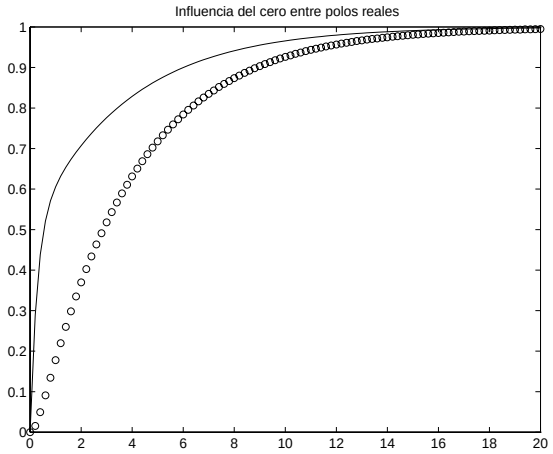

Figura 2.26: Influencia de un cero "dominante" en la respuesta temporal de un sistema de segundo orden

```
y2 = step (num2,den,t);
plot (t,y2,t,y,'o');
title('Influencia del cero cercano al eje imag.');
disp 'Magnitud de los polos:';
abs(roots(den))
```

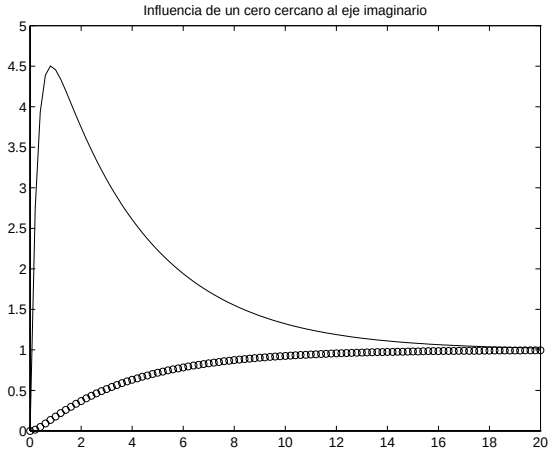

Figura 2.27: Influencia de un cero "muy dominante" en la respuesta temporal de un sistema de segundo orden

Si el cero pasa al semiplano derecho (sistema de fase no mínima) modifica sensiblemente la oscilación (típico por ejemplo en arranque de turbinas). Ver Fig. 2.28. En este caso, la acción derivativa inicial va en sentido contrario a la salida del sistema, por lo que si dicha acción derivativa es grande (cero cercano al eje imaginario), la salida irá inicialmente en sentido contrario a la de régimen permanente.

```
t = [0:0.2:20]';
```

```
K=1;  wn=1;  d=2;  c=-0.5;
num  = K*wn^2;
num2 = K*(wn^2/c)*[1 c];
den  = [1 2*d*wn wn^2];

y  = step (num,den,t);
y2 = step (num2,den,t);
plot (t,y2,t,y,'o');
title('Influencia de un cero positivo');
```

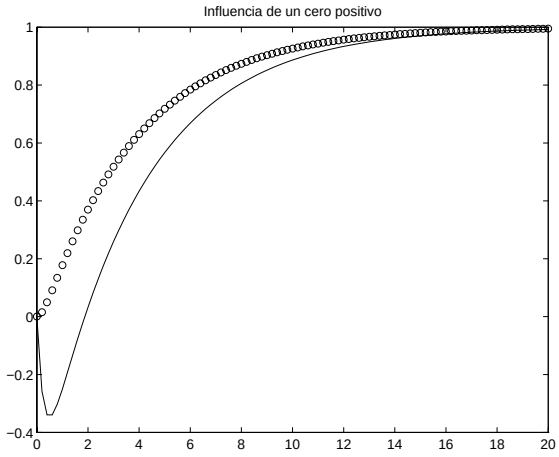

Figura 2.28: Influencia de un cero en el semiplano derecho en la respuesta temporal de un sistema de segundo orden

