\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CODIGO 2

El código es una implementación de un sistema de ecuaciones diferenciales no lineales que se resuelve numéricamente para diferentes

condiciones iniciales.

import numpy as np: La librería NumPy es utilizada para trabajar con arrays y realizar operaciones matemáticas en ellos. En este código

se utiliza para generar valores aleatorios para las condiciones iniciales, así como para calcular la constante e.

import matplotlib.pyplot as plt: La librería Matplotlib es utilizada para realizar visualizaciones en 2D y 3D. En este código se utiliza

para crear un gráfico que muestra las soluciones del sistema de ecuaciones diferenciales para diferentes condiciones iniciales.

from scipy.integrate import odeint: La función odeint es una función de la librería Scipy que se utiliza para resolver sistemas de

ecuaciones diferenciales ordinarias. Esta función es la que se utiliza para resolver el sistema de ecuaciones en este código.

def equations(x, t, L, C, R, e, u): Esta función define las ecuaciones del sistema de ecuaciones diferenciales que se van a resolver.

La función toma como argumentos el vector de variables dependientes x, el tiempo t, y los parámetros L, C, R, e y u.

L = 1, C = 1, R = -0.5, u = 1: Estos son los valores de los parámetros del sistema de ecuaciones diferenciales.

e = np.sqrt(L/C): Esta línea calcula el valor de la constante e utilizando los valores de L y C.

num\_conditions = 100: Este es el número de condiciones iniciales que se van a generar y resolver.

x0\_list = np.random.uniform(low=-1, high=1, size=(num\_conditions, 2)): Esta línea genera num\_conditions valores aleatorios para las

condiciones iniciales del sistema de ecuaciones diferenciales.

num\_points = 1000: Este es el número de puntos en el tiempo en el que se va a resolver el sistema de ecuaciones diferenciales.

t = np.linspace(0, 100, num\_points): Esta línea genera un array de num\_points puntos en el tiempo, desde 0 hasta 100.

sol\_list = []: Esta lista va a contener las soluciones del sistema de ecuaciones diferenciales para cada una de las condiciones iniciales.

for i in range(num\_conditions):: Este loop itera sobre cada una de las condiciones iniciales generadas.

sol = odeint(equations, x0\_list[i], t, args=(L, C, R, e, u)): Esta línea resuelve el sistema de ecuaciones diferenciales utilizando la

función odeint. Los argumentos son la función que define las ecuaciones, la condición inicial x0\_list[i], el array de puntos en el tiempo

t, y los parámetros L, C, R, e y u.

sol\_list.append(sol): La solución del sistema de ecuaciones diferenciales se añade a la lista sol\_list.

fig, ax = plt.subplots(figsize=(8, 6)): Esta línea crea un objeto figura y un objeto ejes para el gráfico.

for i in range(num\_conditions):: Este loop itera sobre cada una de las soluciones

La primera línea importa la biblioteca NumPy y la renombra como np.

La segunda línea importa la biblioteca Matplotlib y la renombra como plt.

La tercera línea importa la función odeint de la biblioteca SciPy. Esta función se utiliza para resolver sistemas de ecuaciones

diferenciales ordinarias.

La función 'equations' define el sistema de ecuaciones diferenciales ordinarias que se resolverá. La función toma cinco argumentos:

x, t, L, C, R, e, y u. La función devuelve un arreglo con las dos ecuaciones diferenciales. El primer argumento, x, es el vector de estado

del sistema. El segundo argumento, t, es el tiempo. L, C, R, e y u son constantes del sistema.

Líneas 5 a 9 definen las constantes del sistema L, C, R, u y e. Luego se calcula la constante e como la raíz cuadrada de L/C.

La línea 12 define el número de condiciones iniciales a utilizar para resolver el sistema de ecuaciones diferenciales. x0\_list es una

matriz de 100 x 2 donde cada fila representa una condición inicial aleatoria para el sistema.

La línea 15 define el tiempo de integración y el número de puntos a resolver. La función linspace de NumPy crea un arreglo con 1000

valores igualmente espaciados entre 0 y 100.

La línea 16 crea una lista vacía para almacenar las soluciones del sistema de ecuaciones diferenciales para cada condición inicial.

El bucle for en las líneas 18-22 resuelve el sistema de ecuaciones diferenciales para cada condición inicial utilizando la función odeint.

Los argumentos de la función odeint son la función equations, la condición inicial, el arreglo de tiempo t, y las constantes del sistema

L, C, R, e y u.

