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 1
Assignment/Lab Title:	Time-Domain Analysis of Discrete-Time Systems -Part 1

Submission Date:	February 4, 2023
Due Date:	February 5, 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

In this lab assignment, the concepts of discrete-time signals were explored. A variety of transformations were applied to discrete signals and the resulting signals were plotted. The energy and power of signals were also calculated. Additionally, the equation of a discrete system was determined and the zero-input, zero-state and total system responses were computed using a recursive method.

A) Signal Transformation

1) Python code for Part 1:

```python
import numpy as np
import matplotlib.pyplot as plt

n = np.arange(-10, 10)

# Plotting δ[n - 3]
delta_n = np.zeros(len(n))
delta_n[n == 3] = 1
plt.stem(n, delta_n)
plt.title(r'$\delta[n-3]$')
plt.xlabel('n')
plt.ylabel(r'$\delta[n-3]$')
plt.grid()
plt.show()

# Plotting u[n + 1]
u_n = (n >= -1).astype(int)
plt.stem(n, u_n)
plt.title(r'$u[n+1]$')
plt.xlabel('n')
plt.ylabel(r'$u[n+1]$')
plt.grid()
plt.show()

# Plotting x[n] =cos (πn/5 ) u[n]
x = np.cos(np.pi * n / 5) * (n >= 0).astype(int)
plt.stem(n, x)
plt.title(r'$x[n] = cos(\frac{\pi n}{5}) u[n]$')
plt.xlabel('n')
plt.ylabel(r'$x[n]$')
plt.grid()
plt.show()

# Plotting x1[n] = x[n - 3]
x1 = np.zeros(len(n))
x1[n >= -7] = x[n >= -7]
plt.stem(n, x1)
plt.title(r'$x1[n] = x[n-3]$')
plt.xlabel('n')
plt.ylabel(r'$x1[n]$')
plt.grid()
plt.show()

# Plotting x2[n] = x[-n]
x2 = x[::-1]
plt.stem(n, x2)
plt.title(r'$x2[n] = x[-n]$')
plt.xlabel('n')
plt.ylabel(r'$x2[n]$')
plt.grid()
plt.show()
```

I) $\delta[n-3]$

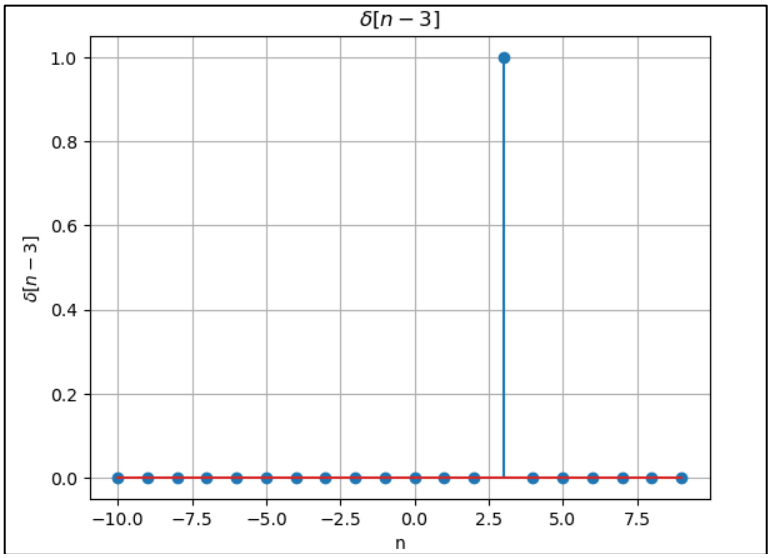

Figure 1: Transformation applied on discrete signal to shift by 3 units to the right

