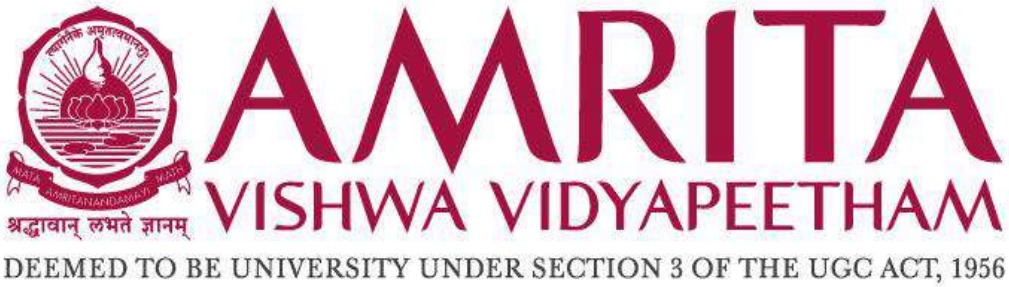
B. Tech – Computer and Communication Engineering (CCE)

Department of Electronics and Communication Engineering

Amrita School of Engineering, Coimbatore – 641112



19CCE281 Signal Processing Lab

Third Semester, Second Year

Name: Santosh

Roll Number: CB.EN.U4CCE20053

Period: September 2021 – January 2022

Academic Year – 2021-22

Faculty In-charge – Dr S. Kirthiga Mam, Ms R. Karthika Mam and Mr K. Pargunarajan Sir

TABLE OF CONTENTS

Lab Title and Code: Signal Processing Lab 19CCE281
Experiment No:1
Date:

INTRODUCTION TO PYTHON

Aim:

Python Background

Brief intro to Python in your own words (200 to 250 words, not exceeding one page, as 4 paragraphs

Python Getting Started and Commands

The following needs to be understood and documented in record with examples

Python Various Modes
Python Indentation
Python Variables
Creating a Comment
Python Numbers
Python Operators
Python Collections (Arrays)
Python Conditions and If statements
Python For Loops
Python Functions
Python Libraries

Inference:

NAME - SANTOSH (CB.EN.U4CLE20053)
DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CLE)
ACADEMIC YEAR - 2020-24 BATCH

①

LAB TITLE AND CODE : SIGNAL PROCESSING LAB 19CLE281
EXPERIMENT NUMBER : 1
DATE : 13/09/2021

INTRODUCTION TO PYTHON

* **AIM :**
A brief introduction to python with the implementation of basic commands.

* **SOFTWARE REQUIRED :**
Spyder IDE (Anaconda 3) - Python 3.9

* **THEORY - PYTHON BACKGROUND :**
Python is a cross-platform programming language, meaning, it runs on multiple platforms like Windows, Mac OS X, Linux, Unix and has even been ported to the Java and .NET virtual machines. It was created by Guido van Rossum, and released in 1991. Python is an open-source (free) programming language that is used in web programming, data science, artificial intelligence, and many scientific applications.

Python can be used on a server to create a web application, alongside software to create workflows. It can connect to database systems, read and modify files. It can also be used to handle big data and perform complex mathematics, for rapid prototyping, or production-ready software development.

Python has a simple syntax similar to the English language that allows developers to write programs with fewer lines

than some other programming languages. It runs on an interpreter system, meaning the code can be executed as soon as it is written. This means that prototyping can be very quick and can be treated procedurally, in an object-oriented way or a functional way.

Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of a program maintenance. Python supports modules and packages, which encourages modularity and code usage. The Python interpreter and the executive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

* ALGORITHM - PYTHON GETTING STARTED AND COMMANDS :

→ PYTHON VARIOUS MODES :-

① Immediate Mode - Typing python in the command line will invoke the interpreter in immediate mode. We can directly type in Python expressions and press enter to get the output.
For example, >>> is the python prompt. It tells us that the interpreter is ready for our output. Try typing in 1+1 and press enter. We get 2 as the output. This prompt can be used as a calculator. To exit this mode type exit() or quit() and press enter.
print ("Hello, World")

② Script Mode - This mode is used to execute a python program written in a file. Such a file is called a script. Scripts can be saved to disk for future use. Python scripts have the extension .py, meaning that the filename ends with .py. For example,
HelloWorld.py

DTO →

③ Integrated Development Environment (IDE) - We just need to save it with the .py extension. But using an IDE can make our work a lot easier. IDE is a piece of software that provides useful features like code hinting, syntax highlighting and checking, file explorers etc. to the programmer for application development. Using an IDE can get rid of redundant tasks and significantly decrease the time required for application development.

→ PYTHON LINES AND INDENTATION :-
Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced. For example,

```
if 5 > 2:
    print ("Five is greater than two!")
# OUTPUT - Five is greater than two!
```

→ PYTHON COMMENTS :-
Comments can be used to explain python code, make the code more readable and prevent execution when testing code.

① Creating a comment - Comments starts with a #, and python will ignore them. For example,

```
# This is a comment
print ("Hello, World!")
```

Comments can be placed at the end of a line, and python will ignore the rest of the line. For example,

```
print ("Hello, World!") # This is a comment
```

A comment does not have to be text that explains the code, it can also be used to prevent python from executing code. For example,

```
# print ("Hello, World!")
print ("Cheers, mate!")
```

② Multi-line Comments - Python does not have a syntax for multi-line comments. To add a multi-line comment, you can insert a # for each line. For example,

```
# This is a comment
# written in
# more than just one line
print("Hello, World!")
```

Or, not quite as intended, you can use a multiline string. Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it. For example,

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

→ PYTHON VARIABLES :-

Variables are containers for storing data values.

① Creating variables - Python has no command for declaring a variable. A variable is cleared the moment you first assign a value to it. For example,

```
x = 5
y = "John"
print(x)
print(y)
```

Output :
5
John

⑤

Variables do not need to be declared with any particular type, and can even change type after they have been set. For example,

```
x = 4          # x is of type int
x = "Sally"    # x is of type str now
print(x)       # OUTPUT - Sally
```

② Casting - If you want to specify the data type of a variable, this can be done with casting. For example,

```
x = str(3)     # x will be '3'
y = int(3)     # y will be 3
z = float(3)   # z will be 3.0
```

③ Get the Type - You can get the data type of a variable with the type() function. For example,

```
x = 5
y = John
print(type(x))   # OUTPUT - <class 'int'>
print(type(y))   # OUTPUT - <class 'str'>
```

④ Single or Double quotes - String variables can be declared either by using single or double-quotes. For example,

```
x = "John"
# is the same as
x = 'John'
```

⑤ Case-Sensitive - Variable names are case-sensitive. For example,

```
# This will create two variables
a = 4
A = "Sally"
# A will not overwrite a
```

→ PYTHON NUMBERS :-

There are three numeric types in python, int, float and complex. Variables of numeric types are created when you assign a value to them. For example,

```
x = 1      # int
y = 2.8    # float
z = 1j     # complex
```

To verify the type of any object in python, use the type() function. For example,

```
print (type (x))  # OUTPUT - <class 'int'>
print (type (y))  # OUTPUT - <class 'float'>
print (type (z))  # OUTPUT - <class 'complex'>
```

→ PYTHON ASSIGN VALUES TO MULTIPLE VARIABLES :-

Python allows you to assign values to multiple variables in one line. For example,

```
x, y, z = "Orange", "Banana", "Cherry"
print (x)
print (y)
print (z)
```

OUTPUT:
Orange
Banana
Cherry

And you can assign the same value to multiple variables in one line. For example,

```
x = y = z = "Orange"
print (x)
print (y)
print (z)
```

OUTPUT:
Orange
Orange
Orange

→ PYTHON USER INAT :-

Python allows for user input. That means we can ask the user for input. The following example asks for the username, it gets printed on the screen.

```
username = input ("Enter username : ")
print ("Username is : " + username)
```

⑦

→ Python Operators :-
operators are used for performing operations on variables and values. Python divides the operators into the following groups:

① Arithmetic operators -

Operator	Description	Syntax
+	Addition : adds two operands	x + y
-	Subtraction : subtracts two operands	x - y
*	Multiplication : multiplies two operands	x * y
/	Division (float) : Divides the first operand by the second	x / y
//	Division (floor) : Divides the first operand by the second	x // y
%	Modulus : Returns the remainder when the first operand is divided by the second	x % y
**	Power : Returns first raised to power second	x ** y

② Assignment operators -

Operator	Description	Syntax
=	Assign a value of right side of expression to left side operand	x = y + z
+=	Add and Assign : Add right-side operand with left side operand and then assign to left operand	a += b
-=	Subtract AND : Subtract right operand from left operand and then assign to left operand : True if both operands are equal	a -= b
*=	Multiply AND : Multiply right operand with left operand and then assign to left operand	a *= b

operator	Description	Syntax
/=	Divide AND : Divide left operand with right operand and then assign to left operand	a/=b
%=	Modulus AND: Takes modulus using left and right operands and assign the result to left operand	a%=b
//=	Divide (floor) AND: Divide left operand with right operand and then assign the value (floor) to left operand	a//=b
**=	Exponent AND : Calculate exponent (raise power) value using operands and assign value to left operand	a**=b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b
^=	Performs xOR on operands and assign value to left operand	a^=b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a<<=b

③ Comparison operators -

operator	Description	Syntax
>	Greater than : True if the left operand is greater than the right	x > y
<	less than : True if the left operand is less than the right	x < y

Operator	Description	Syntax
==	Equal to: True if both operands are equal	x == y
!=	Not equal to: True if both operands are not equal	x != y
>=	Greater than or equal to: True if the left operand is greater than or equal to the right	x >= y
<=	less than or equal to: True if the left operand is less than or equal to the right	x <= y

⑭ Logical operators -

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if ~~both~~ either the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

⑮ Identity operators -

Operator	Description	Syntax
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. Here, it results in 1 if id(x) equals id(y)	x is y
is not	Evaluates to ~~true~~ false if the variables on either side of the operator point to the same object and false otherwise. Here, not in results in a 1 if x is not a member of sequence y.	x is not y

⑥ Membership operators -

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise	x in y, here in results in a 1 if x is a member of sequence y
not in	Evaluates to true if does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

⑦ Bitwise operators -

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands.	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

→ PYTHON COLLECTIONS (ARRAYS) :-

There are four collection data types in the python programming language -

① List - List items are ordered, changeable, and allow duplicate values. They are indexed, the first item has index [0], the second item has index [1] etc. They are created using square brackets.

For example,

```
thislist = ["apple", "banana", "cherry"]
print(thislist) # OUTPUT - ["apple", "banana", "cherry"]
```

② Tuple - Tuple items are ordered, unchangeable, and allow duplicate values. They are indexed, the first item has index [0], the second item has index [1] etc. They are written with round brackets.

For example,

```
thistuple = ("apple", "banana", "cherry")
print(thistuple) # OUTPUT - ("apple", "banana", "cherry")
```

③ Set - Set items are unordered, unchangeable, and do not allow duplicate values. Sets are unordered, so you cannot be sure in which order the items will appear. They are written in curly brackets. For example,

```
thisset = {"apple", "banana", "cherry"}
print(thisset) # OUTPUT - {"banana", "cherry", "apple"}
```

④ Dictionaries - Dictionary items are ordered, changeable, and does not allow duplicates. They have been presented in key : value pairs, and can be referred to by using the key name. They are written with curly brackets and have keys and values. For example,

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

OUTPUT -
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

→ Python If ... Else:-

① Python Conditions and If statements - Python supports the usual logical conditions from mathematics.

(i) Equals : a==b
(ii) Not Equals : a!=b
(iii) Less than : a<b
(iv) Less than or equal to : a<=b
(v) Greater than : a>b
(vi) Greater than or equal to : a>=b

Syntax for If :
if expression :
  statement(s)

These conditions can be used in several ways, most commonly in "if statements" and loops. An "if statement" is written by using the if keyword. For example,

```
a = 33
b = 200
if b>a:  # IF
  print ("b is greater than a")
# OUTPUT - b is greater than a
```

Syntax for If..else
if expression :
  statement(s)
else :
  statement(s)

② Elif - The elif keyword is python's way of saying "if the previous conditions were not true, then try this condition". For example,

```
a = 33
b = 33
if b>a:
  print ("b is greater than a")
elif a==b:  # ELIF
  print ("a and b are equal")
  # OUTPUT - a and b are equal
```

③ Else - The else keyword catches anything which isn't caught by the preceding conditions. For example,

PTO

```
a = 200
b = 33
if b > a:  # IF
  print ("b is greater than a")
elif a == b:  # ELIF
  print ("a and b are equal")
else :  # ELSE
  print ("a is greater than b")
  # a is greater than b
```

```
syntax
if expression 1:
  statement(s)
elif expression 2:
  statement(s)
elif expression 3:
  statement(s)
else:
  statement(s)
```

④ And - The and keyword is a logical operator and is used to combine conditional statements. For example,

```
a = 200
b = 33
c = 500
if a > b and c > a:  # AND
  print (" Both conditions are True")
  # Both conditions are True
```

⑤ Or - The or keyword is a logical operator and is used to combine conditional statements. For example,

```
a = 200
b = 33
c = 500
if a > b or a > c:  # OR
  print ("At least one of the conditions is True")
  # At least one of the conditions is True
```

⑥ Nested If - You can have if statements inside if statements, this is called nested if statements. For example,

```
x = 41
if x > 10 :
  print ("Above ten,")
if x > 20
  print ("and also above 20!")
else :
  print ("but not above 20.")
```

(Nested)

OUTPUT:
Above ten,
and also above 20!

(7) The Pass statement - If statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error. For example,

```
a = 33
b = 200
if b > a:
  pass  # No functionality implemented
```

→ PYTHON WHILE LOOPS :-

With the while loop, we can execute a set of statements as long as a condition is true. For example,

```
i = 1
while i < 6:
  print(i)
  i += 1
```

OUTPUT :
1
2
3
4
5

Syntax :
while test_expression :
    Body of while

The while loop requires relevant variables to be ready. In this example, we need to define an indexing variable, i, which we set to 1.

→ PYTHON FOR LOOPS :-

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). This is less like the for keyword in other programming languages and works more like an iterator method as found in other object-oriented programming languages. With the for loop, we can execute a set of statements, once for each item in a list, tuple, set, etc. The for loop does not require an indexing variable to set beforehand. For example,

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

OUTPUT :
apple
banana
cherry

Syntax :
for val in sequence :
  loop body

→ PYTHON CONTROL STATEMENTS :-

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements :-

① break - Terminates the loop statement and transfers execution to the statement immediately following the loop.

② continue - Causes the loop to skip the remainder of its body and immediately retest its condition before reiterating.

③ pass - Used when a statement is required syntactically but you do not want any command or code to execute.

→ PYTHON FUNCTIONS :-

A function is a block of code that only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

① Creating a function - In python, a function is defined using the def keyword. For example,

```
def my_function1():
    print("Hello from a function!")
```

② Calling a function - To call a function, use the function name followed by parenthesis. For example,

```
def my_function1():
    print("Hello from a function!")
my_function1()
# OUTPUT - Hello from a function!
```

PTO

③ Arguments - Information can be passed into functions as arguments. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just seperate them with a comma. The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name -

```
def my_function2 (fname):
    print (fname + "Refsnes")
my_function2 ("Emil")
my_function2 ("Tobias")
my_function2 ("Linus")
```

OUTPUT:
Emil Refsnes
Tobias Refsnes
Linus Refsnes .....

④ Number of Arguments - By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the functions with 2 arguments, not more, and not less. The following function expects 2 arguments, and gets 2 arguments -

```
def my_function3 (fname, lname):
    print (fname + " " + lname)
my_function ("Emil", "Refsnes")
```

If you try to call the function with 1 or 3 arguments, you will get an error.

```
# OUTPUT - Emil Refsnes
```

⑤ Arbitrary Arguments, *args - If you do not know how many arguments will be passed into your function, add a * before the parameter name in the function definition. This way the function will receive a tuple of arguments and can access the items accordingly. For example,

```
def my_function4 (*kids):
    print ("The youngest child is " + kids[2])
my_function4 ("Emil", "Tobias", "Linus")
# OUTPUT - The youngest child is Linus
```

6. Keyword Arguments - You can also send arguments with the key = value syntax. This way the order of the arguments does not matter. For example,

```
def my_function5 (child3, child2, child1):
    print ("The youngest child is " + child3)
my_function5 (child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

The phrase Keyword Arguments are often shortened to kwargs in python documentations.

```
# OUTPUT - The youngest child is Linus
```

7. Arbitrary Keyword Arguments, **kwargs - If you do not know how many keyword arguments will be passed into your function, add two asterisks: ** before the parameter name in the function definition. This way the function will receive a dictionary of arguments and can access the items accordingly. For example,

```
def my_function6 (**kid):
    print ("His last name is " + kid["lname"])
my_function6 (fname = "Tobias", lname = "Refsnes")
```

Arbitrary Keyword Arguments are often shortened to **kwargs in python documentations.

```
# OUTPUT - His last name is Refsnes
```

8. Default Parameter Value - The following example shows how to use a default parameter value. If we call the function without argument, it uses the default values

```
def my_function7 (country = "Norway"):
    print ("I am from " + country)
my_function7 ("Sweden")
my_function7 ("India")
my_function7 ()
my_function7 ("Brazil")
```

OUTPUTS
I am from Sweden
I am from India
I am from Norway
I am from Brazil

PTO →

(9) Passing a List as an Argument - You can send any data type of argument to a function (string, number, list, dictionary, etc.) and it will be treated as the same data type inside the function. For example, if you send a list as an argument, it will still be a list when it reaches the function,

```
def my_function8(food):
    for x in food:
        print(x)
fruits = ["apple", "banana", "cherry"]
my_function8(fruits)
```

OUTPUT:
apple
banana
cherry

(10) Return Values - To let a function return a value, use the return statement. For example,

```
def my_function9(x)
    return 5 * x
print(my_function9(3))
print(my_function9(5))
print(my_function9(9))
```

OUTPUT:
15
25
45

→ PYTHON IMPORT STATEMENTS :-

You can use any python source file as a module by executing an import statement in some other python source file. When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example,

```
import random # imports random source file.
val1 = random.randrange(100);
val2 = random.randrange(100);
print("Following are the two generated random numbers under 100")
print("First: ", val1); # Gives a random number <100.
print("Second: ", val2); # Gives a random number <100.
```

→ PYTHON HBRARIES :-

① Numpy - Numerical Python (NumPy) is a python library used for working with arrays. It also has functions per working in the domain of linear algebra, fourier transform and matrices. It was created in 2005 by Travis oliphant, is an open-source project and we can use it freely. It aims to provide an array object that is up to 50x faster than traditional python lists.

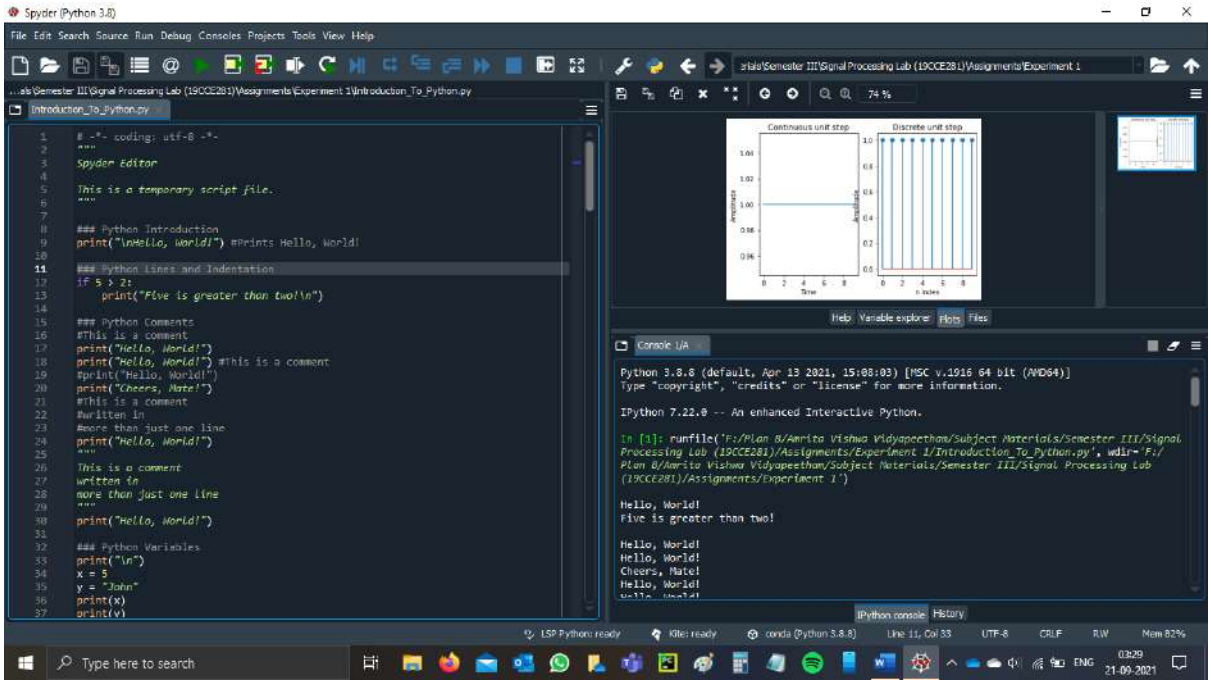
② Matplotlib - Matplotlib is a low-level graph plotting library in python that serves as a visualization utility. It was created by John D. Hunter, is open source and we can use it freely. It is now a comprehensive library for creating static, animated, and interactive visualizations in python.

For example, [sample code]

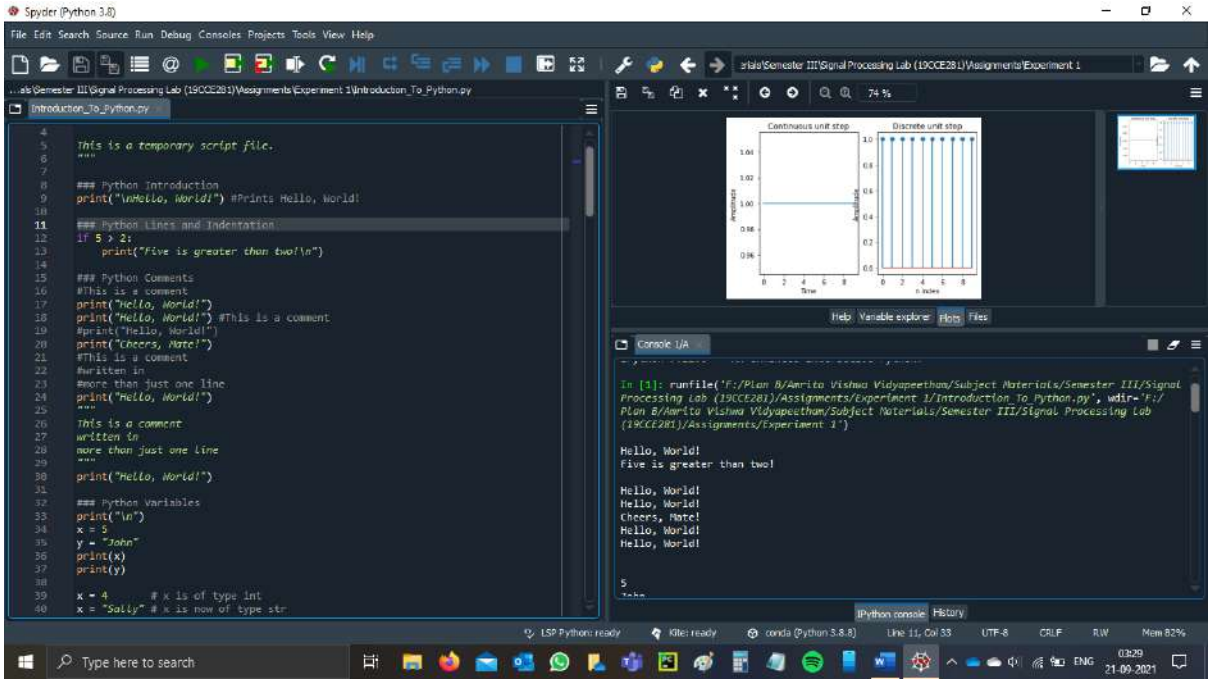
```
import numpy as np
import matplotlib.pyplot as plt
x= np.ones(10)
plt.subplot(121)
plt.plot(x)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('continuous unit step')
plt.subplot(122)
plt.stem(x)
plt.xlabel('n index')
plt.ylabel('Amplitude')
plt.title('Discrete unit step')
```

+ INFERENCE:
A brief introduction to python is documented along with syntaxes and examples for basic concepts and commands.

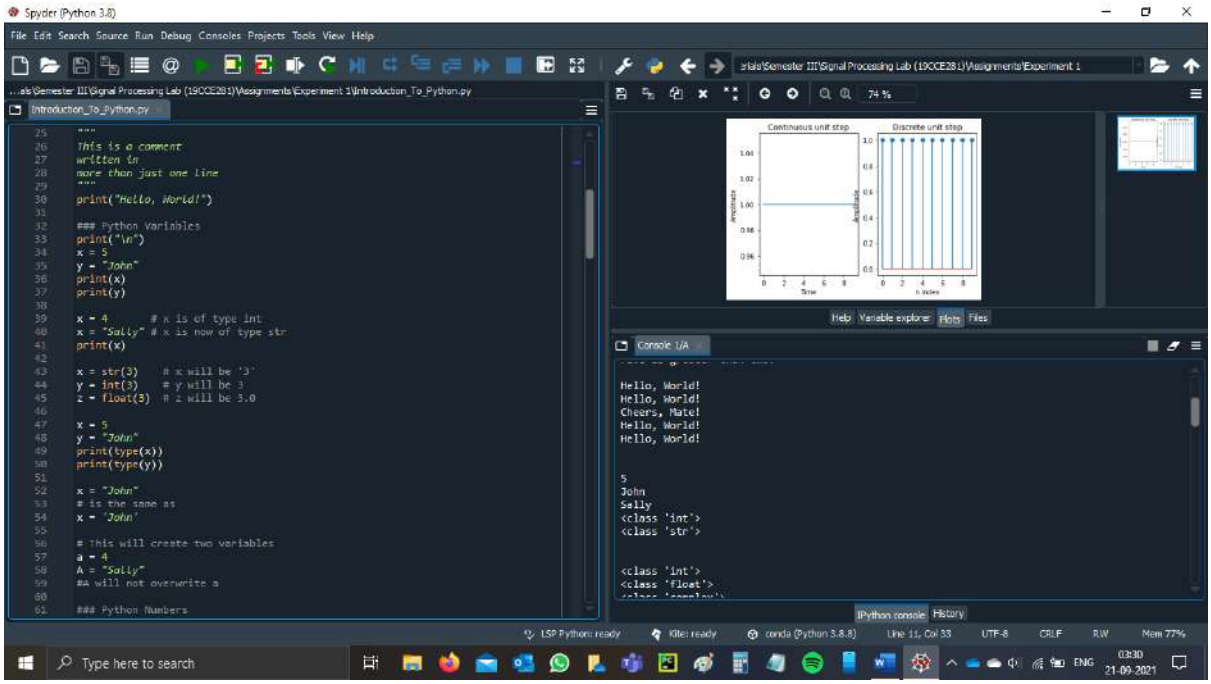
Python Lines and Indentation



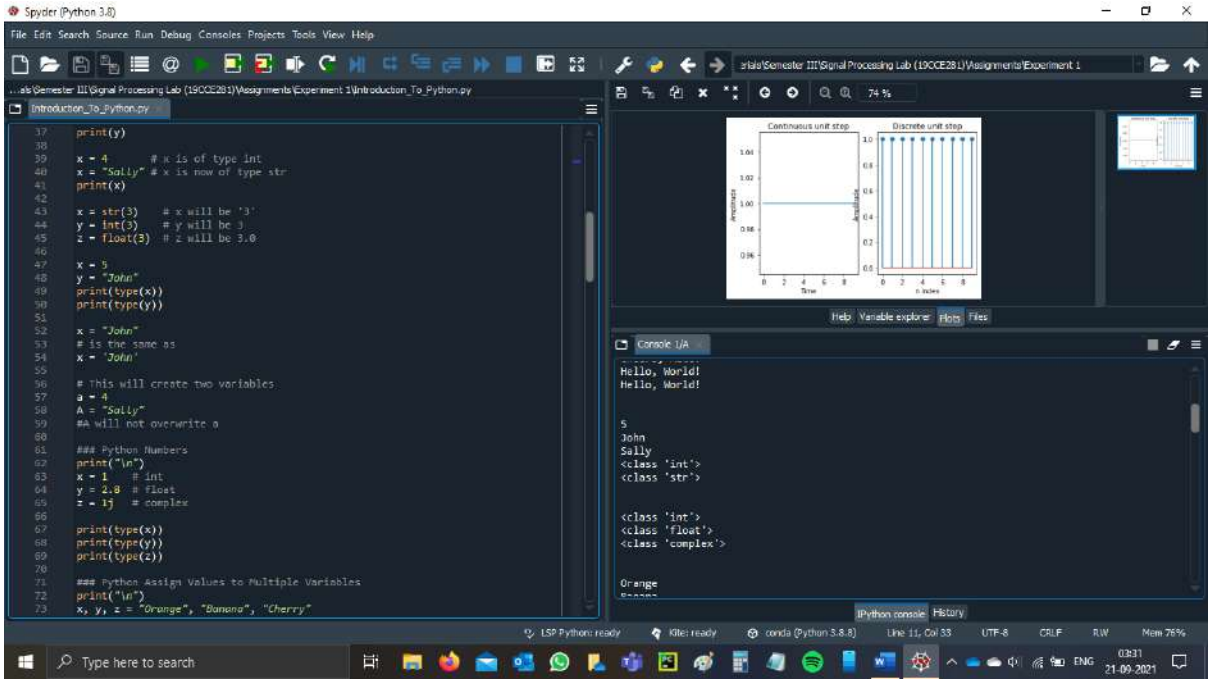
Python Comments



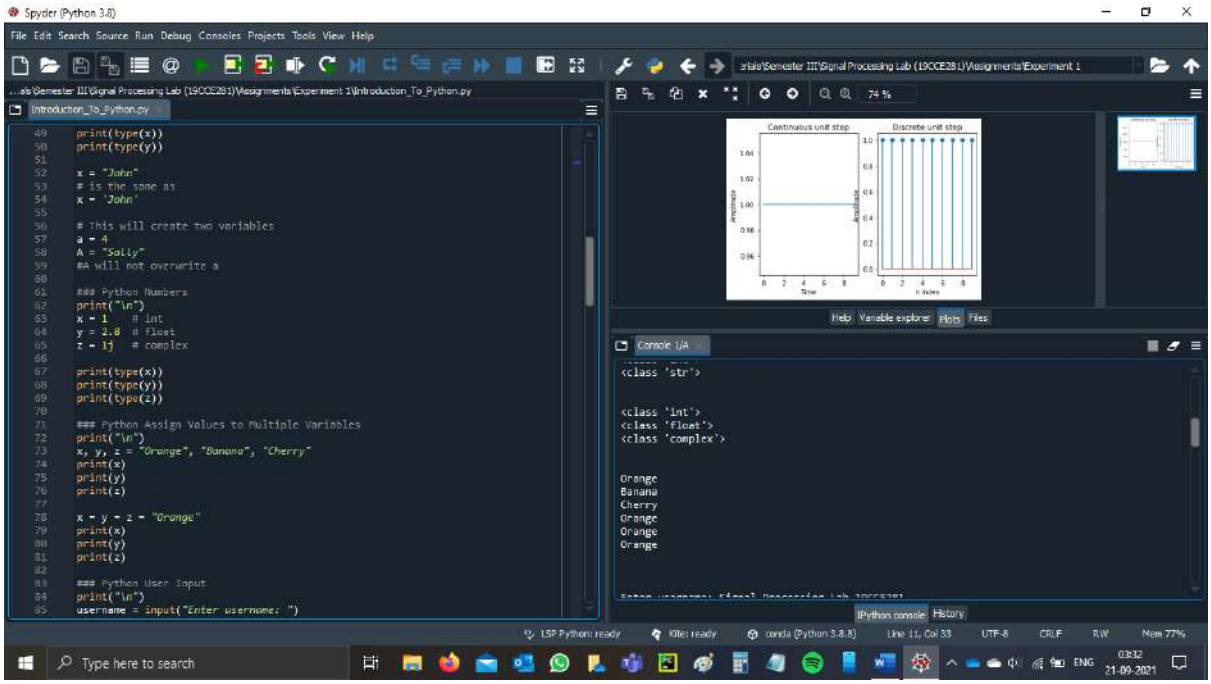
Python Variables



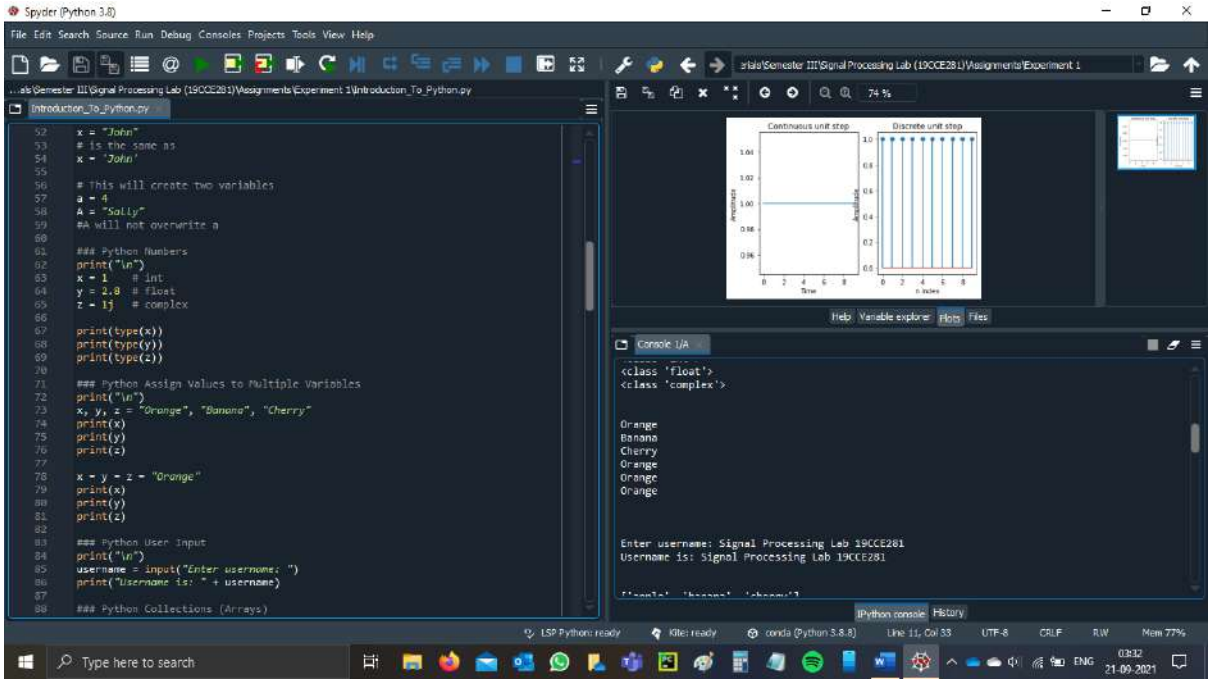
Python Numbers



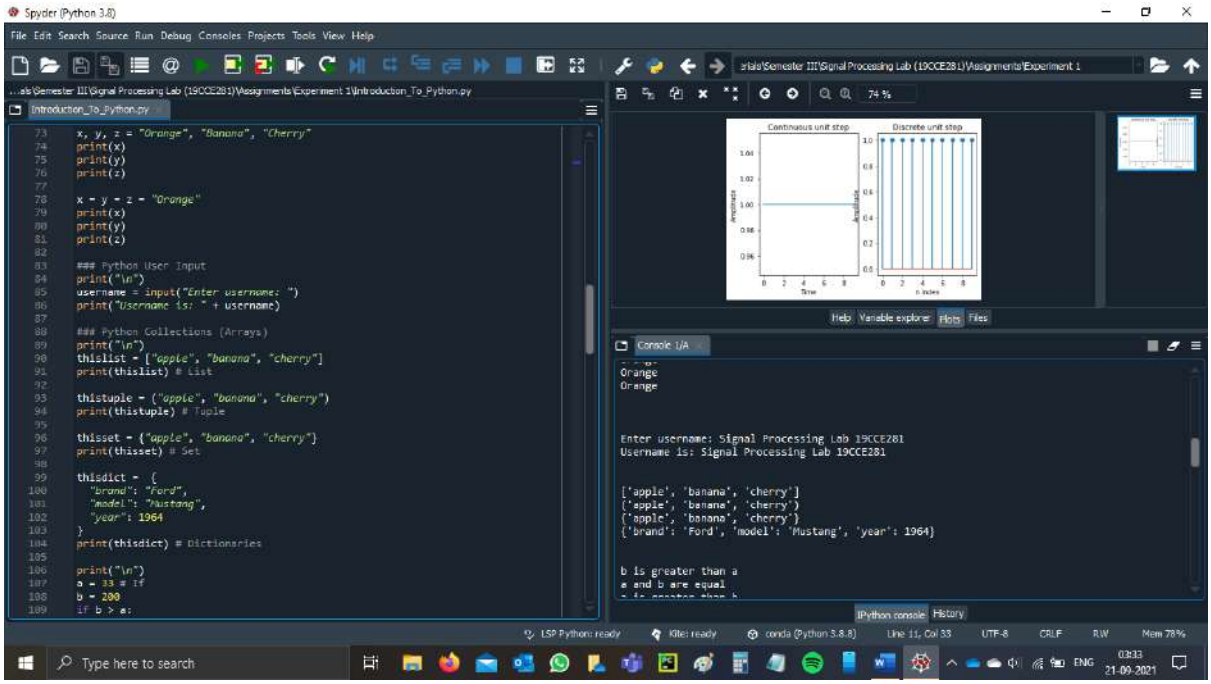
Python Assign Values to Multiple Variables



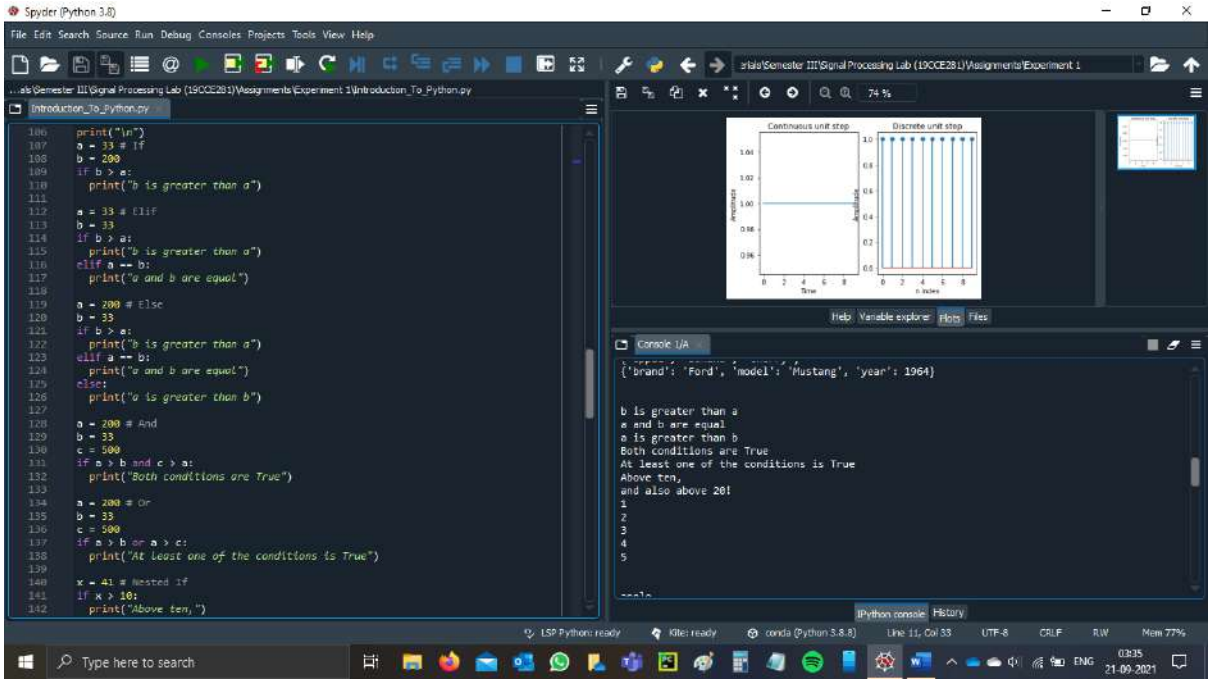
Python User Input



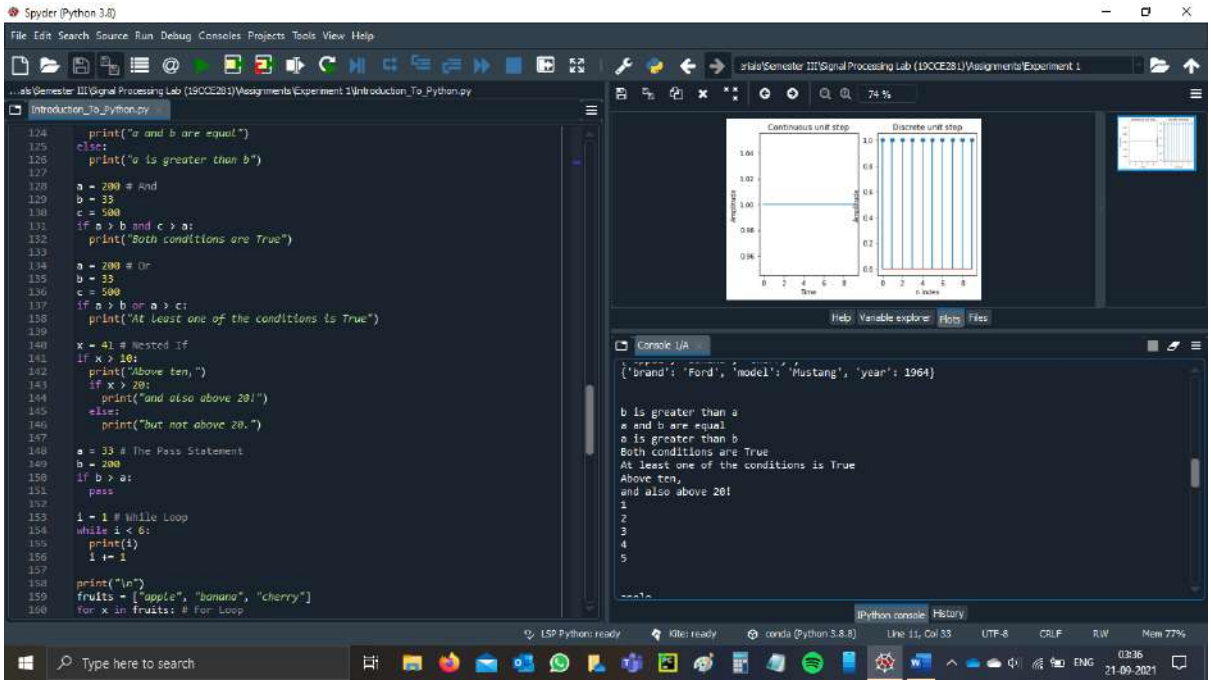
Python Collections (Arrays)



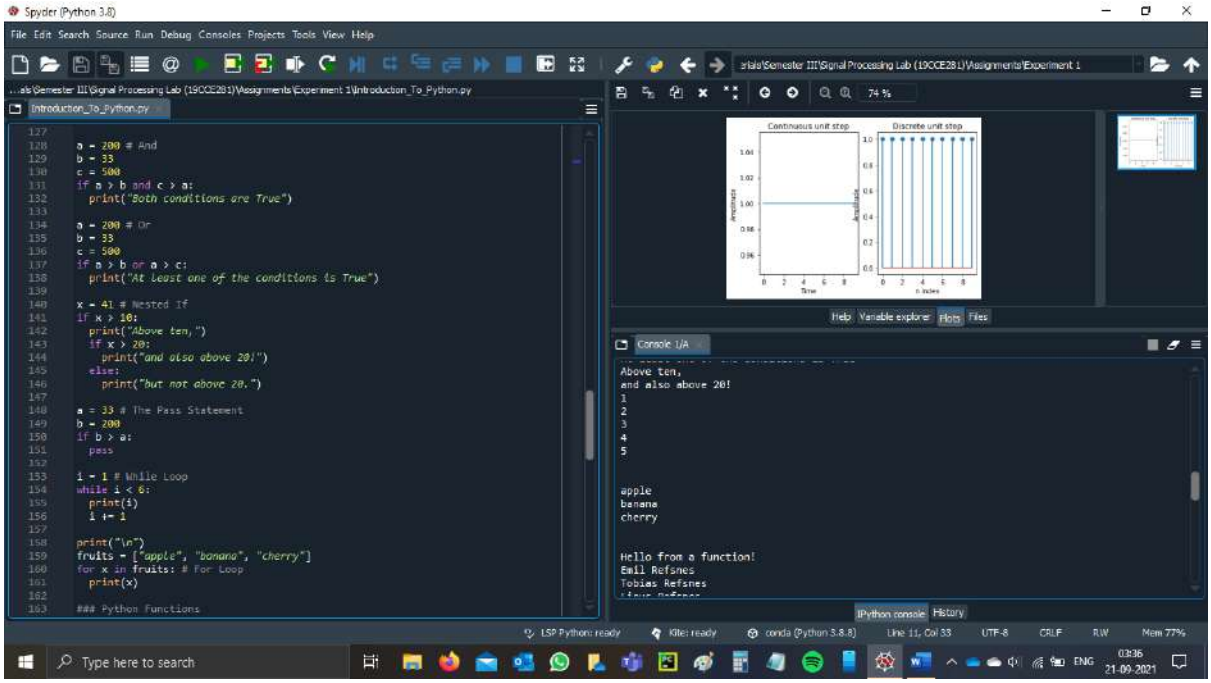
Python If … Else



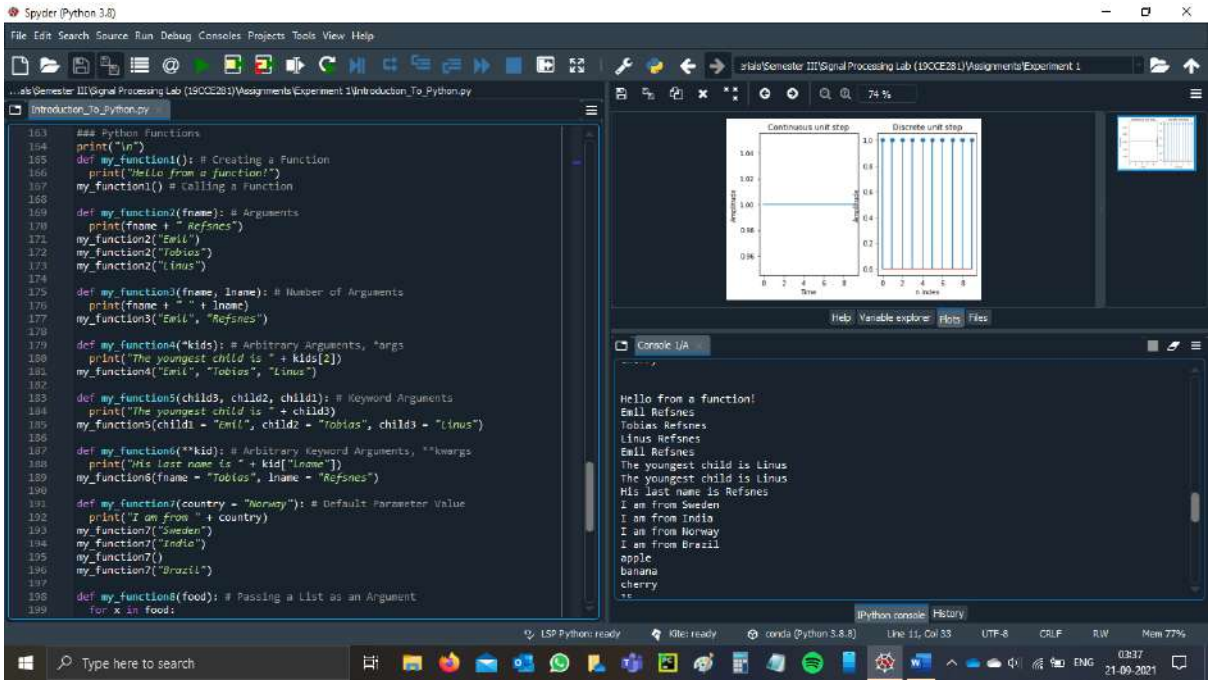
Python While Loop



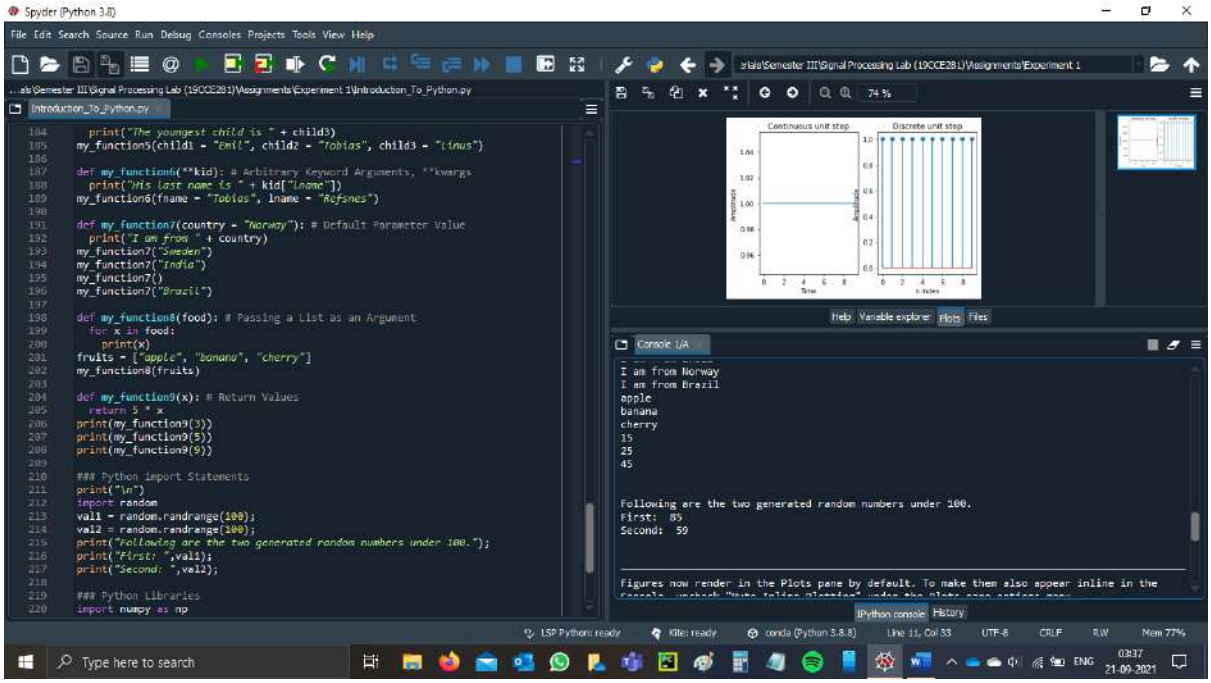
Python For Loop



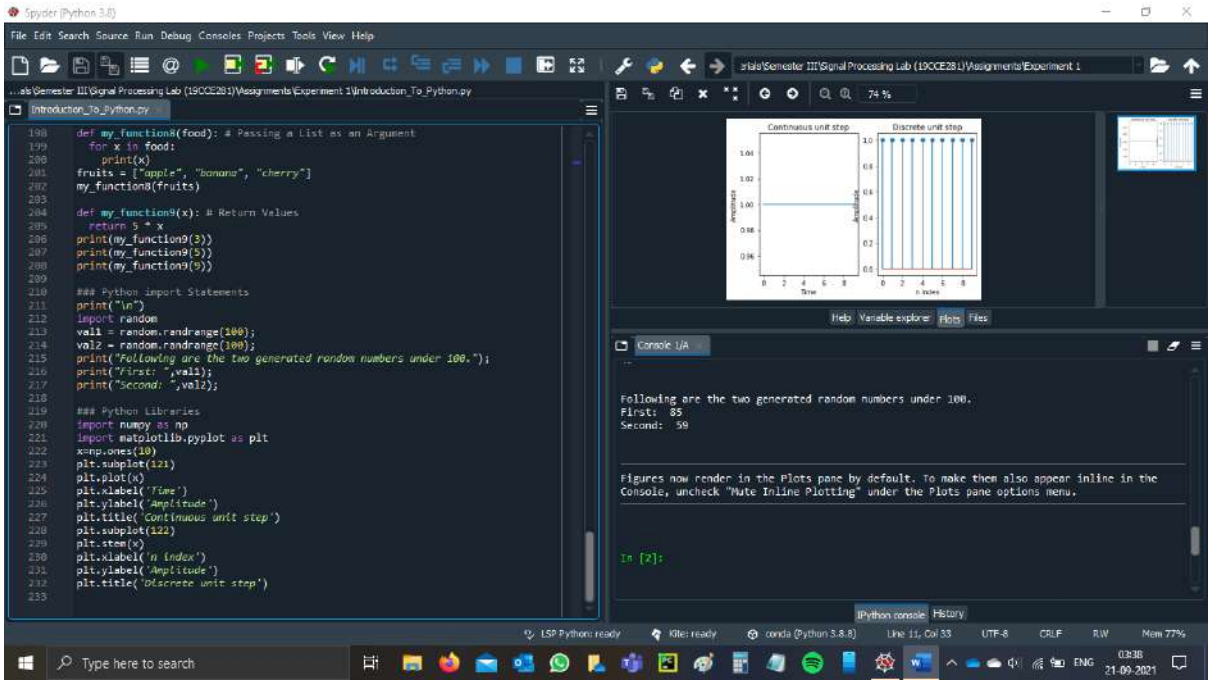
Python Functions

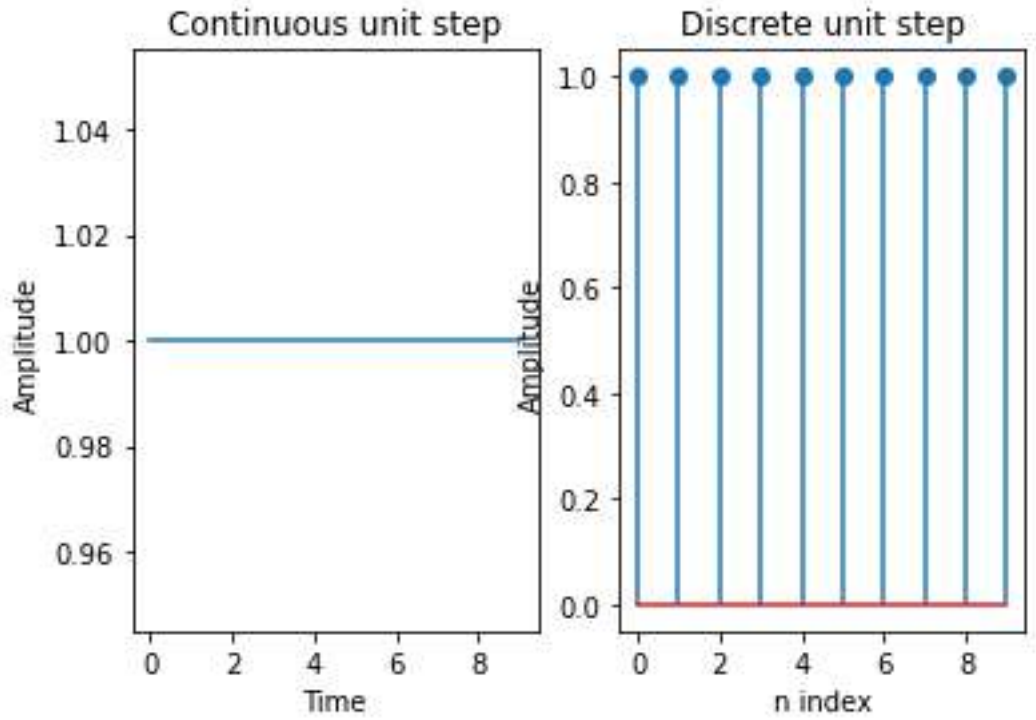


Python Import Statements



Python Libraries





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.

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

Hello, World!
Five is greater than two!

Hello, World!
Hello, World!
Cheers, Mate!
Hello, World!
Hello, World!


5
John
Sally
<class 'int'>
<class 'str'>


<class 'int'>
<class 'float'>
<class 'complex'>


Orange
Banana
Cherry
Orange
Orange
Orange



Enter username: Signal Processing Lab 19CCE281
Username is: Signal Processing Lab 19CCE281


['apple', 'banana', 'cherry']
('apple', 'banana', 'cherry')
{'apple', 'cherry', 'banana'}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}


b is greater than a
a and b are equal
a is greater than b
Both conditions are True
```

```
At least one of the conditions is True
Above ten,
and also above 20!
1
2
3
4
5


apple
banana
cherry


Hello from a function!
Emil Refsnes
Tobias Refsnes
Linus Refsnes
Emil Refsnes
The youngest child is Linus
The youngest child is Linus
His last name is Refsnes
I am from Sweden
I am from India
I am from Norway
I am from Brazil
apple
banana
cherry
15
25
45


Following are the two generated random numbers under 100.
First:  34
Second:  86
```

Warning
Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.

