BIRZEIT UNIVERSITY

DIGITAL SIGNAL PROCESSING (DSP) ENCS4310

Assignment 1

Name: Faris Abufarha

Student ID: 1200546

Instructor: Dr. Qadri Mayyala

Section: 3

Date: 13/1/2023

Abstract

The main objective of the assignment is to get familiar with the technical side of the course using programming languages like MATLAB or Python, for this assignment most codes were written in Python using libraries like NumPy for the computations, and matplotlib for plotting, the last two questions were solved in MATLAB. The Project is published in Github you can see it by clicking Github

Part 1

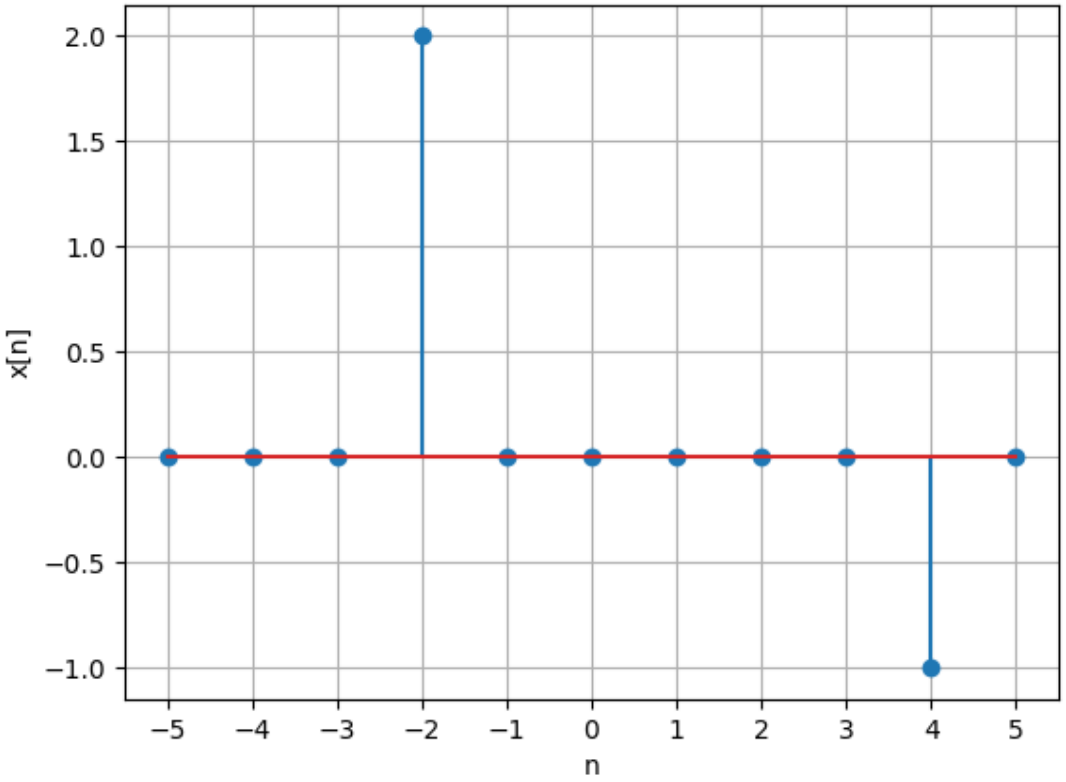
Q1

Generate and plot each of the following sequences over the indicated interval.

A

$$x[n] = 2\delta(n + 2) - \delta(n - 4), \quad -5 \leq n \leq 5.$$

Plot:



Code:

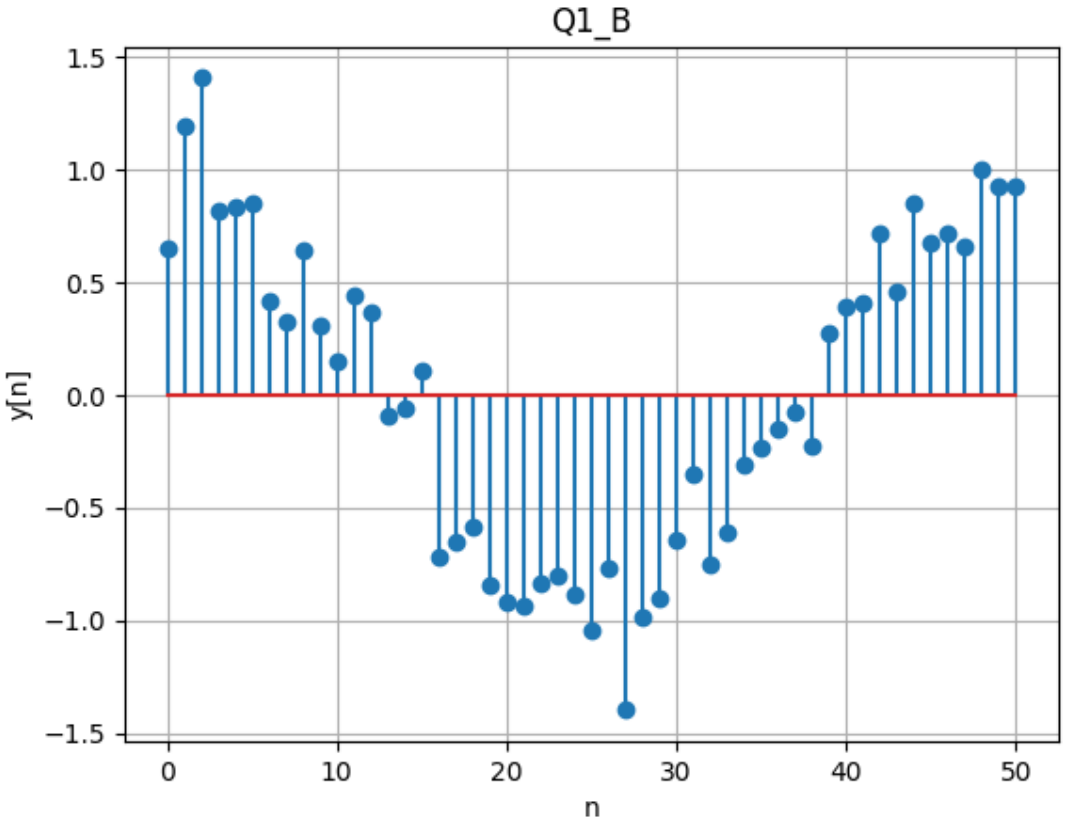
```
import matplotlib.pyplot as plt
import numpy as np
```

```
#  δ(n - a )   a: delay or advance
def unit_impulse(interval, a) -> list:  # returns a list
    unit = []
    for sample in interval:
        if sample == -a:
            unit.append(1)
        else:
            unit.append(0)
    return unit


LL = -5
UL = 5
n = np.arange(LL, UL + 1)
x1 = unit_impulse(n, 2)
x2 = unit_impulse(n, -4)
x = []
for i in range(len(x1)):
    x.append(2 * x1[i] - x2[i])

plt.grid()
plt.stem(n, x)
plt.xlabel('n')
plt.xticks(n)
plt.ylabel('x[n]')
plt.savefig("q1_a.png")
plt.show()
```

Comments:
To Implement the Impulse signal; i made a function that makes a List of integers so on the delayed time we get 1 else we get 0, and then we plot this list

B

$y[n] = cos(0.04\pi n) + 0.2w(n)$, $0 \leq n \leq 50$. where w(n) is a Gaussian random sequence with zero mean and unit variance



Code:

