ECE 351 Lab 6
Partial Fraction Expansion

Andrew Hartman

October 2019

https://github.com/HartmanAndrew

Contents

1 Introduction

Partial fraction expansion is a common method to use to solve for inverse Laplace transforms. It allows for complicated fractions to be split into simpler forms that can then be reference on a Laplace table. This lab will explore the use of the scipy function scipy.signal.residue() to complete partial fraction expansion. This will then be compared to the step response using scipy.signal.step().

2 Equations

2.1 Part 1

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

1. Find the transfer function:

$$\begin{aligned} s^2Y(s) + 10sY(s) + 24Y(s) &= s^2X(s) + 6sX(s) + 12X(s) \\ Y(s) * (s^2 + 6s + 12) &= X(s) * (s^2 + 10s + 24) \\ H(s) &= \frac{s^2 + 6s + 12}{(s+6)(s+4)} \end{aligned}$$

2.Find the step response using the transfer function:

$$\begin{aligned} H(s) * u(s) &= \frac{1}{s} * \frac{s^2 + 6s + 12}{(s+6)(s+4)} \\ &= \frac{A}{s} + \frac{B}{s+6} + \frac{C}{s+4} \\ y(t) &= (\frac{1}{2} + e^{-6t} - 0.5e^{-4t})u(t) \end{aligned}$$

2.2 Part 2

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

Cosine Method Equation:

$$\begin{aligned} p &= \alpha + \omega j \\ k &= \left.\frac{F_0 s}{s - p^*}\right|_{s=p} = |k|\angle k \\ y_c(t) &= |k|e^{\alpha t}cos(\omega t + \angle k)u(t) \end{aligned}$$

3 Methodology

This first step of implementing the step function solved by hand and then graphing it was very simple, this was done using the same plotting methods of all the previous lab. The step response was then taken using scipy.step() like in lab 5 and plotted as well. Both output plots are shown in Figure 1 in the results section. Scipy.residue was also used on the step response in the Laplace domain to solve for the partial fraction expansion. This is shown in the Appendix.

Part 2 began by asking for the partial fraction expansion of the transfer function of the differential equation shown in equations section. This was done using the residue function of scipy. The result is attached in the Appendix. After that a method needed to be implemented in order to complete the cosine method to change the partial fraction expansion back into the time domain.

```
def cosineMethod(r, p, t):
    y = np.zeros((t.shape))
    for i in range(len(r)):
        k = np.abs(r[i])
        phi = np.angle(r[i])
        preal = p[i].real
        pimag = p[i].imag
        y = y+(k*np.exp(preal * t) * np.cos((pimag * t) + phi) * step(t))

    return y
```

This function works by being passed in the roots and poles returns by residue as well as the t time matrix. It then loops through every pair of roots and poles and extracts the different necessary parameters to input into the cosine method equation. Each cosine method is added to the previous ones to get the final output and then returned.

This function was used to find the step response of the provided differential equation and then graphed. Scipy.step() was also used and the two graphs are compared in Figure 2.

4 Results

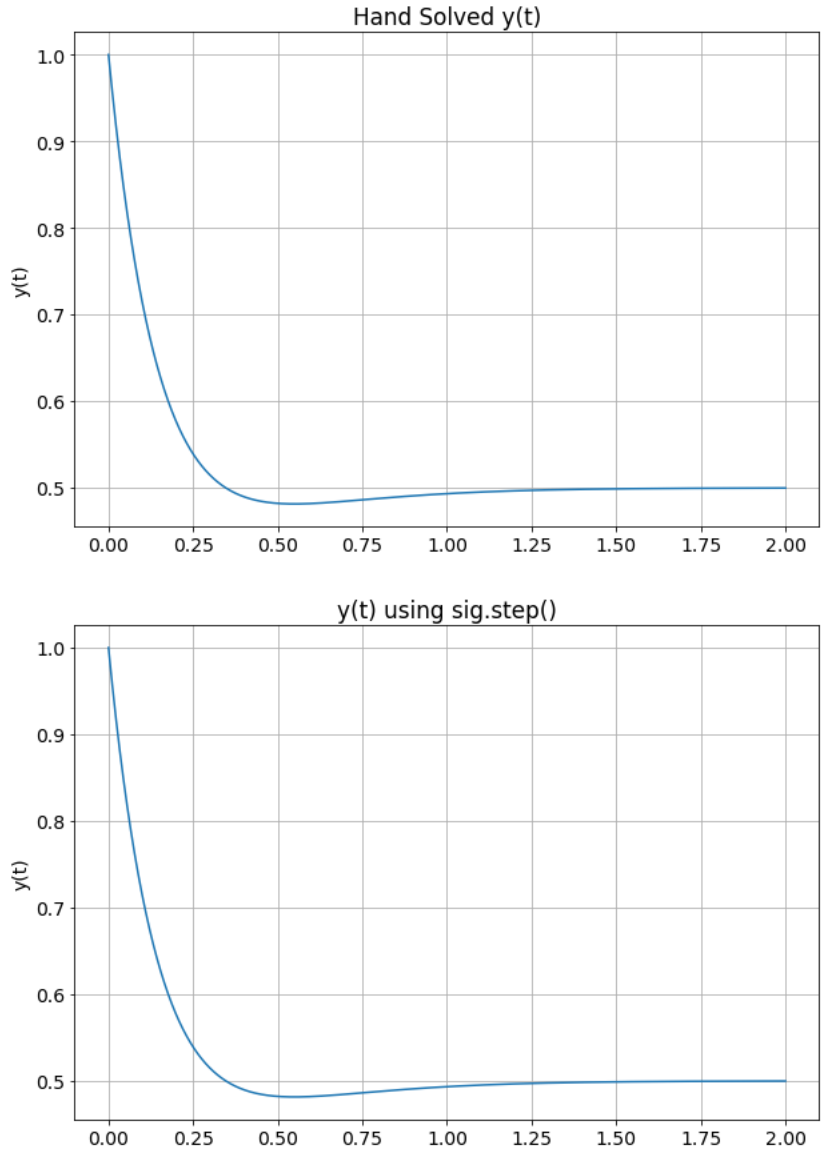

Figure 1: Hand solved and Scipy.step() step responses

These two graphs show once again that solving for a step response by hand and then solving it using scipy.step() both produce the same response and the graphs are identical. The results of the scipy.residue function being ran on this first transfer function is attached in the Appendix.

