NAME - SANTOSH (LB.ETN.U4CLE20053)

ⓓ DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING

LAB TITLE AND CODE: SIGNAL PROCESSING LAB 19CLE281

EXPERIMENT NO: 3

DATE : 28/09/2021

OPERATION ON SIGNALS

* AIM:

Generate and visualize the basic set of signal operations using Python code.

* SOFTWARE REQUIRED:

Spyder IDE (Anaconda 3) - Python 3.9.7 (64-bit)

* THEORY:

The basic set of signal operations can be broadly classified as -

→ BASIC SIGNAL OPERATIONS PERFORMED ON DEPENDENT VARIABLES :-

In this transformation, only the quadrature axis values are modified i.e. magnitude of the signal changes, with no effects on the horizontal axis values or periodicity of signals. Let us look into these types in detail.

ⓐ Amplitude Scaling of Signals -

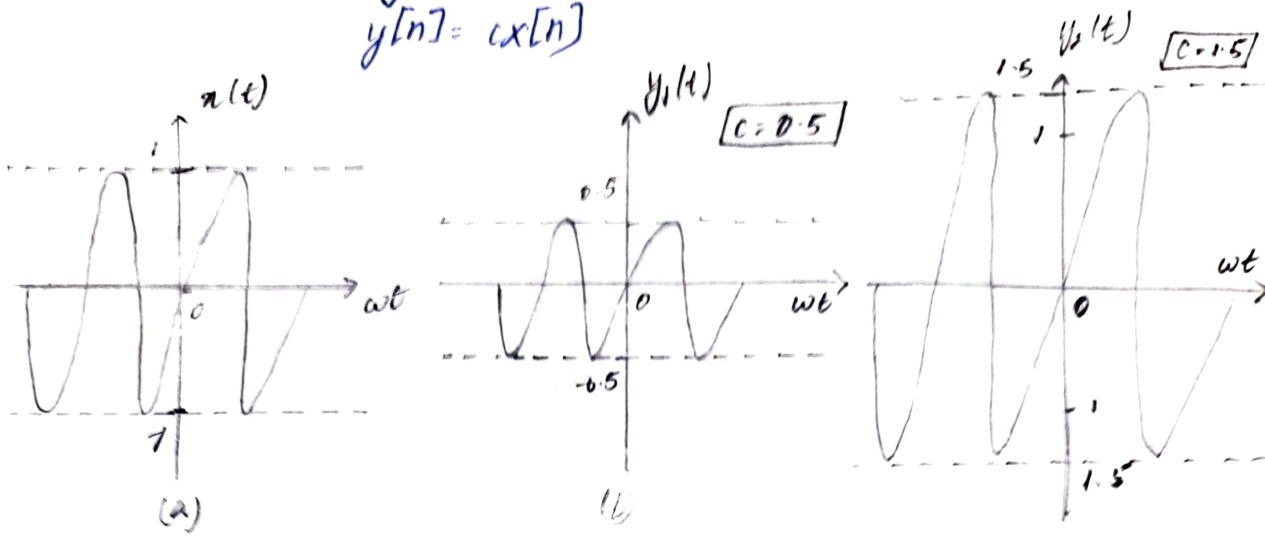
Amplitude scaling is a very basic operation performed on signals to vary its strength. It can be mathematically represented as -

$$y(t) = cx(t)$$

where $x(t)$ denotes continuous time signal,
$y(t)$ denotes resulting signal,
and c denotes the scaling factor.
If $c < 1$ → signal is attenuated.
$c > 1$ → signal is amplified.

For discrete time signals,

$$y[n] = cx[n]$$



This is illustrated in the diagram, where the signal is attenuated when $c = 0.5$ in figure (b) and amplified when $c = 1.5$ in figure (c).

Examples of amplitude scaling include amplifier, attenuator and resistor.

(3) Addition of signals -

This particular operation involves the addition of amplitude of two or more signals at each instance of time or any other independent variables which are common between the signals. Addition of signals is illustrated in the diagram below, where $x_1(t)$ and $x_2(t)$ are two time dependent signals, performing the additional operation on them, we get -

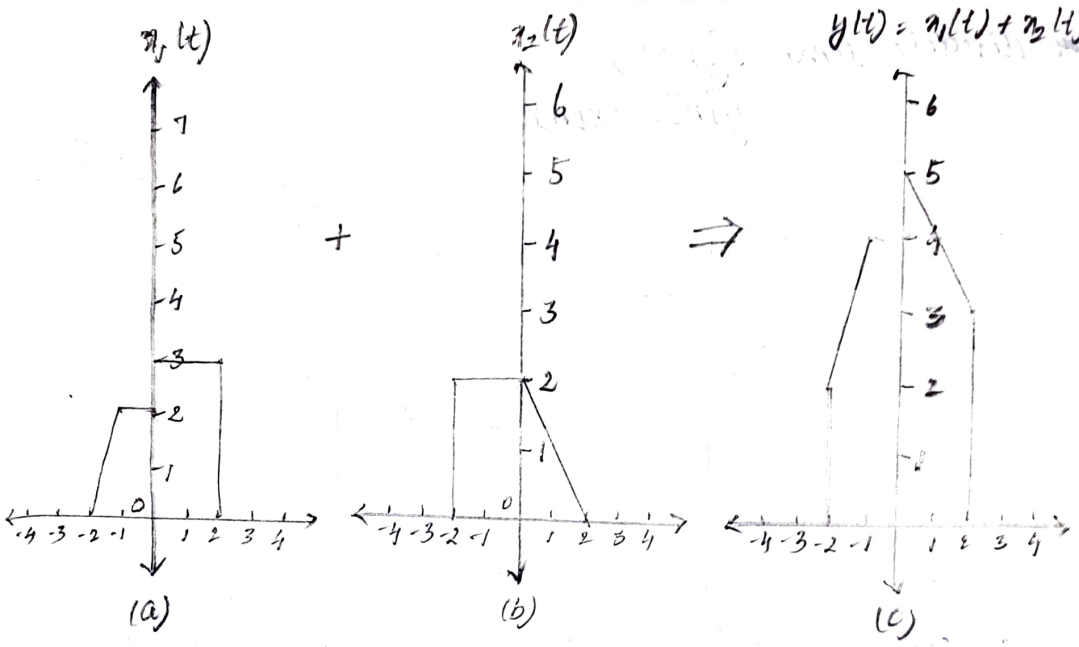
$$y(t) = x_1(t) + x_2(t)$$

For discrete time signals,

$$y[n] = x_1[n] + x_2[n]$$

A practical aspect in which signal addition plays its role is in the case of transmission of a signal through a communication channel. Other examples include dithering (Intentionally applied form of noise) and audio mixers.

