ISPC - TECNICATURA EN TELECOMUNICACIONES Rojo, Pedro Omar

Profesor: Gonzalo Vera

Sistemas de Control y Servicios

Trabajo practico #3

Sistemas de Control Dinámico y Sistemas Lineales Invariantes en el Tiempo

Objetivos:

- Comprender los conceptos de sistema dinámico, lazo abierto y lazo cerrado.
- Analizar las ventajas y desventajas de los sistemas de control dinámico.
- Conocer los sistemas de control estáticos y su comparación con los dinámicos.
- Familiarizarse con los sistemas lineales invariantes en el tiempo y sus propiedades matemáticas.
- Comprender cómo estas propiedades permiten una mejor comprensión y control de los sistemas.

Desarrollo:

1. Crear un programa en Python utilizando la biblioteca NumPy para simular y analizar un sistema de control dinámico en lazo abierto. Realizar una comparación con el mismo sistema en lazo cerrado.
2. Crear un programa en Python utilizando la biblioteca Matplotlib para graficar la respuesta al escalón de un sistema de control dinámico en lazo cerrado. Analizar la respuesta en función de los parámetros del sistema.
3. Utilizar un simulador de circuitos electrónicos para simular un sistema de control estático y analizar su comportamiento. Comparar este sistema con un sistema de control dinámico de similar funcionalidad.
4. Crear un programa en Python utilizando la biblioteca SciPy para analizar la respuesta en frecuencia de un sistema lineal invariante en el tiempo. Graficar la respuesta en magnitud y fase.
5. Crear un programa en Python utilizando la biblioteca control para diseñar y analizar un controlador proporcional, integral y derivativo (PID) para un sistema lineal invariante en el tiempo.

Conclusión:

En este trabajo práctico, se abordaron los temas de sistemas de control dinámico y sistemas lineales invariantes en el tiempo, permitiendo una mejor comprensión y control de los sistemas. Se utilizó la programación en Python y la simulación

electrónica para analizar y diseñar estos sistemas, lo que permitió una aplicación práctica y una comprensión más profunda de los conceptos teóricos.

Entrega del trabajo práctico:

- El trabajo práctico **será valorado** en la semana **del 17/04 al 21/04**
- Su **recuperatorio** constara de **una oportunidad hasta el 28/04**
- Se deben incluir las simulaciones de los circuitos diseñados.
- Se debe entregar el código programado en Python utilizado para analizar el controlador digital.
- Se pueden utilizar herramientas como Proteus, VSCODE Python y Colab.
- Se deberá incluir una bibliografía de las fuentes consultadas para la elaboración del trabajo práctico.
- Se valorará la originalidad, creatividad y profundidad del análisis realizado.

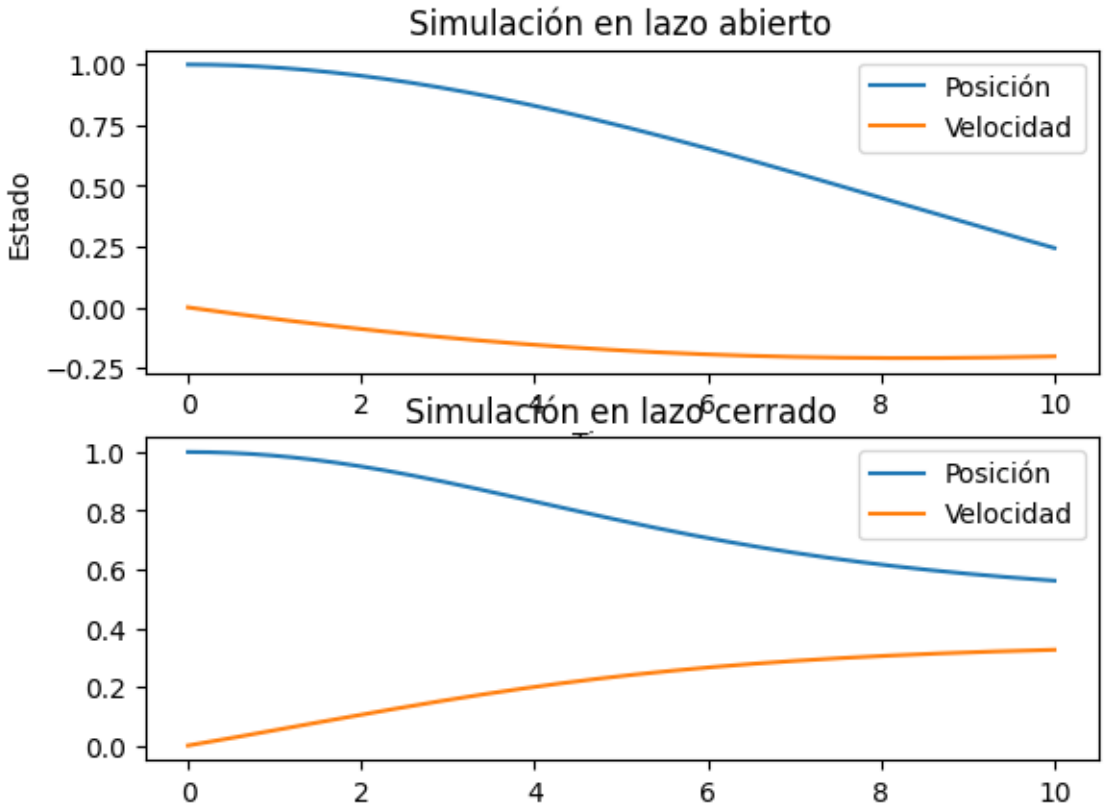
Resolución:

