

LAB TITLE AND CODE : SOFTWARE DEFINED RADIO LAB IACCE284  
LAB SESSION : 1  
DATE : 11/04/2022 (MONDAY)

### FAMILIARIZATION WITH PYTHON, SIMPLE EXPERIMENTS

#### \* AIM:

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

#### \* SOFTWARE REQUIRED :

Synder IDE - 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 been ported to the Java and .NET virtual machines. It was created by Guido van Rossum, and released in 1991. Python is an open-source (free) programming language that is used in web programming, data science, artificial intelligence, and many scientific applications.

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

Python has a simple syntax similar to the English language that allows developers to write programs with fewer lines than some other programming languages. It runs on an interpreter system, meaning the code can be executed as soon as it is written.

Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of a program maintenance. Python supports modules and packages, which encourages modularity and code 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.

#### \* ALGORITHM - PYTHON GETTING STARTED AND COMMANDS :

##### → PYTHON VARIOUS MODES :-

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

```
print ("Hello, World!")
```

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

- ③ Integrated Development 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 in the time required for application development.

## ① Immediate Mode -

In[1]: 1+1

Out[1]: 2

In[2]: print("Hello, world!")

Hello, world!

Ques. Ques. In immediate mode, when we enter a command, it is executed immediately and the output is displayed. For example, if we enter "1+1", the result "2" is displayed immediately. This is because the input is read and processed by the interpreter as soon as it is entered.

Ans. Ans. In immediate mode, when we enter a command, it is executed immediately and the output is displayed. For example, if we enter "1+1", the result "2" is displayed immediately. This is because the input is read and processed by the interpreter as soon as it is entered.

→ 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!")
```

→ PYTHON COMMENTS :-

Comments can be used to explain python code, make the code more readable and prevent executing when testing code.

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

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

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

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

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

```
# print ("Hello, world!")  
print ("Cheers, Mate!")
```

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

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

→ PYTHON LINES AND INDENTATION :-

Five is greater than two!

→ PYTHON COMMENTS:-

Hello, world!

Hello, world!

Cheers, Mate!

Hello, world!

Hello, world!

} Creating a comment

} Multi-line comments

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 will ignore it, and you have made a multiline comment.

#### → PYTHON VARIABLES :-

Variables are containers for storing data values.

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

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

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)
```

## → PYTHON VARIABLES :-

### ① Creating variables :-

5

John

Sally

③ 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
```

④ 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))  
print(type(y))
```

⑤ 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'
```

⑥ Case-sensitive - Variable names are case-sensitive. For example,

# This will create two variables

```
a = 4  
A = "sally"  
# A will not overwrite a
```

→ PYTHON NUMBERS :-

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

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

### ③ Get the Type :-

```
<class 'int'>  
<class 'str'>
```

### → PYTHON NUMBERS :-

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

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

```
print(type(x))
print(type(y))
print(type(z))
```

#### → PYTHON ASSIGN VALUES TO MULTIPLE VARIABLES :-

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

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

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)
```

#### → 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)
```

#### → PYTHON OPERATORS :-

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

→ PYTHON ASSIGN VALUES TO MULTIPLE VARIABLES :-

Orange  
Banana  
Cherry

Orange  
orange  
orange

→ PYTHON USER INPUT :-

Enter username : Santosh

Username is : Santosh

## ① Arithmetic operators -

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

## ② Assignment operators -

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

operator	Description	Syntax
$/=$	Divide AND: Divide left operand with right operand and then assign to left operand	$a/b$
$\% =$	Modulus AND: Takes modulus using left and right operand and assign to left operand	$a \% b$
$\lceil \rfloor =$	Divide (float) AND: Divide left operand with right operand and then assign the value (float) to left operand	$a/\lceil b$
$** =$	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	$a\mid b$
$\oplus =$	Performs XOR on operands and assign value to left operand	$a^{\oplus}b$
$>> =$	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	$x<y$
$\equiv$	Equal to : True if both operands are equal	$x\equiv y$
$\neq$	Not equal to : True if both operands are not equal	$x\neq y$

operator	Description	Syntax
$>=$	Greater than or equal to: True if the left operand is greater than or equal to the right	$x >= y$
$<=$	Less than or equal to: True if the left operand is less than or equal to the right	$x <= y$

#### ④ 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 are true	$x \text{ or } y$
not	Logical NOT: True if the operand is false	$\text{not } x$

#### ⑤ 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 true otherwise. Here, it results in 1 if $x$ is not a member of sequence $y$ .	$x \text{ is not } y$

$\xrightarrow{\text{PTD}}$

## ⑥ Membership operators -

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

→ PYTHON COLLECTIONS (ARRAYS) :-

There are four collection data types in the Python programming language :

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

For example,

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

- 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 using round brackets.

For example,

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

- 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 using curly brackets. For example,

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

- 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 using curly brackets and have keys and values.

For example ,

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

→ PYTHON COLLECTIONS (ARRAYS) :-

① List-

{'apple', 'banana', 'cherry'}

② Tuple-

('apple', 'banana', 'cherry')

③ Set-

{'banana', 'cherry', 'apple'}

④

Dictionaries-

{'brand': 'Ford', 'model': 'mustang', 'year': 1964}

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

### → PYTHON IF...ELSE :-

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

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

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  $b > a$ :

print ("b is greater than a")

Syntax for 'if' :-  
if expression:  
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  $a == b$ :

print ("a and b are equal")

→ PYTHON IF...ELSE

① Python conditions and If statements -

b is greater than a

② elif -

a and b are equal

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

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

→  
P7D

③ Else -

a is greater than b

④ And -

Both conditions are true!

⑤ Or -

At least one of the conditions is true.

$x = 41$

If  $x > 10:$

    print ("Above ten,")

If  $x > 20:$

        print ("and also above 20!")

else:

        print ("but not above 20.")

} Nested

- ③ 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)

$i += 1$

Syntax for while :-

    while test\_expression:

        statement(s)

    Iteration

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, one for each item in a list, tuple, set, etc. The for loop does not require an indexing variable to set beforehand.

## ⑥ Nested If

above ten,

and also above 20!

## → PYTHON WHILE LOOPS :-

1

2

3

4

5

For example,

```
fruits = ["apple", "banana", "cherry"]
```

for x in fruits:

```
    print(x)
```

Syntax for for-loop:-  
for value in sequence  
body of loop  
statement(s)

#### → 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!")
```

→ PYTHON FOR LOOPS :-

apple  
banana  
cherry

It is used to iterate over the elements of a sequence like lists, tuples, strings, etc. It can also be used to iterate over other iterable objects like dictionaries, sets, files, etc.

Programmatic Statement

→ PYTHON FUNCTIONS :-

The Python function is a block of code which performs a specific task. It can take zero or more arguments and can return zero or more values.

There are two types of functions in Python: built-in functions and user-defined functions. Built-in functions are pre-defined in Python and are available for use without importing any module. User-defined functions are created by the programmer to perform a specific task. They are defined using the def keyword followed by the function name and parentheses. The code inside the parentheses is the body of the function.

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

```
def my_function1():
    pass
```

```
    print("Hello from a function!")
```

```
my_function1()
```

- ③ Arguments - Information can be passed into functions as arguments. Arguments are specified after the function name, inside the parenthesis. You can add as many arguments as you want, just separate them with a comma. The following example has a function with one argument (frame). 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(frame):
    pass
```

```
    print(frame + "Refanes")
```

```
my_function2("Amil")
```

```
my_function2("Tabias")
```

```
my_function2("Linus")
```

- 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(frame, lname):
    pass
```

```
    print(frame + " " + lname)
```

```
my_function3("Amil", "Refanes")
```

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

## ② Calling a function

Hello from a function!

## ③ Arguments

Emil Refsnes

Tobias Refsnes

Linus Refsnes

## ④ Number of Arguments

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 ("amil", "Tobias", "linus")
```

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

```
def my_function5 (child3, child2, child1):
    print ("the youngest child is " + child3)
```

```
my_function5 (child1 = "amil", child2 = "Tobias", child3 = "linus")
The phrase Keyword Arguments are often shortened to kwargs in python documentations.
```

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

```
def my_function6 (**kid)
    print ("His last name is " + kid['ename'])
```

```
my_function6 (fname = "Tobias", ename = "Reffenes")
```

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

- ⑤ Arbitrary Arguments, + args -  
the youngest child is thus
- ⑥ Keyword Arguments -  
the youngest child is thus
- ⑦ Arbitrary Keyword Arguments, + + args -  
his last name is Reffner

- ⑨ 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 value

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

- ⑩ Passing a List as an Argument - You can send any data type of argument to a function (string, number, list, dictionary, etc), and it will be treated as the same data type inside the function. For example, if you send a list as an argument, it will still be a list when it reaches the function -

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

- ⑪ 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))
```

#### → 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.

## ⑧ Default Parameter Value

I am from Sweden

I am from India

I am from Norway

I am from Brazil

## ⑨ Passing a List as an Argument

apple

banana

cherry

## ⑩ Return Values

15

25

45

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 less than 100.
print ("Second : ", val2); # Gives a random number less than 100.
```

#### → PYTHON CLASSES AND OBJECTS:-

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A class is like an object constructor, or a "blueprint" for creating objects.

##### ① Create a Class - To create a class, use the keyword `class`:

```
# Create a class named MyClass, with a property named x
class MyClass:
    x=5
```

##### ② Create Object - Now, we can use the class named `MyClass` to create objects.

```
# Create an object named p1, and print the value of x
p1 = MyClass()
print (p1.x)
```

##### ③ The `_init_()` Function - The examples above are classes and objects in their simplest form, and are not really useful in real life applications. To understand the meaning of classes, we have to understand the built-in `_init_()` function. All classes have a function called `_init_()`, which is always executed when the class is being initiated. Use the `_init_()` function to assign values to object

→ PYTHON IMPORT STATEMENTS

Following are the two generated random numbers under 100 :-

First : 82

Second : 15

→ PYTHON CLASSES AND OBJECTS

② Create object

5

properties, or other operations that are necessary to do when the object is being created:-

# Create a class named Person, use the `-init-`() function to assign values for name and age

class Person:

```
def __init__(self, name, age):
    self.name = name
    self.age = age
```

p1 = Person ("John", 36)

print(p1.name)

print(p1.age)

(7) Object Methods - Object can also contain methods. Methods in objects are functions that belong to the object. Let us create a method in the Person class:-

# Insert a function that prints a greeting, and execute it on the p1 object

class Person:

```
def __init__(self, name, age):
    self.name = name
    self.age = age
```

def myfunc(self)

print("Hello! My name is " + self.name)

p1 = Person ("John", 36)

p1.myfunc()

## ③ The `__init__()` Function -

John

36

## ④ Object Methods -

Hello! My name is John

⑤ The self Parameter - The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class. It does not have to be named self, you can call it whatever you like, but it has to be the first parameter of any function in the class:-

# Use the words mysillyobject and abc instead of self  
class Person :

```
def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age
```

```
def myfunc(abc):
    print ("Hello! My name is " + abc.name)
```

```
p1 = Person ("John", 30)
p1.myfunc()
```

⑥ Modify Object Properties - You can modify properties on objects like this :-

# Set the age of p1 to 40
p1.age = 40

⑦ Delete Object Properties - You can delete properties of an object by using the del keyword:-

# Delete the age property from the p1 object
del p1.age

## ⑤ the self Parameter

Hello! my name is John

- ⑧ Delete Objects - You can delete objects by using the `del` keyword:-  
 # Delete the p1 object  
`del p1`

- ⑨ The pass Statement - class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the `pass` statement to avoid getting an error:-

```
class Person:  
    pass
```

→ PYTHON LIBRARIES :-

A Python library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. It makes Python Programming simpler and convenient for the programmer. As we don't need to write the same code again and again for different programs. Python libraries play a very vital role in fields of machine learning, Data Science, Data Visualization, etc.

- ⑩ 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 E. 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.
- ⑪ matplotlib - It 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,

```
import numpy as np # Support for large, multi-dimensional arrays
and matrices
import matplotlib.pyplot as plt # Provides an implicit way of plotting
```

```
x = np.ones(10) # Returns a new array of given shape and type,
with ones
```

```
plt.subplot(111) # subplot(rows, columns, index) describes the
figure layout
```

```
plt.plot(x) # Used to draw points (markers) in a diagram,
draws a line from point to point
```

```
plt.xlabel('Time')
```

```
plt.ylabel('Amplitude')
```

```
plt.title('Continuous unit step')
```

```
plt.subplot(122) # subplot(rows, columns, index) describes the
figure layout
```

```
plt.stem(x) # stem plot plots vertical lines at a position covered
under the graph from the baseline to y, and places a marker there
```

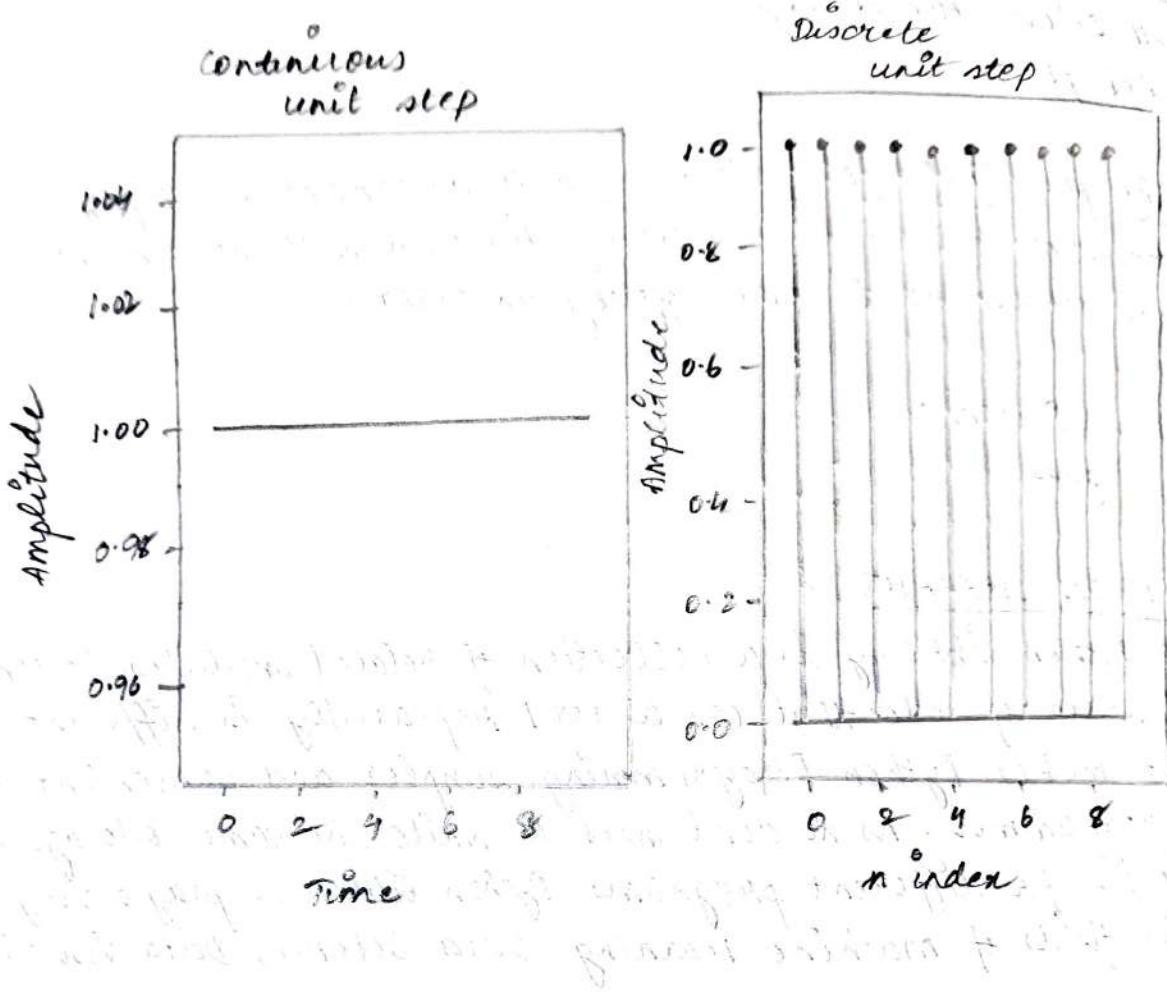
```
plt.xlabel('n index')
```

```
plt.ylabel('Amplitude')
```

```
plt.title('Discrete unit step')
```

- ③ Pandas - It is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. The library is built on top of the NumPy library. Pandas is fast and it has high performance and productivity for users.

→ PYTHON LIBRARIES :-



in this graph with the 'Sampling' value as 1.00  
so the discrete signal is zero and we have to multiply it by 1.00  
to get the continuous signal. So the continuous signal is 1.00  
for all time values and the discrete signal is 1.00 for all  
the previous values of time and 0.00 for all the next values.

So the discrete signal is zero and we have to multiply it by 1.00  
to get the continuous signal. So the continuous signal is 1.00  
for all time values and the discrete signal is 1.00 for all  
the previous values of time and 0.00 for all the next values.

- (B) scipy -
- Fundamental Algorithms :- Scipy provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics and many other classes of problems.
  - Broadly Applicable :- The algorithms and data structures provided by Scipy are broadly applicable across domains.
  - Foundational :- Extends NumPy providing additional tools for array computing and provides specialized data structures, such as sparse matrices and k-dimensional trees.
  - Performant :- Scipy wraps highly-optimized implementations written in low-level languages like Fortran, C, and C++. Enjoy the flexibility of Python with the speed of compiled code.
  - Easy to use :- Scipy's high level syntax makes it accessible and productive for programmers from any background or experience level.
  - Open source :- Distributed under a liberal BSD license, Scipy is developed and maintained publicly on GitHub by a vibrant, responsive, and diverse community.

\* INFERENCES :

A brief introduction to Python is documented along with the syntaxes and examples for basic concepts and commands.

LAB TITLE AND CODE : SOFTWARE DEFINED RADIO LAB 19CLF284

EXPERIMENT NUMBER 5)

DATE : 18/04/2022 (MONDAY)

### SAMPLING OF LOW PASS SIGNALS

\* AIM:

Generating signal and sketching the spectrum for various sampling rates. Perform the analysis for both low pass and band limited signal of bandwidth B.

\* SOFTWARE REQUIRED :

Spyder IDE (Python 3.9.7)

\* SOURCE CODE :

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

# Compute DFT coefficients using linear transformation method:

```
def DFT(x):
```

# Compute w(N) 1D array:

