MACHINE PROBLEM: LAPLACE TRANSFORM AND S-DOMAIN ANALYSIS USING PYTHON

OBJECTIVES

- To apply the principles of Laplace Transform and Inverse Laplace Transform to solve engineering problems.
- To understand the concept of zeros and poles in the s-domain and their significance in system stability and response.
- To utilize Python for computational analysis and visualization of mathematical concepts.

LEARNING OUTCOMES

Upon completion of this machine problem, students will be able to:

- Compute Laplace Transforms and Inverse Laplace Transforms of time-domain functions using Python.
- Identify and plot zeros and poles in the s-domain for given transfer functions.
- Interpret the s-domain plots in terms of system behavior and stability.

DISCUSSION

The Laplace Transform is an integral tool in control systems and signal processing that transforms a time-domain function into the complex frequency domain, making it easier to handle differential equations and convolution. Zeros and poles in the s-domain are critical to understanding system behavior, where zeros cancel out poles, and the position of poles relative to the imaginary axis indicates system stability.

- **Handling Differential Equations**: The Laplace Transform simplifies differential equations into algebraic equations in the s-domain. This simplification is possible because differentiation in the time domain corresponds to multiplication by *s* in the s-domain, turning complex differential equations into simpler polynomial forms.
- **Convolution**: Convolution in the time domain, which is a complex operation involving integrals, becomes multiplication in the s-domain. This property significantly simplifies the analysis of systems described by convolution, such as linear time-invariant (LTI) systems.
- **Zeros** of a transfer function are the values of s that make the numerator of the function zero. They represent the frequencies at which the system's output is minimized. In the s-domain plot, zeros can affect the system's frequency response but do not directly determine system stability.
- **Poles** are the roots of the denominator of a transfer function and are critical to the system's behavior. The system's stability is determined by the location of its poles in the s-domain:
    - A system is considered stable if all poles are in the left half of the complex plane (having negative real parts).
    - Poles on the right half-plane indicate an unstable system.
    - Poles exactly on the imaginary axis suggest a marginally stable system, potentially leading to sustained oscillations.

Libraries Used in the Program

- **SymPy**: A Python library for symbolic mathematics. It provides tools to perform algebraic manipulations, solve equations symbolically, and perform calculus operations among many others. In the context of the Laplace Transform, SymPy allows for the definition of symbolic expressions for

time-domain functions, computes their Laplace and inverse Laplace transforms, and solves for zeros and poles symbolically.

- **Matplotlib**: A plotting library in Python that provides functions for a wide range of static, animated, and interactive visualizations. It is used here to plot the zeros and poles on the complex s-plane, providing a visual interpretation of the system's stability and frequency response characteristics.

General Syntax for Laplace Transforms:

- **Defining Symbols**: **sp.symbols('t s', real=True)** is a standard way to declare symbolic variables in SymPy, where **t** and **s** are commonly used symbols representing time and the complex frequency domain variable, respectively.
- **Heaviside and DiracDelta Functions**: Utilizing **sp.Heaviside(t)** and **sp.DiracDelta(t)** to represent step inputs and impulse functions, respectively, is a direct application of SymPy's capabilities to model common functions encountered in control systems and signal processing.
- **Laplace Transform**: The function **sp.laplace_transform(f, t, s, noconds=True)** is the standard method provided by SymPy to compute the Laplace Transform of a time-domain function **f(t)**. The argument **noconds=True** is used to return the transform directly without additional conditions that might sometimes accompany the result.
- **Inverse Laplace Transform**: Similarly, **sp.inverse_laplace_transform(F, s, t)** computes the inverse Laplace Transform, converting a function from the s-domain back to the time domain.
- **Solving for Zeros and Poles**: Using **sp.solveset(equation, variable, domain=sp.S.Complexes)** to find the roots of the numerator and denominator for zeros and poles, respectively, is a general approach in symbolic computation to solve equations. Specifying the domain as **sp.S.Complexes** ensures that the solution set includes complex numbers, which is essential for analyzing systems in the s-domain.
- **Simplification and Fraction Extraction**: **sp.simplify(F)** and **sp.fraction(F_simplified)** are used to simplify the Laplace Transform expression and then extract its numerator and denominator. This step is crucial for identifying the transfer function's zeros and poles.

Plotting with Matplotlib:

- **Visualization**: The use of Matplotlib's **plt.scatter** for plotting zeros and poles provides a visual interpretation of their distribution in the complex plane. This visualization is key to understanding the system's stability and response characteristics based on the location of poles and zeros.

GUIDELINES

Step 1: Environment Setup

Ensure Python and necessary libraries (sympy and matplotlib) are installed:

```
pip install sympy matplotlib
```

Step 2: Import Libraries

In your Python script, import the required libraries:

```
import sympy as sp
import matplotlib.pyplot as plt
```

Step 3: Define Time-Domain Functions

Set up the symbolic variables and define your function

```
# Define symbols for time t and complex frequency s
t, s = sp.symbols( names: 't s', real=True)

# Define the Heaviside step function u(t) for the unit step function
u = sp.Heaviside(t)

# Define the time-domain function f(t)
f = sp.DiracDelta(t) + (8 * sp.exp(-3*t) + 2 * sp.exp(-2*t) * (sp.cos(3*t) + sp.sin(3*t)) - 5 * sp.exp(-4*t)) * u
```

Here, t and s are defined as symbolic variables with t representing time and s the complex frequency domain variable. The function f(t) is defined using SymPy's symbolic expressions, including the Dirac delta function, exponentials, and trigonometric functions, modulated by the Heaviside step function u.

