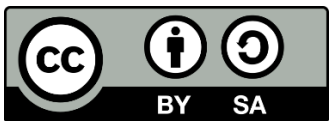
FUNDAMENTOS DE AUTOMÁTICA

GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

Universidad
Rey Juan Carlos



Anotaciones de prácticas

Autores: *Susana Borromeo López, Diego Martín Martín y Enrique Hernández Balaguera*

Curso 2023/2024

Índice

Universidad
Rey Juan Carlos

Área de Tecnología Electrónica

Práctica 1

Uso de MyApps URJC

Introducción a MATLAB:

Comandos básicos y creación de scripts

2023-2024

ÍNDICE

1. Instalación y uso de MyApps URJC

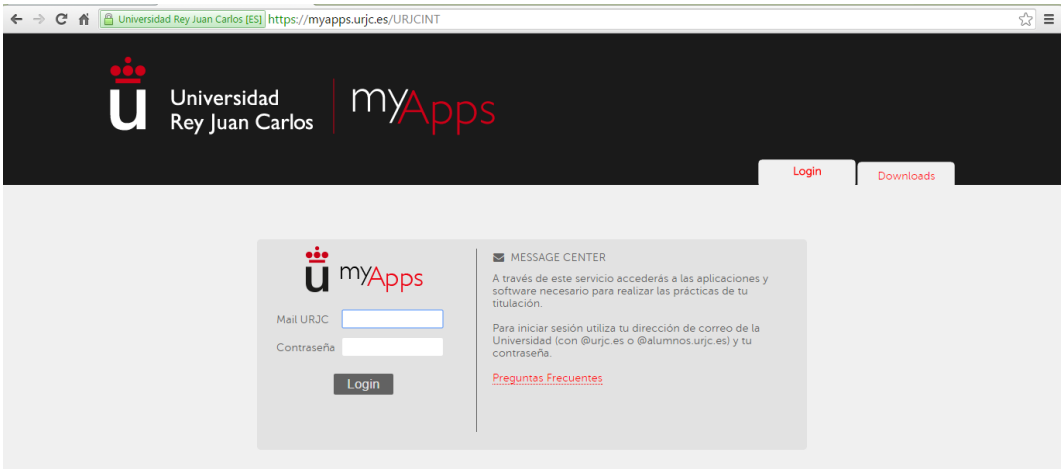
1.1 Instalación guiada de MyApps en un equipo informático propio

Para la realización de las prácticas de la asignatura de Fundamentos de Automática es necesario el software de cálculo numérico y programación denominado MATLAB.

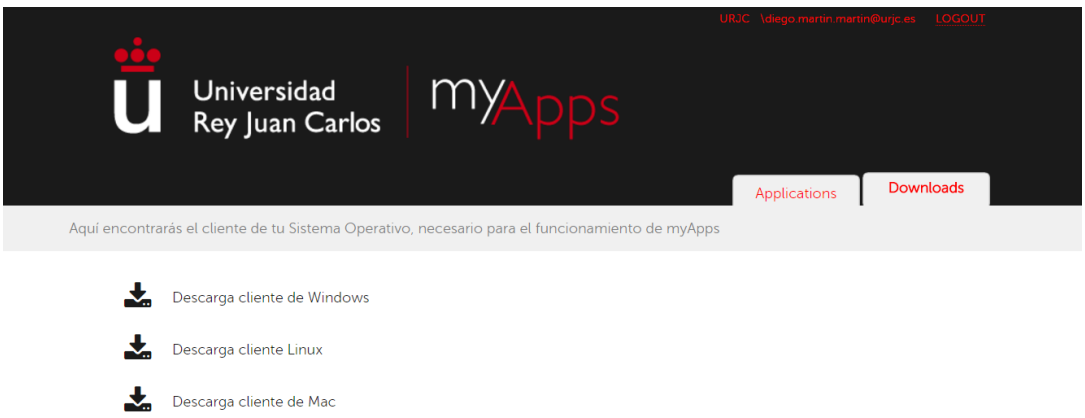
Desde hace algunos años la Universidad Rey Juan Carlos pone a disposición de los alumnos una herramienta de virtualización de aplicaciones informáticas denominada MyApps. Para acceder a la misma y usar sus aplicaciones necesitas tener una conexión a internet.

Abre el navegador e ingresa en la siguiente URL utilizando tu correo electrónico URJC y tu contraseña:

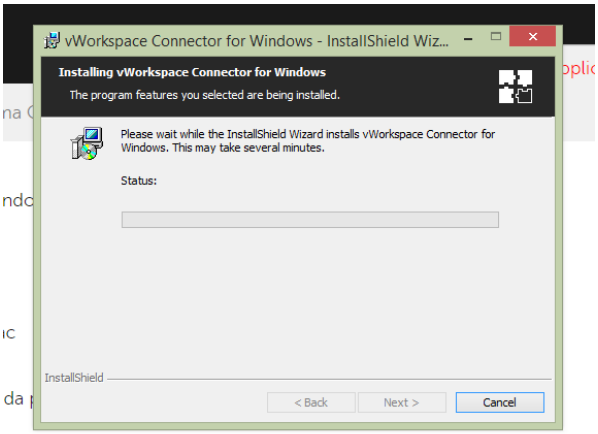
https://myapps.urjc.es



1. La primera vez que accedas a MyApps deberás instalar un cliente en tu equipo. Para ello, haz clic en la versión de tu Sistema Operativo y sigue las instrucciones de descarga e instalación.



2. Usa las opciones de instalación que aparecen por defecto:

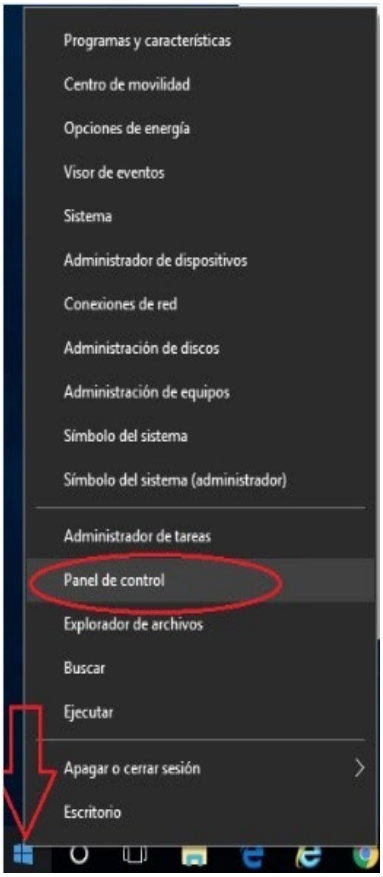


1.2 Acceso a MyApps desde Windows 10

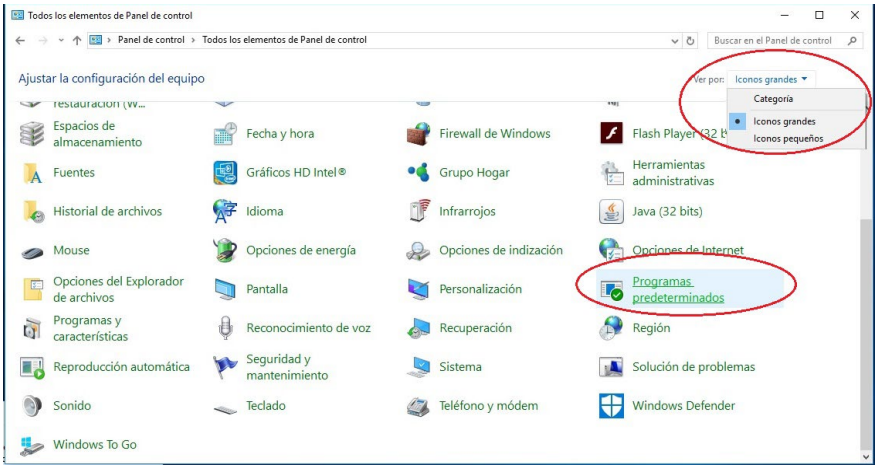
(Extraído de las "Preguntas frecuentes sobre MyApps" de https://cau.urjc.es/myapps/)

Si al acceder a myApps, y ejecutar una de las aplicaciones, se te descarga un fichero ".pit" y no consigues asociarlo al programa vWorkspace, debes realizar los siguientes pasos:

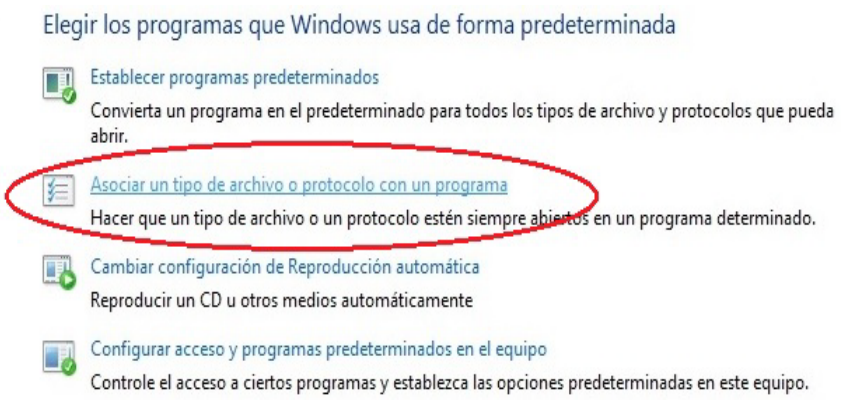
1. Iremos al Panel de Control (pulsando con el botón derecho del ratón sobre el menú de inicio),



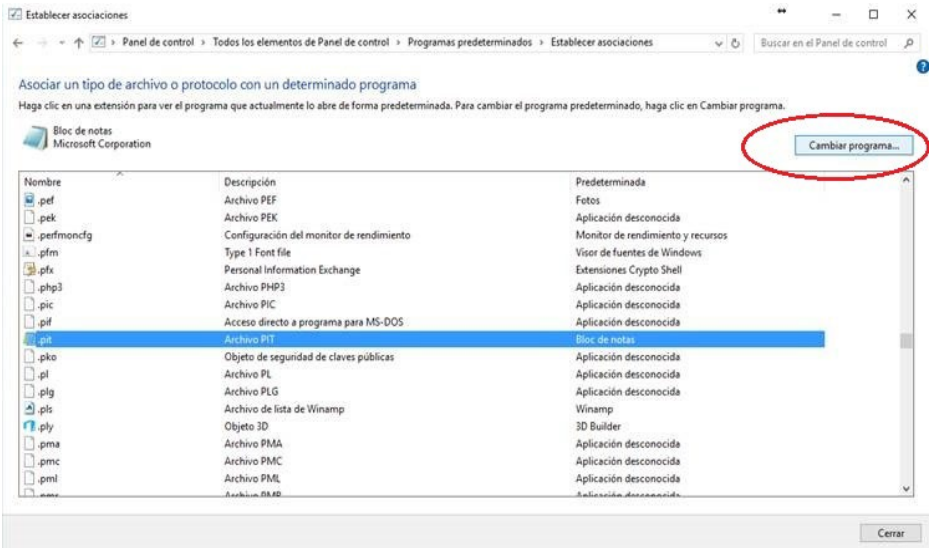
En la siguiente ventana que se nos muestra, lo configuraremos como "iconos grandes" y seleccionaremos "Programas Predeterminados"



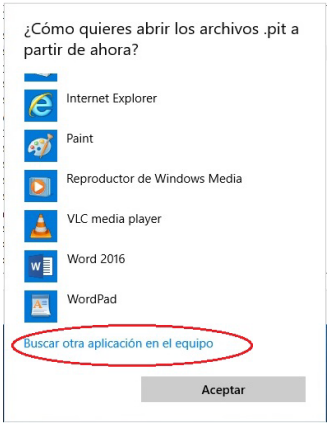
2. Seleccionaremos "Asociar un tipo de archivo o protocolo con un programa"



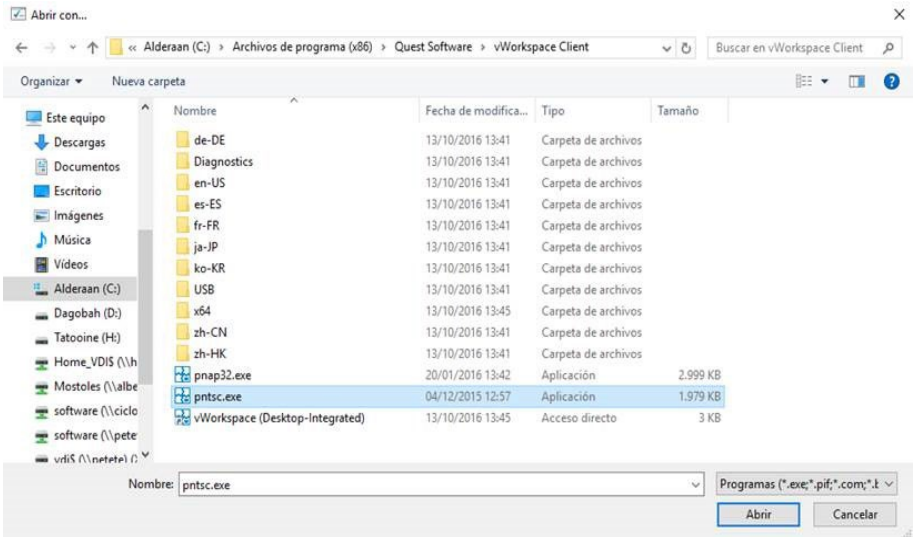
3. Tenemos que seleccionar la extensión **.pit** y darle a "Cambiar Programa"



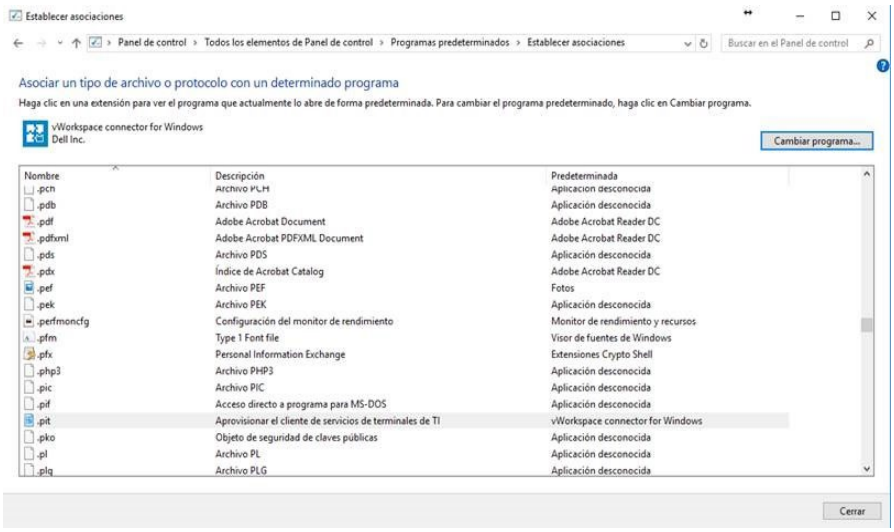
4. Hay que darle a "Buscar otra aplicación"



5. Le tenemos que indicar la siguiente ruta C:\Program Files (x86)\Quest Software\vWorkspace Client y seleccionar **pntsc.exe**

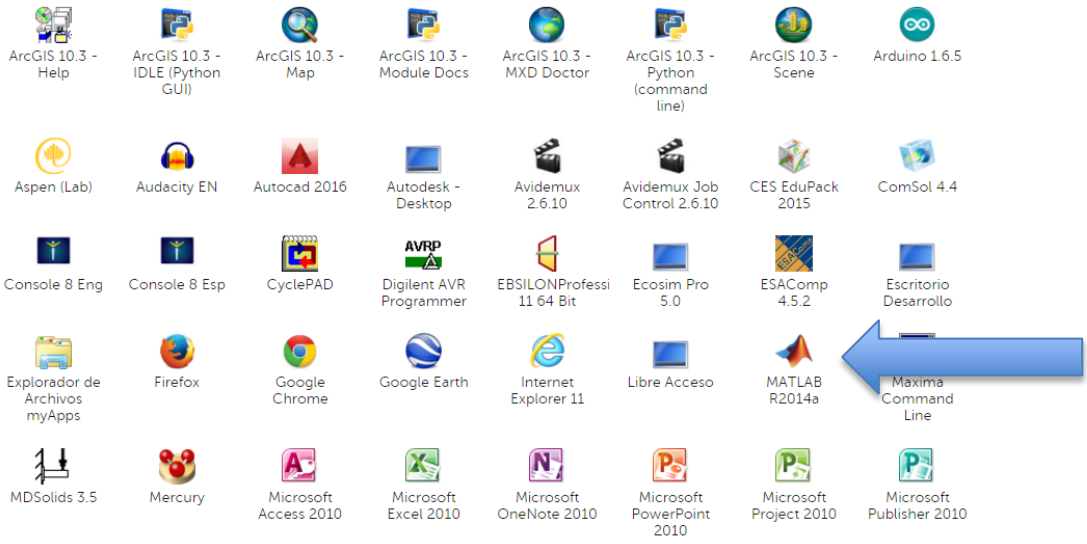


6. Y al ejecutarlo a partir de ahora, ya se asocia correctamente, y nos lo va a abrir automáticamente.



1.3 Arranque de MATLAB en MyApps

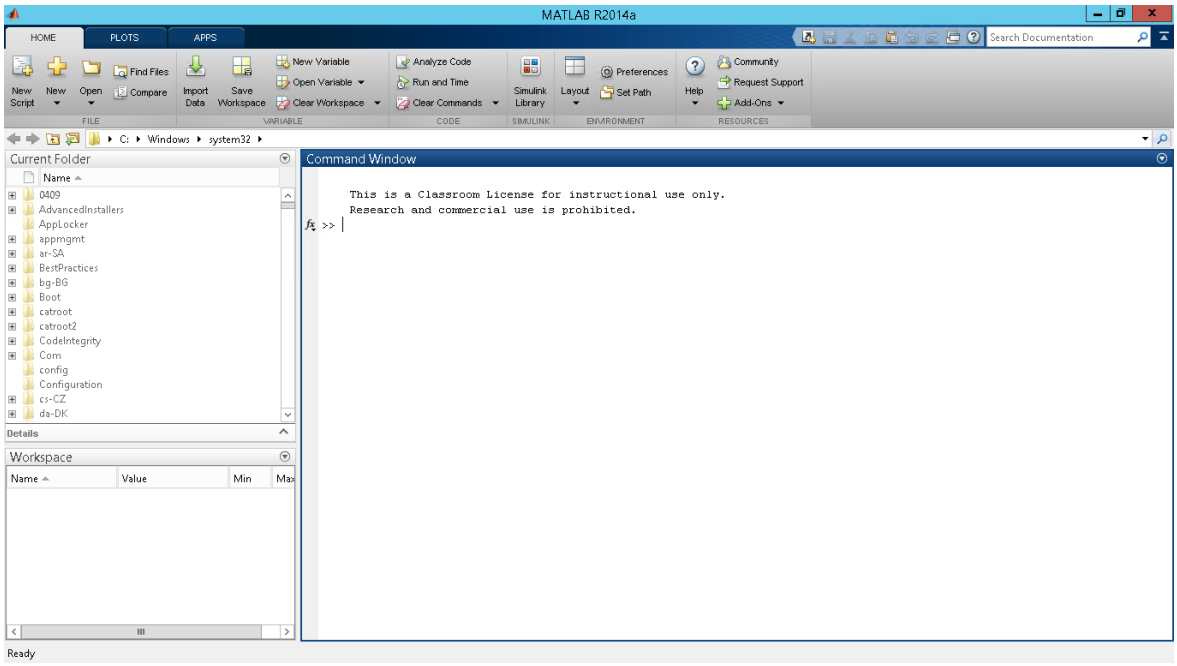
Una vez terminada la instalación ya podrás acceder al conjunto de aplicaciones disponibles para tu usuario a través del navegador. Comprueba que todo funcione correctamente y que MATLAB se encuentre entre las aplicaciones instaladas para tu usuario. Anota el número de versión que te aparece e intenta arrancar el programa.



