

Manual de Prácticas

Biomecatrónica

Andrés Quintero Zea, PhD.

Universidad ELA

Manual de prácticas – Biomecatrónica

Andrés Quintero Zea, PhD. (andres.quintero27@eia.edu.co)

MATLAB®, Simulink® y Control Systems Toolbox™ son marcas comerciales de The MathWorks, Inc. The MathWorks no garantiza la exactitud del texto ni de los ejercicios de este libro. El uso o discusión de este libro sobre el software MATLAB®, Simulink® y Control Systems Toolbox™ o productos relacionados no constituye respaldo o patrocinio por parte de The MathWorks de un enfoque pedagógico particular o uso particular del software MATLAB®, Simulink® y Control Systems Toolbox™.

Este libro fue escrito por el autor en \LaTeX usando las fuentes tipográficas Myriad Pro para encabezados de sección, Cronos Pro para texto corrido, Minion Pro para ecuaciones matemáticas y Roboto Mono para fragmentos de código.

Copyright ©2023 Andrés Quintero Zea

Imagen de la portada creada por Inteligencia Artificial usando la herramienta [Microsoft Copilot](#)



Esta obra está bajo una Licencia Creative Commons
Atribución-NoComercial-CompartirIgual 4.0 Internacional

Resumen: <https://creativecommons.org/licenses/by-nc-sa/4.0/>

Licencia: <https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>

Prefacio

Bienvenido al “Manual de Prácticas – Biomecatrónica”. Este recurso ha sido diseñado específicamente para estudiantes de Ingeniería Biomédica de la Universidad EIA, con el fin de explorar los fundamentos del control analógico y aplicar estos conocimientos a la solución de problemas en el ámbito de la salud.

La ingeniería biomédica desempeña un papel crucial en la mejora de la atención médica mediante la aplicación de principios y técnicas de ingeniería a los desafíos en el campo de la medicina. Este manual se centra en la aplicación de conceptos de control analógico en situaciones relevantes para la ingeniería biomédica, como el diseño de sistemas de control para dispositivos médicos, monitoreo de señales fisiológicas y mejora de la precisión en instrumentación biomédica.

Este manual ha sido diseñado para proporcionar una experiencia de aprendizaje práctica y relevante para los estudiantes de ingeniería biomédica. A medida que te sumerjas en los ejercicios prácticos y estudios de caso, te alentamos a considerar el impacto positivo que la ingeniería biomédica puede tener en la salud y el bienestar de las personas.

Esperamos que este recurso sirva como un valioso compañero en tu trayectoria académica y profesional en el emocionante campo de la ingeniería biomédica. ¡Disfruta del viaje!

Andrés Quintero Zea, PhD.

Índice general

Prefacio	i
1. Introducción al <i>Control System Toolbox</i>	1
1.1. Introducción al <i>Control System Toolbox</i>	2
1.2. Creación y Manipulación de Sistemas Dinámicos	3
1.2.1. Función de Transferencia	3
1.2.2. Cero – Polo – Ganancia	5
1.2.3. Espacio de Estado	6
1.3. Análisis de Respuesta Transitoria y en Frecuencia	8
1.3.1. Respuesta en el tiempo	8
1.3.2. Respuesta en la frecuencia	11
1.4. Trabajo práctico	13
1.4.1. Prodecimiento	13
1.4.2. Evaluación	14
2. Modelos en función de transferencia	15
2.1. Funciones polinómicas básicas	16
2.1.1. Ejemplos	17
2.2. Análisis de funciones racionales en fracciones parciales	19
2.2.1. Ejemplos	20
2.3. Funciones de transferencia simbólicas	23
2.3.1. Ejemplos	24
2.4. Trabajo práctico	26
2.4.1. Procedimiento	26
2.4.2. Evaluación	27
3. Modelos en ecuaciones de estado	29
3.1. Ecuaciones de estado de sistemas no lineales	30
3.1.1. Ejemplos	30
3.2. Representación en espacio de estados (SSR)	33
3.2.1. Ejemplos	33
3.3. Retrato de fase de sistemas de segundo orden	35

3.3.1.	Ejemplos	35
3.4.	Trabajo práctico	37
3.4.1.	Procedimiento	37
3.4.2.	Evaluación	39

Índice de tablas

1.1.	Funciones para el análisis de respuesta temporal	9
1.3.	Funciones para el análisis de respuesta frecuencial	11
1.5.	Parámetros del modelo	14

Índice de figuras

1.1.	Propiedades de un objeto tipo <code>tf</code>	4
1.2.	Propiedades de un objeto tipo <code>zpk</code>	6
1.3.	Propiedades de un objeto tipo <code>ss</code>	8
1.4.	Respuestas del sistema	9
1.5.	Respuesta del sistema usando <code>lsim</code>	11
1.6.	Gráficas para análisis de respuesta en frecuencia	12
1.7.	Sistema de suspensión utilizado en el análisis	13
3.1.	Simulación del oscilador de Van der Pol	31
3.2.	Simulación del atractor de Lorenz	33
3.3.	Respuesta del sistema usando simulación numérica	34
3.4.	Respuesta del sistema usando la función <code>step</code>	35
3.5.	Retrato de fase del oscilador de Van der Pol	36
3.6.	Retrato de fase de un sistema lineal	37
3.7.	Circuito con diodo túnel	38

Práctica 1

Introducción al *Control System Toolbox*

Introducción

En esta sesión, nos embarcaremos en un emocionante viaje para aprender a utilizar las funciones poderosas que ofrece el *Control System Toolbox* de MATLAB. Este conjunto de herramientas especializado nos permitirá diseñar y analizar sistemas de control analógico de manera eficiente y precisa.

Objetivo de la Práctica

El objetivo principal de esta práctica es familiarizarnos con las capacidades del *Control System Toolbox* y desarrollar habilidades prácticas en su aplicación. Al finalizar esta sesión, podrás:

1. Comprender las funciones clave del *Control System Toolbox*.
2. Simular sistemas de control analógico básicos utilizando herramientas específicas.
3. Ganar confianza en la manipulación de funciones y comandos para resolver problemas específicos de sistemas de control.

Contenido de la Práctica

Ejercicio 1

Introducción al *Control System Toolbox*

Una visión general de las capacidades y herramientas disponibles en el *Control System Toolbox*, destacando su importancia en el diseño y análisis de sistemas de control.

Creación y Manipulación de Sistemas Dinámicos

Aprenderemos a utilizar funciones para crear modelos de sistemas dinámicos y a manipular sus características clave.

Análisis de Respuesta Transitoria y en Frecuencia

Exploraremos las funciones que nos permiten analizar la respuesta transitoria y en frecuencia de nuestros sistemas de control.

Metodología

La práctica consistirá en una serie de ejercicios prácticos guiados que te llevarán paso a paso a través de las funciones esenciales del *Control System Toolbox*. Te animamos a que experimentes con los comandos y explores cómo afectan a los sistemas de control que estamos diseñando.

Este conocimiento será fundamental para las prácticas posteriores, donde aplicaremos estas habilidades en contextos específicos.

Al final de la práctica, deberás construir un informe escrito con algunos ejercicios que desarrollarás, posteriormente, de manera independiente.

1.1**Introducción al *Control System Toolbox***

El *Control System Toolbox* es un conjunto de rutinas diseñadas para MATLAB, destinadas a aplicar las diversas herramientas derivadas de la teoría de control en sistemas lineales.

Este *toolbox* está integrado por cuatro grupos de funciones:

- Modelos de sistemas dinámicos.
- Análisis lineales.
- Diseño y ajuste de sistemas de control.
- Cálculos con matrices especiales de la teoría de control.

Cuando abordamos el análisis de un sistema específico, el primer paso implica la obtención de las ecuaciones dinámicas que describen su comportamiento, en un proceso conocido como *modelado*. En caso de que estas ecuaciones sean no lineales, se procede con un proceso conocido como *linealización* con el fin de obtener un modelo de comportamiento aproximadamente lineal en las vecindades de un punto de operación. Esto proporciona una descripción del sistema, ya sea en forma de función de transferencia o de variables de estado, que servirá como punto de partida para la *simulación* del comportamiento dinámico del sistema.

En esta práctica, comenzaremos explorando las diversas formas de ingresar modelos y luego nos adentraremos en las herramientas de análisis, examinando las respuestas en tiempo y frecuencia. Finalmente, concluiremos esta exploración con un enfoque en algunas funciones para la visualización de los resultados de simulación.

1.2 Creación y Manipulación de Sistemas Dinámicos

Todo intento de diseñar un sistema debe iniciarse con la anticipación de su rendimiento antes de que el sistema pueda ser detalladamente diseñado o construido en la realidad. Esta anticipación se fundamenta en una *representación matemática* de las características dinámicas del sistema, conocida como *modelo matemático*. En el caso de numerosos sistemas físicos, los modelos matemáticos útiles se articulan en términos de *ecuaciones diferenciales*. Estas ecuaciones diferenciales se pueden manipular con el fin de llevarlas a formas estandarizadas. Para el caso de los sistemas de control, contamos con tres *representaciones estándar básicas*: función de transferencia, cero-polo-ganancia y variables de estado.