Step 4: Compute Laplace Transform

Calculate the Laplace Transform

```
F = sp.laplace_transform(f, t, s, noconds=True)
```

Step 5: Zeros and Poles

Determine zeros and poles of the function

```
# Extract the numerator and denominator
numerator, denominator = sp.fraction(F_simplified)

# Determine zeros and poles
zeros = sp.solveset(numerator, s, domain=sp.S.Complexes)
poles = sp.solveset(denominator, s, domain=sp.S.Complexes)
```

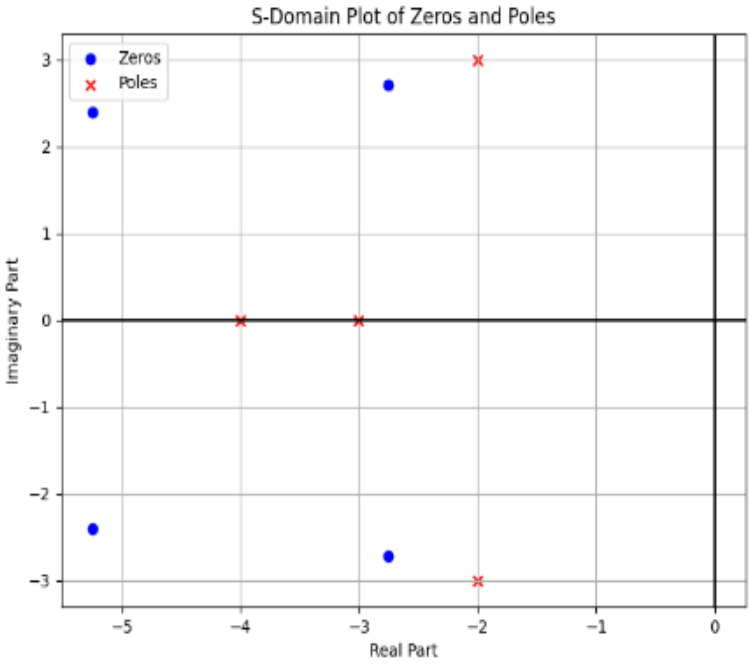
Step 6: Plot S-Domain

Plot the zeros (o) and poles (x) on the s-plane. Convert zeros and poles to numerical values and plots them on the complex s-plane. Zeros are marked with , and poles with crosses, illustrating their distribution in the s-domain.

```
# Create a new figure with specified size
plt.figure(figsize=(8, 6))

# Plot zeros on the complex plane
plt.plot( *args: real_parts_of_zeros, imaginary_parts_of_zeros, 'o', label='Zeros', markersize=10)

# Plot poles on the complex plane
plt.plot( *args: real_parts_of_poles, imaginary_parts_of_poles, 'x', label='Poles', markersize=10)

# Set the labels for the axes
plt.xlabel('Real Part')
plt.ylabel('Imaginary Part')

# Set the title of the plot
plt.title('S-Domain Plot: Zeros and Poles')

# Add a legend to the plot
plt.legend()

# Enable the grid
plt.grid(True)

# Draw horizontal and vertical axis lines
plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')

# Display the plot
plt.show()
```

Step 8: Visualization and Interpretation

Discuss the results. Explain how the location of zeros and poles affects the system's response and stability. This discussion will be based on the plotted s-domain graph, analyzing how the proximity of poles to the imaginary axis or right half-plane can indicate potential instability or oscillatory behavior.



TASKS

- The console output will display the Laplace Transforms of each function.
- A plot will visualize the zeros and poles on the complex s-plane, providing insight into the system's stability and response characteristics. Use the tick mark ‘x’ for poles and ‘o’ for zeroes.
- The report will contain a detailed interpretation of the plot in terms of system stability and response, emphasizing the significance of the zeroes and poles' locations.

A. PROBLEMS:

1. $f(t) = 3\delta(t) + [4e^{-2t} + 8e^{-5t}\cos 3t - 2e^{-4t}\sin 5t - \ 6e^{-3t}]\mu(t)$

2. $f(t) = 8\delta(t) + [7e^{-3t} - 8e^{-4t} + 4cosh2t + \ 6sinh5t]\mu(t)$

3. $f(t) = [7t^3e^{-3t} + 5e^{-3t}cosh4t - 5\sin 4tcos2t + \ 2t^3coshh3t]\mu(t)$

4. $f(t) = \sin t\, u(t) + 2\sin(t-\pi)u(t-\pi) - e^{-2(t-2\pi)}u(t-2\pi)$

5. $f(t) = \dfrac{cosh3t\sin 2t}{te^{4t}}$

QUESTIONS:

1. What is the Laplace Transform, and how does it apply to signal processing and control systems?
2. Explain the significance of zeros and poles in the s-domain and how they affect system stability.
3. How do the sympy and matplotlib libraries facilitate the computation and visualization of Laplace Transforms in Python? Provide examples.
4. How zeros near the poles affect the system's frequency response?