II) $u[n+1]$

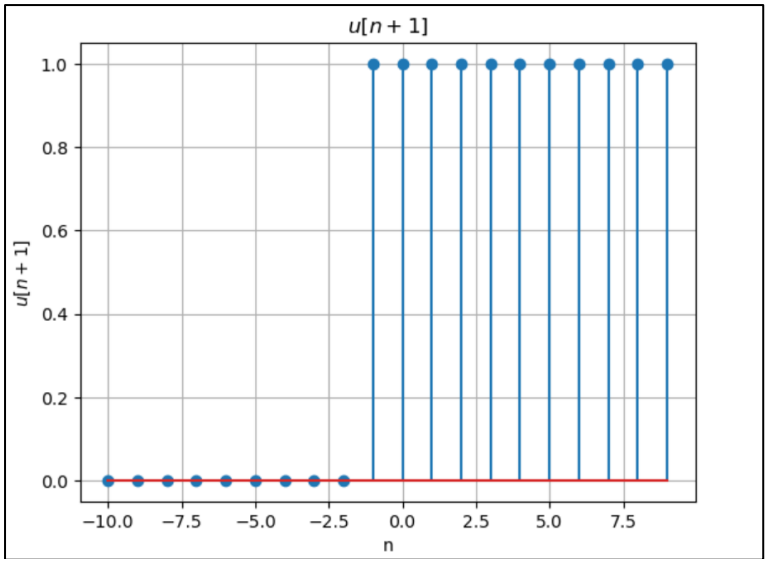

Figure 2: Transformation applied on discrete signal to shift left by 1 unit

III) $x[n] = \cos\left(\frac{\pi n}{5}\right) u[n]$

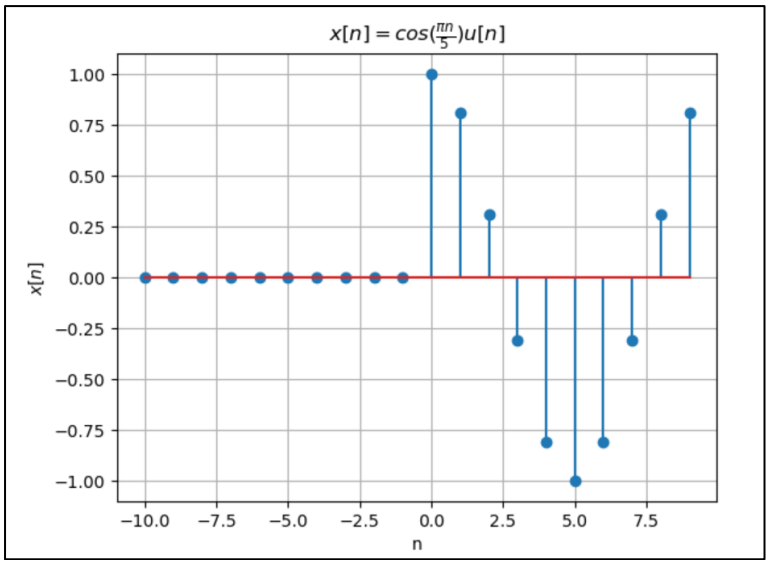

Figure 3: Transformation applied on discrete signal

IV) $x_1[n] = x[n-3]$

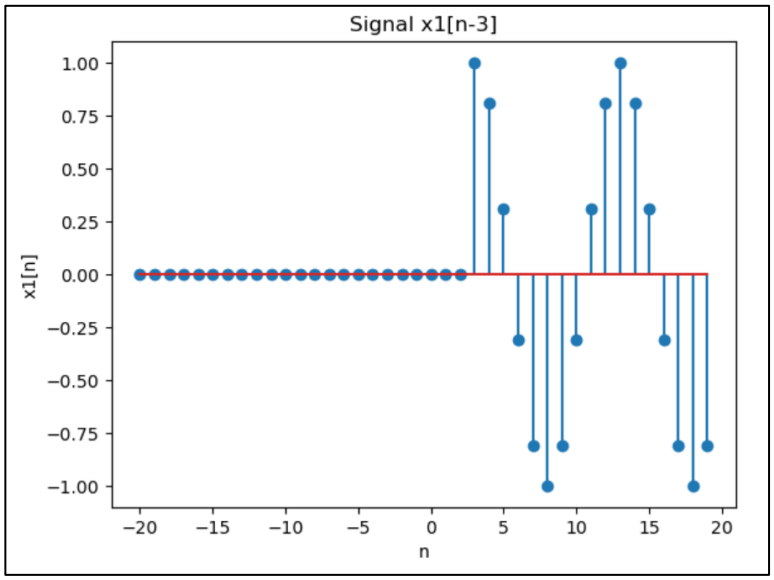

Figure 4: Transformation applied on discrete signal to shift x[n] right by 3

V) $x_2[n] = x[-n]$

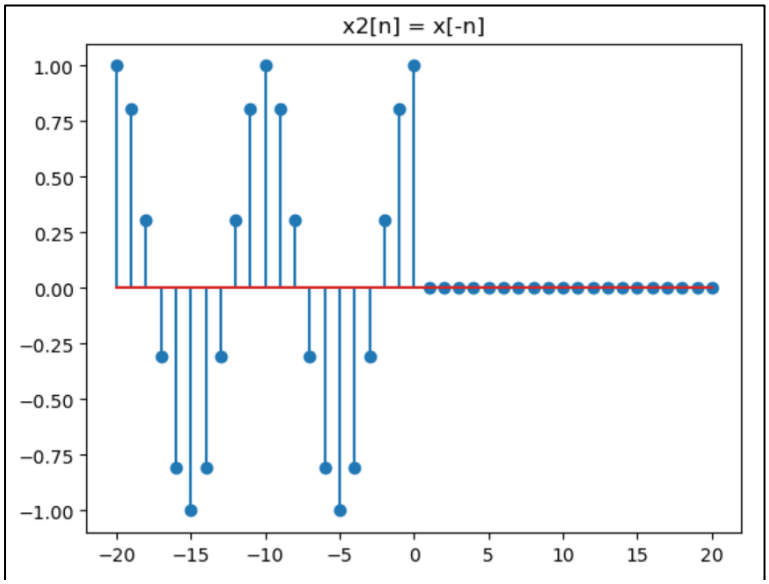

Figure 5: Transformation applied on discrete signal to flip x[n] on the x-axis

In x1[n], the original signal of x[n] shifts to the right by 3 units. For x2[n], the signal goes through time reversal as the original signal x[n] is flipped along the y-axis.

2) Python code for Part 2:

```
import numpy as np
import matplotlib.pyplot as plt

n = np.arange(-10,70)

def u(n):
    return 1.0 * ((n >= 0) & (n <= 70))

def y(n):
    return 5 * np.exp(-n/8) * (u(n) - u(n - 10))

def y1(n):
    return y(3 * n)

def y2(n):
    return y(n/3)

plt.figure(1)
plt.stem(n, y(n))
plt.xlabel('n')
plt.ylabel('y[n]')
plt.title('Signal y[n]')

plt.figure(2)
plt.stem(n, y1(n))
plt.xlabel('n')
plt.ylabel('y1[n]')
plt.title('Signal y1[n]')

plt.figure(3)
plt.stem(n, y2(n))
plt.xlabel('n')
plt.ylabel('y2[n]')
plt.title('Signal y2[n]')

plt.show()
```