El *Control System Toolbox* incorpora funciones para la creación de objetos con atributos propios de cada uno de los tres tipos de representación. Es bueno aclarar que un mismo sistema puede ser representado en cualquiera de las tres formas y estas son *equivalentes* entre sí, pues representan un mismo sistema.

Función de Transferencia

Una función de transferencia es una representación matemática que describe la relación entre la entrada y la salida de un SLIT. La función de transferencia se expresa como la razón de Laplace de la transformada de la salida a la transformada de la entrada, bajo la suposición de condiciones iniciales nulas. La forma general de una función de transferencia $H(s)$:

$$G(s) = \frac{C(s)}{R(s)} = \frac{\text{num}(s)}{\text{den}(s)},$$

donde $H(s)$ es la función de transferencia, $C(s)$ es la transformada de Laplace de la salida, $R(s)$ es la transformada de Laplace de la entrada y s es la variable compleja de Laplace.

En MATLAB, un sistema representado en forma de función de transferencia se crea por medio de la función `tf` que recibe como argumentos los vectores numéricos que representan los polinomios asociados al numerador (`num`) y al denominador (`den`) de la fracción racional $G(s)$. Recuerda que en MATLAB el polinomio $s^5 + 4s^3 + 2s^2 + 3s + 7$ se representa con el vector `[1 0 4 2 3 7]`.

Por ejemplo, si tenemos el modelo de un sistema representado por la función de transferencia

$$G(s) = \frac{s^4 + 17s^3 + 99s^2 + 223s + 140}{s^5 + 32s^4 + 363s^3 + 2092s^2 + 5052s + 4320}$$

en MATLAB crearemos el objeto `tf` de la siguiente forma:

```
num = [1 17 99 223 140];  
den = [1 32 363 2092 5052 4320];  
G = tf(num, den)
```

que producirá como salida en el *Command Window*

G =

$$\frac{s^4 + 17s^3 + 99s^2 + 223s + 140}{s^5 + 32s^4 + 363s^3 + 2092s^2 + 5052s + 4320}$$

Continuous-time transfer **function**.

Una vez creado nuestro objeto `tf`, podemos acceder a las propiedades de este usando la función `tfdata`,

```
[NUM,DEN] = tfdata(G)
```

que produce la salida

NUM =

1×1 cell array
 {[0 1 17 99 223 140]}

DEN =

1×1 cell array
 {[1 32 363 2092 5052 4320]}

donde NUM y DEN son variables de clase `cell` que contienen los polinomios del numerador y denominador, respectivamente. El objeto `tf` tiene otras propiedades que permiten personalizarlo con los datos del sistema. En la Figura 1.1 se muestran estas propiedades, que pueden ser visualizadas dando doble click sobre el nombre de la variable del objeto `tf`.

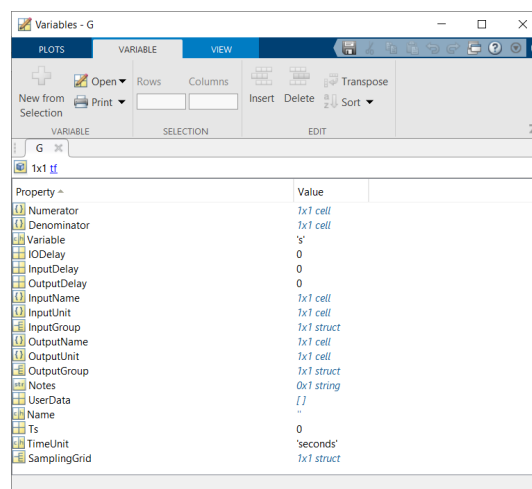


Figura 1.1: Propiedades de un objeto tipo `tf`

Cero – Polo – Ganancia

Un modelo de cero–polo–ganancia es una forma específica de representar una función de transferencia, especialmente útil en el análisis y diseño de sistemas dinámicos. En este tipo de modelo, la función de transferencia se expresa en función de los ceros y polos del sistema, junto con un término de ganancia. La forma general de una función de transferencia de cero-polo-ganancia es:

$$G(s) = K \frac{(s - z_1)(s - z_2) \dots (s - z_m)}{(s - p_1)(s - p_2) \dots (s - p_n)},$$

donde $G(s)$ es la función de transferencia, K es la ganancia del sistema, z_1, z_2, \dots, z_m son los ceros y p_1, p_2, \dots, p_n son los polos del sistema.

En este contexto:

- Los ceros (z) son los valores de s que hacen que el numerador de la función de transferencia sea cero, lo que significa que la salida es cero en esos puntos específicos.
- Los polos (p) son los valores de s que hacen que el denominador de la función de transferencia sea cero, indicando las ubicaciones en las cuales el sistema puede volverse inestable o donde la respuesta puede tener singularidades.
- La ganancia (K) es un factor de escala que afecta la amplitud de la respuesta del sistema.

Este tipo de representación es especialmente útil para comprender cómo los ceros, polos y la ganancia afectan el comportamiento del sistema en el dominio de Laplace. Facilita el análisis y diseño de sistemas lineales al proporcionar una descripción estructurada y modular de la función de transferencia.

Si el modelo de nuestro sistema se encuentra expresado en función de los ceros, los polos y la ganancia, en MATLAB utilizaremos la función `zpk`, la cual crea un objeto `zpk`. Por ejemplo, si tenemos el siguiente modelo:

$$G(s) = \frac{20(s + 2)(s + 3)(s + 6)(s + 8)}{s^2(s + 7)(s + 9)(s + 10)(s + 15)}$$

ingresaremos en MATLAB el siguiente código:

```
Z = [-2, -3, -6, -8];  
P = [0, 0, -7, -9, -10, -15];  
K = 20;  
G = zpk(Z, P, K)
```

que tiene como salida

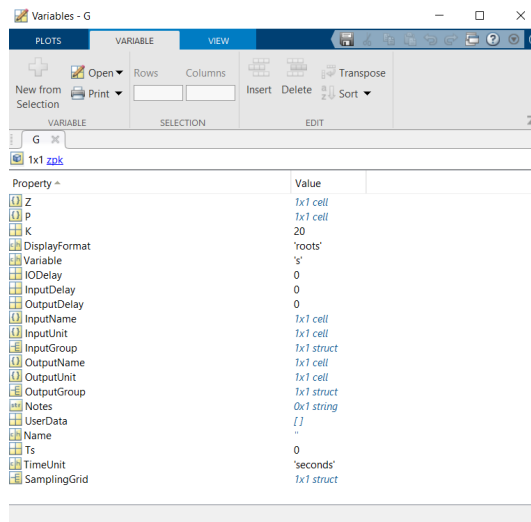
G =

$$\frac{20 (s+2) (s+3) (s+6) (s+8)}{s^2 (s+7) (s+9) (s+10) (s+15)}$$

Continuous-time zero/pole/gain model.

Es importante notar que los vectores de polos y ceros contienen las *ubicaciones* de los polos y ceros, respectivamente, sobre el plano complejo, por lo que es muy importante conservar su signo, es decir, cada polo asociado con un término $s + a$ se debe ingresar como $-a$. Además, en este ejemplo en concreto, debido al término s^2 en el numerador, fue necesario ingresar dos veces el 0 para representar el polo doble en el origen.

Las propiedades del objeto tipo zpk se muestran en la Figura 1.2



Property	Value
Z	1x1 cell
P	1x1 cell
K	20
DisplayFormat	'roots'
Variable	's'
IODelay	0
InputDelay	0
OutputDelay	0
InputName	1x1 cell
InputUnit	1x1 cell
InputGroup	1x1 struct
OutputName	1x1 cell
OutputUnit	1x1 cell
OutputGroup	1x1 struct
Notes	0x1 string
UserData	[]
Name	''
Ts	0
TimeUnit	'seconds'
SamplingGrid	1x1 struct

Figura 1.2: Propiedades de un objeto tipo zpk

Espacio de Estado

La representación en espacio de estado es otra forma de describir sistemas dinámicos, pero a diferencia de la representación en función de transferencia y cero–polo–ganancia, se enfoca en describir las ecuaciones diferenciales de primer orden que modelan el comportamiento del sistema. Esta representación es particularmente útil cuando se trabaja con sistemas de múltiples entradas y múltiples salidas.

En el contexto de un SLIT, la representación en espacio de estado se expresa mediante un conjunto de ecuaciones diferenciales ordinarias, conocidas como ecuaciones de estado, en la siguiente forma:

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

donde $x(t)$ es el vector de estado que describe la condición interna del sistema, $\dot{x}(t)$ es la derivada temporal de $x(t)$, $u(t)$ es el vector de entrada, $y(t)$ es el vector de salida y A , B , C , y D son matrices constantes que caracterizan al sistema y definen cómo las variables de estado, entrada y salida se relacionan entre sí.

La representación en espacio de estado ofrece una visión completa del comportamiento dinámico del sistema y es particularmente útil para el análisis y diseño de sistemas complejos. Puede capturar información sobre la estructura interna y la interconexión de las variables del sistema, lo que facilita la comprensión de su comportamiento en diferentes condiciones. Además, esta representación es fundamental en el diseño de controladores y en el análisis de la estabilidad del sistema.

