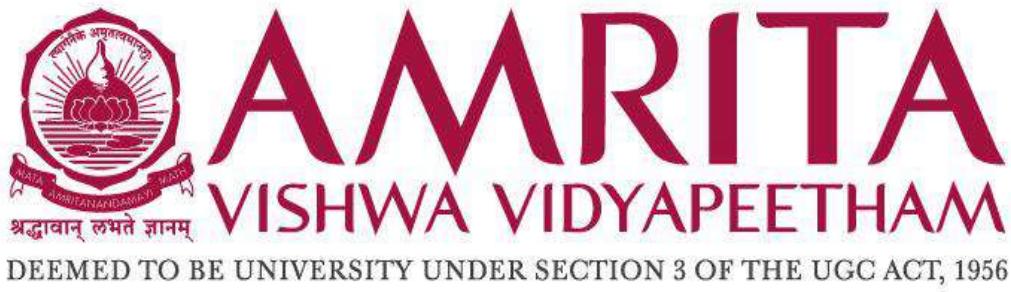


B. Tech – Computer and Communication Engineering (CCE)
Department of Electronics and Communication Engineering
Amrita School of Engineering, Coimbatore – 641112



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

Experiment Number	Experiment Name	Page Number
1	Introduction to Python	3
2	Generation of Elementary Signals	32
3	Operations on Signals	62
4	System Properties	86
5	Convolution Sum	107
6	Fourier Series Representation of Periodic Signals	119
7	Properties of DTFS: Time Shifting	131
8	Frequency Response of LTI Systems	150
9	Discrete Fourier Transform	162
10	FIR Filter Design	178

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:

① NAME - SANTOSH (CB.EN.V4CLC 20053)

DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE)

ACADEMIC YEAR - 2020-21 BATCH

LAB TITLE AND CODE : SIGNAL PROCESSING LAB IACLC 281

EXPERIMENT NUMBER : 1

DATE : 13/09/2021

INTRODUCTION TO PYTHON

* AIM :

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

* SOFTWARE REQUIRED :

Synder 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

②

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 program maintenance. Python supports modules and packages, which encourages modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

+ H ALGORITHM - PYTHON GETTING STARTED AND COMMANDS :

→ P YTHON 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

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.

④ 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!")

(4)

(5)

- ② Multiline 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.

- ③ Creating variables - Python has no command for declaring a variable. A variable is created the moment you first assign a value to it. For example,

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

Output :
5
John

③

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

6

→ 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 INPUT :-

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)
```

①

→ Python Operators :-

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

② 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 ^{o left} right 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	$a \% = b$
$\lfloor =$	Divide (floor) AND: Divide left operand with right operand and then assign the value (floor) to left operand	$a \lfloor = b$
$** =$	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	$a \& = b$
$\mid =$	Performs Bitwise OR on operands and assign value to left operand	$a \mid = b$
$\wedge =$	Performs XOR on operands and assign value to left operand	$a \wedge = b$
$>> =$	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	$x < y$

①

Operator	Description	Syntax
$=$	Equal to : True if both operands are equal	$x = y$
\neq	Not equal to : True if both operands are not equal	$x \neq y$
\geq	Greater than or equal to : True if the left operand is greater than or equal to the right	$x \geq y$
\leq	Less than or equal to : True if the left operand is less than or equal to the right	$x \leq y$

② logical operators -

operator	Description	Syntax
and	Logical AND : True if both the operands are true.	$x \text{ and } y$
or	Logical OR : True if either of the operands is true.	$x \text{ or } y$
not	Logical NOT : True if the operand is false	$\text{not } a$

③ 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 $\text{id}(x)$ equals $\text{id}(y)$	$x \text{ is } y$
is not	Evaluates to false if the variables on either side of the operator point to the same object and false otherwise. Here, it results in 1 if x is not a member of sequence y .	$x \text{ is not } y$

⑥ Membership operators -

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise	a in y, here in results in a 1 if n is a member of sequence y.
not in	Evaluates to true if does not find a variable in the specified sequence and false otherwise.	a not in y, here not in results in a 1 if a 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 operand's 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 operand's value is moved right by the number of bits specified by the right operand.	a>>2 = 15 (means 0000 1111)

①

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,

OUTPUT -

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

```
print(thisdict)
```

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 : $a \leq b$

(v) Greater than : $a > b$

(vi) Greater than or equal to : $a \geq b$

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 $a == b$: # ELSE

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,

13

```

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_function():
 print("Hello from a function!")
```
- ② Calling a function - To call a function, use the function name followed by parenthesis. For example,
- ```
def my_function():
    print("Hello from a function!")

my_function()
# OUTPUT - Hello from a function!
```

- 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 separate 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 function 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_function3("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_function_5(child3, child2, child1):
    print("The youngest child is " + child3)
```

```
my_function_5(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_function_6(**kid):
```

```
    print("His last name is " + kid["name"])
```

```
my_function_6(name = "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_function_7(country = "Norway"):
```

```
    print("I am from " + country)
```

```
my_function_7("Sweden")
```

```
my_function_7("India")
```

```
my_function_7()
```

```
my_function_7("Brazil")
```

OUTPUTS

I am from Sweden

I am from India

I am from Norway

I am from Brazil

- ② 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 n in food:
        print(n)
```

```
fruits = ["apple", "banana", "cherry"]
my_function8(fruits)
```

OUTPUT:
apple
banana
cherry

- ③ 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.
```

(19)

→ PYTHON LIBRARIES :-

- ① Numpy - Numerical Python (NumPy) is a python library used for working with arrays. It also has functions for 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
a = np.ones(10)
plt.subplot(121)
plt.plot(a)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('continuous unit step')
plt.subplot(122)
plt.stem(a)
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

The screenshot shows the Spyder Python 3.8 IDE interface. In the top left, the code editor displays a script named `Introduction_To_Python.py` with the following content:

```
# -*- coding: utf-8 -*-
"""
This is a temporary script file.

### Python Introduction
print("Hello, World!") #Prints Hello, World!
if 5 > 2:
    print("Five is greater than two!\n")

### Python Comments
#This is a comment
print("Hello, World!")
print("Hello, World!") #this is a comment
print("Hello, World!")
print("Cheers, Mate!")
#This is a comment
written in
more than just one line
print("Hello, World!")

### Python Variables
print("\n")
x = 5
y = "John"
print(x)
print(y)

x = 4      # x is of type int
x = "Sally" # x is now of type str
```

In the top right, there are two plots: "Continuous unit step" and "Discrete unit step". Below the plots is the IPython console window, which shows the output of the script's execution.

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

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

In [2]: 
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

The bottom of the screen shows the Windows taskbar with various pinned icons.

Python Comments

This screenshot is identical to the previous one, showing the same version of the `Introduction_To_Python.py` script in the code editor. The plots in the top right remain the same. The IPython console window shows the same output as the first screenshot, demonstrating the execution of the script.

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

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

In [2]: 
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

Python Variables

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays a script named `Introduction_to_Python.py` with the following content:

```
25 """
26     This is a comment
27     written in
28     more than just one line
29 """
30 print("Hello, World!")
31
32 ### Python variables
33 print("\n")
34 x = 5
35 y = "John"
36 print(x)
37 print(y)
38
39 x = 4      # x is of type int
40 x = "Sally" # x is now of type str
41 print(x)
42
43 x = str(3) # x will be '3'
44 y = int(3) # y will be 3
45 z = float(3) # z will be 3.0
46
47 x = 5
48 y = "John"
49 print(type(x))
50 print(type(y))
51
52 x = "John"
53 # is the same as
54 x = "John"
55
56 # This will create two variables
57 a = 4
58 A = "Sally"
59 #A will not overwrite a
60
61 ### Python Numbers
```

The IPython console shows the output of the code, including variable types and values. The plot area displays two side-by-side plots: "Continuous unit step" and "Discrete unit step".

Python Numbers

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays a script named `Introduction_to_Python.py` with the following content:

```
37 print(y)
38
39 x = 4      # x is of type int
40 x = "Sally" # x is now of type str
41 print(x)
42
43 x = str(3) # x will be '3'
44 y = int(3) # y will be 3
45 z = float(3) # z will be 3.0
46
47 x = 5
48 y = "John"
49 print(type(x))
50 print(type(y))
51
52 x = "John"
53 # is the same as
54 x = "John"
55
56 # This will create two variables
57 a = 4
58 A = "Sally"
59 #A will not overwrite a
60
61
62 ### Python Numbers
63 print("\n")
64 x = 1      # int
65 y = 2.8    # float
66 z = 1j     # complex
67
68 print(type(x))
69 print(type(y))
70 print(type(z))
71
72 ### Python Assign Values to Multiple Variables
73 print("\n")
74 x, y, z = "Orange", "Banana", "Cherry"
```

The IPython console shows the output of the code, including variable types and values. The plot area displays two side-by-side plots: "Continuous unit step" and "Discrete unit step".

Python Assign Values to Multiple Variables

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays a script named `Introduction_to_Python.py` with the following content:

```
49 print(type(x))
50 print(type(y))
51
52 x = "John"
53 # is the same as
54 x = "John"
55
56 # this will create two variables
57 a = 4
58 A = "Sally"
59 #A will not overwrite a
60
61 ### Python Numbers
62 print("\n")
63 x = 1 # int
64 y = 1.0 # float
65 z = 1j # complex
66
67 print(type(x))
68 print(type(y))
69 print(type(z))
70
71 ### Python Assign Values to Multiple Variables
72 print("\n")
73 x, y, z = "Orange", "Banana", "Cherry"
74 print(x)
75 print(y)
76 print(z)
77
78 x = y = z = "Orange"
79 print(x)
80 print(y)
81 print(z)
82
83 ### Python User Input
84 print("\n")
85 username = input("Enter username: ")
86
87 print(username)
```

The IPython console shows the output of the code, including type annotations and the user input:

```
<class 'Str'>
<class 'Int'>
<class 'Float'>
<class 'Complex'>

Orange
Banana
Cherry
Orange
Orange
Orange

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

Orange
Banana
Cherry
Orange
Orange
Orange

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

Python User Input

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays the same script `Introduction_to_Python.py` as the previous screenshot, but with additional code at the bottom:

```
88
89 print(username)
```

The IPython console shows the output of the code, including user input:

```
<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
```

Python Collections (Arrays)

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays a script named `Introduction_to_Python.py` with the following content:

```
73 x, y, z = "Orange", "Banana", "Cherry"
74 print(x)
75 print(y)
76 print(z)
77
78 x = y = z = "Orange"
79 print(x)
80 print(y)
81 print(z)
82
83 ## Python User Input
84 print("\n")
85 username = input("Enter username: ")
86 print("Username is: " + username)
87
88 ## Python Collections (Arrays)
89 print("\n")
90 thislist = ["apple", "banana", "cherry"]
91 print(thislist) # List
92
93 thistuple = ("apple", "banana", "cherry")
94 print(thistuple) # Tuple
95
96 thisset = {"apple", "banana", "cherry"}
97 print(thisset) # Set
98
99 thisdict = {
100     "brand": "Ford",
101     "model": "Mustang",
102     "year": 1964
103 }
104 print(thisdict) # Dictionaries
105
106 print("\n")
107 a = 33 # If
108 b = 200
109 if b > a:
110     print("b is greater than a")
111
112 a = 33 # Elif
113 b = 33
114 if b > a:
115     print("b is greater than a")
116 elif a == b:
117     print("a and b are equal")
118
119 a = 200 # Else
120 b = 33
121 if b > a:
122     print("b is greater than a")
123 elif a == b:
124     print("a and b are equal")
125 else:
126     print("a is greater than b")
127
128 a = 200 # And
129 b = 33
130 c = 200
131 if a > b and c > a:
132     print("Both conditions are True")
133
134 a = 200 # Or
135 b = 33
136 c = 500
137 if a > b or a > c:
138     print("At least one of the conditions is True")
139
140 x = 41 # Nested If
141 if x > 10:
142     print("Above ten,")

The IPython console shows the execution of the code, including user input for a username and various print statements for lists, tuples, sets, and dictionaries. Two plots are visible in the background: a continuous unit step function from 0 to 10 and a discrete unit step function from 0 to 8.
```

Python If ... Else

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays a script named `Introduction_to_Python.py` with the following content:

```
106 print("\n")
107 a = 33 # If
108 b = 200
109 if b > a:
110     print("b is greater than a")
111
112 a = 33 # Elif
113 b = 33
114 if b > a:
115     print("b is greater than a")
116 elif a == b:
117     print("a and b are equal")
118
119 a = 200 # Else
120 b = 33
121 if b > a:
122     print("b is greater than a")
123 elif a == b:
124     print("a and b are equal")
125 else:
126     print("a is greater than b")
127
128 a = 200 # And
129 b = 33
130 c = 200
131 if a > b and c > a:
132     print("Both conditions are True")
133
134 a = 200 # Or
135 b = 33
136 c = 500
137 if a > b or a > c:
138     print("At least one of the conditions is True")
139
140 x = 41 # Nested If
141 if x > 10:
142     print("Above ten,")

The IPython console shows the execution of the code, including user input for a variable x and various print statements. The output includes a list of car details and several conditional print statements. Two plots are visible in the background: a continuous unit step function from 0 to 10 and a discrete unit step function from 0 to 8.
```

Python While Loop

The screenshot shows the Spyder Python 3.8 IDE interface. On the left, the code editor displays `Introduction_to_Python.py` with the following content:

```
124     print("a and b are equal")
125 else:
126     print("a is greater than b")
127
128 a = 200 # And
129 b = 33
130 c = 500
131 if a > b and c > a:
132     print("Both conditions are True")
133
134 a = 200 # Or
135 b = 33
136 c = 500
137 if a < b or a > c:
138     print("At least one of the conditions is True")
139
140 x = 41 # Nested If
141 if x > 10:
142     print("Above ten:")
143     if x > 20:
144         print("and also above 20!")
145     else:
146         print("but not above 20.")
147
148 a = 33 # The Pass Statement
149 b = 200
150 if b > a:
151     pass
152
153 i = 1 # While Loop
154 while i < 6:
155     print(i)
156     i += 1
157
158 print("\n")
159 fruits = ["apple", "banana", "cherry"]
160 for x in fruits: # For Loop
```

The IPython console window below shows the output of the code:

```
{'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
```

The status bar at the bottom indicates the current time as 03:36 and date as 21-09-2021.

Python For Loop

The screenshot shows the Spyder Python 3.8 IDE interface. On the left, the code editor displays `Introduction_to_Python.py` with the same code as the previous screenshot, including the nested if statement and the while loop.

The IPython console window shows the output of the code:

```
Above ten,
and also above 20!
1
2
3
4
5

apple
banana
cherry

Hello from a function!
Emil Refnes
Tobias Refnes
Eduardo Rodriguez
```

The status bar at the bottom indicates the current time as 03:36 and date as 21-09-2021.

Python Functions

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays a script named `Introduction_to_Python.py` with the following content:

```
161     ## Python Functions
162     print("\n")
163     def my_function(): # Creating a Function
164         print("Hello from a function!")
165     my_function() # Calling a Function
166
167     def my_function2(fname): # Arguments
168         print(fname + " Refsnes")
169     my_function2("Emil")
170     my_function2("Tobias")
171     my_function2("Linus")
172
173     def my_function3(fname, lname): # Number of Arguments
174         print(fname + " " + lname)
175     my_function3("Emil", "Refsnes")
176
177     def my_function4(kids): # Arbitrary Arguments, *args
178         print("The youngest child is " + kids[2])
179     my_function4("Emil", "Tobias", "Linus")
180
181     def my_function5(child1, child2, child3): # Keyword Arguments
182         print("The youngest child is " + child3)
183     my_function5(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
184
185     def my_function6(**kid): # Arbitrary Keyword Arguments, **kwargs
186         print("His last name is " + kid["lname"])
187     my_function6(fname = "Tobias", lname = "Refsnes")
188
189     def my_function7(country = "Norway"): # Default Parameter Value
190         print("I am from " + country)
191     my_function7("Sweden")
192     my_function7("India")
193     my_function7()
194     my_function7("Brazil")
195
196     def my_function8(food): # Passing a List as an Argument
197         for x in food:
198             print(x)
199
200     fruits = ["apple", "banana", "cherry"]
201     my_function8(fruits)
202
203     def my_function9(x): # Return Values
204         return 5 * x
205     print(my_function9(3))
206     print(my_function9(5))
207     print(my_function9(9))
208
209     ## Python Import Statements
210     print("\n")
211     import random
212     val1 = random.randrange(100);
213     val2 = random.randrange(100);
214
215     print("Following are the two generated random numbers under 100.")
216     print("First: ",val1);
217     print("Second: ",val2);
218
219     ## Python Libraries
220     import numpy as np
```

The IPython console shows the output of the script, including the execution of various functions and the generation of two random numbers. Two plots are displayed in the plots pane: a continuous unit step function and a discrete unit step function.

Python Import Statements

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays a script named `Introduction_to_Python.py` with the following content:

```
164     . print("The youngest child is " + child3)
165     my_function5(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
166
167     def my_function6(**kid): # Arbitrary Keyword Arguments, **kwargs
168         print("His last name is " + kid["lname"])
169     my_function6(fname = "Tobias", lname = "Refsnes")
170
171     def my_function7(country = "Norway"): # Default Parameter Value
172         print("I am from " + country)
173     my_function7("Sweden")
174     my_function7("India")
175     my_function7()
176     my_function7("Brazil")
177
178     def my_function8(food): # Passing a List as an Argument
179         for x in food:
180             print(x)
181
182     fruits = ["apple", "banana", "cherry"]
183     my_function8(fruits)
184
185     def my_function9(x): # Return Values
186         return 5 * x
187     print(my_function9(3))
188     print(my_function9(5))
189     print(my_function9(9))
190
191     ## Python Import Statements
192     print("\n")
193     import random
194     val1 = random.randrange(100);
195     val2 = random.randrange(100);
196
197     print("Following are the two generated random numbers under 100.")
198     print("First: ",val1);
199     print("Second: ",val2);
200
201     ## Python Libraries
202     import numpy as np
```

The IPython console shows the output of the script, including the execution of various functions and the generation of two random numbers. Two plots are displayed in the plots pane: a continuous unit step function and a discrete unit step function.

Python Libraries

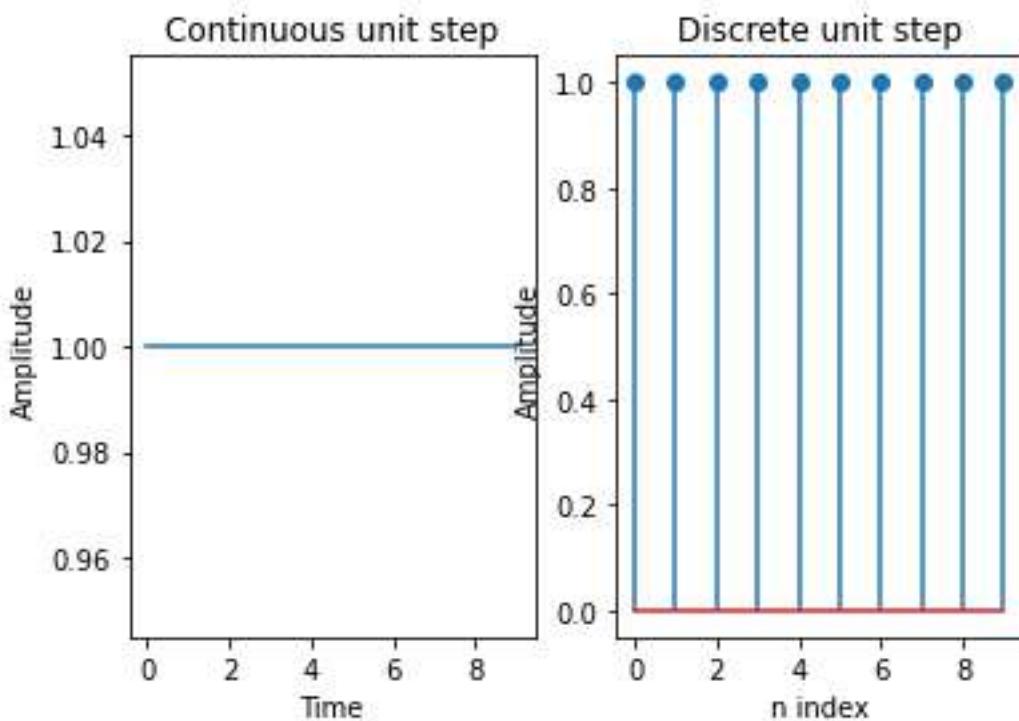
The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named `Introduction_to_Python.py` with the following content:

```
198 def my_functionA(food): # Passing a List as an Argument
199     for x in food:
200         print(x)
201 fruits = ["apple", "banana", "cherry"]
202 my_functionA(fruits)
203
204 def my_functionB(x): # Return Values
205     return 5 * x
206 print(my_functionB(3))
207 print(my_functionB(5))
208 print(my_functionB(9))
209
210 ## Python Import Statements
211 import random
212
213 val1 = random.randrange(100);
214 val2 = random.randrange(100);
215 print("Following are the two generated random numbers under 100.");
216 print("First: ",val1);
217 print("Second: ",val2);
218
219 ### Python Libraries
220 import numpy as np
221 import matplotlib.pyplot as plt
222 x=np.arange(10)
223 plt.subplot(121)
224 plt.plot(x)
225 plt.xlabel('Time')
226 plt.ylabel('Amplitude')
227 plt.title('Continuous unit step')
228 plt.subplot(122)
229 plt.stem(x)
230 plt.xlabel('n index')
231 plt.ylabel('Amplitude')
232 plt.title('Discrete unit step')
```

The right side of the interface shows two plots in the Plots pane:

- Continuous unit step:** A plot of Amplitude versus Time. The amplitude is constant at 1.00 for all time indices from 0 to 9.
- Discrete unit step:** A plot of Amplitude versus n index. The amplitude is 1.00 at every integer index from 0 to 9, represented by vertical blue lines.

The Spyder status bar at the bottom indicates: LSP Python ready, Kite ready, conda Python 3.8.8, Line: 11, Col: 33, UTF-8, CRLF, R/W, Mem: 77%, 02:38, 21-09-2021.



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) $325\sin(100\pi t)$ for $-.02 \leq t \leq 0.05$ with 1000 samples ; (ii) $325\sin(100\pi n)$ for $0 \leq n \leq 50$;
- b) Exponential: (i) $\exp(j100\pi t)$ for $-.02 \leq t \leq 0.05$ with 1000 samples, plot real and imaginary parts;
- (ii) (2^n) for $-10 \leq n \leq 10$;
- c) Unit step ; unit impulse and ramp functions to be generated for $0 \leq n \leq 9$;

① NAME - SANTOSH (CB.EN. V4CLLE 20053)

DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE) A

LAB TITLE AND CODE : SIGNAL PROCESSING LAB V4CLLE 281

EXPERIMENT NO: 2

DATE : 21/07/2021

GENERATION OF ELEMENTARY SIGNALS

* AIM:

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

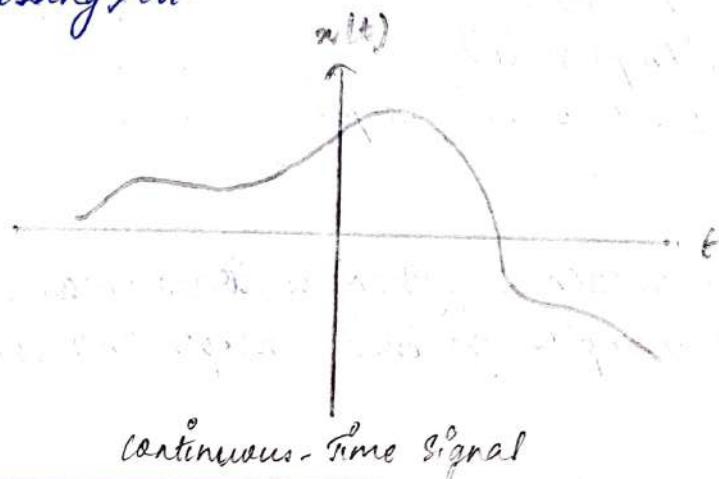
* SOFTWARE REQUIRED :

Synder 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.

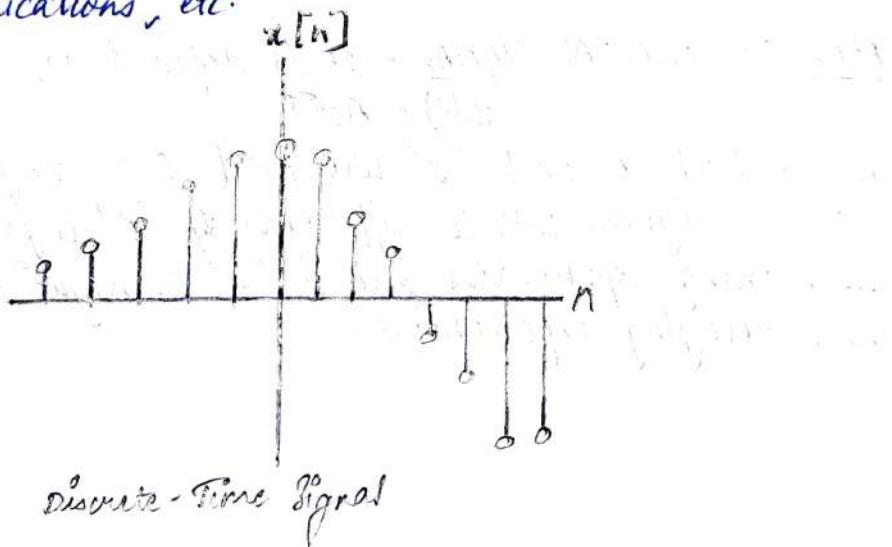
Defined for all time instants and not at specific time instants. The electrical signals derived in proportion with the physical quantities such as temperature, pressure, sound etc. are generally continuous. Applications of these signals include Public Switched Telephone Network (PSTN), FM radio broadcasting,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



②

- 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, separate "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.

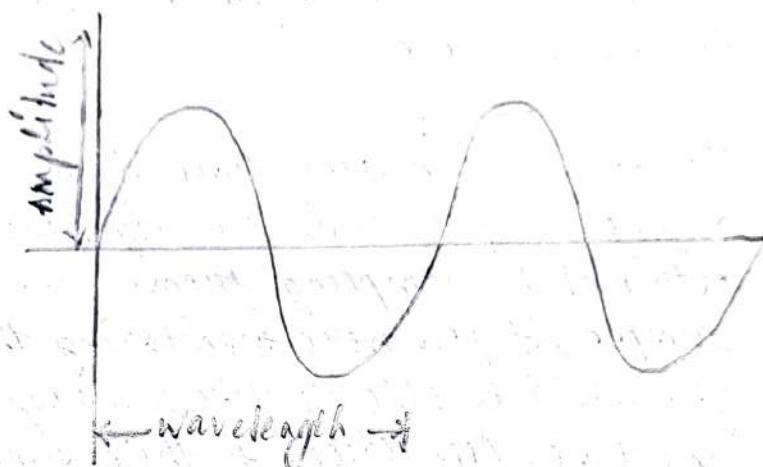


- ③ Sinusoidal signals - These are periodic functions that are based on the sine or cosine function from trigonometry. When the sine wave starts from zero and covers positive values, reaches zero; and again covers negative values, reaches zero, it is said to have completed one cycle or single cycle.

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.

(3)

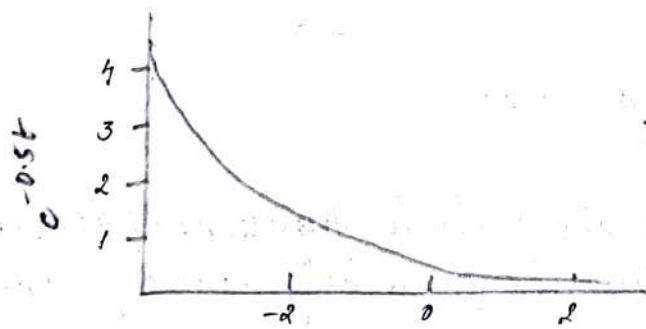
The discrete-time version of a sinusoidal signal, in its most general form, may be written as $a[n] = A \cos(\omega n + \phi)$, where the angular frequency ω must be rational multiple of 2π for the signal to be periodic. Here, the period is measured in samples.



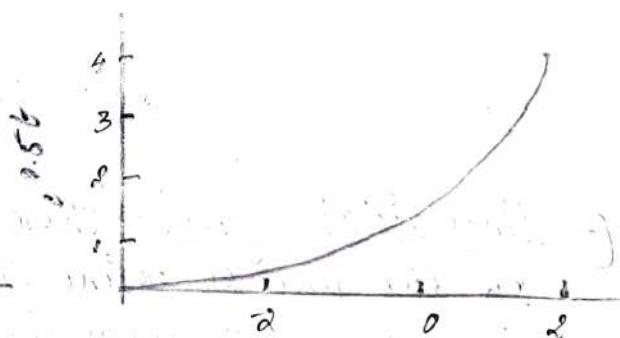
④ REAL EXPONENTIAL SIGNAL - It is defined as,

$$a(t) = Ae^{\delta t}$$

where both 'A' and ' δ ' are real. Depending on the value of the ' δ ' the signals will be different. If ' δ ' is positive, the signal $a(t)$ is growing exponential and if ' δ ' is negative, then the signal $a(t)$ is a decaying exponential.



(a) Decaying Exponential ($\delta < 0$)



(b) Growing Exponential ($\delta > 0$)

⑤ COMPLEX EXPONENTIAL SIGNAL - It is defined as,

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

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

$$s = \delta + j\omega$$

④

$$\text{Therefore, } x(t) = e^{st} = e^{(6+j\omega)t} = e^{st} e^{j\omega t} - 0 \quad (1)$$

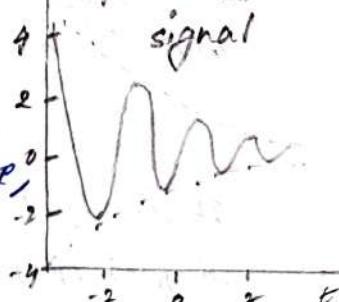
Using Euler's identity,

$$e^{j\omega t} = e^{j\theta} = \cos \theta + j \sin \theta - 0$$

Substituting equation ② in equation ①, we have,

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

Decaying complex exponential signal



⑤

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.

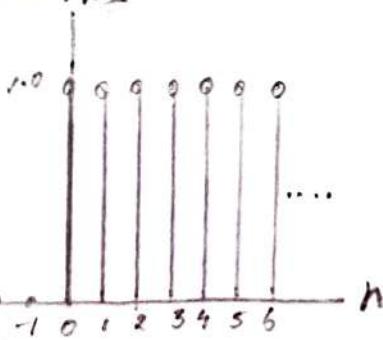
⑥

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

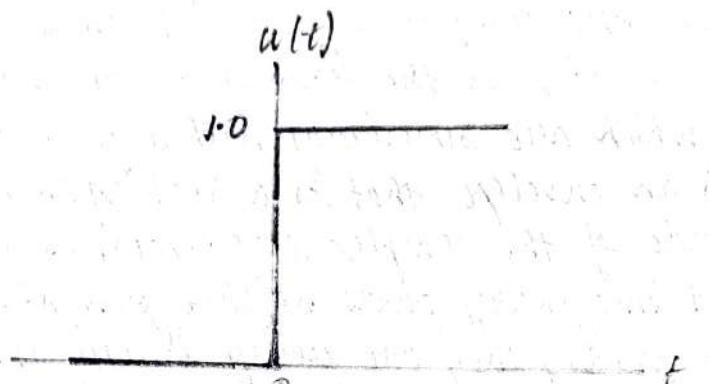


(15)

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 $t=0$.

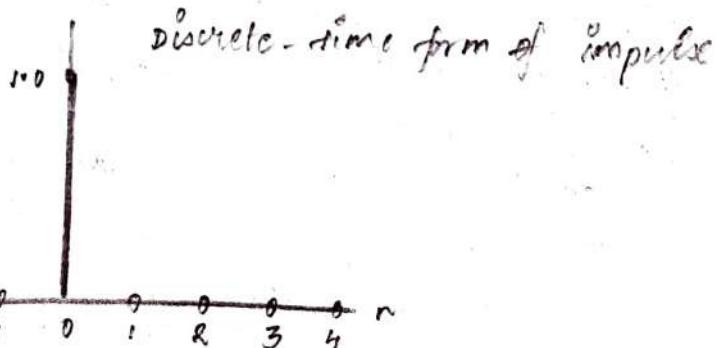


continuous-time version of step function of unit impulse

⑧ 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 -

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



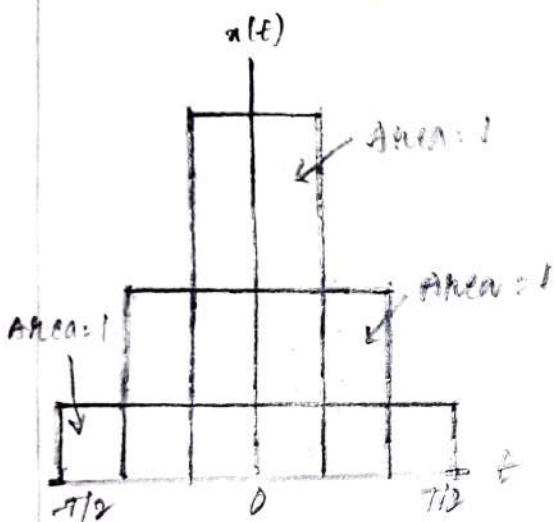
①

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

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

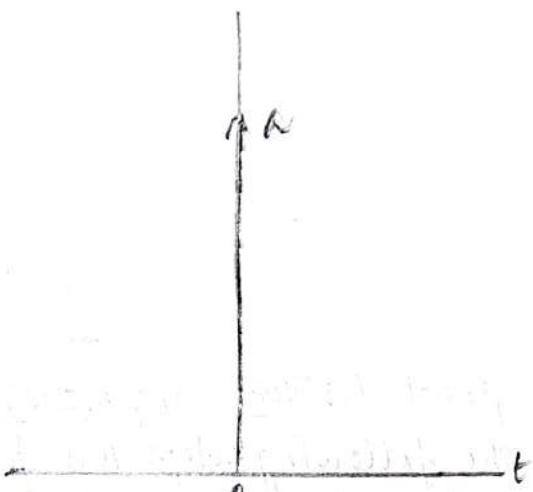
and

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



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

$$a\delta(t)$$



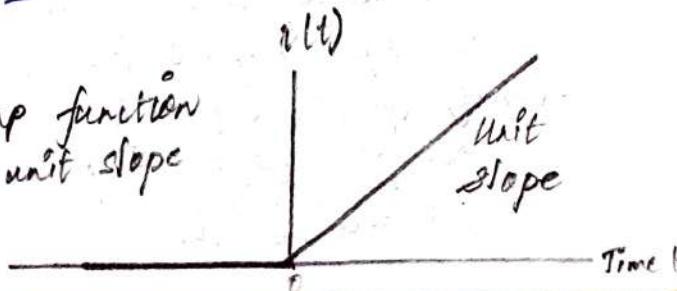
Graphical symbol for an impulse of strength a

- ② 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 the unit ramp function (slope 1, starting at 0).

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

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

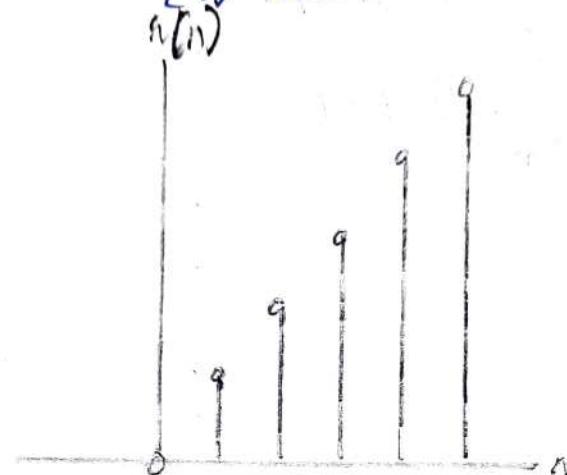
ramp function
of unit slope



⑦

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

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



Discrete-time version of the impulse 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 plots 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')
```

⑧

```
plt.ylabel('x(t)')
plt.title('Plot of CT signal: x(t) = 325 * sin(100 * pi * t)')
plt.ylim([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_d(): # Exponential Continuous-Time Signal
    t = np.linspace(0.02, 0.05, 1000) # linspace() creates evenly
    spaced sequences
    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.ylim([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
```

⑨

```

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 axis
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 array
        if n[sample] < 0:
            temp = j(2**n[sample])) # conversion of negative power to fractions ( $2^{-n} = \frac{1}{2^n}$ )
        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 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

```

(10)

```

else
    temp = 0
arr.append(temp) # adds a single item to the existing list
plt.xlabel('t')
plt.ylabel('arr')
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.arange(-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('arr[n]')
    plt.title('Unit Step Discrete - Time Function for DSNSA')
    plt.stem(n, arr) # stem plot plots vertical lines at each
n 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.arange(-2, 10) # Get evenly spaced values within a given
interval
    arr = [] # Null Array Declaration

```

⑪

```

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.arange(0, 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 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

```

10

```

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('n(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 n
    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": In 1. Sinusoidal Continuous-Time Signal. In
        2. Sinusoidal Discrete-Time Signal. In 3. Exponential
        continuous-Time Signal. In 4. Exponential discrete-Time
        signal. In 5. Unit Step Continuous-Time Signal. In 6. Unit Step
        Discrete-Step Signal. In 7. Unit Impulse Continuous-Time
        Signal. In 8. Unit Impulse Discrete-Time Signal. In 9. Ramp
        Function Continuous-Time Signal. In 10. Ramp Function Discrete-
        Time signal. In 11. Exit In Enter the number corresponding to
        the menu to implement the choice: ') # Menu Driven
        Implementation

```

(13)

```
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):  
    step_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
```

(14)

else :

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

+ INFEREN CE:

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

Sinusoidal Continuous-Time Signal

Spawner (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Experiment 2.py

```
# coding: utf-8 -*-
...
Created on Tue Sep 21 13:56:09 2021
Author: Dell
...
import numpy as np
import matplotlib.pyplot as plt

def sinusoidal_ct(): # Sinusoidal Continuous-Time Signal
    t = np.linspace(-0.02, 0.05, 1000) # linspace() creates evenly spaced sequences
    x = 325 * np.sin(2 * np.pi * 50 * t)
    plt.title('Plot of CT signal: x(t) = 325*sin(100*pi*t)')
    plt.xlabel('t')
    plt.ylabel('x(t)')
    plt.ylim([-300, 300])
    plt.show() # To view your plot

def sinusoidal_dt(): # Sinusoidal Discrete-Time Signal
    n = np.arange(50) # Get evenly spaced values within a given interval
    dt = 0.001 # Set the signal source Resolution
    x = 325 * np.cos(2 * np.pi * 50 * n * dt)
    plt.title('Plot of DT signal: x[n] = 325*cos(100*pi*n*dt)')
    plt.xlabel('n')
    plt.ylabel('x[n]')
    plt.ylim([-300, 300])
    plt.stem(n, x) # Stem plot plots vertical lines at each x position covered under the graph from the bottom to the top of the marker there.
    plt.show() # To view your plot

def exponential_ct(): # Exponential Continuous-Time Signal
    t = np.linspace(-0.02, 0.05, 1000) # linspace() creates evenly spaced sequences
    x = np.exp(2 * np.pi * 50 * t) * np.cos(100 * np.pi * t) # Returns the real part of the complex argument
    plt.plot(t, x)
    plt.title('Plot of CT signal: x(t) = e^(2*pi*50*t)*cos(100*pi*t)')
    plt.xlabel('t')
    plt.ylabel('x(t)')
    plt.ylim([-300, 300])
    plt.show() # Set the x-limits of the current axes
    print('Plot of CT signal: x(t) = 325*sin(100*pi*t)')

def exponential_dt(): # Exponential Discrete-Time Signal
    n = np.arange(50) # Get evenly spaced values within a given interval
    dt = 0.001 # Set the signal source Resolution
    x = np.exp(2 * np.pi * 50 * n * dt) * np.cos(100 * np.pi * n * dt) # Returns the real part of the complex argument
    plt.plot(n, x)
    plt.title('Plot of DT signal: x[n] = e^(2*pi*50*n*dt)*cos(100*pi*n*dt)')
    plt.xlabel('n')
    plt.ylabel('x[n]')
    plt.ylim([-300, 300])
    plt.show() # Set the x-limits of the current axes
    print('Plot of DT signal: x[n] = 325*cos(100*pi*n*dt)')

def unit_step_ct():
    t = np.linspace(-0.02, 0.05, 1000)
    x = np.heaviside(t, 1)
    plt.title('Plot of CT signal: x(t) = u(t)')
    plt.xlabel('t')
    plt.ylabel('x(t)')
    plt.ylim([-0.5, 1.5])
    plt.show() # To view your plot

def unit_step_dt():
    n = np.arange(50)
    x = np.heaviside(n, 1)
    plt.title('Plot of DT signal: x[n] = u(n)')
    plt.xlabel('n')
    plt.ylabel('x[n]')
    plt.ylim([-0.5, 1.5])
    plt.stem(n, x)
    plt.show() # To view your plot

def unit_impulse_ct():
    t = np.linspace(-0.02, 0.05, 1000)
    x = np.zeros(t.shape)
    x[0] = 1
    plt.title('Plot of CT signal: x(t) = delta(t)')
    plt.xlabel('t')
    plt.ylabel('x(t)')
    plt.ylim([-0.5, 1.5])
    plt.show() # To view your plot

def unit_impulse_dt():
    n = np.arange(50)
    x = np.zeros(n.shape)
    x[0] = 1
    plt.title('Plot of DT signal: x[n] = delta(n)')
    plt.xlabel('n')
    plt.ylabel('x[n]')
    plt.ylim([-0.5, 1.5])
    plt.stem(n, x)
    plt.show() # To view your plot

def ramp_function_ct():
    t = np.linspace(-0.02, 0.05, 1000)
    x = t
    plt.title('Plot of CT signal: x(t) = t')
    plt.xlabel('t')
    plt.ylabel('x(t)')
    plt.ylim([-0.5, 1.5])
    plt.show() # To view your plot

def ramp_function_dt():
    n = np.arange(50)
    x = n
    plt.title('Plot of DT signal: x[n] = n')
    plt.xlabel('n')
    plt.ylabel('x[n]')
    plt.ylim([-0.5, 1.5])
    plt.stem(n, x)
    plt.show() # To view your plot
```

Plot of CT signal: x(t) = 325*sin(100*pi*t)

Variable explorer Help Index files

In [1]: runfile('E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2/Experiment 2.py', wdir='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2')

MEMO:

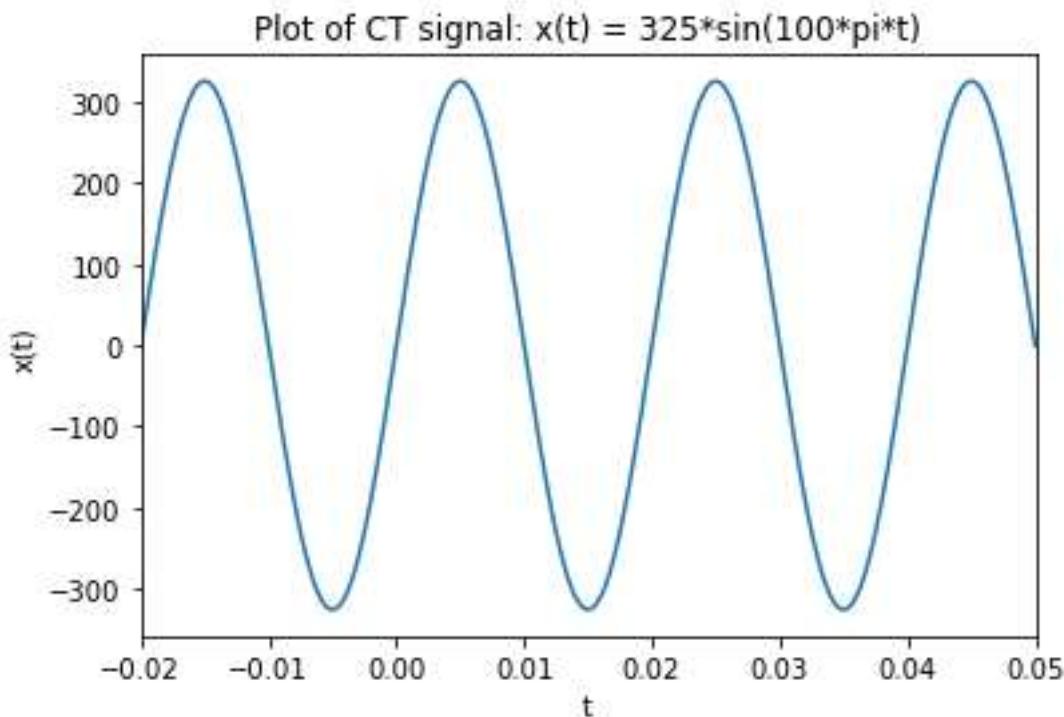
1. Sinusoidal Continuous-Time Signal.
2. Sinusoidal Discrete-Time Signal.
3. Exponential Continuous-Time Signal.
4. Exponential Discrete-Time Signal.
5. Unit Step Continuous-Time Signal.
6. Unit Step Discrete-Time Signal.
7. Unit Impulse Continuous-Time Signal.
8. Unit Impulse Discrete-Time Signal.
9. 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