Recientemente se ha incluido en MyApps la opción de visualizar las aplicaciones **directamente en el navegador web.** Esto permite acceder a las herramientas sin necesidad de tener el cliente de MyApps instalado, algo útil para acceder desde dispositivos móviles, por ejemplo.

Para activar esta opción debes hacer clic el marcador "HTML5" que aparece en la esquina superior derecha de la página de inicio de MyApps:

Arranca MATLAB y comprueba que puedes acceder a la aplicación. Si no dispones de una conexión rápida a internet, a veces puede tardar varios segundos. El resultado de este primer apartado de la práctica debería ser una ventana parecida a la siguiente:



2. Introducción a MATLAB

MATLAB, acrónimo de **MATrix LABoratory**, es hoy en día una de las principales herramientas *software* existentes en el mercado para el cálculo matemático, análisis de datos, simulación y visualización de resultados. Todas las operaciones que realiza MATLAB se basan en una estructura de datos matricial. Dentro del entorno de trabajo de MATLAB, se pueden definir nuevos comandos o funciones, programadas por el propio usuario, a través de ficheros .**m.** Este tipo de ficheros se encuentran en las llamadas *toolbox* de MATLAB, que son una colección de funciones ya programadas y disponibles para el usuario.

Dentro del campo del control, MATLAB ha desarrollado un gran número de funciones para el análisis de los sistemas de control automático. Todas ellas se encuentran dentro del ***Control System Toolbox,*** que permite el análisis de sistemas de control en el dominio del tiempo y de la frecuencia, tanto de sistemas continuos como discretos.

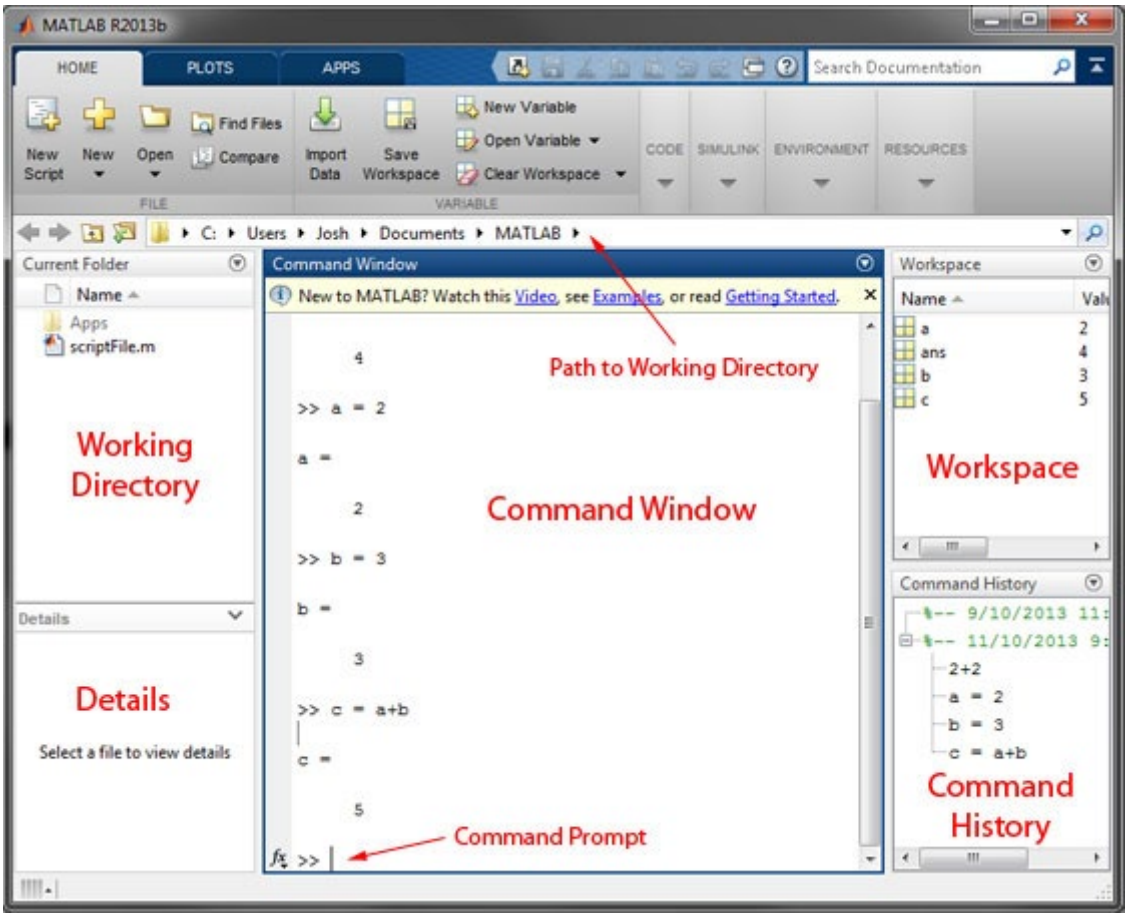
MATLAB incorpora interfaces gráficas como la *Root Locus Tool* para el análisis y diseño de sistemas de control mediante el método del lugar de las raíces o la *System Identification Tool* para identificación de sistemas.

La interfaz gráfica por excelencia es *Simulink*, que es una aplicación integrada dentro de MATLAB, y que permite simular mediante la construcción gráfica de diagrama de bloques.

2.1 Interfaz de usuario de MATLAB

Aunque depende de la versión del programa que estemos usando, al iniciar MATLAB aparece una interfaz gráfica de usuario que contiene tres elementos básicos que debemos conocer bien. Estos elementos básicos son los siguientes:

- **Consola de comandos:** Situada en la parte central de la ventana de la aplicación, nos permite introducir y ejecutar comandos en el prompt (>>) y visualizar los últimos comandos ejecutados.
- **Directorio de trabajo:** Situado a la izquierda, muestra los archivos contenidos en el directorio de trabajo. Dicho directorio se puede seleccionar desde la barra superior.
- **Workspace o espacio de trabajo:** conjunto de variables introducidas en MATLAB en la sesión actual, junto con sus valores actuales (se puede guardar y volver a cargar más adelante):



Cada una de estas áreas de la ventana principal se puede maximizar, minimizar o extraer a una ventana independiente ("undock"). Para recuperar la apariencia original, se puede utilizar la opción "Default" dentro del menú "Home ->Layout"

2.2 Comandos de ayuda y de sistema en MATLAB

Con el comando **help**, seguido del nombre de una instrucción o comando, se obtiene una ayuda **en línea** con información sobre la utilidad dicho comando. Además, muestra un resumen de las diferentes sintaxis y argumentos de los mismos.

Para obtener una información más detallada de una instrucción o comando, denominada "página de referencia" (ventana independiente, que incluye la sintaxis, descripción ejemplos de usos, comandos relacionados, etc.) se debe utilizar el comando **doc**, seguido del nombre de dicho comando.

Si se ejecuta simplemente **doc** se accede a la página principal de la documentación de MATLAB, categorizada por bloques de herramientas o Toolboxes.

¿Y qué ocurre si no sabemos o no recordamos el nombre del comando? En ese caso, podemos buscarlo con el comando **lookfor**, seguido de alguna palabra clave relacionada con la funcionalidad u objetivo del comando que estamos buscando.

Ejercicio práctico 1. Uso de comandos de ayuda

1. Averigua los Toolboxes que están instalados en tu versión de MATLAB, y si alguno de ellos tiene que ver con *Sistemas de Control.*
2. Busca los comandos de MATLAB que tienen que ver con funciones de transferencia (*transfer function*).

Mediante **memory** podrás conocer la memoria principal (RAM) instalada en tu equipo (sea virtual o no), además de la disponible para MATLAB

```
>> memory
Maximum possible array:        6877 MB (7.211e+09 bytes) *
Memory available for all arrays:        6877 MB (7.211e+09 bytes) *
Memory used by MATLAB:          635 MB (6.663e+08 bytes)
Physical Memory (RAM):         8192 MB (8.589e+09 bytes)

*  Limited by System Memory (physical + swap file) available.
>>
```

2.1 Vectores y matrices

En MATLAB los **vectores y matrices** se definen entre corchetes, con sus elementos separados por espacios o por comas. En el caso de las matrices, cada fila se delimita con un punto y coma.

>> vector = [1 2 3]

vector =

1 2 3

>> matriz= [1 3 5;2 4 6; 7 9 0]

matriz =

1 3 5

2 4 6

7 9 0

Los vectores también se pueden definir automáticamente si sus componentes están equiespaciados, indicando el valor inicial, incremento y valor final:

>> vector2 = 1:2:19 *% del 1 al 19, con un incremento de 2 unidades entre componentes*

vector2 =

1 3 5 7 9 11 13 15 17 19

Un vector columna se construye separando elementos con punto y coma o transponiendo un vector fila con el operador '.

>> vector3 = [4;3;2]

vector3 =

4

3

2

Ejercicio práctico 2. Creación de vectores y matrices

1 Averigua para qué sirven las instrucciones **vector_1 = linspace(0,10)** y **vector_2 = logspace(1,1000).**

2. Utilizando una única instrucción, crea una matriz A de dimensiones 5 x 10 donde sus elementos sean los 50 primeros números enteros, en orden de menor a mayor.

En MATLAB, cuando se necesita realizar operaciones **elemento a elemento** con una matriz se deben utilizar los operadores precedidos por un punto:

- Multiplicación elemento a elemento de dos matrices A y B: **A.*B**
- División elemento a elemento de dos matrices A y B: **A./B**
- Potenciación de una matriz A a la potencia n elemento a elemento: **A.^n**

Ejercicio práctico 3. Operaciones elemento a elemento en matrices

1. Crea una nueva matriz B donde cada elemento B_{ij} sea el cuadrado de cada elemento Aij de la matriz A del ejercicio práctico 2, punto 2.

2. Multiplica elemento a elemento las matrices A y B

2.3 Definición de polinomios

En MATLAB los polinomios se definen como vectores filas, siempre entre corchetes, los coeficientes de sus elementos en orden de potencia **descendente.** Se debe añadir un cero en la posición de aquellos elementos que no existen dentro del polinomio.

>> p1 = [1 6 5 -3] *%definición del polinomio* $x^3 + 6x^2 + 5x - 3$

>> p2 = [2 0 1 -1 1] *%definición del polinomio* $2x^4 + x^2 - x + 1$

Para obtener las **raíces de un polinomio** se utiliza el comando **roots:**

>> r1 = roots(p1)

r1 =

-4.8385

-1.5592

0.3977

Para definir un polinomio a través de sus raíces se utiliza el comando **poly:**

>> r3 = [-1;0.5+i;0.5-i];

>> p3= poly(r3)

p3 =

1.0000 0 0.250 1.250 *% el resultado es el polinomio* $x^3 + 0.25\,x + 1.25$

Ejercicio práctico 4. Manejo de polinomios

1. Halla los tres polinomios que tienen una raíz quíntuple igual a -1, a 2 y a -3, respectivamente.
2. ¿Qué característica tienen los polinomios con todas sus raíces reales y negativas?
2. Halla los polinomios que tiene las raíces complejas iguales a –i, i, tanto simples como dobles y triples. ¿Encuentras algo en común entre ellos?
3. Calculas las raíces del polinomio $x^{10} + x^5 - x + 1$

2.4 Creación de gráficos

Mediante la función **"plot"** de MATLAB se pueden representar funciones bidimensionales del tipo **y = f(t)** de la siguiente forma: dados los vectores de igual longitud "t" y "y" la función se dibuja con el comando **plot(t,y)**.

Si t es tiempo, se puede generar un vector de la siguiente forma t**=[0:1:100].** Este *array* genera un vector de 101 elementos de valores comprendidos entre el 0 y el 100.

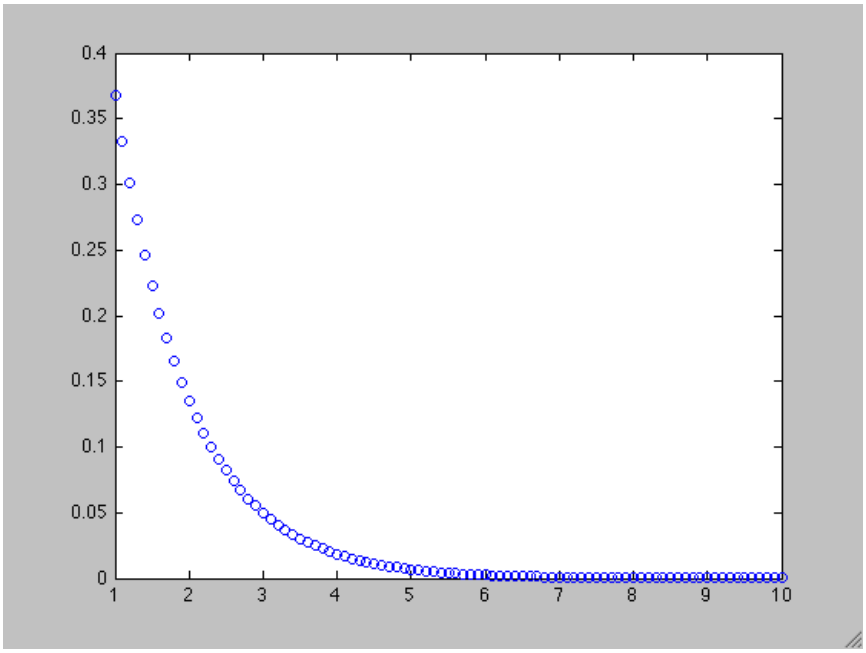
Si se quisiera representar $y=e^{-t}$, en un rango de t entre 0 y 10 a intervalos de 0.1 marcando los puntos de la gráfica con un símbolo circular 'o', entonces, las instrucciones serían:

>> t=[0:0.1:10]

>> y=exp(-1*t)

>> plot(t,y,'o')

que daría como resultado:



MATLAB genera una sola gráfica para la representación de resultados con el nombre ***Figure***. Si se desea representar sucesivos resultados sobre la misma gráfica, es necesario llamar a la función plot con sucesivos argumentos x,y, por ejemplo **plot(x,y1,x,y2)**. También se puede activar el comando **hold,** mediante la instrucción **hold on.** La desactivación del comando hold se realiza mediante la instrucción **hold off.**

La creación de nuevas ventanas de gráficas se realiza con el comando **figure.**

Los resultados gráficos se pueden personalizar especificando tres tipos de atributos: **color, símbolo y estilo (tipo, ancho) de línea**. La opción elegida se implementa añadiendo al comando **plot** una cadena de tres caracteres, entre comillas simples, indicando la opción deseada para cada atributo. Para ver los distintos tipos de opciones consultar la ayuda mediante los comandos **doc plot** y **doc LineSpec**

Ejercicio práctico 5. Gráficas de funciones reales. Utiliza distintas opciones de color, símbolo y tipo de línea que aparecen explicadas en la ayuda (doc plot) para:

1. Realizar las gráficas de $f(t) = t^2$ y $f(t) = t^3$ en el intervalo t=[-3,3] en el mismo gráfico
2. Realizar la gráfica de $f(t) = e^{-a \cdot t} \cdot \cos(t)$ en el intervalo t=[0,5], para distintos valores de a
3. Realizar la gráfica de $f(t) = e^{-t} \cdot \cos(bt)$ en el intervalo t=[0,5], para distintos valores de b

Los comandos **grid, xlabel, ylabel y title** añaden a la gráfica una rejilla, un texto a los ejes y un título respectivamente.

La modificación del escalado de los ejes X-Y en una gráfica se realiza con el comando **axis,** indicando mediante un vector de cuatro elementos el valor mínimo y máximo del eje de abscisa y de ordenada.

>>axis([minX maxX minY maxY])

Ejercicio práctico 6. Prueba los comandos anteriores en las gráficas realizadas en el ejercicio práctico 5.