I) $y[n] = 5e^{-\frac{n}{8}}(u[n] - u[n-10])$

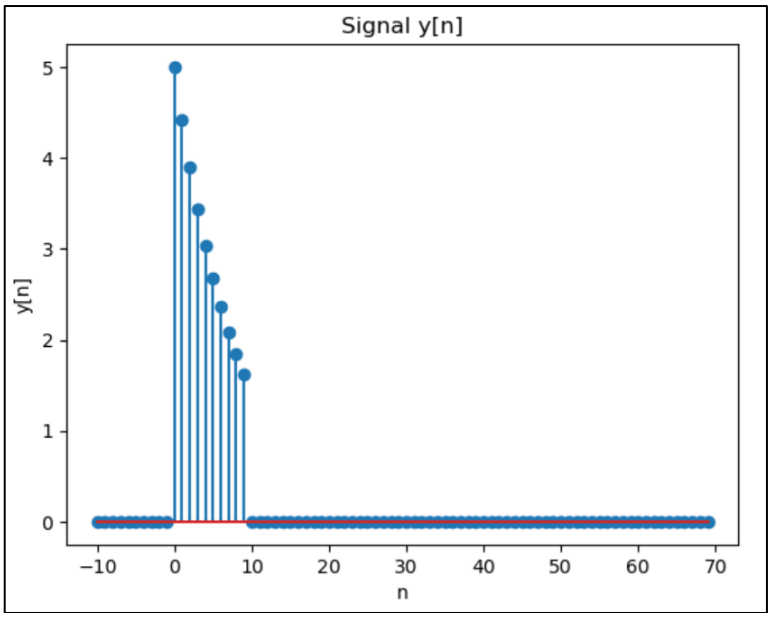

Figure 6: Discrete time signal

II) $y_1[n] = y[3n]$

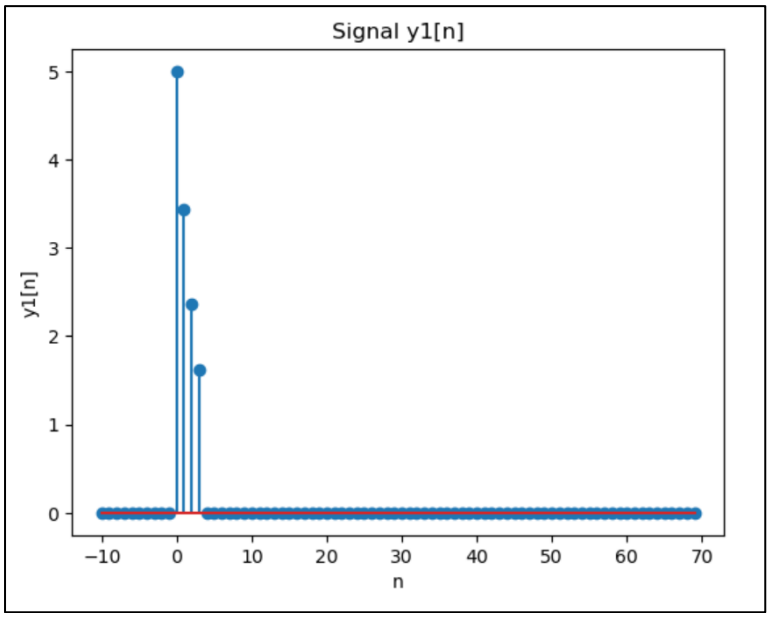

Figure 7: Transformation applied on discrete signal Y[n] to compress by 3

III) $y_2[n] = y[\frac{n}{3}]$

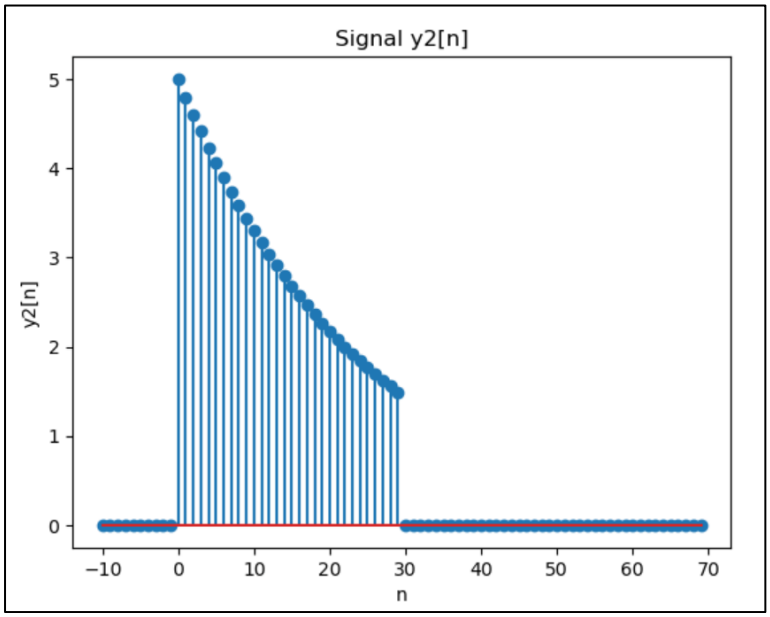

Figure 8: Transformation applied on discrete signal Y[n] to expand by 3

The effect of transformation on y1[n] is that it goes through a compression by a factor of 3 whereas signal y2[n] is expanded by a factor of 3.

3) Python code for Part 3:

```
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(-10, 70, 0.1)
z = lambda t: 5 * np.exp(-t/8) * ((t >= 0) & (t < 10))
y3 = lambda t: z(t/3)

plt.stem(t, y3(t))
plt.xlabel('t')
plt.ylabel('y3(t)')
plt.title('Continuous signal z(t) and its discrete representation y3[n]')
plt.show()
```