2.3.4 Influencia de polos adicionales. Polos dominantes

Al régimen transitorio afectan fundamentalmente los polos y ceros cercanos al eje imaginario. El efecto de un cero adicional se ha visto anteriormente. Analizamos ahora el de un polo adicional. Supongamos la función de transferencia siguiente:

$$G(s) = \frac{p\,K\,w_n^2}{(s^2 + 2\delta w_n s + w_n^2)(s+p)}$$

Se va a tomar para el estudio el caso de comportamiento subamortiguado analizado en las secciones anteriores. En dicho caso, la función de transferencia correspondiente tenía los polos con su parte real en -0.5. Si la distancia entre esos polos y el resto es del orden de $5w_n$ (con δ cercano a 1) todos los polos y ceros a partir de esa distancia no afectan al transitorio. De hecho, en el análisis de sistemas, normalmente cuando un sistema tiene un número grande de polos, se trata de simplificar para obtener un sistema aproximado de segundo orden con sus polos localizados en los polos dominantes citados. La influencia de un polo lejano y uno cercano puede verse en las Fig. 2.29 y 2.30. En ambas figuras se muestra la salida del sistema con y sin el polo p adicional comparadas ante una entrada en escalón.

```
t = [0:0.2:20]';
K=1;  wn=1;  d=0.5;  p=10;
num  = K*wn^2;
num2 = wn^2*K*p;
den  = [1 2*d*wn wn^2];
den2 = conv (den,[1 p]);
y  = step (num,den,t);
y2 = step (num2,den2,t);
plot (t,y2,t,y,'o');
title ('Influencia de un polo lejano');
disp 'Situacion de los polos:';
roots(den2)
```

Como se observa en la Fig. 2.29 la influencia es muy pequeña o casi inexistente en este caso.

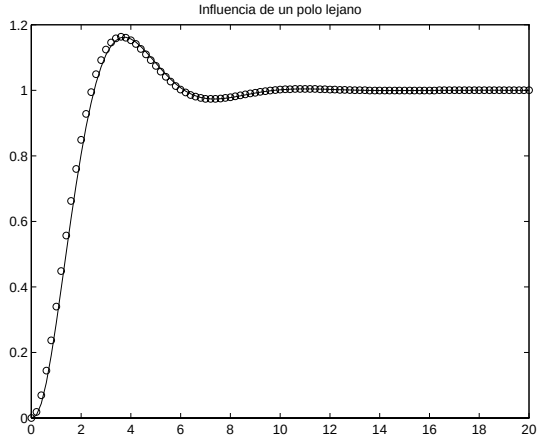

Figura 2.29: Influencia de un polo lejano en la respuesta temporal

Si se analiza el caso de un polo cercano al eje imaginario:

```
t = [0:0.2:20]';
K=1;  wn=1;  d=0.5;  p=0.2;
num  = K*wn^2;
num2 = wn^2*K*p;
den  = [1 2*d*wn wn^2];
den2 = conv (den,[1 p]);
y  = step (num,den,t);
y2 = step (num2,den2,t);
plot (t,y2,t,y,'o');
title ('Influencia de un polo lejano');
disp 'Situacion de los polos:';
roots(den2)
```

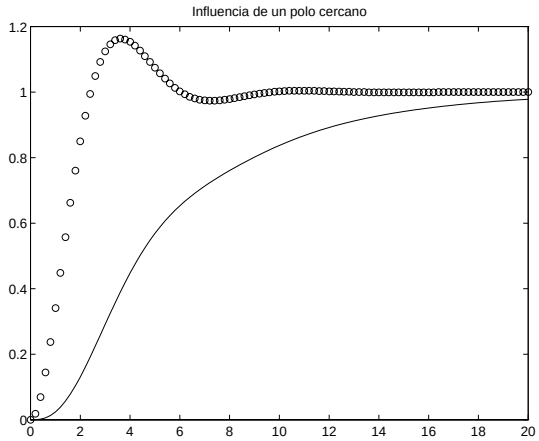

Figura 2.30: Influencia de un polo cercano al eje imaginario

2.4 TRATAMIENTO MEDIANTE FUNCIONES DE TRANSFERENCIA. SISTEMAS DISCRETOS

Se incluyen en esta sección algunas aclaraciones sobre comandos para tiempo discreto, cuya sintaxis suele ser similar, en su caso, a su equivalente continuo, añadiendo una `d` delante. Se van a exponer sólo unas pocas, dejando al lector el análisis por su cuenta del resto.

