BERZIET UNIVERSITY

Faculty of Engineering & Technology – Electrical & Computer Engineering Department

First Semester 2022/2023

DIGITAL SIGNAL PROCESSING (DSP)

ENCS4310

Assignment#1

Prepared by: Mohammad AbuJaber

ID: 1190298

Instructor: Dr. Qadri Mayyala

Section: 3

Date: 13th January 2023

Abstract:

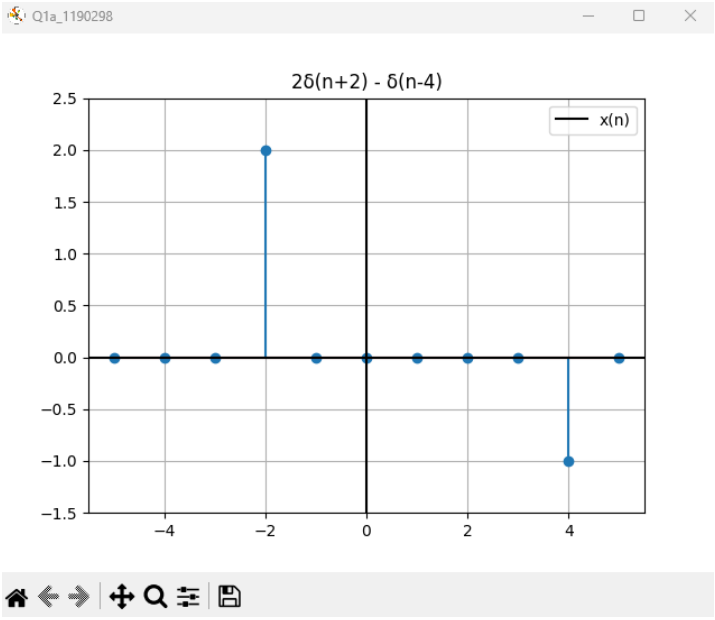
This assignment focused on the practical application of various signal processing concepts, including convolution, correlation of sequences, difference equations, and the discrete-time Fourier transform (DTFT). We practiced plotting different types of signals, such as rectangular and sinusoidal pulses, and analyzed the magnitude, angle, real, and imaginary parts of the DTFT. Additionally, we explored the properties of Gaussian random sequences and their effect on a given signal. Overall, this assignment provided hands-on experience in working with these important signal processing concepts and techniques.

Table of Contents

Part1

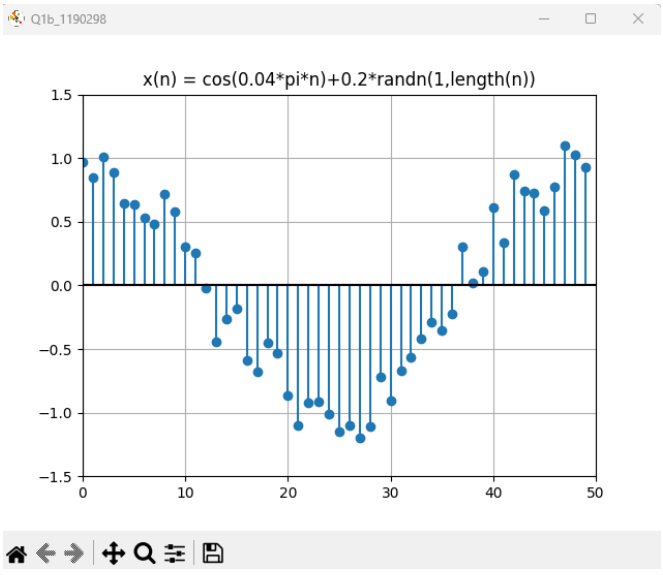
Q1. Generate and plot each of the following sequences over the indicated interval.

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



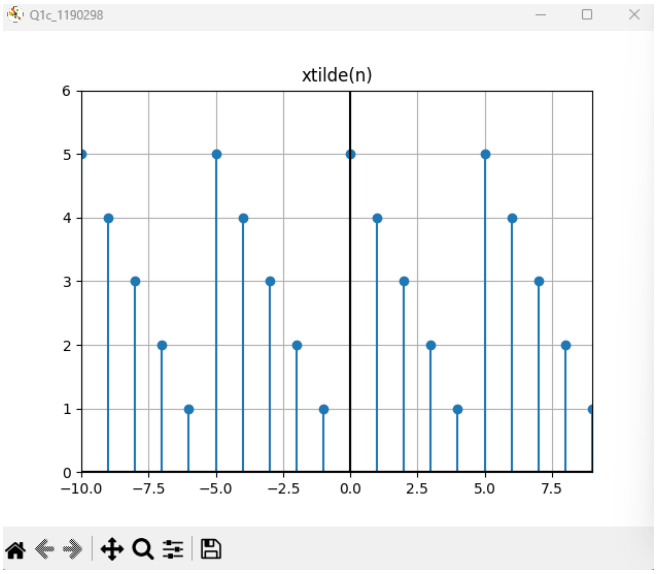
It is what expected, as it has zero value in the range [-5,5] except at 4 and -2.

B) $y[n] = \cos(0.04\pi n) + 0.2w(n), 0 \leq n \leq 50$



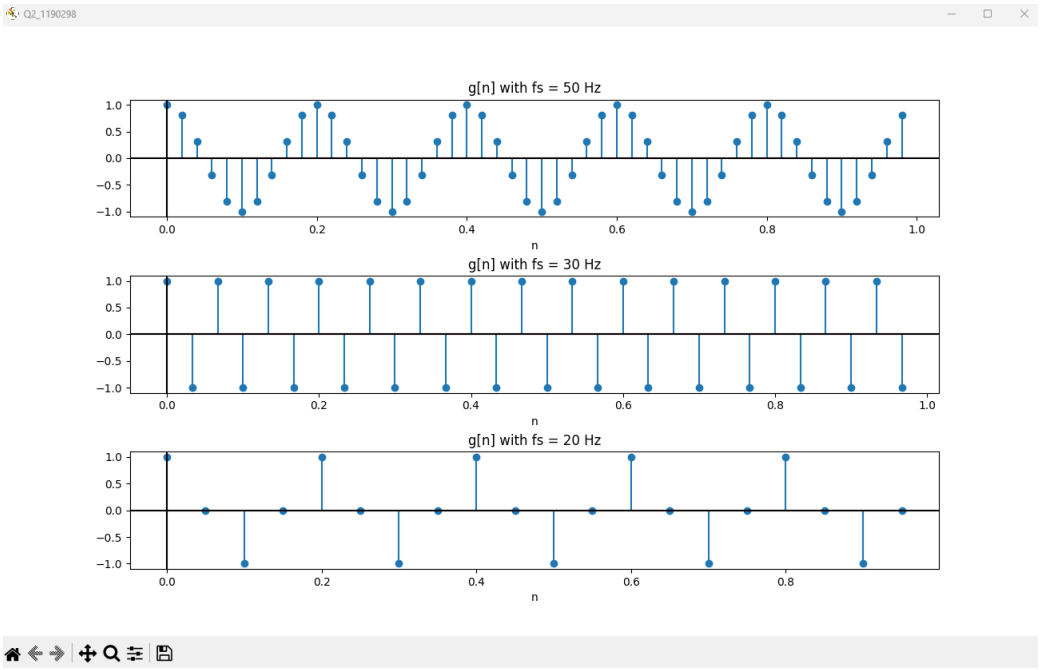
The output is a sequence of values, y[n], that represents the sum of a cosine signal and a Gaussian random sequence. The cosine signal has a frequency of 0.04π radians per sample, which corresponds to a period of 50/0.04 = 1250 samples. The amplitude of the cosine signal is 1, and it is shifted by a constant value of 0.2. The Gaussian random sequence, w(n), represents additive noise with zero mean and unit variance. This means that the values of the sequence will have an average value of 0 and a standard deviation of 1. The y[n] values will be a combination of the cosine signal and the random noise, where the cosine signal will be visible but with some random fluctuation due to the Gaussian noise.

C) z[n] = {…,5,4,3,2,1,5,4,3,2,1,5,4,3,2,1,…}, $-10 \leq n \leq 9$



We have found that the output is as expected.