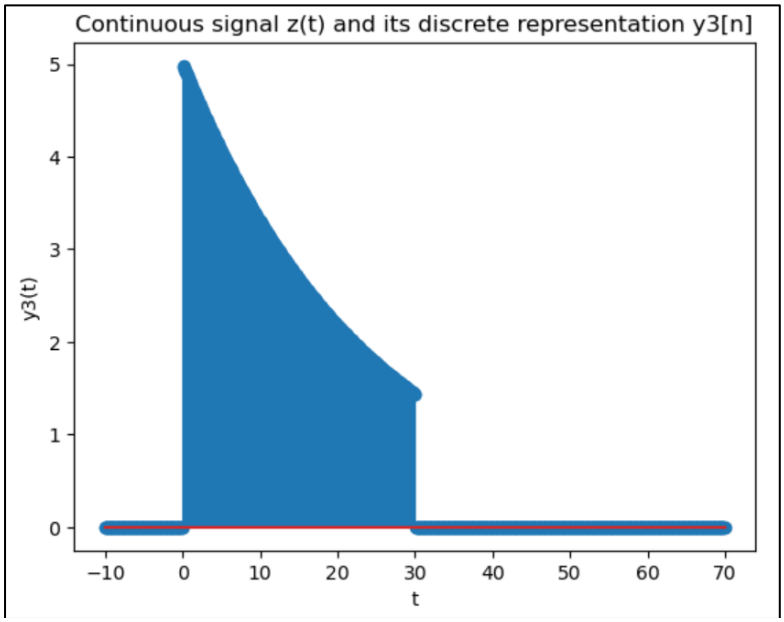

Figure 9: Continuous signal with linear transformation applied

The order of the operations (scaling and sampling) matters, y2[n] and y3[n] are not the same. In the first instance, the original signal y[n] was first sampled, and the resulting discrete signal was then scaled, yielding y2[n]. On the other hand, to obtain y3[n], the continuous signal z(t) was first scaled, and the resulting continuous signal was then sampled. Because the operations are not commutative, i.e., the order of the operations matters, there is a difference between the two. In general, sampling and transformation of a continuous signal do not yield the same results as sampling and transformation of a discrete signal.

B) Recursive solution of difference equation

1) The equation relating the output y[n] to the input x[n] is: $\boldsymbol{y[n] = 1.03 * y[n-1] + x[n]}$

2) Python code for part 2:

```
import numpy as np
import matplotlib.pyplot as plt

s = 13
y = np.zeros(s)
y[0] = 1000
for n in range(s-1):
    yround = round(1.03 * y[n])
    y[n+1] = yround

plt.figure(8)
n = np.arange(12) + 1
plt.stem(n, y[n])
plt.xlabel('Months')
plt.ylabel('Balance')
plt.title('Account balance overtime with zero input')
plt.show()
```

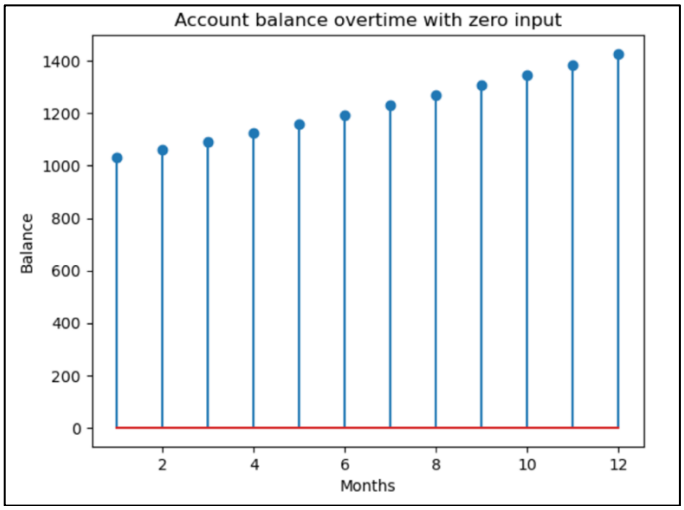

Figure 10: Account balance over time with no deposit in the new year

3) Python code for part 3:

```
import numpy as np
import matplotlib.pyplot as plt

s = 13
y = np.zeros(s)
y[0] = 1000
for n in range(s-1):
    x = 100 * (n + 1)
    yround = round(1.03 * y[n] + x)
    y[n+1] = yround

plt.figure(8)
n = np.arange(12) + 1
plt.stem(n, y[n])
plt.xlabel('Months')
plt.ylabel('Balance')
plt.title('Account balance overtime with monthly deposit of 100n')
plt.show()
```