Una gráfica se puede subdividir en zonas para sucesivas representaciones con el comando **subplot.** La gráfica, a modo de matriz, se divide en celdas a través del número de filas y columnas que se indique en el comando; a su vez se debe especificar qué celda activar para la representación de resultados.

```
>>subplot(1,2,1)    % Dividir en dos zonas una figura (1 fila, 2 columnas,).
                    % Activando la zona izquierda (zona 1, tercer argumento) para el plot.
```

Para obtener más información sobre el commando subplot, consulta la ayuda de MATLAB mediante **doc subplot**

Ejercicio práctico 7. Utilizando el comando subplot:

1. Introduce las gráficas b y c del ejercicio práctico 5 en una única gráfica dividida en una fila y dos columnas.
2. Crea una gráfica con subplot que contenga cuatro subgráficas de las funciones trigonométricas sen(t), cos(t), tan(t), sen(t)*cos(t).

Aunque en esta asignatura serán menos importantes, MATLAB también permite realizar gráficas de funciones de varias variables, histogramas, gráficos de barras, etc. Para obtener más información se puede consultar "Types of MATLAB Plots" en la ayuda de MATLAB, o consultar la tabla de comandos de gráficos de la siguiente página.

Types of MATLAB Plots

There are various functions that you can use to plot data in MATLAB®. This table classifies and illustrates the common graphics functions.

Line Plots	Pie Charts, Bar Plots, and Histograms	Discrete Data Plots	Polar Plots	Contour Plots	Vector Fields	Surface and Mesh Plots		Volume Visualization	Animation	Images
plot	area	stairs	polarplot	contour	quiver	surf	mesh	streamline	animatedline	image
plot3	pie	stem	polarhistogram	contourf	quiver3	surfc	meshc	streamslice	comet	imagesc
semilogx	pie3	stem3	polarscatter	contour3	feather	surfl	meshz	streamparticles	comet3	
semilogy	bar	scatter	compass	contourslice		ribbon	waterfall	streamribbon		
loglog	barh	scatter3	ezpolar	fcontour		pcolor	fmesh	streamtube		
errorbar	bar3	spy				fsurf		coneplot		
fplot	bar3h	plotmatrix				fimplicit3		slice		
fplot3	histogram	heatmap								
fimplicit	histogram2									
	pareto									

Ejercicio práctico 8. Gráficas 3D de funciones de varias variables usando "mesh()":

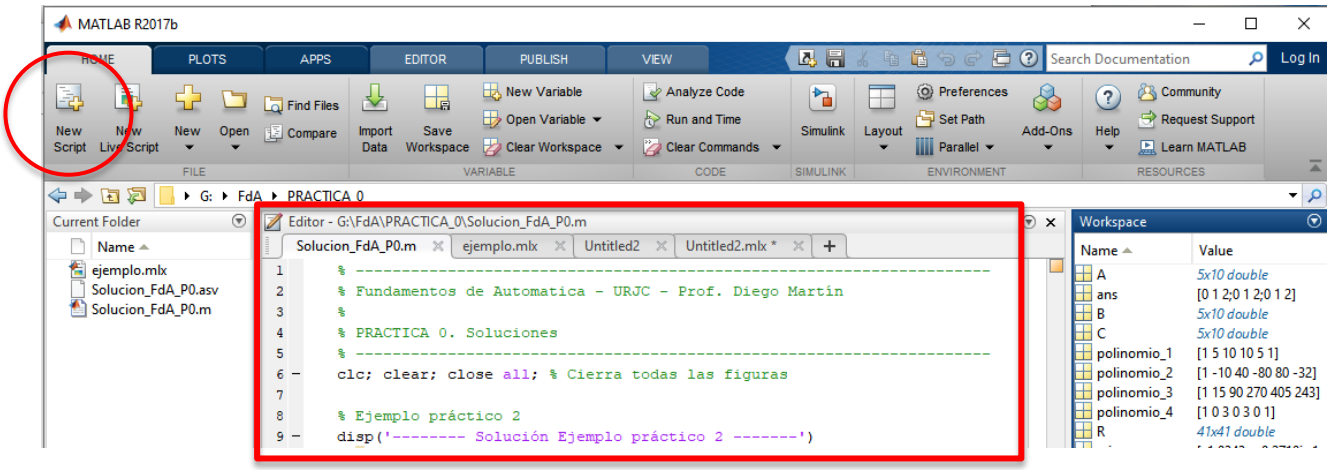
1. Realiza una gráfica de la función de dos variables $f(x,y) = x^2 + y^2$ en el intervalo [-2,2] para x e y.
2. Realiza una gráfica de la función de dos variables $f(x,y) = \sin(x^2 + y^2)/(x^2 + y^2)$ en el intervalo [-10,10] para x e y.
3. Realiza una gráfica de la función de dos variables f(x,y) = sin(x)*cos(y) en el intervalo [0,10] para x e y. Cambia el mapa de color del gráfico a la opción "jet" mediante **colormap(jet)** y observa el resultado.

Nota: Estudia el comando mesh(), y encontrarás que para usarlo las variables x e y deben ser una malla (matriz), no un vector. Dicha malla puede crearse con el comando "meshgrid".

2.5 Creación de scripts .m (ficheros para ejecución de comandos por lotes)

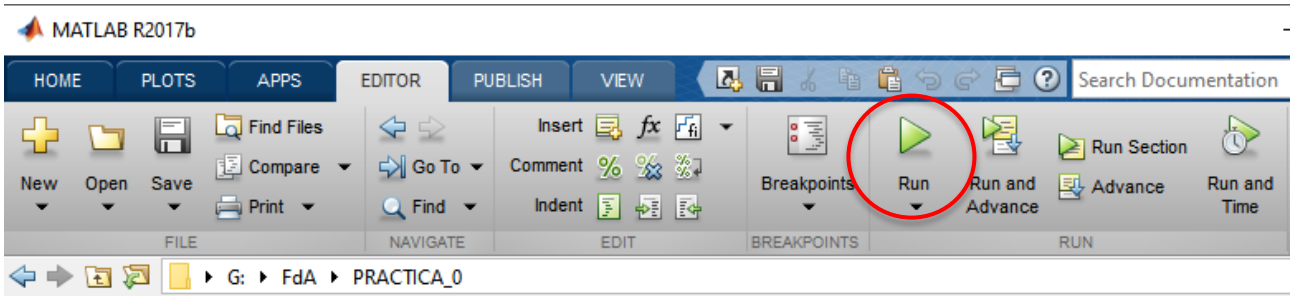
En MATLAB tienen especial importancia los **ficheros de extensión .m.** Dichos ficheros o scripts contienen conjuntos de comandos a ejecutar (o definiciones de funciones)

Un fichero .m es un fichero de texto plano (sin formato) y que pueden crearse a partir de un editor de textos cualquiera. No obstante, lo mejor es utilizar el editor del propio MATLAB al que se accede por defecto al abrir un nuevo fichero (Pestaña HOME -→ New Script):



Un fichero .m se ejecuta al teclear su nombre en la línea de comandos y pulsar intro. Es esencial que el fichero se encuentre almacenado y guardado en el directorio de trabajo seleccionado (*Current Folder*).

Otra opción para su ejecución es hacer clic en el icono RUN desde la pestaña EDITOR. Para ello el fichero debe estar guardado. En caso contrario MATLAB lo guardará automáticamente. Las salidas de los diferentes comandos se mostrarán en la consola.



Debes saber que en MATLAB la ejecución de un script .m es secuencial (de manera equivalente a los lenguajes de programación interpretados, como Python), es decir, se ejecutarán los comandos uno por uno, en orden de aparición, hasta el final del fichero o hasta que aparezca una línea que contenga un error de sintaxis.

En caso de error, la ejecución se parará y en la consola aparecerá, en color rojo, una descripción del error y de la línea del comando que lo contiene.

En los ficheros .m resulta muy recomendable introducir una cabecera con el nombre del autor, la fecha y una breve descripción de su contenido. Se pueden realizar mediante el uso de comentarios (utilizando % al principio de la línea de comentario). Su uso resulta fundamental para la organización y reutilización del trabajo que realicemos

```
% Esto es una línea de comentario en un fichero .m de MATLAB
```

Además, es recomendable incluir tras la cabecera los siguientes comandos, que nos permiten borrar el espacio de trabajo, la consola de comandos y las figuras previamente existentes al ejecutar el archivo .m:

```
clc; clear; close all;
```

Puedes usar el siguiente comando para mostrar un texto por la línea de comandos:

```
disp('-------- Solución Ejemplo práctico 2 -------')
```

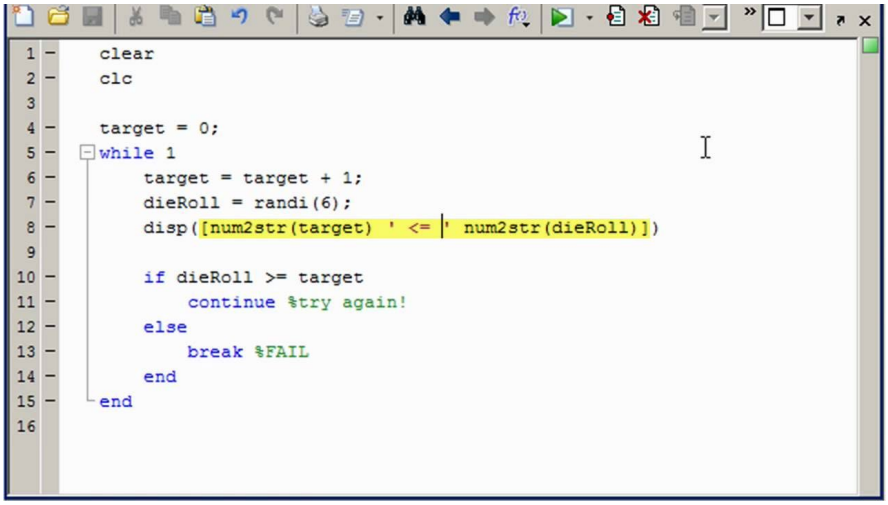
Por último, se puede dividir el archivo en secciones de código, para trabajar y ejecutar el código en diferentes fragmentos, controlando el flujo de ejecución, utilizando %%:

```
%% Un doble % indica el inicio de una sección de código
```

Ejercicio práctico 9

Crea un archivo .m para cada uno de los ejercicios prácticos anteriores, que contenga una cabecera, la sentencia de borrado descrita, comentarios y todos los comandos que has utilizado previamente.

Utilizando los archivos .m, en MATLAB se puede programar, ya que también están disponibles siguientes instrucciones de recursividad (control de flujo) **for** y **while (continue, break)** y sentencias condicionales **if (elseif, else, end)** y **switch (case, otherwise)**, haciendo de MATLAB un **lenguaje de programación estructurada**:

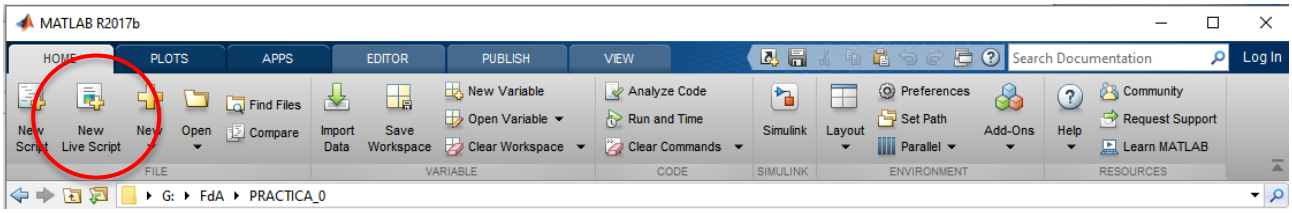


```
clear
clc

target = 0;
while 1
    target = target + 1;
    dieRoll = randi(6);
    disp([num2str(target) ' <= ' num2str(dieRoll)])

    if dieRoll >= target
        continue %try again!
    else
        break %FAIL
    end
end
```

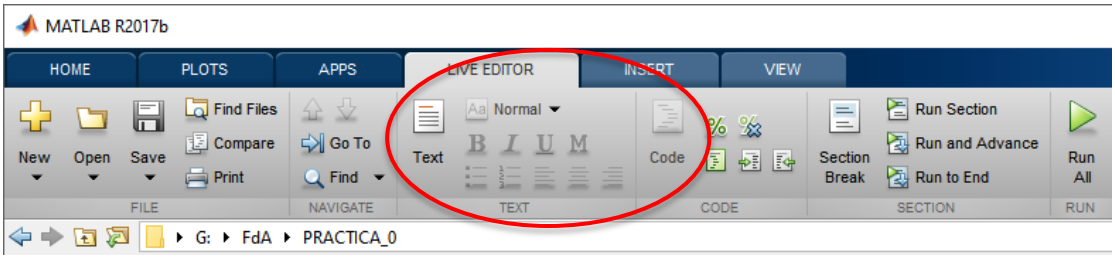
2.6 Y entonces, ¿qué es son los "live scripts" de MATLAB?

En las últimas versiones de MATLAB han aparecido otro tipo de scripts denominados "*Live Scripts*" o "*Guiones en vivo*". Se trata de ficheros con extensión .mlx que funcionan como documentos interactivos, combinando texto formateado con código de MATLAB ejecutable, cuya ejecución aparece insertada debajo (o al lado) de cada celda del *live script*, lo que permite generar archivos HTML, PDF o LaTeX para publicar

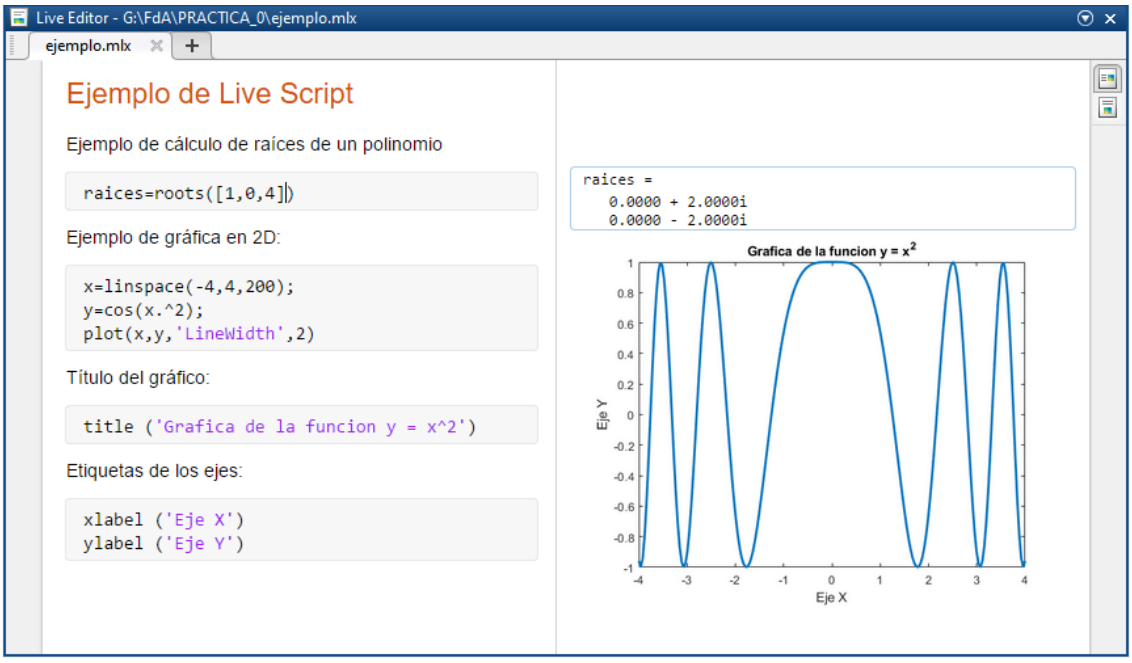
Para crear un Live Script puedes hacer clic en HOME → New Live Script:



Aparecerá un editor denominado "Live Editor" que lleva asociada una barra de herramientas propia, que te permitirá insertar código y celdas de texto intercaladas:



Este es el aspecto de un Live Script sencillo. Para más información, teclea **doc Live Script.**

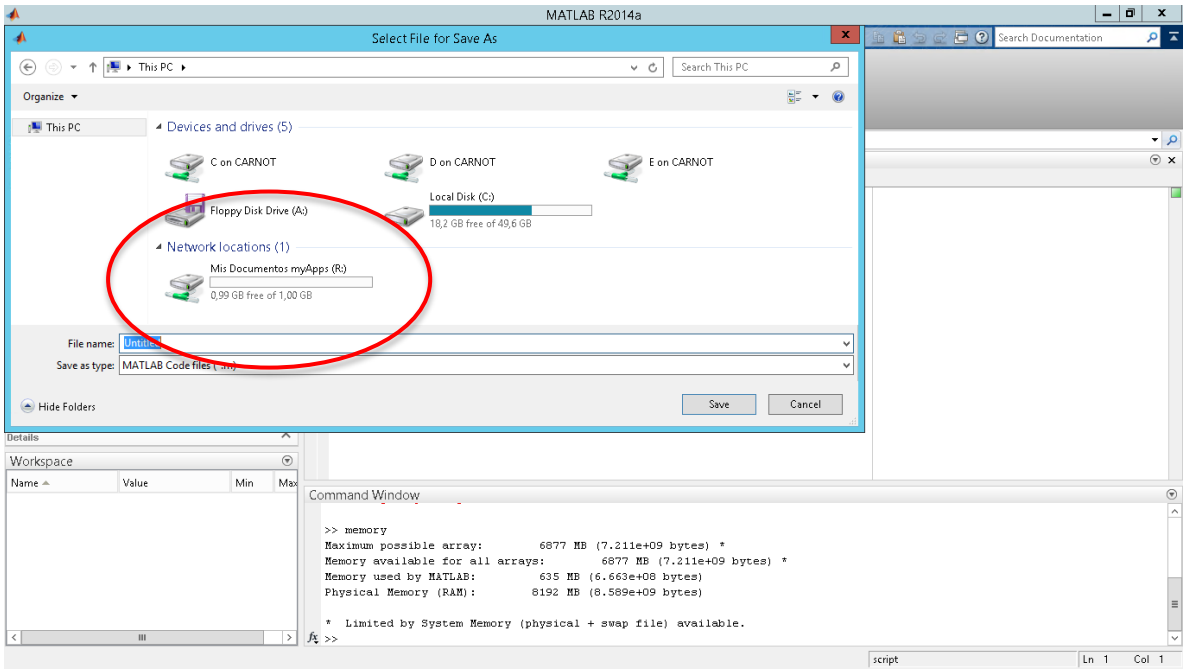


Ejercicio práctico 10.