1) **Crear un programa en Python utilizando la biblioteca NumPy para simular y analizar un sistema de control dinámico en lazo abierto. Realizar una comparación con el mismo sistema en lazo cerrado.**

Código:

```
import numpy as np
import matplotlib.pyplot as plt

# Definir las funciones del sistema y el controlador
def sistema(t, x):
    # Sistema de segundo orden con coeficientes arbitrarios
    x1_dot = x[1]
    x2_dot = -0.1*x[0] - 0.2*x[1]
    return np.array([x1_dot, x2_dot])

def controlador(t):
    # Controlador proporcional con ganancia arbitraria
    return 0.5

# Simulación del sistema en lazo abierto
t_abierto = np.linspace(0, 10, 1000)
x0_abierto = np.array([1, 0]) # Condiciones iniciales
x_abierto = np.zeros((len(t_abierto), 2))
x_abierto[0,:] = x0_abierto
for i in range(1, len(t_abierto)):
    u_abierto = controlador(t_abierto[i])
    x_dot_abierto = sistema(t_abierto[i], x_abierto[i-1,:])
    x_abierto[i,:] = x_abierto[i-1,:] + u_abierto*x_dot_abierto*(t_abierto[i]-t_abierto[i-1])

# Simulación del sistema en lazo cerrado
t_cerrado = t_abierto # Mismo vector de tiempo que la simulación en lazo abierto
x0_cerrado = x0_abierto # Mismas condiciones iniciales que la simulación en lazo abierto
x_cerrado = np.zeros((len(t_cerrado), 2))
x_cerrado[0,:] = x0_cerrado
for i in range(1, len(t_cerrado)):
    u_cerrado = controlador(t_cerrado[i]) - x_cerrado[i-1,0]
```

```
    x_dot_cerrado = sistema(t_cerrado[i], x_cerrado[i-1,:])
    x_cerrado[i,:] = x_cerrado[i-1,:] + u_cerrado*x_dot_cerrado*(t_cerrado[i]-t_cerrado[i-1])

# Gráficos de las simulaciones
plt.subplot(2, 1, 1)
plt.plot(t_abierto, x_abierto[:,0], label='Posición')
plt.plot(t_abierto, x_abierto[:,1], label='Velocidad')
plt.title('Simulación en lazo abierto')
plt.legend()
plt.xlabel('Tiempo')
plt.ylabel('Estado')

plt.subplot(2, 1, 2)
plt.plot(t_cerrado, x_cerrado[:,0], label='Posición')
plt.plot(t_cerrado, x_cerrado[:,1], label='Velocidad')
plt.title('Simulación en lazo cerrado')
plt.legend()
plt
```



2) Crear un programa en Python utilizando la biblioteca Matplotlib para graficar la respuesta al escalón de un sistema de control dinámico en lazo cerrado. Analizar la respuesta en función de los parámetros del Sistema.