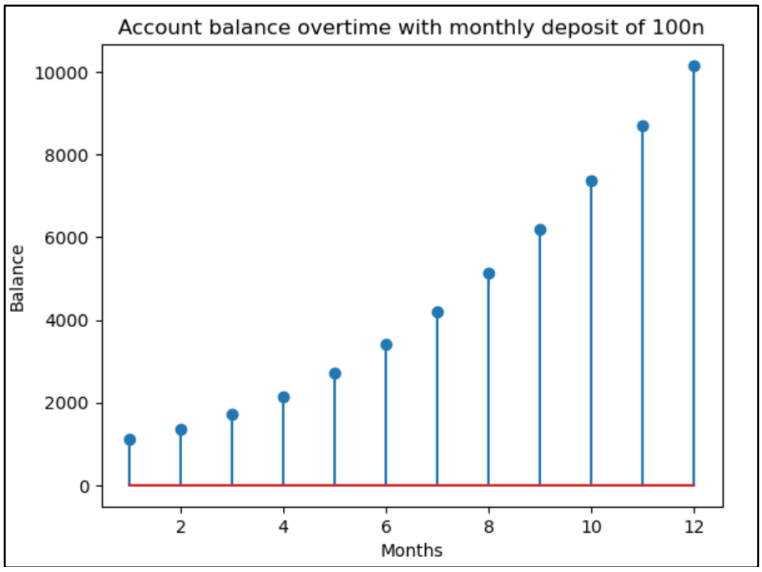

Figure 11: Account balance over time with 100n monthly deposit

C) Design a filter: N-point maximum filter

1) Python function code for Part 1:

```
import numpy as np

def maximum_filter(x, N):
    M = len(x)
    y = np.zeros(M)
    for i in range(M - N + 1):
        y[i + N - 1] = np.amax(x[i : i + N])
    y = np.concatenate([np.zeros(N - 1), y])
    return y
```

2) Python code for Part 2:

```
import numpy as np
import matplotlib.pyplot as plt


def max_filter(x, N):
    M = len(x)
    y = np.zeros(M)
    for n in range(M):
        if n < N - 1:
            y[n] = 0
        else:
            y[n] = np.amax(x[n - N + 1:n + 1])
    return y


n = np.arange(0, 45)
x = np.cos(np.pi * n / 1) - (n == 1) + (n == 5)
y1 = max_filter(x, 4)
y2 = max_filter(x, 8)
y3 = max_filter(x, 12)

plt.figure()
plt.stem(n, x)
plt.grid(True)
plt.xlabel('n')
plt.ylabel('x')
plt.title('Input')

plt.figure()
plt.stem(n, y1)
plt.grid(True)
plt.xlabel('n')
plt.ylabel('Y1')
plt.title('4 Point Filter')

plt.figure()
plt.stem(n, y2)
plt.grid(True)
plt.xlabel('n')
plt.ylabel('Y2')
plt.title('8 Point Filter')

plt.figure()
plt.stem(n, y3)
plt.grid(True)
plt.xlabel('n')
plt.ylabel('Y3')
plt.title('12 Point Filter')

plt.show()
```

