Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Signals and Systems II
Course Number:	ELE 632
Semester/Year (e.g.F2016)	W2023

Instructor:	Dimitri Androutsos

Assignment/Lab Number:	Lab 2
Assignment/Lab Title:	Time-Domain Analysis of Discrete-Time Systems-Part 2

Submission Date:	February 19, 2023
Due Date:	February 19, 2023

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Saini	Harsanjam	501055402	10	Harsanjam

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Introduction

Welcome to this lab report where we will explore LTID systems and their response analysis using Python. In this lab assignment, we will delve into the determination of unit impulse response, zero state response, and total response for LTID systems. These responses are fundamental in understanding how a system behaves and are essential in various engineering applications. In Lab Assignment 1, we solved similar problems recursively. However, in this report, we will approach these problems differently by finding the characteristic modes of the system equation, which is another effective method to solve these problems. Additionally, we will examine the stability of the system to gain a more comprehensive understanding of its behavior.

A) Unit Impulse Response

1) Python code for Part 1:

I. $y[n] - \frac{1}{(F+1)}\, y[n-1] - \frac{1}{(F+1)}[y-2] = \frac{1}{(G+1)}\, x[n]$, **G=0 and F=5**

II. $y[n] - \frac{1}{(H+1)}\, y[n-2] = x[n]$, **H=0**

```
import numpy as np
import matplotlib.pyplot as plt
import scipy as scipy
from scipy import signal
n = np.arange(0,8)
delta = signal.unit_impulse(8)
a = [1,1/6,-1/6]
b = [1/5,0,0]
h = signal.lfilter(b,a,delta)
plt.stem(n,h)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Unit Impulse Response I")
plt.grid()
```

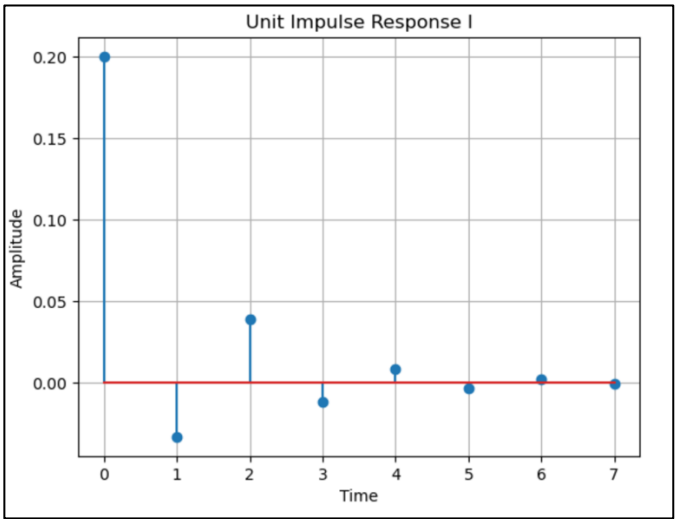

Figure 1: Unit impulse response from equation 1

```
import numpy as np
import matplotlib.pyplot as plt
import scipy as scipy
from scipy import signal
n = np.arange(0,8)
delta = signal.unit_impulse(8)
a = [1,0,1/1]
b = [1,0,0]
h = signal.lfilter(b,a,delta)
plt.stem(n,h)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Unit Impulse Response II")
plt.grid()
```