Crea un Live Script como el de la imagen anterior o de alguno de los ejercicios prácticos anteriores. Prueba a exportarlo tanto a PDF (Save → Export to PDF) como a HTML (Save → Export to HTML), visualizando la salida desde el visor de archivos PDF y desde un navegador de internet, respectivamente.

2.7 Directorio de trabajo para MATLAB usando MyApps

Como ya seguramente sabes, en el servicio de MyApps de la URJC se dispone de un disco duro virtual (unidad R:) de 3 GB de espacio total, al que se puede acceder desde cualquier aplicación ejecutada en MyApps y desde cualquier equipo. Cuando trabajes con MATLAB desde MyApps, debes seleccionar siempre una carpeta de R: como directorio de trabajo:



Sin embargo, si usas MATLAB desde un ordenador propio resulta recomendable crear un directorio en un disco duro local, (C:, D: …), establecerlo como directorio de trabajo y utilizarlo para almacenar todos los archivos .m que se vayan creando durante las prácticas de esta asignatura.

2.8 Para saber más sobre MATLAB

Si deseas profundizar en tu conocimiento de MATLAB, aprender más operaciones básicas con matrices, importar ficheros de datos o realizar programas más completos, te recomendamos que consultes tanto el documento PDF titulado "*MANUAL DE MATLAB*" que se adjunta a esta práctica como la ayuda y los cursos online de MATLAB disponibles en https://matlabacademy.mathworks.com/es.

Universidad
Rey Juan Carlos

Área de Tecnología Electrónica

Práctica 2

MATLAB *Control System Toolbox*

Funciones de transferencia, transformadas de Laplace y álgebra de bloques con MATLAB

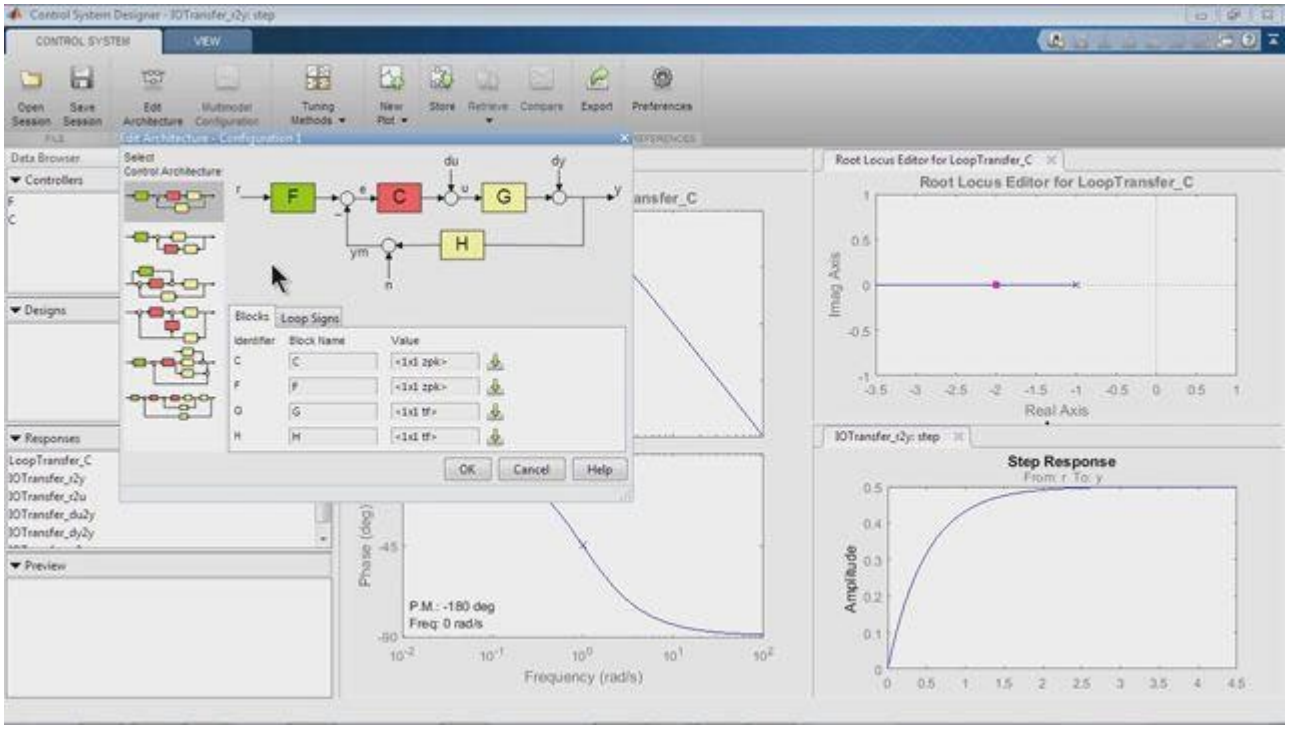
2023-2024

ÍNDICE

1. Introducción al *Control System Toolbox* de MATLAB

MATLAB proporciona un entorno muy completo para trabajar con sistemas de control. Dicho entorno se basa en las funciones, aplicaciones y algoritmos incluidos en el llamado *Control System Toolbox* del software.

Mediante el *Control System Toolbox* podremos crear modelos de sistemas de control LTI (lineales e invariantes en el tiempo) mediante representaciones de funciones de transferencia o modelos en el espacio de estados, ya sean sistemas continuos o sistemas discretos, y de una o varias entradas y salidas (SISO, MIMO). Además, dicho Toolbox contiene funciones que nos permitirán estudiar la respuesta transitoria y estacionaria de los sistemas a estímulos de entrada típicos como la función impulso, escalón, rampa, etc. También podremos analizar y diseñar sistemas de control mediante técnicas como el *lugar de las raíces* o la *respuesta en frecuencia*, así como diseñar y sintonizar controladores, estudiar el efecto de las perturbaciones sobre los sistemas, etc.



Para obtener una descripción más precisa de la potencia y versatilidad del *Control System Toolbox*, consulta el siguiente enlace:

https://es.mathworks.com/products/control.html

O visualiza el siguiente vídeo introductorio (en inglés):

https://www.youtube.com/watch?v=NfSPmBwD71g

2. Modelado de sistemas LTI mediante Funciones de Transferencia

En este apartado vamos a aprender a crear funciones de transferencia en MATLAB mediante diferentes comandos pertenecientes al *Control System Toolbox*. También veremos cómo pasar de una representación del sistema a otra.

Nota: resuelve los ejercicios prácticos que encontrarás a continuación utilizando scripts (.m) o LiveScripts (.mlx) de MATLAB.

2.1 Creación de funciones de transferencia en forma polinómica (TF)

Una función de transferencia G(s) en el dominio de la frecuencia compleja (dominio s) típicamente se expresa de forma polinómica o polinomial, es decir, como un cociente de polinomios de s, G(s) = num(s)/den(s):

$$G(s) = \frac{2s + 3}{4s^2 + 5s + 6}$$

Por ello, el comando más utilizado para definirlas es **G = tf(num,den)**, donde "num" y "den" son los polinomios del numerador y del denominador de la función de transferencia, respectivamente:

```
>> num = [2 3];
>> den = [4 5 6];
>> G = tf (num, den)
```

También se pueden pasar los polinomios directamente como argumentos del comando TF:

```
>> G = tf ([2 3], [4 5 6])
```

Ejercicio práctico 1. Definición de funciones de transferencia mediante TF

1. Define las tres siguientes funciones de transferencia en MATLAB:

$$G_1(s) = \frac{1}{s^2 + 1} \qquad G_2(s) = \frac{s + 1}{s^2 - 5s + 6} \qquad G_3(s) = \frac{s^2 + 2s + 3}{s^5 + 4s^3 + 5s + 6}$$

2. Una vez definidas, observa qué tipo de objeto usa MATLAB para almacenarlas (columna *Class* del Workspace)

Si el numerador o el denominador de la función de transferencia no están expresados en forma de un único polinomio, puedes ayudarte del comando de convolución de vectores en MATLAB, que permite realizar una multiplicación de polinomios:

```
>> conv (poli_1, poli_2)
```

Ejercicio práctico 2. Definición de funciones de transferencia no polinómicas

1. Define las dos siguientes funciones de transferencia ayudándote de conv():

$$G_4(s) = \frac{10(s+5)}{(s+1)(s-3)(s^2+3s+6)} \qquad G_5(s) = \frac{s^2+3}{(s^2+s+1)(s^3+2s+3)}$$

2.2 Creación de funciones de transferencia en forma factorizada (ZPK)

En otras ocasiones las funciones de transferencia no se presentan en forma polinominal sino que lo hacen en forma factorizada, es decir, como producto de las raíces $\mathbf{z}_i$ del numerador (que denominaremos ceros del sistema) entre el producto de las raíces $\mathbf{p}_i$ del denominador (que en este caso son los **polos del sistema**):

$$G(s) = \frac{K(s-z_0)(s-z_1)\dots(s-z_m)}{(s-p_0)(s-p_1)\dots(s-p_n)}$$

La función de transferencia, expresada en función de los polos y los ceros del sistema se puede definir mediante el comando **zpk** *(zero-pole-gain)*. Para ello se pasan como argumentos tres vectores, el que contiene los ceros (z), el de los polos (p) y el factor de ganancia estática (K). En un sistema que no contenga ceros el vector correspondiente al vector z debe estar vacío, indicándose con [].

>> G = zpk ([z_m … z_1 z_0], [p_n … p_1 p_0], K)

Ejercicio práctico 3. Definición de funciones de transferencia factorizadas

1. Define la función de transferencia $G_6(s)$ para un sistema que tiene una ganancia K igual a 2, un cero situado en -1 y tres polos situados en -2, -3 y -4
2. Repite el apartado anterior definiendo $G_7(s)$ para un sistema que tiene una ganancia K igual a 5, un cero situado en +1 y tres polos situados en -2, +3 y -4
3. Define la función de transferencia $G_8(s)$ que no tiene ceros y tiene un polo triple en -1 (con K=1)
4. Define la función de transferencia $G_9(s)$ que no tiene ceros y tiene tres polos, uno en -1 y los otros dos polos complejos conjugados -1 + 2i, -1 -2i (con K=1)

2.3 Conversión entre funciones de transferencia polinómicas y factorizadas

Los comandos tf y zpk también pueden tomar argumentos que ya sean una función de transferencia, de tal manera que:

- Si GP es una función de transferencia en forma polinómica, el comando **GZPK = zpk(GP)** convierte la función GP a forma factorizada.

- Si GZPK es una función de transferencia en forma factorizada, el comando **GP = tf(GZK)** convierte la función GP a forma polinomial.

Ejercicio práctico 4. Conversión de funciones de transferencia

1. Convierte a forma factorizada las funciones de transferencia G1, G2, G3, G4 y G5 definidas en los ejercicios prácticos 1 y 2
2. Convierte a forma polinómica las funciones de transferencia G6, G7, G8 y G9 definidas en los ejercicios prácticos 1 y 2

2.4 Forma simplificada: uso de s = tf('s')

Una vez definidas varias funciones de transferencia, MATLAB permite realizar operaciones algebraicas simples entre varias funciones (suma, resta, multiplicación y división de funciones de transferencia) y/o producto de funciones de transferencia por escalares.

Por ello, una última forma muy simplificada de definir funciones de transferencia se basa en definir la variable "s" como función de transferencia (mediante **s = tf ('s')**) y a continuación especificar la función buscada mediante una **expresión racional** que incluya la variable s (es decir, con sumas, restas, multiplicaciones y divisiones de "s"), por ejemplo:

```
>> s = tf ('s')

>> G = (s + 1)/(s^3 + 3*s + 1)
```

El anterior método sería equivalente al uso de la siguiente expresión

```
>> G = tf ( [1 1] , [1 0 3 1] )
```

Ejercicio práctico 5. Uso de s = tf('s')

1. Utilizado este método simplificado para definir una función de transferencia del ejercicio práctico 1, las dos funciones del ejercicio práctico 2 y una función del ejercicio práctico 3.

3. Transformadas inversas de Laplace en MATLAB

3.1 Descomposición (expansión) en fracciones simples

Con MATLAB podemos realizar descomposiciones (también denominadas expansiones) en fracciones simples del cociente de dos polinomios, utilizando el comando **residue**.

Para ello, si queremos descomponer una transformada de Laplace Y(s) con **residue**, ésta tiene que estar en forma polinómica Y(s) = b(s)/a(s). La sintaxis habitual es la siguiente:

>> [r, p, k] = residue (b, a)

- **b** y **a** son los polinomios del numerador y del denominador, respectivamente.
- **r** es un vector que contiene los residuos buscados.
- **p** es otro vector que contiene las raíces (polos) del denominador.
- En caso de que el grado del numerador b(s) sea igual o mayor que el del denominador a(s), el vector **k** contendrá el polinomio resultado del **cociente entre ambos**, y la descomposición se realizará sobre el resto de dicho cociente. Si dicho grado del numerador es inferior al del denominador, **k** será un vector vacío.

$$\frac{b(s)}{a(s)} = \frac{b_m s^m + b_{m-1}s^{m-1} + \ldots + b_1 s + b_0}{a_n s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_0} = \frac{r_n}{s - p_n} + \ldots + \frac{r_2}{s - p_2} + \frac{r_1}{s - p_1} + k(s)$$

Ejercicio práctico 6. Descomposición en fracciones simples

1. Calcula los residuos y los polos de las descomposición en fracciones simples de las siguientes funciones Y(s):

$$Y_1(s) = \frac{32}{s^3 + 12s^2 + 32s} \qquad Y_2(s) = \frac{6s^2 + 22s + 18}{s^3 + 6s^2 + 11s + 6}$$

$$Y_3(s) = \frac{s + 7}{s^2 + 10s + 25} \qquad Y_4(s) = \frac{2s^3 + 25s^2 + 95s + 110}{s^3 + 9s^2 + 26 + 24}$$

2. Una vez descompuestas, calcula sus transformadas inversas de Laplace (tabla).

Residue también opera de forma inversa, es decir, si se le proporcionan como argumentos los valores de las raíces, los polos y el vector cociente k, proporciona como salida el numerador y denominador de la función original, con la siguiente sintaxis:

>> [num, den] = residue (raices, polos, k)

Para más información y ejemplos, consulta la ayuda de MATLAB (**doc residue**).

3.2 Transformadas simbólicas usando el *Symbolic Math Toolbox*

En MATLAB también existe la posibilidad de trabajar con variables simbólicas, utilizando el *Symbolic Math Toolbox*. Este potente Toolbox contiene comandos que permiten manipular expresiones, resolver ecuaciones simbólicas, realizar derivadas, integrales, etc. Entre sus capacidades también se encuentra el manejo de transformadas de Laplace.

Para usar el *Symbolic Math Toolbox* es imprescindible definir previamente aquellas variables que serán simbólicas utilizando el comando **syms**. Por ejemplo, para definir la variable s como simbólica, usaremos:

```
>> syms s
```

Nota: no es compatible definir s como variable simbólica (**syms s**) y como función de transferencia (**s = tf('s')**) simultáneamente.

Para calcular transformadas inversas de Laplace usaremos el comando **ilaplace.** Por ejemplo, para calcular la salida en el dominio del tiempo de un sistema con una función de transferencia G(s) ante una entrada escalón R(s) = 1/s, haremos:

```
>> syms s
>> G = 1/(s^2 + 3*s + 2)
>> R = 1/s
>> Y = G*R
>> y = ilaplace (Y)
```

Para trabajar con expresiones simbólicas resulta muy recomendable utilizar **LiveScritps de MATLAB**, principalmente porque la salida de cada comando se muestra con un renderizado tipográfico avanzado. Si queremos obtener un formateado similar por línea de comandos, podremos usar la función **pretty()**, aunque el resultado es bastante peor que el que se obtiene con un LiveScript.

Ejercicio práctico 7. Transformadas inversas de Laplace con MATLAB

1. Utilizando un LiveScript, calcula la respuesta en el dominio del tiempo de las siguientes funciones de transferencia ante una entrada escalón unitario:

$$G_1(s) = \frac{1}{(s+1)} \qquad G_2(s) = \frac{5}{(s+1)(s+5)}$$

$$G_3(s) = \frac{1}{(s+1)^3} \qquad G_4(s) = \frac{1}{s^2+s+1}$$

2. Calcula también la respuesta en el dominio del tiempo de las funciones de transferencia anteriores ante una entrada rampa unitaria.

Se pueden realizar gráficas de una función simbólica en un intervalo determinado utilizando el comando **fplot**. Para incluir varias funciones en la misma gráfica, el primer argumento de fplot debe ser un vector que contenga dichas funciones. Por ejemplo:

```
>> syms t
>> x = cos (t);
>> y = sin (t);
```

>> fplot ([x, y] [0, 4*pi])

>> legend ("cos(t)",'sin(t)") *% Permite modificar y mostrar la leyenda en el gráfico.*

Nota: Si las funciones argumento de **fplot** provienen de una transformada inversa de Laplace obtenida con el comando **ilaplace**, no es necesario especificar que t es una variable simbólica (es decir, no hace falta introducir **syms t**).