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

# Q1_B
LL = 0  # UPPER LIMIT
UL = 50  # LOWER LIMIT
n = np.arange(LL, UL + 1)
mean = 0
variance = 1
sigma = np.sqrt(variance)
w = np.random.normal(mean, sigma,len(n)) # to generate the 'randn'
y = np.cos(0.04 * np.pi * n) + 0.2 * w
plt.grid() # TO SHOW THE GRID
plt.stem(n, y) # TO PLOT THE GRAPH in discrete form
plt.xlabel('n')
plt.ylabel('y[n]')
plt.title('Q1_B')
plt.savefig("q1_B.png") # SAVING THE PLOT AS A PNG FILE
plt.show()
```

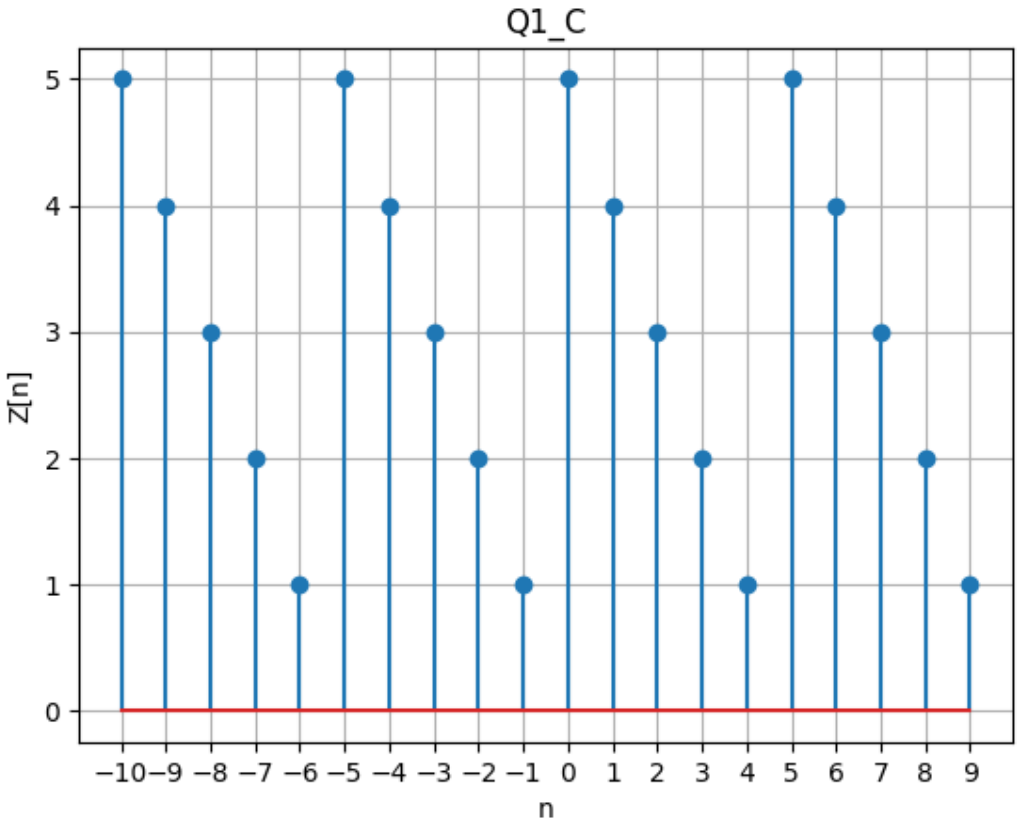
Comments:

We can notice that in each run the plot may differ that because the differ in the Gaussian random numbers

C

$$z[n] = \{..., 5,4,3,2,1,\underline{5},4,3,2,1,5,4,3,2,1, ...\}; \ -10 \leq n \leq 9,$$

Plot



Code:

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

# Q1_C
LL = -10  # UPPER LIMIT
UL = 9  # LOWER LIMIT
```

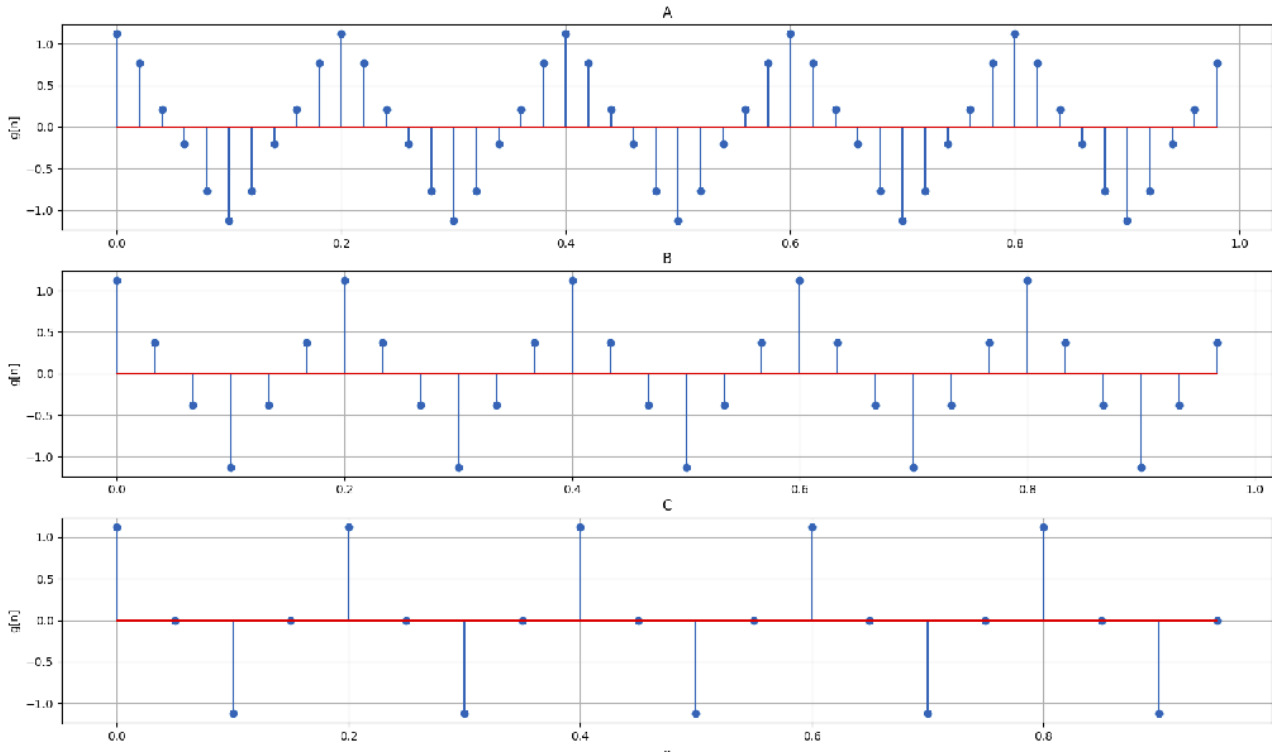
```
n = np.arange(LL, UL + 1)
z = [5 - i % 5 for i in n] # faster way to generate Z


plt.grid()
plt.stem(n, z)
plt.xlabel('n')
plt.xticks(n)
plt.ylabel('Z[n]')
plt.title('Q1_C')
plt.savefig("q1_C.png")
plt.show()
```

Q2

$g(t) = cos(2\pi F_1 t) + 0.125cos(2\pi F_2 t), F_1 = 5Hz, F_2 = 15,$, plot g[n] for one second.

A) For $Fs = 50Hz$
B) For $Fs = 30Hz$
C) For $Fs = 20Hz$

Plot:



Code:

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

f1 = 5
f2 = 15
na = np.arange(0, 1, 1 / 50)
nb = np.arange(0, 1, 1 / 30)
nc = np.arange(0, 1, 1 / 20)

gna = [(np.cos(2 * np.pi * f1 * t) + 0.125 * np.cos(2 * np.pi * f2
* t)) for t in na]
gnb = [(np.cos(2 * np.pi * f1 * t) + 0.125 * np.cos(2 * np.pi * f2
* t)) for t in nb]
gnc = [(np.cos(2 * np.pi * f1 * t) + 0.125 * np.cos(2 * np.pi * f2
* t)) for t in nc]

plt.subplot(3, 1, 1)
plt.stem(na, gna)
plt.grid()
plt.title('A')
plt.ylabel('g[n]')

plt.subplot(3, 1, 2)
plt.stem(nb, gnb)
plt.grid()
plt.title('B')
plt.ylabel('g[n]')

plt.subplot(3, 1, 3)
plt.stem(nc, gnc)
plt.grid()
plt.title('C')

plt.xlabel('n')
plt.ylabel('g[n]')
```

```
# plt.savefig("q1_C.png")
# plt.subplot()
plt.subplots_adjust(top=0.9)
fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)
plt.savefig('Q2.png',dpi=100)