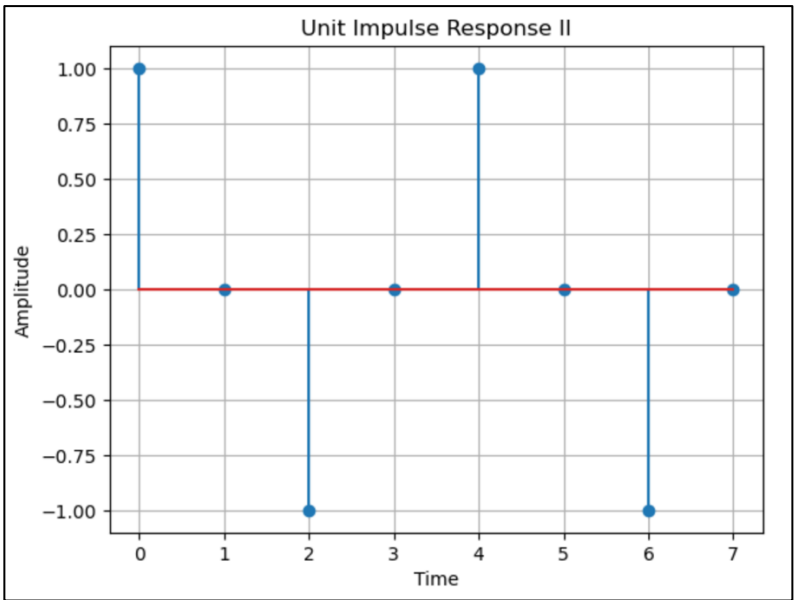

Figure 2: Unit impulse response from equation 2

Part 2 and 3) Hand Calculations and comparison for equation 1

part 2) Hand calculation

1) $y(n) + \frac{1}{6}y(n-1) - \frac{1}{6}y(n-2) = \frac{1}{5}x[n]$

$y[n+2] + \frac{1}{6}y[n+1] - \frac{1}{6}y[n] = \frac{1}{5}x[n+2]$

$Q(E)\, y[n] = P(E)\, x[n]$

$\left(E^2 + \frac{1}{6}E - \frac{1}{6}\right) y[n] = \frac{1}{5}E^2 x[n]$

$E^2 + \frac{1}{6}E - \frac{1}{6} = 0$

$\left(E + \frac{1}{2}\right)\left(E - \frac{1}{3}\right) = 0$

roots $= -\frac{1}{2}, \frac{1}{3}$

characteristic

$y[n] = c_1\left(\frac{1}{3}\right)^n + c_2\left(-\frac{1}{2}\right)^n$

$h[n] + \frac{1}{6}h[n-1] - \frac{1}{6}h[n-2] = \frac{1}{5}\delta[n]$

$h[-1] = h[-2] = 0$

@ n=0

$h[0] + \frac{1}{6}h[n-1] - \frac{1}{6}h[n-2] = \frac{1}{5}(1)$

$h[0] = \frac{1}{5}$

@ n=1

$h[1] + \frac{1}{6}h[0] - \frac{1}{6}h[n-1]$

$h[1] + \frac{1}{6}\left(\frac{1}{5}\right) - 0 = \frac{1}{5}(0)$

$h[1] = -\frac{1}{30}$

$h[n] = c_1\left(\frac{1}{3}\right)^n + c_2\left(-\frac{1}{2}\right)^n$

$h[0] = 1/3 \qquad h[1] = -\frac{1}{30}$

@ n=0

$\frac{1}{3} = c_1 + c_2$ ①

$-1/30 = c_1(1/3) + c_2(-1/2)$ ②

$\therefore c_1 = \frac{1}{3} - c_2$

$-\frac{1}{30} = \left(\frac{1}{3} - c_2\right)\left(\frac{1}{3}\right) + c_2\left(-\frac{1}{2}\right)$

$-\frac{1}{30} = \frac{1}{9} - \frac{c_2}{3} - \frac{c_2}{2}$

$c_2\left(\frac{1}{3} + \frac{1}{2}\right) = \frac{1}{9} + \frac{1}{30}$

$c_2\left(\frac{5}{6}\right) = \frac{13}{90}$

$c_2 = \frac{13}{75}$ or 0.173

$c_1 = \frac{1}{3} - c_2 = \frac{1}{3} - \frac{13}{75} = \frac{4}{25}$ or 0.16

hence

$$h[n] = \left[\frac{4}{25}\left(1/3\right)^n + \frac{13}{75}\left(-\frac{1}{2}\right)^n\right] u[n]$$

part 3 comparison of hand calculation and python

$$h[n] = \left[\frac{4}{25}\left(\frac{1}{3}\right)^n + \frac{13}{75}\left(-\frac{1}{2}\right)^n\right] u[n]$$

$h[3] = \frac{4}{25}\left(\frac{1}{3}\right)^3 + \frac{13}{75}\left(-\frac{1}{2}\right)^3$

$= \frac{4}{25}\left(\frac{1}{27}\right) + \frac{13}{75}\left(-\frac{1}{8}\right)$