La función odeint devuelve un arreglo de soluciones para el sistema de ecuaciones diferenciales para cada condición inicial. La lista

sol\_list almacena cada arreglo de soluciones.

Las líneas 25-30 crean una gráfica utilizando Matplotlib para mostrar las soluciones del sistema de ecuaciones diferenciales para cada

condición inicial. El bucle for traza la primera columna de la lista sol\_list, que representa el voltaje en el sistema. La función

set\_xlabel y set\_ylabel establecen las etiquetas del eje x e y, respectivamente, y la función set\_title establece el título de la gráfica.

La última línea muestra la gráfica.

El código utiliza las siguientes funciones de Python:

import numpy as np: importa la biblioteca NumPy y renombra el módulo como "np". NumPy es una biblioteca de Python que proporciona soporte para cálculos numéricos y matriciales de alta velocidad y eficiencia.

import matplotlib.pyplot as plt: importa la biblioteca Matplotlib y renombra el módulo como "plt". Matplotlib es una biblioteca de visualización de datos de Python que permite crear gráficos, diagramas y otros tipos de visualizaciones.

from scipy.signal import StateSpace, step: importa las funciones StateSpace y step del módulo scipy.signal. Estas funciones son utilizadas para crear un sistema de espacio de estados y simular su respuesta a una entrada tipo escalón.

En resumen, el código utiliza las bibliotecas NumPy y Matplotlib para crear un sistema de espacio de estados y simular su respuesta a una entrada tipo escalón. La biblioteca scipy.signal proporciona las funciones necesarias para crear el sistema y simular su respuesta. Finalmente, se grafica la respuesta del sistema utilizando la biblioteca Matplotlib.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CODIGO 3

Este código utiliza varias funciones y librerías de Python para simular y graficar la respuesta de un sistema de espacio de estados a una

entrada tipo escalón. A continuación se explica cada componente del código:

import numpy as np: Esta línea importa la librería NumPy, que es una de las principales librerías de Python para la computación científica. El alias np se utiliza comúnmente para hacer referencia a NumPy en el código.

import matplotlib.pyplot as plt: Esta línea importa la sublibrería pyplot de la librería Matplotlib. La sublibrería pyplot proporciona

funciones para crear gráficas y visualizar datos. El alias plt se utiliza comúnmente para hacer referencia a pyplot en el código.

from scipy.signal import StateSpace, step: Esta línea importa las funciones StateSpace y step de la librería SciPy. La función StateSpace

se utiliza para definir un sistema de espacio de estados, mientras que la función step se utiliza para simular la respuesta del sistema a

una entrada tipo escalón.

A = np.array([[0, 1], [-2, -0.5]]): Esta línea define la matriz A del sistema de espacio de estados. En este caso, la matriz A tiene

dimensiones 2x2 y representa la relación entre las variables de estado del sistema.

B = np.array([[0], [1]]): Esta línea define la matriz B del sistema de espacio de estados. En este caso, la matriz B tiene dimensiones 2x1

y representa la relación entre la entrada del sistema y las variables de estado.

C = np.array([[0, 1]]): Esta línea define la matriz C del sistema de espacio de estados. En este caso, la matriz C tiene dimensiones 1x2 y

representa la relación entre las variables de estado y la salida del sistema.

D = np.array([[0]]): Esta línea define la matriz D del sistema de espacio de estados. En este caso, la matriz D tiene dimensiones 1x1 y

representa la relación entre la entrada del sistema y la salida del sistema.

sys = StateSpace(A, B, C, D): Esta línea crea el sistema de espacio de estados utilizando las matrices A, B, C y D definidas

anteriormente.

t, y = step(sys): Esta línea utiliza la función step de SciPy para simular la respuesta del sistema a una entrada tipo escalón. La función

step devuelve dos arreglos NumPy: t es el arreglo de tiempo y y es el arreglo de la salida del sistema.

plt.plot(t, y): Esta línea utiliza la función plot de Matplotlib para graficar la salida del sistema en función del tiempo.

plt.xlabel('Tiempo (s)') y plt.ylabel('Salida'): Estas líneas definen las etiquetas del eje x y del eje y de la gráfica, respectivamente.

plt.title('Respuesta a una entrada tipo escalón'): Esta línea define el título de la gráfica.

plt.show(): Esta línea muestra la gráfica en una ventana separada.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CODIGO 4

Este código se utiliza para graficar los retratos de fase de un sistema dinámico no lineal. A continuación, se explican los componentes

del código y su función:

import numpy as np: Esta línea importa la biblioteca NumPy y la renombra como "np" para abreviar el código. NumPy es una biblioteca de

Python para realizar cálculos numéricos con matrices y arreglos multidimensionales.

import matplotlib.pyplot as plt: Esta línea importa la biblioteca Matplotlib y la renombra como "plt" para abreviar el código. Matplotlib

es una biblioteca de Python para crear gráficos y visualizaciones.