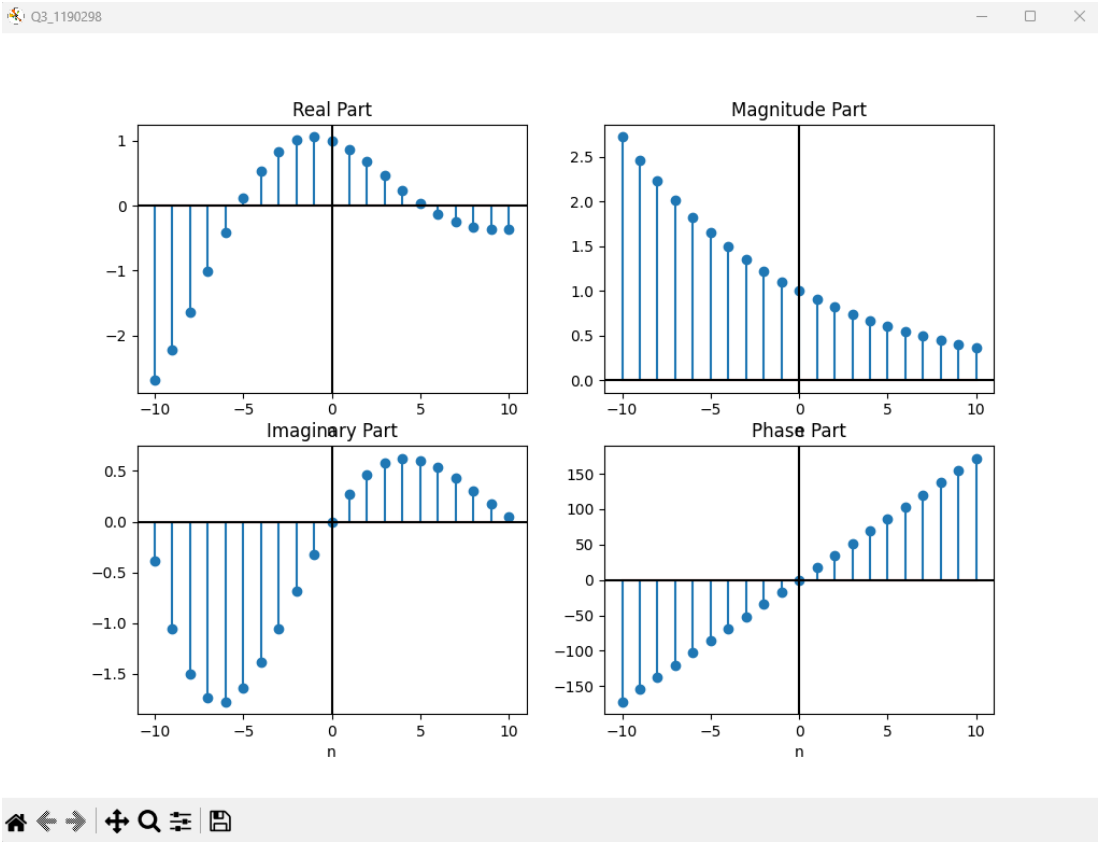
Q2. Generate and plot each of the following sequences over the indicated interval.



This code defines a function plotCosine() that plots a cosine function with different sampling frequencies. The function first defines the sampling frequencies fs1, fs2, fs3 and the frequencies f1 and f2 used in the cosine function. Then it calculates the cosine function for each sampling frequency using the numpy library's np.cos() and np.arange() functions. It also creates time axes for each sampling frequency using np.arange() and then plots the cosine function using plt.stem() and plt.subplot() functions from matplotlib library. It makes use of the stem plot to show the discrete signal. It shows 3 subplots, one for each sampling frequency and it uses matplotlib's plt.subplots_adjust() function to adjust the spacing between the subplots. The code will display 3 plots, one for each sampling frequency, showing the cosine function over a period of one second. Each plot will have a horizontal axis showing the time values, and a vertical axis showing the amplitude of the cosine function. It will also add horizontal and vertical lines to indicate the x and y axis. It would also have a title above each subplot with the corresponding sampling frequency and it will display the final plots using plt.show().

Q3. Generate the complex-valued signal.

$e^{(-0.1+j0.3)n}$, $-10 \leq n \leq 10$



The real part of the sequence is the cosine component of the sequence and is given by the equation Re{e^(-0.1+j0.3)n} which is similar to cos function. The imaginary part of the sequence is the sine component of the sequence and is given by the equation Im{e^(-0.1+j0.3)n} which is similar to sin function. To summarize, The plot of the real part will be a decaying cosine wave, the plot of the imaginary part will be a decaying sine wave, the plot of the magnitude will be a decaying exponential, and the plot of the phase will be a linearly increasing wave.

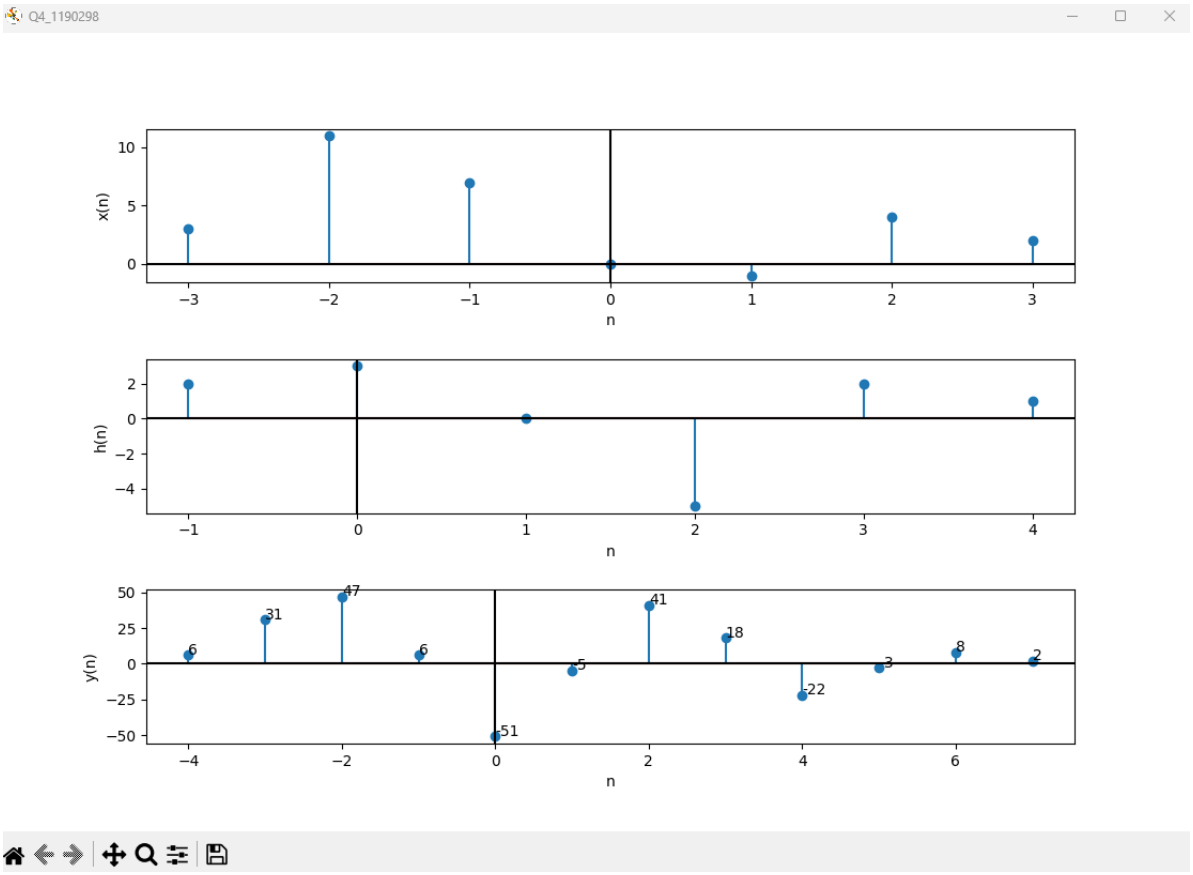
Part2 Convolution

Q4. Find the convolution of x[n] and h[n].