- `[Nz,Dz] = c2dm (N,D,Ts,'metodo')`: Discretización de un modelo en tiempo continuo, cuya función de transferencia viene dada por los polinomios numerador y denominador. Como tercer parámetro se especifica el periodo de muestreo. El último parámetro proporciona una cadena de caracteres que indica el método con el que se va a hacer la discretización, las posibilidades son:
  - `'zoh'`: Discretización utilizando mantenedor de orden cero (ZOH). Es la opción por defecto.
  - `'foh'`: Discretización utilizando mantenedor de orden uno (FOH).
  - `'tustin'`: Discretización mediante aproximación trapezoidal.
  - `'prewarp'`: Discretización trapezoidal con *prewarping*.
  - `'matched'`: Discretización mediante emparejamiento de polos y ceros (ver [2], pag. 147).

  Se echa de menos en esta función la posibilidad de usar otros métodos de discretización como son el rectangular hacia delante o hacia atrás. También da numerosos problemas cuando se intenta discretizar una función no propia (como pueda ser la función de transferencia de un controlador PID). No es el único comando de MATLAB que tiene esta limitación.

- `[N,D] = d2cm(Nz,Dz,Ts,'metodo')`: Se trata de la función contraria a la anterior. Transforma un sistema discreto en uno continuo, mediante alguno de los métodos citados.

- `dstep(Nz,Dz)`: Calcula la respuesta temporal de un sistema discreto a una secuencia escalón. Esta función dibuja directamente la respuesta. Esta representación no tendrá en el eje horizontal valores temporales absolutos, sino que aparecerán múltiplos del periodo de muestreo. Por otro lado, si indicamos un parámetro de salida en la llamada al comando, `y = dstep (Nz,Dz);`, no se realiza la representación, deberemos hacerlo nosotros mismos. En este caso, sería conveniente pintar la curva con puntos, en lugar de con un trazo continuo (como por defecto hace el comando `plot`). Veámoslo con el siguiente ejemplo, que dará como resultado la figura que aparece a la izquierda en Fig. 2.31:

```
N = [0.2  0.3  1];
D = [1  0.9  1.2  0.5];
[Nz,Dz] = c2dm (N,D,1,'zoh');
y = dstep (Nz,Dz);
plot (y,'.');
title ('Respuesta escal{\'o}n de un sistema discreto');
xlabel ('Periodo de muestreo');
ylabel ('Salida');
grid;
```