Ejercicio práctico 8. Gráficas de funciones simbólicas

1. Añade el comando fplot al LiveScript del ejercicio práctico 7 para realizar la gráfica, por separado, de cada una de las transformadas inversas de Laplace que has ido calculando, en el intervalo [0, 10].
2. Realiza una única gráfica en la que aparezcan las respuestas ante una entrada escalón de las tres funciones de transferencia siguientes. ¿Observas alguna relación entre ellas?

$$G_1(s) = \frac{1}{(s+1)} \qquad G_2(s) = \frac{5}{(s+5)} \qquad G_3(s) = \frac{5}{(s+1)(s+5)}$$

El *Symbolic Math Toolbox* también nos permite calcular transformadas de Laplace, mediante el comando **laplace()**. Esto resulta de mucha utilidad si no tenemos a nuestro alcance la tabla de transformadas. Por ejemplo:

>> syms t

>> y = t

>> Y = laplace (y)

Ejercicio práctico 9. Transformadas de Laplace con MATLAB

1. Calcula las transformadas de Laplace de las siguientes funciones del tiempo:

$$x_1(t) = t^2 \qquad x_2(t) = \cos(t) \qquad x_3(t) = \sin(2t)$$

$$x_4(t) = e^{-2t}\cos(t) \qquad x_5(t) = t^2 e^{-2t}\sin(t)$$

Por último, indicar que existen funciones en el *Symbolic Math Toolbox* que nos permiten pasar de expresiones simbólicas a polinomios (**sym2poly**) y viceversa (**poly2sym**), lo que nos permitiría, en caso de que fuera necesario, conectar los dos Toolboxes que hemos utilizado en esta práctica. Para más información, consulta la ayuda de MATLAB.

4. Álgebra de bloques mediante MATLAB

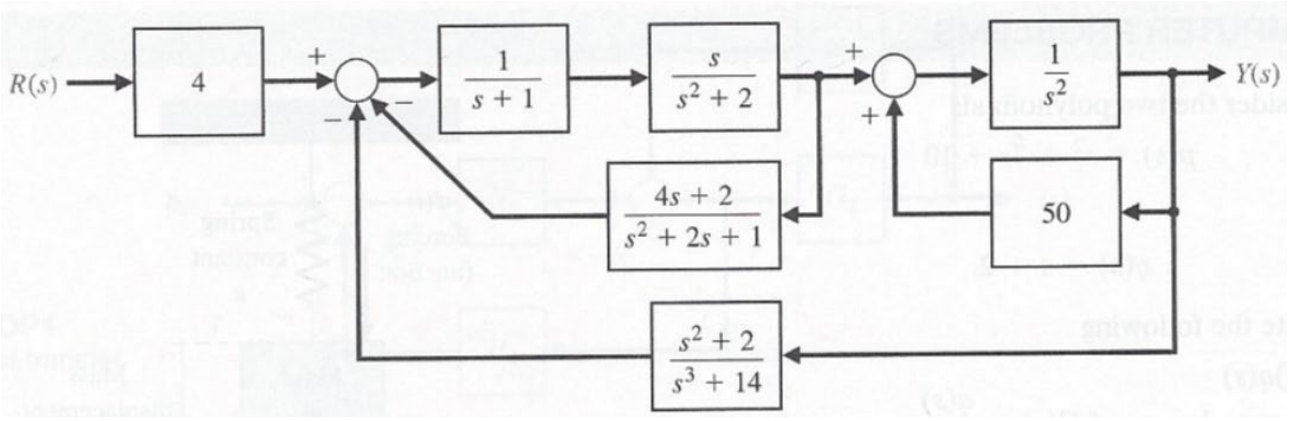
4.1 Herramientas para la interconexión y reducción de diagramas de bloques

En el *Control System Toolbox* encontramos varios comandos que nos permiten tanto conectar como reducir diagramas de bloques. De todos ellos destacan los tres siguientes:

- **Asociación en cascada o serie:** el comando **series (G1, G2)** obtiene la función de transferencia equivalente del conjunto de dos bloques G1 y G2 conectados en serie.
- **Asociación en paralelo:** el comando **parallel (G1, G2)** obtiene la función de transferencia equivalente del conjunto de dos bloques G1 y G2 conectados en paralelo.
- **Sistemas realimentados:** el comando **feedback (G, H)** obtiene la función de transferencia equivalente de un sistema realimentado. A este comando se le pasan, por este orden, la función de transferencia G del lazo directo y H, la de la realimentación, separadas por comas. Por defecto se considera una realimentación negativa. En el caso de la realimentación positiva, se debe añadir un 1 a continuación de las funciones de transferencia, es decir, la sintaxis debe ser **feedback (G, H, 1)**

Ejercicio práctico 10. Reducción de diagramas de bloques

1. Utiliza los comandos de reducción de diagramas de bloques para calcular la función de transferencia en lazo cerrado del siguiente sistema de control:



Otros comandos menos utilizados en el contexto de esta asignatura para la interconexión de bloques son **connect** y **append**. Utiliza la ayuda de MATLAB para descubrir su utilidad y sintaxis.

Universidad
Rey Juan Carlos

Área de Tecnología Electrónica

Práctica 3

Polos y ceros de una función de transferencia, respuesta transitoria de un sistema de control con MATLAB y LTIViewer, precisión y error en estado estacionario

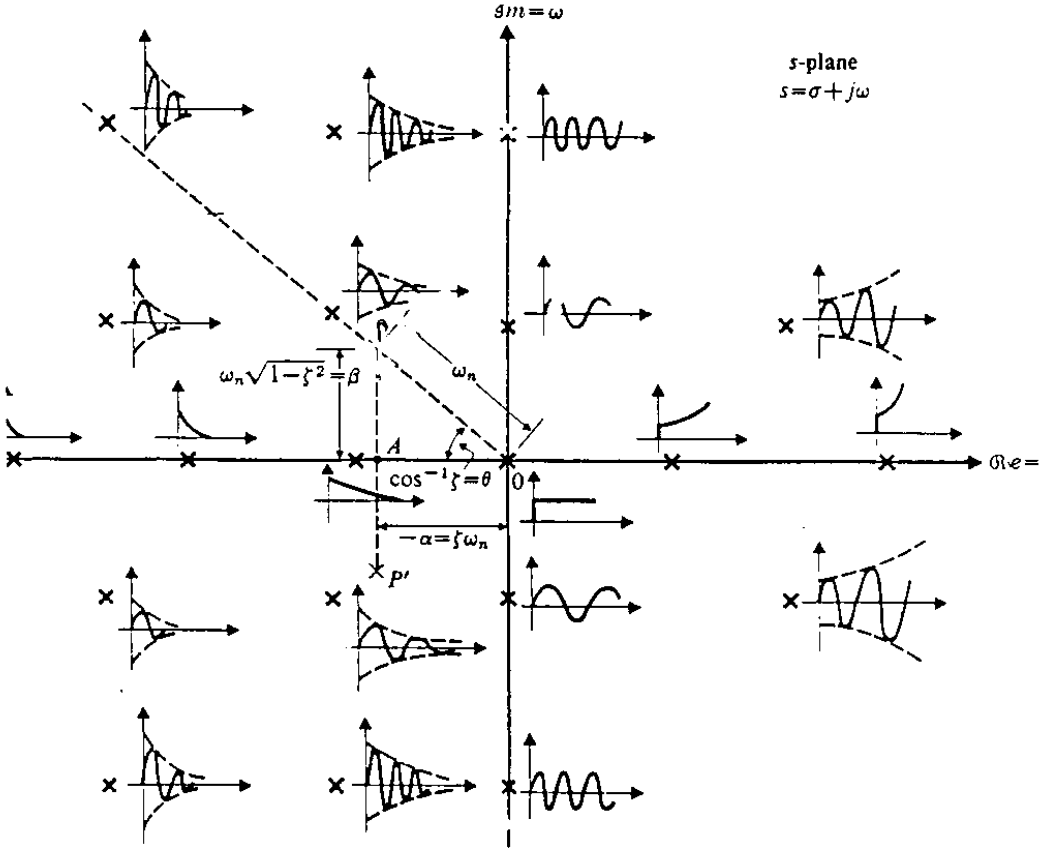
2023-2024

ÍNDICE

1. Polos y ceros de una función de transferencia con MATLAB

En las clases de teoría hemos visto que la respuesta temporal y(t) de un sistema de control depende esencialmente de la posición en el plano complejo (plano s) de los **polos (raíces del denominador)** de la función de transferencia en lazo cerrado $G_{LC}(s)$ y del tipo de entrada r(t) del sistema. Los ceros (raíces del denominador) también influyen en dicha respuesta, aunque en menor medida, ya que sólo afectan a los coeficientes de cada término en el dominio del tiempo.

En resumen, en la relación entre la respuesta transitoria y la posición de los polos viene dada por:



Existen diferentes maneras de extraer numéricamente los polos y los ceros de una función de transferencia G(s) mediante MATLAB. Los comandos más habituales son:

>> **polos = pole (G)**

>> **ceros = zero (G)**

Para dibujar el mapa de polos y ceros de un sistema de control en el plano complejo se utiliza el comando **pzmap(G)**. Al ejecutarlo, los polos aparecen representados con cruces (**x**) y los ceros con círculos (**o**).

- **pzmap (G1,G2,…)** dibuja los polos y ceros de varias funciones de transferencia en el mismo plan complejo, utilizando diferentes colores.

Con **pzmap** también se pueden extraer los valores numéricos de polos y ceros, agrupados en vectores columna, sin más que ejecutar el comando "*pzmap*" utilizando argumentos de salida de la siguiente manera:

>> [polos, ceros] = pzmap (G)

Ejercicio práctico 1. Extracción y dibujo de polos y ceros de funciones de transferencia

1. Extrae los valores numéricos de los polos y ceros de las siguientes funciones de transferencia:

$$G_1(s) = \frac{1}{s^2+4} \qquad G_2(s) = \frac{1}{s^4+4s^3+6s^2+4s+1}$$

$$G_3(s) = \frac{s^2+2s+1}{s^2-s} \qquad G_4(s) = \frac{s^2-1}{s^6+5s^5+11s^4+25s^3+34s^2+20s+24}$$

2. Para cada una de las funciones de transferencia, realiza una gráfica del plano complejo donde aparezcan dichos polos y ceros.

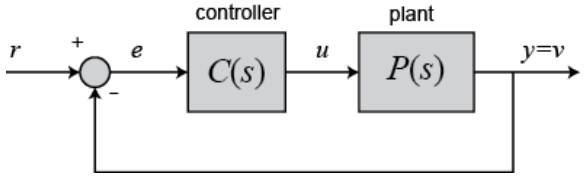
Nota: resuelve los ejercicios prácticos utilizando scripts (.m) o LiveScripts (.mlx) de MATLAB.

Ejercicio práctico 2. Polos y ceros en lazo abierto y en lazo cerrado en función de K

1. Realiza una gráfica del plano complejo con los polos y ceros de P(s):

$$P(s) = \frac{s+1}{(s-1)(s+2)(s+4)}$$

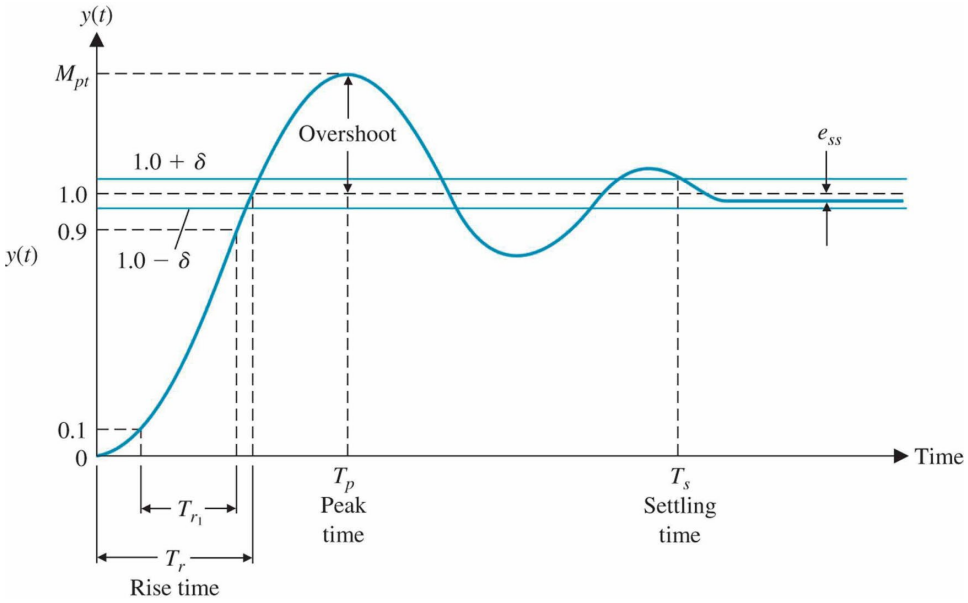
2. Realiza ahora una <u>única gráfica</u> de los polos y ceros de la función de transferencia en lazo cerrado $G_{LC}(s)$ del sistema de la figura, con la misma P(s) y con C(s) = K, para tres valores de la constante K, K=1, K=3 y K=10.



¿Influye el valor de K en la posición de los polos, de los ceros o de ambos en el sistema en lazo cerrado? Justifica tu respuesta

2. Estudio de la respuesta transitoria de sistemas de control

En este apartado vamos a aprender los comandos del *Control System Toolbox* de MATLAB que nos permiten conocer la respuesta temporal de un sistema de control y calcular los parámetros característicos del transitorio, que se han visto en teoría y que quedan resumidos en la siguiente gráfica:



2.1 Respuesta temporal a entradas impulso y escalón unitarios

El comando "*impulse*" nos permite visualizar gráficamente la respuesta de un sistema de control definido por una función de transferencia G(s) a una entrada impulso unitario, mientras que el comando "*step*" hace lo propio para una entrada escalón unitario:

>> impulse (G)

>> step (G)

MATLAB decide automáticamente la duración del intervalo de tiempo a mostrar, comenzando siempre desde t = 0. Si se quiere controlar la duración del intervalo de tiempo de interés para la visualización, su valor final debe aparecer como un argumento de entrada. Por ejemplo:

>> step (G, 20) *% Respuesta al escalón para un intervalo desde t = 0 s hasta t = 20 s.*

También se puede forzar un intervalo específico, con un tiempo inicial, final y paso temporal determinados, mediante:

>> **intervalo = 10:0.1:20;**

>> step (G, intervalo) % *Respuesta al escalón desde t = 10 s hasta t = 20 s en pasos de 0.1 s.*

En todos los casos anteriores se puede almacenar tanto el vector temporal como el vector de salida tanto de "*impulse*" como de "*step*" incluyendo argumentos de salida de la siguiente manera:

>> [y, t] = step (G) *% Almacenamiento de la curva de salida en los vectores y,t.*

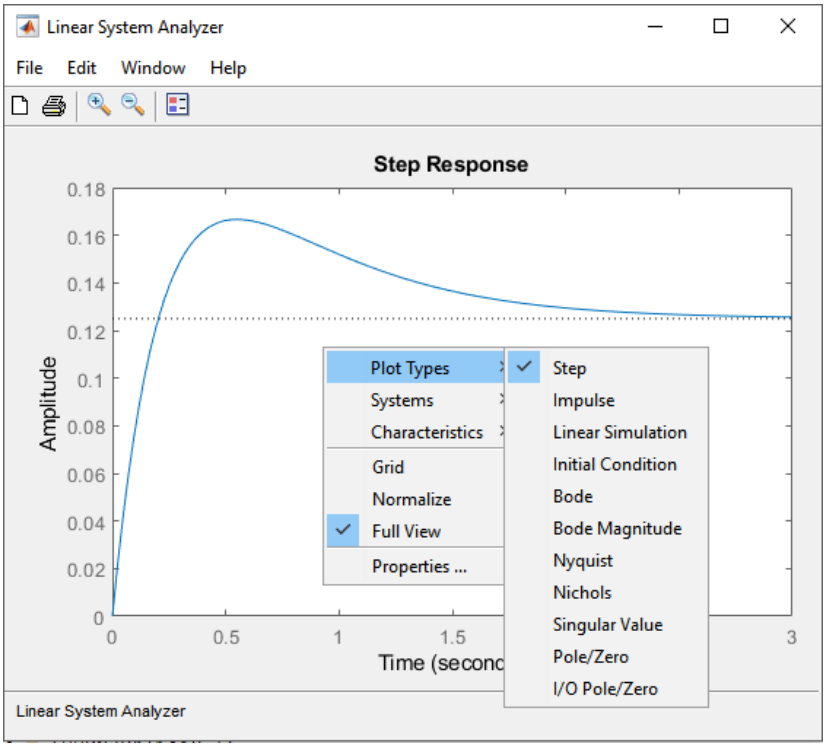
Por último, indicar que tanto "*impulse*" como "*step*" admiten varias funciones de transferencia como argumentos. Por ejemplo:

>> impulse (G1, G2, G3) *% Respuesta al impulso unitario de G1, G2 y G3*

Ejercicio práctico 3. Respuesta temporal de sistemas de primer orden

1. Construye cinco funciones de transferencia de primer orden situando sus polos en s = -0.5, s=-1, s = -3, s = -5 y s = -10 respectivamente.
2. Realiza una gráfica que incluya la respuesta al impulso unitario de todas ellas hasta un valor final de t = 5 s. Incluye la leyenda. Justifica el resultado obtenido.
3. Realiza ahora una gráfica que incluya la respuesta al escalón unitario de todas ellas hasta un valor final de t = 10 s. Incluye la leyenda. Justifica el resultado obtenido.
4. Modifica la ganancia estática de cada una de las funciones anteriores para la respuesta al escalón de todas ellas tenga el mismo valor final igual a 1.

Una forma alternativa de acceder a la gráfica tanto al mapa de polos y ceros como a la respuesta temporal de un sistema es a través del "*Linear System Analyzer*" de MATLAB:



Para acceder al "*Linear System Analyzer*" de MATLAB se debe usar el siguiente comando:

>> ltiview (G)

Una vez abierta la ventana del analizador, se puede modificar la visualización mostrada por defecto haciendo clic con el botón derecho del ratón, y cambiando la selección en el menú "*Plot Types*".