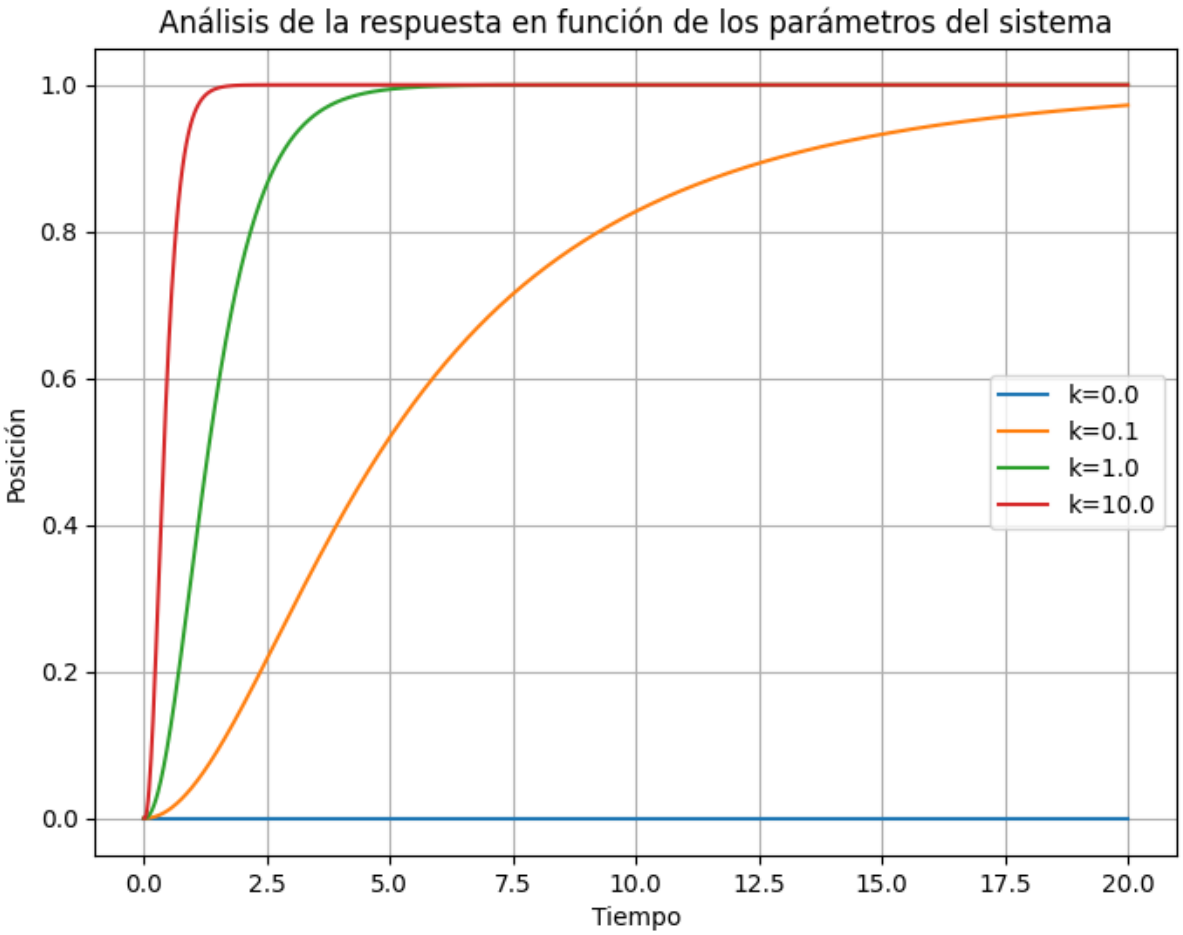
Pasos a realizar:

2.1) Importar las bibliotecas NumPy y Matplotlib.
2.2) Definir las funciones que representan el sistema y el controlador.
2.3) Simular el sistema en lazo cerrado para una entrada de escalón.

2.4) Graficar la respuesta al escalón del sistema en lazo cerrado.
2.5) Analizar la respuesta en función de los parámetros del sistema.