C = 1, L = 1, R = 0.5, u = 1: Estas líneas definen los parámetros del sistema. En este caso, se trata de un sistema eléctrico RLC con una

fuente de voltaje constante.

e = np.sqrt(L/C): Esta línea calcula el valor de la constante e, que es la raíz cuadrada de L/C. Esta constante aparece en la ecuación

diferencial del sistema.

def f(t, x): Esta línea define la ecuación diferencial del sistema en la forma f(t, x) = dx/dt. En este caso, la función f toma un

argumento de tiempo t y un arreglo de estado x, que contiene las variables de estado x1 y x2. La función devuelve una lista con las

derivadas de x1 y x2, dx1 y dx2, respectivamente.

x1 = np.linspace(-2, 2, 20), x2 = np.linspace(-2, 2, 20): Estas líneas crean un arreglo de 20 puntos igualmente espaciados en el rango de

-2 a 2 para las variables de estado x1 y x2.

X1, X2 = np.meshgrid(x1, x2): Esta línea crea una cuadrícula de puntos en el espacio de estados a partir de los arreglos x1 y x2. La

matriz X1 contiene valores de x1 para cada punto en la cuadrícula, mientras que la matriz X2 contiene valores de x2.

DX1, DX2 = f(0, [X1, X2]): Esta línea calcula el campo vectorial en cada punto de la cuadrícula de puntos. La función f se evalúa en cada

punto de la cuadrícula con t = 0 y con el arreglo de estado x igual a [X1, X2]. La matriz DX1 contiene los valores de la derivada de x1

para cada punto, mientras que la matriz DX2 contiene los valores de la derivada de x2.

fig, ax = plt.subplots(figsize=(8, 8)): Esta línea crea una figura de Matplotlib con una sola subfigura (eje) y un tamaño de 8 por 8

pulgadas. La variable fig representa la figura, mientras que la variable ax representa el eje.

ax.set\_xlim([-2, 2]), ax.set\_ylim([-2, 2]): Estas líneas establecen los límites del eje x e y para que estén entre -2 y 2.

ax.set\_xlabel('x1'), ax.set\_ylabel('x2'): Estas líneas etiquetan los ejes x e y con los nombres x1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CODIGO 5.1

Este código utiliza la biblioteca NumPy (importada como np) para calcular los valores y vectores propios de una matriz. El objetivo es

analizar un sistema dinámico linealizado.

El primer paso es definir la matriz A del sistema linealizado, que representa la dinámica del sistema en torno a un punto de equilibrio.

En este caso, la matriz A es una matriz 2x2.

A continuación, se utiliza la función np.linalg.eig() de NumPy para calcular los valores y vectores propios de la matriz A. Los valores

propios son los valores λ tales que Ax = λx para algún vector x, y los vectores propios son los vectores x correspondientes.

El resultado se imprime utilizando la función print(), mostrando los valores propios y los vectores propios del sistema linealizado. Esto

puede ayudar a analizar la estabilidad y el comportamiento del sistema en torno a su punto de equilibrio.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CODIGO 5.2

Este código simula el comportamiento de un sistema dinámico lineal a través de la resolución numérica de sus ecuaciones diferenciales

usando la función odeint del módulo scipy.integrate. Los pasos del código son los siguientes:

Importar los módulos necesarios:

python

Copy code

import numpy as np

from scipy.integrate import odeint

import matplotlib.pyplot as plt

Definir las matrices del sistema lineal:

python

Copy code

A = np.array([[0, 1], [-2, -0.5]])

B = np.array([[0], [1]])

C = np.array([0, 1])

D = np.array([0])

Definir la función que describe el sistema dinámico en forma de ecuaciones diferenciales:

python

Copy code

def system(state, t):

x = state.reshape((2, 1))

dxdt = A @ x + B \* 1

return dxdt.flatten()

En este caso, la función system toma como entrada un vector de estado state y el tiempo t, y devuelve la derivada de state en función del

tiempo. La función reshape se utiliza para convertir el vector de estado en una matriz columna de 2x1, de modo que se pueda multiplicar

por la matriz de coeficientes A. La expresión B \* 1 representa la entrada del sistema, que se fija en 1 en este caso.