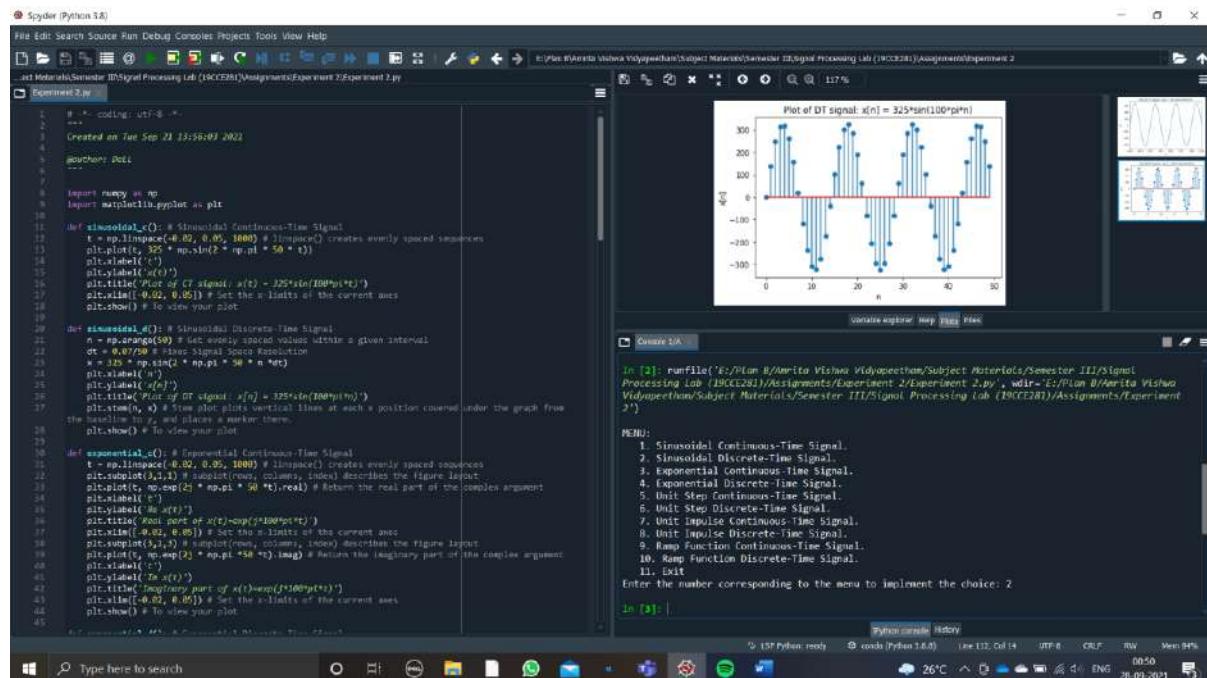
In [2]:

Python console History

26°C 20:47 28-09-2021



Sinusoidal Discrete-Time Signal



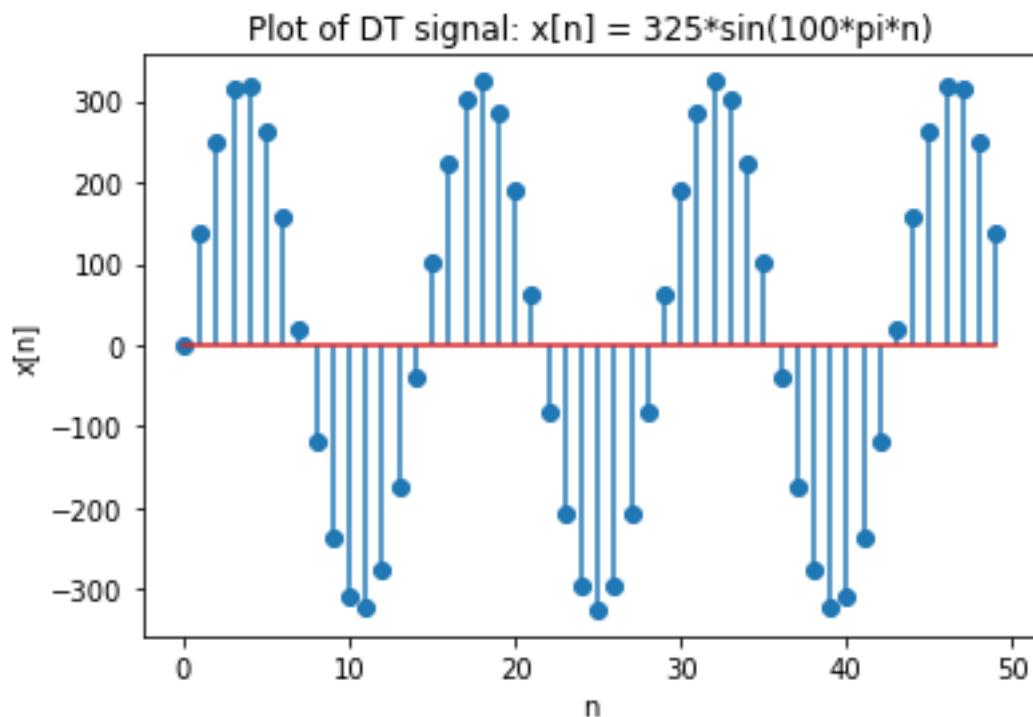
The screenshot shows the Spyder Python IDE interface. The code in the editor window is as follows:

```

# coding: utf-8
# Created on Tue Sep 21 13:56:09 2021
# @author: Dell
# ...
# import numpy as np
# import matplotlib.pyplot as plt
#
# def sinusoidal_c(): # Sinusoidal Continuous-Time Signal
#     n = np.linspace(-0.02, 0.05, 1000) # linspace() creates evenly spaced sequences
#     dt = 0.0001 # gives signal source resolution
#     x = 325 * np.sin(2 * np.pi * 50 * n)
#     plt.xlabel('t')
#     plt.ylabel('x(t)')
#     plt.title('Plot of CT signal: x(t) = 325sin(100pi*t)')
#     plt.ylim([-0.02, 0.05]) # Set the x-axis 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 = 1 # gives signal source 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] = 325sin(100pi*n)')
#     plt.ylim([-0.02, 0.05]) # Set the x-axis limits of the current axes
#     plt.stem(n, x) # Stem plot plots vertical lines at each x position covered under the graph from the bottom to the top of the plot
#     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 sequences
#     plt.plot(t, 1.1 * np.exp(0.1 * t)) # subplot(m,n,1) describes the figure layout
#     plt.plot(t, 0.9 * np.exp(0.1 * t)) # Return the real part of the complex argument
#     plt.xlabel('t')
#     plt.ylabel('x(t)')
#     plt.title('Plot of CT signal: x(t) = (1.1)^t + (0.9)^t')
#     plt.ylim([-0.02, 0.05]) # Set the x-axis limits of the current axes
#     plt.show() # To view your plot
#
# def exponential_d(): # Exponential Discrete-Time Signal
#     n = np.arange(50) # Get evenly spaced values within a given interval
#     dt = 1 # gives signal source resolution
#     x = (1.1)**n + (0.9)**n
#     plt.xlabel('n')
#     plt.ylabel('x[n]')
#     plt.title('Plot of DT signal: x[n] = (1.1)^n + (0.9)^n')
#     plt.ylim([-0.02, 0.05]) # Set the x-axis limits of the current axes
#     plt.show() # To view your plot

```

The plot window displays the discrete-time signal $x[n] = 325\sin(100\pi n)$. The x-axis is labeled n and ranges from 0 to 50. The y-axis is labeled $x[n]$ and ranges from -300 to 300. The signal is a periodic oscillation with an amplitude of 325 and a frequency of 50 Hz. A red horizontal line is drawn at $y=0$.



Exponential Continuous-Time Signal

Spawner (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Experiment 2.ipynb

```
# coding: utf-8
# Created on Tue Sep 21 13:56:09 2021
# Author: Dell
# ...

import numpy as np
import matplotlib.pyplot as plt

def sinusoidal_c(t): # Sinusoidal Continuous-time Signal
    t = np.linspace(-0.02, 0.05, 1000) # linspace() creates evenly spaced sequences
    dt = 0.0001 # Set signal source resolution
    n = 325 # np.pi * 50 / dt
    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) = 325sin(100pi*t)')
    plt.ylim([-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 = 0.0001 # Set signal source resolution
    t = 125 * np.pi * n / 50 # np.pi * 50 / dt
    plt.plot(t, 325 * np.cos(2 * np.pi * 50 * t))
    plt.xlabel('t')
    plt.ylabel('x(n)')
    plt.title('Plot of DT signal: x[n] = 325cos(100pi*n)')
    plt.ylim([-0.02, 0.05]) # Set plot plots vertical lines at each x-position covered under the graph from the command line, so we can see the plot better here.
    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 sequences
    plt.plot(t, 1.0 * np.exp(2 * np.pi * 50 * t)) # Returns 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.ylim([-0.02, 0.05]) # Set the x-limits of the current axes
    plt.show() # To view your plot

def exponential_d(): # Exponential Discrete-Time Signal
    n = np.arange(50) # Get evenly spaced values within a given interval
    dt = 0.0001 # Set signal source resolution
    t = 125 * np.pi * n / 50 # np.pi * 50 / dt
    plt.plot(t, 1.0 * np.exp(2 * np.pi * 50 * t).real) # Returns 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.ylim([-0.02, 0.05]) # Set the x-limits of the current axes
    plt.show() # To view your plot

def exponential_im(): # Exponential Continuous-Time Signal
    t = np.linspace(-0.02, 0.05, 1000) # linspace() creates evenly spaced sequences
    plt.plot(t, 1.0 * np.exp(2 * np.pi * 50 * t).imag) # Returns 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.ylim([-0.02, 0.05]) # Set the x-limits of the current axes
    plt.show() # To view your plot
```

In [4]: runfile('E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2/Experiment 2.ipynb', wdir='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2')

MEMO:

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. Exponential Continuous-Time Signal.
10. Ramp Function Discrete-Time Signal.
11. Exit

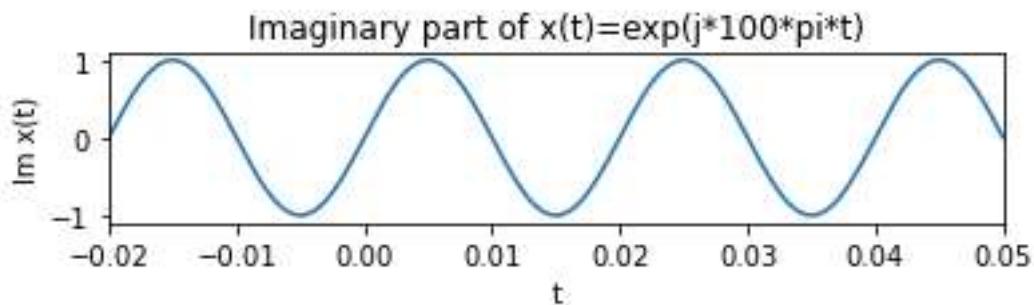
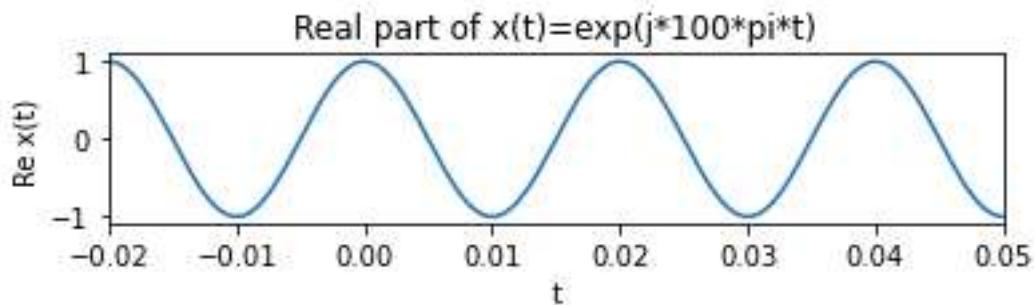
Enter the number corresponding to the menu to implement the choice: 3

In [4]:

Python console History

Type here to search

26°C 20:45 28-09-2021



Exponential Discrete-Time Signal

File Edit Search Source Run Debug Console Projects Tools View Help

Experiment 2.ipynb

```

1. sinoidal_d(): # Sineoidal Discrete-Time Signal.
2. n = np.arange(50) # Get evenly spaced values within a given interval
3. dt = 0.0750 # Fixes Signal Source Resolution
4. x = 125 * np.sin(2 * np.pi * 50 * n * dt)
5. plt.plot(x)
6. plt.title('Plot of DT signal: x[n] = 325*sin(100pi*n)')
7. plt.stem(n, x) # Stem plot plots vertical lines at each n position covered under the graph from
the baseline to y_n and places a marker there.
8. plt.show() # To view your plot

9. Net exponential_d(): # Exponential Continuous-Time Signal
10. t = np.linspace(-0.02, 0.05, 1000) # linspace() creates evenly spaced sequences
11. plt.subplot(3,1,1) # subplot(rows, columns, index) describes the figure layout.
12. plt.plot(t, np.exp(2 * np.pi * 50 * t)) # Returns the real part of the complex argument
13. plt.xlabel('t')
14. plt.ylabel('x(t)')
15. plt.title('Real part of x(t)=exp(j*100pi*t)')
16. plt.ylim([-0.02, 0.05]) # Set the x-limits of the current axes
17. plt.plot(np.real(np.exp(2 * np.pi * 50 * t)), 'o') # Add subplot(1,1,1) describes the figure layout.
18. plt.title('Imaginary part of x(t)=exp(j*100pi*t)')
19. plt.ylim([-0.02, 0.05]) # Set the x-limits of the current axes
20. plt.show() # To view your plot

21. Net exponential_d(): # Exponential Discrete-time Signal
22. n=np.arange(-2,5) # Get evenly spaced values within a given interval
23. arr=[ ] # Null Array Declaration
24. for sample in range(len(n)): # Returns the length of the array
25.     if n[sample]>0:
26.         temp=1/(2**n[sample]) # Conversion of negative power to fractions (2^-1 + 1/2)
27.     else:
28.         temp=2**(-n[sample])
29.     arr.append(temp) # Adds a single item to the existing list
30. plt.plot(n, arr)
31. plt.title('Plot of DT signal: x[n]=2^n')
32. plt.stem(n, arr) # Stem plot plots vertical lines at each n position covered under the graph from
the baseline to y_n and places a marker there.
33. plt.show()

34. Net step_d(): # Unit Step Continuous-Time Signal
35. t=np.linspace(-2, 10, 1000) # linspace() creates evenly spaced sequences
36. arr=[ ] # Null Array Declaration
37.
```

Plot of DT signal: $x[n]=2^n$

In [4]: runfile('E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2.py', wdir='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2')

MEMO:

1. Sineoidal Continuous-Time Signal.
2. Sineoidal 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

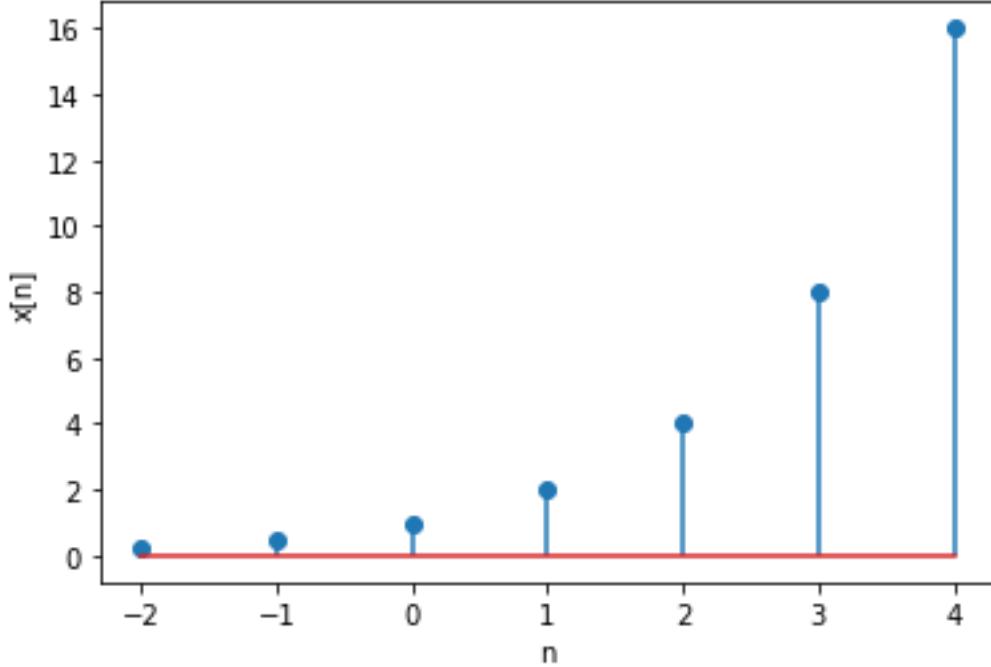
Enter the number corresponding to the menu to implement the choice: 4

20. [5]:

Python console History

Type here to search

Plot of DT signal: $x[n]=2^n$



Unit Step Continuous-Time Signal

Spawner (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Experiment 2.py

```
30:     plt.plot(t, np.exp(2j * np.pi * 50 * t).real) # Return the real part of the complex argument
31:     plt.xlabel('t')
32:     plt.ylabel('x(t)')
33:     plt.title('Real part of x(t)=exp(j*100pi*t)')
34:     plt.ylim([-0.02, 0.05]) # Set the y-limits of the current axes
35:     plt.show() # Create a new figure, column_index describes the figure layout
36:     plt.plot(np.arange(-2, 10), np.pi*50*t).imag) # Return the imaginary part of the complex argument
37:     plt.xlabel('t')
38:     plt.ylabel('Imaginary part of x(t)=exp(j*100pi*t)')
39:     plt.ylim([-0.02, 0.05]) # Set the y-limits of the current axes
40:     plt.show() # To view your plot
41:
42: def exponential_d(): # Exponential Discrete-Time Signal
43:     n=np.arange(-2, 5) # Get evenly spaced values within a given interval
44:     arr=[0]*len(n) # Initialize array
45:     for sample in range(len(n)): # Returns the length of the array
46:         if (sample!=0):
47:             temp=1/(2**(-n[sample])) # Conversion of negative power to fractions (2^(-1) = 1/2)
48:         else:
49:             temp=0
50:         arr.append(temp) # Adds a single item to the existing list
51:     plt.xlabel('n')
52:     plt.ylabel('x(n)')
53:     plt.title('Plot of DT signal: x(n)=2^n')
54:     plt.stem(n,arr) # Stem plot plots vertical lines at each n position covered under the graph from the bottom to top and places a marker where
55:     plt.show() # To view your plot
56:
57: def step_c(): # Unit Step Continuous-Time Signal
58:     t=np.arange(-2, 10, 1000) # linspace() creates evenly spaced sequences
59:     arr=[0]*len(t) # Initialize array
60:     for sample in range(len(t)): # Returns the length of the array
61:         if t[sample]>0:
62:             temp=1
63:         else:
64:             temp=0
65:         arr.append(temp) # Adds a single item to the existing list
66:     plt.xlabel('t')
67:     plt.ylabel('x(t)')
68:     plt.title('Unit Step Continuous-Time Function')
69:     plt.step(t,arr) # Line forming a series of steps between data points
70:     plt.show() # To view your plot
71:
72: def step_d(): # Unit Step Discrete-Time Signal
73:     n=np.arange(-2, 10) # Get evenly spaced values within a given interval
74:     arr=[0]*len(n) # Initialize array
```

Unit Step Continuous-Time Function

Variables Explorer Help File Size

In [4]: runfile('E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2/Experiment 2.py', wdir='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2')

MEMO:

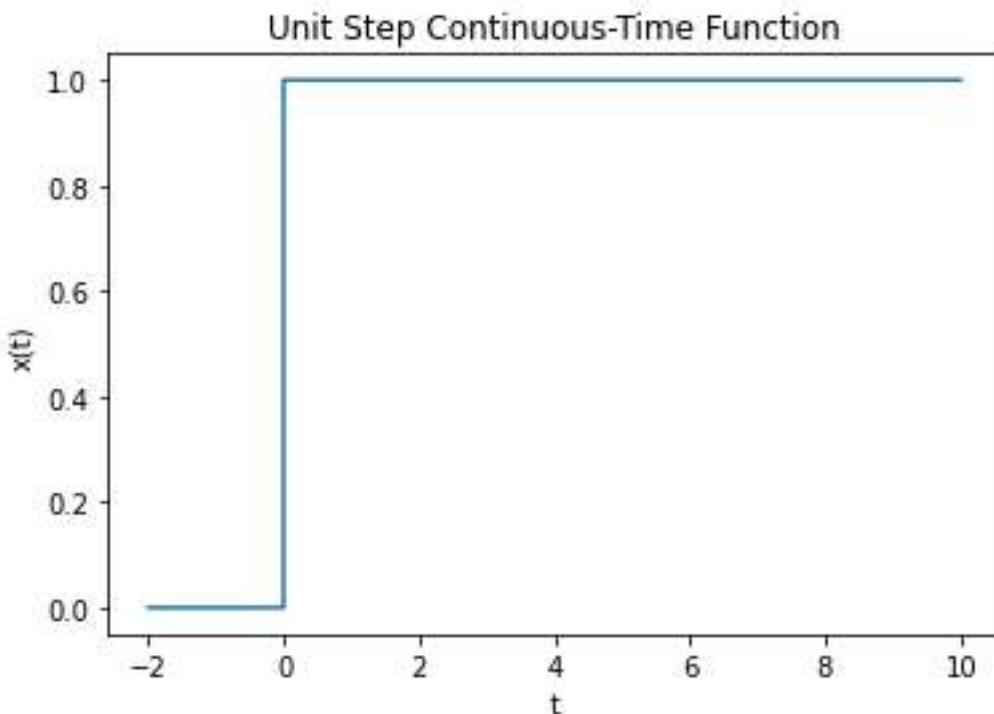
1. Sineoidal Continuous-Time Signal.
2. Sineoidal Discrete-Time Signal.
3. Exponential Continuous-Time Signal.
4. Exponential Discrete-Time Signal.
5. Unit Step Continuous-Time Signal.
6. Unit Step Discrete-Time Signal.
7. Unit Impulse Continuous-Time Signal.
8. Unit Impulse Discrete-Time Signal.
9. 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

20 / 41 Python console History

26°C 11:22 Col 14 UTF-8 CPU: RW Mem: 85%

28-09-2021



Unit Step Discrete-Time Signal

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named 'Experiment 2.py' containing Python functions for generating various signals. The right side shows a plot titled 'Unit Step Discrete-Time Function for 0 ≤ n ≤ 9'. The plot displays a discrete-time signal where the value is 0 for n < 0 and 1 for n ≥ 0, plotted against n from -2 to 9. Below the plot, a menu lists ten signal types, and the user is prompted to enter a choice.

```
def arr1(): # Null Array Declaration
    arr1 = []
    return arr1

def sample_id():
    if [sample] == 0:
        temp = 1/(2**(-n+[sample])) # Conversion of negative power to fractions (2^-1 = 1/2)
    else:
        temp = (2**(n-[sample]))
    arr.append(temp) # Adds a single item to the existing list
    print('arr:', arr)
    print('temp:', temp)
    print('n:', n)
    print('title:', 'Plot of DT signal: x(n)=2^n')
    print('show()') # Shows plot after placing vertical lines at each x position covered under the graph from the baseline to 1, and places a marker chord.
    print('show()') # To view your plot

def step_c(): # Unit Step Continuous-Time Signal
    temp = np.arange(-2, 10, 1) # np.arange() creates evenly spaced sequences
    arr1 = arr1()
    for sample in range(len(arr1)):
        if [sample] == 0:
            temp[0] = 1
        else:
            temp[0] = 0
        arr.append(temp) # Adds a single item to the existing list
    print('arr:', arr)
    print('temp:', temp)
    print('n:', n)
    print('title:', 'Unit Step Continuous-Time Function')
    print('show()') # Line forming a series of steps between data points
    print('show()') # To view your plot

def step_d(): # Unit Step Discrete-Time Signal
    temp = np.arange(-2, 10) # Get evenly spaced values within a given interval
    arr1 = arr1()
    for sample in range(len(arr1)):
        if [sample] == 0:
            temp[0] = 1
        else:
            temp[0] = 0
        arr.append(temp) # Adds a single item to the existing list
    print('arr:', arr)
    print('temp:', temp)
    print('n:', n)
    print('title:', 'Unit Step Discrete-Time Function for 0 ≤ n ≤ 9')
    print('show()') # Shows plot after placing vertical lines at each x position covered under the graph from the baseline to 1, and places a marker chord.
    print('show()') # To view your plot

def impulse_d(): # Unit Impulse Continuous-Time Signal
    temp = np.arange(-2, 10) # Not evenly spaced values within a given interval
```

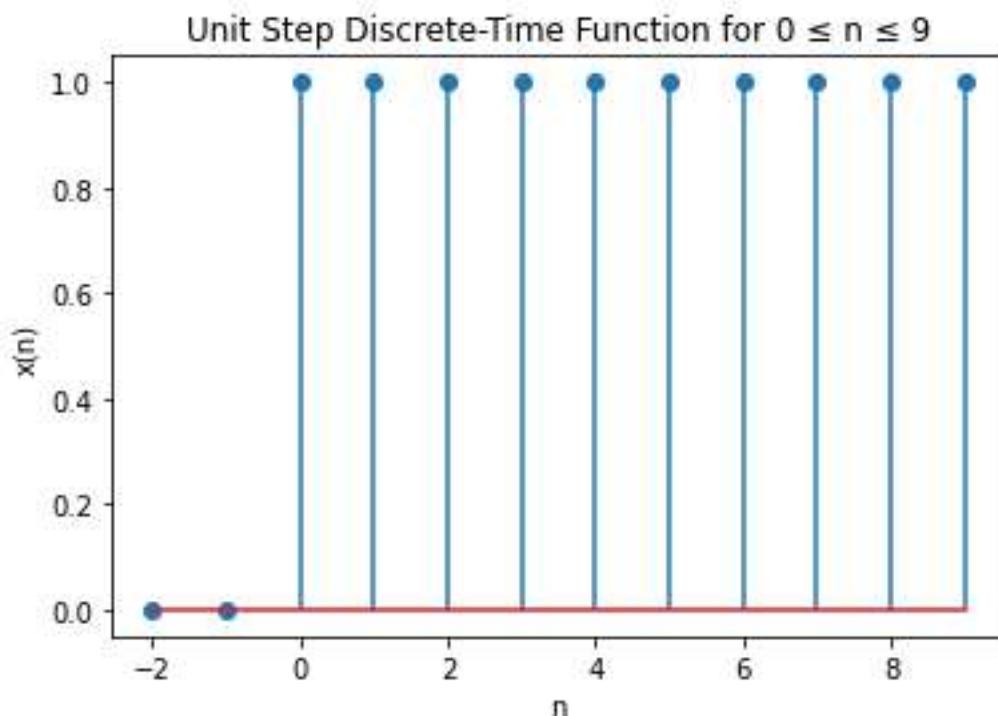
In [6]: runfile('E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2/Experiment 2.py', wdir='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2')

MEMO:

1. Sineoidal Continuous-Time Signal.
2. Sineoidal 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

In [7]:



Unit Impulse Continuous-Time Signal

The screenshot shows the Spyder Python IDE interface. On the left is the code editor with the file 'Experiment 2.py' open. The code defines several functions: `unit_step_d()`, `step_d()`, `impulse_c()`, and `impulse_d()`. The `impulse_c()` function is highlighted. On the right is a plot window titled 'Unit Impulse Continuous-Time Function' showing a single sharp peak at $t=0$ with a value of 1.0. Below the plot is a menu with numbered options from 1 to 11. The bottom status bar shows the date and time.

```
def unit_step_d():
    t = np.linspace(0, 10, 1000) # linspace() creates evenly spaced sequences
    arr = [] # Null Array Declaration
    for sample in range(len(t)): # Returns the length of the 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.title('Unit Step Continuous-Time Function')
    plt.plot(arr) # Line forming a series of steps between data points
    plt.show() # To view your plot

def step_d():
    t = np.linspace(0, 10, 10) # Get evenly spaced values within a given interval
    arr = [] # Null Array Declaration
    for sample in range(len(t)): # Returns the length of the 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.title('Unit Step Discrete-Time Function for 0 < n < 10')
    plt.stem(arr) # Stem plot places vertical lines at each position covered under the graph from the baseline to y, and places a marker where.
    plt.show() # To view your plot

def impulse_c():
    t = np.linspace(-2, 10, 1000) # Get evenly spaced values within a given interval
    arr = [] # Null Array Declaration
    for sample in range(len(t)): # Returns the length of the 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.title('Unit Impulse Continuous-Time Function')
    plt.plot(arr) # Line forming a series of steps between data points
    plt.show() # To view your plot

def impulse_d():
    t = np.linspace(0, 10, 1000) # Get evenly spaced values within a given interval
    arr = [] # Null Array Declaration
    for sample in range(len(t)): # Returns the length of the 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.title('Unit Impulse Discrete-Time Signal')
    plt.stem(arr) # Stem plot places vertical lines at each position covered under the graph from the baseline to y, and places a marker where.
    plt.show() # To view your plot
```

Unit Impulse Continuous-Time Function

Variables explorer Help Index File

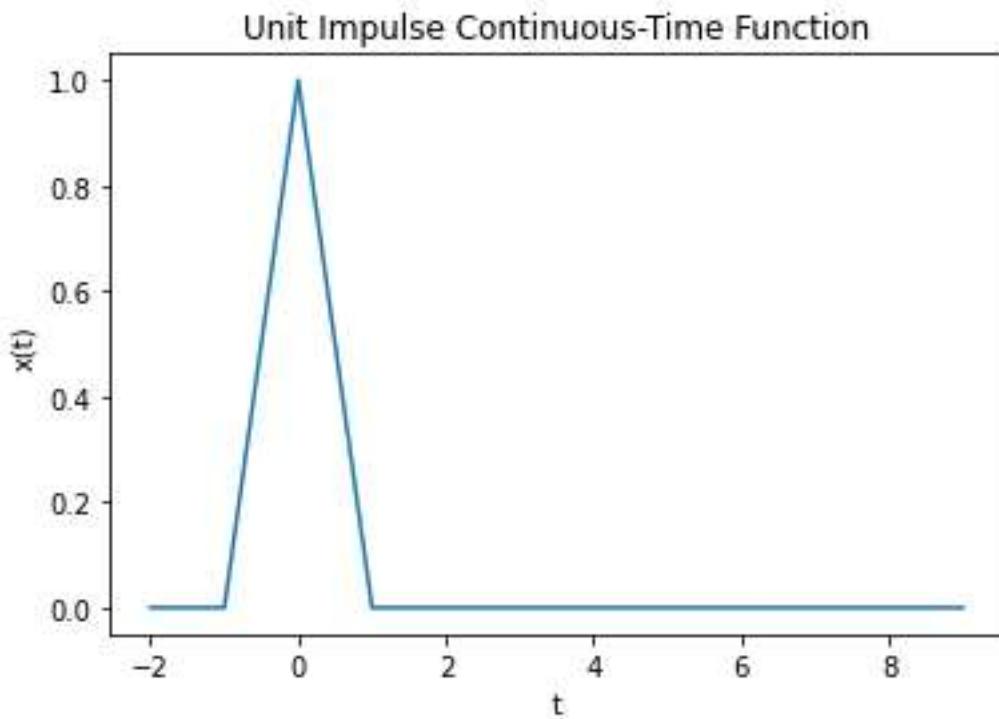
In [7]: runfile('E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2/Experiment 2.py', wdir='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2')

MEMO:

1. Sineoidal Continuous-Time Signal.
2. Sineoidal 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

26/9/2021 26°C 100% 08:55



Unit Impulse Discrete-Time Signal

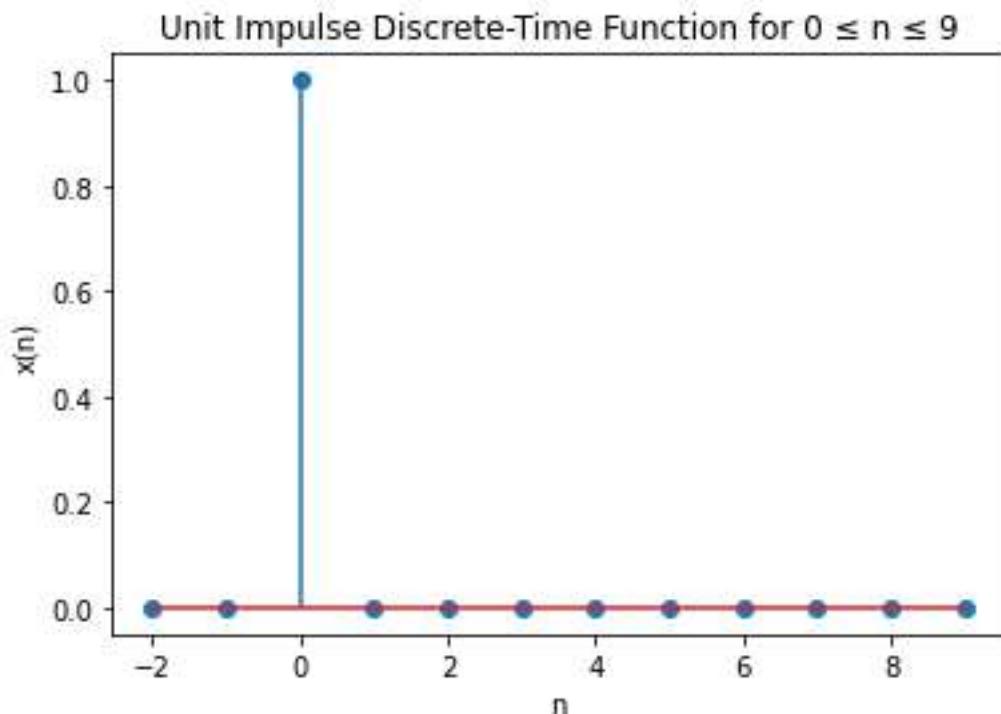
The screenshot shows the Spyder Python IDE interface. The code editor window displays a script named 'Experiment 2.ipynb' containing Python code for generating discrete-time signals. The code includes functions for unit impulse, unit step, ramp, and sinc signals. The main part of the code plots a unit impulse signal for $n \in [-2, 9]$. The plot is titled 'Unit Impulse Discrete-Time Function for $0 \leq n \leq 9$ ' and shows a single vertical blue line at $n=0$ with a value of 1.0, while all other points are at 0.0. The x-axis is labeled 'n' and ranges from -2 to 9. The y-axis is labeled 'x(n)' and ranges from 0.0 to 1.0. The plot window also contains three smaller subplots showing different signal types.

```
if n[sample]==0:
    temp1
else:
    temp0
err.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,are) # Stem plot plots vertical lines at each position covered under the graph from the baseline to y_n and places a marker there.
plt.show() # To view your plot

def impulse_c(): # Unit Impulse Continuous-Time Signal
temp=arange(-2, 10) # Get evenly spaced values within a given interval
arr1[] # Null Array Declaration
for sample in range(len(c)): # Returns the length of the array
    if [sample]!=[0]:
        temp1
    else:
        temp0
    err.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,are) # Line forming a series of steps between data points
plt.show() # To view your plot

def impulse_d(): # Unit Impulse Discrete-Time Signal
temp=arange(-2, 10) # Get evenly spaced values within a given interval
arr1[] # Null Array Declaration
for sample in range(len(c)): # Returns the length of the array
    if [sample]!=[0]:
        temp1
    else:
        temp0
    err.append(temp) # Adds a single item to the existing list
plt.xlabel('n')
plt.ylabel('x(n)')
plt.title('Unit Impulse Discrete-Time Function for 0 ≤ n ≤ 9')
plt.stem(n,are) # Stem plot plots vertical lines at each position covered under the graph from the baseline to y_n and places a marker there.
plt.show() # To view your plot

def ramp_c(): # Ramp Function Continuous-Time Signal
temp=linspace(0, 10, 1000) # linspace() creates evenly spaced sequences
plt.plot(temp)
plt.show()
```



Ramp Function Continuous-Time Signal

Spawner (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

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

Experiment 2.ipynb

```
the baseline to  $r_0$  and places a marker there.
plt.show() # To view your plot

def impulse_c(): # Unit Impulse Continuous-Time Signal
    t=np.arange(0, 10) # Get evenly spaced values within a given interval
    arr=[0] # Array declaration
    for sample in range(len(t)): # Returns the length of the 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.title('Unit Impulse Continuous-Time Function')
    plt.plot(t,arr) # Line forcing a series of steps between data points
    plt.show() # To view your plot

def impulse_d(): # Unit Impulse Discrete-Time Signal
    n=np.arange(0, 10) # Get evenly spaced values within a given interval
    arr=[0] # Array declaration
    for sample in range(n): # returns the length of the 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.title('Unit Impulse Discrete-Time Function for  $\theta = n \neq 0$ ')
    plt.stem(n,arr) # stem plot plots vertical lines at each position covered under the graph from the baseline to  $r_0$  and places a marker there.
    plt.show() # To view your plot

def ramp_c(): # Ramp Function Continuous-Time Signal
    t=np.arange(0, 10, 0.001) # linspace() creates evenly spaced sequences
    arr=[0] # Array declaration
    for sample in range(len(t)):
        arr.append(t[sample])
    plt.xlabel('t')
    plt.title('Ramp Continuous-Time Function')
    plt.plot(t,arr) # Set the xlimits 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
    arr=[0] # Array declaration
    plt.xlabel('n')
```

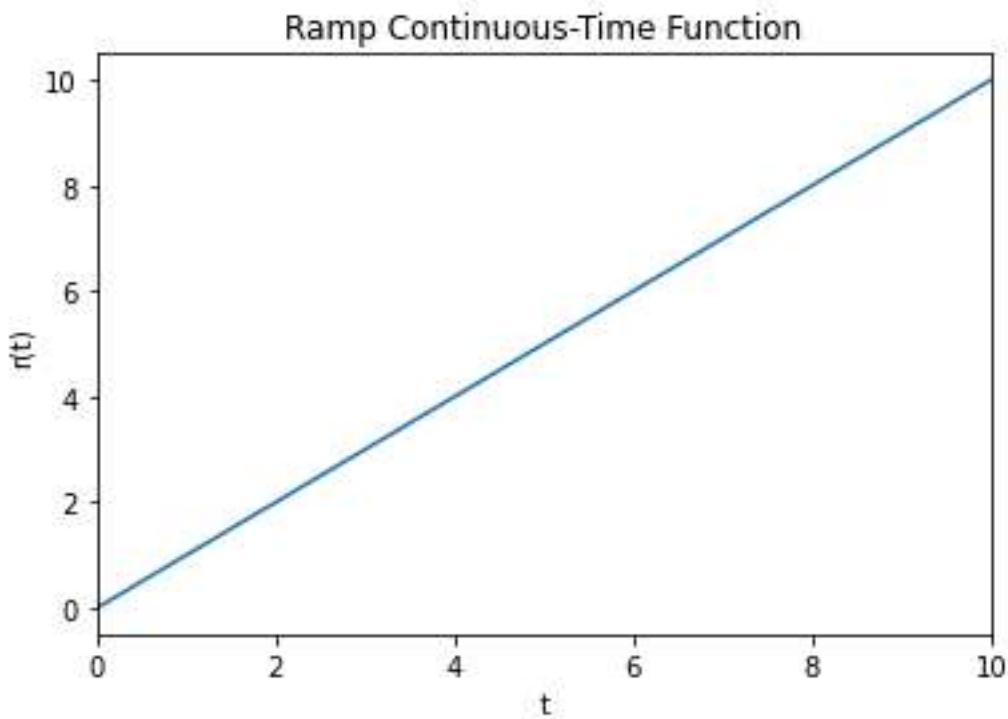
In [1]: runfile('E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2/Experiment 2.py', wdir='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 2')

MEMO:

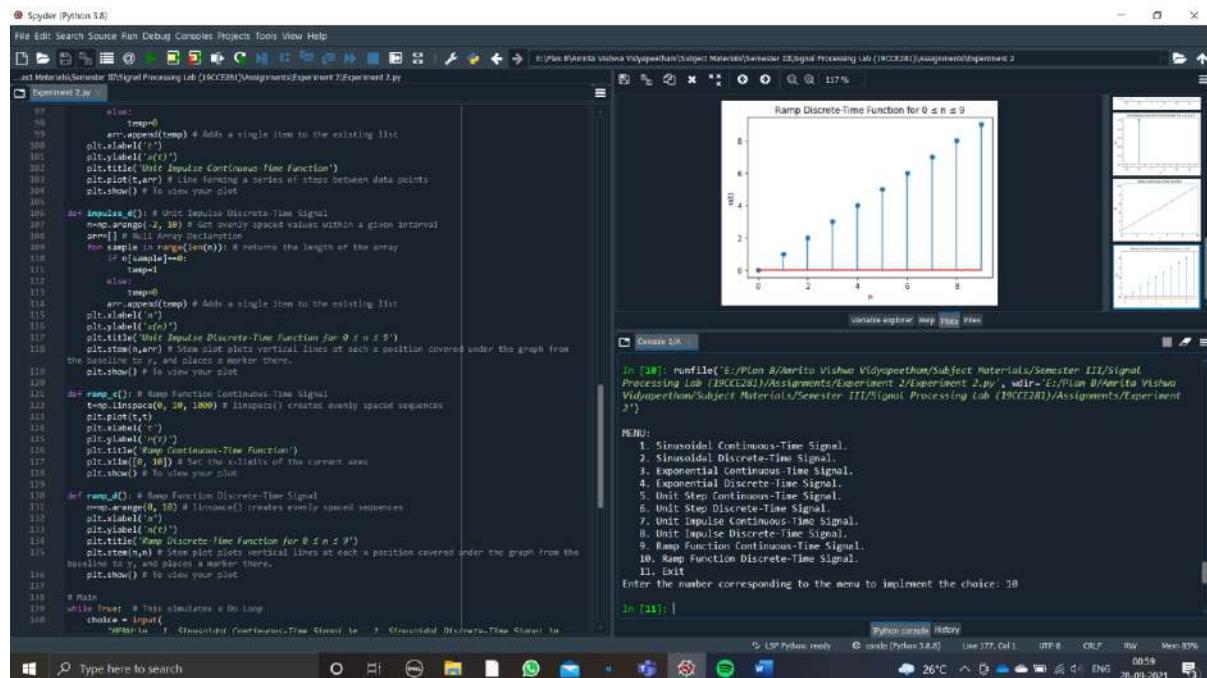
1. Sineoidal Continuous-Time Signal.
2. Sineoidal Discrete-Time Signal.
3. Exponential Continuous-Time Signal.
4. Exponential Discrete-Time Signal.
5. Unit Step Continuous-Time Signal.
6. Unit Step Discrete-Time Signal.
7. Unit Impulse Continuous-Time Signal.
8. Unit Impulse Discrete-Time Signal.
9.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

Enter the number corresponding to the menu to implement the choice: 9

26°C 28-09-2021 08:58 Python console History



Ramp Function Discrete-Time Signal



The screenshot shows the Spyder Python IDE interface. The code in the editor window is as follows:

```

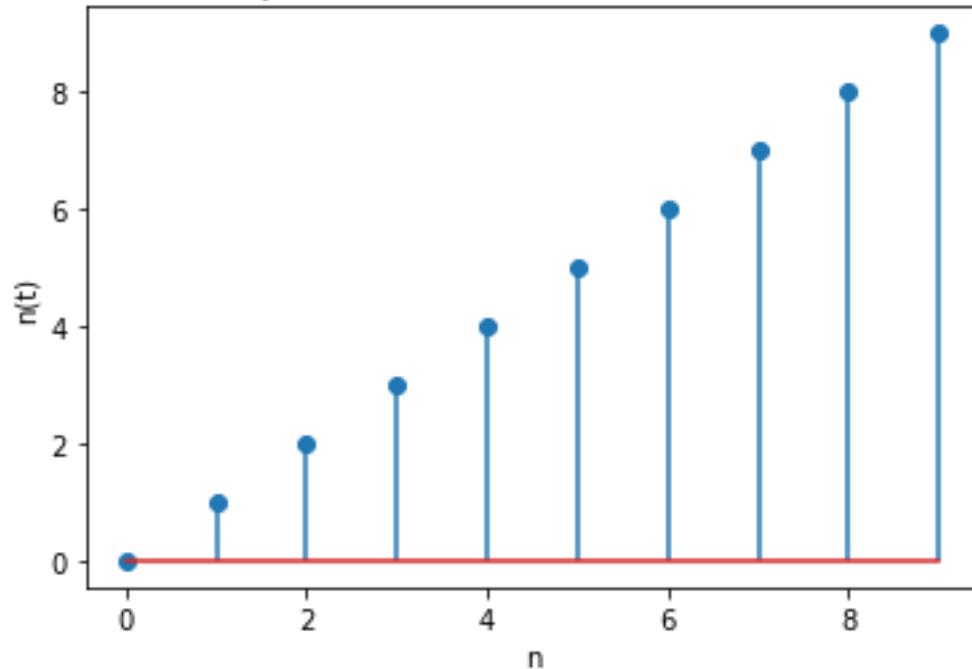
# Experiment 2.py
# File Edit Search Source Run Debug Projects Tools View Help
# Experiment 2.py
# 1. Sineoidal Continuous-Time Signal
# 2. Sineoidal Discrete-Time Signal
# 3. Exponential Continuous-Time Signal
# 4. Exponential Discrete-Time Signal
# 5. Unit Step Continuous-Time Signal
# 6. Unit Step 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
# Main
# while True: # This simulates a Do Loop
#     choice = input('
#         1. Sineoidal Continuous-Time Signal
#         2. Sineoidal 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
#     ')
#     if choice == '10':
#         break
#     else:
#         print('Invalid choice. Please enter a valid choice from 1 to 11.')
#     print('')

# 10. Ramp Function Discrete-Time Signal
# n=np.arange(0,10) # linspace() creates evenly spaced sequences
# n=[int(i) for i in n] # Convert float array to integer array
# plt.stem(n,ramp_d) # 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 your plot

# 11. Exit
# 
```

The plot area displays a discrete-time ramp function for $0 \leq n \leq 9$. The x-axis is labeled n and ranges from 0 to 9. The y-axis is labeled $r_n(n)$ and ranges from 0 to 9. The plot shows discrete points connected by vertical lines, forming a staircase-like ramp.