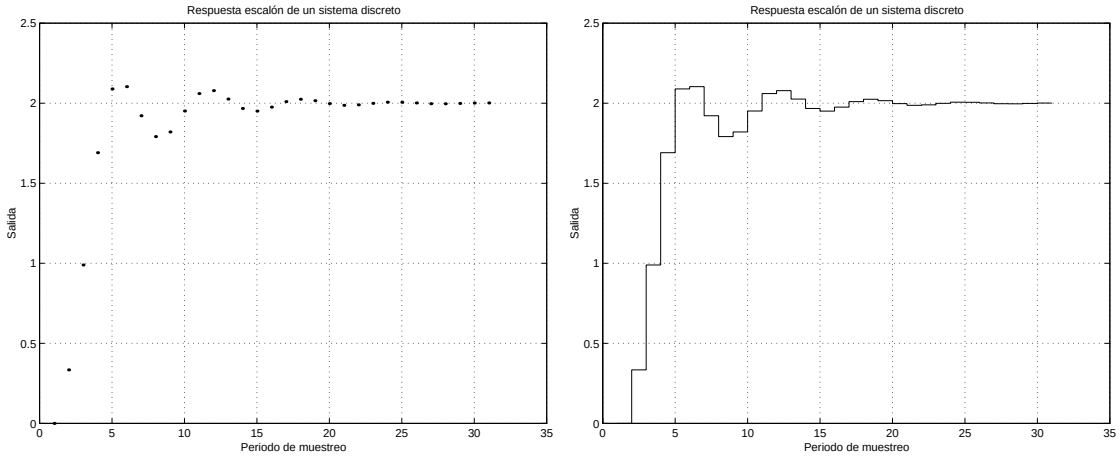

Figura 2.31: Respuesta ante escalón de un sistema discreto, mostrando sólo valores en instantes de muestreo (izq); mostrando salida continua, mantenida entre periodos de muestreo (der)

- `stairs(y)`: Para obtener una respuesta como la de que aparece a la derecha en Fig. 2.31, con la forma escalonada típica de sistemas digitales, simulando que la salida se mantiene constante entre dos periodos de muestreo, bastaría con reemplazar la instrucción `plot(y,'.')` del ejemplo anterior por `stairs(y)`.

- `dimpulse(Nz,Dz)`: Versión discreta de `impulse`.

- `dlsim(Nz,Dz)`: Versión discreto de `lsim`.

- `[mag,phase] = dbode(Nz,Dz,Ts,w)`: Calcula el diagrama de bode de un sistema en tiempo discreto. Siendo w un vector con las frecuencias donde queremos que se evalúen magnitud y fase de la función de transferencia.
- Existen también los correspondientes `dnyquist` y `dnichols`.
- El lugar de las raíces se calcula igual que en dominio s, usando `rlocus`. Sin embargo, se usa una rejilla distinta para que la representación tenga en cuenta el círculo de radio unidad. Por ejemplo, para el sistema discretizado anterior, si hacemos:

```
rlocus (Nz,Dz);
zgrid;
```

  podremos obtener analizar el lugar de las raíces en relación con los lugares geométricos de δ constante y $w_n T_s$ constante, en el plano z (Fig. 2.32).

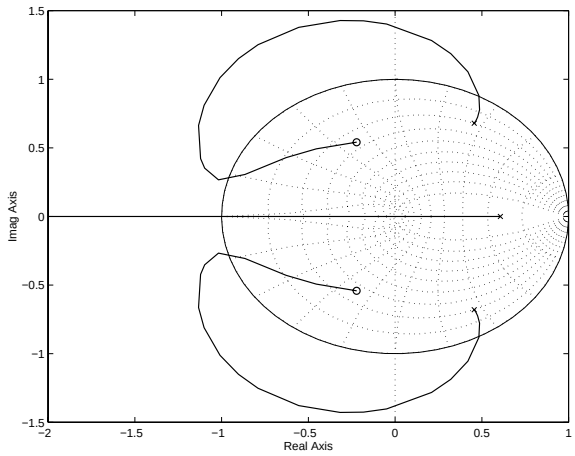

Figura 2.32: Lugar de las raíces del sistema discreto, mostrando rejilla plano Z

2.5 TRATAMIENTO MEDIANTE DESCRIPCIÓN EN EL ESPACIO DE ESTADOS

Se va a utilizar para el análisis la misma función de transferencia que ya apareció anteriormente:

$$H(s) = \frac{0.2s^2 + 0.3s + 1}{(s^2 + 0.4s + 1)(s + 0.5)}$$

El sistema se puede representar en el espacio de estados

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

Existen comandos que permiten obtener una descripción de un sistema en espacio de estados (*ss*) a partir de una función de transferencia, venga esta dada mediante polinomios numerador-denominador (*tf*) o mediante polos-ceros (*zp*), y viceversa:

`[A,B,C,D] = tf2ss(num,den)`	Paso de función de transferencia a espacio de estados
`[A,B,C,D] = zp2ss(z,p,k)`	Paso de descripción polo-cero a espacio de estados
`[num,den] = ss2tf(A,B,C,D)`	Paso de espacio de estados a función de transferencia
`[z,p,k] = ss2zp(A,B,C,D)`	Paso de espacio de estados a descripción polo-cero

Supongamos que disponemos de nuestra función de transferencia dada mediante:

```
num = [.2  .3  1];
den = conv([1 .4 1],[1 .5]);
```

con el comando `tf2ss`, podemos obtener las matrices correspondientes a su descripción en espacio de estados: `[A,B,C,D] = tf2ss (num,den)`.