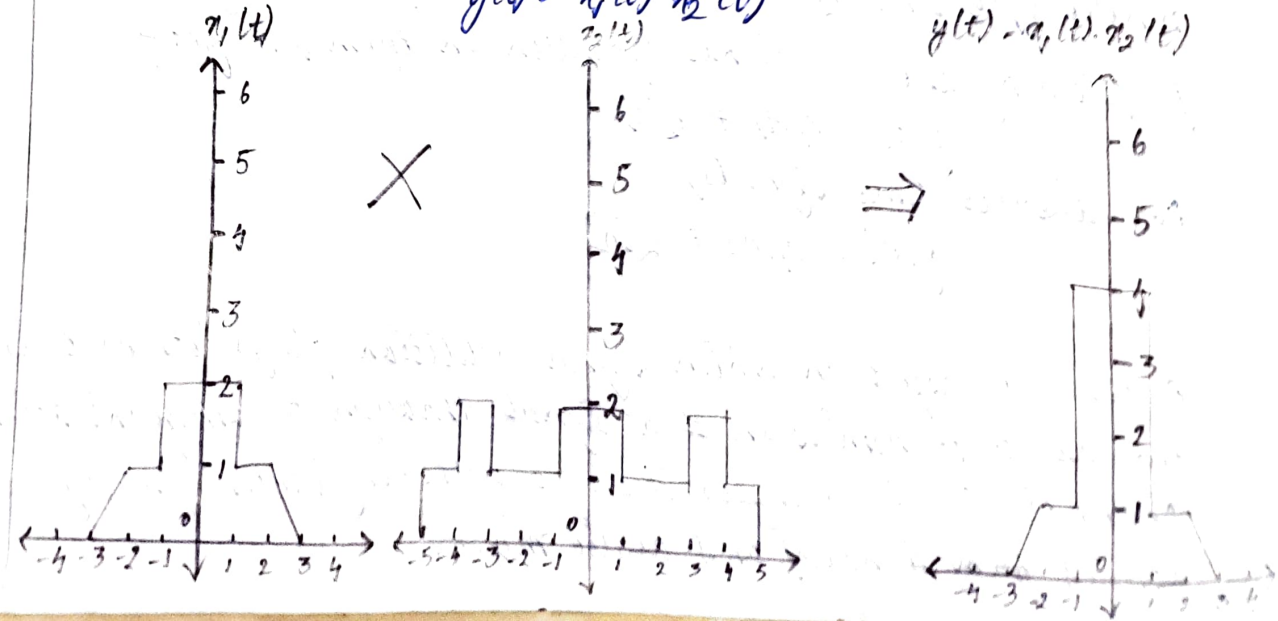
③



③ Multiplication of Signals -

Like addition, multiplication of signals also falls under the category of basic signal operations. Here multiplication of amplitude of two or more signals at each instance of time or any other independent variables is done which are common between the signals. The resultant signal we get has values equal to the product of amplitude of the parent signals for each instance of time. Multiplication of signals is illustrated in the diagram below, where $x_1(t)$ and $x_2(t)$ are two time independent signals, on whom after performing the multiplication operation we get -

$$y(t) = x_1(t)\, x_2(t)$$

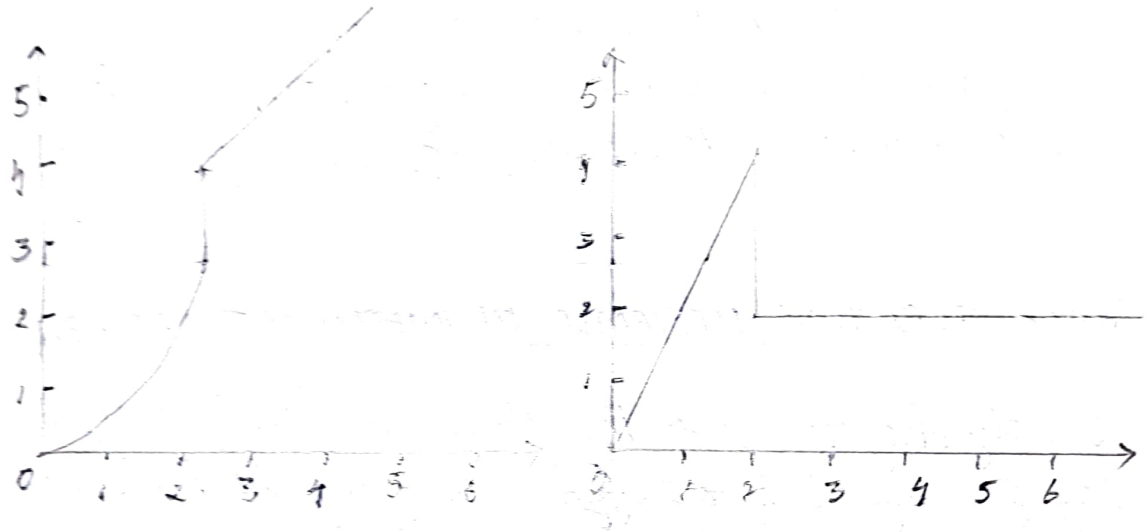


(4)

(6) Differentiation of Signals -

For differentiation of signals, it must be denoted that this operation is only applicable for continuous signals, as a discrete signal cannot be differentiated. The modified signal we get on differentiation has tangential values of the parent signal at all instance of time. Mathematically, it can be expressed as -

$$y(t) = \frac{d}{dt} x(t)$$

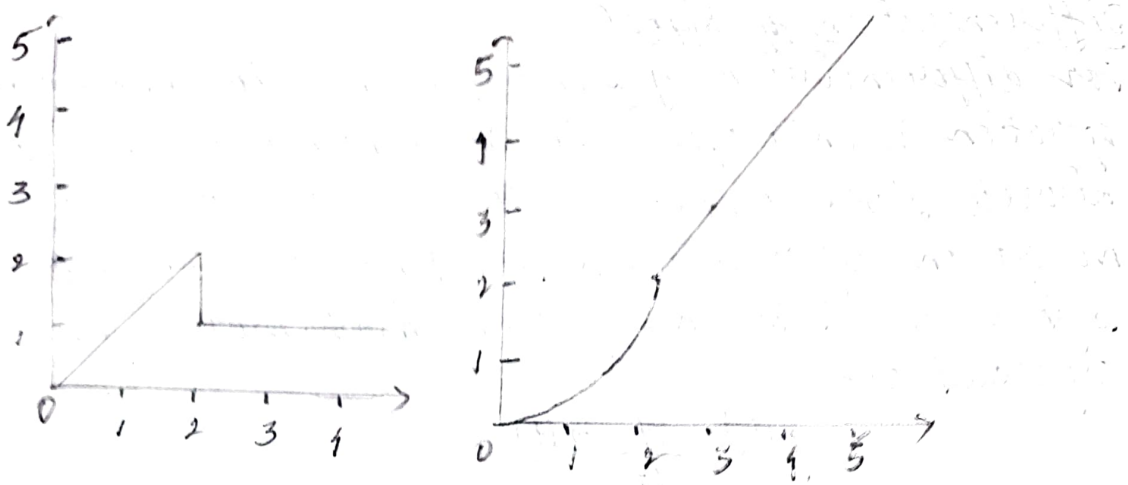


Differentiation of a signal takes the foum of the gradient operator in the field of image or video processing. In the case of image processing, the gradient technique is a popular method which is used to detect the edges in the given image. With video processing, this operator is used for motion detection. This kind of processing is important in the field of robotics.

(5) Integration of Signals -

Like differentiation, integration of signals is also applicable to only continuous time signals. The limits of integration will be from $-\infty$ to present instance of time t. It is mathematically expressed as -

$$y(t) = \int_{-\infty}^{t} x(\tau)\, d\tau$$



Integration is fundamental in signal-processing operations such as the Fourier transform, correlation, and convolution. These are, in turn, used to analyze different properties of a signal.

→ BASIC SIGNAL OPERATIONS PERFORMED ON INDEPENDENT VARIABLES :-

This is exactly the opposite of the above mentioned cases, here the periodicity of a signal is ~~modified~~ modifying the horizontal axis values, while the amplitude or the strength remains constant. Let us look into these operations in detail.