Ramp Discrete-Time Function for $0 \leq n \leq 9$



Exit

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays a script named 'Experiment 2.py' which contains a menu loop for generating discrete-time signals. The code includes a series of if-else statements for each signal type, with a final catch-all case for invalid input. The code is annotated with comments explaining the menu items. To the right of the code editor, there are three plots: a stem plot of a discrete-time ramp function, a line plot of a continuous-time ramp function, and a step plot of a discrete-time unit step function. The bottom status bar shows system information like temperature (26°C), date (28-09-2021), and battery level (84%).

```
137 # Main
138 while True: # This simulates a Do Loop
139     choice = input()
140     # MENU (a - 1. Sinusoidal Continuous-Time Signal, n - 2. Sinusoidal Discrete-Time Signal, In - 3. Unit Step Continuous-Time Signal, n - 4. Exponential Continuous-Time Signal, n - 5. Unit Step Discrete-Time Signal, n - 6. Unit Impulse Continuous-Time Signal, n - 7. Unit Impulse Discrete-Time Signal, n - 8. Ramp Function Continuous-Time Signal, n - 9. Ramp Function Discrete-Time Signal, n - 10. Ramp Function Discrete-Time Signal, n - 11. Exit) Enter the number corresponding to the menu to implement the choice: "choice"
141
142     if choice == str(1): # returns the string version of the variable "choice"
143         sinusoidal_ct() # Sinusoidal Continuous-Time Signal
144         break
145     elif choice == str(2):
146         sinusoidal_dt() # Sinusoidal Discrete-Time Signal
147         break
148     elif choice == str(3):
149         unit_step_ct() # Unit Step Continuous-Time Signal
150         break
151     elif choice == str(4):
152         exponential_ct() # Exponential Continuous-Time Signal
153         break
154     elif choice == str(5):
155         exponential_dt() # Exponential Discrete-Time Signal
156         break
157     elif choice == str(6):
158         unit_step_ct() # Unit Step Continuous-Time Signal
159         break
160     elif choice == str(7):
161         unit_impulse_ct() # Unit Impulse Continuous-Time Signal
162         break
163     elif choice == str(8):
164         unit_impulse_dt() # Unit Impulse Discrete-Time Signal
165         break
166     elif choice == str(9):
167         ramp_ct() # Ramp Function Continuous-Time Signal
168         break
169     elif choice == str(10):
170         ramp_dt() # Ramp Function Discrete-Time Signal
171         break
172     elif choice == str(11):
173         print("Enter the number corresponding to the menu to implement the choice: ")
174     else:
175         print("Error: Invalid Input! Please try again.\n")
```

This screenshot is identical to the one above, showing the same code in 'Experiment 2.py', the same three plots, and the same system status at the bottom. The only difference is the text in the command line window, which now shows 'In [12]:' indicating the next input prompt.

```
137 # Main
138 while True: # This simulates a Do Loop
139     choice = input()
140     # MENU (a - 1. Sinusoidal Continuous-Time Signal, n - 2. Sinusoidal Discrete-Time Signal, In - 3. Unit Step Continuous-Time Signal, n - 4. Exponential Continuous-Time Signal, n - 5. Unit Step Discrete-Time Signal, n - 6. Unit Impulse Continuous-Time Signal, n - 7. Unit Impulse Discrete-Time Signal, n - 8. Ramp Function Continuous-Time Signal, n - 9. Ramp Function Discrete-Time Signal, n - 10. Ramp Function Discrete-Time Signal, n - 11. Exit) Enter the number corresponding to the menu to implement the choice: "choice"
141
142     if choice == str(1): # returns the string version of the variable "choice"
143         sinusoidal_ct() # Sinusoidal Continuous-Time Signal
144         break
145     elif choice == str(2):
146         sinusoidal_dt() # Sinusoidal Discrete-Time Signal
147         break
148     elif choice == str(3):
149         unit_step_ct() # Unit Step Continuous-Time Signal
150         break
151     elif choice == str(4):
152         exponential_ct() # Exponential Continuous-Time Signal
153         break
154     elif choice == str(5):
155         exponential_dt() # Exponential Discrete-Time Signal
156         break
157     elif choice == str(6):
158         unit_step_ct() # Unit Step Continuous-Time Signal
159         break
160     elif choice == str(7):
161         unit_impulse_ct() # Unit Impulse Continuous-Time Signal
162         break
163     elif choice == str(8):
164         unit_impulse_dt() # Unit Impulse Discrete-Time Signal
165         break
166     elif choice == str(9):
167         ramp_ct() # Ramp Function Continuous-Time Signal
168         break
169     elif choice == str(10):
170         ramp_dt() # Ramp Function Discrete-Time Signal
171         break
172     elif choice == str(11):
173         print("Enter the number corresponding to the menu to implement the choice: ")
174     else:
175         print("Error: Invalid Input! Please try again.\n")
```

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:

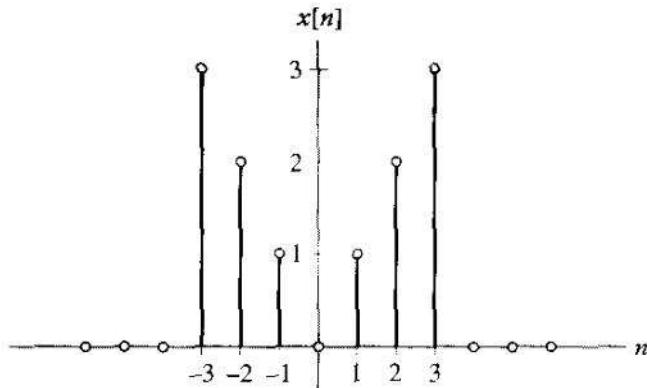
1. Sinusoidal Continuous-Time Signal.
2. Sinusoidal Discrete-Time Signal.
3. Exponential Continuous-Time Signal.
4. Exponential Discrete-Time Signal.
5. Unit Step Continuous-Time Signal.
6. Unit Step Discrete-Time Signal.
7. Unit Impulse Continuous-Time Signal.
8. Unit Impulse Discrete-Time Signal.
9. 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

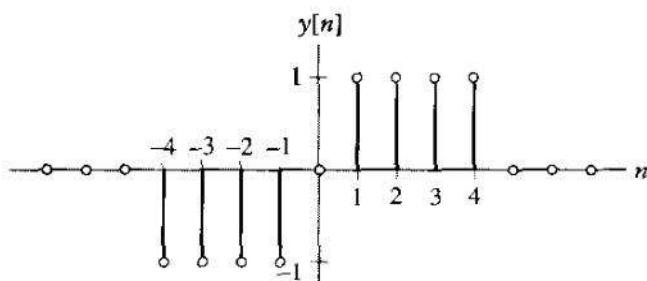
In [2]:

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.EN.VHCE20053)

① DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING

LAB TITLE AND CODE: SIGNAL PROCESSING LAB MCL6281

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 -

$$y(t) = c x(t)$$

where $x(t)$ denotes continuous time signal,

$y(t)$ denotes resulting signal,

and c denotes the scaling factor.

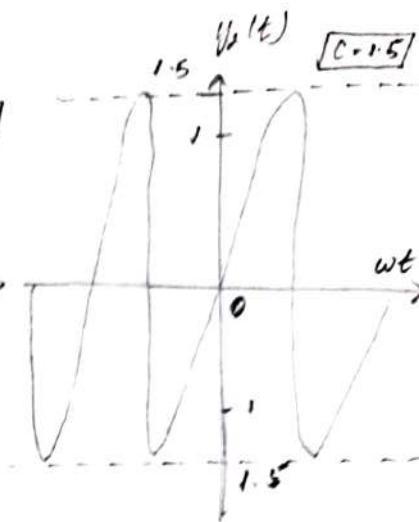
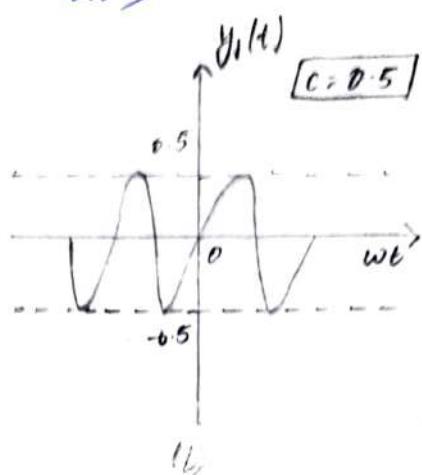
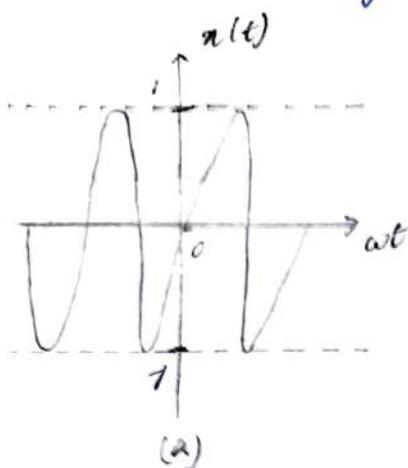
If $c < 1 \rightarrow$ signal is attenuated.

$c > 1 \rightarrow$ signal is amplified.

⑨

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.

⑩ Addition of signals-

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

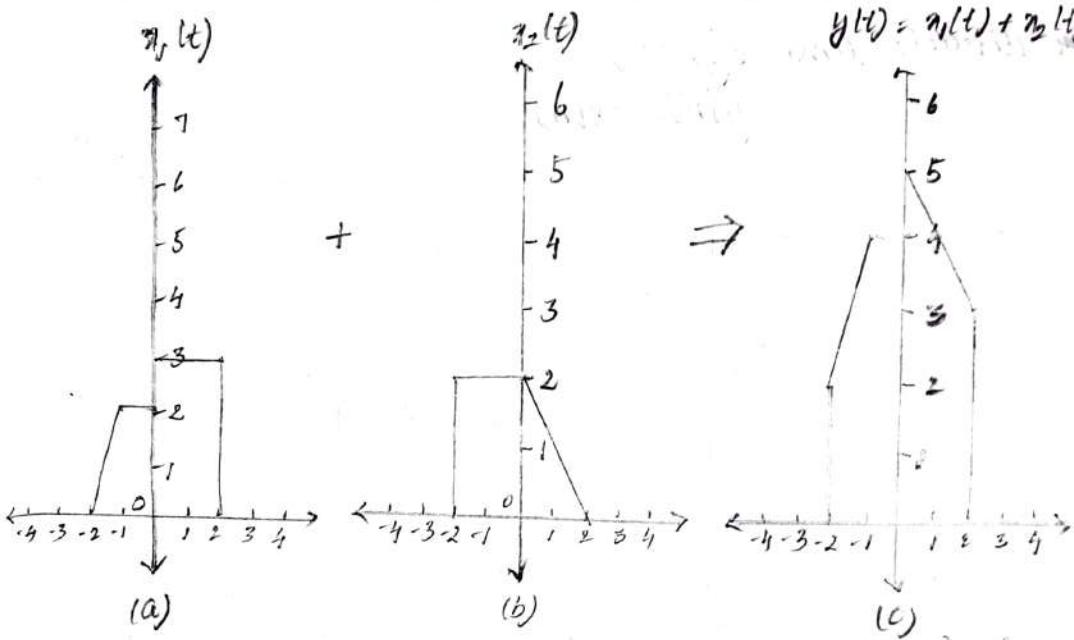
$$y(t) = x_1(t) + x_2(t)$$

For discrete time signals,

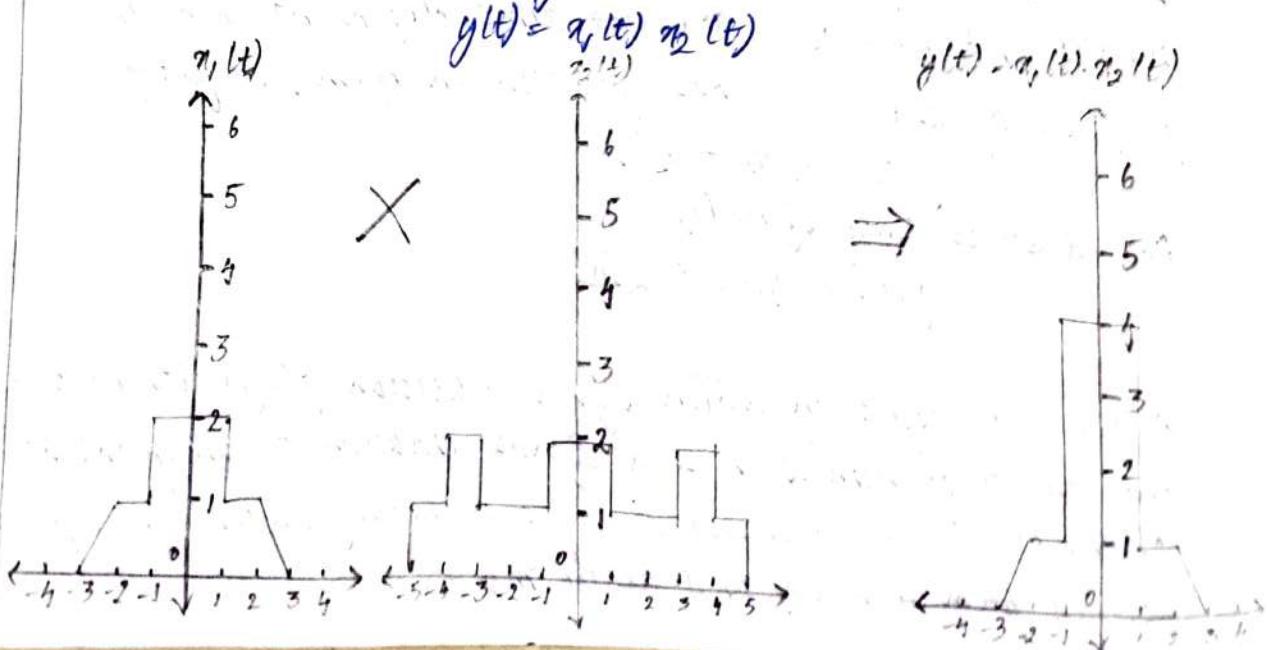
$$y[n] = x_1[n] + x_2[n]$$

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

3



③ Multiplication of Signals -
 Like addition, multiplication of signals also fall 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 $r_1(t)$ and $r_2(t)$ are two time dependent signals, on whom after performing the multiplication operation we get -

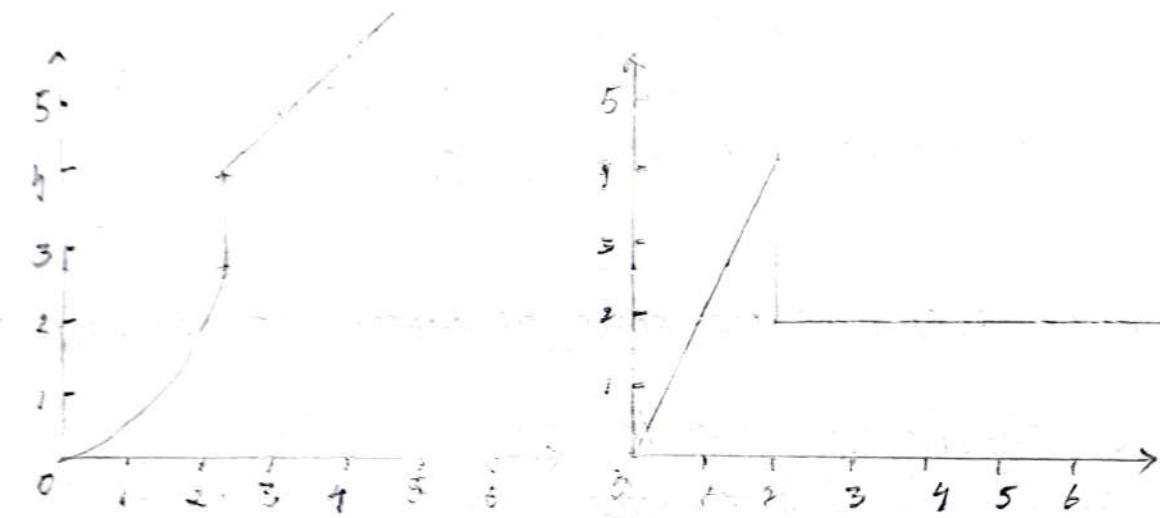


④

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 -

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



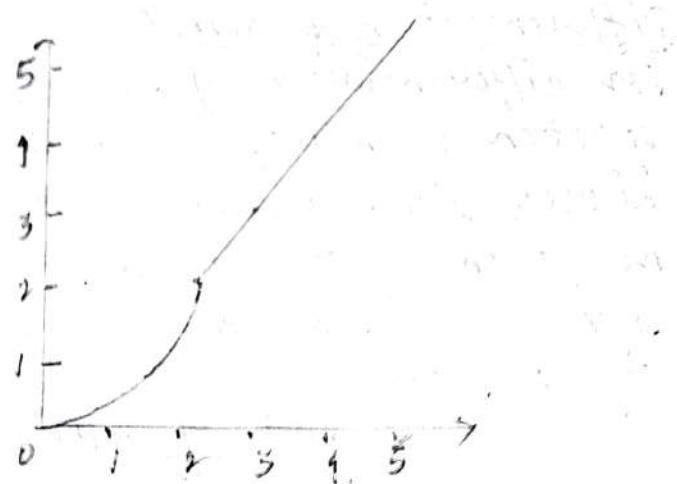
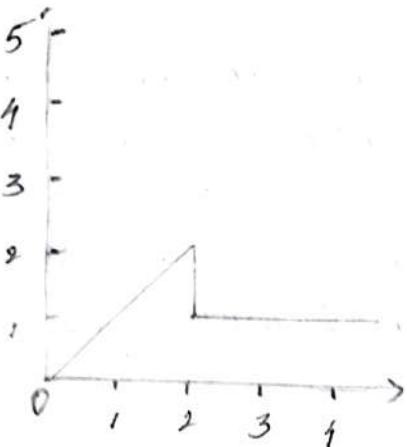
Differentiation of a signal takes the form of the gradient operator in the field of image or video processing. In the case of image processing, the gradient technique is a popular method which is used to detect the edges in the given image. With video processing, this operator is used for motion detection.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

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 -

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

5



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.

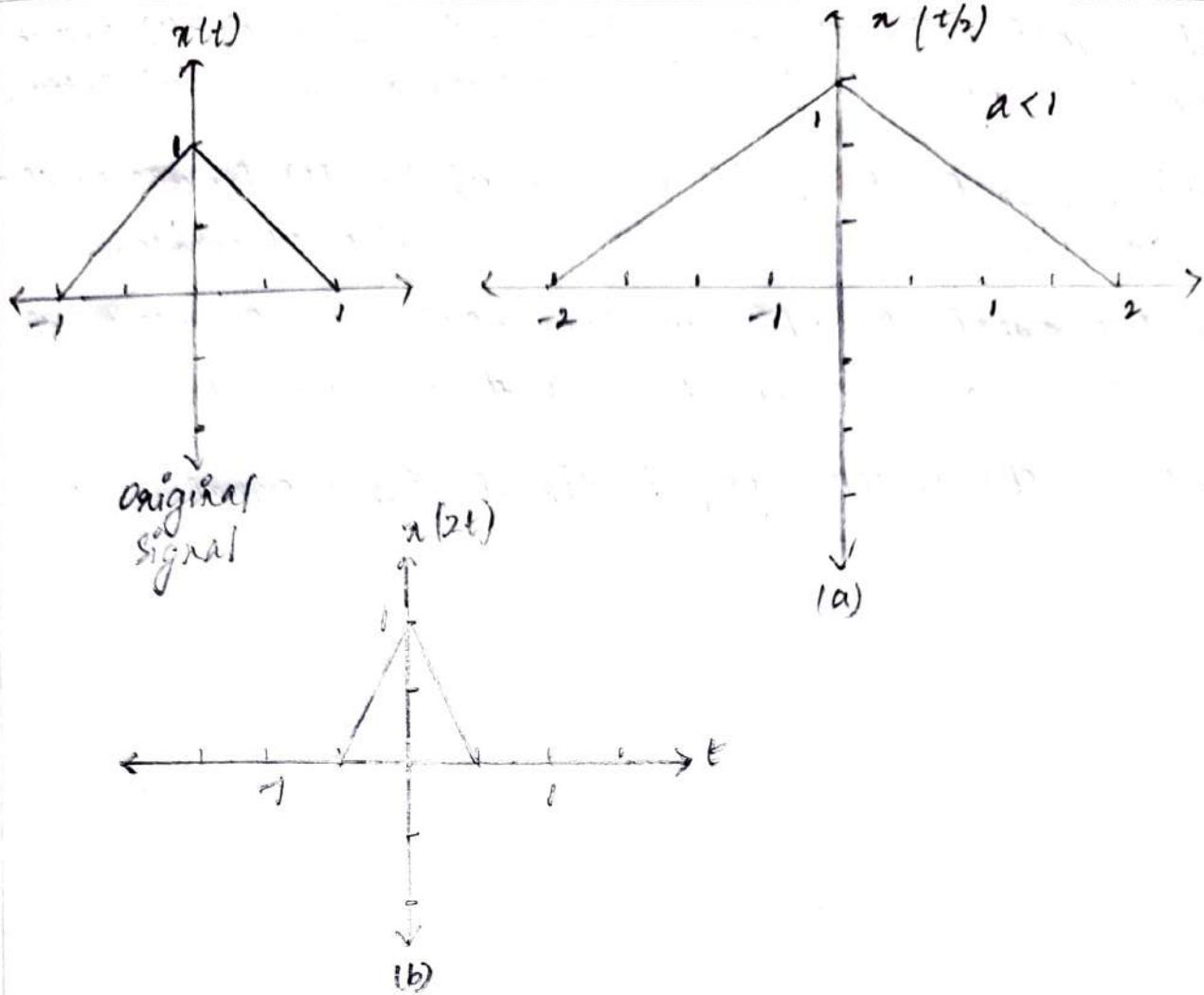
① Time Scaling of Signals -

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

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

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

$\xrightarrow{\text{PSD}}$



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

② Reflection of signals -

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

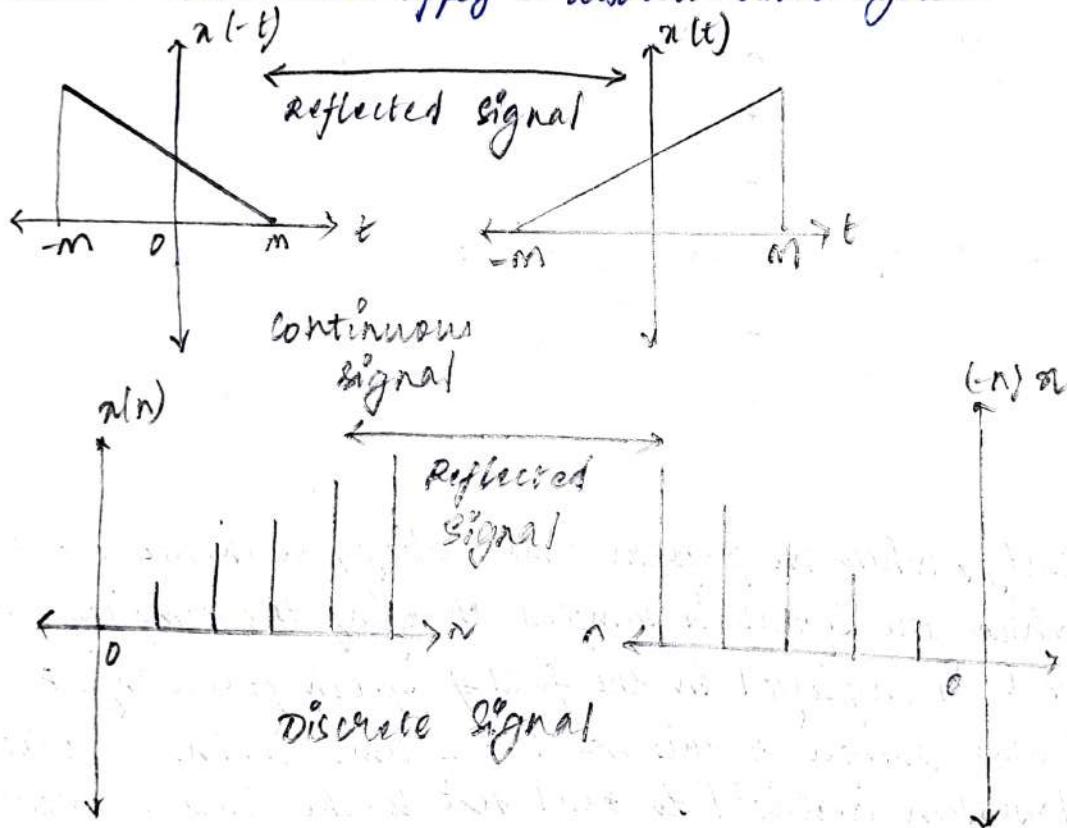
Let $n(t)$ denote continuous time signal. Let $y(t)$ denote the signal obtained by replacing time t with $-t$ as shown by -

$$y(t) = n(-t)$$

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

- 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.
- 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.

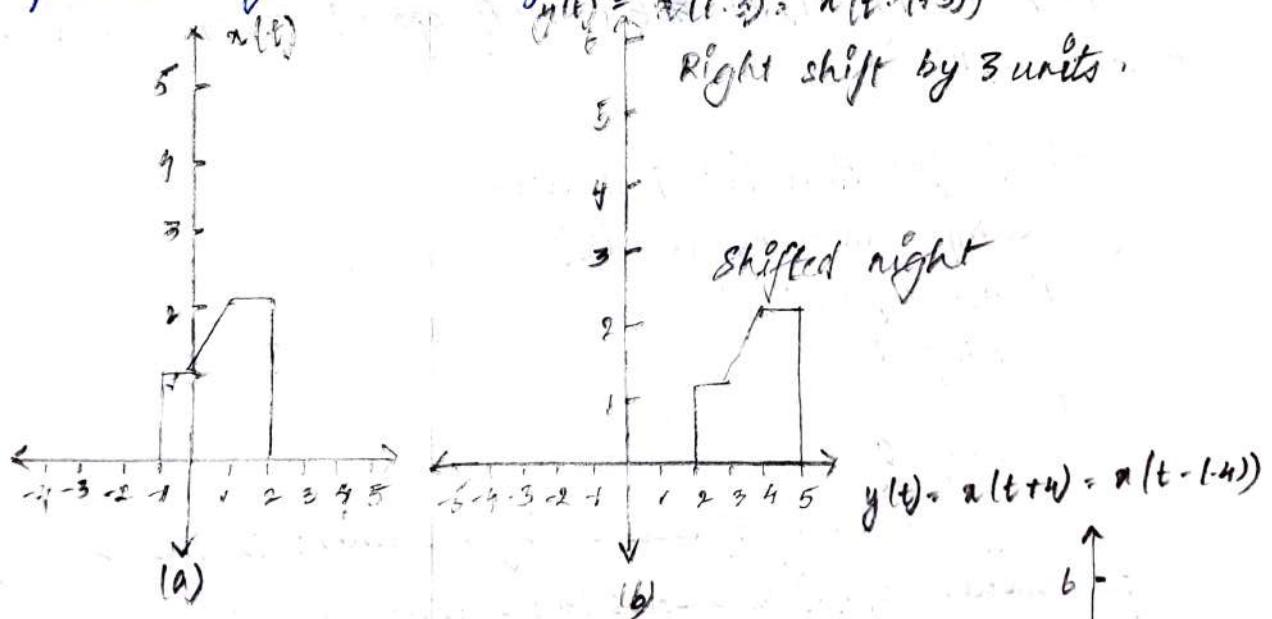
(8) Time Shifting of Signals -

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

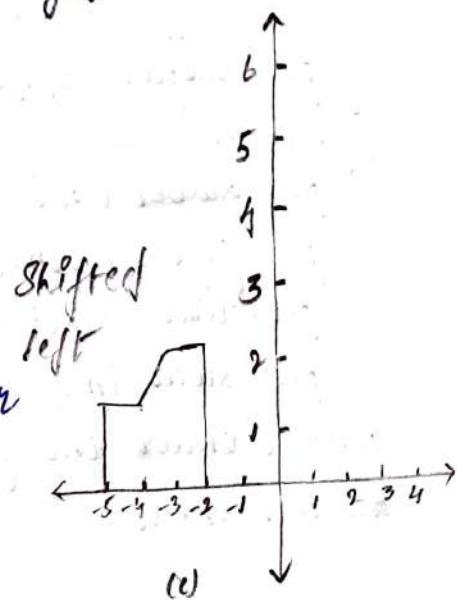
$$y(t) = a(t-t_0)$$

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

For a signal $a(t)$, if the position shift to >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 ~~are~~ performed autocorrelation. Other examples include artificial intelligence, such as in systems that use Time Delay Neural Networks.



(9)

* 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()` function.
- 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)
```

```
    u[n] # 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.
```

⑩

$b = [] \# u[n-2]$ Null list declaration
for sample in range (len(n)):

If $n[\text{sample}] >= 2$:

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

11

```
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_t_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) # sends the value of list back and exits the  
    function.
```

```
def y_t_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.
```

(12)

else :

list.append(0) # Adds a single item to the existing list.

else :

list.append(0) # Adds a single item to the existing list.

return(list) # send the value of list back and exits the function

def sketch_i(): # sketch $x[3n-1]$

n = np.arange(-6, 1) # get evenly spaced values within the interval [-6, 1]

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() 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/N

(13)

```
def sketch_2() # sketch  $x[n-2] + y[n+2]$ 
    n = np.arange(-6, 7) # get evenly spaced values within the
    intervals [-6, 7]
```

$\text{list_x}[]$ # Null List Declaration
for sample in n :

$\text{list_x.append(sample_r)}$ # adds a single item to the
existing $\text{list_x}[n-2]$

$\text{plt.subplot}(5, 1, 1)$ # subplot (rows, columns, index) describes
the figure layout

plt.xlabel('n')

$\text{plt.ylabel('x[n-2]')}$

$\text{plt.stem}(n, x_of_n)$ # calls the function $x_of_n(\text{list_x})$ and
displays $x[n-2]$ plot.

$\text{list_y}[]$ # Null List Declaration
for sample in n :

$\text{list_y.append(sample_r)}$ # adds a single item to the
existing $\text{list_y}[n+2]$

$\text{plt.subplot}(5, 1, 3)$ # subplot (rows, columns, index) describes
the figure layout

plt.xlabel('n')

$\text{plt.ylabel('y[n+2]')}$

$\text{plt.title('Function for y[n+2]')}$

$\text{plt.stem}(n, y_of_n(\text{list_y}))$ # calls the function $y_of_n(\text{list_y})$
and displays $y[n+2]$ plot.

$\text{result} = \text{np.add}(x_of_n(\text{list_x}), y_of_n(\text{list_y}))$ # display the
resultant graph, $x[n-2] + y[n+2]$

$\text{plt.subplot}(5, 1, 5)$ # subplot (rows, columns, index)
describes the figure layout

(14)

```

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": In 1. Step signal Subtraction \n 2. Plot Impulse
 signal. \n 3. sketch x[3n-1]. \n 4. sketch x[n-2] + y[n+3]. \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+3]
 break

elif choice == str(5):

break # Exit loop

else:

print ("Error & Invalid Input! Please try again.\n")

* INFERENC E:

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

Step Signal Subtraction

Spyder (Python 3.8)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
Experiment 3.py
1 #-*- coding: utf-8 -*-
2 """
3 Created on Tue Sep 28 13:58:38 2021
4
5 Author: Santosh (dell)
6
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10
11 def step_sub(): # Step Signal Subtraction
12     n = np.arange(0,10) # Get evenly spaced values within the interval [0,10]
13
14     a=[1] # u[n] Null List Declaration
15     for sample in range(len(n)):
16         if n[sample]>=0:
17             temp1
18         else:
19             temp0
20             a.append(temp0) # Adds a single item to the existing list
21     plt.subplot(3,1,1) # subplot(rows, columns, index) describes the figure layout
22     plt.xlabel('n')
23     plt.ylabel('u[n]')
24     plt.title('Unit Step Discrete-Time Function for u[n]')
25     plt.stem(n,a) # Stem plot plots vertical lines at each x position covered under the
graph from the baseline to y, and places a marker there.
26
27     b=[1] # u[n-2] Null List Declaration
28     for sample in range(len(n)):
29         if n[sample]>=2:
30             temp1
31         else:
32             temp0
33             b.append(temp0) # Adds a single item to the existing list
34     plt.subplot(3,1,2) # subplot(rows, columns, index) describes the figure layout
35     plt.xlabel('n')
36     plt.ylabel('u[n-2]')
37     plt.title('Unit Step Discrete-Time Function for u[n-2]')
38     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.
39
40     result=[1] # u[n] - u[n-2] Null List Declaration
41
42     for sample in range(len(n)):
43         if n[sample]>=0:
44             result.append(1)
45         else:
46             result.append(0)
47
48     plt.subplot(3,1,3) # subplot(rows, columns, index) describes the figure layout
49     plt.xlabel('n')
50     plt.ylabel('u[n] - u[n-2]')
51     plt.title('Unit Step Discrete-Time Function for u[n] - u[n-2]')
52     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.
53
54
55 result # u[n] - u[n-2] Null List Declaration

```

Unit Step Discrete-Time Function for $u[n]$

Unit Step Discrete-Time Function for $u[n-2]$

Unit Step Discrete-Time Function for $u[n] - u[n-2]$

Variable explorer Help Index Files

Console 2/4

Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1936 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

iPython 7.22.0 -- An enhanced Interactive Python.

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

MENU:

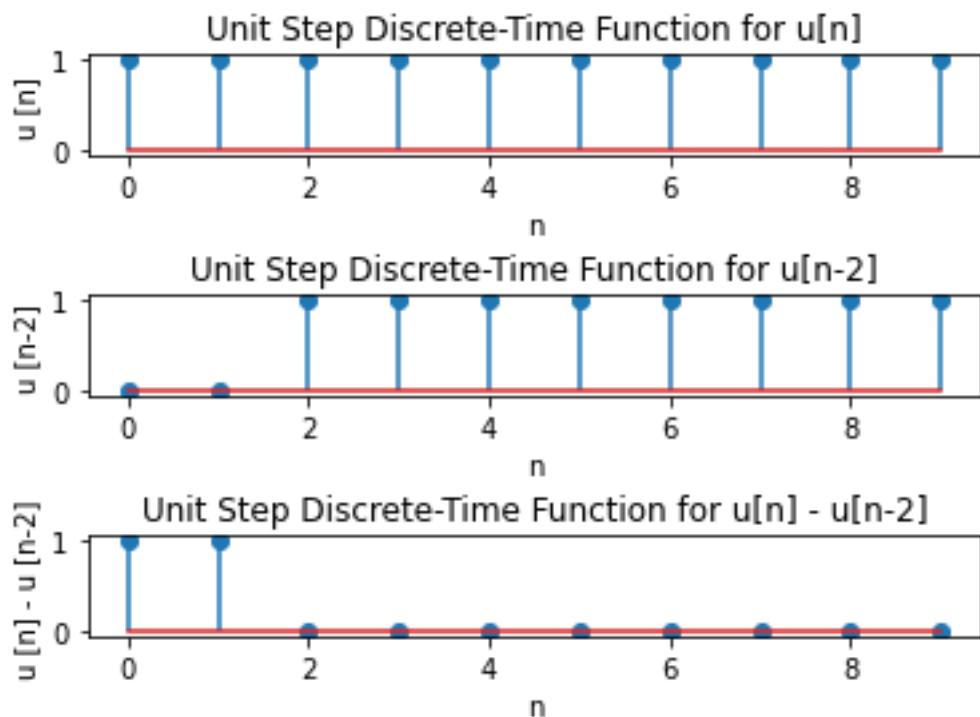
- 1. Step Signal Subtraction.
- 2. Plot Impulse Signal.
- 3. Sketch $x[n-1]$.
- 4. Sketch $x[n-2] + y[n+2]$.
- 5. Exit

Enter the number corresponding to the menu to implement the choice: 1

Plots now render in the Plots pane by default. To make them also appear inline in the Console.

Python console history

QUIT PYTHON (F5) File (F6) code (Python 3.8) Line (F9) Cell (F10) IPython (F11) CWD (F12) RW (F13) Open (F14) 27°C Mostly cloudy ⌂ 80% 08:01 07-10-2021



Plot Impulse Signal

Spyder (Python 3.8)

File Edit Search Source Run Debug Console Projects Tools View Help

Experiment 3.ipynb

```
22 plt.xlabel('n')
23 plt.ylabel('x[n]')
24 plt.title('Unit Step Discrete-Time Function for u[n]')
25 plt.stem(n,a) # Stem plot plots vertical lines at each x position covered under the graph from the baseline to y, and places a marker there.
26
27 b[:] # u[n-2] Null List Declaration
28 for sample in range(len(b)):
29     if n[sample]>-2:
30         temp=1
31     else:
32         temp=0
33     b.append(temp) # Adds a single item to the existing list
34 plt.subplot(5,1,3) # subplot(rows, columns, index) describes the figure layout
35 plt.xlabel('n')
36 plt.ylabel('u[n-2]')
37 plt.title('Unit Step Discrete-Time Function for u[n-2]')
38 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.
39
40 result=[0]*len(n) - u[n-2] Null List Declaration
41 for sample in range(len(n)):
42     result.append(a[sample]-b[sample]) # Adds a single item to the existing list by subtracting u[n-2] from u[n] list.
43 plt.subplot(5,1,4) # subplot(rows, columns, index) describes the figure layout
44 plt.xlabel('n')
45 plt.ylabel('u[n] - u[n-2]')
46 plt.title('Unit Step Discrete-Time Function for u[n] - u[n-2]')
47 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.
48 plt.show() # To view all the three sub-plots.
49
50 def impulse_plot(): # Plot Impulse Signal
51     n=np.arange(0, 5) # Get evenly spaced values within the interval [0,5]
52     arr=[2,3,4,5,0] # Initialize array with values as given in question.
53     plt.xlabel('n')
54     plt.ylabel('x[n]')
55     plt.title('Unit Impulse Discrete-Time Function for x[n] = [2 3 4 5 0]')
56     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.
57     plt.show() # To view the plot.
58
```

Variables Explorer Help Index Files

In [2]: runfile('E:/Plan R/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE21)/Assignments/Experiment 3/Experiment 3.ipynb', wdir='E:/Plan R/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE21)/Assignments/Experiment 3')

Console

MENU:

1. Stop Signal Subtraction.
2. Plot Impulse Signal.
3. Sketch $x[3n - 1]$.
4. Sketch $x[3n - 2] + y[n + 2]$.
5. Exit.

Enter the number corresponding to the menu to implement the choice: 2

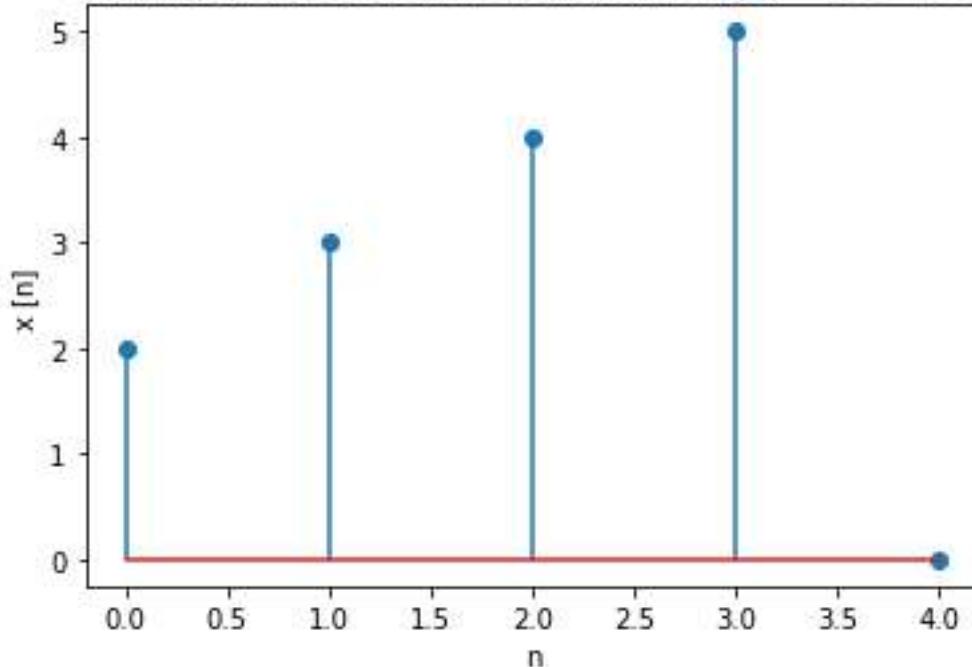
Python console History

Type here to search

27°C Mostly cloudy

07-10-2021

Unit Impulse Discrete-Time Function for $x[n] = [2 3 4 5 0]$



Generate $x[n]$ as given in the question

Spyder (Python 3.8)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
art Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py
Experiment 3.py
1 #!/usr/bin/python3
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 n = np.arange(-6, 6) # Get evenly spaced values within the interval [-6, 6]
7 arr=[2,3,4,5,6] # Initialize array with values as given in question.
8 plt.xlabel('n')
9 plt.title('Unit Impulse Discrete-Time Function for x[n] = [2 3 4 5 6]')
10 plt.stem(n,arr) # stem plot prints vertical lines at each x position covered under the
graph from the baseline to y, and places a marker there.
11 plt.show() # To view all the three subplots.

12 def impulse_plot(): # Plot Impulse Signal.
13     n=np.arange(0, 5) # Get evenly spaced values within the interval [0, 5]
14     arr=[2,3,4,5,6] # Initialize array with values as given in question.
15     plt.xlabel('n')
16     plt.title('Unit Impulse Discrete-Time Function for x[n] = [2 3 4 5 6]')
17     plt.stem(n,arr) # stem plot prints vertical lines at each x position covered under the
graph from the baseline to y, and places a marker there.
18     plt.show()

19 def x_of_n(): # Generate x[n] as given in the question
20     list=[] # Null list Declaration
21     for sample in n:
22         if sample<3:
23             if sample<2:
24                 list.append(0) # Adds a single item to the existing list.
25             else:
26                 list.append(0) # Adds a single item to the existing list.
27         else:
28             list.append(0) # Adds a single item to the existing list.
29     return (list) # Send the value of list back and exits the function.

30 def y_of_n(): # Generate y[n] as given in the question
31     list=[] # Null list Declaration
32     for sample in n:
33         if sample<0:
34             list.append(0) # Adds a single item to the existing list.
35         elif sample<1:
36             list.append(0) # Adds a single item to the existing list.
37         elif sample<2:
38             if sample>=2:
39                 list.append(sample/abs(sample)) # abs() returns the absolute value of a
number.
40             else:
41                 list.append(0) # Adds a single item to the existing list.
42         else:
43             list.append(0) # Adds a single item to the existing list.
44     return(list) # Send the value of list back and exits the function.

45 def sketch_1(): # Sketch x[3n - 1]
46     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
47     plt.subplot(3,1,1) # subplot(rows, columns, index) describes the figure layout
48     plt.xlabel('n')
49     plt.title('Function for x[n]')
50     plt.stem(n,x_of_n())
51     plt.show()

52 def sketch_2(): # Sketch x[3n - 1]
53     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
54     plt.subplot(3,1,2) # subplot(rows, columns, index) describes the figure layout
55     plt.xlabel('n')
56     plt.title('Function for x[3n - 1]')
57     plt.stem(n,y_of_n())
58     plt.show()

59 def sketch_3(): # Sketch x[n - 1]
60     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
61     plt.subplot(3,1,3) # subplot(rows, columns, index) describes the figure layout
62     plt.xlabel('n')
63     plt.title('Function for x[n - 1]')
64     plt.stem(n,y_of_n())
65     plt.show()

66 def sketch_4(): # Sketch x[n - 2] + y[n + 2].
67     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
68     plt.subplot(3,1,4) # subplot(rows, columns, index) describes the figure layout
69     plt.xlabel('n')
70     plt.title('Function for x[n - 2] + y[n + 2]')
71     plt.stem(n,y_of_n())
72     plt.show()

```

Variables explorer Help File View

Console 1/A

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

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

In [4]: runfile('E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py', wdir='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3')

Python console history

OS: USP Python ready CPU ready Code Python 3.8.0 Line 156, Col 1 UTF-8 CRLF INV ALT Shift 88%

27°C Mostly cloudy 08:04 07-10-2021

Generate $y[n]$ as given in the question

Spyder (Python 3.8)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
art Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py
Experiment 3.py
1 #!/usr/bin/python3
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 n = np.arange(-6, 6) # Get evenly spaced values within the interval [-6, 6]
7 arr=[2,3,4,5,6] # Initialize array with values as given in question.
8 plt.xlabel('n')
9 plt.title('Unit Impulse Discrete-Time Function for x[n] = [2 3 4 5 6]')
10 plt.stem(n,arr) # stem plot prints vertical lines at each x position covered under the
graph from the baseline to y, and places a marker there.
11 plt.show()

12 def x_of_n(): # Generate x[n] as given in the question
13     list=[] # Null list Declaration
14     for sample in n:
15         if sample<3:
16             if sample<2:
17                 list.append(0) # Adds a single item to the existing list.
18             else:
19                 list.append(0) # Adds a single item to the existing list.
20         else:
21             list.append(0) # Adds a single item to the existing list.
22     return (list) # Send the value of list back and exits the function.

