1 Introduction

The goal of this lab is to learn how to use Python's residue function to perform partial fraction expansion.

2 Equations

We use function from the per-lab in time-domain and in s-domain:

$$Y(s) = H(s)X(s) = H(s)\frac{1}{s} = \frac{s^2 + 6s + 12}{s^3 + 10s^2 + 24s} \tag{1}$$

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

Also, we use function given in the lab manual:

$$Y(s) = H(s)\frac{1}{s} = \frac{25250}{s(s^5 + 18s^4 + 218s^3 + 2036s^2 + 9285s + 25250)} \tag{3}$$

$$y(t) = 2|k|e^{\alpha t}cos(wt + \angle k)u(t) \tag{4}$$

3 Methodology and Results

This lab consists of two parts:

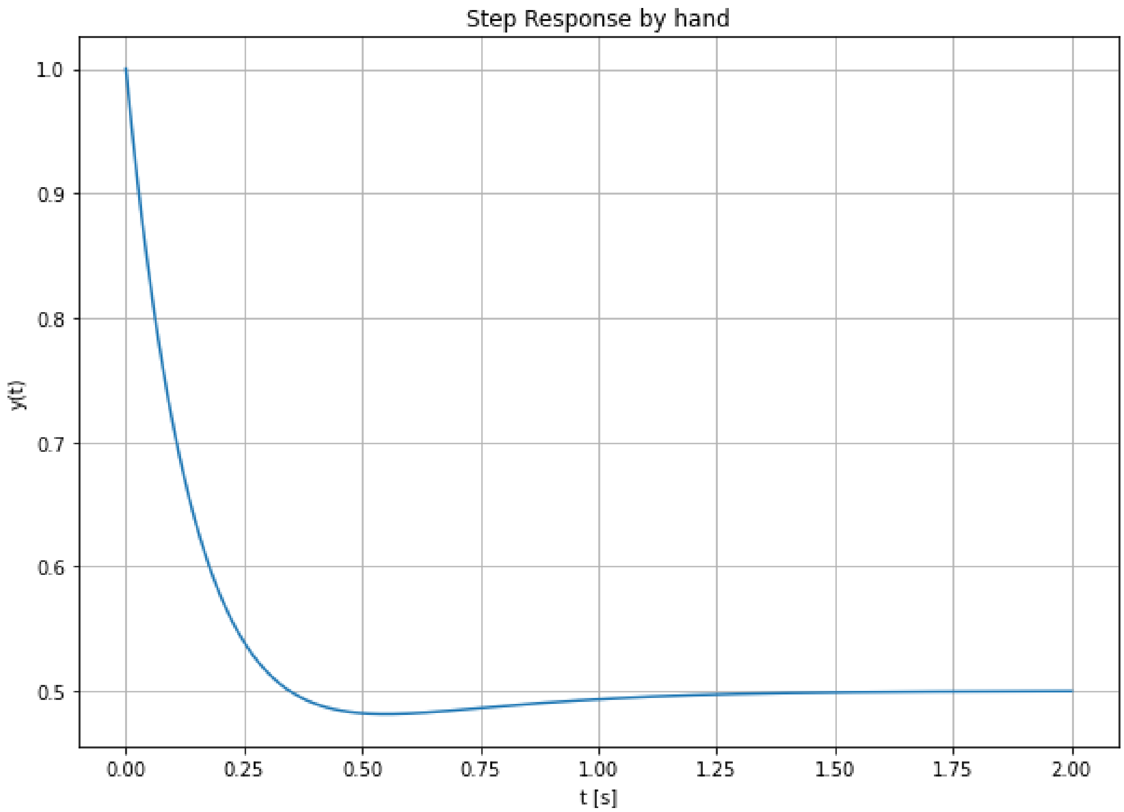
- Part 1:

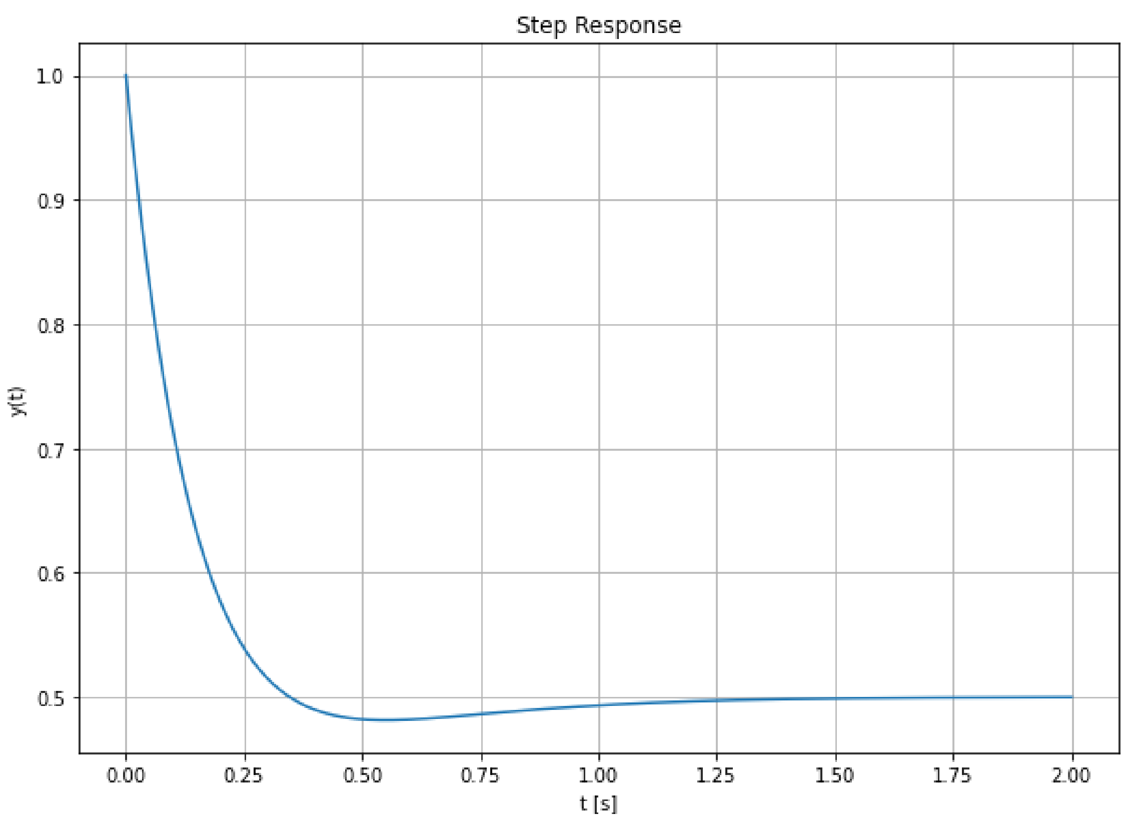
  In this part, we plot the step response twice. The first one by using y(t) that we found by hand in the prelab and second one by using H(s) and the scipy.signal.step() command. Then, we print the partial fraction expansion results R, P, and K by using scipy.signal.residue() command:

```
#%% part 1 task 1

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
t = np.arange(0, 2 + steps, steps)

def h(t):
h = (1/2 - (1/2)*np.exp(-4*t)+np.exp(-6*t))*u(t)
return h
plt.figure(figsize=(10,7))
plt.subplot(1, 1, 1)
```

```
    plt.plot(t,h(t))
    plt.grid()
    plt.ylabel ('y(t)')
    plt.xlabel('t [s]')
    plt.title ('Step Response by hand')

    #%% part 1 task 2

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

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

    plt.figure(figsize=(10,7))
    plt.subplot(1, 1, 1)
    plt.plot(tout, yout)
    plt.grid()
    plt.ylabel ('y(t)')
    plt.xlabel('t [s]')
    plt.title ('Step Response')


    #%% part 1 task 3

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

    R, P, K = sig.residue(num, den)

    print("Residues:\n", R)

    print("Poles:\n", P)

    print("Gain:\n", K)


```

The output plots of this code is:


Step Response by hand
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 [s]


Step Response
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 [s]