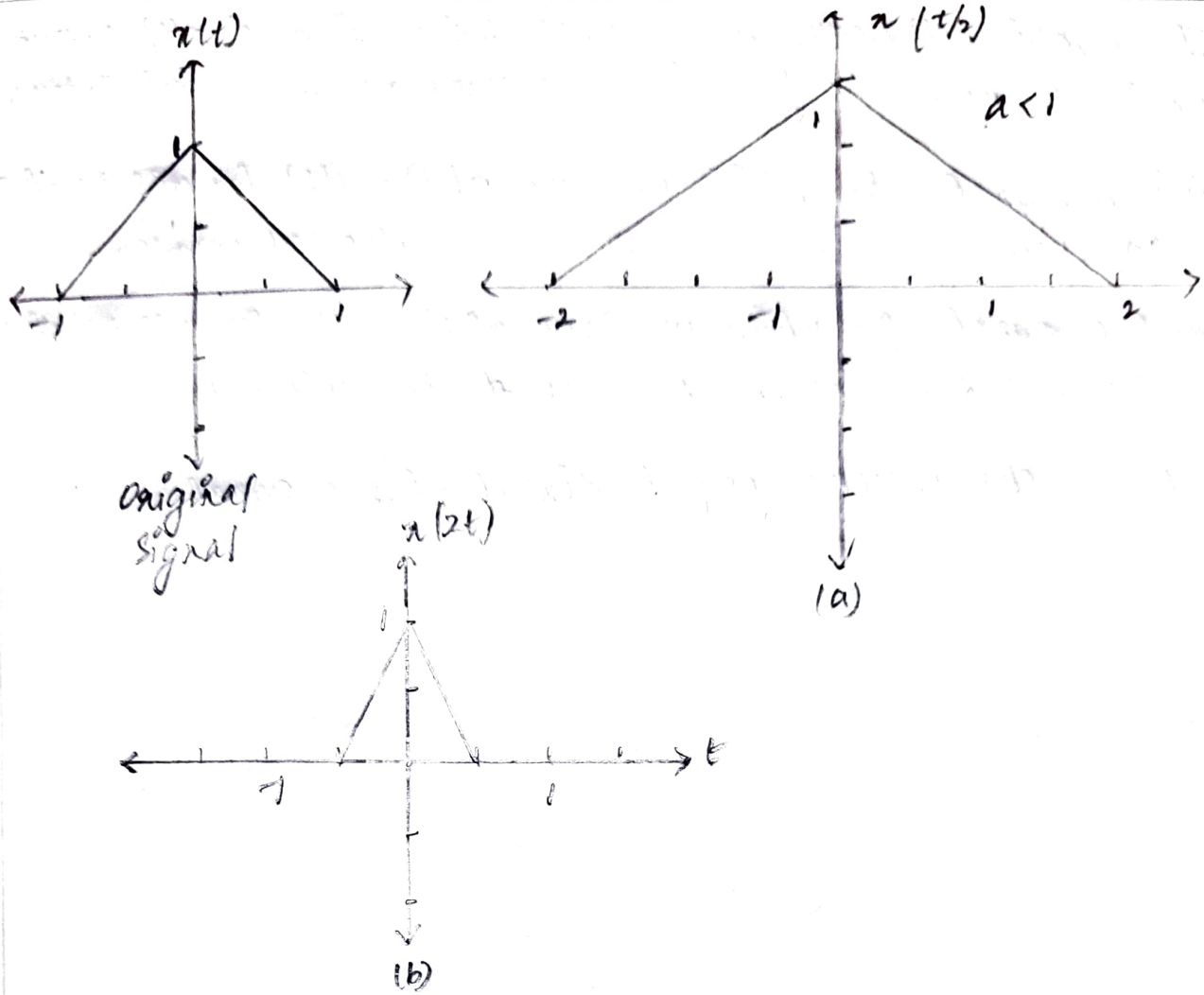
① Time scaling of Signals -

Time scaling of signals involves the modification of a periodicity of the signal, keeping its amplitude constant. Its mathematically expressed as,

$$y(t) = x(at)$$

If $a > 1$ implies the signal is compressed and $a < 1$ implies the signal is expanded. This is illustrated diagrammatically for better understanding.

PTO



Basically, when we perform time scaling, we change the rate at which the signal is sampled. Changing the sampling rate of a signal is employed in the field of speech processing. A particular example of this would be a time-scaling-algorithm-based system developed to read text to the visually impaired.

② Reflection of signals -

Reflection of signal is a very interesting operation applicable on both continuous and discrete signals. Here in this case, the vertical axis acts as the mirror, and the transformed image obtained is exactly the mirror image of the parent signal.

Let x(t) denote continuous time signal. Let y(t) denote the signal obtained by replacing time t with -t as shown by -

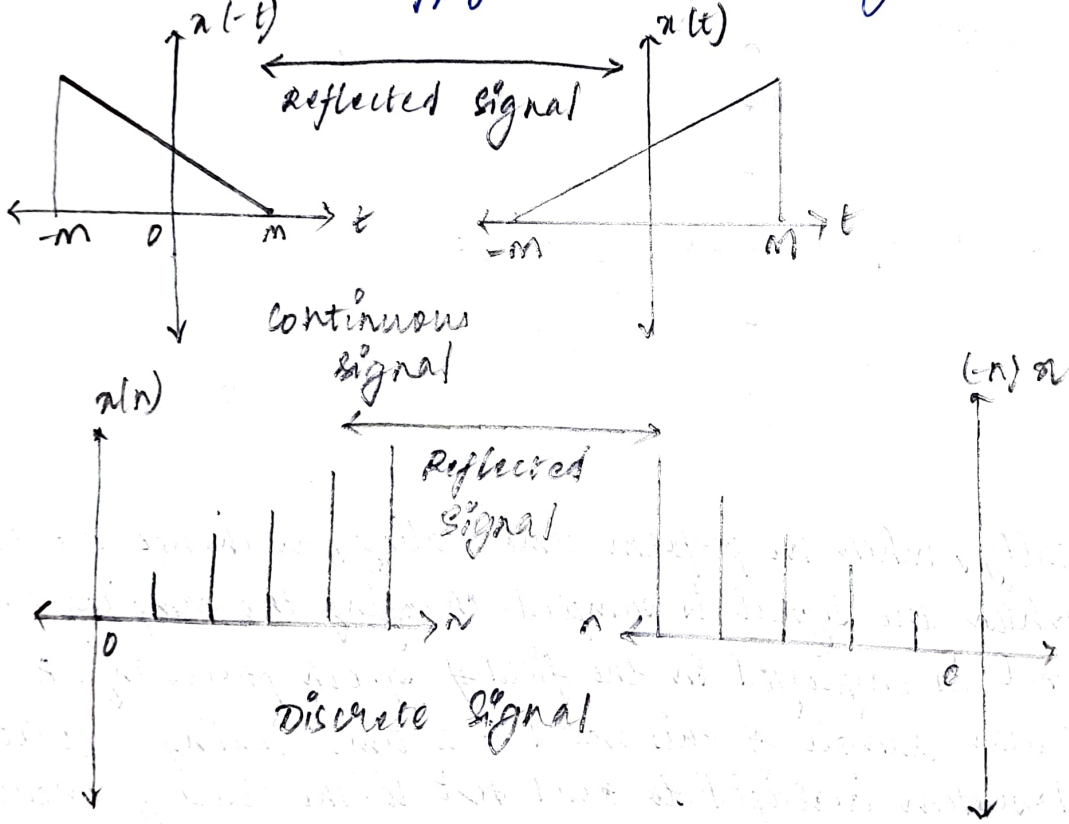
$$y(t) = x(-t)$$

The signal $y(t)$ represents a reflected version of $x(t)$ about the amplitude axis. The following two cases are of special interest –

(i) Even signals, for which we have $x(-t) = x(t)$ for all t; that is, an even signal is the same as its reflected version.

(ii) Odd signals, for which we have $x(-t) = -x(t)$ for all t; that is, an odd signal is the negative of its reflected version.

Similar observations apply to discrete-time signals.



Time reversal is an important preliminary step when computing the convolution of signals; one signal is kept in its original state while the other is mirror-imaged and slid along the former signal to obtain the result. Time-reversal operations, therefore, are useful in various image-processing procedures, such as edge detection.