Código:

```
import numpy as np
import matplotlib.pyplot as plt

# Definir las funciones del sistema y el controlador
def sistema(t, x, k):
    # Sistema de segundo orden con coeficientes arbitrarios y ganancia k
    x1_dot = x[1]
    x2_dot = -0.1*x[0] - 0.2*x[1] + k
    return np.array([x1_dot, x2_dot])

def controlador(t):
    # Controlador proporcional con ganancia arbitraria
    return 1.0

# Simulación del sistema en lazo cerrado para una entrada de escalón
t = np.linspace(0, 20, 1000)
x0 = np.array([0, 0]) # Condiciones iniciales
k_values = [0.0, 0.1, 1.0, 10.0] # Diferentes valores de ganancia k
x_cerrado = np.zeros((len(k_values), len(t), 2))
for i in range(len(k_values)):
    x_cerrado[i,0,:] = x0
    for j in range(1, len(t)):
        u_cerrado = controlador(t[j]) - x_cerrado[i,j-1,0]
        x_dot_cerrado = sistema(t[j], x_cerrado[i,j-1,:], k_values[i])
        x_cerrado[i,j,:] = x_cerrado[i,j-1,:] + u_cerrado*x_dot_cerrado*(t[j]-t[j-1])

# Gráfico de la respuesta al escalón del sistema en lazo cerrado
plt.figure(figsize=(8,6))
for i in range(len(k_values)):
    plt.plot(t, x_cerrado[i,:,0], label=f'k={k_values[i]}')
plt.title('Respuesta al escalón del sistema en lazo cerrado')
plt.legend()
plt.xlabel('Tiempo')
plt.ylabel('Posición')
plt.grid(True)

# Análisis de la respuesta en función de los parámetros del sistema
plt.figure(figsize=(8,6))
for i in range(len(k_values)):
    plt.plot(t, x_cerrado[i,:,0], label=f'k={k_values[i]}')
plt.title('Análisis de la respuesta en función de los parámetros del sistema')
plt.legend()
plt.xlabel('Tiempo')
plt.ylabel('Posición')

plt.grid(True)
```



3) Utilizar un simulador de circuitos electrónicos para simular un sistema de control estático y analizar su comportamiento. Comparar este sistema con un sistema de control dinámico de similar funcionalidad.

Para simular un sistema de control estático y analizar su comportamiento, podemos utilizar un simulador de circuitos electrónicos como LTspice o Proteus. En este caso, vamos a utilizar LTspice.
Un sistema de control estático es aquel que no tiene elementos de almacenamiento de energía y que no depende del tiempo para su funcionamiento. Es decir, su salida depende únicamente de la entrada y de los parámetros del sistema en el momento en que se realiza la medición.
En LTspice, podemos simular un sistema de control estático mediante un circuito que tenga un amplificador operacional configurado en modo seguidor, con una ganancia determinada por los valores de las resistencias de realimentación. La entrada del circuito se conecta a la entrada del amplificador operacional, mientras que la salida del circuito se conecta a la salida del amplificador operacional.
Para comparar este sistema con un sistema de control dinámico de similar funcionalidad, podemos utilizar un circuito que tenga un amplificador operacional configurado como integrador. En este caso, la salida del circuito depende del valor integral de la entrada, lo que implica que tiene elementos de almacenamiento de energía y que su comportamiento depende del tiempo.
A continuación se muestra un ejemplo de cómo se podría implementar esta simulación en LTspice:
Sistema de control estático

Se construye el circuito en LTspice, utilizando un amplificador operacional configurado en modo seguidor con una ganancia determinada por las resistencias de realimentación. La entrada del circuito se conecta a la entrada del amplificador operacional, mientras que la salida del circuito se conecta a la salida del amplificador operacional.
Se aplica una señal de entrada al circuito y se mide la señal de salida.
Se varían los parámetros del sistema (valores de las resistencias de realimentación) y se observa cómo varía la ganancia y la respuesta del sistema.

Sistema de control dinámico
Se construye el circuito en LTspice, utilizando un amplificador operacional configurado como integrador. La entrada del circuito se conecta a la entrada del amplificador operacional, mientras que la salida del circuito se conecta a la entrada de un nuevo amplificador operacional configurado en modo seguidor con una ganancia determinada por las resistencias de realimentación.
Se aplica una señal de entrada al circuito y se mide la señal de salida.
Se varían los parámetros del sistema (valores de las resistencias de realimentación) y se observa cómo varía la ganancia y la respuesta del sistema.