x[n] = [3, 11, 7, 0, -1, 4, 2], -3≤n≤3

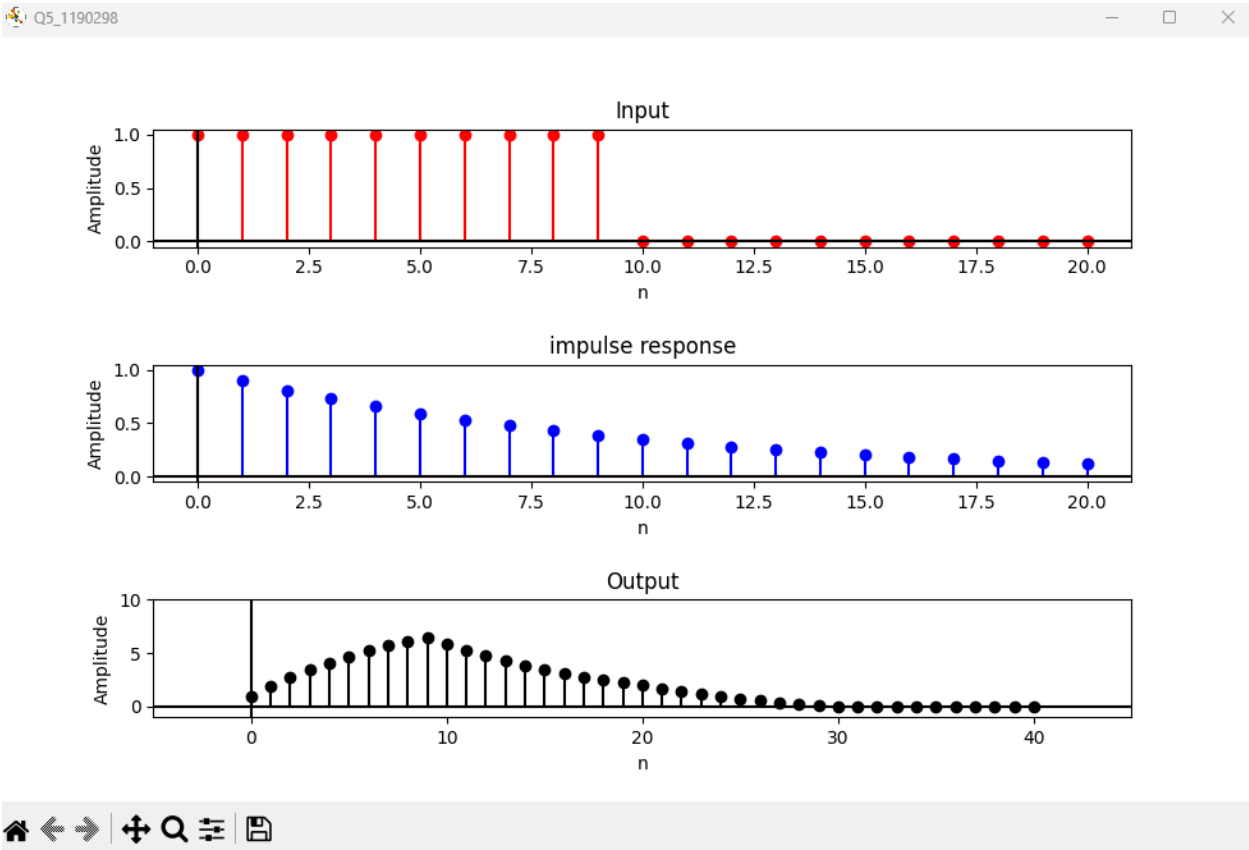
h[n] = [2, 3, 0, -5, 2, 1], -1≤n≤4

y[n] = [6 31 47   6 -51 -5 41 18 -22 -3   8   2], -4≤n≤7



n = [-3+-1,3+4] = [-4,7].

Q5. Let the rectangular pulse x(n) = u(n) − u(n − 10) be an input to an LTI system with impulse response $h[n] = (0.9)^n u(n)$, Find and plot y[n]

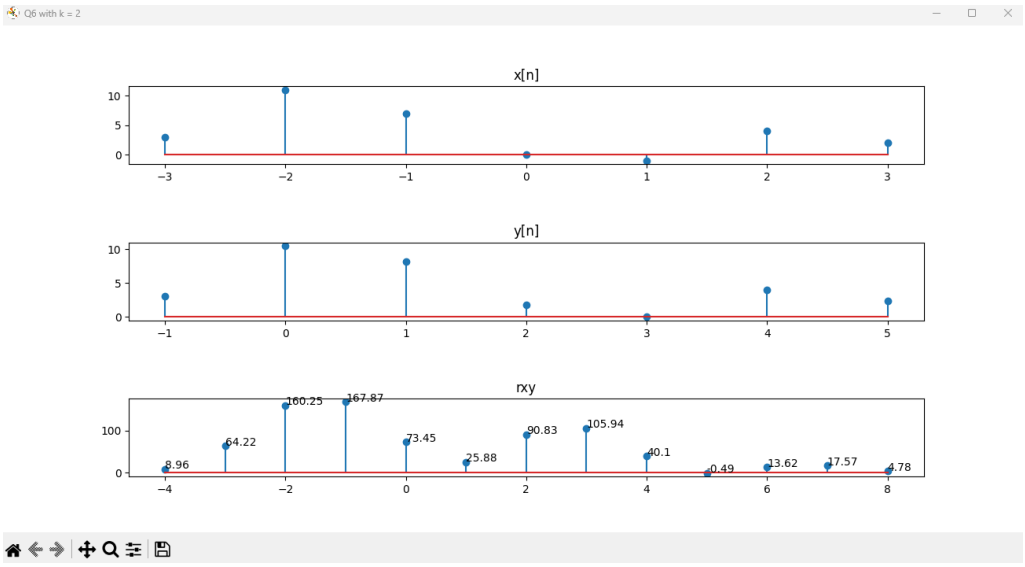


n = [0+0,20+20] = [0,40].

Part3 Correlations of sequences

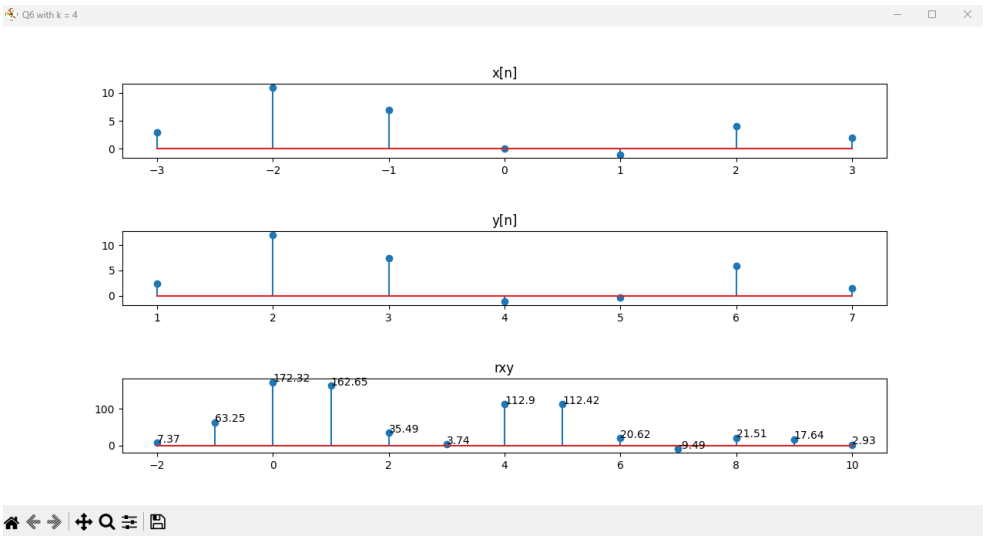
Q6. To demonstrate one application of the cross-correlation sequence.

A) y[n] = x[n-k] + w[n], k =2



It agrees with the expected output and the range is also as expected: [-3+-1,3+5] = [-4,8].