plt.show()
```

Comments

We can notice that the higher the frequency the narrower the signal gets,

f = 1/T

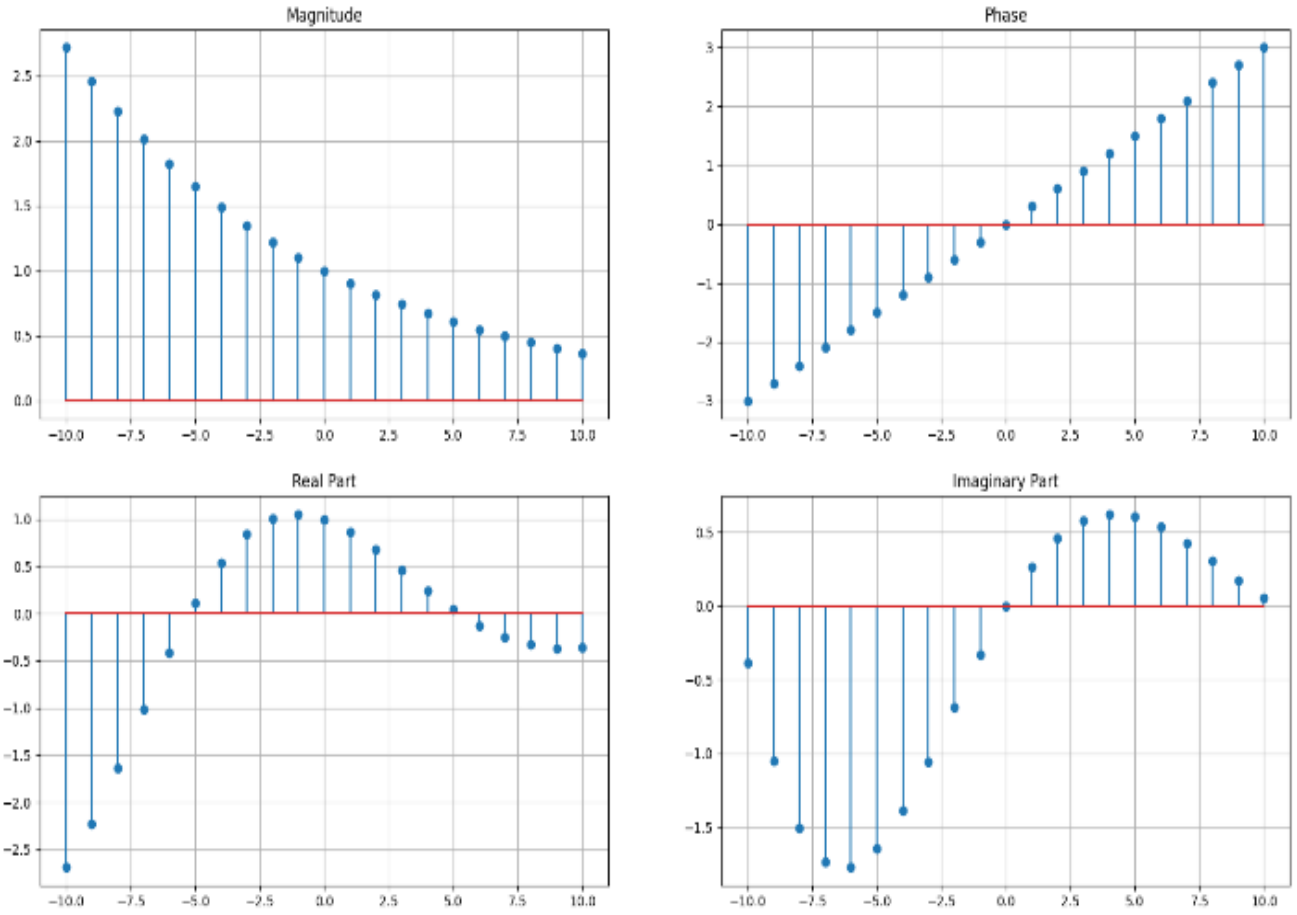
Q3

Generate the complex-valued signal

$$x[n] = e^{(-0.1+j0.3)n}, \quad -10 \leq n \leq 10$$

and plot its magnitude, phase, the real part, and the imaginary part in four separate subplots.

Plot:



Code:

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

LL = -10
UL = 10
n = np.arange(LL, UL + 1)
x = np.exp((-0.1 + 1j * 0.3) * n)

# Plot the magnitude of x[n]
plt.subplot(2, 2, 1)
plt.stem(n, np.abs(x))
plt.title('Magnitude')
plt.grid()

# Plot the phase of x[n]
```

```
plt.subplot(2, 2, 2)
plt.stem(n, np.angle(x))
plt.title('Phase')
plt.grid()

# Plot the real part of x[n]
plt.subplot(2, 2, 3)
plt.stem(n, np.real(x))
plt.title('Real Part')
plt.grid()
# Plot the imaginary part of x[n]
plt.subplot(2, 2, 4)
plt.stem(n, np.imag(x))
plt.title('Imaginary Part')
plt.grid()

# to change plot dimensions
plt.subplots_adjust(top=0.9)
fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)
plt.savefig('Q3.png',dpi=100)


plt.show()
```

Part 2

Q5

Q5. For

$$x[n] = [3, 11, 7, 0, -1, 4, 2], \quad -3 \le n \le 3;$$

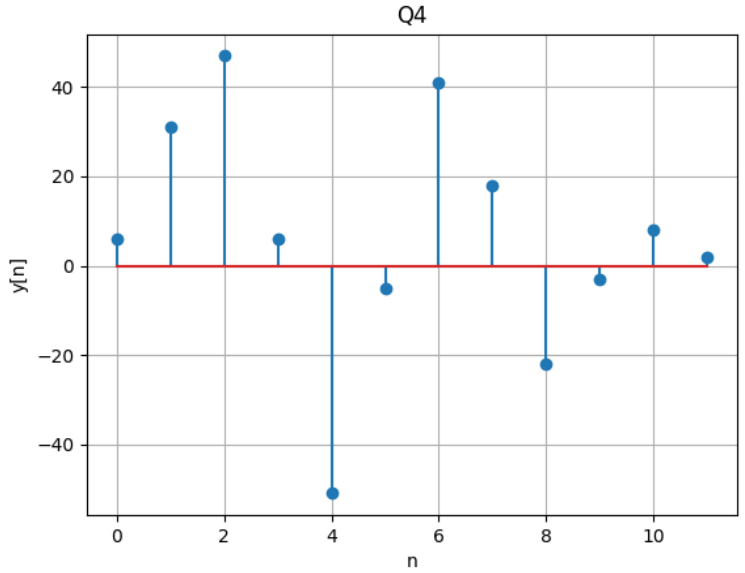
$$h[n] = [2, 3, 0, -5, 2, 1], \quad -1 \le n \le 4$$

Find and plot y[n].

Code:

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

nx = np.arange(-3, 3 + 1)
nh = np.arange(-1, 4 + 1)

xn = np.array([3, 11, 7, 0, -1, 4, 2])

hn = np.array([2, 3, 0, -5, 2, 1])

ny = np.arange(-4, 7 + 1)