Si tenemos un sistema representado en espacio de estado con las siguientes matrices:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -3 & 3 \end{pmatrix} \quad B = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \quad D = 0$$

Ingresaremos el siguiente código para la creación del objeto ss que lo representa:

```
A = [0, 1, 0; 0, 0, 1; 1, -3, 3];
B = [1; 1; 1];
C = [1, 0, 0];
D = 0;
sys = ss(A,B,C,D)
```

que produce la salida

sys =

A =

	x1	x2	x3
x1	0	1	0
x2	0	0	1
x3	1	-3	3

B =

	u1
x1	1
x2	1
x3	1

C =

	x1	x2	x3
y1	1	0	0

$$D = \begin{bmatrix} & u1 \\ y1 & 0 \end{bmatrix}$$

Continuous-time state-space model.

El objeto ss creado tiene las propiedades mostradas en la Figura 1.3

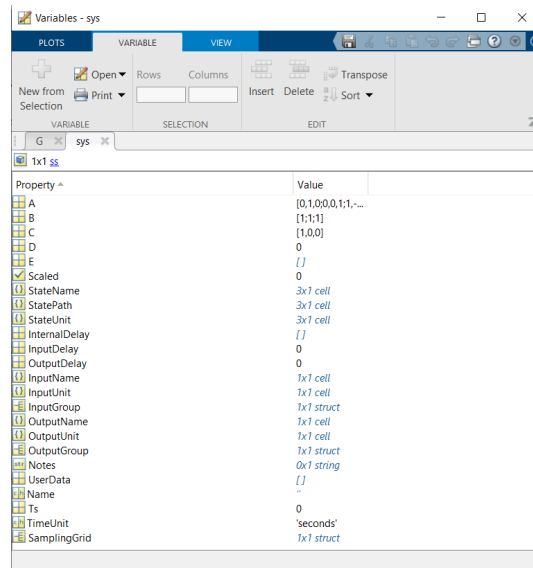


Figura 1.3: Propiedades de un objeto tipo ss

1.3 Análisis de Respuesta Transitoria y en Frecuencia

Una vez que hemos comprendido cómo ingresar sistemas en MATLAB, podemos avanzar hacia la exploración de las funciones especializadas en el análisis.

Respuesta en el tiempo

En lo que respecta al cálculo de la respuesta temporal de un SLIT, el *Control Systems Toolbox* dispone de funciones específicas, las cuales están detalladas en la Tabla 1.1.

Las funciones `step` e `impz` se aplican de la misma forma. Por ejemplo, dado el sistema

```
num = [1 17 99 223 140];
den = [1 32 363 2092 5052 4320];
G = tf(num,den);
```

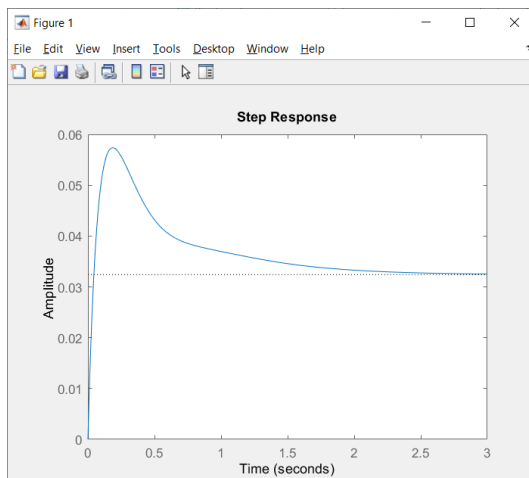
Tabla 1.1: Funciones para el análisis de respuesta temporal

Función	Descripción
step	Respuesta al escalón
impulse	Respuesta al impulso
lsim	Respuesta ante una entrada arbitraria
gensig	Genera señales de entrada

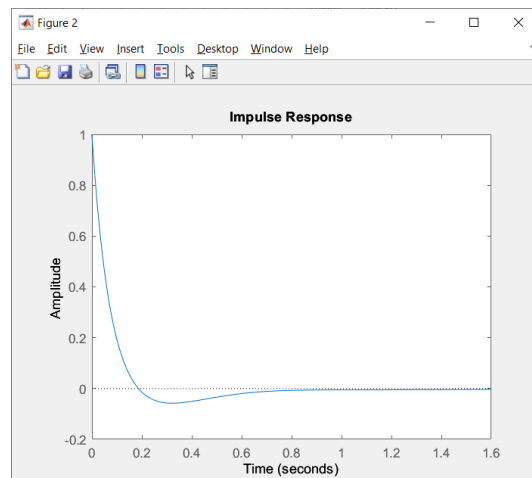
podemos obtener la respuesta al escalón y al impulso de la siguiente forma:

```
figure(1)
step(G)
figure(2)
impulse(G)
```

cuyo resultado son las dos figuras mostradas en la Figura 1.4.



(a) Respuesta al escalón (step)



(b) Respuesta al impulso (impulse)

Figura 1.4: Respuestas del sistema

A las funciones step e impulse se les puede variar el tiempo de simulación mediante un argumento adicional, como se muestra a continuación:

```
tsim = linspace(0,5,1500);
figure(1)
step(G,tsim)
figure(2)
impulse(G,tsim)
```

Además, permiten la simulación simultanea de varios sistemas, mediante el siguiente código:

```
num = [1 17 99 223 140];  
den = [1 32 363 2092 5052 4320];  
G = tf(num,den);  
G1 = tf([1 7],[1 10 216]);  
  
tsim = linspace(0,4,1500);  
figure(1)  
step(G,G1,tsim)  
figure(2)  
impulse(G,G1,tsim)
```

En los casos que necesitemos la respuesta temporal ante otras entradas podemos utilizar la función `lsim`. El llamado a esta función tiene la siguiente forma:

```
lsim(sys,u,t,xo)
```

donde `sys` es el sistema, `u` el vector de entrada, `t` el vector de tiempo y `xo` las condiciones iniciales (solo para sistemas en variables de estado). El vector `u` debe tener tantas filas como elementos tiene `t` y tantas columnas con entradas tenga el sistema.

Es posible usar la función `gensig` para generar el vector de entradas, `u`. Esta tiene la siguiente sintaxis:

```
[u,t] = gensig(tipo,tau,Tf,Ts);
```

donde `tipo` puede ser `'square'` para onda cuadrada, `'sin'` para onda senoidal y `'pulse'` para onda de pulsos. El argumento `tau` indica el periodo, `Tf` el tiempo final y `Ts` el tiempo de muestreo.

```
G = tf([1 0.5],2,[1 4 3]);  
[u1,t] = gensig('sin',1,10,0.01);  
u2 = [zeros(100,1);ones(901,1)];  
u = [0.1*u1,u2];  
lsim(G,u,t)
```

El resultado lo podemos ver en la Figura 1.5

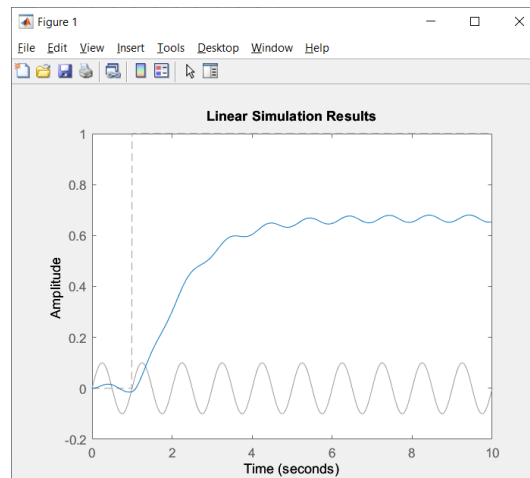


Figura 1.5: Respuesta del sistema usando lsim

Respuesta en la frecuencia

En la Tabla 1.3 se enumeran algunas de las funciones que dispone el *Control Systems Toolbox* para obtener la respuesta en frecuencia de un sistema.