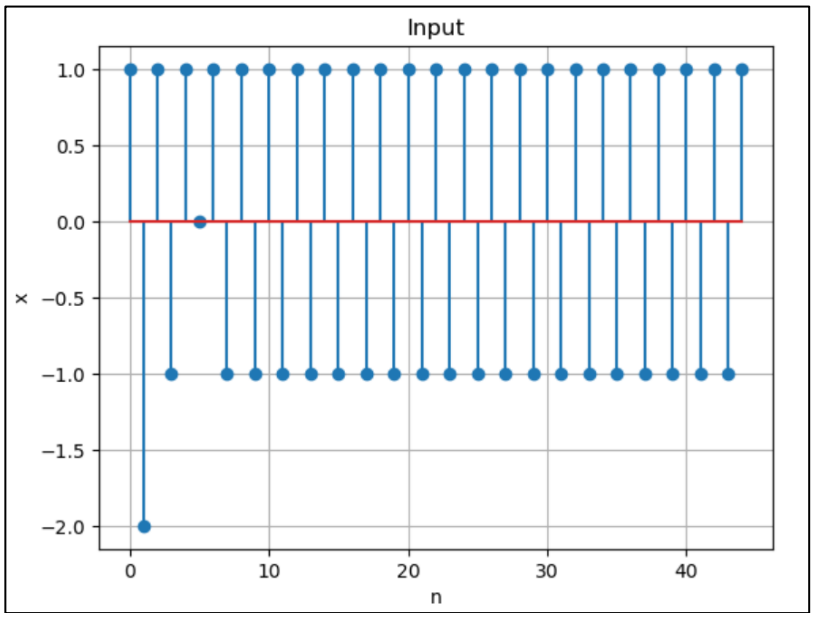

Figure 12: Maximum filter input

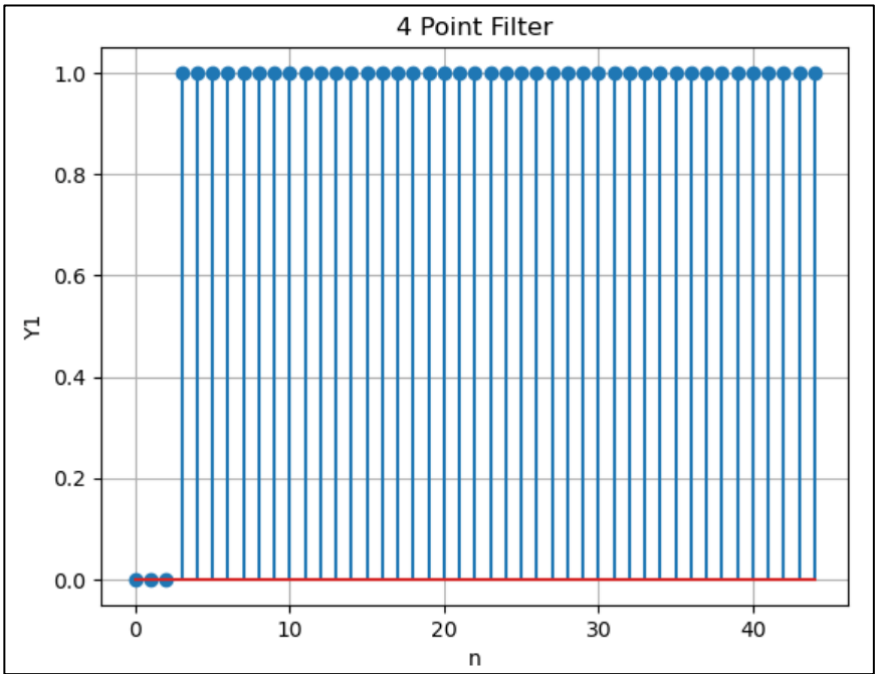

Figure 13: Maximum filter with N=4

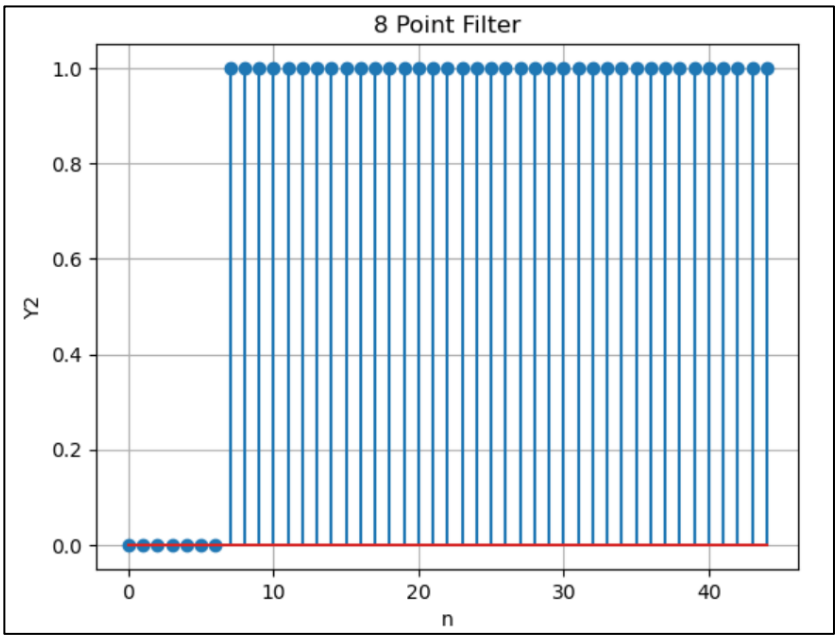

Figure 14: Maximum filter with N=8

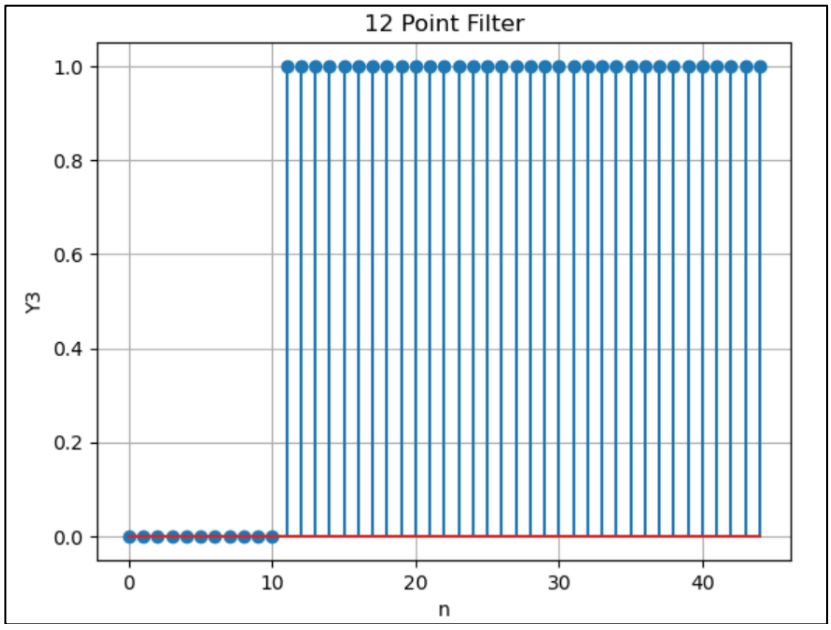

Figure 15: Maximum filter with N=12

3) As N increases, the filter takes a more common maximum within an interval and filter out more oscillations. The filter becomes more aggressive in smoothing out the fluctuations in the signal, resulting in a smoother output.

D) Energy and power of a discrete signal