Also, it outputs the values of R, P, and K which agree with our resluts from pre-lab:

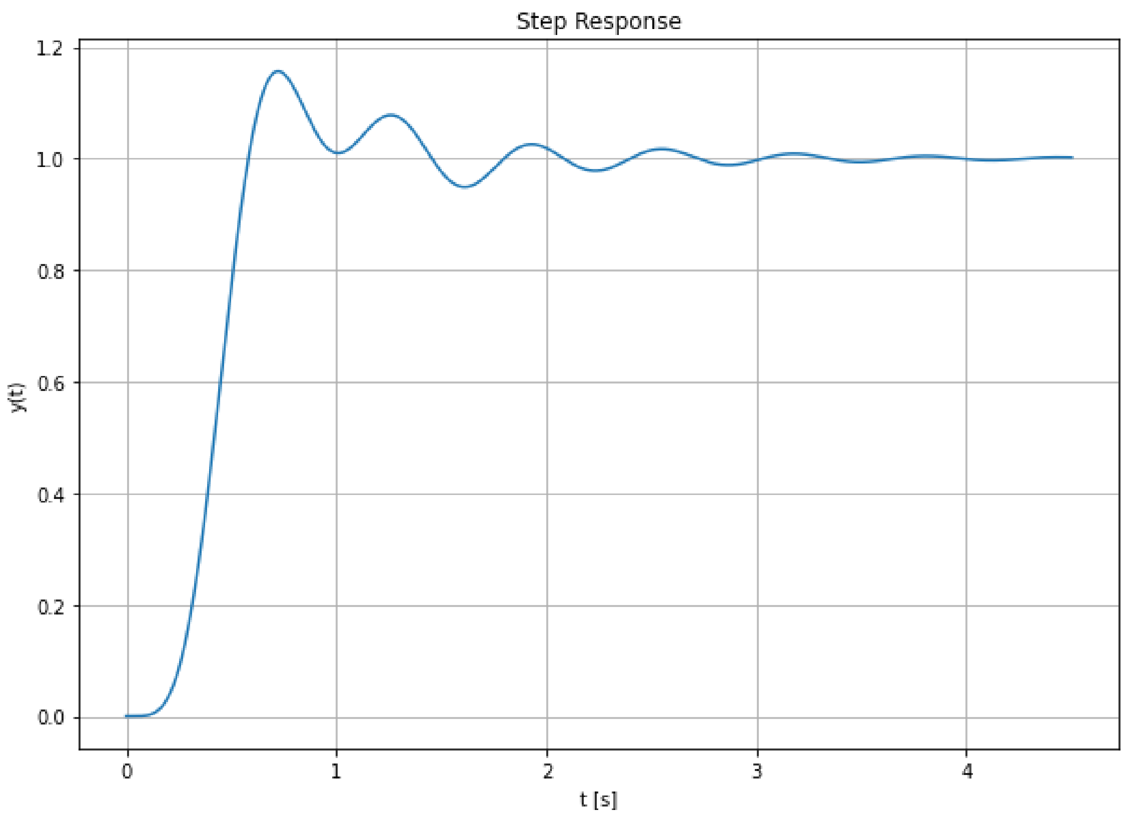
```
Residues:
[ 0.5 -0.5  1. ]
Poles:
[ 0. -4. -6.]
Gain:
[]

