ECE 351 - Section 53

Partial Fraction Expansion

Lab 6

Submitted By :
Chadwick Goodall

Contents

1 Introduction

The objective of this lab was to utilize the scipy.signal.residue function in order to calculate the partial fraction expansion of a differential equation.

The code for this project can be found at the below github repository.

GitHub: `https://github.com/Chadwick-g`

2 Equations

Prelab transfer function:

$$H(s) = \frac{S^2 + 6S + 12}{(S+4)(S+6)} \tag{1}$$

Prelab equation:

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

Part 2 equation:

$$y^{(5)}(t) + 18y^{(4)}(t) + 218y^{(3)}(t) + 2036y^{(2)}(t) + 9085y^{(1)}(t) = 25250x(t) \tag{3}$$

3 Methodology

For this lab I utilized the scipy.signal.residue function in order to verify that the hand calculated step response of the prelab was properly calculated. Additionally, I used the matplotlib.pyplot module to verify this visually after completing the calculation with the residue function.

```
y_t = (0.5 - 0.5*np.exp(-4*t) + np.exp(-6*t))*cs.u(t)

num = [1, 6, 12]
den = [1, 10, 24]

lti = sig.lti(num,den)
tout, yout = sig.step(lti)

# graphs

num = [0, 1, 6, 12]
den = [1, 10, 24, 0]
```

```
r, p, k = sig.residue(num, den)

print("residues: " + str(r) + "\npoles: " + str(p) + "\ncoefficients: " +
```

The above code listing illustrates the minimal code necessary for calculating the step response of the prelab function as well as the calculation for the partial fraction expansion of the Y(S) function (Transfer function multiplied by the step function).

Moving forward I then repeated a similar process for part two. I began by using the residue function on the lengthy expression from part 2 (equation 3 in the equations section). After finding the partial fraction expansion of this equation I then developed a custom cosine function to calculate the time-domain response using the cosine method on the results. The function is listed below.

```
def cos_pfd(r, p, t):
    y = 0
    for i in range (0, len(r)):
        k_mag = np.absolute(r[i])
        k_angle = np.angle(r[i])
        a = np.real(p[i])
        w = np.imag(p[i])
        y += k_mag*np.exp(a*t)*np.cos(w*t + k_angle) * cs.u(t)
    return y
```

The function effectively sums all of the results from the output of the residue function in order to arrive at a result for the cosine method.

Lastly I then utilized the scipy.signal.step function once more in order to find the step response of the transfer function from the residue function.

4 Results

Below are the resulting graphs that I constructed for this lab.