In [2]:

Expt 2: Question is as follows

Write Python code for the generation of sequences for Sinusoidal, Exponential, Impulse function, Unit Step function, and Ramp function. (Note: Continuous and Discrete time sequence to be plotted for all signals)

Specification are as follows

a) Sinusoidal: (i) 325sin(100πt) for -.02≤ t ≤ 0.05 with 1000 samples ; (ii) 325sin(100πn) for 0 ≤ n ≤ 50 ;

b) Exponential: (i) exp(j100πt) for -.02≤ t ≤ 0.05 with 1000 samples, plot real and imaginary parts;

(ii) (2^n) for -10 ≤ n ≤ 10;

c) Unit step ; unit impulse and ramp functions to be generated for 0 ≤ n ≤ 9;

NAME - SANTOSH (CB.EN.U4CCE20053)

DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE) A

LAB TITLE AND CODE : SIGNAL PROCESSING LAB 19CCE281

EXPERIMENT NO: 2

DATE : 21/09/2021

GENERATION OF ELEMENTARY SIGNALS

★ AIMS :

Plot continuous-time and discrete-time sequence for the generation of sinusoidal, exponential, impulse, unit step and ramp functions using Python code.

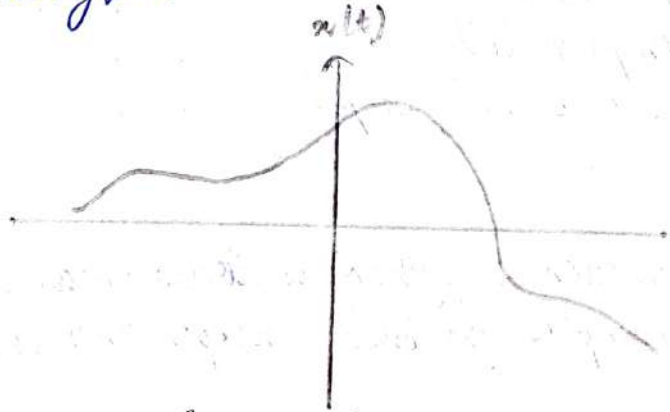
★ SOFTWARE REQUIRED :

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

★ THEORY :

① Continuous-Time Signal - It is a varying quantity (a signal) whose domain, which is often time, is a continuum (e.g.: a connected interval of the reals). That is, the function's domain is an uncountable set. The function itself need not be continuous.

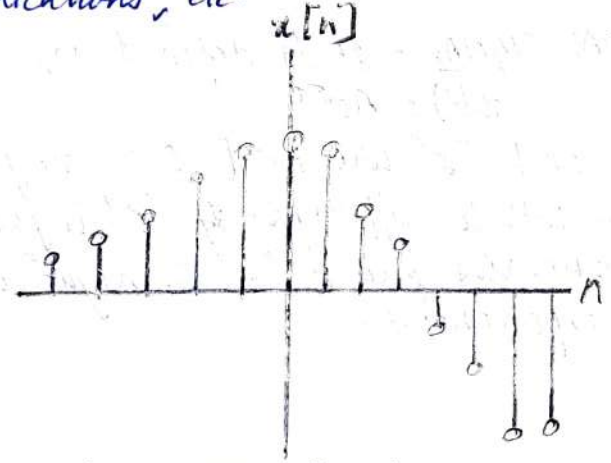
Defined for all time instants and not at specific time instants. The electrical signals derived in proportion with the physical quantities such as temperature, pressure, sound etc. are generally continuous. Applications of these signals include Public Switched Telephone Network (PSTN), FM radio Broadcasting, Image Processing, etc.



Continuous-Time Signal

(2) Discrete-Time Signal – It is a time series consisting of a sequence of a sequence of quantities. Discrete-time views values of variables as occurring at distinct, seperate "points in time", or equivalently as being unchanged throughout each non-zero region of time ("time period") - that is, time is viewed as a discrete variable.

Unlike a continuous-time signal, a discrete-time signal is not a function of a continuous argument; however, it may have been obtained by sampling from a continuous-time signal. For example, if you were monitoring the temperature of a room, you would be able to take a measured value of temperature at any time. Applications of these signals include speech processing, SONAR, communications, etc.

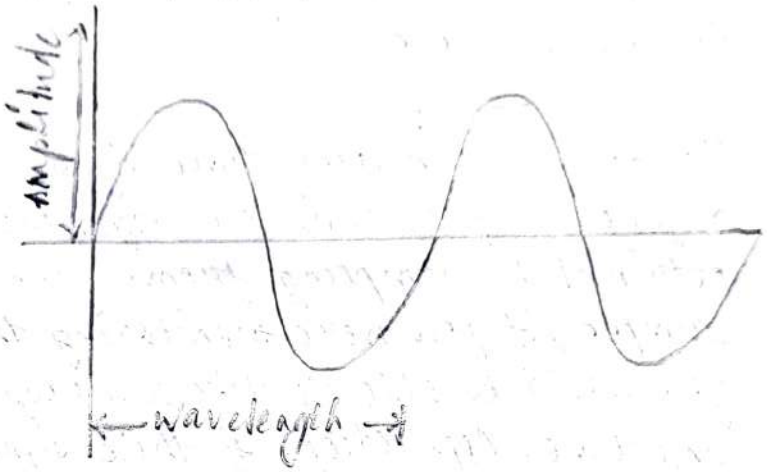


Discrete-Time Signal

(3) Sinusoidal signal - These are periodic functions that are based on the sine or cosine function from trigonometry. When the sine wave starts from zero and covers positive values, reaches zero; and again covers negative values, reaches zero, it is said to have completed one cycle or single cycle.

The continuous-time version of a sinusoidal signal, in its most general form, may be written as $x(t) = A\cos(\omega t + \phi)$ where A is the amplitude, ω is the frequency in radians per second, and ϕ is the angle in radians.

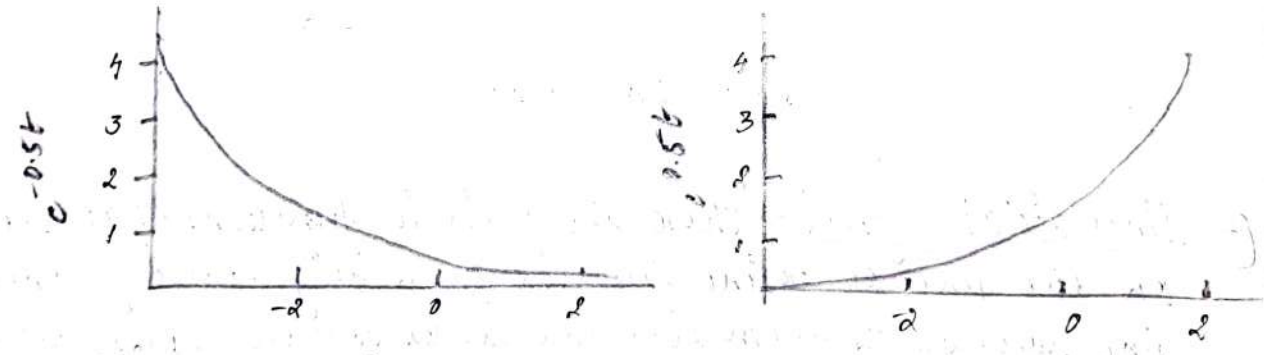
The discrete-time version of a sinusoidal signal, in its most general form, may be written as $x[n] = A\cos(\Omega n + \phi)$, where the angular frequency Ω must be rational multiple of 2π for the signal to be periodic. Here, the period is measured in samples.



(4) REAL EXPONENTIAL SIGNAL - It is defined as,

$$x(t) = Ae^{\sigma t}$$

where both "A" and "σ" are real. Depending on the value of the "σ" the signals will be different. If "σ" is positive, the signal $x(t)$ is growing exponential and if "σ" is negative, then the signal $x(t)$ is a decaying exponential.



(a) Decaying exponential ($\sigma < 0$) (b) Growing Exponential ($\sigma > 0$)

(5) COMPLEX EXPONENTIAL SIGNAL - It is defined as,

$$x(t) = Ae^{st}$$

where "s" is a complex variable and it is defined as,

$$s = \sigma + j\omega$$

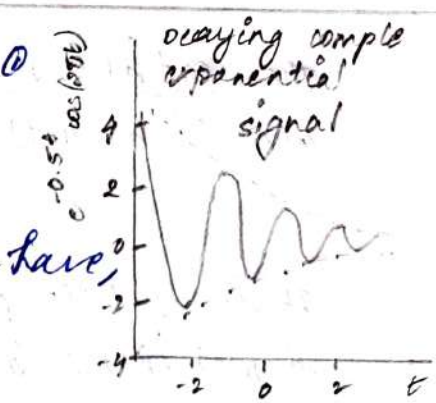
Therefore, $x(t) = e^{st} = e^{(\sigma + j\omega)t} = e^{\sigma t} e^{j\omega t}$ — (1)

Using Euler's identity,

$$e^{j\omega t} = e^{j\theta} = \cos\theta + j\sin\theta \quad \text{— (2)}$$

Substituting equation (2) in equation (1), we have,

$$x(t) = e^{\sigma t}[\cos\theta + j\sin\theta]$$



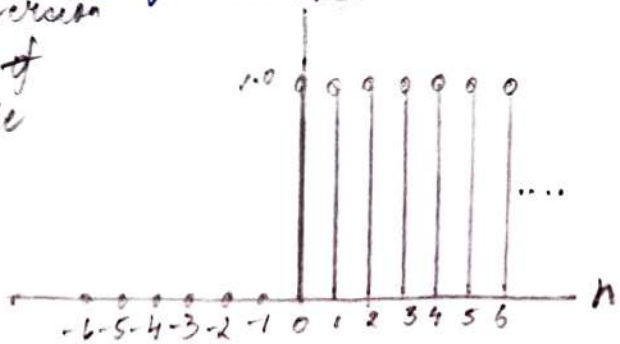
(6) CONTINUOUS-TIME AND DISCRETE-TIME COMPLEX EXPONENTIALS - In both cases, the complex exponential can be expressed through Euler's relation in the form of a real and an imaginary part, both of which are sinusoidal with a phase difference of $\pi/2$ and with an envelope that is a real exponential. When the magnitude of the complex exponential is a constant, then the real and imaginary parts neither grow nor decay with time; in other words, they are purely sinusoidal. In this case for continuous-time, the complex exponential is periodic. For discrete-time, the complex exponential may or may not be periodic depending on whether the sinusoidal real and imaginary components are periodic.

(7) UNIT STEP FUNCTION - It is the value of which is zero for negative arguments and one for positive arguments. It is an example of the general class of step functions, all of which can be represented as linear combinations of translation of this one.

The discrete-time version of the step function, commonly denoted by $u[n]$, is defined by -

$$u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

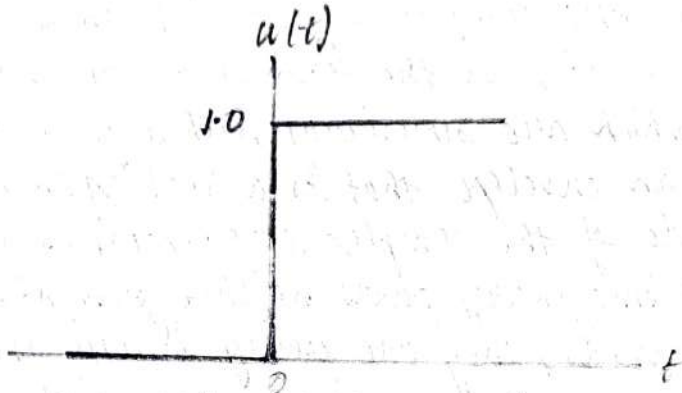
Discrete-time version of step function of unit amplitude



The continuous-time version of the step function, commonly denoted by $u(t)$, is defined by -

$$u(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

It is said to exhibit a discontinuity at $t=0$, since the value of $u(t)$ changes instantaneously from 0 to 1 when $t=0$.

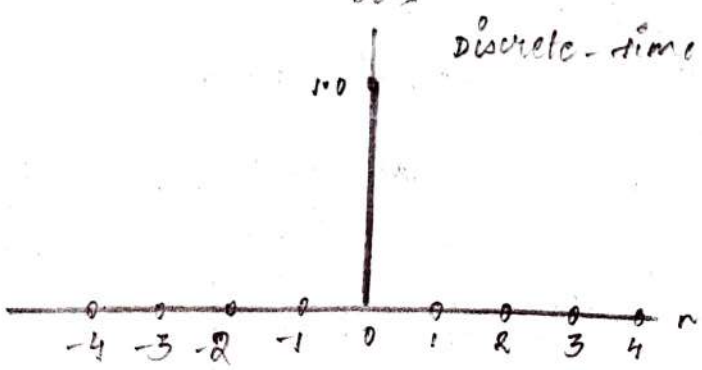


continuous-time version of step function of unit impulse

⑥ UNIT IMPULSE FUNCTION - It is a generalized function or distribution over the real numbers, whose value is zero everywhere except at zero, and whose integral over the entire real line is equal to one. It is used as a tool for the normalization of state vectors and also, has uses in probability theory and signal processing.

The discrete-time version of the impulse, commonly denoted by $\delta[n]$, is defined by -

$$\delta[n] = \begin{cases} 1, & n=0 \\ 0, & n \neq 0 \end{cases}$$

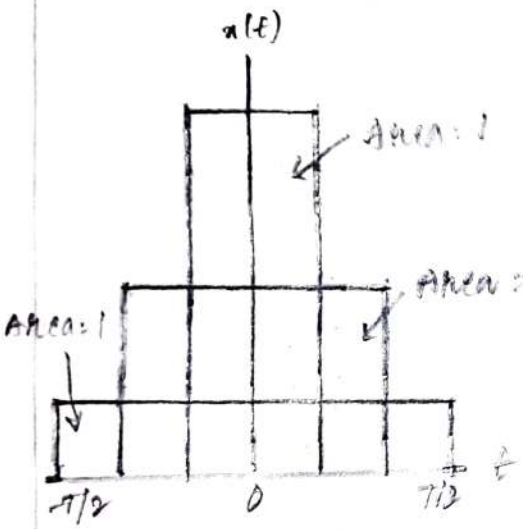


The continuous-time version of the unit impulse, commonly denoted by $\delta(t)$, is defined by the following pair of relations -

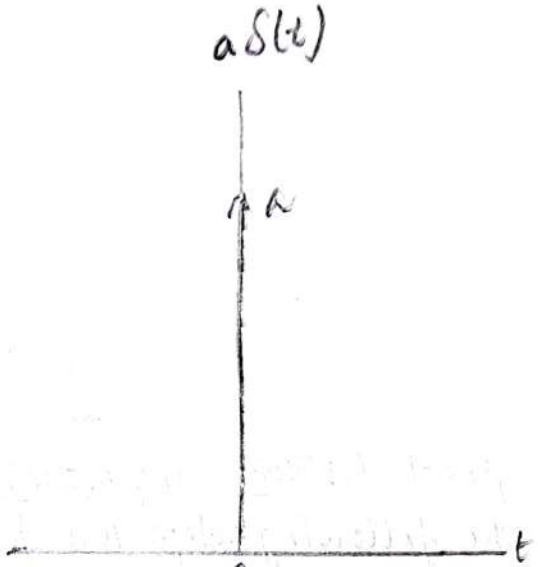
$$\delta(t) = 0 \quad \text{for } t \neq 0$$

and

$$\int_{-\infty}^{\infty} \delta(t)\, dt = 1$$



Evolution of a rectangular pulse of unit area into an impulse of unit strength

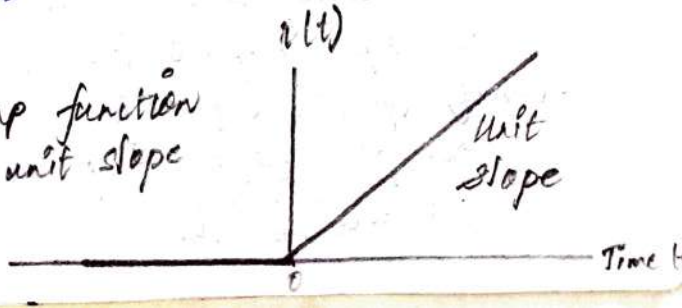


Graphical symbol for an impulse of strength a

⑦ RAMP FUNCTION - It is a unary real function, whose graph is shaped like a ramp. It can be expressed by numerous definitions for example, "0 for negative inputs, output equals input for non-negative inputs." The term "ramp" can also be used for other functions obtained by scaling and shifting, and the function is is the unit ramp function (slope 1, starting at 0).

The continuous-time version of the ramp function, commonly denoted by $r(t)$, is defined by -

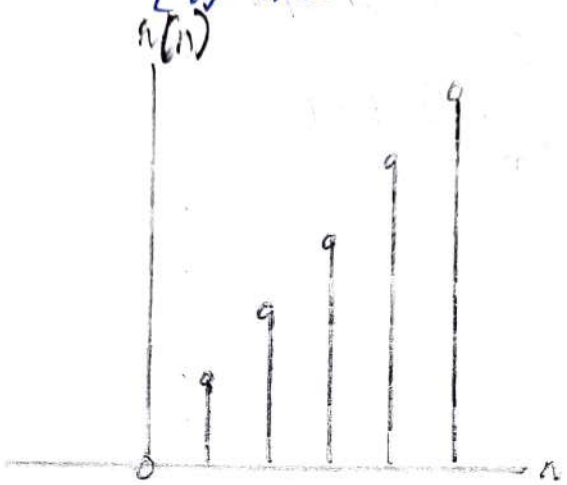
$$r(t) = \begin{cases} t, & t \geq 0 \\ 0, & t < 0 \end{cases}$$



ramp function of unit slope

The discrete-time version of the Impulse, commonly denoted by r[n], is defined by-

$$r[n] = \begin{cases} n, & n > 0 \\ 0, & n < 0 \end{cases}$$



Discrete-time version of the ramp function

+ GRAPH PLOTTING ALGORITHM:

The following steps are 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 sinusoidal_c(): # Sinusoidal continuous-Time Signal
    t = np.linspace(-0.02, 0.05, 1000) # linspace() creates
evenly spaced sequences
    plt.plot(t, 325*np.sin(2*np.pi*50*t))
    plt.xlabel('t')
```

```
    plt.ylabel ['x(t)']
    plt.title ['plot of CT signal : x(t) = 325*sin(100*pi*t)']
    plt.xlim ([-0.02, 0.05]) # set the x-limits of the current axes
    plt.show() # To view your plot

def sinusoidal_d(): # Sinusoidal Discrete-Time Signal
    n = np.arange(50) # Get evenly spaced values within a given interval
    dt = 0.07/50 # Fixes Signal Space Resolution
    x = 325*np.sin(2*np.pi*50*n*dt)
    plt.xlabel('n')
    plt.ylabel('x[n]')
    plt.title('plot of DT signal : x[n]= 325*sin(100*pi*n)')
    plt.stem(n, x) # 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 your plot.

def exponential_c(): # Exponential Continuous-Time Signal
    t = np.linspace(-0.02, 0.05, 1000) # linspace() creates evenly spaced sequencies
    plt.subplot(3,1,1) # subplot(rows, columns, index) describes the figure layout
    plt.plot(t, np.exp(2j*np.pi*50*t).real) # Return the real part of the complex argument
    plt.xlabel('t')
    plt.ylabel('Re x(t)')
    plt.title('Real part of x(t) = exp(j*100*pi*t)')
    plt.xlim([-0.02, 0.05]) # set the x-limits of the current axes
    plt.subplot(3,1,3) # subplot(rows, columns, index) describes the figure layout
    plt.plot(t, np.exp(2j*np.pi*50*t).imag) # Return the imaginary part of the complex argument
```

```
    plt.xlabel('t')
    plt.ylabel('Im x(t)')
    plt.title('Imaginary part of x(t) = exp(j*100*pi*t)')
    plt.xlim([-0.02, 0.05]) # Set the x-limits of the current ax
    plt.show() # To view your plot

def exponential_d(): # Exponential Discrete-Time Signal
    n = np.arange(-2, 5) # Get evenly spaced values within a
given interval
    arr=[] # Null Array Declaration
    for sample in range(len(n)): # Returns the length of the arr
        if n[sample] < 0:
            temp = 1/(2**(-(n[sample]))) # conversion of negativ
power to fractions (2^-1 = 1/2)
        else:
            temp = (2**(n[sample]))
        arr.append(temp) # Adds a single item to the
existing list
    plt.xlabel('n')
    plt.ylabel('x[n]')
    plt.title('Plot of DT signal : x[n] = 2^n')
    plt.stem(n, arr) # 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 your plot

def step_c(): # Unit Step Continuous-Time Signal
    t= np.linspace(-2, 10, 1000) # linspace() creates evenly
spaced sequences
    arr=[] # Null Array Declaration
    for sample in range(len(t)): # Returns the length of array
        if t[sample] >= 0:
            temp =1
```

```
    else
     temp=0
    arr.append(temp) # Adds a single item to the existing list
  plt.xlabel('t')
  plt.ylabel('x(t)')
  plt.title('Unit Step Continuous-Time Function')
  plt.step(t,arr) # line forming a series of steps between
data points
  plt.show() # To view your plot

def step_d(): # Unit Step Discrete-Time Signal
    n=np.arrange[-2,10) # Get evenly spaced values within
a given interval
    arr=[] # Null Array Declaration
    for sample in range(len(n)): # Returns the length of array
        if n[sample]>=0:
          temp=1
        else
          temp=0
        arr.append(temp) # Adds a single item to the
existing list
    plt.xlabel('n')
    plt.ylabel('x(n)')
    plt.title('Unit Step Discrete-Time Function for 0≤n≤9')
    plt.stem(n, arr) # stem plot plots vertical lines at each
x position under the graph from the baseline to y, and places
a marker there.
    plt.show() # To view your plot

def impulse_c(): # Unit Impulse Continuous-Time Signal
    t=np.arrange(-2,10) # Get evenly spaced values within a given
interval
    arr=[] # Null Array Declaration
```

```
for sample in range (len(t)): # Returns the length of array
    if t[sample] >=0 :
      temp =1
    else
      temp =0
    arr. append (temp) # Adds a single item to the existing list
  plt. xlabel ('t')
  plt. ylabel ('x(t)')
  plt. title ('Unit Impulse Continuous-Time Function')
  plt. plot (t, arr) # Line forming a series of steps within a
given interval
  plt. show () To view your plot

def impulse_d() : # Unit Impulse Discrete-Time signal
    n= np. arrange (-2, 10) # Get evenly spaced values within a
given interval
    arr = [] # Null Array Declaration
    for sample in range (len(n)) : # Returns the length of array
      if n[sample] == 0:
        temp = 1
      else
        temp = 0
      arr. append (temp) # Adds a single item to the existing li
  plt. xlabel ('n')
  plt. ylabel ('x(n)')
  plt. title ('Unit Impulse Discrete-Time Function for 0≤n≤9')
  plt. stem (n, arr) # stem plot plots vertical lines at each
position covered under the graph from the baseline to y. and
places a marker there.
  plt. show () # To view your plot
```

```
def ramp_c(): # Ramp Function Continuous-Time Signal
    t = np.linspace(0, 10, 1000) # linspace() creates evenly spaced sequences
    plt.plot(t, t)
    plt.xlabel('t')
    plt.ylabel('x(t)')
    plt.title('Ramp Continuous-Time Function')
    plt.xlim([0, 10]) # Set the x-limits of the current axes
    plt.show() # To view your plot

def ramp_d(): # Ramp Function Discrete-Time Signal
    n = np.arange(0, 10) # linspace() creates evenly spaced sequences
    plt.xlabel('n')
    plt.ylabel('n(t)')
    plt.title('Ramp Discrete-Time Function for 0≤n≤9')
    plt.stem(n, n) # 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 your plot

# Main
while True: # This simulates a Do Loop
    choice = input(
        "MENU: \n 1. Sinusoidal Continuous-Time Signal. \n 2. Sinusoidal Discrete-Time Signal. \n 3. Exponential Continuous-Time Signal. \n 4. Exponential Discrete-Time signal. \n 5. Unit step Continuous-Time Signal \n 6. Unit Step Discrete-Step Signal. \n 7. Unit Impulse Continuous-Time Signal. \n 8. Unit Impulse Discrete-Time Signal. \n 9. Ramp Function Continuous-Time Signal. \n 10. Ramp Function Discrete-Time Signal. \n 11. Exit \n Enter the number corresponding to the menu to implement the choice: ") # Menu Driven Implementation
```

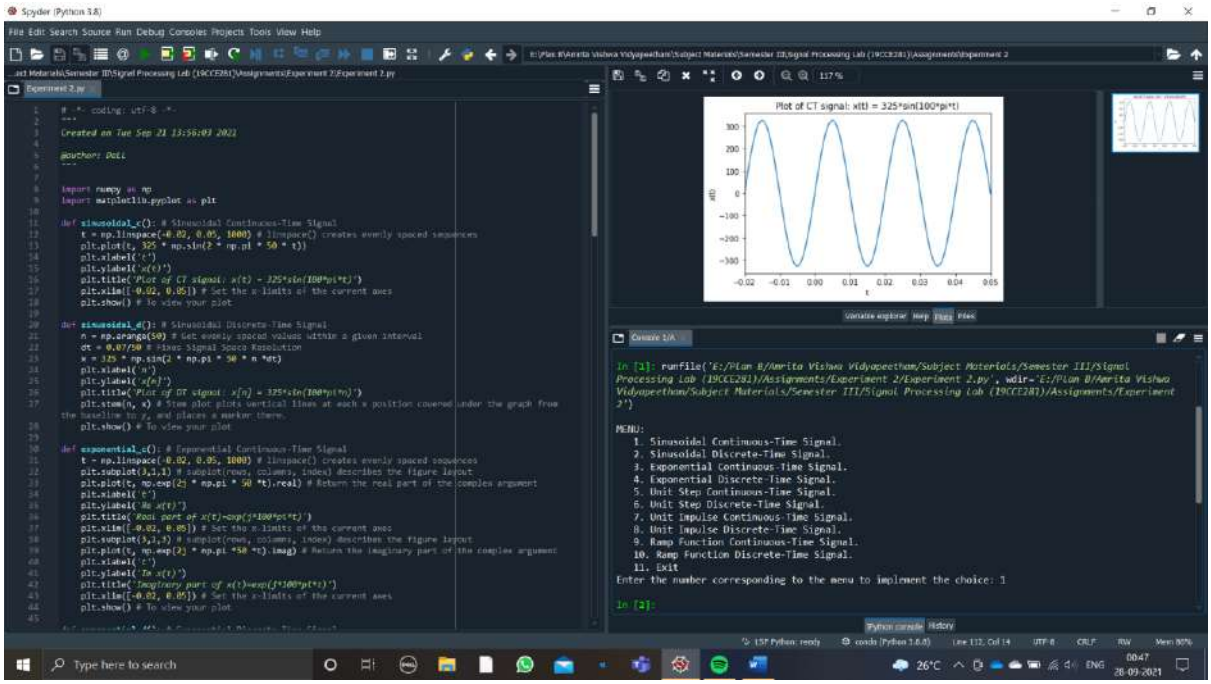
```
if choice == str(1): # str() returns the string version of
the variable "choice"
    sinusoidal_c() # Sinusoidal Continuous-Time Signal
    break
elif choice == str(2):
    sinusoidal_d() # Sinusoidal Discrete-Time signal
    break
elif choice == str(3):
    exponential_c() # Exponential Continuous-Time signal
    break
elif choice == str(4):
    exponential_d() # Exponential Discrete-Time Signal
    break
elif choice == str(5):
    step_c() # Unit Step Continuous-Time Signal
    break
elif choice == str(6):
    step_d() # Unit step Discrete-Time Signal
    break
elif choice == str(7):
    impulse_c() # Unit Impulse Continuous-Time Signal
    break
elif choice == str(8):
    impulse_d() # Unit Impulse Discrete-Time Signal
    break
elif choice == str(9):
    ramp_c() # Ramp Function Continuous-Time Signal
    break
elif choice == str(10):
    ramp_d() # Ramp Function Discrete-Time signal
    break
elif choice == str(11):
    break # Exit Loop
```

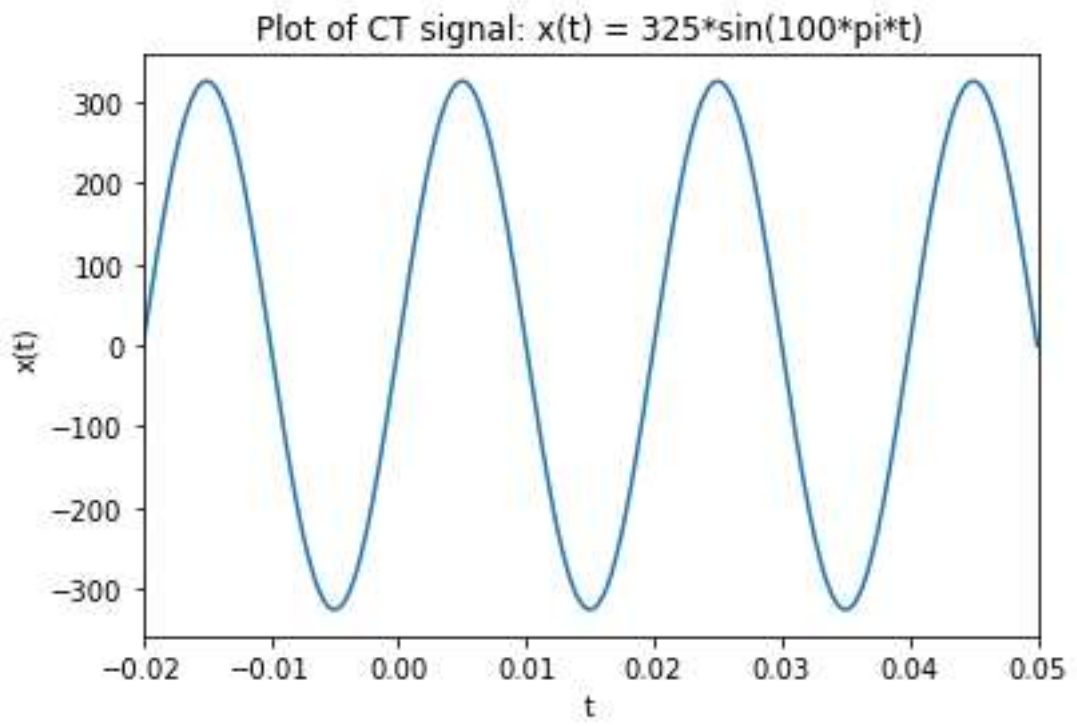
```
else :
    print ("Error : Invalid Input! Please try again. \n")
```

+ INFERENCE :

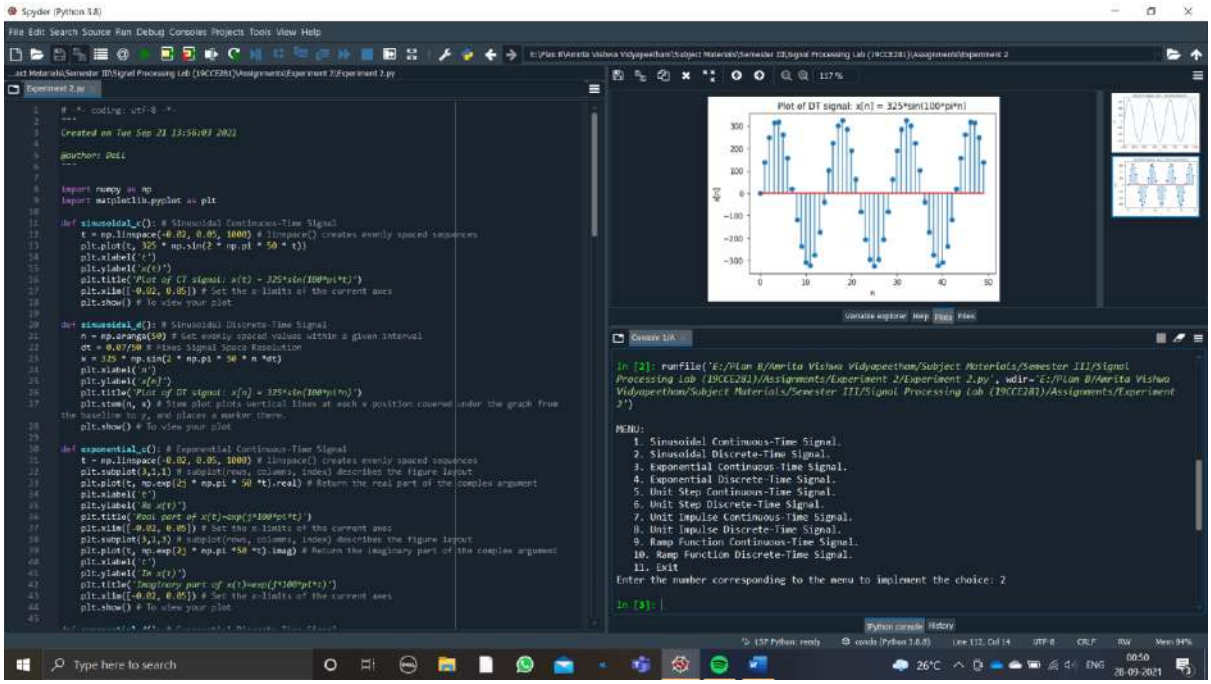
Visualize the complex exponential signal and real sinusoids and demonstrate other simple elementary signals and results verified.

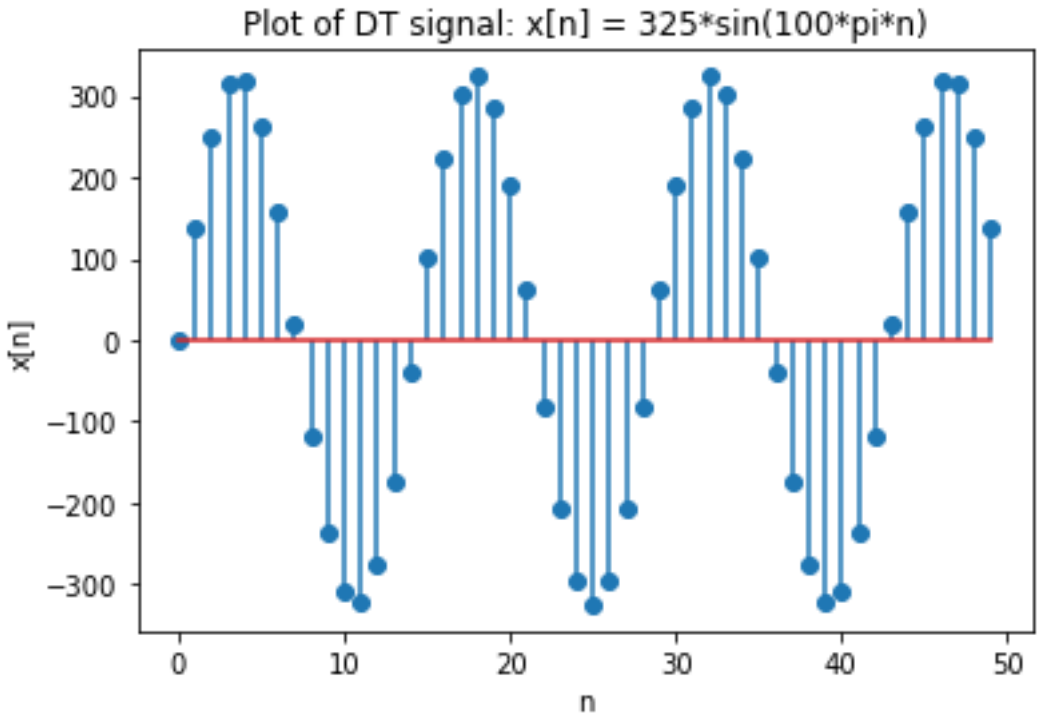
Sinusoidal Continuous-Time Signal



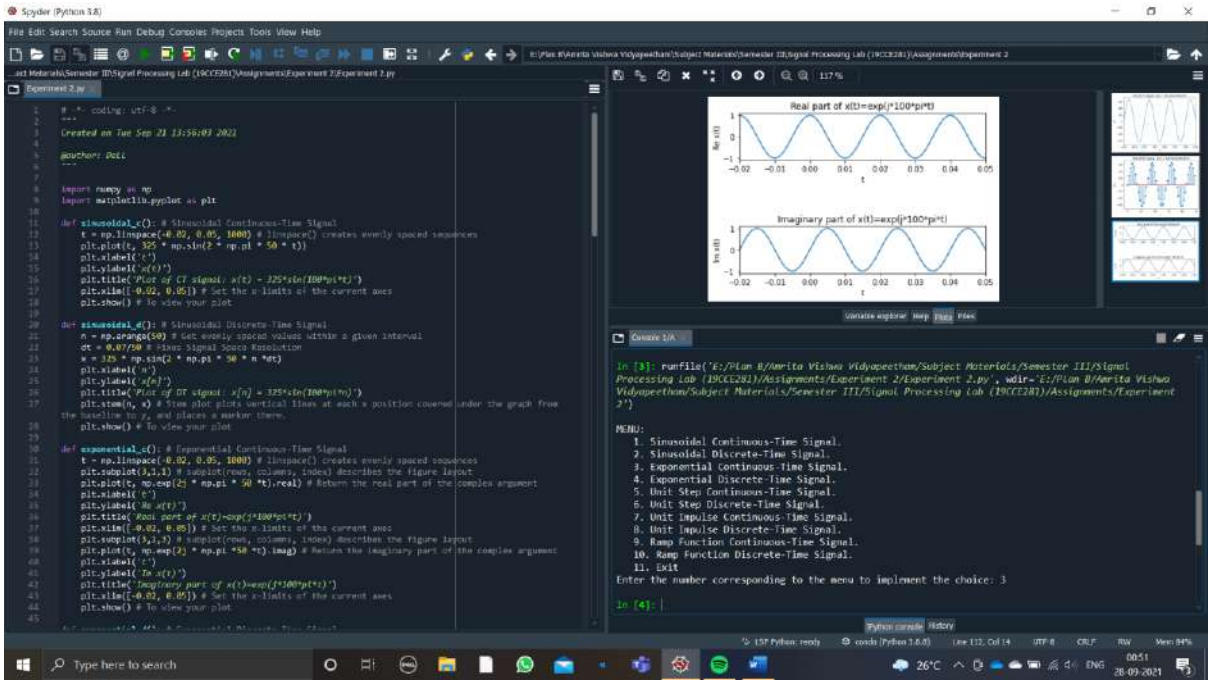


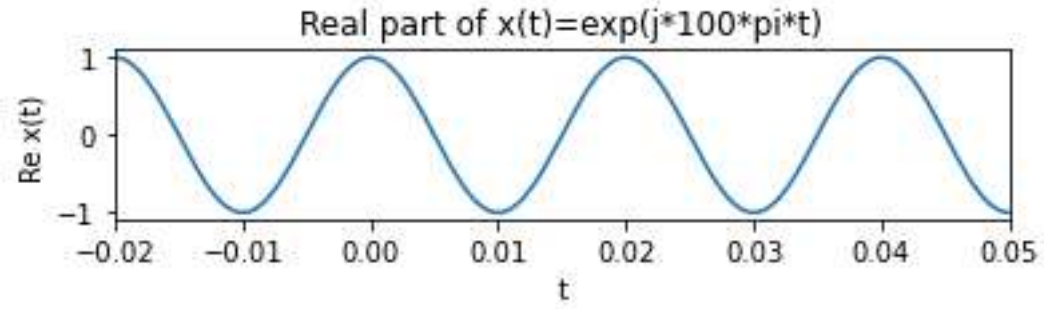
Sinusoidal Discrete-Time Signal

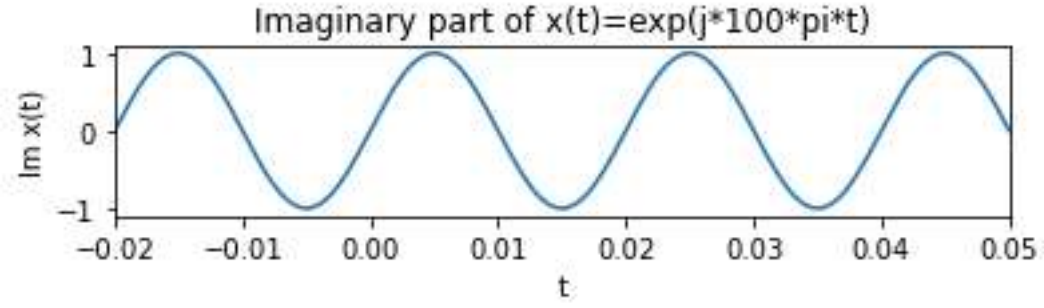




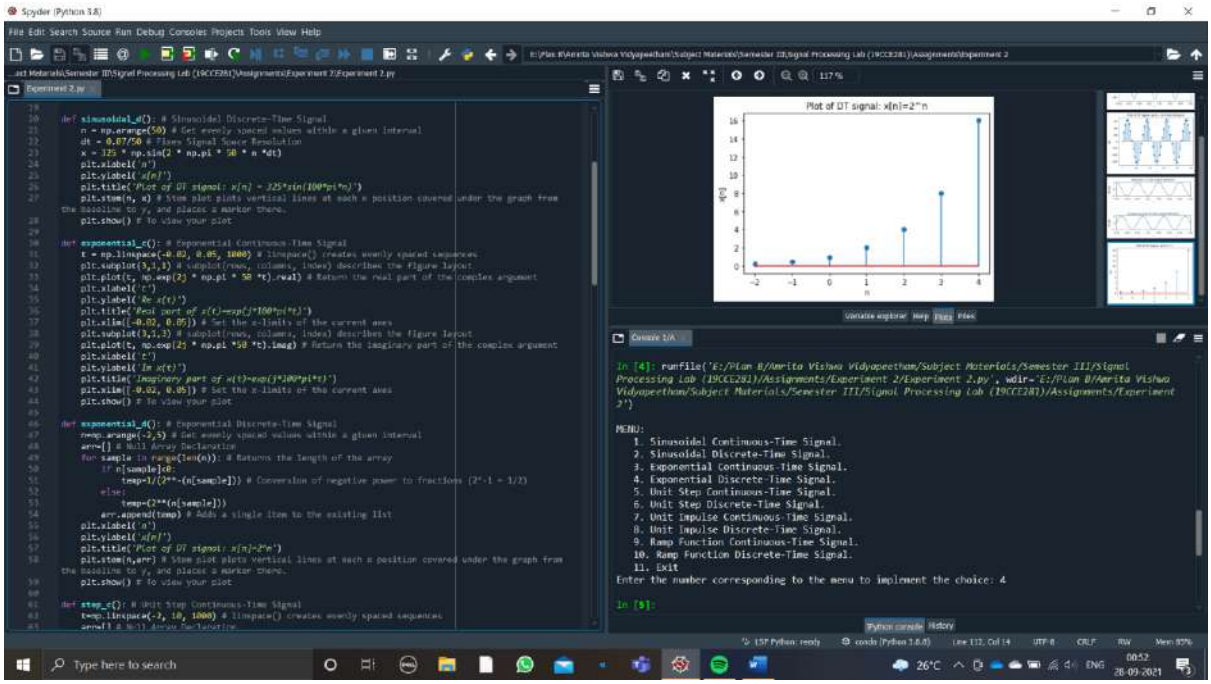
Exponential Continuous-Time Signal

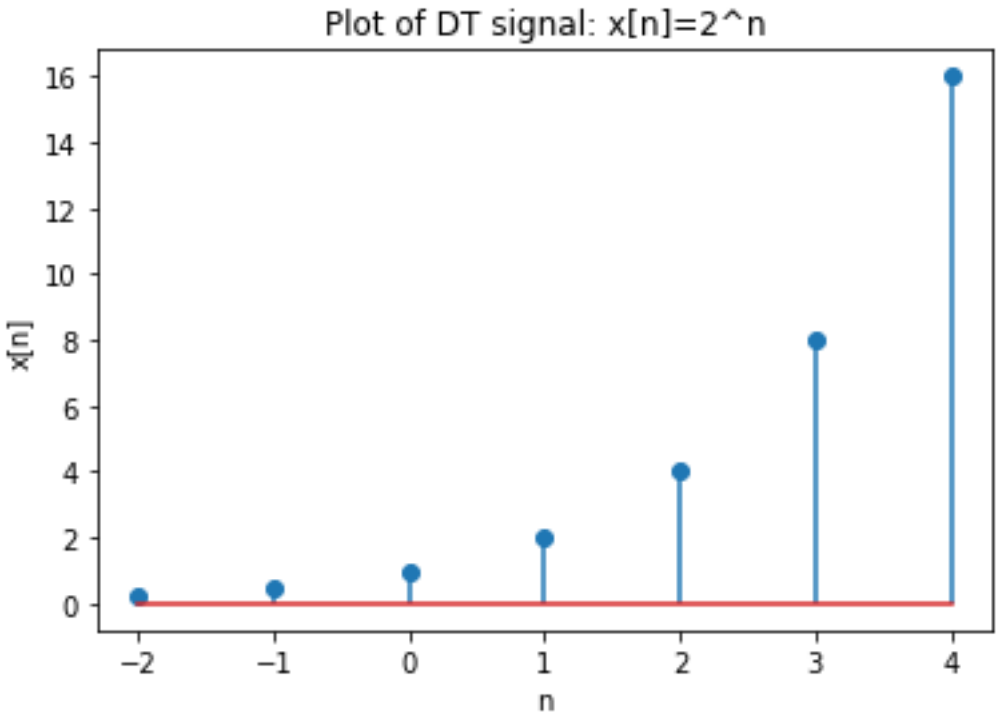




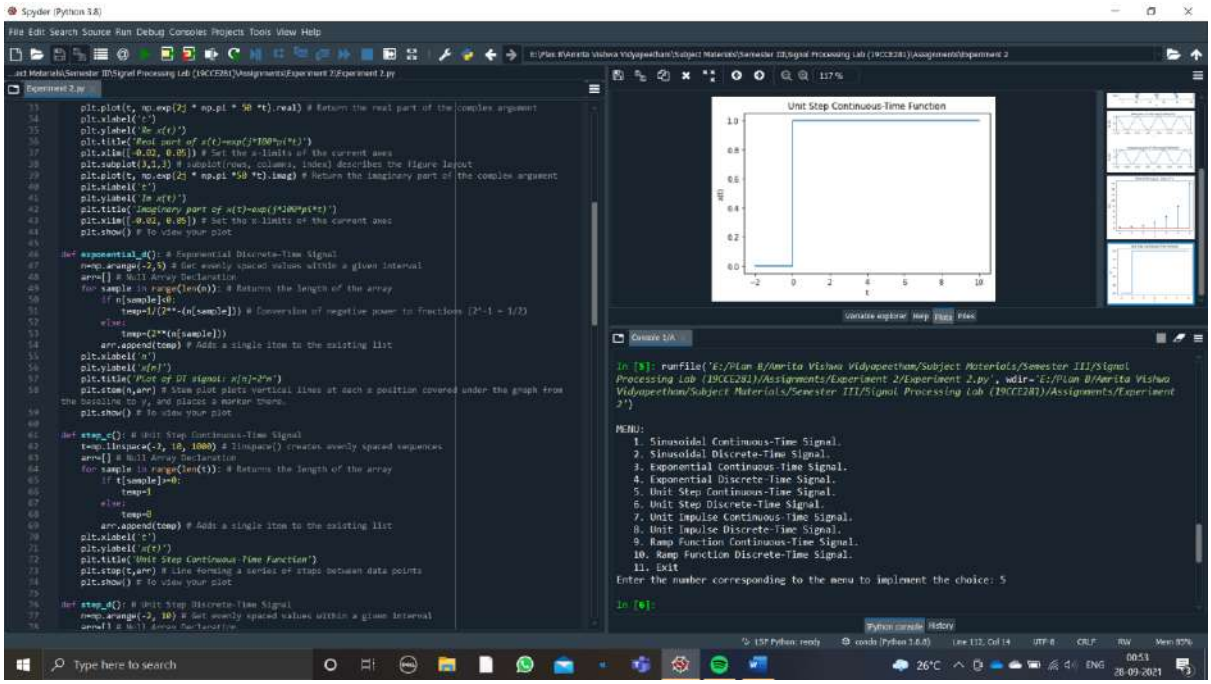


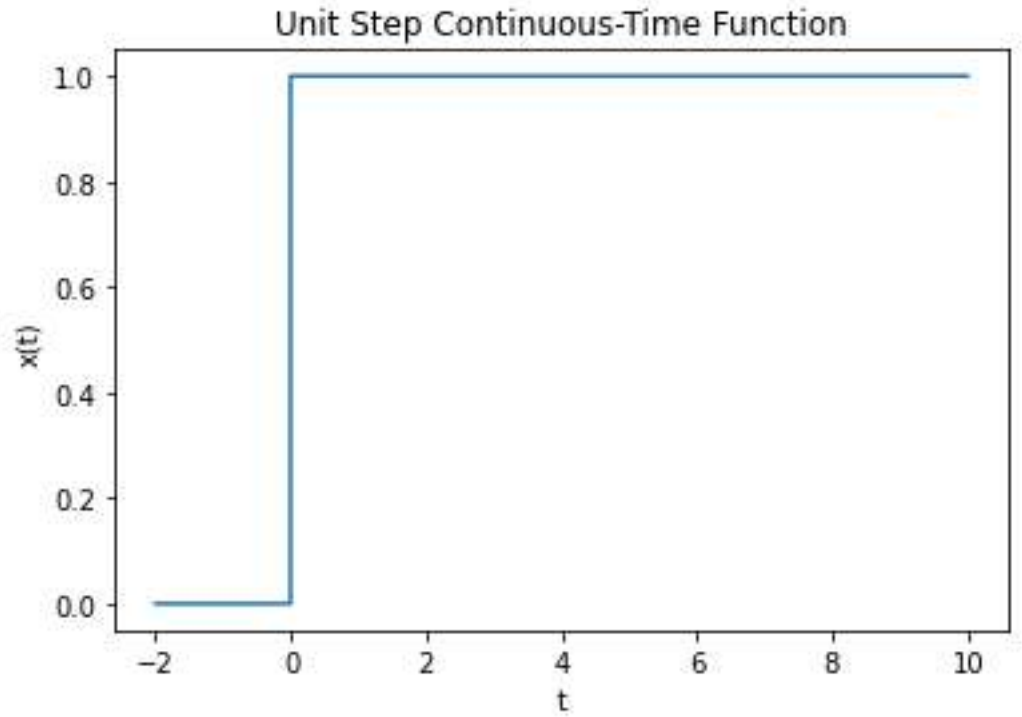
Exponential Discrete-Time Signal



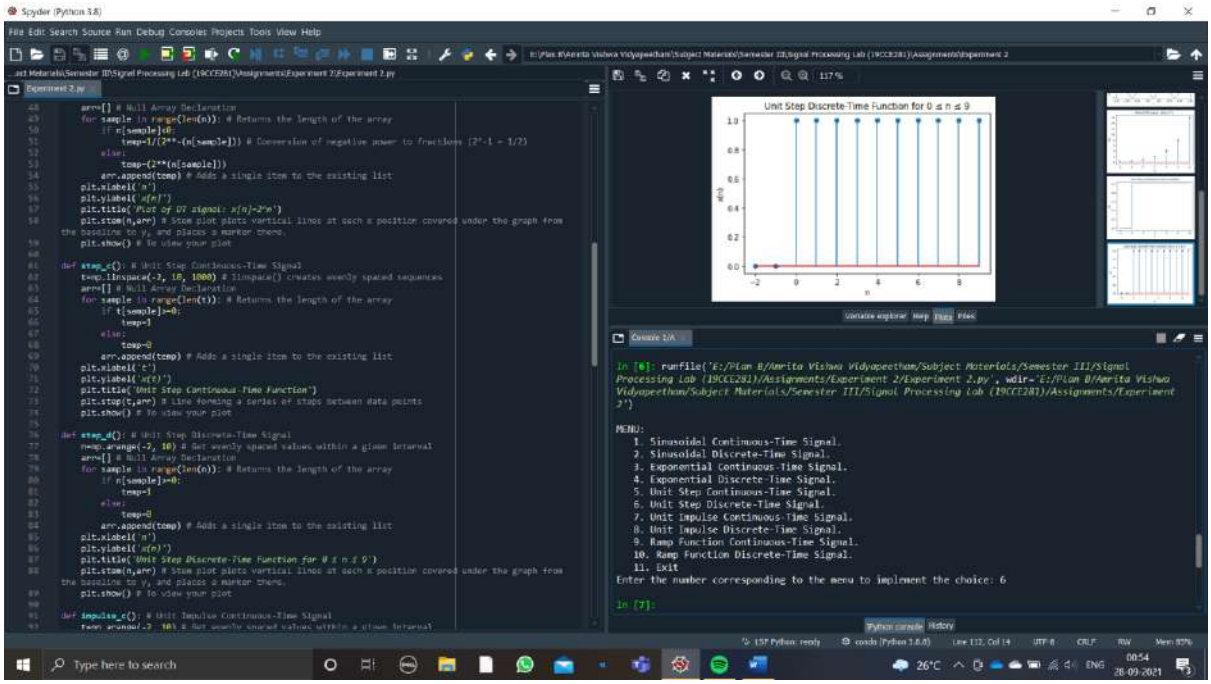


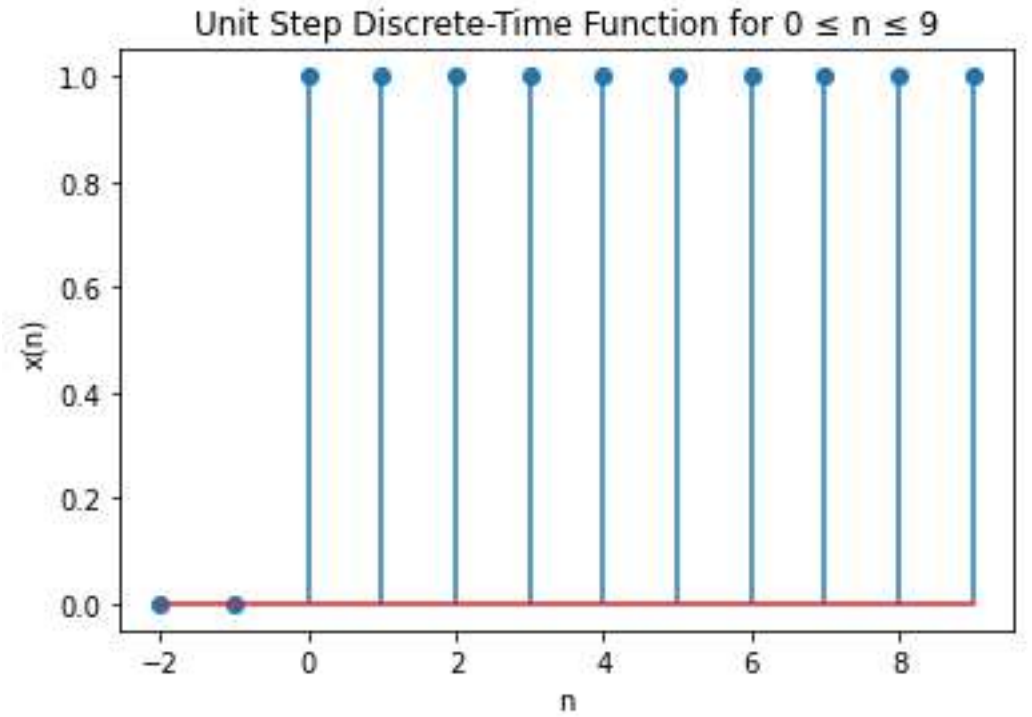
Unit Step Continuous-Time Signal



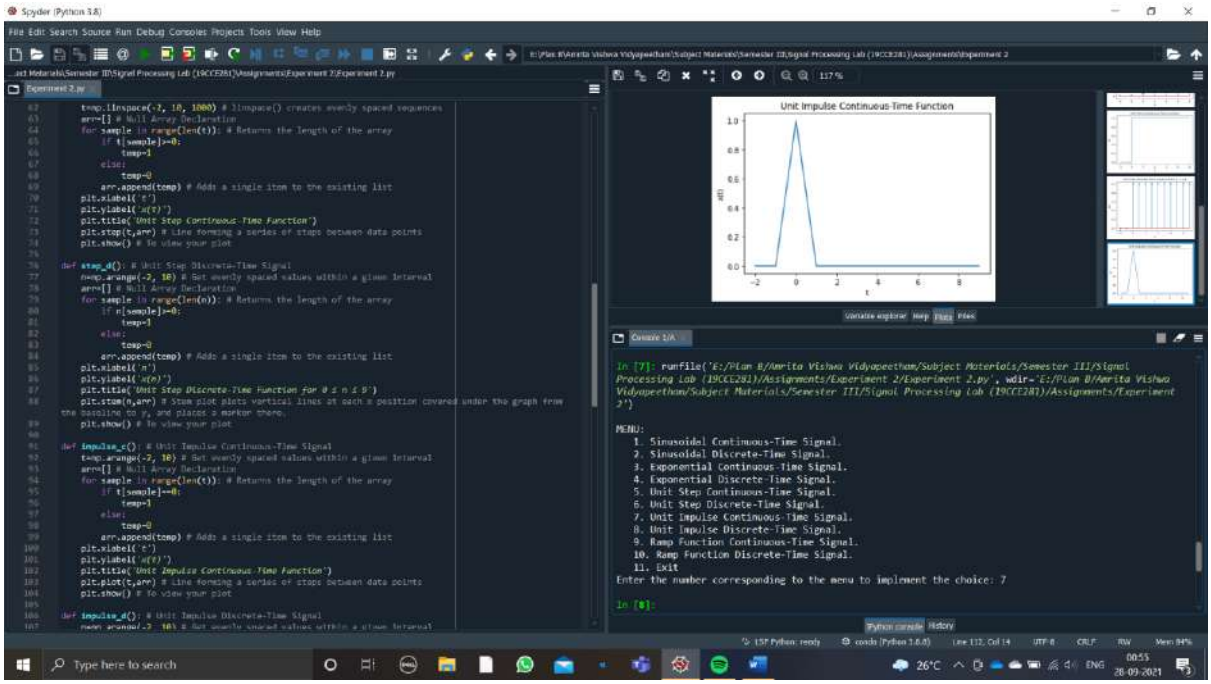


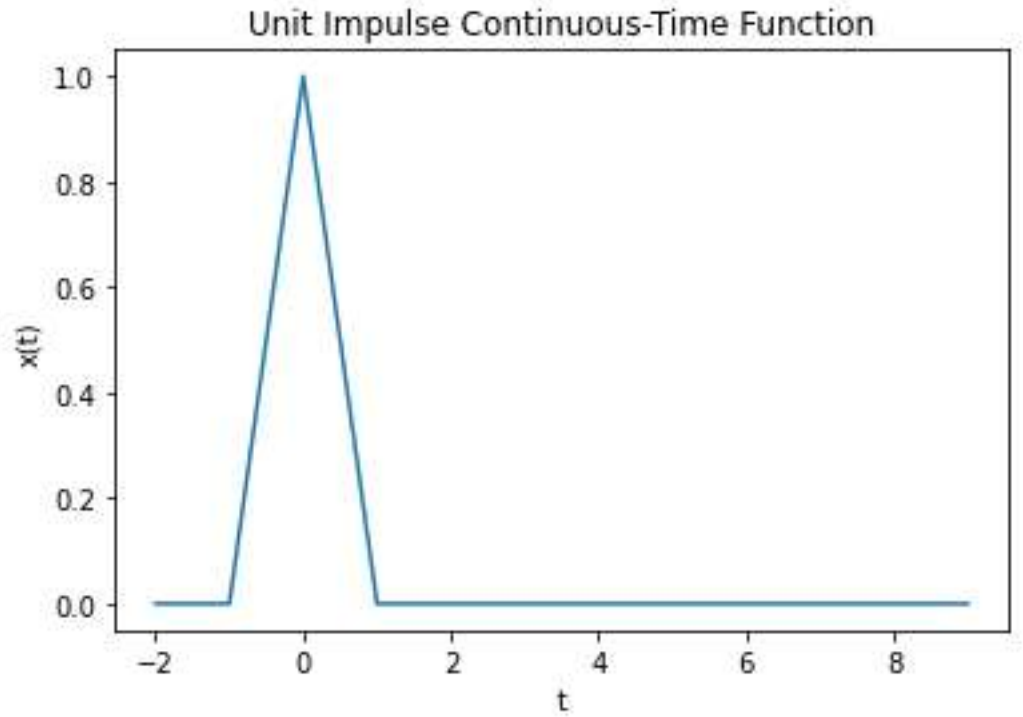
Unit Step Discrete-Time Signal



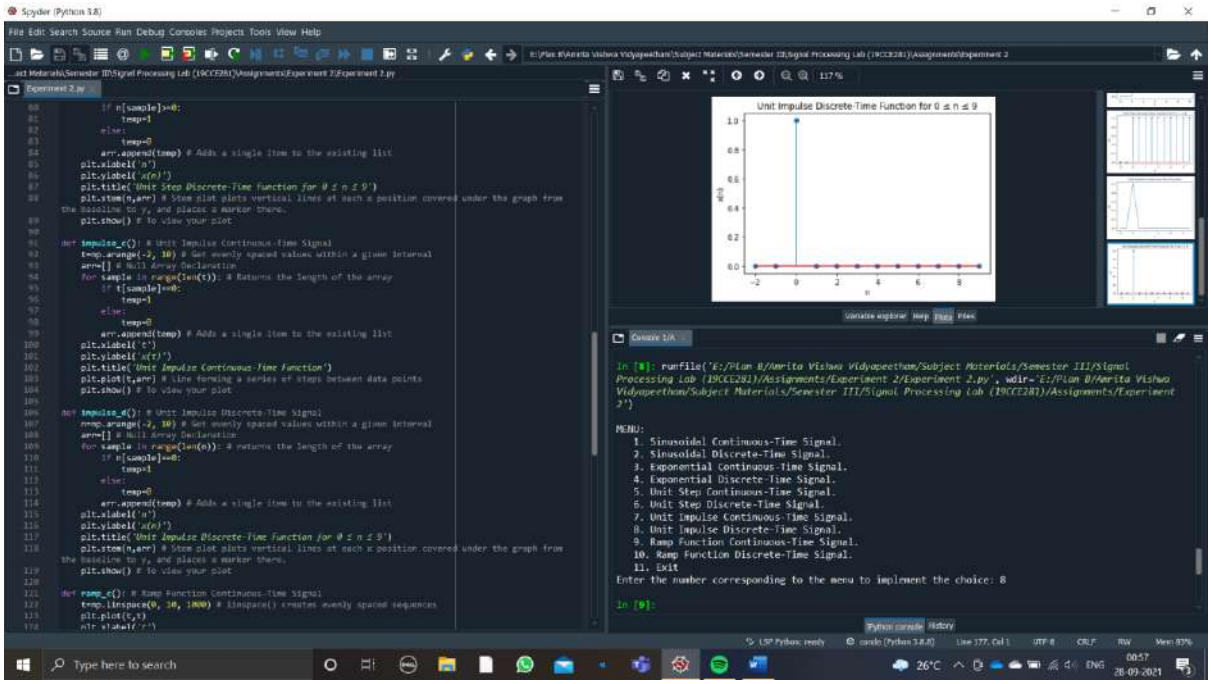


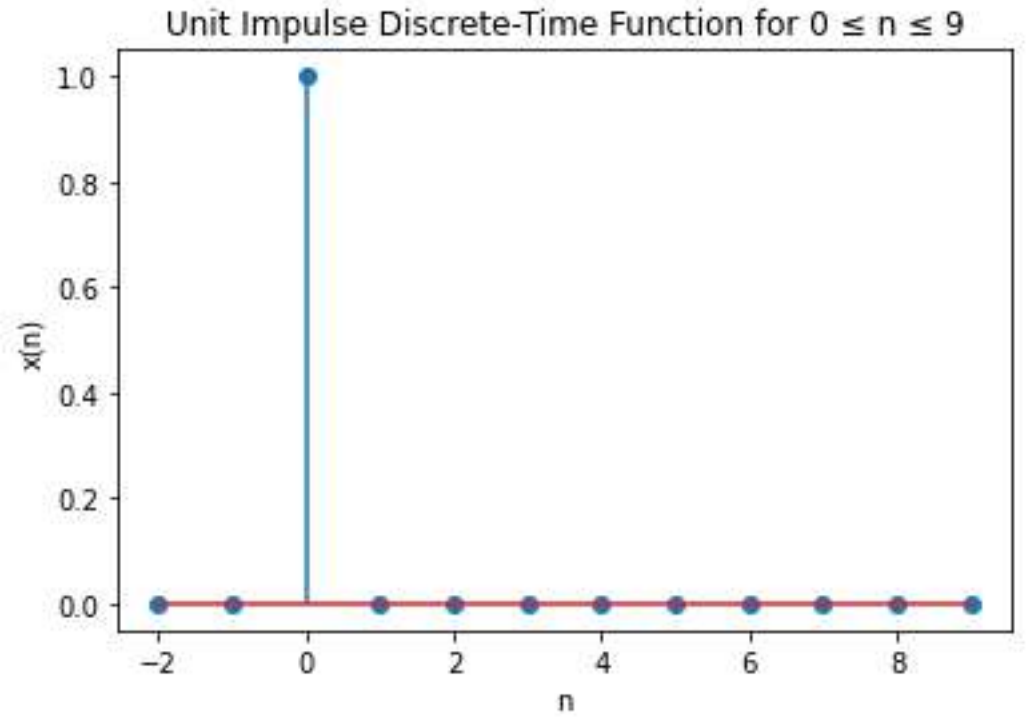
Unit Impulse Continuous-Time Signal



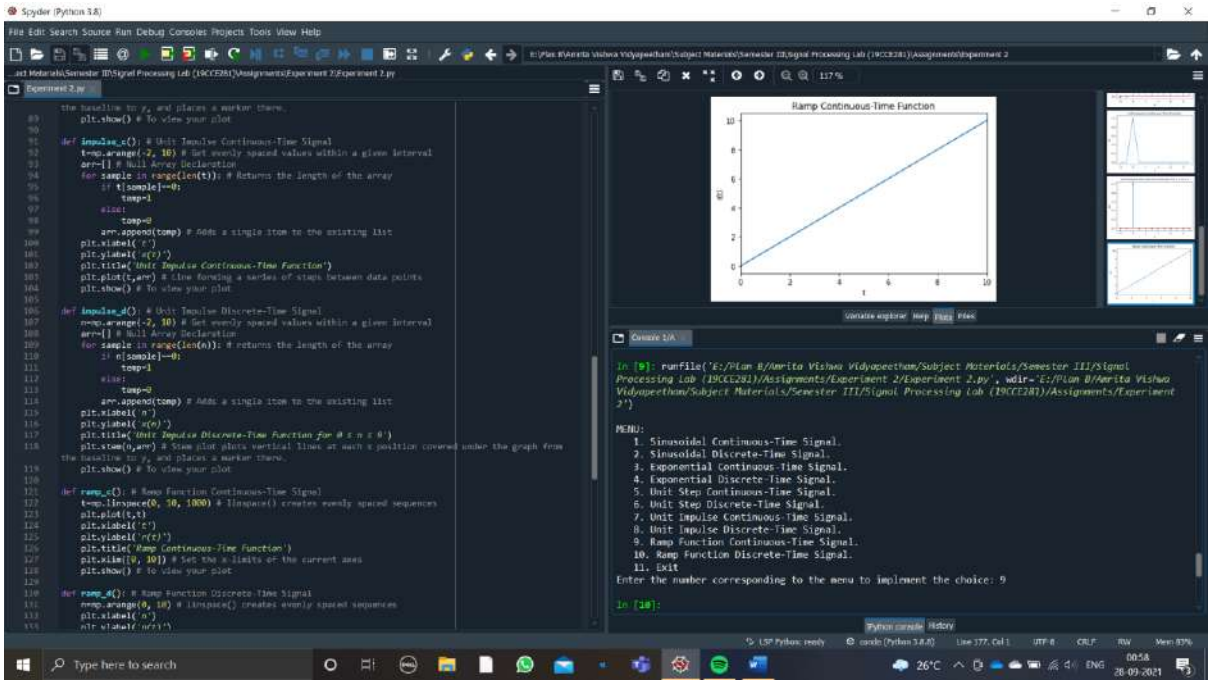


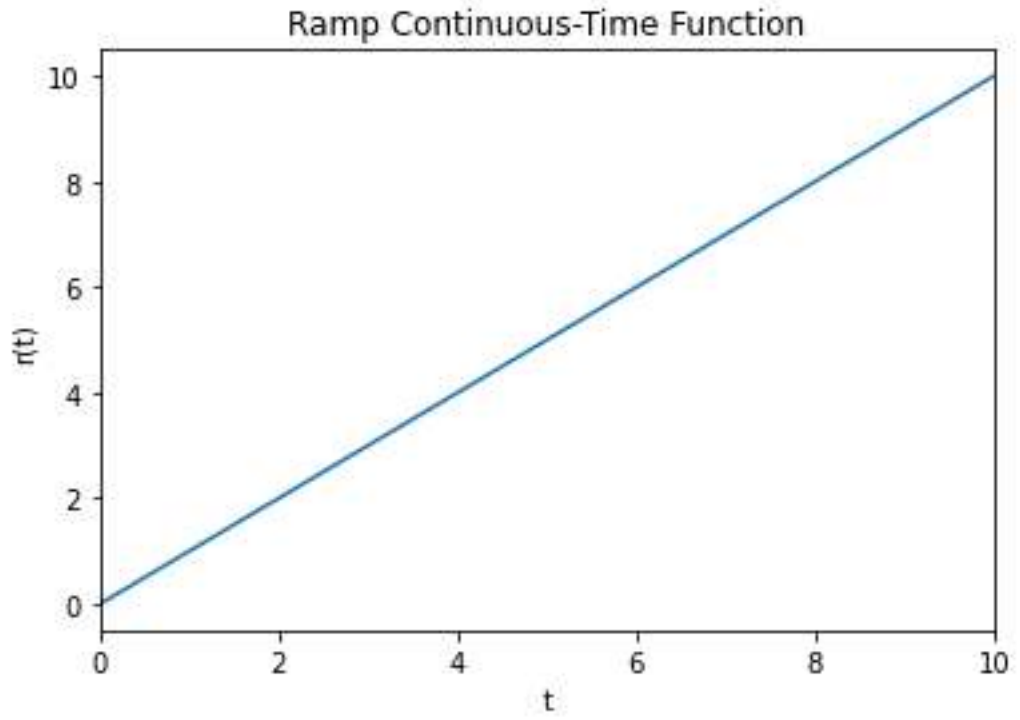
Unit Impulse Discrete-Time Signal



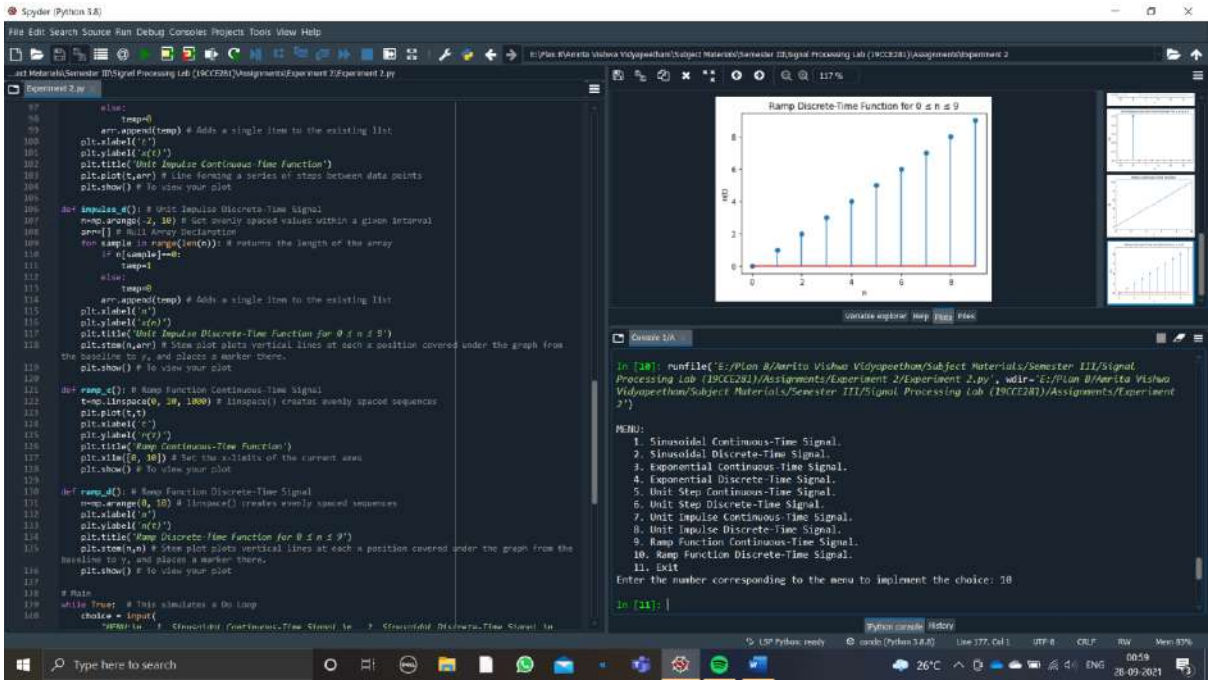


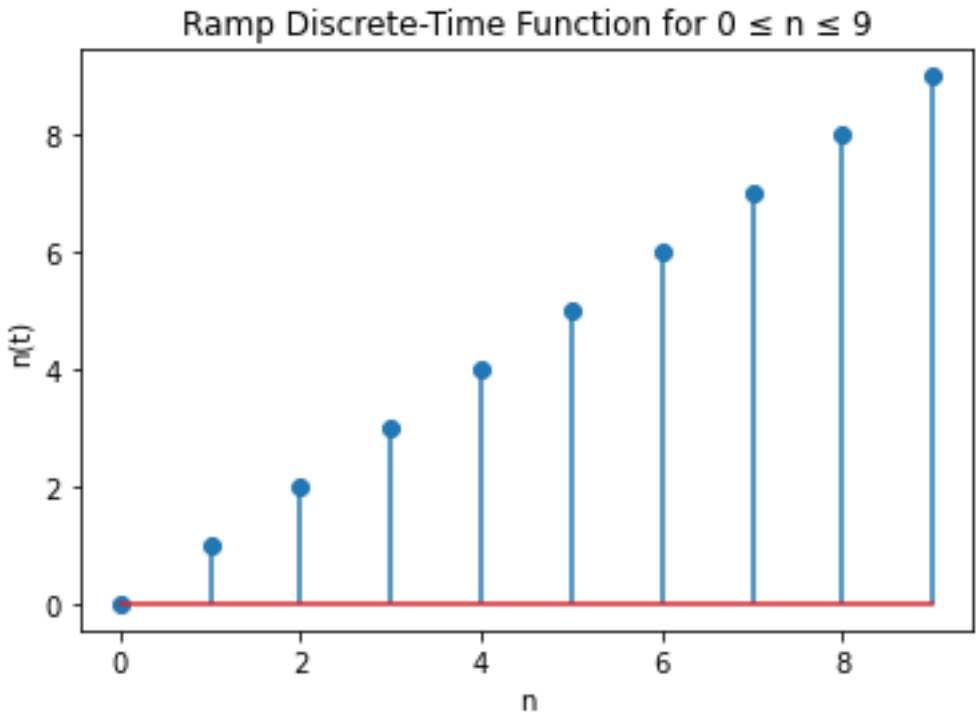
Ramp Function Continuous-Time Signal



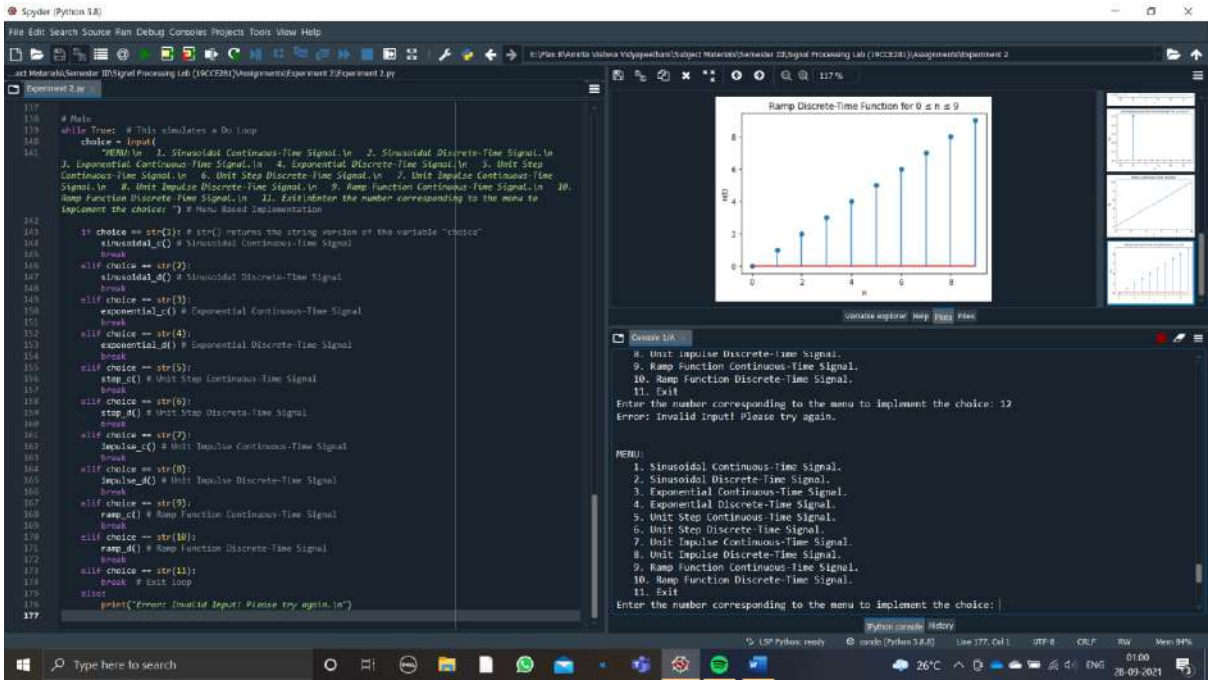


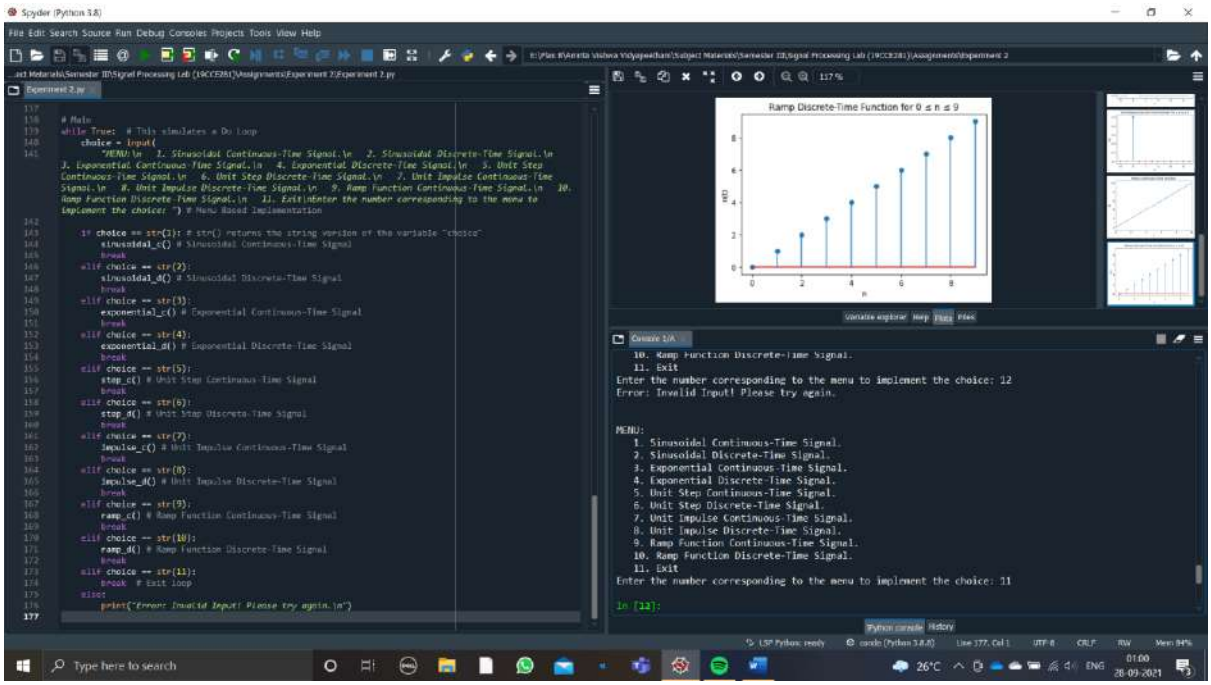
Ramp Function Discrete-Time Signal





Exit





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 2/Experiment 2.py'      ='E:/Plan
B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab
(19CCE281)/Assignments/Experiment 2'

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
Enter the number corresponding to the menu to implement the choice: 1

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
Enter the number corresponding to the menu to implement the choice: 2

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
Enter the number corresponding to the menu to implement the choice: 3
```

```
MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
Enter the number corresponding to the menu to implement the choice: 4

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
Enter the number corresponding to the menu to implement the choice: 5

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
Enter the number corresponding to the menu to implement the choice: 6

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
```

```
Enter the number corresponding to the menu to implement the choice: 7

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
Enter the number corresponding to the menu to implement the choice: 8

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
Enter the number corresponding to the menu to implement the choice: 9

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
Enter the number corresponding to the menu to implement the choice: 10

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
```