# find y[n]
yn = np.convolve(xn, hn)

# plot y[n]
plt.stem(yn)
plt.grid()
plt.title('Q4')
plt.xlabel('n')
plt.ylabel('y[n]')
plt.savefig('Q4.png', dpi=100)
plt.show()
```

Plot



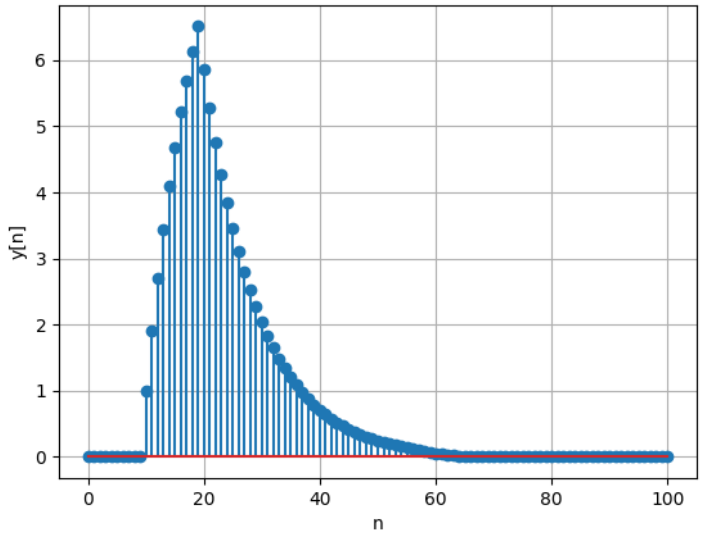
Q6

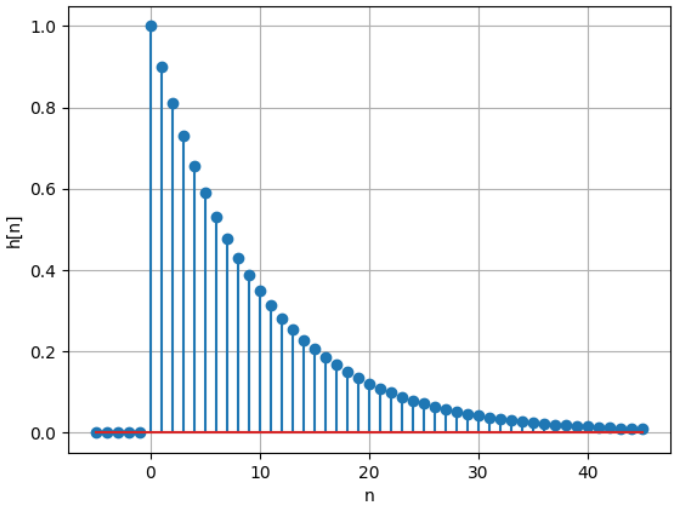
Q6. Let the rectangular pulse x(n) = u(n) − u(n − 10) be an input to an LTI system with impulse response $h[n] = (0.9)^n u(n)$

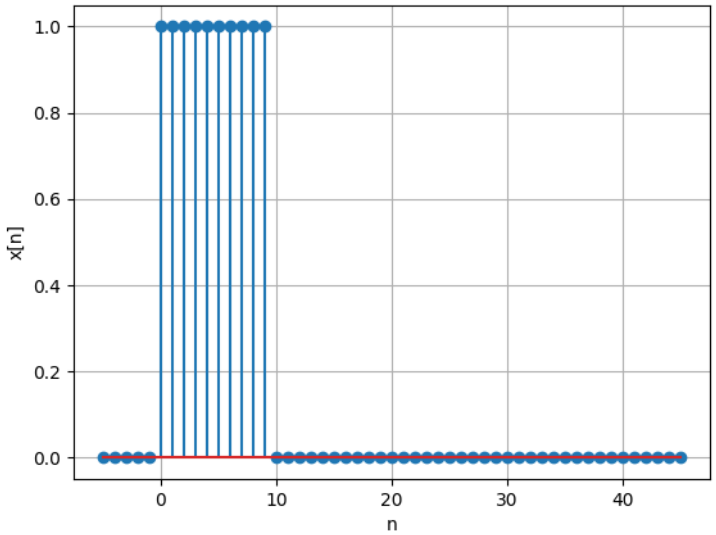
Plot x[n], h[n], Find and plot the output y(n). Consider the interval [−5, 45].

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


# u(n-5)  interval n, delay -5
def unit_step(interval, delay) -> list:
    unit = []
    for sample in interval:
        if sample >= -delay:
            unit.append(1)
        else:
```

```
            unit.append(0)

    return unit


LL = -5
UL = 45
n = np.arange(LL, UL + 1)

xn = [(unit_step(n, 0)[i + -LL] - unit_step(n, -10)[i +-LL]) for i in n]
plt.stem(n, xn)
plt.grid()
plt.xlabel('n')
plt.ylabel('x[n]')
plt.savefig('plots/Q6_X.png') # to save in a certain directory
plt.show()

hn = [(unit_step(n, 0)[i + -LL] * (0.9 ** i)) for i in
      n]  # -LL = --5 = + is for the relative index, there's no index
-5, but relatively it's same as index 0 for the list
plt.stem(n, hn)
plt.grid()
plt.xlabel('n')
plt.ylabel('h[n]')
plt.savefig('plots/Q6_H.png')
plt.show()

yn = np.convolve(xn, hn)
plt.stem(yn)
plt.xlabel('n')
plt.ylabel('y[n]')
plt.grid()
plt.savefig('plots/Q6_Y.png')
plt.show()
```


y[n]
n
0
1
2
3
4
5
6
0
20
40
60
80
100


h[n]
n
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40


x[n]
n
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40

Q7

Q7. To demonstrate one application of the crosscorrelation sequence.

Let $x[n] = [3,11,7,\underline{0},-1,4,2]$ be a prototype sequence,

A) let y(n) be its noise-corrupted-and-shifted version

$$y[n]=x[n-2]+w[n]$$

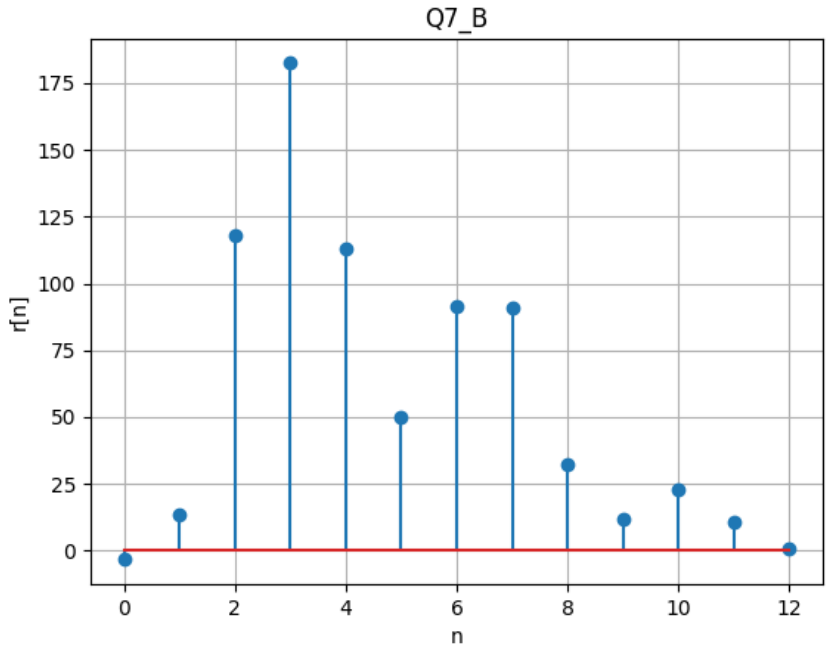
where w[n] is Gaussian sequence with mean 0 and variance 1. Compute the crosscorrelation between y[n] and x[n] and comment on the results.

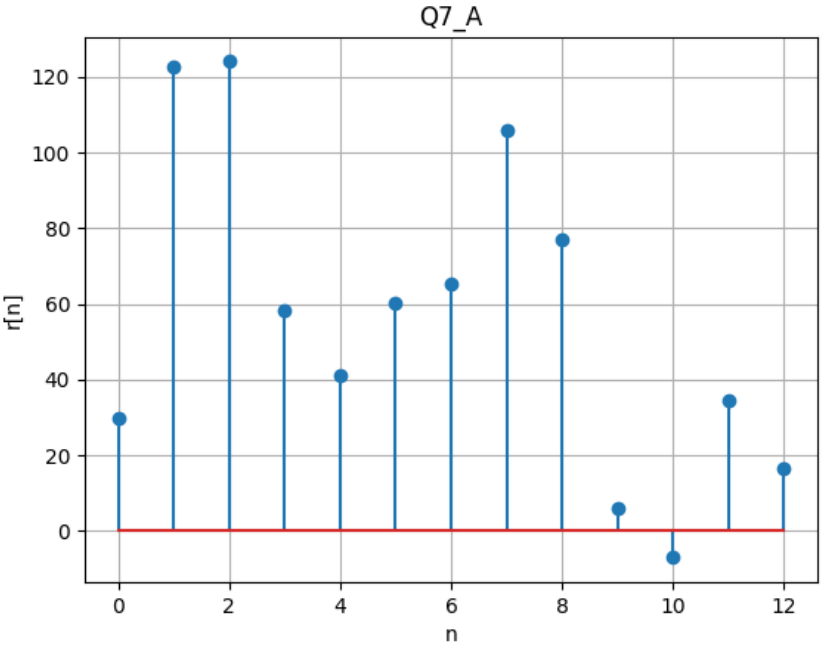
B) Repeat part (a) for $y[n]=x[n-4]+w[n]$

Code:

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

LL = -3
UL = 3
n = np.arange(LL, UL + 1)
xn = np.array([3, 11, 7, 0, -1, 4, 2])
w = np.random.normal(0, 1, size=len(xn))
# yn = [xn[i - 2 - LL] + w[i - LL] for i in n]
yn = np.roll(xn, -2) + w  # shift x by 2 and add noise
r = np.correlate(xn, yn, mode='full')

plt.stem( r)
plt.grid()
plt.title('Q7_A')
plt.xlabel('n')
plt.ylabel('r[n]')
plt.yticks()
plt.show()
plt.savefig('Q7_A.png', dpi=100)