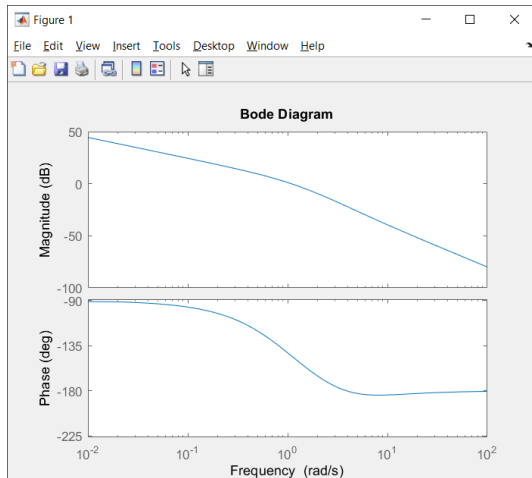
Tabla 1.3: Funciones para el análisis de respuesta frecuencial

Función	Descripción
bode	Diagrama de Bode
rlocus	Lugar de raíces
pzmap	Diagrama de polos y ceros
margin	Márgenes de fase y ganancia

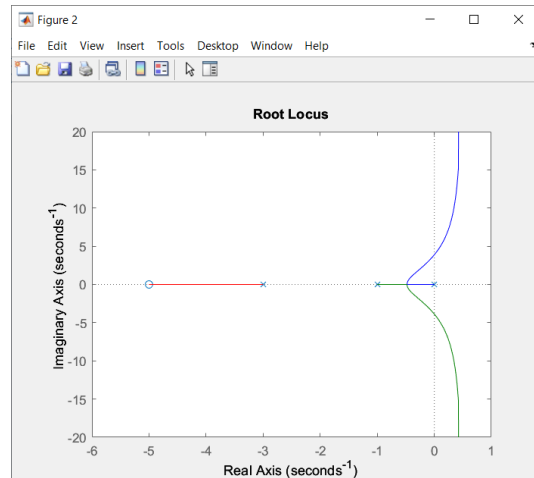
El uso de tales funciones se muestra a continuación:

```
G = zpk([-5],[0;-1;-3],1);
figure(1)
bode(G)
figure(2)
rlocus(G)
figure(3)
pzmap(G)
figure(4)
margin(G)
```

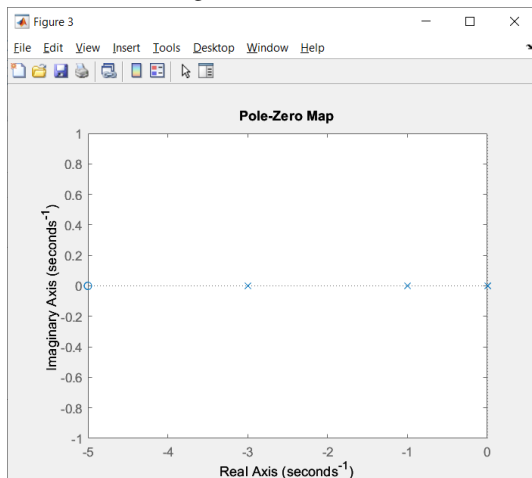
y sus resultados se pueden ver en la Figura 1.6



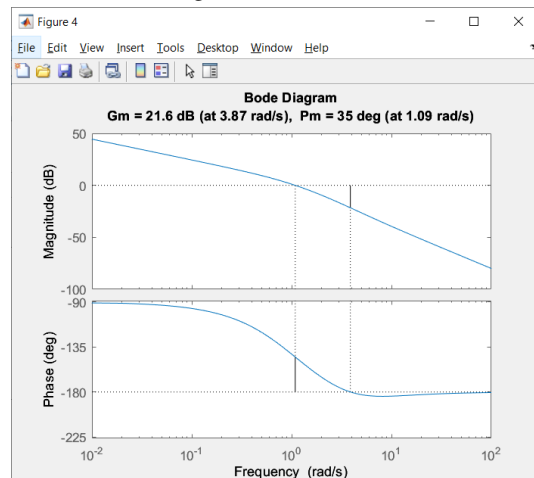
(a) Diagrama de Bode (bode)



(b) Lugar de raíces (rlocus)



(c) Diagrama de polos y ceros (step)



(d) Márgenes de fase y ganancia (impulse)

Figura 1.6: Gráficas para análisis de respuesta en frecuencia

Observaciones finales

El *Control Systems Toolbox* cuenta con la función `ltiview` que constituye un resumen de las funciones que hemos visto hasta aquí para obtener las respuestas temporal y frecuencial. al usarlo, se abre una ventana con menús que permiten importar a esta ventana cualquiera de los sistemas definidos en el *Workspace* y seleccionar el tipo de gráfico.

En la página web <https://la.mathworks.com/products/control.html> podrás expandir tus conocimientos acerca del *Control System Toolbox*, encontrar ejemplos de uso y explorar todas las funciones que vienen en él.

1.4 Trabajo práctico

Procedimiento

La Figura 1.7 muestra el sistema de suspensión de un automóvil con el que se trabajará en esta práctica. La variable de interés es la posición central del eje que sostiene la llanta, $x(t)$. La entrada al sistema es la superficie de la carretera $y(t)$, que puede variar de forma independiente. Suponemos que la masa de la carrocería no se mueve con respecto al eje del auto. Esto elimina el movimiento vertical de la carrocería, lo que simplifica el análisis. Puede que esta no sea una suposición realista, pero proporciona una primera aproximación del rendimiento de la suspensión.

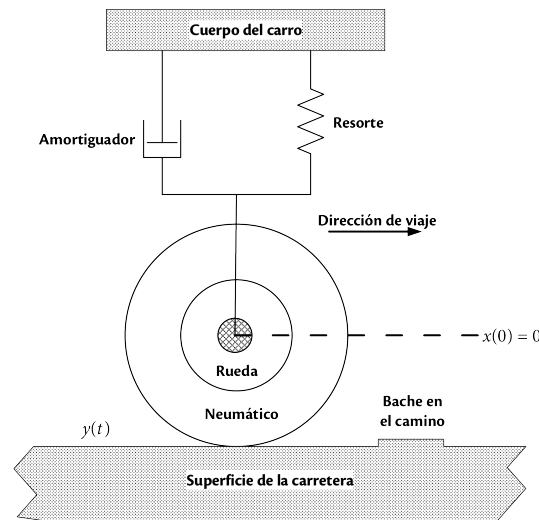


Figura 1.7: Sistema de suspensión utilizado en el análisis

El modelo matemático del sistema viene dado por la siguiente ecuación diferencial:

$$\frac{M}{K_1} \frac{d^2 x(t)}{dt^2} + \frac{B}{K_1} \frac{dx(t)}{dt} + \frac{K + K_1}{K_1} x(t) = y(t)$$

1. Deberá crear un archivo en MATLAB para simular la respuesta temporal del sistema de suspensión ante diferentes entradas:
 - a) Escalón de amplitud 5 cm (debe buscar cómo se modifican los parámetros de la función step para simular escalones de amplitud diferente a la unidad)
 - b) Rampa saturada en 5 cm y pendiente 0.2 cm/ms
 - c) Función sigmoide saturada en 5 cm en 25 ms

Para todos los casos, use un vector de tiempo de 500 ms con pasos de 100 μ s y los parámetros del modelo mostrados en la Tabla 1.5.

Tabla 1.5: Parámetros del modelo

Parámetro	Valor
K	$1.751 \times 10^5 \text{ N/m}$
K_1	$3.5 \times 10^4 \text{ N/m}$
M	14 kg
B	536 N s/m

2. Repita las simulaciones dejando los valores de K , K_1 y M fijos y variando el valor de B tal que sea 400, 750, 1500 y 3000 N s/m. Grafique todas las respuestas en un mismo eje.
3. Obtenga *a)* el diagrama de márgenes de fase y ganancia y *b)* el mapa de polos y ceros del sistema original y de las variaciones del inciso anterior.

Evaluación

Envíe un documento con la respuesta a los siguientes elementos, junto con el archivo de MATLAB y las gráficas obtenidas.

1. Para todos los casos, establezca inicialmente la función de transferencia.
2. Describa los cambios en la respuesta temporal de la suspensión a medida que varía el amortiguamiento (tiempo de estabilización, máximo sobreimpulso, etc.).
3. Describa los cambios en los márgenes de fase y ganancia, relacionados con la variación del amortiguamiento.
4. Describa el efecto del amortiguamiento sobre la ubicación de los polos del sistema.

Práctica 2

Modelos en función de transferencia

Introducción

En esta práctica, exploraremos cómo trabajar con modelos en función de transferencia en MATLAB. Nos centraremos en tres aspectos clave: la manipulación de funciones polinómicas básicas, el análisis de funciones racionales mediante fracciones parciales y la creación de funciones de transferencia utilizando el *Symbolic Math Toolbox*. Estos conceptos son fundamentales en el análisis y diseño de sistemas de control y nos permitirán comprender mejor su comportamiento.

Objetivo de la Práctica

El objetivo de la práctica es que adquieras habilidades prácticas en la manipulación y análisis de funciones de transferencia, elementos fundamentales en el diseño y análisis de sistemas de control. Al finalizar esta sesión, podrás:

1. Comprender y aplicar el uso de funciones polinómicas básicas, permitiéndote analizar sistemas de control desde una perspectiva algebraica.
2. Realizar el análisis de funciones racionales mediante fracciones parciales, facilitando la simplificación y comprensión de sistemas complejos.
3. Crear funciones de transferencia utilizando el *Symbolic Math Toolbox*, lo que te permitirá modelar sistemas dinámicos de manera precisa y eficiente en MATLAB.

Contenido de la Práctica

Funciones polinómicas básicas

Estudiaremos funciones polinómicas simples, y cómo operarlas, así como su relevancia en el análisis de sistemas de control.

Análisis de funciones racionales

Analizaremos una función racional y cómo descomponerla en fracciones parciales.

Funciones de transferencia simbólicas

Usaremos el *Symbolic Math Toolbox* para crear y analizar funciones de transferencia en términos de variables simbólicas.

Metodología

La práctica se desarrollará de forma teórico-práctica, enfocada en la resolución de ejercicios y la manipulación de funciones en MATLAB y consistirá en tres etapas:

1. **Introducción teórica:** Breve explicación de los conceptos fundamentales relacionados con funciones polinómicas, fracciones parciales y funciones de transferencia.
2. **Demostración práctica:** Ejemplos guiados de cómo realizar operaciones con funciones polinómicas básicas, descomposición en fracciones parciales y creación de funciones de transferencia simbólicas utilizando MATLAB.
3. **Práctica independiente:** Resolverás ejercicios prácticos relacionados con los temas tratados, aplicando los conceptos aprendidos de forma autónoma.