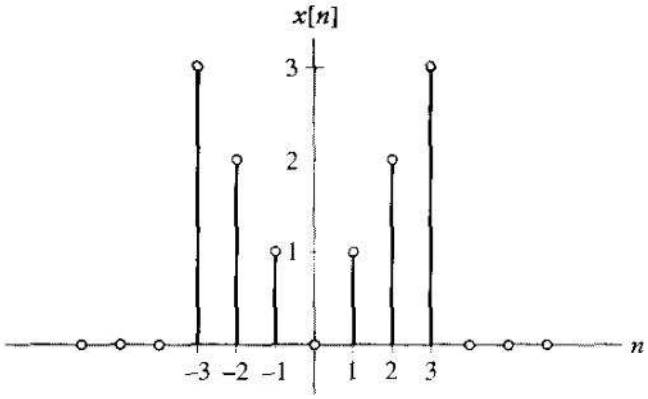
```
   11. Exit
Enter the number corresponding to the menu to implement the choice: 12
Error: Invalid Input! Please try again.

MENU:
   1. Sinusoidal Continuous-Time Signal.
   2. Sinusoidal Discrete-Time Signal.
   3. Exponential Continuous-Time Signal.
   4. Exponential Discrete-Time Signal.
   5. Unit Step Continuous-Time Signal.
   6. Unit Step Discrete-Time Signal.
   7. Unit Impulse Continuous-Time Signal.
   8. Unit Impulse Discrete-Time Signal.
   9. Ramp Function Continuous-Time Signal.
   10. Ramp Function Discrete-Time Signal.
   11. Exit
Enter the number corresponding to the menu to implement the choice: 11

In [2]:
```

Expt 3: Operations on Signals

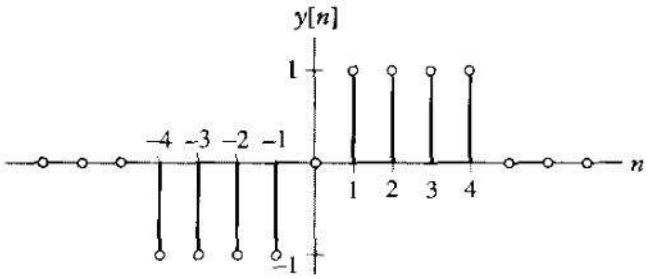
1)Generate and visualize the unit-step u[n] with the following modification in the time axis: u[n] - u[n - 2] for the time duration $0 \leq n \leq 9$; .

2)Generate and visualize the signal x[n] = [2 3 4 5 0] in terms of impulse functions.

3)Generate x[n] and y[n] as given in figure below and sketch the following signals
(i) x[3n - 1] (ii) x[n – 2] + y[n + 2]



(a)



(b)

NAME - SANTOSH (LB.ENV.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.

① Amplitude Scaling of Signals -

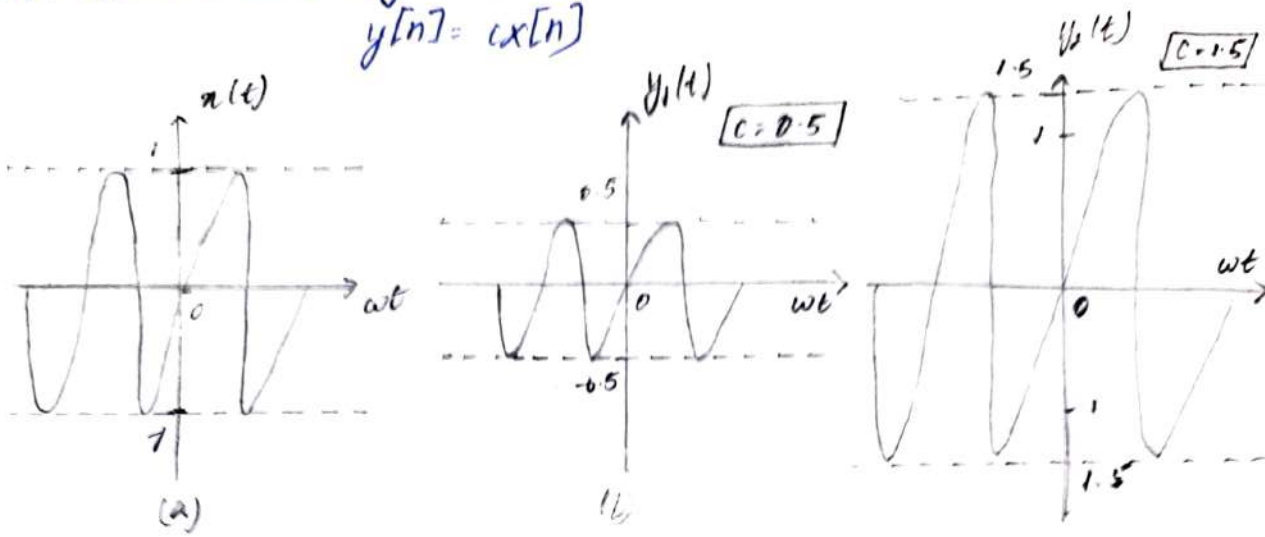
Amplitude scaling is a very basic operation performed on signals to vary its strength. It can be mathematically represented as -

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

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 -

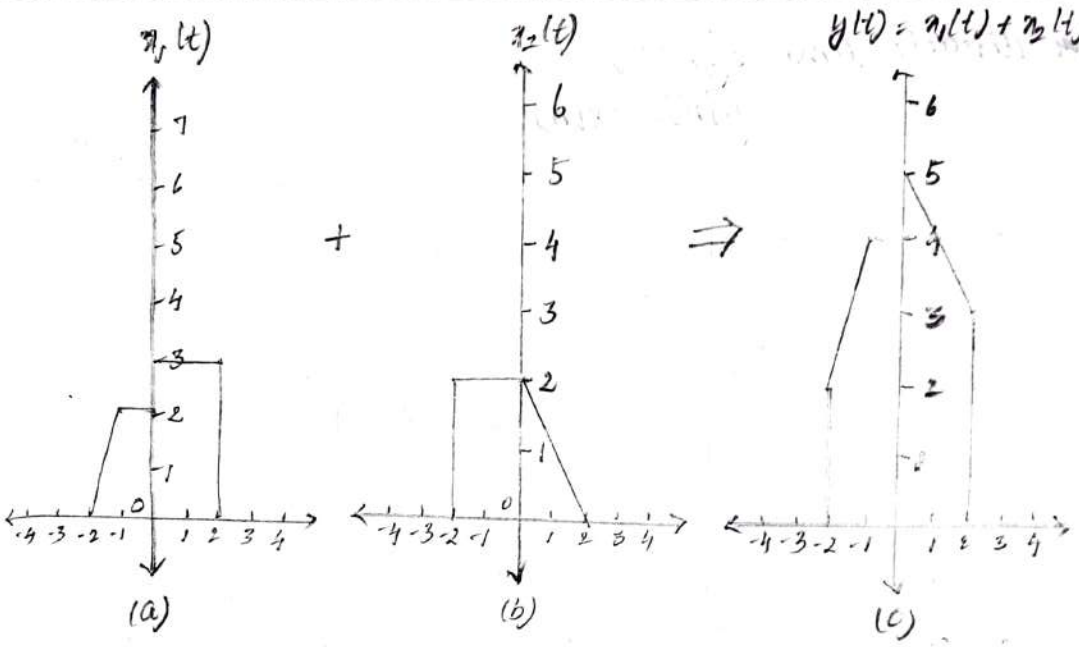
This particular operation involves the addition of amplitude of two or more signals at each instance of time or any other independent variables which are common between the signals. Addition of signals is illustrated in the diagram below, where $x_1(t)$ and $x_2(t)$ are two time dependent signals, performing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

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

For discrete time signals,

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

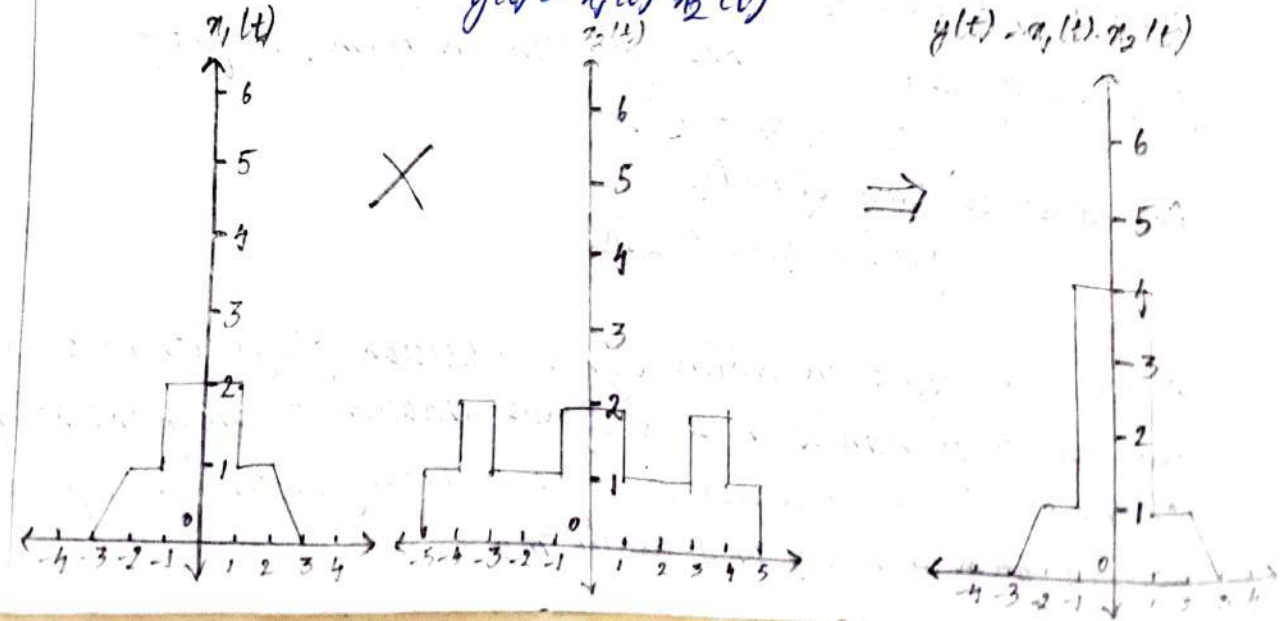
A practical aspect in which signal addition plays its role is in the case of transmission of a signal through a communication channel. Other examples include dithering (intentionally applied form of noise) and audio mixers.



③ 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 -

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

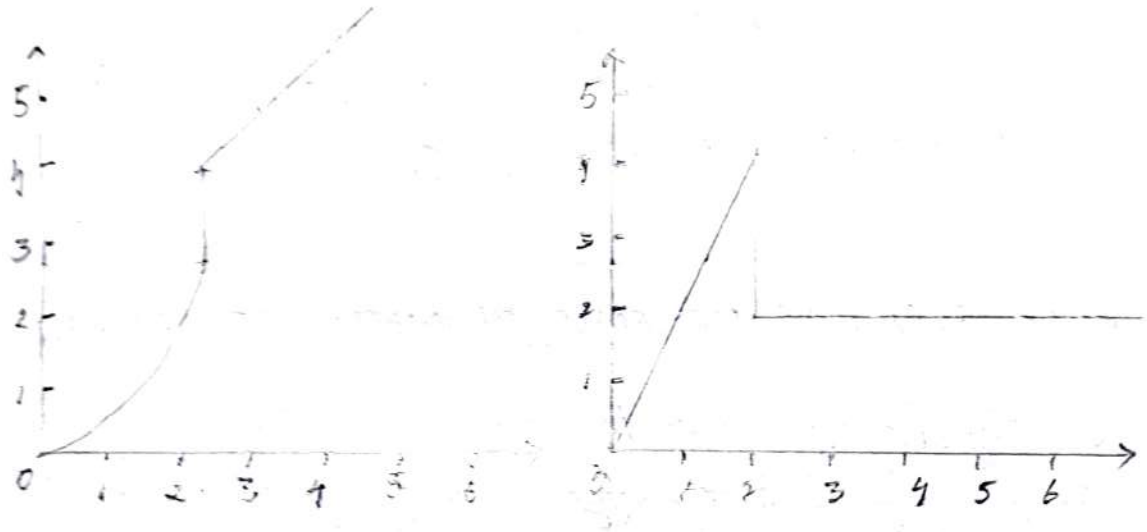


(4)

(a) 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 -

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

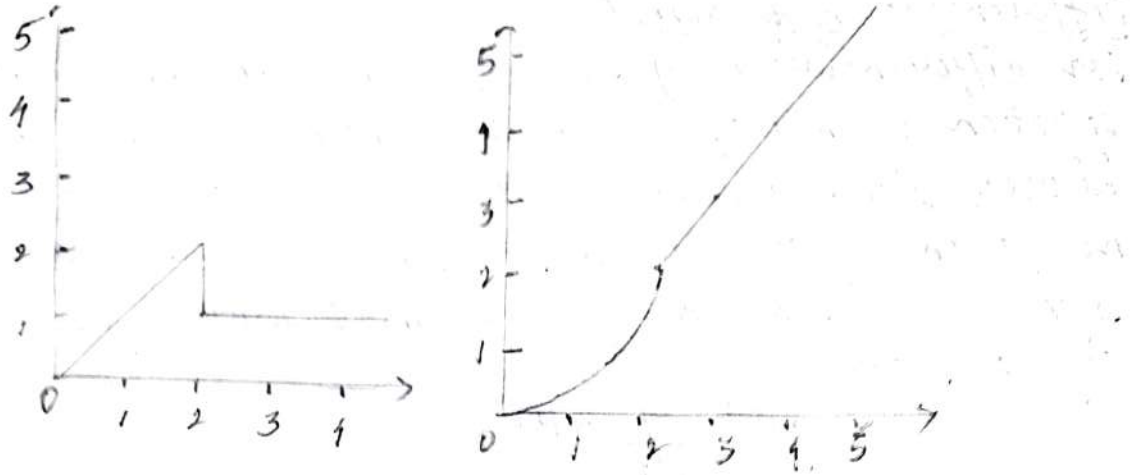


Differentiation of a signal takes the form of the gradient operator in the field of image or video processing. In the case of image processing, the gradient technique is a popular method which is used to detect the edges in the given image. With video processing, this operator is used for motion detection.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

(b) 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 -

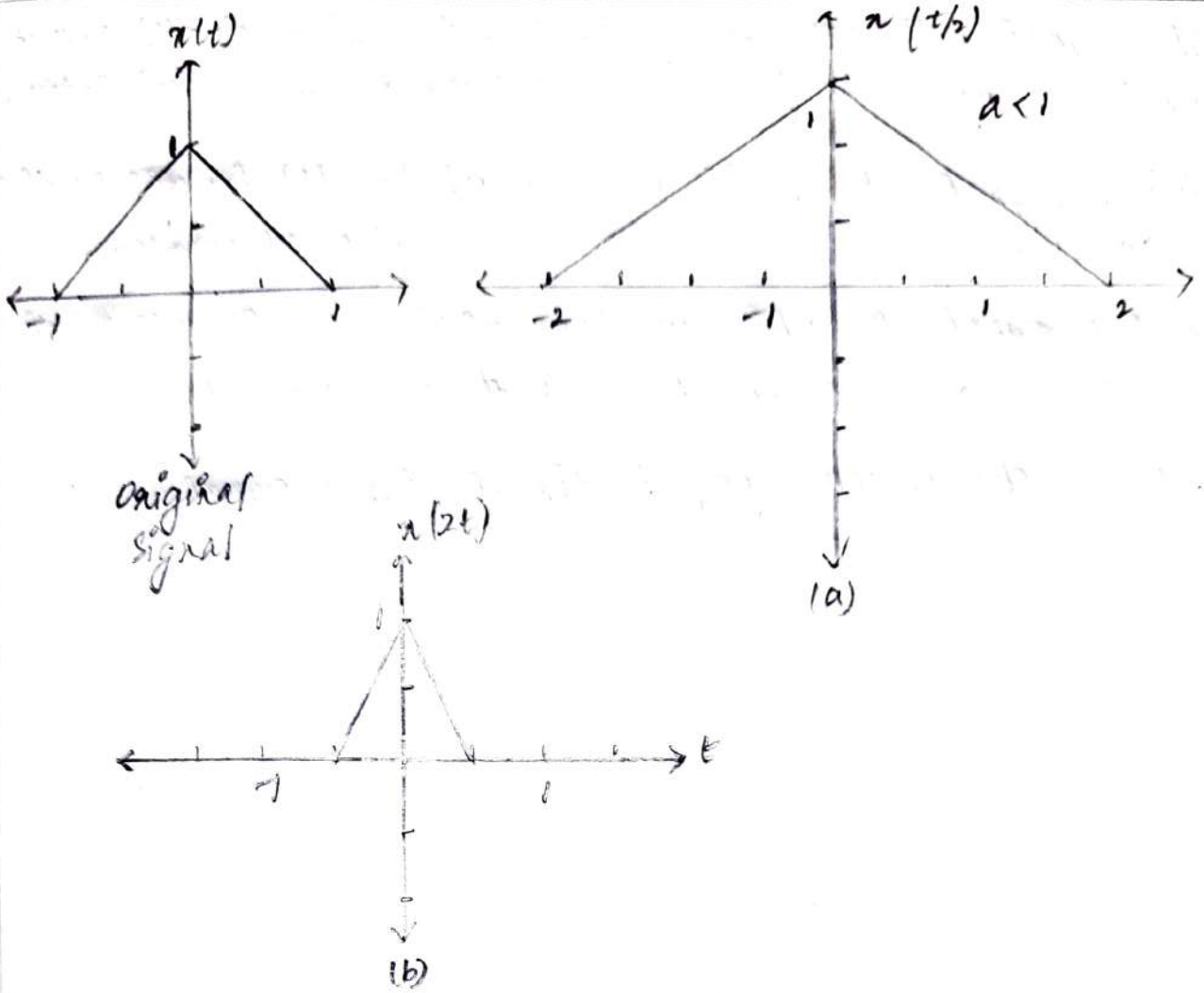
Time scaling of signals involves the modification of a periodicity of the signal, keeping its amplitude constant. Its mathematically expressed as,

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

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

PTO

(a)



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.

(b) 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 -

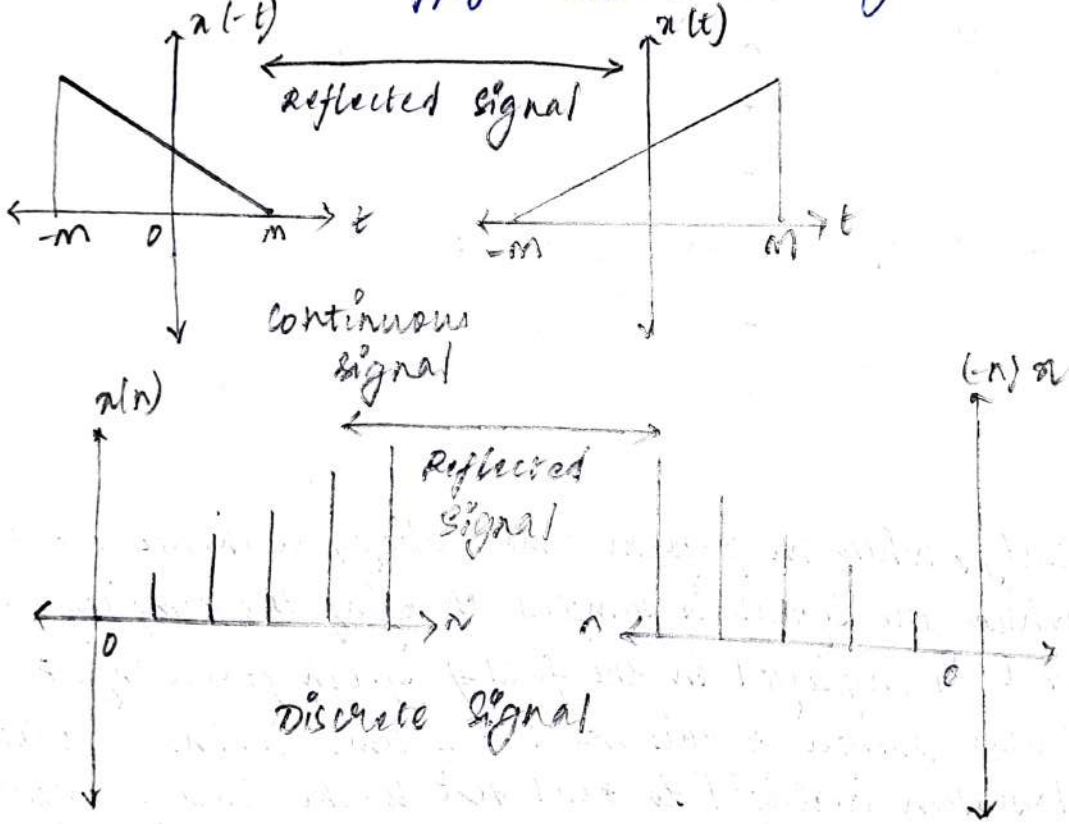
$$y(t) = x(-t)$$

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.

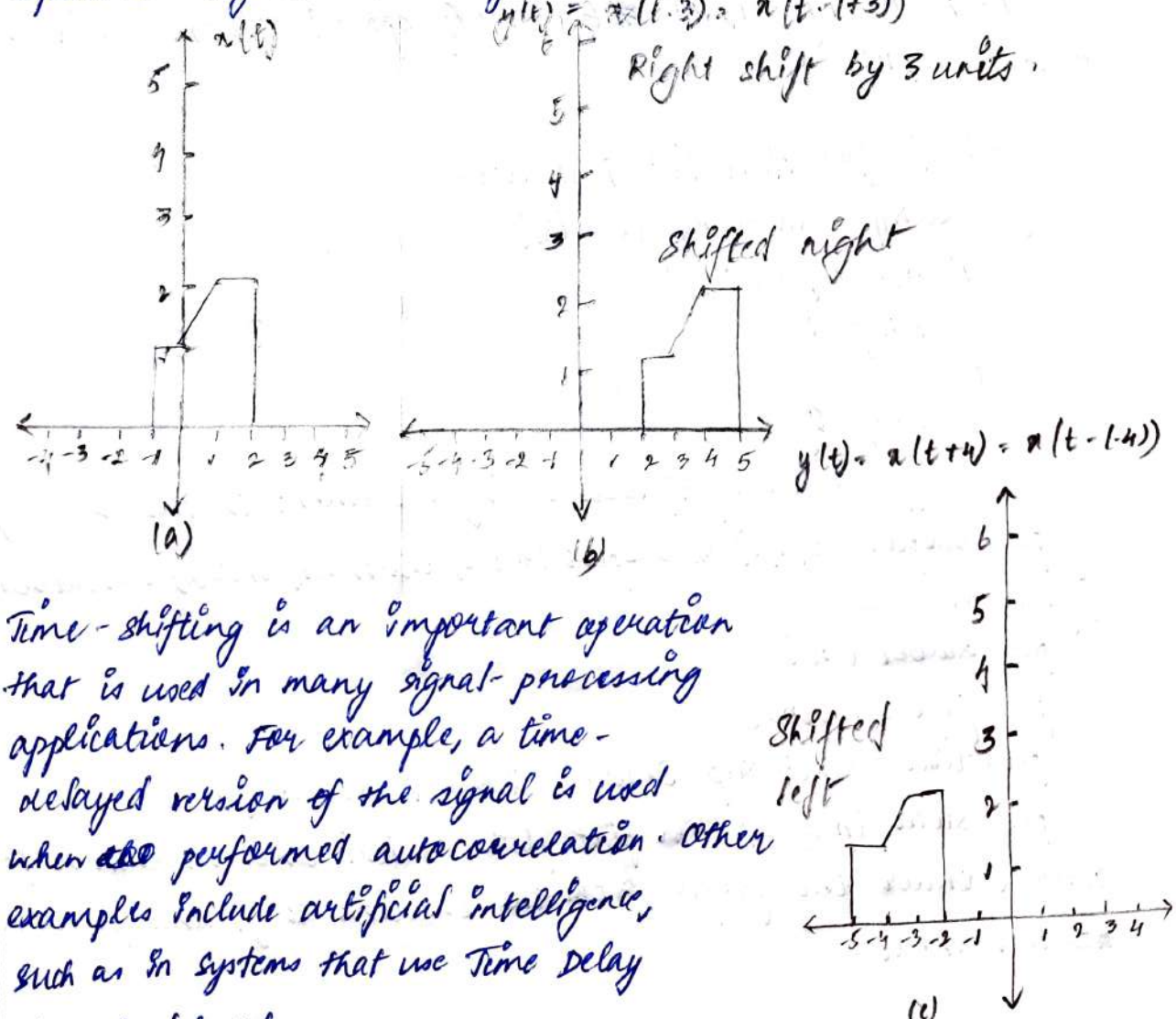
(9) Time Shifting of Signals -

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
interval [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 n 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.
```

P.T.O

```
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_1(): # 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.