## Q7_B
# yn = [xn[i - 4 - LL] + w[i - LL] for i in n]
yn = np.roll(xn, -4) + w  # shift x by 4 and add noise
r = np.correlate(xn, yn, mode='full') # same as xcorr in matlab

# nr = np.arange(0, 26)
plt.stem( r)
plt.grid()
plt.title('Q7_B')
plt.xlabel('n')
plt.ylabel('r[n]')
plt.yticks()
plt.show()
plt.savefig('Q7_B.png', dpi=100)
```

Plot


Q7_B
r[n]
n
175
150
125
100
75
50
25
0
0
2
4
6
8
10
12


Q7_A
r[n]
n
120
100
80
60
40
20
0
0
2
4
6
8
10
12

Q8

Q8. Given the following difference equation

$$y[n] - y[n-1] + 0.9y[n-2] = x[n]$$

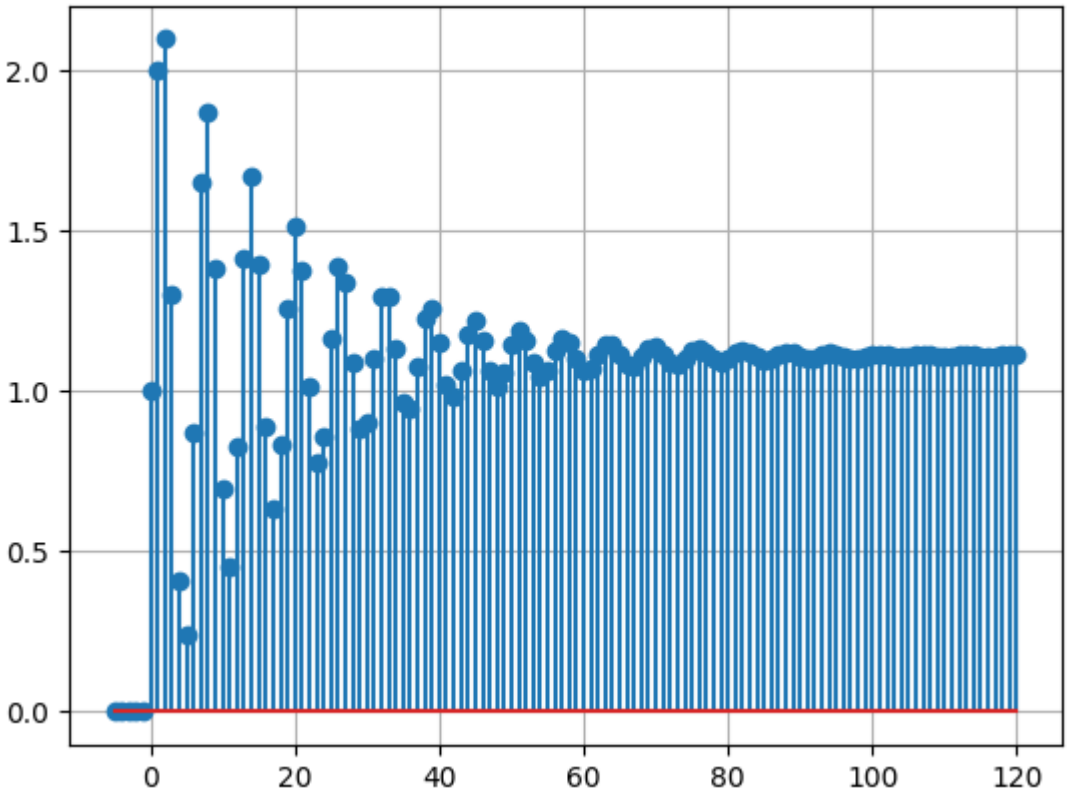
A) Calculate and plot the impulse response h(n) at n = −5,..., 120.
B) Calculate and plot the unit step response s(n) at n = −5,..., 120.
C) Is the system specified by h(n) stable?

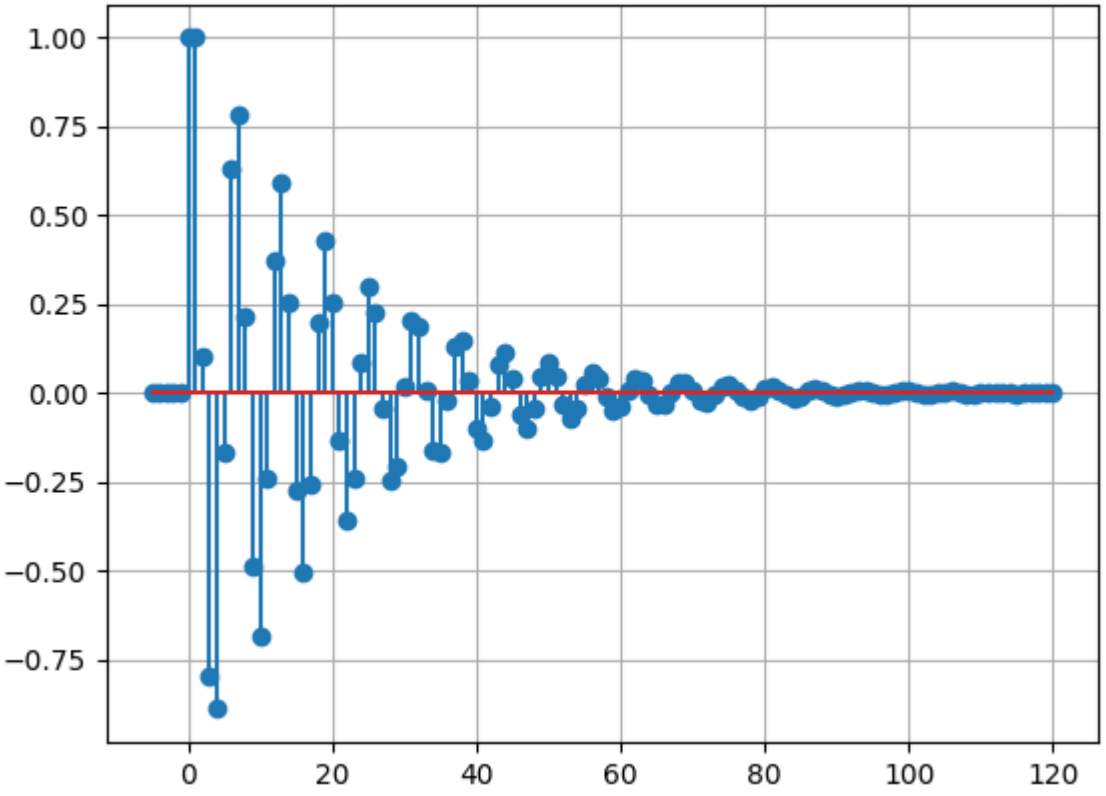
Code:

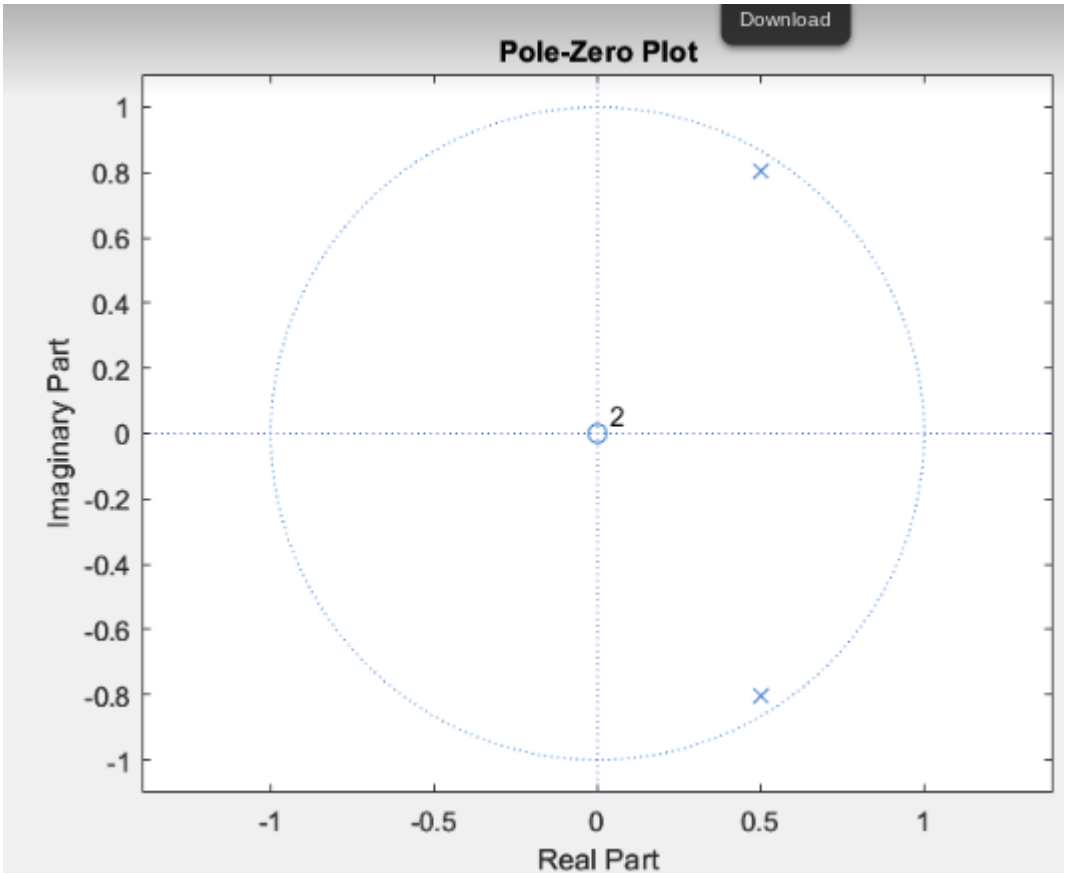
```
import numpy as np
import matplotlib.pyplot as plt
import os
import random
import scipy as scp
from scipy.signal import lfilter, lfilter_zi