B) y[n] = x[n-k] + w[n], k =4

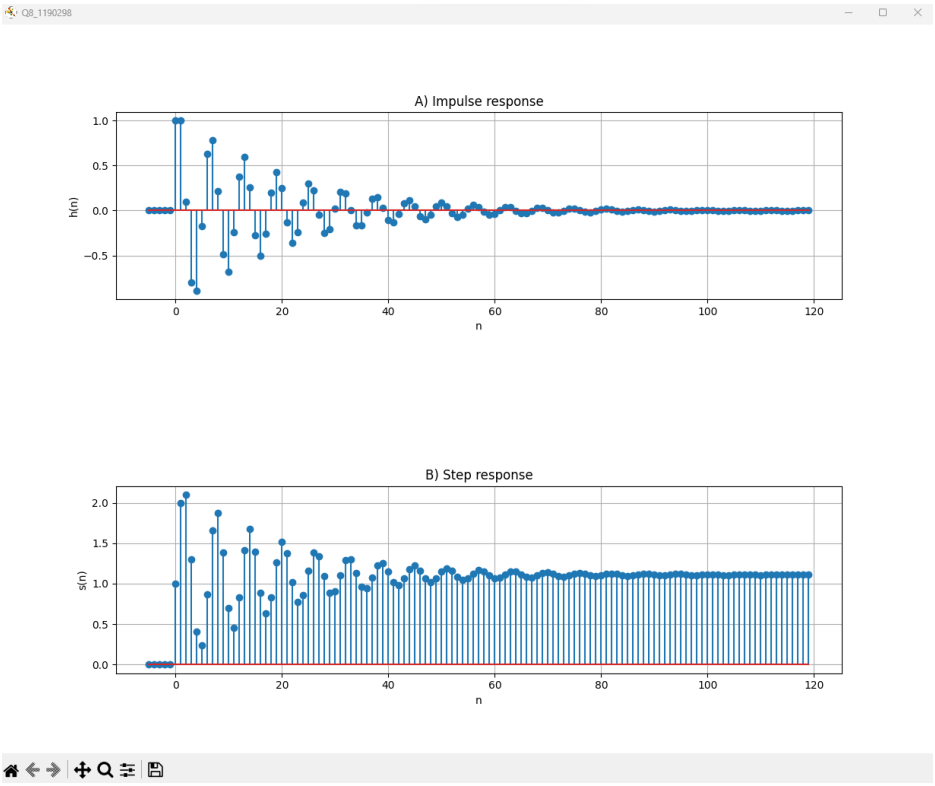


n =[-3+1,3+7] = [-2,10].

Part4 Difference Equation

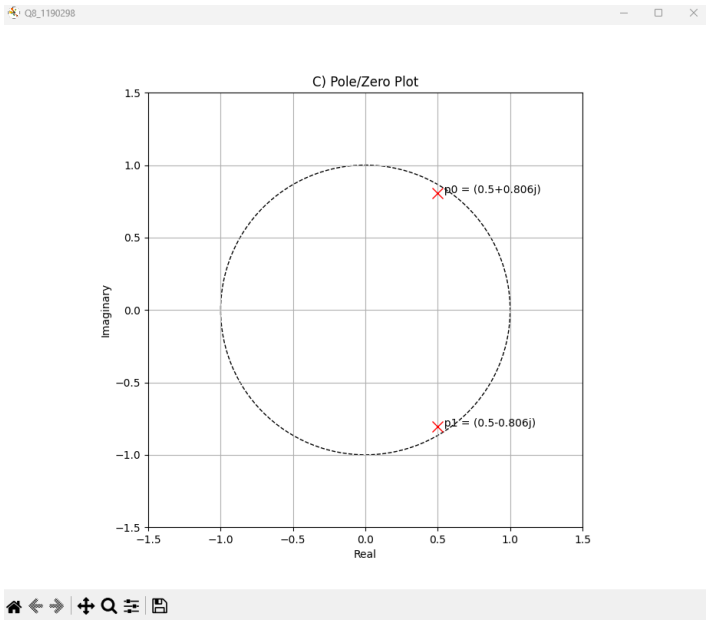
Q7. Given the following difference equation y[n] – y[n-1] +0.9y[n-2] = x[n]

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.



The first subplot shows the impulse response h(n) as a stem plot, with the time index n on the x-axis and the impulse response values on the y-axis. The second subplot shows the step response s(n) in a similar fashion, with the time index n on the x-axis and the step response values on the y-axis. It should be noted that the time range for both sequences is defined as n = -5:120.

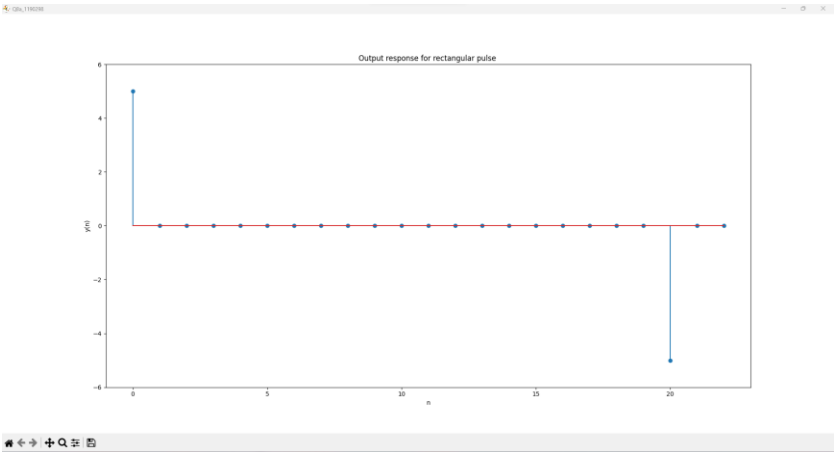
C) Is the system specified by h(n) stable?



Since the poles are inside the unit circle, the system is stable

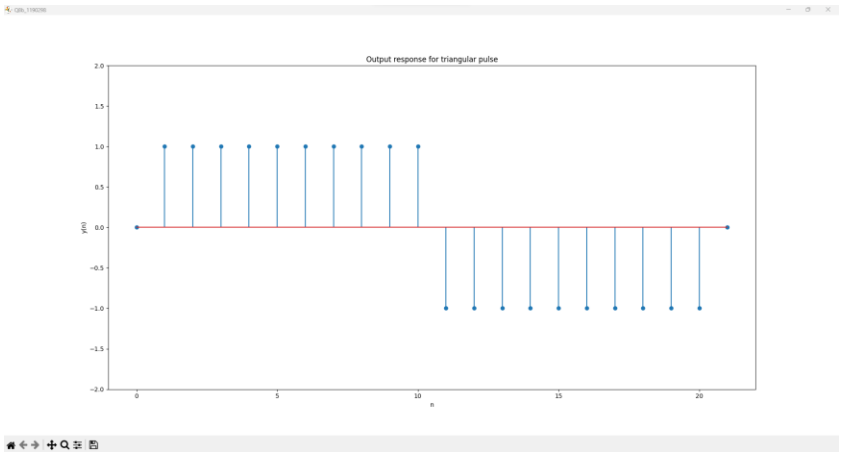
Q8. A “simple” digital differentiator is given by

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



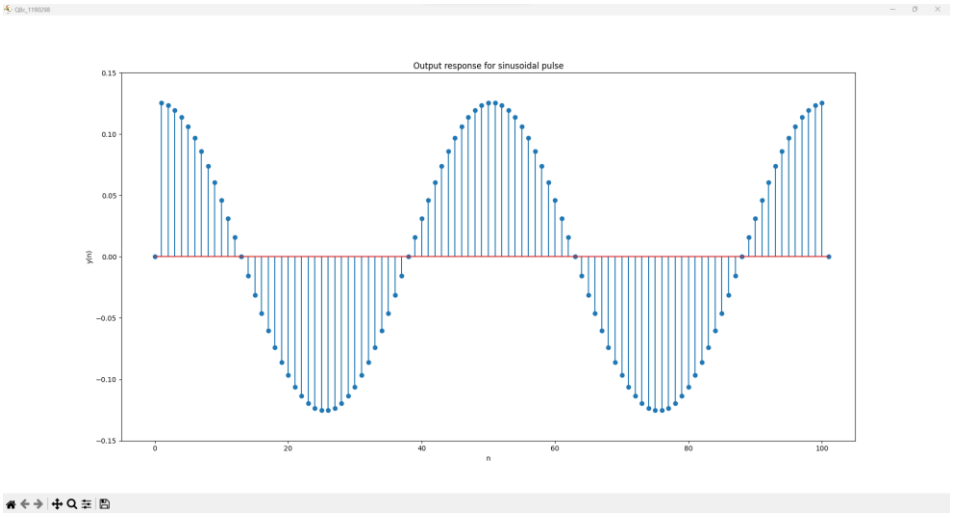
Which is the expected output.

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



Which also agree with the expected output.