2.1 Funciones polinómicas básicas

Definimos polinomios en MATLAB usando vectores de fila. En concreto, para definir un polinomio creamos un vector fila donde los elementos de este último son los coeficientes del polinomio que queremos insertar, en orden descendente.

En general, si tenemos el polinomio

$$p(t) = a_n t^n + a_{n-1} t^{n-1} + \dots + a_1 t + a_0$$

podemos definirlo en MATLAB mediante la instrucción

```
p = [an, an_1, . . . , a1, a0]
```

Una vez definido el polinomio, si deseamos evaluarlo en un punto $t = k$ específico, usaremos la instrucción

```
p_val = polyval(p, k)
```

También se pueden calcular las raíces de un polinomio, usando

```
r = roots(p)
```

En caso de conocerse las raíces del polinomio de grado n : r_1, r_2, \dots, r_n , se pueden hallar los coeficientes del polinomio mediante

```
p = poly(r)
```

Además, es posible convertir de un polinomio simbólico a una matriz polinómica y viceversa. Esto lo podemos lograr con los comandos

```
sym2poly(simb)
poly2sym(vect)
```

La variable `simb` denota el polinomio simbólico que usamos para convertir a un vector polinomio y la variable `vect` denota el polinomio vectorial que usamos para convertir a un polinomio simbólico.

Un comando similar a `sym2poly(simb)`, pero que devuelve los coeficientes en forma simbólica, es el comando

```
coefs = coeffs(simb, 'All')
```

Ejemplos

2.1.1 Considere el polinomio $p(t) = t^2 - 2t + 1$

- a) Define el polinomio
- b) Calcula las raíces del polinomio
- c) Calcula el valor del polinomio para $t = 1$

Solución:

```
p = [1 -2 1];
r = roots(p)
p_value = polyval(p,1)
```

`r =`

```
1
1
```

`p_value =`

```
0
```

2.1.2 Construya, en cada caso, la matriz polinomial asociada a las raíces:

- a) $\{-1, -2, -3\}$
- b) $\{-4 + 2j, -4 - 2j, -8, 0\}$
- c) $\{-1, -1, -2 - j, -2 + j\}$

Solución:

```
p1 = poly([-1, -2, -3])
p2 = poly([-4+2j, -4-2j, -8, 0])
p3 = poly([-1, -1, -2-1j, -2+1j])
```

p1 =

1 6 11 6

p2 =

1 16 84 160 0

p3 =

1 6 14 14 5

2.1.3 Considere el polinomio simbólico $p(t) = t^3 + 2t + 6$

- a) Defina el polinomio usando variables simbólicas
- b) Convierta el polinomio simbólico en matriz polinomial
- c) Convierta la matriz polinómica nuevamente al polinomio simbólico

Solución:

```
syms t
symb = t^3+2*t+6
pol = sym2poly(symb)
symb_1 = poly2sym(pol)
```

symb =

$t^3 + 2*t + 6$

pol =

1 0 2 6

symb_1 =

$x^3 + 2*x + 6$

2.1.4 Considere el polinomio $p(t) = 8t^6 + 4t^5 - 2t^3 + 7t + 2$

- a) Defina el polinomio usando variables simbólicas
- b) Extraiga los coeficientes simbólicos del polinomio

Solución:

```
syms t
p = 8*t^6+4*t^5-2*t^3+7*t+2;
coeff = coeffs(p, 'All')
```

coeff =

[8, 4, 0, -2, 0, 7, 2]

2.2 Análisis de funciones racionales en fracciones parciales

Analizar una función racional en fracciones parciales es importante debido a que tenemos la capacidad de representar la función de una manera diferente, como por ejemplo teniendo los polos del sistema como denominadores de las fracciones parciales. También hay razones prácticas, como facilitar la búsqueda de la transformada inversa de Laplace de una función de transferencia. Una expansión en fracciones parciales de una función racional es la siguiente:

$$\frac{b(s)}{a(s)} = \frac{r_1}{s - p_1} + \frac{r_2}{s - p_2} + \cdots + \frac{r_n}{s - p_n} + k$$

donde p_1, \dots, p_n son las singularidades de la función racional, c_1, \dots, c_n los residuos asociados a cada singularidad y k el término directo para el caso de fracciones impropias. En MATLAB, la expansión en fracciones parciales la realizamos mediante el comando

```
[r, k, p] = residue(num, den)
```

MATLAB retorna los residuos r correspondientes a los polos p y el término k en forma de vectores columna. En el caso de que k sea cero (fracción propia), MATLAB retorna $[]$. Por otro lado, `residue` también puede construir la fracción racional a partir de r , p , k conocidos:

```
[num,den] = residue(r,p,k)
```

Adicionalmente, se puede hallar la expansión en fracciones parciales de una función racional simbólica `frac` mediante las siguientes líneas de comandos:

```
syms s
G = partfrac(fract,s)
pretty(G)
```

La función `pretty` se invoca al final con el fin de tener una mejor visualización del resultado. Para el caso contrario, se puede hallar la función racional simbólica a partir de las fracciones parciales por medio de las líneas de comandos

```
syms s
G = collect(expr,s)
pretty(G)
```

Ejemplos

2.2.1 Encuentre la expansión en fracciones parciales, numérica y simbólica, de la función racional

$$X(s) = \frac{s+2}{s^3+4s^2+3s}$$

Solución:

De forma numérica:

```
num=[1 2];
den=[1 4 3 0];
[c,p,k]=residue(num,den)
```

`r =`

```
-0.1667
-0.5000
0.6667
```

`p =`

```
-3
-1
0
```

`k =`

```
[]
```

Y en forma simbólica:

```
syms s
X = partfrac((s+2)/(s^3 + 4*s^2 + 3*s),s)
pretty(X)
```

X =

$$2/(3*s) - 1/(6*(s + 3)) - 1/(2*(s + 1))$$

$$\frac{2}{3s} - \frac{1}{6(s+3)} - \frac{1}{2(s+1)}$$

2.2.2 Expanda en fracciones parciales, numéricas y simbólicas, la función racional:

$$Y(s) = \frac{3s^4}{s^4 + 6s^3 + 29s^2 + 44s + 20}$$

Solución:

```
syms s
num = [3 0 0 0 0];
den = poly([-1, -1, -2-4j, -2+4j]);
[r,p,k]=residue(num,den)
Y = partfrac((3*s^4)/(s^4 + 6*s^3 + 29*s^2 + 44*s + 20),s)
pretty(Y)
```

r =

$$\begin{aligned} &-8.6367 + 1.8062i \\ &-8.6367 - 1.8062i \\ &-0.7266 + 0.0000i \\ &0.1765 + 0.0000i \end{aligned}$$

p =

$$\begin{aligned} &-2.0000 + 4.0000i \\ &-2.0000 - 4.0000i \\ &-1.0000 + 0.0000i \\ &-1.0000 + 0.0000i \end{aligned}$$

k =

$$3$$

Y =

$$3/(17*(s+1)^2) - 210/(289*(s+1)) - ((4992*s)/289 + 14160/289)/(s^2 + 4*s + 20) + 3$$

$$\frac{3}{17(s+1)^2} - \frac{210}{289(s+1)} - \frac{\frac{4992s}{289} + \frac{14160}{289}}{s^2 + 4s + 20} + 3$$

Nota: En el caso numérico, cuando el denominador tiene raíces complejas conjugadas, estas se consideran por separado como raíces simples. Para el caso de raíces repetidas, estas aparecen en orden ascendente.

2.2.3 Encuentre, de forma numérica y simbólica, la fracción racional resultante de sumar las siguientes fracciones:

$$X(s) = 2 + \frac{3}{s-1} + \frac{1.5}{s+4.3} - \frac{1}{s-2}$$

Solución:

```
syms s
r = [3;1.5;-1];
p = [1;-4.3;2];
k = 2;
[num,den] = residue(r,p,k)
X = collect(2 + 3/(s-1) + 1.5/(s+4.3) - 1/(s-2))
pretty(X)
```

num =

2.0000 6.1000 -22.7000 -1.3000

den =

1.0000 1.3000 -10.9000 8.6000

X =

$$(20*s^3 + 61*s^2 - 227*s - 13)/(10*s^3 + 13*s^2 - 109*s + 86)$$

$$\begin{array}{r}
 3 \qquad 2 \\
 20 s^3 + 61 s^2 - 227 s - 13 \\
 \hline
 3 \qquad 2 \\
 10 s^3 + 13 s^2 - 109 s + 86
 \end{array}$$

2.3 Funciones de transferencia simbólicas

Como vimos en la Práctica 1, el modelo de un sistema se puede representar en forma de función de transferencia (`tf(num, den)`) o de cero-polo-ganancia (`zpk(zeros, poles, gain)`), con el fin de construir modelos que puedan ser simulados usando `step`, `impz` o `lsim`. Si el objetivo es hallar la expresión simbólica de la respuesta temporal, i.e. hallar la transformada inversa de Laplace de una función racional, estos modelos no serán de utilidad y se deberá representar el modelo como un objeto simbólico, usando las herramientas proveídas por el *Symbolic Math Toolbox*.