def unit_step(interval, delay) -> list:
   unit = []
   for sample in interval:
       if sample >= -delay:
           unit.append(1)
       else:
           unit.append(0)

   return unit

def unit_impulse(interval, a) -> list:  # returns a list
   unit = []
   for sample in interval:
       if sample == -a:
           unit.append(1)
       else:
           unit.append(0)
   return unit


# n = [i for i in range(-5, 120 + 1)]
n = np.arange(-5, 120 + 1)

xn = unit_impulse(n, 0)

ak = [1, -1, 0.9]
bk = [1]
hn = lfilter(b=bk, a=ak, x=xn)
plt.stem(n,hn)
plt.grid()
plt.savefig('Q8_A.png')
plt.show() # ok

#b
xn = unit_step(n, 0)
sn= lfilter(b=bk, a=ak, x=xn)
```

```
plt.stem(n, sn)
plt.grid()
plt.savefig('Q8_B.png')
plt.show()


#c
#check if hn is stable or not

rts = np.roots(hn)
if all(abs(rts) < 1):
   print("System is stable")
else:
   print("System is not stable")
```

Plots (A,B, and C):


2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
120


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
20
40
60
80
100
120



Comments:
In section C, we just can determine if it's stable or not by checking if it's less than , s=which means all points inside the unit circle

Q9

Q9. A "simple" digital differentiator is given by

$$y[n] = x[n] - x[n-1]$$

which computes a backward first-order difference of the input sequence. Implement this differentiator on the following sequences, and plot the results. Comment on the appropriateness of this simple differentiator.

A) Rectangular pulse: **x[n]= 5[u(n) − u(n − 20)]**

B) Triangular pulse: **x[n] = n (u[n] − u[n − 10]) + (20−n) (u[n − 10] − u[n − 20])**

C) Sinusoidal pulse: $x[n] = \sin\left(\frac{\pi n}{25}\right)(u[n] - u[n-100])$

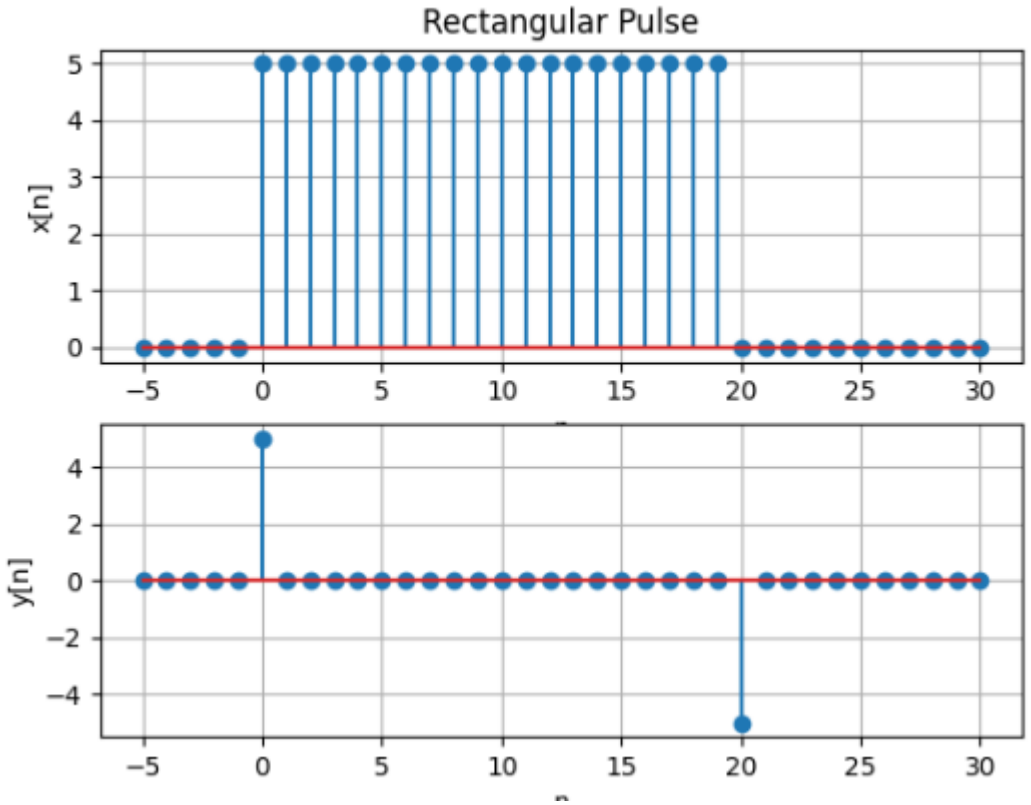
Code

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

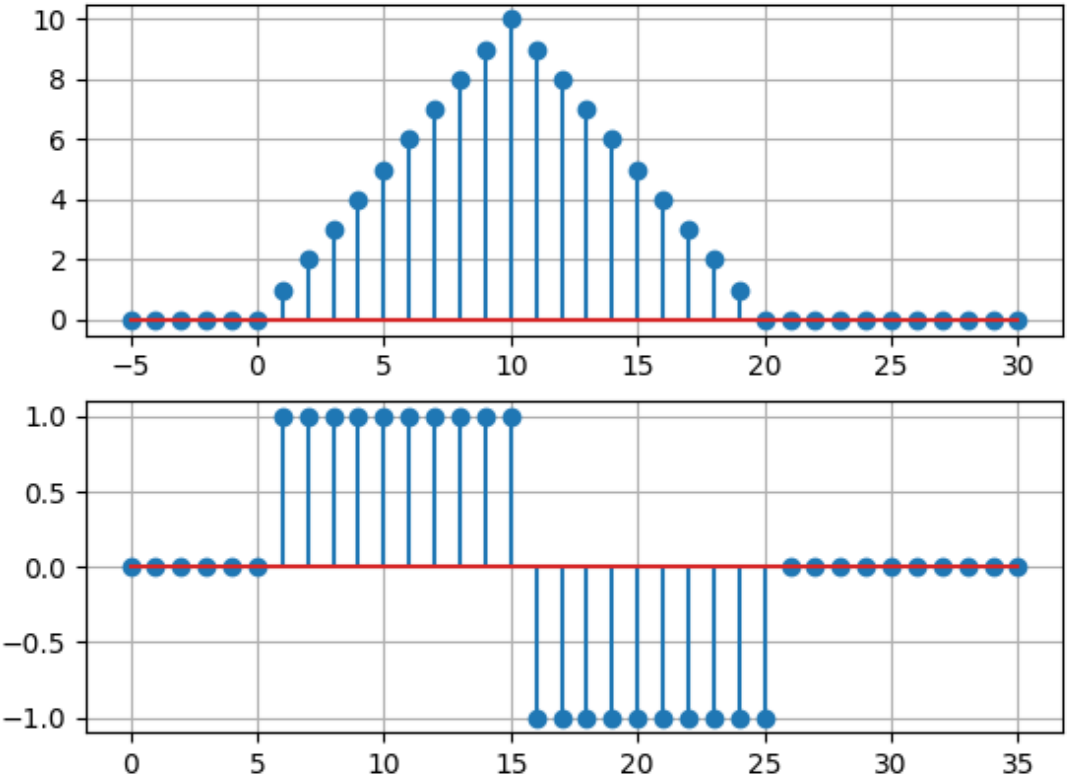
```
import random
import scipy as scp
from scipy.signal import lfilter, lfilter_zi