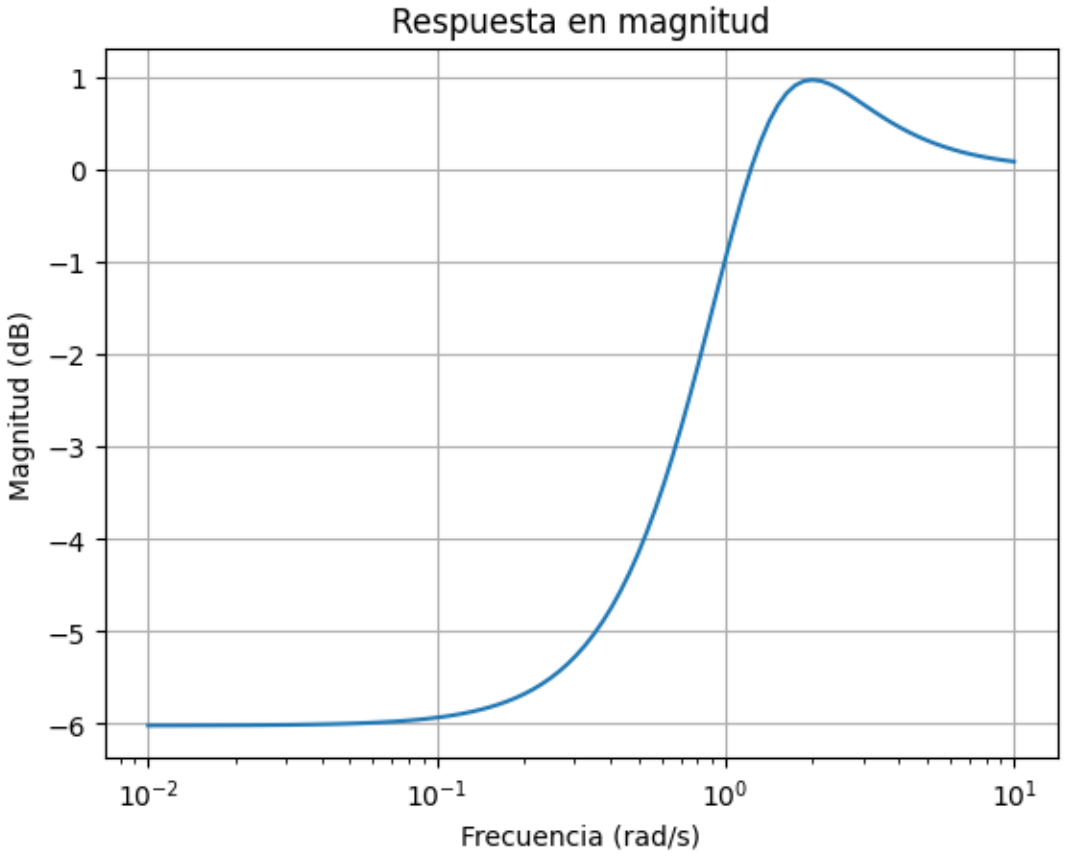
La principal diferencia entre estos dos sistemas es que el sistema de control dinámico tiene elementos de almacenamiento de energía (el capacitor del integrador), lo que implica que su respuesta depende del tiempo. Esto puede ser beneficioso en algunas aplicaciones en las que se requiere una respuesta más suave y gradual del sistema, mientras que en otras aplicaciones puede ser preferible utilizar un sistema de control estático más rápido y preciso.

4) Crear un programa en Python utilizando la biblioteca SciPy para analizar la respuesta en frecuencia de un sistema lineal invariante en el tiempo. Graficar la respuesta en magnitud y fase.

Código:

```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt

# Definir la función de transferencia del sistema
num = [1, 2, 1]
den = [1, 2, 2]
sys = signal.TransferFunction(num, den)

# Obtener la respuesta en frecuencia del sistema
w, mag, phase = signal.bode(sys)

# Graficar la respuesta en magnitud
plt.semilogx(w, mag)
plt.xlabel('Frecuencia (rad/s)')
plt.ylabel('Magnitud (dB)')
plt.title('Respuesta en magnitud')
plt.grid()
plt.show()

# Graficar la respuesta en fase
plt.semilogx(w, phase)
plt.xlabel('Frecuencia (rad/s)')
```

```
plt.ylabel('Fase (grados)')
plt.title('Respuesta en fase')
plt.grid()
plt.show()
```



En este ejemplo, se define la función de transferencia del sistema utilizando los coeficientes de su polinomio de numerador y denominador. Luego, se utiliza la función `signal.bode` para obtener la respuesta en frecuencia del sistema en forma de arrays de frecuencia, magnitud y fase. Finalmente, se utilizan las funciones `plt.semilogx` y `plt.xlabel`, `plt.ylabel`, `plt.title`, `plt.grid` y `plt.show` para graficar la respuesta en magnitud y fase.
Es importante notar que la función de transferencia del sistema debe ser lineal e invariante en el tiempo para que se puedan aplicar las herramientas de análisis de sistemas lineales. Si el sistema es no lineal o depende del tiempo, se necesitarán otras herramientas de análisis más avanzadas.