```
n1 = c1 = len(x)
```

```
wn = [ ]
```

```
for i in range(n1):
```

```
    for j in range(c1):
```

```
        wn.append(np.exp(-2j * np.pi * i * j / len(x)))
```

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

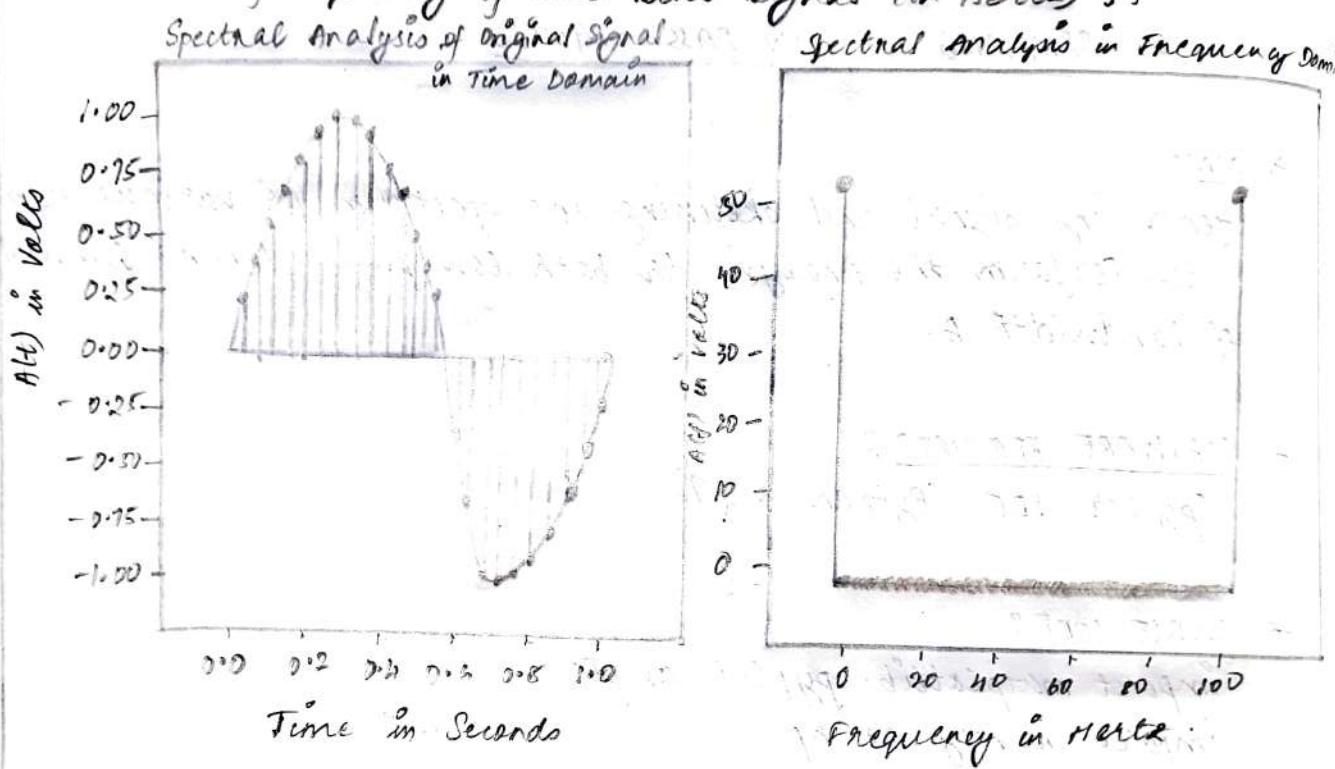
```
wn_multidim = np.reshape(wn, (n1, c1)) # An N*N w(N) matrix
```

```
n2 = len(x); c2=1
```

```
x_multidim = np.reshape(x, (n2, c2)) # An N*1 x(N) matrix
```

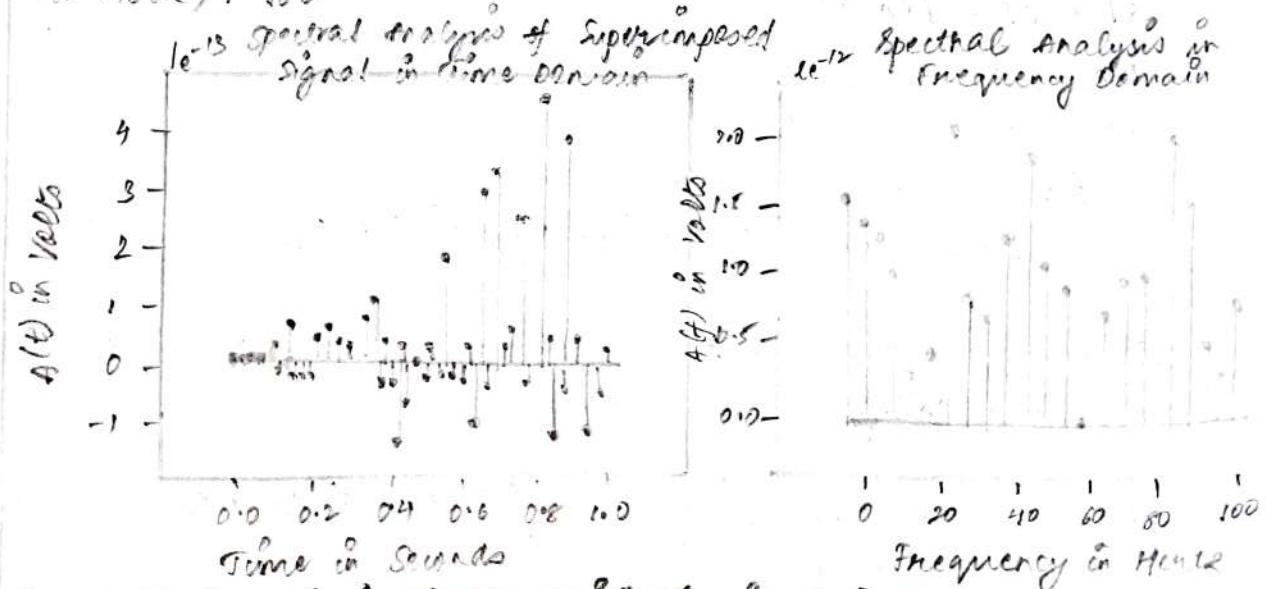
\* CONSOLE WINDOW OUTPUT

Enter the desired sampling frequency (in Hertz) : 100  
 Enter the frequency of the sine signal (in Hertz) : 1



The spectral analysis of the original signal in time and frequency domain is shown in the above plot.

Enter the bandwidth for the analysis of the compas. signal (in Hertz) : 100



The spectral analysis of the original signal in time and frequency domain is shown in the above plot.

```

# Compute  $x(N) = w(N) * x(N)$ , an  $N \times 1$  matrix
fourier_transform_multidim = [[0] * c2] * N # Null
Multidimensional Array
fourier_transform_l_t = [] # Convert Multidimensional Array
to 1D
for i in range (a1):
    for j in range (c2):
        fourier_transform_multidim[i][j] = 0
    for k in range (c1):
        fourier_transform_multidim[i][j] += wN_multidim[i][k] * float (x_multidim[k][j])
    fourier_transform_l_t.append (abs (fourier_transform_multidim[i][j]))
return fourier_transform_l_t

```

# sketch the spectrum for given sampling rate:

```
def frequency_domain (signal):
```

```
length_of_signal = len (signal)
```

```
time_period = length_of_signal / fs
```

```
temp_frequency_axis = np.arange (length_of_signal)
```

```
frequency_axis = temp_frequency_axis / time_period
```

```
plt.xlabel ("Frequency Domain")
```

```
plt.ylabel ("A(f)")
```

```
plt.title ("Spectral analysis in Frequency Domain")
```

```
plt.stem (frequency_axis, signal)
```

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

```
plt.show()
```

# Generate the sine signals:

```
fs = int (input ("Enter the desired sampling frequency in Hertz: "))

time_interval = 1 / fs
```

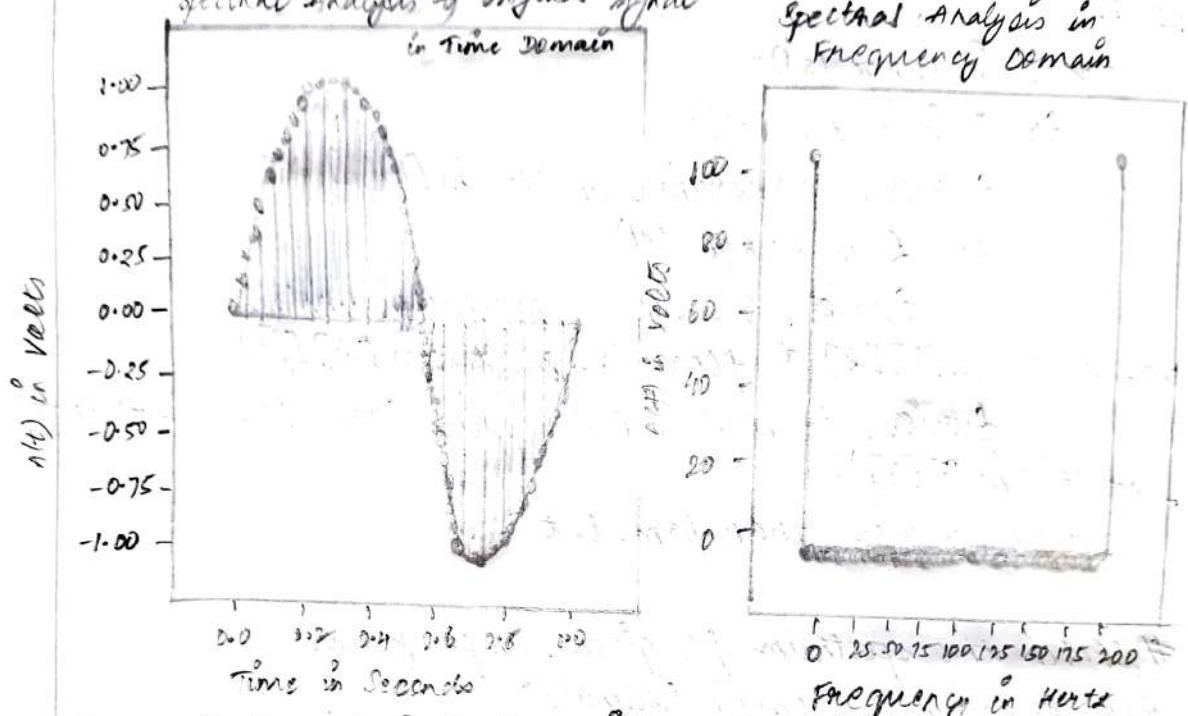
\* CONSOLE WINDOW OUTPUT:

enter the desired sampling frequency (in Hertz): 200

enter the frequency of the sine signal (in Hertz): 10

spectral analysis of original signal

spectral analysis in  
Frequency domain

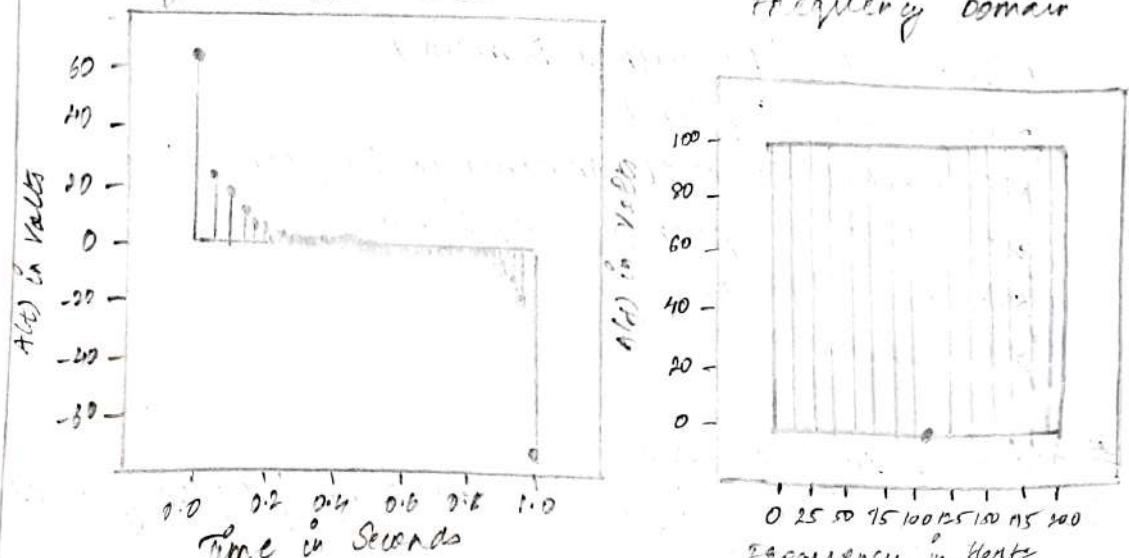


The spectral analysis of the original signal in time and frequency domain is shown in the above plot.

enter the bandwidth for the analysis of the low pass signal (in Hertz): 100

spectral analysis of superimposed  
signal in Time domain

spectral analysis in  
Frequency domain



The spectral analysis of the superimposed signal in time and frequency domain is shown in the above plot.

time-axis = np.arange(0, 1, time-interval) # Define the time axis

sine-frequency = int(input("Enter the frequency of the sine signal : "))

plt.xlabel("Time domain")

plt.ylabel("A(t)")

plt.title("Spectral Analysis of Original Signal in Time Domain")  
 plt.stem(time-axis, np.sin(2 \* np.pi \* sine-frequency \* time-axis))

plt.grid(True)

plt.show()

original-signal = np.sin(2 \* np.pi \* sine-frequency \* time-axis)

frequency-domain(DFT(original-signal))

print("The spectral analysis of the original signal in frequency domain is shown in the above plot.")

# Perform the analysis for low pass signal of given bandwidth:

superimposed-signal = 0

for i in range(1, 101, 1):

superimposed-signal = superimposed-signal + np.sin(2 \* np.pi \* i \* time-axis)

plt.xlabel("Time domain")

plt.ylabel("A(t)")

plt.title("Spectral Analysis of Superimposed Signal in Time Domain")

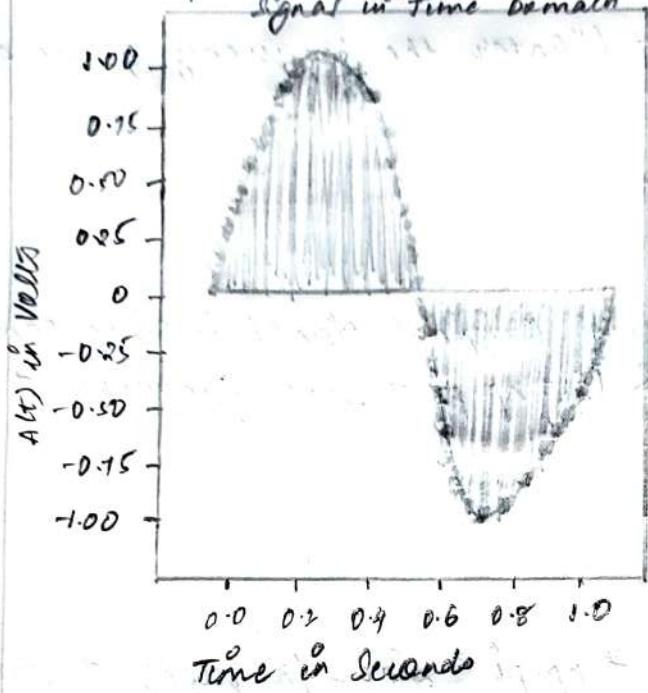
plt.stem(time-axis, superimposed-signal)

plt.grid(True)

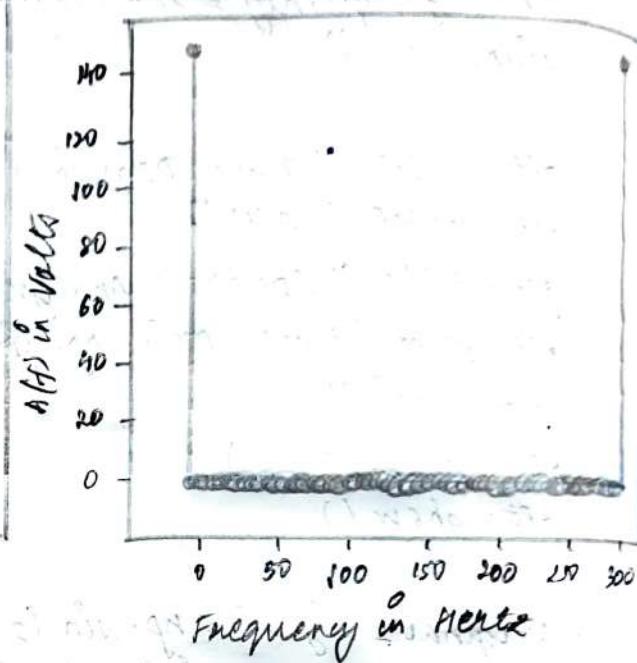
plt.show()

+ CONSOLIDATION OUTPUT :

Spectral analysis of original signal in Time domain



Spectral Analysis in Frequency Domain



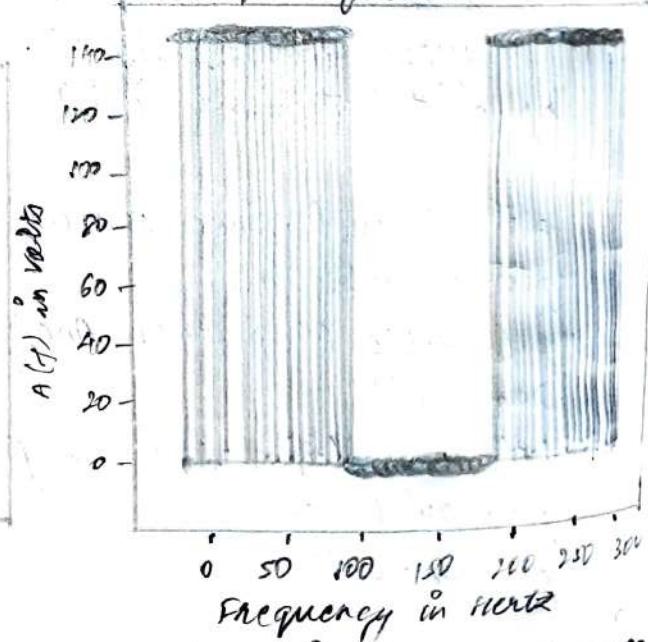
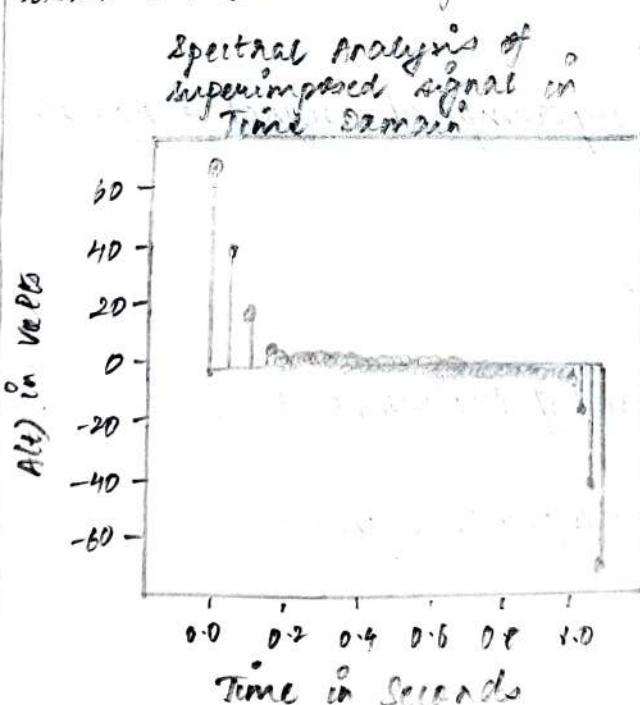
Enter the desired sampling frequency (in Hertz) : 300

Enter the frequency of the sine signal (in Hertz) : 1

The spectral analysis of the original signal in time and frequency domain is shown in the above plot.

Enter the bandwidth for the analysis of the low pass signal (in Hertz)

Spectral Analysis in Frequency Domain



The spectral analysis of the superimposed signal in time and frequency domain is shown in the above plot.

frequency-domain (DFT (superimposed-signal))

point ("In The spectral analysis of the superimposed signal in the frequency domain is shown in the above plot.")

\* INFERENCES:

Generated signal and sketched the spectrum for various sampling rates. Also, performed the analysis for low pass signal of given bandwidth. All the simulation results were verified successfully.

LAB TITLE AND CODE : SOFTWARE DEFINED RADIO LAB 19CCE284  
 EXPERIMENT NUMBER 6 2  
 DATE : 25/04/2022, MONDAY

## INTERPOLATION AND DECIMATION - TIME AND FREQUENCY

### DOMAIN ANALYSIS

\* AIM:

Generate signals of definite frequency and apply sampling rate conversion using decimation and interpolation. study the time domain and frequency domain characteristics.

\* SOFTWARE REQUIRED:

Spyder IDE (3.9.7)

\* SOURCE CODE:

```
import matplotlib.pyplot as plt # Provides an implicit of plotting
import numpy as np # support for large, multi-dimensional arrays
and matrices
```

```
# Compute DFT coefficients using linear transformation method;
def DFT(x, plot_name):
```

```
# Compute w(n) 1D Array:
```

```
w1 = c1 = len(x)
```

```
wn = []
```

```
for i in range (n1):
```

```
    for j in range (c1):
```

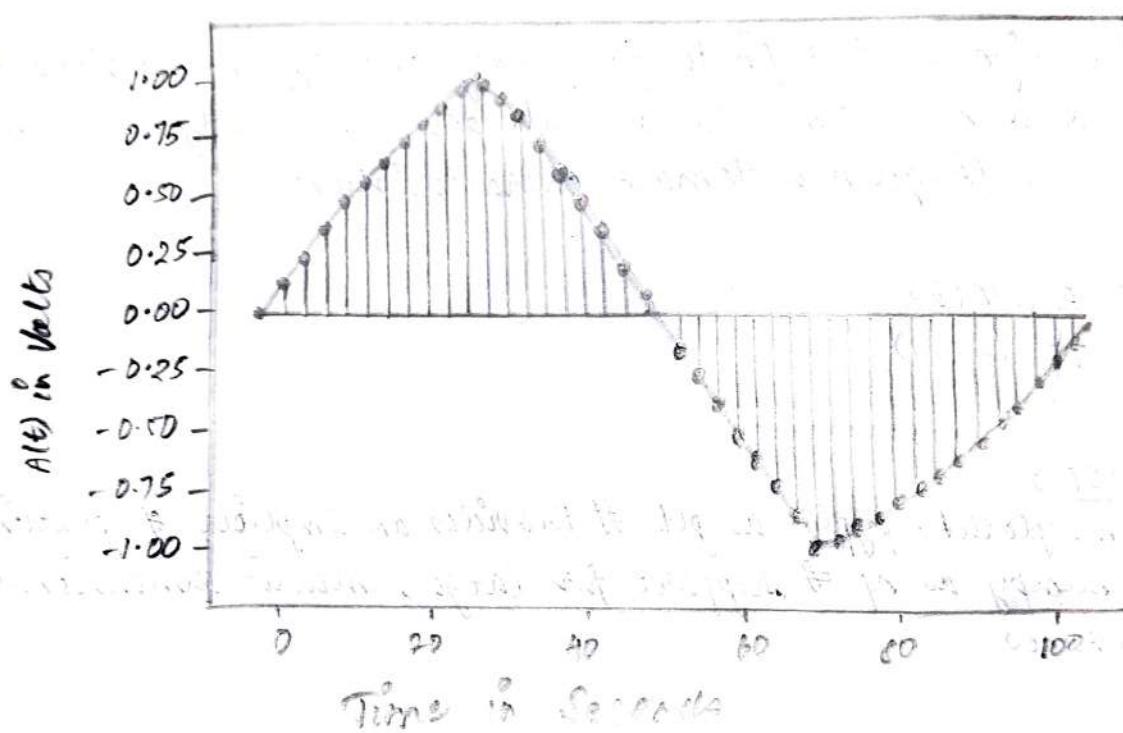
```
        wn.append(np.exp(-2j * np.pi * i * j / len(x)))
```

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

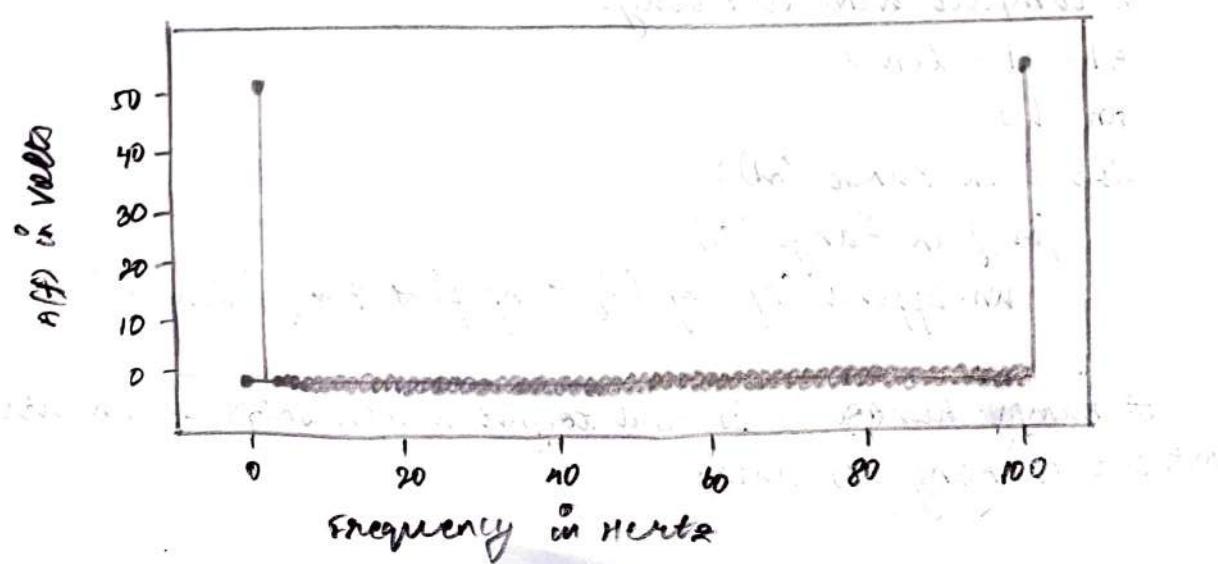
OUTPUT :

enter the desired sampling frequency (in Hertz) : 100  
enter the frequency of the sine signal (in Hertz) : 1

### Spectral Analysis of original signal in Time Domain



### Spectral Analysis of original Signal in Frequency Domain



`wn_multidim = np.reshape (wn, (a1, c1)) # An N*N w(N) matrix  
 $a2 = \text{len}(x); c2 = 1$`

`x_multidim = np.reshape (x, (a2, c2)) # An N*1 x(N) matrix`

`# Compute  $x(N) = w(N) * x(N)$ , an N*1 matrix`

`fourier_transform_multidim = [[0]*c2]*a1 # Null  
 Multidimensional Array`

`fourier_transform_l_t = [] # Convert multidimensional  
 Array to 1D`

`for i in range (a1):`

`for j in range (c2):`

`fourier_transform_multidim [i][j] +=`

`wn_multidim [i][k] * float (x_multidim [k][j]))`

`fourier_transform_l_t.append (abs (fourier_transform  
 - multidim [i][j])))`

`plt.xlabel ("Frequency in Hertz")`

`plt.ylabel ("A(f) in Volts")`

`plt.title ("Spectral Analysis of " + str(plot_name) + " in  
 Frequency Domain")`

`plt.stem (np.arange (0, len (fourier_transform_l_t)),  
 fourier_transform_l_t)`

`plt.grid (True)`

`plt.show ()`

`# sketch the spectrum for given sampling rate in time domain:`

`def time_domain (signal, plot_name):`

`plt.xlabel ("Time in seconds")`

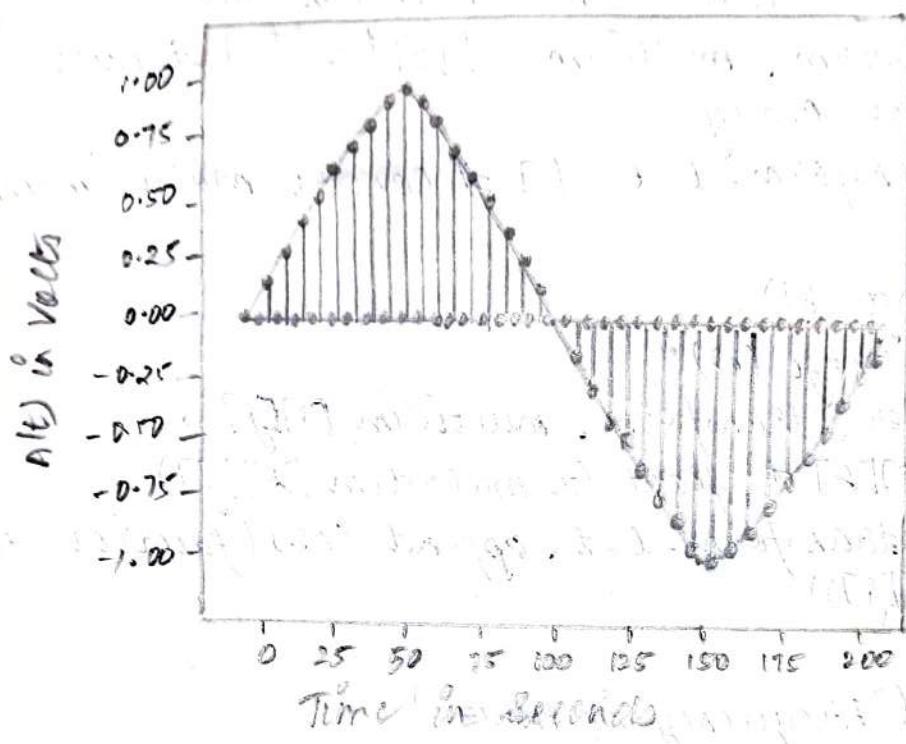
`plt.ylabel ("A(t) in Volts")`

`plt.title ("Spectral Analysis of " + str(plot_name) + " in Time  
 Domain")`

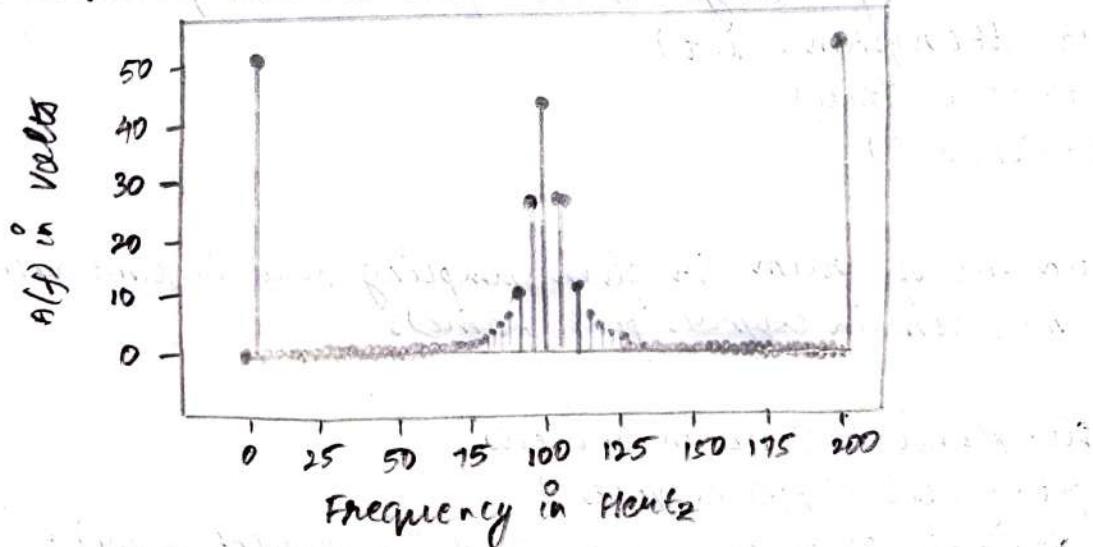
`plt.stem (np.arange (0, len (signal)), signal)`

Enter the value of 4-fold expander in the following

### Spectral Analysis of Up-Sampler in Time Domain



### Spectral Analysis of Down-Sampler in Frequency Domain



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

DFT(signal, plot-name) # Frequency domain analysis of the given signal

# Driver code: main()

# Generate the sine signal:

```
fs = int(input("Enter the desired sampling frequency (in Hertz):"))
time_axis = np.arange(0, 1, 1/fs) # Define the time axis
sine_frequency = int(input("Enter the frequency of the sine signal (in Hertz):"))
```

title = "Original Signal"

original\_signal = []

```
original_signal = np.sin(2 * np.pi * sine_frequency * time_axis)
```

title = "Up-sampler"

up\_sampler = []

l = int(input("Enter the value of l-fold expander:"))

for i in range(len(original\_signal)):

```
    up_sampler.append(original_signal[i]) # Copy the element value from original signal to up sampler array
```

if i != len(original\_signal) - 1:

for k in range(l - 1):

```
        up_sampler.append(0) # Insert "l-1" zeros between each elements of the array
```

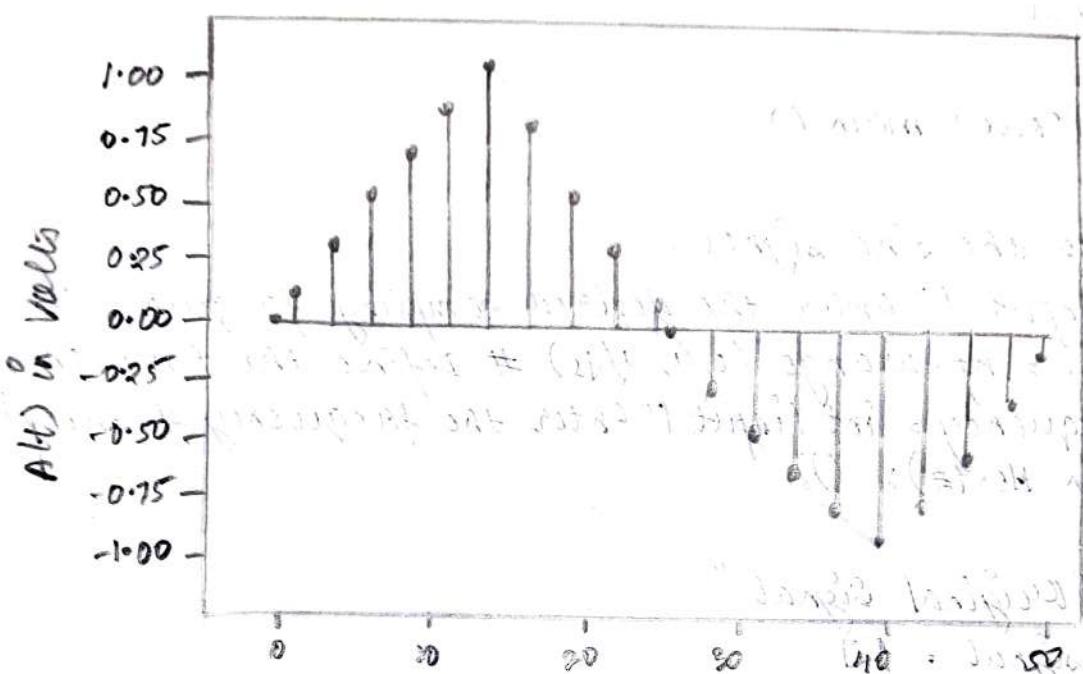
else:

break

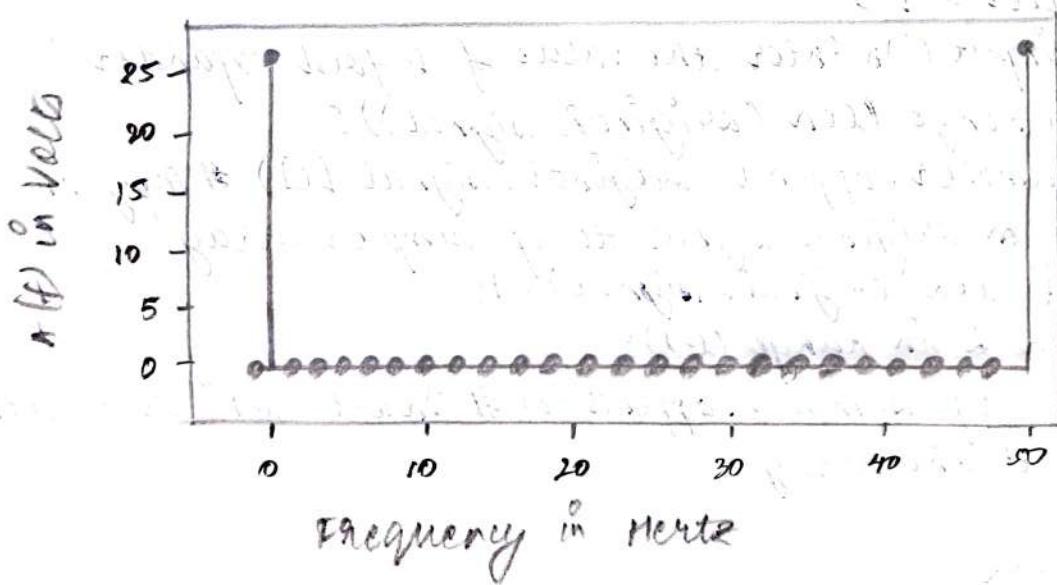
time\_domain(up\_sampler, title)

Enter the value of M-fold expander : 2

### Spectral Analysis of Down-Sampler in Time Domain



### Spectral Analysis of Down-Sampler in Frequency Domain



```
title = "Down-sampler"
down-sampler = []
m = int(input("Enter the value of m-fold expander: ")) # Copy
the element value from original signal to down sampler array with
the increment value set to "m":
for i in range(0, len(original.signal), m):
    down-sampler.append(original.signal[i])
time-domain(down-sampler, title)
```

RESULT:

Generated signals of definite frequency and applied sampling rate conversion using decimation and interpolation. Also, studied the time domain and frequency domain characteristics. All the simulation results were verified successfully.

LAB CODE AND TITLE : SOFTWARE DEFINED RADIO LAB MILE 284  
 EXPERIMENT NUMBER : 3  
 DATE : 25/04/2022

### FRACTIONAL RATE CONVERSION

\* AIM:

Using the concept of interpolation and decimation, perform the fractional rate conversion for achieving the given sampling rate. Study the time domain and frequency domain characteristics.

\* SOFTWARE REQUIRED :

spyder IDE (3.9.7)

\* PYTHON CODE:

```
import matplotlib.pyplot as plt # Provides an implicit way of
plotting
import numpy as np # support for large, multi-dimensional
array and matrices
```

```
# Compute DFT coefficients using linear transformation method:
def DFT(x, plot_name):
```

```
# Compute W(N) 1D Array:
```

```
w1 = cl = len(x)
```

```
wn = []
```

```
for i in range (cl):
```

```
    for j in range (cl):
```

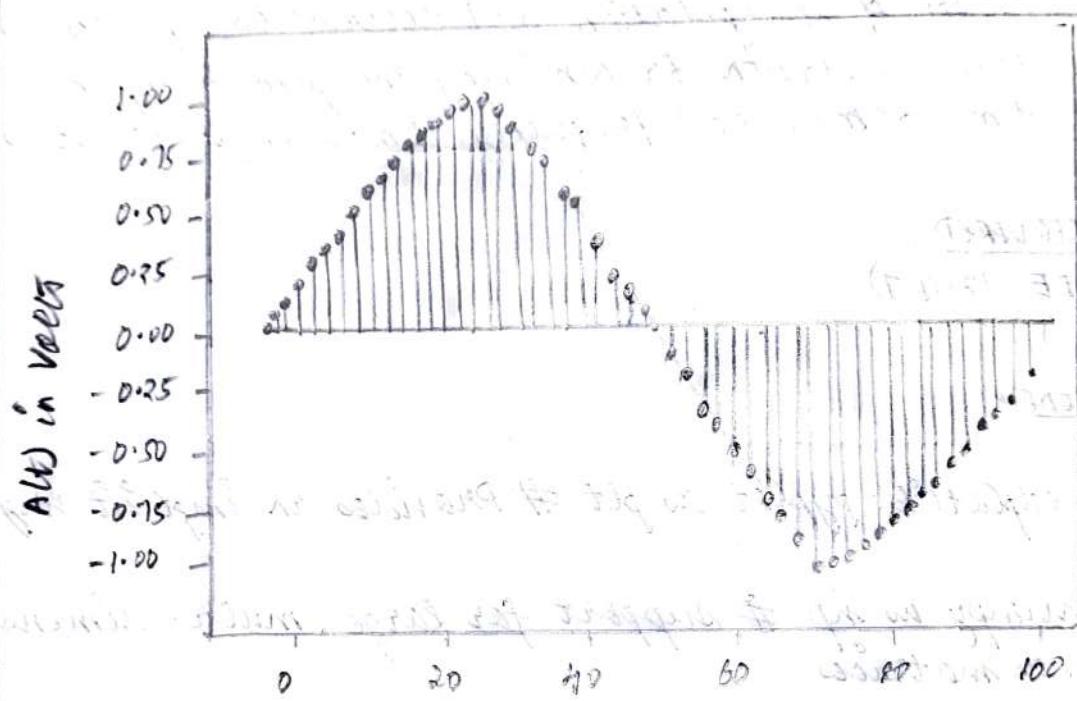
```
        wn.append (np.exp (-j * np.pi * i * j / len(x)))
```

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

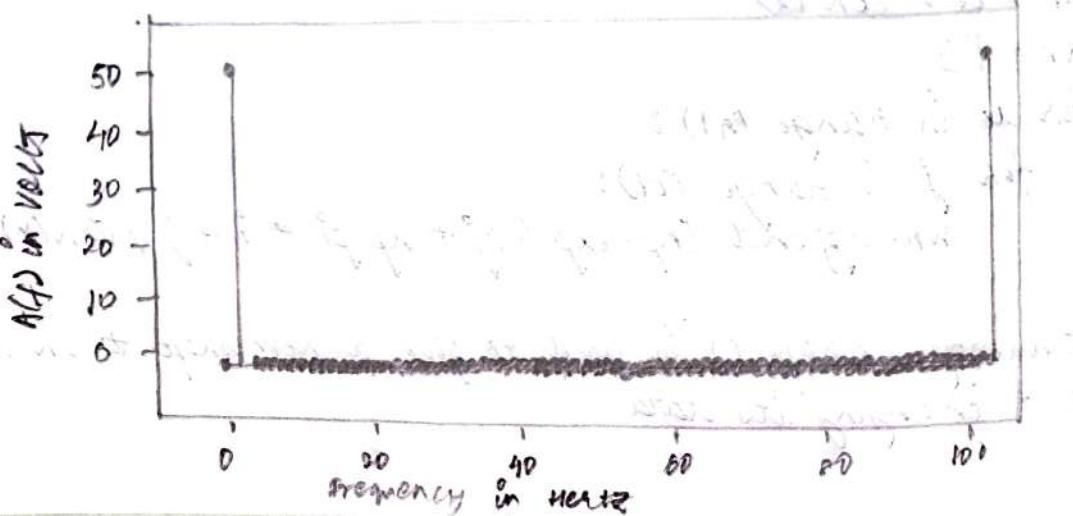
\* INPUT :

enter the desired sampling frequency (in Hertz) : 100  
enter the frequency of the sine signal (in Hertz) : 1

### Spectral Analysis of Original Signal in Time Domain



### Spectral Analysis of Original Signal in Frequency Domain



~~multidim + np.reshape () is used to give a new shape to an array without changing its data.~~

nn-multidim = np.reshape(nn, (n1, (1))) # An N\*1 x (N) matrix

n2= len(x); c2=1

x-multidim = np.reshape(x, (n2, (c2))) # An N\*1 x (N) matrix

# Compute X(N) = NN \* x(N), an N\*1 matrix

fourier-transform-multidim = [[0]\*c2]\*n1 # Null multidimensional Array

fourier-transform-l-t = [] # Convert multidimensional Array to 1D

for i in range (n1):

    for j in range (c2):

        fourier-transform-multidim[i][j] = 0

        for k in range (c1):

            fourier-transform-multidim[i][j] += nn-multidim[i][k] \* float(x-multidim[k][j])

        fourier-transform-l-t.append (abs (fourier-transform-multidim[i][j])))

plt.xlabel ("Frequency in Hertz")

plt.ylabel ("A(f) in Volts")

plt.title ("spectral Analysis of " + str(plot-name) + " in Frequency Domain")

plt.stem (np.arange (0, len (fourier-transform-l-t)), fourier-transform-l-t)

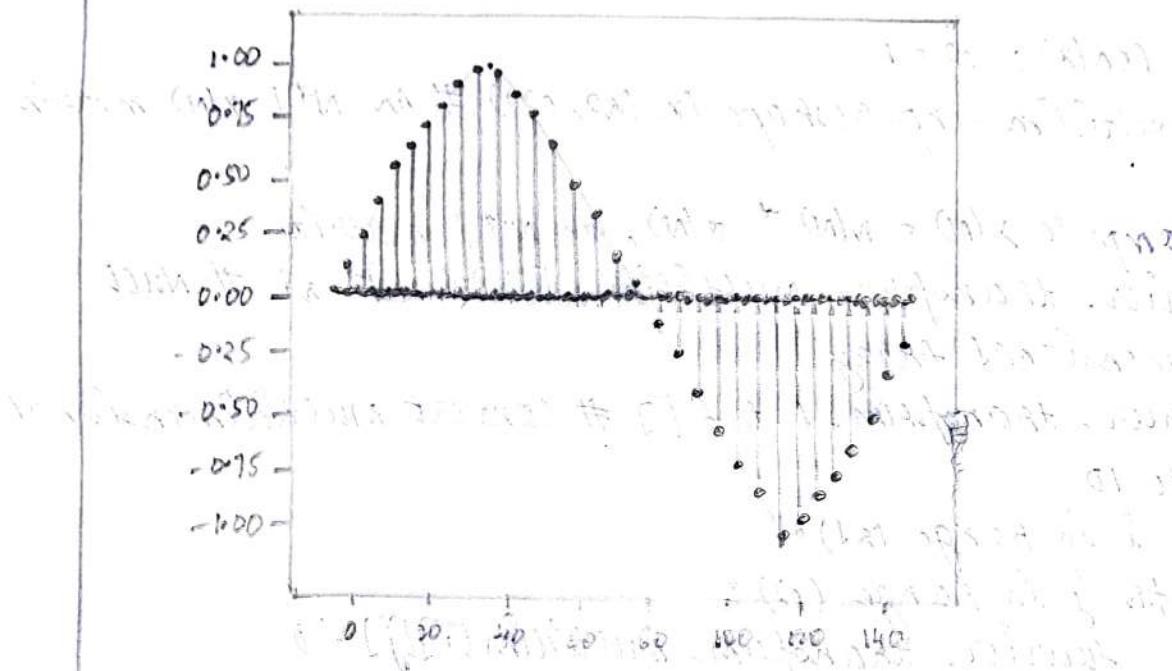
plt.grid (True)

plt.show ()

Enter the value of fractional rate conversion : 1.5

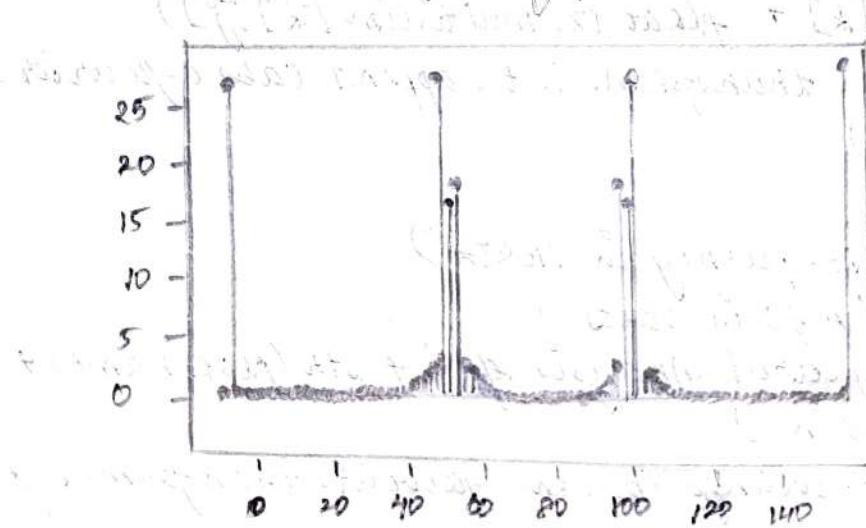
### Spectral Analysis of Final Plot

\* in Time Domain



### Spectral Analysis of Final Plot in Frequency Domain

\* in Frequency Domain



The original signal has been up-sampled by a factor of 3 and down-sampled by a factor of 2.

# sketch the spectrum for given sampling rate in time domain:

def time-domain (signal, plot-name):

plt.xlabel ("Time in seconds")

plt.ylabel ("A(t) in volts")

plt.title ("Spectral Analysis of " + str(plot-name) + " in Time Domain")

plt.stem (np.arange (0, len(signal)), signal)

plt.grid ()

plt.show ()

DFT (signal, plot-name) # Frequency domain analysis of the given signal

def down-sample (signal, m):

down-sampler = []

# Copy the element value from original signal to down sampler array with the increment value set to "m" -

for i in range (0, len(signal), m):

down-sampler.append (signal[i])

title = "Final Plot"

time-domain (down-sampler, title)

def up-sample (signal, m, l):

up-sampler = []

for i in range (len(signal)):

up-sampler.append (signal[i]) # Copy the element value from original signal to up sampler array

if i == len(signal) - 1:

for k in range (l - 1):

up-sampler.append (0) # Insert "l-1" zeros

between the elements of the array

else :

break

down-sample (up-sample, n)

# Driver code - main ()

# Generate the sine signal :-

fs = int(input("Enter the desired sampling frequency (in Hertz): "))  
 time-axis = np.arange(0, 1, 1/fs) # Define the time axis  
 sine-frequency = int(input("Enter the frequency of the sine signal (in Hertz): "))

title = 'Original signal'

original-signal = []

original-signal = np.sin(2 \* np.pi \* sine-frequency \* time-axis)

time-domain (original-signal, title)

fractional-rate = float(input("Enter the value of fractional rate conversion : "))

for x in range (1, 100, 1) :

y = fractional-rate + x

if y - int(y) == 0.5

up-sample (original-signal, x, int(y))

break

print ("The original signal has been up-sampled by a factor of " + str(int(y)) + " and down-sampled by a factor of " + str(x) + ".")

**+ RESULT:**

Performed the fractional rate conversion for achieving the given sampling rate using the concept of interpolation and decimation. Also, studied the time domain and frequency domain characteristics and all simulation results were verified successfully.

✓  
9/15/22

LAB TITLE AND CODE : MACLE 2024 SOFTWARE DEFINED RADIO LAB  
 EXPERIMENT NUMBER : 4  
 DATE : 09/05/2022, MONDAY

### POLYPHASE STRUCTURES

\* AIM:

For the filters designed to be used in interpolators and decimators, implement the simplified polyphase architecture. Analyze the complexity and latency of the implemented filters.

\* SOFTWARE REQUIRED :

Spyder IDE - Python 3.9

\* CODE: ORDER OF FILTER = 2

```
#import matplotlib.pyplot as plt # Provides an implicit way of
#plotting
```

```
#import numpy as np # support for large, multi-dimensional
#arrays and matrices
```

```
#import time # Handle time-related tasks
```

# Plot time-domain spectrum of given signal :-

```
def time_domain(signal, plot_name):
```

```
plt.xlabel("Time in seconds")
```

```
plt.ylabel("Alt in Volts")
```

```
plt.title(plot_name + " in Time Domain")
```

```
plt.stem(np.arange(0, len(signal)), signal)
```

```
plt.grid()
```

```
plt.show()
```

# Down-sample the given signal by a factor of "m": -

```
def down-sample(signal, m):
    down-sampler = []
```

# Copy the element value from original signal to down sampler array with the increment value set to "m":

```
for i in range(0, len(signal), m):
```

```
    down-sampler.append(signal[i])
```

```
return down-sampler
```

# Up-sample the given signal by a factor of "l": -

```
def up-sample(signal, l):
```

```
    up-sampler = []
```

```
    for i in range(len(signal)):
```

# Copy the element value from original signal to up sampler array

```
        if i == len(signal) - 1:
```

```
            for k in range(l - 1):
```

# Insert "l-1" zeros between the elements of the array

```
        else:
```

```
            break
```

```
    return up-sampler
```

# Shift the samples of given signal by "phase-shift" units: -

```
def shift-sample(signal, phase-shift):
```

```
    for i in range(phase-shift):
```

signal.append(0) # Adds required zeros at the end of list

```
    for i in range(len(signal) - 1, 0, -1):
```

signal[i] = signal[i-1] # shifts the zeros to the beginning of the list

```
return signal
```

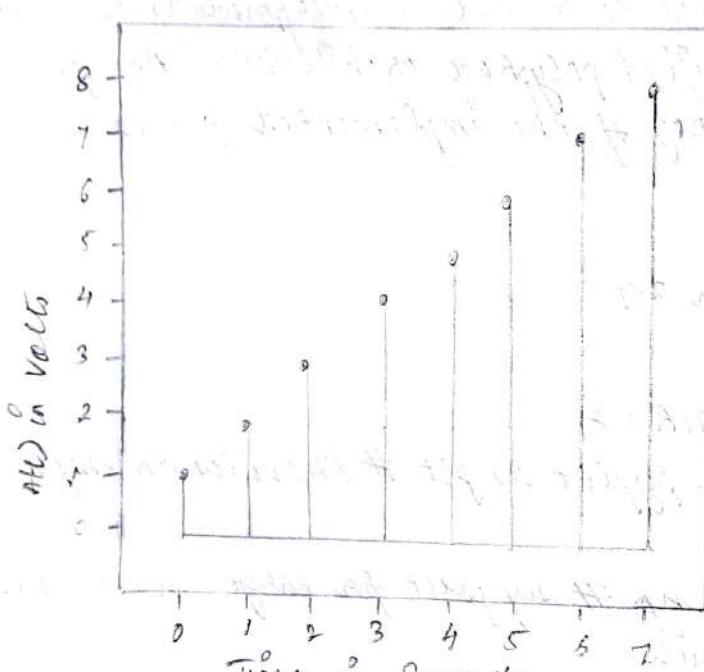
+ OUTPUT (CONSOLE & PLOTS) : (ORDER OF FILTER = 2)

The original signal is : [1, 2, 3, 4, 3, 6, 7, 8]

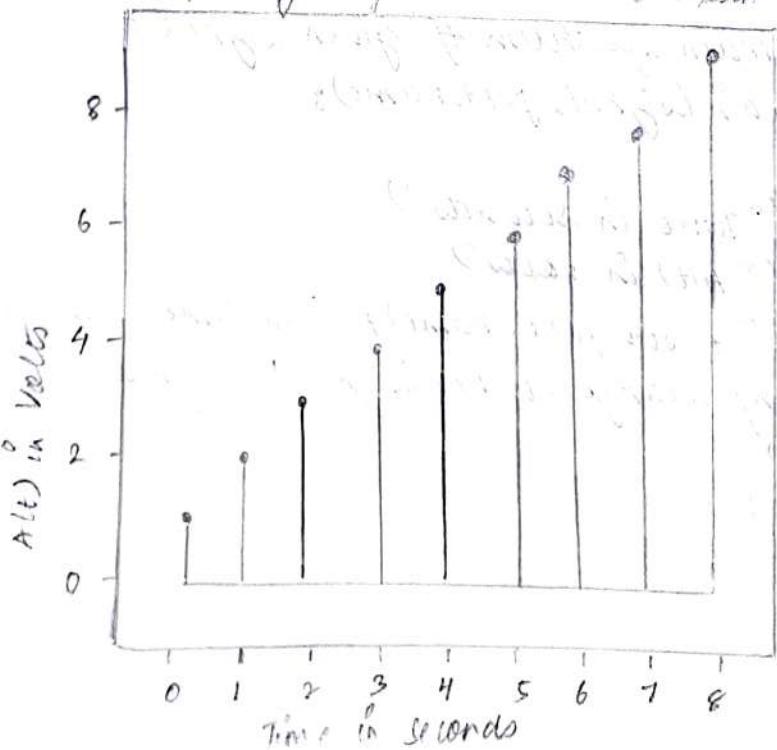
The frequency response is : [1, 2, 3, 4, 5, 6, 7, 8, 9]

The order of the filter is 2.

original signal in Time Domain



Frequency Response in Time Domain



# Perform convolution,  $x[n] * h[n]$  :-  
def convolution(x, h):

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

size = (len(x) + len(h)) - 1 # compute the size of system response

x = np.pad(x, (0, size - len(x)), 'constant')

h = np.pad(h, (0, size - len(h)), 'constant')

y = np.zeros(size, dtype=int) # Returns a new array of given shape and type, with zeros.

for i in range(size):

    for j in range(size):

        if i >= j:

            y[i] = int(y[i]) + (int(x[i-j]) \* int(h[j]))

return y # system Response y[n]

# Driver Code : main(); Execution starts here.

# Generate the original signal:-

title = "original signal"

original\_signal = [1, 2, 3, 4, 5, 6, 7, 8]

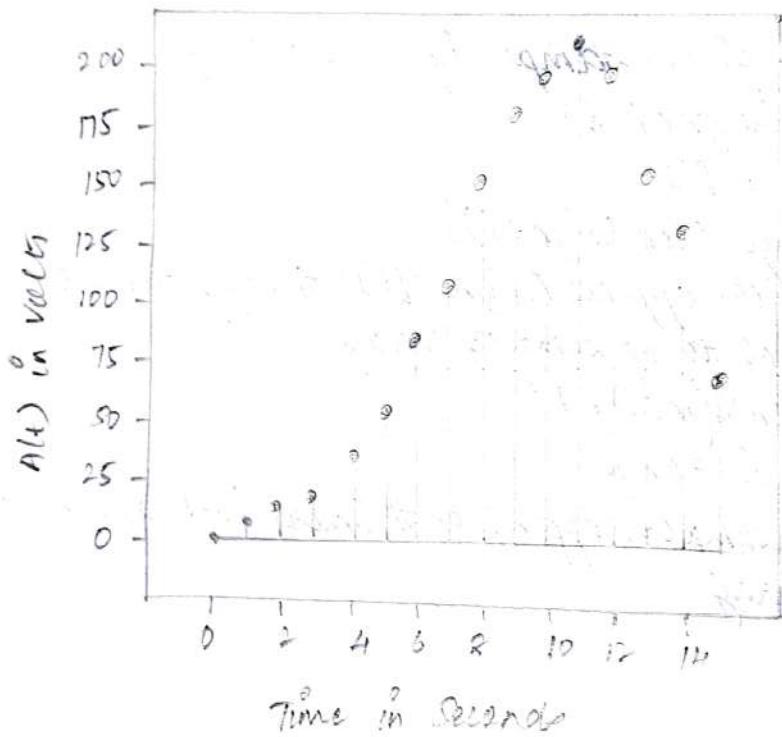
print("The original response is: ", original\_signal)

time\_domain(original\_signal, title)

The system response after convolution for 'Traditional Filter - Direct Representation' is : [14 10 20 35 56 84 120 156 182 197 200 190 166 127 72]

The time of execution of 'Traditional Filter - Direct Representation' is (in seconds) : 0.11633872441101074

### Traditional Filter - Direct Representation in Time Domain



$$E0[n] : [1, 0, 3, 0, 5, 0, 7, 0, 9]$$

$$E1[n] : [0, 2, 0, 4, 0, 6, 0, 8, 0]$$

The system response after summing up the individual convolutions for 'Polyphase Filter Representation' is : [14 10 20 35 56 84 120 156 182 197 200 190 166 127 72]

The time of execution of 'Polyphase Filter Representation' is (in seconds) : 0.1625978946685191