23 def y_of_n(): # Generate y[n] as given in the question
24     list=[] # Null list Declaration
25     for sample in n:
26         if sample<0:
27             list.append(0) # Adds a single item to the existing list.
28         elif sample<1:
29             list.append(0) # Adds a single item to the existing list.
30         elif sample<2:
31             if sample>=2:
32                 list.append(sample/abs(sample)) # abs() returns the absolute value of a
number.
33             else:
34                 list.append(0) # Adds a single item to the existing list.
35         else:
36             list.append(0) # Adds a single item to the existing list.
37     return(list) # Send the value of list back and exits the function.

38 def sketch_1(): # Sketch x[3n - 1]
39     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
40     plt.subplot(3,1,1) # subplot(rows, columns, index) describes the figure layout
41     plt.xlabel('n')
42     plt.title('Function for x[n]')
43     plt.stem(n,x_of_n())
44     plt.show()

45 def sketch_2(): # Sketch x[3n - 1]
46     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
47     plt.subplot(3,1,2) # subplot(rows, columns, index) describes the figure layout
48     plt.xlabel('n')
49     plt.title('Function for x[3n - 1]')
50     plt.stem(n,y_of_n())
51     plt.show()

52 def sketch_3(): # Sketch x[n - 1]
53     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
54     plt.subplot(3,1,3) # subplot(rows, columns, index) describes the figure layout
55     plt.xlabel('n')
56     plt.title('Function for x[n - 1]')
57     plt.stem(n,y_of_n())
58     plt.show()

59 def sketch_4(): # Sketch x[n - 2] + y[n + 2].
60     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
61     plt.subplot(3,1,4) # subplot(rows, columns, index) describes the figure layout
62     plt.xlabel('n')
63     plt.title('Function for x[n - 2] + y[n + 2]')
64     plt.stem(n,y_of_n())
65     plt.show()

```

Variables explorer Help File View

Console 1/A

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

In [4]: runfile('E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py', wdir='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: 4

In [4]: runfile('E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py', wdir='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3')

Python console history

OS: USP Python ready CPU ready Code Python 3.8.0 Line 156, Col 1 UTF-8 CRLF INV ALT Shift 88%

27°C Mostly cloudy 08:05 07-10-2021

Sketch $x[3n - 1]$

Spyder (Python 3.8)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
art Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py
Experiment 3.py
  98         list.append(0) # Adds a single item to the existing list.
  99     return (list) # Send the value of list back and exits the function.
100
101 def x_of_n(n): # Generate x[n] as given in the question
102     list=[] # Null List Declaration
103     for sample in n:
104         if sample==0:
105             list.append(0) # Adds a single item to the existing list.
106         elif sample<0:
107             if sample<-4:
108                 list.append(sample/abs(sample)) # abs() returns the absolute value of a number.
109             else:
110                 list.append(0) # Adds a single item to the existing list.
111         else:
112             list.append(0) # Adds a single item to the existing list.
113     return(list) # Send the value of list back and exits the function.
114
115 def sketch_1(y): # Sketch x[3n + 1]
116     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
117     plt.subplot(3,1,1) # subplot(rows, columns, index) describes the figure layout
118     plt.xlabel('n')
119     plt.ylabel('x [n]')
120     plt.title('Function for x[n]')
121     plt.stem(n, x_of_n(n)) # Calls the function x_of_n(n) and displays x[n] plot.
122
123     list=[] # Null List Declaration
124     for sample in n:
125         list.append(3*sample + 1) # Adds a single item to the existing list, x[3n+1]
126
127     plt.subplot(3,1,3) # subplot(rows, columns, index) describes the figure layout
128     plt.xlabel('n')
129     plt.ylabel('x [3n-1]')
130     plt.title('Function for x[3n-1]')
131     plt.stem(n, x_of_n(list)) # Calls the function x_of_n(list) and displays x[3n-1] plot.
132     plt.show() # To view both the sub-plots.
133
134 def sketch_2(): # Sketch x[n - 2] + y[n + 2]
135     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
136
137     list_x1=[] # Null List Declaration
138     for sample in n:
139

```

In [4]: runfile('E:/Plan B/Amita Vishwa Vidyaapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py', wdir='E:/Plan B/Amita Vishwa Vidyaapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3')

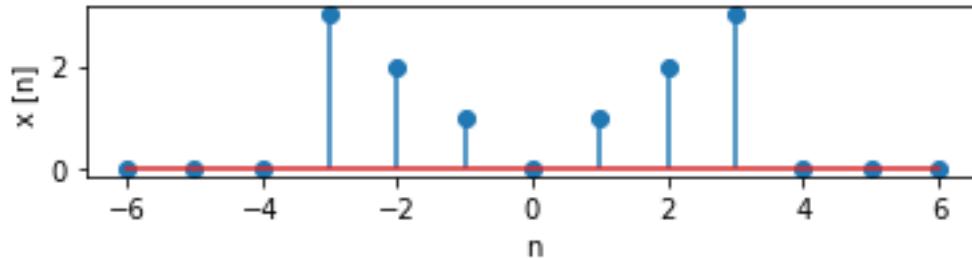
Out[4]:

MENU:

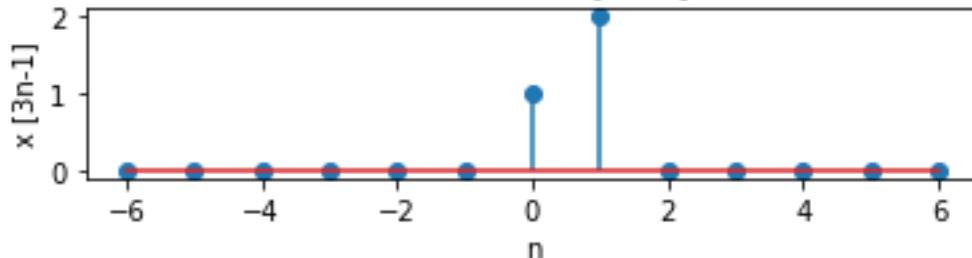
1. Step Signal Subtraction.
2. Plot Impulse Signal.
3. Sketch $x[n - 1]$.
4. Sketch $x[n - 2] + y[n + 2]$.
5. Exit

Enter the number corresponding to the menu to implement the choice: 3

Function for $x[n]$



Function for $x[3n-1]$



Sketch $x[n - 2] + y[n + 2]$

Scipy (Python 3.8)

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
.../Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py
Experiment 3.py
100     plt.title('Function for x[n-2]')
101     plt.stem(n, x_of_n(list)) # Calls the function x_of_n(list) and displays x[n-2] plot.
102     plt.show() # To view both the subplots.
103
104 def sketch_2(): # Sketch x[n - 2] + y[n + 2]
105     n = np.arange(-6,7) # Get evenly spaced values within the interval [-6,7]
106
107     list_x = [] # Null List Declaration
108     for sample in n:
109         list_x.append(sample - 2) # Adds a single item to the existing list, x[n-2]
110
111     plt.subplot(3,1,1) # subplot(rows, columns, index) describes the figure layout
112     plt.xlabel('n')
113     plt.ylabel('x[n-2]')
114     plt.title('Function for x[n-2]')
115     plt.stem(n, x_of_n(list_x)) # Calls the function x_of_n(list_x) and displays x[n-2]
116     plot.
117
118     list_y = [] # Null List Declaration
119     for sample in n:
120         list_y.append(sample + 2) # Adds a single item to the existing list, y[n+2]
121
122     plt.subplot(3,1,2) # subplot(rows, columns, index) describes the figure layout
123     plt.xlabel('n')
124     plt.ylabel('y[n+2]')
125     plt.title('Function for y[n+2]')
126     plt.stem(n, y_of_n(list_y)) # Calls the function y_of_n(list_y) and displays y[n+2]
127     plot.
128
129     result = np.add(x_of_n(list_x),y_of_n(list_y)) # Display the resultant graph, x[n - 2]
130     + y[n + 2]
131     plt.subplot(3,1,3) # subplot(rows, columns, index) describes the figure layout
132     plt.xlabel('n')
133     plt.ylabel('x[n-2]+y[n+3]')
134     plt.title('Function for x[n-2] + y[n+3]')
135     plt.stem(n, result) # Stem plot plots vertical lines at each n position covered under
136     the graph from the baseline to y, and places a marker there.
137     plt.show() # To view all the three sub-plots.
138
139 # Main
140 while True: # This simulates a Do loop
141     choice = input('

```

Type here to search

Console 1/A

4. Sketch $x[n - 2] + y[n + 2]$.
5. Exit
Enter the number corresponding to the menu to implement the choice: 3

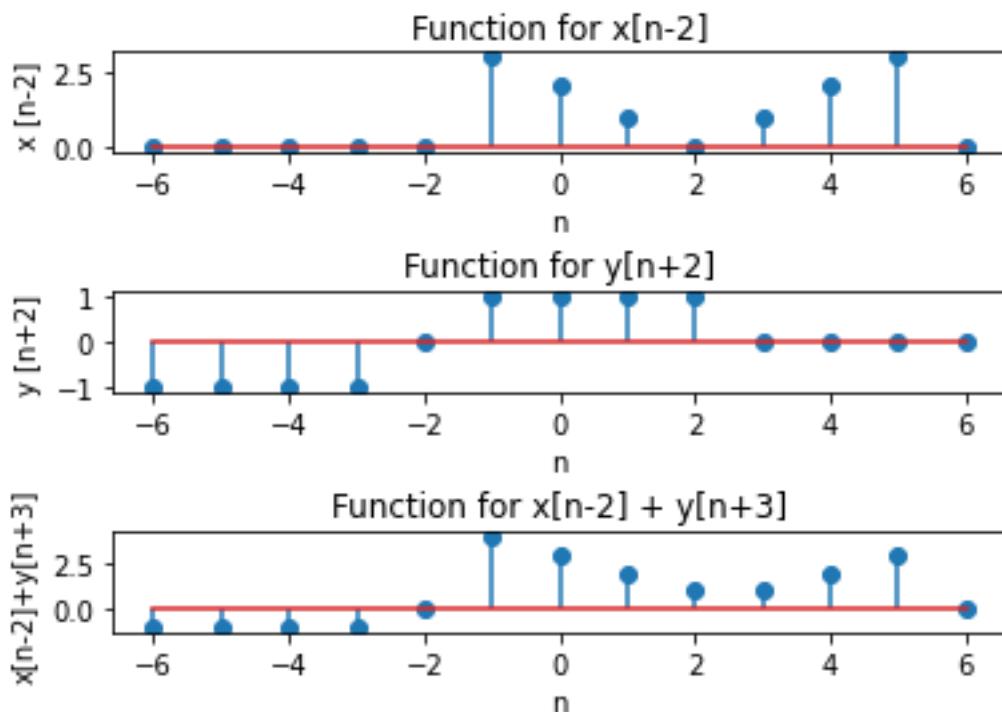
In [4]: runfile('E:/Plan B/Amita Vishwa Vidyapeeth/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py', wdir='E:/Plan B/Amita Vishwa Vidyapeeth/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3')

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

In [5]: runfile('E:/Plan B/Amita Vishwa Vidyapeeth/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3/Experiment 3.py', wdir='E:/Plan B/Amita Vishwa Vidyapeeth/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 3')

Python console Ready

27°C Mostly cloudy



Menu Driven Implementation

The screenshot shows the Spyder Python 3.8 IDE interface. On the left, the code editor displays a script named 'Experiment 3.py' containing Python code for signal processing tasks. The code includes imports for numpy and matplotlib, defines functions for signal operations, and implements a menu-driven loop for user interaction. On the right, three plots are displayed side-by-side:

- The top plot is titled "Function for $x[n-2]$ " and shows a discrete-time signal $x[n-2]$ plotted against n . The signal has values 1 at $n = -4, -2, 0, 2, 4$ and 0 elsewhere.
- The middle plot is titled "Function for $y[n+2]$ " and shows a discrete-time signal $y[n+2]$ plotted against n . The signal has values 1 at $n = -4, -2, 0, 2, 4$ and 0 elsewhere.
- The bottom plot is titled "Function for $x[n-2] + y[n+2]$ " and shows the sum of the two signals. It has values 2 at $n = -4, -2, 0, 2, 4$ and 0 elsewhere.

The terminal window below the plots shows the execution of the script, including the menu options and user input. The system tray at the bottom indicates the date (02-10-2021), time (08:08), and weather (27°C Mostly cloudy).

```
121 plt.subplot(3,1,1) # subplot(rows, columns, index) describes the figure layout
122 plt.xlabel('n')
123 plt.ylabel('y[n]')
124 plt.title('Function for y[n+2]')
125 plt.stem(n, y_of_n(list_y)) # Calls the function y_of_n(list_y) and displays y[n+2]
126 plot.
127 result = np.add(x_of_n(list_x),y_of_n(list_y)) # display the resultant graph, x[n-2]
+ y[n+2]
128 plt.subplot(3,1,2) # subplot(rows, columns, index) describes the figure layout
129 plt.xlabel('n')
130 plt.ylabel('x[n-2]')
131 plt.title('Function for x[n-2]')
132 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.
133 plt.show() # To view all the three sub-plots.
134
135 # Main
136 while True: # This simulates a Do Loop
137     choice = input()
138     if choice == '1': Step Signal Subtraction, In 2. Plot Impulse Signal, In 3. Sketch
x[n - 1], In 4. Sketch x[n - 2] + y[n + 2], In 5. Exit Enter the number corresponding
to the menu to implement the choice: " # Menu Driven Implementation
139     if choice == str(1): # str() returns the string version of the variable "choice"
140         step_sub()
141         break
142     elif choice == str(2):
143         impulse_plot() # Plot Impulse Signal
144         break
145     elif choice == str(3):
146         sketch_1() # Sketch x[n - 1]
147         break
148     elif choice == str(4):
149         sketch_2() # Sketch x[n - 2] + y[n + 2]
150         break
151     elif choice == str(5):
152         break # Exit loop
153     else:
154         print("Error: Invalid Input! Please try again.\n")
```

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

$$x_1 = [2, 4, 7, 3]$$

$$x_2 = [1, 2, 5, 3]$$

Sketch the input and system responses appropriately.

①

NAME - SANTOSH (CB.EN.U4 CLE 20053)

DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CLE) A

LAB TITLE AND CODE : SIGNAL PROCESSING LAB IACCE 281

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:

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

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:

$$x_3(t) \rightarrow S \rightarrow y_3(s), \text{ where } y_3(s) = \alpha y_1(s) + \beta y_2(s)$$

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.

$\overrightarrow{\text{PTD}}$

②

$$f(t) \xrightarrow{\otimes} \boxed{H} \rightarrow y(t) \equiv f(t) \xrightarrow{\otimes} \boxed{H} \rightarrow \otimes \rightarrow y'(t)$$

\uparrow
 K

\uparrow
 K

$H(Kf(t)) = KH(f(t)) \rightarrow 1^{\text{st}}$ step where K is the scaling factor
A block diagram demonstrating the scaling property of linearity.

$$\begin{array}{c} f_1 \xrightarrow{\otimes} \boxed{H} \rightarrow y \\ f_2 \xrightarrow{\otimes} \boxed{H} \rightarrow y \\ \hline f_1 + f_2 \xrightarrow{\otimes} \boxed{H} \rightarrow y \end{array}$$

$H(f_1(t) + f_2(t)) = H(f_1(t)) + H(f_2(t))$
A block diagram demonstrating the superposition property of linearity.

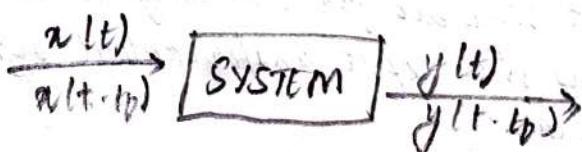
$$H(k_1 f_1(t) + k_2 f_2(t)) = k_1 H(f_1(t)) + k_2 H(f_2(t))$$

→ 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)

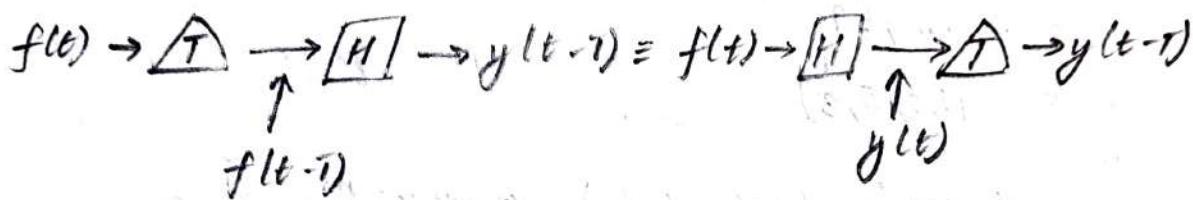
In a time-invariant system, there are no changes in system structure as a function of a time t . A system is time-invariant if -



③

From the diagram, we can observe that for any time t_0 and any input $x(t)$ that produces response $y(t)$, the response to the time-shifted input $x(t-t_0)$ is equal to the time-shifted output $y(t-t_0)$ 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 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.

(A)

+ PROGRAM WITH COMMENTS :

```
1 # Import library source files
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 n = np.arange(0, 4) # Get evenly spaced values within the
6 Interval [0, 4]
7 # Given Input sequences:
8 x1 = [2, 4, 2, 3]
9 x2 = [1, 2, 5, 3]
10
11 def scaling (scale, choice): # check Homogeneity condition
12     before_scaling = []
13     after_scaling = []
14
15     if choice == str(1): # calculate for x1 = [2, 4, 1, 3]
16
17         for sample in range(len(n)):
18             temp = sample + x1[sample]
19             before_scaling.append (int(scale) * temp) # scaling the
20             output
21
22             for sample in range(len(n)):
23                 temp = int(scale) + x1[sample]
24                 after_scaling.append (sample * temp) # sending output
25                 into system
26
27     else: # calculate for x2 = [1, 2, 5, 3]
28
29         for sample in range(len(n)):
30             temp = sample + x2[sample]
31             before_scaling.append (int(scale) * temp) # scaling the
32             output
```

⑤

```
30     for sample in range (len(n)):  
31         temp = float (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) + '{3}'.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 n 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) + '{3}'.format  
(after_scaling)) # Formats the specified values and insert  
them inside the string's placeholder.  
44     plt.stem (n, after_scaling) # stem plot plots vertical  
lines at each n 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)
```

⑥

```

51 else:
52     print("The given system does not satisfy Homogeneity")
      Homogeneity condition.')
53     return(0)
54
55 def superposition (): # check additivity condition
56     before_adding = []
57     after_adding = []
58     y1 = []
59     y2 = []
60     x = []
61
62     for sample in range (len(n)):
63         y1.append (sample + x1[sample])
64         y2.append (sample + x2[sample])
65         before_adding.append (y1[sample] + y2[sample])
66
67     # Processing System Individually
68
69     plt.subplot (3,1,1) # subplot (rows, columns, index) describes
    the figure layout
70     plt.xlabel ('n')
71     plt.ylabel ('y[n]')
72     plt.title ('H(x1(t)) + H(x2(t))')
73     plt.stem (n, before_adding)
74
75     for sample in range (len(n)):
76         x.append (x1[sample] + x2[sample])
77         after_adding.append (sample + x[sample])
78
79     # Processing system Together
80
81     plt.subplot (3,1,3) # subplot (rows, columns, index)
    describes the figure layout
82     plt.xlabel ('n')
83     plt.ylabel ('y[n]')
84     plt.title ('H(x1(t)) + H(x2(t))')
85     plt.stem (n, after_adding) # stem plot plots vertical
lines at each a 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('In 2. Superposition condition :- ')
82 if (before_adding == after_adding): # Condition to check
83     if both the systems are equal or not after adding
84         print("The given system satisfies superposition condition.")
85     return (1)
86 else:
87     print("The given system does not satisfy superposition
88 condition.")
89     return (0)
90
91
92 def time (shift, choice): # check Time Variance
93     Input_delay = []; output_delay = []
94
95     if choice == str(1): # calculate for x1 = [2, 4, 2, 3]
96
97         # Output Delay
98         for sample in range (len(x1)):
99             output_delay.append (sample + all_sample) # sending
100            output into system
101
102         for sample in range (int(shift)):
103             output_delay.append (0) # adds extra sample at the end
104            for shifting
105
106         for sample in range (int(shift)):
107             for sample in range (len(output_delay)-1, 0, -1):
108                 output_delay[sample] = output_delay[sample-1]
109
110             # shift the sample

```

```

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

```

```

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

```

10

```

156 plt.stem(np.arange(len(input_delay)), input_delay) # stem
plot plots vertical lines at each n 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 ' +
str(choice) + '3'.format(output_delay)) # Formats the
specified values 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 n 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

```

(11)

177 choice
~~choice~~ -> Input /

178 "MENU: Determine whether the system $y[n] = nx[n]$ is linear and time invariant. In 1. $x1 = [2, 4, 1, 3]$ In 2. $x2 = [1, 3, 5, 3]$
 In 3. Exit In 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). In 1. Homogeneity Condition: -In enter the value for scaling factor: ")

183

184 if scaling (scale, choice) == superposition (1): # Condition to check if both the systems are linear or not

185 print ("In Since both the properties are satisfied; the given system is linear.")

186 first_condition = 1

187 else:

188 print ("In 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. In 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 ("In The given system is a Linear Time Invariant (LTI) system. ")

195 else:

(1)

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

+ INFERENCE:

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

Check Homogeneity Condition

Spyder (Python 3.8)

```
# import library source files
# import numpy as np
# import matplotlib.pyplot as plt
# Given Input sequences:
x1 = [2,4,7,3]
x2 = [1,2,5,3]
def scaling (scale, choice): # Check Homogeneity Condition
    before_scaling=[]
    after_scaling=[]
    if choice == str(1): # Calculate for x1 = [2,4,7,3]
        for sample in range(len(x1)):
            temp = sample * x1[sample]
            before_scaling.append(int(scale)*temp) # Scaling the output
    for sample in range(len(x1)):
        temp = int(scale) * x1[sample]
        after_scaling.append(sample*temp) # Sending output into system
    else: # Calculate for x2 = [1,2,5,3]
        for sample in range(len(x2)):
            temp = sample * x2[sample]
            before_scaling.append(int(scale)*temp) # Scaling the output
    for sample in range(len(x2)):
        temp = int(scale) * x2[sample]
        after_scaling.append(sample*temp) # Sending output into system
    plt.subplot(2,1,1) # subplot(rows, columns, index) describes the figure layout
    plt.xlabel('n')
    plt.ylabel('y') + str(choice) + '(n)'
    plt.title('Before Scaling x1[0, 8, 28, 18]') # Formats the specified value(s) and insert them inside the string's placeholder.
    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.
    plt.subplot(2,1,2) # subplot(rows, columns, index) describes the figure layout
    plt.xlabel('n')
    plt.ylabel('y') + str(choice) + '(n)'
    plt.title('After Scaling x1[0, 8, 28, 18]') # Formats the specified value(s) and insert them inside the string's placeholder.
    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.
    plt.show() # To view both the sub-plots
    if (before_scaling == after_scaling): # Condition to check if both the systems are equal or not after scaling
        print("The given system satisfies Homogeneity Condition.")
        return()
    else:
        print("The given system does not satisfy Homogeneity Condition.")
        return(0)
def superposition(): # Check Additivity Condition
    pass
```

File Edit Search Source Run Debug Consoles Projects Tools View Help

E:\Plan B\Anirita Vishwa Vidyaapeeth\Subject Materials\Semester III\Signal Processing Lab (19CE281)\Assignments\Experiment 4\Experiment 4.py

Experiment 4.py

117%

Before Scaling x1[0, 8, 28, 18]

After Scaling x1[0, 8, 28, 18]

Variables explorer Help File menu

Console 1/A

Python 3.8.8 (default, Apr 13 2021, 25:08:03) [MSC v.1936 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

iPython 7.22.0 -- An enhanced Interactive Python.

In [3]: runfile('E:/Plan B/Anirita Vishwa Vidyaapeeth/Subject Materials/Semester III/Signal Processing Lab (19CE281)/Assignments/Experiment 4/Experiment 4.py', wdir='E:/Plan B/Anirita Vishwa Vidyaapeeth/Subject Materials/Semester III/Signal Processing Lab (19CE281)/Assignments/Experiment 4')

MENU: Determining 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

2. Additivity Condition: -
Enter the value for scaling factor: 2

3. Superposition Condition: -
Enter the value for scaling factor: 2

Type here to search

OS: LSP Python ready 0 code (Python 3.8.8) 1 line 4 Col 1 ASCII CRLF RW Mem 91% 26°C 20:44 ENG 13-10-2021

Spyder (Python 3.8)

```
# import library source files
# import numpy as np
# import matplotlib.pyplot as plt
# Given Input sequences:
x1 = [2,4,7,3]
x2 = [1,2,5,3]
def scaling (scale, choice): # Check Homogeneity Condition
    before_scaling=[]
    after_scaling=[]
    if choice == str(1): # Calculate for x1 = [2,4,7,3]
        for sample in range(len(x1)):
            temp = sample * x1[sample]
            before_scaling.append(int(scale)*temp) # Scaling the output
    for sample in range(len(x1)):
        temp = int(scale) * x1[sample]
        after_scaling.append(sample*temp) # Sending output into system
    else: # Calculate for x2 = [1,2,5,3]
        for sample in range(len(x2)):
            temp = sample * x2[sample]
            before_scaling.append(int(scale)*temp) # Scaling the output
    for sample in range(len(x2)):
        temp = int(scale) * x2[sample]
        after_scaling.append(sample*temp) # Sending output into system
    plt.subplot(2,1,1) # subplot(rows, columns, index) describes the figure layout
    plt.xlabel('n')
    plt.ylabel('y') + str(choice) + '(n)'
    plt.title('Before Scaling x1[0, 8, 28, 18]') # Formats the specified value(s) and insert them inside the string's placeholder.
    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.
    plt.subplot(2,1,2) # subplot(rows, columns, index) describes the figure layout
    plt.xlabel('n')
    plt.ylabel('y') + str(choice) + '(n)'
    plt.title('After Scaling x1[0, 8, 28, 18]') # Formats the specified value(s) and insert them inside the string's placeholder.
    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.
    plt.show() # To view both the sub-plots
    if (before_scaling == after_scaling): # Condition to check if both the systems are equal or not after scaling
        print("The given system satisfies Homogeneity Condition.")
        return()
    else:
        print("The given system does not satisfy Homogeneity Condition.")
        return(0)
def superposition(): # Check Additivity Condition
    pass
```

File Edit Search Source Run Debug Consoles Projects Tools View Help

E:\Plan B\Anirita Vishwa Vidyaapeeth\Subject Materials\Semester III\Signal Processing Lab (19CE281)\Assignments\Experiment 4\Experiment 4.py

Experiment 4.py

117%

Before Scaling x1[0, 8, 28, 18]

After Scaling x1[0, 8, 28, 18]

Variables explorer Help File menu

Console 1/A

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

2. Additivity Condition: -
Enter the value for scaling factor: 2

3. Superposition Condition: -
Enter the value for scaling factor: 2

The given system satisfies Homogeneity Condition.

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

Type here to search

OS: LSP Python ready 0 code (Python 3.8.8) 1 line 4 Col 1 ASCII CRLF RW Mem 91% 26°C 20:45 ENG 13-10-2021

Step 1: Scale the output.

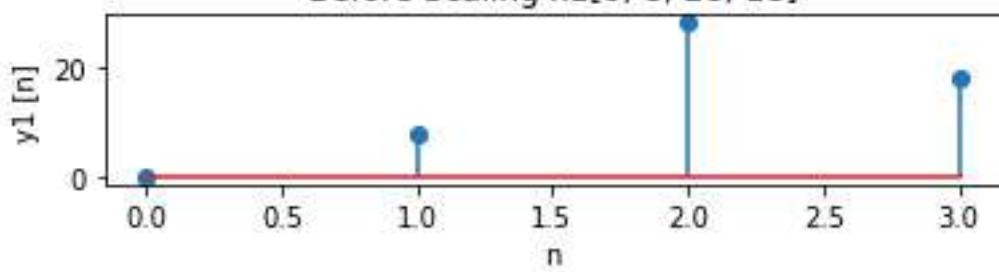
Step 2: Send output into the system.

Step 3: Plot graphs before and after scaling.

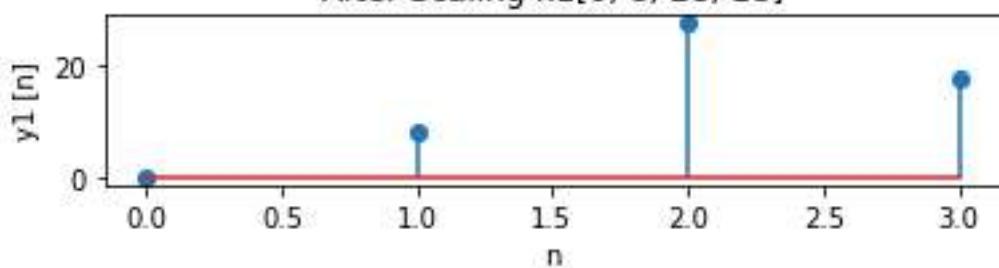
Step 4: Check if both the systems are equal or not after scaling.

$$x1 = [2, 4, 7, 3]$$

Before Scaling $x1[0, 8, 28, 18]$

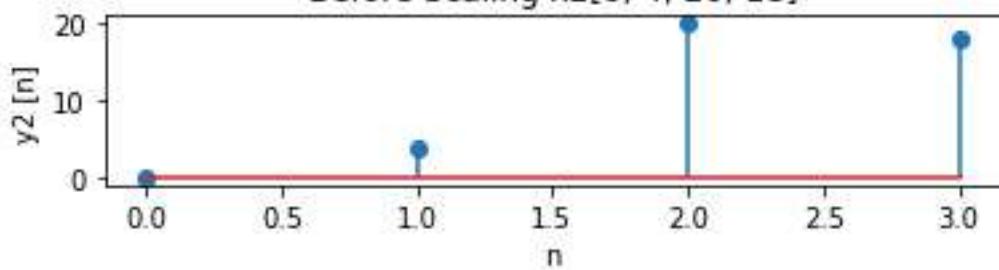


After Scaling $x1[0, 8, 28, 18]$

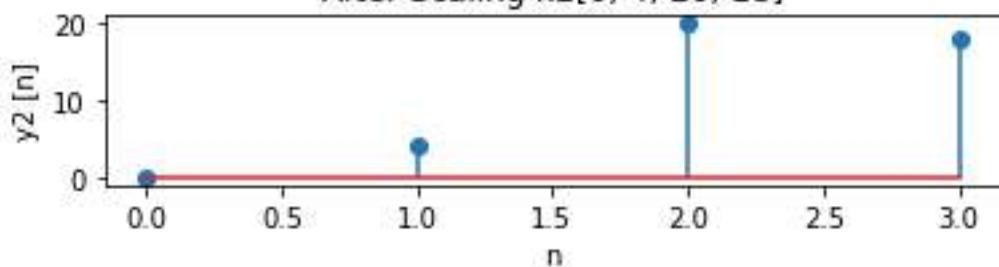


$$x2 = [1, 2, 5, 3]$$

Before Scaling $x2[0, 4, 20, 18]$



After Scaling $x2[0, 4, 20, 18]$



The given system satisfies the Homogeneity Condition.

Check Additivity Condition

The screenshot shows the Spyder Python IDE interface. The code in the editor window is as follows:

```
def superposition():
    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[0] + x2[0])')
    plt.stem(n, before_adding) # Stem plot plots vertical lines at each x position covered under the graph from the baseline to y, and places a marker there.

    for sample in range(n):
        x.append(x1[sample])
        after_adding.append(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) + 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.

    plt.show() # To view both the sub-plots

    print('1. Superposition Condition: -')
    if (before_adding == after_adding): # condition to check if both the systems are equal or not often adding
        print("The given system satisfies Superposition Condition.")
        return(1)
    else:
        print("The given system does not satisfy Superposition Condition.")
        return(0)

def time(shift, choice): # Check Time Variance
    input_delay = []
    output_delay = []
```

The plots show two vertically aligned stem plots. The top plot is titled "H(x1[0] + x2[0])" and the bottom plot is titled "H(x1(t) + x2(t))". Both plots have an x-axis labeled "n" ranging from 0 to 30 and a y-axis ranging from 0 to 20. Vertical stems are plotted at n=10, n=20, and n=30, with markers at the top of each stem.

The screenshot shows the Spyder Python IDE interface. The code is identical to the one above, but the console output is different. It includes a menu for determining linearity and time-invariance based on input sequences 1, 2, and 3.

```
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 plots are identical to the ones in the first screenshot.

Step 1: Process System Individually.

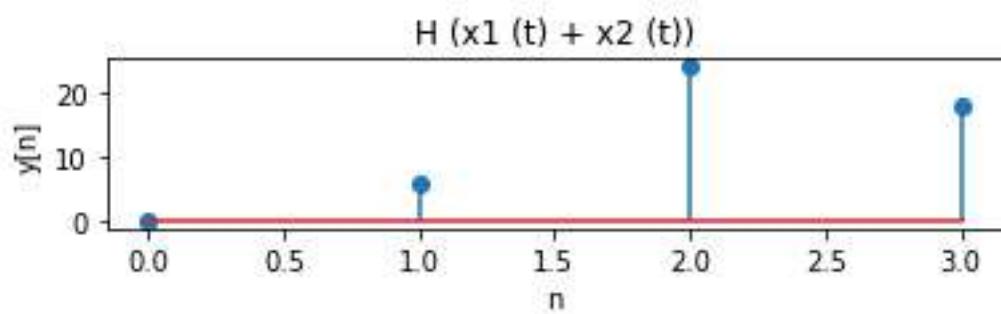
Step 2: Plot graph before adding.

Step 3: Process System Together.

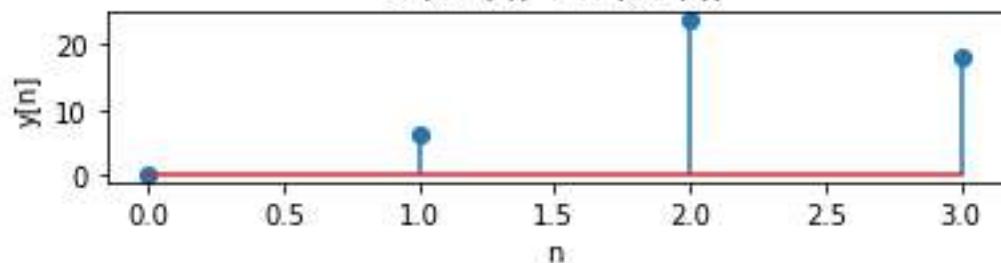
Step 4: Plot graph after adding.

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

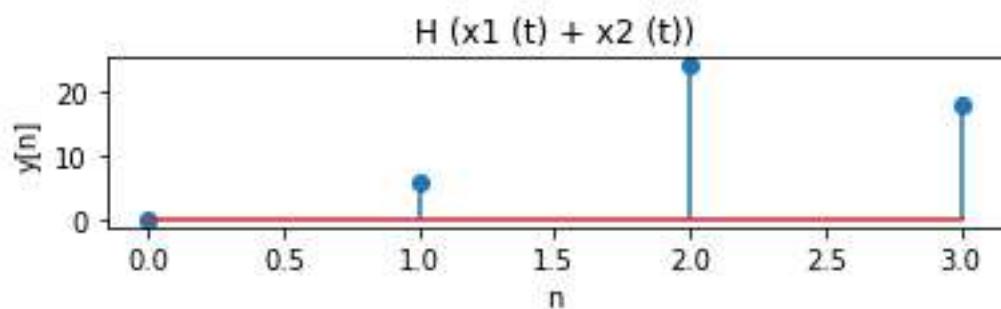
$$x1 = [2, 4, 7, 3]$$



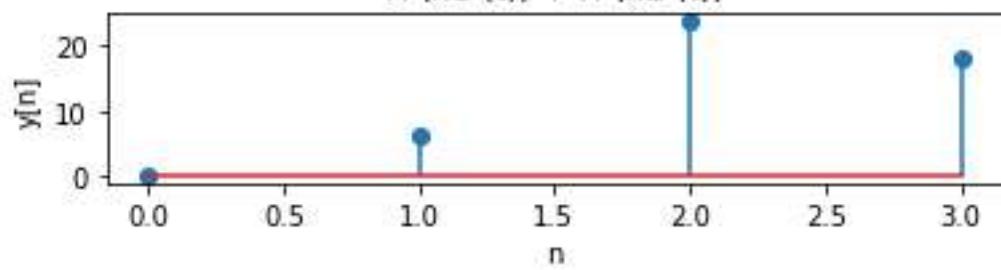
$$H(x_1(t)) + H(x_2(t))$$



$$x2 = [1, 2, 5, 3]$$



$$H(x_1(t)) + H(x_2(t))$$



The given system satisfies Superposition Condition.

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

Check Time Variance

```
Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Experiment 4.py
def time(shift, choice):
    # Check Time Variance
    input_delay = []
    if choice == str(1): # Calculate for x1 = [2,4,7,3]
        # Output Delay
        for sample in range(len(x1)):
            output_delay.append(sample*x1[sample]) # Sending output into system
    for sample in range(int(shift)):
        output_delay.append(0) # Adds extra sample at the end for shifting
    for sample in range(int(shift)):
        for sample in range(len(x1)-1,0,-1):
            output_delay[sample]=output_delay[sample]-1 # Shift the sample
    for sample in range(int(shift)):
        output_delay[sample]=0 # Assign zero to the empty elements for the shifted bit
    # Input Delay
    for sample in range(intShift):
        x1.append(0) # Adds extra sample at the end for shifting
    for sample in range(intShift):
        for sample in range(len(x1)-1,0,-1):
            x1[sample]=x1[sample]-1 # Shift the sample
    for sample in range(intShift):
        x1[sample]=0 # Assign zero to the empty elements for the shifted bit
    for sample in range(len(x1)):
        input_delay.append(sample*x1[sample]) # Sending input into system
    else: # Calculate for x2 = [1,2,3,5]
        # Output Delay
        for sample in range(len(x2)):
            output_delay.append(sample*x2[sample]) # Sending output into system
    for sample in range(intShift):
        output_delay.append(0) # Adds extra sample at the end for shifting

Figure 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.
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.
Python console Ready
O: C:\SP Python ready O: OneDrive O: code (Python 3.8.0) V: The RIO_Cat1 ASCII OLEF RW: New 89%
26°C ↻ 4V ENG 13-10-2021
```

```
Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Experiment 4.py
for sample in range(intShift):
    for sample in range(len(output_delay)-1,0,-1):
        output_delay[sample]=output_delay[sample]-1 # shift the sample
    for sample in range(intShift):
        output_delay[sample]=0 # Assign zero to the empty elements for the shifted bit
    # Input Delay
    for sample in range(intShift):
        x2.append(0) # Adds extra sample at the end for shifting
    for sample in range(intShift):
        for sample in range(len(x2)-1,0,-1):
            x2[sample]=x2[sample]-1 # Shift the sample
    for sample in range(intShift):
        x2[sample]=0 # Assign zero to the empty elements for the shifted bit
    for sample in range(len(x2)):
        input_delay.append(sample*x2[sample]) # Sending input into system
    plt.subplot(3,1,1) # subplot(rows, columns, index) describes the figure layout
    plt.xlabel('t')
    plt.ylabel('y(t)')
    plt.title('Shifting input by ' + str(shift) + ' for x' + str(choice) +
              '(' + str(input_delay) + ')') # Formats the specified value(s) and insert them inside the string's placeholder.
    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.
    plt.subplot(3,1,3) # subplot(rows, columns, index) describes the figure layout
    plt.xlabel('t')
    plt.ylabel('y(t)')
    plt.title('Shifting output by ' + str(shift) + ' for x' + str(choice) +
              '(' + str(output_delay) + ')') # Formats the specified value(s) and insert them inside the string's placeholder.
    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.
    plt.show() # To view both the sub-plots

Enter the number corresponding to the menu to implement the input sequence: 2
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.
Python console Ready
O: C:\SP Python ready O: OneDrive O: code (Python 3.8.0) V: The RIO_Cat1 ASCII OLEF RW: New 89%
26°C ↻ 4V ENG 13-10-2021
```

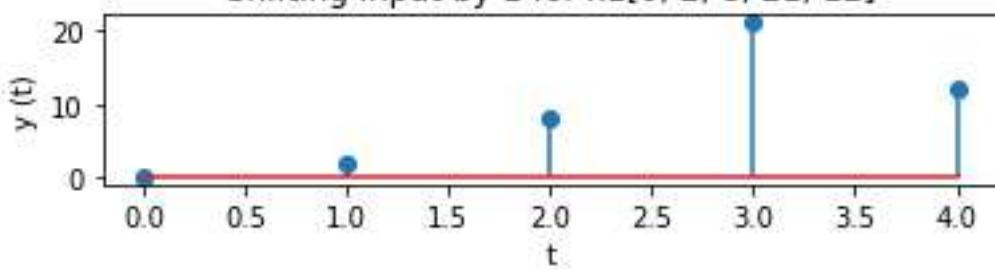
Step 1: Process input and output delay.

Step 2: Plot graphs before and after shifting.