$h[3] = \frac{4}{675} - \frac{13}{600}$

$h[3] = \frac{-17}{1080}$ or -0.0157

Part 2 and 3) Hand Calculations and comparison for equation 2

part 2) Hand calculation

11) $y[n+2] + \frac{1}{1} y[n] = x[n+2]$

$Q(\epsilon)\, y[n] = P(\epsilon)\, x(n)$

$(\epsilon^2 + 1) y[n] = \epsilon^2 x(n)$

$\epsilon^2 + 1 = 0$

roots = j & $-j$

characteristic

$y[n] = c_1 (j)^n + c_2 (-j)^n$

$h[n] + \frac{1}{4} h[n-2] = \delta[n]$

@ n=0

$h[0] + \frac{1}{4} h[-2] = 1$

$h[-2] = 0$

$h[0] = 1$

@ n=1

$h[1] + \frac{1}{4} h[-1] = 0$ $\quad h[-1] = 0$

$h[1] = 0$

$h[n] = c_1 (j)^n + c_2 (-j)^n$

@ n=0

$h[0] = c_1 (j)^0 + c_2 (-j)^0$

$1 = c_1 + c_2$

$c_1 = 1 - c_2$

@ n=1

$h[1] = c_1 (j)^1 + c_2 (-j)^1$

$0 = (1 - c_2)(j) + c_2 (-j)$

$0 = j + c_2 (-j) + c_2 (-j)$

$-j = 2 c_2 (-j)$

$\left(\frac{1}{2}\right)\left(\frac{-j}{-j}\right) = c_2$

$\therefore c_2 = \frac{1}{2}$

$c_1 = 1 - 1/2 = 1/2$

$h[n] = \left[\frac{1}{2}(j)^n + \frac{1}{2}(-j)^n\right] u[n]$

part 3 comparison of hand calculation & python

$h[n] = \left[\frac{1}{2}(j)^n + \frac{1}{2}(-j)^n\right] u[n]$

$h[3] = \frac{1}{2}(j)^3 + \frac{1}{2}(-j)^3$

$h[3] = -\frac{1}{2} j + \frac{1}{2} j$

$h[3] = 0$

Therefore the values of h[3] is same in both the methods. We get the same result from the graph and handwritten calculations

B) Zero Input Response

1) The equation $\boldsymbol{y[n] - \frac{(D+1)}{10}\, y[n-1] - \frac{1}{10}[y-2] = 2x[n]}$ **where D = 0**
y [-1] = 1 and y[-2] = 2

Python code for Part 1:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter, lfiltic
#NOTE My D value is = 0
N = 8; D = 0
a = [1,-(D+1)/10,-0.1]
b = [2,0,0]
zi = lfiltic(b, a, [1, 2])
u = np.zeros(N)
y,zo = lfilter(b, a, u, zi=zi)
plt.stem(y)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Zero Input Response")
plt.grid()
```

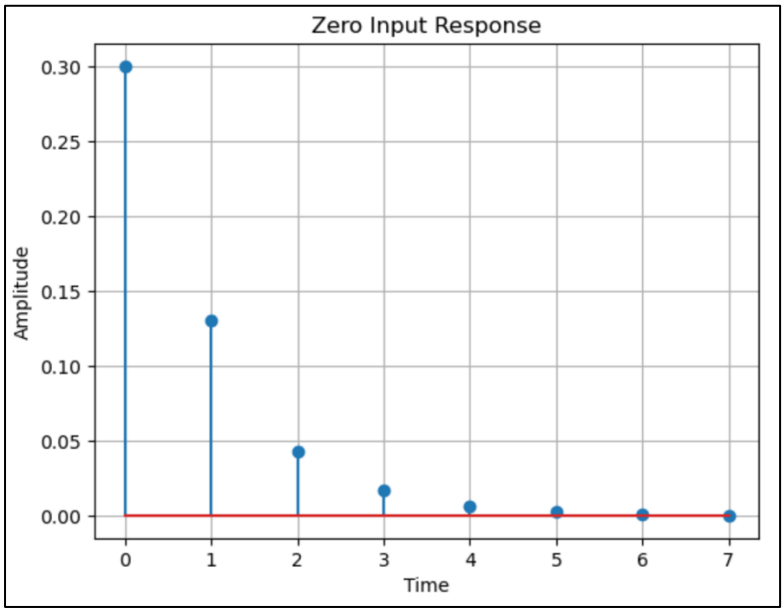

Figure 3: Zero input response of system from equation 1

C) Zero-State Response

1) The equation $\boldsymbol{x[n] = 2cos\left(\frac{2\pi n}{(D+1)}\right)(u[n] - u[n-(D+1)])}$ **where D = 0**

Python code for Part 1:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter, lfiltic, unit_impulse
def u(a, n):
    unit = np.array([])
    for sample in n:
        if sample<a:
            unit = np.append(unit,0)
        else:
            unit = np.append(unit,1)
    return(unit)
#NOTE My D value is = 0
D = 0; n = np.arange(0,15)
a = [1,-(D+1)/10,-0.1]
b = [2,0,0]
x = 2*np.cos((2*np.pi*n)/(D+1))*(u(0,n)-u((D+1),n))
y = lfilter(b,a,x)
plt.figure(2)
plt.stem(n,y)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Zero State Response")
plt.grid()
```