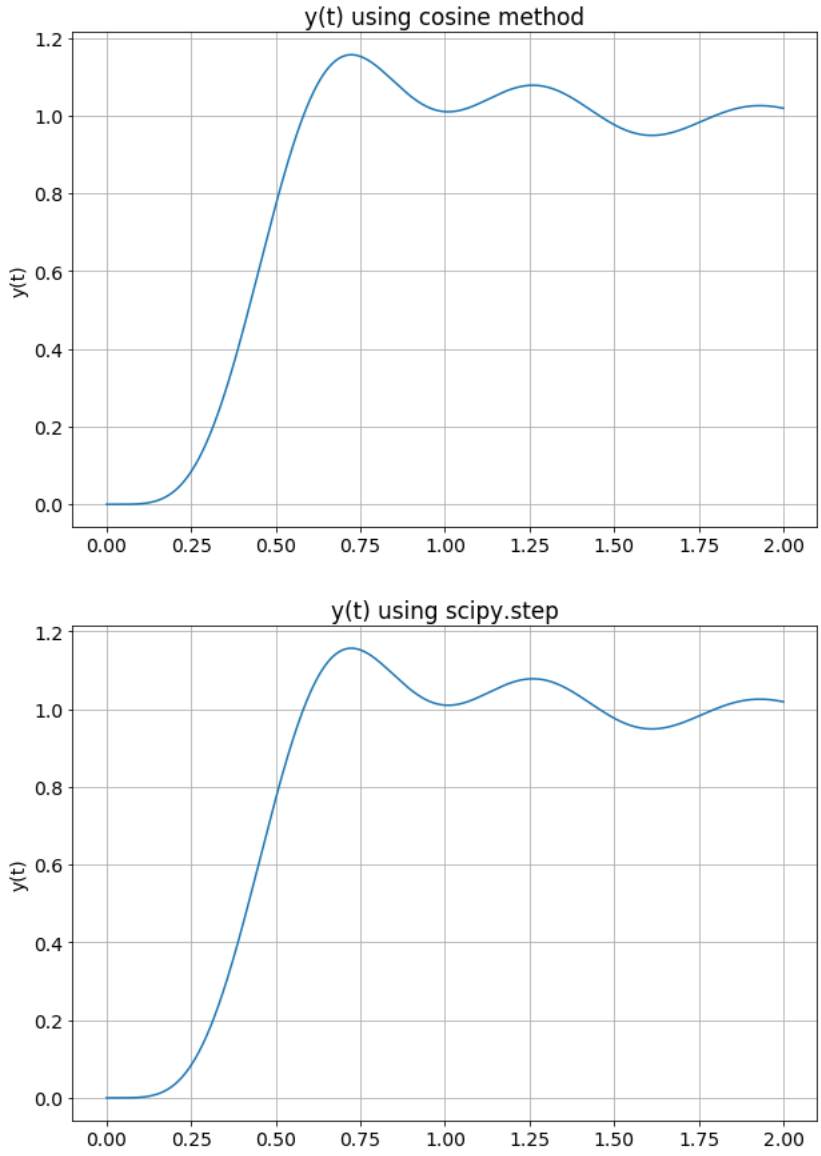

Figure 2: Impulse Response

These two graphs compare the results of using the cosine method or scipy.step(). The cosine method was ran by first running scipy.residue on the transfer function multiplied by a step function to get the partial fraction expansion poles and roots. These were passed into my cosine method function which evaluated the proper $e^t cosine$ term for each pair and added them together. As can be seen on the graphs in Figure 2, both methods produced the same results.

5 Error Analysis

The only source of error that could occur for the lab would be solving the step response wrong by hand. As the two graphs show above though, this did not happen. One issue that did come up though was that the cosine method in the textbook had an extra 2 before the —k— term which caused the magnitude of my cosine method function to be doubled.

6 Questions

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

If the poles and roots are both non complex then the cosine method ends up being:

$$y_c(t) = |r|e^{pt}cos(0)u(t) \qquad = re^{pt}u(t)$$

This is the same solution if you took the inverse Laplace of $\frac{r}{s-p}$.

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

This lab was very well laid out and clear as to what it expected and wanted for deliverables.

7 Conclusion

Being able to complete partial fraction expansion on complex transfer functions is a difficult task to accomplish by hand. This lab showed however that there are functions inside python that can help accomplish this. It also showed how simple it can be to implement mathematical methods into a function to be able to repeatedly solve similar problems. This lab makes me excited to see what other functions the scipy library or other python libraries include to help solve common math and engineering problems.

8 Appendix

```
Part 1:
r=[ 1.   -0.5   0.5]

p=[-6. -4.   0.]
```

Figure 3: Part 1 Residue Output

```
Part 2:
r=[ 1.         +0.j          -0.48557692+0.72836538j -0.48557692-0.72836538j
 -0.21461963+0.j          0.09288674-0.04765193j  0.09288674+0.04765193j]

p=[  0. +0.j  -3. +4.j  -3. -4.j -10. +0.j  -1.+10.j  -1.-10.j]
```

Figure 4: Part 2 Residue Output