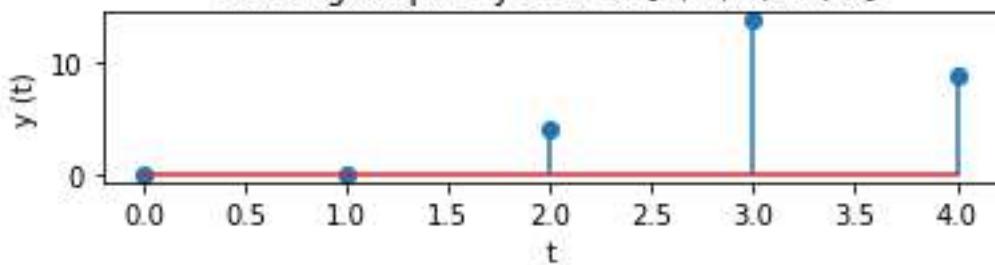
Step 3: Check if both the systems are equal or not after shifting.

$$x1 = [2, 4, 7, 3]$$

Shifting input by 1 for $x1[0, 2, 8, 21, 12]$

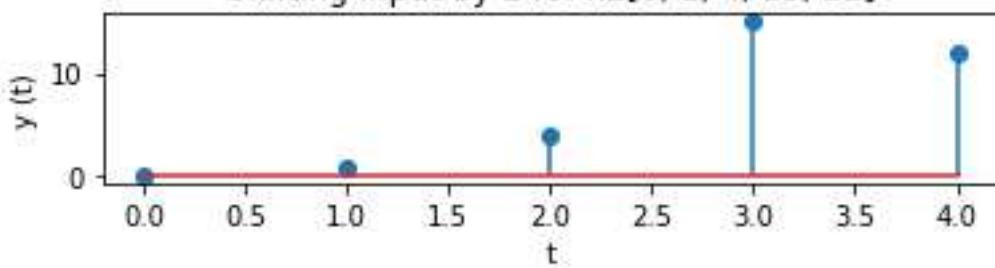


Shifting output by 1 for $x1[0, 0, 4, 14, 9]$

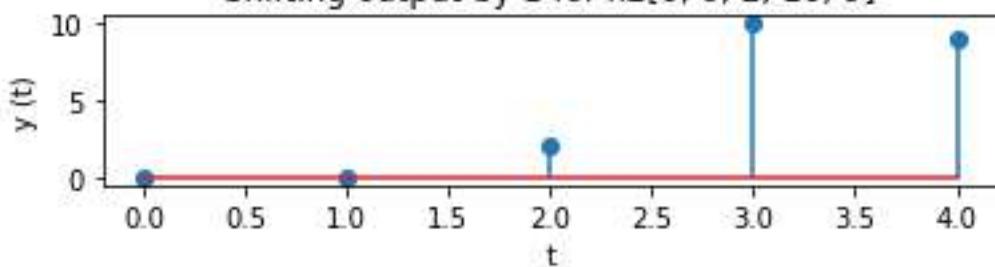


$$x2 = [1, 2, 5, 3]$$

Shifting input by 1 for $x2[0, 1, 4, 15, 12]$



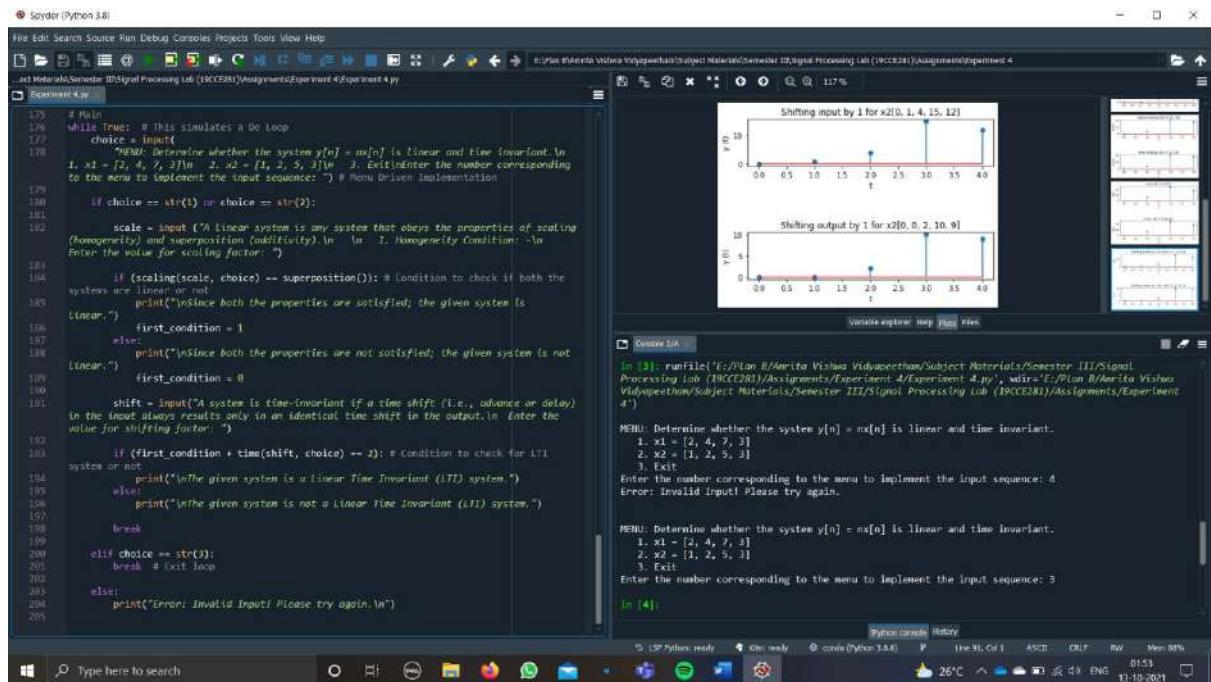
Shifting output by 1 for $x2[0, 0, 2, 10, 9]$



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



The screenshot shows the Spyder Python IDE interface. On the left, the code for 'Experiment 4.py' is displayed:

```
125 # Main
126 while True: # This simulates a Do Loop
127     choice = input()
128     #MENU: Determine whether the system y[n] = nx[n] is linear and time invariant. An
129     #      1. x1 = [2, 4, 7, 3]\n    2. x2 = [1, 2, 5, 3]\n    3. Exit\nEnter the number corresponding
130     # to the menu to implement the input sequence: 3 # Menu Driven Implementation
131
132     if choice == str(1) or choice == str(2):
133
134         scale = input("A linear system is any system that obeys the properties of scaling
135         (Homogeneity) and superposition (additivity). In\n    1. Homogeneity Condition: -In
136         Enter the value for scaling factor: ")
137
138         if (scaling(scale, choice) == superposition()): # Condition to check if both the
139             print("\nSince both the properties are satisfied; the given system is
140             linear.")
141         else:
142             print("\nSince both the properties are not satisfied; the given system is not
143             linear.")
144
145         first_condition = 0
146
147         shift = input("A system is time-invariant if a time shift (i.e., advance or delay)
148         in the input always results only in an identical time shift in the output. In Enter the
149         value for shifting factor: ")
150
151         if (first_condition + time(shift, choice) == 2): # Condition to check for LTI
152             print("\nThe given system is a Linear Time Invariant (LTI) system.")
153         else:
154             print("\nThe given system is not a Linear Time Invariant (LTI) system.")
155
156         break
157
158     elif choice == str(3):
159         break # Exit loop
160
161     else:
162         print("Error: Invalid Input! Please try again.\n")
```

On the right, there are two plots. The top plot is titled "Shifting input by 1 for x2[0, 1, 4, 15, 12]" and shows a discrete-time signal x2 with values at t=0, 1, 4, 15, and 12. The bottom plot is titled "Shifting output by 1 for x2[0, 0, 2, 10, 9]" and shows a discrete-time signal x2 with values at t=0, 1, 2, 10, and 9.

In the Spyder interface, the status bar shows "Python console Ready". The taskbar at the bottom includes icons for File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, Help, and a search bar.

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. $x_1 = [2, 4, 7, 3]$
2. $x_2 = [1, 2, 5, 3]$
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.

① NAME - SANJOSH (CB.EN.V4 CIE20053)
DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE) A

LAB TITLE AND CODE : SIGNAL PROCESSING LAB 19 CCE 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 $n[k]$ 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, $n[n] = [2, 3, 4]$

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

To find, $y[n]$

(3)

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

$$\begin{array}{r}
 & 1 & 2 & 3 \\
 & x & 2 & 3 & 4 \\
 \times & 2 & 4 & 6 & 8 & 0 \\
 & 0 & 3 & 6 & 9 & 0 \\
 + & 0 & 0 & 1 & 8 & 12 \\
 \hline
 & 2 & 7 & 16 & 17 & 12
 \end{array}
 \quad
 \begin{array}{l}
 \leftarrow R[n] \\
 \leftarrow x[n] \\
 \leftarrow x[0] \cdot h[n], i=0 \\
 \leftarrow x[1] \cdot h[n-1], i=1 \\
 \leftarrow x[2] \cdot h[n-2], i=2 \\
 \leftarrow y[n]
 \end{array}$$

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

* PROGRAM WITH COMMENTS :

```

1 # Import library source files
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 size_n = int(input("Enter the size of input x[n]: "))
6 x = [0] * (size_n) # 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_n, 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(n) + ']') # formats the specified values
and insert them inside the string's placeholder.
16 plt.stem(np.arange(0, size_n), x)
17

```

④

```

18 size_h = int(input("Enter the size of impulse response h[n]: "))
19 h = [0] * (size_h) # declare an array with the number of elements
20      equals to value of size.
21 print ("Enter the elements of the impulse response h[n]: ")
22 for i in range (0, size_h, 1):
23     h[i] = int(input("Element " + str(i+1) + ": "))
24 print ("The entered impulse response h[n] is:- " + str(h) + "\n")
25 plt.subplot (3, 1, 3)
26 plt.xlabel ('n')
27 plt.ylabel ('h[n]')
28 plt.title ('Impulse Response h').format(h) # formats the
29 specified values(s) and insert them inside the string's placeholder
30 plt.stem (np.arange (0, size_h), h)
31 plt.show()
32
33 temp = [] # duplicate for impulse response
34 for i in range (size_x):
35     temp = h.copy() # creates a copy of an existing list
36     for j in range (len(temp)):
37         temp[j] = int(temp[j]) * int(x[i])
38     for k in range (1):
39         temp.append (0) # adds required zeros at the end of list
40     for l in range (len(temp)-1, 0, -1):
41         temp[l] = temp[l-1] # shifts the zeros at the
42 beginning of list
43     temp[0] = 0 # prepares for next iteration
44 plt.title ('Intermediate plot x[{i}] * h[n-{i}] - {i}'.format(i=i, temp=temp))
45 plt.xlabel ('n')
46 plt.ylabel ('x[{i}] * h[n-{i}]'.format(i=i))

```

(5)

```

46 plt. stem(np.arange(0, len(temp)), temp)
47 plt.show()
48
49 # 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.
50
51 size = (size_n + size_R) - 1 # Compute the size of system response
52 x = np.pad(x, (0, size - size_x), 'constant')
53 h = np.pad(h, (0, size - size_h), 'constant')
54 y = np.zeros(size, dtype = int) # Returns a new array of given
  shape and type, with zeros.
55 iteration = 1 # Variable used for displaying the iteration sequence
56
57 for i in range(size):
58     for j in range(size):
59         if i >= j:
60             y[i] = int(y[i]) + (int(x[i-j]) + int(h[j]))
61             print("Iteration " + str(iteration) + ": " + str(y))
62 # Display result of each iteration
63 iteration += 1
64
65 print("\nThe system response is :- " + str(y) + "\n")
66
67 plt.xlabel('n')
68 plt.ylabel('y[n]')
69 plt.title('System Response y{3}'.format(y)) # formats the
  specified value(s) and insert them inside the string's placeholder.
70 plt.stem(np.arange(0, size), y)
71 plt.show()

```

70

⑥

* INFERENCE:

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

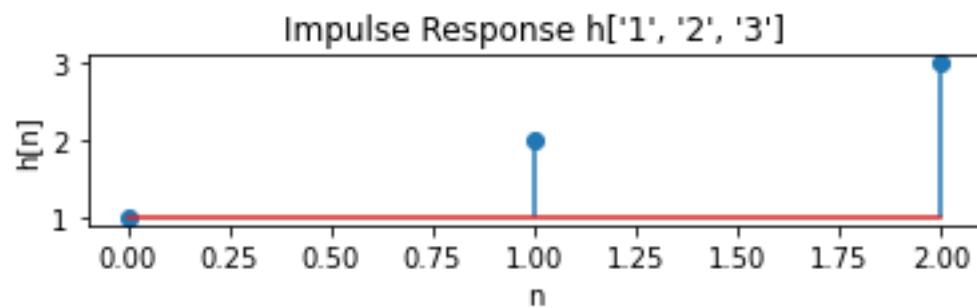
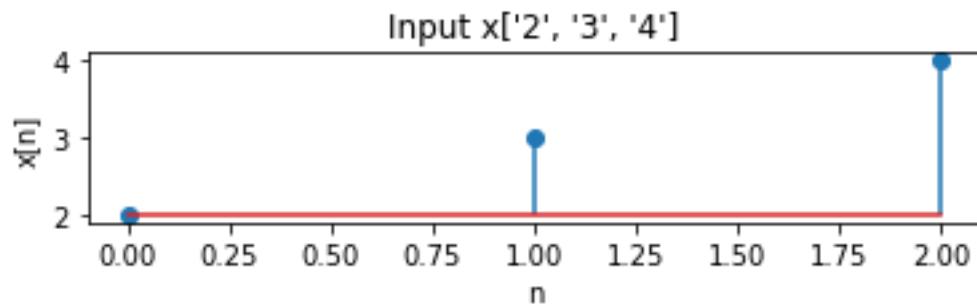
Input and Impulse Response

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named 'Experiment 5.py' containing Python code for generating impulse response plots. On the right, there are two plots: 'Input x[2, 3, 4]' and 'Impulse Response h[1, 2, 3]'. The 'Input' plot shows a discrete-time signal x[n] with values 2, 3, and 4 at n=0, 1, and 2 respectively. The 'Impulse Response' plot shows a discrete-time signal h[n] with values 1, 2, and 3 at n=0, 1, and 2 respectively. Below the plots, the Python console window shows the input values entered by the user.

```

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) # Declares an array with the number of elements equal to value of 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[2, 3, 4]')
16 plt.stem(np.arange(0,size_x),x)
17
18 size_h = int(input("Enter the size of impulse response h[n]: "))
19 h = [0] * (size_h) # Declares an array with the number of elements equal to value of size.
20 print("Enter the elements of the impulse response h[n] one-by-one as follows: -")
21 for i in range(0, size_h, 1):
22     h[i] = input("Element " + str(i+1) + ": ")
23     print("The entered impulse response h[n] is: - " + str(h) + "\n")
24
25 plt.subplot(3,1,3)
26 plt.xlabel('n')
27 plt.ylabel('h[n]')
28 plt.title('Impulse Response h[1, 2, 3]')
29 plt.stem(np.arange(0,size_h),h)
30
31 plt.show()
32
33 temp = [] # Duplicate for impulse response
34 for i in range(size_x):
35     temp.append([i]) # Creates a copy of an existing list
36     for j in range(len(temp)):
37         temp[j].append(int(x[i]))
38     for k in range(i):
39         temp.append([k]) # Adds required zeros at the end of list
40         for l in range(len(temp)-1,0,-1):
41             if l == 0:
42                 break
43

```



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

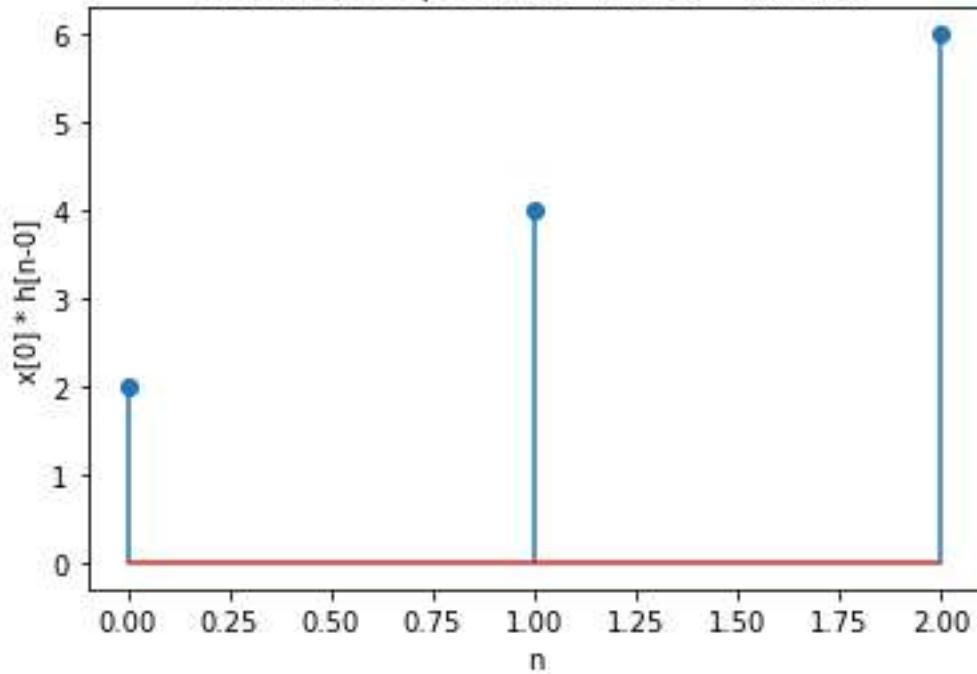
The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named 'Experiment 5.py' containing Python code for convolution. The code initializes input x[0] as [2, 3, 4], creates a copy temp, performs convolution sum, adds zeros, shifts elements, and plots the intermediate result. On the right, the IPython console shows the entered input x[0] and the impulse response h[n]. The plots pane shows three plots: the input signal x[0] as a step function, the impulse response h[n] as a unit impulse at n=0, and the intermediate plot x[0] * h[n-0] = [2, 4, 6] as a discrete-time signal with values at n=0, 100, and 200.

```

27         temp[i]=int(temp[j]) * int(x[i])
28     for k in range(i):
29         temp.append(0) # Adds required zeros at the end of list
30     for l in range(len(temp)-1,0,-1):
31         temp[l] = temp[l-1] # Shifts the zeros to the beginning of the list
32     temp[0] = 0 # Prepares for next iteration
33     plt.title('Intermediate plot x[0] * h[n-0] = [2, 4, 6]')
34     plt.xlabel('n')
35     plt.ylabel('x[0] * h[n-0]')
36     plt.stem(np.arange(0,size),y)
37     plt.show()
38
39 # 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.
40
41 size_x = (size_x + size_h) - 1 # Compute the size of system response
42 x = np.pad(x,(0, size_x - size_x), 'constant')
43 h = np.pad(h,(0, size_h - size_h), 'constant')
44 y = np.zeros(size, dtype = 'int') # Returns a new array of given shape and type, with zeros.
45 iteration = 1 # Variable used for displaying the iteration sequence
46
47 for i in range (size):
48     for j in range (size):
49         if i > j:
50             y[i] = int(x[i] * (int(x[i-j])*int(h[j])))
51             print("Iteration " + str(iteration) + ":" + str(y)) # Display result of each iteration
52             iteration += 1
53 print("The system response is: " + str(y) + "\n")
54
55 plt.xlabel('n')
56 plt.ylabel('y(n)')
57 plt.title('System Response y(n)',format(y)) # Formats the specified value(s) and insert them inside the string's placeholder.
58 plt.stem(np.arange(0,size),y)
59 plt.show()

```

Intermediate plot $x[0] * h[n-0] = [2, 4, 6]$



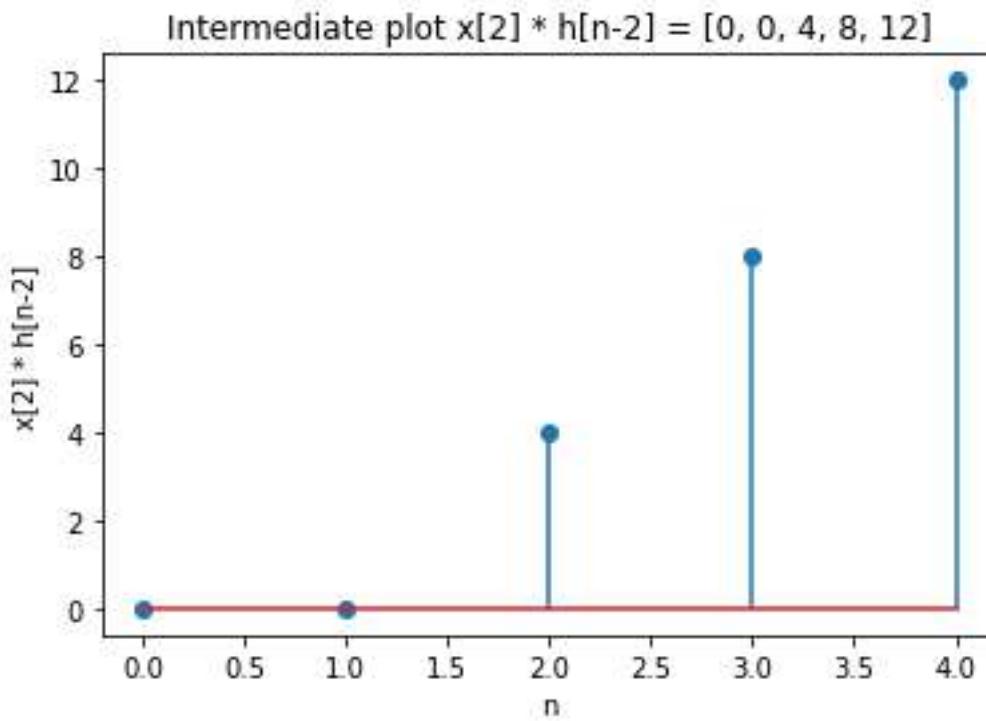
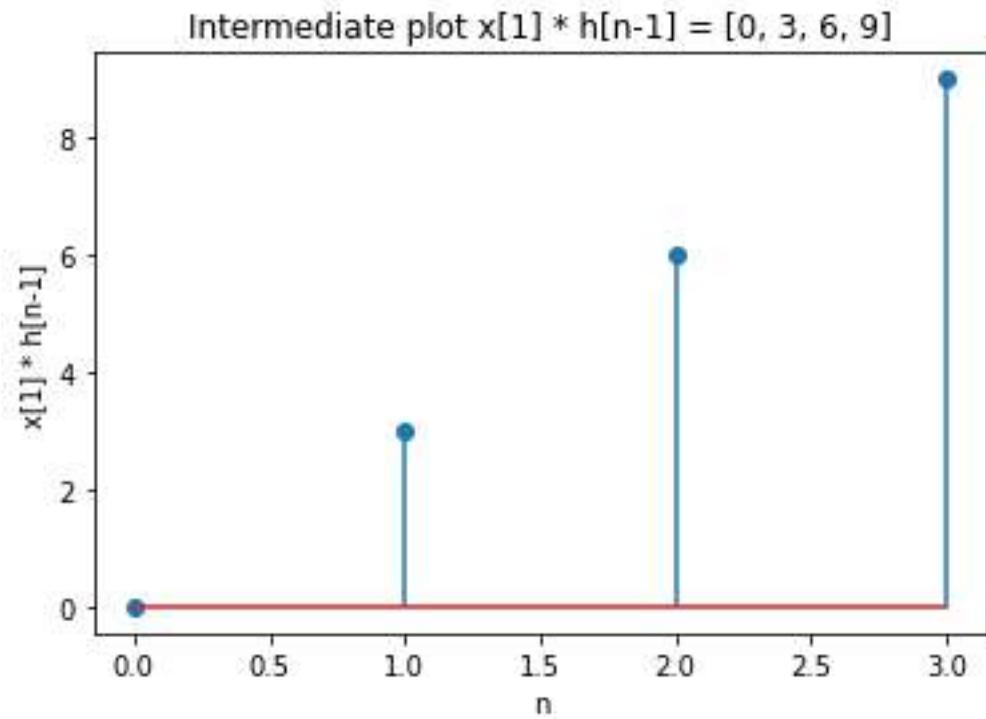
Step 1: Create a duplicate list temp[] and copy the elements impulse response to the duplicate list.

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:

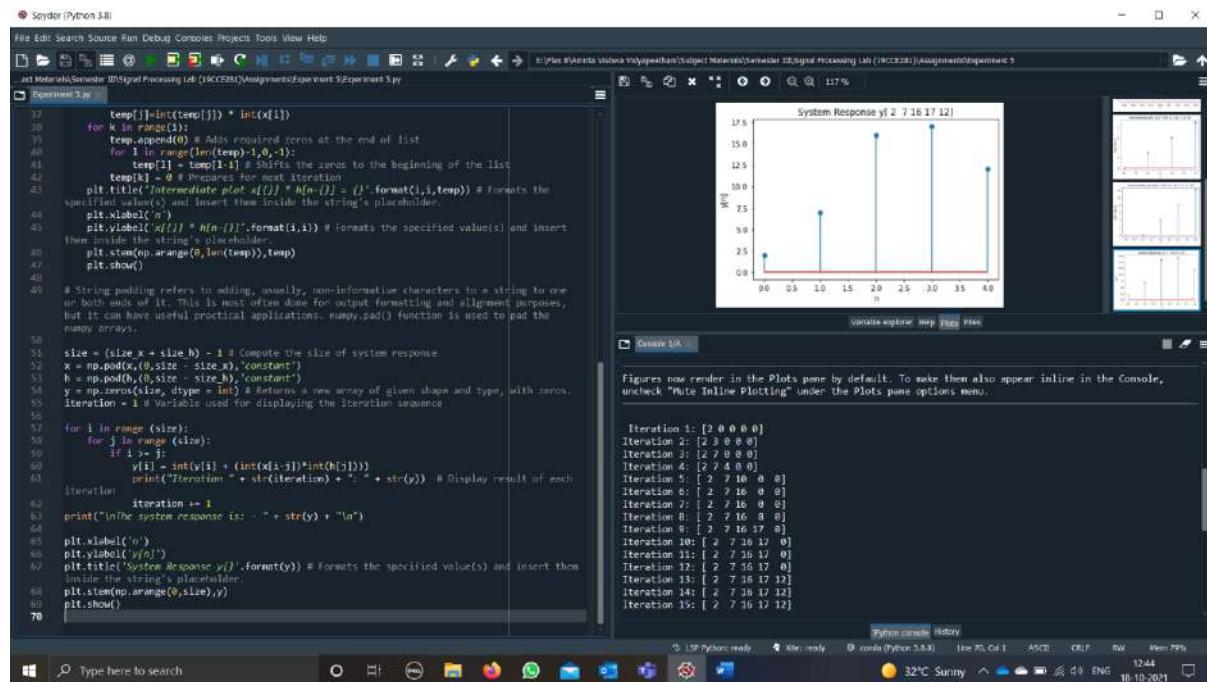
$$x[0] * h[n] = [2, 4, 6] \text{ for } i = 0$$

$$x[1] * h[n-1] = [0, 3, 6, 9] \text{ for } i = 1$$

$$x[2] * h[n-2] = [0, 0, 4, 8, 12] \text{ for } i = 2$$

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

Iterations and System Response

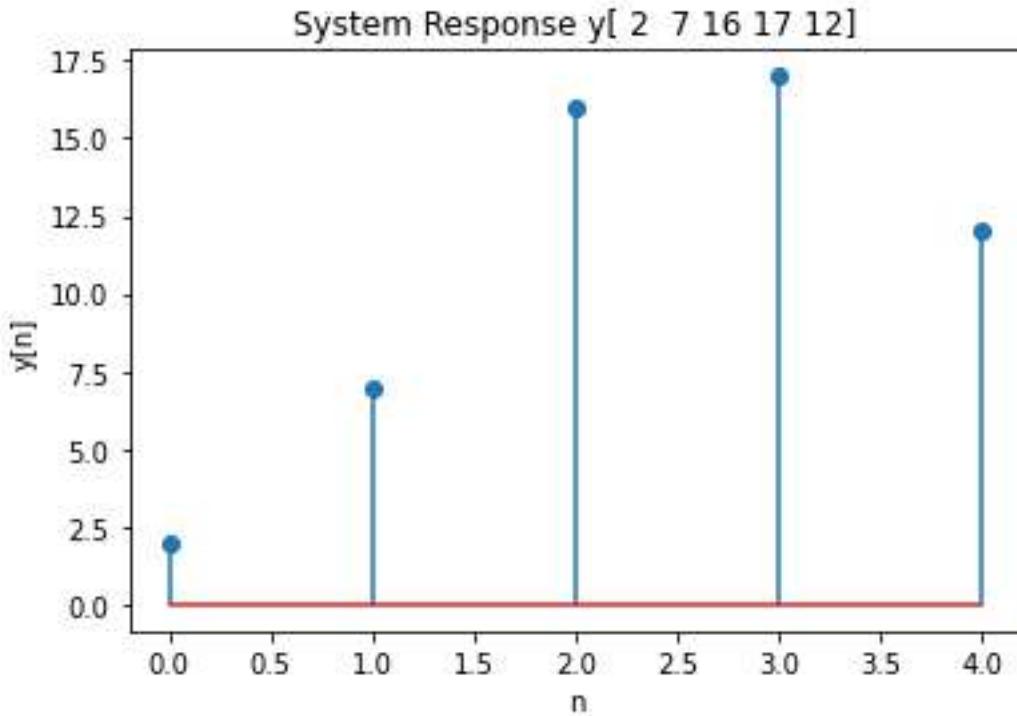


The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named 'Experiment 5.py' containing Python code for convolution. The code initializes input arrays x and h , pads them, and then uses nested loops to calculate the convolution sum $y[i] = \sum x[j] * h[i-j]$ for each iteration i . It also prints the intermediate results and shows the final system response y as a stem plot.

```

27 temp[1].insert(0,0) * int(x[1])
28 for n in range(3):
29     temp.append(0) # Adds required zeros at the end of list
30     for i in range(len(temp)-1,0,-1):
31         temp[i] = temp[i-1] # Shifts the zeros to the beginning of the list
32     temp[0] = 0 # Prepares for next iteration
33     plt.title('Intermediate plot: x[1] * h[n-1] = (' + str(i) + ', ' + str(temp) + ')')
34     plt.xlabel('n')
35     plt.ylabel('x[1] * h[n-1]')
36     plt.plot(np.arange(0, len(temp)), temp)
37     plt.show()
38
39 # 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.
40
41 size_x = (size_x + size_h) - 1 # Compute the size of system response
42 x = np.pad(x, (0, size_x - size_x), 'constant')
43 h = np.pad(h, (0, size_x - size_h), 'constant')
44 y = np.zeros(size, dtype = float) # Returns a new array of given shape and type, with zeros.
45 iteration = 1 # Variable used for displaying the iteration sequence
46
47 for i in range (size):
48     for j in range (size):
49         if i > j:
50             y[i] = int(x[j] * (int(x[i-j]) * int(h[j]))) # Compute convolution sum
51             print("Iteration " + str(iteration) + " : " + str(y)) # Display result of each iteration
52             iteration += 1
53 print("The system response is: " + str(y) + "\n")
54
55 plt.xlabel('n')
56 plt.ylabel('y[n]')
57 plt.title('System Response y[ ]'.format(y)) # Formats the specified value(s) and insert them inside the string's placeholder.
58 plt.stem(np.arange(0, size), y)
59 plt.show()
    
```

The right side of the interface shows a plot titled 'System Response y[2 7 16 17 12]' with the y-axis labeled 'y[n]' and the x-axis labeled 'n'. The plot shows discrete values at specific indices: (0, 2), (1, 7), (2, 16), (3, 17), and (4, 12). A red line at y=0 serves as the baseline.



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]
```

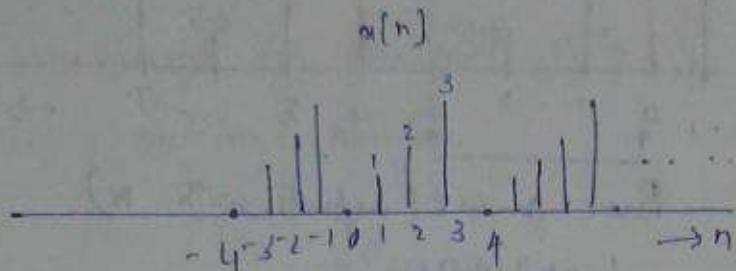
```
Iteration 14: [ 2 7 16 17 12]
```

```
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:

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

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

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

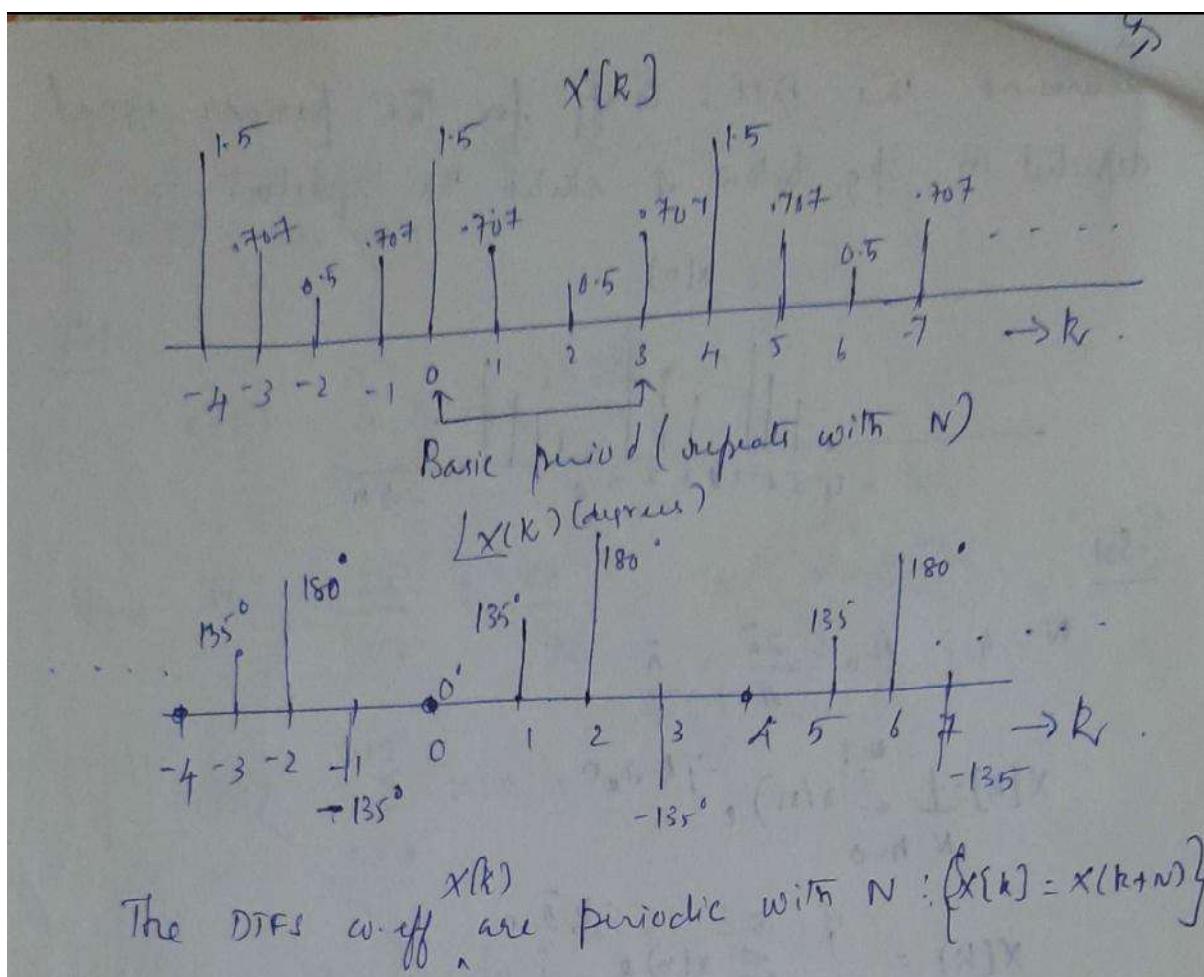
$$= \frac{1}{4} [x(0)e^0 + x(1)e^{-j \frac{\pi}{2}} + x(2)e^{-j \frac{\pi}{2} \cdot 2} + x(3)e^{-j \frac{\pi}{2} \cdot 3}]$$

$$= \frac{1}{4} [0 + 1 \cdot e^{-j \frac{\pi}{2}} + 2 \cdot e^{-j \frac{\pi}{2} \cdot 2} + 3 \cdot e^{-j \frac{\pi}{2} \cdot 3}] \quad \textcircled{1}$$

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

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	$x(k)$	$ x(k) $	$\angle x(k)$
0	1.5	1.5	0
1	-0.5 + j0.5	0.707	135° ($2 \cdot 357$ radians)
2	-0.5	0.5	180° (π rad)
3	-0.5 - j0.5	0.707	-135° ($-2 \cdot 357$ radians)



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

0
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 $f = \frac{1}{N}, \frac{2}{N}, \dots, \frac{(N-1)}{N}$ 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 -
$$e^{j2\pi kn/N}, k = 0, 1, \dots, N-1$$

The Fourier series is expressed as -

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

②

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,

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

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

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

$$\text{Angular frequency } (\omega_0) = \frac{2\pi}{N} = \frac{2\pi}{5}$$

$$\begin{aligned} x[k] &= \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-jk \frac{2\pi}{N} n} = \frac{1}{5} \sum_{n=0}^4 x[n] e^{-jk \frac{2\pi}{5} n} \\ &= \frac{1}{5} \left[x[0] e^0 + x[1] e^{-jk \frac{2\pi}{5}} + x[2] e^{-jk \frac{4\pi}{5}} \right. \\ &\quad \left. + x[3] e^{-jk \frac{6\pi}{5}} + x[4] e^{-jk \frac{8\pi}{5}} \right] \end{aligned}$$

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

Taking 5 common, we get -

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

(3)

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

$$x[0] = 15 + 0j, \text{ for } k=0$$

$$x[1] = -2.5 + 3.44j, \text{ for } k=1$$

$$x[2] = -2.5 + 0.81j, \text{ for } k=2$$

$$x[3] = -2.5 - 0.81j, \text{ for } k=3$$

$$x[4] = -2.5 - 3.44j, \text{ for } k=4$$

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 -

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

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

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

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

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



PROGRAM WITH COMMENTS:

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

```

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(user_defined_input) + "\n")
11 print("\nThe entered Input x[n] is : " + str(user_defined_input) + "\n")
12
13 # Null Array Declaration
14 x = []
15 fourier_series = []
16
17 # Compute Fundamental Time Period
18 for sample in range(1, size_input - 1):
19     x.append(user_defined_input[sample - 1])
20     if user_defined_input[0] == user_defined_input[sample]:
21         breakpoint = sample - 2 # How many times does the
22         sequence repeat?
23         break
24
25 # Obtain Frequency Domain Representation
26 boldface = int(input("Enter the element position that
27 indicates n=0 index : "))
28 index = int((boldface - 1) / len(x))
29
30 # Sketch Input Sequence
31 plt.xlabel('n')
32 plt.ylabel('x[n]')
33 plt.title('Input sequence : ... {3...}' .format(x))
34 plt.stem(np.arange(index * len(x)), len(x) * (breakpoint -
35 index)), x[breakpoint]

```

⑤

```

33 plt.show()
34
35 # Compute Fourier series
36 def summation():
37     sum = 0
38     for n in range(len(x)):
39         exponential = np.exp(-2j * np.pi * k * n / len(x))
40         sum = sum + float(x[n]) * exponential
41     return sum
42
43 # Compute Fourier series coefficients
44 for k in range(len(x)):
45     fourier_series.append(summation(k) / len(x))
46 print ("The Fourier Series Coefficients are as follows : { } .")
47 format(fourier_series)
48 # Compute Magnitude and Phase spectrum
49 magnitude_spectrum = []
50 phase_spectrum = []
51 for sample in range(len(fourier_series)):
52     magnitude_spectrum.append(np.sqrt(fourier_series[sample].real**2 + fourier_series[sample].imag**2) ** 0.5)
53     phase = np.arctan(fourier_series[sample].imag / fourier_series[sample].real)
54     phase_spectrum.append(phase)
55
56 # Sketch magnitude spectrum
57 plt.xlabel('k')
58 plt.ylabel('|x[k]|')
59 plt.title("Magnitude spectrum : ...{ }... .format(magnitude_spectrum))")
60 plt.stem(np.arange(0, len(magnitude_spectrum)), magnitude_spectrum)

```

⑥

```

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

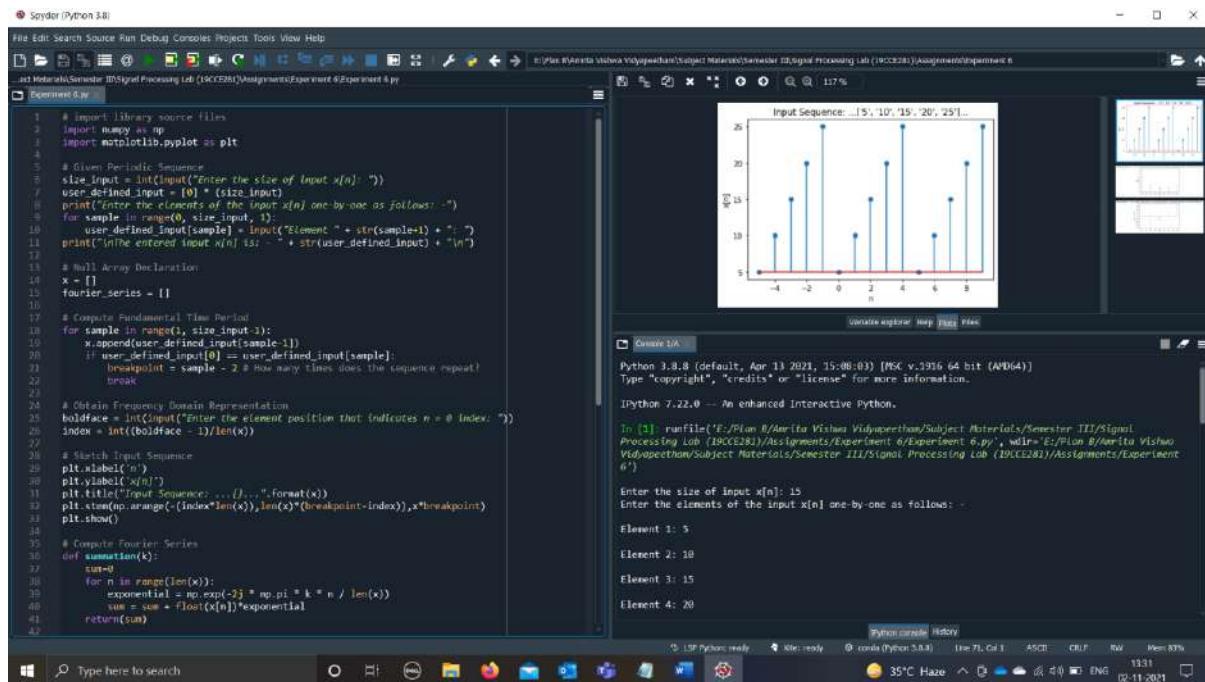
```

* INFERENCES :

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

RESULTS

VERIFIED

Given Discrete Periodic Sequence


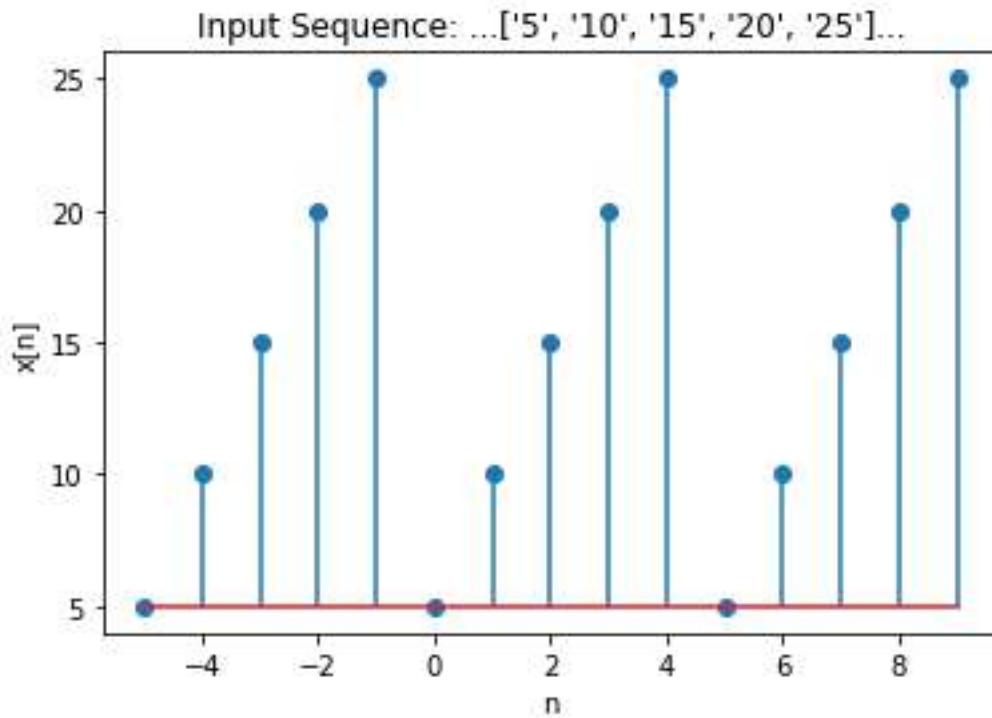
The screenshot shows the Spyder Python IDE interface. The code in the editor window is as follows:

```

1 # Import library source files
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Given Periodic Sequence
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("The entered input x[n] is: - " + str(user_defined_input) + "\n")
12
13 # Null Array Declaration
14 x = []
15 fourier_series = []
16
17 # Compute Fundamental Time Period
18 for sample in range(1, size_input-1):
19     x.append(user_defined_input[sample])
20     if user_defined_input[0] == user_defined_input[sample]:
21         breakpoint = sample - 2 # How many times does the sequence repeat?
22         break
23
24 # Obtain Frequency Domain Representation
25 boldface = int(input("Enter the element position that indicates n = 0 index: "))
26 index = int((boldface - 1)/len(x))
27
28 # Sketch Input Sequence
29 plt.xlabel('n')
30 plt.ylabel('x[n]')
31 plt.title('Input Sequence: ...[5, 10, 15, 20, 25]...')
32 plt.stem(np.arange(-index*len(x), len(x)*(breakpoint+index)), x*breakpoint)
33 plt.show()
34
35 # Compute Fourier Series
36 def summation(k):
37     sum=0
38     for n in range(len(x)):
39         exponential = np.exp(-2j * np.pi * k * n / len(x))
40         sum = sum + float(x[n])*exponential
41     return(sum)
42

```

The plot window shows a discrete periodic sequence with values 5, 10, 15, 20, and 25 at regular intervals. The x-axis is labeled 'n' and ranges from -4 to 8. The y-axis is labeled 'x[n]' and ranges from 5 to 25.