Ejercicio práctico 4. Respuesta temporal de sistemas de segundo orden

1. Construye cuatro funciones de transferencia de segundo orden de tal manera que todas tengan una K = 1, una frecuencia natural de 2 rad/s y una coeficiente de amortiguamiento igual a 0, 0.25, 1 y 5, respectivamente.
2. Extrae el valor numérico y realiza una única gráfica con la posición de los polos y ceros de las cuatro funciones de transferencia.
3. Realiza una gráfica que incluya la respuesta al escalón unitario de todas ellas hasta un valor final que permita que todas las respuestas se vean con comodidad. Incluye la leyenda. Indica el *tipo* de cada una de las respuestas temporales (*oscilatoria, subamortiguada, con amortiguamiento crítico o sobreamortiguada*).
4. Calcula manualmente la posición de los polos para cada uno de los tipos de respuesta temporal al escalón, comprueba tus resultados y justifica las respuestas.

   Nota: mediante el comando **ord2()** también se pueden definir funciones de transferencia de segundo orden. Consulta su sintaxis mediante **doc ord2**.

2.2 Extracción de parámetros característicos de la respuesta temporal

Existen dos opciones principales para extraer los parámetros característicos que definen la respuesta temporal a una entrada escalón unitario, que son:

- Tiempo de asentamiento o establecimiento (*settling time*, t_s).
- Tiempo de subida (*rise time*, t_r)
- Valor final del estacionario, $y(\infty)$

En caso de respuestas subamortiguadas, a los anteriores se añaden:

- Sobreelongación (*overshoot*, M_p)
- Tiempo de pico (*peak time*, t_p)

Para acceder a dichos datos numéricamente debemos hacer uso del comando "*stepinfo*" de MATLAB.

>> respuesta_G = stepinfo (G)

De esa manera, la salida del comando se guarda en la variable "*respuesta_G*", que es de tipo "*struct*" (*structure array* o array de estructura). Un **array de estructura** es un tipo de datos de MATLAB que agrupa datos relacionados mediante contenedores de datos denominados **campos**. Cada campo puede contener cualquier tipo de datos.

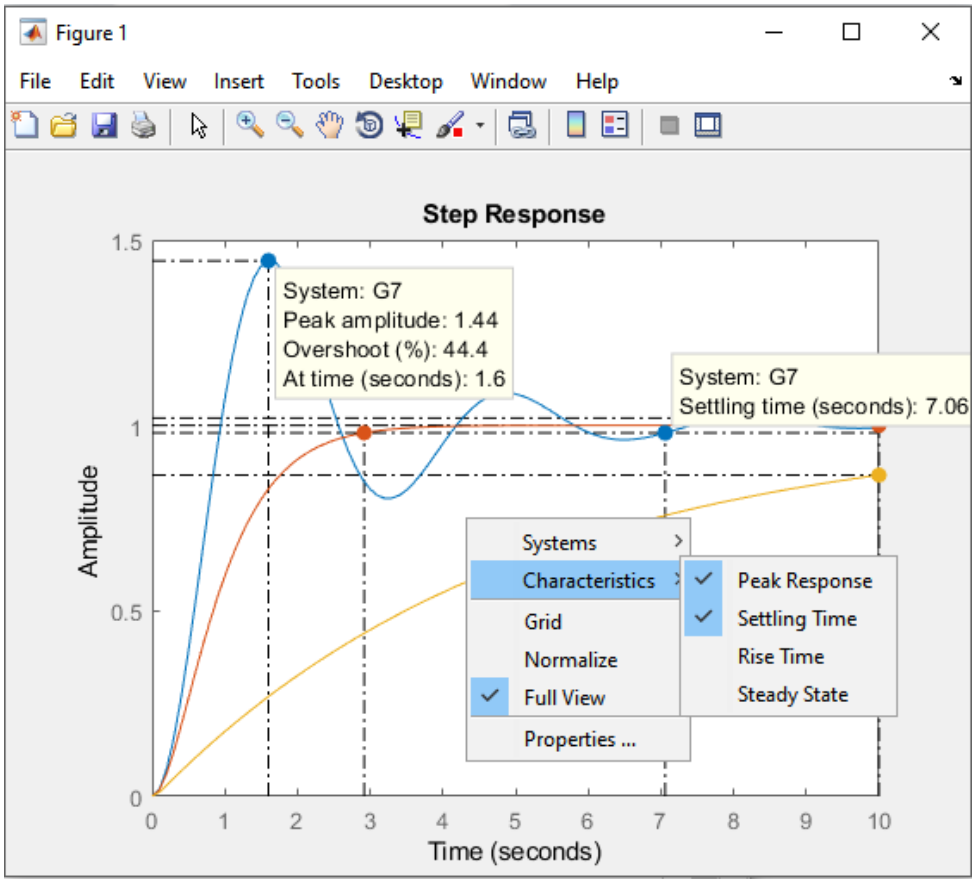
Para acceder a los datos almacenados en un campo de un array de estructura se debe utilizar la notación de puntos similar a la usada en programación orientada a objetos, es decir:

variable = structName.fieldName

Se puede llamar al comando stepinfo utilizando los argumentos '*SettlingTimeThreshold*' y/o '*RiseTimeThreshold*' que permiten modificar la definición del tiempo de establecimiento o del tiempo de subida, respectivamente. Por ejemplo, para calcular el tiempo de establecimiento en una banda del 0.5% y el tiempo de subida del 5 al 95% haríamos:

>> stepinfo(G, 'SettlingTimeThreshold', 0.005, 'RiseTimeThreshold', [0.05 0.95])

Aparte del comando "*stepinfo*", los parámetros característicos de la respuesta temporal de un sistema se pueden extraer de la gráfica de la respuesta a un escalón unitario o del "*Linear System Analyzer*", haciendo clic con el botón derecho del ratón sobre la gráfica, como se observa en la siguiente figura:



Ejercicio práctico 5. Extracción de los parámetros de la respuesta temporal

1. Extrae los valores numéricos del tiempo de subida y del tiempo de asentamiento para los sistemas de primer orden del ejercicio práctico 3, guardándolos en variables independientes. Comprueba dichos valores con los extraídos de la gráfica de la respuesta al escalón unitario
2. Repite el apartado anterior para las cuatro funciones de transferencia de sistemas de segundo orden especificadas en el ejercicio práctico 4.
3. Contrasta algunos tiempos de asentamiento con los valores teóricos esperados.

Ejercicio práctico 6. Diseño de sistemas de control

1. Utilizando comandos de control de flujo (*for, while, if*) realiza un script de MATLAB que calcule el valor que debe tener el polo de un sistema de primer orden para que su tiempo de subida (del 10 al 90%) sea igual a 1 ms.
2. Repite el apartado anterior para extraer la frecuencia natural de un sistema de segundo orden con respuesta críticamente amortiguada que permite que el tiempo de asentamiento del sistema sea igual a 10 ms.

2.3 Respuesta a otro tipo de entradas: rampa unitaria, aceleración unitaria, etc.

Para estudiar la respuesta de un sistema de control a una entrada de cualquier otro tipo (rampa unitaria, aceleración unitaria, suma de diferentes entradas, etc.) tenemos a nuestra disposición un comando muy importante, denominado "*lsim*". La sintaxis más habitual de "*lsim*" es la siguiente:

>> lsim (G, entrada, tiempo)

Donde:

1. "tiempo" es un vector que debe contener el intervalo de valores temporales equiespaciados para la simulación, definido por ejemplo de la forma:

   >> tiempo = 0 : dt : Tfinal

2. "entrada" es un vector que debe contener los valores de la señal de entrada para cada uno de los datos temporales del vector tiempo. Por ejemplo, para una señal de entrada de tipo aceleración unitaria, tendríamos:

   >> aceleracion = tiempo.^2

Por último, comentar que "*lsim*":

- Permite simular la respuesta de varios sistemas de control simultáneamente (en la misma gráfica).
- Se puede extraer el valor numérico de la respuesta especificando argumentos de salida de la misma manera que se vio con "*step*".

Ejercicio práctico 7. Respuesta de sistemas de 1º orden a entradas rampa y aceleración

1. Realiza una gráfica con la respuesta temporal de los sistemas de primer orden del ejercicio práctico 3 (una vez ajustadas las K) a una entrada rampa unitaria.
2. Repite el apartado anterior para una rampa de pendiente igual a 3.
3. Realiza ahora una gráfica con la respuesta temporal de dichos sistemas a una entrada aceleración unitaria

Ejercicio práctico 8. Respuesta de sistemas de 2º orden a entradas rampa y aceleración

1. Repite los tres puntos del apartado anterior para las cuatro funciones de transferencia de sistemas de segundo orden especificadas en el ejercicio práctico 4.

3. Respuesta transitoria de sistemas de orden superior

Hasta ahora se ha estudiado la respuesta temporal de sistemas de primer y segundo orden (es decir, con uno o dos polos en lazo cerrado y sin ceros). Bajo determinadas condiciones, los sistemas de orden superior (con tres o más polos) y/o los sistemas con ceros **se comportan como un sistema de primer o de segundo orden sin ceros**, donde su dinámica viene determinada por sólo uno o dos polos, denominados "**polo o polos dominantes"**, que son los que encuentran más cerca del eje imaginario (polos lentos).

En este caso se suele hablar de "aproximación mediante polos dominantes", y la respuesta temporal de dicho sistema se puede aproximar por la de un sistema de primer o de segundo orden, sin ceros, sólo con el polo o los dos polos dominantes.

3.1 Respuesta de un sistema de tercer orden. Polos dominantes

¿Qué condiciones se tienen que dar en un sistema de tercer orden para que se pueda realizar la aproximación de su respuesta temporal mediante los polos dominantes?

Vamos a estudiarlo mediante un ejercicio práctico:

Ejercicio práctico 9. Respuesta de sistemas de 3º orden. Polos dominantes

1. Construye una función de transferencia de primer orden G1(s) donde la posición de su único polo sea configurable mediante la variable P (s = -P).

$$G_1(s) = \frac{1}{(1 + \frac{s}{P})} = \frac{P}{(s + P)}$$

2. Construye la siguiente función de transferencia de segundo orden G2(s). Extrae la posición (fija) de sus dos polos:

$$G_2(s) = \frac{10}{s^2 + 2s + 10}$$

3. Define una nueva función de transferencia de tercer orden, G3(s), como el producto (cascada) de G1·G2. Realiza dos gráficas, por un lado una con posición de los polos de G1 y G2 (que serán los polos de G3) y por otro la respuesta al escalón de las tres funciones de transferencia, G1, G2 y G3. Activa la leyenda de la gráfica.

4. Da valores a P desde 0,1 hasta 20 y relaciona la respuesta al escalón de G3 con las respuestas al escalón de G1 y G2, a través de la posición de sus polos. ¿Qué polos dominantes tiene G3 en cada caso? Razona tu respuesta

5. ¿A partir de qué valor de P podrías decir que la respuesta de G3 es equivalente a la de G2, es decir, que el sistema de tercer orden se comporta como si tuviera sólo dos polos dominantes (los de G2)? Coteja tu respuesta en la bibliografía de la asignatura.

Ejercicio práctico 10. Respuesta de sistemas de 4º orden

1. En base al ejercicio anterior, añade a la función G3(s) del ejercicio anterior otro polo simple en posición s = -P2, construyendo una nueva función de 4º orden G4 = G1·G1b*G2 y comprueba si tus conclusiones del ejercicio anterior siguen siendo válidas para un sistema de 4º orden:

$$G_{1b}(s) = \frac{1}{(1 + \frac{s}{P_2})} = \frac{P_2}{(s + P_2)}$$

4. Precisión y error en estado estacionario de un sistema de control

4.1 Error de posición en estado estacionario

Para calcular el error verdadero en estado estacionario de un sistema de control utilizando MATLAB es necesario:

a) Comprobar que el sistema es estable. El concepto de error en estado estacionario sólo tiene sentido para que tienen una respuesta temporal que se estabiliza, es decir, para sistemas con todos sus polos en el semiplano complejo negativo.

b) Calcular la diferencia entre la entrada del sistema y la salida en estado estacionario, en términos absolutos o relativos

Por ejemplo, para calcular el error verdadero de posición en estado estacionario ante una entrada escalón de una amplitud determinada, para un sistema cuya función de transferencia en lazo cerrado viene dada por GLC, debemos hacer:

>> Amplitud = 2; *% Amplitud (configurable) del escalón*

>> [y,t] = step (Amplitud * GLC, t_final) *% Almacenamos respuesta hasta t_final en el vector y*

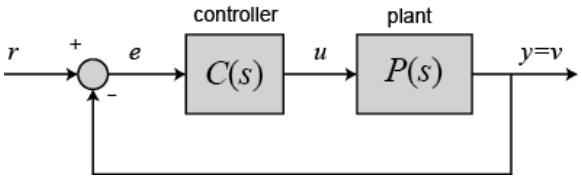
>> ess_pos_abs = Amplitud – y(end) *%Error absoluto, con su signo*

>> ess_pos_rel = 100*(Amplitud –y(end)) / Amplitud *% Error relativo, en porcentaje, con signo*

Para obtener valores de los errores absoluto y relativo precisos, el tiempo t_final de evaluación del escalón debe ser sensiblemente superior al tiempo de asentamiento (comienzo del estacionario) del sistema.

Ejercicio práctico 11. Precisión y error de posición en estado estacionario

1. Construye el sistema en lazo cerrado de la figura, donde C(s) es igual a 1 y P(s) es un sistema sin ceros y con dos polos en s= -3 y s= -1



2. ¿De qué TIPO es el sistema? Calcula su constante de error de posición y el error verdadero de posición en estado estacionario con las expresiones vistas en teoría.
3. Comprueba dichos cálculos extrayendo los errores de posición absoluto y relativo utilizando MATLAB, mediante un escalón de amplitud igual a 2.
4. Repite el ejercicio asumiendo que ahora C(s) es igual a 50. ¿Ha cambiado el error de posición? ¿Se ha visto afectada la respuesta transitoria?

4.2 Tipos de sistemas frente al error. Error en estado estacionario de velocidad

En las clases de teoría hemos visto que, con respecto al error en estado estacionario, los sistemas de control se pueden definir por su TIPO, que viene dado por el número de polos en el origen de su función de transferencia en lazo abierto G(s)H(s) (error de control) o G(s) error verdadero:

$$\lim_{s\to 0} G(s).H(s) = \lim_{s\to 0} \frac{K.\prod_{i=1}^{z}\left(\frac{s}{z_1}+1\right)}{s^n.\prod_{j=1}^{p}\left(\frac{s}{z_p}+1\right)} = \lim_{s\to 0}\frac{K}{s^n}$$

Valor de "n" = Tipo del sistema

El error verdadero del sistema depende de la entrada según su tipo, como se resume en la siguiente tabla:

	Constantes			Error para diferentes entradas		
Tipo	k_p	k_v	k_a	Salto	Rampa	Parábola
0	K	0	0	M/(1+K)	∞	∞
1	∞	K	0	0	M/K	∞
2	∞	∞	K	0	0	M/K

Ejercicio práctico 12. Aumento del tipo de sistema frente al error.

1. Repite el ejercicio 11 introduciendo en C(s) las modificaciones necesarias que hagan aumentar en una unidad el TIPO del sistema frente al error.
2. Vuelve a calcular el error de posición con MATLAB y justifica el resultado obtenido

Ejercicio práctico 13. Precisión y error de velocidad en estado estacionario

1. Calcula ahora la constante de precisión de velocidad y el error verdadero en estado estacionario de velocidad del sistema que has definido ejercicio 12, mediante las expresiones teóricas.
2. Utilizando el comando "lsim" y la metodología de cálculo aprendida para los errores de posición, determina el error en estado estacionario de velocidad utilizando MATLAB
3. Realiza la gráfica de la respuesta temporal del sistema ante una entrada de velocidad y explica los resultados obtenidos.
4. ¿En qué unidades crees que debe darse el error de velocidad absoluto?

Universidad
Rey Juan Carlos

Área de Tecnología Electrónica

Práctica 4

Análisis y diseño de sistemas de control mediante el Lugar de las Raíces con MATLAB

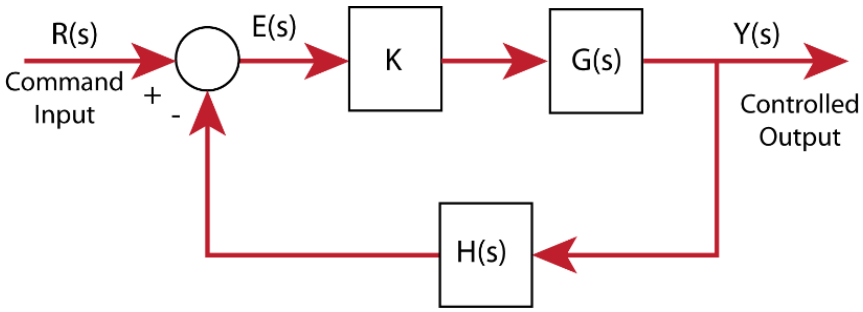
2023-2024

ÍNDICE

1. Gráfica del Lugar de las Raíces (LDR) con MATLAB

Recordemos que el **método o técnica del lugar de las raíces** (LDR) es un procedimiento gráfico diseñado por W.R. Evans en 1948 para representar los polos en lazo cerrado de un sistema de control con realimentación negativa para todos los valores de un parámetro. Dicho parámetro es, en general, la ganancia K de lazo, que varía de 0 a ∞.