La mayoría de las funciones que se han comentado en secciones anteriores para la manipulación y simulación de sistemas lineales dados por un par numerador-denominador, tienen su correspondencia para espacio de estados. Valgan los ejemplos siguientes:

```
step (A,B,C,D,1,t);
impulse (A,B,C,D,1,t);
[Abc,Bbc,Cbc,Dbc] = cloop (A,B,C,D,-1);
[A12,B12,C12,D12] = series (A1,B1,C1,D1,A2,B2,C2,D2);
[Ad,Bd] = c2dm (A,B,Ts,'metodo')};
```

y muchas otras.

Como es bien sabido, una propiedad fundamental de los sistemas es el concepto de estabilidad. Si se considera la ecuación no forzada $\dot{x} = Ax, \quad x(0) = x_0$, se dice que el sistema es asintóticamente estable si el estado alcanza el valor cero asintóticamente con el tiempo, es decir, $x(t) \to 0$ con $t \to \infty$. Se puede demostrar que esto ocurre cuando los autovalores de la matriz A tienen partes reales negativas. Por tanto, se puede analizar la estabilidad encontrando los autovalores de la matriz A, usando el comando:

```
evalues = eig (A)
```

2.5.1 Diseño de reguladores en el espacio de estados

Se analizan en esta sección una serie de comandos de MATLAB de gran utilidad para el diseño y simulación de esquemas de control por realimentación lineal del vector de estados:

- `K = place(A,B,P)`: Calcula la matriz o vector K de tal forma que los autovalores de $A - B * K$ (matriz de transición de estados del sistema en bucle cerrado) sean los especificados en el vector P.
- `K = acker(A,B,P)`: Calcula la matriz o vector de ganancias K tal que el sistema de una sola entrada $\dot{x} = Ax + Bu$, con una ley de control $u = -Kx$, tenga los polos en bucle cerrado en los valores especificados en el vector P. Es una implementación de la fórmula de Ackerman).
- `[y,x,t] = initial(A,B,C,D,x0)`: Proporciona la respuesta que describe el comportamiento de un sistema lineal continuo de la forma:

  $$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

  ante una cierta condición inicial `x0` de los estados. Devuelve la evolución temporal de la salida y de los estados Existe la correspondiente versión discreta de este comando `dinitial`.
- `Co = ctrb(A,B)`: Devuelve la matriz de controlabilidad del sistema
- `Ob = obsv(A,C)`: Devuelve la matriz de observabilidad.
- `[Ac,Bc,Cc,T,K] = ctrbf(A,B,C)`: Devuelve una descripción en espacio de estados en forma canónica de control, separando los subespacios controlables y no controlables.
- `[Ao,Bo,Co,T,K] = obsvf(A,B,C)`: Devuelve una descripción en espacio de estados en forma canónica de observación, separando los subespacios observables y no observables.

Ejemplo de diseño: Mostramos a continuación un breve ejemplo de un diseño de un controlador por realimentación lineal del vector de estados con un observador de orden completo:

```
% Se obtiene una descripcion continua en espacio de estados:
[A,B,C,D] = tf2ss (num,den);
% Se calcula, por ejemplo, la forma canonica de observacion:
[Ao,Bo,Co,T,J] = obsvf (A,B,C);
% Se calcula la dinamica deseada para el bucle cerrado (3 polos en s=-2):
Pd = poly ([-2,-2,-2]);
% Se calcula el vector de ganancias para realimentacion del vector de estados:
K = acker (Ao,Bo,[-2,-2,-2]);
% O bien se realiza manualmente:
```

```
K_ = [0 0 1] * inv([Bo, Ao*Bo, Ao^2*Bo]) * polyvalm(Pd,Ao);
% Se calcula un observador de orden completo, con una din{\'a}mica deseada:
L = place (Ao',Co',[-5,-5+j,-5-j]);
L = L';
% Formamos el sistema en bucle cerrado:
Abc = [Ao-Bo*K, -Bo*K; zeros(size(Ao)), Ao-L*Co];
Bbc = 0 * [Bo;Bo];
Cbc = [Co Co];
Dbc = 0;
x0 = [1;1;1;-1;-1;-1];
[y,x,t] = initial (Abc,Bbc,Cbc,Dbc,x0);
plot (t, [y, x]);
title ('Control por realim. vector estados con observador');
xlabel ('tiempo (s)');
ylabel ('Salida y estados');
```