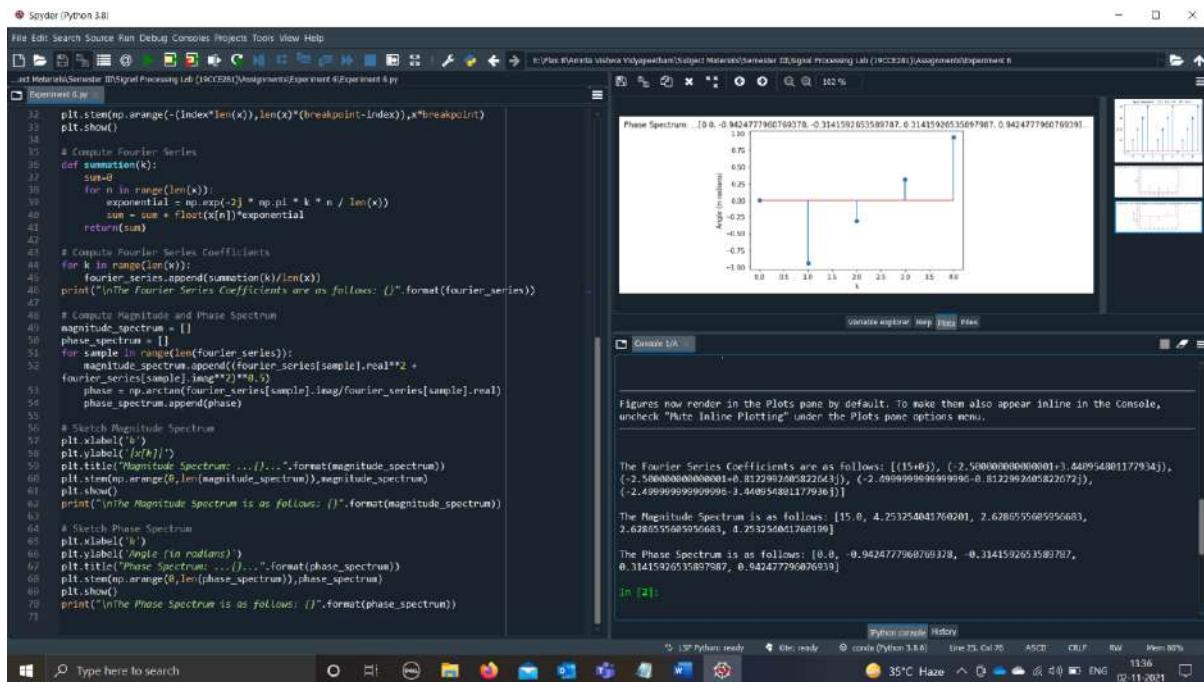
Step 1: Enter the size of input $x[n]$ and declare an array with the number of elements equal to the value of size.

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



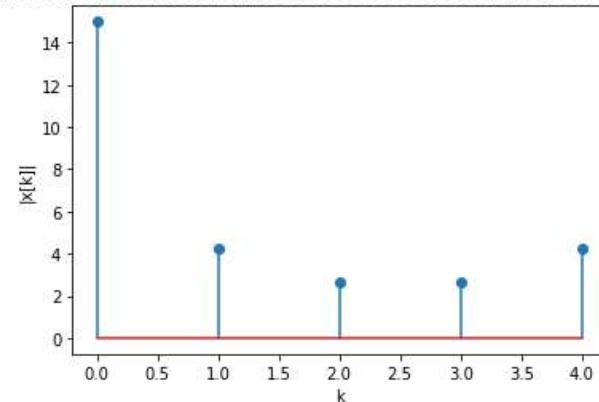
The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named 'Experiment 6.ipynb' containing Python code for signal processing tasks. The code includes functions for computing the Fourier series, magnitude, and phase spectra. On the right, there are two plots: one titled 'Phase Spectrum' showing discrete values for different k values, and another titled 'Magnitude Spectrum' showing discrete values for different k values. Below the plots, the IPython console window shows the printed results of the magnitude and phase spectra.

```

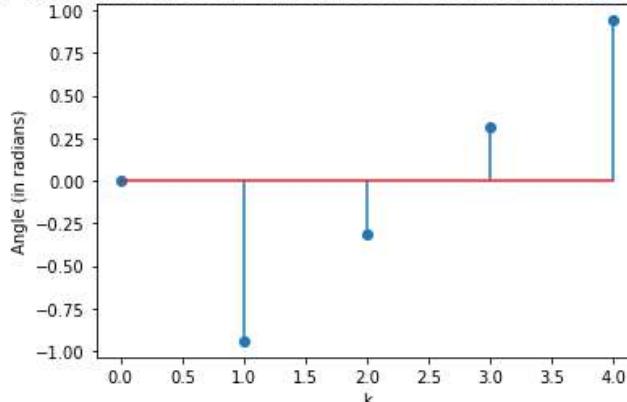
22 plt.stem(np.arange(-[index]*len(x), len(x)*(breakpoint-index)),x*breakpoint)
23 plt.show()
24
25 # Compute Fourier Series
26 def summation(k):
27     sum=0
28     for n in range(len(x)):
29         exponential = np.exp(-2j * np.pi * k * n / len(x))
30         sum = sum + float(x[n])*exponential
31     return(sum)
32
33 # Compute Fourier Series Coefficients
34 for k in range(len(x)):
35     fourier_series.append(summation(k)/len(x))
36 print("The Fourier Series Coefficients are as follows: {}".format(fourier_series))
37
38 # Compute Magnitude and Phase Spectrum
39 magnitude_spectrum = []
40 phase_spectrum = []
41 for sample in range(len(fourier_series)):
42     magnitude_spectrum.append(np.abs(fourier_series[sample]).real**2 +
43         fourier_series[sample].imag**2)**0.5
44     phase = np.arctan(fourier_series[sample].imag/fourier_series[sample].real)
45     phase_spectrum.append(phase)
46
47 # Sketch Magnitude Spectrum
48 plt.xlabel('k')
49 plt.ylabel('Angle (in radians)')
50 plt.title("Phase Spectrum: ...".format(phase_spectrum))
51 plt.stem(np.arange(0,len(phase_spectrum)),phase_spectrum)
52 plt.show()
53 print("The Phase Spectrum is as follows: {}".format(phase_spectrum))
54
55 # Sketch Magnitude Spectrum
56 plt.xlabel('k')
57 plt.ylabel('Angle (in radians)')
58 plt.title("Magnitude Spectrum: ...".format(magnitude_spectrum))
59 plt.stem(np.arange(0,len(magnitude_spectrum)),magnitude_spectrum)
60 plt.show()
61 print("The Magnitude Spectrum is as follows: {}".format(magnitude_spectrum))
62
63 # Sketch Phase Spectrum
64 plt.xlabel('k')
65 plt.ylabel('Angle (in radians)')
66 plt.title("Phase Spectrum: ...".format(phase_spectrum))
67 plt.stem(np.arange(0,len(phase_spectrum)),phase_spectrum)
68 plt.show()
69 print("The Phase Spectrum is as follows: [0.0, -0.9424777960769378, -0.3141592653589787, 0.3141592653589787, 0.942477796076939]...")
70
71

```

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



Phase Spectrum: ...[0.0, -0.9424777960769378, -0.3141592653589787, 0.3141592653589787, 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),
```

(-2.499999999999996-0.8122992405822672j), (-2.49999999999996-3.440954801177936j)]

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: PROPERTIES OF DTFS: TIME SHIFTING

Verify Time shifting property for the sequence $y[n] = x[n-2]$ where $x[n] = [\dots \mathbf{9} \ 18 \ 27 \ 36 \ 45 \ \dots]$ (*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 $y[n]$.

① NAME - SANTOSH - [CB.EN.UACLE20053]
DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING

LAB TITLE AND CODE : SIGNAL PROCESSING LAB IACLE281
EXPERIMENT NUMBER : 7
DATE : 02/10/2021

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 (Anaconda3) - Python 3.8.8 (64-bit)

* THEORY :

① Discrete-time signals -

A discrete-time signal of fundamental period N can consist of frequency components $f = \frac{1}{N}, \frac{2}{N}, \dots, \frac{(N-1)}{N}$ 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 -

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

The Fourier series is expressed as -

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

②

③ Fourier Coefficients -

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

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

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

$$x[n] = [\dots 9 18 27 36 45 \dots]$$

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

By keen observation, we find that fundamental period, N^5

$$\text{Angular frequency, } \omega_0 = \frac{2\pi}{N} = \frac{2\pi}{5}$$

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

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

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

Taking q common,

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

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

$$x[0] = 27 + 0^\circ, \text{ for } k=0$$

$$x[1] = -4.5 + 6.2^\circ, \text{ for } k=1$$

$$x[2] = -4.5 + 1.5^\circ, \text{ for } k=2$$

$$x[3] = -4.5 - 1.5^\circ, \text{ for } k=3$$

$$x[4] = -4.5 - 6.2^\circ, \text{ for } k=4$$

Upon computing the magnitude spectrum, we find that -

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

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

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

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

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

Upon computing the phase spectrum, we find that -

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

Therefore,

$$\arg(x[0]) = 0 \text{ radians}$$

$$\arg(x[1]) = -0.94 \text{ radians}$$

$$\arg(x[2]) = -0.31 \text{ radians}$$

$$\arg(x[3]) = 0.31 \text{ radians}$$

$$\arg(x[4]) = 0.94 \text{ radians}$$

(4)

Given shifting factor, $n_0 = 2$.

Therefore, obtained resulting sequence,

$$y[n] = [\dots \underline{36} \ 45 \ 9 \ 18 \ 27 \ \dots]$$

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

Again, fundamental period (N) = 5

$$\text{Angular frequency } (\omega_0) = \frac{2\pi}{N} = \frac{2\pi}{5}$$

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 -

$$\angle(y[0]) = \tan^{-1}\left(\frac{0}{27}\right) = 0^\circ = -\times \frac{\pi}{180} \text{ radians}$$

$$\angle(y[1]) = -0.31 \text{ radians}$$

$$\angle(y[2]) = 0.94 \text{ radians}$$

$$\angle(y[3]) = -0.94 \text{ radians}$$

$$\angle(y[4]) = 0.31 \text{ radians}$$

⑧

* 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
30         break

```

⑥

```

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

```

①

```

61 plt.show()
62
63 # Compute Magnitude and phase Spectrum for x[n]
64 for sample in range (len (fourier_series - x)):
65     magnitude_spectrum - x.append (fourier_series - x[sample])
66     real ** 2 + fourier_series - x[sample].Imag ** 2) ** 0.5)
67     phase = np.arctan (fourier_series - x[sample].Imag /
68     fourier_series - x[sample].real)
69     phase_spectrum - x.append (phase)
70
71 # sketch magnitude Spectrum of x[n]
72 plt.xlabel ('k')
73 plt.ylabel ('|x[k]|')
74 plt.title ("Magnitude Spectrum of x[n]: ... {{}}... ".format
75 (magnitude_spectrum - x))
76 plt.stem (np.arange (0, len (magnitude_spectrum - x)),
77 magnitude_spectrum - x)
78 plt.show ()
79 print ("The magnitude Spectrum of x[n] is as follows: {{}}".
80 format (magnitude_spectrum - x))
81
82 # sketch Phase Spectrum of x[n]
83 plt.xlabel ('k')
84 plt.ylabel ('Angle (in radians)')
85 plt.title ("Phase Spectrum of x[n]: ... {{}}... ".format
86 (phase_spectrum - x))
87 plt.stem (np.arange (0, len (phase_spectrum - x)),
88 phase_spectrum - x)
89 plt.show ()
90 print ("The phase spectrum of x[n] is as follows: {{}}".format
91 (phase_spectrum - x))
92
93 intermediate = []
94
95 intermediate = []
96

```

⑧

```

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

```

⑨

```

118 plt.title ("Resulting sequence: {{}}".format(y))
119 plt.stem(np.arange(0, index + len(y)), len(y) + breakpoints_index,
120          y + breakpoint)
121 plt.show()
122
123 # Compute Fourier coefficients for y[n]
124 for k in range (len(x)):
125     fourier_series_y.append(np.exp (-2j * np.pi * k *
126         shifting_factor / len(x)) * fourier_series_x[k])
127 print ("The Fourier coefficients for y[n] are as follows: {{}}".
128       format(fourier_series_y))
129
130 # sketch Fourier coefficients of y[n]
131 plt.xlabel ('n')
132 plt.ylabel ('y[n]')
133 plt.title ("Fourier coefficients of y[n]")
134 plt.stem(np.arange (0, len(fourier_series_y)),
135          np.real(fourier_series_y))
136 plt.show()
137
138 # Compute magnitude and Phase Spectrum for y[n]
139 for sample in range (len(fourier_series_y)):
140     magnitude_spectrum_y.append ((fourier_series_y[sample]
141         .real ** 2 + fourier_series_y[sample].imag ** 2) ** 0.5)
142     phase = np.arctan (fourier_series_y[sample].imag /
143                         fourier_series_y[sample].real)
144     phase_spectrum_y.append (phase)
145
146 # sketch magnitude Spectrum of y[n]
147 plt.xlabel ('k')
148 plt.ylabel ('|y[k]|')
149 plt.title ("Magnitude spectrum of y[n]: {{}}".format
150             (magnitude_spectrum_y))

```

(10)

- 144 plt. stem (np.arange(0, len(magnitude_spectrum_y)), magnitude_spectrum_y),
magnitude-spectrum-y)
- 145 plt. show()
- 146 print ("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_y), phase-spectrum
- 153 plt.show()
- 154 print ("The Phase Spectrum of y[n] is as follows: {} ".format(phase_spectrum_y))
- 155

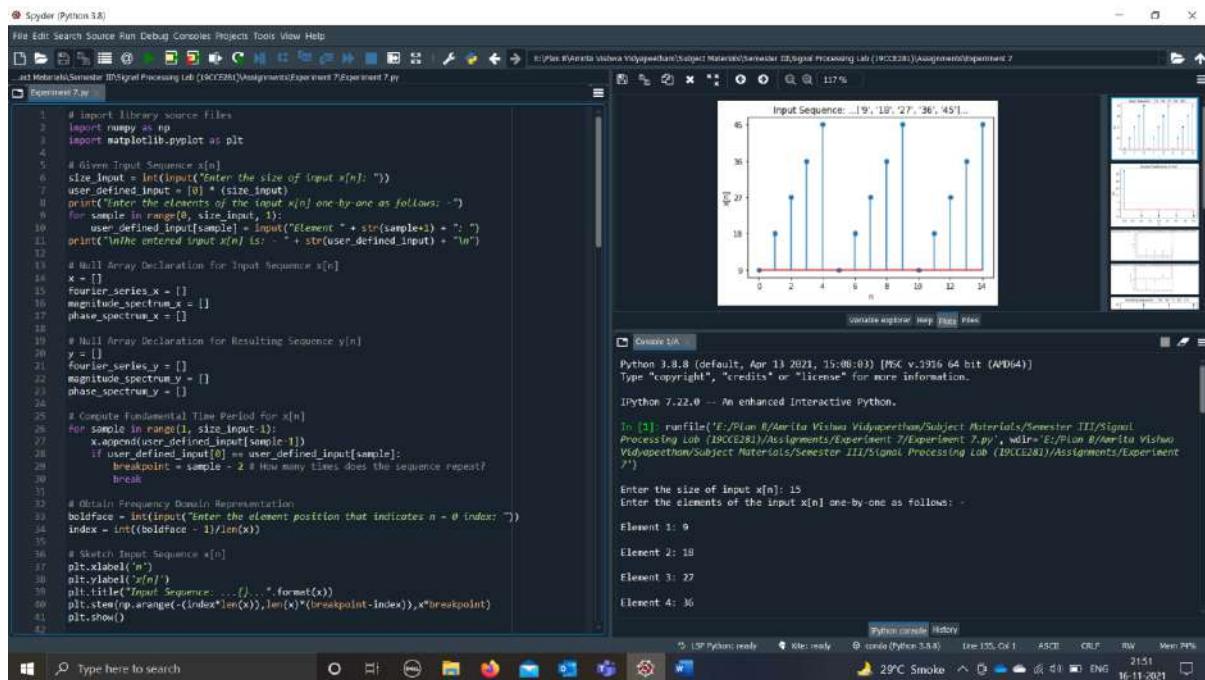
*

INFERENCES:

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.

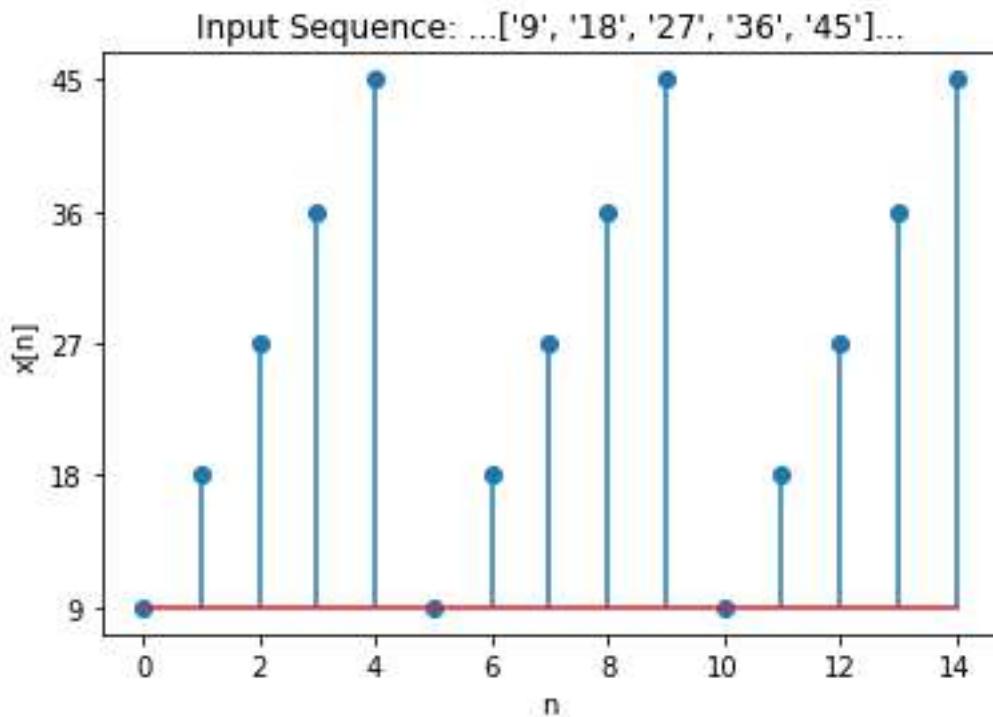
Given Input Sequence


The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a Python script named 'Experiment 7.py'. The code implements a function to plot a discrete-time signal. It starts by importing numpy and matplotlib.pyplot. It then prompts the user for the size of the input sequence and initializes arrays for the input and output. It loops through the user-defined input to build the input array. The plot window on the right shows a discrete-time signal x[n] plotted against n. The x-axis ranges from 0 to 14, and the y-axis ranges from 9 to 45. The signal consists of vertical spikes at specific indices: n=0, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14. The values of these spikes are 9, 18, 36, 45, 9, 27, 36, 45, 18, 27, 36, 45 respectively. The plot title is 'Input Sequence: ...[9, 18, 27, 36, 45]...'. The bottom status bar shows the Python version (3.8.8), system information (Windows 10 Pro), and current time (16-11-2021).

```

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("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])
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+1, len(x)), [x]*len(x), x[breakpoint])
41 plt.show()
42

```



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

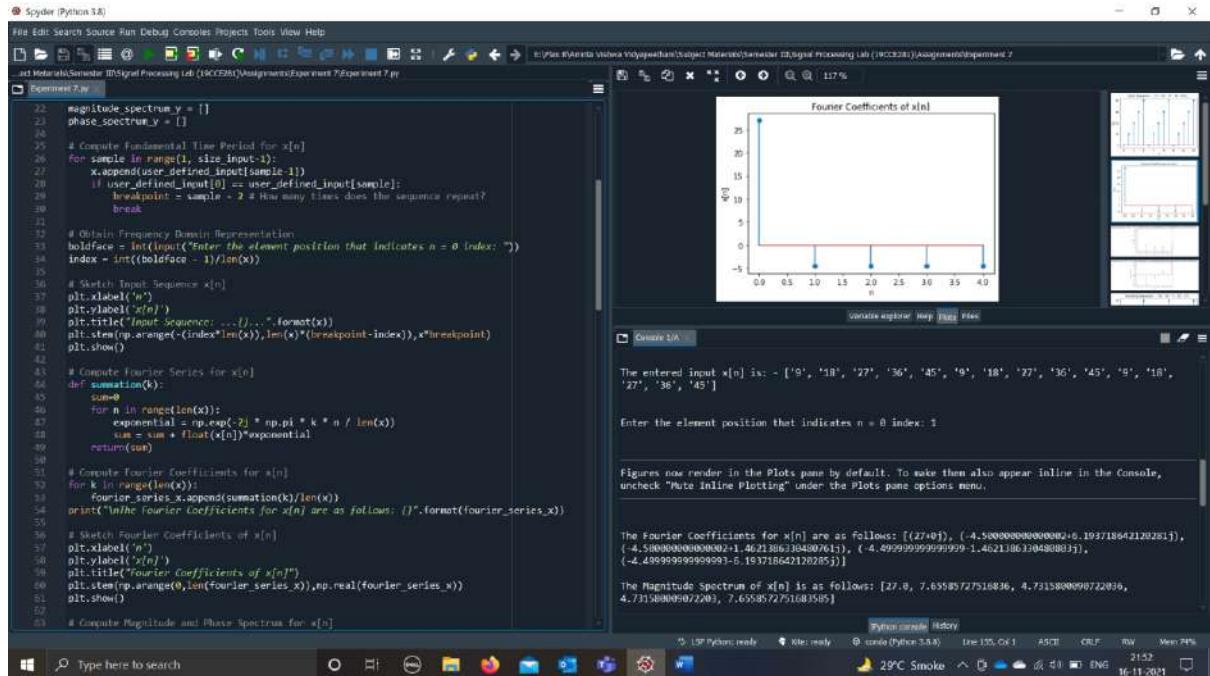
Step 2: Enter the size of the input and declare an array with the number of elements equal to the value of size.

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 $x[n]$


The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named 'Experiment 7.py' containing Python code for signal processing. The code includes functions for generating input sequences, computing Fourier series, and plotting magnitude and phase spectra. On the right, the IPython console window shows the execution of the script. It prompts for the input sequence, which is provided as a list of integers: [9, 18, 27, 36, 45, 9, 18, 27, 36, 45]. The console then asks for the element position that indicates $n = 0$, and the user inputs index 1. The output shows the Fourier coefficients for $x[n]$ and the magnitude spectrum of $x[n]$.

```

# Compute Fundamental Time Period for x[n]
for sample in range(1, size_input-1):
    x.append(user_defined_input[sample])
    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 x[n]
plt.xlabel('n')
plt.ylabel('x[n]')
plt.title('Input Sequence: ...'.format(x))
plt.stem(np.arange(-(index)*len(x)), np.real(x)*(breakpoint-index), x*breakpoint)
plt.show()

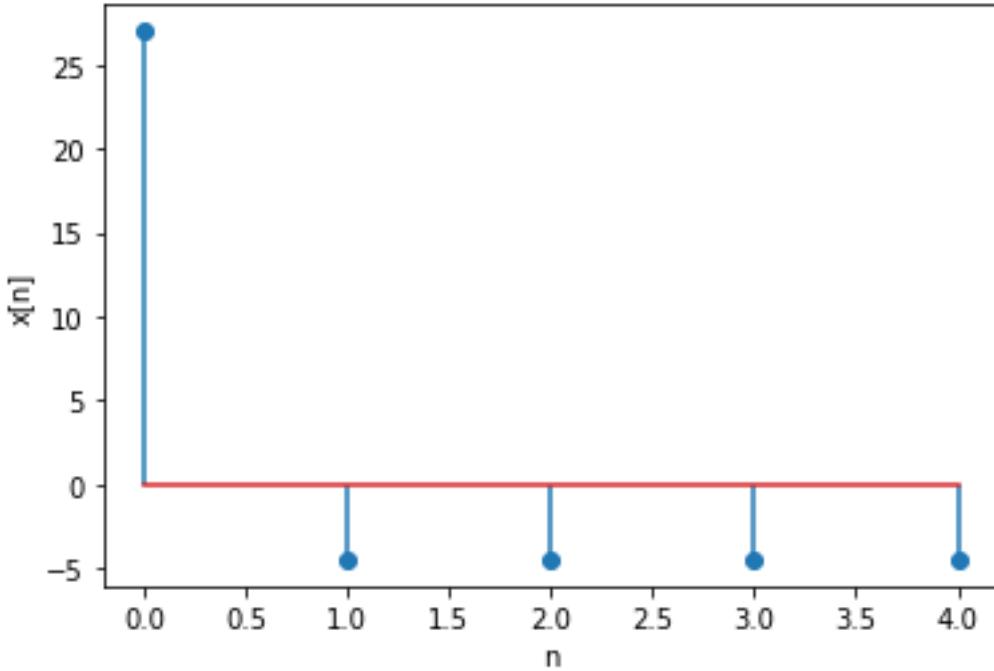
# Compute Fourier Series for x[n]
def summation(k):
    sum=0
    for n in range(len(x)):
        exponential = np.exp(-j * np.pi * k * n / len(x))
        sum = sum + float(x[n])*exponential
    return(sum)

# Compute Fourier Coefficients for x[n]
for k in range(len(x)):
    fourier_series_x.append(summation(k)/len(x))
print("The Fourier Coefficients for x(n) are as follows: {}".format(fourier_series_x))

# Sketch Fourier Coefficients of x[n]
plt.xlabel('n')
plt.ylabel('x[n]')
plt.title('Fourier Coefficients of x[n]')
plt.stem(np.arange(0,len(fourier_series_x)),np.real(fourier_series_x))
plt.show()

# Compute Magnitude and Phase Spectra for x[n]

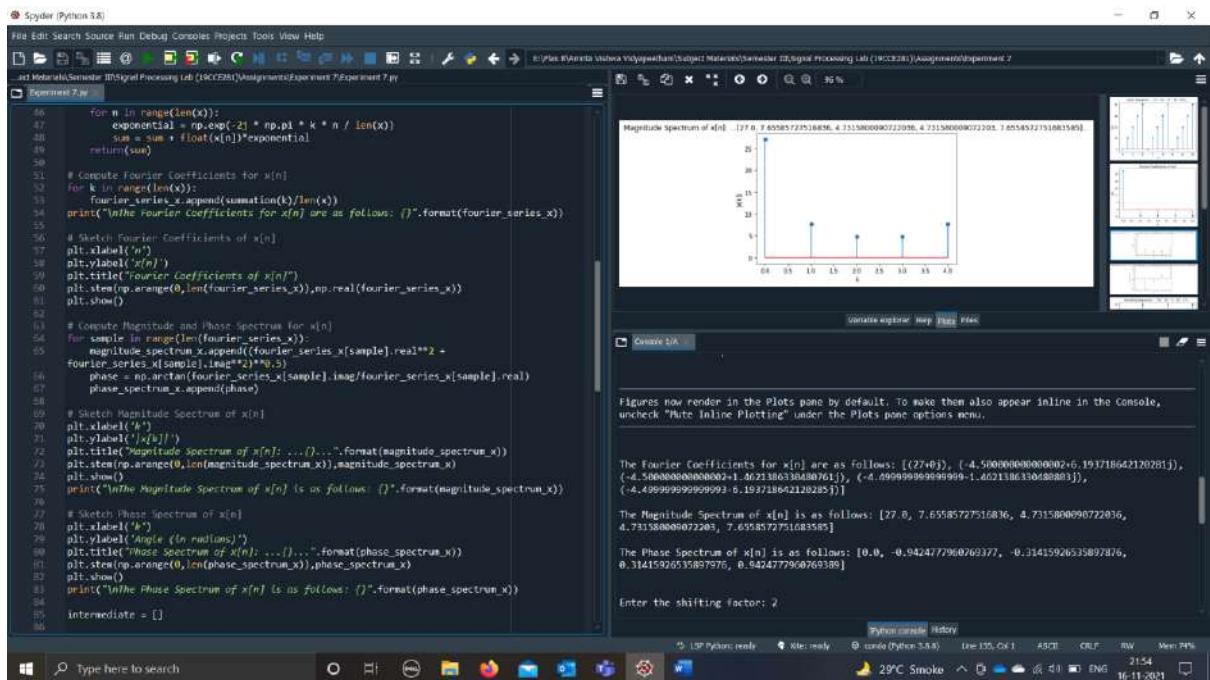
```

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 $x[n]$


```

# Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
art Materials Semester 10 Signal Processing Lab (19CCE20053) Assignments Experiment 7/Experiment 7.py
Experiment 7.py
1 # Compute exponential sum for x[n]
2 for n in range(len(x)):
3     exponential = np.exp(-2j * np.pi * k * n / len(x))
4     sum = sum + float(x[n])*exponential
5
6 return(sum)
7
8 # Compute Fourier Coefficients for x[n]
9 for k in range(len(x)):
10    fourier_series_x.append(summation(k)/len(x))
11    print("The Fourier Coefficients for x[n] are as follows: {}".format(fourier_series_x))
12
13 # Sketch Fourier Coefficients of x[n]
14 plt.xlabel('n')
15 plt.ylabel('x[n]')
16 plt.title('Fourier Coefficients of x[n]')
17 plt.stem(np.arange(0,len(fourier_series_x)),np.real(fourier_series_x))
18 plt.show()
19
20 # Compute Magnitude and Phase Spectrums for x[n]
21 for sample in range(len(fourier_series_x)):
22    magnitude_spectrum_x.append(np.sqrt(fourier_series_x[sample].real**2 +
23        fourier_series_x[sample].imag**2))
24    phase = np.arctan(fourier_series_x[sample].imag/fourier_series_x[sample].real)
25    phase_spectrum_x.append(phase)
26
27 # Sketch Magnitude Spectrum of x[n]
28 plt.xlabel('k')
29 plt.ylabel('|x[k]|')
30 plt.title('Magnitude Spectrum of x[n]: ...'.format(magnitude_spectrum_x))
31 plt.stem(np.arange(0,len(magnitude_spectrum_x)),magnitude_spectrum_x)
32 plt.show()
33 print("The Magnitude Spectrum of x[n] is as follows: {}".format(magnitude_spectrum_x))
34
35 # Sketch Phase Spectrum of x[n]
36 plt.xlabel('k')
37 plt.ylabel('Angle (in radians)')
38 plt.title('Phase Spectrum of x[n]: ...'.format(phase_spectrum_x))
39 plt.stem(np.arange(0,len(phase_spectrum_x)),phase_spectrum_x)
40 plt.show()
41 print("The Phase Spectrum of x[n] is as follows: {}".format(phase_spectrum_x))
42 intermediate = []

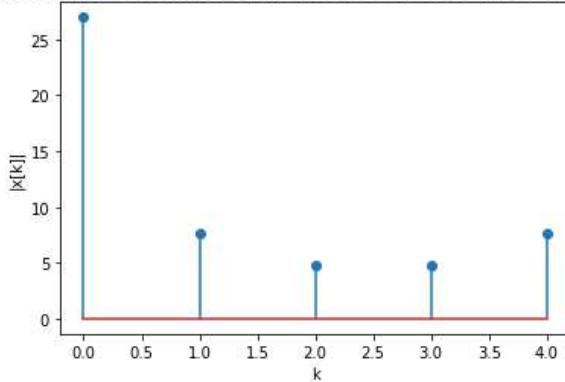
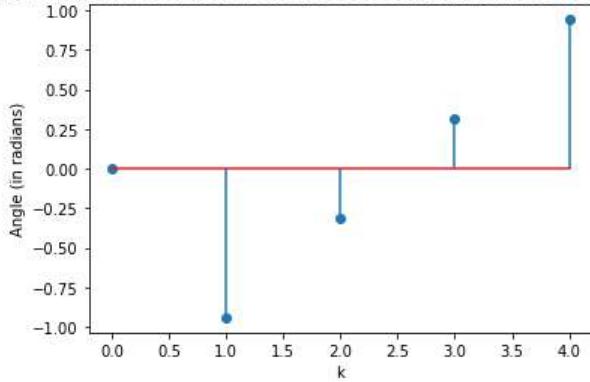
```

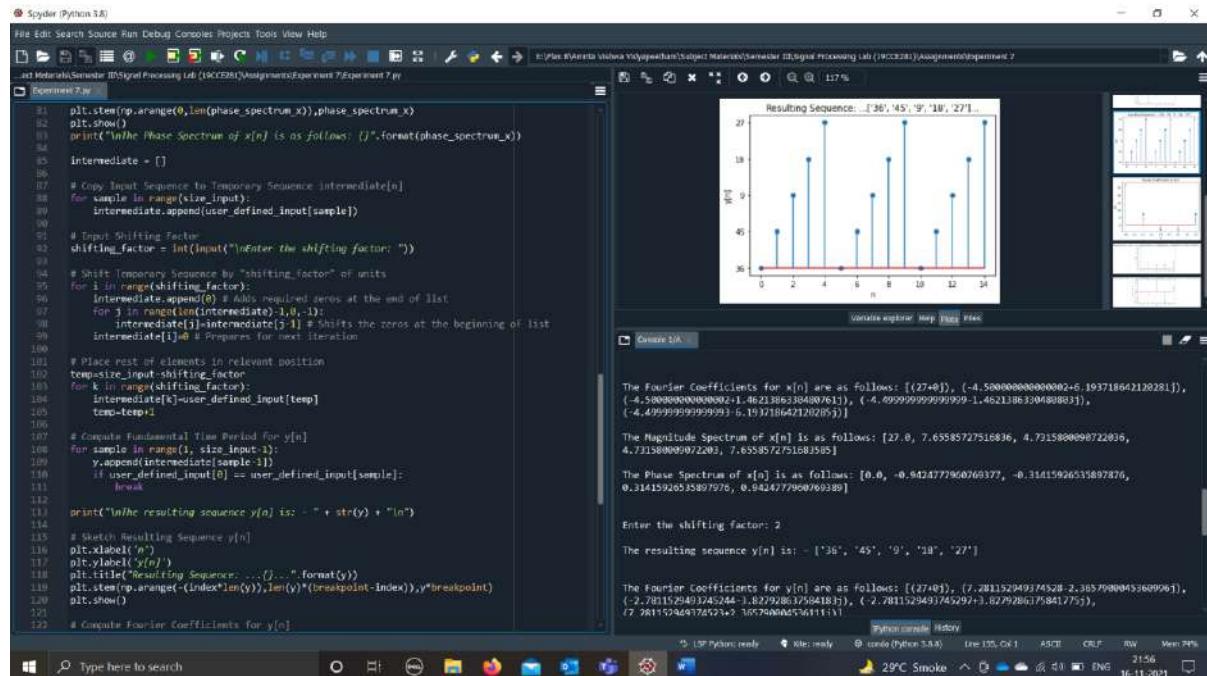
The Fourier Coefficients for $x[n]$ are as follows: [(27.0), (-4.5000000000000002+6.1937186422081j), (-4.5000000000000002+1.462130330480761j), (-4.899999999999999-1.402130633048083j), (-4.499999999999999-6.19371864220825j)]

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

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 – $y[n]$


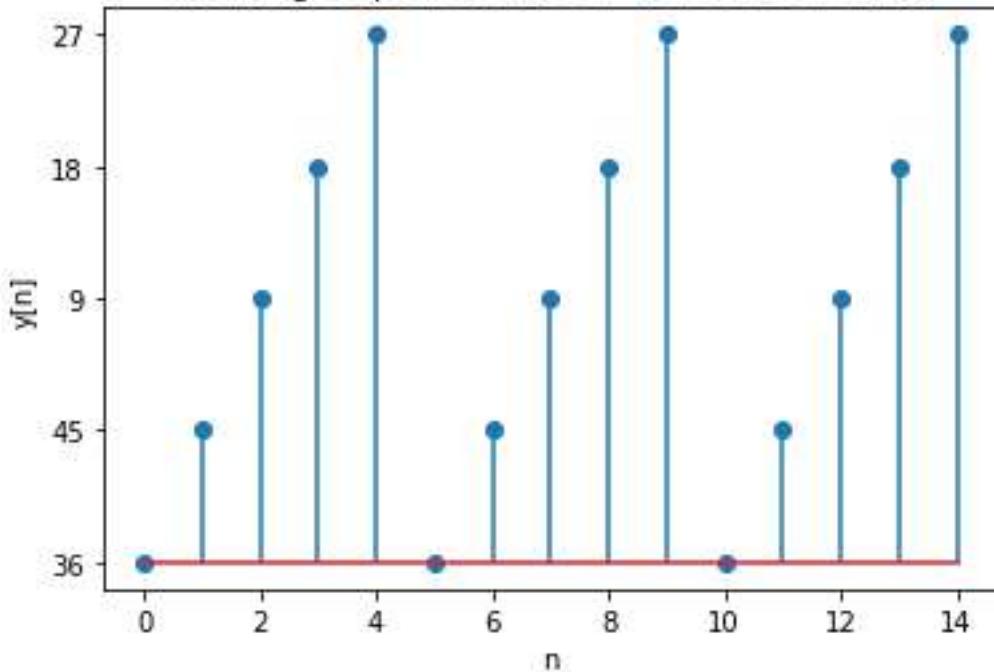
The screenshot shows the Spyder Python IDE interface. On the left, the code for Experiment 7 is displayed:

```

1 #!/usr/bin/python
2 # Experiment 7.py
3
4 import numpy as np
5 import matplotlib.pyplot as plt
6
7 print("The Phase Spectrum of x[n] is as follows: " + str(phase_spectrum_x))
8
9 intermediate = []
10
11 # Copy Input Sequence to Temporary Sequence intermediate[n]
12 for sample in range(size_input):
13     intermediate.append(user_defined_input[sample])
14
15 # Input Shifting Factor
16 shifting_factor = int(input("\nEnter the shifting factor: "))
17
18 # Shift Temporary Sequence by "shifting_factor" of units
19 for i in range(shifting_factor):
20     intermediate.append(0) # Adds required zeros at the end of list
21
22 for j in range(len(intermediate)-1,0,-1):
23     intermediate[j]=intermediate[j-1] # Shifts the zeros at the beginning of list
24     intermediate[0]=0 # Prepares for next iteration
25
26 # Place rest of elements in relevant position
27 temp=intermediate[0]
28 for k in range(shifting_factor):
29     intermediate[k]=user_defined_input[temp]
30     temp+=1
31
32 # Compute Fundamental Time Period for y[n]
33 for sample in range(1, size_input-1):
34     y.append(intermediate[sample-1])
35     if user_defined_input[0] == user_defined_input[sample]:
36         break
37
38 print("The resulting sequence y[n] is: " + str(y) + "\n")
39
40 # Sketch Resulting Sequence y[n]
41 plt.xlabel('n')
42 plt.ylabel('y[n]')
43 plt.title('Resulting Sequence: ...' + str(y[-5:]) + '... ')
44 plt.ylim(36, 27)
45 plt.yticks([36, 45, 9, 18, 27])
46 plt.plot(np.arange(-len(y), len(y)), y)
47 plt.show()
48
49 # Compute Fourier Coefficients for y[n]
50

```

The right side of the interface shows the resulting sequence plot titled "Resulting Sequence: ...[36, 45, 9, 18, 27]...". The x-axis is labeled "n" and ranges from 0 to 14. The y-axis is labeled "y[n]" and ranges from 36 to 27. The plot shows discrete values at specific indices: (0, 36), (1, 45), (2, 9), (3, 18), (4, 27), (5, 36), (6, 45), (7, 9), (8, 18), (9, 27), (10, 36), (11, 45), (12, 9), (13, 18), (14, 27).

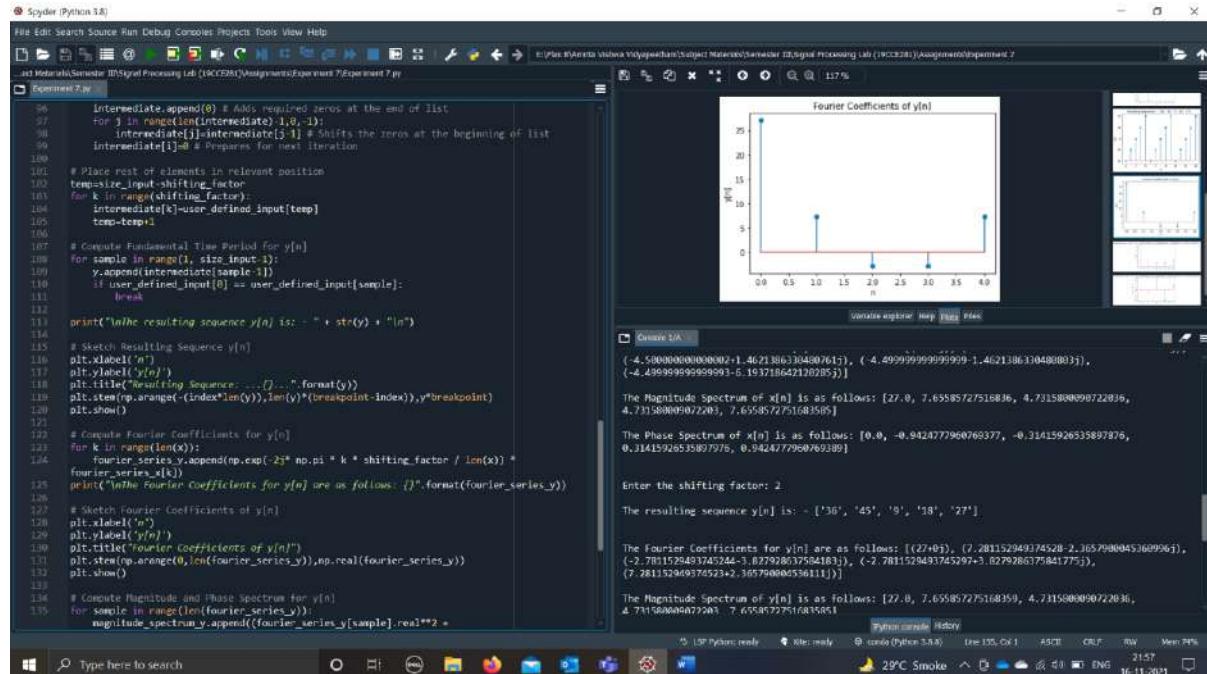
Resulting Sequence: ...[36, '45', '9', '18', '27']...