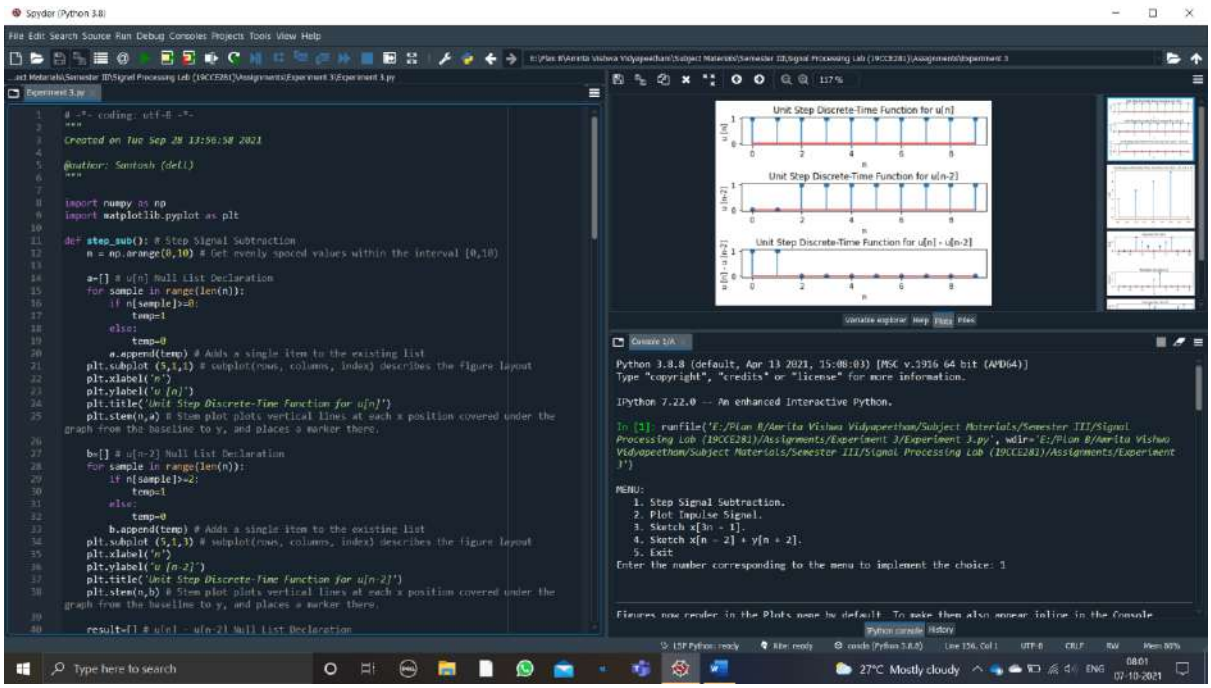
```
def sketch_x1() # Sketch x[n-2] + y[n+2]
    n= np.arange(-6,7) # Get evenly spaced values within the interval [-6,7)
    list_x[] # Null List Declaration
    for sample in n:
        list_x.append(sample-2) # Adds a single item to the existing list, x[n-2]

    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
```

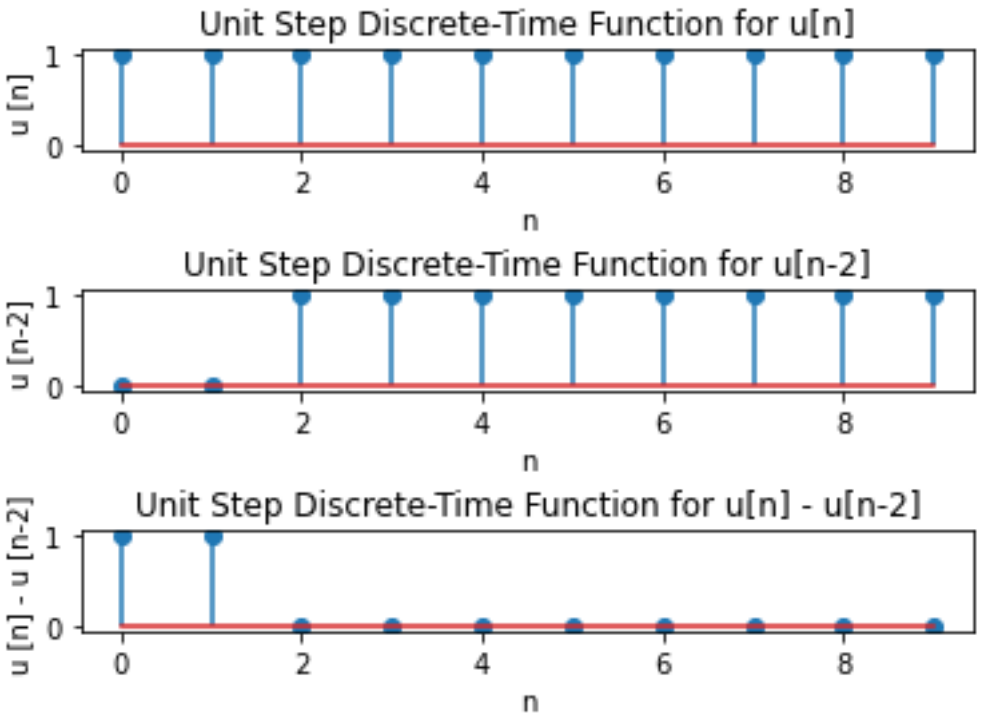
```
    plt.xlabel('n')
    plt.ylabel('x[n-2] + y[n+3]')
    plt.title('Function for x[n-2] + y[n+3]')
    plt.stem(n, result) # stem plot plots vertical lines at each n position covered under the graph from the baseline to y, and places a marker there.
    plt.show() # To view all the three sub-plots.

# 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:**

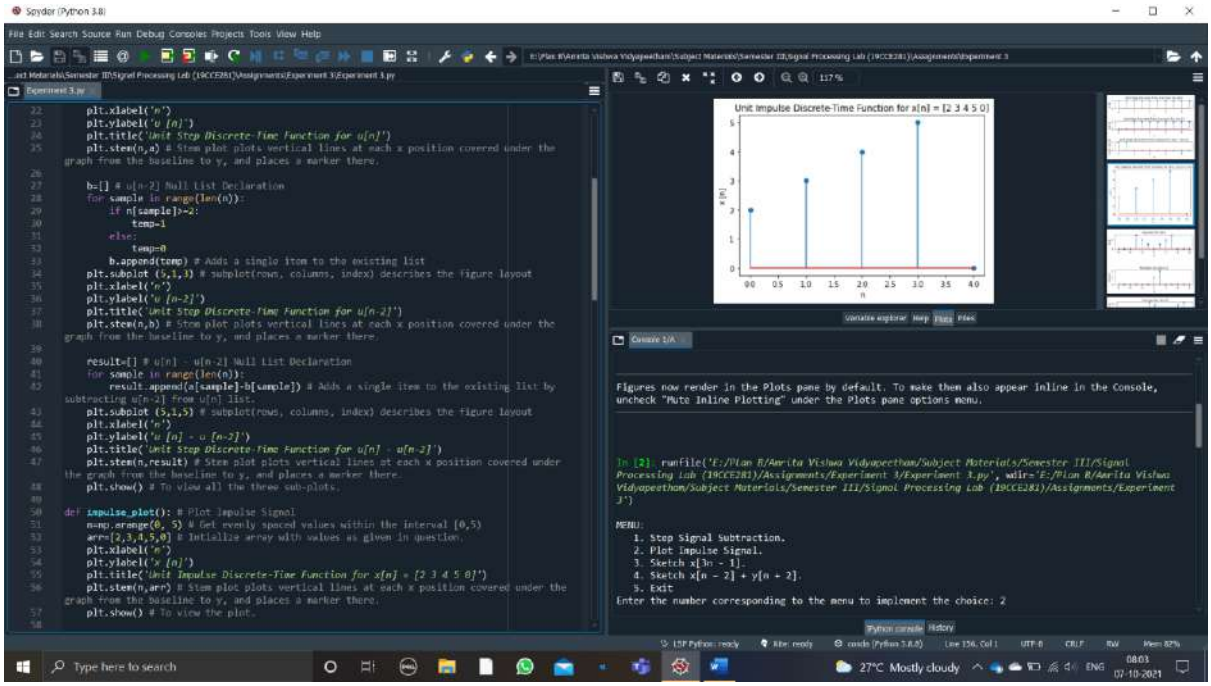
Performed the basic set of signal processing operations using Python code and results verified.

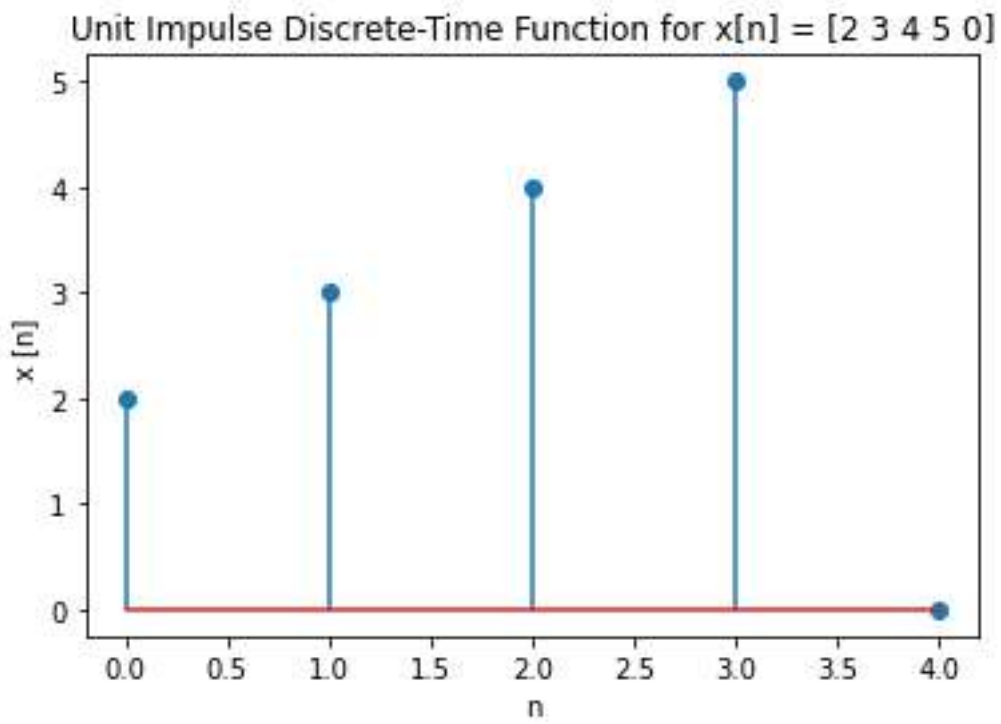
Step Signal Subtraction



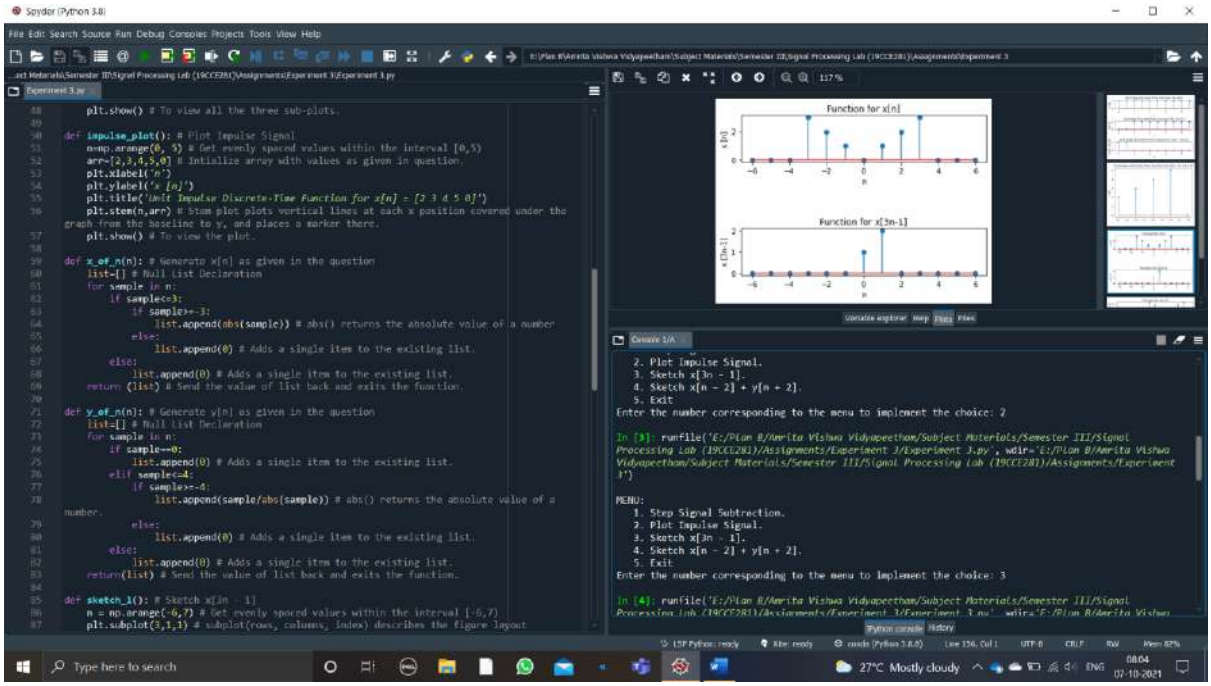


Plot Impulse Signal

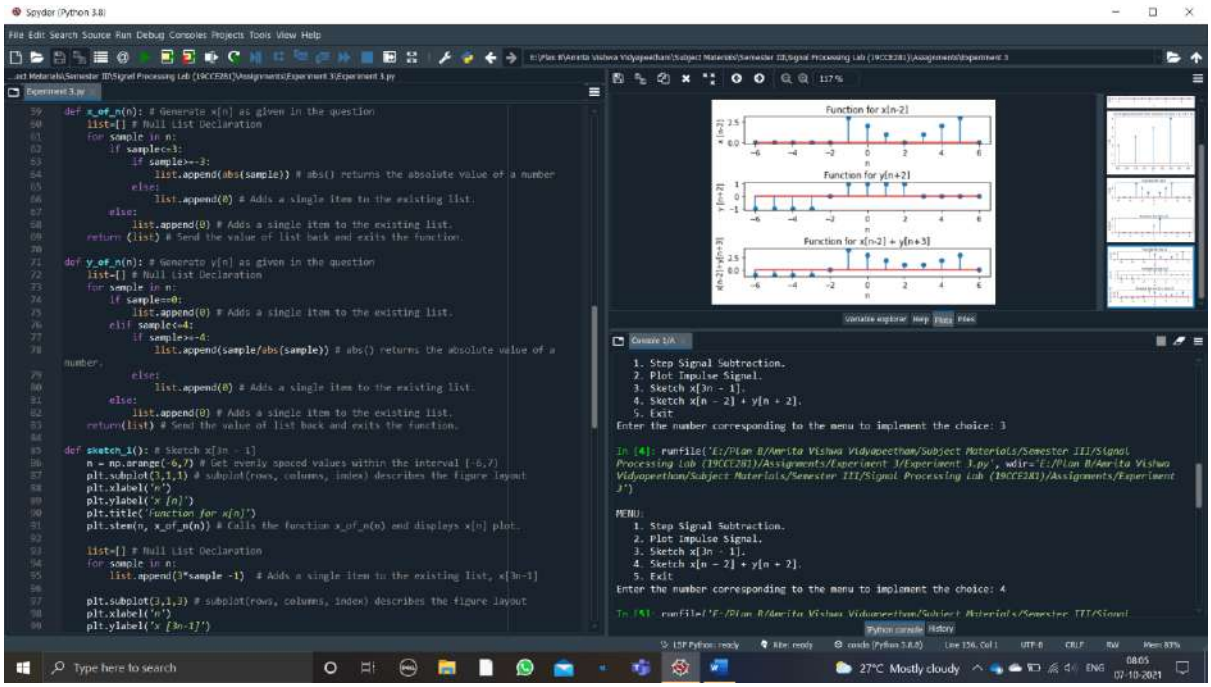




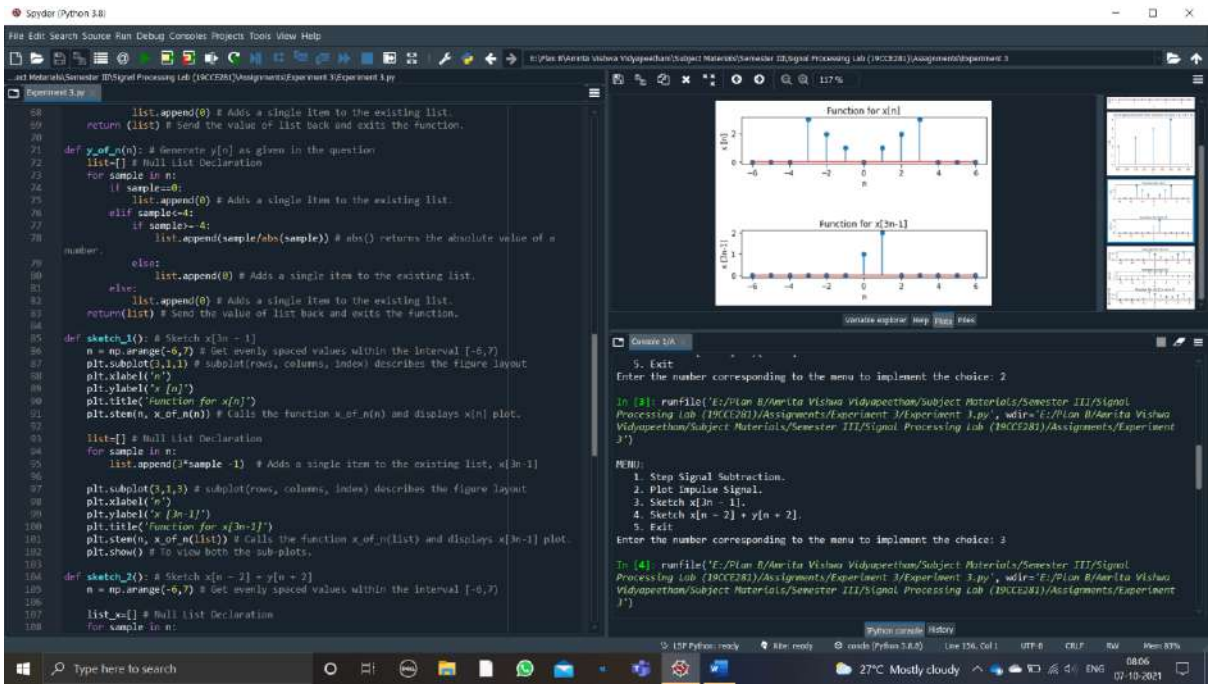
Generate x[n] as given in the question

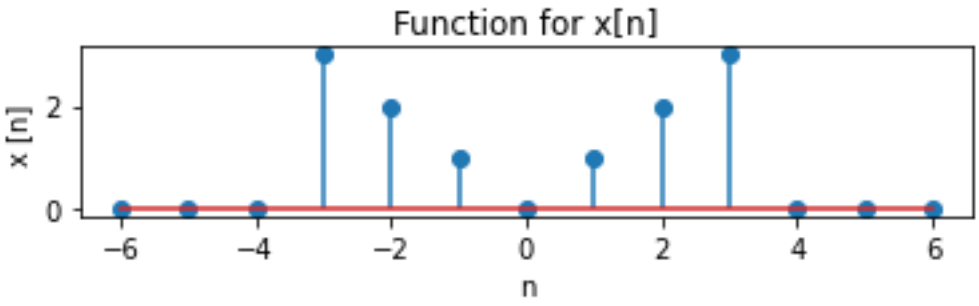


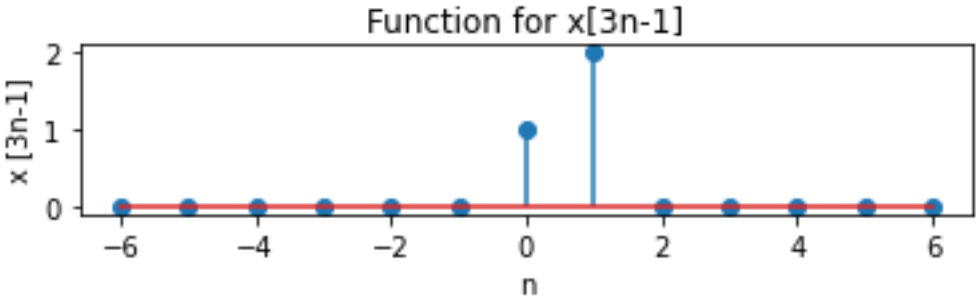
Generate y[n] as given in the question



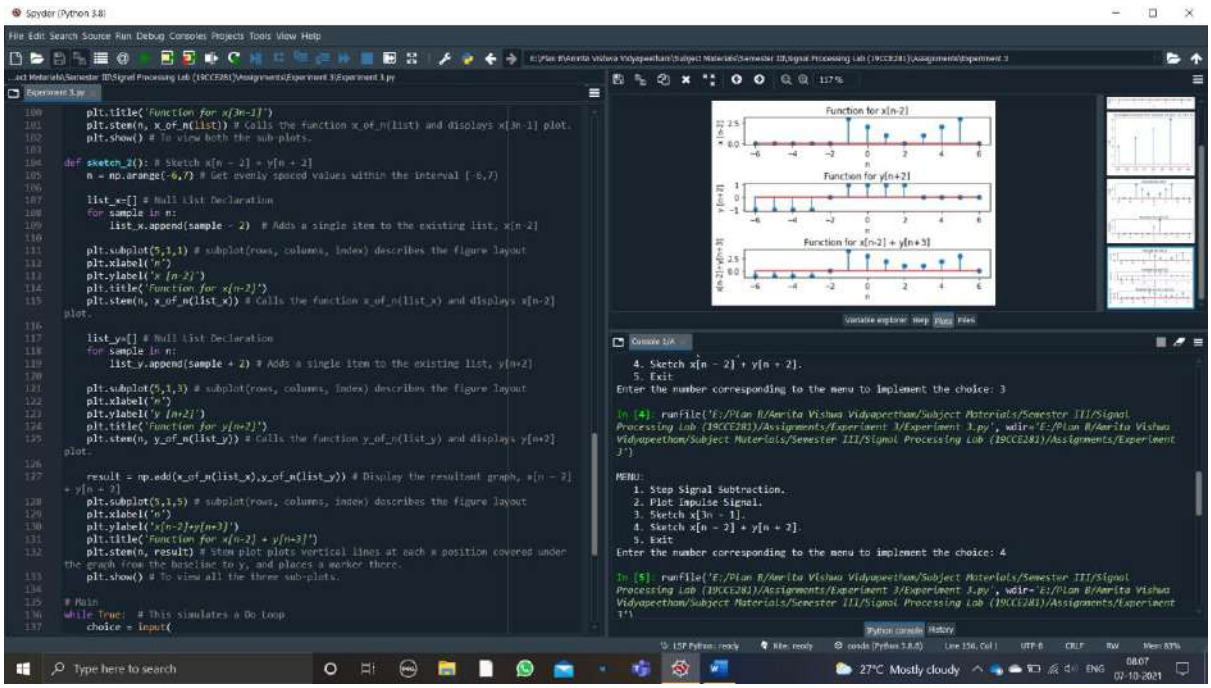
Sketch x[3n - 1]

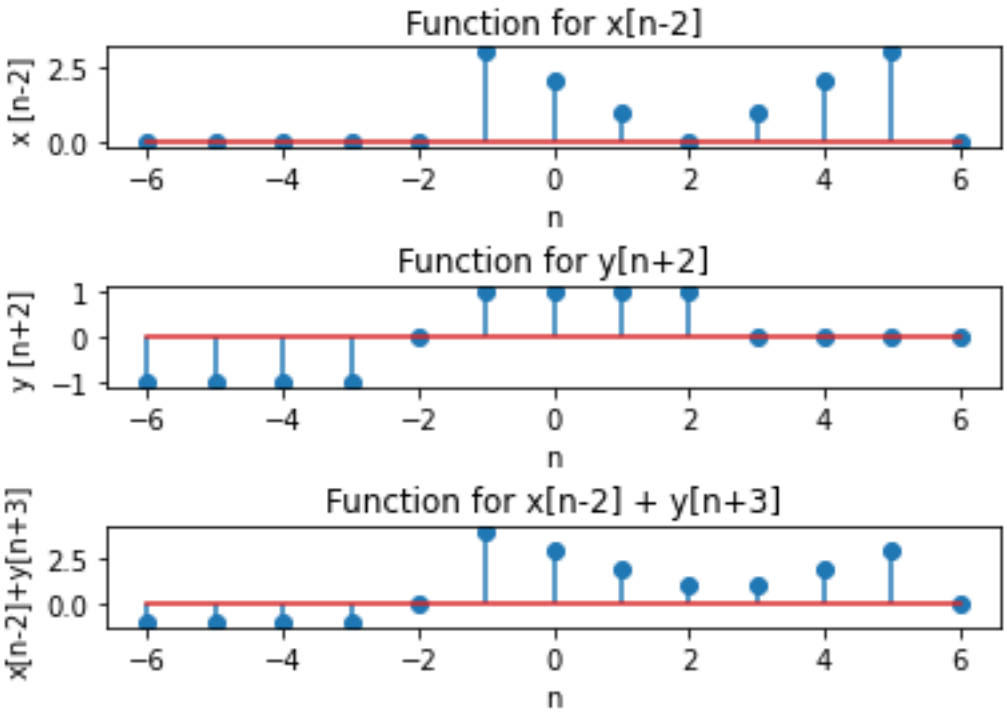

Spyder (Python 3.8)
Function for x[n]
Function for x[3n-1]
5. Exit
Enter the number corresponding to the menu to implement the choice: 2
MENU:
1. Step Signal Subtraction.
2. Plot Impulse Signal.
3. Sketch x[3n - 1].
4. Sketch x[n - 2] + y[n + 2].
5. Exit
Enter the number corresponding to the menu to implement the choice: 3
Type here to search
27°C Mostly cloudy


Function for x[n]
x [n]
n

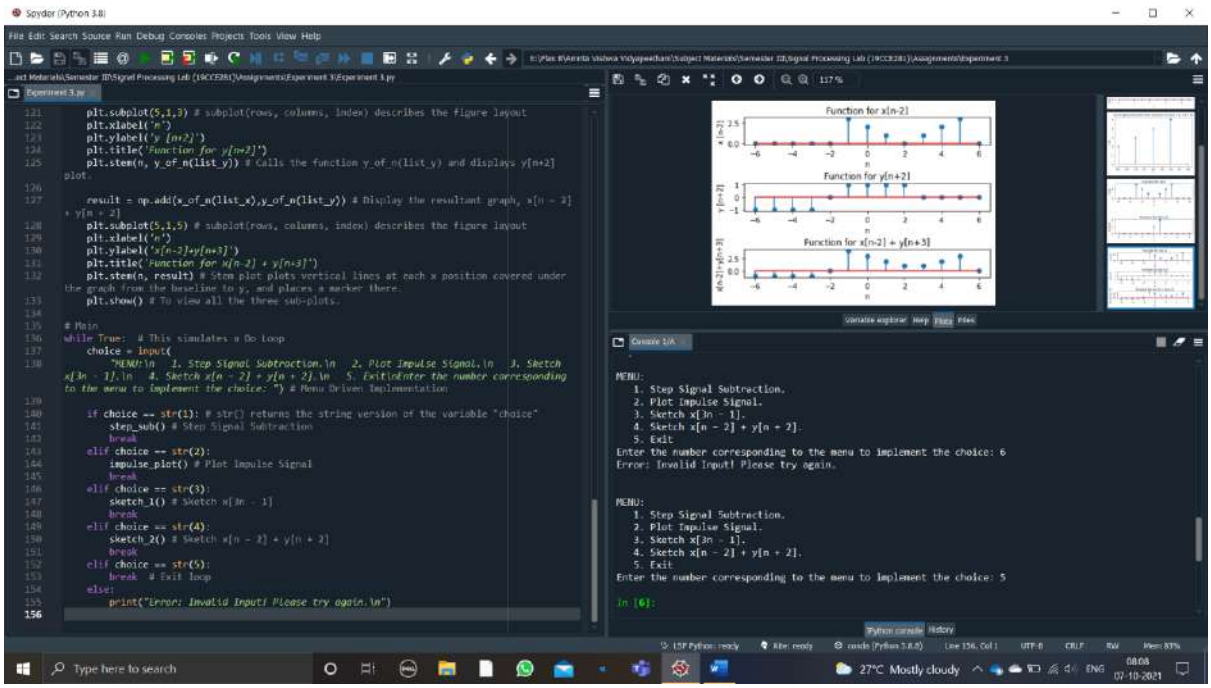

Function for x[3n-1]
x [3n-1]
n

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]:
```

Expt 4: System Properties

Determine whether the system y[n] = nx[n] is linear and time invariant.
Assume the following input sequences
x1=[2,4,7,3]
x2=[1,2,5,3]
Sketch the input and system responses appropriately.

NAME - SANTOSH (CB.EN.U4CLE20053)

① DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCLE) A

LAB TITLE AND CODE : SIGNAL PROCESSING LAB 19CLE281

EXPERIMENT NUMBER : 4

DATE : 05/10/2021

PROPERTIES OF SYSTEM

* AIM :

To determine whether a given input sequence is a Linear Time-Invariant (LTI) System or not.

* SOFTWARE REQUIRED :

Spyder IDE (Anaconda 3) - Python 3.9.1 (64-bit)

* THEORY : (LINEARITY)

The principle of linearity is equivalent to the principle of superposition, i.e, a system can be said to be linear if, for any two input signals, their linear combination yields as output the same linear combination of the corresponding output signals.

→ Definition -

Given any two inputs, acted on by the system as follows:

$x_1(t) \rightarrow S \rightarrow y_1(t)$; $x_2 \rightarrow S \rightarrow y_2(t)$

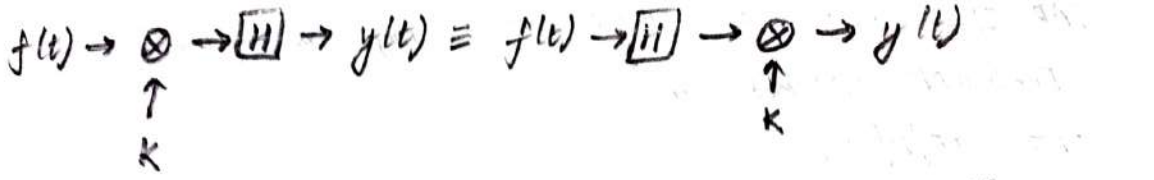
The system is said to be linear if, for any two constants α and β, and the signal : $x_3(t) = \alpha x_1(t) + \beta x_2(t)$.

The system acts on it as follows:

$x_3(t) \rightarrow S \rightarrow y(t)$, where $y(t) = \alpha y_1(t) + \beta y_2(t)$

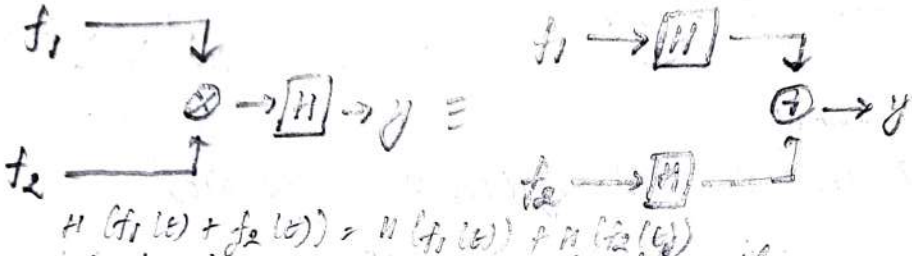
It is not necessary for the input and output signals to have the same independent variable for linearity to make sense. The definition for systems with input and/or output signal being discrete-time is similar.

PTO



$H(Kf(t)) = KH(f(t))$ → 1st step where K is the scaling factor

A block diagram demonstrating the scaling property of linearity.



$H(f_1(t) + f_2(t)) = H(f_1(t)) + H(f_2(t))$

A block diagram demonstrating the superposition property of linearity.

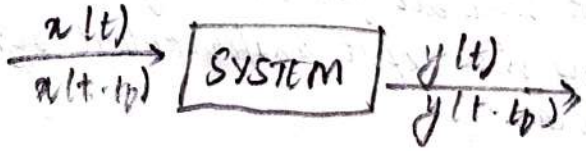
$H(k_1 f_1(t) + k_2 f_2(t)) = k_1 H(f_1(t)) + k_2 H(f_2(t))$

→ Example -

A capacitor, an inductor, a resistor or any combination of these are all linear systems. If we consider the voltage applied across them as an input signal, and the current through them as an output signal. This is because these simple passive circuit components follow the principle of superposition within the ranges of operation.

* THEORY: (TIME INVARIANCE)

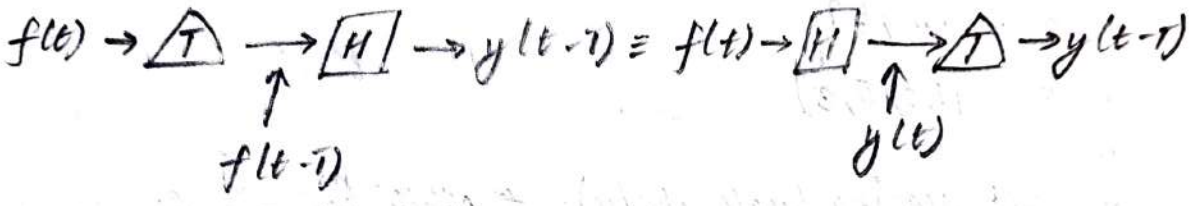
In a time-invariant system, there are no changes in system structure as a function of a time t. A system is time-invariant if -



PTO

From the diagram, we can observe that for any time to and any input x(t) that produces response y(t), the response to the time-shifted input x(t-to) is equal to the time-shifted output y(t-to) of y(t).

Therefore, in a time-invariant system, the response to a left or right shift of the input x(t) is equal to a corresponding left or right shift in the response y(t).



This block diagram shows the condition for time invariance. The output is the same whether the delay is put on the input or the output.

→ Example -
An electric circuit with fixed values of Resistance R, Inductance L, and capacitance C can be considered as time invariant.

+ GRAPH PLOTTING ALGORITHM :
The followed steps were followed -

① Define the x-axis and corresponding y-axis 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.

PTO

+ PROGRAM WITH COMMENTS:

```
# import library source files
import numpy as np
import matplotlib.pyplot as plt

n = np.arange(0,4) # Get evenly spaced values within the
                     Interval [0,4)
# Given Input Sequences:
x1 = [2,4,1,3]
x2 = [1,2,5,3]

def scaling(scale, choice): # check Homogeneity condition
    before_scaling = []; after_scaling = []

if choice = str(1): # Calculate for x1 = [2,4,1,3]

    for sample in range(len(n)):
        temp = sample * x1[sample]
        before_scaling.append(int(scale) * temp) # Scaling the
                                                   output

  for sample in range(len(n)):
        temp = int(scale) * x1[sample]
        after_scaling.appena(sample * temp) # sending output
                                              into system

else: # Calculate for x2 = [1,2,5,3]

  for sample in range(len(n)):
        temp = sample * x2[sample]
        before_scaling.append(int(scale) * temp) # scaling the
                                                   output
```

```
30      for sample in range (len(n)):
31          temp = int (scale) * a2[sample]
32          after_scaling. append (sample * temp) # Sending
33  output into system
34    plt.subplot (3,1,1) # subplot (rows, columns, index
    describes the figure layout
35    plt. xlabel ('n')
36    plt. ylabel ('y' + str (choice) + '[n]')
37.   plt. title ('Before scaling x' + str (choice) + '{}'. format
    (before_scaling) # Formats the specified value(s) and insert
    the string's placeholder.
38    plt. stem (n, before_scaling) # stem plot plots vertical
    lines at each x position covered under the graph from the
    baseline to y, and places a marker there.
39
40    plt. subplot (3, 1, 3) # subplot (rows, columns, index) describes
    the figure layout
41    plt. xlabel ('n')
42    plt. ylabel ('y' + str (choice) + '[n]')
43    plt. title ('After scaling x' + str (choice) + '{}'. format
    (after_scaling)) # Formats the specified value(s) and insert
    them inside the string's placeholder.
44    plt. stem (n, after_scaling) # stem plot plots vertical
    lines at each x position covered under the graph from the
    baseline to y, and places a marker there.
45
46    plt. show() # To view both the sub-plots
47
48    if (before_scaling == after. scaling): # Condition to check if
    both the systems are equal or not after scaling
49        print (" The given system satisfies Homogeneity condition.")
50        return (1)
```

P.T.O

```
    else:
        print (" The given system does not satisfy Homogeneity
Homogeneity condition.")
        return (0)

def superposition (): # check additivity condition
    before_adding = []; after_adding = []; y1 = []; y2 = []; x = []

    for sample in range (len(n)):
        y1.append (sample * x1[sample])
        y2.append (sample * x2[sample])
        before_adding.append (y1[sample] + y2[sample])
# Processing System Individually

    plt.subplot (3,1,1) # subplot (rows, columns, index) describes
the figure layout
    plt.xlabel ('n')
    plt.ylabel ('y[n]')
    plt.title ('H (x1 (t) + x2(t))')
    plt.stem (n, before_adding)

    for sample in range (len(n)):
        x.append (x1[sample] + x2[sample])
        after_adding.append (sample * x[sample])
# Processing system Together

    plt.subplot (3,1,3) # subplot (rows, columns, index)
describes the figure layout
    plt.xlabel ('n')
    plt.ylabel ('y[n]')
    plt.title ('H (x1 (t)) + H (x2(t))')
    plt.stem (n, after_adding) # stem plot plots vertical
lines at each x position covered under the graph from the
baseline to y, and places a marker there.
```

```
78
79  plt.show() # To view both the sub-plots
80
81  print('\n 2. Superposition Condition :-')
82  if (before_adding == after_adding): # Condition to check if both the systems are equal or not after adding
83      print(" The given system satisfies superposition condition.")
84      return(1)
85  else:
86      print(" The given system does not satisfy superposition condition.")
87      return(0)
88
89  def time(shift, choice): # Check Time Variance
90      input_delay = []; output_delay = []
91
92      if choice == str(1): # Calculate for x1 = [2, 4, 7, 3]
93
94          # Output Delay
95          for sample in range(len(x1)):
96              output_delay.append(sample * x1[sample]) # sending output into system
97
98          for sample in range(int(shift)):
99              output_delay.append(0) # Adds extra sample at the end for shifting
100
101         for sample in range(int(shift)):
102             for sample in range(len(output_delay)-1, 0, -1):
103                 output_delay[sample] = output_delay[sample-1]
    # shift the sample
104
```

P10

```
105     for sample in range (int (shift)):
106         output_delay [sample] = 0 # Assign zero to the empty
elements for the shifted bit
107
108     # Input Delay
109     for sample in range (int (shift)):
110         x1. append (0) # Adds extra sample at the end for
shifting
111
112     for sample in range (int (shift)) :
113         for sample in range (len (x1) - 1, 0, -1) :
114             x1 [sample] = x1 [sample-1] # shift the sample
115
116     for sample in range (int (shift)) :
117         x1 [sample] = 0 # Assign zero to the empty elements
for the shifted bit
118
119     for sample in range (len (x1)):
120         input_delay. append (sample + x1 [sample]) # sending
input into system
121
122 else : # Calculate for x2=[1, 2, 5, 3]
123
124     ## output Delay
125     for sample in range (len (x2)):
126         output_delay. append (sample + x2 [sample]) # sending
output into system
127
128     for sample in range (int (shift)):
129         output_delay. append (0) # Adds extra sample at
the end for shifting
130
131
```

P10

```
131     for sample in range (int (shift)):
132         for sample in range (len (output_delay)-1, 0, -1):
133             output_delay [sample] = output_delay [sample -1]
    # shift the output
134
135     for sample in range (int (shift)):
136         output_delay [sample] = 0 # Assign zero to the empty
    elements for the shifted bit
137
138     # Input Delay
139     for sample in range (int (shift)):
140         x2. append (0) # Adds extra sample at the end for shifting
141
142     for sample in range (int (shift)):
143         for sample in range (len (x2) -1, 0, -1):
144             x2 [sample] = x2 [sample -1] # shift the sample
145
146     for sample in range (int (shift)):
147         x2 [sample] = 0 # Assign zero to the empty elements for
    the shifted bit
148
149     for sample in range (len (x2)):
150         input_delay. append (sample * x2 [sample]) # sending
    input into system
151
152  plt. subplot (3,1,1) # subplot (rows, columns, index) describes
    the figure layout
153  plt. xlabel ('t')
154  plt. ylabel ('y(t)')
155  plt. title ('shifting input by ' + str (shift) + ' for x' + str (choice)
    + '{}'. format (input_delay)) # Formats the specified value(s)
    and insert them inside the string's placeholder.
```

PTO

```
156   plt.stem(np.arange(len(input_delay)), input_delay) # stem plot plots vertical lines at each x position covered under the graph from the baseline to y, and places a marker there.
157
158   plt.subplot(3,1,3) # subplot(rows, columns, index) describes the figure layout
159   plt.xlabel('t')
160   plt.ylabel('y(t)')
161   plt.title('shifting output by ' + str(shift) + ' for x' + str(choice) + '{3}'.format(output_delay)) # Formats the specified value(s) and insert them inside the string's placeholder.
162   plt.stem(np.arange(len(output_delay)), output_delay) # stem plot plots vertical lines at each x position covered under the graph from the baseline to y, and places a marker there.
163
164   plt.show() # To view both the sub-plots
165
166   if(input_delay == output_delay): # Condition to check if both the system are equal or not after shifting
167       print("Since both the outputs are same, the given system is time invariant.")
168       second_condition = 1
169   else:
170       print("Since both the outputs are not same, the given system is time variant.")
171       second_condition = 0
172
173   return(second_condition)
174
175 # Main
176 while True: # This simulates a Do Loop
```

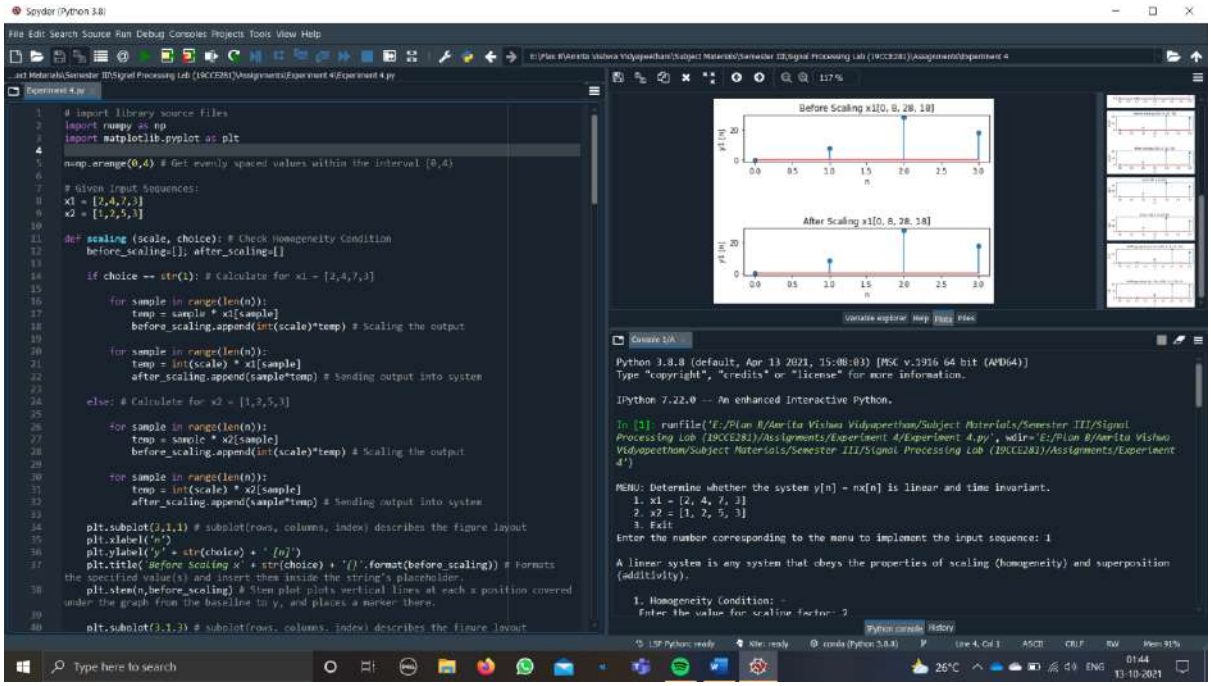
```
choice
177  ~~value~~ = input(
178  "MENU: Determine whether the system y[n] = nx[n] is linear and time invariant.\n 1. x1 = [3,4,7,3]\n 2. x2 = [1,2,5,3]\n 3. Exit\n Enter the number corresponding to the menu to Implement the Input sequence: ") # Menu Driven Implementation
179
180  if choice == str(1) or choice == str(2):
181
182      scale = input("A linear system is any system that obeys the properties of scaling (homogeneity) and superposition (additivity).\n \n 1. Homogeneity Condition: -\n Enter the value for scaling factor: ")
183
184      if (scaling (scale, choice) == superposition(1): # Condition to check if both the systems or linear or not
185          print("\n Since both the properties are satisfied; the given system is linear.")
186          first_condition = 1
187      else:
188          print("\n Since both the properties are not satisfied; the given system is not linear.")
189          first_condition = 0
190
191      shift = input("A system is time-invariant if a time shift (i.e., advance or delay in the input always results only in an identical time shift in the output.\n Enter the value for shifting factor: ")
192
193      if (first_condition + time (shift, choice)==2): # Condition to check for LTI system or not
194          print("\n The given system is a linear Time Invariant (LTI) system.")
195      else:
```

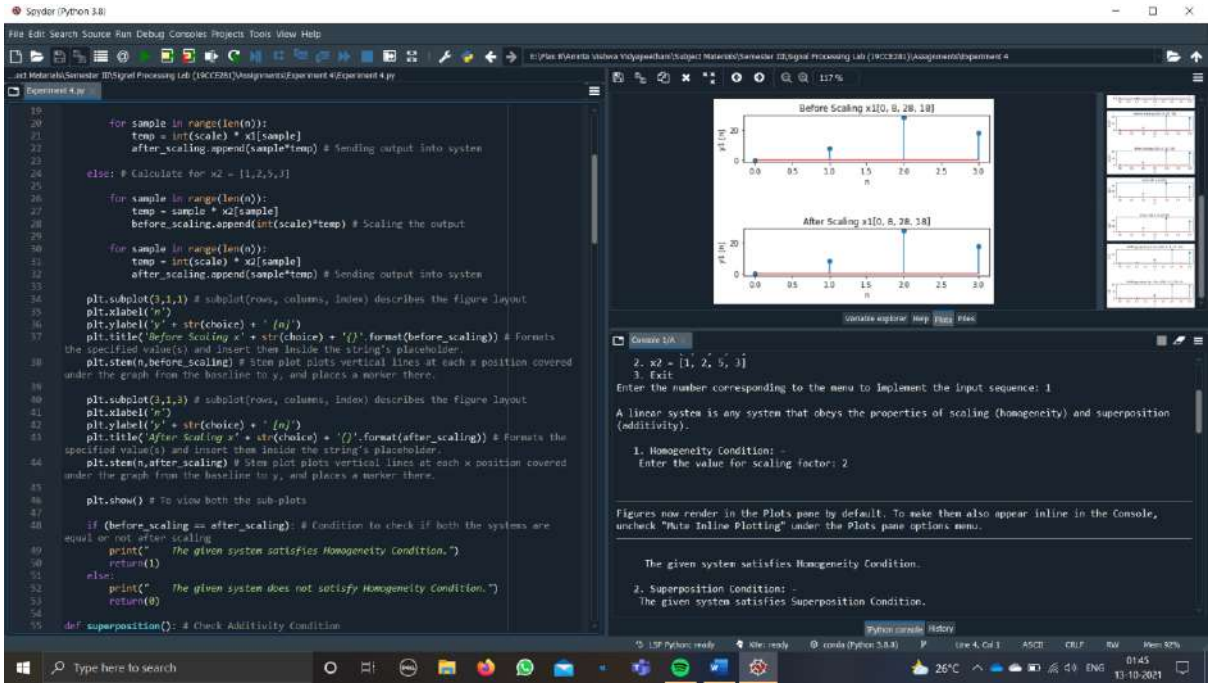
```
196         print("\n The given system is not a Linear Time Invariant (LTI) system.")
197
198     break
199
200 elif choice == str(3):
201         break : # Exit Loop
202
203 else:
204     print("Error: Invalid Input! Please try again.\n")
```

+ INFERENCE:

Examine the given time sequence as Linear Time-Invariant (LTI) system or not using Python code and results verified.

Check Homogeneity Condition





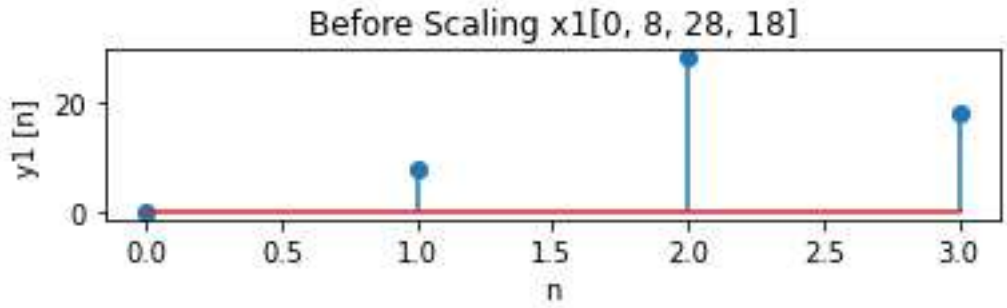
Step 1: Scale the output.

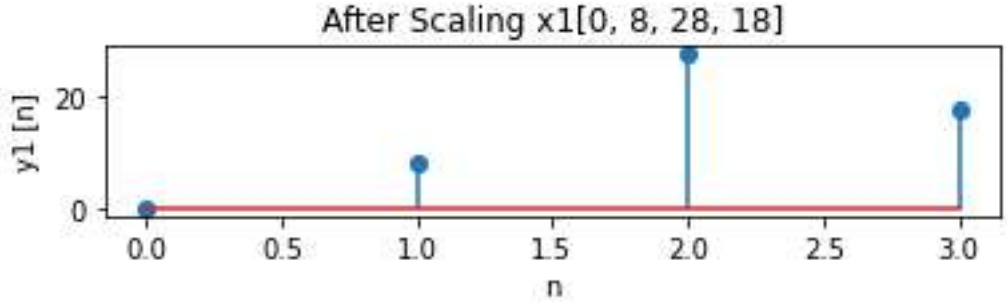
Step 2: Send output into the system.

Step 3: Plot graphs before and after scaling.

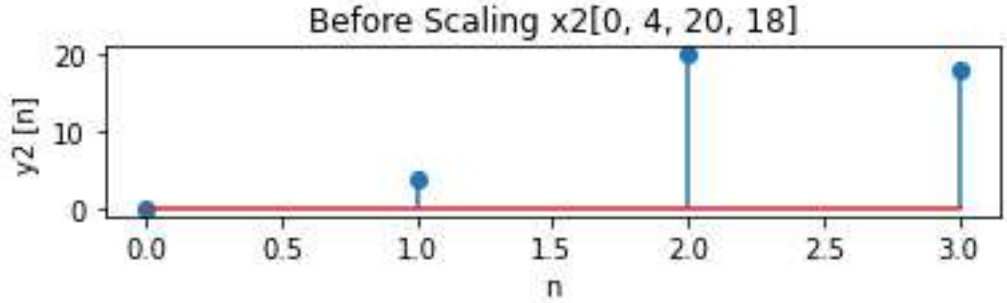
Step 4: Check if both the systems are equal or not after scaling.

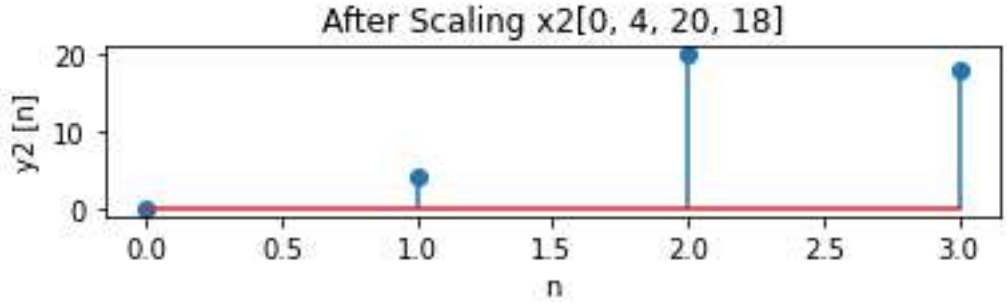
x1 = [2,4,7,3]





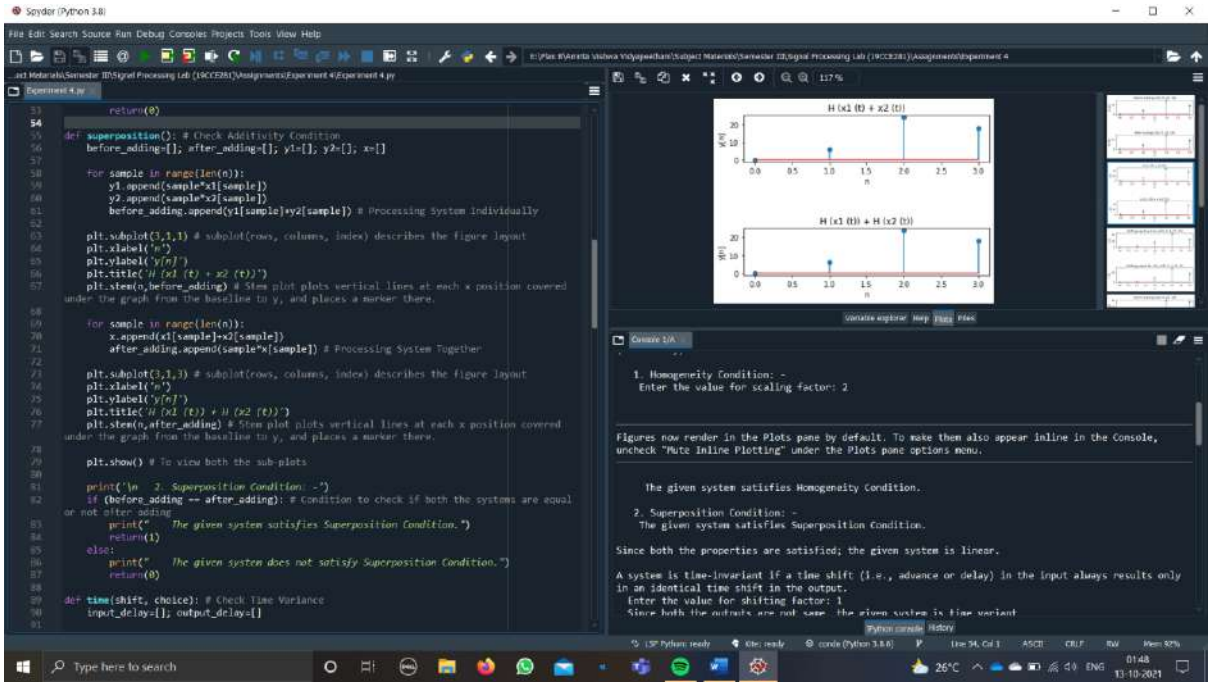
x2 = [1,2,5,3]

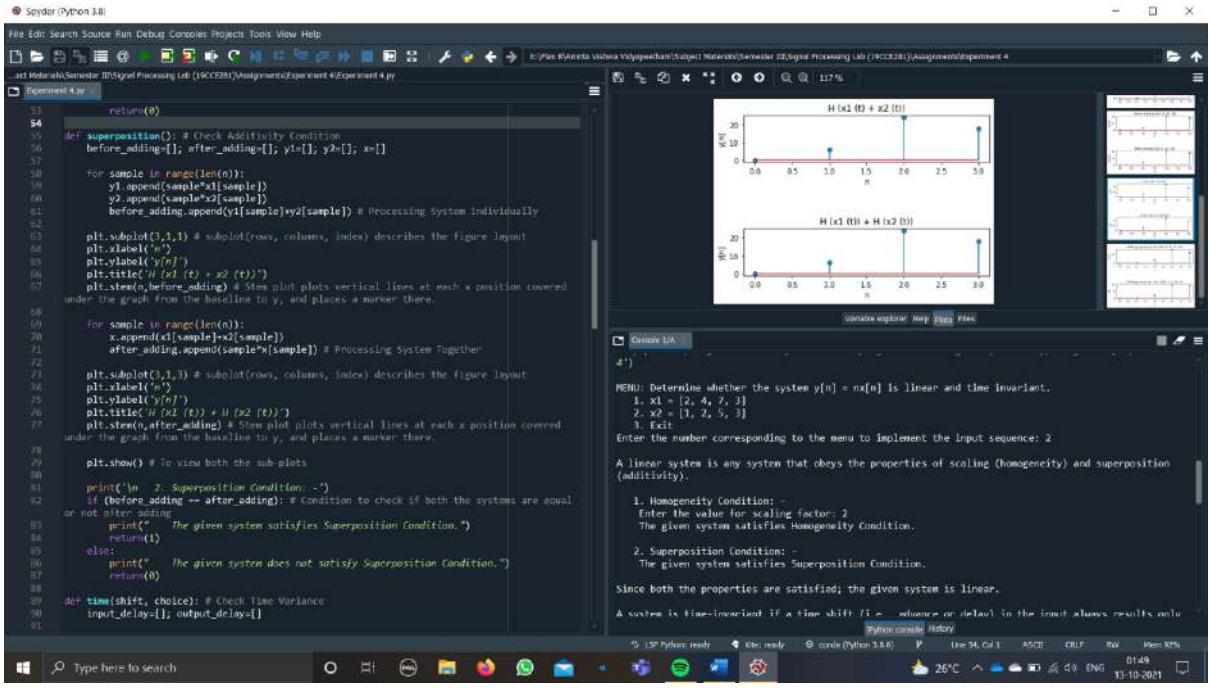




The given system satisfies the Homogeneity Condition.

