LABORATORY EXPERIMENT: INVERSE LAPLACE TRANSFORM USING PYTHON

OBJECTIVES

- To apply the principles of Inverse Laplace Transform to convert functions from the frequency domain back to the time domain.
- To enhance understanding of the relationship between s-domain and time-domain representations in engineering applications.
- To utilize Python for computational analysis and visualization of mathematical concepts.

LEARNING OUTCOMES

Upon completion of this laboratory experiment, students will be able to:

1. Compute Inverse Laplace Transforms of s-domain functions using Python.
2. Relate the behavior of complex systems in the time-domain by interpreting their s-domain representations.
3. Visualize time-domain responses based on given s-domain functions.

DISCUSSION

The Inverse Laplace Transform is crucial for analyzing systems in the time domain when their behavior is known in the s-domain. This process is essential for understanding how electronic and control systems behave in real-time applications.

- Concept Review: The Inverse Laplace Transform converts functions from the frequency domain (s-domain) back to the time domain (t-domain). This is particularly useful in control systems and electronic circuits where understanding the time-domain behavior is necessary for design and analysis.
- Mathematical Foundations: Using properties of linearity, complex conjugation, and frequency shifting, the Inverse Laplace Transform can be applied to complex functions to deduce their behavior in the time domain.

LIBRARIES USED IN THE PROGRAM

- SymPy: For symbolic mathematics to perform algebraic manipulations and compute Inverse Laplace Transforms.
- Matplotlib: For plotting time-domain responses to visualize the effects of different s-domain 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

1. Environment Setup: Ensure Python and necessary libraries (SymPy and Matplotlib) are installed.

```
pip install sympy matplotlib
```

2. Import Libraries: Import required libraries in your Python script.

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

3. Define s-Domain Functions: Set up symbolic variables for s (complex frequency domain) and define your s-domain function, F(s).

```python
s = sp.symbols('s')
F = (3*s + 5)/(s**2 + 2*s + 5)  # Example function
```

4. Compute Inverse Laplace Transform: Calculate the Inverse Laplace Transform to convert F(s) back to f(t), the time-domain function.

```python
t = sp.symbols('t', positive=True)
f = sp.inverse_laplace_transform(F, s, t)
```

5. Visualization: Plot the resulting time-domain function f(t) to visualize how the system behaves over time.

```python
sp.plot(f, (t, 0, 10), title='Time Domain Response', ylabel='f(t)')
```

6. Analysis and Interpretation: Discuss how the characteristics observed in the time-domain plot relate to the s-domain representation. Analyze stability and transient behavior based on the time-domain response.

GUIDELINES FOR SOLVING DIFFERENTIAL EQUATIONS USING INVERSE LAPLACE TRANSFORM

1. Environment Setup:

Ensure that Python and the necessary libraries (SymPy and Matplotlib) are installed. These can be installed using pip if not already available:

```
pip install sympy matplotlib
```

2. Import Libraries:
   Import the required libraries to handle symbolic mathematics and plotting:

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

3. Define the Differential Equation:

Define the symbolic variables and the differential equation in the time domain. Also, specify initial conditions if available:

```
t, s = sp.symbols('t s')
Y = sp.Function('Y')(s)  # Laplace transform of y(t)

# Initial conditions
y0 = 0  # y(0)
yp0 = 0  # y'(0)
```

4. Apply the Laplace Transform:

Transform the differential equation to the s-domain by applying the Laplace Transform, considering initial conditions:

```
# Example differential equation: y'' + 3y' + 2y = delta(t)

# Laplace transform of the equation considering initial conditions

equation = sp.Eq(s**2 * Y - s * y0 - yp0 + 3 * (s * Y - y0) + 2 * Y, sp.DiracDelta(t))
```

5. Solve the Algebraic Equation in the s-domain:
   Solve the transformed equation for $Y(s)$

```
solution_s = sp.solve(equation, Y)
F_s = solution_s[0]  # The Laplace Transform of the solution
```

6. Compute the Inverse Laplace Transform:
   Convert the s-domain function back to the time domain using the Inverse Laplace Transform:

```
f_t = sp.inverse_laplace_transform(F_s, s, t)
```

TASKS

1. The console output will display the Inverse Laplace Transforms of each s-domain function and differential equations
2. A plot will visualize the time-domain response, providing insight into the system's behavior over time.

PROBLEMS

1. $F(s) = \frac{6s+20}{s^3+8s^2+20s+32}$
2. $F(s) = \frac{s^2+6s+34}{s^3+7s^2+25s+50}$
3. $F(s) = \frac{s^3+18s^2+115s+250}{s^5+12s^4+74s^3+232s^2+376s+240}$
4. $y'' + 4y' + 3y = cos\,(2t),\;\; y(o) = 1\;\; y'(0) = 0$
5. $y'' + 2y' + 5y = e^{-t},\;\; y(o) = 0\;\; y'(0) = 1$

QUESTIONS

1. What is the Inverse Laplace Transform and its significance in analyzing systems?
2. How do time-domain responses help in understanding system stability and behavior?
3. Discuss the role of SymPy and Matplotlib in transforming and visualizing Inverse Laplace Transforms.
4. How does the Laplace Transform simplify the solution process for differential equations in system analysis?