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.

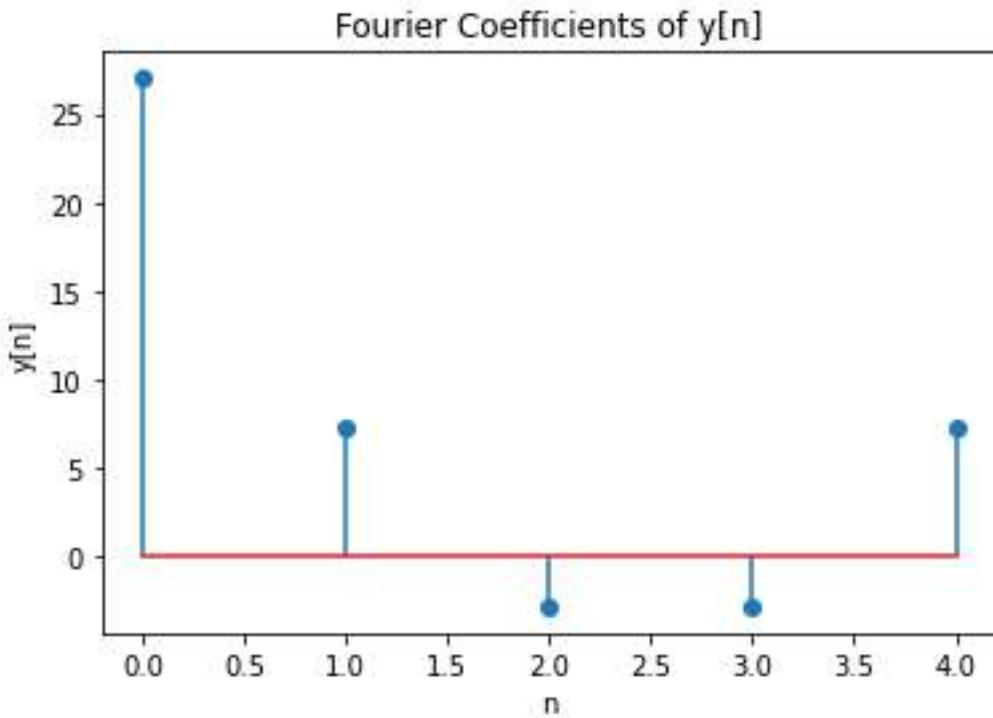
Step 3: Place the rest of the elements in the relevant position and compute the fundamental time period for $y[n]$.

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

Plot Fourier Coefficients of $y[n]$


```

# Spyder (Python 3.8)
File Edit Search Source Run Debug Console Projects Tools View Help
art Materials/Semester 10/Signal Processing Lab (19CCE20053)/Assignments/Experiment 7/Experiment 7.py
Experiment 7.py
113 intermediate.append(0) # Adds required zeros at the end of list
114 for i in range(len(intermediate)-1,0,-1):
115     intermediate[i]=intermediate[i-1] # Shifts the zeros at the beginning of list
116     intermediate[i]=0 # Prepares for next iteration
117
118 # Place rest of elements in relevant position
119 temp=1/input_shifting_factor
120 for k in range(input_shifting_factor):
121     intermediate[k]=user_defined_input[temp]
122     temp+=1
123
124 # Compute Fundamental Time Period for  $y[n]$ 
125 for sample in range(1, size_input+1):
126     y.append(intermediate[sample-1])
127     if user_defined_input[0] == user_defined_input[sample]:
128         break
129
130 print("The resulting sequence  $y[n]$  is: " + str(y) + "\n")
131
132 # Sketch Resulting Sequence  $y[n]$ 
133 plt.xlabel('n')
134 plt.ylabel('y[n]')
135 plt.title("Resulting Sequence: ... " + "...".format(y))
136 plt.stem(np.arange(-len(y)+1, len(y)*(breakpoint_index)), y*breakpoint)
137 plt.show()
138
139 # Compute Fourier Coefficients for  $y[n]$ 
140 for K in range(len(x)):
141     fourier_series_y.append(np.exp(-2j * np.pi * k * shifting_factor / len(x)) *
142     y[K])
143 fourier_series_y[K]
144
145 print("The Fourier Coefficients for  $y[n]$  are as follows: " + str(fourier_series_y))
146
147 # Sketch Fourier Coefficients of  $y[n]$ 
148 plt.xlabel('n')
149 plt.ylabel('y[n]')
150 plt.title("Fourier Coefficients of  $y[n]$ ")
151 plt.stem(np.arange(0, len(fourier_series_y)), np.real(fourier_series_y))
152 plt.show()
153
154 # Compute Magnitude and Phase spectrum for  $y[n]$ 
155 for sample in range(len(fourier_series_y)):
156     magnitude_spectrum_y.append((fourier_series_y[sample].real)**2 +
157
158                                         Fourier Coefficients of y(n)
159                                         n
160                                         y[n]
161                                         0 0.5 1 1.5 2 2.5 3 3.5 4
162                                         0 5 10 15 20 25
163                                         -4.500000000000002, -1.462136339400761j, (-4.499999999999999-5.193718642120285j),
164                                         (-4.499999999999999-5.193718642120285j)
165                                         The Magnitude Spectrum of  $x[n]$  is as follows: [27.0, 7.65585727516836, 4.715800099722836,
166                                         4.715800099722836, 7.65585727516835]
167                                         The Phase Spectrum of  $x[n]$  is as follows: [0.0, -0.9424777968769377, -0.31415926535897876,
168                                         0.3141592653589786, 0.9424777968769389]
169                                         Enter the shifting factor: 2
170                                         The resulting sequence  $y[n]$  is: -['36', '45', '9', '38', '27']
171                                         The Fourier Coefficients for  $y[n]$  are as follows: [(27+0j), (7.28115294917452j-2.1657900045360996j),
172                                         (-2.78115294917452j-3.82792863794181j), (-2.78115294917452j+3.827928637584177j),
173                                         (7.28115294917452j+2.36579000536111j)]
174                                         The Magnitude Spectrum of  $y[n]$  is as follows: [27.0, 7.655857275168359, 4.715800099722836,
175                                         4.715800099722836, 7.655857275168359]
176                                         Python console History
177                                         0 100% Python ready 0 100% Python 3.8 0 100% the ISS OK! ASCE CRU RW Men 2157 29°C Smoke 29°C Smoke 16-11-2021
```



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

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

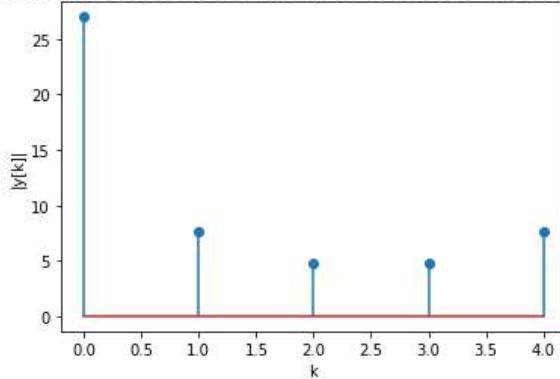
Plot Magnitude and Phase Spectrum of $y[n]$

```

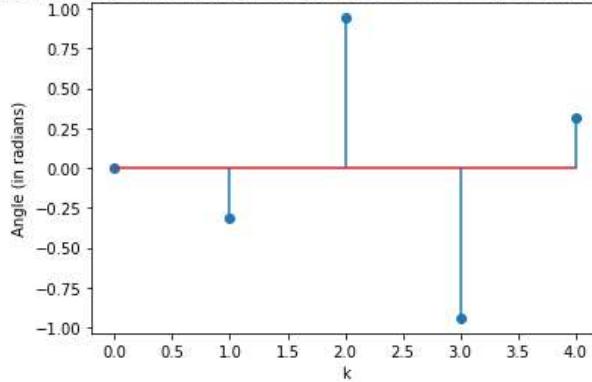
1.7 plt.ylabel('y[n]')
1.8 plt.title('Resulting Sequence: ...')
1.9 plt.stem(np.arange((index*len(y)), len(y)*(breakpoint-index)), y*breakpoint)
2.0 plt.show()
2.1
2.2 # Compute Fourier Coefficients for y[n]
2.3 for k in range(len(x)):
2.4     fourier_series_y.append(np.exp(-j*2*pi*k * shifting_factor / len(x)) *
2.5         fourier_series_x[k])
2.6 print("The Fourier Coefficients for y[n] are as follows: {}".format(fourier_series_y))
2.7
2.8 # Sketch Fourier Coefficients of y[n]
2.9 plt.xlabel('n')
3.0 plt.ylabel('y[n]')
3.1 plt.title('Fourier Coefficients of y[n]')
3.2 plt.stem(np.arange(0, len(fourier_series_y)), np.real(fourier_series_y))
3.3 plt.show()
3.4
3.5 # Compute Magnitude Spectrum for y[n]
3.6 for sample in range(0, len(fourier_series_y)):
3.7     magnitude_spectrum_y.append(np.sqrt(fourier_series_y[sample].real**2 +
3.8         fourier_series_y[sample].imag**2)*0.5)
3.9     phase = np.arctan(fourier_series_y[sample].imag/fourier_series_y[sample].real)
3.10     phase_spectrum_y.append(phase)
3.11
3.12 # Sketch Magnitude Spectrum of y[n]
3.13 plt.xlabel('k')
3.14 plt.ylabel('|y[k]|')
3.15 plt.title('Magnitude Spectrum of y[n]: ...'.format(magnitude_spectrum_y))
3.16 plt.stem(np.arange(0, len(magnitude_spectrum_y)), magnitude_spectrum_y)
3.17 plt.show()
3.18 print("The Magnitude Spectrum of y[n] is as follows: {}".format(magnitude_spectrum_y))
3.19
3.20 # Sketch Phase Spectrum of y[n]
3.21 plt.xlabel('k')
3.22 plt.ylabel('Angle (in radians)')
3.23 plt.title('Phase Spectrum of y[n]: ...'.format(phase_spectrum_y))
3.24 plt.stem(np.arange(0, len(phase_spectrum_y)), phase_spectrum_y)
3.25 plt.show()
3.26 print("The Phase Spectrum of y[n] is as follows: {}".format(phase_spectrum_y))

```

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, 0.31415926535898075]...



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.V4(CIE20053)]

DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CIE) A

LAB TITLE AND CODE : SIGNAL PROCESSING LAB 19CIE251

EXPERIMENT NUMBER : 8

DATE : 23/10/2024

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 -

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

②

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 -

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

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)

③

```

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]')

```

```

38 plt.title('Unit Step Discrete-Time Function for  $u[n-3] - u[n]$ ')
39 plt.stem(n, result)
40 plt.show()
41
42 temp = [] #  $(u[n-3] - u[n]) * 0.5^n$  Null List Declaration
43 for sample in range(len(n)):
44     temp.append((b[sample] - a[sample]) * 0.5 ** sample)
45 plt.xlabel('n')
46 plt.ylabel('( $u[n-3] - u[n]$ ) * 0.5^n')
47 plt.title('Impulse Response h[n]')
48 plt.stem(n, temp)
49 plt.show()
50
51 # Compute Frequency Response
52 frequency_response = []
53 def summation(f):
54     sum = 0
55     for k in range(len(temp)):
56         exponential = np.exp(-2j * np.pi * k * f)
57         sum = sum + float(temp[k]) * exponential
58     return sum
59
60 for w in range(len(temp)):
61     frequency_response.append(summation(w))
62
63 # Sketch Frequency Response
64 plt.xlabel('w')
65 plt.ylabel('h(w)')
66 plt.title("Frequency Response")
67 plt.plot(np.arange(0, len(frequency_response)),
68          np.real(frequency_response))
69 plt.show()
70 print("The Frequency Response of h[n] is as follows : ")
71 print(format(frequency_response))

```

⑤

```

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],
77         real ** 2 + frequency-response[sample].imag ** 2) ** 0.5)
78     phase = np.arctan (frequency-response[sample].imag /
79         frequency-response[sample].real)
80     phase-spectrum.append (phase)
81
82 # sketch Magnitude Spectrum
83 plt.xlabel ('w')
84 plt.ylabel ('|h(w)|')
85 plt.title ("Magnitude spectrum : ... {}... ".format
86             (magnitude-spectrum))
87 plt.plot (np.arange (0, len (magnitude-spectrum)),
88             magnitude-spectrum)
89 plt.show ()
90 print ("The Magnitude spectrum of h[n] is as follows : {}".
91       format (magnitude-spectrum))
92
93 # sketch Phase Spectrum
94 plt.xlabel ('w')
95 plt.ylabel ('Angle (in radians)')
96 plt.title ("Phase spectrum \phi(w)")
97 plt.plot (np.arange (0, len (phase-spectrum)), phase-spectrum)
98 plt.show ()
99 print ("The Phase spectrum of h[n] is as follows : {}".
100       format (phase-spectrum))

```

* INFEERENCE :

For the given LTI system characterized by the Impulse response $h[n]$, compute frequency response $H(\omega)$ and plot the same along with magnitude and phase spectrums.

RESULTS

VERIFIED:

Unit Step Discrete-Time Functions

The screenshot shows the Spyder Python IDE interface. On the left is the code editor with the following Python script:

```

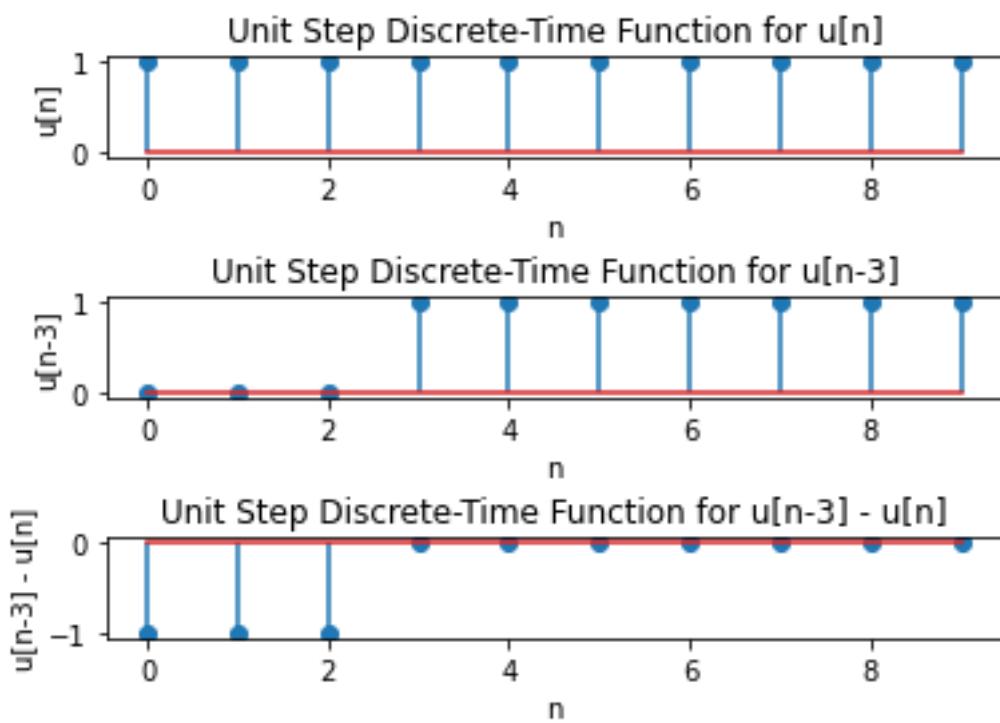
1 import numpy as np
2 import matplotlib.pyplot as plt
3 
4 n = np.arange(0,10) # Get evenly spaced values within the interval [0,10]
5 
6 a=[ ] # u[n] Null List Declaration
7 for sample in range(len(n)):
8     if n[sample]>=0:
9         temp1
10    else:
11        temp0
12        a.append(temp0)
13    plt.subplot (5,2,1)
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         temp1
23     else:
24         temp0
25     b.append(temp0)
26    plt.subplot (5,2,2)
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 results=[ ] # 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 (5,2,3)
36    plt.xlabel('n')
37    plt.ylabel('u[n-3] - u[n]')
38    plt.title('Unit Step Discrete-Time Function for u[n-3] - u[n]')
39    plt.stem(n,result)
40    plt.show()
41 
42 temp=[ ] # (u[n-3] - u[n])*0.5^n Null List Declaration
43

```

The right side of the interface shows three plots generated by the code:

- Unit Step Discrete-Time Function for $u[n]$:** A plot showing a unit step function where $u[n] = 1$ for $n \geq 0$ and $u[n] = 0$ for $n < 0$.
- Unit Step Discrete-Time Function for $u[n-3]$:** A plot showing a unit step function where $u[n-3] = 1$ for $n \geq 3$ and $u[n-3] = 0$ for $n < 3$.
- Unit Step Discrete-Time Function for $u[n-3] - u[n]$:** A plot showing the difference between the two functions, which is 1 for $n < 3$ and -1 for $n \geq 3$.

Below the plots, the terminal window shows the command runfile and its output.

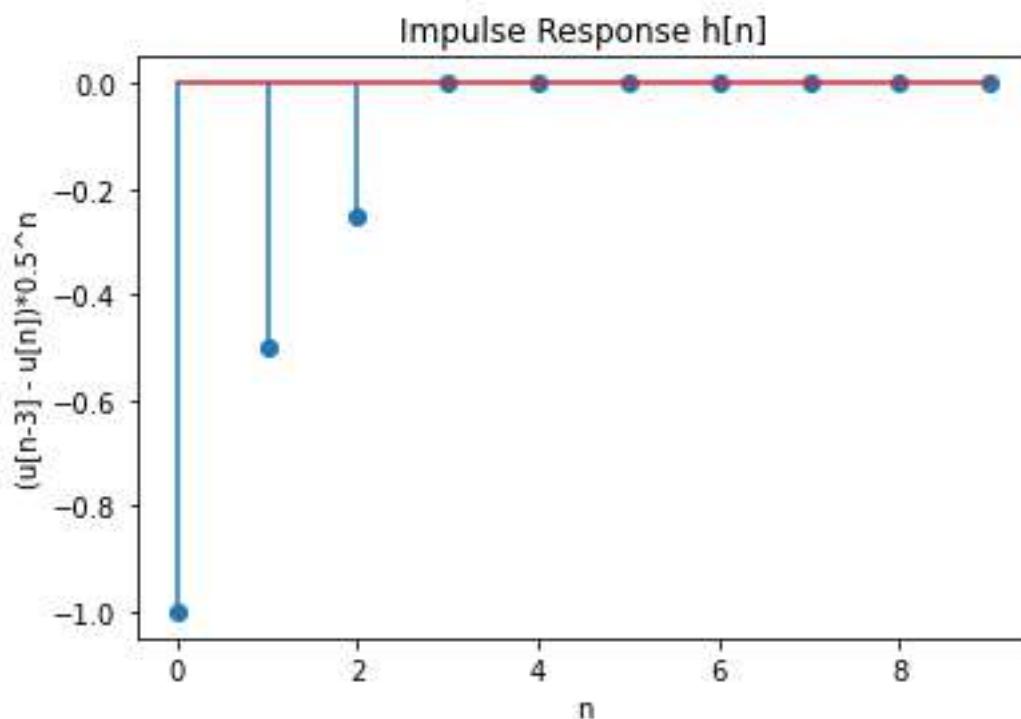
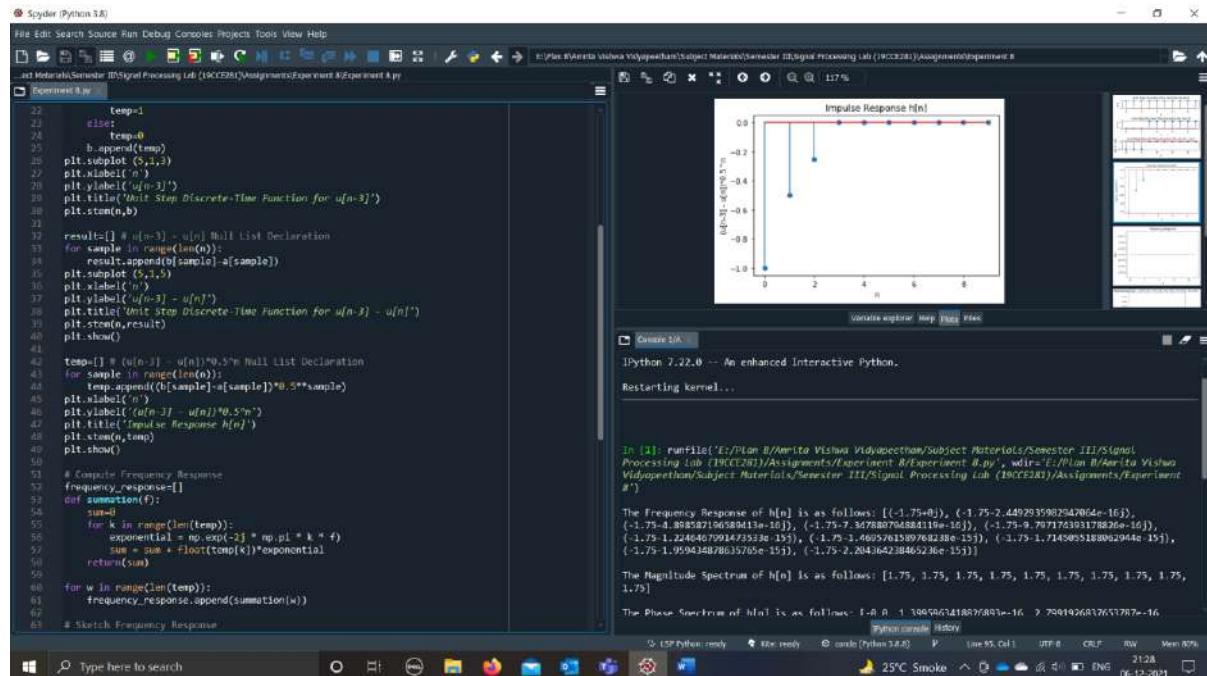


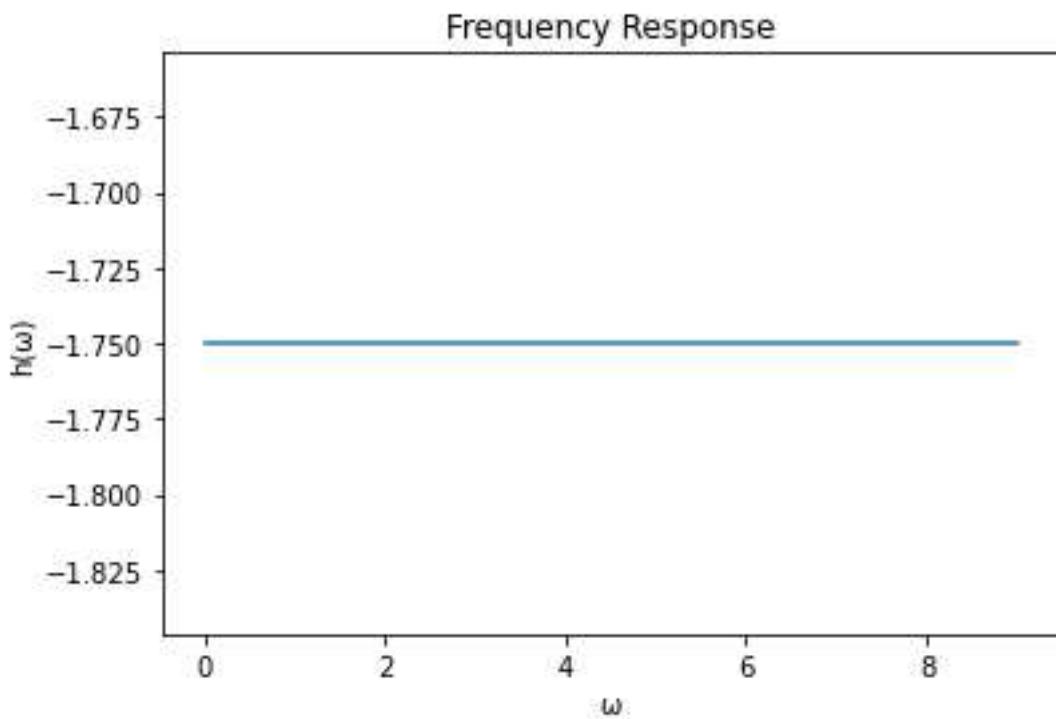
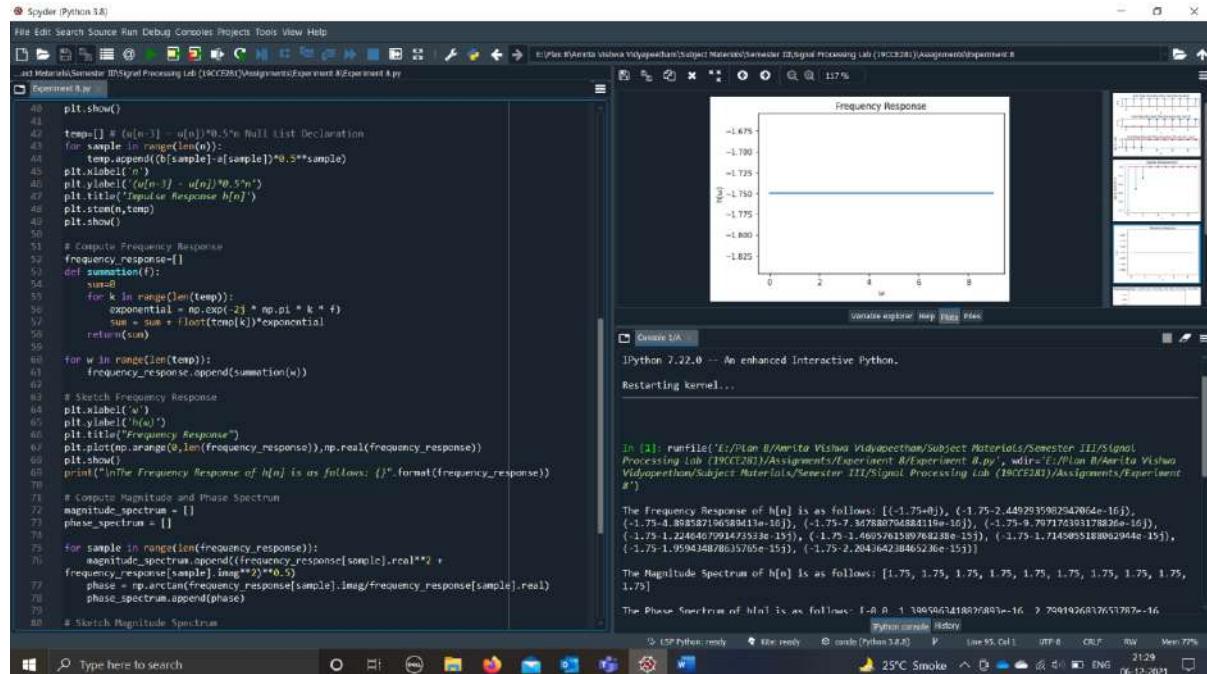
Step 1: Import library source files, numpy and matplotlib.pyplot.

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 – $h(\omega)$ 

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

Step 2: Print and sketch the frequency response, $h(\omega)$.

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

Step 4: Print and sketch magnitude spectrum, $|h(\omega)|$.

Step 5: Print and sketch phase spectrum, $\Phi(\omega)$.

Magnitude Spectrum – $|h(\omega)|$ & Phase Spectrum – $\Phi(\omega)$

```

# Spyder (Python 3.8)
File Edit Search Source Run Debug Console Projects Tools View Help
art Materials/Semester III/Signal Processing Lab (19CCE28)/Assignments/Experiment 8/Experiment 8.py
Experiment 8.py
56     exponential = np.exp(-2j * np.pi * k * f)
57     sum = sum + float(temp[k])*exponential
58   return(sum)
59
60   for w in range(len(temp)):
61     frequency_response.append(summation(w))
62
63   # Sketch Frequency Response
64   plt.xlabel('w')
65   plt.ylabel('h(w)')
66   plt.title("Frequency Response")
67   plt.plot(np.arange(0,1),frequency_response),np.real(frequency_response))
68   plt.show()
69   print("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(np.abs(frequency_response[sample].real)**2 +
77       frequency_response[sample].imag**2)**0.5
78     phase = np.arctan(frequency_response[sample].imag/frequency_response[sample].real)
79     phase_spectrum.append(phase)
80
81   # Sketch Magnitude Spectrum
82   plt.xlabel('w')
83   plt.ylabel('Angle (in radians)')
84   plt.title("Magnitude Spectrum: ...".format(magnitude_spectrum))
85   plt.plot(np.arange(0,1),magnitude_spectrum),magnitude_spectrum)
86   plt.show()
87   print("The Magnitude Spectrum of h[n] is as follows: {}".format(magnitude_spectrum))
88
89   # Sketch Phase Spectrum
90   plt.xlabel('w')
91   plt.ylabel('Angle (in radians)')
92   plt.title("Phase Spectrum \Phi(w)")
93   plt.plot(np.arange(0,1),phase_spectrum),phase_spectrum)
94   plt.show()
95   print("The Phase Spectrum of h[n] is as follows: {}".format(phase_spectrum))

```

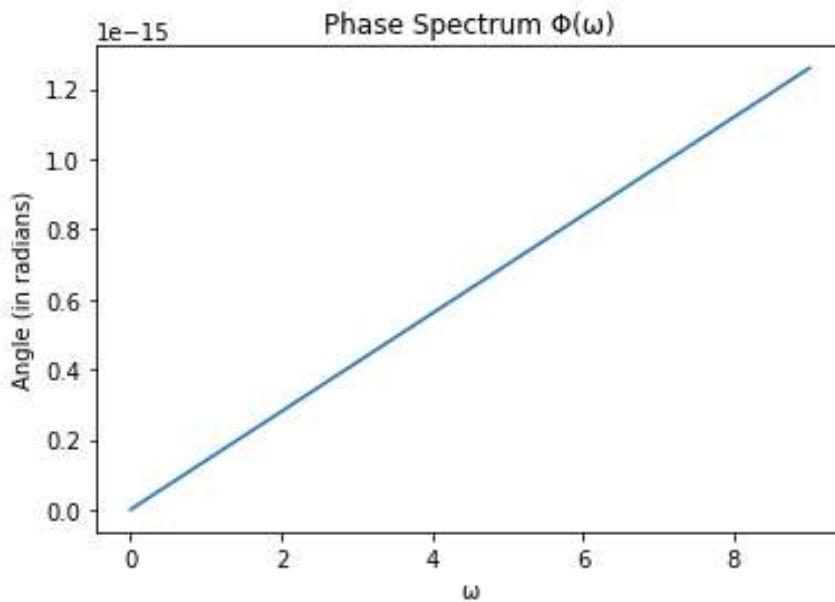
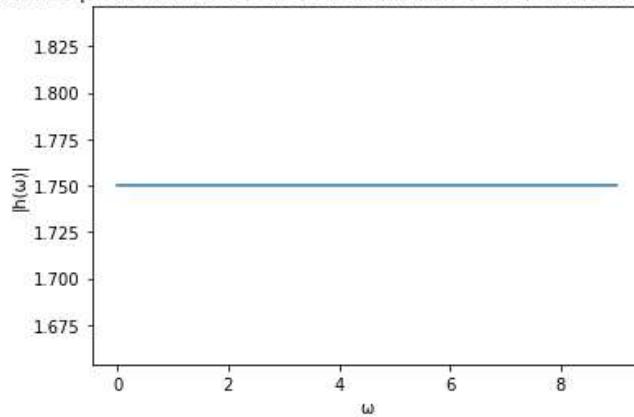
In [2]: runfile('E:/Dhanvi Vidyapeeth/Subject Materials/Semester III/Signal Processing Lab (19CCE28)/Assignments/Experiment 8/Experiment 8.py', wdir='E:/Dhanvi Vidyapeeth/Subject Materials/Semester III/Signal Processing Lab (19CCE28)/Assignments/Experiment 8')

The Frequency Response of $h[n]$ is as follows: $\{(-1.75+0j), (-1.75-2.402235982947064e-15j), (-1.75-4.09858719658941e-16j), (-1.75-7.347880754884119e-16j), (-1.75-9.757174393178826e-16j), (-1.75-1.224646799147733e-15j), (-1.75-1.469576159976928e-15j), (-1.75-1.71455180862944e-15j), (-1.75-1.95943042805379e-15j), (-1.75-2.20436623846523e-15j)\}$

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: $[0.0, 1.3995063418826893e-16, 2.7991926837653787e-16, 4.19879025648080e-16, 5.59838536753075e-16, 6.99798179413447e-16, 8.397578051296136e-16, 9.7717439178825e-16, 1.15677073506151e-15, 1.2596367076944294e-15]$

Magnitude Spectrum: ...[1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75]



```
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: $[1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75]$

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.V4.CCE20053]

DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE)

LAB TITLE AND CODE: SIGNAL PROCESSING LAB 1A [CCE281]

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, \dots, x_{N-1} to a new sequence of N complex numbers,

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

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

The x_n are thought of as the values of a function, or signal, at equally spaced times $t = 0, 1, \dots, N-1$. 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.

$$\begin{aligned} & \left[\text{This comes from Euler's formula -} \right] \\ & e^{-j 2\pi k n / N} = \cos(2\pi k n / N) + j \sin(2\pi k n / N) \end{aligned}$$

⑧

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 -

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

→ WIDDLER FACTOR -

It is denoted as W_N and defined as $W_N = e^{-j 2\pi / N}$. 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 -

$$W_N^n = W_N^{n+2N} = W_N^{n+4N} = \dots$$

It is a function of n and period N .

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

$$\Leftrightarrow W_8^0 = W_8^8 = W_8^{16} = \dots = \dots = W_8^{32} = \dots = 1 = 1 \text{ L0}$$

$$W_8^1 = W_8^9 = W_8^{17} = \dots = \dots = W_8^{33} = \dots = \frac{1}{\sqrt{2}} - j \frac{1}{\sqrt{2}} = 1 \text{ L1/4}$$

→ LINEAR TRANSFORMATION -

We know that -

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

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

$$x(n) = \text{IDFT}[X(k)]$$

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

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

(3)

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

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

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

N -point DFT in matrix form is given by $X_N = W_N x_N$
 $\Rightarrow w_N = W_N^{-1} X_N$

IDFT in matrix form is given by -

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

Comparing these expressions of x_N ,

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

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,

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

(4)

* 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, $N=4$

$$\text{Angular frequency, } \omega_0 = \frac{2\pi}{4}$$

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

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

$$+ x[2] e^{-j k \frac{2\pi}{4}(2)} + x[3] e^{-j k \frac{2\pi}{4}(3)}$$

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

Substituting the values of k , we get-

$$x[0] = 6 + 0^\circ, \text{ for } k=0$$

$$x[1] = -2 + 2^\circ, \text{ for } k=1$$

$$x[2] = -2 + 0^\circ, \text{ for } k=2$$

$$x[3] = -2 - 2^\circ, \text{ for } k=3$$

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 -

$$(y[n]) = \tan^{-1}\left(\frac{y_1}{y_0}\right) = -^{\circ} = -\times \frac{\pi}{180} \text{ radians}$$

$$(y[0]) = 0^{\circ} = 0 \text{ radians}$$

$$(y[1]) = 135^{\circ} = -0.79 \text{ radians}$$

$$(y[2]) = 180^{\circ} = 3.68 \text{ radians}$$

$$(y[3]) = -135^{\circ} = 0.79 \text{ radians}$$

LINEAR TRANSFORMATION -

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

$W_N^{k+N} = -W_N^k$
the matrix W_y may be expressed as -

$$W_y = \begin{bmatrix} W_y^0 & W_y^1 & W_y^2 & W_y^3 \\ W_y^1 & W_y^1 & W_y^2 & W_y^3 \\ W_y^2 & W_y^2 & W_y^3 & W_y^0 \\ W_y^3 & W_y^3 & W_y^0 & W_y^1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_y^1 & W_y^2 & W_y^3 \\ 1 & W_y^2 & W_y^3 & W_y^0 \\ 1 & W_y^3 & W_y^0 & W_y^1 \end{bmatrix}$$

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

Then,

$$x_y = W_y y_y = \begin{bmatrix} 6 \\ -2+2j \\ -2 \\ -2-2j \end{bmatrix}$$

Upon computing the magnitude spectrum, we find that -

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

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

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

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

6

Upon computing the phase spectrum, we find that -
 $(\alpha[n]) = \tan^{-1}\left(\frac{b}{a}\right) = -8^\circ \rightarrow -\frac{\pi}{180} \text{ radians}$

Therefore,

$$\begin{aligned} (\alpha[0]) &= 0^\circ = 0 \text{ radians} \\ (\alpha[1]) &= -135^\circ = -0.79 \text{ radians} \\ (\alpha[2]) &= 180^\circ = 3.14 \text{ radians} \\ (\alpha[3]) &= 135^\circ = 0.79 \text{ radians} \end{aligned}$$

* PROGRAM WITH COMMENTS:

```

1 # Function to calculate x(k) for direct method
2 def summation
3     sum = 0
4     for n in range(size-n):
5         exponential = np.exp(-2j * np.pi * k + n / size-n)
6         sum = sum + float(a[n]) * exponential
7     return sum
8
9 # import library source files
10 import numpy as np
11 import matplotlib.pyplot as plt
12
13 # Given Input Sequence a[n]
14 size_n = int(input("Enter the size of input a[n]: "))
15 a = [0] * (size_n)
16 print("Enter the elements of the input a[n] one-by-one
17 as follows: -")
18 for sample in range(0, size_n, 1):
19     a[sample] = input("Element " + str(sample+1) + ": ")
20
21 # sketch Input sequence
22 plt.xlabel('n')
23 plt.ylabel('a[n]')

```

```

①
23 plt.title ("Input Sequence  $a[n]$ . format (%))")
24 plt.stem (np.arange (0, size_n), x)
25 plt.grid (True) # Configure the grid lines
26 plt.show ()
27 print ("In The entered input  $a[n]$  is: - " + str(x) + "\n")
28
29 # Compute Discrete Fourier Transform Coefficients Using
30 # Direct Method
31 fourier_transform_direct = []
32 for k in range (size_n):
33     fourier_transform_direct.append (summation (k))
34     print ("In The Discrete Fourier Transform (DFT) coefficients of
35     the sequence  $a[n]$  obtained using direct method are as
36     follows: { } ". format (fourier_transform_direct))
37
38 # Compute Magnitude and Phase Spectrum of DFT Coefficients
39 magnitude_spectrum_direct = []
40 phase_spectrum_direct = []
41
42 for sample in range (len (fourier_transform_direct)):
43     magnitude_spectrum_direct.append ((fourier_transform-
44         direct [sample]. real ** 2 + fourier_transform_direct [sample].
45         imag ** 2) ** 0.5)
46     phase_direct = np.arctan (fourier_transform_direct
47         [sample]. imag / fourier_transform_direct [sample]. real)
48     phase_spectrum_direct.append (phase_direct)
49
50 # Sketch Magnitude Spectrum Using Direct Method
51 plt.xlabel ('k')
52 plt.ylabel ('| $a[k]|$ ')
53 plt.title ("Magnitude spectrum obtained Using Direct
54 Method")
55

```

```

41 plt. stem (np.arange (0, len (magnitude_spectrum_direct)),  

        magnitude_spectrum_direct)  

42 plt.grid (True) #Configure the grid lines  

43 plt.show ()  

44 print ("In The magnitude spectrum of DFT coefficients obtained  

        using direct method are as follows : {} ".format  

        (magnitude_spectrum_direct))  

45  

46 # Sketch Phase Spectrum Using Direct Method  

47 plt.xlabel ('z')  

48 plt.ylabel ('Angle (in radians)')  

49 plt.title ("Phase Spectrum obtained using Direct Method")  

50 plt.stem (np.arange (0, len (phase_spectrum_direct)),  

        phase_spectrum_direct)  

51 plt.show ()  

52 plt.grid (True) # Configure the grid lines  

53 print ("In The phase spectrum of DFT coefficients obtained  

        using direct method are as follows : {} ".format  

        (phase_spectrum_direct))  

54  

55 # Compute DFT Coefficients Using Linear Transformation  

      Method  

56  

57 # Compute W(N) 1D Array  

58 n1 = c1 = size_n  

59 nn = []  

60 for i in range (n1):  

61     for j in range (c1):  

62         nn.append (np.exp (-2j * np.pi * i + j / size_n))  

63  

64 # numpy.reshape() is used to give a new shape to an array  

       without changing its data.

```

①

```

72 wn_multidim = np.reshape(wn, (n1, c1)) # An N*N w(N) matrix
73 print ("In The matrix w(N) is as follows: \n { } ".format
(wn_multidim))
74 n2 = size_x; c2 = 1
75 x_multidim = np.reshape(x, (n2, c2)) # An N*1 x(N) matrix
76 print ("In The matrix x(N) is as follows: \n { } ".format
(x_multidim))
77
78 # Compute x(N) = w(N) * x(N), an N*1 matrix
79 fourier_transform_multidim = [(0) + c2] * a1
80 # Null Multidimensional Array
81 fourier_transform_l_t = [] # Convert multidimensional
82 # Array to 1D
83 for i in range (n1):
84     for j in range (c2):
85         fourier_transform_multidim[i][j] = 0
86         for k in range (c1):
87             fourier_transform_multidim[i][j] += 
88             wn_multidim[i][k] * float (x_multidim[k][j])
89             temp = fourier_transform_multidim[i][j]
90             fourier_transform_l_t.append (fourier_transform
91             - multidim[i][j])
92
93 print ("In The Discrete Fourier Transform (DFT) coefficients
94 of the sequence x[n] obtained using linear transformation
95 are as follows: { } ".format (fourier_transform_l_t))
96
97 # Compute magnitude and Phase spectrum of DFT Coefficients
98 magnitude_spectrum_l_t = []
99 phase_spectrum_l_t = []
100 for sample in range (len (fourier_transform_l_t)):
101     magnitude_spectrum_l_t.append ((fourier_transform_l_t
102     [sample]).real ** 2 + fourier_transform_l_t [sample].imag ** 2) ** 0.5
103     phase_spectrum_l_t.append (np.angle (fourier_transform_l_t
104     [sample]))

```

10

```

96 phase - l - t = np. arctan (fourier - transform - l - t [sample]
97 imag / fourier - transform - l - t [sample] . real)
98 phase - spectrum - l - t . append (phase - l - t)
99 # sketch Magnitude spectrum using Linear Transformation
100 method
101 plt . xlabel ('k')
102 plt . ylabel ('|x[k]|')
103 plt . title ("Magnitude spectrum Obtained Using Linear
Transformation Method")
104 plt . grid (True)
105 plt . stem (np . arange (0, len (magnitude - spectrum - l - t)), 
magnitude - spectrum - l - t)
106 plt . show ()
107 print ("In The magnitude spectrum of DFT coefficients obtained
using linear transformation are as follows: {}" . format
(magnitude - spectrum - l - t))
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 ("In The phase spectrum of DFT coefficients obtained
using linear transformation method are as follows: {}" .
format (phase - spectrum - l - t))
116

```

RESULTS

VERIFIED

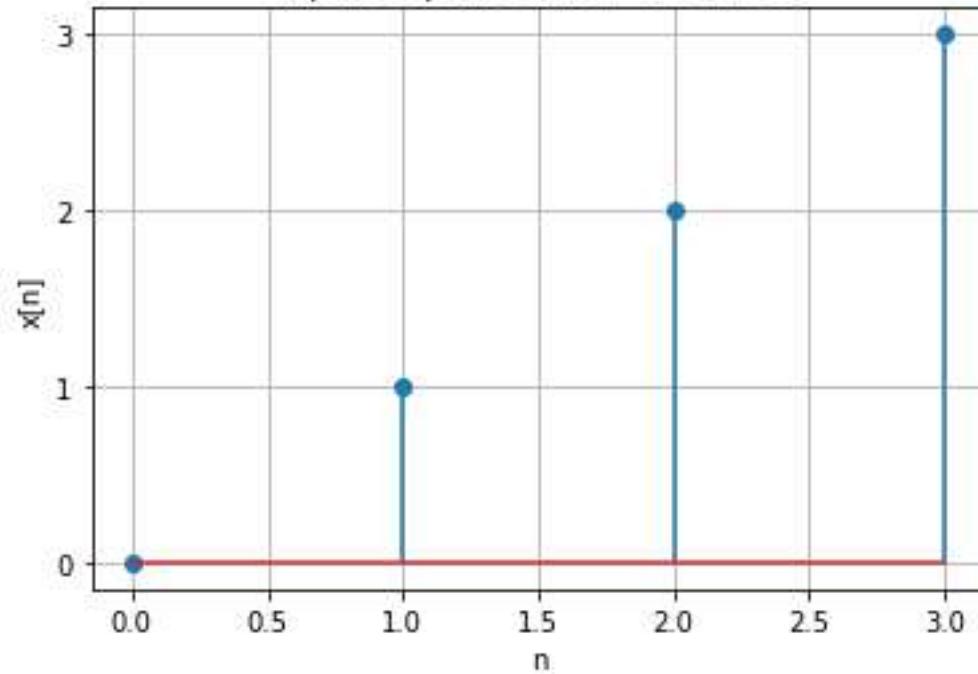
Given Input Sequence x[n]

The screenshot shows the Spyder Python 3.9 IDE interface. On the left, the code editor displays a script named `Experiment 9.py` containing Python code for calculating the Discrete Fourier Transform (DFT) coefficients using the direct method. The code includes functions for summing terms, defining the input sequence, and plotting the result. On the right, there are two panes: a plot window titled "Input Sequence x['0', '1', '2', '3']" showing discrete values at n=0, 10, 20, and 30, and a console window showing the execution of the script and the resulting output.