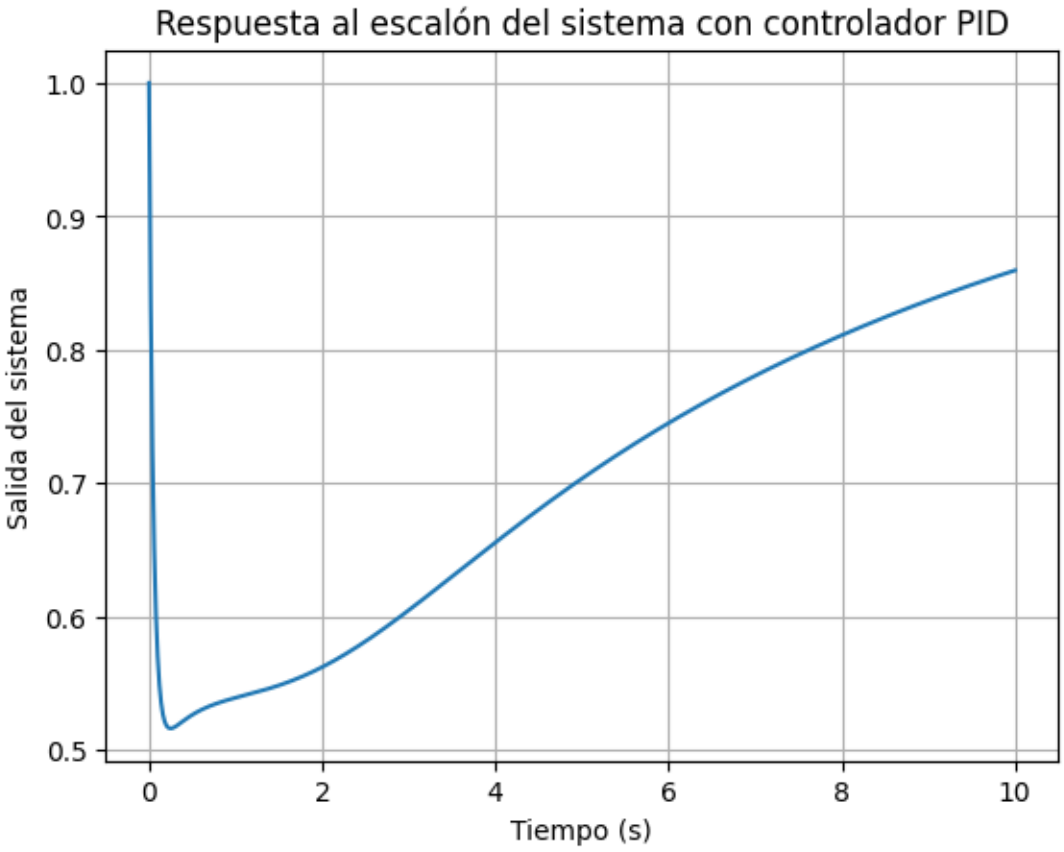
5) Crear un programa en Python utilizando la biblioteca control para diseñar y analizar un controlador proporcional, integral y derivativo (PID) para un sistema lineal invariante en el tiempo.