```

# Generate the frequency response:-
title = "Frequency Response"
frequency_response = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print("The frequency response is: ", frequency_response)
time_domain(frequency_response, title)

m = int(input("Enter the order of filter: "))

# Case - I:-
start = time.time()
title_I = "Traditional Filter- Direct Representation"
output_I = convolution(original_signal, frequency_response)
print('The system response after convolution for "' + str(title_I) + "' is: ", output_I)
time_domain(output_I, title_I)
end = time.time()
print("The time of execution of " + str(title_I) + " is (in seconds): ", end - start)

print("\n")

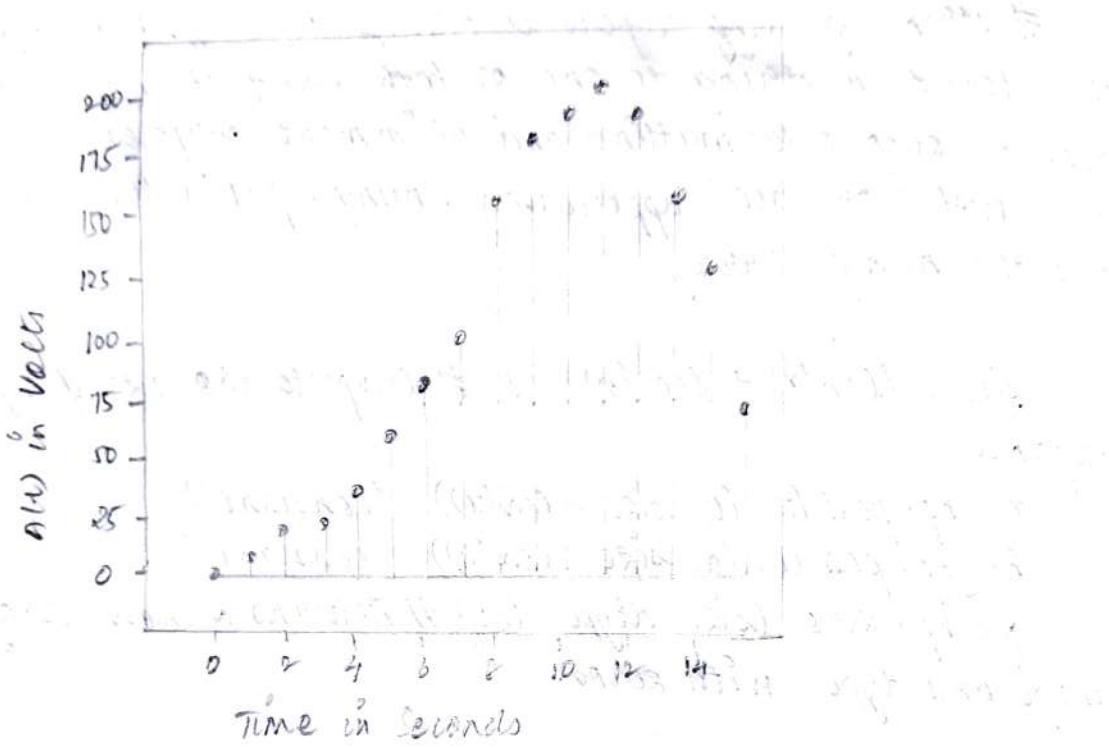
Case - II:-
start = time.time()

title_II = "Polyphase Filter Representation"
down_sampler_0 = down_sample(frequency_response, m)
up_sampler_0 = up_sample(down_sampler_0, m)
print("ED[n]: ", up_sampler_0)

phase_shift = m^-1
delay_signal = shift_samples(frequency_response, phase_shift)
down_sampler_1 = down_sample(delay_signal, m)
up_sampler_1 = up_sample(down_sampler_1, m)

```

## Polyphase Filter Representation in Time Domain



↳ This is a polyphase filter representation in the time domain.  
↳ It is a sequence of discrete voltage values.

↳ This is a sequence of discrete voltage values.

↳ This is a sequence of discrete voltage values.

↳ This is a sequence of discrete voltage values.

```

for i in range(m-1):
    up_sampler_1.remove(up_sampler_1[i])
    up_sampler_1.append(0)
print("E1[n]: ", up_sampler_1)

```

```

convolve_I = convolve(original_signal, up_sampler_0)
convolve_II = convolve(original_signal, up_sampler_1)
output_II = convolve_I + convolve_II
print("In The system response after summing up the individual
convolutions for " + str(title_II) + " is: ", output_II)
time_domain(output_II, title_II)
end = time.time()

```

```

print("The time of execution of " + str(title_II) + " is (in
seconds): ", end - start)

```

#### \* CODE: ORDER OF FILTER = 3)

```

import matplotlib.pyplot as plt # Provides an implicit way of
plotting

```

```

import numpy as np # Support for large, multi-dimensional
arrays and matrices

```

```

import time # Handle time-related tasks

```

# Plot time-domain spectrum of given signals:-

```

def time_domain(signal, plot_name):

```

```

    plt.xlabel("Time in Seconds")
    plt.ylabel("Alt) in Volts")
    plt.title("spectral Analysis of " + str(plot_name) + " in
Time Domain")
    plt.stem(np.arange(0, len(signal)), signal)
    plt.grid()
    plt.show()

```

# Down-sample the given signal by a factor of "m":-

```
def down_sample(signal, m):
    down_sampler = []
    down_sampler = []
```

# Copy the element value from original signal to down sampler array with the increment value set to "m":

```
for i in range(0, len(signal), m):
    down_sampler.append(signals[i])
return down_sampler
```

# Up-sample the given signal by a factor of "l":-

```
def up_sample(signal, l):
```

```
    up_sampler = []
    up_sampler = []
```

```
    for i in range(len(signal)):
```

up\_sampler.append(signals[i]) # Copy the element value from original signal to up sampler array

```
    if i != len(signal) - 1:
```

```
        for k in range(l-1):
```

up\_sampler.append(0) # Insert "l-1" zeros

between the elements of the array

```
    else:
```

```
        break
```

```
    return up_sampler
```

# Perform convolution,  $x[n] * h[n]$  :-

```
def convolution(x, h):
```

# 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.

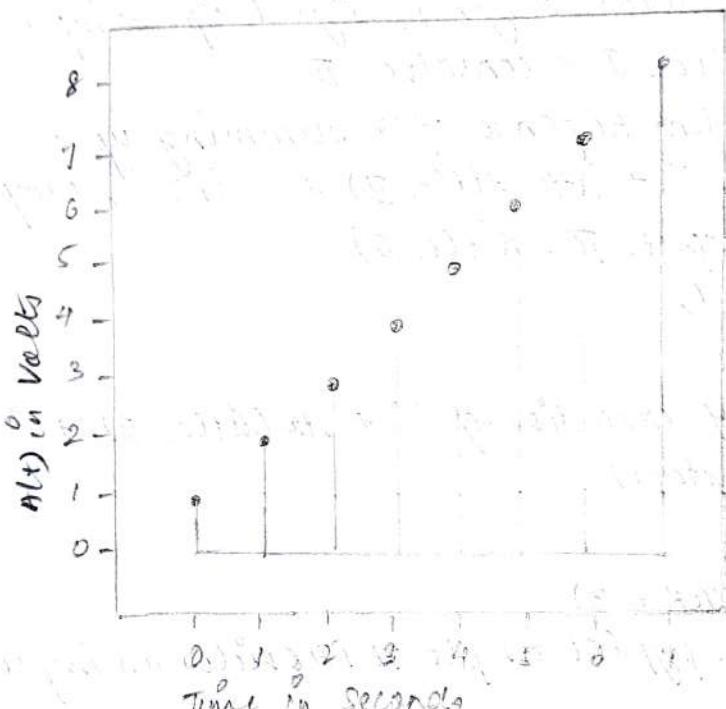
\* OUTPUT (CONSOLE & PLOTS): (ORDER OF FILTER = 3)

The original signal is  $\{1, 2, 3, 4, 5, 6, 7, 8\}$

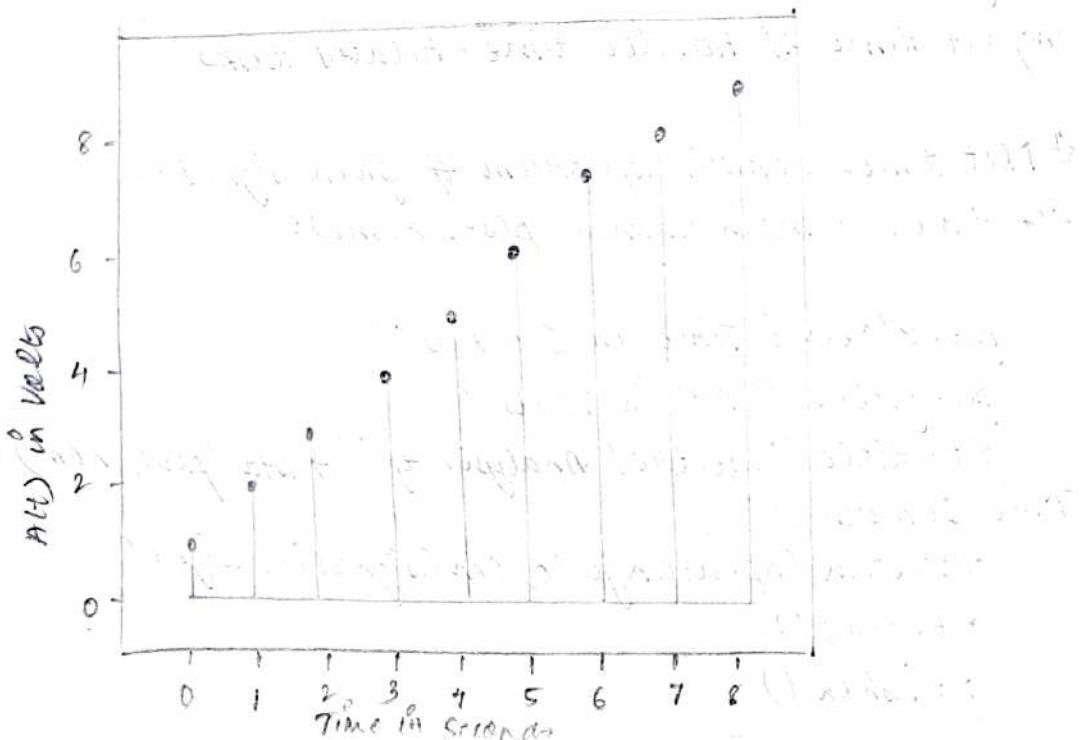
The frequency response is  $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

The order of filter is 3.

Original Signal in Time Domain



Frequency Response in Time Domain



```

size = (len(x) + len(h)) - 1 # Compute the size of system
response
x = np.pad(x, 10, size - len(x)), 'constant')
h = np.pad(h, 10, size - len(h)), 'constant')
y = np.zeros(size, dtype=int) # Returns a new array of given
shape and type, with zeros.

```

```

for i in range (size):
    for j in range (size):
        if i >= j:
            y[i] = int(y[i] + (int(x[i-j])) * int(h[j])))
return y # system Response y[n]

```

# Driver Code: main(); Execution starts here.

# Generate the original signal:-

```

title = "Original Signal"
original_signal = [1, 2, 3, 4, 5, 6, 7, 8]
print ("The original signal is: ", original_signal)
time_domain(original_signal, title)

```

# Generate the frequency response:-

```

title = "Frequency Response"
frequency_response = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print ("The frequency response is: ", frequency_response)
time_domain(frequency_response, title)

```

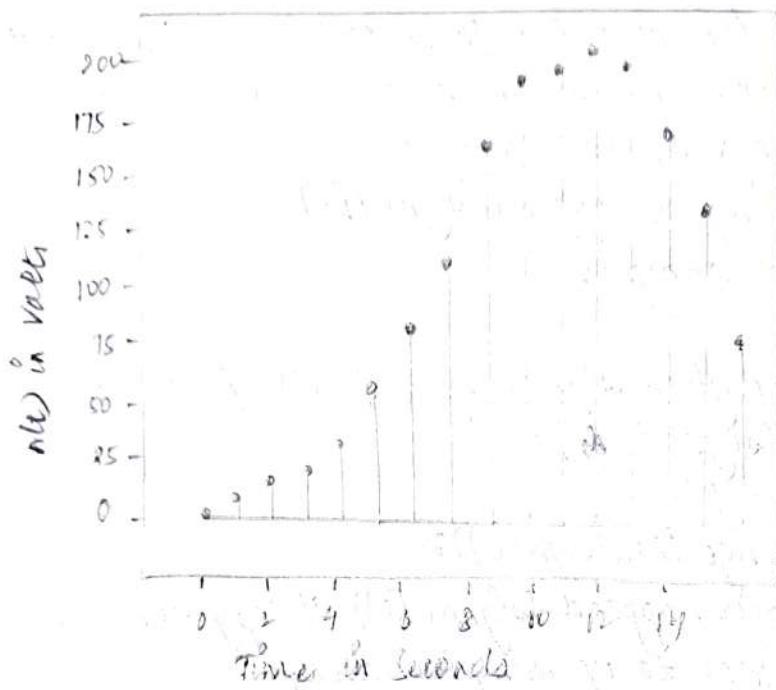
m = int(input("Enter the order of filter: "))

# Case - I:-

start = time.time()

title\_I = "Traditional Filter - Direct Representation"

## Traditional Filter - Direct Representation in Time Domain



The system response after convolution for 'Traditional Filter - Direct Representation' is : [1 4 10 20 35 56 84 120 156 182 197 201 190 166 121 72]

The time of execution of 'Traditional Filter - Direct Representation' is (in seconds) : 0.19136570526123047

$$E[0] := [1, 0, 0, 4, 0, 0, 7, 0, 0]$$

$$e[1] := [0, 1, 0, 0, 5, 0, 0, 8, 0]$$

$$e[2] := [0, 0, 3, 0, 0, 6, 0, 0, 9]$$

The system response after summing up the individual convolution for 'Polyphase Filter Representation' is : [1 4 10 20 35 56 84 10 156 182 197 200 190 166 121 72]

The time of execution of 'Polyphase Filter Representation' is (in seconds) : 0.16875076273945312

```

output_I = convolution(original_signal, frequency_response)
print("In the system response after convolution for '" + str(title_I) + "' is : ", output_I)
time_domain(output_I, title_I)
end = time.time()
print("The time of execution of '" + str(title_I) + "' is (in seconds) : ", end - start)

print("\n")

```

# Case-II :-

```
start = time.time()
```

title\_II = "Polyphase Filter Representation"

down\_sampler\_0 = down\_sample(original\_signal, m)

up\_sampler\_0 = up\_sample(down\_sampler\_0, m)

for i in range(m-1):

```
    up_sampler_0.append(0)
print("E0[n] : ", up_sampler_0)
```

pad\_frequency\_response = np.pad(frequency\_response, (m-1, 0), 'constant', constant\_values=0)

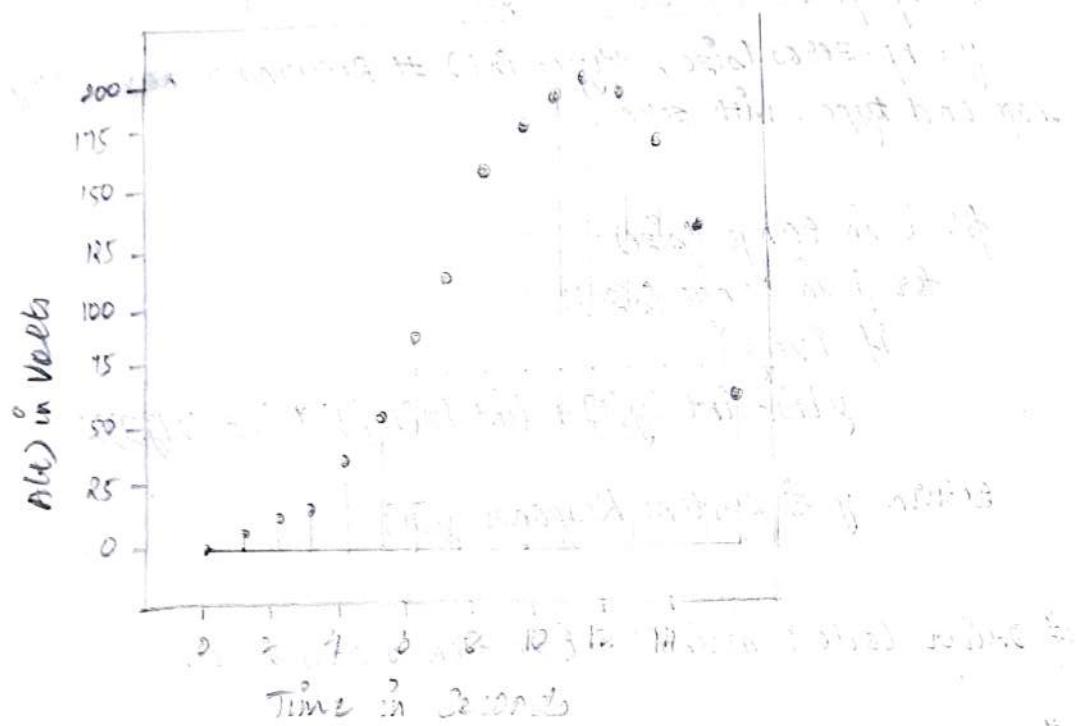
down\_sampler\_1 = down\_sample(pad\_frequency\_response, m)

up\_sampler\_1 = up\_sample(down\_sampler\_1, m)

for i in range(m-1):

```
    up_sampler_1.remove(up_sampler_1[i])
size_up_sampler_1 = len(up_sampler_0) - len(up_sampler_1)
for i in range(size_up_sampler_1):
    up_sampler_1.append(0)
print("E1[n] : ", up_sampler_1)
```

## Polyphase Filter Representation in Time Domain