Recordemos que si se tiene la función de transferencia, o su equivalente cero-polo-ganancia:

$$G(s) = \frac{b_n s^n + b_{n-1} s^{n-1} + \dots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0} = K \frac{(s - z_1)(s - z_2) \dots (s - z_m)}{(s - p_1)(s - p_2) \dots (s - p_n)}$$

se puede ingresar el modelo en MATLAB como

```
sys_tf = tf(num, den)
sys_zpk = zpk(Z, P, K)
```

Esta forma puede resultar a veces engorrosa, principalmente cuando manejemos subsistemas, cada uno de ellos con su propia función de transferencia y aún se desconoce la función de transferencia del sistema final. Para este caso es mejor recurrir a crear la variable `s` como objeto función de transferencia o cero-polo-ganancia (`s = tf('s')` o `s = zpk('s')`) y luego proceder a escribir la función de transferencia como si fuera un objeto simbólico.

Para el caso en que deseemos analizar el modelo del sistema de forma simbólica, podemos establecer la función de transferencia o cero-polo-ganancia a partir de la definición de `s` como variable simbólica. En este caso, el objetivo es trabajar con el modelo en el dominio de Laplace, para hallar luego su inversa en el dominio del tiempo. O en el caso inverso, partir de la definición en tiempo y hallar la función de transferencia asociada.

```
syms s t
laplace(expr_t)
ilaplace(expr_s)
```

Ejemplos

2.3.1 Cree el objeto función de transferencia del sistema representado por la siguiente función racional:

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

Luego, encuentre la representación cero-polo-ganancia.

Solución:

```
s = tf('s');
sys_tf = (s+2)/(s^2+2*s+1)
s = zpk('s');
sys_zpk = (s+2)/(s^2+2*s+1)
```

sys_tf =

$$\frac{s + 2}{s^2 + 2s + 1}$$

Continuous-time transfer **function**.

sys_zpk =

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

Continuous-time zero/pole/gain model.

2.3.2 Cree el objeto cero-polo-ganancia del sistema representado por la siguiente función racional:

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

Luego, encuentre la representación en función de transferencia.

Solución:

```
s = zpk('s');
sys_zpk = ((s+2)*(s-1))/((s+3)^2*(s-5))
s = tf('s');
sys_tf = ((s+2)*(s-1))/((s+3)^2*(s-5))
```

sys_zpk =

$$\frac{(s+2)(s-1)}{(s+3)^2(s-5)}$$

Continuous-time zero/pole/gain model.

sys_tf =

$$\frac{s^2 + s - 2}{s^3 + s^2 - 21s - 45}$$

Continuous-time transfer **function**.

2.3.3 Considere el sistema representado por la función de transferencia:

$$G(s) = \frac{s^3 + 4s^2 + 3s + 12}{s^4 + 11s^3 + 46s^2 + 76s + 40}$$

- a) Cree un objeto simbólico para esta función de transferencia
- b) Halle la respectiva representación cero-polo-ganancia
- c) A partir de la representación cero-polo-ganancia, halle nuevamente la función de transferencia

Solución:

```
syms s;
G_tf = (s^3 + 4*s^2 + 3*s + 12)/(s^4 + 11*s^3 + 46*s^2 + 76*s + 40);
pretty(G_tf)
G_zpk = simplify(G_tf, 'Steps', 50);
pretty(G_zpk)
G_tf2 = collect(G_zpk);
pretty(G_tf2)
```

$$\frac{s^3 + 4s^2 + 3s + 12}{s^4 + 11s^3 + 46s^2 + 76s + 40}$$

$$\frac{(s + 3)(s + 4)}{(s + 3)(s + 4)}$$

$$\frac{(s+1)(s+2)(s^2+8s+20)}{s^3+4s^2+3s+12}$$

$$\frac{s^4+11s^3+46s^2+76s+40}{s^3+4s^2+3s+12}$$

Nota: El argumento adicional '**Steps**' en la función `simplify` es opcional y controla la cantidad de iteraciones para llegar a la convergencia de la factorización, principalmente del numerador, si lo quitas, notarás que el numerador no se factoriza.

2.3.4 Halle la expresión matemática de la respuesta al escalón del sistema del ejemplo anterior.

Solución:

```
syms s
G = (s^3 + 4*s^2 + 3*s + 12)/(s^4 + 11*s^3 + 46*s^2 + 76*s + 40);
U = 1/s;
y = ilaplace(G*U);
pretty(y)
```

$$\frac{\exp(-2t) \left(\frac{7}{8} + \frac{12 \exp(-t)}{13} + \frac{\exp(-4t) \left(\cos(2t) - \frac{\sin(2t)}{131} \right)}{520} \right)}{10} + \frac{3}{10}$$

2.4 Trabajo práctico

Procedimiento

Para cada uno de los ejercicios siguientes cree un *script* en MATLAB que provea la solución.

Ejercicio 1

Considera la función $f(x) = 3x^2 - 2x + 1$.

- Encuentra las raíces de $f(x)$
- Grafica la función $f(x)$ en el intervalo $[-5, 5]$

Ejercicio 2

Dada la función racional

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

- a) Encuentra la expansión en fracciones parciales de $F(s)$
- b) Halla la transformada inversa de Laplace de cada fracción parcial
- c) Encuentra la representación cero-polo-ganancia de $F(s)$
- d) Grafica la respuesta del sistema representado por $F(s)$ ante la entrada $u(t) = 3 \cos(4t)$

Ejercicio 3

Utilizando Symbolic Toolbox, crea una función de transferencia simbólica $H(s)$ de la forma

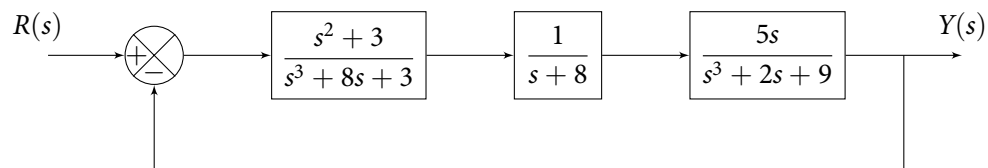
$$\frac{bs^2 + as + c}{ds^2 + es + f},$$

donde a, b, c, d, e y f son parámetros simbólicos.

- a) Define los parámetros simbólicos y la función de transferencia $H(s)$
- b) Calcula la inversa de Laplace de $H(s)$

Ejercicio 4

Considere el sistema realimentado mostrado a continuación



- a) Cree objetos simbólicos para cada subsistema
- b) Encuentre la función de transferencia simbólica del sistema completo y la representación cero-polo-ganancia

Evaluación

Envíe un documento con los resultados de cada ejercicio, junto con el archivo de MATLAB y las gráficas obtenidas.

Práctica 3

Modelos en ecuaciones de estado

Introducción

En esta práctica, exploraremos cómo trabajar con modelos en variables de estado en MATLAB. Simularemos el comportamiento de sistemas no lineales, usando métodos numéricos de solución del sistema de ODEs. Además, revisaremos los conceptos de simulación de sistemas lineales descritos por una SSR.

Objetivo de la Práctica

El objetivo de la práctica es que adquieras destreza para la implementación y análisis de sistemas dinámicos descritos por modelos en ecuaciones de estado utilizando MATLAB. Al finalizar esta sesión, podrás:

1. Definir un sistema dinámico mediante un conjunto de ecuaciones de estado.
2. Crear la representación en espacio de estados del sistema.
3. Visualizar el retrato de fase de sistemas de segundo orden.

Contenido de la Práctica

Ecuaciones de estado de sistemas no lineales

Estudiaremos la forma de resolver numéricamente el comportamiento dinámico de un sistema no lineal descrito por ecuaciones de estado.

Representación en espacio de estados (SSR)

Para el caso de sistemas lineales, veremos las herramientas disponibles en MATLAB para trabajar con la SSR.

Retrato de fase de sistemas de segundo orden

Estudiaremos el concepto de retrato de fase y cómo obtenerlo en MATLAB.

Metodología

La práctica se desarrollará de forma teórico-práctica, enfocada en la resolución de ejercicios y la manipulación de funciones en MATLAB y consistirá en tres etapas:

1. **Introducción teórica:** Breve explicación de los conceptos fundamentales relacionados con modelos en ecuaciones de estado de sistemas lineales y no lineales, al igual que el concepto de retrato de fase.
2. **Demostración práctica:** Ejemplos guiados de cómo realizar la simulación del comportamiento dinámico de sistema.
3. **Práctica independiente:** Resolverás ejercicios prácticos relacionados con los temas tratados, aplicando los conceptos aprendidos de forma autónoma.

3.1 Ecuaciones de estado de sistemas no lineales

En sistemas lineales, siempre es posible encontrar la solución explícita a las ecuaciones diferenciales que representan su dinámica utilizando métodos analíticos. Este no es el caso de los sistemas no lineales, donde el cálculo de la solución normalmente sólo puede lograrse utilizando métodos de análisis numérico. Además, surgen nuevas preguntas sobre la existencia de una solución para determinadas condiciones iniciales.