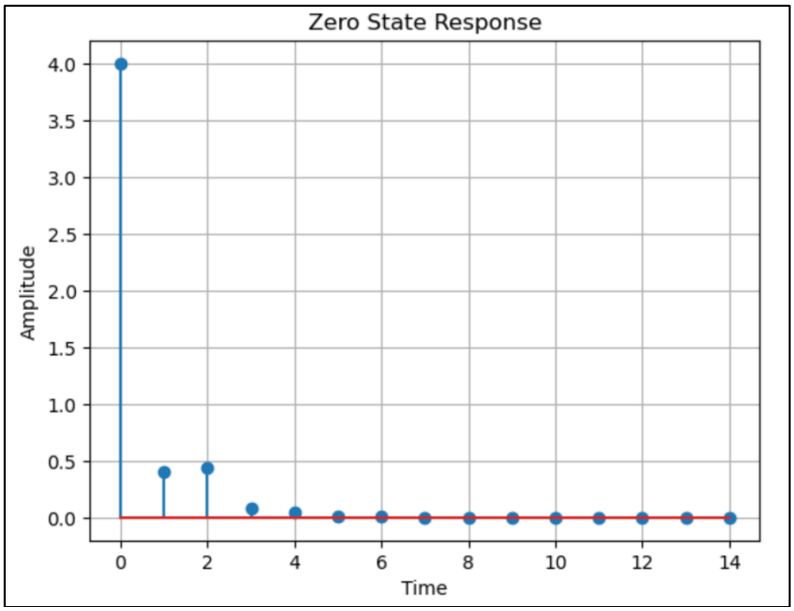

Figure 4: Zero state response of system from part B and input x[n]

D) Total Response

1. Use Python to find and sketch the total response of the system in PART B to the input $x[n]$ (PART C) with initial conditions $y[-1] = 1$ and $y[-2] = 2$.

Python code for Part 1:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter, lfiltic
def u(a, n):
    unit = np.array([])
    for sample in n:
        if sample<a:
            unit = np.append(unit,0)
        else:
            unit = np.append(unit,1)
    return(unit)
#NOTE My D value is = 0
D = 0; n = np.arange(0,15)
a = [1,-(D+1)/10,-0.1]
b = [2,0,0]
x = 2*np.cos((2*np.pi*n)/(D+1))*(u(0,n)-u((D+1),n))
zi = lfiltic(b,a,[1,2])
y,zi = lfilter(b,a,x,zi=zi)
plt.stem(n,y)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Full Response")
plt.grid()
```

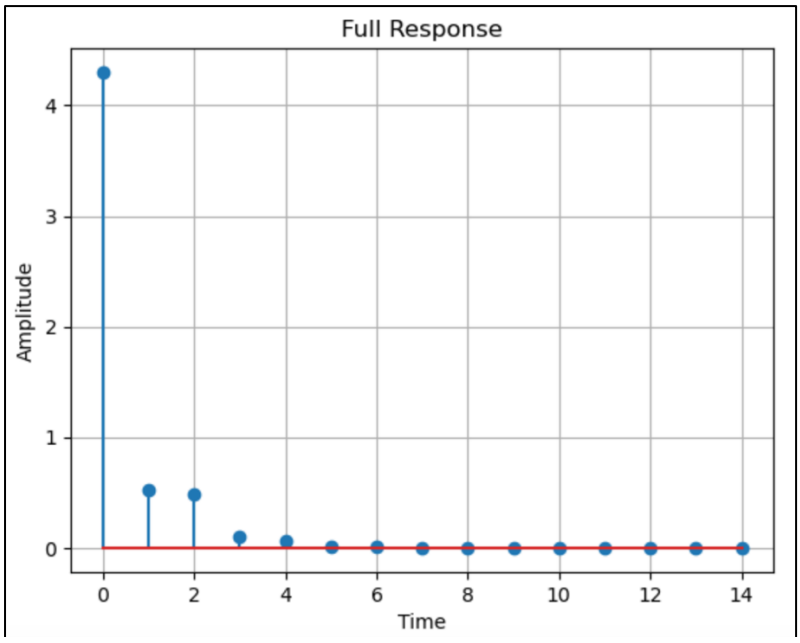

Figure 5: Total response of the system from part B

2. Examine if the total response can be found by adding the zero-input response from PART B and the zero-state response in PART C

Python code for Part 2:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter, lfiltic
def u(a, n):
    unit = np.array([])
    for sample in n:
        if sample<a:
            unit = np.append(unit,0)
        else:
            unit = np.append(unit,1)
    return(unit)
#NOTE My D value is = 0
D = 0; n = np.arange(0,15)
a = [1,-(D+1)/10,-0.1]
b = [2,0,0]
u1 = np.zeros(15)
zi = lfiltic(b,a,[1,2])
y1,zo = lfilter(b, a, u1, zi=zi) #ZERO INPUT RESPONSE
x = 2*np.cos((2*np.pi*n)/(D+1))*(u(0,n)-u((D+1),n))
y2 = lfilter(b,a,x) # ZERO STATE RESPONSE
fig, (ax1, ax2) = plt.subplots(2)
ax1.stem(n,y1);
ax2.stem(n,y2);
```