Check Additivity Condition





Step 1: Process System Individually.

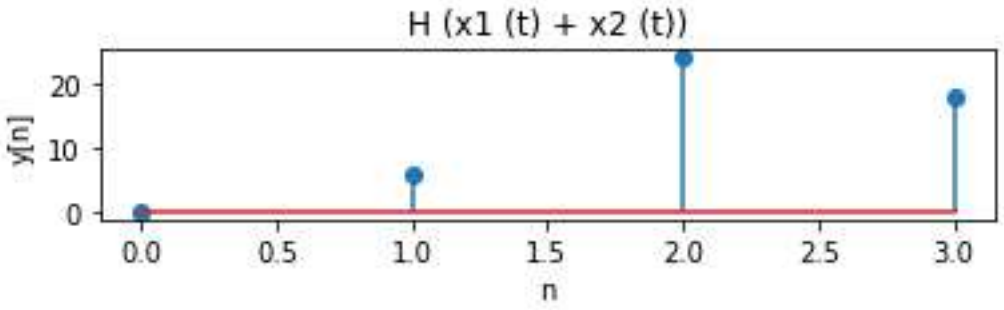
Step 2: Plot graph before adding.

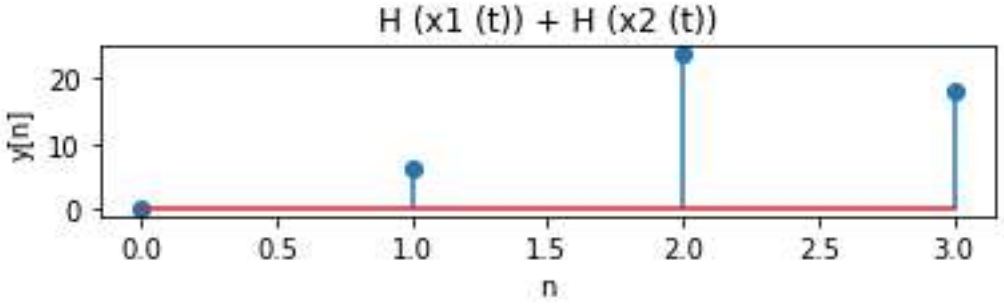
Step 3: Process System Together.

Step 4: Plot graph after adding.

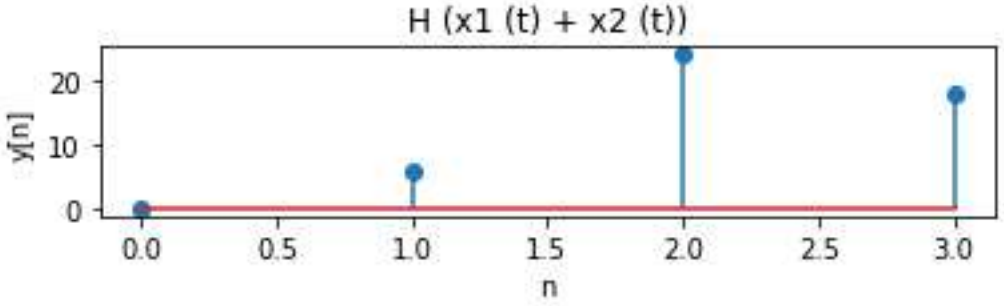
Step 5: Check if both the systems are equal or not after adding.

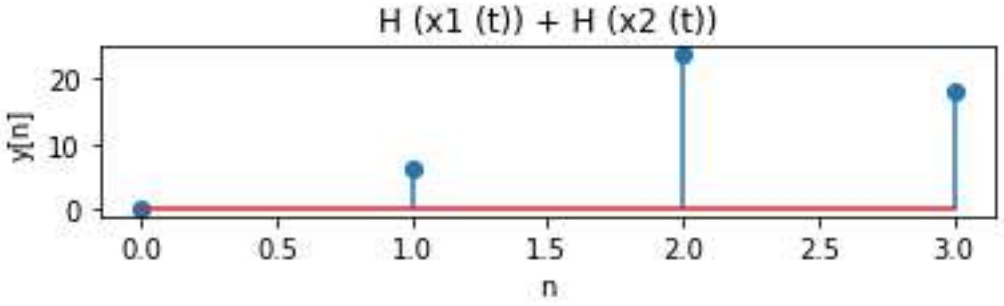
x1 = [2,4,7,3]





x2 = [1,2,5,3]

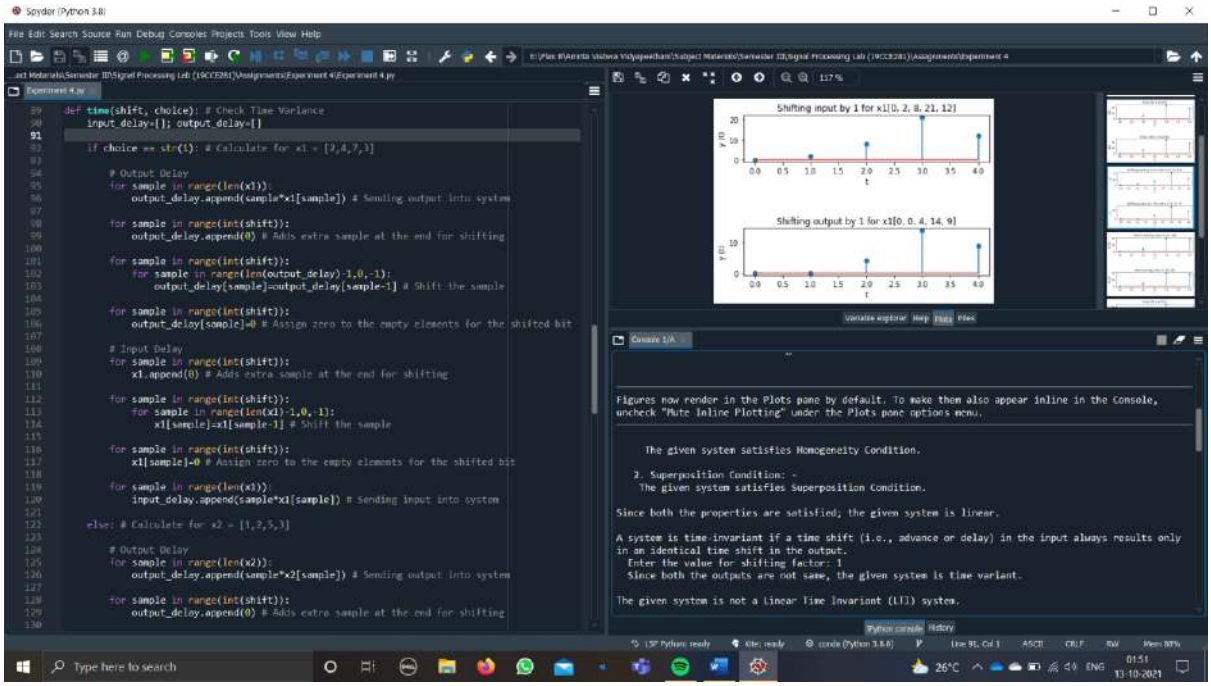


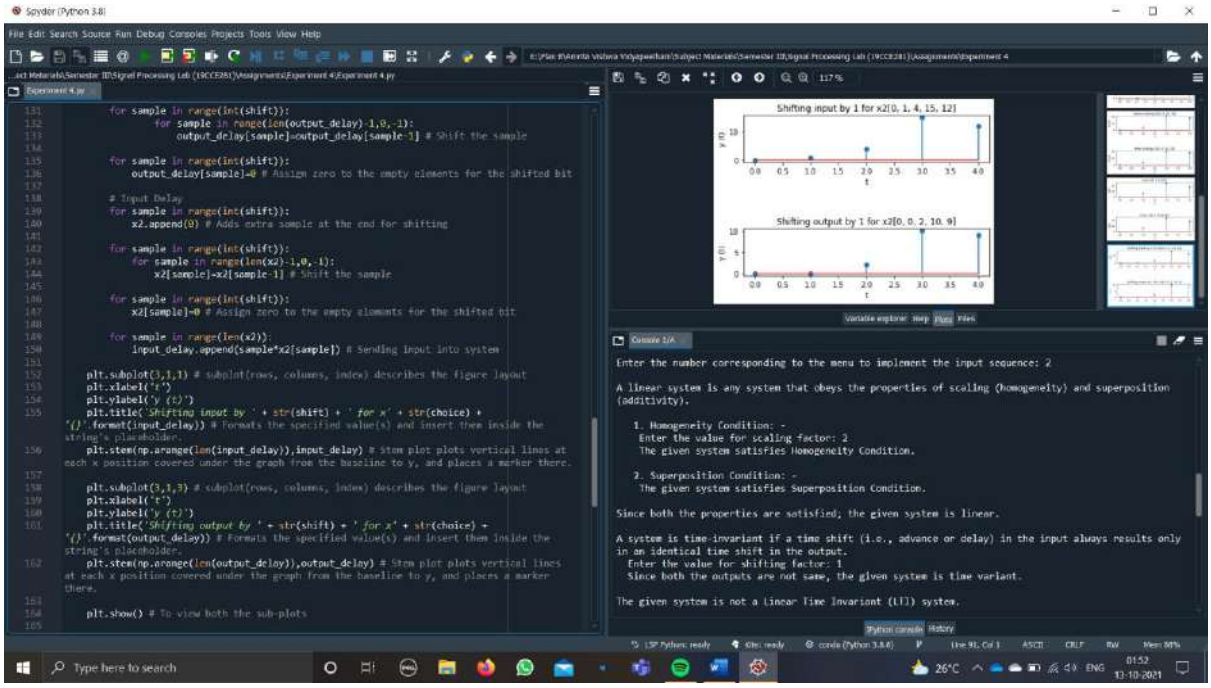


The given system satisfies Superposition Condition.

Since both the properties are satisfied; the given system is linear.

Check Time Variance



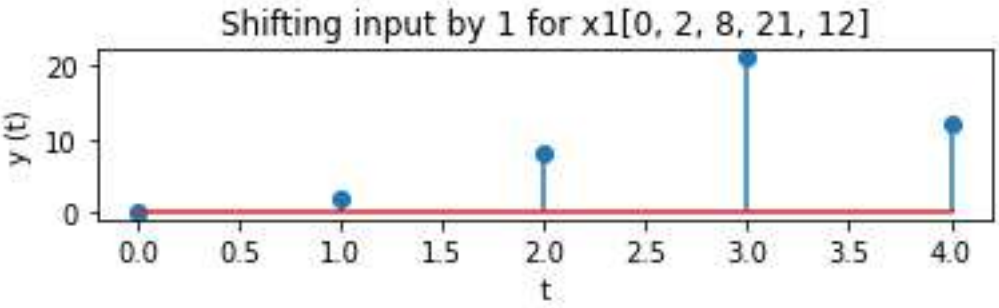


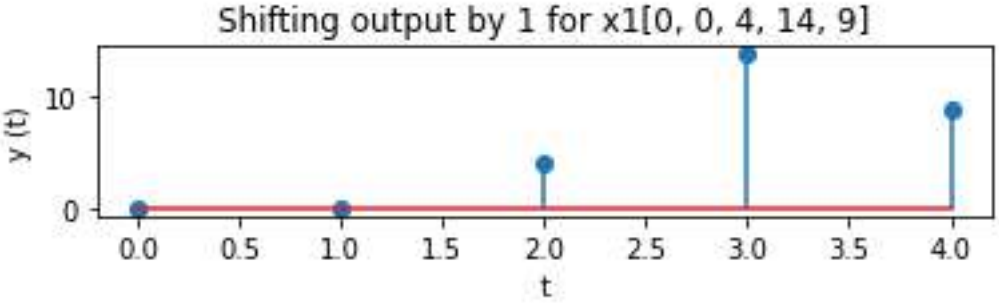
Step 1: Process input and output delay.

Step 2: Plot graphs before and after shifting.

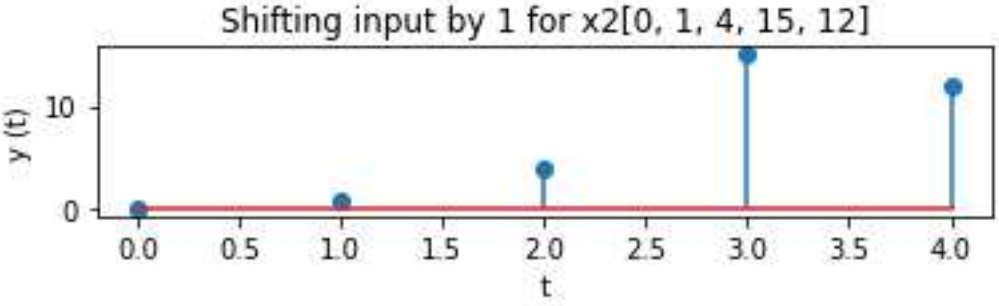
Step 3: Check if both the systems are equal or not after shifting.

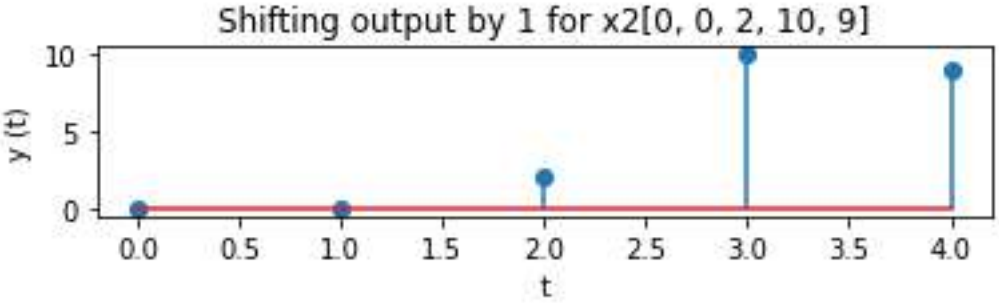
x1 = [2,4,7,3]





x2 = [1,2,5,3]

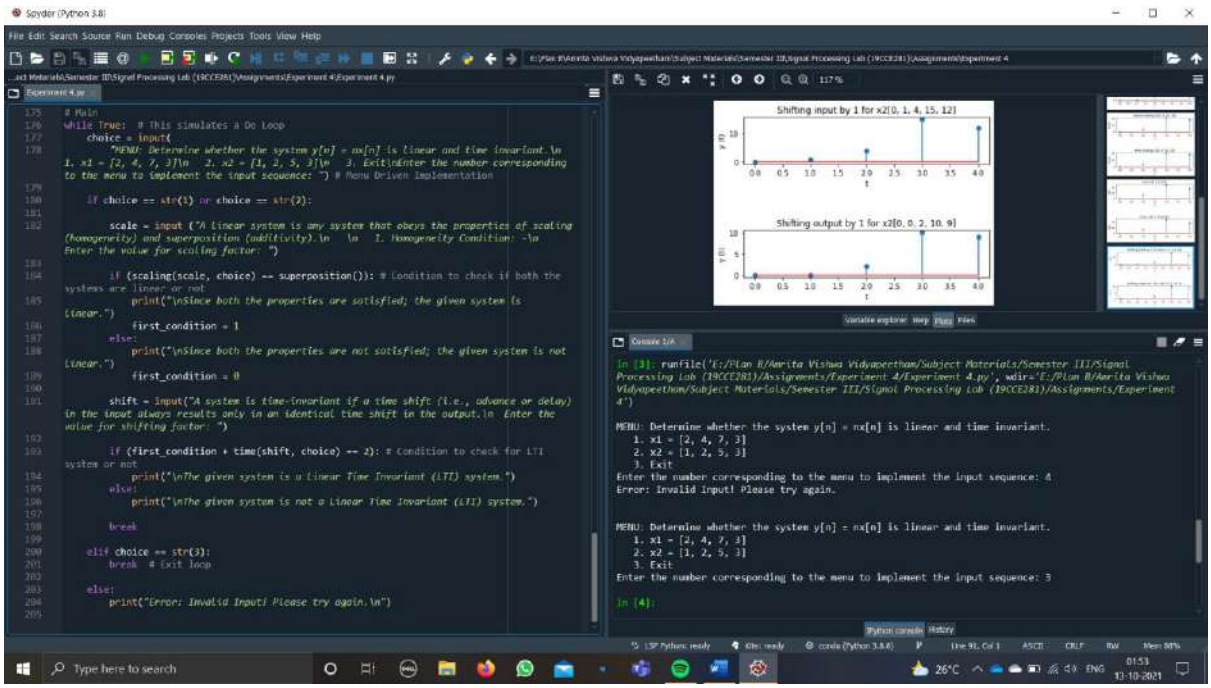




Since both the outputs are not the same, the given system is time-variant.

The given system is not a Linear Time-Invariant (LTI) system.

Main 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 4/Experiment 4.py'      ='E:/Plan
B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab
(19CCE281)/Assignments/Experiment 4'

MENU: Determine whether the system y[n] = nx[n] is linear and time invariant.
   1. x1 = [2, 4, 7, 3]
   2. x2 = [1, 2, 5, 3]
   3. Exit
Enter the number corresponding to the menu to implement the input sequence: 1

A linear system is any system that obeys the properties of scaling (homogeneity) and
superposition (additivity).

   1. Homogeneity Condition: -
    Enter the value for scaling factor: 2
    The given system satisfies Homogeneity Condition.

   2. Superposition Condition: -
    The given system satisfies Superposition Condition.

Since both the properties are satisfied; the given system is linear.

A system is time-invariant if a time shift (i.e., advance or delay) in the input always
results only in an identical time shift in the output.
  Enter the value for shifting factor: 1
  Since both the outputs are not same, the given system is time variant.

The given system is not a Linear Time Invariant (LTI) system.

In [2]:
```

Expt 5: Convolution Sum

Assume discrete-time LTI system with input x[n] = [2,3,4] and impulse response h[n] =[1,2,3] for $n \geq 0$. Determine the system response y[n]. Sketch the input, impulse response and system responses appropriately.

NAME - SANTOSH (CB.EN.U4 CLE20053)
DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CLE) A

LAB TITLE AND CODE : SIGNAL PROCESSING LAB 19 CLE 281

EXPERIMENT NUMBER : 5

DATE : 12/10/2021

DISCRETE - TIME CONVOLUTION SUM

* **AIM :**

given the input $x[n]$ and impulse response $h[n]$, determine the system response $y[n]$ in discrete-time LTI system.

* **SOFTWARE REQUIRED :**

Spyder IDE (Anaconda3) - Python 3.9.7 (64-bit)

* **THEORY :**

convolution can be used to determine the output of a system produces for a given input signal. It can be shown that a linear time-invariant system is completely characterized by its impulse response. The shifting property of the discrete-time impulse function tells us that the input signal to a system can be represented as a sum of scaled and shifted unit impulses.

Thus, by linearity, it would seem reasonable to compute the output signal as the sum of scaled and shifted unit impulse responses. That is exactly what the operation of convolution accomplishes. Hence, convolution can be used to determine a linear time-invariant system's output from knowledge of the input and the impulse response.

Discrete time convolution is an operation on two discrete time signals defined by the integral -

$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[k]\, g[n-k]$$

for all signals f, g defined on $\mathbb{Z}$. It is important to note that the operation of convolution is commutative, meaning that -

$$f * g = g * f$$

for all signals f, g defined on $\mathbb{Z}$. Thus, the convolution operation could have been just as easily stated using the equivalent definition -

$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[n-k]\, g[k]$$

for all signals f, g defined on $\mathbb{Z}$.

The convolution sum is realized as follows -

1. Invert $h[k]$ about $k=0$ to obtain $h[-k]$.
2. The function $h[n-k]$ is given by $h[-k]$ shifted to the right by n (if n is positive) and to the left (if n is negative) (note the sign of the independent variable).
3. Multiply $x[k]$ and $h[n-k]$ for the same coordinates on the k axis. The value obtained is the response at n i.e., the value of $y[n]$ at a particular n, the value chosen in step 2.

* 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.

* THEORETICAL CALCULATION :

Given, $x[n] = [2, 3, 4]$

$h[n] = [1, 2, 3]$

To find, $y[n]$

(3)

We know that, $y[n] = x[n] * h[n]$. Therefore,

	1	2	3			← $h[n]$
×		2	3	4		← $x[n]$
	2	4	6	0	0	← $x[0] \cdot h[n]$, $i=0$
	0	3	6	9	0	← $x[1] \cdot h[n-1]$, $i=1$
+	0	0	4	8	12	← $x[2] \cdot h[n-2]$, $i=2$
	2	7	16	17	12	← $y[n]$

$$\boxed{y[n] = [2, 7, 16, 17, 12]}$$

* PROGRAM WITH COMMENTS:

```
1  # Import library source files
2  Import numpy as np
3  Import matplotlib.pyplot as plt
4
5  size_x = int (input ("Enter the size of input x[n]: "))
6  x = [0] * (size_x) # Declare an array with the number of
   elements equal to size
7  print ("Enter the elements of the input x[n] one-by-one as
   follows :- ")
8  for i in range (0, size_x, 1):
9      x[i] = input ("Element " + str(i+1) + ": ")
10 print ("The entered input x[n] is :- " + str(x) + "\n")
11
12 plt.subplot (3, 1, 1)
13 plt.xlabel ('n')
14 plt.ylabel ('x[n]')
15 plt.title ('Input x[]'.format(x)) # Formats the specified value(s)
   and insert them inside the string's placeholder.
16 plt.stem (np.arange (0, size_x), x)
17
```

P.T.O.

```
size_h = int(input("Enter the size of impulse response h[n]: "))
h = [0] * (size_h) # Declare an array with the number of elements equals to value of size.
print("Enter the elements of the impulse response h[n]: ")
for i in range(0, size_h, 1):
    h[i] = input("Element " + str(i+1) + ": ")
print("The entered impulse response h[n] is :- " + str(h) + "\n")

plt.subplot(3,1,3)
plt.xlabel('n')
plt.ylabel('h[n]')
plt.title('Impulse Response h{}'.format(h)) # Formats the specified value(s) and insert them inside the string's placeholder.
plt.stem(np.arange(0, size_h), h)

plt.show()

temp = [] # duplicate for impulse response
for i in range(size_x):
    temp = h.copy() # creates a copy of an existing list
    for j in range(len(temp)):
        temp[j] = int(temp[j]) * int(x[i])
    for k in range(i):
        temp.append(0) # Adds required zeros at the end of list
        for l in range(len(temp)-1, 0, -1):
            temp[l] = temp[l-1] # shifts the zeros at the beginning of list
        temp[k] = 0 # prepares for next iteration
    plt.title('Intermediate plot x[{}] * h[n-{}] = {}'.format(i, i, temp)) # Formats the specified value(s) and insert them inside the string's placeholder.
    plt.xlabel('n')
    plt.ylabel('x[{}] * h[n-{}]'.format(i,i))
```

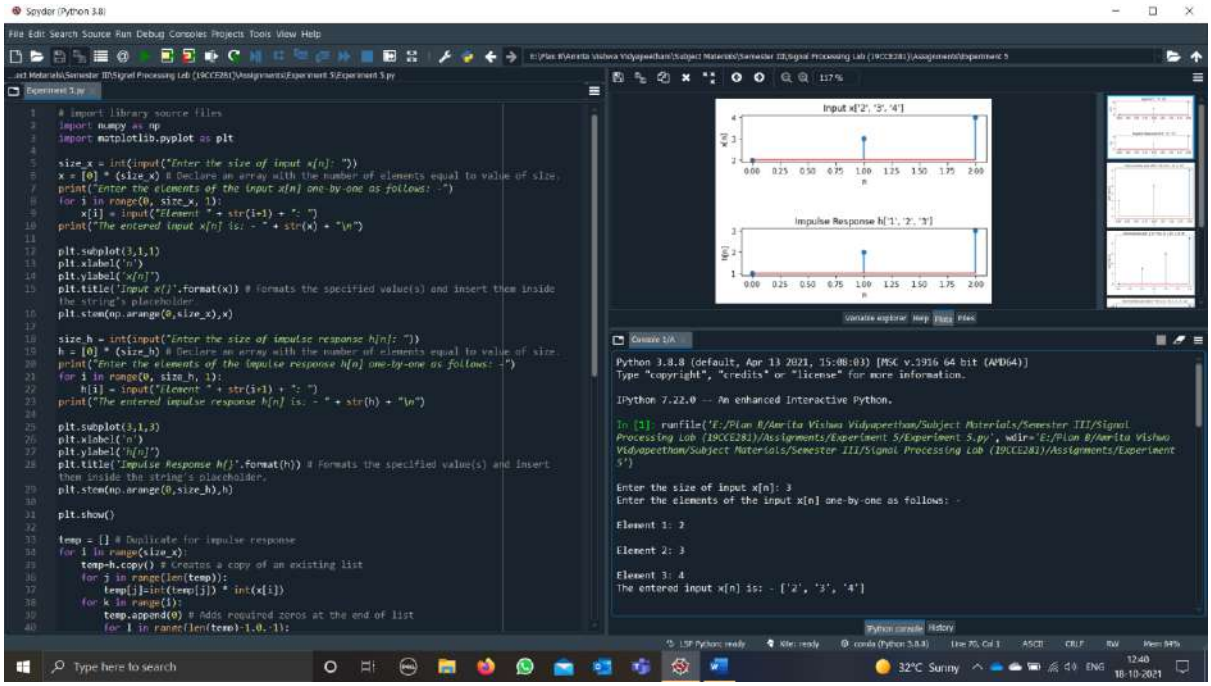
```
    plt.stem(np.arange(0, len(temp)), temp)
    plt.show()

# string padding refers to adding, usually, non-informative characters to a string to one or both ends of it. This is most often done for output formatting and alignment purposes, but it can have useful practical applications. numpy.pad() function is used to pad the numpy arrays.

size = (size_x + size_h) - 1 # Compute the size of system response
x = np.pad(x, (0, size - size_x), 'constant')
h = np.pad(h, (0, size - size_h), 'constant')
y = np.zeros(size, dtype = int) # Returns a new array of given shape and type, with zeros.
iteration = 1 # Variable used for displaying the iteration sequence

for i in range(size):
    for j in range(size):
        if i >= j:
            y[i] = int(y[i] + (int(x[i-j]) * int(h[j])))
            print("Iteration " + str(iteration) + ": " + str(y))
# Display result of each iteration
            iteration += 1
print("\n The system response is :- " + str(y) + "\n")

plt.xlabel('n')
plt.ylabel('y[n]')
plt.title('System Response y[]'.format(y)) # Formats the specified value(s) and insert them inside the string's placeholder.
plt.stem(np.arange(0, size), y)
plt.show()
```

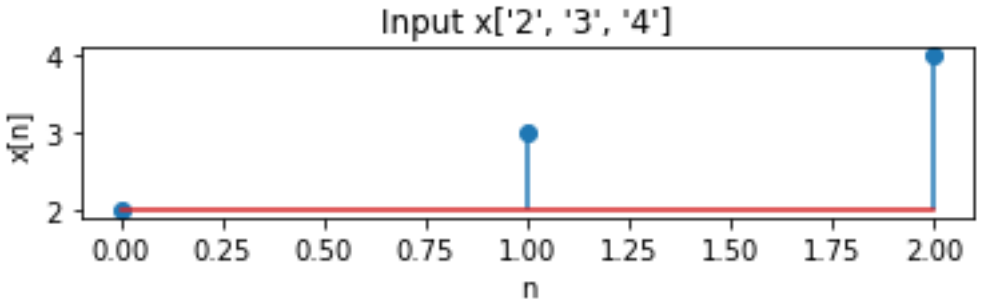
(6)

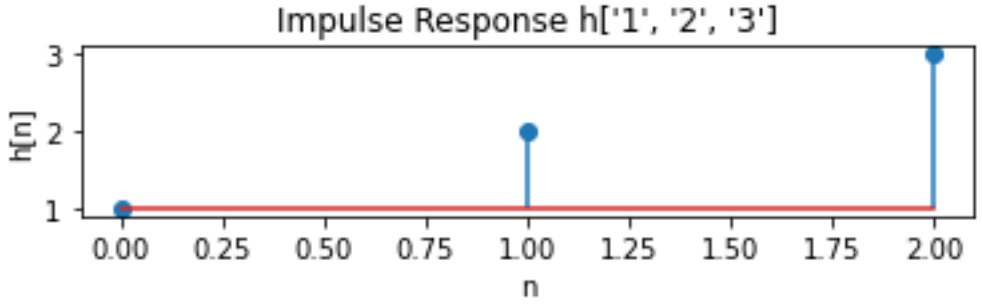
* INFERENCE:

System response $y[n]$ calculated from input $x[n]$ and impulse response $h[n]$, responses sketched and results verified.

Input and Impulse Response







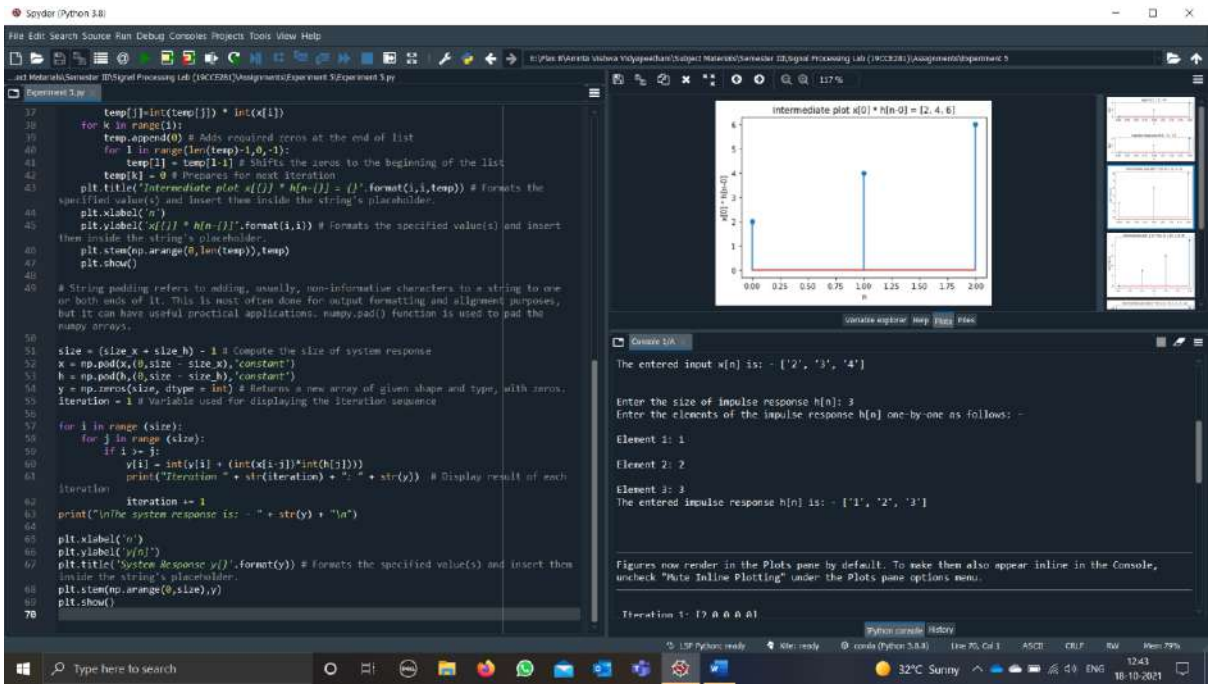
Step 1: Enter the size of input x[n] and declare an array with the number of elements equal to the value of size.

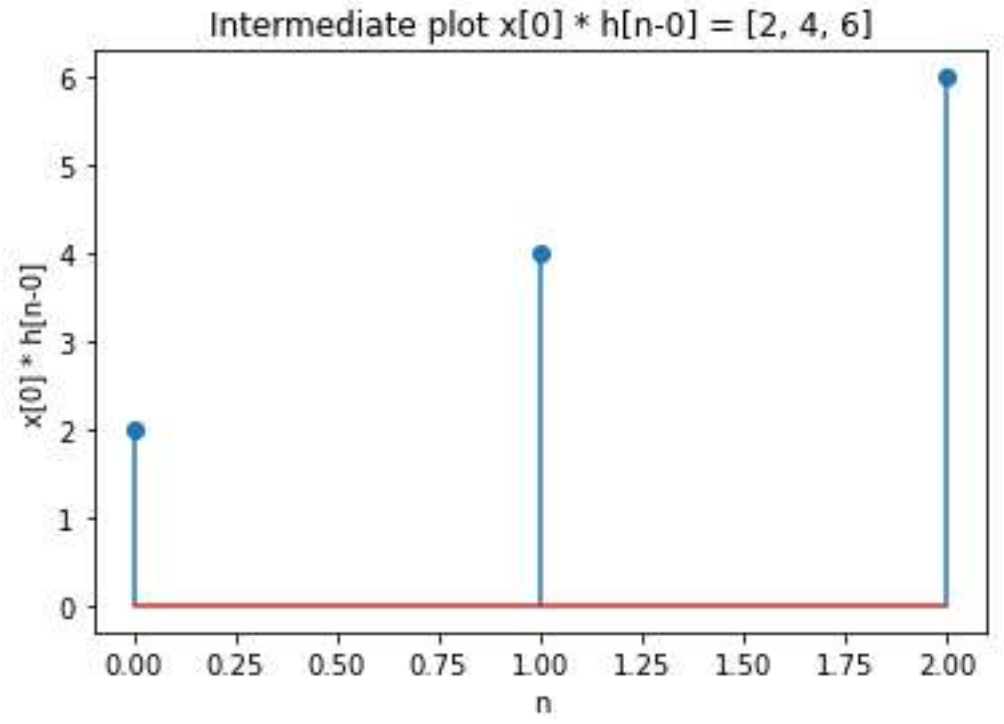
Step 2: Enter the elements of the input x[n].

Step 3: Show the labelled plot x[n].

Step 4: Repeat steps 1, 2 and 3 for impulse response h[n].

Display Intermediate Plot





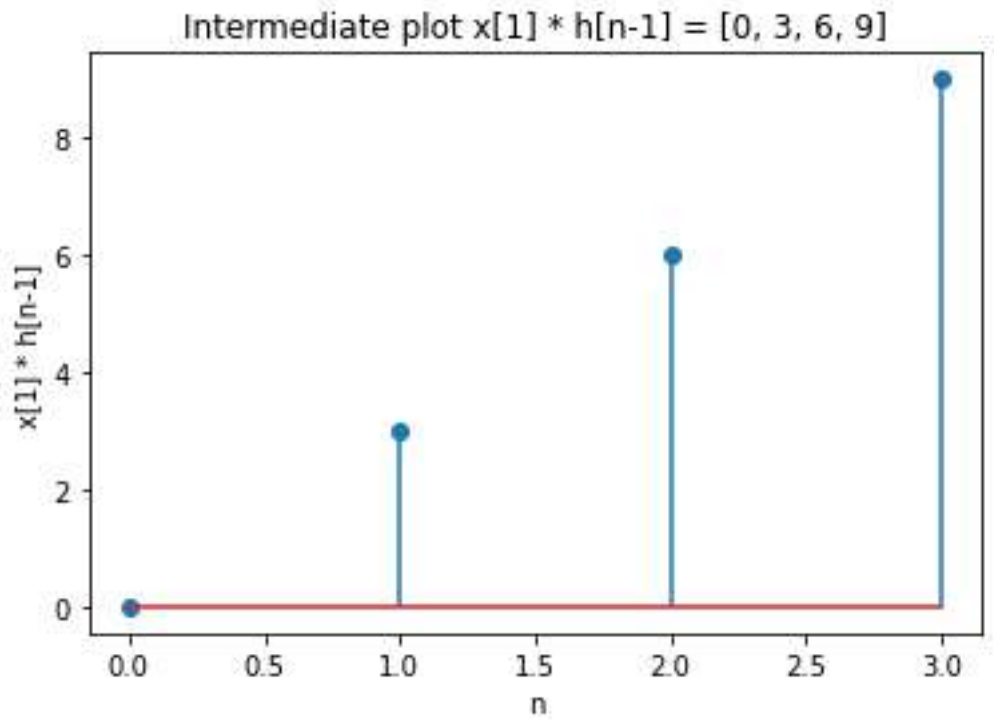
Step 1: Create a duplicate list temp[] and copy the elements impulse response to the duplicate list.

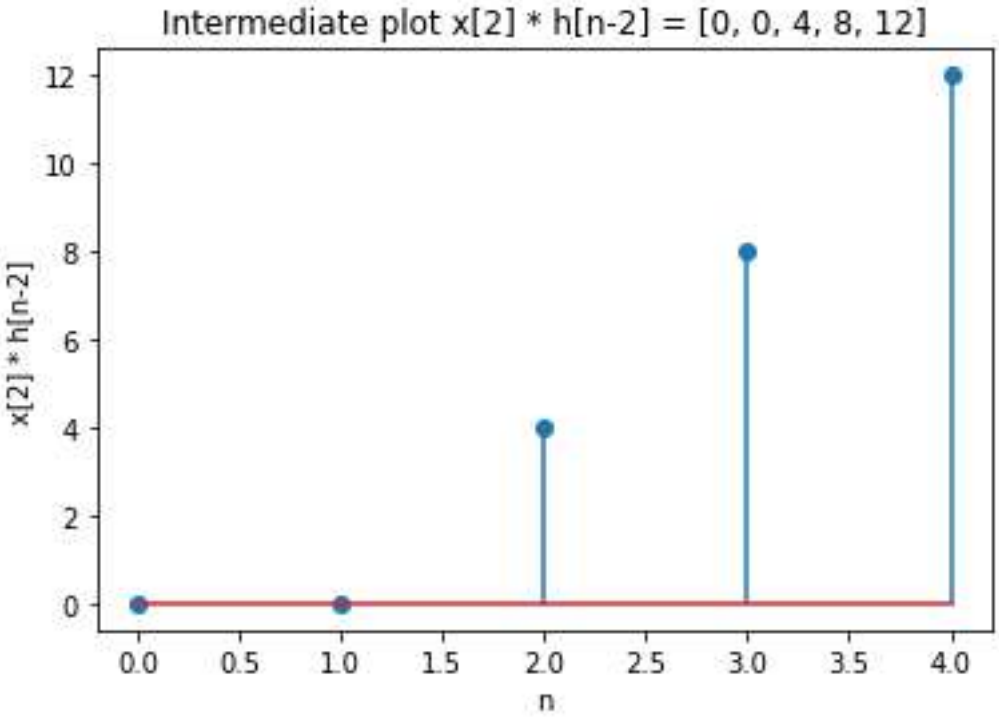
Step 2: Perform convolution sum integral (x[n] * h[n]).

Step 3: Adds required zeros at the end of the list.

Step 4: Shifts the elements to bring the above-added zeros to the beginning of the list.

Step 5: Show the labelled intermediate plot.





For the given input x[n] = [2, 3, 4] and impulse response = h[n] = [1, 2, 3], we have three intermediates:

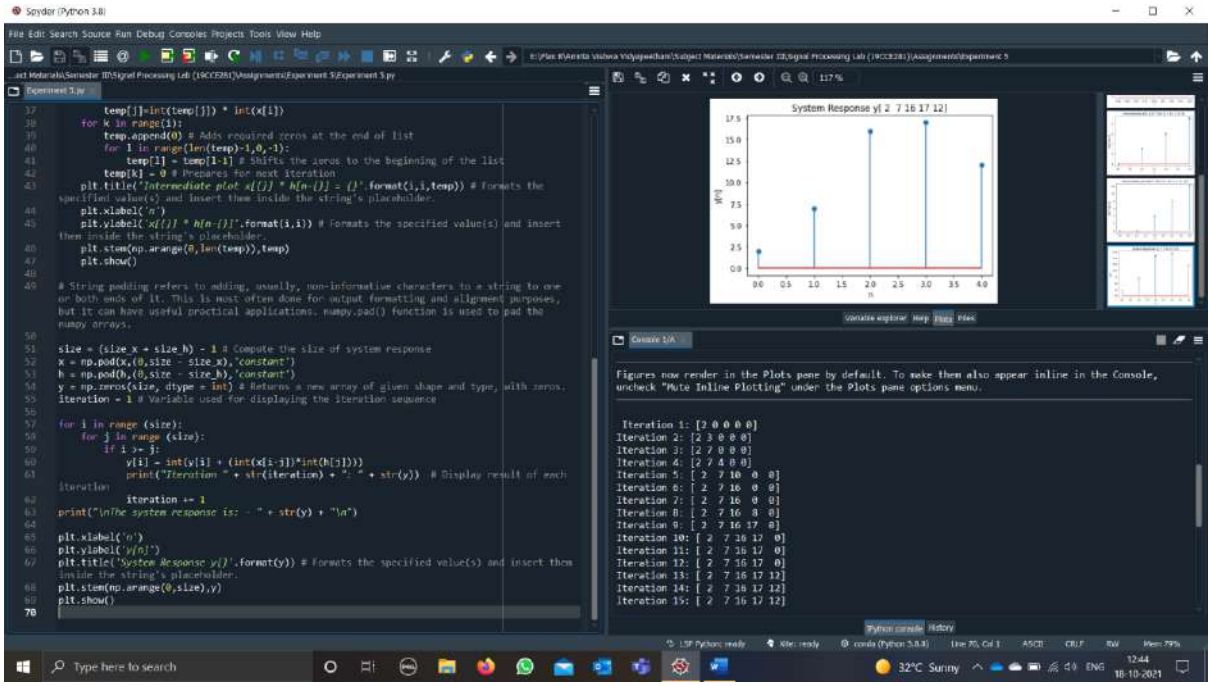
x[0] * h[n] = [2, 4, 6] for i = 0

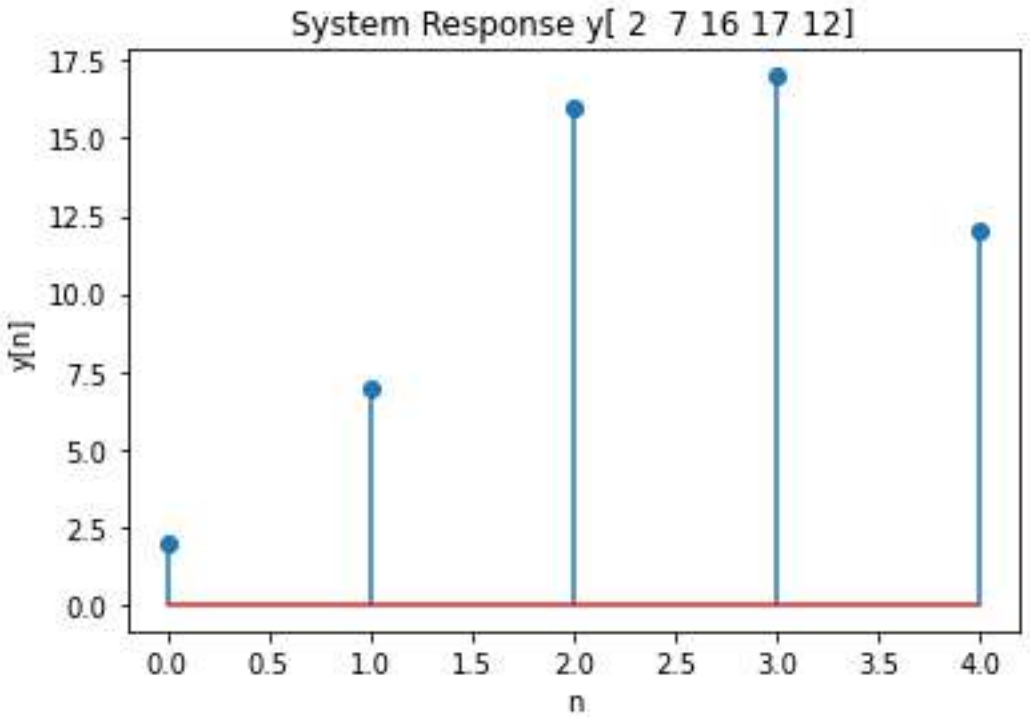
x[1] * h[n-1] = [0, 3, 6, 9] for i = 1

x[2] * h[n-2] = [0, 0, 4, 8, 12] for i = 2

Upon adding these, we compute that y[n] = [2, 7, 16, 17, 12]

Iterations and System Response





Step 1: Compute the size of the system response and pad for input x[] and impulse response h[].

Step 2: Initialize all the elements of system response y[n] as zeros.

Step 3: Using two nested for loops, perform convolution sum integral (x[n] * h[n]).

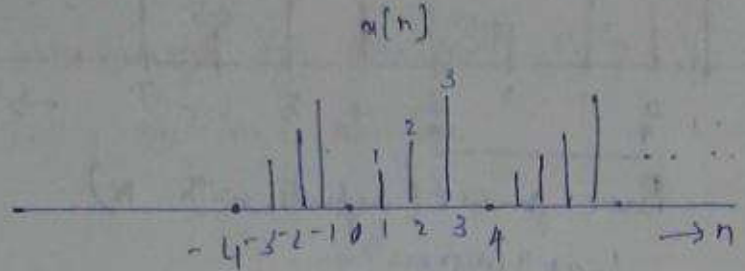
Step 4: Print the result of each iteration.

Step 5: Show the labelled plot for system response.

```
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 5/Experiment 5.py'      ='E:/Plan
B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab
(19CCE281)/Assignments/Experiment 5'

Enter the size of input x[n]: 3
Enter the elements of the input x[n] one-by-one as follows: -

Element 1: 2

Element 2: 3

Element 3: 4
The entered input x[n] is: - ['2', '3', '4']


Enter the size of impulse response h[n]: 3
Enter the elements of the impulse response h[n] one-by-one as follows: -

Element 1: 1

Element 2: 2

Element 3: 3
The entered impulse response h[n] is: - ['1', '2', '3']

Iteration 1: [2 0 0 0 0]
Iteration 2: [2 3 0 0 0]
Iteration 3: [2 7 0 0 0]
Iteration 4: [2 7 4 0 0]
Iteration 5: [ 2  7 10  0  0]
Iteration 6: [ 2  7 16  0  0]
Iteration 7: [ 2  7 16  0  0]
Iteration 8: [ 2  7 16  8  0]
Iteration 9: [ 2  7 16 17  0]
Iteration 10: [ 2  7 16 17  0]
Iteration 11: [ 2  7 16 17  0]
Iteration 12: [ 2  7 16 17  0]
Iteration 13: [ 2  7 16 17 12]
Iteration 14: [ 2  7 16 17 12]
Iteration 15: [ 2  7 16 17 12]

The system response is: - [ 2  7 16 17 12]


In [2]:
```

Determine the DTFS coeff for the periodic signal depicted in fig. below & sketch the spectra.



Sol:

$$N = 4; \quad \Omega_0 = \frac{2\pi}{4} = \frac{\pi}{2}$$

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x(n)\, e^{-jk\Omega_0 n}$$

$$X(k) = \frac{1}{4} \sum_{n=0}^{3} x(n)\, e^{-jk\frac{\pi}{2}n}$$

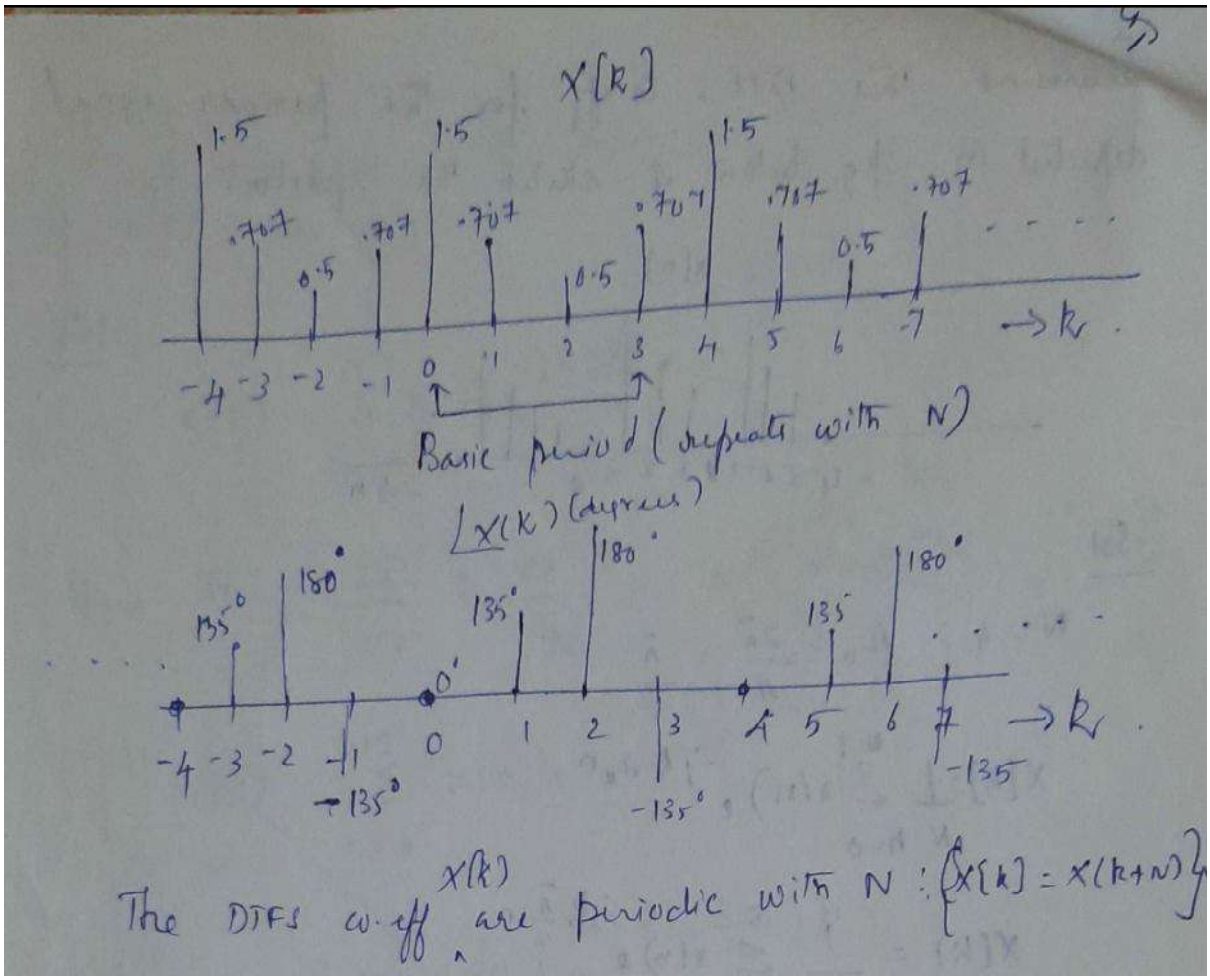
$$= \frac{1}{4}\left[x(0)e^{0} + x(1)e^{-j\frac{\pi}{2}k} + x(2)e^{-jk\frac{\pi}{2}\cdot 2} + x(3)e^{-jk\frac{\pi}{2}\cdot 3} - \right.$$

$$= \frac{1}{4}\left[0 + 1.e^{-jk\frac{\pi}{2}} + 2e^{-jk\pi} + 3e^{-j\frac{3k\pi}{2}}\right] \longleftarrow (1)$$

In this problem, $k = 0, 1, 2, 3$ (as $N = 4$).

Hence evaluate $x[0]$, $x[1]$, $x[2]$, $x[3]$ by putting k value in eqn (1). At the same time evaluate $|x(k)|$ & $\angle x(k)$.

k	$X(k)$	$\lvert X(k)\rvert$	$\angle X(k)$
0	1.5	1.5	0
1	$-0.5+j0.5$	0.707	135° (2.357 radians)
2	-0.5	0.5	180° (π rad)
3	$-0.5-j0.5$	0.707	-135° (-2.357 radians)



The DTFS coeff $X(k)$ are periodic with N : $\{X[k] = X(k+N)\}$

NAME - SANTOSH [CB.EN.U4CCE20053]
DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE)

LAB TITLE AND CODE : SIGNAL PROCESSING LAB 19CCE281

EXPERIMENT NUMBER : 6

DATE : 26/10/2021

FOURIER SERIES REPRESENTATION OF PERIODIC SIGNALS

* AIM :

Obtain the frequency domain representation of the given discrete-time periodic sequence and sketch the input sequence and Fourier coefficients magnitude and phase spectrum

* SOFTWARE REQUIRED :

Spyder IDE (Anaconda3) - Python 3.9.7 (64-bit)

* THEORY :

① Discrete-time signals -

A discrete-time signal of fundamental period N can consist of frequency components $f = \frac{1}{N}, \frac{2}{N}, \cdots, \frac{(N-1)}{N}$ besides $f = 0$, the DC component. Therefore, the Fourier series representation of the discrete-time periodic signal contains only N complex exponential basis functions.

② Fourier series for discrete-time periodic signals -

Given a periodic sequence $x[k]$ with period N, the Fourier series representation for $x[k]$ uses N harmonically related exponential functions -

$$e^{j2\pi kn/N}, \quad k = 0, 1, \cdots, N-1$$

The Fourier series is expressed as -

$$x[k] = \sum_{n=0}^{N-1} c_n e^{j2\pi kn/N}$$

(3) Fourier coefficients -

The Fourier coefficients $\{c_n\}$ are given by -

$$c_n = \frac{1}{N} \sum_{k=0}^{N-1} x[k] e^{-j2\pi kn/N}$$

* GRAPH PLOTTING ALGORITHM :

The following steps are 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.

* THEORETICAL CALCULATION :

Given periodic sequence,

$x[n] = [\ldots\ 5\ 10\ 15\ 20\ 25\ \mathbf{5}\ 10\ 15\ 20\ 25\ 5\ 10\ 15\ 20\ 25\ \ldots]$

(Note: Boldface indicates $n=0$ index).

By keen observation, we find that fundamental period (N) = 5

Angular frequency $(\Omega_0) = \frac{2\pi}{N} = \frac{2\pi}{5}$

$$x[k] = \frac{1}{N} \sum_{n=0}^{4} x[n] e^{-jk\frac{2\pi}{5}n} = \frac{1}{5} \sum_{n=0}^{4} x[n] e^{-jk\frac{2\pi}{5}n}$$

$$= \frac{1}{5} \left[x[0] e^{0} + x[1] e^{-jk\frac{2\pi}{5}} + x[2] e^{-jk\frac{4\pi}{5}} + x[3] e^{-jk\frac{6\pi}{5}} + x[4] e^{-jk\frac{8\pi}{5}} \right]$$

$$= \frac{1}{5} \left[5 + 10 e^{-jk\frac{2\pi}{5}} + 15 e^{-jk\frac{4\pi}{5}} + 20 e^{-jk\frac{6\pi}{5}} + 25 e^{-jk\frac{8\pi}{5}} \right]$$

Taking 5 common, we get -

$$x[k] = 1 + 2e^{-jk\frac{2\pi}{5}} + 3e^{-jk\frac{4\pi}{5}} + 4e^{-jk\frac{6\pi}{5}} + 5e^{-jk\frac{8\pi}{5}}$$

Substituting the values of k in $x[k]$, we find that -

$x[0] = 15 + 0j$, for $k = 0$

$x[1] = -2.5 + 3.44j$, for $k = 1$

$x[2] = -2.5 + 0.81j$, for $k = 2$

$x[3] = -2.5 - 0.81j$, for $k = 3$

$x[4] = -2.5 - 3.44j$, for $k = 4$

Upon computing the magnitude spectrum, we find that -

$|x[0]| = \sqrt{15^2 + 0^2} = 15$

$|x[1]| = \sqrt{(-2.5)^2 + (3.44)^2} = 4.25$

$|x[2]| = \sqrt{(-2.5)^2 + (0.81)^2} = 2.63$

$|x[3]| = \sqrt{(-2.5)^2 + (-0.81)^2} = 2.63$

$|x[4]| = \sqrt{(-2.5)^2 + (-3.44)^2} = 4.25$

Upon computing the phase spectrum, we find that -

$(x[0]) = \tan^{-1}\left(\frac{0}{15}\right) = 0^\circ = 0 \times \frac{\pi}{180}$ radians $= 0$ radians

$(x[1]) = \tan^{-1}\left(\frac{3.44}{-2.5}\right) = -54^\circ = -\frac{3}{54} \times \frac{\pi}{180\ 10}$ radians $= -0.94$ radians

$(x[2]) = \tan^{-1}\left(\frac{0.81}{-2.5}\right) = -18^\circ = -\frac{1}{18} \times \frac{\pi}{180\ 10}$ radians $= -0.31$ radians

$(x[3]) = \tan^{-1}\left(\frac{+0.81}{+2.5}\right) = 54^\circ = \frac{3}{54} \times \frac{\pi}{180\ 10}$ radians $= 0.94$ radians

$(x[4]) = \tan^{-1}\left(\frac{+3.44}{+2.5}\right) = 18^\circ = \frac{1}{18} \times \frac{\pi}{180\ 10}$ radians $= 0.31$ radians

* PROGRAM WITH COMMENTS :