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



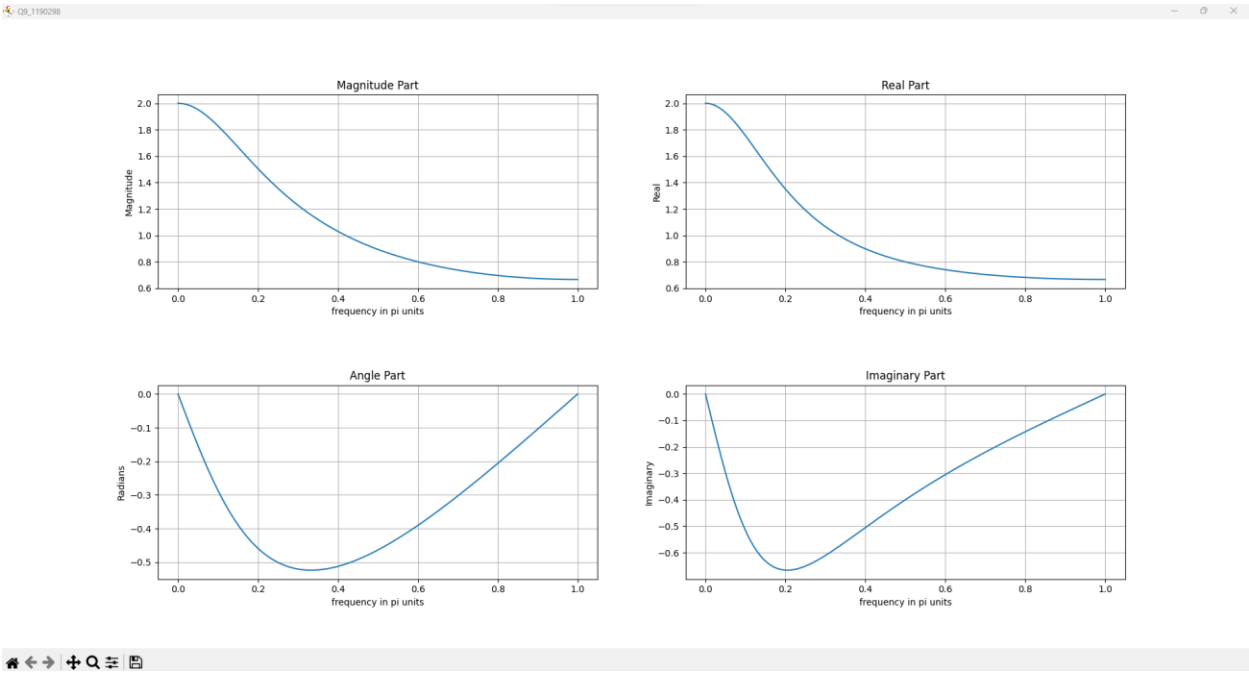
The amplitude of the sinusoidal pulse is modulated by the sin(πn/25) term, which varies between -0.1253 and 0.1253 over the range of n. The pulse starts at n=0 and lasts for 100 samples, which is determined by the u[n] and u[n-100] terms. The pulse is centered around n=50 (half of the pulse duration) and has a frequency of 1/50, which corresponds to 2π/50=π/25 radians.

In the output, we can see the sinusoidal pulse with amplitude modulated by the sine function. The value of the sinusoidal pulse is 1 for the values of n between 0 to 100, and 0 for all other values of n. We can also see that the amplitude of the sinusoidal pulse varies sinusoidally over the range of n.

Part4 DTFT

Q9. For $x[n] = (0.5)^n u[n]$. *The corresponding DTFT is* $x(e^{jw}) = \frac{e^{jw}}{e^{jw}-0.5}$

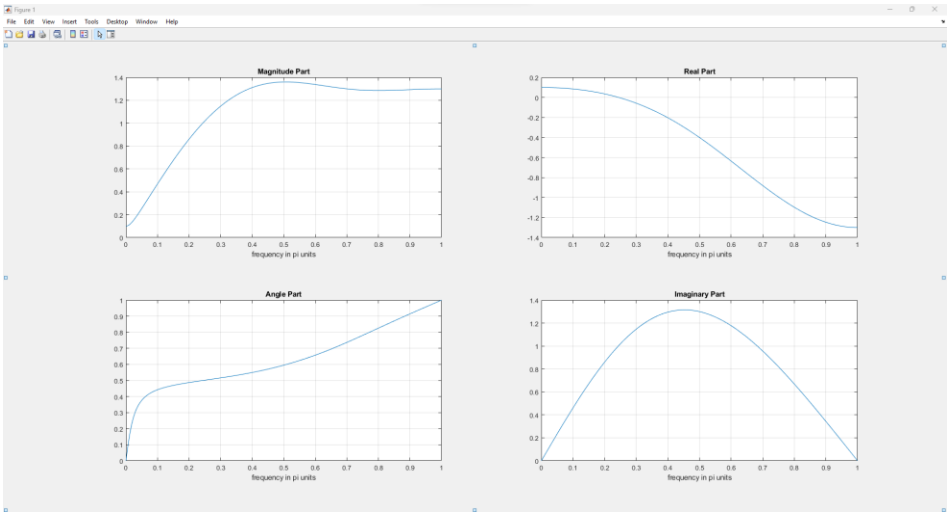
Evaluate at 501 equispaced points between [0,π] and plot its magnitude, angle, real, and imaginary parts.



The code above is generating a discrete-time Fourier transform (DTFT) of a specific function. It starts by defining an array of 501 evenly spaced values from 0 to pi, which will serve as the frequency index for the DTFT. Then it uses this array of frequencies to compute the DTFT of the function X using the given formula. The formula is X = $\frac{e^{jw}}{e^{jw}-0.5}$, where w is the array of frequencies and e^(j*w) is the complex exponential function. The function X is the transfer function of a discrete-time system. It then computes the magnitude, angle, real and imaginary parts of X using numpy functions. Finally, it plots the magnitude, angle, real and imaginary parts of X in four separate subplots. Each subplot has the frequency in pi units on the x-axis, and the value of the magnitude, angle, real and imaginary parts on the y-axis respectively.

Q10. 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,π].
B) plot its magnitude, angle, real, and imaginary parts



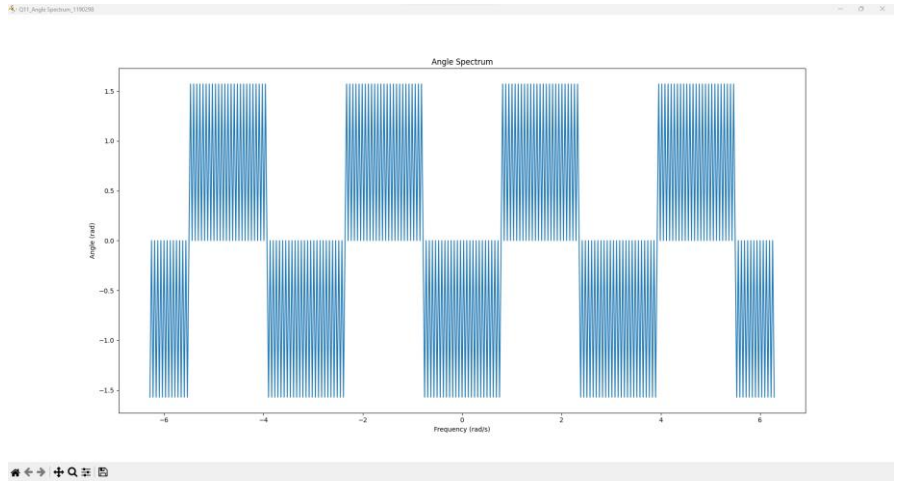
This figure defines a discrete-time signal x[n] with four samples, where n ranges from -1 to 2. Then it defines the frequency index for the Discrete-Time Fourier Transform (DTFT) of this signal, k, ranging from 0 to 500. The variable w is defined as the angular frequency in rad/s, scaled by a factor of pi/500.

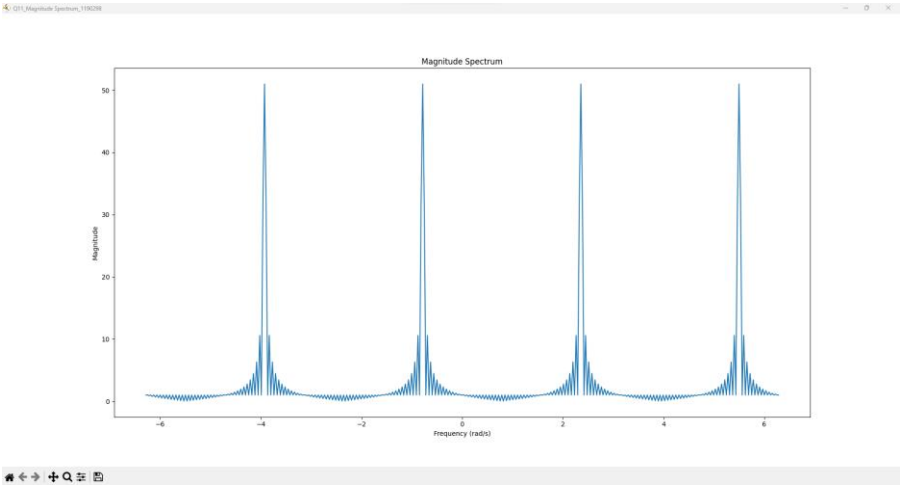
The DTFT of the input sequence x[n] is then computed by time-frequency scaling and stored in the variable X. The magnitude of the DTFT is computed and stored in the variable magX, the phase/angle of the DTFT is computed and stored in the variable angX. The real and imaginary parts of the DTFT are computed and stored in the variables realX and imagX respectively.