Definir las condiciones iniciales del sistema y el tiempo de integración:

python

Copy code

x0 = np.array([1, 0])

t = np.linspace(0, 10, 1000)

Resolver las ecuaciones diferenciales usando odeint:

python

Copy code

x = odeint(system, x0, t)

La función odeint integra numéricamente las ecuaciones diferenciales y devuelve la solución del sistema en forma de un arreglo x de

dimensión (1000, 2), donde cada fila corresponde a un tiempo diferente y las columnas contienen los valores de x1 y x2 respectivamente.

Graficar la solución en el plano x1-x2:

python

Copy code

plt.plot(x[:, 0], x[:, 1])

plt.xlabel("x1")

plt.ylabel("x2")

plt.title("Diagrama de espacio de estados")

plt.grid()

plt.show()

Este código grafica la solución del sistema en el plano x1-x2 mediante la función plot de matplotlib.pyplot. Los valores de x1 y x2 se

obtienen de las columnas correspondientes del arreglo x. El resto de las líneas de código se encargan de agregar etiquetas y títulos a la

gráfica y mostrarla en la pantalla.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CODIGO 6

Este código utiliza la biblioteca NumPy para realizar operaciones de álgebra lineal y diagonalización de matrices. Se trata de un proceso

para obtener la forma diagonal de un sistema de espacio de estados lineal.

En el primer bloque de código, se definen las matrices A, B, C y D que describen el sistema de espacio de estados lineal.

En el Paso 1, se utiliza la función eig de NumPy para obtener los valores y vectores propios de la matriz A.

En el Paso 2, se crea la matriz de cambio de base P utilizando los vectores propios de la matriz A como columnas.

En el Paso 3, se crea la matriz diagonal D a partir de los valores propios de la matriz A.

En el Paso 4, se calcula la matriz inversa de P.

En el Paso 5, se utiliza la matriz de cambio de base y su inversa para obtener la forma diagonal del sistema. En particular, se obtiene la

matriz A en su forma diagonal, la matriz B transformada, la matriz C transformada y la matriz D sin cambios.

La forma diagonal de un sistema de espacio de estados lineal es útil para analizar su estabilidad y comportamiento. En esta forma, los

valores propios de la matriz A corresponden a los polos del sistema, y su ubicación en el plano complejo determina su estabilidad y

dinámica.

El código utiliza varias funciones de la biblioteca NumPy de Python para realizar operaciones matemáticas. A continuación, se explica el uso de cada una de ellas:

np.array: Crea una matriz o arreglo NumPy a partir de una lista o tupla de valores.

np.linalg.eig: Calcula los valores y vectores propios de una matriz dada.

np.diag: Crea una matriz diagonal a partir de una lista de valores.

np.matmul: Realiza la multiplicación matricial entre dos matrices dadas.

np.linalg.inv: Calcula la inversa de una matriz dada.

En el código, estas funciones se utilizan para diagonalizar un sistema de ecuaciones lineales. Primero, se calculan los valores y vectores

propios de la matriz A utilizando np.linalg.eig. Luego, se crea una matriz de cambio de base P utilizando los vectores propios, y se

calcula la forma diagonal del sistema utilizando las funciones np.diag, np.matmul y np.linalg.inv. Finalmente, se imprimen los valores de

A, B, C y D en su forma diagonalizada.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CODIGO 7

Este código utiliza la librería NumPy y Matplotlib para graficar la solución fundamental de un sistema de ecuaciones diferenciales lineales.

Primero, se importan las bibliotecas necesarias, numpy y matplotlib.pyplot.

Se define la matriz A del sistema y se encuentran los valores y vectores propios usando la función eig de numpy.

Se define la función solucion\_fundamental que toma como entrada el tiempo, c1 y c2 que son constantes, y utiliza los valores y vectores

propios para calcular la solución fundamental del sistema.

La función utiliza un bucle for para iterar sobre los valores y vectores propios y calcular la solución fundamental correspondiente. Se

utiliza la función outer de numpy para realizar el producto externo entre el vector propio y las constantes c1 y c2 multiplicadas por la

exponencial del valor propio correspondiente y su conjugado.

La solución fundamental se devuelve como una matriz 2xN donde N es la longitud de la matriz de tiempo.

