Project Report for ECE 351

Lab 06 - Partial Fraction Expansion

Skyler Corrigan

September 29, 2022

ECE351 Code Repository:
$https://github.com/ElfinPeach/ECE351_Code.git$

ECE351 Report Repository:
$https://github.com/ElfinPeach/ECE351_Report.git$

Contents

1 Objective

This lab is designed to demonstrate partial fraction expansion.

2 Part 1

The equation that is analyzed is as follows:

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

The transfer function that was hand calculated is as follows:

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

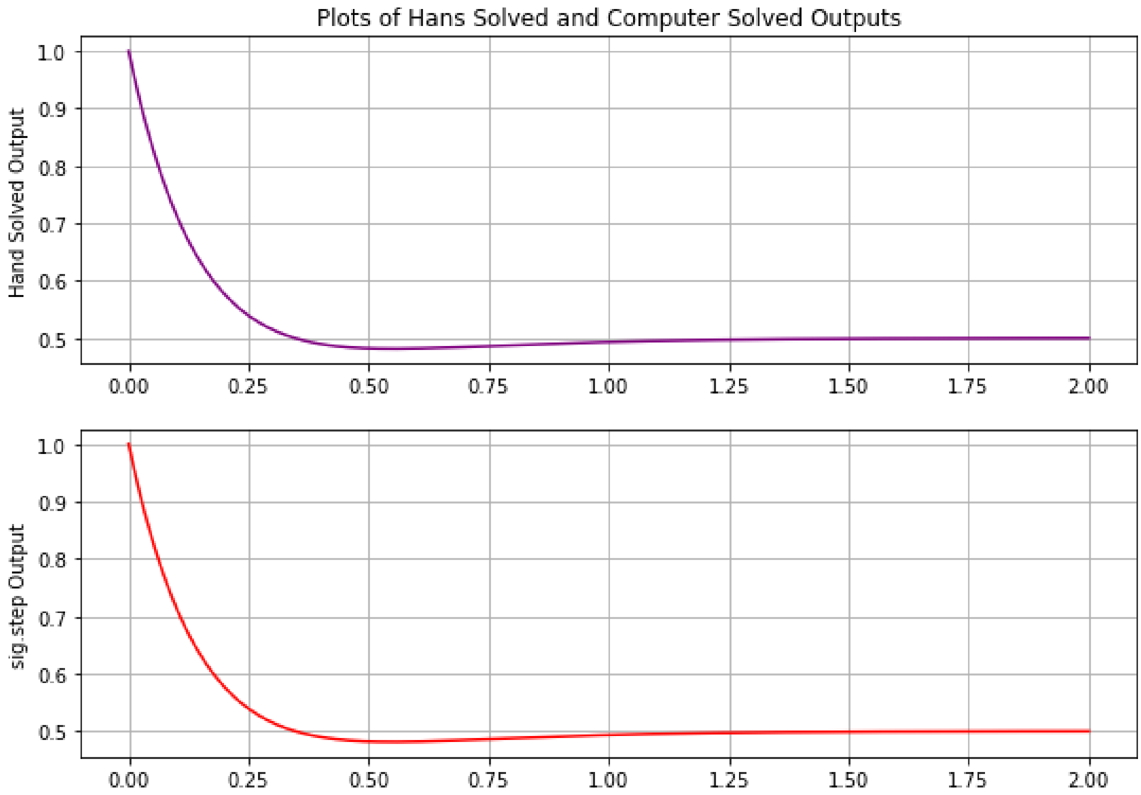
Using the Python function to find the roots and poles, they are as follows:

Roots = [0.5, -0.5, 1]
Poles = [0, -4, -6]

These roots and poles show that the Python function is equivalent to the hand defined one.

The hand defined and computer defined functions plot the graphs below.

Figure 1


Plots of Hans Solved and Computer Solved Outputs
Hand Solved Output
sig.step Output
1.0
0.9
0.8
0.7
0.6
0.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

3 Part 2

The function evaluated in this section is as follows:

$$y^{(5)} + 18y^{(4)} + 218y'''(t) + 2036y''(t) + 9085y'(t) + 25250y(t) = 25250x(t)$$

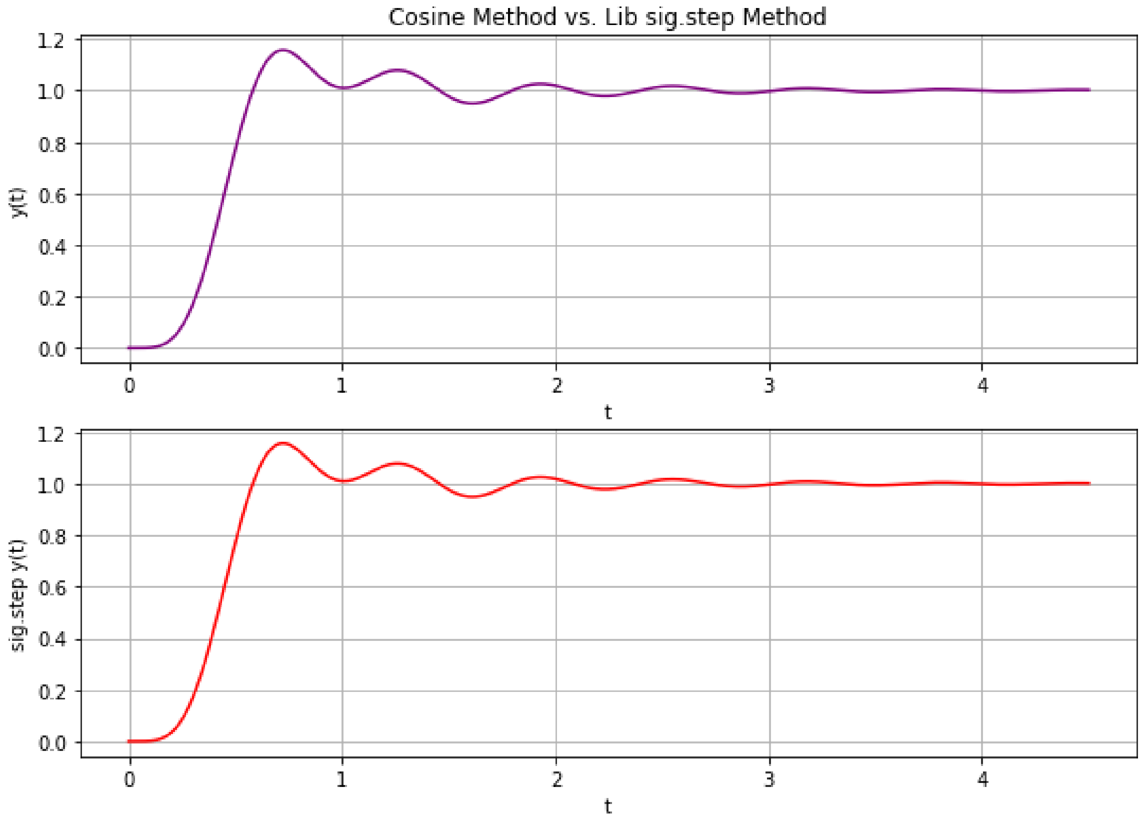
Using the scipy.signal.residue() function, the roots, poles, and K value are as follows:

Roots = [1+0.j, -0.486+0.728j, -0.486-0.728j, -0.215+0.j, 0.0929-0.047j 0.093+0.0477j]
Poles = [0.+0.j, -3.+4.j, -3.-4.j, -10.+0.j, -1.+10.j, -1.-10.j]
K = 0.104 ∠0.474

These were used to create a plot using the Cosine method, which is:

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

This was compared to the build in step function, and both were plotted as seen below:
Figure 2



Since both of these plots are equivalent, it shows that both methods result in the same outcome.

4 Appendix

The following is the code used for the entirety of the lab.
Lab Code

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

#--------------FUNCTIONS--------------------------------------------#

#Make a step function using an array t, stepTime, and stepHeight
def stepFunc(t, startTime, stepHeight):
    y = np.zeros(t.shape)
    for i in range(len(t)):
        if(t[i] >= startTime):
            y[i] = stepHeight
    return y

#------------------END FUNCTIONS------------------------------------#

#----------Part 1---------------------------------------------------#
    #Define step size
steps = 1e-2

    #t for part 1
start = 0
stop = 2
    #Define a range of t. Start at 0 and go to 20 (+a step)
t = np.arange(start, stop + steps, steps)

    #Prelab stuff
h = ((np.exp(-6 * t)) + (-0.5 * np.exp(-4 * t))+ 0.5) * stepFunc(t, 0, 1)

    #Make the H(s) using the sig.step()

num = [1, 6, 12] #Creates a matrix for the numerator
den = [1, 10, 24] #Creates a matrix for the denominator

tout , yStep = sig.step((num , den), T = t)

den_residue = [1, 10, 24, 0]

    #Make and print the partial fraction decomp
roots, poles, _ = sig.residue(num, den_residue)
```

```
print("Part 1")
print("Roots =", roots)
print("Poles =", poles)


    #Make plots for pt1
plt.figure(figsize=(10,7))
plt.subplot(2,1,1)
plt.plot(t,h, 'purple')
plt.grid()
plt.ylabel('Hand Solved Output')
plt.title('Plots of Hans Solved and Computer Solved Outputs')

plt.subplot(2,1,2)
plt.plot(t,yStep, 'red')
plt.grid()
plt.ylabel('sig.step Output')


#------------PART 2----------------------

    #Define step size
steps = 1e-2

    #t for part 1
start = 0
stop = 4.5
    #Define a range of t2. Start at 0 and go to 20 (+a step)
t2 = np.arange(start, stop + steps, steps)

    #Make numerator and denomentaor for sig.residue()
num2 = [25250]
den2 = [1, 18, 218, 2036, 9085, 25250, 0]

R, P, K = sig.residue(num2, den2)

print("")
print("Part 2")
print("Roots =", R)
print("Poles =", P)

    #cosine vethod
yt = 0

    #Range iterates through each root
```

```
for i in range(len(R)):
    angleK = np.angle(R[i])
    magK = np.abs(R[i])
    W = np.imag(P[i])
    a = np.real(P[i])

    yt += magK * np.exp(a * t2) * np.cos(W * t2 + angleK) *
        stepFunc(t2, 0, 1)

print("K value =", magK)
print("K angle =", angleK)

#Make the lib generated step response
den2_step = [1, 18, 218, 2036, 9085, 25250]
tStep2, yStep2 = sig.step((num2,den2_step), T = t2)

    #Show Plots
plt.figure(figsize=(10,7))
plt.subplot(2,1,1)
plt.plot(t2, yt,'purple')
plt.grid()
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Cosine Method vs. Lib sig.step Method')


plt.subplot(2,1,2)
plt.plot(tStep2, yStep2, 'red')
plt.grid()
plt.xlabel('t')
plt.ylabel('sig.step y(t)')

#---------------SHOW ALL PLOTS---------------------
plt.show()
```