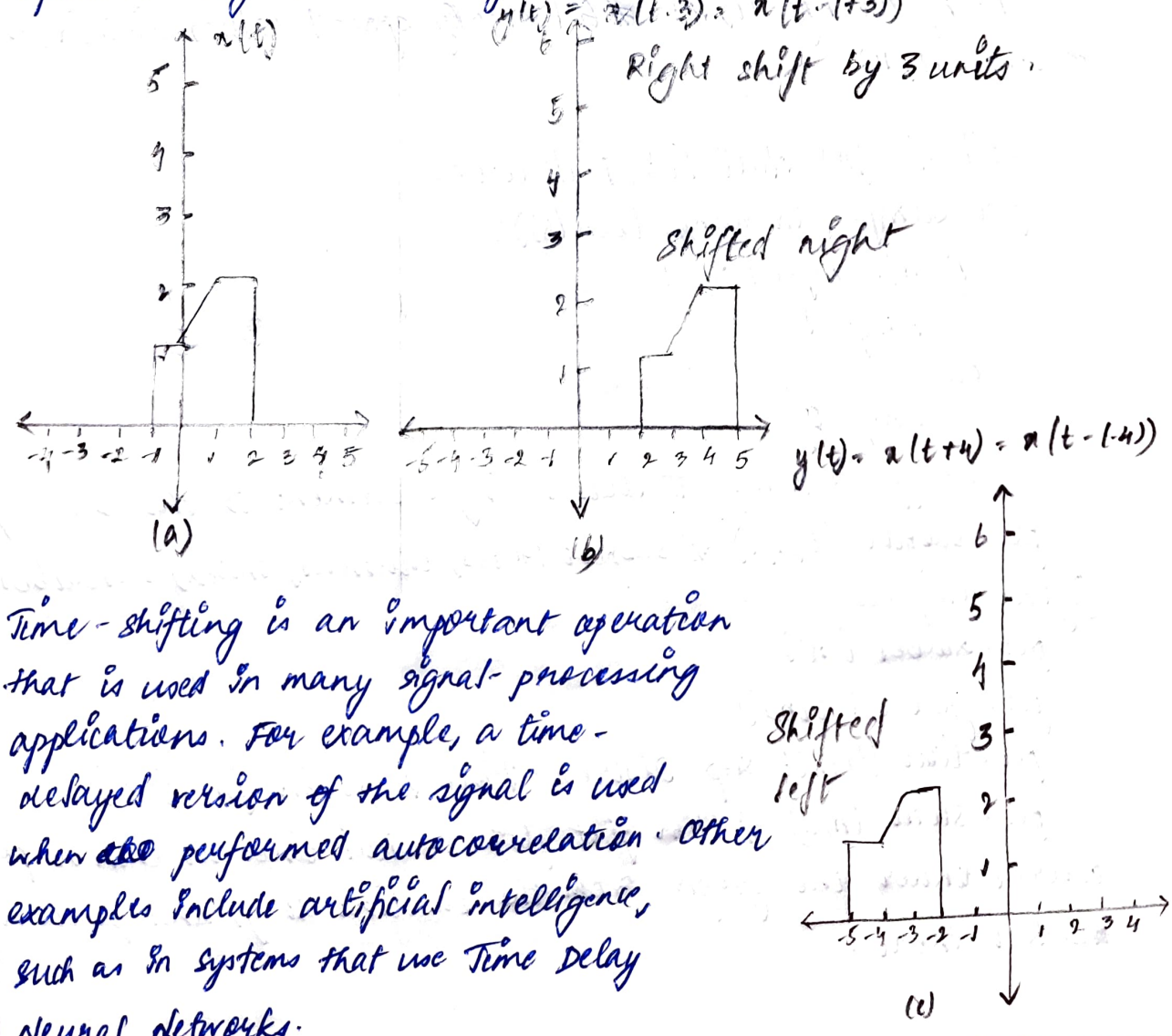
(9) **Time shifting of signals -**

Time shifting of signals is probably the most important one, and most widely used amongst all basic signal operations. Its generally used to fast-forward or delay a signal, as is necessary in most practical circumstances. Time shifting is mathematically expressed as,

$$y(t) = x(t - t_0)$$

where, $x(t)$ is the original signal,
and t_0 represents the shift in time.

For a signal $x(t)$, if the position shift $t_0 > 0$, then the signal is said to be right shifted or delayed. In the same manner, if $t_0 > 0$, implies a signal is left shifted or delayed. This has been explained diagrammatically below.



Time-shifting is an important operation that is used in many signal-processing applications. For example, a time-delayed version of the signal is used when performed autocorrelation. Other examples include artificial intelligence, such as in systems that use Time Delay Neural Networks.

* GRAPH PLOTTING ALGORITHM :

The following steps were followed -

1. Define the x-axis and corresponding y-axis values as lists.
2. Plot them on canvas using .plot() function.
3. Give a name to x-axis and y-axis using .xlabel() and .ylabel() functions.
4. Give a title to your plot using the .title() function.
5. Finally, to view your plot, we use the .show() function.

* PROGRAM WITH COMMENTS :

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

def step_sub # Step Signal Subtraction
    n = np.arange (0, 10) # Get evenly spaced values within the intervals [0, 10)

    a[] # u[n] Null List Declaration
    for sample in range (len(n)):
        if n[sample] >= 0
            temp = 1
        else :
            temp = 0
        a.append (temp) # Adds a single element to the existing list
    plt.subplot (5,1,1) # subplot (rows, columns, index) describes the figure layout
    plt.xlabel ('n')
    plt.ylabel ('u[n]')
    plt.title ("Unit Step Discrete-Time Function for u[n]')
    plt.stem (n,a) # Stem plot plots vertical lines at each x position covered under the graph from the baseline to y, and places a marker there.
```

```
b=[] # u[n-2] Null list Declaration
for sample in range (len(n)):
    if n[sample] >= 2:
        temp = 1
    else
        temp = 0
    b.append (temp) # Adds a single item to the existing list
plt.subplot (5, 1, 3) # subplot (rows, columns, index) describes the figure layout
plt.xlabel ('n')
plt.ylabel ('u[n-2]')
plt.title ('Unit Step Discrete-Time Function for u[n-2]')
plt.stem (n,b) # Stem plot plots vertical lines at each x position covered under the graph from the baseline to y, and places a marker there.

result = [] # u[n] - u[n-2] Null list Declaration
for sample in range (len(n)):
    result.append (a[sample] - b[sample]) # adds a single item to the existing list by subtracting u[n-2] from u[n] list.
plt.subplot (5, 1, 5) # subplot (rows, columns, index) describes the figure layout
plt.xlabel ('n')
plt.ylabel ('u[n] - u[n-2]')
plt.title ('Unit Step Discrete-Time Function for u[n] - u[n-2]')
plt.stem (n, result) # stem plot plots vertical lines at each x position covered under the graph from the baseline to y, and places a marker there.
plt.show () # to view all the three sub-plots.
```

PTO

```
def impulse_plot() # Plot Impulse signal
    n= np.arange(0,5) # Get evenly spaced values within the interval [0,5)
    arr = [2,3,4,5,0] # Initialize array with values as given in the question.
    plt.xlabel('n')
    plt.ylabel('x[n]')
    plt.stem(n,arr) # stem plot plots vertical lines at each n position covered under the graph from the baseline to y, and places a marker there.
    plt.show() # To view the plot.

def x_of_n(n): # Generate x[n] as given in the question
    list = [] # Null Array Declaration
    for sample in n: ⟵ if sample <=3
        if sample >= -3:
            list.append(abs(sample)) # abs() returns the absolute value of a number
        else:
            list.append(0) # Adds a single item to the existing list
    else:
        list.append(0) # Adds a single item to the existing list
    return(list) # send the value of list back and exits the function.

def y_of_n(n): # Generate y[n]
    list = [] # Null Array Declaration
    for sample in n:
        if sample == 0:
            list.append(0) # Adds a single item to the existing list.
        elif sample <= 4:
            if sample >= 4:
                list.append(sample/abs(sample)) # abs returns the absolute value of a number.
```

```
    else:
        list.append(0) # Adds a single item to the existing list.
    else:
        list.append(0) # Adds a single item to the existing list.
    return (list) # send the value of list back and exits the function

def sketch_i(): # sketch x[3n-1]
    n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7)
    plt.subplot(3,1,1) # subplot(rows, columns, index) describes the figure layout
    plt.xlabel('n')
    plt.ylabel('x[n]')
    plt.title('Function of x[n]')
    plt.stem(n, x_of_n(n)) # calls the function x_of_n(n) and displays x[n] plot.

    list = [] # Null List Declaration
    for sample in n:
        list.append(3*sample-1) # Adds a single item to the existing list, x[3n-1].