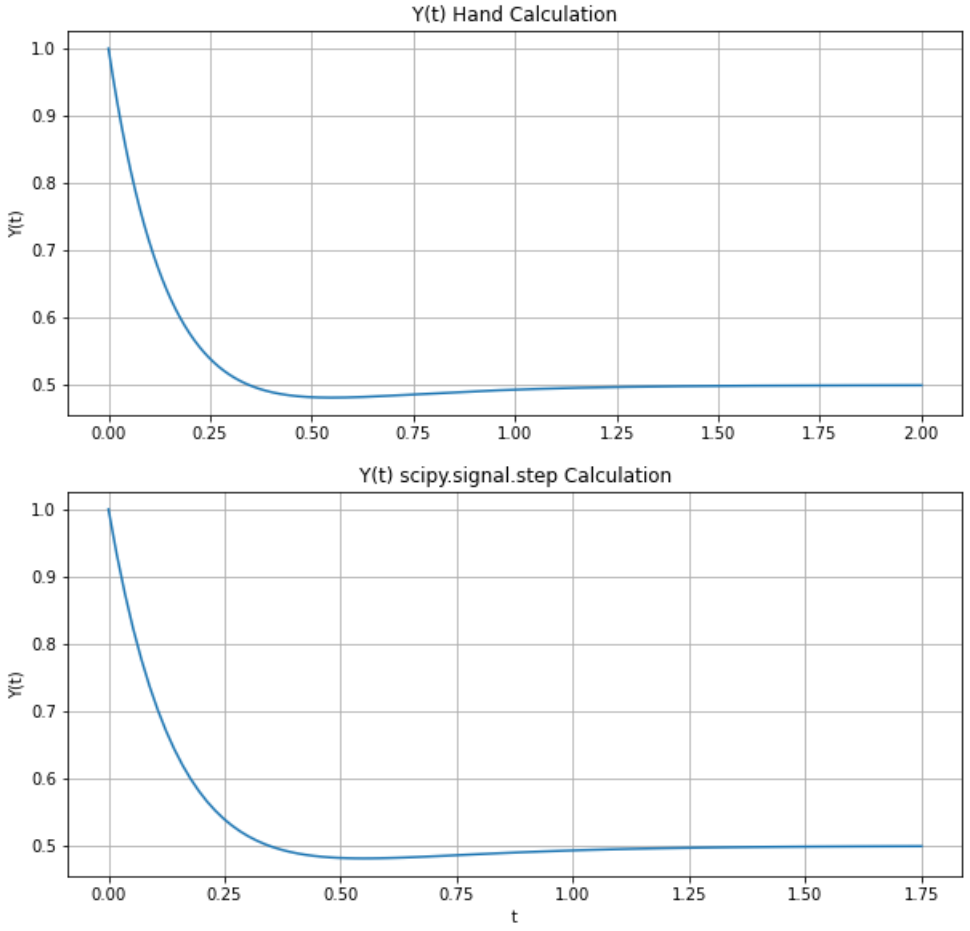

Figure 1: Step response of the prelab function

Here it can be observed that the two step response graphs from the hand solved prelab equation as well as the the scipy.signal.step function are Identical as they should be, indicating the hand solved equation was in fact correct.

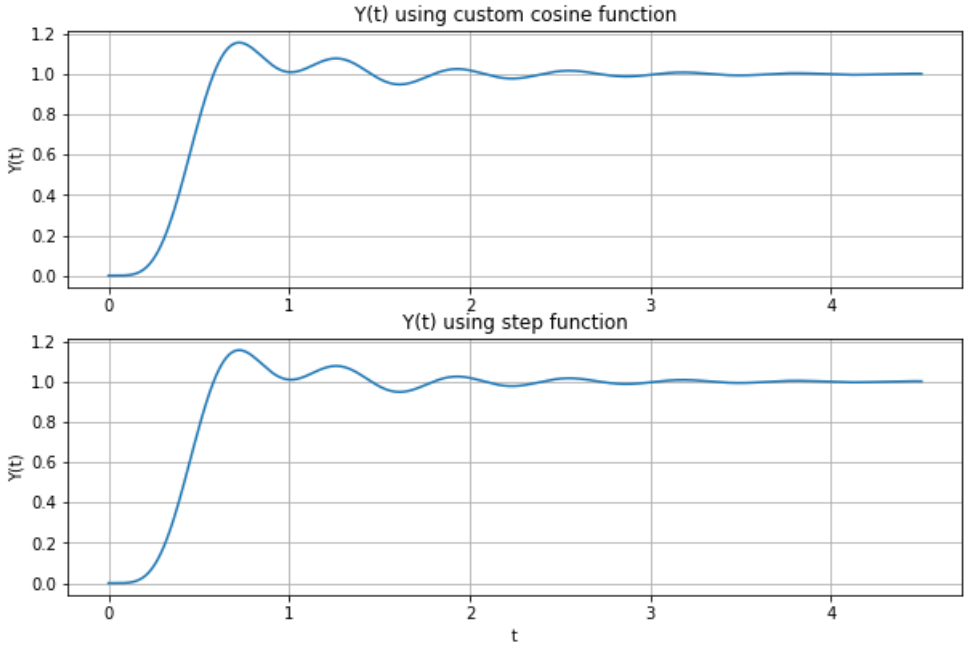

Figure 2: Step response

Furthermore analyzing the graph constructed by my custom cosine function and the step function clearly shows that they are identical further verifying that my user defined function is correct.

The printed outputs of the residue function can be viewed at the end of the document in the appendix section.

5 Error Analysis and Difficulties

The most challenging portion of this lab I would say personally was the development of the function to conduct the the partial fraction decomposition of the rather complex differential equation in Part 2.

6 Questions

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

For a non-complex pole-residue term, it is still possible to use the cosine method due to the fact that the only portion of the solution where the complex component

manifests itself is inside of the cosine term. As a result, if this term is zero due to the pole being non-complex, the input to the cosine function is zero and thus becomes a multiplication by a factor of one.

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

The lab was clear in its objectives.

7 Conclusion

Concluding this lab I feel that I now have a solid understanding of how the sine/-cosine method can be utilized to expand the Laplace tranformed functions derived from various differential equations with both complex and noncomplex poles.

8 Appendix

Below are the recorded results of the residue function from part 1 followed by part 2.

residues: [0.5 -0.5 1.] poles: [0. -4. -6.] coefficients: []

residues: [1. +0.j -0.48557692+0.72836538j -0.48557692-0.72836538j -0.21461963+0.j 0.09288674-0.04765193j 0.09288674+0.04765193j] poles: [0. +0.j -3. +4.j -3. -4.j -10. +0.j -1.+10.j -1.-10.j] coefficients: []