```

pad-frequency-response = np.pad (frequency-response, (m-2, 0),
'constant', constant-values = 0)
down-sampler-2 = down-sample (pad-frequency-response, m)
up-sampler-2 = up-sample (down-sampler-2, m)
for i in range (m-2):
    up-sampler-2.remove (up-sampler-1[i])
print ("D[n] : ", up-sampler-2)

convolve-I = convolution (original-signal, up-sampler-0)
convolve-II = convolution (original-signal, up-sampler-1)
convolve-III = convolution (original-signal, up-sampler-2)
output-II = convolve-I + convolve-II + convolve-III
print ("In The system response after summing up the individual
convolutions for " + str(title-II) + " is : ", output-II)
time-domain (output-II, title-II)
end = time.time()
print ("The time of execution of " + str(title-II) + " is
(in seconds) : ", end-start)

```

#### \* RESULTS:

For the filters designed to be used in interpolators and decimators implemented the simplified polyphase architecture. Also, analyzed the complexity and latency of the implemented filters. All the simulation results were verified successfully.

LAB CODE AND NAME - 19CCE284 SOFTWARE DEFINED RADIO LAB  
 EXPERIMENT NUMBER - 5  
 DATE - 30/05/2022

### DESIGN AND ANALYSE FILTERS FOR GIVEN SPECIFICATIONS

#### \* AIM:

Given the order, cutoff frequency, and filter type, analyze the filter response and magnitude response. Also, include finite word length effects in the analysis.

#### \* SOFTWARE REQUIRED:

Synder IDE - Python 3.9

#### \* CODE:

```
import matplotlib.pyplot as plt # Provides an implicit way of plotting
import numpy as np # Support for large, multi-dimensional arrays
and matrices
```

```
import time # Handle time-related tasks
```

```
# Compute DFT coefficients using linear transformation method:-
```

```
def DFT(x, plot_name, round_off):
```

```
start = time.time()
```

```
# Compute N(N) 1D Array :-
```

```
z1 = c1 = len(x)
```

```
wn = []
```

```
for i in range (n1):
```

```
    for j in range (n1):
```

```
        wn.append(np.exp(-2j * np.pi * i * j / len(x)))
```

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

Wn. multidiim = np. reshape (wn, (1, c)) # An  $N \times N$   $N(N)$  matrix  
 $\sigma_2 = \text{exp}(\pi) : \omega^2 \approx 1$

$$n_2 = \text{len}(W); \quad \omega^2 = 1$$

- multidims = np. reshape (x, (n2, n1)) # An  $N \times 1 \times N$  matrix

# compute  $x(N) = w(N)^* x(N)$ , an  $N \times 1$  matrix

fourier-transform-multidim =  $[10]^\dagger [02]^\dagger \alpha_1 \# \text{null}$

## Multidimensional Array Declaration

```
fourier_transform -l -t = [] # convert Multidimensional Array
```

卷之二

for  $i$  in range (n):

for  $j$  in range (0,2):

fourier-transform multidims  $[i][j] = 0$

for  $k$  in range (0, 5):

fourier-transform-multidim[i][j] += nn\_multidim[i][k] +

float [ $\alpha$ -multidims [k][j])

```
fourier_transform_l_t.append(around(abs(fourier_transform.  
multidim[i][j]), round_off))
```

```
plt.xlabel('Frequency in Hertz')
```

plt. ylabel ('A(f) in Volts')

```
plt.title (" "+ str(plt_name) + " in Frequency Domain")
```

plot plot (np.array(10), np.array(np.exp(-l-t)))

fourier-transform- $e^{-t}$ )

pet. grid (True)

```
plt.show()
```

end = time.time()

print('The time taken for execution of ' + str(plate-name) + ' Design (in seconds) is : ', end-start)

MAIN MENU

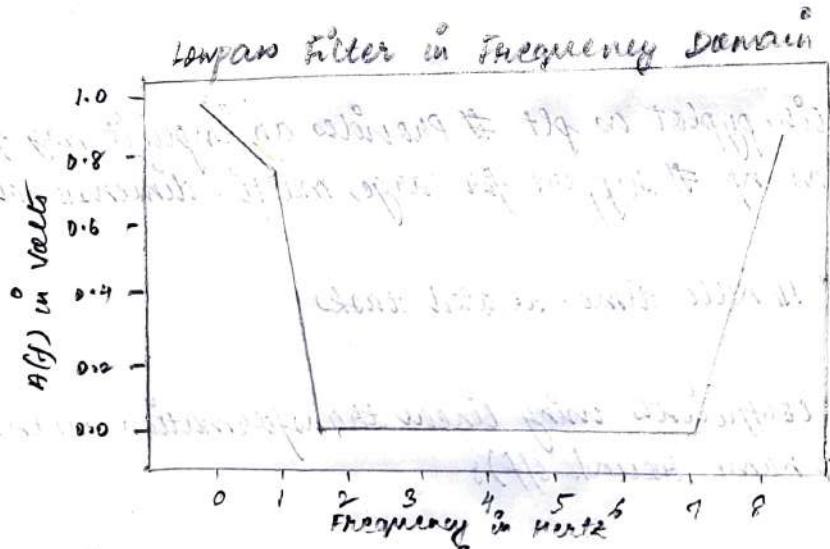
1. Lowpass Filter
2. Highpass Filter
3. Bandpass Filter
4. Bandstop Filter
5. Exit

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

Enter the order of the filter: 9

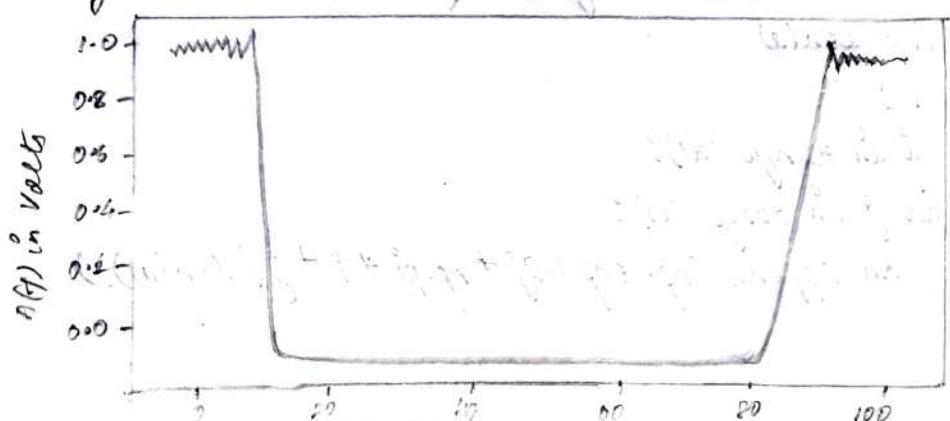
The truncation value for finite word analysis is taken to be

Enter the value of wc coefficient for the lowpass filter: 0.3



The time taken for execution of 'Lowpass Filter Design' in second  
is: 0.10024905204772949

similarly, for order of the filter: 100



```
def plot_desired_impulse_response(hd_n, plot_name):
```

```
plt.xlabel("Frequency in Hertz")
plt.ylabel("A(f) in Volts")
plt.title("Desired Impulse Response of " + str(plot_name))
plt.stem(np.arange(0, len(hd_n)), hd_n)
plt.grid(True)
plt.show()
```

```
def lowpass_filter(N, round_off):
```

```
title = "Lowpass Filter"
w_c = float(input("Enter the value of w_c coefficient for the "
+ str(title) + ":"))
w_c = w_c * np.pi
```

```
hd_n = [] # Desired Frequency Response - Numpy Array Declaration
for n in range(-N//2+1, N//2+1, 1):
```

If  $n=0$ :

```
hd_n.append((w_c + np.sin(w_c * n)) / np.pi) # Geometric
Interpretation
```

else:

```
hd_n.append(np.sin(w_c * n) / (np.pi + n)) # computation
purposes.
```

```
plot_desired_impulse_response(hd_n, title)
DFT(hd_n, title, round_off)
```

```
def highpass_filter(N, round_off):
```

```
title = "Highpass Filter"
w_c = float(input("Enter the value of w_c coefficient for the "
+ str(title) + ":"))
```

MAIN MENU

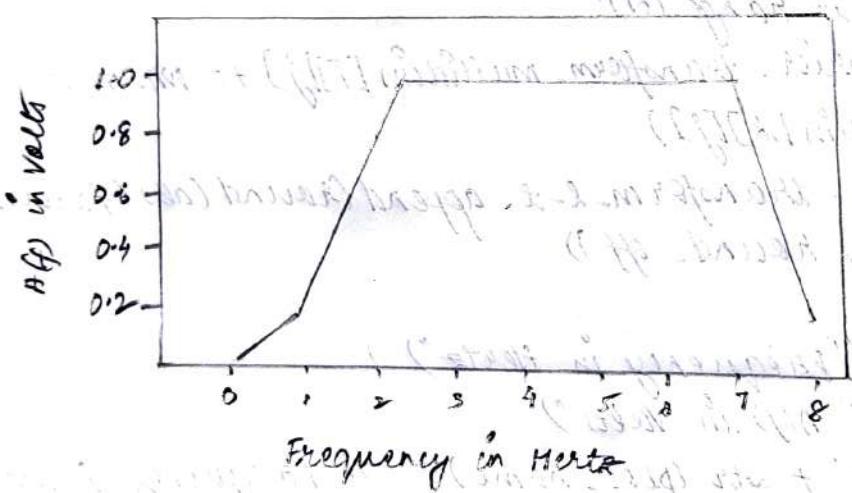
1. Lowpass Filter
2. Highpass Filter
3. Bandpass Filter
4. Bandstop Filter
5. Exit

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

Enter the order of the filter : 9  
The truncation value for finite word length analysis is taken to be 4.

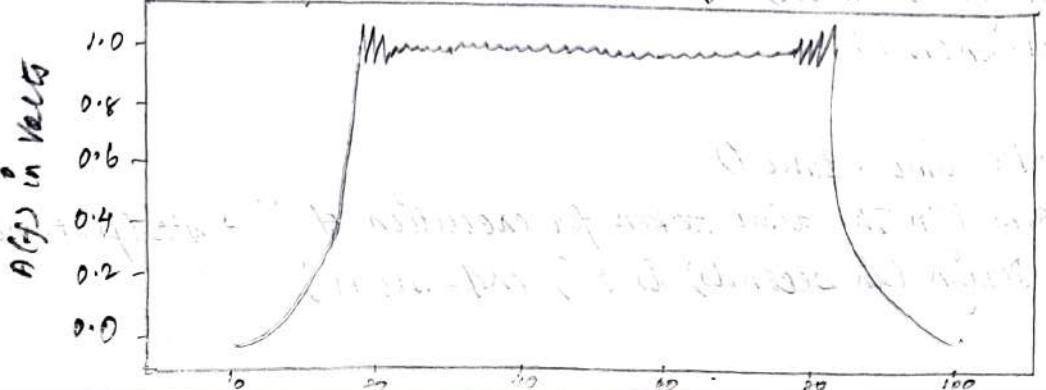
Enter the value of  $\omega_0$  coefficient for the Highpass Filter : 0.3

Highpass Filter in Frequency Domain



Similarly, for order of the filter : 100

Highpass Filter in Frequency domain



$$w_c = w_c * np.pi^{\circ}$$

hd\_n = [] # Desired Frequency Response - Null Array Declaration

for n in range (-N//2 + 1, N//2 + 1, 1):  
if n == 0:

hd\_n.append ((np.sin((np.pi \* n)) - (w\_c + np.sin(w\_c + n)) / (np.pi))) # Geometric Interpretation  
else:

hd\_n.append ((np.sin((np.pi) \* n)) - (np.sin(w\_c + n)) / (np.pi) \* n)) # computation purposes

plot\_desired\_impulse\_response (hd\_n, title)  
DFT (hd\_n, title, round-off)

def bandpass\_filter (N, round-off):

title = "Bandpass Filter"

w\_h = float (input ("Enter the value of wH coefficient for the" + str (title) + ":"))

w\_h = w\_h + np.pi

w\_l = float (input ("Enter the value of wL coefficient for the" + str (title) + ":"))

w\_l = w\_l + np.pi

hd\_n = [] # Desired Frequency Response - Null Array Declaration

for n in range (-N//2 + 1, N//2 + 1, 1):  
if n == 0:

hd\_n.append ((w\_h \* np.sin (w\_h \* n) / np.pi) - (w\_l + np.sin (w\_l \* n) / np.pi)) # Geometric Interpretation  
else:

MAIN MENU

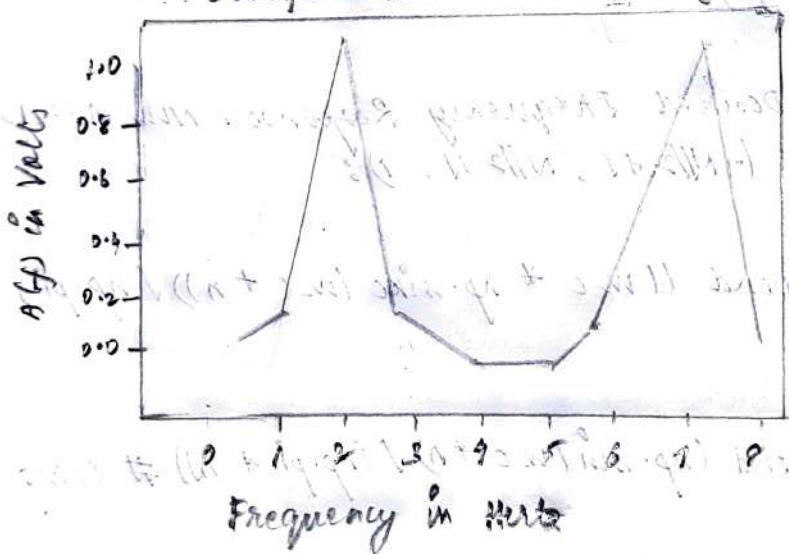
1. Lowpass Filter
2. Highpass Filter
3. Bandpass Filter
4. Bandstop Filter
5. Exit

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

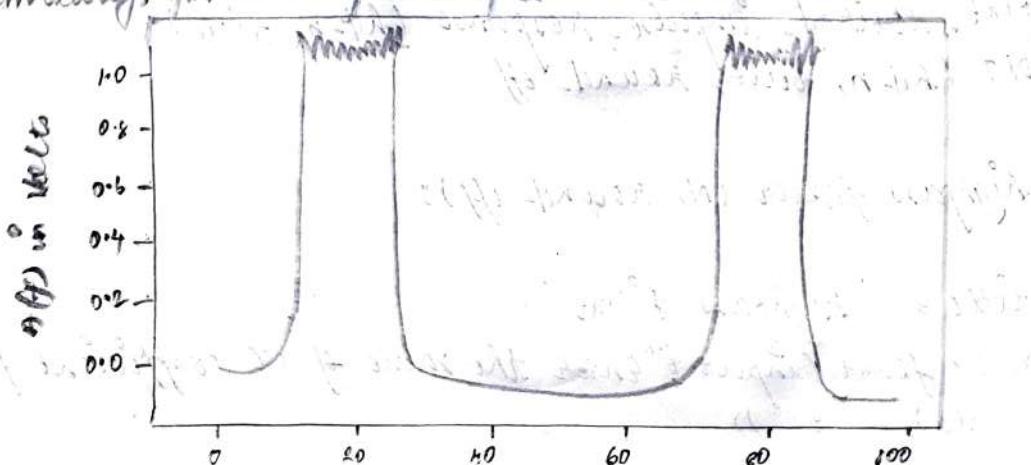
Enter the order of the filter : 9  
The truncation value for finite word length analysis is taken to be 4.

Enter the value of  $w_0$  coefficient for the Bandpass Filter : 0.6  
Enter the value of  $w_1$  coefficient for the Bandpass Filter : 0.3

Bandpass Filter in Frequency Domain



Similarly, for order of the filter : 100



$(np.pi + n)) \# computation purposes$

`plot_desired_impulse_response(hd-n, title)`

`DFT(hd-n, title, round-off)`

`def bandstop_filter(N, round-off):`

`title = "Bandstop Filter"`

`w_h = float(input("Enter the value of wH coefficient for the " + str(title) + ": "))`

`w_d = w_h + np.pi`

`w_l = float(input("Enter the value of wL coefficient for the " + str(title) + ": "))`

`w_d = w_d + np.pi`

`hd-n = [] # Desired Frequency Response - Null array declaration`

`for n in range(-N//2+1, N//2+1, 1):`

`if n == 0:`

`hd-n.append((np.sin(np.pi + n)) + (w_d + np.sin(w_d + n) * np.pi) - (w_h + np.sin(w_h + n)) * np.pi))`

`# Geometric Interpretation`

`else:`

`hd-n.append(((np.sin(np.pi + n)) + (np.sin(w_d + n)) - (np.sin(w_h + n))) / (np.pi + n)) \# computation purposes`

`plot_desired_impulse_response(hd-n, title)`

`DFT(hd-n, title, round-off)`

`# Driver Code: main(); Execution starts here.`

### MAIN MENU

1. Lowpass Filter
2. Highpass Filter
3. Bandpass Filter
4. Bandstop Filter
5. Exit

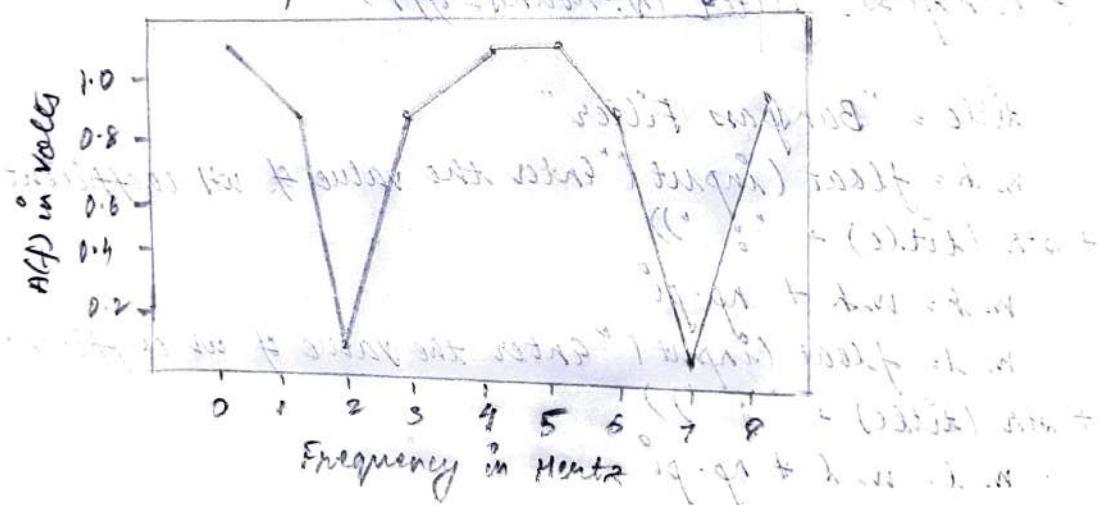
Enter the number corresponding to the menu to implement the choice : 4

Enter the order of the filter : 9

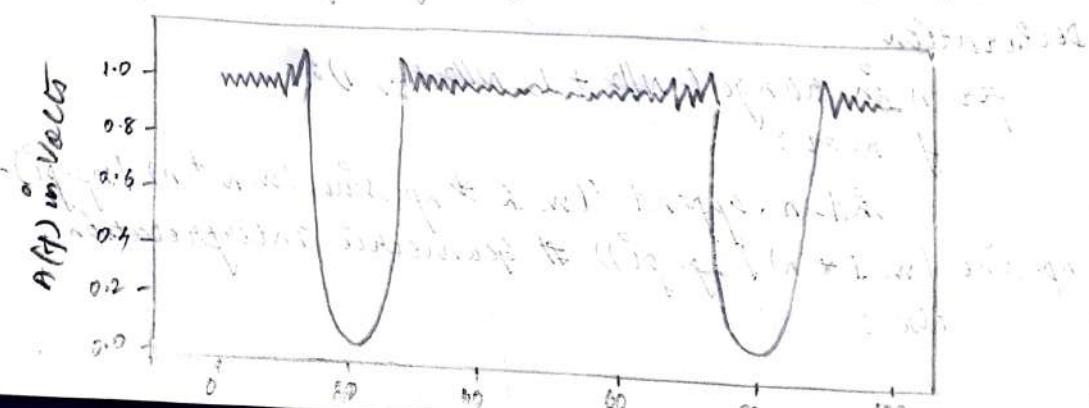
The truncation value for finite word length analysis is taken to be 4.

Enter the value of  $w_L$  coefficient for the Bandstop Filter : 0.6  
Enter the value of  $w_R$  coefficient for the bandstop filter : 0.3

### Bandpass Filter in Frequency Domain



Similarly, for order of the filter : 100



while True: # This simulates a Do Loop