    plt.subplot(3,1,3) # subplot(rows, columns, index) describes the figure layout
    plt.xlabel('n')
    plt.ylabel('x[3n-1]')
    plt.title('Function for x[3n-1]')
    plt.stem(n, x_of_n(list)) # calls the function x_of_n(list) and displays x[3n-1] plot.
    plt.show() # To view both the sub-plots.
```

P.T.O →

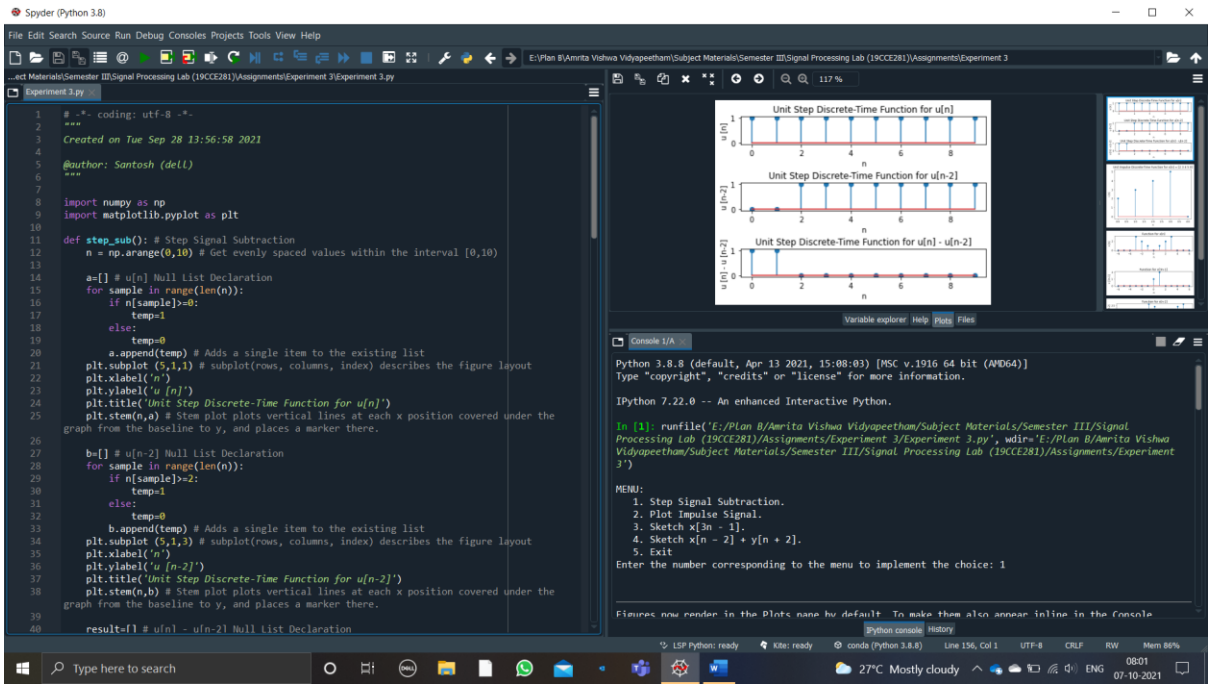
```
def sketch_2():  # sketch x[n-2] + y[n+2]
    n= np.arange(-6,7)  # Get evenly spaced values within the interval [-6, 7)
    list_x[]  # Null List Declaration
    for sample in n:
        list_x.append(sample-2)  # Adds a single item to the existing list, x[n-2]

    plt.subplot(5,1,1)  # subplot(rows, columns, index) describes the figure layout
    plt.xlabel('n')
    plt.ylabel('x[n-2]')
    plt.stem(n, x_of_n(list_x))  # Calls the function x_of_n(list_x) and displays x[n-2] plot.

    list_y=[]  # Null List Declaration
    for sample in n:
        list_y.append(sample+2)  # Adds a single item to the existing list, y[n+2]

    plt.subplot(5,1,3)  # subplot(rows, columns, index) describes the figure layout
    plt.xlabel('n')
    plt.ylabel('y[n+2]')
    plt.title('Function for y[n+2]')
    plt.stem(n,y_of_n(list_y))  # Calls the function y_of_n(list_y) and displays y[n+2] plot.

    result = np.add(x_of_n(list_x), y_of_n(list_y))  # Display the resultant graph, x[n-2] + y[n+2]
    plt.subplot(5,1,5)  # subplot(rows, columns, index) describes the figure layout
```

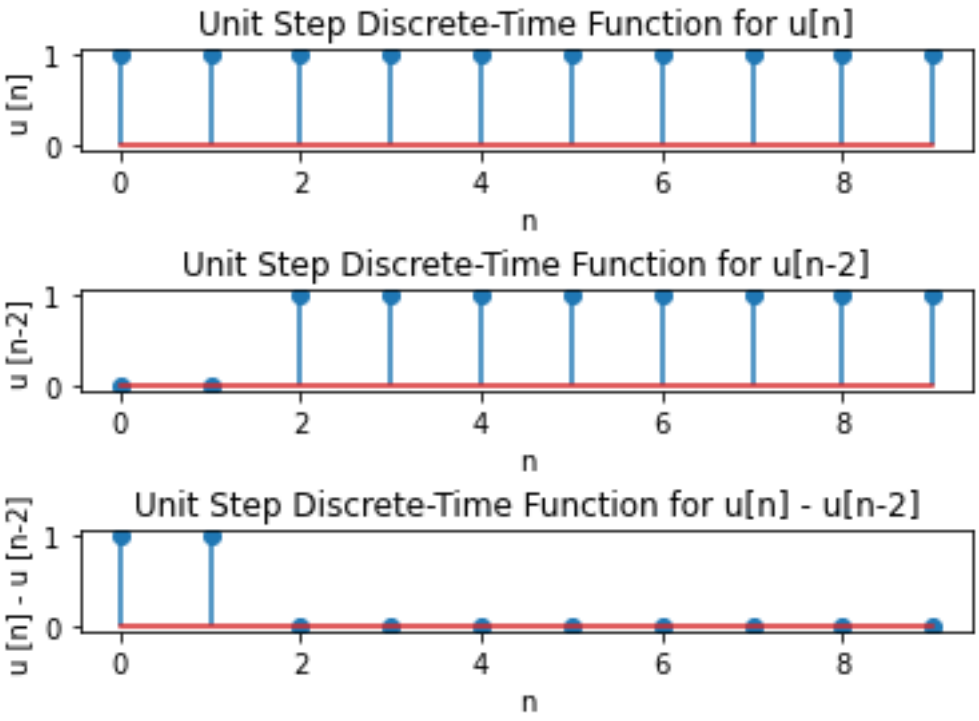
```
    plt.xlabel('n')
    plt.ylabel('x[n-2] + y[n+3]')
    plt.title('Function for x[n-2] + y[n+3]')
    plt.stem(n, result) # stem plot plots vertical lines at each n
position covered under the graph from the baseline to y, and
places a marker there.
    plt.show() # To view all the three sub-plots.

# Main
while True: # This simulates a Do Loop
    choice = input(
        "MENU: \n 1. Step Signal Subtraction \n 2. Plot Impulse
signal. \n 3. Sketch x[3n-1]. \n 4. Sketch x[n-2] + y[n+2]. \n
5. Exit \n Enter the number corresponding to the menu to
implement the choice: ") # Menu Driven Implementation