The code then creates four subplots using the subplot() function. The first subplot plots the magnitude of the DTFT in the k/500 vs magX domain. The second subplot plots the phase/angle of the DTFT in the k/500 vs angX/pi domain. The third subplot plots the real part of the DTFT in the k/500 vs realX domain. The fourth subplot plots the imaginary part of the DTFT in the k/500 vs imagX domain. Each subplot includes labels, grid, and title.

Q11. Let x[n] = cos(0.5πn), 0 ≤ n ≤ 100 and y[n] = $e^{jn\pi/4}\ x[n]$

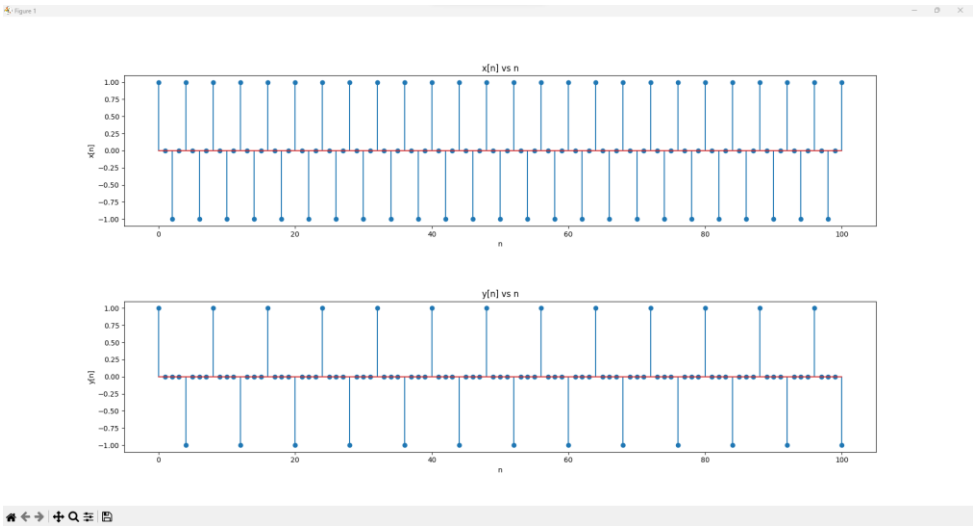
A) Numerically compute the discrete-time Fourier transform of at 401 equispaced frequencies between [-2π,2π].
B) Plot its magnitude, angle spectrum.





This question defines two sequences x[n] and y[n]. x[n] is a sequence of cosine values with an angular frequency of pi/2 and defined for the range of n = 0 to n = 100. y[n] is a sequence that is obtained by multiplying x[n] with a complex exponential sequence with an angular frequency of pi/4. The code in the appendex defines the Discrete Time Fourier Transform (DTFT) of y[n] by computing the sum of y[n] multiplied by the complex exponential sequence with different angular frequencies. The angular frequencies are defined in the k array, with the range of -2π to 2π with 401 points. The DTFT is then computed for each of these angular frequencies and stored in the Y array. Then the code plots the magnitude and angle spectrum of Y. The magnitude spectrum shows the magnitude of the complex numbers in the Y array, indicating the strength of different frequency components. The angle spectrum shows the phase of the complex numbers in the Y array, indicating the phase relationship between different frequency components.

C) Comment on the relation between x[n] and y[n].



The signal x[n] is defined as a cosine function with a frequency of π/2, while y[n] is a scaled and phase-shifted version of x[n]. The scaling factor is e^(jπn/4) which means that the magnitude of y[n] is same as x[n] but the phase has been shifted by π/4. In other words, y[n] and x[n] are related by a complex exponential multiplication, which shifts the phase of x[n] by π/4 and preserves its magnitude.

Conclusion

In this assignment, we practiced plotting several types of signals, including step sequences, impulse sequences, rectangular pulses, and sinusoidal pulses. We also looked at the concepts of convolution, correlation of sequences, and difference equations. Additionally, we explored the Discrete-Time Fourier Transform (DTFT) and computed the magnitude, angle, real, and imaginary parts of a signal. Overall, this assignment provided a comprehensive overview of several important concepts in signal processing and helped us gain a better understanding of how these concepts are applied in practice.

Appendix

Q1)

a-

```
import matplotlib.pyplot as plt
def plotImpulse(lst,lowerBound=0,upperBound=0):
    plt.figure("Q1a_1190298")
    if lowerBound==0 and upperBound==0:
        lowerBound=min(lst)
        upperBound=max(lst)
    minvalue=min(lst)
    maxvalue=max(lst)
    plt.stem(range(lowerBound,upperBound+1),lst)
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.axis([lowerBound-0.5,upperBound+0.5,minvalue-0.5,maxvalue+0.5])
    plt.grid(True)
    plt.legend(['x(n)'])
    plt.title('2δ(n+2) - δ(n-4)')
    plt.show()

plotImpulse([0,0,0,2,0,0,0,0,0,0,-1,0],-5,5)
```

b-

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

def plotCosine():
    plt.figure("Q1b_1190298")
    n = np.arange(0, 50, 1)
    x = np.cos(0.04*np.pi*n)+0.2*np.random.randn(len(n)) # x =
cos(0.04*pi*n)+0.2*randn(1,length(n))
    plt.stem(n,x)
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.axis([0,50,-1.5,1.5]) # plt.axis([xmin,xmax,ymin,ymax])
    plt.grid(True)
    plt.title("x(n) = cos(0.04*pi*n)+0.2*randn(1,length(n))")
    plt.show()
plotCosine()
```

c-

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

def plotXtilde():
    plt.figure("Q1c_1190298")
    n = np.arange(-10, 10, 1) # n = [-10,-9,...,9]
    x = [5,4,3,2,1]
    xtilde = np.tile(x,4) # xtilde = [5,4,3,2,1,5,4,3,2,1,5,4,3,2,1,5,4,3,2,1]
    plt.stem(n,xtilde)
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.axis([-10,9,0,6]) # plt.axis([xmin,xmax,ymin,ymax])
    plt.grid(True)
    plt.title('xtilde(n)')
    plt.show()
plotXtilde()
```

Q2)

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

# g(t) = cos(2*pi*f1*t) + 0.125 * cos(2*pi*2*f2*t)
# plot g[n]  for one second with sampling frequency fs

def plotCosine():
    # sampling frequencies
    fs1 = 50
    fs2 = 30
    fs3 = 20
    # Figure name
    plt.figure('Q2_1190298')
    # frequencies
    f1 = 5
    f2 = 15
    # cosine function with different sampling frequencies
    g1 = np.cos(2*np.pi*f1*np.arange(0, 1, 1/fs1))
    g2 = np.cos(2*np.pi*f2*np.arange(0, 1, 1/fs2))
    g3 = np.cos(2*np.pi*f2*np.arange(0, 1, 1/fs3))
    # time axis for each sampling frequency
    t1 = np.arange(0, 1, 1/fs1)
    t2 = np.arange(0, 1, 1/fs2)
    t3 = np.arange(0, 1, 1/fs3)
    # plot g[n] for each sampling frequency
    plt.subplot(3,1,1)
    plt.stem(t1,g1)
```