```

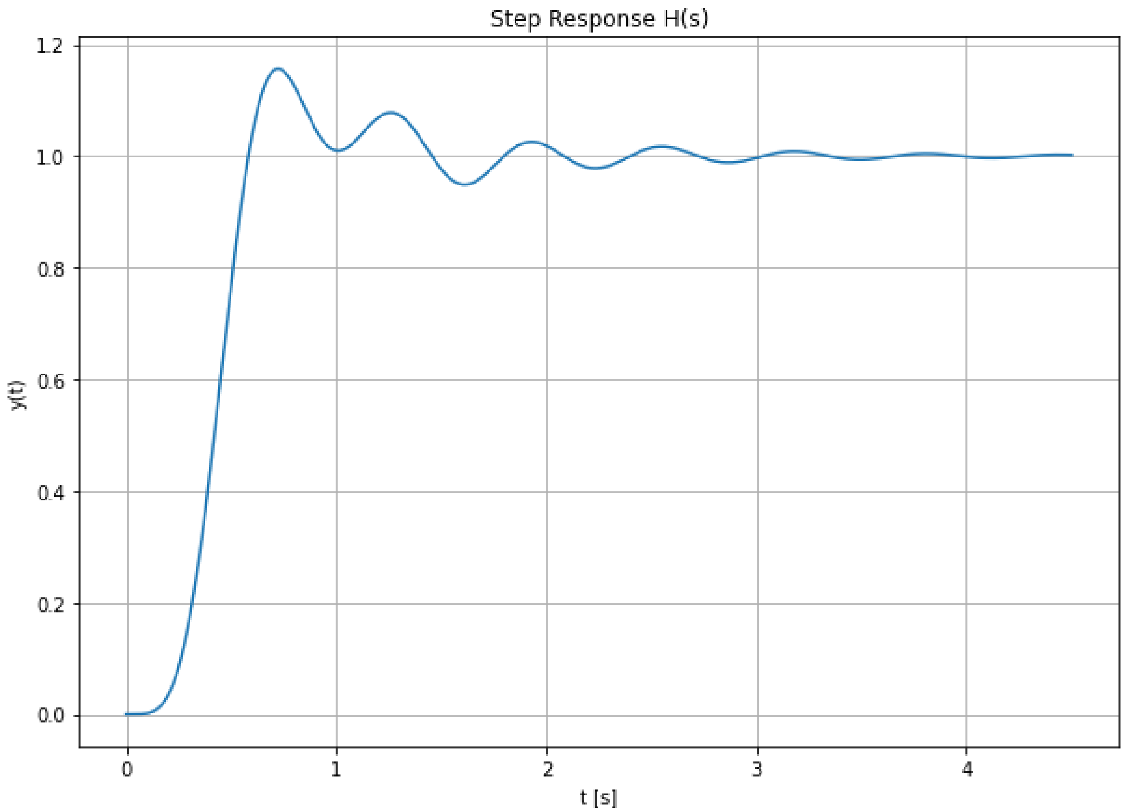
- Part 2:
  In this part, we print the partial fraction expansion results R, P, and K by using scipy.signal.residue() command. Then, we plot the time-domain response using the cosine method and the step response by using H(s) and the scipy.signal.step() command:

```
#%% part 2 task 1

num = [25250]
den = [1 , 18 , 218, 2036, 9085, 25250, 0]

R1, P1, K1 = sig.residue(num, den)

print("Residues:\n", R1)

print("Poles:\n", P1)

print("Gain:\n", K1)

#%% part 2 task 2

def cosine_method(R1,P1,t):
y = 0
for i in range(len(R1)):
kmag = np.abs(R1[i])
kang = np.angle(R1[i])
alpha = np.real(P1[i])
omega = np.imag(P1[i])
y = y + kmag*np.exp(alpha*t)*np.cos(omega*t + kang)*u(t)
return y

t =np.arange(0, 4.5 + steps, steps)
plt.figure(figsize=(10,7))
plt.subplot(1,1,1)
plt.plot(t,cosine_method(R1,P1,t))
plt.grid()
plt.ylabel ('y(t)')
plt.xlabel('t [s]')
plt.title ('Step Response')
plt.show()

#%% part 2 task 3

num = [25250]
```

```
    den = [1 , 18 , 218, 2036, 9085, 25250]

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

    plt.figure(figsize=(10,7))
    plt.subplot(1, 1, 1)
    plt.plot(tout, yout)
    plt.grid()
    plt.ylabel ('y(t)')
    plt.xlabel('t [s]')
    plt.title ('Step Response H(s)')
    plt.show()


```

The output plots of this code is:





The R, P, and K values:

```
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]
Gain:
[]

```

4 Questions

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

   The angle of the cosine will be zero cos(0)=1. So the result of y(t) will be exponential.

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

5 Conclusion

In this lab, we learned how to use scipy.signal.residue() function to perform partial fraction expansion.