#  δ(n - a )   a: delay or advance
def unit_impulse(interval, a) -> list:  # returns a list
   unit = []
   for sample in interval:
       if sample == -a:
           unit.append(1)
       else:
           unit.append(0)
   return unit


def unit_step(interval, delay) -> list:
   unit = []
   for sample in interval:
       if sample >= -delay:
           unit.append(1)
       else:
           unit.append(0)

   return unit

LL = -5
UL = 30
n = np.arange(LL, UL + 1)
rec = [5 * (unit_step(n, 0)[i - LL] - unit_step(n, -20)[i- LL]) for i in n]
plt.subplot(2, 1, 1)
plt.stem(n, rec)

plt.grid()
plt.xlabel('n')
plt.ylabel('x[n]')
plt.title('Rectangular Pulse')

yn = rec - np.roll(rec, +1) # ask about this

plt.subplot(2, 1, 2)
plt.stem(n, yn)
plt.grid()
plt.xlabel('n')
plt.ylabel('y[n]')
plt.show()

# b

trig = [i * (unit_step(n, 0)[i- LL] - unit_step(n, -10)[i- LL]) + (20 - i) *
(unit_step(n, -10)[i- LL] - unit_step(n, -20)[i- LL]) for i
       in n]
plt.subplot(2, 1, 1)
plt.stem(n, trig)
plt.grid()
plt.savefig('Q9_B.png')
yn = trig - np.roll(trig, 1) # ask about this
plt.subplot(2, 1, 2)
plt.stem( yn)
```

```
plt.grid()
plt.show()

# c
LL = 0
UL = 100
n = np.arange(LL, UL + 1)
sinus = [np.sin(np.pi * i / 25) * (unit_step(n, 0)[i- LL] - unit_step(n, -100)[i-
LL]) for i in n]
plt.subplot(2, 1, 1)
plt.stem(n, sinus)
plt.grid()

yn = sinus - np.roll(sinus, 1) # ask about this
plt.subplot(2, 1, 2)
plt.stem(n, yn)
plt.grid()
plt.show()
```

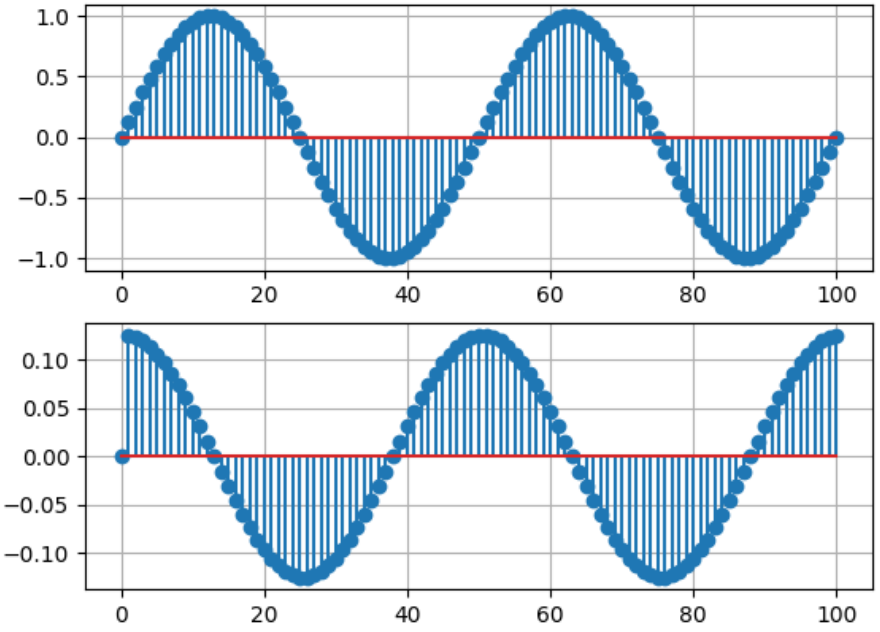
Plot
A:



B:



C:



Part 4

Q10

Q10. For $\mathbf{x[n]=(0.5)^n\ u[n]}$. The corresponding DTFT is $X(e^{j\omega})=\frac{e^{j\omega}}{e^{j\omega}-0.5}$.

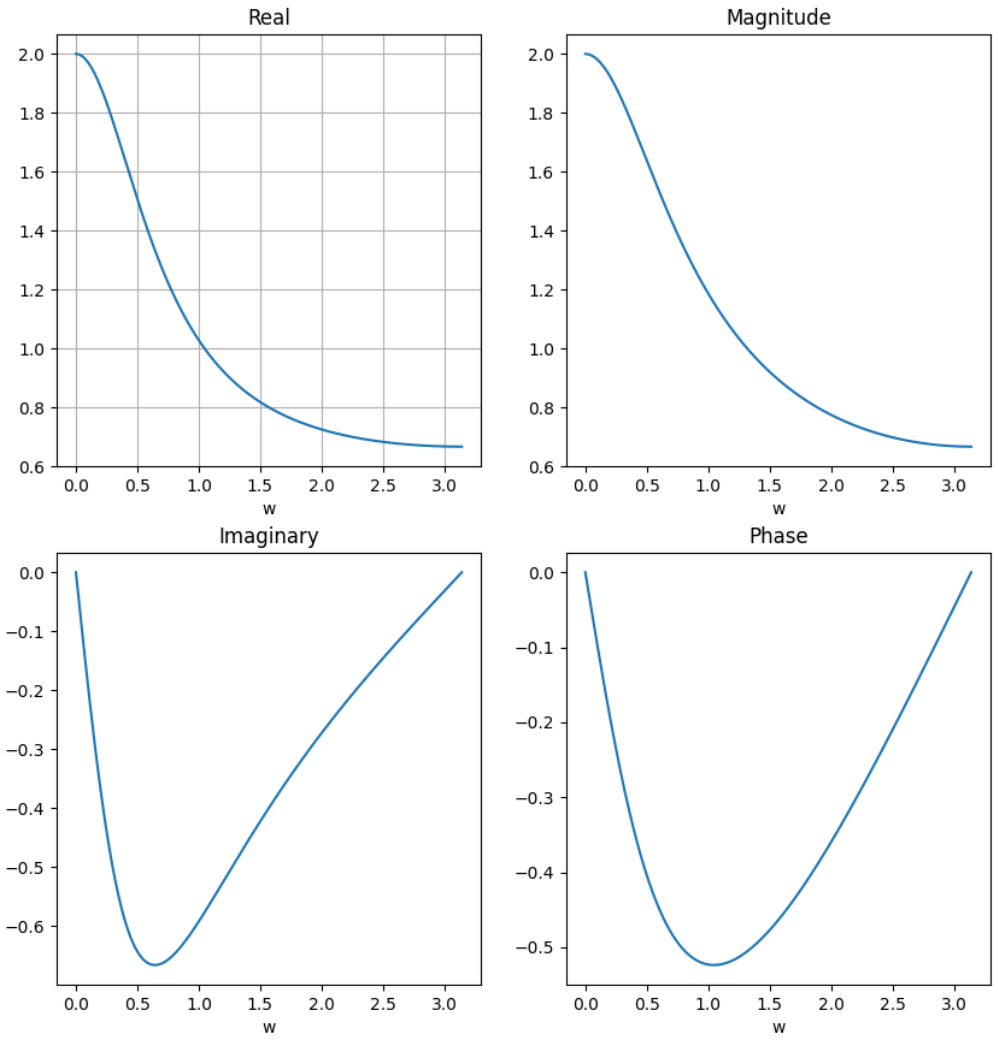
Evaluate $X(e^{j\omega})$ at 501 equispaced points between $[0,\pi]$ and plot its magnitude, angle, real, and imaginary parts.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import os
import random
import scipy as scp
from scipy.signal import lfilter, lfilter_zi

LL = 0
UL = 500
w = np.arange(LL, UL + 1) * np.pi / 500 # 500 samples
X_ejw = np.exp(1j*w) / (np.exp(1j*w) - 0.5*np.ones(UL + 1))


angle = np.angle(X_ejw)
plt.grid()
plt.subplot(2, 2, 4)
```

```
plt.title('Phase')
plt.xlabel('w')
plt.plot(w, angle)

real = np.real(X_ejw)
plt.subplot(2, 2, 1)
plt.grid()
plt.title('Real')
plt.xlabel('w')
plt.plot(w, real)

imag = np.imag(X_ejw)
plt.subplot(2, 2,3)
plt.grid()
plt.title('Imaginary')
plt.xlabel('w')
plt.plot(w, imag)

mag = np.abs(X_ejw)
plt.grid()
plt.subplot(2, 2, 2)
plt.title('Magnitude')
plt.xlabel('w')
plt.plot(w, mag)


plt.subplots_adjust(top=0.9)
fig = plt.gcf()
fig.set_size_inches(10, 10)
plt.savefig('Q10.png',dpi=100)
plt.show()
```