    if choice == str(1): # str() returns the string version of the
variable "choice"
        step_sub() # Step Signal subtraction
        break
    elif choice == str(2):
        Impulse_plot() # Plot Impulse signal
        break
    elif choice == str(3):
        sketch_1() # sketch x[3n-1]
        break
    elif choice == str(4):
        sketch_2() # sketch x[n-2] + y[n+2]
        break
    elif choice == str(5):
        break # Exit loop
    else:
        print("Error! Invalid Input! Please try again. \n")
```

* **INFERENCE:**

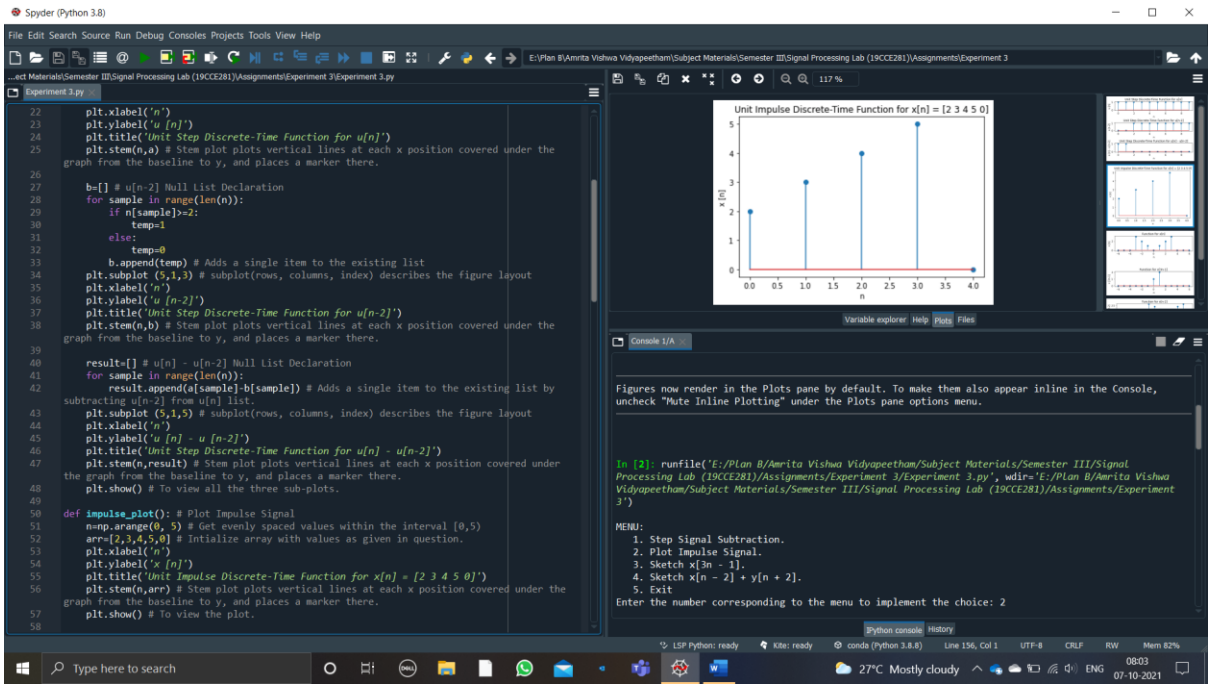
performed the basic set of signal processing operations using Python code and results verified.

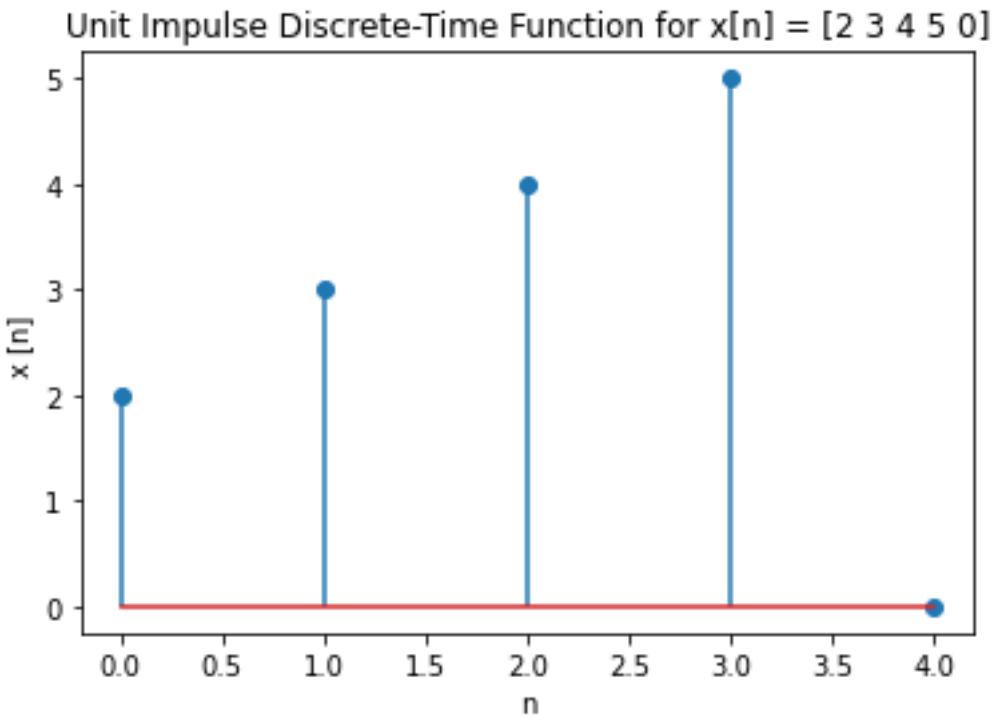
Step Signal Subtraction



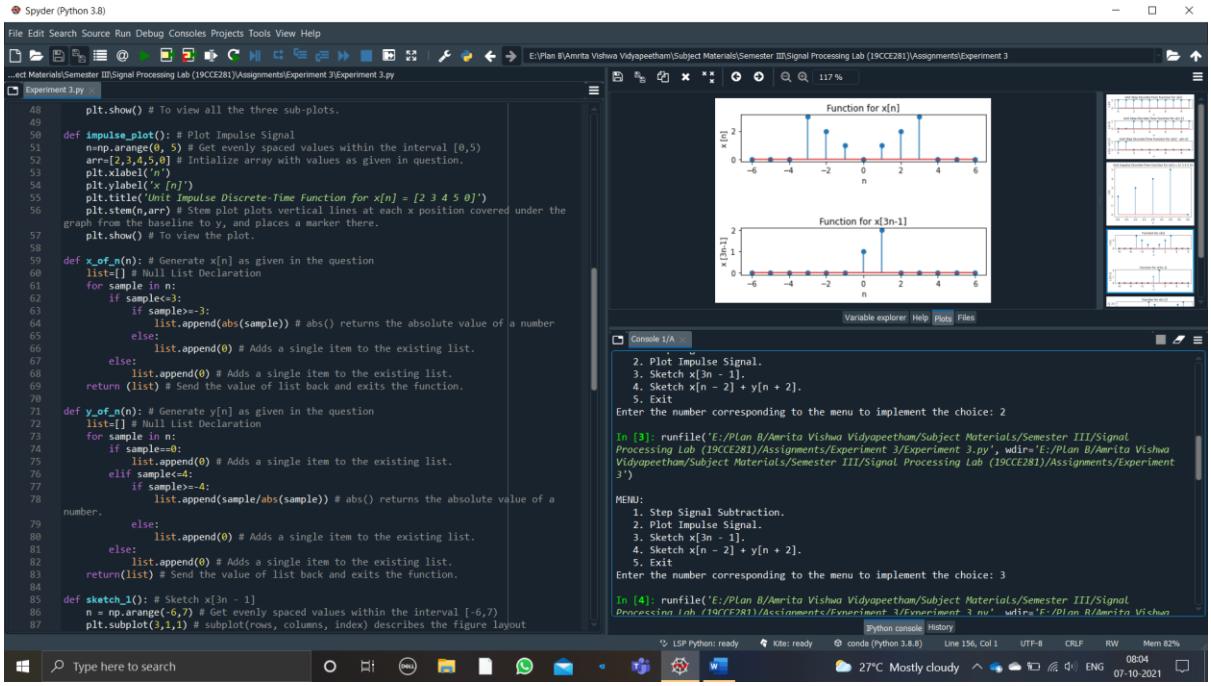