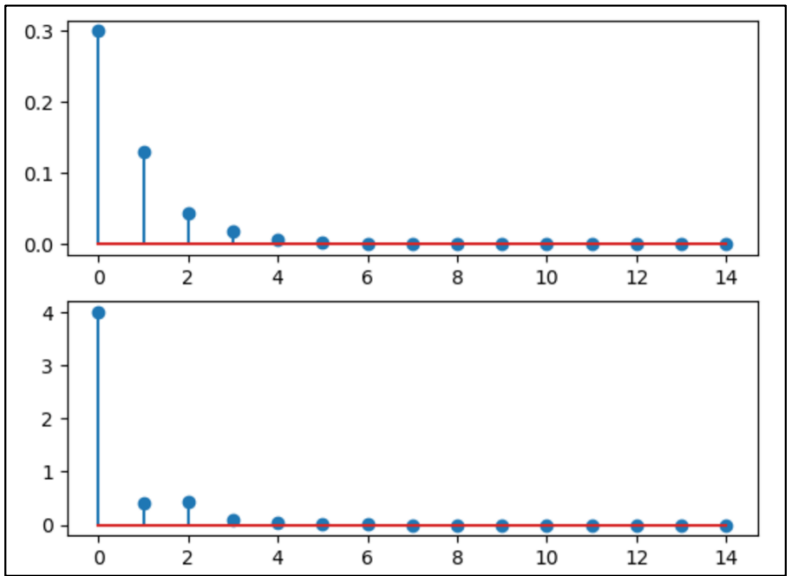

Figure 6: Separate waveforms for zero input and zero state response

Python code for Part 2 adding zero input and zero state:

```
# ZERO INPUT + ZERO
y = y1 + y2
plt.stem(y)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Zero-Input + Zero-State")
plt.grid()
```

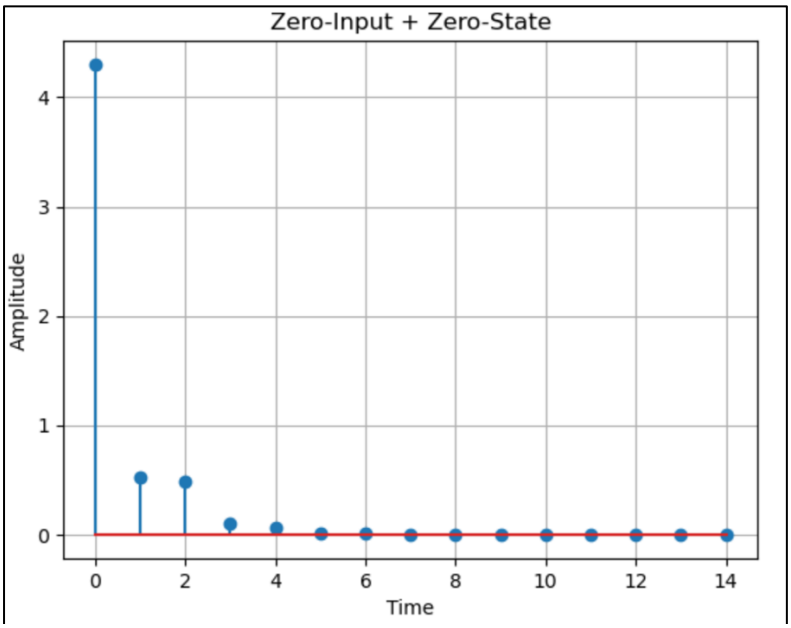

Figure 7: Addition of zero-input and zero-state response

As shown in the plots above we can see that the total response can be found by adding the zero-state response and the zero-input response. This is true because both plots are the same.