El resultado del programa indicado aparece en la Fig. 2.33, donde se muestran tanto la evolución temporal de los estados reales partiendo de condiciones iniciales no nulas como los errores de estimación (suponiendo también un valor no nulo inicialmente en dicho error de estimación).

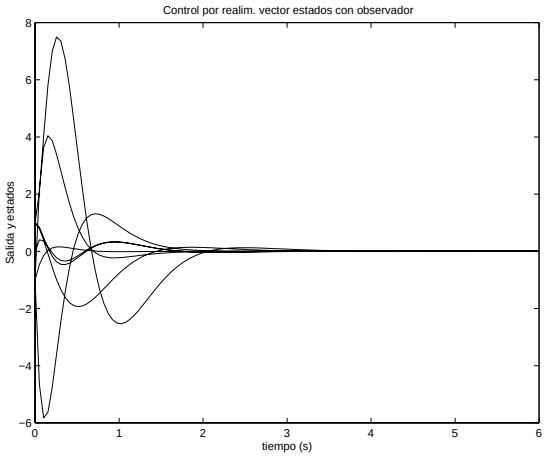

Figura 2.33: Simulación de la evolución de la salida y los estados reales

2.6 Manipulación mediante objetos

En las nuevas versiones del paquete de control (en consonancia con la nueva filosofía de MATLAB), se ofrece la posibilidad de manipular los sistemas mediante unas estructuras de datos específicas para modelos de sistemas lineales. Estos "objetos" harán más fácil la manipulación de los sistemas, pudiendo ser tratados de forma más abstracta, independientemente de que vinieran descritos mediante funciones de transferencia, conjuntos de polos y ceros o matrices correspondientes a una descripción en espacio de estados.

Para empezar, veamos justamente las tres formas de crear un "objeto de tipo sistema", en función de cómo proporcionemos la descripción de dicho sistema:

- `sys = tf(num,den)`
- `sys = zpk(ceros,polos,k)`
- `sys = ss(A,B,C,D)`

En cualquiera de los casos, se puede añadir un parámetro adicional que sería indicativo de un periodo de muestreo, y por tanto, el sistema sería discreto.

Para cada una de las tres posibilidades el objeto que se crea tiene una estructura diferente, puesto que tiene que almacenar datos de distinto tipo.

Supongamos que creamos dos sistemas mediante sendas funciones de transferencia:

```
sys1 = tf (1,[1 3 1]);
sys2 = tf ([1,2],1);
sys1

    Transfer function:
             1
       -------------
       s^2 + 3 s + 1
```

Como vemos, al pedirle el valor de *sys1* nos lo da de una forma monolítica, que hará muy cómoda su manipulación. Por ejemplo, podríamos sumar o poner en serie ambos sistemas sin más que:

```
sys1+sys2
       Transfer function:
       s^3 + 5 s^2 + 7 s + 3
       ---------------------
           s^2 + 3 s + 1
sys12 = sys1*sys2
       Transfer function:
           s + 2
       -------------
       s^2 + 3 s + 1
```

Por otra parte, las propiedades que están asociadas a estos objetos pueden verse mediante:

```
get(sys1)
              num: {[0 0 1]}
              den: {[1 3 1]}
         Variable: 's'
               Ts: 0
       InputDelay: 0
      OutputDelay: 0
    ioDelayMatrix: 0
        InputName: {''}
       OutputName: {''}
       InputGroup: {0x2 cell}
      OutputGroup: {0x2 cell}
            Notes: {}
         UserData: []
```