Finalmente, se grafica la solución fundamental utilizando la biblioteca matplotlib.pyplot. Se definen las condiciones iniciales y las

constantes c1 y c2, se llama a la función solucion\_fundamental y se grafican las soluciones x e y en función del tiempo.

La función np.array() de NumPy se utiliza para crear matrices a partir de listas.

La función np.linalg.eig() se utiliza para calcular los valores y vectores propios de una matriz cuadrada.

La función np.outer() se utiliza para calcular el producto tensorial entre dos vectores.

La función np.exp() se utiliza para calcular la exponencial de un número.

La función np.real() se utiliza para obtener la parte real de un número complejo.

La función plt.plot() de Matplotlib se utiliza para graficar la solución fundamental en el tiempo.

La función plt.xlabel() y plt.ylabel() se utilizan para agregar etiquetas a los ejes de la gráfica.

La función plt.legend() se utiliza para agregar una leyenda a la gráfica.

La función plt.show() se utiliza para mostrar la gráfica.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CODIGO 8

El código comienza importando las librerías numpy y matplotlib.pyplot bajo los alias np y plt, respectivamente.

Luego, se definen las constantes del sistema, incluyendo la inductancia L, la capacitancia C, la resistencia R, la fuente de voltaje u y

la constante e, que es la raíz cuadrada de la división de L por C.

A continuación, se define la función f(x, u) que describe la dinámica no lineal del sistema. Esta función toma como entrada un vector de

estado x y la señal de entrada u, y devuelve la derivada del vector de estado x.

Luego se define el tiempo de simulación t\_sim, el paso de tiempo dt y se genera un vector de tiempo t que va desde cero hasta t\_sim con un

paso de dt.

Se genera una señal aleatoria de perturbación usando la función numpy.random.normal() que toma una media mu, una desviación estándar sigma

y la longitud del vector t.

A continuación, se simula la dinámica no lineal del sistema con perturbación aleatoria. Primero se crea un array vacío x con la misma

forma que el vector de tiempo t, luego se asignan las condiciones iniciales a la primera fila del array x. En el bucle for, la función

f(x[i-1], u) se usa para calcular la derivada del vector de estado x en el tiempo i-1, luego se multiplica por el paso de tiempo dt para

obtener la aproximación de x en el siguiente instante de tiempo. Este resultado se suma a la fila anterior de x y se le agrega la señal de

perturbación en ese instante de tiempo, multiplicada por dt.

A continuación, se calculan las energías de las señales x1 y x2 usando la expresión para la energía en circuitos eléctricos, que es igual

a la mitad del producto de la constante asociada (inductancia o capacitancia) por el cuadrado de la señal Luego se

suman estas dos energías para obtener la energía total del sistema.

Por último, se grafican las energías de las señales x1 y x2, la energía total y la perturbación aleatoria a lo largo del tiempo. También

se grafican las barras de energía de las señales x1 y x2 al final del tiempo de simulación.

El código utiliza principalmente dos módulos de Python: numpy y matplotlib.pyplot.

numpy es una biblioteca de Python utilizada para realizar cálculos numéricos con arrays y matrices. En este código, numpy se utiliza para realizar operaciones matriciales y vectoriales, calcular valores de energía de las señales, generar señales aleatorias y generar un rango de tiempo para la simulación.

matplotlib.pyplot es una biblioteca de visualización de datos utilizada para crear gráficos y visualizaciones. En este código, matplotlib.pyplot se utiliza para crear gráficos de barras y gráficos de líneas que muestran la energía de las señales, la energía total y la perturbación aleatoria.

Además de estas bibliotecas, el código utiliza varias funciones de Python. A continuación, se explica brevemente el uso de cada una de ellas:

np.sqrt(): Calcula la raíz cuadrada de un número o una matriz.

np.arange(): Genera un array con valores equiespaciados dentro de un rango especificado.

np.random.normal(): Genera una distribución de valores aleatorios con una media y una desviación estándar especificadas.

np.zeros(): Crea un array de ceros con la forma especificada.

np.array(): Crea un array a partir de una lista o una matriz.

plt.subplots(): Crea una figura y un conjunto de subplots en una sola llamada de función.

ax.plot(): Traza líneas o marcadores en el gráfico.

ax.bar(): Crea un gráfico de barras.

ax.set\_xlabel(): Establece la etiqueta del eje x del gráfico.

ax.set\_ylabel(): Establece la etiqueta del eje y del gráfico.

ax.set\_title(): Establece el título del gráfico.

ax.legend(): Agrega una leyenda al gráfico.