```
print ("\\n")
```

```
heading = "MAIN MENU"
```

```
print ('{:=^50}'.format ('' * 50).join (heading.center(50))))
```

```
choice = input ()
```

```
    1. Lowpass Filter \\n 2. Highpass Filter \\n 3.
```

```
Bandpass Filter \\n 4. Bandstop Filter \\n 5. Exit \\n Enter the  
number corresponding to the menu to implement the choice : ")
```

# Menu Driven Implementation

```
# str() returns the string version of the variable "choice"
```

```
if (choice == str(1) or choice == str(2) or choice == str(3) or  
choice == str(4)):
```

```
N = int(input ("\\nEnter the order of the filter : "))
```

```
print ("The truncation value for finite word length analysis  
is taken to be 4. ")
```

```
round_off = 4
```

```
if choice == str(1):
```

```
    lowpass_filter (N, round_off)
```

```
elif choice == str(2):
```

```
    highpass_filter (N, round_off)
```

```
elif choice == str(3):
```

```
    bandpass_filter (N, round_off)
```

```
elif choice == str(4):
```

```
    bandstop_filter (N, round_off)
```

```
elif choice == str(5):
```

```
    break # Exit loop
```

else :

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

\* RESULT:

For the given order, cutoff frequency, and filter type, analyzed the filter response and magnitude response. Also, included finite word length effects in the analysis. All the simulation results were verified successfully.

LAB CODE AND TITLE : 19CCE284 - SOFTWARE DEFINED RADIO LAB  
 EXPERIMENT NUMBER : 6  
 DATE : 06/06/2022

### WAVELETS :

\* AIM:  
 Develop a code for signal denoising using Haar wavelets. Analyse the effectiveness of denoising for varying levels of decomposition. Repeat the analysis for varying degrees of noise added to the clear sinusoidal signal.

\* SOFTWARE REQUIRED :

Synder IDE - Python 3.9

\* CODE:

```
import matplotlib.pyplot as plt # Provides an implicit way of plotting
import numpy as np # support for large, multi-dimensional arrays and matrices
import scipy.fftpack as sp # module to calculate discrete fast Fourier transform
```

# Compute DFT coefficients using linear transformation method:-  
 def DFT(x, plot\_name):

# Compute w(N) 1D array :-

nl = cl = len(x)

wn = []

for i in range (nl):

    for j in range (cl):

        wn.append (np.exp (-2j \* np.pi \* i \* j / len(x)))

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

wn\_multidim = np.reshape(wn, (n1, c1)) # An N\*N w(N) matrix  
 $n_1 = \text{len}(x); c_1 = 1$

x\_multidim = np.reshape(x, (n2, c2)) # An N\*1 x(N) matrix

# Compute  $x(N) = w(N) * x(N)$ , an N\*1 matrix :-

fourier\_transform\_multidim = [[0]\*c2]\*n1 # Null

Multidimensional Array Declaration

fourier\_transform\_l\_t = [] # Convert multidimensional array to 1D

for i in range(n1):

    for j in range(c2):

        fourier\_transform\_multidim[i][j] = 0

        for k in range(c1):

            fourier\_transform\_multidim[i][j] += wn\_multidim[i][k] \* float(x\_multidim[k][j])

        fourier\_transform\_l\_t.append(abs(fourier\_transform\_multidim[i][j]))

plt.xlabel("Frequency (in Hertz)")

plt.ylabel("Amplitude (in Volts)")

plt.title(plot\_name + " in Frequency Domain")

plt.stem(np.arange(0, len(fourier\_transform\_l\_t)), fourier\_transform\_l\_t)

plt.grid(True)

plt.show()

# Down-sample the given signal by a factor of "m":-

def down\_sample(signal, m):

    down\_sampler = []

# Copy the element value from original signal to down sampler array with the increment value set to "m":-

```
for i in range (0, len(signal), m):
```

```
    down-sampler.append (signal[i])
```

```
return down-sampler
```

# Up-sample the given sample by a factor of "l":-

```
def up-sample (signal, l):
```

```
    up-sampler = []
```

```
    for i in range (len(signal)):
```

```
        up-sampler.append (signal[i]) # Copy the element value  
from original array to up sampler array
```

```
        if i != len(signal):
```

```
            for k in range (l-1):
```

```
                up-sampler.append (0) # Insert "l-1" zeros between  
the elements of the array
```

```
        else:
```

```
            break
```

```
    return up-sampler
```

# Perform convolution,  $x[n] * h[n]$ :-

```
def convolution (x, h):
```

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

```
size = (len(x) + len(h)) - 1 # Compute the size of system response
```

```
x = np.pad (x, (0, size - len (x)), 'constant')
```

```
h = np.pad (h, (0, size - len (h)), 'constant')
```

```
y = np.zeros (size, dtype = float) # Returns a new array of  
given shape and type, with zeros.
```

```

for i in range (size):
    for j in range (size):
        if i >= j:
            y[i] = float (y[i] + (float (x[i-j]) * float (A[j])))
return y # system response y[n]

```

# Driver Code : main(); Execution starts here.

# Generate the sine signal:-

```

fs = int (input ("Enter the desired sampling frequency (in Hertz): "))
time_axis = np.arange (0, 1, 1/fs) # Define the time axis
sine_frequency = int (input ("Enter the frequency of the sine
signal (in Hertz): "))
original_signal = np.sin (2 * np.pi * sine_frequency * time_axis)

```

title = "Original Signal"

plt.xlabel ("Time in Seconds")

plt.ylabel ("Alt) in Volts")

plt.title (" + str (title) + " in Time Domain")

plt.stem (time\_axis, original\_signal)

plt.grid (True)

plt.show()

DFT (original\_signal, title)

print ("The spectral analysis of the original signal in time and frequency domain is shown in the above plots.")

# Generate a Noise :- Draw random samples from a normal distribution)

y = np.random.normal (0, 0.2, np.size (original\_signal)); # Additive Mixed Gaussian Noise

original\_signal = np.add (original\_signal, y)

```

plt.title ("Histograms Representation - Gaussian Noise")
plt.hist(y) # Plot a histogram
plt.grid (True)
plt.show()

```

# Plot the Noisy signal:-

```

plt.ylabel ('Amplitude (in Volts)')
plt.title ('Noisy Sinusoidal Wave')
plt.stem(original_signal)
plt.grid (True)
plt.show()

```

# Perform spectral Analysis:-

```

x-f = abs (sf. fft (original_signal)) # Return the absolute value of
the fast Fourier transform
l = np.size (original_signal)
fr = (fs/2) * np.linspace (0, 1, int (l/2))
al-m = (2/l) * abs (x-f [0 : np.size (fr)]) ; # Compute Magnitude Spectrum

```

# Plot Magnitude spectrum:-

```

plt.title ('Spectrum of Noisy Signal')
plt.xlabel ('Frequency (in Hertz)')
plt.ylabel ('Magnitude (in dB)')
plt.stem (fr, 20 + np.log10 (al-m))
plt.grid (True)
plt.show()

```

Print ("In the spectral analysis of the noisy signal in time and  
frequency domain is shown in the above plots.")

$h_1 = [1/\sqrt{2}, j/\sqrt{2}]$ ;  $h_2 = [1/\sqrt{2}, -j/\sqrt{2}]$   
# same coefficients