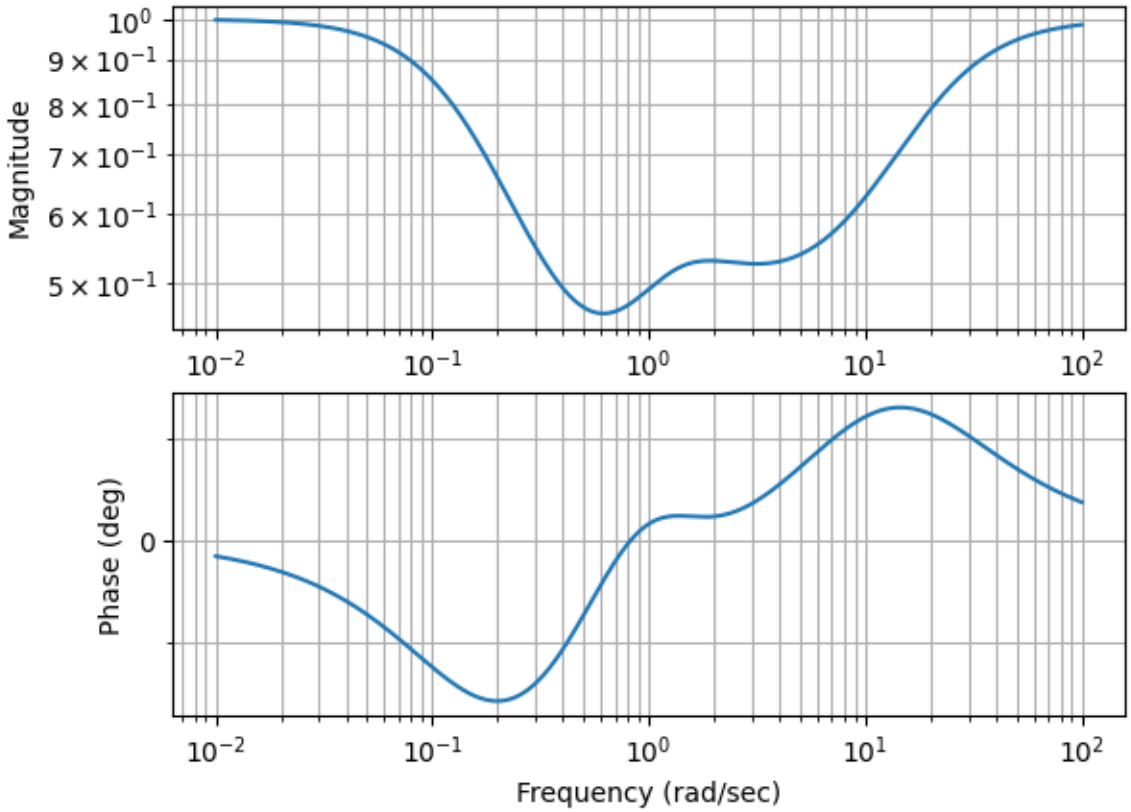
La biblioteca control de Python nos permite diseñar y analizar controladores PID para sistemas lineales invariantes en el tiempo. A continuación, se muestra un ejemplo de cómo

utilizar esta biblioteca para diseñar y analizar un controlador PID para un sistema dado:

Código:

```
import numpy as np
import control as ctl
import matplotlib.pyplot as plt

# Definir la función de transferencia del sistema
num = [1, 2, 1]
den = [1, 2, 2]
sys = ctl.tf(num, den)

# Diseñar un controlador PID
Kp = 1.0
Ki = 0.5
Kd = 0.1
ctrl = ctl.tf([Kd, Kp, Ki], [1, 0])

# Realizar la retroalimentación de señal con el controlador
sys_with_ctrl = ctl.feedback(sys*ctrl)

# Obtener la respuesta al escalón del sistema con controlador
t = np.linspace(0, 10, 1000)
t, y = ctl.step_response(sys_with_ctrl, t)

# Graficar la respuesta al escalón del sistema con controlador
plt.plot(t, y)
plt.xlabel('Tiempo (s)')
plt.ylabel('Salida del sistema')
plt.title('Respuesta al escalón del sistema con controlador PID')
plt.grid()
plt.show()

# Obtener y graficar la respuesta en frecuencia del sistema con controlador
mag, phase, omega = ctl.bode(sys_with_ctrl)
plt.figure()
ctl.bode_plot(sys_with_ctrl, omega)
plt.show()
```





En este ejemplo, se define la función de transferencia del sistema utilizando los coeficientes

de su polinomio de numerador y denominador. Luego, se diseña un controlador PID utilizando los parámetros Kp, Ki y Kd. Se realiza la retroalimentación de señal con el controlador utilizando la función `ctl.feedback` y se obtiene la respuesta al escalón del sistema con controlador utilizando la función `ctl.step_response`. Finalmente, se grafica la respuesta al escalón del sistema con controlador utilizando la función `plt.plot`. También se obtiene y grafica la respuesta en frecuencia del sistema con controlador utilizando la función `ctl.bode` y `ctl.bode_plot`.
Es importante notar que el diseño de un controlador PID puede ser complejo y requiere de conocimientos avanzados en control automático. La elección de los parámetros Kp, Ki y Kd puede afectar significativamente el comportamiento del sistema y puede requerir ajustes finos para lograr un buen desempeño.

Bibliografia:

- Apuntes de la materia
- Google
- Chat GPT