```
    plt.title('g[n] with fs = ' + str(fs1) + ' Hz')
    plt.xlabel('n')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.subplot(3,1,2)
    plt.stem(t2,g2)
    plt.title('g[n] with fs = ' + str(fs2) + ' Hz')
    plt.xlabel('n')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.subplot(3,1,3)
    plt.stem(t3,g3)
    plt.title('g[n] with fs = ' + str(fs3) + ' Hz')
    plt.xlabel('n')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    # edit spacing between subplots
    plt.subplots_adjust(hspace=0.5)
    plt.show()

plotCosine()
```

Q3)

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

def plotRealAndImaginaryPart():
    plt.figure('Q3_1190298')
    n = np.arange(-10, 11, 1)
    alpha = -0.1+0.3j
    x = np.exp(alpha*n)
    ######################################
    plt.subplot(2,2,1)
    plt.stem(n,np.real(x))
    plt.title('Real Part')
    plt.xlabel('n')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    ######################################
    plt.subplot(2,2,2)
    plt.stem(n,np.abs(x))
    plt.title('Magnitude Part')
    plt.xlabel('n')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    ######################################
    plt.subplot(2,2,3)
    plt.stem(n,np.imag(x))
```

```
    plt.title('Imaginary Part')
    plt.xlabel('n')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    ####################################
    plt.subplot(2,2,4)
    plt.stem(n,(180/np.pi)*np.angle(x)) # convert to degree
    plt.title('Phase Part')
    plt.xlabel('n')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    ####################################
    plt.show()

plotRealAndImaginaryPart()
```

Q4)

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

def findAndPlotConv(x,h):
    plt.figure('Q4_1190298')
    y = np.convolve(x,h)
    plt.subplot(3,1,1)
    plt.stem(range(-3,4),x) # x[n] = [3, 11, 7, 0, -1, 4, 2],  -3<=n<=3
    plt.ylabel('x(n)')
    plt.xlabel('n')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.subplot(3,1,2)
    plt.stem(range(-1,5),h) # h[n] = [2, 3, 0, -5, 2, 1],  -1<=n<=4
    plt.ylabel('h(n)')
    plt.xlabel('n')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.subplot(3,1,3)
    plt.stem(range(-4,8),y) # y[n] = [  6  31  47   6 -51  -5  41  18 -22  -
3   8   2],  -4<=n<=7
    plt.ylabel('y(n)')
    plt.xlabel('n')
    for i in range(len(y)):
        plt.text(i-4, y[i]+0.5, str(y[i]))
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    print(y)
    plt.subplots_adjust(hspace=0.5)
    plt.show()
```

```
x = [3, 11, 7, 0, -1, 4, 2]
h = [2, 3, 0, -5, 2, 1]
findAndPlotConv(x,h)
```

Q5)

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

def conv(x,nx,h,nh): # it will return the convolution of x and h
    nyb = nx[0] + nh[0] # nyb is the starting point of y
    nye = nx[-1] + nh[-1] # nye is the ending point of y
    ny = np.arange(nyb, nye+1) # ny is the range of y
    y = np.convolve(x,h) # y is the convolution of x and h
    return y, ny

def plotConv(x,nx,h,nh):
    plt.figure('Q5_1190298')
    y, ny = conv(x,nx,h,nh)
    plt.subplot(3,1,1)
    plt.stem(nx,x, 'r')
    plt.title('Input')
    plt.xlabel('n'),plt.ylabel('Amplitude')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')

    plt.subplot(3,1,2)
    plt.stem(nh,h, 'b')
    plt.title('impulse response')
    plt.xlabel('n'),plt.ylabel('Amplitude')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')

    plt.subplot(3,1,3)
    plt.stem(ny, y, 'k')
    plt.title('Output')
    plt.xlabel('n'),plt.ylabel('Amplitude')
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')

    plt.axis([-5, 45, -1, 10])
    plt.subplots_adjust(hspace=1)
    plt.show()

#x(n) = u(n) – u(n – 10)
x = np.array([1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0])
nx = np.arange(0,21)
h = 0.9**np.arange(0,21)
nh = np.arange(0,21)
```

```
plotConv(x,nx,h,nh)
```

Q6)

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

def sigshift(x,nx,k): # it will return the signal x[n-k]
    nx = nx + k
    return x,nx

def sigadd(y,ny,w , nw): # it will return the signal x[n] + w[n]
    #where w[n] is Gaussian sequence with mean 0 and variance 1
    #and x[n] is the signal x[n-k]
    y = y + w
    ny = nw
    return y,ny

def sigfold(x,nx): # it will return the signal x[-n]
    nx = -nx
    return x,nx

def conv(x,nx,h,nh): # it will return the convolution of x and h
    nyb = nx[0] + nh[0] # nyb is the starting point of y
    nye = nx[-1] + nh[-1] # nye is the ending point of y
    ny = np.arange(nyb, nye+1) # ny is the range of y
    y = np.convolve(x,h) # y is the convolution of x and h
    return y, ny


#[rxy,nrxy] = conv_m(y,ny,x,nx); % crosscorrelation
def crosscorrelation(x,nx,y,ny):
    y,ny = sigfold(y,ny) # obtain x[-n]
    rxy,nrxy = conv(x,nx,y,ny) # calculate the convolution of x and y
    return rxy,nrxy

def plotEverything(k,x,nx):
    plt.figure("Q6 with k = " + str(k))
    plt.subplot(3,1,1)
    plt.stem(range(-3,4),x)
    plt.title('x[n]')
    y,ny = sigshift(x,nx,k) # obtain x[n-k]
    w = np.random.randn(len(y))
    nw = ny # generate w[n]
    y,ny = sigadd(y,ny,w,nw) # obtain y[n] = x[n-k] + w[n]
    plt.subplot(3,1,2)
    plt.stem(range(-3 + k,4 + k),y) # the range is equal to the range of x plus k
    plt.title('y[n]')
    x,nx = sigfold(x,nx) # obtain x[-n]
```

```
    rxy,nrxy = crosscorrelation(x,nx,y,ny) # crosscorrelation
    plt.subplot(3,1,3)
    plt.stem(range(-6 + k,7 + k),rxy) # the range is equal to the sum of the
ranges of x and y
    for i in range(len(rxy)):
        plt.text(i-6 + k, rxy[i]+0.5, str(round((rxy[i]),2)))
    plt.title('rxy')
    plt.subplots_adjust(hspace=1)
    plt.show()

if __name__ == '__main__':
    x = [3, 11, 7, 0, -1, 4, 2];
    nx = np.arange(-3,4) # given signal x[n] nx = [-3 -2 -1  0  1  2  3]
    plotEverything(2,x,nx) # Q7.a: k = 2 ==> y[n] = x[n-2] + w[n]
    plotEverything(4,x,nx) # Q7.b: k = 3 ==> y[n] = x[n-3] + w[n]
```

Q7)

```
import numpy as np
import matplotlib.pyplot as plt
b = [1] # this is the coefficients of x[n]
a = [1, -1, 0.9] # these are the coefficients of y[n] to get y[n] = 0.9y[n-1] -
y[n-2]
n= np.arange(-5, 120, 1)

x = np.zeros(len(n))
for i in range(len(n)):
    if n[i] == 0:
        x[i] = 1

def filter(b, a, x):
    y = np.zeros(len(x))
    for i in range(len(x)):
        for j in range(len(b)):
            if i-j >= 0:
                y[i] += b[j]*x[i-j]
        for j in range(1, len(a)):
            if i-j >= 0:
                y[i] -= a[j]*y[i-j]
        y[i] /= a[0]
    return y

y = filter(b, a, x)
plt.figure("Q8_1190298")
plt.subplot(211)
plt.stem(n, y)
plt.grid()
plt.xlabel('n')
plt.ylabel('h(n)')
```

```
plt.title('A) Impulse response')

x = np.zeros(len(n))
for i in range(len(n)):
    if n[i] >= 0:
        x[i] = 1

y = filter(b, a, x)
plt.subplot(212)
plt.stem(n, y)
plt.grid()
plt.xlabel('n')
plt.ylabel('s(n)')
plt.title('B) Step response')
plt.subplots_adjust(hspace=1)
plt.show()

##############################################
# Zplane
##############################################
def zplane(b,a):
    plt.figure("Q8_1190298")
    ax = plt.subplot(111)
    unit_circle = plt.Circle((0,0), radius=1, fill=False, color='black',
ls='dashed')
    ax.add_patch(unit_circle)
    plt.axis('scaled')
    plt.axis([-1.5, 1.5, -1.5, 1.5])
    plt.grid(True)
    plt.xlabel('Real')
    plt.ylabel('Imaginary')
    # get the zeros and poles
    zeros = np.roots(b)
    print("zeros: ", zeros)
    poles = np.roots(a)
    print("poles: ", poles)
    # plot the zeros
    plt.plot(np.real(zeros), np.imag(zeros), 'go', ms=10)
    # plot the poles
    plt.plot(np.real(poles), np.imag(poles), 'rx', ms=10)
    # set the plot title
    plt.title('C) Pole/Zero Plot')
    # print the zeros and poles on the plot
    for n in range(len(zeros)):
        plt.text(np.real(zeros[n]), np.imag(zeros[n]), "  z{0}".format(n), '= ' +
str(round(zeros[n], 3)))
    for n in range(len(poles)):
        plt.text(np.real(poles[n]), np.imag(poles[n]), "  p{0}".format(n) + ' = '
+ str(round(poles[n], 3)))
```

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

zplane(b, a)
```

