ECE 351-51

March 8, 2022

Lab 7

Submitted By :
Kennedy Beach

Contents

1 Introduction

The purpose of this lab was to use Laplace-domain block diagrams and factored transfer functions to judge system stability. The poles and zeros of the individual components of the diagrams as well as the overall transfer functions were looked at to determine how stable the system was.

2 Equations

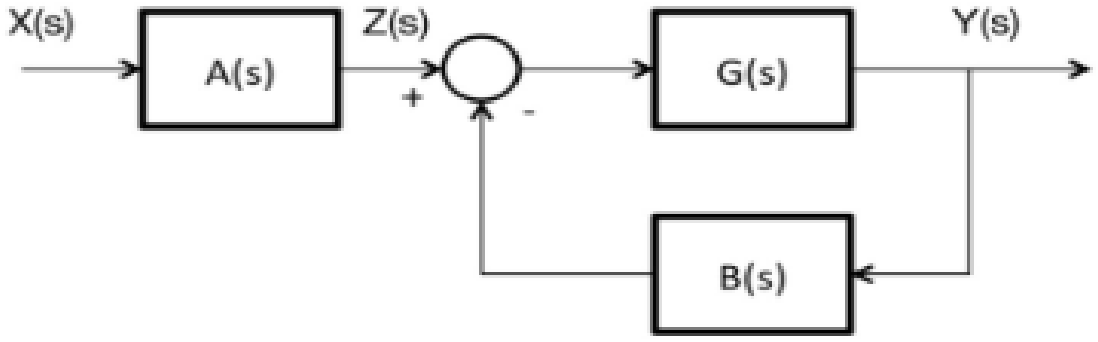

Figure 1: Block Diagram

$$G(s) = \frac{s+9}{(s^2-6s-16)(s+4)}$$

$$A(s) = \frac{s+4}{s^2+4s+3}$$

$$B(s) = s^2+26s+168$$

Figure 2: Component Equations

The factored forms of the component functions are seen below.

$$G(s) = \frac{s+9}{(s+2)(s+4)(s-8)}$$

The zeros of G(s) are s = -9 and the poles are s = -4, -2, and 8.

$$A(s) = \frac{s+4}{(s+1)(s+3)}$$

The zeros of A(s) are s = -4, and the poles are s = -3 and -1.

$$B(s) = (s + 12)(s + 14)$$

B(s) only has zeros at s = -14 and -12.

The open-loop transfer function was then derived by multiplying component A(s) and G(s):

$$H_{open}(s) = \frac{(s + 4)(s + 9)}{(s + 1)(s + 3)(s + 2)(s + 4)(s - 8)}$$

The closed-loop transfer function was derived symbolically, then the unfactored form was calculated through Python.

$$H_{closed}(s) = \frac{A * G}{1 + B * G}$$

$$H_{closed}(s) = \frac{numA * numG}{denA * denG + denA * B * numG}$$

$$H_{closed}(s) = \frac{s^2 + 13s + 36}{2s^5 + 41s^4 + 500s^3 + 2995s^2 + 6878s + 4344}$$

3 Methodology

The first part involved using scipy.signal.tf2zpk() to get the poles and zeros of the component functions to verify the calculations. numpy.roots() was used to compute the zeros for B since it is not a proper transfer function. The printed output is seen in the Results section.

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

numG = [1,9]
denG = sig.convolve([1,-6,-16], [1,4])

numA = [1,4]
denA = [1,4,3]

B = [1,26,168]

G_z, G_p, G_k = sig.tf2zpk(numG, denG)
print('G zeros: ', G_z, '\nG poles: ', G_p)
```

```
15
16 A_z, A_p, A_k = sig.tf2zpk(numA, denA)
17 print('\nA zeros: ', A_z, '\nA poles: ', A_p)
18
19 B_roots = np.roots(B)
20 print('\nB roots: ', B_roots)
```

Listing 1: Poles and zeros of the components

The numerator and denominator of the open-loop transfer function were then calculated with scipy.signal.convolve(), and poles and zeros of the transfer function were calculated. The step response of the open-loop transfer function was then plotted.