1 & 2) Python code for function in Part 1:

```
def E(x):
    energy = np.sun((np.abs (x))^2)
    return (energy)
```

```
n = np.arange(-3,4)
def x(n):
    return 3*n * (u(-3,n)-u(4,n))
def energy(x):
    return  sum(val**2 for val in x)
def power(x):
    return (1/len(n))* sum(val**2 for val in x)
plt.stem(n,x(n))
energyResult = energy(x(n))
powerResult = power (x(n))

print("Energy:", energyResult)
print("Power:", powerResult)
```

```
Energy: 252.0
Power: 36.0
```

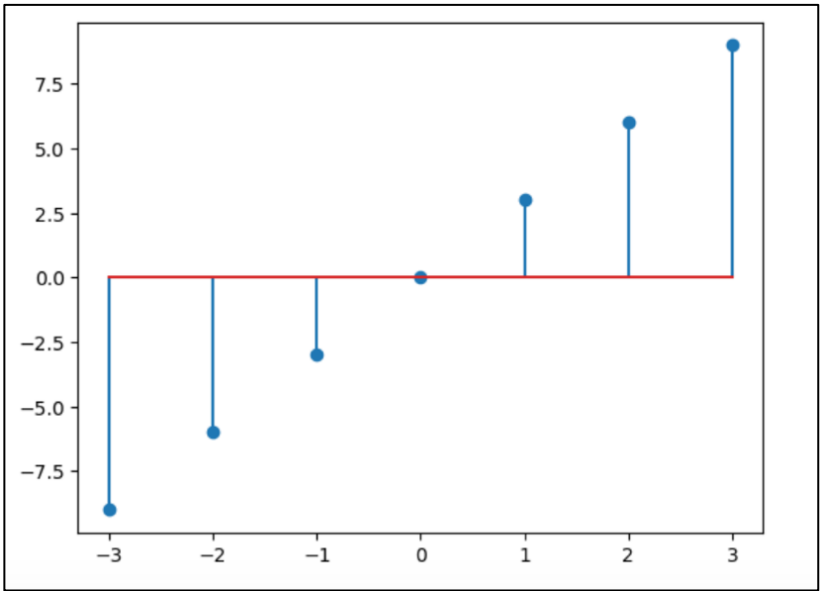

Figure 16: Signal x[n] with power and energy