Dado el diagrama de bloques del sistema en lazo cerrado genérico:



Cuya función de transferencia en lazo cerrado es: $$G_{LC}(s) = \frac{K \cdot G(s)}{1 + K \cdot G(s) \cdot H(s)}$$

Por ello, el lugar de las raíces del sistema es la representación gráfica (en el plano complejo) de las raíces de la ecuación característica, es decir, del denominador de la función de transferencia en lazo cerrado del sistema, cuando K va variando desde 0 hasta ∞:

$$1 + \frac{K(s + z_1)(s + z_2) \cdots (s + z_m)}{(s + p_1)(s + p_2) \cdots (s + p_n)} = 0$$

En la ecuación característica anterior, los valores p_i son los polos de la función de transferencia en lazo abierto G(s)H(s) y los valores z_i son los ceros de G(s)H(s). Recordemos que, según hemos visto en teoría, la representación del lugar de las raíces parte de los polos en lazo abierto p_i para k=0 (o del infinito) y llega a los ceros en lazo abierto z_i (o al infinito) para valores de k muy elevados.

En MATLAB, el comando del *Control Systems Toolbox* para dibujar el lugar de las raíces se denomina "rlocus". Rlocus determina de forma automática los valores de ganancias K necesarios para realizar el gráfico completo del LDR. Su sintaxis básica es la siguiente:

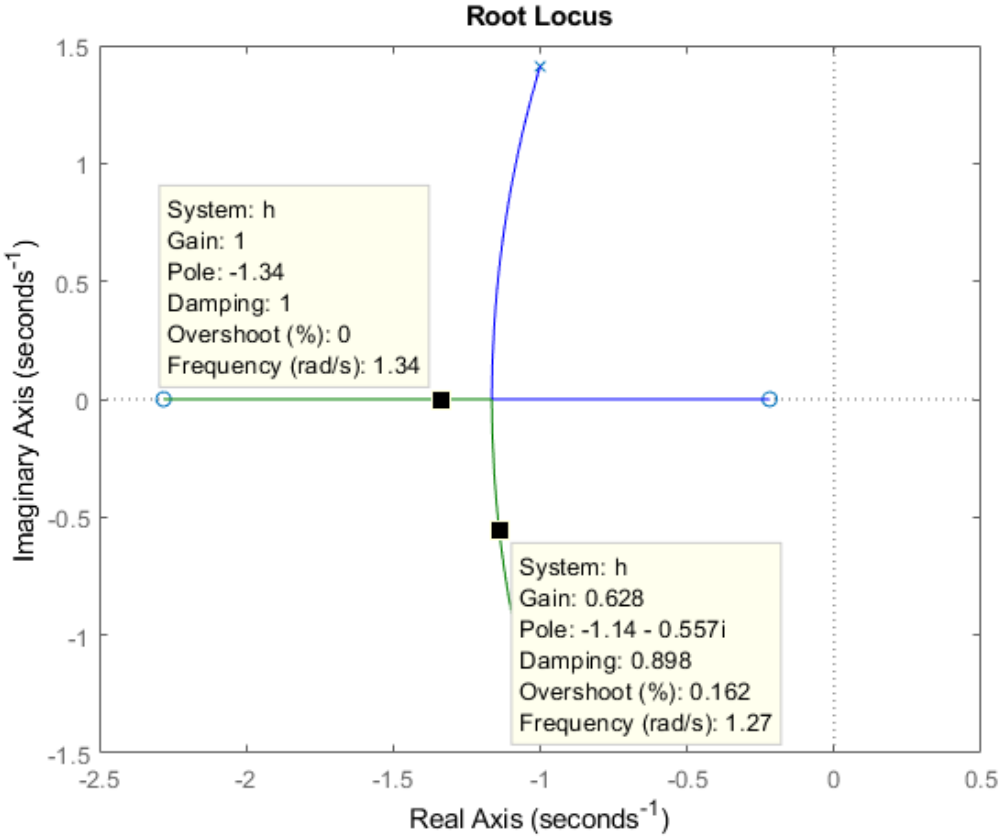
>> rlocus (G·H)

Donde G y H deben ser funciones de transferencias definidas con los comandos *tf* o *zpk* de MATLAB. Cada una de las ramas del LDR aparece de un color diferente.

Rlocus también acepta múltiples sistemas de control como parámetros de entrada. En ese caso, realizará la gráfica del LDR de todos los sistemas en una única figura:

>> rlocus (G1·H1, G2·H2, G3·H3)

Una vez realizada la gráfica del LDR, se puede hacer clic sobre la misma (una o varias veces) y MATLAB mostrará un "*data marker*" mostrando tanto el valor de la ganancia que se corresponde con dicho punto como la posición exacta del polo en lazo cerrado, factor de amortiguamiento, sobreelongación y frecuencia asociada a dicho punto del plano complejo:



Ejercicio práctico 1: Dibujo básico del lugar de las raíces con MATLAB. Extracción de K

1. Crea un script de MATLAB y dibuja el lugar de las raíces de al menos 6 sistemas de control extraídos de los ejercicios 4.2 al 4.13 de la relación de ejercicios del Tema 4.
2. Para todas las figuras anteriores, determina gráficamente el punto del LDR en el que la ganancia toma el valor de K =1
3. Si en la gráfica aparecen puntos de ruptura o reencuentro, halla el valor de K asociado a los mismos. Comprueba alguno de ellos realizando los cálculos manualmente.

Si necesitamos calcular numéricamente el valor de los polos en lazo cerrado para un valor determinado de K, debemos incluir dicho valor como segundo argumento de *rlocus* y guardarlo en una variable de salida (por ejemplo *PolosLC*):

>> PolosLC = rlocus (G·H, 3) *% Valor numérico de todos los polos en LC para K = 3*

K también se puede definir como un *vector* conteniendo un conjunto o intervalo de valores numéricos, en cuyo caso el comportamiento del comando *rlocus* será el siguiente:

>> K = 1:1:10; *% Vector de valores de K de 1 a 10 (en unidades enteras)*

>> PolosLC = rlocus (G·H, K) *% Matriz de polos en LC para cada uno de los valores de K*

>> rlocus (G·H, K) *% Gráfica del lugar de las raíces sólo para K entre 1 y 10.*

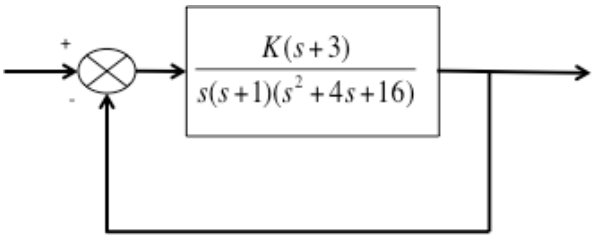
El control de los valores máximos y mínimos de los ejes Real e Imaginario en la figura del LDR se puede realizar mediante el comando "*axis*", ya visto en prácticas anteriores.

>> axis([xmin xmax ymin ymax])

También se pueden modificar otras opciones de visualización más complejas mediante el comando "*rlocusplot*". Si tienes interés en conocerlas, consulta la ayuda de dicho comando.

Ejercicio práctico 2: Lugar de las raíces con MATLAB para un vector de valores de K.

1. Dibuja la gráfica del LDR del sistema de la figura:



2. Modifica los ejes de la gráfica para visualizarla entre -6 y 1 (en el eje real) y -6 y 6 (en el eje imaginario)
3. Calcula el valor de K para el que el sistema sea hace inestable y el valor de la frecuencia de oscilación de la respuesta del sistema para esa K en el límite de la estabilidad.

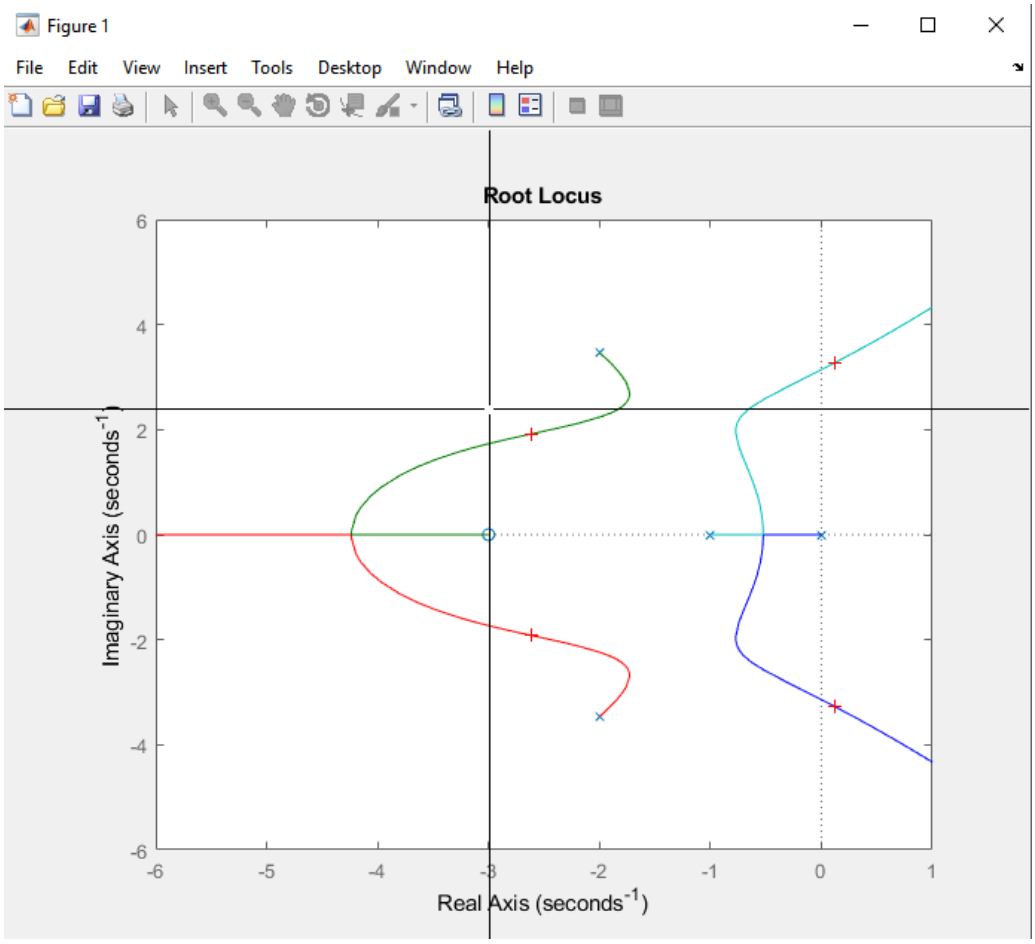
1.1 Cálculo gráfico de todos los polos en lazo cerrado mediante "rlocfind"

Uno de los problemas de la representación gráfica del LDR con *rlocus* es que, al hacer clic sobre uno de los puntos del gráfico, sólo se obtiene el valor del polo correspondiente a dichas coordenadas para el valor de K correspondiente a dicho punto. Si el sistema tiene más ramas, no aparecen los valores del resto de polos en LC asociados con dicho valor de K.

Con la orden **rlocfind,** que debe seguir a la orden rlocus, se puede calcular gráficamente el valor de la ganancia en un punto y la posición de TODOS los polos en lazo cerrado correspondientes a ese punto del lugar de las raíces.

Para ello, debemos mover la cruz que aparece en la gráfica y hacer clic en el punto deseado. El valor de la ganancia se guardará en K y el valor de los polos en la variable *PolosLC*:

>> [K,PolosLC] = rlocfind(G·H)

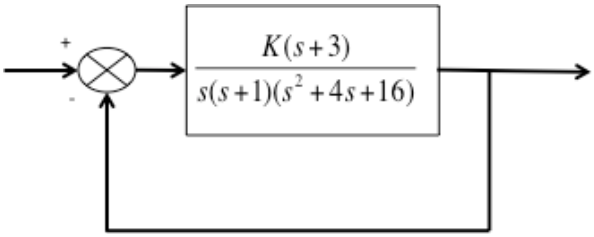


Por último, el comando "*rlocfind*" acepta un segundo argumento de entrada, un vector "P", en el que se pueden especificar las coordenadas (valores complejos) de una serie de polos. Al ejecutar "*rlocfind*" el comando devolverá un vector de ganancias K para los que existe un polo en lazo cerrado de G·H próximo a los valores que aparecen en por P. Además, los valores exactos de dichos polos aparecerán en la matriz *polosLC*

>> [K,PolosLC] = rlocfind(G·H,P)

Ejercicio práctico 3: Uso de rlocfind para ampliar las capacidades de rlocus en MATLAB

Para el sistema de control en lazo cerrado del ejercicio anterior:



1. Encuentra gráficamente la posición de todos los polos en lazo cerrado del sistema para el límite de la estabilidad (sistema oscilante o no amortiguado).
2. Calcula mediante *rlocfind* el valor de K para el que uno de los polos está en s = -5. Indica si el sistema de control será estable en ese caso, razonando tu respuesta.

2. Diseño de sistemas mediante el Lugar de las Raíces

2.1 Lugar de las raíces con ξ constante y ω_n constante

En las clases de teoría ya se ha visto que en la gráfica del lugar de las raíces, las líneas con coeficiente de amortiguamiento ζ constante son radiales que pasan por el origen. El coeficiente de amortiguamiento determina la localización angular de los polos, mientras que la distancia del polo al origen determina la frecuencia natural no amortiguada ω_n del sistema**.** Por ello, el lugar de las raíces para ω_n constante son circunferencias con centro en el origen del plano complejo.

Para superponer sobre el LDR una o varias líneas con coeficiente de amortiguamiento constante ζ y una o varias circunferencias con ω_n constante se utiliza el comando ***sgrid*.**

La orden **sgrid** tiene como parámetros de entrada **ζ y ω_n**, pudiendo elegir representar uno o varios valores concretos de las dos variables (usando vectores). Si se desea eliminar todas las líneas de ζ o todas las circunferencias de ω_n se utilizan corchetes vacíos.

Ejercicio práctico 4: Uso de sgrid para dibujar líneas de coeficiente de amortiguamiento constante

1. Construye un sistema de control en lazo cerrado con H(s) = 1 y donde la función de transferencia en lazo abierto G(s) tenga dos polos situados en s = -1 y s = -3, y ningún cero.
2. Superpón al LDR las líneas de coeficientes de amortiguamiento **ζ= 0.3 y ζ= 0.7** y calcula el valor de K (controlador de tipo proporcional) que permite que los polos en lazo cerrado se correspondan con dichos coeficientes. Reajusta los límites de la gráfica si es necesario.

3. Extrae del LDR la posición de los polos en lazo cerrado (LC) para cada valor de K y la sobreelongación que tendrá la respuesta al escalón del sistema. Calcula el coeficiente de amortiguamiento (manualmente) utilizando la posición de los polos en LC, y comprueba con la expresión de Mp la estimación de sobreelongación extraída de la gráfica del LDR.
4. Conocida la posición de los polos en LC para cada valor de K, estima el valor del tiempo de asentamiento del sistema.
4. Realiza una gráfica de la respuesta al escalón del sistema en LC con ambos valores de K y comprueba en ambos casos los valores de sobreelongación y tiempo de asentamiento estimados mediante el LDR (recuerda, para la banda del 1.8%). ¿Coinciden?
5. ¿Con qué valor de K el sistema presenta un mayor error frente a la entrada escalón? ¿Por qué?

Ejercicio práctico 5: Uso de sgrid para dibujar circunferencias de ω_n constante

1. Construye ahora un sistema de control en lazo cerrado similar al del ejercicio anterior pero con G(s) teniendo dos polos situados en s = 0 y s = -3, y ningún cero.
2. Repite el ejercicio anterior y realiza una nueva gráfica del LDR para el mismo sistema, calculando ahora los tres valores de K que nos permiten intersectar con las circunferencias de las tres frecuencias naturales siguientes: ω_n=1, ω_n=1,5 y ω_n=3.
3. ¿Qué tipo de respuesta transitoria es esperable para cada uno de los valores de K? ¿Cuál de los tres será el sistema con un transitorio más largo? ¿Por qué?
4. Con respecto a la respuesta estacionaria, ¿alguno de los tres valores de K hará que el sistema sea inestable? ¿Se puede prever el error frente a la entrada escalón simplemente observando la gráfica del LDR?
5. Realiza en una misma gráfica la respuesta al escalón del sistema en LC para los tres valores de K hallados y comprueba las respuestas de los apartados 3 y 4.

2.2 Diseño de sistemas de control con ceros mediante el LDR

En el siguiente ejercicio práctico vamos a comprobar si las estimaciones extraídas del LDR (en concreto, la estimación de sobreelongación y de tiempo de asentamiento que se obtiene de los polos dominantes) son válidas en el caso de sistemas de control con que contienen ceros en la función de transferencia en lazo abierto.

Ejercicio práctico 6: Diseño de sistemas con ceros mediante el LDR

1. Construye un sistema de control en lazo cerrado donde la función en lazo abierto G(s) tenga dos polos situados en s = 0 y s = -1, y un cero situado en s = -3.
2. Dibuja la gráfica del lugar de las raíces del sistema. Extrae los valores de K que permiten intersectar con la línea de coeficiente de amortiguamiento constante **ζ= 0.7.** Anota en cada caso la posición de los polos en LC y la sobreelongación máxima asociada con dicho coeficiente de amortiguamiento.
3. Razona qué diferencias y qué similitudes esperas en la respuesta al escalón utilizando cada uno de los valores de K anteriores.
4. Representa la respuesta al escalón unitario del sistema completo para ambos valores de K y comprueba las afirmaciones que has hecho en el apartado 3. ¿Se cumplen las predicciones de sobreelongación máxima en ambos casos? ¿Por qué?
5. Resitúa el cero en s = +3 y vuelve a realizar la gráfica de la respuesta al escalón del sistema, para cualquier valor de K. Justifica en resultado en base al nuevo LDR.