```
1 numOpen = sig.convolve(numA, numG)
2 denOpen = sig.convolve(denA, denG)
3 print('\nOpen num: ', numOpen, '\nOpen den: ', denOpen)
4
5 open_z, open_p, open_k = sig.tf2zpk(numOpen, denOpen)
6 print('\nOpen loop zeros: ', open_z, '\nOpen loop poles: ', open_p)
7
8 steps = 1e-5
9 t = np.arange(0, 5 + steps, steps)
10
11 tout, yout = sig.step((numOpen, denOpen), T = t)
12
13 plt.figure(figsize = (10, 7))
14 plt.ylabel('h(t)')
15 plt.xlabel('t')
16 plt.plot(tout, yout)
17 plt.grid()
18 plt.suptitle('Step Response of the Open Loop')
```

Listing 2: Open-loop transfer function calculation and poles and zeros

Next, the closed-loop numerator and denominator were found using scipy.signal.convolve() with the various element numerators and denominators. The equation that explains this is in the Equations section.

```
1 numClosed = sig.convolve(numA,numG)
2 denClosed = sig.convolve((denG + sig.convolve(B, numG)), denA)
3 print('\nClosed num: ', numClosed, '\nClosed den: ', denClosed)
4
5 closed_z, closed_p, closed_k = sig.tf2zpk(numClosed, denClosed)
6 print('\nClosed loop zeros: ', closed_z, '\nClosed loop poles: ',
    closed_p)
7
8 tout2, yout2 = sig.step((numClosed, denClosed), T = t)
9
10 plt.figure(figsize = (10, 7))
```