Nos informa del numerador y denominador de la función de transferencia en s. Al ser continua, su periodo de muestreo es $T_s = 0$. Se pueden asociar retardos a las entradas o salidas, asociar nombres a entradas o salidas, etc. Si en un momento dado, quisiéramos cambiar alguna propiedad podríamos hacerlo mediante el operador `.`:

```
sys1.Ts = 1;
sys1.num{1} = [0 0 2];
sys1
      Transfer function:
            2
      -------------
      z^2 + 3 z + 1
```

el nuevo sistema es discreto, con periodo de muestreo $T_s = 1$ y con un polinomio ganancia 2 en el numerador (no confundir esta manipulación con una discretización, en este caso el sistema continuo original y el discreto no tienen porqué guardar ninguna relación).

Estos objetos también están pensados para poder manipular cómodamente sistemas de múltiples entradas y múltiples salidas. De hecho, podríamos haber creado un sistema de la siguiente forma:

```
num1 = 0.5;   num2 = [1 1];
den1 = [1 2 1]; den2 = [1 0];
sys = tf ({num1,num2},{den1,den2})

            Transfer function from input 1 to output:
                 0.5
            -------------
            s^2 + 2 s + 1
```

```
            Transfer function from input 2 to output:
            s + 1
            -----
              s
% O lo que hubiera sido igual:
sys1 = tf (num1,den1);
sys2 = tf (num2,den2);
sys = [sys1, sys2];
```

2.7 Resumen de los comandos más importantes del Control System Toolbox

Normalmente, en la nomenclatura usada en este paquete de control, los comandos referentes a tiempo discreto que posean un equivalente para tiempo continuo, se denominan igual que éstos, pero precedidos de una `d`. Por ejemplo: `step` y `dstep`.

	Construcción de modelos
`append`	Agrupa dinámica de varios sistemas
`blkbuild`	Construye un sistema en representación en espacio de estados a partir del diagrama de bloques
`cloop`	Calcula el bucle cerrado de un sistema
`connect`	Modelado con diagrama de bloques
`feedback`	Conexión de sistemas realimentados
`ord2`	Genera las matrices A, B, C y D para un sistema de segundo orden
`pade`	Aproximación de Padé a un retardo
`parallel`	Conexión de sistemas en paralelo
`series`	Conexión de sistemas en serie

	Conversión de modelos
c2d	Conversión de sistema continuo a tiempo discreto
c2dm	Conversión de sistema continuo a tiempo discreto por varios métodos
c2dt	Conversión de sistema continuo a discreto con retardo
d2c	Conversión de sistema en tiempo discreto a continuo
d2cm	Conversión de sistema en tiempo discreto a continuo
ss2tf	Paso de representación en espacio de estados a función de transferencia
ss2zp	Conversión de espacio de estados a representación polo-cero
tf2ss	Paso de representación externa (función de transferencia) a interna (espacio de estados)
tf2zp	Conversión de función de transferencia a representación polo-cero
zp2tf	Paso de representación polo-cero a función de transferencia
zp2ss	Paso de representación polo-cero a espacio de estados

	Reducción de modelos
minreal	Realización mínima y cancelación polo-cero
modred	Reducción del orden del modelo

	Realización de modelos
canon	Conversión de un sistema a forma canónica
ctrbf	Matriz de controlabilidad en escalera
obsvf	Matriz de observabilidad en escalera

	Propiedades de los modelos
ctrb	Matriz de controlabilidad
damp	Factores de amortiguamiento y frecuencias naturales
dcgain	Ganancia en régimen permanente
ddamp	Factores de amortiguamiento y frecuencias naturales discretas
ddcgain	Ganancia discreta en régimen permanente
eig	Autovalores del sistema
esort	Clasifica los autovalores según su parte real
obsv	Matriz de observabilidad
roots	Raíces del polinomio