Plot Impulse Signal

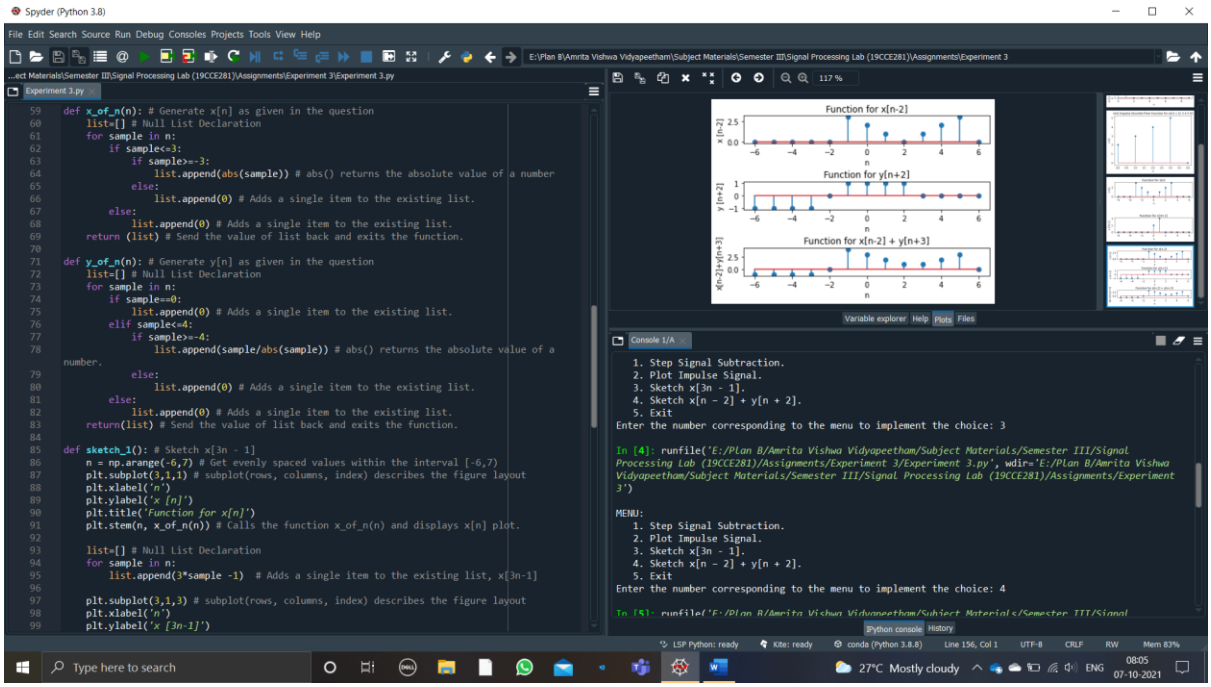




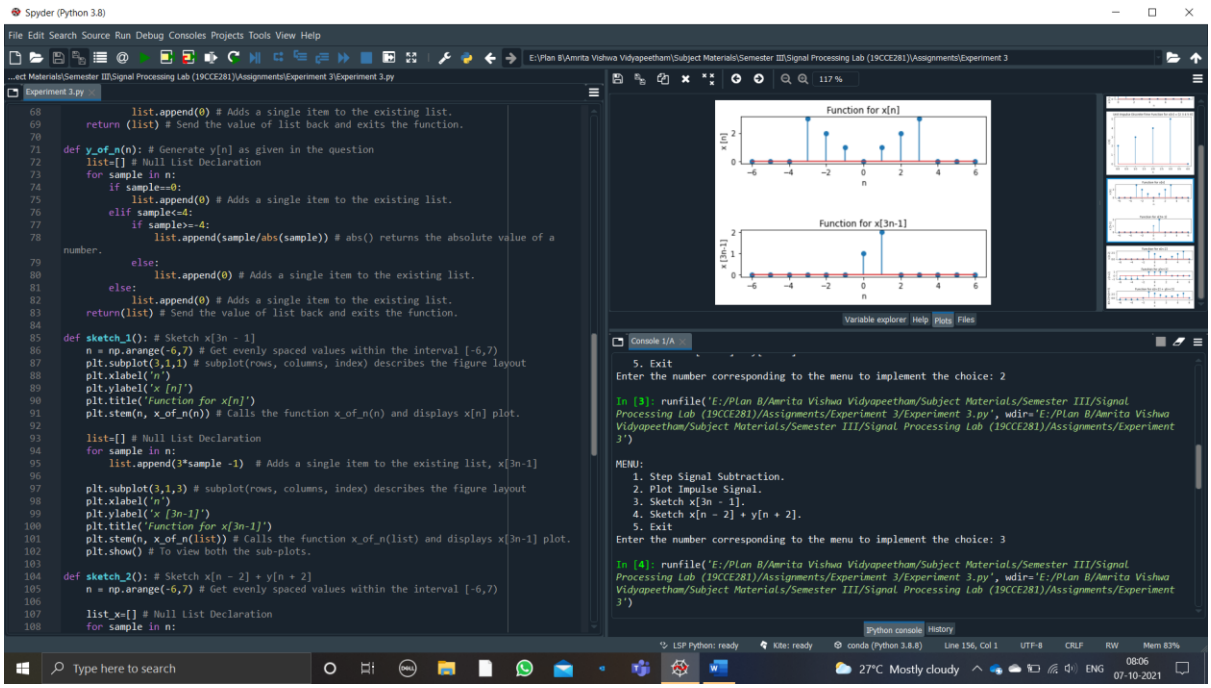
Generate x[n] as given in the question

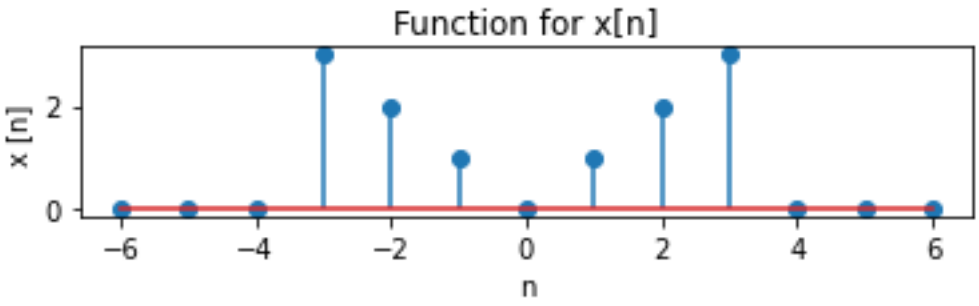


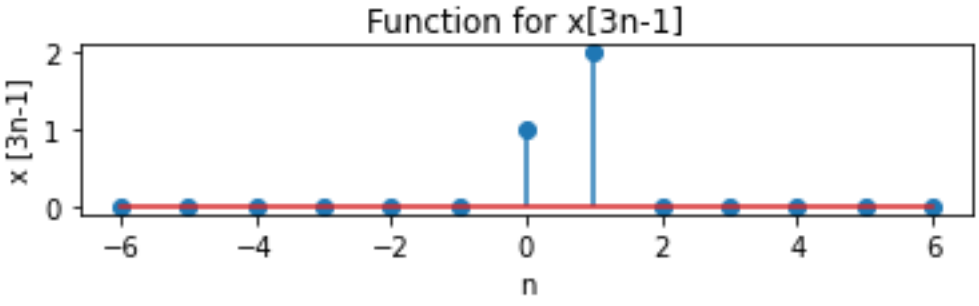
Generate y[n] as given in the question



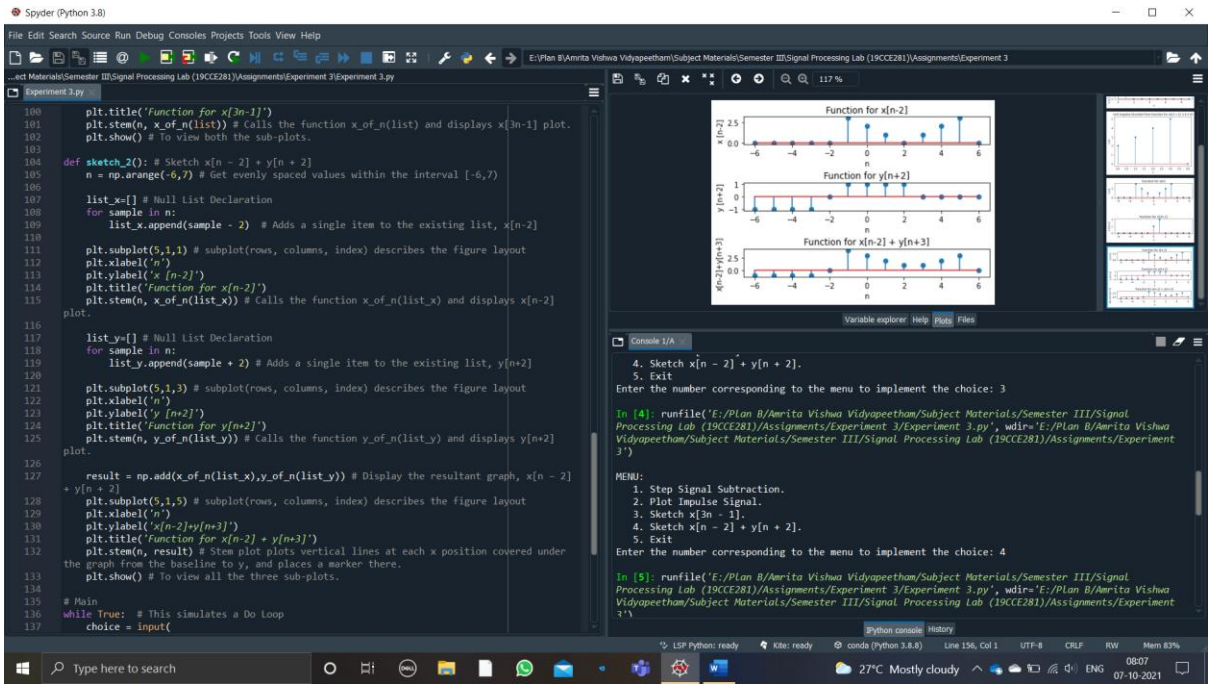
Sketch x[3n - 1]

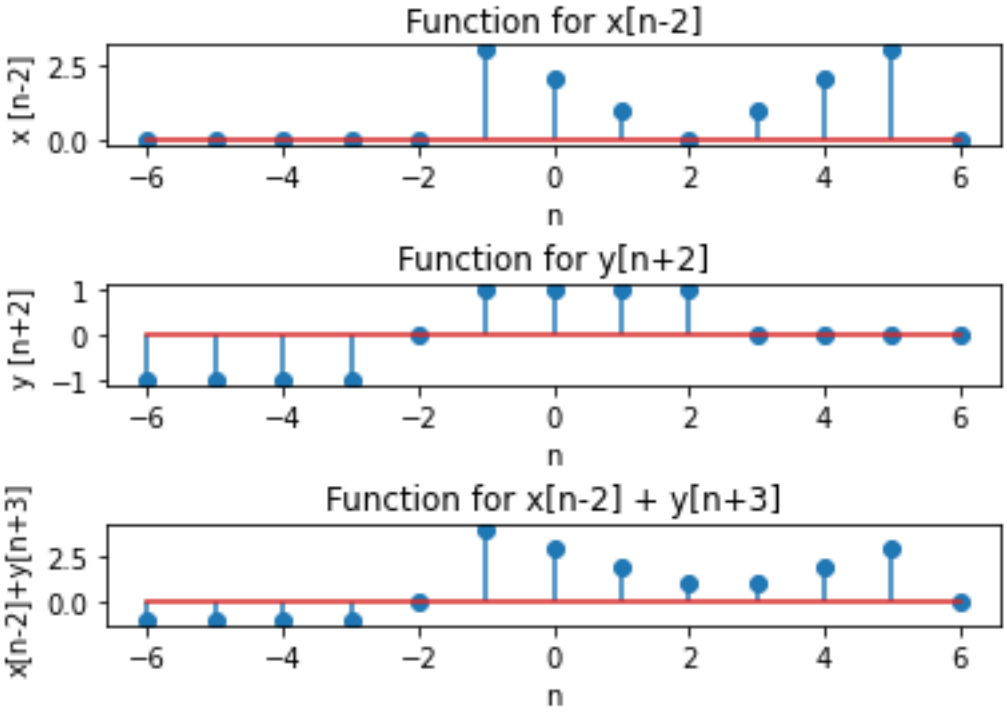

Spyder (Python 3.8)
Function for x[n]
Function for x[3n-1]
Enter the number corresponding to the menu to implement the choice: 2
Enter the number corresponding to the menu to implement the choice: 3


Function for x[n]
x [n]
n

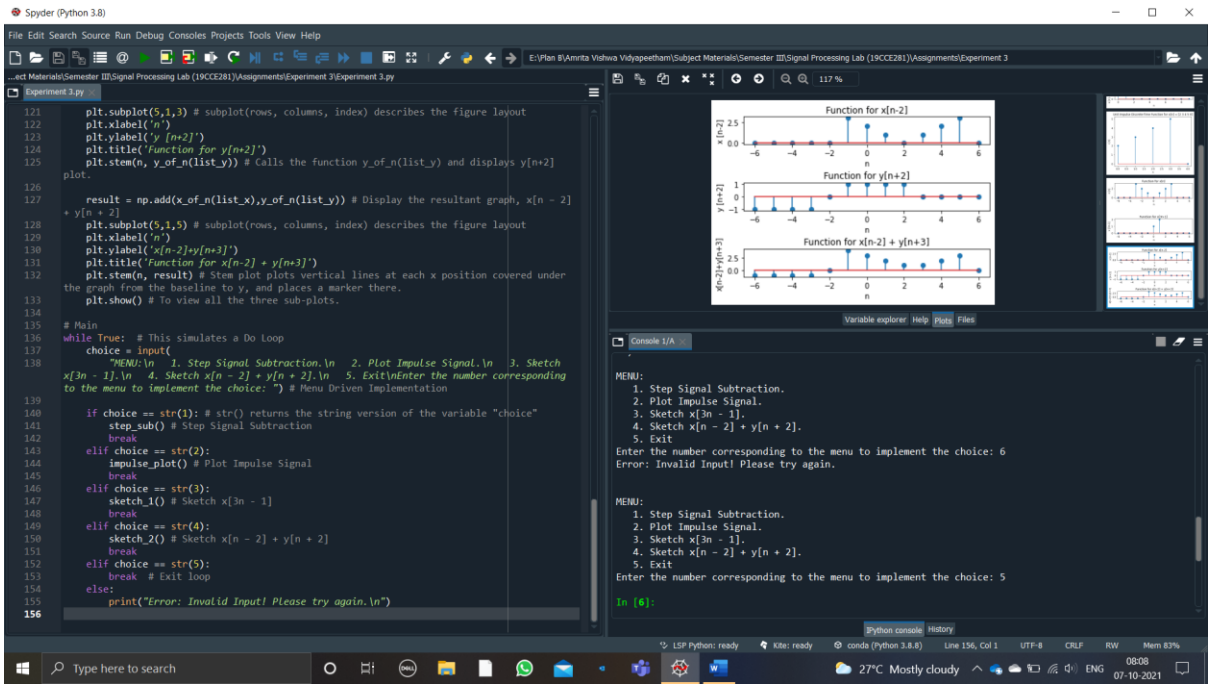

Function for x[3n-1]
x [3n-1]
n

Sketch x[n – 2] + y[n + 2]





Menu Driven Implementation



Thank You!