```

1 # Function to calculate X(k) for Direct Method
2 def summation(k):
3     sum = 0
4     for n in range(size_x):
5         exponential = np.exp(-2j * np.pi * k * n / size_x)
6         sum = sum + float(x[n]) * exponential
7     return(sum)
8
9 # Import library source files
10 import numpy as np
11 import matplotlib.pyplot as plt
12
13 # Given Input Sequence x[n]
14 size_x = int(input("Enter the size of input x[n]: "))
15 x = [0] * (size_x)
16 print("Enter the elements of the input x[n] one-by-one as follows: -")
17 for sample in range(0, size_x, 1):
18     x[sample] = input("Element " + str(sample + 1) + ": ")
19
20 # Sketch Input Sequence
21 plt.xlabel('n')
22 plt.ylabel('x[n]')
23 plt.title('Input Sequence x[n]', fontweight='bold')
24 plt.stem(range(0, size_x), x)
25 plt.grid(True) # Configure the grid lines
26 plt.show()
27 print("\nThe entered input x[n] is: - " + str(x) + "\n")
28
29 # Compute Discrete Fourier Transform Coefficients Using Direct Method
30 fourier_transform_direct = []
31 for k in range(size_x):
32     fourier_transform_direct.append(summation(k))
33 print("The Discrete Fourier Transform (DFT) coefficients of the sequence x[n] obtained using direct method are as follows: /", fourier_transform_direct)
34
35 # Compute Magnitude and Phase Spectrum of DFT Coefficients
36 magnitude_spectrum_direct = []
37 phase_spectrum_direct = []
38 for sample in range(0, size_x):
39     magnitude_spectrum_direct.append(np.abs(fourier_transform_direct[sample].real)**2 +
40         fourier_transform_direct[sample].imag**2)**0.5
41     phase_direct = np.arctan(fourier_transform_direct[sample].imag /
42         fourier_transform_direct[sample].real)
43     phase_spectrum_direct.append(phase_direct)
44
45 print(magnitude_spectrum_direct)
46 print(phase_spectrum_direct)

```

Input Sequence x['0', '1', '2', '3']

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 $N \times 1$ matrix.

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

Compute DFT Coefficients Using Direct Method

The screenshot shows the Spyder Python 3.9 IDE interface. On the left, the code editor displays a script named `Experiment 8.py` containing Python code for DFT computation using the direct method. The code includes functions for Fourier transform, magnitude spectrum, phase spectrum, and DFT coefficient calculation. It also includes a section for generating a matrix $\mathbf{H}(N)$. On the right, there are two plots: "Magnitude Spectrum Obtained Using Direct Method" and "Phase Spectrum Obtained Using Direct Method". Below the plots is a "Console" window showing the input sequence `x[n]` and the calculated DFT coefficients. The bottom status bar shows system information like temperature and date.

```

# Compute DFT Coefficients Using Direct Method

# Compute Magnitude Spectrum Using Direct Method
def fourier_transform_direct(sample):
    phase_spectrum_direct.append(phase_spectrum)
    plt.xlabel('k')
    plt.ylabel('|x[k]|')
    plt.title("Magnitude Spectrum Obtained Using Direct Method")
    plt.stem(np.arange(0, len(magnitude_spectrum_direct)), magnitude_spectrum_direct)
    plt.grid(True) # Configure the grid lines
    plt.show()

print("The magnitude spectrum of DFT coefficients obtained using direct method are as follows: \n", format(magnitude_spectrum))

# Compute Phase Spectrum Using Direct Method
def phase_spectrum_direct():
    plt.xlabel('k')
    plt.ylabel('Angle (in radians)')
    plt.title("Phase Spectrum Obtained Using Direct Method")
    plt.stem(np.arange(0, len(phase_spectrum)), phase_spectrum)
    plt.grid(True) # Configure the grid lines
    plt.show()

print("The phase spectrum of DFT coefficients obtained using direct method are as follows: \n", format(phase_spectrum))

# Compute DFT Coefficients Using Linear Transformation Method

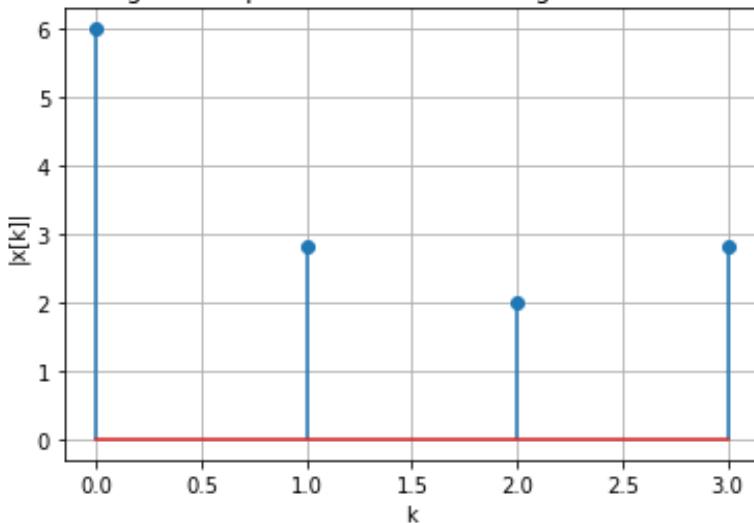
# Compute W(N) 1D Array
rl = cl = size_N
wn = []
for i in range(size_N):
    for j in range(cl):
        wn.append(np.exp(-j * np.pi * i * j / size_N))

# numpy.reshape() is used to give a new shape to an array without changing its data.
wn_multidim = np.reshape(wn, (rl, cl)) # An MMxN(M) matrix
print("The matrix W(N) is as follows: \n", format(wn_multidim))
rl = size_N; cl = 1
x_multidim = np.reshape(x, (rl, cl)) # An R1xN(M) matrix
print("The matrix x(N) is as follows: \n", format(x_multidim))

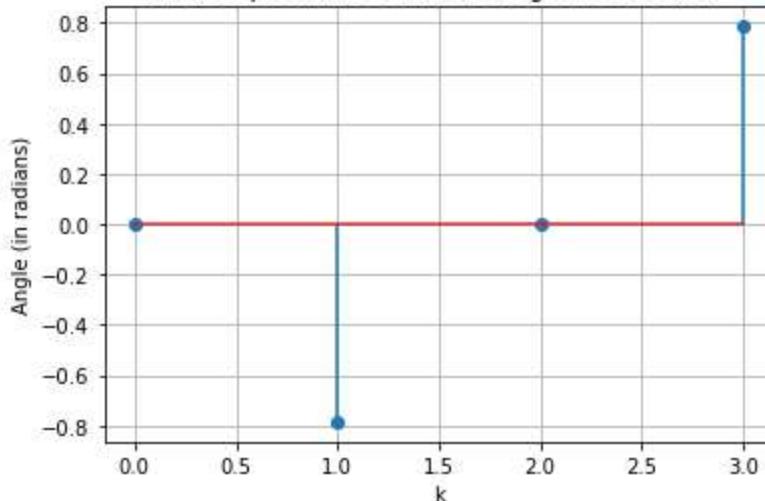
# Compute X(N) = W(N) * x(N), an Mx1 matrix
fourier_transform_multidim = [[0]*cl]*rl # Null Multidimensional Array
fourier_transform_1D = [] # Convert Multidimensional Array to 1D
for i in range(rl):
    for j in range(cl):
        .....

```

Magnitude Spectrum Obtained Using Direct Method



Phase Spectrum Obtained Using Direct Method



Compute DFT Coefficients Using Linear Transformation Method

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a Python script for calculating DFT coefficients using linear transformation. The code includes imports for numpy, defines a sequence $x[n]$, performs linear transformation, and then calculates magnitude and phase spectra. It also includes plotting code for both magnitude and phase spectra.

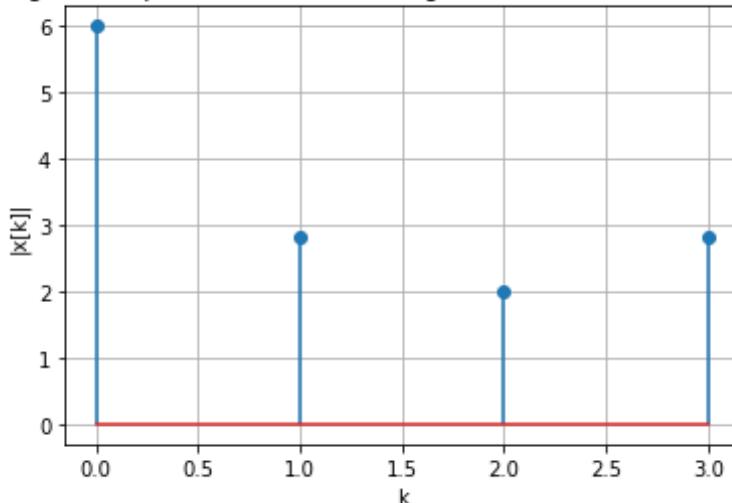
```

77: # Compute X(k) = u(k) * x(n), as 2D matrix
78: fourier_transform_multidim = [[0]*4]*4 # Multi Dimensional array
79: fourier_transform_1_t = [] # Convert Multi dimensional array to 1D
80: for l in range(4):
81:     for j in range(4):
82:         fourier_transform_multidim[l][j] = 0
83:         for k in range(4):
84:             fourier_transform_multidim[l][j] += u_multidim[l][k]*float(x_multidim[k][j])
85:             temp = fourier_transform_multidim[l][j]
86:             fourier_transform_multidim[l][j] = float(x_multidim[k][j])
87:             fourier_transform_1_t.append(fourier_transform_multidim[l][j])
88:
89: print("The Discrete Fourier Transform (DFT) coefficients of the sequence x[n] obtained using linear transformation method are as follows: ", fourier_transform_1_t)
90
91: # Compute Magnitude and Phase spectrum of DFT Coefficients
92: magnitude_spectrum_1_t = []
93: phase_spectrum_1_t = []
94: for sample in range(len(fourier_transform_1_t)):
95:     magnitude_spectrum_1_t.append((fourier_transform_1_t[sample].real)**2 + fourier_transform_1_t[sample].imag**2)**0.5
96:     phase_spectrum_1_t.append(fourier_transform_1_t[sample].imag / fourier_transform_1_t[sample].real)
97:     phase_spectrum_1_t.append(phase_spectrum_1_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.stem(np.arange(0, len(magnitude_spectrum_1_t)), magnitude_spectrum_1_t)
104: plt.grid()
105: plt.show()
106: print("The magnitude spectrum of DFT coefficients obtained using linear transformation method are as follows: ", magnitude_spectrum_1_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_1_t)), phase_spectrum_1_t)
113: plt.grid()
114: plt.show()
115: print("The phase spectrum of DFT coefficients obtained using linear transformation method are as follows: ", phase_spectrum_1_t)

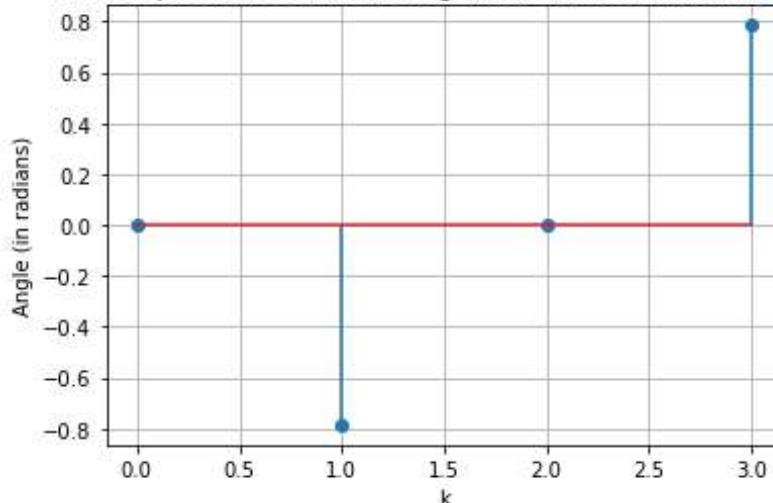
```

The right side of the interface shows two plots. The top plot, titled "Phase Spectrum Obtained Using Linear Transformation Method", shows the angle in radians for each DFT coefficient. The bottom plot, titled "Magnitude Spectrum Obtained Using Linear Transformation Method", shows the magnitude |x[k]| for each coefficient. Both plots have the x-axis labeled 'k' from 0.0 to 3.0.

Magnitude Spectrum Obtained Using Linear Transformation Method



Phase Spectrum Obtained 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), (-2.000000000000004+1.999999999999998j),  
(-2-7.34788079488412e-16j), (-1.999999999999984-2.00000000000001j)]
```

```
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+0.000000e+00j  1.0000000e+00+0.000000e+00j  
  1.0000000e+00+0.000000e+00j  1.0000000e+00+0.000000e+00j]  
 [ 1.0000000e+00+0.000000e+00j  6.1232340e-17-1.000000e+00j  
  -1.0000000e+00-1.2246468e-16j -1.8369702e-16+1.000000e+00j]  
 [ 1.0000000e+00+0.000000e+00j -1.0000000e+00-1.2246468e-16j  
  1.0000000e+00+2.4492936e-16j -1.0000000e+00-3.6739404e-16j]  
 [ 1.0000000e+00+0.000000e+00j -1.8369702e-16+1.000000e+00j  
  -1.0000000e+00-3.6739404e-16j  5.5109106e-16-1.000000e+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),
(-2.000000000000004+1.999999999999998j), (-2-7.34788079488412e-16j),
(-1.999999999999984-2.000000000000001j)]

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.

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

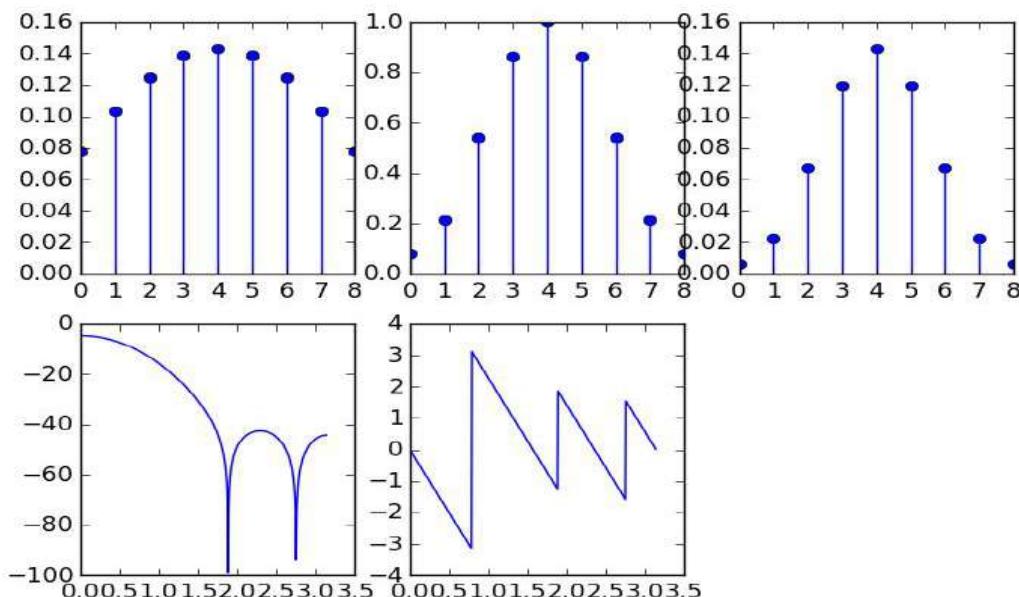
Determine the frequency response $H(e^{j\omega})$ by direct substitution of the expression {even symmetry and length is odd integer}

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.VHCE20053]

DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE)

LAB TITLE AND CODE - SIGNAL PROCESSING LAB 19CLE281

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 :

Synder 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 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.

→
PTD

→ 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 $m-1$ is same as multiplying $h_d(n)$ by the rectangular window defined as -

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

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

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

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 -

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

③

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

$$H(\omega) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(\nu) w(\omega - \nu) d\nu$$

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

$$H(\omega) = \sum_{n=0}^{M-1} R(n) e^{-j\omega n}$$

HANNING WINDOW METHOD:-

Similar to rectangular window,

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

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

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

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

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

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

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

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.

* MEDREITICAL CALCULATION :

Given -

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

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

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

$$\Rightarrow \text{for } n=0, h_d[0] = \frac{1}{2\pi} \int_{-\pi}^{\pi} 1 \times e^{jwn} dw = 0.4$$

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

⑤

$$\text{for } n=7, h_7[7] = 0.3027$$

$$\text{for } n=8, R_8[8] = 0.4$$

① RECTANGULAR WINDOW TECHNIQUE:-

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

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

$$h[n] = h_7[n] \times w[n]$$

$$= \begin{cases} h_7[n], & 0 \leq n \leq 8 \\ 0, & \text{otherwise} \end{cases}$$

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

$$H(\omega) = H_7(\omega) e^{-j\omega \left(\frac{M-1}{2}\right)} = H_7(\omega) e^{-j\omega \left(\frac{8}{2}\right)},$$

where $H_7(\omega)$ is a real function of ω and can be expressed as -

$$H_7(\omega) = h\left(\frac{M-1}{2}\right) + 2 \sum_{n=0}^{\frac{(M-3)/2}{2}} h(n) \cos\left(\frac{m-1}{2} - n\right); \text{ when } m \text{ is odd}$$

$$\text{and } H_7(\omega) = 2 \sum_{n=0}^{\frac{(M/2)-1}{2}} a(n) \cos \omega \left(\frac{m-1}{2} - n\right); \text{ when } M \text{ is even}$$

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 -

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

→ HANNING WINDOW TECHNIQUE:-

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

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

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

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

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("Enter the number of coefficients for FIR filter:"))
- 6 hd_w = int(input("Enter the frequency response value:"))

```

    0
    1
    2
    3
    4
    5
    6
    7
    8 # Inverse discrete-time Fourier transform:
    9 Rd_n = []
    10 for n in range(N//2 + 1):
    11     expression = lambda w: Rd_w + np.exp(1j * w * n)
    12     # Expression within the integral
    13     temp = integrate.quad(expression, (-2 + np.pi) / 5,
    14         (-2 - np.pi) / 5)
    15     integral = temp[0] / (2 * np.pi) # quad() gives a tuple,
    16     integral_value[0] + constant[1]
    17     Rd_n.append(integral)
    18
    19 # Compute desired filter coefficients:
    20
    21 for n in reversed(range(N//2)): # Backward iteration will start
    22     occurring from the rear
    23     symmetric = Rd_n[n]
    24     Rd_n.append(symmetric)
    25
    26 print("The desired filter coefficients a_d[n] is as follows:\n")
    27     f3 = format(Rd_n)
    28
    29 # Design FIR filter using rectangular window technique
    30 # (Impulse Response):
    31
    32 # Rectangular window function:
    33 wn_rectangular = []
    34 for n in range(N):
    35     if n < N:
    36         temp = 1
    37     else:
    38         temp = 0
    39     wn_rectangular.append(temp)

```

⑧

- 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 techniques:
- 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_rectangular)
- 44 plt.grid (True)
- 45 plt.show ()
- 46 print ("The expected filter coefficients h[n] obtained using rectangular window technique is as follows: In fig format (hn_rectangular))
- 47
- 48 # Design FIR filter using rectangular window technique (Frequency Response):
- 49
- 50 hr_w_rectangular = [] # Real function of ω
- 51 h_w_rectangular = [] # Frequency response (magnitude)
- 52 h_w_rectangular_decibels = [] # Frequency response (decibels)
- 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, 180, 10):
- 57
- 58 w_rectangular = degrees + (np.pi / 180)

①

59 sum_rectangular = 0

60 if ($N \% 2 == 1$):61 for n in range (($N-3$) // 2) + 1):

62 summation = hn_rectangular[n] +

63 np.cos(w_rectangular) + (($N-1$) // 2) - n)

64 sum_rectangular = sum_rectangular + summation

65 hr_w_rectangular.append(hn_rectangular[$(N-1)$ // 2] +
(2 * sum_rectangular))

66

67 else:

68

69 for n in range ((N) // 2) - 1):

70 summation = hn_rectangular[n] +

71 np.cos(w_rectangular) + (($N-1$) // 2) - n)

72 sum_rectangular = sum_rectangular + summation

73 hr_w_rectangular.append(2 * sum_rectangular)

74

75 hr_w_rectangular.append(hr_w_rectangular[Index_rectangular] + np.exp(-1j * w_rectangular * (($N-1$) // 2)))

76

77 hr_w_rectangular_decibels.append(20 + np.log10(abs(hr_w_rectangular[Index_rectangular]))))

78

79 Index_rectangular = Index_rectangular + 1

80

81 x_axis_rectangular.append(degrees)

82

83 # sketch frequency response obtained using rectangular window technique:

84 plt.xlabel('w in degrees')

85 plt.ylabel('magnitude of H[w]')

```

86 plt.title ("Frequency Response Using Rectangular Window
Technique")
87 plt.plot (x-axis - rectangular, h-w - rectangular - decibels)
88 plt.grid (True)
89 plt.show ()
90 print ("The frequency response H[w] obtained using,
rectangular window technique is as follows: \n {}".
format (h-w - rectangular - decibels))
91
92 # Design FIR filter using Hanning window technique
(Impulse Response):
93
94 # Hanning window function:
95 wh-hanning = []
96 for n in range (N):
97     wh-hanning.append (0.5 + (1 - np.cos ((2 * np.pi * n) /
(N-1))) )
98
99 # expected filter coefficients obtained using Hanning window
technique:
100 hn-hanning = []
101 for n in range (N):
102     hn-hanning.append (hd-n[n] + wh-hanning [n])
103
104 # Sketch expected filter coefficients obtained using Hanning
window technique:
105 plt.xlabel ('n')
106 plt.ylabel ('magnitude of h[n]')
107 plt.title ("Impulse Response Using Hanning Window
Technique")
108 plt.stem (np.arange (0, len (hn-hanning)), hn-hanning)
109 plt.grid (True)
110 plt.show ()

```

11

111 print ("The expected filter coefficients $h[n]$ obtained using Hanning window technique is as follows: \n f").
format (%.2f)

112

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

114

115 hr_w_hanning = [] # Real function of w

116 R_w_hanning = [] # Frequency response in magnitude

117 h_w_hanning - decibels = [] # Frequency response (decibels)

118 index_hanning = 0 # Index variable to iterate through
'hr_w_hanning' list

119 x_axis_hanning = [] # Define the angle axis

120

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

122

123 w_hanning = degrees + (np.pi/180)

124 sum_hanning = 0

125

126 if (N%2 == 1):

127

128 for n in range (((N-3)/2) + 1):

129 summation = hr_w_hanning[n] + np.cos(w_hanning) *
(((N-1)/2) - n)

130 sum_hanning = sum_hanning + summation

131 hr_w_hanning.append(hr_w_hanning[((N-1)/2) + 1 +
sum_hanning])

132

133 else:

134

135 for n in range ((N-2) - 1):

136 summation = hr_w_hanning[n] + np.cos(w_hanning) *
(((N-1)/2) - n)

(P)

```

137 sum_hanning = sum_hanning + summation
138 h_w_hanning.append(2 * sum_hanning)
139
140 h_w_hanning.append(h_w_hanning[index_hanning] +
    np.exp(j * 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("The frequency response H(w) obtained using
Hanning window technique is as follows: In f3",
format(h_w_hanning_decibels))
156

```

* **INFERENCES:**

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

RESULTS
VERIFIED.

Impulse Response Using Rectangular Window Technique

The screenshot shows the Spyder Python 3.9 IDE interface. On the left, the code editor displays a script named 'Experiment 10.py' containing Python code for calculating FIR filter coefficients and plotting the impulse response. On the right, there are two panes: a plot window titled 'Impulse Response Using Rectangular Window Technique' showing discrete-time Fourier transform magnitude, and a console window showing the execution of the script.

```

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 # Computes a definite integral
4
5 N = int(input("\nEnter the number of coefficients for FIR Filter: "))
6 hd_n = int(input("Enter the frequency response value: "))
7
8 # Inverse discrete-time Fourier transform.
9 hd_m = []
10 for n in range(N//2 + 1):
11     expression = lambda w: hd_n * np.exp(-j * w * n) # Expression within the integral
12     temp = integrate.quad(expression, (-2*np.pi)/N, (2*np.pi)/N)
13     integral = temp[0] / (2*np.pi) # quad() gives a tuple, integral_value[0] + constant[1]
14     hd_m.append(integral)
15
16 # Compute desired filter coefficients.
17 for n in reversed(range(N//2)): # Backward iteration will start occurring from the rear.
18     symmetric = hd_m[n]
19     hd_m.append(symmetric)
20
21 print("\nthe desired filter coefficients h_d(n) is as follows:\n ", format(hd_m))
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 == 0:
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_m[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
44 plt.show()

```

In the console window:

```

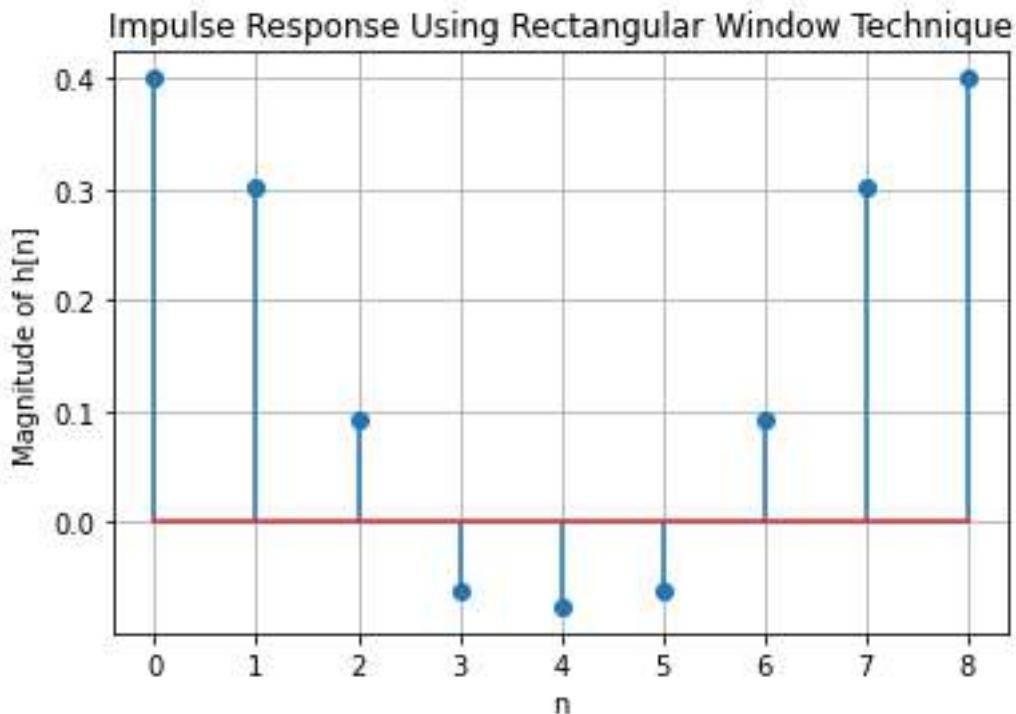
In [1]: runfile('E:/Plm B/Aerita Vishnu Vidyameethan/Subject Materials/Semester - III/Signal Processing Lab (20CE28)/Assignment/Experiment 10/Experiment 10.py', wdir='E:/Plm B/Aerita Vishnu Vidyameethan/Subject Materials/Semester - III/Signal Processing Lab (20CE28)/Assignment')
Python 3.9.2 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

Python 7.29.0 -- An enhanced Interactive Python.

Restarting kernel...

In [1]: Enter the number of coefficients for FIR Filter: 9
Enter the frequency response value: 1

```



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.

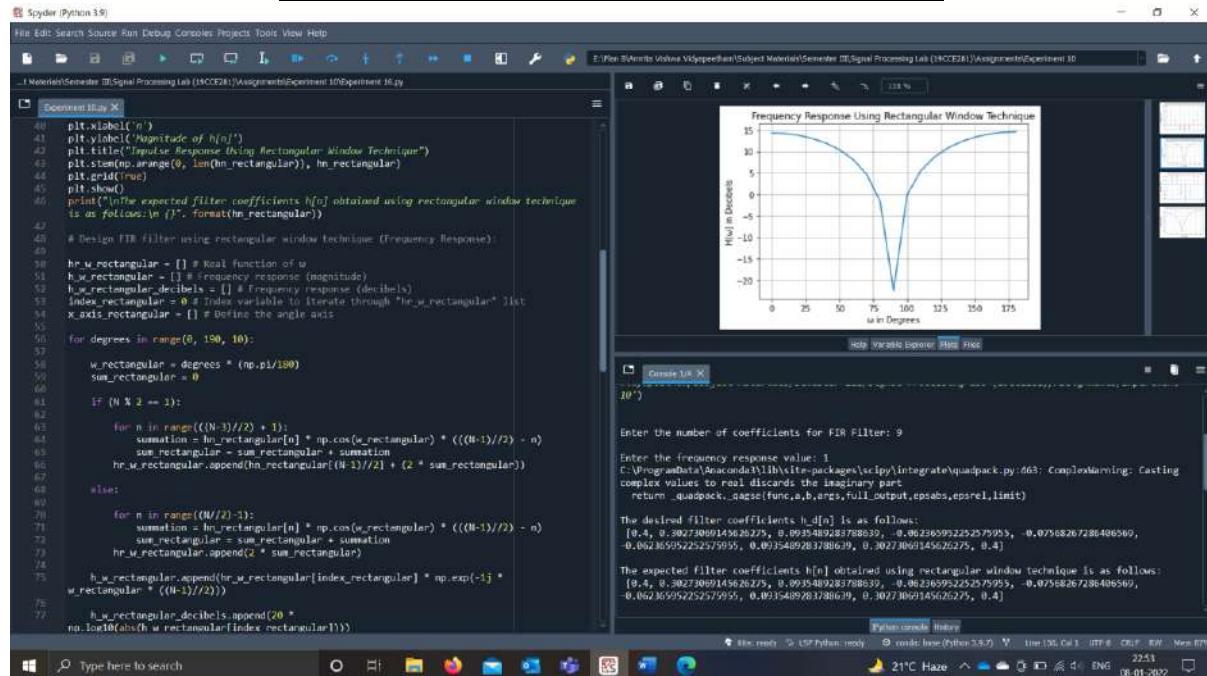
Step 3: Calculate inverse discrete-time Fourier transform and compute desired filter coefficients.

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

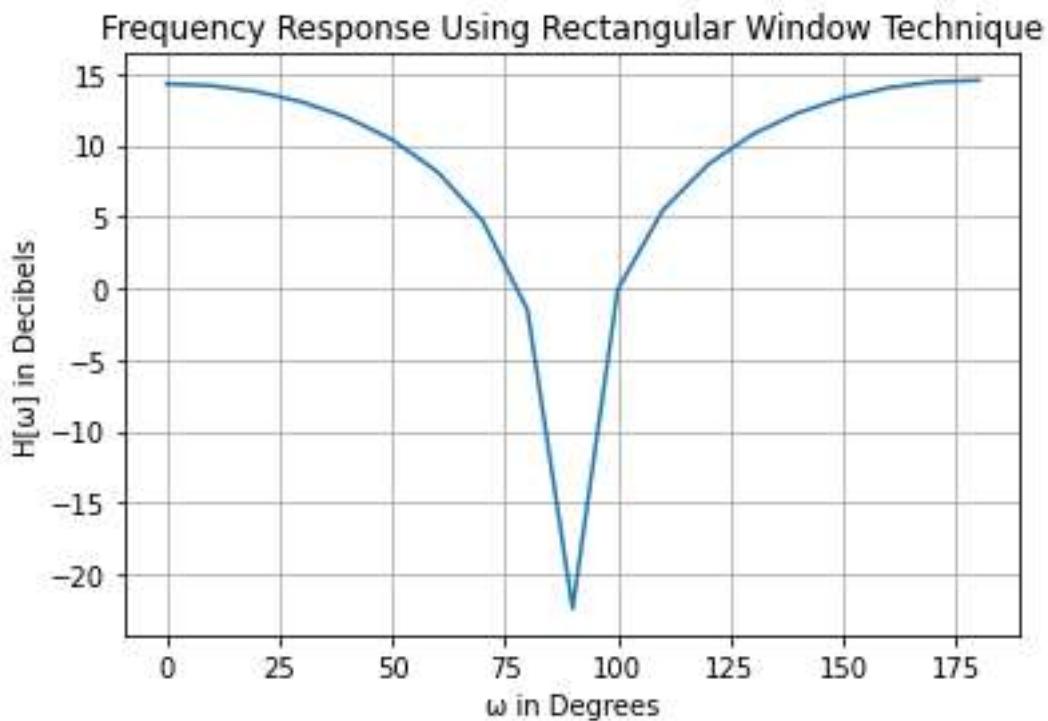


The screenshot shows the Spyder Python 3.9 IDE interface. On the left, the code editor displays a script named `Experiment 10.py` containing Python code for designing an FIR filter using the rectangular window technique. On the right, the IPython console shows the resulting frequency response plot titled "Frequency Response Using Rectangular Window Technique". The plot shows a magnitude in decibels (dB) on the y-axis ranging from -20 to 15, and an angle ω in degrees on the x-axis ranging from 0 to 375. The plot exhibits a deep null at approximately 90 degrees, indicating a stopband rejection. Below the plot, the console displays the calculated filter coefficients $h[n]$ and the frequency response magnitude $H[\omega]$ in decibels.

```

40 plt.xlabel('ω')
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_rectangular)
44 plt.grid(True)
45 plt.show()
46 print("The expected filter coefficients h[n] obtained using rectangular window technique is as follows: In dB format: ", hn_rectangular)
47
48 # Design FIR filter using rectangular window technique (Frequency Response):
49
50 hr_w_rectangular = [] # Real function of w
51 h_w_rectangular = [] # Frequency response (magnitude)
52 h_w_rectangular_decibels = [] # Frequency response (decibels)
53 index_rectangular = 0 # Index variable to iterate through "h_w_rectangular" list
54 x_axis_rectangular = [] # Define the angle axis
55
56
57 for degree in range(0, 180, 10):
58     w_rectangular = degrees * (np.pi/180)
59     sum_rectangular = 0
60
61     if (N // 2 == 1):
62
63         for n in range((N-1)//2 + 1):
64             summation = hn_rectangular[n] * np.cos(w_rectangular) + (((N-1)//2) - n)
65             sum_rectangular = sum_rectangular + summation
66             hn_rectangular.append(hn_rectangular[(N-1)//2] + (2 * sum_rectangular))
67
68     else:
69
70         for n in range((N//2)-1):
71             summation = hn_rectangular[n] * np.cos(w_rectangular) + (((N-1)//2) - n)
72             sum_rectangular = sum_rectangular + summation
73             hn_rectangular.append(hn_rectangular[(N-1)//2])
74
75     h_w_rectangular.append(hr_w_rectangular[index_rectangular] * np.exp(-1j * w_rectangular * ((N-1)//2)))
76
77     h_w_rectangular_decibels.append(20 * np.log10(abs(h_w_rectangular[index_rectangular])))


```



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.

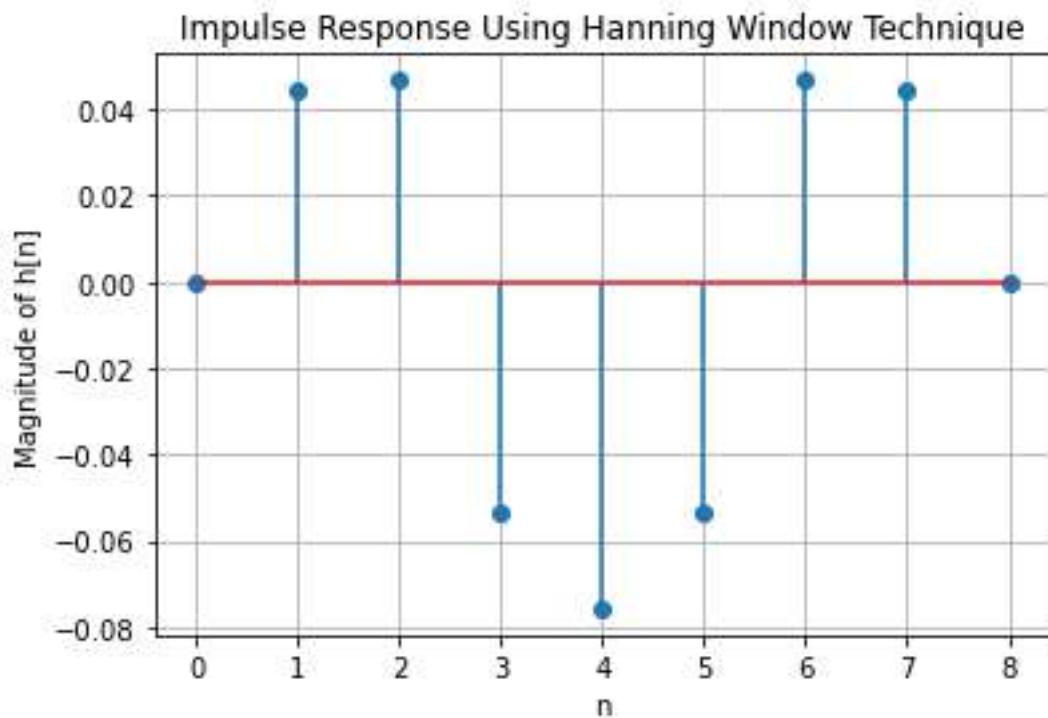
Impulse Response Using Hanning Window Technique

The screenshot shows the Spyder Python 3.9 IDE interface. On the left, the code editor displays a script named `Experiment 16.py` containing Python code for designing an FIR filter using the Hanning window technique. The code includes plotting the impulse response $h[n]$ and its magnitude in decibels. On the right, the IPython console shows the generated plot titled "Impulse Response Using Hanning Window Technique". The plot shows a discrete-time signal $h[n]$ with values at $n = 0, 1, 2, 3, 4, 5, 6, 7, 8$. The magnitude of $h[n]$ is zero for $n \neq 1, 2, 6, 7$, and reaches approximately ±0.045 for $n = 1, 2, 6, 7$. The plot also includes a red horizontal line at zero magnitude.

```

# Design FIR filter using Hanning window technique (Impulse Response)
# Hanning window function:
# Expected filter coefficients obtained using Hanning window technique:
# The expected filter coefficients h[n] obtained using Hanning window technique is as follows:
# [-0.0, 0.0443338312712346, -0.0512326706927642, -0.0756827286465569,
# -0.0932326700027643, 0.04677446418943196, 0.0441338833712349, 0.0]
# The Frequency response H(u) in decibels obtained using Hanning window technique is as follows:
# [-11.34211510661129, -11.512984881667998, -12.895916796134286, -12.97517980045179]
# Design FIR filter using Hanning window technique (Frequency Response):
# Frequency response in magnitude:
# Frequency response (decibels):

```



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.

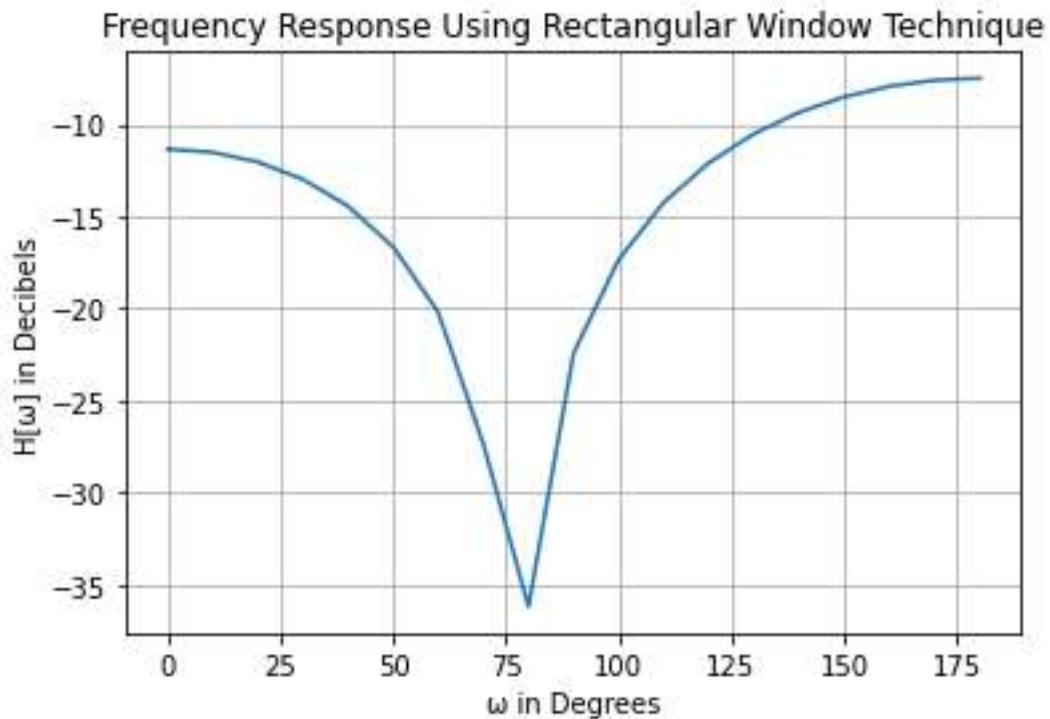
Frequency Response Using Rectangular Window Technique

The screenshot shows the Spyder Python 3.9 IDE interface. On the left, the code editor displays a script named 'Experiment 10.py' containing Python code for calculating the frequency response using a Hanning window. On the right, a plot titled 'Frequency Response Using Rectangular Window Technique' shows the magnitude in decibels (H[ω] in Decibels) versus frequency in degrees (ω in Degrees). The plot has a deep null at approximately 80 degrees.

```

1.0 index_hanning = 0 # Index variable to iterate through "hr_w_hanning" list
1.1 x_axis_hanning = [] # Declaring the angle axis
1.2
1.3 for degrees in range(0, 180, 10):
1.4     w_hanning = degrees * (np.pi/180)
1.5     sum_hanning = 0
1.6
1.7     if (N % 2 == 1):
1.8         for n in range(((N-1)//2) + 1):
1.9             summation = hn_hanning[n] * np.cos(w_hanning) * (((N-1)//2) - n)
1.10            sum_hanning = sum_hanning + summation
1.11            hr_w_hanning.append(hr_hanning[((N-1)//2) + (2 * sum_hanning)])
1.12        else:
1.13            for n in range((N//2)-1):
1.14                summation = hn_hanning[n] * np.cos(w_hanning) * (((N-1)//2) - n)
1.15                sum_hanning = sum_hanning + summation
1.16            hr_w_hanning.append(2 * sum_hanning)
1.17
1.18    h_w_hanning.append(hr_w_hanning[index_hanning] * np.exp(-1j * w_hanning * ((N-1)//2)))
1.19    h_w_hanning_decibels.append(20 * np.log10(np.abs(h_w_hanning[index_hanning])))
1.20
1.21    index_hanning = index_hanning + 1
1.22
1.23    x_axis_hanning.append(degrees)
1.24
1.25 # Sketch frequency response obtained using Hanning window technique:
1.26 plt.xlabel('ω in Degrees')
1.27 plt.ylabel('H[ω] in Decibels')
1.28 plt.title('Frequency Response Using Rectangular Window Technique')
1.29 plt.plot(x_axis_hanning, h_w_hanning_decibels)
1.30 plt.show()
1.31 print("The frequency response H[ω] in decibels obtained using Hanning window technique is as follows:\n", h_w_hanning_decibels)

```



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 $H[\omega]$ 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 $H[\omega]$ 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]: