ECE 351-51

March 1, 2022

Lab 6

Submitted By :
Kennedy Beach

Contents

1 Introduction

The purpose of this lab was to compare the hand-calculated and Python generated partial fraction expansions of some differential equations. A comparison of the step response generated by the cosine method, using the residue and poles determined by scipy.signal.residue(), and scipy.signal.step() was also performed.

2 Equations

The following equation was transformed into a transfer function:

$$y^{''}(t) + 10y^{'}(t) + 24y(t) = x^{''}(t) + 6x^{'}(t) + 12x(t)$$

$$H(s) = \frac{s^2 + 6s + 12}{s^2 + 10s + 24}$$

From the transfer function, the step response was found:

$$y(t) = (\frac{1}{2} - \frac{1}{2}e^{-4t} + e^{-6t})u(t)$$

The second equation analyzed is:

$$y^{(5)}(t) + 18y^{(4)}(t) + 218y^{(3)}(t) + 2036y^{(2)}(t) + 25250y^{(1)}(t) = 25250x(t)$$

3 Methodology

The first part involved plotting the step response y(t) found in the prelab from 0 to 2 seconds. The transfer function found in the prelab was also plotted over the same time using scipy.signal.step().

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


def u(t):
    y = np.zeros(t.shape)

    for i in range(len(t)):
        if t[i] >= 0:
            y[i] = 1
        else:
```

```
            y[i] = 0
    return y

steps = 1e-5
t1 = np.arange(0, 2 + steps, steps)

num1 = [1, 6, 12]
den1 = [1, 10, 24]

y1 = (0.5-0.5*np.exp(-4*t1)+np.exp(-6*t1))*u(t1)
tout, yout = sig.step((num1, den1), T = t1)
```

Listing 1: Step response and transfer function plots

Using scipy.signal.residue(), the residues, poles, and gain were output. The function used was the transer function multiplied by the Laplace transform of a step function which is 1/s.

```
num2 = [1, 6, 12]
den2 = [1, 10, 24, 0]

R1, P1, K1 = sig.residue(num2,den2)
```

Listing 2: sig.residue() of the first differential equation

Next was to use scipy.signal.residue() to find the residue, poles, and gain of the second equation differential equation.

```
num3 = [25250]
den3 = [1, 18, 218, 2036, 9085, 25250, 0]

R2, P2, K2 = sig.residue(num3,den3)
```

Listing 3: sig.residue() of the second differential equation

These results were then input into a function that performs the Cosine Method, and then that function was plotted from 0 to 4.5 seconds. A comparison using scipy.signal.step() was made.

```
def cosine_method(R, P, t):
    y = np.zeros(t.shape)

    for i in range(len(R)):
        mag_k = np.abs(R[i])
        angle_k = np.angle(R[i])
        alpha = np.real(P[i])
        omega = np.imag(P[i])

        y = y + mag_k * np.exp(alpha*t) * np.cos(omega*t + angle_k)
    * u(t)
```

```
    return y

t2 = np.arange(0, 4.5 + steps, steps)

y2 = cosine_method(R2, P2, t2)

num4 = [25250]
den4 = [1, 18, 218, 2036, 9085, 25250]
tout2, yout2 = sig.step((num4, den4), T = t2)
```

Listing 4: Cosine Method

4 Results

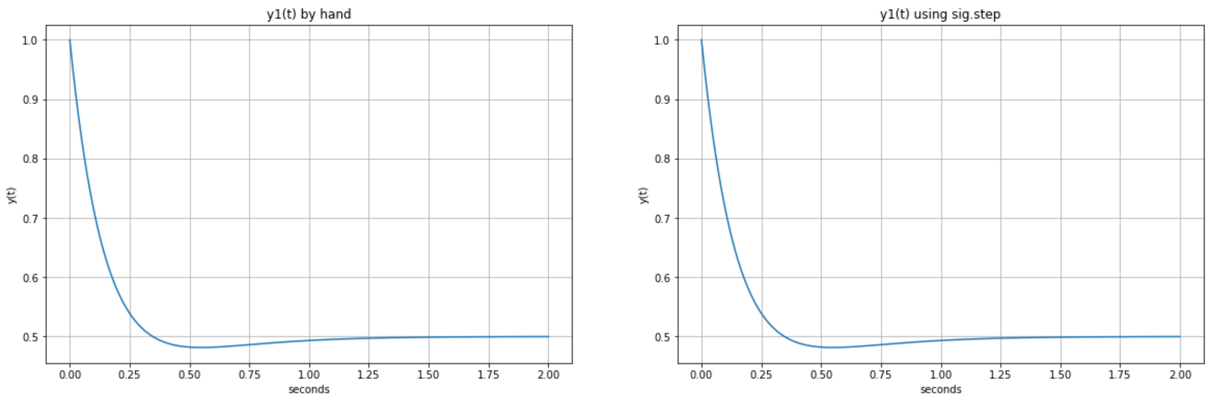

Figure 1: Plots of the first differential equation

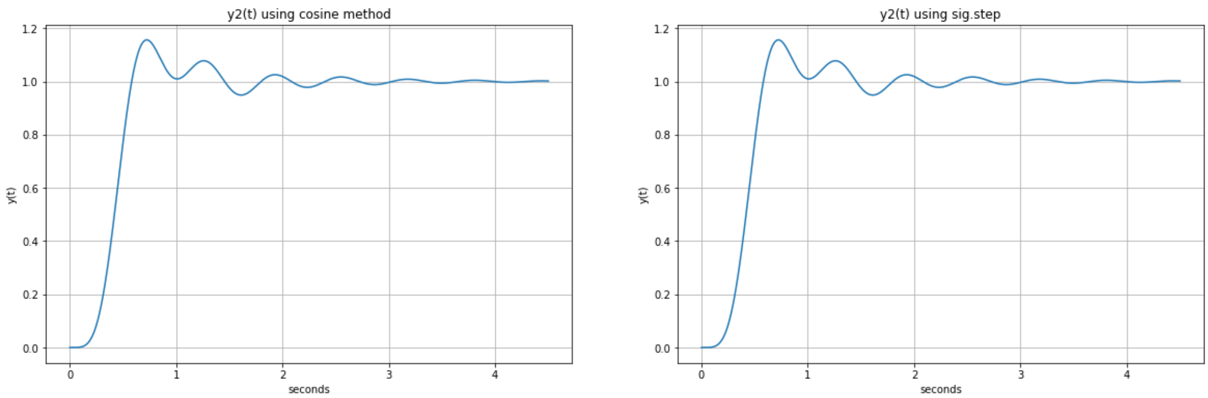

Figure 2: Plots of the second differential equation

```
In [63]: runfile('D:/College/Spring 2022/ECE_351/Lab6/Beach_Kennedy_ECE351_Lab6.py', wdir='D:/College/Spring 2022/ECE_351/Lab6')
R1:  [ 0.5 -0.5  1. ]
P1:  [ 0. -4. -6.]
K1:  []
R2:  [ 1.        +0.j         -0.48557692+0.72836538j -0.48557692-0.72836538j
 -0.21461963+0.j          0.09288674-0.04765193j  0.09288674+0.04765193j]
P2:  [  0. +0.j   -3. +4.j   -3. -4.j  -10. +0.j   -1.+10.j   -1.-10.j]
K2:  []
```

Figure 3: Printed output of the residue (R), poles (P), and gain (K)

5 Error Analysis

I ran into an issue where the y of my cosine method function wouldn't return. I had to change my step function from what it was in previous labs in order for the function to return. An empty array with the length of t was defined, then a for loop went through each value of i to see if the step function would return a 1 or 0. Previously, an if/else statement was used where if t was equal to or greater than zero, 1 would be returned. Otherwise, a 0 would be returned.

6 Questions

1. For a non-complex pole-residue term, you can still use the cosine method, explain why this works.

If a pole-residue term is not complex, then omega and angle of k are zero. Since these terms are in the cosine function, the cosine would return a value of 1. The new equation would be

$$y_c(t) = |k|e^{\alpha t}u(t)$$

2. Leave any feedback on the clarity of the expectations, instructions, and deliverables.

Everything was clear on what needed to be done and turned in.

7 Conclusion

I was a little confused about how to implement the cosine method until I realized that the numpy library has functions to get the real and imaginary parts of a result. Once I realized that, a new error where I couldn't return the value of the cosine method appeared. This was fixed by redoing my step function that I have used in previous labs. The Python and LaTeX code are seen in `https://github.com/Eniac618/ECE351_Code` and `https://github.com/Eniac618/ECE351_Reports` respectively.