2.3 Diseño de sistemas de orden superior mediante el LDR

Ejercicio práctico 7: Máquina-herramienta con control en velocidad

Para controlar la velocidad de una máquina herramienta se dispone de un sistema formado por un actuador que acciona directamente (en cascada) el motor de la misma. **El sistema acepta órdenes de velocidad** (no de posición) y:

1. El modelo del actuador, según su hoja de características, es un **sistema de primer orden** con ganancia estática de valor 1.2 y constante de tiempo de 2.56 segundos.
2. El motor dispone de un sensor para medir la velocidad del mismo. Del modelo motor-sensor se ha obtenido un modelo empírico, dando como resultado un **sistema de segundo orden** con un valor de K de 0.1, y dos polos en las posiciones -0.3 y -0.1.

Construye las funciones de transferencia del actuador y del motor-sensor en MATLAB y:

a) Obtén la evolución de la salida (velocidad), ante una entrada escalón unitario, cuando el sistema se encuentra en lazo abierto. Calcula el error en estado estacionario. ¿Es un error de velocidad?

b) Para el sistema de control en lazo cerrado con realimentación unitaria, obtén la evolución de la salida (velocidad) ante una entrada escalón unitario, con una ganancia de lazo K unitaria. Determina de nuevo el error en estado estacionario y compara con el anterior. ¿Es este error nulo? ¿Por qué? ¿Cómo podría ser siempre nulo?

c) Dibuja el lugar de las raíces e indica si es posible aumentar el valor de la ganancia de lazo K al sistema para mejorar la respuesta en estado estacionario sin que éste se convierta en inestable. Justifica tu respuesta.

d) Representa la respuesta del sistema ante una entrada escalón unitario para dos valores de $K > 1$. Relaciona la frecuencia de oscilación y sobreelongación máxima de estas dos respuestas con la nueva posición de los polos dominantes en lazo cerrado según el lugar de las raíces. Calcula la posición de polo rápido del sistema para dichos valores de K. ¿Tiene alguna influencia en la respuesta temporal?

Universidad Rey Juan Carlos

Grado en Ingeniería de Robótica Software

Práctica 5

Diseño de controladores PID con MATLAB y RLTOOL

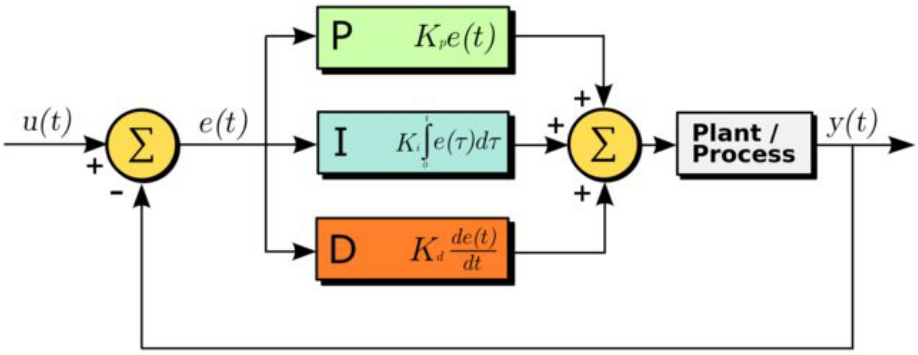
2023-2024

ÍNDICE

1. La familia de controladores tipo PID

Como se ha visto en la parte de teoría de la asignatura, un controlador PID es un tipo de regulador o compensador que aplica una señal de control u(t) a una planta o proceso, donde dicha señal de control u(t) es una combinación lineal (es decir, una suma) de acciones **proporcional, integral** y **derivativa** de la señal de error e(t).

Su esquema típico es el siguiente:



La familia de controladores tipo PID está formada por diferentes combinaciones de los elementos proporcional, integral y/o derivativo. Las configuraciones más importantes son:

- **Proporcional (P):** cuenta sólo con una ganancia positiva, K.
- **Proporcional-Derivativo (PD):** cuenta con una ganancia K y un cero cuya posición es configurable.
- **Proporcional-Integral (PI):** ganancia K, polo en el origen y cero en posición configurable.
- **Proporcional-Integral-Derivativo (PID):** ganancia K, polo en el origen y dos ceros en posiciones configurables.

Las funciones de transferencia de PD, PI y PID son las siguientes:

$$PI \quad G_c(s) = K\left(1 + \frac{1}{T_i s}\right)$$

$$PD \quad G_c(s) = K\left(1 + T_d s\right)$$

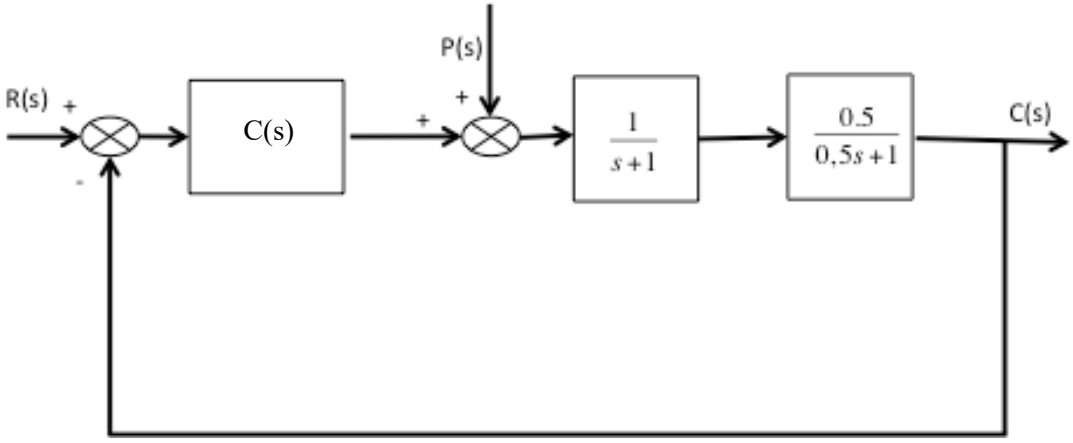
$$PID \quad G_c(s) = K\left(1 + T_d s + \frac{1}{T_i s}\right)$$

Ejercicio práctico 1: Posiciones de los polos y los ceros de los controladores PID

1. Extrae la posición de los ceros de los controladores PD, PI y PID en función de las constantes de tiempo integral (T_i) y derivativa (T_d), para el caso general.

Ejercicio práctico 2: Respuesta de sistemas con controladores PID frente a perturbaciones

La figura representa un sistema de control con una entrada R(s) y con una perturbación (señal no deseada) P(s) situada entre el controlador y la planta:



El controlador C(S) del sistema puede ser P, PI o PID. Se han obtenido por métodos empíricos los siguientes parámetros de tres controladores tipo PID:

- **Caso 1:** Controlador *P* (K = 2,35)
- **Caso 2:** Controlador *PI* (K = 0,235 T_i= 0,1s)
- **Caso 3:** Controlador *PID* (K = 2,55 T_i= 1,28s T_d=0,09)

Se requiere:

a.- Obtener la respuesta temporal c(t) del sistema ante una entrada r(t) escalón unitario considerando que no hay perturbación (p(t) = 0), comparando las respuestas del sistema sin controlador y con los tres controladores en la misma gráfica. Obtener en cada caso la posición de los polos en lazo cerrado del sistema y justificar el resultado obtenido.

b.- Considerando que la entrada r(t) permanece a cero (p(t) = 0), obtener cómo evoluciona la salida cuando se añade una perturbación p(t) de tipo escalón unitario con los tres controladores anteriores. ¿Con qué controlador se obtiene una mejor respuesta frente a la perturbación?

Ayuda: *Para resolver el apartado b) debes calcular la nueva función de transferencia que relaciona la salida C(s) con la entrada perturbación P(s), eliminando la entrada R(s) del sistema.*

2. La interfaz RLTOOL de MATLAB

MATLAB incorpora distintas interfaces de usuario aplicadas al control, entre las que cabe destacar *Sisotool* para el análisis y diseño de sistemas y *Simulink* para la simulación de modelos matemáticos mediante diagrama de bloques.

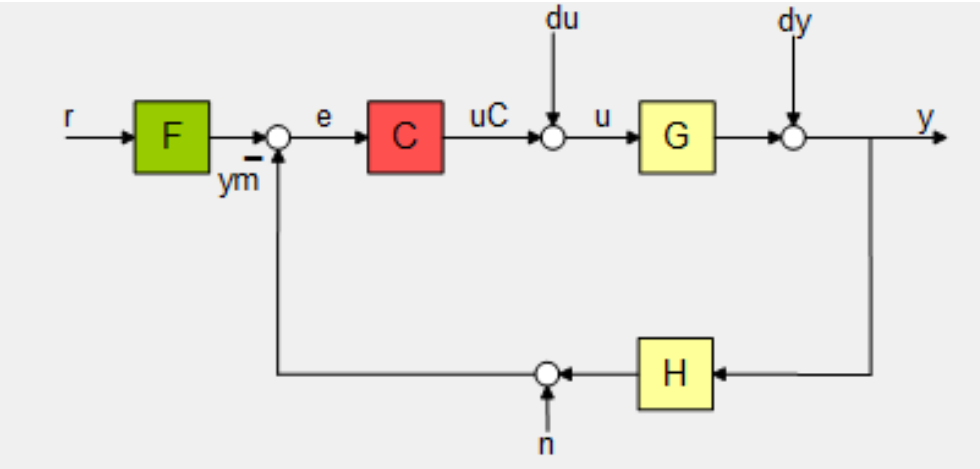
SISOTOOL (*Single Input Single Output Tool)*, permite el análisis de sistemas de control de una entrada y una salida (sistemas SISO), así como el diseño de compensadores de adelanto o retardo y de controladores tipo PID.

RLTOOL es una parte de la herramienta SISOTOOL que se centra el usar el Lugar de las Raíces como técnica principal de análisis y diseño de los sistemas. Proporciona una forma rápida, fácil y útil de diseñar controladores, y ver su influencia en el lugar de las raíces dibujado sobre el plano complejo. Con RLTOOL podemos añadir, mover y eliminar los polos y los ceros del controlador de forma rápida, cambiar su ganancia y ver los resultados de forma inmediata en la nueva localización de los polos del sistema en bucle cerrado y en la respuesta del sistema a la señal de entrada

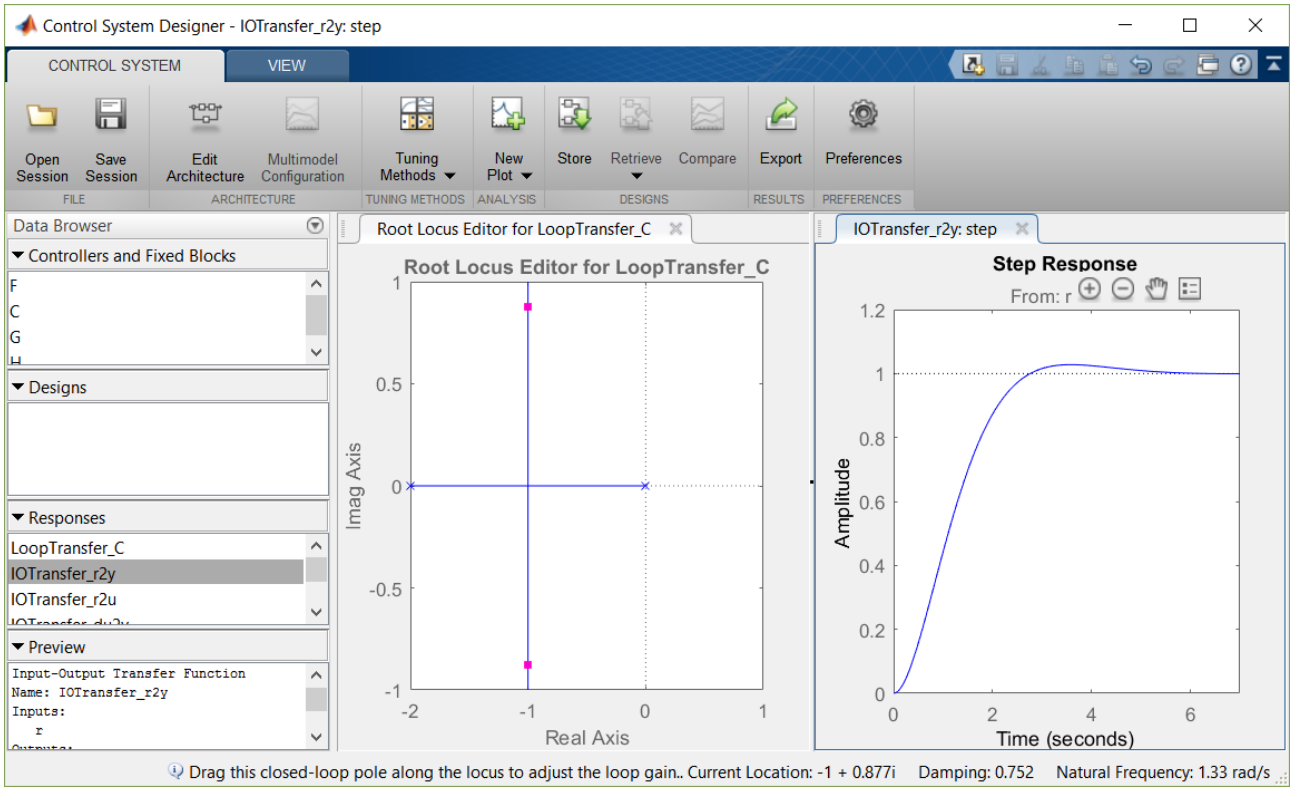
¿COMO ARRANCO RLTOOL?

RLTOOL puede ser invocado con una gran variedad de parámetros. Escribiendo "help rltool" la ventana de comandos de MATLAB podemos ver las posibles opciones:

- rltool(G), siendo G la función de transferencia de la planta, creada con zpk o tf.
- rltool(G,C), siendo C el controlador para el sistema en bucle cerrado con realimentación negativa.



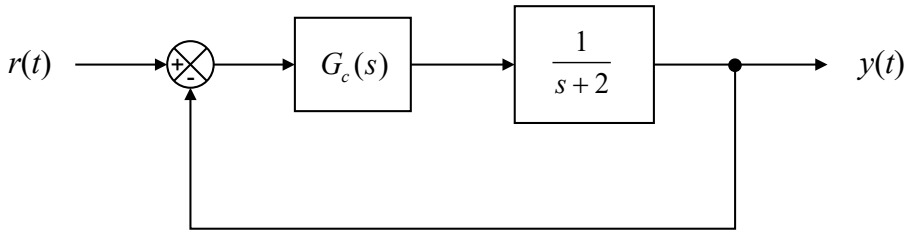
Una vez se ha ejecutado el comando aparecerá la interfaz de usuario (*Control System Designer*), a través de la cual se puede configurar y guiar todo el diseño y ajuste del sistema. En MATLAB R2016a tiene este aspecto:



Para descubrir todo el potencial de *RLTOOL* se va a abordar el proceso de ajuste y análisis de varios sistemas de control en los siguientes ejercicios:

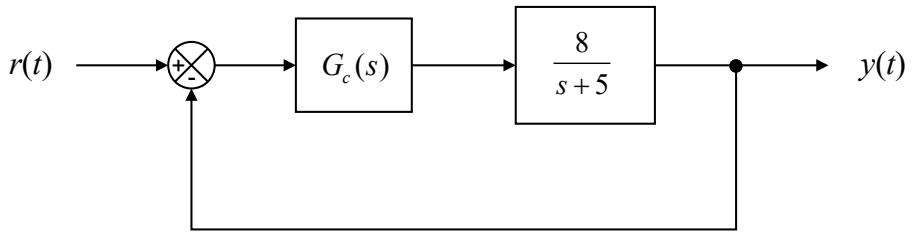
Ejercicio práctico 3: Utiliza la herramienta RLTOOL para averiguar el controlador más sencillo de tipo PID (P, PI, PD o PID, en ese orden) a colocar en el sistema de la figura de forma que dicho sistema tenga un error en estado estacionario frente a una entrada escalón exactamente igual a $e_{ss} = 30\%$.

Una vez averiguado el tipo de controlador, calcula los parámetros del mismo y comprueba el resultado:



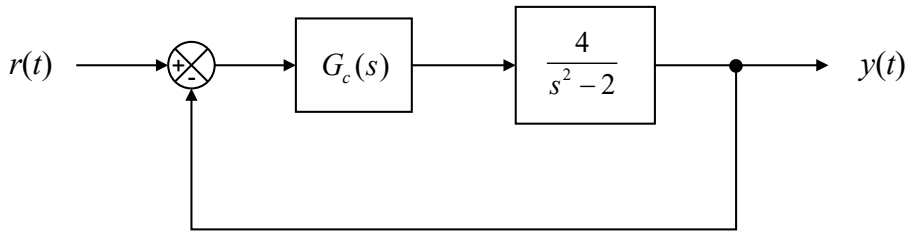
Ejercicio práctico 4: Utiliza la herramienta RLTOOL para averiguar el controlador más sencillo de tipo PID (P, PI, PD ó PID) a colocar en el sistema de la figura de forma que dicho sistema tenga una sobreelongación del $M_p = 10\%$ y un tiempo de asentamiento $t_s = 1$ s (para la banda del 2%)

Una vez averiguado el tipo de controlador, calcula los parámetros del mismo:



Ejercicio práctico 5: Utiliza la herramienta RLTOOL para averiguar el controlador más sencillo de tipo PID (P, PI, PD ó PID) a colocar en el sistema de la figura de forma que dicho sistema tenga ahora $M_p = 20\%$ y t_s (2%) = 1 s.

Una vez averiguado el tipo de controlador, calcula los parámetros del mismo:



Ejercicio práctico 6: Utiliza la herramienta RLTOOL para averiguar el controlador más sencillo de tipo PID (P, PI, PD ó PID) que es necesario incorporar como controlador en el sistema para el control de la velocidad de la máquina herramienta del ejercicio de la práctica del tema anterior, "Lugar de las Raíces".

Las especificaciones del sistema ante una respuesta escalón unitario que deberán cumplirse una vez elegido y configurado el controlador serán las siguientes:

$M_p < 15\%$ **$t_s < 30s$** **error de posición = 0**