Plot:



Q11

Q.11 Consider the sequence $x[n]=\{1,-0.5,-0.3,-0.1\}$

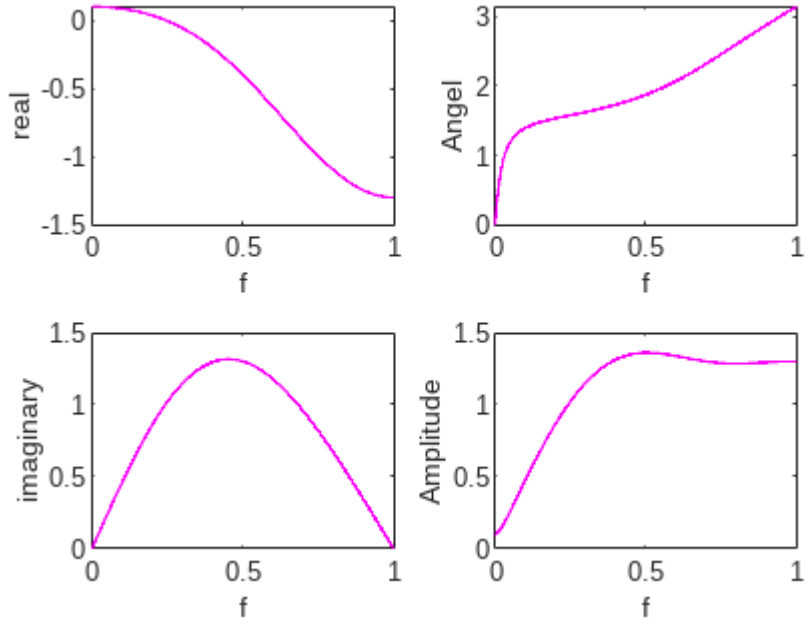
A) Numerically compute the discrete-time Fourier transform of at 501 equispaced frequencies between $[0,\pi]$.

B) plot its magnitude, angle, real, and imaginary parts.

Code:

```
w = [0:1:500]*pi/500
x = [1,-0.5,-0.3,-0.1]
y = zeros(1,length(w))
for i =1 : length(w)
   for n=-1 :2
       y(i) = y(i) + x(n+2)*exp(-j*w(i)*n);

   end
end

subplot (2,2,1);
plot(w/pi, real(y),"m");
xlabel("f");
ylabel('real');
subplot (2,2,2);
plot(w/pi, angle(y),"m");
xlabel("f");
ylabel ("Angel");
subplot(2,2,3);
plot(w/pi, imag(y),"m");
xlabel("f");
ylabel ("imaginary");
subplot(2,2,4);
plot (w/pi,abs(y),"m");
xlabel("f");
ylabel("Amplitude")
```

Plot:



Q12

Q.12 Let $x[n]=\cos(\frac{\pi n}{2})$, $0 \le n \le 100$ and $y[n]=e^{j\pi n/4}x[n]$

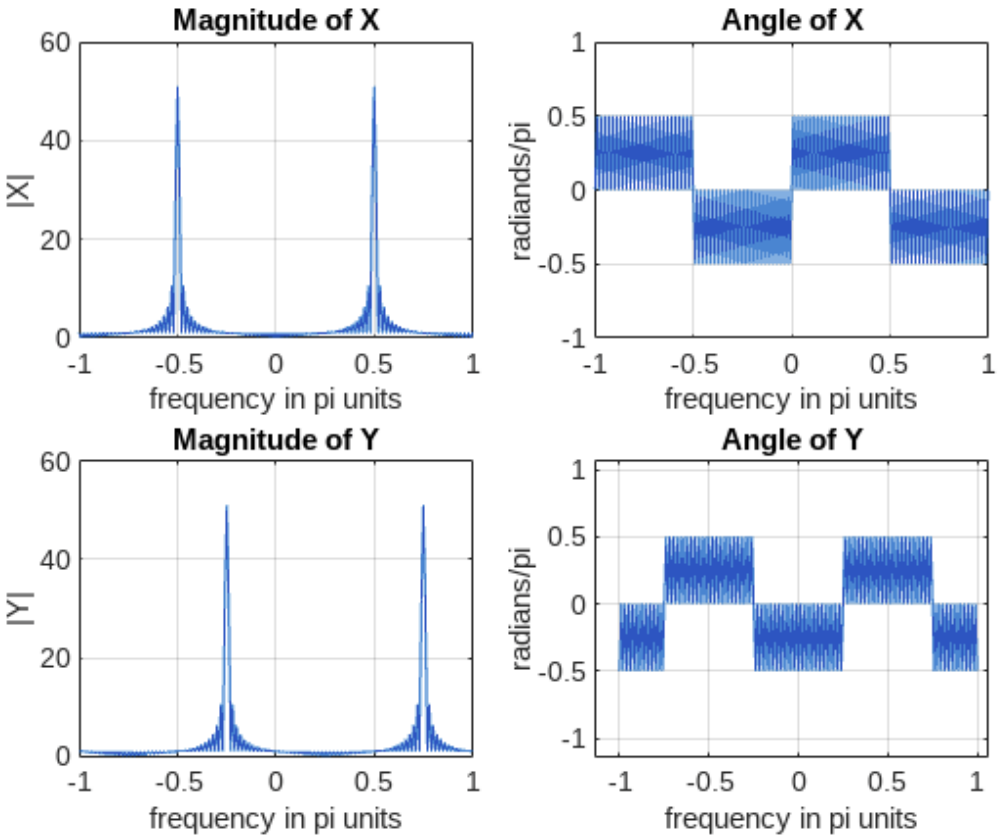
A) Numerically compute the discrete-time Fourier transform of at 401 equispaced frequencies between $[-2\pi,2\pi]$.
B) plot its magnitude, angle spectrum.
C) Comment on the relation between x[n] and y[n].

Code:

```
n = 0:100;
 x = cos(pi*n/2);
 k = -100:100;
 w = (pi/100)*k; % frequency between -pi and +pi
  X = x * (exp(-j*pi/100)).^(n'.*k); % DTFT of x%
  y = exp(j*pi*n/4).*x; % signal multiplied by exp(j*pi*n/4)
  Y = y * (exp(-j*pi/100)).^(n'.*k); % DTFT of y
  % Graphical verification
   subplot(2,2,1);
   plot(w/pi,abs(X));
   grid;
   axis([-1,1,0,60])
   xlabel('frequency in pi units');
   ylabel('|X|')
```

```
title('Magnitude of X')
subplot(2,2,2);
plot(w/pi,angle(X)/pi);
grid; axis([-1,1,-1,1])
xlabel('frequency in pi units');
ylabel('radiands/pi')
title('Angle of X')
subplot(2,2,3);
plot(w/pi,abs(Y));
grid;
axis([-1,1,0,60])
xlabel('frequency in pi units');
ylabel('|Y|')
title('Magnitude of Y')
subplot(2,2,4);
plot(w/pi,angle(Y)/pi);
grid;
axis([-1,1,-1,1])
xlabel('frequency in pi units');
ylabel('radians/pi')
title('Angle of Y')
```

Plot:



C:

From the above code, we can see that x[n] is a cosine function and y[n] is a product of a cosine function and an exponential function. The Fourier Transform of x[n] will have magnitude spectrum with two peaks at positive and negative pi/2 and the angle spectrum will have a constant value of pi/2. On the other hand, y[n] will have magnitude spectrum with four peaks at positive and negative pi/4 and pi/2 and pi*3/4 and the angle spectrum will have a linear phase with a slope of pi/4. The relation between x[n] and y[n] is that y[n] is a phase-shifted version of x[n] by pi