```
1  # import library source files
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # Given Periodic Sequence
```

PTO

④

```
size_input = int(input("Enter the size of input x[n] : "))
user_defined_input = [0] * (size_input)
print("Enter the elements of the input x[n] one-by-one as follows:
for sample in range(0, size_input, 1):
    user_defined_input[sample] = input("element " + str(
user_defined_input) + "\n")
print("\n The entered input x[n] is : - " + str(user_defined_input)
+ "\n")

# Null Array Declaration
x = []
fourier_series = []

# Compute Fundamental Time Period
for sample in range(1, size_input-1):
    x.append(user_defined_input[sample-1])
    if user_defined_input[0] == user_defined_input[sample]:
        breakpoint = sample-2 # How many times does the
sequence repeat?
        break

# Obtain Frequency Domain Representation
boldface = int(input("Enter the element position that
indicates n=0 index : "))
index = int((boldface-1)/len(x))

# Sketch Input sequence
plt.xlabel('n')
plt.ylabel('x[n]')
plt.title("Input sequence : ... {} ... ".format(x))
plt.stem(np.arange(-(index * len(x)), len(x) * (breakpoint'
index)), x* breakpoint)
```

```
plt.show()

# Compute Fourier series
def summation(k):
    sum = 0
    for n in range(len(x)):
        exponential = np.exp(-2j * np.pi * k * n / len(x))
        sum = sum + float(x[n]) * exponential
    return(sum)

# Compute Fourier series coefficients
for k in range(len(x)):
    fourier_series.append(summation(k)/len(x))
print("\nThe Fourier Series Coefficients are as follows: {}".
 format(fourier_series))

# Compute magnitude and Phase spectrum
magnitude_spectrum = []
phase_spectrum = []
for sample in range(len(fourier_series)):
    magnitude_spectrum.append((fourier_series[sample].
real**2 + fourier_series[sample].imag**2)**0.5)
    phase = np.arctan(fourier_series[sample].imag /
fourier_series[sample].real)
    phase_spectrum.append(phase)

# sketch magnitude Spectrum
plt.xlabel('k')
plt.ylabel('|x[k]|')
plt.title("magnitude spectrum : ...{}...".format
    (magnitude_spectrum))
plt.stem(np.arange(0, len(magnitude_spectrum)),
    magnitude_spectrum)
```

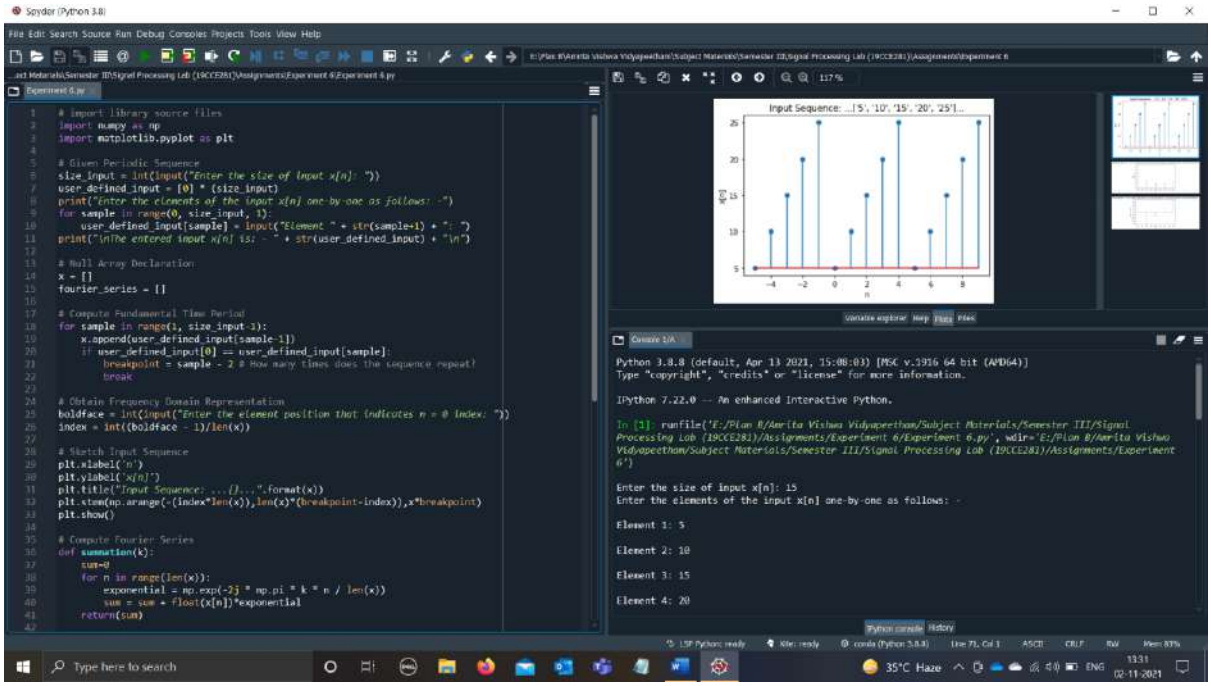
```
61 plt.show()
62 print("\n The Magnitude Spectrum is as follows : {}".
       format(magnitude_spectrum))
63
64 # sketch Phase Spectrum
65 plt.xlabel('k')
66 plt.ylabel('Angle(in radians)')
67 plt.title("Phase spectrum : ... {} ...".format(phase_spectrum))
68 plt.stem(np.arange(0, len(phase_spectrum)), phase_spectrum)
69 plt.show()
70 print("\n The Phase Spectrum is as follows : {}".format
       (phase_spectrum))
71
```

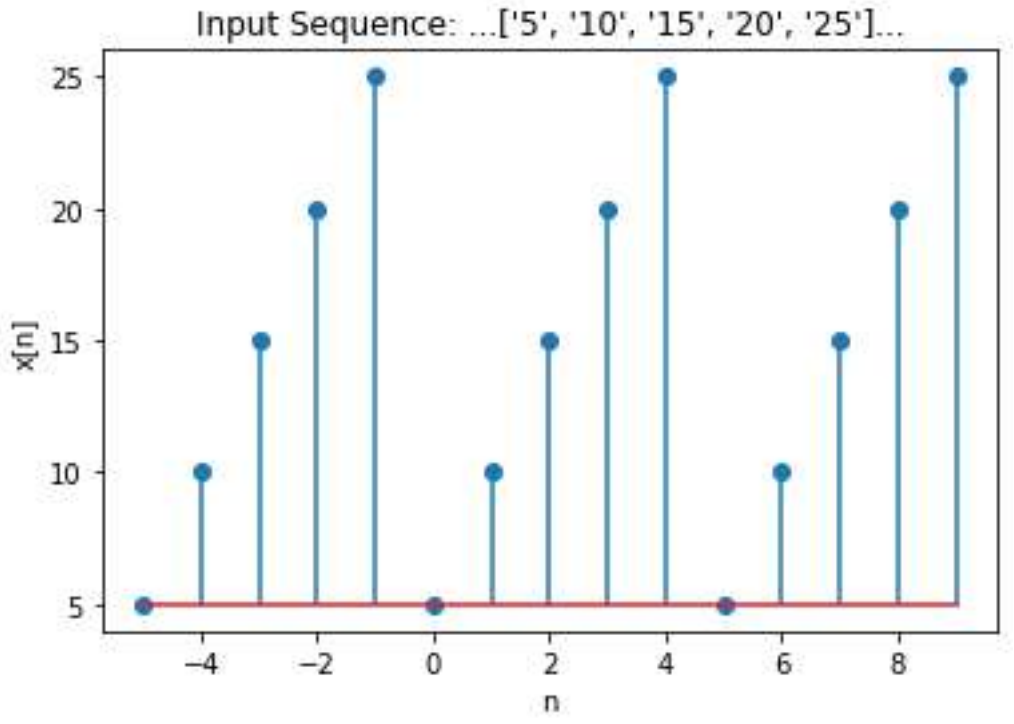
* **INFERENCE :**

For the given input periodic sequence x[n], obtain frequency domain representation along with fundamental time-period and compute Fourier series, magnitude and phase spectrum.

RESULTS
VERIFIED.

Given Discrete Periodic Sequence





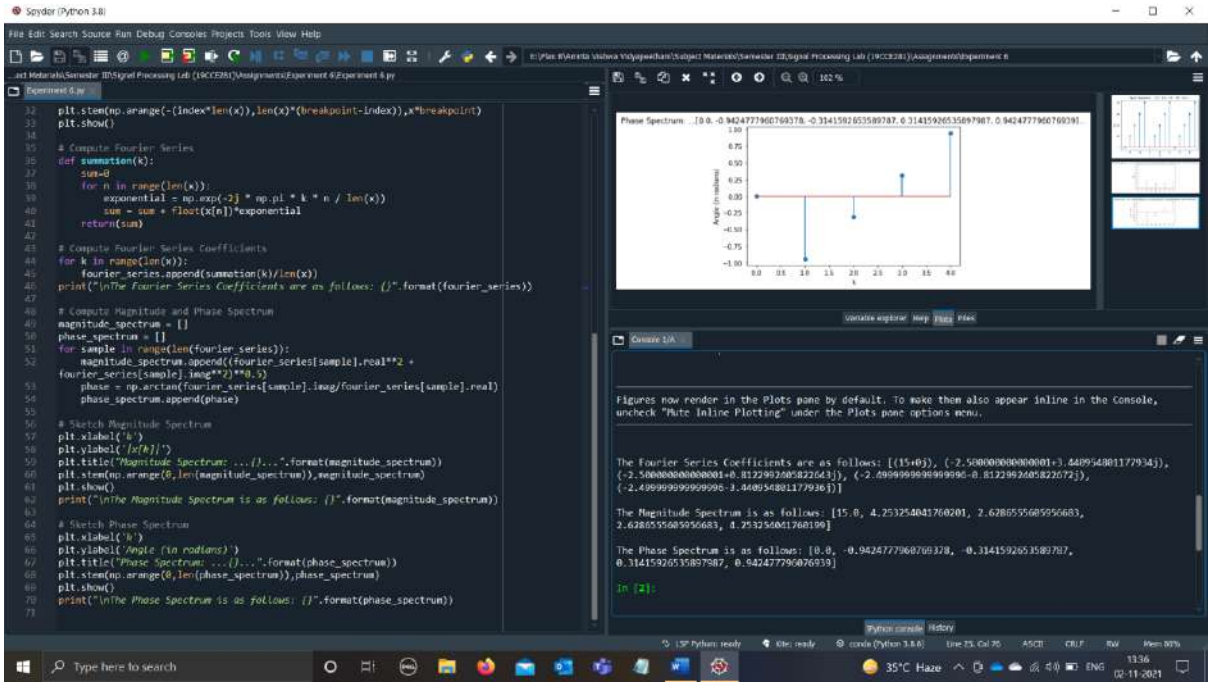
Step 1: Enter the size of input x[n] and declare an array with the number of elements equal to the value of size.

Step 2: Enter the elements of the input x[n] along with the element position that indicates n = 0 index.

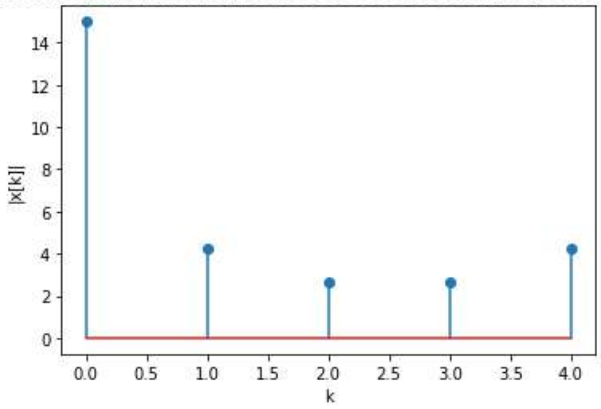
Step 3: Compute fundamental time period and obtain frequency domain representation.

Step 4: Print the input sequence and show its labelled plot x[n].

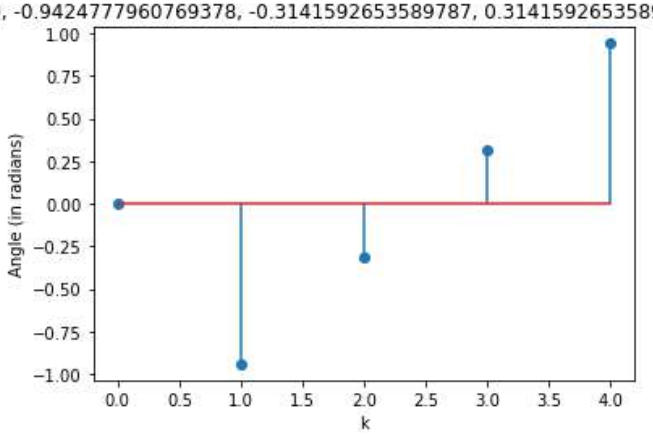
Plot Magnitude and Phase Spectrum



Magnitude Spectrum: ...[15.0, 4.253254041760201, 2.6286555605956683, 2.6286555605956683, 4.253254041760199]...



Phase Spectrum: ...[0.0, -0.9424777960769378, -0.3141592653589787, 0.31415926535897987, 0.942477796076939]...



Step 1: Compute Fourier series and their coefficients.

Step 2: Compute magnitude and phase spectrum.

Step 3: Sketch the two spectrums and print them.

```
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 6/Experiment 6.py'      ='E:/Plan
B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab
(19CCE281)/Assignments/Experiment 6'

Enter the size of input x[n]: 15
Enter the elements of the input x[n] one-by-one as follows: -

Element 1: 5

Element 2: 10

Element 3: 15

Element 4: 20

Element 5: 25

Element 6: 5

Element 7: 10

Element 8: 15

Element 9: 20

Element 10: 25

Element 11: 5

Element 12: 10

Element 13: 15

Element 14: 20

Element 15: 25

The entered input x[n] is: - ['5', '10', '15', '20', '25', '5', '10', '15', '20', '25',
'5', '10', '15', '20', '25']


Enter the element position that indicates n = 0 index: 6

The Fourier Series Coefficients are as follows: [(15+0j),
(-2.500000000000001+3.440954801177934j), (-2.500000000000001+0.8122992405822643j),
```

```
(-2.4999999999999996-0.8122992405822672j), (-2.499999999999996-3.440954801177936j)]

The Magnitude Spectrum is as follows: [15.0, 4.253254041760201, 2.6286555605956683,
2.6286555605956683, 4.253254041760199]

The Phase Spectrum is as follows: [0.0, -0.9424777960769378, -0.3141592653589787,
0.31415926535897987, 0.942477796076939]

In [2]:
```

Expt 7: <u>PROPERTIES OF DTFS: TIME SHIFTING</u>

Verify Time shifting property for the sequence y[n] = x[n-2] where x[n] = […. **9** 18 27 36 45 …..] (*Note: Boldface indicates n=0 index).*

Evaluate Fourier coefficients of (i) x[n] using DTFS and (ii) y[n] using time shifting property of DTFS.

Sketch the input sequence x[n], y[n] and Fourier coefficients of x[n] and y[n].

NAME - SANTOSH - [CB.EN.U4CCE20053]

DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING

LAB TITLE AND CODE : SIGNAL PROCESSING LAB 19CCE281

EXPERIMENT NUMBER : 7

DATE : 07/10/2021

TIME-SHIFTING PROPERTY OF DISCRETE-TIME FOURIER SERIES (DTFS)

* **AIM :**

Given an input sequence $x[n]$, verify the time-shifting property and evaluate Fourier coefficients for the resulting sequence $y[n]$ using DTFS.

* **SOFTWARE REQUIRED :**

Spyder IDE (Anaconda 3) - Python 3.8.8 (64-bit)

* **THEORY :**

① *Discrete-time signals* -

A discrete-time signal of fundamental period N can consist of frequency components $f = \frac{1}{N}, \frac{2}{N}, \cdots, \frac{(N-1)}{N}$ besides $f=0$, the DC component. Therefore, the Fourier series representation of the discrete-time periodic signal contains only N complex exponential basis functions.

② *Fourier series for discrete-time periodic signals* -

Given a periodic sequence $x[k]$ with period N, the Fourier series representation for $x[k]$ uses N harmonically related exponential functions -

$$e^{j2\pi kn/N}, \quad k = 0, 1, \cdots, N-1$$

The Fourier series is expressed as -

$$x[k] = \sum_{n=0}^{N-1} c_n e^{j2\pi kn/N}$$

(3) Fourier coefficients -

The Fourier coefficients $\{c_n\}$ are given by -

$$c_n = \frac{1}{N} \sum_{k=0}^{N-1} x[k] e^{-j 2\pi kn/N}$$

(4) Time shifting -

Let n_0 be any integer. If $x[n]$ is a discrete-time signal to period N, then $y[n] = x[n-n_0]$. Therefore, the fourier series is -

$$x[n] \longleftrightarrow x[k]$$

which gives, $x[n-n_0] \xleftrightarrow{FS} e^{-j 2\pi k n_0/N} x[k]$,

the time-shifted Fourier coefficients.

* GRAPH PLOTTING ALGORITHM :

The following steps are 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.

* THEORETICAL CALCULATION :

Given periodic sequence,

$x[n] = [\ldots\ 9\ \ 18\ \ 27\ \ 36\ \ \mathbf{45}\ \ \ldots]$

(Note : Boldface indicates n=0 index).

By keen observation, we find that fundamental period, $N = 5$

Angular frequency, $\Omega_0 = \frac{2\pi}{N} = \frac{2\pi}{5}$.

$$x[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-jk\frac{2\pi}{5}n} = \frac{1}{5} \sum_{n=0}^{4} x[n] e^{-jk\frac{2\pi}{5}n}$$

$$= \frac{1}{5}\left[x[0]e^{0} + x[1]e^{-jk\frac{2\pi}{5}} + x[2]e^{-jk\frac{4\pi}{5}} + x[3]e^{-jk\frac{6\pi}{5}} + x[4]e^{-jk\frac{8\pi}{5}}\right]$$

$$= \frac{1}{5}\left[9 + 18e^{-jk\frac{2\pi}{5}} + 27e^{-jk\frac{4\pi}{5}} + 36e^{-jk\frac{6\pi}{5}} + 45e^{-jk\frac{8\pi}{5}}\right]$$

Taking 9 common,

$$x[k] = \frac{9}{5}\left[1 + 2e^{-jk\frac{2\pi}{5}} + 3e^{-jk\frac{4\pi}{5}} + 4e^{-jk\frac{6\pi}{5}} + 5e^{-jk\frac{8\pi}{5}}\right]$$

Substituting the values of k in $x[k]$, we find that -

$x[0] = 21 + 0j$, for $k=0$

$x[1] = -4.5 + 6.2j$, for $k=1$

$x[2] = -4.5 + 1.5j$, for $k=2$

$x[3] = -4.5 - 1.5j$, for $k=3$

$x[4] = -4.5 - 6.2j$, for $k=4$

Upon computing the magnitude spectrum, we find that -

$|x[0]| = \sqrt{(21)^2 + (0)^2} = 21.0$

$|x[1]| = \sqrt{(-4.5)^2 + (6.2)^2} = 7.66$

$|x[2]| = \sqrt{(-4.5)^2 + (1.5)^2} = 4.73$

$|x[3]| = \sqrt{(-4.5)^2 + (-1.5)^2} = 4.73$

$|x[4]| = \sqrt{(-4.5)^2 + (-6.2)^2} = 7.66$

Upon computing the phase spectrum, we find that -

$$(x[n]) = \tan^{-1}\left(\frac{y}{x}\right) = \_\_^{\circ} = \_\_ \times \frac{\pi}{180} \text{ radians}$$

Therefore,

$(x[0]) = 0$ radians

$(x[1]) = -0.94$ radians

$(x[2]) = -0.31$ radians

$(x[3]) = 0.31$ radians

$(x[4]) = 0.94$ radians

Given shifting factor, $n_0 = 2$.
Therefore, obtained resulting sequence,

$y[n] = [\ldots \mathbf{36}\ 45\ 9\ 18\ 27 \ldots]$

(Note: Boldface indicates $n=0$ index)

Again, fundamental period $(N) = 5$

Angular frequency $(\Omega_0) = \frac{2\pi}{N} = \frac{2\pi}{5}$

Therefore, Fourier coefficients of $y[n]$ using time shifting property of DTFS are -

$y[0] = e^{-j2\pi k n_0/N} \cdot x[0] = e^{-j2\pi(0)2/5} \cdot (27+0j) = (27+0j)$

$y[1] = e^{-j2\pi(1)2/5} \cdot x[1] = e^{-j2\pi(1)2/5} \cdot (-4.5+6.2j) = (7.28-2.37j)$

$y[2] = e^{-j2\pi(2)2/5} \cdot x[2] = e^{-j2\pi(2)2/5} \cdot (-4.5+1.5j) = (-2.78-3.83j)$

$y[3] = e^{-j2\pi(3)2/5} \cdot x[3] = e^{-j2\pi(3)2/5} \cdot (-4.5-1.5j) = (-2.78+3.83j)$

$y[4] = e^{-j2\pi(4)2/5} \cdot x[4] = e^{-j2\pi(4)2/5} \cdot (-4.5-6.2j) = (7.28+2.37j)$

Upon computing the magnitude spectrum, we find that -

$|y[0]| = \sqrt{(27)^2+(0)^2} = 27.0$

$|y[1]| = \sqrt{(7.28)^2+(-2.37)^2} = 7.66$

$|y[2]| = \sqrt{(-2.78)^2+(-3.83)^2} = 4.73$

$|y[3]| = \sqrt{(-2.78)^2+(3.83)^2} = 4.73$

$|y[4]| = \sqrt{(7.28)^2+(2.37)^2} = 7.66$

Upon computing the phase spectrum, we find that -

$(y[n]) = \tan^{-1}\left(\frac{y}{x}\right) = \_\_^{\circ} = \_\_ \times \frac{\pi}{180}$ radians

$(y[0]) = 0.0$ radians

$(y[1]) = -0.31$ radians

$(y[2]) = 0.94$ radians

$(y[3]) = -0.94$ radians

$(y[4]) = 0.31$ radians

* PROGRAM WITH COMMENTS:

```
1  # Import library source files
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # Given Input Sequence x[n]
6  size_input = int (input ("Enter the size of input x[n]: "))
7  user_defined_input = [0] * (size_input)
8  print ("Enter the elements of the input x[n] one-by-one
   as follows :- ")
9  for sample in range (0, size_input, 1):
10     user_defined_input [sample] = input ("Element " +
   str (sample + 1) + ": ")
11 print ("\n The entered input x[n] is : -" + str (user_defined_
   input) + "\n")
12
13 # Null Array Declaration for Input Sequence x[n]
14 x = []
15 fourier_series_x = []
16 magnitude_spectrum_x = []
17 phase_spectrum_x = []
18
19 # Null Array Declaration for Resulting sequence y[n]
20 y = []
21 fourier_series_y = []
22 magnitude_spectrum_y = []
23 phase_spectrum_y = []
24
25 # Compute Fundamental Time Period for x[n]
26 for sample in range (1, size_input - 1):
27     x.append (user_defined_input [sample-1])
28     if user_defined_input [0] == user_defined_input [sample]:
29     → breakpoint = sample - 2 # How many times does the sequence repeat?
30         break
```

```
31
32 # Obtain frequency domain representation
33 boldface = int (input ("Enter the element position that
   indicates n=0 index : "))
34 index = int ((boldface -1) / len (x))
35
36 # sketch Input sequence x[n]
37 plt. xlabel ('n')
38 plt. ylabel ('x[n]')
39 plt. title ("Input sequence : ... {} ... ". format (x))
40 plt. stem (np. arange (-(index * len(x)), len(x) *(breakpoint -index),
   x * breakpoint)
41 plt. show ()
42
43 # Compute Fourier series for x[n]
44 def summation (k):
45     sum = 0
46     for n in range (len(x)):
47         exponential = np. exp (-2j * np.pi * k * n / len (x))
48         sum = sum + float (x[n]) * exponential
49     return (sum)
50
51 # Compute Fourier coefficients for x[n]
52 for k in range (len(x)):
53     fourier_series_x. append (summation (k) / len(x))
54 print ("\n The Fourier coefficients for x[n] are as follows : {}
   ". format (fourier_series_x))
55
56 # sketch Fourier coefficients of x[n]
57 plt. xlabel ('n')
58 plt. ylabel ('x[n]')
59 plt. title ("Fourier coefficients of x[n]")
60 plt. stem (np. arange (0, len (fourier_series_x)). np. real (fourier-
   series_x))
```

```
plt.show()

# Compute Magnitude and Phase Spectrum for x[n]
for sample in range(len(fourier_series_x)):
    magnitude_spectrum_x.append((fourier_series_x[sample].real**2 + fourier_series_x[sample].imag**2)**0.5)
    phase = np.arctan(fourier_series_x[sample].imag/fourier_series_x[sample].real)
    phase_spectrum_x.append(phase)

# Sketch Magnitude Spectrum of x[n]
plt.xlabel('k')
plt.ylabel('|x[k]|')
plt.title("Magnitude Spectrum of x[n]: ...{}...".format(magnitude_spectrum_x))
plt.stem(np.arange(0, len(magnitude_spectrum_x)), magnitude_spectrum_x)
plt.show()
print("\n The Magnitude Spectrum of x[n] is as follows: {}".format(magnitude_spectrum_x))

# Sketch Phase Spectrum of x[n]
plt.xlabel('k')
plt.ylabel('Angle (in radians)')
plt.title("Phase Spectrum of x[n]: ...{}...".format(phase_spectrum_x))
plt.stem(np.arange(0, len(phase_spectrum_x)), phase_spectrum_x)
plt.show()
print("\n The Phase Spectrum of x[n] is as follows: {}".format(phase_spectrum_x))

Intermediate = []
```

```
87  # Copy Input sequence to Temporary sequence intermediate [y]
88  for sample in range (size_input):
89      intermediate . append (user_defined_input [sample])
90
91  # Input Shifting Factor
92  shifting_factor = int (input ("\n Enter the shifting factor : "))
93
94  # shift Temporary sequence by "shifting_factor" of units
95  for i in range (shifting_factor):
96      intermediate . append (0) # Adds required zeros at the end
    of list
97      for j in range (len (intermediate) -1, 0, -1):
98          intermediate [j] = intermediate [j-1] # shifts the zeros
    at the beginning of list
99      intermediate [i] = 0 # Prepares for next iteration
100
101 # Place rest of elements in relevant position
102 temp = size_input - shifting_factor
103 for k in range (shifting_factor)
104     intermediate [k] = user_defined_input [temp]
105     temp = temp + 1
106
107 # Compute Fundamental Time Period for y[n]
108 for sample in range (1, size_input -1):
109     y.append (intermediate [sample -1])
110     if user_defined_input [0] == user_defined_input [sample]:
111         break
112
113 print ("\n The resulting sequence y[n] is :-" + str(y) + "\n")
114
115 # sketch resulting sequence y[n]
116 plt. xlabel ('n')
117 plt. ylabel ('y[n]')
```

```
plt.title("Resulting sequence: ...{}... ".format(y))
plt.stem(np.arange(-index*len(y)), len(y)*(breakpoint-index),
y*breakpoint)
plt.show()

# Compute Fourier coefficients for y[n]
for k in range(len(x))
    fourier_series_y.append(np.exp(-2j*np.pi*k*
shifting_factor/len(x))*fourier_series_x[k])
print("\n The Fourier coefficients for y[n] are as follows: {}"
.format(fourier_series_y))

# sketch Fourier coefficients of y[n]
plt.xlabel('n')
plt.ylabel('y[n]')
plt.title("Fourier coefficients of y[n]")
plt.stem(np.arange(0, len(fourier_series_y)),
np.real(fourier_series_y))
plt.show()

# Compute magnitude and Phase Spectrum for y[n]
for sample in range(len(fourier_series_y)):
    magnitude_spectrum_y.append((fourier_series_y[sample]
.real**2 + fourier_series_y[sample].imag**2)**0.5)
    phase=np.arctan(fourier_series_y[sample].imag/
fourier_series_y[sample].real)
    phase_spectrum_y.append(phase)

# sketch magnitude Spectrum of y[n]
plt.xlabel('k')
plt.ylabel('|y[k]|')
plt.title("magnitude Spectrum of y[n]:...{}...".format
(magnitude_spectrum_y))
```

```
144 plt.stem(np.arange(0, len(magnitude_spectrum_y)),
    magnitude_spectrum_y)
145 plt.show()
146 print("\n The magnitude spectrum of y[n] is as follows: {}".
    format(magnitude_spectrum_y))
147
148 #sketch Phase Spectrum of y[n]
149 plt.xlabel('k')
150 plt.ylabel('Angle(in radians)')
151 plt.title("Phase Spectrum of y[n]:...{}...".format
    (phase_spectrum_y))
152 plt.stem(np.arange(0, len(phase_spectrum_y)), phase_spectrum
153 plt.show()
154 print("\n The Phase Spectrum of y[n] is as follows: {}"
    .format(phase_spectrum_y))
155
```

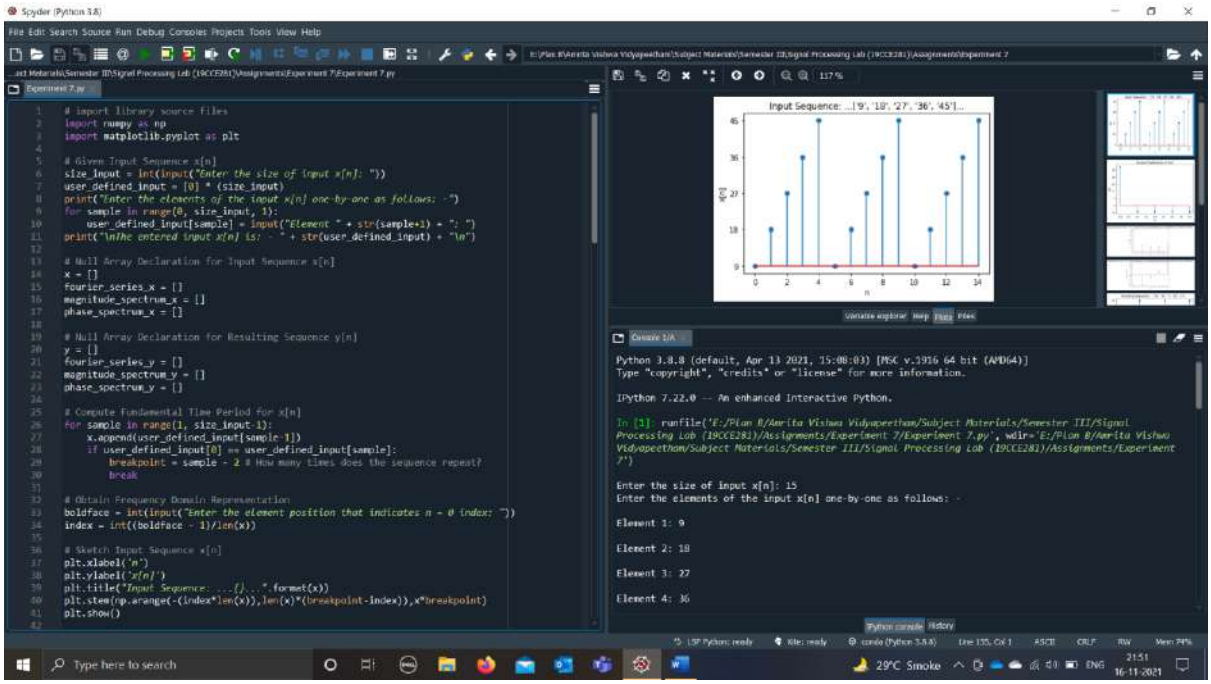
* **INFERENCE:**

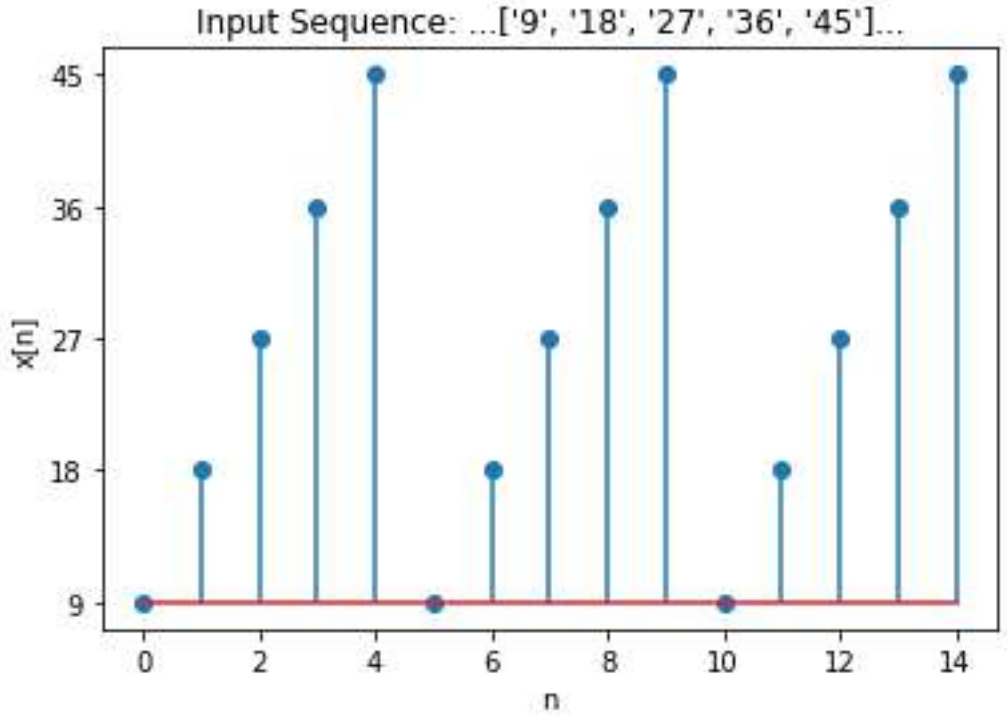
For the given input sequence x[n], demonstrated the time-shifting property and compute Fourier coefficients for the resulting sequence y[n] using DTFS.

RESULTS

VERIFIED.

Given Input Sequence





Step 1: Import library source files, numpy and matplotlib.pyplot.

Step 2: Enter the size of the input and declare an array with the number of elements equal to the value of size.

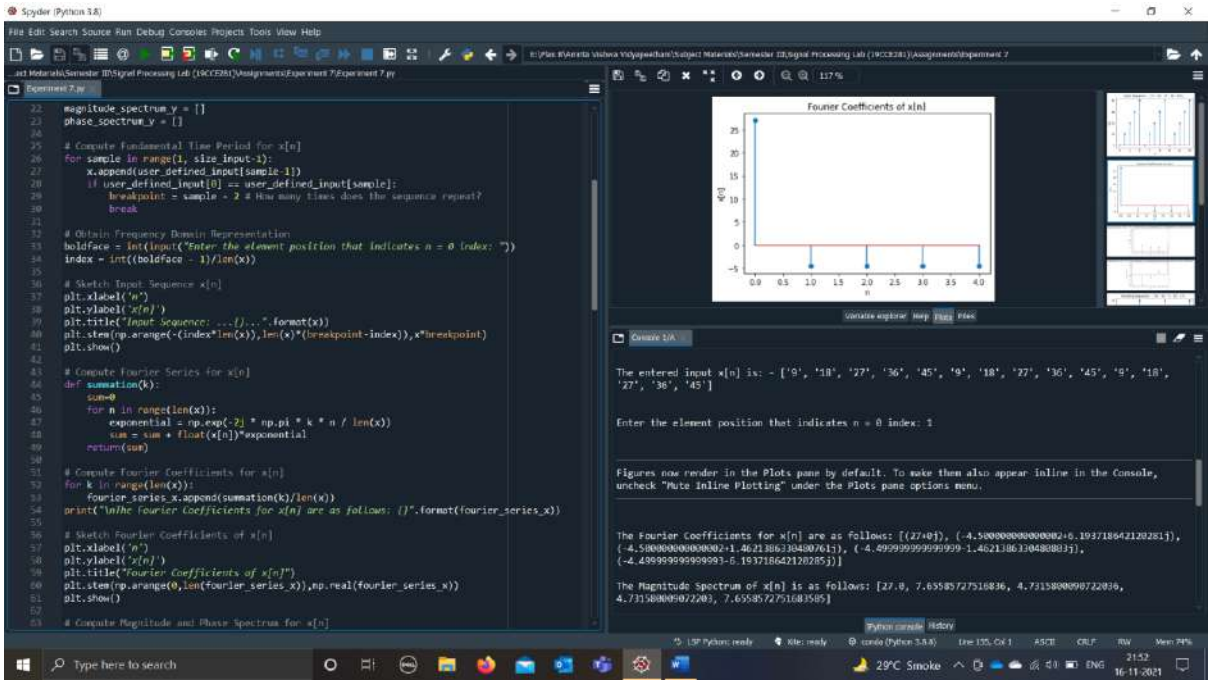
Step 3: Enter the elements of the input along with the element position that indicates n = 0 index.

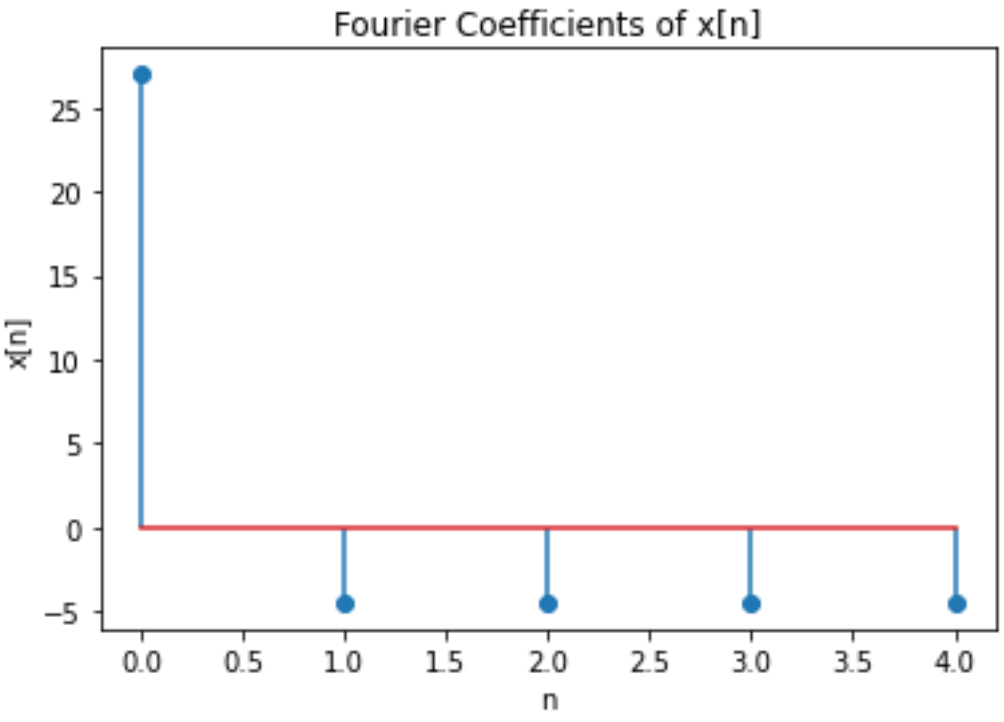
Step 4: Declare required null arrays required for input sequence x[n] and resulting sequence y[n].

Step 5: Compute fundamental time period and obtain frequency domain representation.

Step 6: Print the input sequence and show its labelled plot x[n].

Plot Fourier Coefficients of x[n]



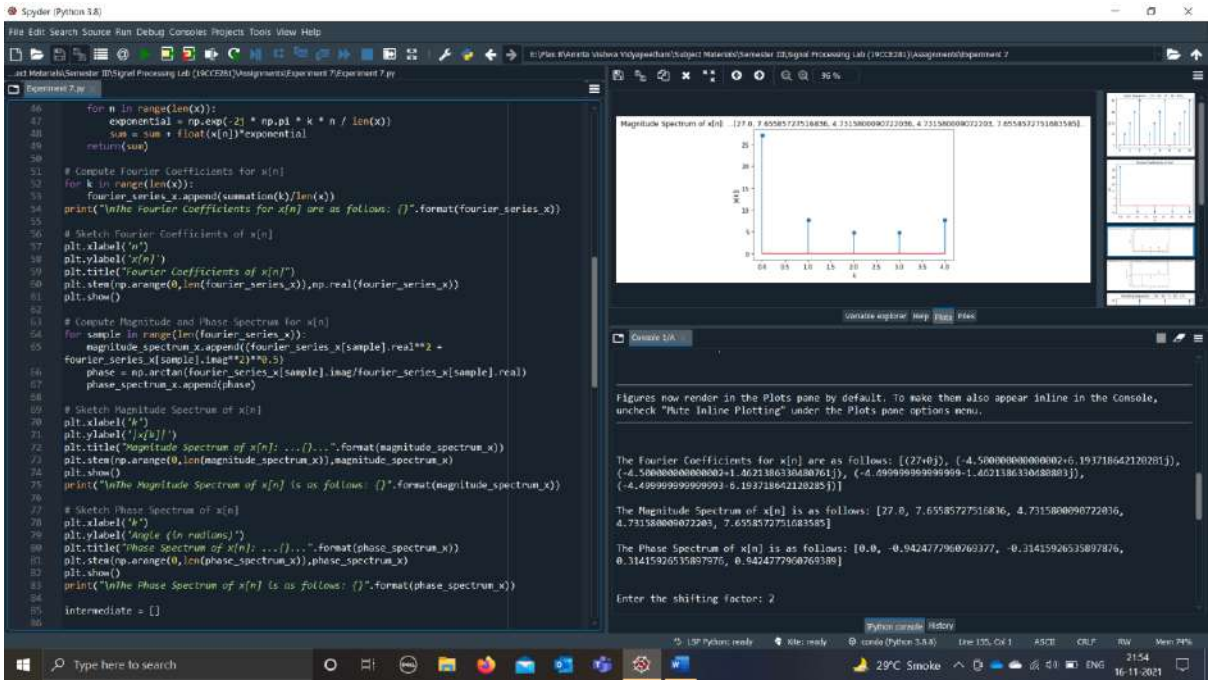


Step 1: Compute Fourier series for x[n].

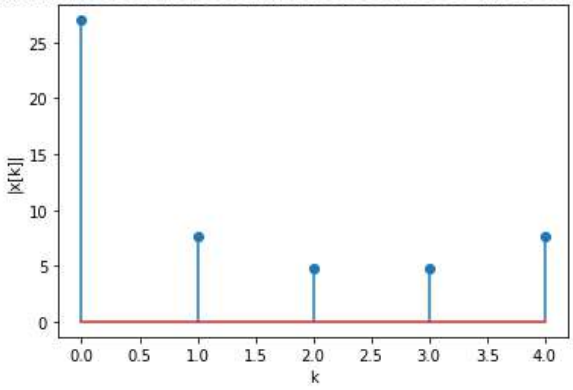
Step 2: Compute Fourier coefficients for x[n].

Step 3: Sketch Fourier coefficients of x[n].

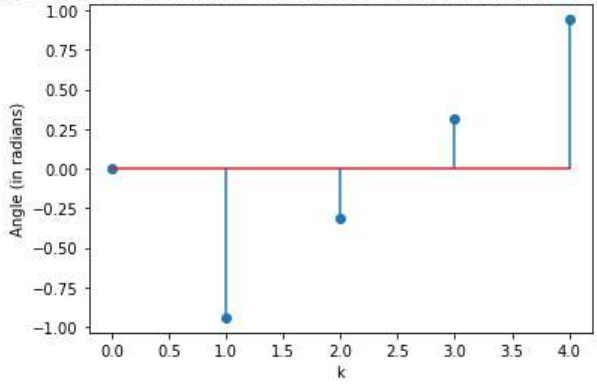
Plot Magnitude and Phase Spectrum of x[n]



Magnitude Spectrum of x[n]: ...[27.0, 7.65585727516836, 4.7315800090722036, 4.731580009072203, 7.6558572751683585]...



Phase Spectrum of x[n]: ...[0.0, -0.9424777960769377, -0.31415926535897876, 0.31415926535897976, 0.9424777960769389]...

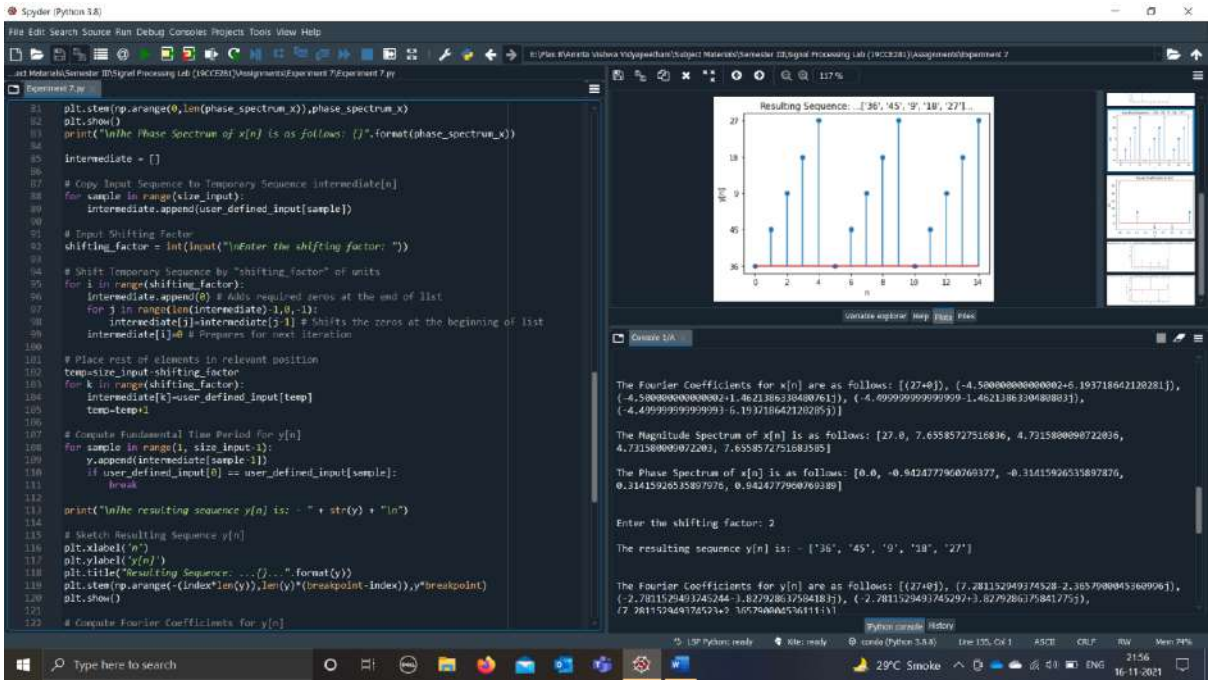


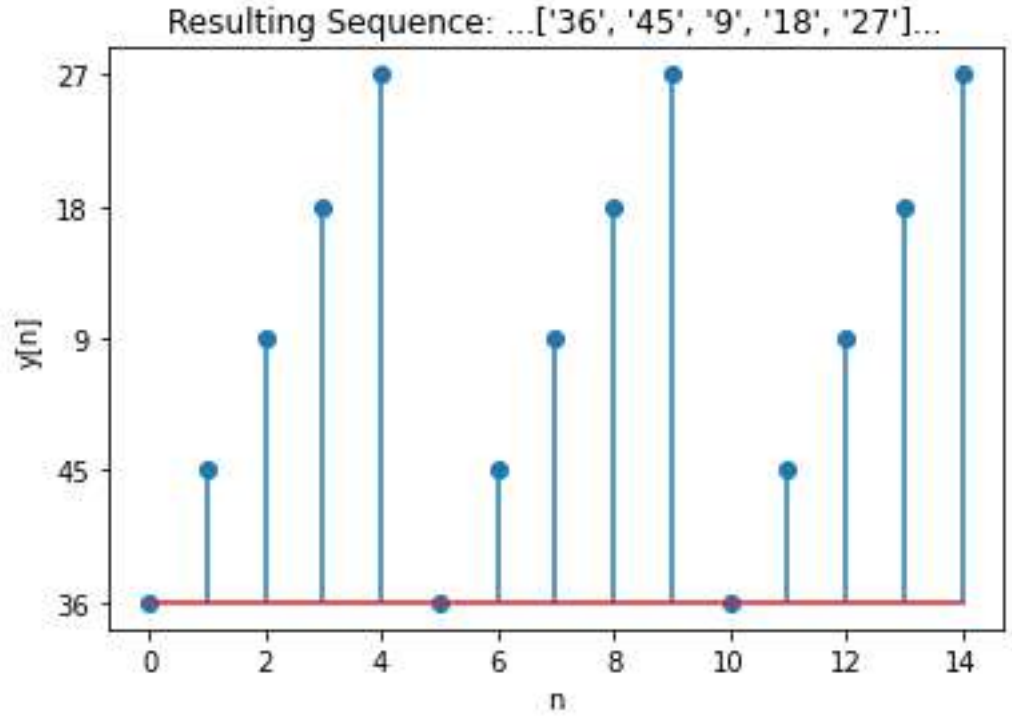
Step 1: Compute magnitude and phase spectrum for x[n].

Step 2: Sketch magnitude spectrum of x[n].

Step 3: Sketch phase spectrum of x[n].

Resulting Sequence – y[n]





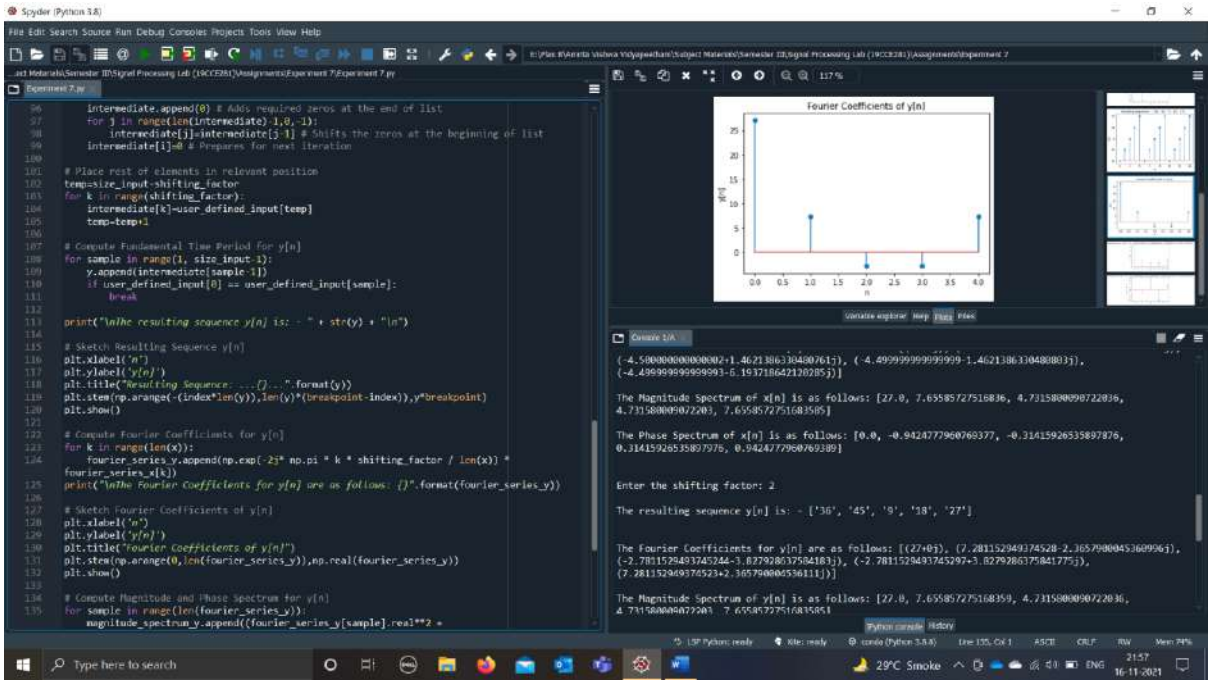
Step 1: Declare a NULL array "intermediate" and copy the input sequence to temporary sequence intermediate[n].

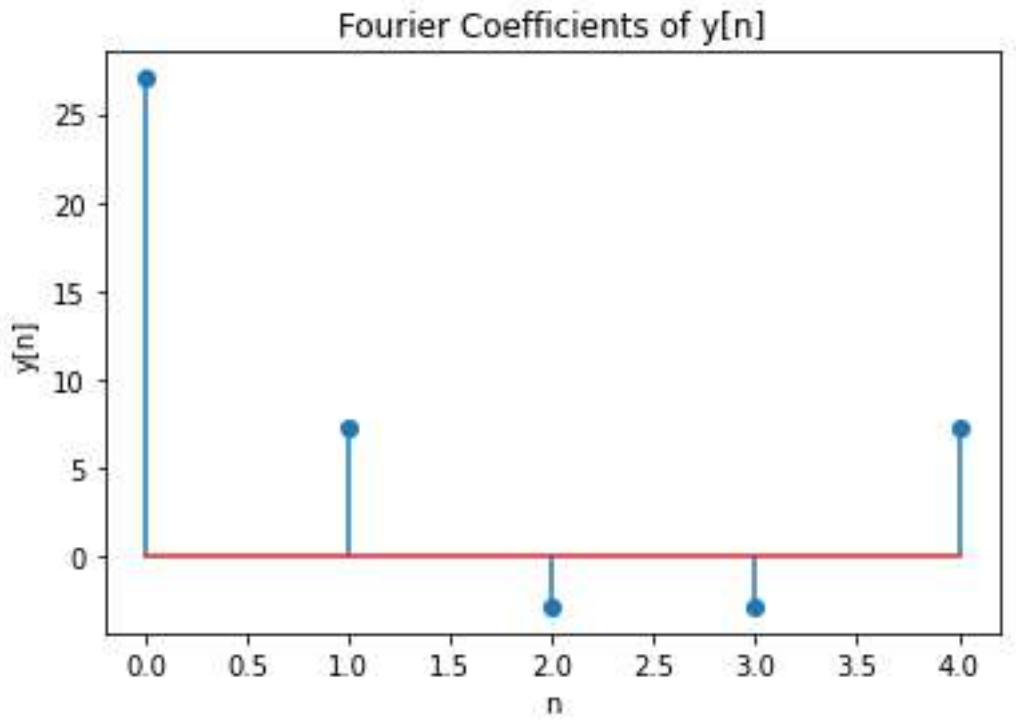
Step 2: Enter the shifting factor and shift temporary sequence by that many numbers of units.

Step 3: Place the rest of the elements in the relevant position and compute the fundamental time period for y[n].

Step 4: Print and sketch the resulting sequence, y[n].

Plot Fourier Coefficients of y[n]

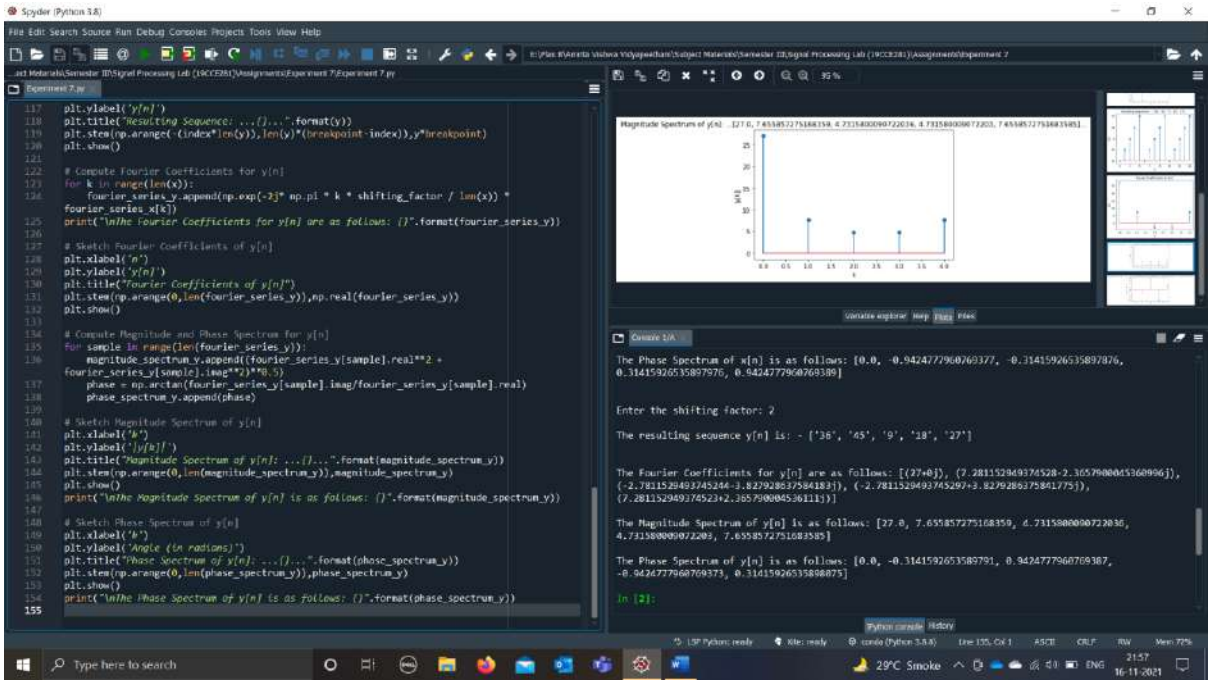




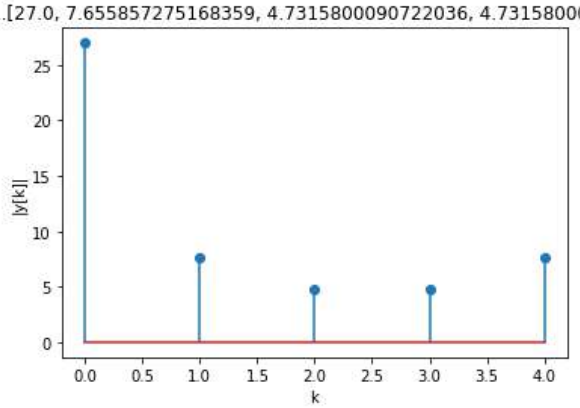
Step 1: Compute Fourier coefficients for y[n].

Step 2: Sketch Fourier coefficients of y[n].

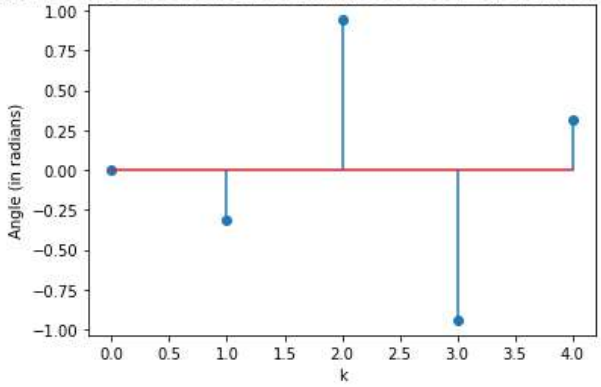
Plot Magnitude and Phase Spectrum of y[n]



Magnitude Spectrum of y[n]: ...[27.0, 7.655857275168359, 4.7315800090722036, 4.731580009072203, 7.6558572751683585]...



Phase Spectrum of y[n]: ...[0.0, -0.3141592653589791, 0.9424777960769387, -0.9424777960769373, 0.3141592653898075]...



Step 1: Compute magnitude and phase spectrum for y[n].

Step 2: Sketch magnitude spectrum of y[n].

Step 3: Sketch phase spectrum of y[n].

```
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 7/Experiment 7.py'      ='E:/Plan
B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab
(19CCE281)/Assignments/Experiment 7'

Enter the size of input x[n]: 15
Enter the elements of the input x[n] one-by-one as follows: -

Element 1: 9

Element 2: 18

Element 3: 27

Element 4: 36

Element 5: 45

Element 6: 9

Element 7: 18

Element 8: 27

Element 9: 36

Element 10: 45

Element 11: 9

Element 12: 18

Element 13: 27

Element 14: 36

Element 15: 45

The entered input x[n] is: - ['9', '18', '27', '36', '45', '9', '18', '27', '36', '45',
'9', '18', '27', '36', '45']


Enter the element position that indicates n = 0 index: 1

The Fourier Coefficients for x[n] are as follows: [(27+0j),
(-4.500000000000002+6.193718642120281j), (-4.500000000000002+1.4621386330480761j),
```

```
(-4.499999999999999-1.4621386330480803j), (-4.499999999999993-6.193718642120285j)]

The Magnitude Spectrum of x[n] is as follows: [27.0, 7.65585727516836, 4.7315800090722036,
4.731580009072203, 7.6558572751683585]

The Phase Spectrum of x[n] is as follows: [0.0, -0.9424777960769377, -0.31415926535897876,
0.31415926535897976, 0.9424777960769389]


Enter the shifting factor: 2

The resulting sequence y[n] is: - ['36', '45', '9', '18', '27']


The Fourier Coefficients for y[n] are as follows: [(27+0j),
(7.281152949374528-2.3657900045360996j), (-2.7811529493745244-3.827928637584183j),
(-2.7811529493745297+3.8279286375841775j), (7.281152949374523+2.365790004536111j)]

The Magnitude Spectrum of y[n] is as follows: [27.0, 7.655857275168359,
4.7315800090722036, 4.731580009072203, 7.6558572751683585]

The Phase Spectrum of y[n] is as follows: [0.0, -0.3141592653589791, 0.9424777960769387,
-0.9424777960769373, 0.31415926535898075]

In [2]:
```

Expt 8 FREQUENCY RESPONSE OF LTI SYSTEM

Determine the frequency response for the given LTI system characterized by the impulse response $h[n] = (u[n-3] - u[n])*0.5^n$ and sketch the same.

NAME - SANTOSH - [CB.EN.U4CLE20053]
DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CLE) A

LAB TITLE AND CODE : SIGNAL PROCESSING LAB 19CLE281
EXPERIMENT NUMBER : 8
DATE : 23/10/2021

FREQUENCY RESPONSE OF LTI SYSTEM

* AIM :

For a given LTI system characterized by an impulse response $h[n]$, determine the frequency response $h(\omega)$ and sketch the same.

* SOFTWARE REQUIRED :

Spyder IDE - Anaconda 3 2021.05 (Python 3.8.8 64-bit)
Publisher - Anaconda, Inc.

* THEORY :

A system is characterized by its impulse $h(n)$ whose DTFT transform is -

$$H(\omega) = \sum_{n=-\infty}^{\infty} h(n)\, e^{-j\omega n}$$

And the inverse DTFT is -

$$h(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(\omega)\, e^{j\omega n}\, d\omega$$

$H(\omega)$ is called the frequency response or frequency characteristic of the system. It is the frequency characterization of the system whereas the impulse response is the time characterization.

FREQUENCY RESPONSE -

Now we use the time convolution property (or convolution theorem) to map the output $y(n)$ in time domain to its transform $Y(\omega)$ in the frequency domain -

$$y(n) = x(n) * h(n) \leftrightarrow Y(\omega) = X(\omega) H(\omega)$$

Or
$$H(\omega) = \frac{Y(\omega)}{X(\omega)}$$

The frequency response $H(\omega)$ is usually a complex quantity, so we write -

$$H(\omega) = H_R(\omega) + jH_I(\omega) = |H(\omega)|\, e^{j\phi(\omega)}$$

where
$$|H(\omega)| = \sqrt{H_R^2(\omega) + H_I^2(\omega)}$$

and
$$\phi(\omega) = \tan^{-1}\frac{H_I(\omega)}{H_R(\omega)}$$

$|H(\omega)|$ is the magnitude response and $\phi(\omega)$ is the phase response.
If the impulse response $h(n)$ is real-valued, then -

$$|H(-\omega)| = |H(\omega)| \quad \text{and} \quad \phi(-\omega) = -\phi(\omega)$$

The frequency response of a system exists if the system is BIBO (Bounded Input Bounded output) stable, i.e.

$$\sum_{n=-\infty}^{\infty} |h(n)| < \infty$$

* 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 :

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  n = np.arange(0,10) #Get evenly spaced values within the
   interval [0,10)
```

```
5
6  a=[] # u[n] Null List Declaration
7  for sample in range (len(n)):
8      if n[sample] >= 0:
9          temp = 1
10     else:
11         temp = 0
12     a.append (temp)
13 plt.subplot (5,1,1)
14 plt.xlabel ('n')
15 plt.ylabel ('u[n]')
16 plt.title ('Unit step Discrete-Time Function for u[n]')
17 plt.stem (n,a)
18
19 b=[] # u[n-3] Null List Declaration
20 for sample in range (len(n)):
21     if n[sample] >= 3:
22         temp=1
23     else
24         temp=0
25     b.append (temp)
26 plt.subplot (5,1,3)
27 plt.xlabel ('n')
28 plt.ylabel ('u[n-3]')
29 plt.title ('Unit step Discrete-Time Function for u[n-3]')
30 plt.stem (n,b)
31
32 result = [] # u[n-3] - u[n] Null List Declaration
33 for sample in range (len(n))
34     result.append (b[sample] - a[sample])
35 plt.subplot (5,1,5)
36 plt.xlabel ('n')
37 plt.ylabel ('u[n-3] - u[n]')
```

```
plt.title('Unit step Discrete-Time Function for u[n-3] - u[n]')
plt.stem(n, result)
plt.show()

temp = []  # (u[n-3] - u[n]) * 0.5^n Null List Declaration
for sample in range(len(n)):
    temp.append((b[sample] - a[sample]) * 0.5 ** sample)
plt.xlabel('n')
plt.ylabel('(u[n-3] - u[n]) * 0.5^n')
plt.title('Impulse Response h[n]')
plt.stem(n, temp)
plt.show()

# Compute Frequency Response
frequency_response = []
def summation(f):
    sum = 0
    for k in range(len(temp))
        exponential = np.exp(-2j * np.pi * k * f)
        sum = sum + float(temp[k]) * exponential
    return(sum)

for w in range(len(temp)):
    frequency_response.append(summation(w))

# sketch Frequency Response
plt.xlabel('w')
plt.ylabel('h(w)')
plt.title("Frequency Response")
plt.plot(np.arange(0, len(frequency_response)),
 np.real(frequency_response))
plt.show()
print("\n The Frequency Response of h[n] is as follows : {}".
 format(frequency_response))
```

```
70
71  # Compute Magnitude and Phase Spectrum
72  magnitude_spectrum = []
73  phase_spectrum = []
74
75  for sample in range (len (frequency_response)):
76      magnitude_spectrum.append ((frequency_response[sample].
    real ** 2 + frequency_response[sample].imag ** 2) ** 0.5)
77      phase = np.arctan (frequency_response[sample].imag/
    frequency_response[sample].real)
78      phase_spectrum.append (phase)
79
80  # Sketch Magnitude Spectrum
81  plt.xlabel ('w')
82  plt.ylabel ('|H(w)|')
83  plt.title ("Magnitude spectrum : ... {} ... ".format
    (magnitude_spectrum))
84  plt.plot (np.arange (0, len (magnitude_spectrum)),
    magnitude_spectrum)
85  plt.show ()
86  print ["\nThe Magnitude Spectrum of h[n] is as follows : {}".
    format (magnitude_spectrum))
87
88  # Sketch Phase Spectrum
89  plt.xlabel ('w')
90  plt.ylabel ('Angle (in radians)')
91  plt.title ("Phase Spectrum $(w)")
92  plt.plot (np.arange (0, len (phase_spectrum)), phase_spectrum)
93  plt.show ()
94  print ["\n The Phase Spectrum of h[n] is as follows : {}".
    format (phase_spectrum)]
95
```

⑥

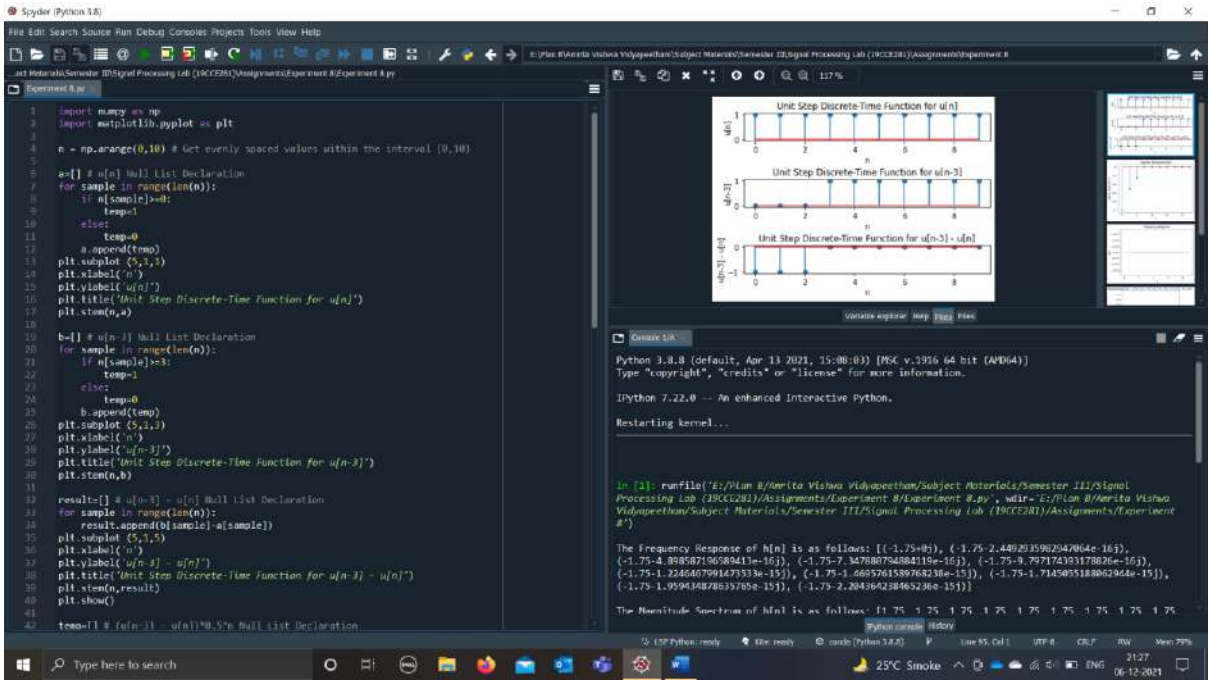
* **INFERENCE:**

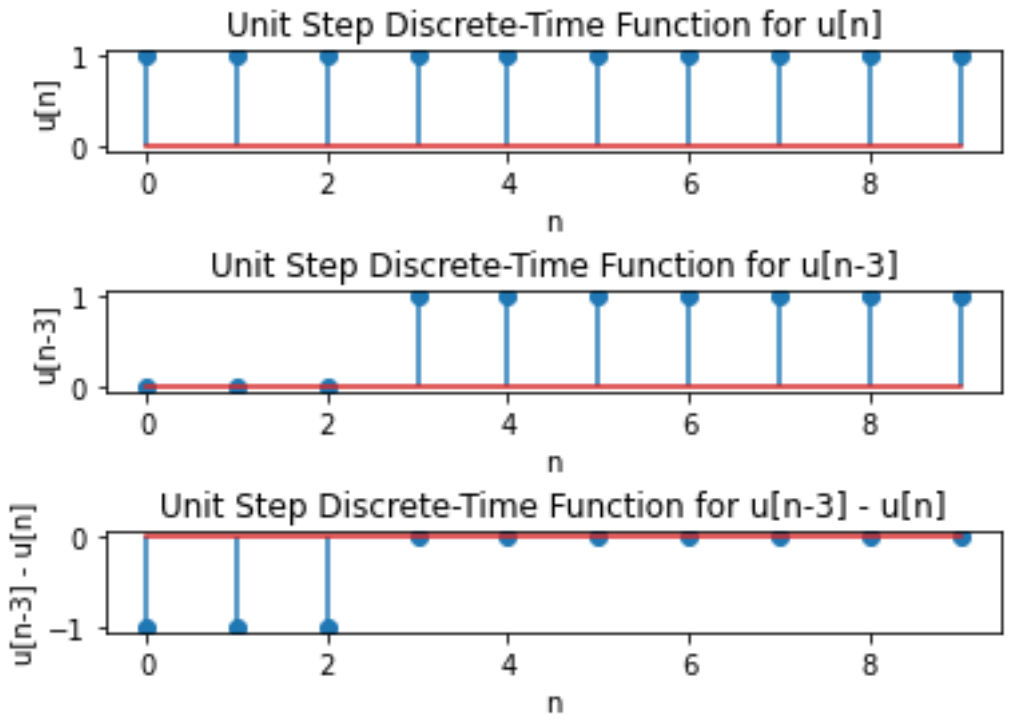
For the given LTI system characterized by the impulse response h[n], compute frequency response h(w) and plot the same along with magnitude and phase spectrums.

RESULTS

VERIFIED.

Unit Step Discrete-Time Functions





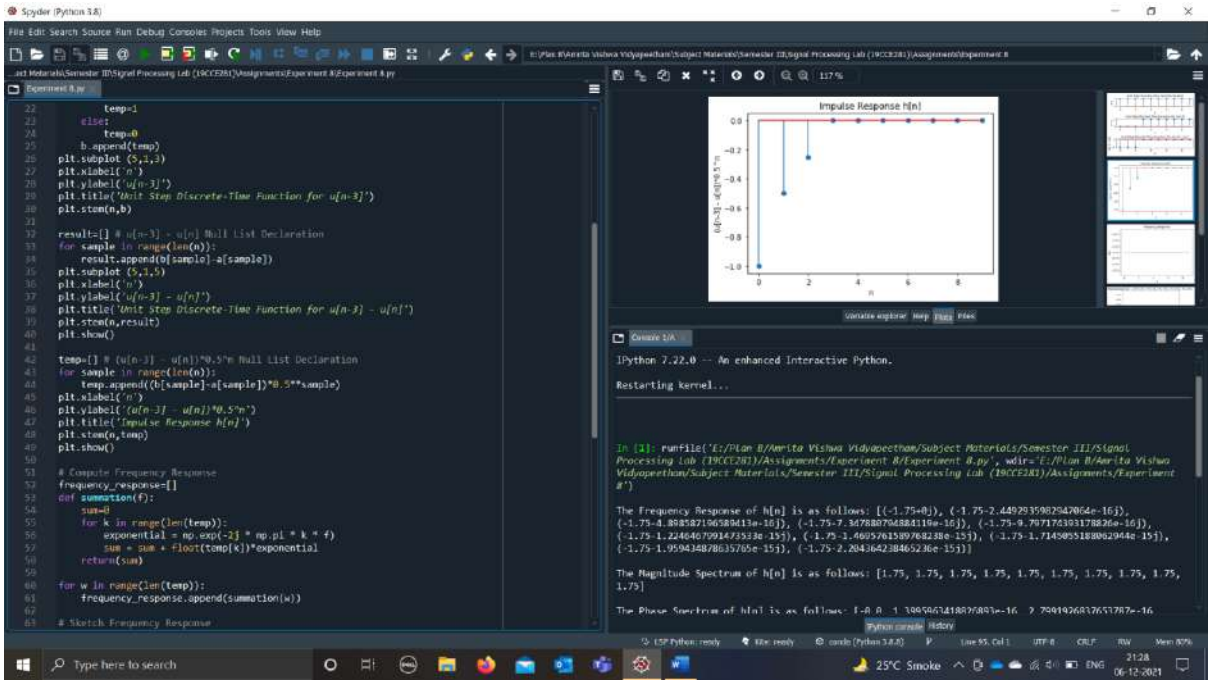
Step 1: Import library source files, numpy and matplotlib.pyplot.

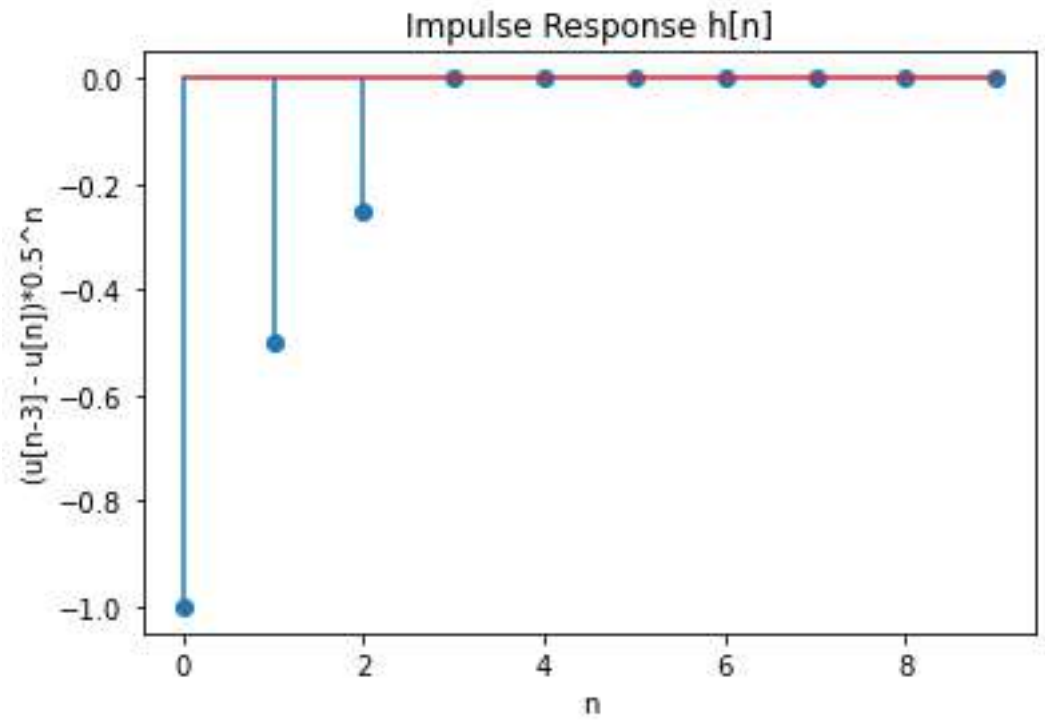
Step 2: Get evenly spaced values within the interval [0,10).

Step 3: Compute u[n] and u[n-3] followed by (u[n-3] - u[n]) and finally, (u[n-3] - u[n])*0.5^n

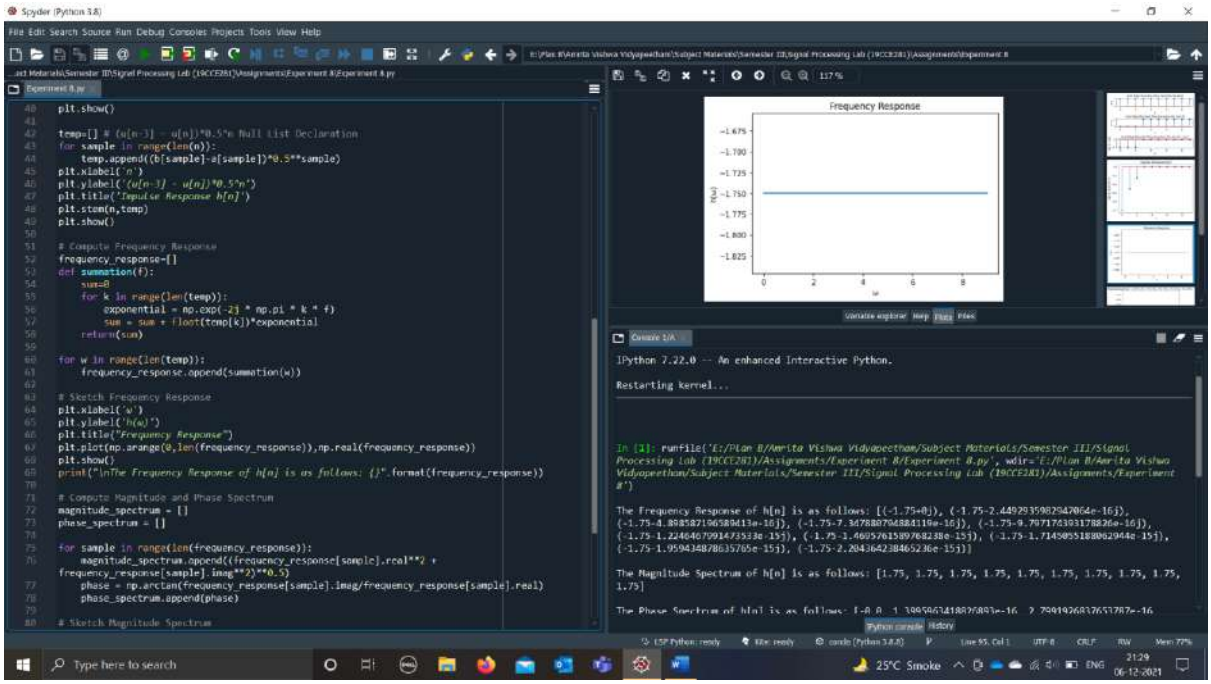
Step 4: Sketch the plot obtained for each function in Step 3.

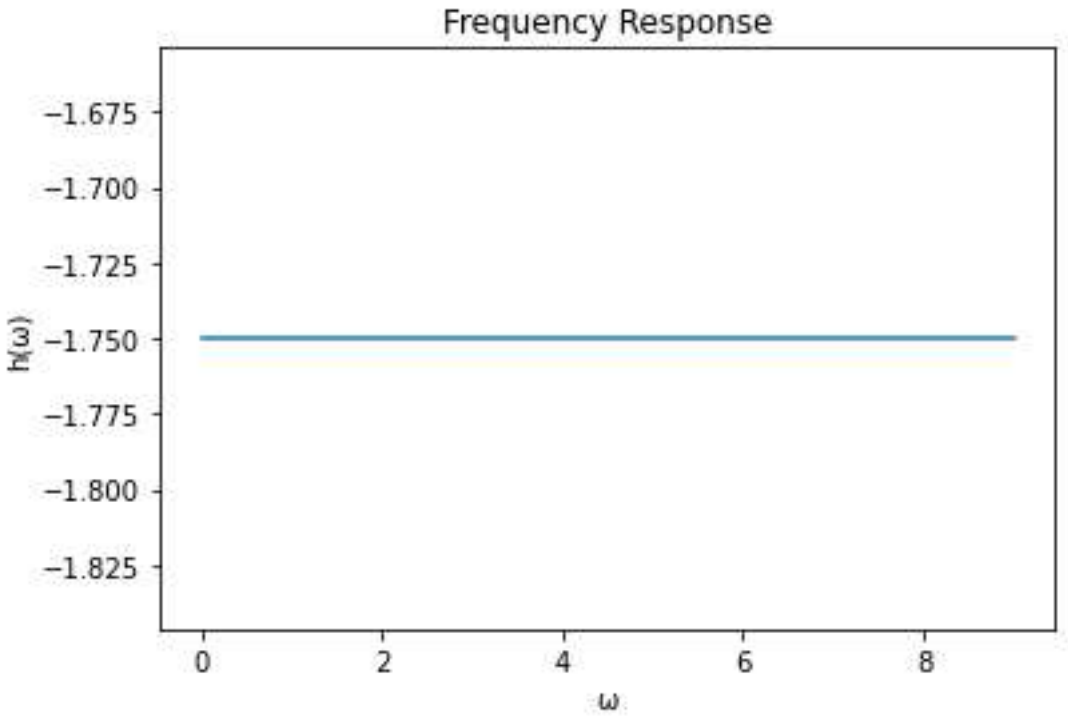
Given Impulse Response h[n]





Frequency Response – h(ω)





Step 1: Compute frequency response for h[n].

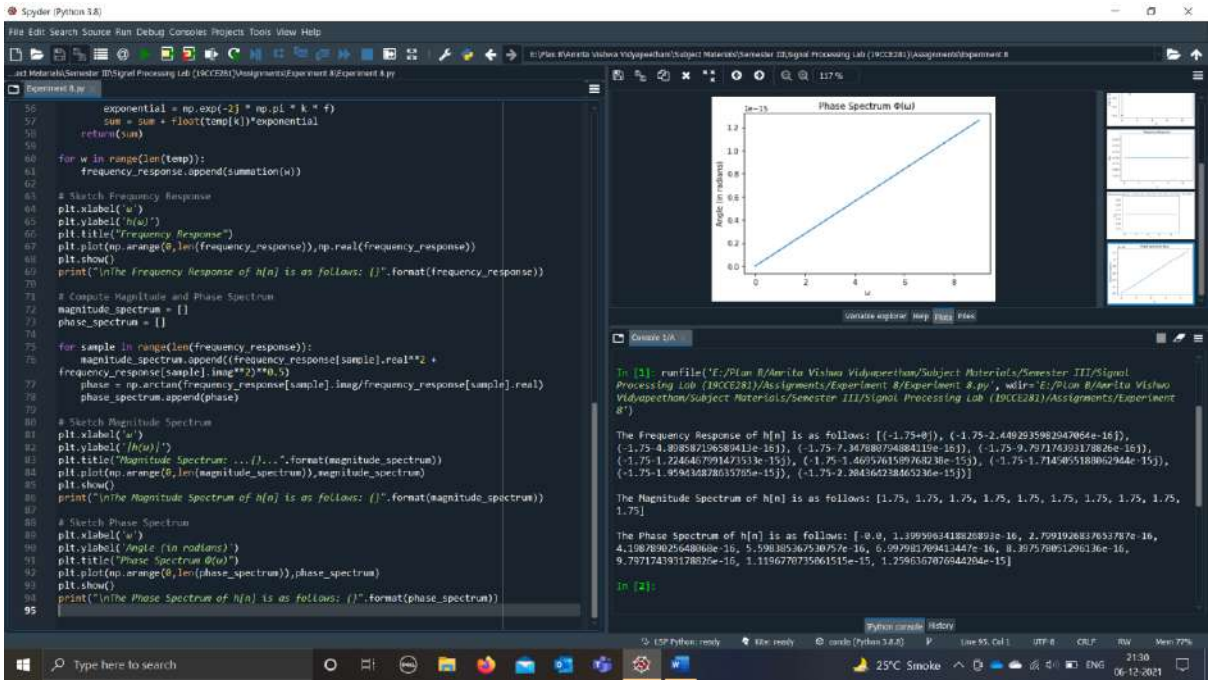
Step 2: Print and sketch the frequency response, h(ω).

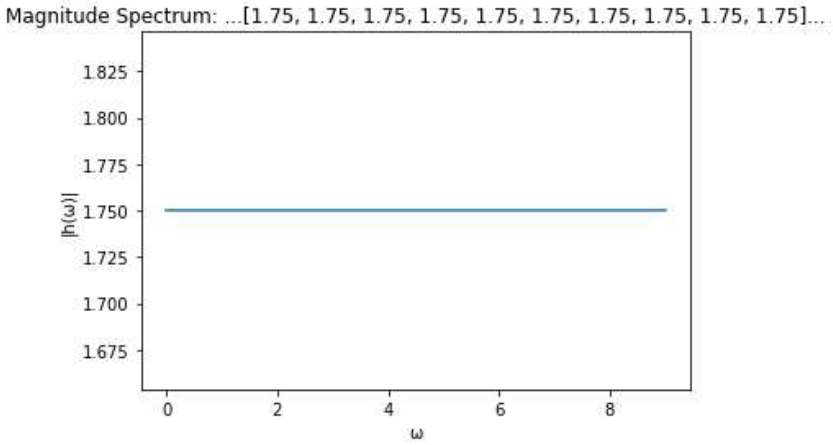
Step 3: Compute magnitude and phase spectrum for h[n].

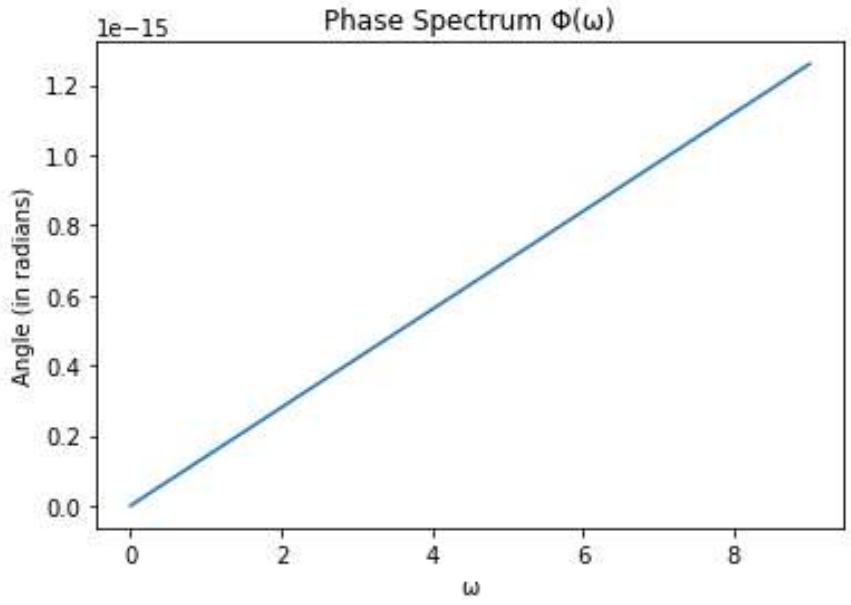
Step 4: Print and sketch magnitude spectrum, |h(ω)|.

Step 5: Print and sketch phase spectrum, Φ(ω).

Magnitude Spectrum – |h(ω)| & Phase Spectrum – Φ(ω)







```
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 8/Experiment 8.py'      ='E:/Plan
B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab
(19CCE281)/Assignments/Experiment 8'

The Frequency Response of h[n] is as follows: [(-1.75+0j),
(-1.75-2.4492935982947064e-16j), (-1.75-4.898587196589413e-16j),
(-1.75-7.347880794884119e-16j), (-1.75-9.797174393178826e-16j),
(-1.75-1.2246467991473533e-15j), (-1.75-1.4695761589768238e-15j),
(-1.75-1.7145055188062944e-15j), (-1.75-1.959434878635765e-15j),
(-1.75-2.204364238465236e-15j)]

The Magnitude Spectrum of h[n] is as follows: [1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75,
1.75, 1.75, 1.75]

The Phase Spectrum of h[n] is as follows: [-0.0, 1.3995963418826893e-16,
2.7991926837653787e-16, 4.198789025648068e-16, 5.598385367530757e-16,
6.997981709413447e-16, 8.397578051296136e-16, 9.797174393178826e-16,
1.1196770735061515e-15, 1.2596367076944204e-15]

In [2]:
```

Expt 9 Discrete Fourier Transform

Compute DFT of the sequence x[n] = [0 1 2 3] using Direct method and Linear Transformation method .

Sketch the input, magnitude and phase spectrum of DFT coefficients obtained using direct method and Linear Transformation method

NAME - SANTOSH - [CB.EN.U4CCE20053]
DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE)

LAB TITLE AND CODE: SIGNAL PROCESSING LAB 19CCE251
EXPERIMENT NUMBER: 9
DATE: 14/12/2021

DISCRETE FOURIER TRANSFORM

* **AIM:**
For a given sequence $x[n]$, compute DFT coefficients using the direct and linear transformation methods. Sketch the magnitude and phase spectrum obtained from the same.

* **SOFTWARE REQUIRED:**
Spyder IDE - Anaconda 3 2021.11 (Python 3.9.7 64-bit)
Publisher - Anaconda, Inc.

* **THEORY:** (DIRECT METHOD)
The discrete Fourier transform (DFT) is a method for converting a sequence of N complex numbers $x_0, x_1, \ldots, x_{N-1}$ to a new sequence of N complex numbers,

$$x_k = \sum_{n=0}^{N-1} x_n e^{-2\pi j kn/N}$$

for $0 \le k \le N-1$.

The x_i are thought of as the values of a function, or signal, at equally spaced times $t = 0, 1, \ldots, N-1$. The output x_k is a complex number which encodes the amplitude and phase of a sinusoidal wave with frequency $\frac{k}{N}$ cycles per unit time.

[This comes from Euler's formula -
$e^{2\pi j kn/N} = \cos(2\pi kn/N) + j \sin(2\pi kn/N)$]

PTO

(b)

The effect of computing the X_k is to find the coefficients of an approximation of the signal by a linear combination of such waves. Since each wave has an integer number of cycles per N time units, the approximation will be periodic with period N. This approximation is given by the inverse Fourier transform -

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \, e^{2\pi j kn/N}$$

→ TWIDDLE FACTOR -

It is denoted as W_N and defined as $W_N = e^{-j2\pi/N}$. Its magnitude is always maintained at unity. Phase of $W_N = -2\pi/N$. It is a vector on unit circle and is used for computational convenience. mathematically, it can be shown as -

$$W_N^n = W_N^{n \pm N} = W_N^{n \pm 2N} = \dots$$

It is a function of n and period N.

consider $N = 8$, $n = 0, 1, 2, 3, \dots, 14, 15, 16, \dots$

$$\Rightarrow W_8^0 = W_8^8 = W_8^{16} = \dots = \dots = W_8^{32} = \dots = 1 = 1 \angle 0$$

$$W_8^1 = W_8^9 = W_8^{17} = \dots = \dots = W_8^{33} = \dots = \frac{1}{\sqrt{2}} - j\frac{1}{\sqrt{2}} = 1 \angle -\frac{\pi}{4}$$

→ LINEAR TRANSFORMATION -

We know that -

$$DFT(k) = DFT[x(n)] = X\left(\frac{2\pi}{N} k\right) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{-nk};$$

$$k = 0, 1, \dots, N-1$$

$$x(n) = IDFT[X(k)]$$

$$= \frac{1}{N} \sum_{n=0}^{N-1} X(k) \cdot W_N^{-nk}; \; n = 0, 1, \dots, N-1$$

Computation of DFT can be performed with N^2 complex multiplication and $N(N-1)$ complex addition

$$x_N = \begin{bmatrix} x(0) \\ x(1) \\ \cdot \\ \cdot \\ x(N-1) \end{bmatrix}$$; N-point vector of signal x_N

$$X_N = \begin{bmatrix} X(0) \\ x(1) \\ \cdot \\ \cdot \\ x(N-1) \end{bmatrix}$$; N-point vector of signal x_N

$$W_N = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & \cdots & W_N^{N-1} \\ \cdot & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix}$$; matrix of linear transformation

N-point DFT in matrix form is given by $X_N = W_N x_N$

$$\Rightarrow x_N = W_N^{-1} X_N$$

IDFT in matrix form is given by -

$$x_N = \frac{1}{N} W_N^* X_N$$

Comparing these expressions of x_N,

$$W_N^{-1} = \frac{1}{N} W_N^* \quad \text{and} \quad W_N \times W_N^* = N[I]_{N \times N}$$

Therefore, W_N is a linear transformation matrix, an orthogonal unitary matrix. From periodic property of W_N and from its symmetric property, it can be concluded that,

$$W_N^{k+N/2} = -W_N^k$$

* **GRAPH PLOTTING ALGORITHM:**

The following steps are followed-

1. Define the x-axis and corresponding y-axis as lists.
2. Plot them on canvas using .plot() function.
3. Give a name to x-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.

* **THEORETICAL CALCULATION: (DIRECT METHOD)**

Given sequence, $x[n] = [0\ 1\ 2\ 3]$

By keen observation, we find that fundamental period, $N = 4$

Angular frequency, $\Omega_0 = \frac{2\pi}{4}$

$$x[k] = \sum_{n=0}^{N-1} x[n]\, e^{-jk\frac{2\pi}{4}n} = \sum_{n=0}^{3} x[n]\, e^{-jk\frac{2\pi}{4}n}$$

$$x[k] = x[0]\, e^{-jk\frac{2\pi}{4}(0)} + x[1]\, e^{-jk\frac{2\pi}{4}(1)} + x[2]\, e^{-jk\frac{2\pi}{4}(2)} + x[3]\, e^{-jk\frac{2\pi}{4}(3)}$$

$$x[k] = 0(1) + 1\left(e^{-jk\frac{2\pi}{4}}\right) + 2\left(e^{-jk\frac{4\pi}{4}}\right) + 3\left(e^{-jk\frac{6\pi}{4}}\right)$$

Substituting the values of k, we get-

$x[0] = 6 + 0j$, for $k = 0$

$x[1] = -2 + 2j$, for $k = 1$

$x[2] = -2 + 0j$, for $k = 2$

$x[3] = -2 - 2j$, for $k = 2$

Upon computing the magnitude spectrum, we find that-

$|x[0]| = \sqrt{6^2 + 0^2} = 6.00$

$|x[1]| = \sqrt{(-2)^2 + (2)^2} = 2.83$

$|x[2]| = \sqrt{(-2)^2 + (0)^2} = 2$

$|x[3]| = \sqrt{(-2)^2 + (-2)^2} = 2.83$

Upon computing the phase spectrum, we find that -

$\angle y[n] = \tan^{-1}\left(\frac{\text{Im}}{\text{Re}}\right) = \_\_^{\circ} = \_\_ * \frac{\pi}{180^{\circ}}$ radians

$\angle y[0] = 0^{\circ} = 0$ radians
$\angle y[1] = 135^{\circ} = -0.79$ radians
$\angle y[2] = 180^{\circ} = 3.68$ radians
$\angle y[3] = -135^{\circ} = 0.79$ radians

→ LINEAR TRANSFORMATION -

The first step is to determine the matrix W_4. By exploiting the periodicity property of W_N and the symmetric property -

$$W_N^{k+N} = -W_N^{k}$$

the matrix W_4 may be expressed as -

$$W_4 = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^1 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

Then,

$$x_4 = W_4 a_4 = \begin{bmatrix} 6 \\ -2+2j \\ -2 \\ -2-2j \end{bmatrix}$$

Upon computing the magnitude spectrum, we find that -

$|x[0]| = \sqrt{6^2+0^2} = 6.00$
$|x[1]| = \sqrt{(-2)^2+(2)^2} = 2.83$
$|x[2]| = \sqrt{(-2)^2+(0)^2} = 2$
$|x[3]| = \sqrt{(-2)^2+(-2)^2} = 2.83$

Upon computing the phase spectrum, we find that -

$$\angle x[n] = \tan^{-1}\left(\frac{H}{R}\right) = -^{\circ}\mathbb{R} = -\times\frac{\pi}{180} \text{ radians}$$

Therefore,

$\angle x[0] = 0^{\circ} = 0$ radians
$\angle x[1] = -135^{\circ} = -0.79$ radians
$\angle x[2] = 180^{\circ} = 3.68$ radians
$\angle x[3] = 135^{\circ} = 0.79$ radians

* PROGRAM WITH COMMENTS:

```
# Function to calculate X(k) for Direct method
def summation
    sum = 0
    for n in range (size_x):
        exponential = np.exp(-2j * np.pi * k * n / size_x)
        sum = sum + float(x[n]) * exponential
    return (sum)

# import library source files
import numpy as np
import matplotlib.pyplot as plt

#Given Input Sequence x[n]
size_x = int(input("Enter the size of input x[n]: "))
x = [0] * (size_x)
print("Enter the elements of the input x[n] one-by-one
as follows: -")
for sample in range(0, size_x, 1):
    x[sample] = input("element " + str(sample+1) + ": ")

#sketch Input Sequence
plt.xlabel('n')
plt.ylabel('x[n]')
```

```
plt.title("Input Sequence x[]".format(x))
plt.stem(np.arange(0, size_x), x)
plt.grid(True) # Configure the grid lines
plt.show()
print("\n The entered input x[n] is: -" + str(x) + "\n")

# Compute Discrete Fourier Transform Coefficients Using Direct Method
fourier_transform_direct = []
for k in range(size_x):
    fourier_transform_direct.append(summation(k))
print("\n The Discrete Fourier Transform (DFT) coefficients of the sequence x[n] obtained using direct method are as follows: {} ".format(fourier_transform_direct))

# Compute Magnitude and Phase Spectrum of DFT Coefficients
magnitude_spectrum_direct = []
phase_spectrum_direct = []

for sample in range(len(fourier_transform_direct)):
    magnitude_spectrum_direct.append((fourier_transform_direct[sample].real**2 + fourier_transform_direct[sample].imag**2)**0.5)
    phase_direct = np.arctan(fourier_transform_direct[sample].imag / fourier_transform_direct[sample].real)
    phase_spectrum_direct.append(phase_direct)

# Sketch Magnitude Spectrum Using Direct Method
plt.xlabel('k')
plt.ylabel('|x[k]|')
plt.title("Magnitude Spectrum obtained Using Direct Method")
```

PTO

```
plt.stem(np.arange(0, len(magnitude_spectrum_direct)), magnitude_spectrum_direct)
plt.grid(True) #Configure the grid lines
plt.show()
print("\n The magnitude spectrum of DFT coefficients obtained using direct method are as follows : {}".format(magnitude_spectrum_direct))

# sketch Phase spectrum using Direct method
plt.xlabel('x')
plt.ylabel('Angle (in radians)')
plt.title("Phase Spectrum obtained using Direct Method")
plt.stem(np.arange(0, len(phase_spectrum_direct)), phase_spectrum_direct)
plt.show()
plt.grid(True) # Configure the grid lines
print("\n The phase spectrum of DFT coefficients obtained using direct method are as follows : {}".format(phase_spectrum_direct))

# Compute DFT coefficients using Linear Transformation Method

# Compute W(N) 1D Array
rl = cl = size_x
wn = []
for i in range(rl):
    for j in range(cl):
        wn.append(np.exp(-2j * np.pi * i * j / size_x))

# numpy.reshape() is used to give a new shape to an array without changing its data.
```

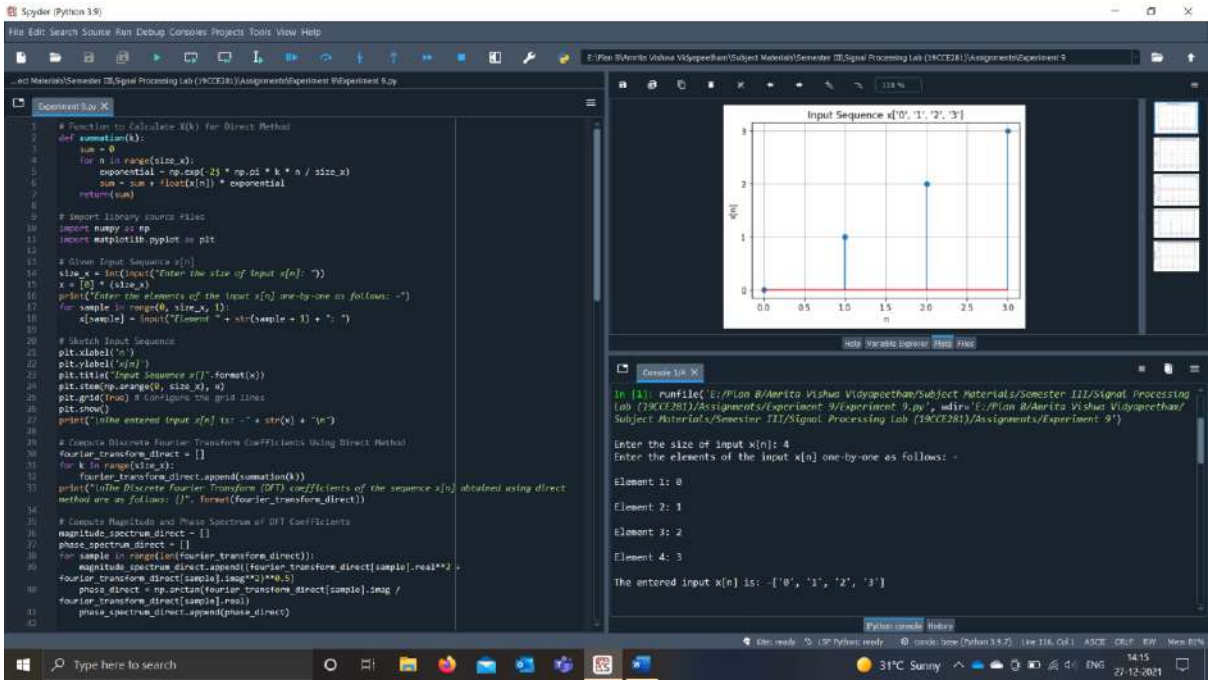
```
wn_multidim = np.reshape(wn, (r1, c1)) # An N*N W(N) matrix
print("\n The matrix W(n) is as follows: \n{}".format
(wn_multidim))
r2 = size_x ; c2 = 1
x_multidim = np.reshape(x,(r2, c2)) # An N*1 x(N) matrix
print("\n The matrix x(N) is as follows: \n{}".format
(x_multidim))

#Compute X(N) = W(N) * x(N), an N*1 matrix
fourier_transform_multidim = [[0] * c2] * r1
#Null Multidimensional Array
fourier_transform_1_t = [] # Convert multidimensional
Array to 1D
for i in range(r1):
    for j in range(c2):
        fourier_transform_multidim[i][j] = 0
        for k in range(c1):
            fourier_transform_multidim[i][j] +=
wn_multidim[i][k] * float(x_multidim[k][j])
        temp = fourier_transform_multidim[i][j]
        fourier_transform_1_t.append(fourier_transform
_multidim[i][j])

print("\n The Discrete Fourier Transform (DFT) coefficients
of the sequence x[n] obtained using linear transformation
are as follows: {}".format(fourier_transform_1_t))

#Compute magnitude and Phase spectrum of DFT Coefficients
magnitude_spectrum_1_t = []
phase_spectrum_1_t = []
for sample in range(len(fourier_transform_1_t)):
    magnitude_spectrum_1_t.append((fourier_transform_1_t
[sample].real**2 + fourier_transform_1_t[sample].imag**2)**0.5)
```

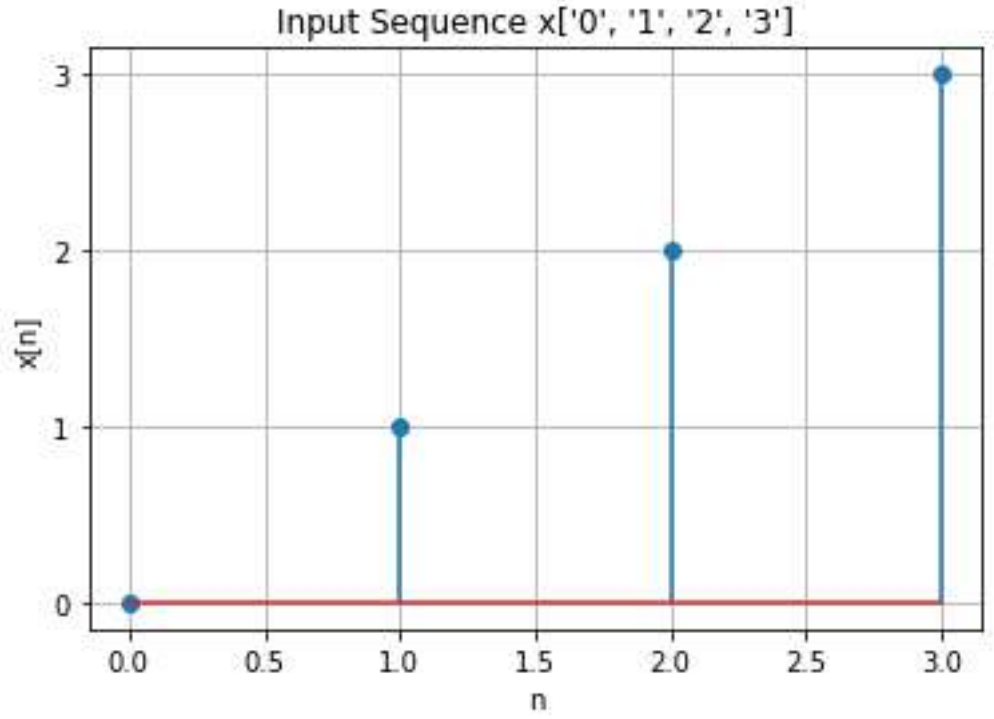
```
96      phase_l_t = np.arctan(fourier_transform_l_t[sample].imag / fourier_transform_l_t[sample].real)
97      phase_spectrum_l_t.append(phase_l_t)
98
99  # sketch magnitude spectrum using Linear Transformation method
100 plt.xlabel('k')
101 plt.ylabel('|x[k]|')
102 plt.title("Magnitude spectrum Obtained Using Linear Transformation Method")
103 plt.grid(True)
104 plt.stem(np.arange(0, len(magnitude_spectrum_l_t)), magnitude_spectrum_l_t)
105 plt.show()
106 print("\n The magnitude spectrum of DFT coefficients obtained using linear transformation are as follows: {}".format(magnitude_spectrum_l_t))
107
108 # sketch Phase spectrum using Linear Transformation Method
109 plt.xlabel('k')
110 plt.ylabel('Angle in radians')
111 plt.title("Phase spectrum Obtained Using Linear Transformation method")
112 plt.stem(np.arange(0, len(phase_spectrum_l_t)), phase_spectrum_l_t)
113 plt.grid(True) # Configure the grid lines
114 plt.show()
115 print("\n The phase spectrum of DFT coefficients obtained using linear transformation method are as follows : {}".format(phase_spectrum_l_t))
116
```

RESULTS

VERIFIED

Given Input Sequence x[n]





Step 1: Import library source files, NumPy and matplotlib.pyplot.

Step 2: Enter the size of input x[n] and declare an array with the number of elements equal to the value of size.

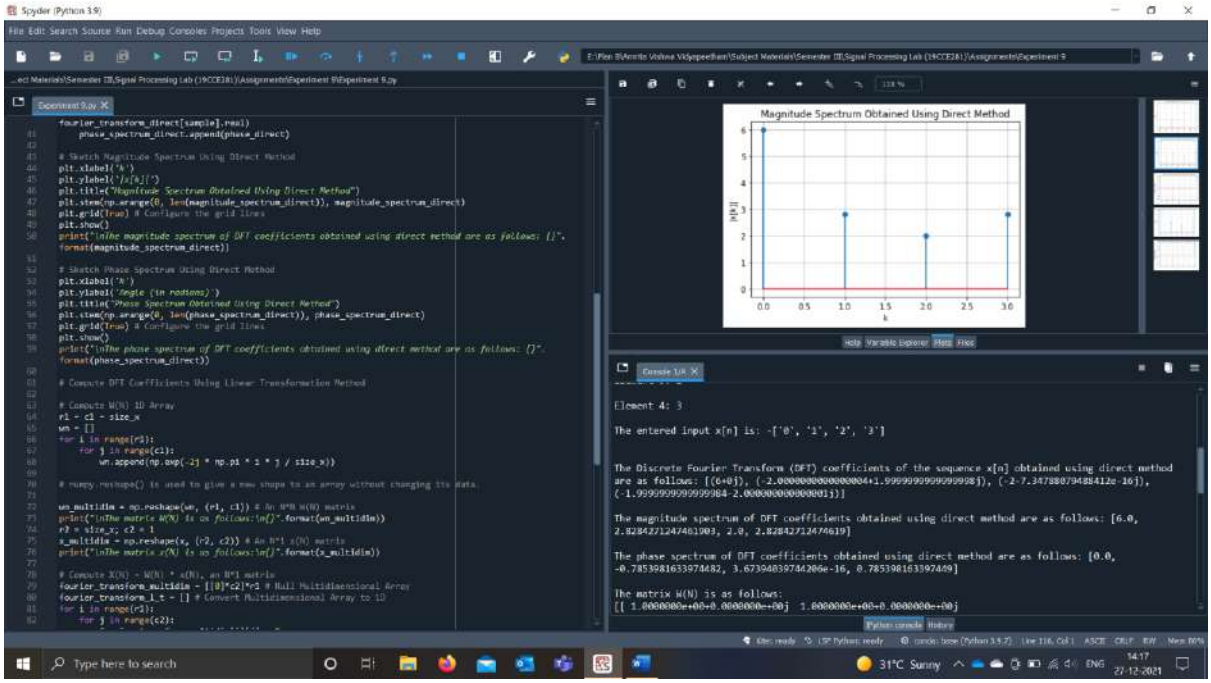
Step 3: Enter the elements of the input x[n] and show the labelled plot.

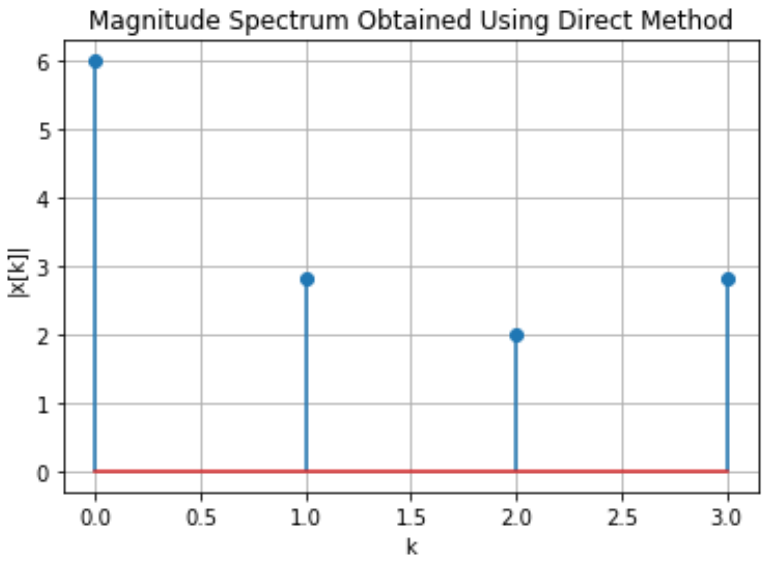
Step 4: Compute Discrete Fourier Transform (DFT) coefficients using the direct method and find its magnitude and phase spectrum.

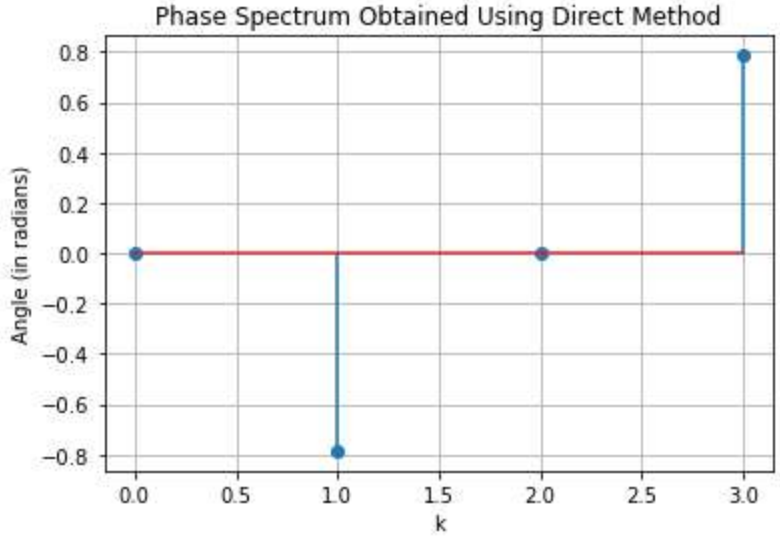
Step 5: Compute W(N) 1D Array followed by X(N) = W(N) * x(N), an N*1 matrix.

Step 6: Repeat step 4 for the linear transformation method.

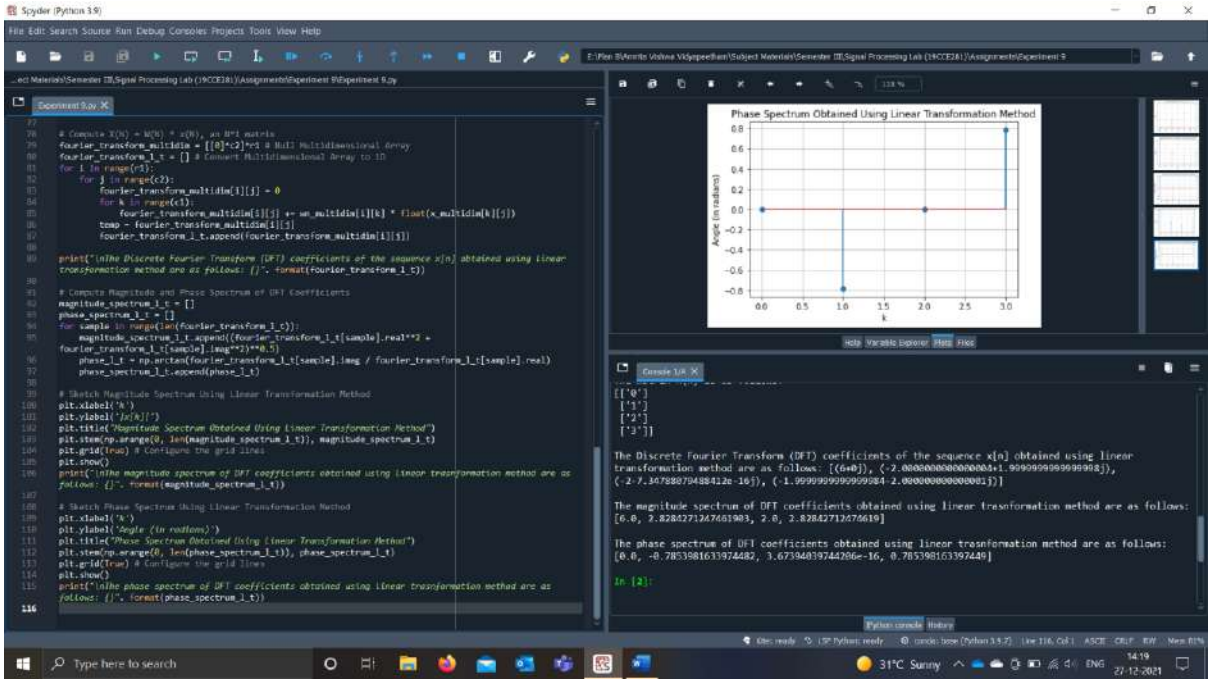
Compute DFT Coefficients Using Direct Method

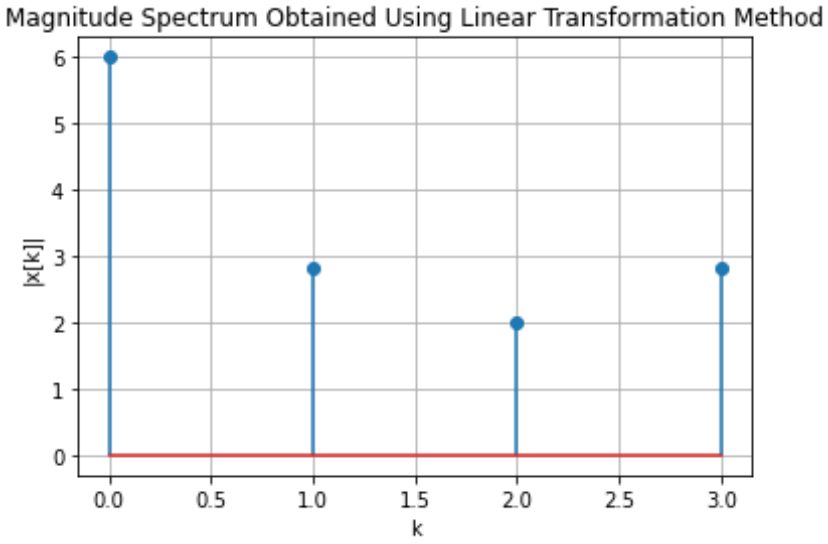


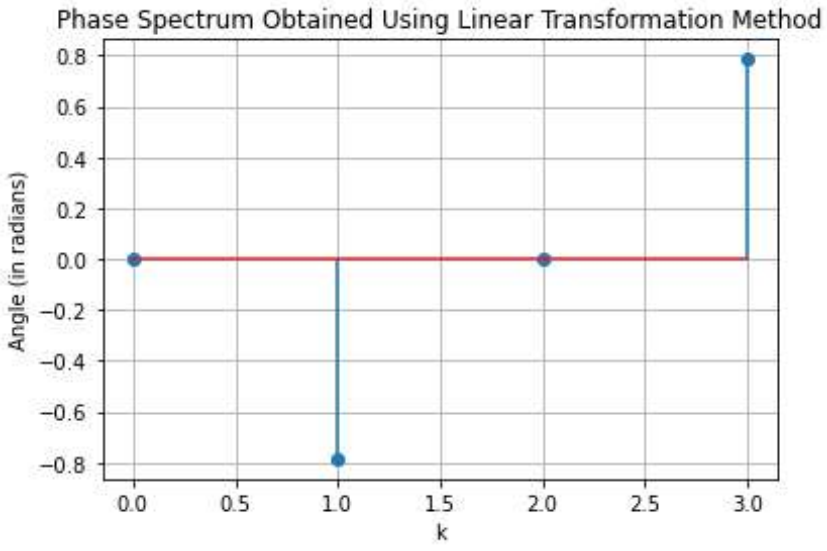




Compute DFT Coefficients Using Linear Transformation Method







```
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 9/Experiment 9.py'      ='E:/Plan
B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab
(19CCE281)/Assignments/Experiment 9'

Enter the size of input x[n]: 4
Enter the elements of the input x[n] one-by-one as follows: -

Element 1: 0

Element 2: 1

Element 3: 2

Element 4: 3

The entered input x[n] is: -['0', '1', '2', '3']


The Discrete Fourier Transform (DFT) coefficients of the sequence x[n] obtained using
direct method are as follows: [(6+0j), (-2.0000000000000004+1.9999999999999998j),
(-2-7.34788079488412e-16j), (-1.9999999999999984-2.000000000000001j)]

The magnitude spectrum of DFT coefficients obtained using direct method are as follows:
[6.0, 2.8284271247461903, 2.0, 2.82842712474619]

The phase spectrum of DFT coefficients obtained using direct method are as follows: [0.0,
-0.7853981633974482, 3.67394039744206e-16, 0.785398163397449]

The matrix W(N) is as follows:
[[ 1.0000000e+00+0.0000000e+00j  1.0000000e+00+0.0000000e+00j
   1.0000000e+00+0.0000000e+00j  1.0000000e+00+0.0000000e+00j]
 [ 1.0000000e+00+0.0000000e+00j  6.1232340e-17-1.0000000e+00j
  -1.0000000e+00-1.2246468e-16j -1.8369702e-16+1.0000000e+00j]
 [ 1.0000000e+00+0.0000000e+00j -1.0000000e+00-1.2246468e-16j
   1.0000000e+00+2.4492936e-16j -1.0000000e+00-3.6739404e-16j]
 [ 1.0000000e+00+0.0000000e+00j -1.8369702e-16+1.0000000e+00j
  -1.0000000e+00-3.6739404e-16j  5.5109106e-16-1.0000000e+00j]]

The matrix x(N) is as follows:
[['0']
 ['1']
 ['2']
 ['3']]

The Discrete Fourier Transform (DFT) coefficients of the sequence x[n] obtained using
```

linear transformation method are as follows: [(6+0j),
(-2.0000000000000004+1.9999999999999998j), (-2-7.34788079488412e-16j),
(-1.9999999999999984-2.000000000000001j)]

The magnitude spectrum of DFT coefficients obtained using linear trasnformation method are as follows: [6.0, 2.8284271247461903, 2.0, 2.82842712474619]

The phase spectrum of DFT coefficients obtained using linear trasnformation method are as follows: [0.0, -0.7853981633974482, 3.67394039744206e-16, 0.785398163397449]

In [2]:

Expt 10 Design of FIR Filter

Design FIR low pass filter whose desired frequency response is
$H(e^{j\omega}) = 1;\ \ -2\pi/5 \leq \omega \leq 2\pi/5$ and 0 otherwise
using Rectangular and Hanning window techniques.

(i) Determine the impulse response {h[n]} for N = 9 and plot the same
(ii) Determine the frequency response $\{H(e^{j\omega})\}$ and plot the same.

****

Solution

Desired filter coefficient $\{h_d[n]\}$ is obtained by taking Inverse Discrete Time Fourier Transform (IDTFT) on $H(e^{j\omega})$ and yields $h_d[n] = \sin((2\pi n/5) / \pi n)$

Expected filter coefficients {h]n]} on multiplying with rectangular window function {w[n]} yields the following.

h[0] = 2/5 = 0.4, h[1] = 0.3027, h[2] = 0.0935 , h[3] = - 0.06236 , h[4] = xx, h[5] = - 0.06236, h[6]= 0.0935, h[7] = 0.3027, h[8] = 0.4

Determine the frequency response $H(e^{j\omega})$ by direct substitution of the expression {even symmetry and length is odd integer}

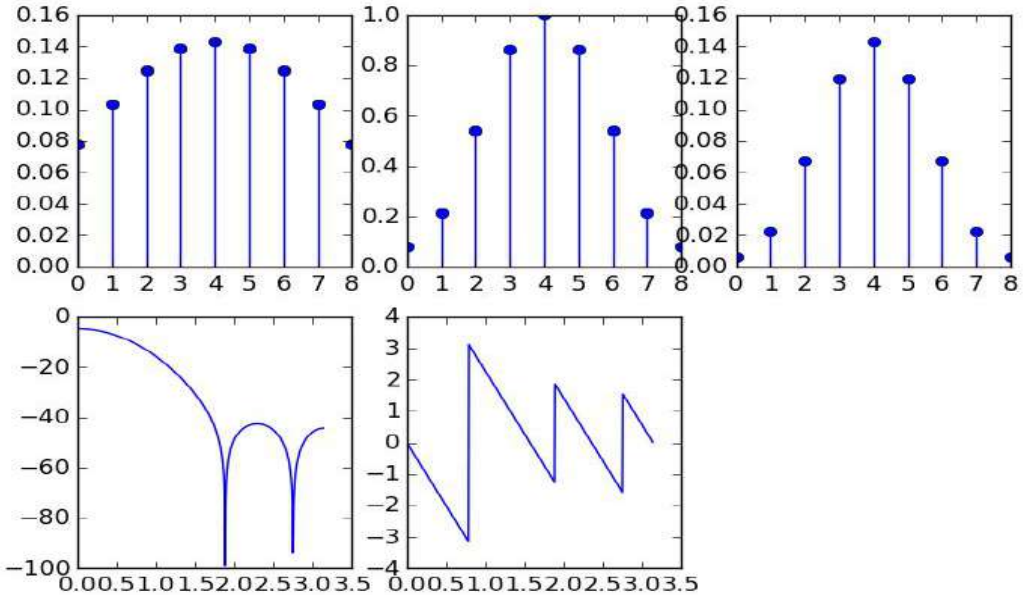
****

In python

Note:

```
from scipy.integrate import quad        ## library that support integration function
from scipy import signal                ## library that supports frequency response function
                                            i.e signal.freqz
```

Sample outputs



NAME - SANTOSH - [CB.EN.U4CCE20053]
DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE)

LAB TITLE AND CODE - SIGNAL PROCESSING LAB 19CCE281
EXPERIMENT NUMBER - 10
DATE - 28/12/2021

DESIGN OF FIR FILTER

* **AIM :**
For a given frequency response $H(e^{j\omega})$, design a Finite Impulse Response (FIR) low pass filter using rectangular and Hanning window techniques and determine the impulse and frequency response.

* **SOFTWARE REQUIRED :**
Spyder IDE - Anaconda 3 2021.11 (Python 3.9.7 64-bit)
Publisher - Anaconda, Inc.

* **THEORY :**
FIR filters are filters having a transfer function of a polynomial in z and is an all-zero filter in the sense that the zeroes in the z-plane determine the frequency response magnitude characteristic. The z transform of a N-point FIR filter is given by-

$$H(z) = \sum_{n=0}^{N-1} h(n)\, z^{-n}$$

FIR filters are particularly useful for applications where exact phase response is required. The FIR filter is generally implemented in a non-recursive way which guarantees a stable filter.

PTO

→ RECTANGULAR WINDOW METHOD:-

In this method, from the desired frequency response specification $H_d(w)$, corresponding unit sample response $h_d(n)$ is determined using the following relation -

$$h_d(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(w)\, e^{jwn}\, dw$$

where, $H_d(w) = \sum_{n=-\infty}^{\infty} h_d(n)\, e^{-jwn}$

In general, unit sample response $h_d(n)$ obtained from the above relation is infinite in duration, so it must be truncated at some point say $n = M-1$ to yield an FIR filter of length M (i.e. 0 to M-1). This truncation of $h_d(n)$ to length M is same as multiplying $h_d(n)$ by the rectangular window defined as -

$$w(n) = \begin{cases} 1, & 0 \le n \le M-1 \\ 0, & \text{otherwise} \end{cases}$$

Thus, the unit sample response of the FIR filter becomes -

$$h(n) = h_d(n)\, w(n)$$
$$= \begin{cases} h_d(n), & 0 \le n \le M-1 \\ 0, & \text{otherwise} \end{cases}$$

Now, the multiplication of the window function $w(n)$ with $h_d(n)$ is equivalent to convolution of $H_d(w)$ with $W(w)$, where $W(w)$ is the frequency domain representation of the window function -

$$W_d(w) = \sum_{n=0}^{M-1} w(n)\, e^{-jwn}$$

Thus, the convolution of $H_d(w)$ with $W(w)$ yields the frequency response of the truncated FIR filter -

$$H(w) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(v)\, W(w-v)\, dw$$

The frequency response can also be obtained using the following relation -

$$H(w) = \sum_{n=0}^{m-1} h(n)\, e^{-jwn}$$

→ HANNING WINDOW METHOD:-

Similar to rectangular window,

$$w[n] = \sum_{k=0}^{k} (-1)^k a_k \cos\left(\frac{2\pi kn}{N}\right), \quad 0 \le n \le N.$$

In most cases, including the given question, all coefficients $a_k \ge 0$. These windows have only $2k+1$ non-zero N-point DFT coefficients.

The customary cosine-sin windows for case $k=1$ have the form -

$$w[n] = a_0 - \underbrace{(1-a_0)}_{a_1} \cdot \cos\left(\frac{2\pi n}{N}\right), \quad 0 \le n \le N$$

which is easily (and often) confused with its zero-phase version -

$$w_0(n) = w\left(n + \frac{N}{2}\right)$$

$$= a_0 + a_1 \cdot \cos\left(\frac{2\pi n}{N}\right), \quad -\frac{N}{2} \le n \le \frac{N}{2}$$

Setting $a_0 = 0.5$ produces a Hann window -

$$w[n] = 0.5\left[1 - \cos\left(\frac{2\pi n}{N}\right)\right] = \sin^2\left(\frac{\pi n}{N}\right)$$

* GRAPH PLOTTING ALGORITHM :

The following steps were followed -

1. Define the x-axis and corresponding y-axis 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.
6. To set the visibility of the grid inside the figure to on, we use the .grid() function.

* THEORETICAL CALCULATION :

Given -

Number of coefficients, $N = 9$ (also, indicated by M)

Frequency response, $H(e^{j\omega}) = \begin{cases} 1, & -2\pi/5 \le \omega \le 2\pi/5 \\ 0, & \text{otherwise} \end{cases}$

Desired filter coefficient ($h_d[n]$) is obtained by taking Inverse Discrete Time Fourier Transform (IDTFT) on $H(e^{j\omega})$ and yields $h_d[n] = \dfrac{\sin\left(\frac{2\pi n}{5}\right)}{\pi n}$

$\Rightarrow$ for $n=0$, $h_d[0] = \frac{1}{2\pi}\int_{-\pi}^{\pi} 1 \times e^{j\omega n}\, d\omega = 0.4$

similarly, for $n=1$, $h_d[1] = 0.3027$

for $n=2$, $h_d[2] = 0.0935$

for $n=3$, $h_d[3] = -0.06236$

for $n=4$, $h_d[4] = -0.0757$

for $n=5$, $h_d[5] = -0.06236$

for $n=6$, $h_d[6] = 0.0935$

for $n=7$, $h_d[7] = 0.3027$

for $n=8$, $h_d[8] = 0.4$

① RECTANGULAR WINDOW TECHNIQUE:-

Expected filter coefficients ($h[n]$) on multiplying with rectangular window function ($w[n]$) yields the following -

$$w[n] = \begin{cases} 1, & 0 \le n \le M-1 = 0 \le n \le 8 \\ 0, & \text{otherwise} \end{cases}$$

$$h[n] = h_d[n] \times w[n]$$

$$= \begin{cases} h_d[n], & 0 \le n \le 8 \\ 0, & \text{otherwise} \end{cases}$$

Determine the frequency response $H(e^{j\omega})$ by direct substitution of the expression (even symmetry and length is odd integer) -

$$H(\omega) = H_R(\omega)\, e^{-j\omega\left(\frac{M-1}{2}\right)} = H_R(\omega)\, e^{-j\omega\left(\frac{8}{2}\right)}, \quad \left(\frac{8}{2}\right) = 4$$

where $H_R(\omega)$ is a real function of ω and can be expressed as -

$$H_R(\omega) = h\left(\frac{M-1}{2}\right) + 2 \sum_{n=0}^{(M-3)/2} h(n) \cos\left(\frac{M-1}{2} - n\right) ; \text{ when } M \text{ is odd}$$

and $$H_R(\omega) = 2 \sum_{n=0}^{(M/2)-1} h(n) \cos\omega\left(\frac{M-1}{2} - n\right) ; \text{ when } M \text{ is even}$$

Taking ω values from 0 to 180°, with a difference of 10° at regular intervals, we obtain 18 values corresponding for $H(\omega)$, which is converted to decibels by -

$$\text{Decibel Representation} = 20 \times \log_{10} |H(\omega)|$$

→ HANNING WINDOW TECHNIQUE:-

Expected filter coefficients ($h[n]$) on multiplying with Hanning window function ($w[n]$) yields the following -

$$w[n] = 0.5\left(1 - \cos\left(\frac{2\pi n}{m-1}\right)\right) = 0.5\left(1 - \cos\left(\frac{2\pi n}{8}\right)\right)$$

$$h[n] = h_d[n] \times w[n]$$

$$h[n] = \begin{cases} 0.00, & \text{for } n=0 \\ 0.0443, & \text{for } n=1 \\ 0.0468, & \text{for } n=2 \\ -0.0532, & \text{for } n=3 \\ -0.0757, & \text{for } n=4 \\ -0.0532, & \text{for } n=5 \\ 0.468, & \text{for } n=6 \\ 0.0443, & \text{for } n=7 \\ 0.00, & \text{for } n=8 \\ 0, & \text{otherwise} \end{cases}$$

Determine the frequency response, similar to rectangular window technique.

* PROGRAM WITH COMMENTS:

```
1 import matplotlib.pyplot as plt # Provides an implicit way of plotting
2 import numpy as np # Support for large, multi-dimensional arrays and matrices
3 from scipy import integrate # Compute a definite integral
4
5 N= int(input("\n Enter the number of coefficients for FIR filter"))
6 hd_w = int(input(" Enter the frequency response value : "))
```

```
7
8  # Inverse discrete-time Fourier transform:
9  hd_n = []
10 for n in range (N//2 + 1):
11     expression = lambda w: Hd_w * np.exp (1j * w * n)
   # Expression within the integral
12     temp= integrate.quad (expression, (-2 * np.pi)/5,
   (-2 * np.pi)/5)
13     integral = temp[0] / (2 * np.pi)  # quad() gives a tuple,
   integral_value[0] + constant[1]
14     hd_n.append (integral)
15
16 # Compute desired filter coefficients:
17 for n in reverse (range (N//2)): # Backward iteration will start
   occurring from the rear
18     symmetric = hd_n[n]
19     hd_n.append (symmetric)
20
21 print ("\n The desired filter coefficients h_d[n] is as follows: \n
   {}".format (hd_n))
22
23 # Design FIR filter using rectangular window technique
   (Impulse Response):
24
25 # Rectangular window function:
26 wn_rectangular = []
27 for n in range (N):
28     if n<N:
29         temp = 1
30     else:
31         temp=0
32     wn_rectangular.append (temp)
33
```

```
34 # Expected filter coefficients obtained using rectangular
   window technique:
35 hn_rectangular = []
36 for n in range(N):
37     hn_rectangular.append(hd_n[n] * wn_rectangular[n])
38
39 # sketch expected filter coefficients obtained using
   rectangular window technique:
40 plt.xlabel('n')
41 plt.ylabel('Magnitude of h[n]')
42 plt.title("Impulse Response Using Rectangular Window
   Technique")
43 plt.stem(np.arange(0, len(hn_rectangular)), hn_rectangu
44 plt.grid(True)
45 plt.show()
46 print("\n The expected filter coefficients h[n] obtained using
   rectangular window technique is as follows: \n {}".
   format(hn_rectangular))
47
48 # Design FIR filter using rectangular window technique
   (Frequency Response):
49
50 hq_w_rectangular = [] # Real function of ω
51 h_w_rectangular = [] # Frequency response (magnitude)
52 h_w_rectangular_decibels = [] # Frequency response (decibe
53 index_rectangular = 0 # Index variable to iterate through
   "hr_w_rectangular" list
54 x_axis_rectangular = [] # Define the angle axis
55
56 for degrees in range(0, 190, 10):
57
58     w_rectangular = degrees * (np.pi/180)
```

```
    sum_rectangular = 0

    if (N % 2 == 1):

        for n in range ((N-3)//2) + 1):
            summation = hn_rectangular[n] *
np.cos(w_rectangular) * ((N-1)//2) - n)
            sum_rectangular = sum_rectangular + summation
        hr_w_rectangular.append(hn_rectangular[(N-1)//2] +
(2 * sum_rectangular))

    else:

        for n in range((N//2) - 1):
            summation = hn_rectangular[n] *
np.cos(w_rectangular) * ((N-1)//2) - n)
            sum_rectangular = sum_rectangular + summation
        hr_w_rectangular.append(2 * sum_rectangular)

    h_w_rectangular.append(hr_w_rectangular[index_
rectangular] * np.exp(-1j * w_rectangular * ((N-1)//2)))

    h_w_rectangular_decibels.append(20 * np.log10(abs(
h_w_rectangular[index_rectangular])))

    index_rectangular = index_rectangular + 1

    x_axis_rectangular.append(degrees)

# sketch frequency response obtained using rectangular
window technique:
plt.xlabel('w in degrees')
plt.ylabel('magnitude of H[w]')
```

```
plt.title("Frequency Response Using Rectangular Window Technique")
plt.plot(x_axis, rectangular, h_w_rectangular_decibels)
plt.grid(True)
plt.show()
print("\n The frequency response H[w] obtained using rectangular window technique is as follows: \n\n",
format(h_w_rectangular_decibels)

# Design FIR filter using Hanning window technique (Impulse Response):

# Hanning window function:
wn_hanning = []
for n in range(N):
    wn_hanning.append(0.5 * (1 - np.cos((2 * np.pi * n)/(N-1))))

# expected filter coefficients obtained using Hanning window technique:
hn_hanning = []
for n in range(N):
    hn_hanning.append(hd_n[n] * wn_hanning[n])

# Sketch expected filter coefficients obtained using Hanning window technique:
plt.xlabel('n')
plt.ylabel('magnitude of h[n]')
plt.title("Impulse Response using Hanning Window Technique")
plt.stem(np.arange(0, len(hn_hanning)), hn_hanning)
plt.grid(True)
plt.show()
```

```
print("\n The expected filter coefficients h[n] obtained using
Hanning window technique is as follows: \n {}".
format(hn_hanning))

# Design FIR filter using Hanning window technique
(Frequency Response):

hn_w_hanning = [] # Real function of w
h_w_hanning = [] # Frequency response in magnitude
h_w_hanning_decibels = [] # Frequency response (decibels)
index_hanning=0 # Index variable to iterate through
'hn_w_hanning' list
x_axis_hanning = [] # Define the angle axis

for degrees in range(0,190,10):

    w_hanning = degrees * (np.pi/180)
    sum_hanning =0

    if (N%2 ==1):

        for n in range(((N-3)/2) + 1):
            summation = hn_Hanning[n] * np.cos(w_hanning) *
((N-1)//2) -n)
            sum_hanning = sum_hanning + summation
        hn_w_hanning.append(hn_hanning[(N-1)//2] + (2 *
sum_hanning))

    else:

        for n in range((N-2)-1):
            summation = hn_hanning[n] * np.cos(w_hanning) *
((N-1)//2) -n)
```

```
137         sum_hanning = sum_hanning + summation
138         hr_w_hanning.append(2 * sum_hanning)
139
140     h_w_hanning.append(hr_w_hanning[index_hanning] *
    np.exp(-1j * w_hanning * (N-1)//2))
141
142     h_w_hanning_decibels.append(20 * np.log10(abs
    (h_w_hanning[index_hanning])))
143
144     index_hanning = index_hanning + 1
145
146     x_axis_hanning.append(degrees)
147
148 # sketch frequency response obtained using Hanning window
    technique:
149 plt.xlabel('w in Degrees')
150 plt.ylabel('H[w] in Decibels')
151 plt.title("Frequency Response using Rectangular Window
    Technique")
152 plt.plot(x_axis_hanning, h_w_hanning_decibels)
153 plt.grid(True)
154 plt.show()
155 print("\n The frequency response H[w] obtained using
    Hanning window technique is as follows: \n {}".
    format(h_w_hanning_decibels))
156
```

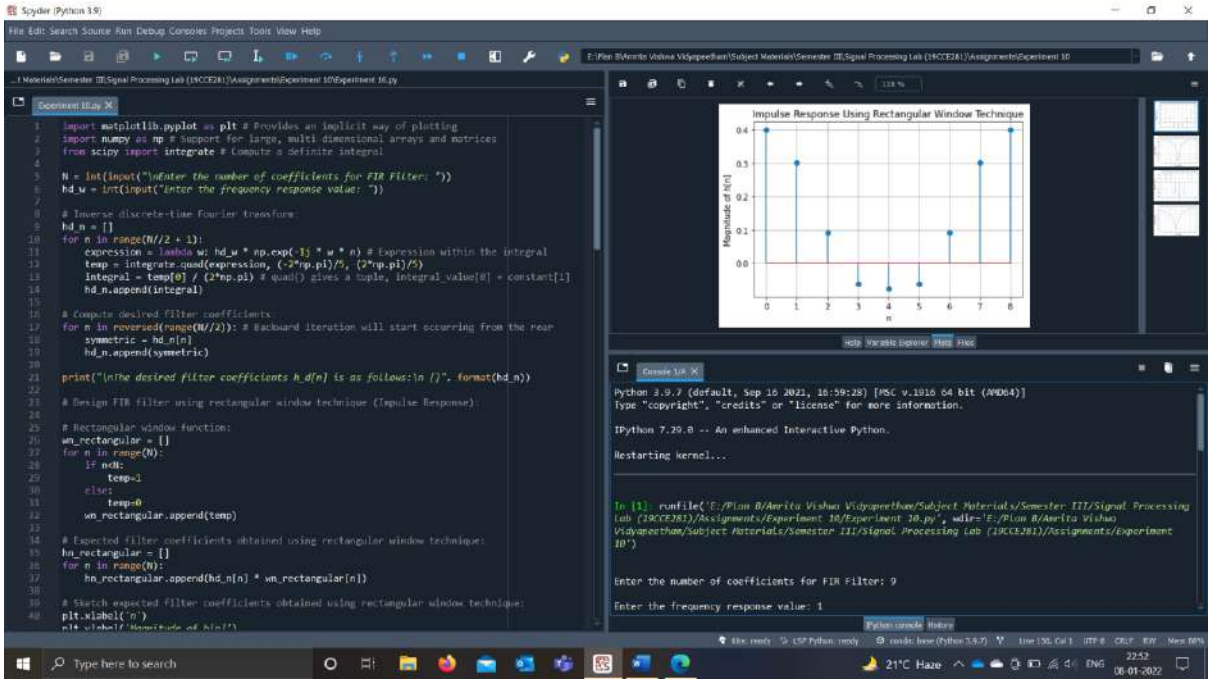
* INFERENCE:

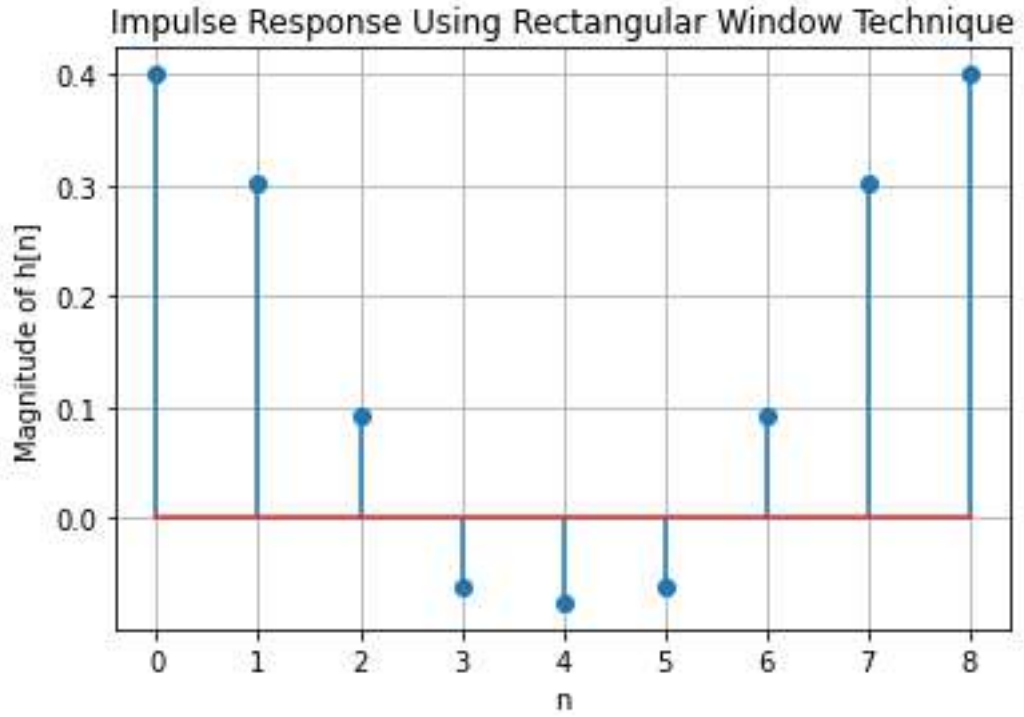
Compute the impulse and frequency response for a FIR low pass filter using rectangular and Hanning window techniques.

RESULTS

VERIFIED.

Impulse Response Using Rectangular Window Technique





Step 1: Import library source files, NumPy, matplotlib.pyplot and SciPy.

Step 2: Enter the number of coefficients for the FIR filter along with the frequency response value.

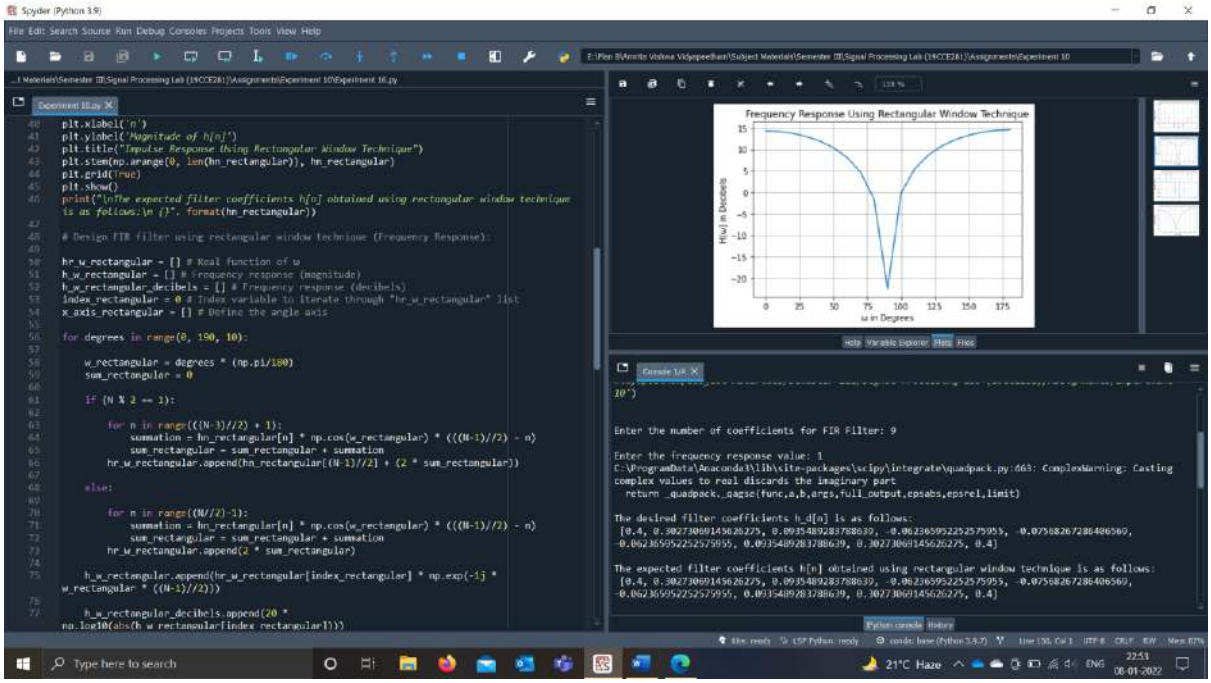
Step 3: Calculate inverse discrete-time Fourier transform and compute desired filter coefficients.

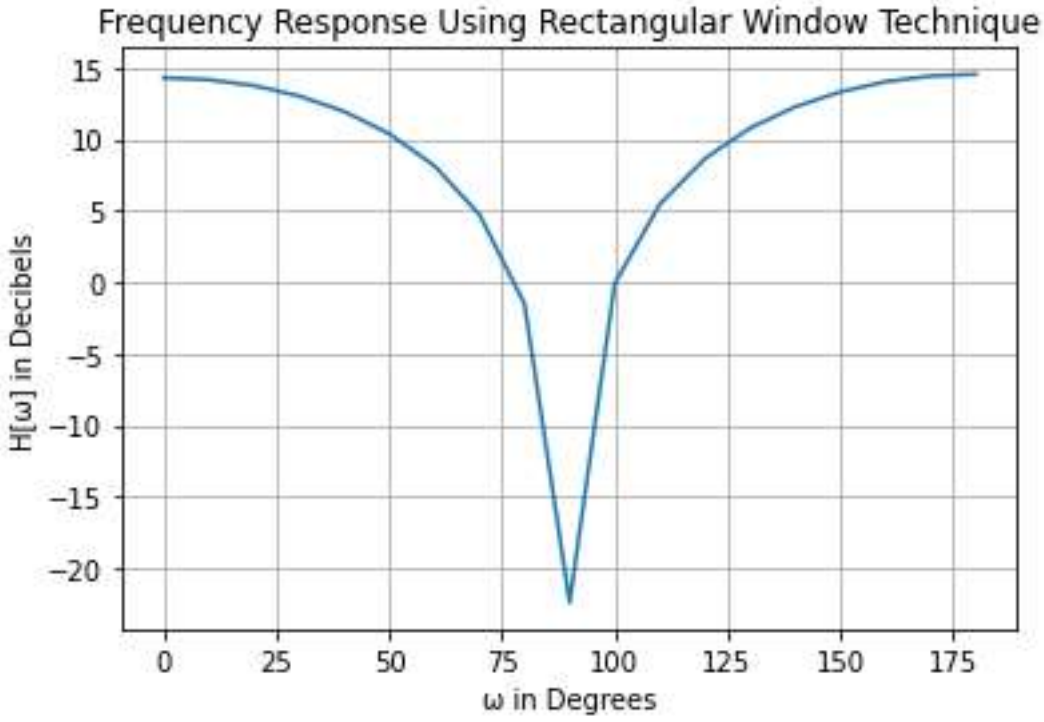
Step 4: Design FIR filter using rectangular window technique (Impulse Response).

Step 5: Compute the rectangular window function.

Step 6: Evaluate expected filter coefficients obtained using the rectangular window technique and sketch the same.

Frequency Response Using Rectangular Window Technique



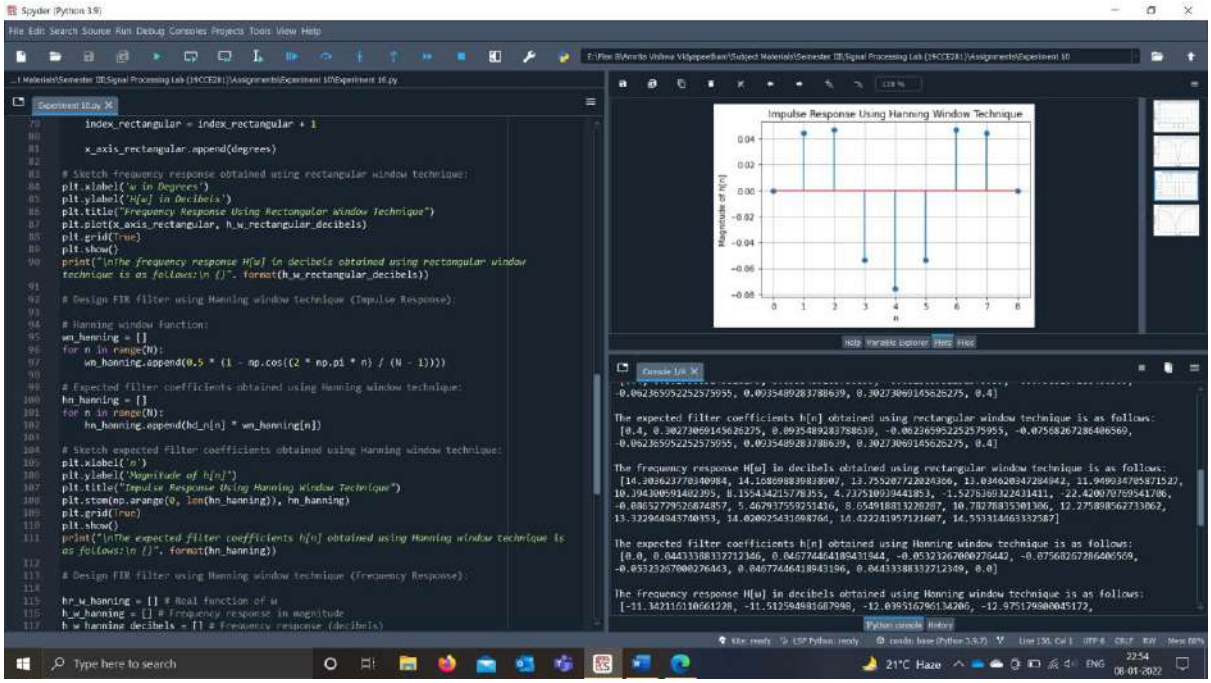


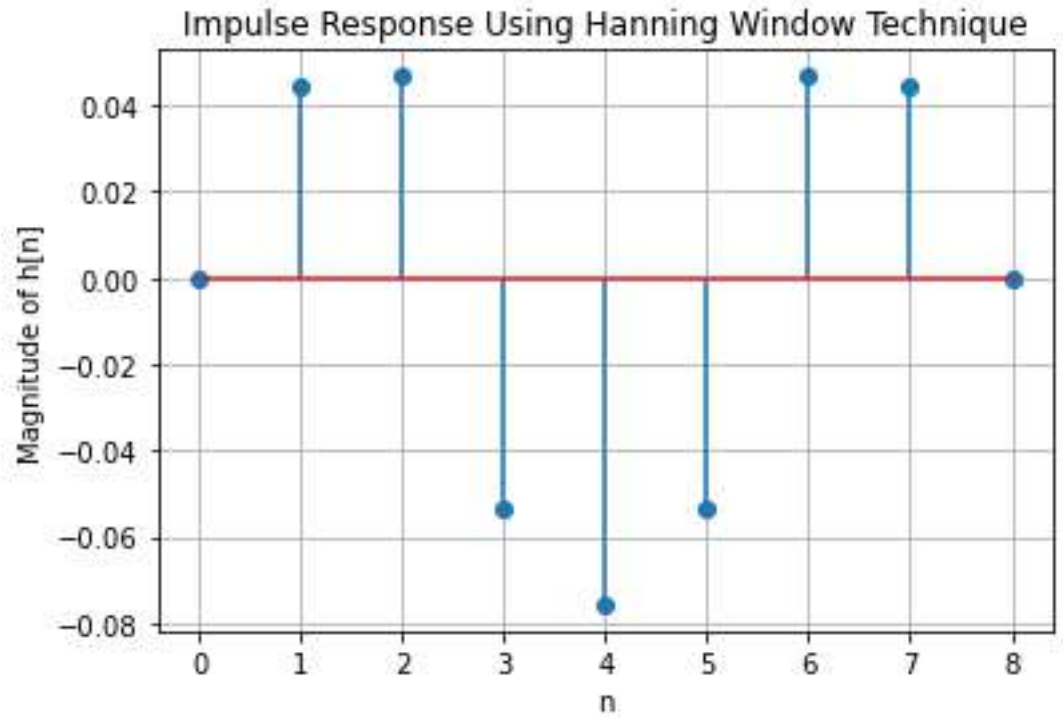
Step 1: Design FIR filter using rectangular window technique (Frequency Response).

Step 2: Initialize the arrays for the real function of ω, frequency response (magnitude), frequency response (decibels) and define the angle axis.

Step 3: Sketch frequency response obtained using rectangular window technique.

Impulse Response Using Hanning Window Technique



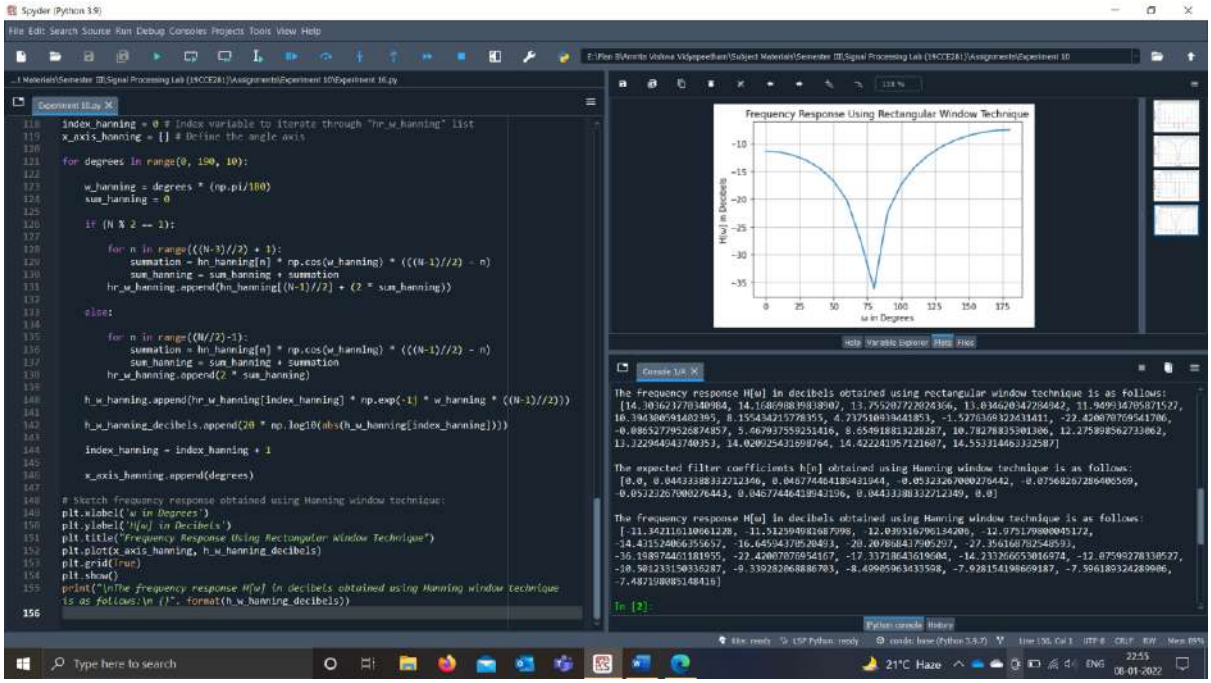


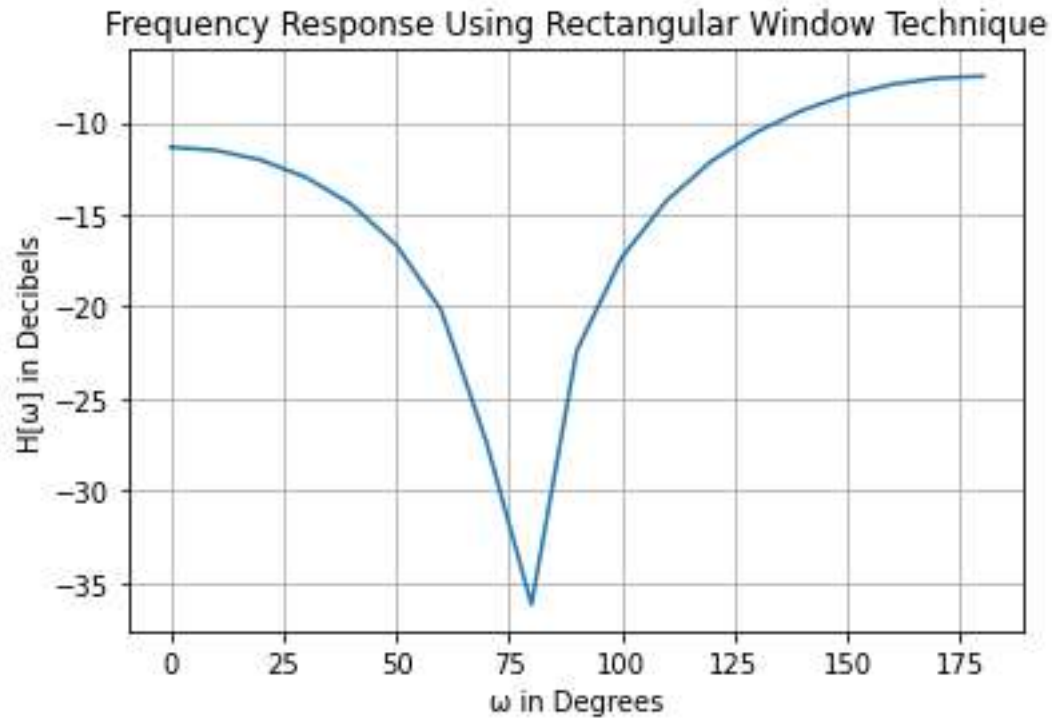
Step 1: Design FIR filter using Hanning window technique (Impulse Response).

Step 2: Compute the Hanning window function.

Step 3: Evaluate expected filter coefficients obtained using the rectangular window technique and sketch the same.

Frequency Response Using Rectangular Window Technique





Step 1: Design FIR filter using Hanning window technique (Frequency Response).

Step 2: Initialize the arrays for the real function of ω, frequency response (magnitude), frequency response (decibels) and define the angle axis.

Step 3: Sketch frequency response obtained using Hanning window technique.

```
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 10/Experiment 10.py'      ='E:/
Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab
(19CCE281)/Assignments/Experiment 10'


Enter the number of coefficients for FIR Filter: 9

Enter the frequency response value: 1
C:\ProgramData\Anaconda3\lib\site-packages\scipy\integrate\quadpack.py:463:
ComplexWarning: Casting complex values to real discards the imaginary part
  return _quadpack._qagse(func,a,b,args,full_output,epsabs,epsrel,limit)

The desired filter coefficients h_d[n] is as follows:
 [0.4, 0.30273069145626275, 0.0935489283788639, -0.062365952252575955,
-0.07568267286406569, -0.062365952252575955, 0.0935489283788639, 0.30273069145626275, 0.4]

The expected filter coefficients h[n] obtained using rectangular window technique is as
follows:
 [0.4, 0.30273069145626275, 0.0935489283788639, -0.062365952252575955,
-0.07568267286406569, -0.062365952252575955, 0.0935489283788639, 0.30273069145626275, 0.4]

The frequency response H[ω] in decibels obtained using rectangular window technique is as
follows:
 [14.303623770340984, 14.168698839838907, 13.755207722024366, 13.034620347284942,
11.949934705871527, 10.394300591402395, 8.155434215778355, 4.737510939441853,
-1.5276369322431411, -22.420070769541706, -0.08652779526874857, 5.467937559251416,
8.654918813228287, 10.78278835301306, 12.275898562733062, 13.322944943740353,
14.020925431698764, 14.422241957121607, 14.553314463332587]

The expected filter coefficients h[n] obtained using Hanning window technique is as
follows:
 [0.0, 0.04433388332712346, 0.046774464189431944, -0.05323267000276442,
-0.07568267286406569, -0.05323267000276443, 0.04677446418943196, 0.04433388332712349, 0.0]

The frequency response H[ω] in decibels obtained using Hanning window technique is as
follows:
 [-11.342116110661228, -11.512594981687998, -12.039516796134206, -12.975179800045172,
-14.431524066355657, -16.64594370520493, -20.207868437905297, -27.356168782548593,
-36.198974461181955, -22.42007076954167, -17.33718643619604, -14.233266653016974,
-12.07599278330527, -10.501233150336287, -9.339282068886703, -8.49905963433598,
-7.928154198669187, -7.596189324289906, -7.487198085148416]

In [2]:
```