```
plt.ylabel('h(t)')
plt.xlabel('t')
plt.plot(tout2, yout2)
plt.grid()
plt.suptitle('Step Response of the Closed Loop')
```

Listing 3: sig.residue() of the second differential equation

4 Results

The open-loop transfer function had a positive pole in the denominator, which means that the function would be increasing at some point rather than continually decreasing if all the poles were negative. This means that the open-loop transfer function is not stable. This is seen in its step response where it increases infinitely as it diverges.

Conversely, in the closed-loop transfer function, the poles are negative which means that the transfer function will be decreasing at a certain point. This means that the closed-loop transfer function is stable. This is seen in the step response where it increases, but then plateaus as it eventually converges.

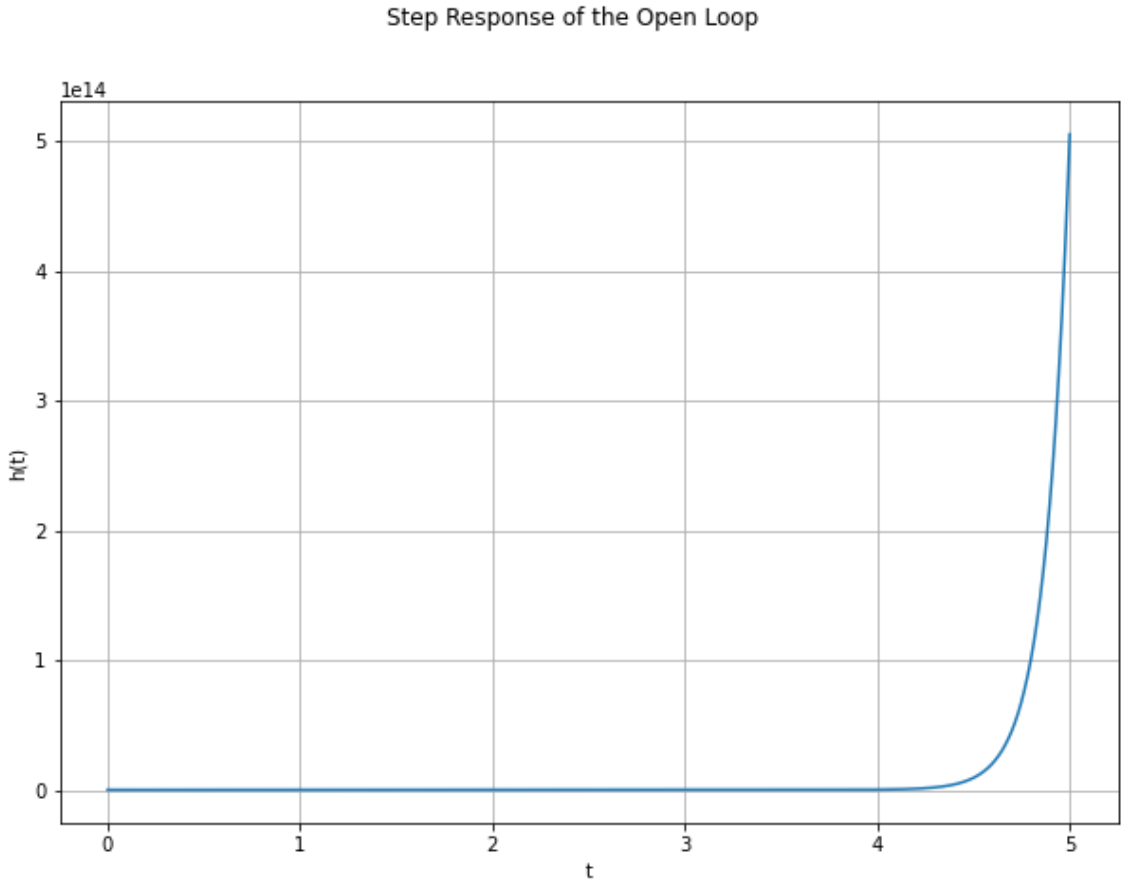

Figure 3: Plot of the Open Loop Step Response

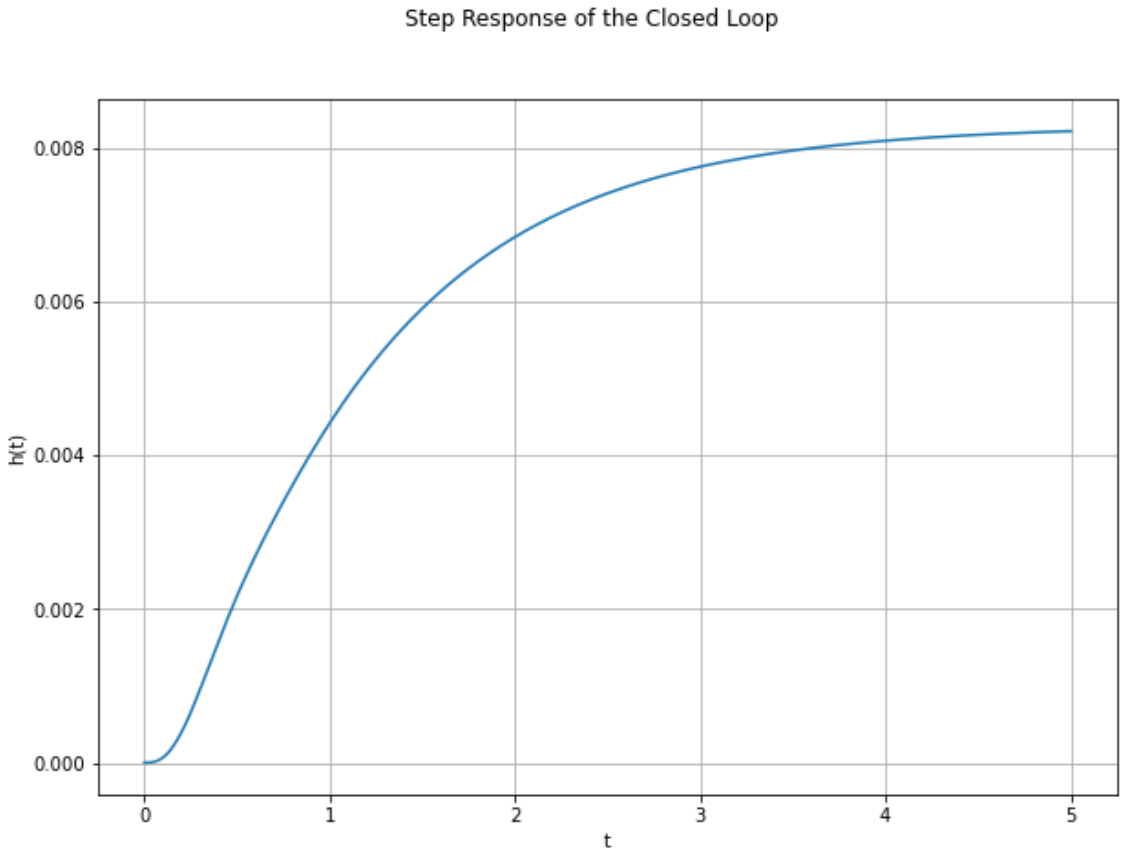

Figure 4: Plot of the Closed Loop Step Response