E) Convolution and System Stability

Python code for Part 1:

```
import numpy as np
from numpy import zeros
import matplotlib.pyplot as plt
from scipy.signal import lfilter, lfiltic, lfilter_zi, convolve, unit_impulse
def u(a, n):
    unit = np.array([])
    for sample in n:
        if sample<a:
            unit = np.append(unit,0)
        else:
            unit = np.append(unit,1)
    return(unit)
#NOTE My D value is = 0
D = 0; n = np.arange(0,15)
a = [1,-(1+D)/10,-0.1]
b = [2,0,0]
h = lfilter(b,a,unit_impulse(15)) #ZERO STATE RESPONSE
x = 2*np.cos((2*np.pi*n)/(D+1))*(u(0,n)-u((D+1),n)) #INPUT
fig, (ax1, ax2) = plt.subplots(2)
ax1.stem(n,h);
ax2.stem(n,x);
```

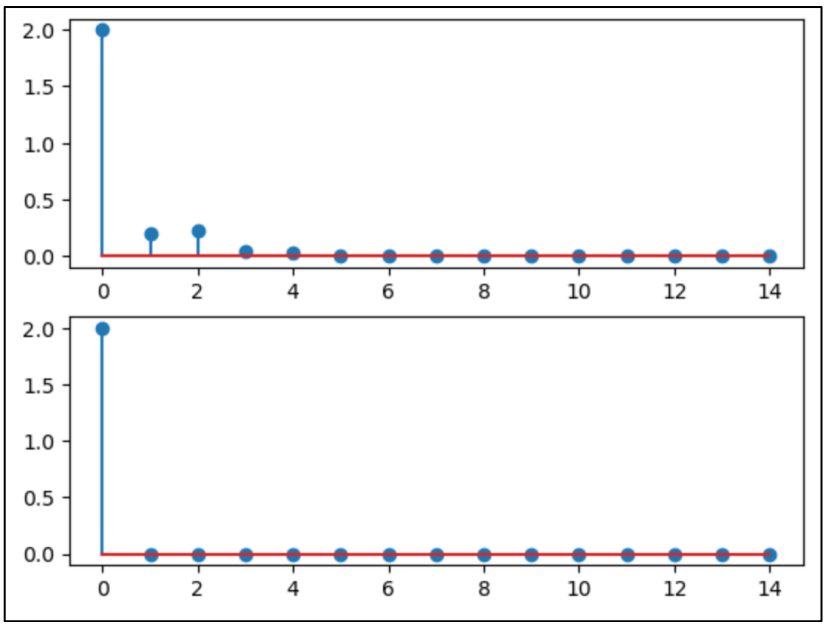

Figure 8: Zero state and zero input response before adding and convolving.

Python code for convolve function:

```
y = convolve(x,h)
plt.stem(y)
```

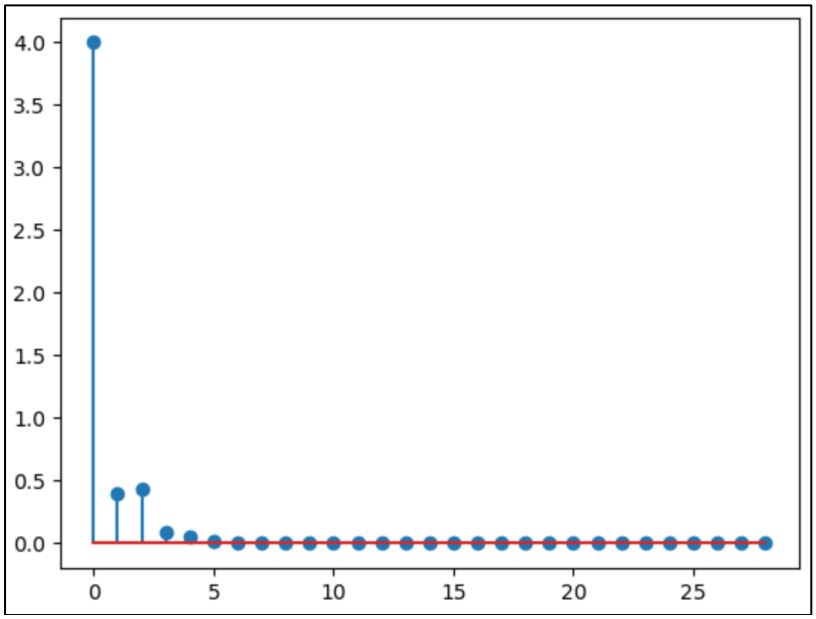

Figure 9: Convolution and system stability

2) It appears that the outcomes in the graphs from Parts C and E are identical since convoluting the data from the impulse-response to the input is equivalent to adding the ZERO-INPUT + ZERO-STATE waveforms.

3) Because the system's characteristics have two real origins, it is also asymptomatically stable. For D = 0, the roots are 0.37015 and -0.27015 which both fall inside the complex plane's unit circle. When the system steadily gets smaller as time passes, it may also be viewed visually.

F) Moving Average Filter

Part 1) Constant coefficient equation:

$h[n] = (u[n] - u[n-N])/N = (1[u[n] - u[n-N]])/N$

Simplify:

$h[n] = 1/N$ for $0 \le n \le N-1$

$h[n] = 0$ for $n < 0$ or $n \ge N$

Constant coefficient equation as:

$y[n] = \left(\frac{1}{N}\right)\left[x[n] + x[n-1] + \ldots\ldots + x[n-N+1]\right]$

Python code for Part 2:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter_zi, lfiltic, convolve,lfilter, unit_impulse
#NOTE My D value is = 0
n = np.arange(0,20); D = 0;G = 4; H =0
def moving_average_filter_parameters(N):
    # y(n) = 1/N * (x(n)+ x(n-1)+ x(n-2)+ ... + x(n-(N-1)))
    # Since h(n) = (u(n) - u(n-N)) / N, 'b' will be a value of '1/N' from 0-->N
    b = np.ones(N) / N # x(n)
    a = 1 # y(n)
    return b, a
def moving_average_filter(N):
    b,a = moving_average_filter_parameters(N)
    x = np.cos((np.pi * n)/(D + 1)) + (unit_impulse(20,(G+1))
                                       + unit_impulse(20,(H+1)))

    h = lfilter(b, a, x)
    return h


plt.figure(0)
plt.stem(np.cos((np.pi * n)/(D + 1)) + (unit_impulse(20,(G+1))
                                        + unit_impulse(20,(H+1))))
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Original x(n)")
plt.grid()
plt.figure(1)
plt.stem(moving_average_filter(4))
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("N = 4")
plt.grid()
```

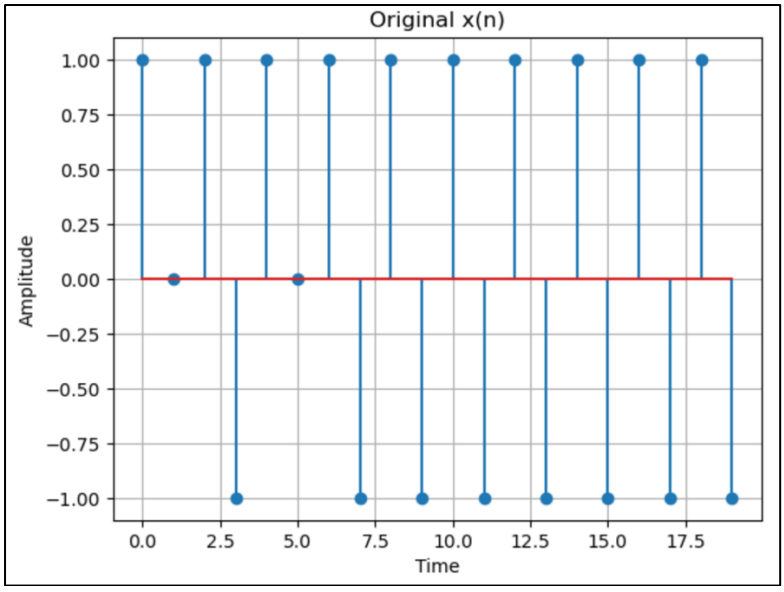

Figure 10: Original function x(n)

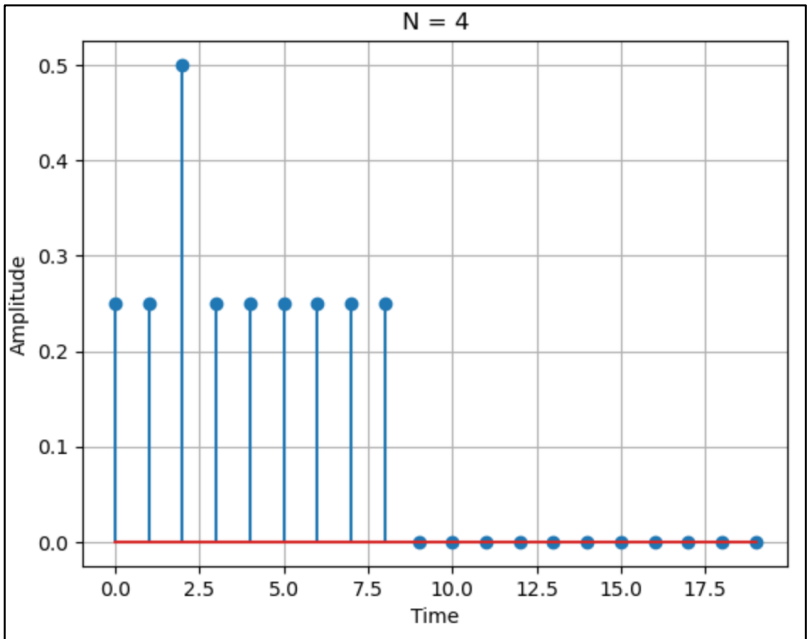

Figure 10: Filter when N=4

```
plt.figure(2)
plt.stem(moving_average_filter(8))
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("N = 8")
plt.grid()
plt.figure(3)
plt.stem(moving_average_filter(12))
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("N = 12")
plt.grid()
```

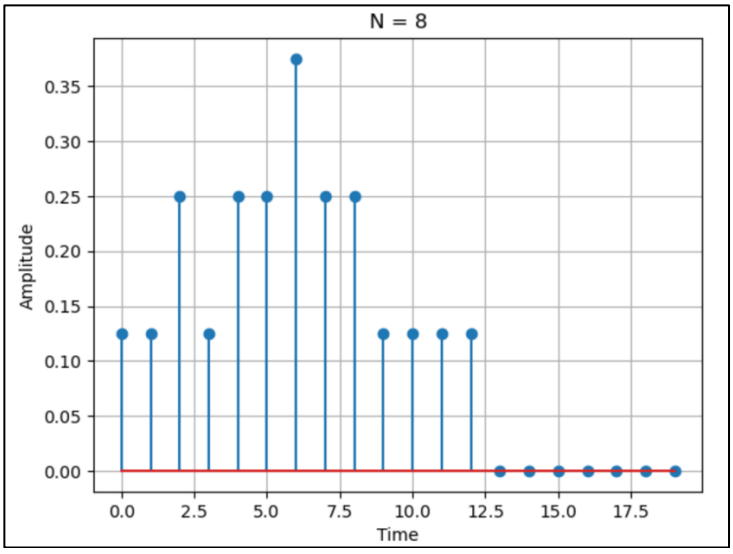

Figure 11: Filter when N=8

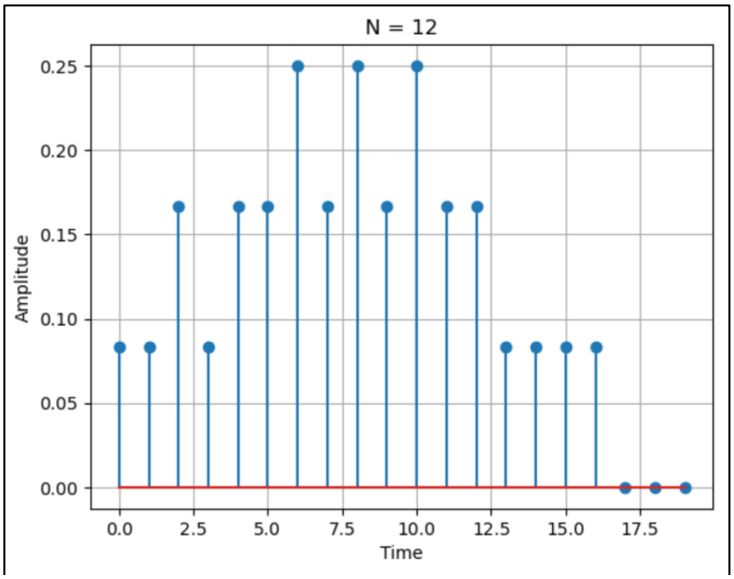

Figure 12: Filter when N=12

The N-Point Moving Average Filter, by definition, averages the input values that lie within the range or window of N and "smooths" them out using the average of the input's prior values. As you are dividing the average by a larger number as the window grows, the output will shrink as N grows. Assuming N = 4, the filter will average the first two values (0 & 1) from the input. where the average will then be divided by N, which in this instance was 4. which will then create the result for n = 2. The N-point moving average filter is typically used to smooth out a signal and reduce surrounding noise. The value of the average signal is 0. As a result, the filtered signal that results approach 0 as N rises, and the effect of the impulse on the size of the output signal diminishes. When the function expands, an average is divided by a greater number; as a result, the output will decrease as N increases due to this inversely proportional connection.