Q8)

a-

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

a = [1]
b = [1, -1]
n1 = np.arange(0, 23, 1)
def stepseq(n0,n1,n2):
    n = np.arange(n1,n2+1)
    x = np.zeros(len(n))
    nx = np.zeros(len(n))
    for i in range(len(n)):
        if n[i] >= n0:
            x[i] = 1
            nx[i] = n[i]
    return x,nx

x11,nx11 = stepseq(0,0,22) # x11 u[n-0]
x12,nx12 = stepseq(20,0,22) # x12 u[n-20]
x1 = 5*(x11 - x12) # x1 = 5*(u[n-0] - u[n-20])

def filter(b, a, x):
    y = np.zeros(len(x))
    for i in range(len(x)):
        for j in range(len(b)):
            if i-j >= 0:
                y[i] += b[j]*x[i-j]
        for j in range(1, len(a)):
            if i-j >= 0:
                y[i] -= a[j]*y[i-j]
        y[i] /= a[0]
    return y
y1 = filter(b,a,x1)

plt.figure("Q8a_1190298")
plt.stem(n1,y1)
plt.axis([-1,23,-6,6])
plt.xlabel('n')
plt.ylabel('y(n)')
plt.title('Output response for rectangular pulse ')
plt.show()
```

b-

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

a = [1]
b = [1, -1]
n2 = np.arange(0, 22, 1)
def stepseq(n0,n1,n2):
    n = np.arange(n1,n2+1)
    x = np.zeros(len(n))
    nx = np.zeros(len(n))
    for i in range(len(n)):
        if n[i] >= n0:
            x[i] = 1
            nx[i] = n[i]
    return x,nx
x11,nx11 = stepseq(0,0,21) # x11 u[n-0]
x12,nx12 = stepseq(10,0,21) # x12 u[n-10]
x13,nx13 = stepseq(20,0,21) # x13 u[n-20]
x2 = n2*(x11 - x12) + (20 - n2)*(x12 - x13) # x2 = n2*(u[n-0] - u[n-10]) +
(20 - n2)*(u[n-10] - u[n-20])

def filter(b, a, x):
    y = np.zeros(len(x))
    for i in range(len(x)):
        for j in range(len(b)):
            if i-j >= 0:
                y[i] += b[j]*x[i-j]
        for j in range(1, len(a)):
            if i-j >= 0:
                y[i] -= a[j]*y[i-j]
        y[i] /= a[0]
    return y

y2 = filter(b,a,x2)
plt.figure("Q8b_1190298")
plt.stem(n2,y2)
plt.axis([min(n2)-1,max(n2)+1,min(y2)-1,max(y2) + 1])
plt.xlabel('n')
plt.ylabel('y(n)')
plt.title('Output response for triangular pulse')
plt.show()
```

c-

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

a = [1]
b = [1, -1]
n3 = np.arange(0, 102, 1)

def stepseq(n0,n1,n2):
    n = np.arange(n1,n2+1)
    x = np.zeros(len(n))
    nx = np.zeros(len(n))
    for i in range(len(n)):
        if n[i] >= n0:
            x[i] = 1
            nx[i] = n[i]
    return x,nx

x11,nx11 = stepseq(0,0,101) # x11 u[n-0]
x12,nx12 = stepseq(100,0,101) # x12 u[n-100]
x13 = x11-x12
x3 = np.sin(np.pi*n3/25)*x13 # x3 = sin(pi*n/25)*(u[n-0] - u[n-100])

def filter(b, a, x):
    y = np.zeros(len(x))
    for i in range(len(x)):
        for j in range(len(b)):
            if i-j >= 0:
                y[i] += b[j]*x[i-j]
        for j in range(1, len(a)):
            if i-j >= 0:
                y[i] -= a[j]*y[i-j]
        y[i] /= a[0]
    return y

y3 = filter(b,a,x3)
plt.figure("Q8c_1190298")
plt.stem(n3,y3)
plt.axis([-5,105,-0.15,0.15])
plt.xlabel('n')
plt.ylabel('y(n)')
plt.title('Output response for sinusoidal pulse')
plt.show()
```

Q9)

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

# Generate an array of 501 evenly spaced values from 0 to pi
w = np.linspace(0, np.pi, 501)

# Compute X using the given formula
X = np.exp(1j*w) / (np.exp(1j*w) - 0.5*np.ones(501))

# Compute the magnitude, angle, real and imaginary parts of X
magX = np.abs(X)
angX = np.angle(X)
realX = np.real(X)
imagX = np.imag(X)

plt.figure("Q9_1190298")
# Plot the magnitude of X in the first subplot
plt.subplot(2, 2, 1)
plt.plot(w/np.pi, magX)
plt.grid()
plt.xlabel('frequency in pi units')
plt.title('Magnitude Part')
plt.ylabel('Magnitude')

# Plot the angle of X in the third subplot
plt.subplot(2, 2, 3)
plt.plot(w/np.pi, angX)
plt.grid()
plt.xlabel('frequency in pi units')
plt.title('Angle Part')
plt.ylabel('Radians')

# Plot the real part of X in the second subplot
plt.subplot(2, 2, 2)
plt.plot(w/np.pi, realX)
plt.grid()
plt.xlabel('frequency in pi units')
plt.title('Real Part')
plt.ylabel('Real')

# Plot the imaginary part of X in the fourth subplot
plt.subplot(2, 2, 4)
plt.plot(w/np.pi, imagX)
plt.grid()
plt.xlabel('frequency in pi units')
plt.title('Imaginary Part')
plt.ylabel('Imaginary')
```

```
plt.subplots_adjust(hspace=0.5)
plt.show()
```

Q10)

```
% Define the time index for the input sequence
n = -1:2;
% Define the input sequence
x = [1 -0.5 -0.3 -0.1];
% Define the frequency index for the DTFT
k = 0:500;
% Define the angular frequency in rad/s
w = (pi/500)*k;
% Compute the DTFT of the input sequence by time-frequency scaling
X = x * (exp(-j*pi/500)) .^ (n'*k);
% Compute the magnitude of the DTFT
magX = abs(X);
% Compute the phase/angle of the DTFT
angX = angle(X);
% Compute the real part of the DTFT
realX = real(X);
% Compute the imaginary part of the DTFT
imagX = imag(X);
% Create the first subplot for the magnitude of DTFT
subplot(2,2,1);
plot(k/500,magX);
grid on
xlabel('frequency in pi units');
title('Magnitude Part')
% Create the second subplot for the phase/angle of DTFT
subplot(2,2,3);
plot(k/500,angX/pi);
grid on
xlabel('frequency in pi units');
title('Angle Part')
% Create the third subplot for the real part of DTFT
subplot(2,2,2);
plot(k/500,realX);
grid on
xlabel('frequency in pi units');
title('Real Part')
% Create the fourth subplot for the imaginary part of DTFT
subplot(2,2,4);
plot(k/500,imagX);
grid on
xlabel('frequency in pi units');
title('Imaginary Part')
```

Q11)

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

# Sample rate
fs = 401

# Create array of n values
n = np.arange(101)

# Calculate x[n] and y[n]
x = np.cos((np.pi * n)/2)
y = np.exp((1j * np.pi * n) / 4) * x


# Calculate DTFT at 401 frequencies between -2*pi and 2*pi
frequencies = np.linspace(-2*np.pi, 2*np.pi, fs)
dtft = np.zeros(fs, dtype=complex)
for k in range(fs):
    dtft[k] = sum(y * np.exp(-1j * frequencies[k] * n))

# Calculate magnitude and angle spectrum
magnitude_spectrum = np.abs(dtft)
angle_spectrum = np.angle(dtft)

# Plot magnitude and angle spectrum
plt.figure("Q11_Magnitude Spectrum_1190298")
plt.plot(frequencies, magnitude_spectrum)
plt.xlabel('Frequency (rad/s)')
plt.ylabel('Magnitude')
plt.title('Magnitude Spectrum')

plt.figure("Q11_Angle Spectrum_1190298")
plt.plot(frequencies, angle_spectrum)
plt.xlabel('Frequency (rad/s)')
plt.ylabel('Angle (rad)')
plt.title('Angle Spectrum')

plt.show()
```