Partial Fraction Expansion

Cameron Williams

ECE 351-51

Lab Report 6

3 March 2020

1 Introduction

The objective of this lab was to use the scipy.signal.residue() function to perform partial fraction expansion in order to check the results calculated during the prelab. The scipy.signal.residue() function will also be used to find the step response of another system that would be considerably more difficult to solve by hand.

2 Methodology

Before attending the lab session, I completed the prelab. I checked my prelab results against what the solution the TA provided and found that my solution was correct. The transfer function and step response I calculated are listed below for reference.

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

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

First, I created a plot of the step response that I calculated. This plot may be seen in Figure 1 of the Results section. I checked the results of this plot by creating a second plot via the scipy.signal.step() function using the transfer function H(s). This plot may be seen in Figure 2 of the Results section. Next, I used the scipy.signal.residue() function to perform partial fraction expansion on the polynomial resulting from the multiplication of the transfer function and the the Laplace transform of the step input (H(s) * 1/s). Similar to the scipy.signal.step() function we used in last weeks lab, the residue function takes the polynomial in the numerator and the polynomial in the denominator as arrays of their respective coeffecients. The values returned by the residue function are R, P, and K. These are arrays of the residue, poles, and gain, respectively. In a non-complex system, the residue values correspond directly to the coefficients of the expanded terms and the poles correspond to the values of s that make that term go to infinity (and are thus the α values in the final step response). The results of the residue function may be seen in the Appendix section under the output titled "Part 1 Residue Results". The results of the residue function match the results of my hand calculation.

For the second part of the lab, I used the scipy.signal.residue function to analyze a system that would be considerably more difficult to analyze by hand. The system may be seen below for reference.

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

I began by using the residue function once again to find the residue and poles (gain K was returned, but it was not used). The results of the residue function for this system may

be seen in Appendix section under the output titled "Part 2 Residue Results". I wrote a function to apply the cosine method algorithm detailed on pages 86-87 of Dr. Sullivan's book *Signals and Systems for Electrical Engineers I*. My cosine method function takes residue, poles, and time arrays as inputs and outputs the function. A key difference to be noted is the factor of 2 is removed since both of the complex conjugates are used in this implementation of the algorithm. I applied my cosine method function to the residue function results of the system and produced a plot of the step response for the system. The plot may be seen in Figure 3 of the Results section. Finally, I checked the Figure 3 plot by generating a plot of the same system using the scipy.signal.step function. This plot may be seen in Figure 4 of the Results Section.