```
In [86]: runfile('D:/College/Spring 2022/ECE_351/Lab7/Beach_Kennedy_ECE351_Lab7.py', wdir='D:/College/Spring 2022/ECE_351/Lab7')
G zeros:  [-9.]
G poles:  [ 8. -4. -2.]

A zeros:  [-4.]
A poles:  [-3. -1.]

B roots:  [-14. -12.]

Open num:  [ 1 13 36]
Open den:  [   1    2  -45 -230 -376 -192]

Open loop zeros:  [-9. -4.]
Open loop poles:  [ 8. -4. -3. -2. -1.]

Closed num:  [ 1 13 36]
Closed den:  [   2   41  500 2995 6878 4344]

Closed loop zeros:  [-9. -4.]
Closed loop poles:  [-5.16237064+9.51798197j -5.16237064-9.51798197j -6.17525872+0.j
 -3.        +0.j        -1.        +0.j        ]
```

Figure 5: Printed output of the zeros and poles of the individual components and the open and closed loop transfer functions

5 Error Analysis

There weren't really any errors besides getting scipy.signal.convolve() to work when calculating the denominator of the closed loop. This just required a minor change in how the convolutions were nested.

6 Questions

1. In Part 1 Task 5, why does convolving the factored terms using scipy.signal.convolve() result in the expanded form of the numerator and denominator? Would this work with your user-defined convolution function from Lab 3? Why or why not?

Using scipy.signal.convolve() results in the expanded form since that is its default behavior as it multiplies two functions. This is the same behavior as our convolution function in Lab 3 since that function output an array of the sum of the products of two functions at certain times.

2. Discuss the difference between the open- and closed-loop systems from Part 1 and Part 2. How does stability differ for each case, and why?

The open-loop system is unstable since it had a positive pole in the denominator. This resulted in it growing exponentially. The closed loop system had negative poles which caused it to converge rather than diverge. The negative feedback loop cancelled out the positive pole found in the open-transfer loop.

3. What is the difference between scipy.signal.residue() used in Lab 6 and scipy.signal.tf2zpk() used in this lab?

The residue results in the partial fraction expansion of non-linear systems whereas the tf2zpk function works with linear systems. Although they produce similar results, the residue function computes the residue of a function while the tf2zpk function computes the zeros.

4. Is it possible for an open-loop system to be stable? What about for a closed-loop system to be unstable? Explain how or how not for each.

It is possible for either to be stable or unstable. The determinant is if there is a positive pole in the denominator.

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

Everything was clear on what needed to be done and turned in.

7 Conclusion

This lab showcased the computational power of Python by using the scipy.signal.convolve() and scipy.signal.tf2zpk functions to factor and expand transfer functions. These functions would make it easy to analyze complex systems. The Python and LaTeX code are seen in `https://github.com/Eniac618/ECE351_Code` and `https://github.com/Eniac618/ECE351_Reports` respectively.