Otra característica notable de los sistemas no lineales es la alta sensibilidad a las perturbaciones en las condiciones iniciales. Un pequeño cambio en la condición inicial de dicho sistema (es decir, debido a un error de redondeo) puede producir una respuesta completamente diferente. Este hecho hace que la predicción a largo plazo de fenómenos no lineales de la vida real sea una tarea extremadamente tediosa.

En MATLAB existen diferentes métodos de solución numérica para ecuaciones diferenciales: ode45, ode23, ode23s, **entre otros**. Todos comparten la misma estructura para su uso:

```
[t, x] = ode45(odefun, tspan, x0)
[t, x] = ode23(odefun, tspan, x0)
[t, x] = ode23s(odefun, tspan, x0)
```

Donde odefun es el conjunto de ecuaciones de estado a resolver, tspan el intervalo de integración y y0 las condiciones iniciales.

Ejemplos

- 3.1.1 Oscilador de Van der Pol.** El oscilador de Van der Pol es una ecuación no lineal que encuentra aplicaciones en diversos campos científicos como la biología y la sismología. Se describe mediante la siguiente ecuación diferencial

$$\ddot{x}(t) - c(1 - x^2(t))\dot{x}(t) + kx(t) = 0$$

Con el fin de poder simular su comportamiento, se debe llevar la EDO a su equivalente en ecuaciones

de estado, dada por

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ c(1 - x_1^2)x_2 - kx_1 \end{bmatrix}$$

A continuación, es necesario crear la función que defina la dinámica de estas ecuaciones

```
function dxdt=vanDerPol(t,x,c,k)

dxdt = [x(2);
        c*(1-x(1)^2)*x(2)-k*x(1)];
```

Finalmente, creamos el *script* para la simulación

```
t = 0:0.001:100;
c = 1;
k = 1;
x0 = 10;
v0 = 3;

[t,x] = ode45(@(t,x) vanDerPol(t,x,c,k), t, [x0,v0]);

subplot(2,1,1);
plot(t,x(:,1));
xlabel('t'); ylabel('x(t)');
title(sprintf('Van Der Pol, posición vs. tiempo, c=%3.2f, k=%3.2f',c,k));

subplot(2,1,2);
plot(t,x(:,2));
xlabel('t'); ylabel('v(t)');
title(sprintf('Van Der Pol, velocidad vs. tiempo, c=%3.2f, k=%3.2f',c,k));
```

cuya salida es la gráfica de la Figura 3.1

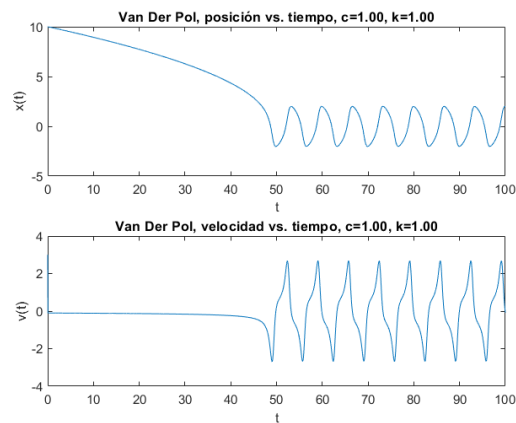


Figura 3.1: Simulación del oscilador de Van der Pol

3.1.2 Atractor de Lorenz. El atractor de Lorenz es un famoso sistema tridimensional no lineal de ecuaciones diferenciales que encuentra aplicación en biología, teoría de circuitos, mecánica, láseres y reacciones químicas. El sistema se describe mediante las siguientes ecuaciones diferenciales.

$$\begin{aligned}\dot{x}(t) &= \sigma(y(t) - x(t)) \\ \dot{y}(t) &= x(t)(\rho - z(t)) - y(t) \\ \dot{z}(t) &= x(t)y(t) - \beta z(t)\end{aligned}$$

Donde $\sigma, \rho, \beta > 0$ son los parámetros del sistema. Los valores de Lorenz para un sistema sensible son $\sigma = 10, \beta = 8/3$ y $\rho = 28$.

La función para las ecuaciones de estado sería

```
function dxdt=lorenz(t,x,a,b,c)

dxdt = [a*(x(2) - x(1));
        x(1)*(b-x(3)) - x(2);
        x(1)*x(2) - c*x(3)];
```

Y el script de simulación

```
t = 0:0.001:100;
a = 10;
b = 28;
c = 8/3;
x0 = 10;
y0 = 20;
z0 = 10;

[t,x] = ode45(@(t,x) lorenz(t,x,a,b,c), t, [x0,y0,z0]);

plot3(x(:,1),x(:,2),x(:,3));
xlabel('x(t)');
ylabel('y(t)');
zlabel('z(t)');
view([-10.0 -2.0])
title('Simulación del atractor de Lorenz')
```

Cuya salida se ilustra en la Figura 3.2

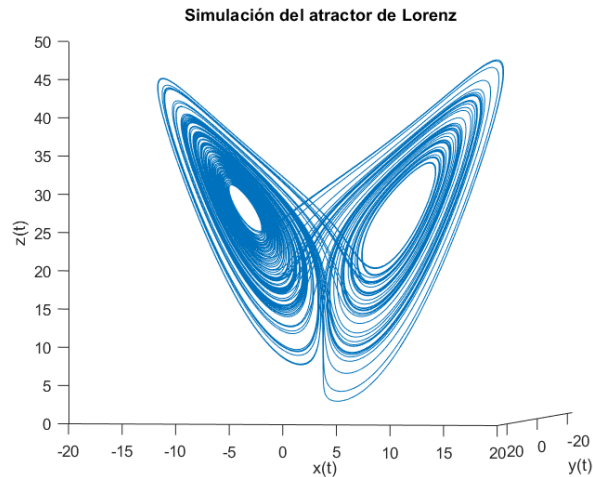


Figura 3.2: Simulación del atractor de Lorenz

3.2 Representación en espacio de estados (SSR)

El enfoque del espacio de estados es un método para modelar, analizar y diseñar una amplia gama de sistemas lineales. La SSR de un sistema viene dada por las ecuaciones

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

La primera ecuación se conoce como *ecuación de estados* y la segunda como *ecuación de salida*.

Ejemplos

3.2.1 Considere el sistema dinámico modelado mediante la SSR

$$\begin{aligned}\dot{x}(t) &= \begin{bmatrix} -1.702 & -50.72 & -263.38 \\ 0.22 & -1.418 & -31.99 \\ 0 & 0 & -14 \end{bmatrix} x(t) + \begin{bmatrix} -272.06 \\ 0 \\ 14 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} x(t)\end{aligned}$$

La simulación de la respuesta al escalón unitario se puede hacer de dos formas

a) **Simulación numérica.** Usando la función de transición de estados

```
function dxdt=sistema1(t,x)

dxdt = [-1.702*x(1) - 50.72*x(2) - 263.38*x(3) - 272.06;
        0.22*x(1) - 1.418*x(2) - 31.99*x(3);
        - 14*x(3) + 14];
```

y el script de simulación

```
t = linspace(0,4.5,2000);
x10 = 0;
x20 = 0;
x30 = 0;

[t,x] = ode45(@(t,x) sistema1(t,x), t, [x10,x20,x30]);

subplot(3,1,1)
plot(t,x(:,1));
xlabel('t'); ylabel('y_1(t)');
subplot(3,1,2)
plot(t,x(:,2));
xlabel('t'); ylabel('y_2(t)');
subplot(3,1,3)
plot(t,x(:,3));
xlabel('t'); ylabel('y_3(t)');
```

Cuyo resultado se muestra en la Figura 3.3

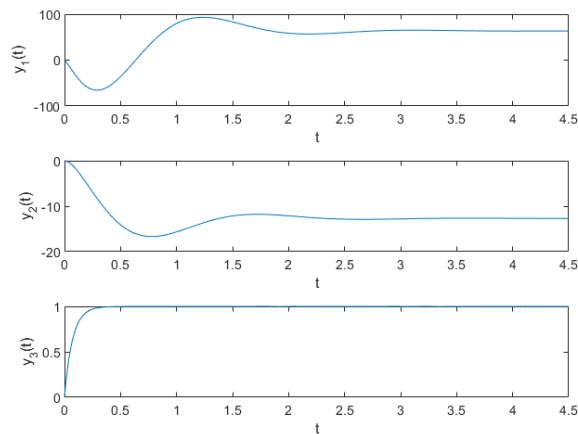


Figura 3.3: Respuesta del sistema usando simulación numérica

b) Usando el Control Systems Toolbox. Por medio de la función `step`

```
A = [-1.702 -50.72 -263.38;  
      0.22 -1.418 -31.99;  
      0 0 -14];  
B = [-272.06;0;14];  
C = eye(3);  
D = 0;  
step(A,B,C,D)
```