```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.29.0 -- An enhanced Interactive Python.

Restarting kernel...
```

```
In [1]:         'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/
Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py'      ='E:/Plan
B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab
(19CCE281)/Assignments/Experiment 3'

MENU:
   1. Step Signal Subtraction.
   2. Plot Impulse Signal.
   3. Sketch x[3n - 1].
   4. Sketch x[n – 2] + y[n + 2].
   5. Exit
Enter the number corresponding to the menu to implement the choice: 1

MENU:
   1. Step Signal Subtraction.
   2. Plot Impulse Signal.
   3. Sketch x[3n - 1].
   4. Sketch x[n – 2] + y[n + 2].
   5. Exit
Enter the number corresponding to the menu to implement the choice: 2

MENU:
   1. Step Signal Subtraction.
   2. Plot Impulse Signal.
   3. Sketch x[3n - 1].
   4. Sketch x[n – 2] + y[n + 2].
   5. Exit
Enter the number corresponding to the menu to implement the choice: 3

MENU:
   1. Step Signal Subtraction.
   2. Plot Impulse Signal.
   3. Sketch x[3n - 1].
   4. Sketch x[n – 2] + y[n + 2].
   5. Exit
Enter the number corresponding to the menu to implement the choice: 4

MENU:
   1. Step Signal Subtraction.
   2. Plot Impulse Signal.
   3. Sketch x[3n - 1].
   4. Sketch x[n – 2] + y[n + 2].
   5. Exit
Enter the number corresponding to the menu to implement the choice: 6
Error: Invalid Input! Please try again.
```

```
MENU:
   1. Step Signal Subtraction.
   2. Plot Impulse Signal.
   3. Sketch x[3n - 1].
   4. Sketch x[n – 2] + y[n + 2].
   5. Exit
Enter the number corresponding to the menu to implement the choice: 5

In [2]:
```