3 Results

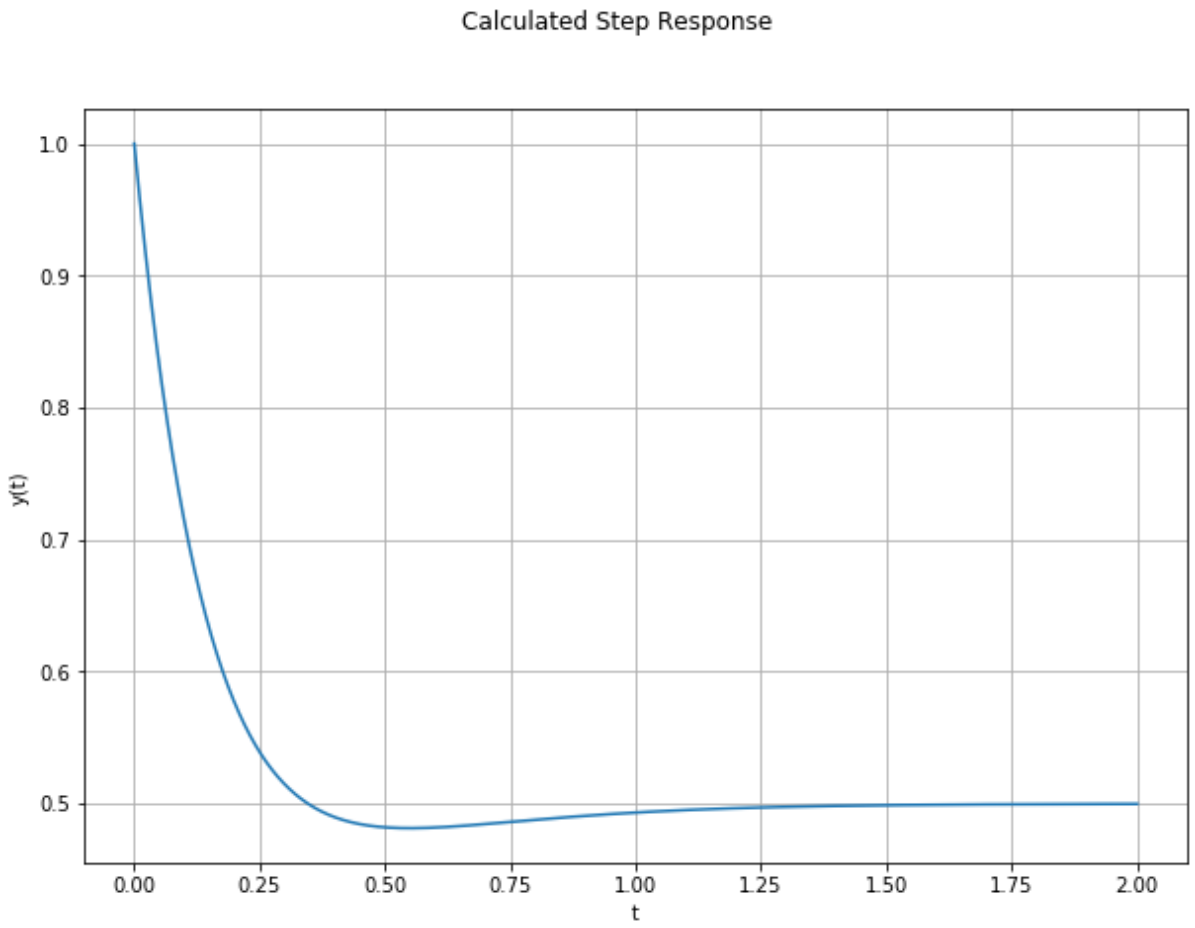

Figure 1

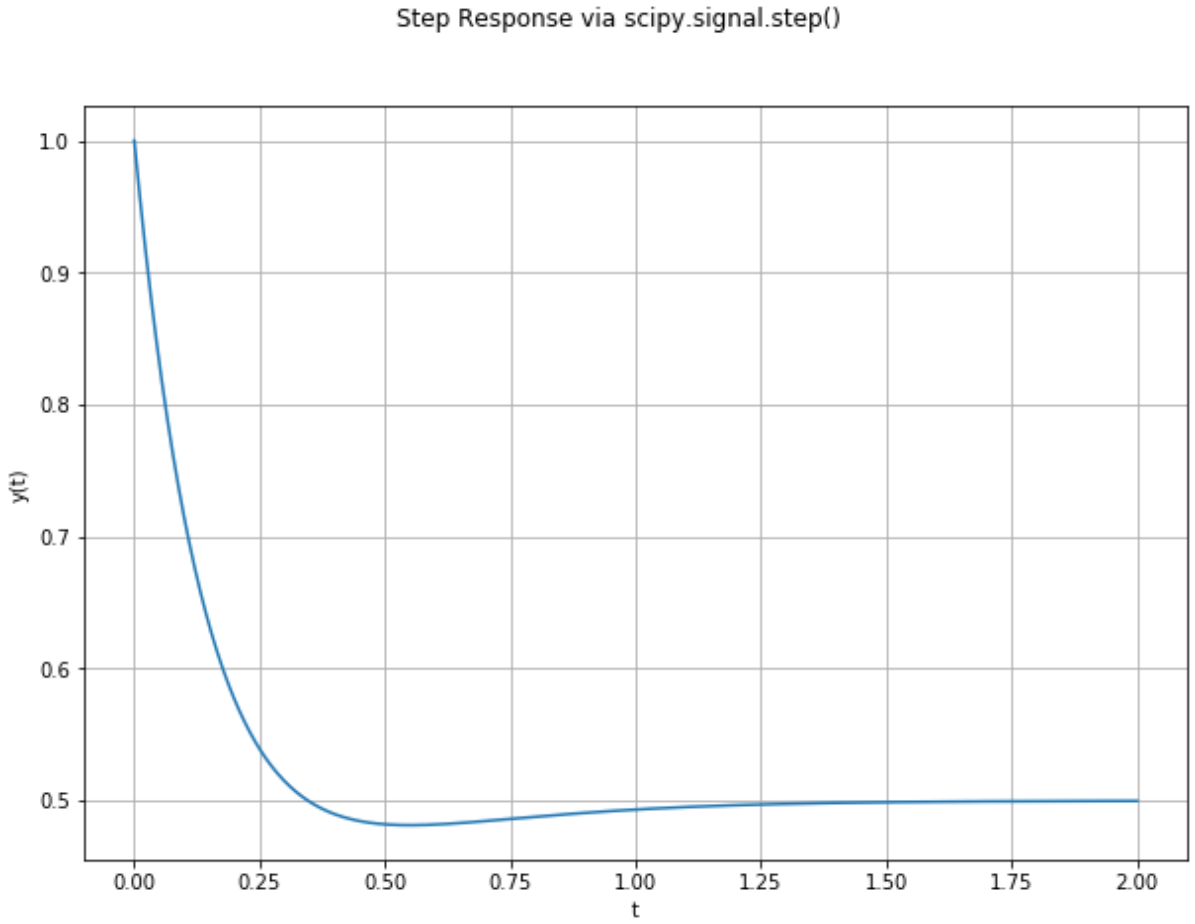
Step Response via scipy.signal.step()
1.0
0.9
0.8
0.7
0.6
0.5
y(t)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
t