	Solución de ecuaciones
are	Solución algebraica de la ecuación de Riccati
dlyap	Solución de la ecuación de Lyapunov discreta
lyap	Solución de la ecuación de Lyapunov continua

	Respuesta temporal
`dimpulse`	Respuesta a impulso unitario en tiempo discreto
`dinitial`	Condiciones iniciales para la respuesta en tiempo discreto
`dlsim`	Simulación para entradas arbitrarias en tiempo discreto
`dstep`	Respuesta a escalón unitario en tiempo discreto
`filter`	Simulación de un sistema SISO en tiempo discreto
`impulse`	Respuesta impulsional continua
`initial`	Condiciones iniciales para la respuesta temporal
`lsim`	Simulación continua para entradas arbitrarias
`step`	Respuesta continua a escalón unitario

	Respuesta frecuencial
`bode`	Diagrama de Bode
`dbode`	Diagrama de Bode para tiempo discreto
`dnichols`	Ábaco de Nichols para tiempo discreto
`dnyquist`	Diagrama de Nyquist para tiempo discreto
`freqs`	Transformada de Laplace
`freqz`	Transformada Z
`margin`	Márgenes de fase y ganancia
`nichols`	Ábaco de Nichols
`ngrid`	Líneas de relleno para el ábaco de Nichols
`nyquist`	Diagrama de Nyquist

	Lugar de las raíces
`pzmap`	Mapeo polo-cero
`rlocfind`	Determinación interactiva de la ganancia en el lugar de las raíces
`rlocus`	Lugar de las raíces
`sgrid`	Relleno del lugar con líneas que indican w_n, δ constantes
`zgrid`	Relleno del lugar para tiempo discreto con líneas de w_n y δ constantes

	Selección de la ganancia
`acker`	Situación de los polos del bucle cerrado en sistemas SISO
`dlqe`	Diseño de un estimador lineal-cuadrático para tiempo discreto
`dlqew`	Diseño de un estimador general lineal-cuadrático para tiempo discreto
`dlqr`	Diseño de un regulador lineal-cuadrático para tiempo discreto
`dlqry`	Diseño de un regulador lineal-cuadrático para tiempo discreto con pesos en las salidas
`lqe`	Estimador lineal-cuadrático
`lqed`	Diseño de un estimador para tiempo discreto partiendo de una función de coste continua
`lqew`	Estimador lineal-cuadrático general
`lqr`	Regulador lineal-cuadrático
`lqrd`	Diseño de un regulador en tiempo discreto a partir de una función de coste continua
`lqry`	Regulador con pesos en las salidas
`place`	Situación de los polos dominantes

	Utilidades
`abcdchk`	Analiza la consistencia del grupo (A, B, C, D)
`dfrqint`	Selector automático de rango para diagramas de Bode en tiempo discreto
`dfrqint2`	Selector automático de rango para diagramas de Nyquist en tiempo discreto
`freqint`	Selector automático de rango para diagramas de Bode
`freqint2`	Selector automático de rango para diagramas de Nyquist

Bibliografía

[1] R.H. Bishop. *Modern Control Systems Analysis and Design Using* MATLAB. Addison-Wesley, 1993.

[2] G.F. Franklin and J.D. Powell. *Digital Control of Dynamic Systems*. Addison-Wesley, 1980.

[3] The MathWorks Inc. CONTROL SYSTEM TOOLBOX *User's Guide*. 1999.

[4] The MathWorks Inc. SIMULINK *User's Guide, version 3*. 1999.

[5] The MathWorks Inc. *Using* MATLAB*, version 5.3.1*. 1999.

[6] K. Ogata. *Solving Control Engineering Problems with* MATLAB*, year=1994, publisher=Prentice Hall International Editions*.

[7] B. Shahian and M. Hassul. *Control System Design using* MATLAB*, year=1993, publisher=Prentice Hall*.

[8] K. Sigmon. *Introducción a* MATLAB*, Segunda Edición*. Department of Mathematics, U. Florida. Traducido del inglés por Celestino Montes, Dep. Matemática Aplicada II, U. Sevilla, 1992.