Cuyo resultado se muestra en la Figura 3.4

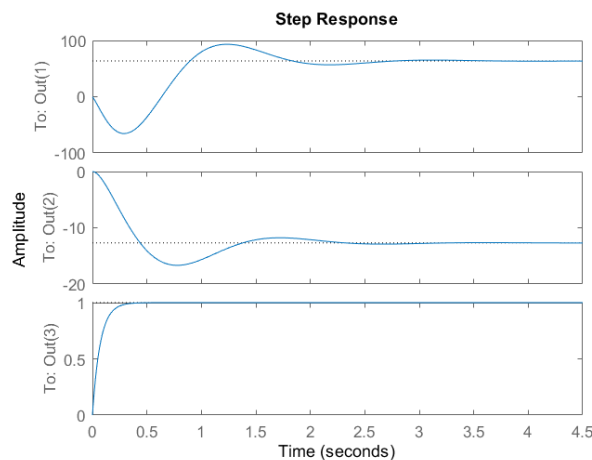


Figura 3.4: Respuesta del sistema usando la función `step`

3.3 Retrato de fase de sistemas de segundo orden

El retrato de fase para sistemas de segundo orden es una representación gráfica que muestra cómo evoluciona el estado de un sistema en el espacio de fase, el cual está definido por las variables de estado y sus derivadas. En el caso de sistemas de segundo orden, como osciladores armónicos o sistemas amortiguados, el retrato de fase suele mostrar la posición y la velocidad (o alguna otra variable relacionada) en un plano bidimensional. Esto permite visualizar cómo cambian las variables de estado en función del tiempo y proporciona información sobre el comportamiento dinámico del sistema, como la estabilidad, la oscilación y la convergencia a un punto de equilibrio.

Para visualizar el retrato de fase, haremos uso de la función `plottp` disponible en [MATLAB Central](#)

Ejemplos

3.3.1 Oscilador de Van der Pol. El retrato de fase para el oscilador del ejemplo 3.1.1 se obtiene mediante el siguiente código y su salida se muestra en la Figura 3.5.

```

c = 1;
k = 1;
odefun = @(t, x) [x(2); -k*x(1) + c*(1 - x(1)^2)*x(2)];
plotpp(odefun, 'tspan', 30, ...
        'quivercolor', [0.6,0.6,0.6], ...
        'linecolor', [0.3,0.3,0.3])

```

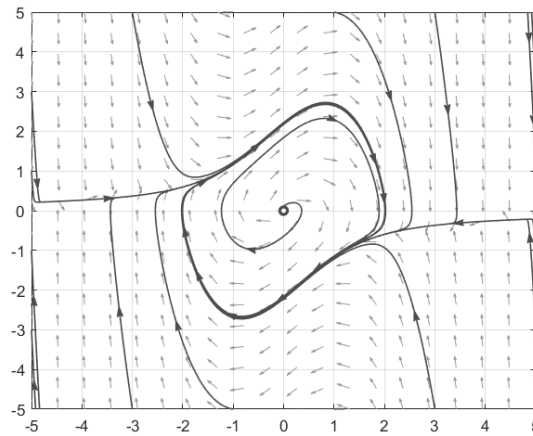


Figura 3.5: Retrato de fase del oscilador de Van der Pol

3.3.2 Considere el sistema lineal descrito por la SSR

$$\begin{aligned}\dot{x}(t) &= \begin{bmatrix} 2 & 1 \\ 0 & -3 \end{bmatrix} x(t) + \begin{bmatrix} 2 \\ 1 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 1 \end{bmatrix} x(t)\end{aligned}$$

El retrato de fase puede ser obtenido usando solo la matriz de transición de estados (A) y mediante el siguiente script

```

odefun = @(t, x) [2*x(1) + x(2); -3*x(2)];
plotpp(odefun, 'tspan', 30, ...
        'quivercolor', [0.6,0.6,0.6], ...
        'linecolor', [0.3,0.3,0.3])

```

Su resultado se muestra en la Figura 3.6

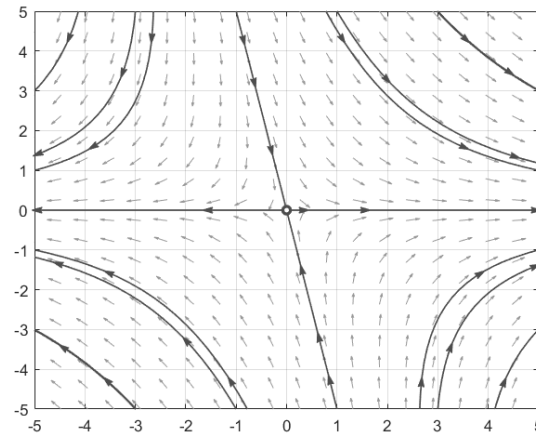


Figura 3.6: Retrato de fase de un sistema lineal

3.4 Trabajo práctico

Procedimiento

Para cada uno de los ejercicios siguientes cree un *script* en MATLAB que provea la solución.

Ejercicio 1

El atractor de Rössler es un sistema de tres ecuaciones diferenciales ordinarias no lineales estudiadas por Otto E. Rössler. Estas ecuaciones diferenciales definen un sistema dinámico de tiempo continuo que muestra dinámicas caóticas asociadas con las propiedades fractales del atractor.

Las ecuaciones que definen el sistema de Rössler son:

$$\begin{aligned}\dot{x} &= -y - z \\ \dot{y} &= x + Ay \\ \dot{z} &= B + z(x + C)\end{aligned}$$

Los valores originales del atractor caótico son $A = 0.2$, $B = 0.2$ y $C = 5.7$, aunque desde entonces los parámetros más comunes han sido $A = 0.1$, $B = 0.1$ y $C = 14$.

Compare el comportamiento de las dos configuraciones del atractor, partiendo de condiciones iniciales cero.

Ejercicio 2

Un generador sincrónico conectado a un bus infinito puede representarse por las ecuaciones

$$\begin{aligned}M\ddot{\delta} &= P - D\dot{\delta} - \eta_1 E_q \sin \delta \\ \tau \dot{E}_q &= -\eta_2 E_q + \eta_3 \cos \delta + E\end{aligned}$$

donde δ es un ángulo en radianes, E_q es voltaje, P es potencia mecánica, E es voltaje de entrada, D es un coeficiente de amortiguamiento, M es un coeficiente de inercia, τ es una constante de tiempo y η_1 , η_2 y η_3 son parámetros constantes.

a) Usando δ , $\dot{\delta}$ y E_q como variables de estado, obtener las ecuaciones de estado.

b) Realizar la simulación del sistema correspondiente a los valores

$$\begin{array}{llll} P = 0.815 & E = 1.22 & \eta_1 = 2 & \eta_2 = 2.7 \\ \eta_3 = 1.7 & \tau = 6.6 & M = 0.0147 & \frac{D}{M} = 4 \end{array}$$

usando dos vectores de condiciones iniciales diferentes.

Ejercicio 3

Simule la respuesta ante el escalón unitario, usando tanto simulación numérica como la función step, del sistema descrito por la SSR

$$\begin{aligned} \dot{x}(t) &= \begin{bmatrix} -0.435 & 0.209 & 0.02 \\ 0.268 & -0.394 & 0 \\ 0.227 & 0 & -0.02 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u(t) \\ y(t) &= [0.0003 \quad 0 \quad 0] x(t) \end{aligned}$$

Ejercicio 4

Circuito con diodo túnel. Considere el circuito con un diodo túnel mostrado en la Figura 3.7

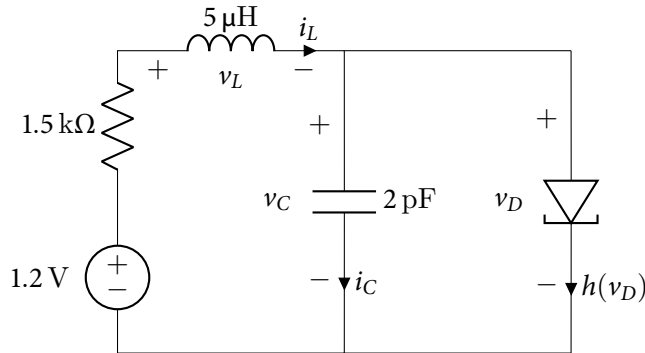


Figura 3.7: Circuito con diodo túnel

Midiendo tiempo en ns y corrientes en mA, el modelo en ecuaciones de estado resultante es

$$\begin{aligned} \dot{x}_1 &= 0.5(-h(x_1) + x_2) \\ \dot{x}_2 &= 0.2(-x_1 - 1.5x_2 + 1.2) \end{aligned}$$

donde

$$h(x_1) = 17.76x_1 - 103.79x_1^2 + 229.62x_1^3 - 226.31x_1^4 + 83.72x_1^5$$

Con base en este modelo, escriba las funciones necesarias para crear el retrato de fase.

Ejercicio 5

Grafique los retratos de fase de los sistemas cuyas matrices de transición de estado son

$$A_1 = \begin{bmatrix} -1 & -1 \\ 0 & -2 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & -2 \\ 0 & -1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} 3 & 5 \\ -8 & -1 \end{bmatrix}$$

$$A_6 = \begin{bmatrix} 2 & 5 \\ -8 & -2 \end{bmatrix}$$

Evaluación

Envíe un documento con los resultados de cada ejercicio, junto con el archivo de MATLAB y las gráficas obtenidas.