Figure 2

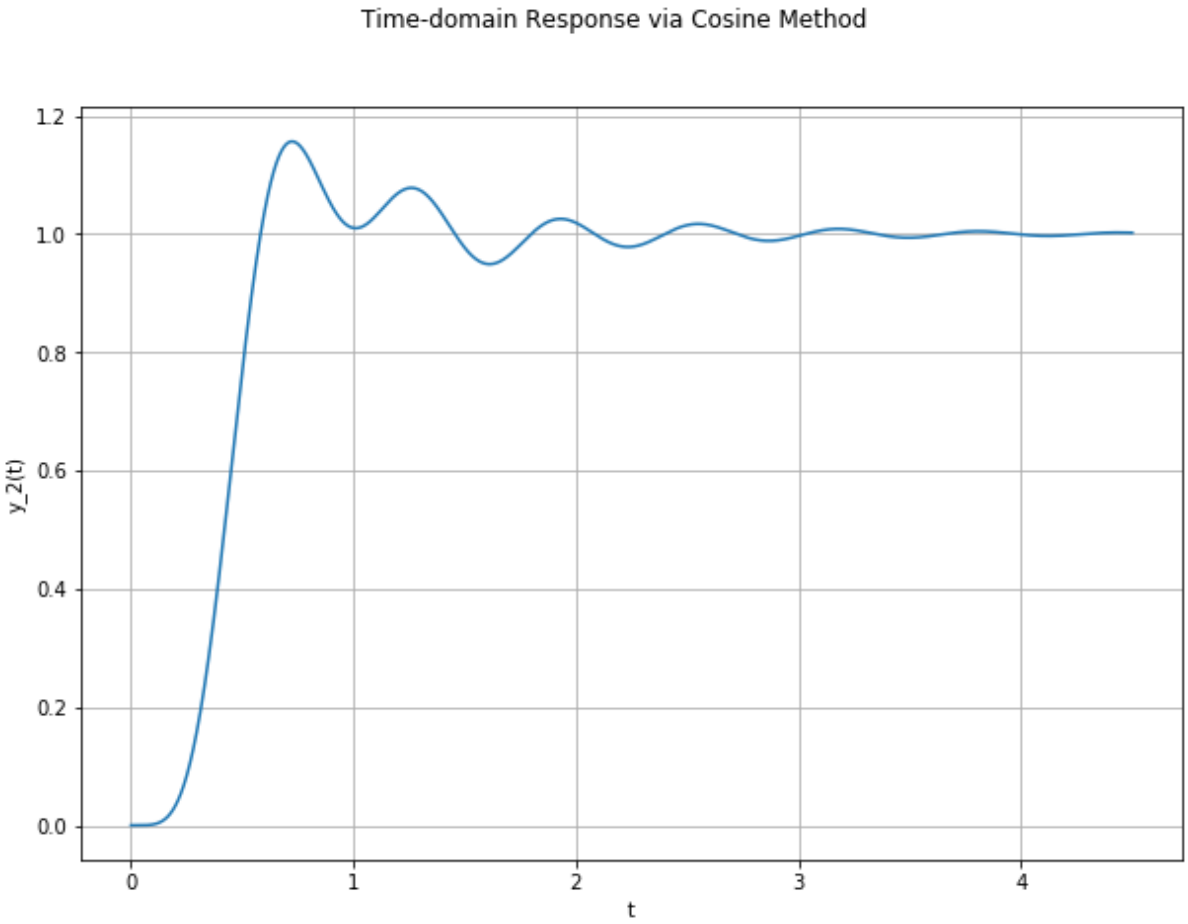
Time-domain Response via Cosine Method
y_2(t)
t
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4

Figure 3

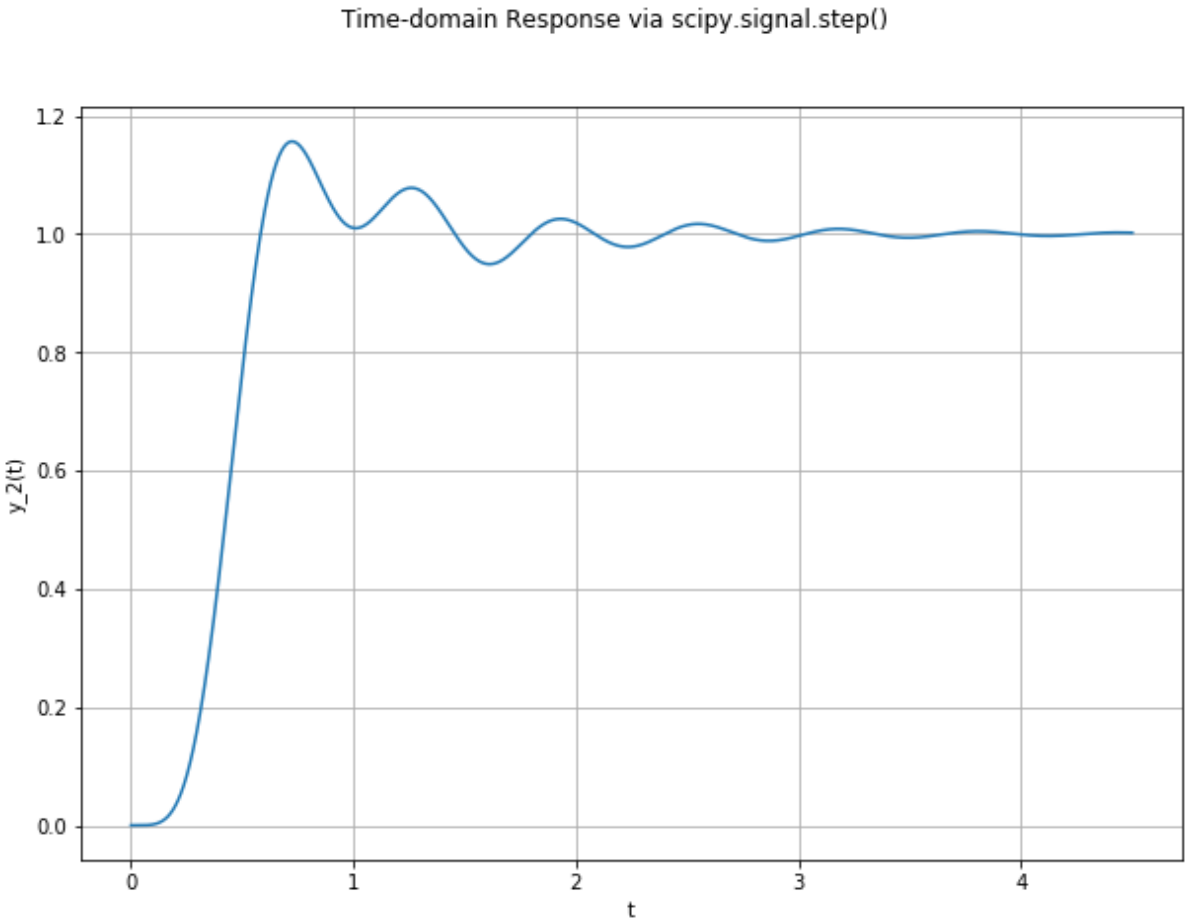
Time-domain Response via scipy.signal.step()
y_2(t)
t
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4

Figure 4

Conclusion

For the first time in the lab section, I was able to complete the lab almost entirely by myself. My cosine method implementation was very close to correct. The exception was an unnecessary absolute value function being used on the omega inside of the cosine term of the summing of y values on the last line of the cosine method function. This small difference added a slight phase shift to the resulting plot which I ended up fixing by simply removing the unneeded absolute value function call. The residue function and cosine method function will undoubtedly be incredibly useful tools in the labs to come.

Questions

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

If the a pole-residue term is not complex, then the omega and angle terms (equation listed below for reference) are zero. Because of this, the value of the cosine term becomes 1, leaving only an exponential term with a coefficient, just like any other non-complex term.

$$y_c(t) = |k|e^{\alpha t}cos(\omega t + \angle k)u(t)$$

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

The expectations and deliverables for this lab were expressed clearly.