```

g1 = [1/np.sqrt(2), -1/np.sqrt(2)]; g2 = [-1/np.sqrt(2), 1/np.sqrt(2)]
# complementary coefficients
l_m_factor_expander = 2

```

decomposition = int(input("In enter the number of levels of decomposition : "))

scaling\_level = int(input("Enter the scaling level, the wavelet noise should be removed (< level of decomposition): "))

for i in range(decomposition):

# Analyze:-

a = convolution(original\_signal, h1); p = convolution(original\_signal, g1)

b = down-sample(a, l\_m\_factor\_expander); q = down-sample(p, l\_m\_factor\_expander)

# synthesize:-

c = up-sample(b, l\_m\_factor\_expander); r = up-sample(q, l\_m\_factor\_expander)

d = convolution(c, h2); s = convolution(r, g2)

# Add :-

denoised\_signal = np.add(d, s)

if i == (decomposition - scaling\_level):

denoised\_signal = d

title = 'Denoised Signal'

plt.ylabel('Amplitude (in Volts)')

plt.title(title);

plt.stem(denoised\_signal)

plt.grid(True)

plt.show()

DFT (denoised - signal, title)

point ("In The spectral analysis of the denoised signal in time and frequency domain is shown in the above plots.")

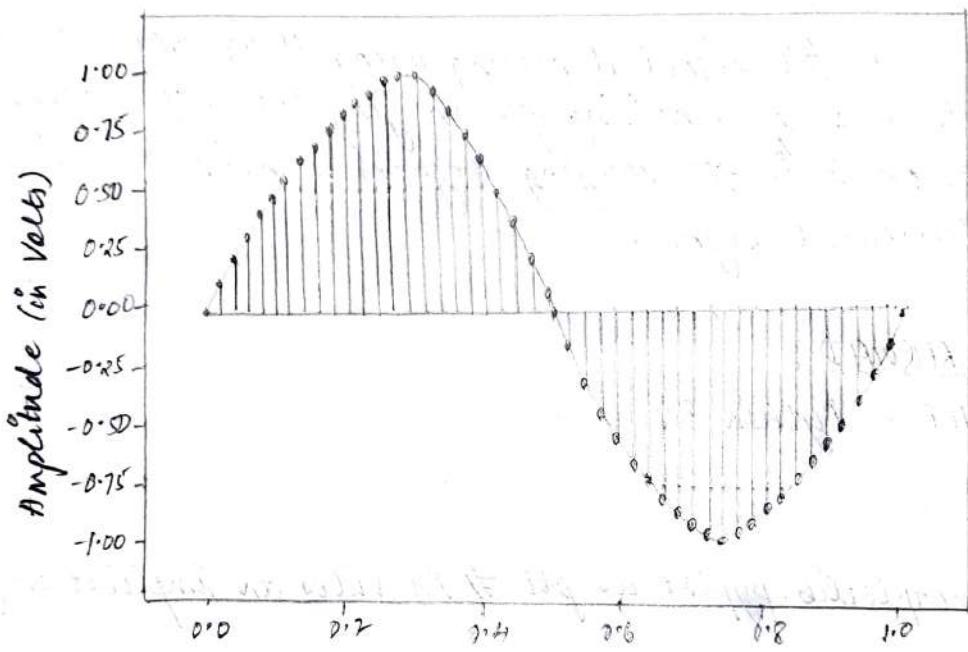
\* RESULT:

for the given sinusoidal signal, performed wavelet decomposition and reconstruction using Haar wavelet system and verified the perfect reconstruction property of wavelets. [one level decomposition using filter bank concept] All simulation results were verified successfully.

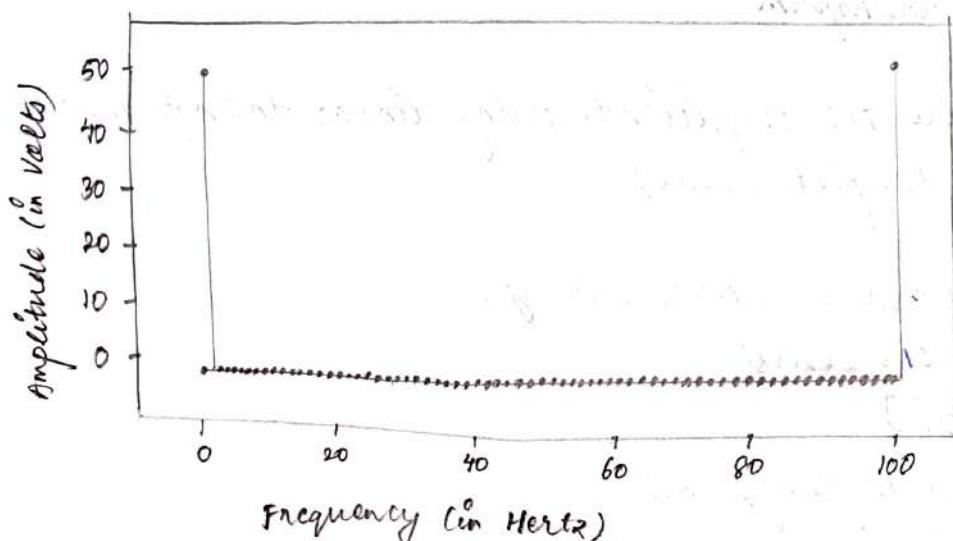
## OUTPUT - CONSOLE WINDOW & PLOTS

Enter the desired sampling frequency (in Hertz) : 100  
Enter the frequency of the sine signal (in Hertz) : 1

original signal in Time Domain

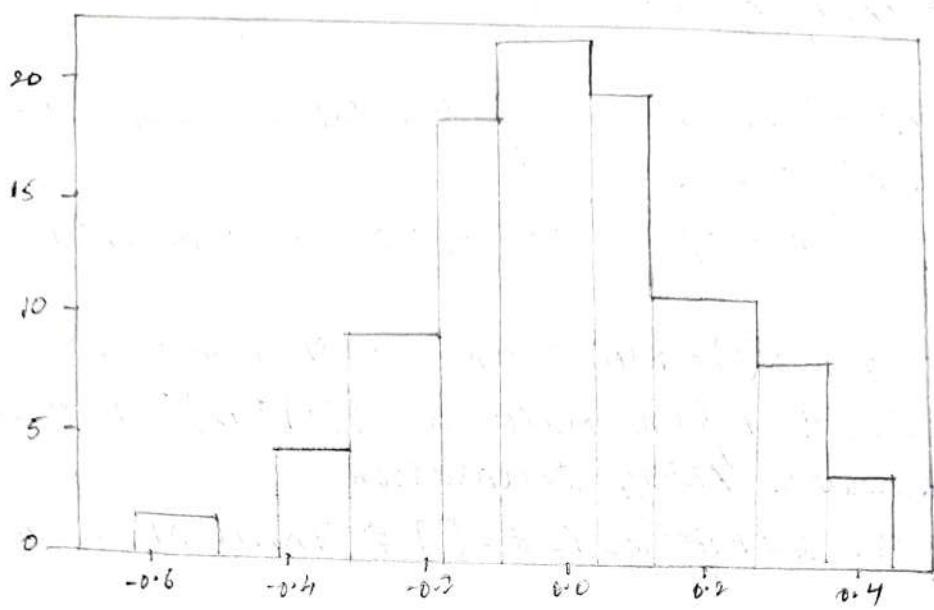


original signal in Frequency Domain

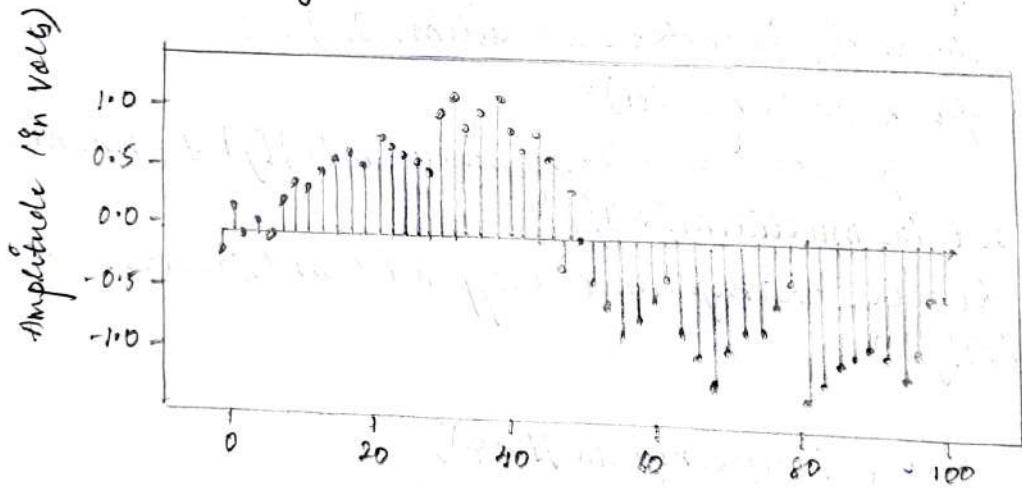


The spectral analysis of the original signal in time and frequency domain is shown in the above plots.

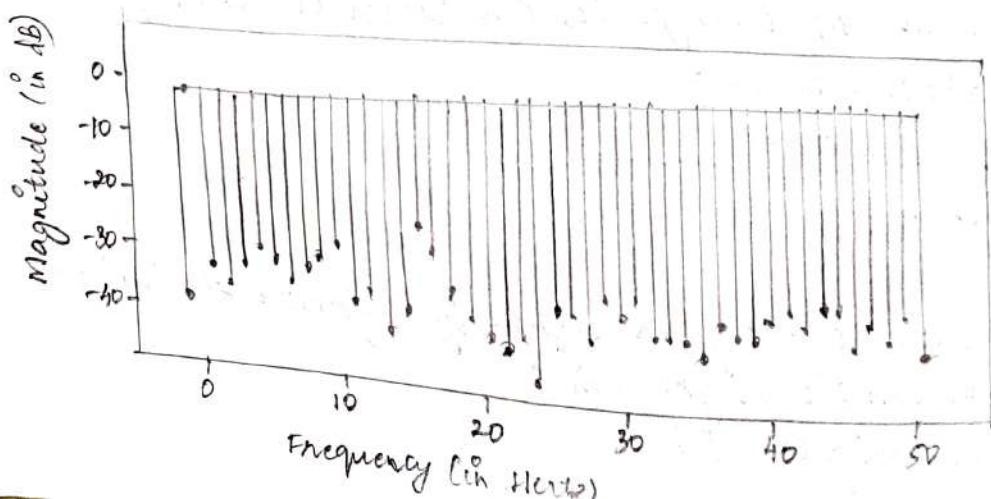
Histogram Representation - Gaussian noise



Noisy Sinusoidal wave

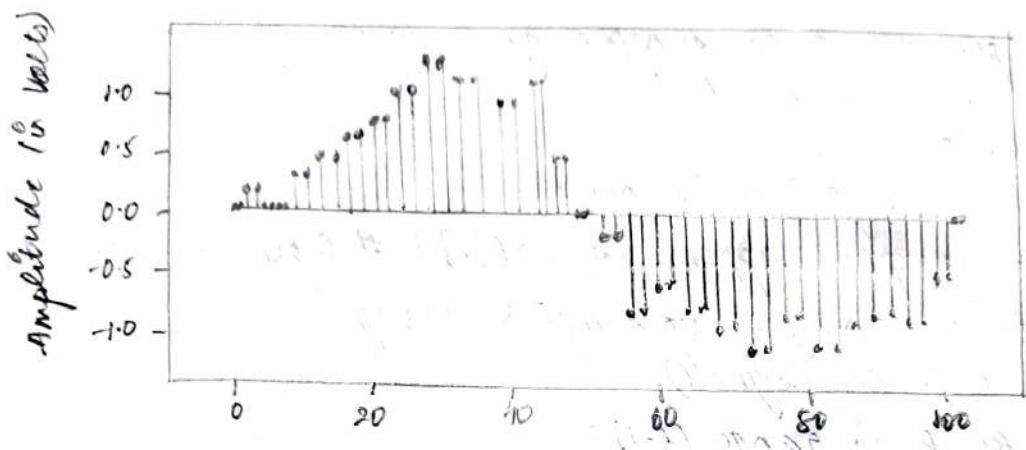


Spectrum of Noisy signal

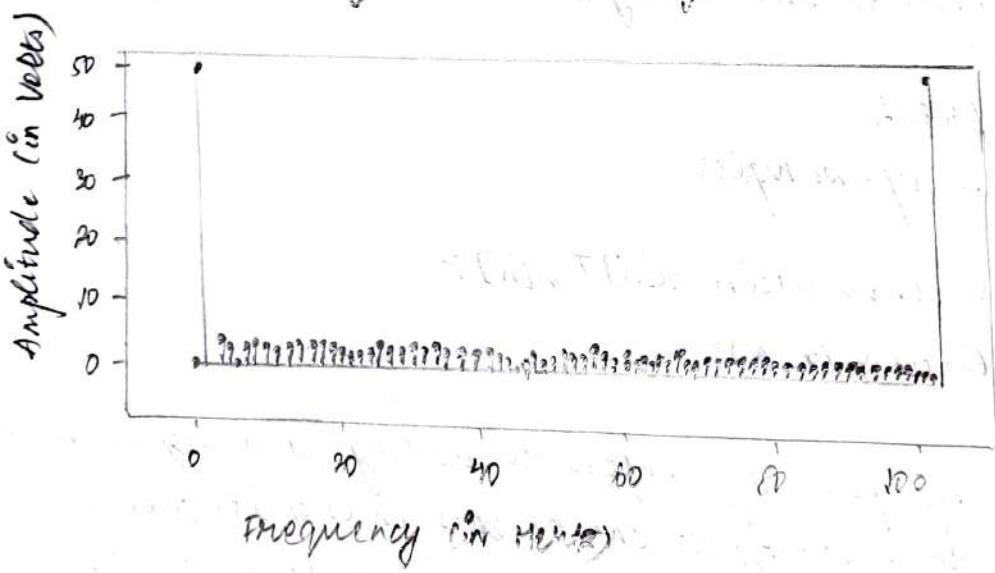


Enter the number of levels of decomposition : 100  
Enter at which scaling level, the wavelet noise should be removed (level of decomposition) : 1

Denoised Sinusoidal Signal



Denoised Signal in Frequency Domain



The spectral analysis of the denoised signal in time and frequency domain is shown in the above plots.

LAB CODE AND TITLE: 19CCE284 - SOFTWARE DEFINED RADIO LAB  
EXPERIMENT NUMBER: 7  
DATE: 20/06/2022

\* AIM:

Design and analyse a simple communication system using software defined radio (SDR) on Gnu Radio Companion.

\* FLOW GRAPH:

$\overrightarrow{Pn}$

### Options

Title: Not titled yet  
Output language: Python  
General options: gqrx

1D: samp. rate  
Value: 32 k

Signal source  
sample rate: 32 k  
waveform: sine  
frequency: 1 k  
amplitude: 2  
offset: 0  
initial phase (radians): 0

Throttle  
in sample rate: 32 k  
out

GT GUS Frequency Sink  
Name: Message S... Time domain  
Number of Points: 1.024 k  
Sample Rate: 32 k  
Interpolate: No  
in

Multiply  
in out  
in out

GT GUS Frequency Sink  
Name: Carrier ... Time domain  
Number of Points: 1.024 k  
Sample Rate: 32 k  
Interpolate: No  
in

Throttle  
in sample rate: 32 k  
out

Multiply  
in out

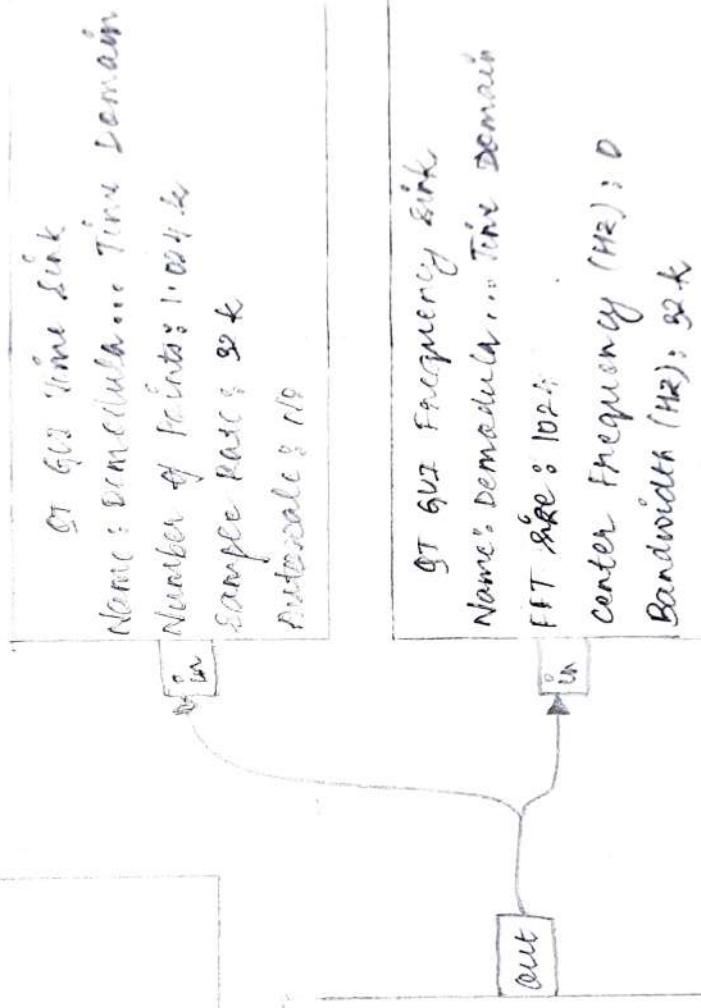
GT GUS Frequency Sink  
Name: Message S... Time domain  
Number of Points: 1.024 k  
Sample Rate: 32 k  
Interpolate: No  
in

GT GUS Frequency Sink  
Name: Carrier ... Time domain  
Number of Points: 1.024 k  
Sample Rate: 32 k  
Interpolate: No  
in

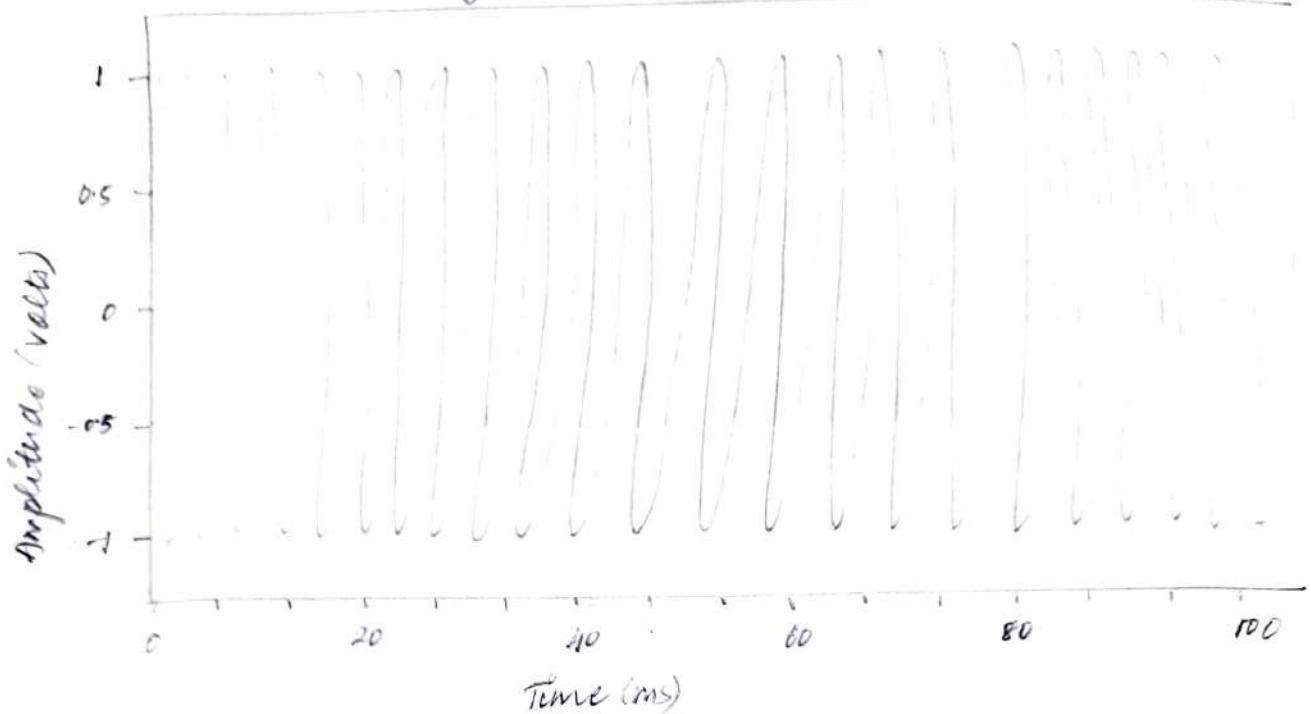
GT GUI Time Sink  
Name: modulate... Time Domain  
Number of Points: 1.024 k  
Sample Rate: 32 k  
Autoscale: No

GT GUI Frequency Sink  
Name: modulate... Frequency Domain  
FFT size: 1024  
Center Frequency (Hz): 0  
Bandwidth (Hz): 32 k

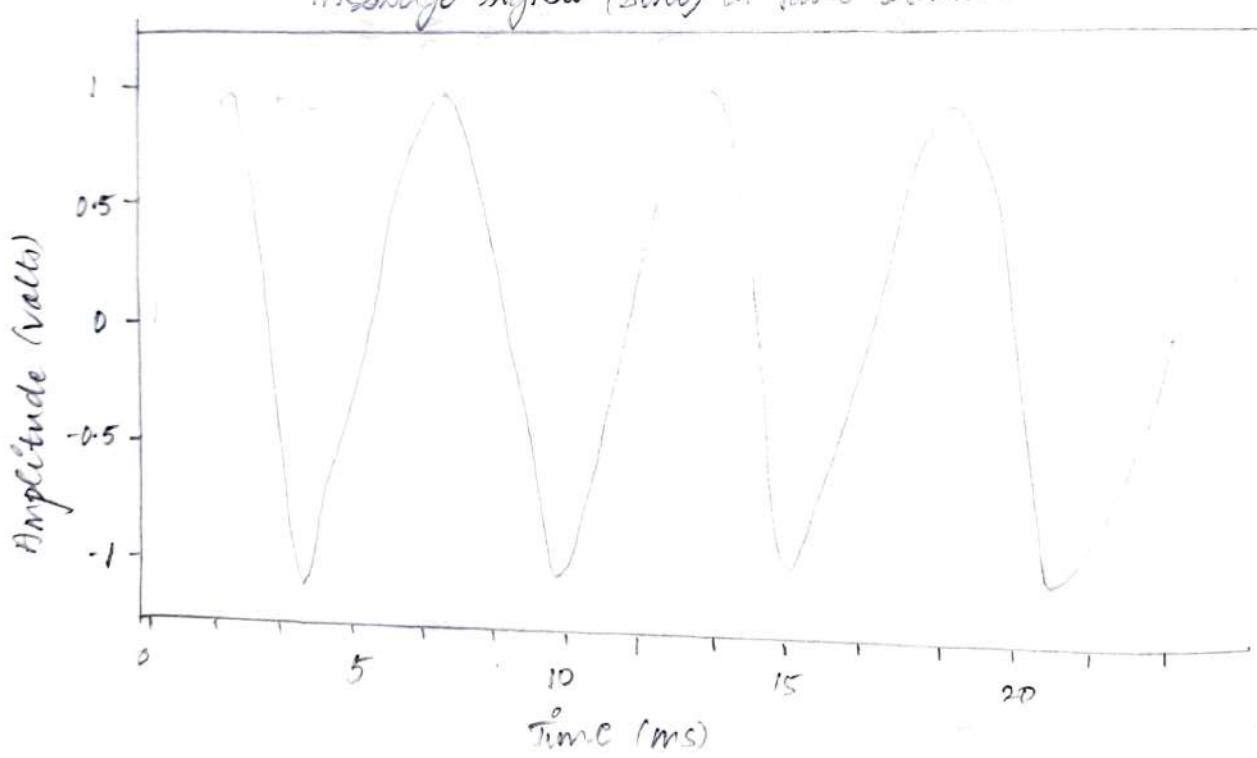
Low Pass Filter  
Decimation: 1  
Gain: 1  
Sample Rate: 32 k  
Cutoff Freq: 1.5 k  
Transition Width: 500  
Window: Rectangular  
Beta: 6.76



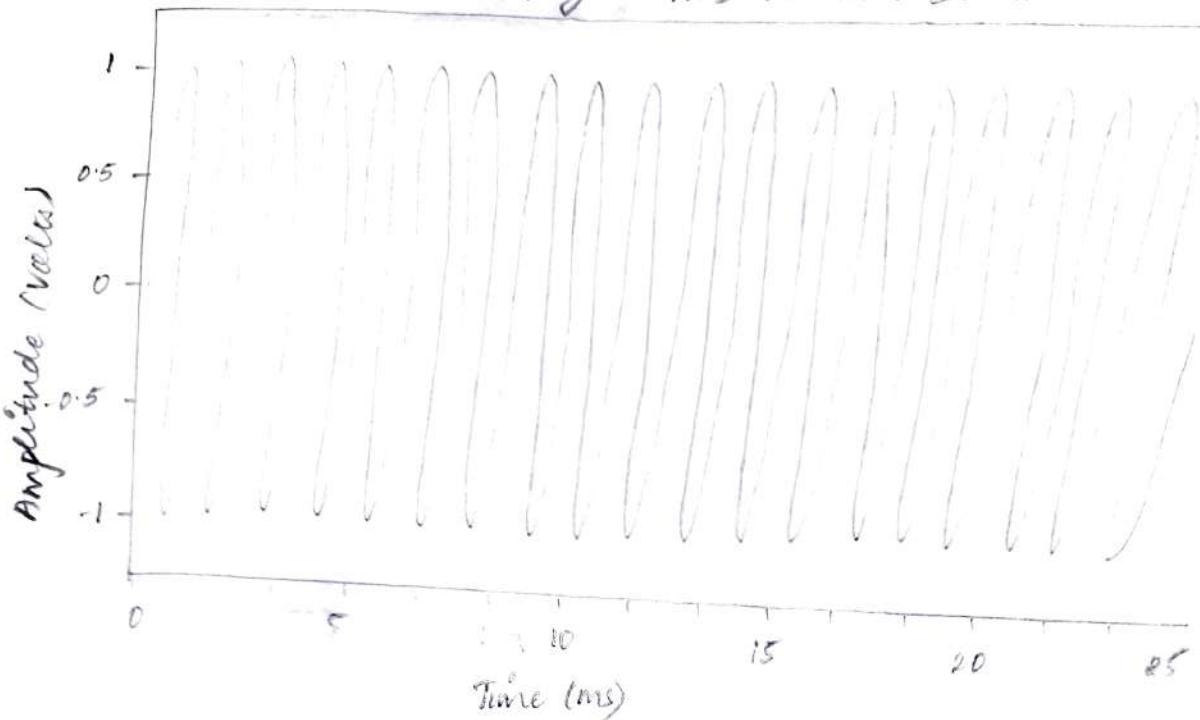
carrier signal (cosine) in Time Domain



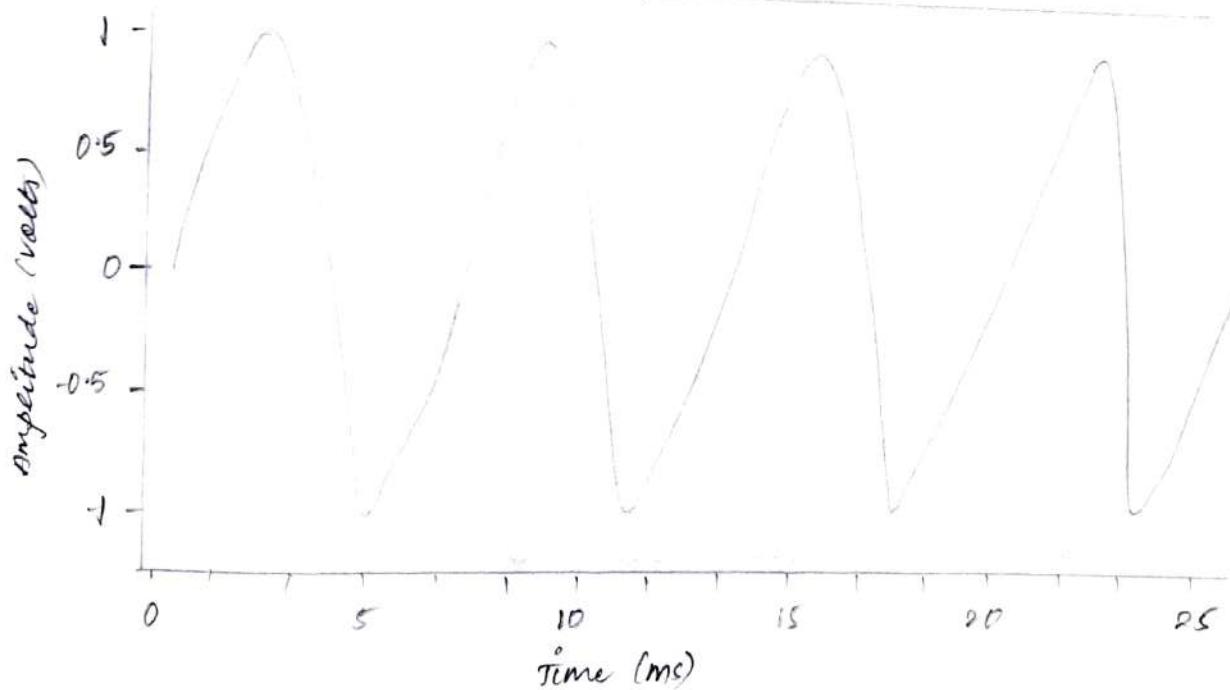
message signal (sinc) in Time domain



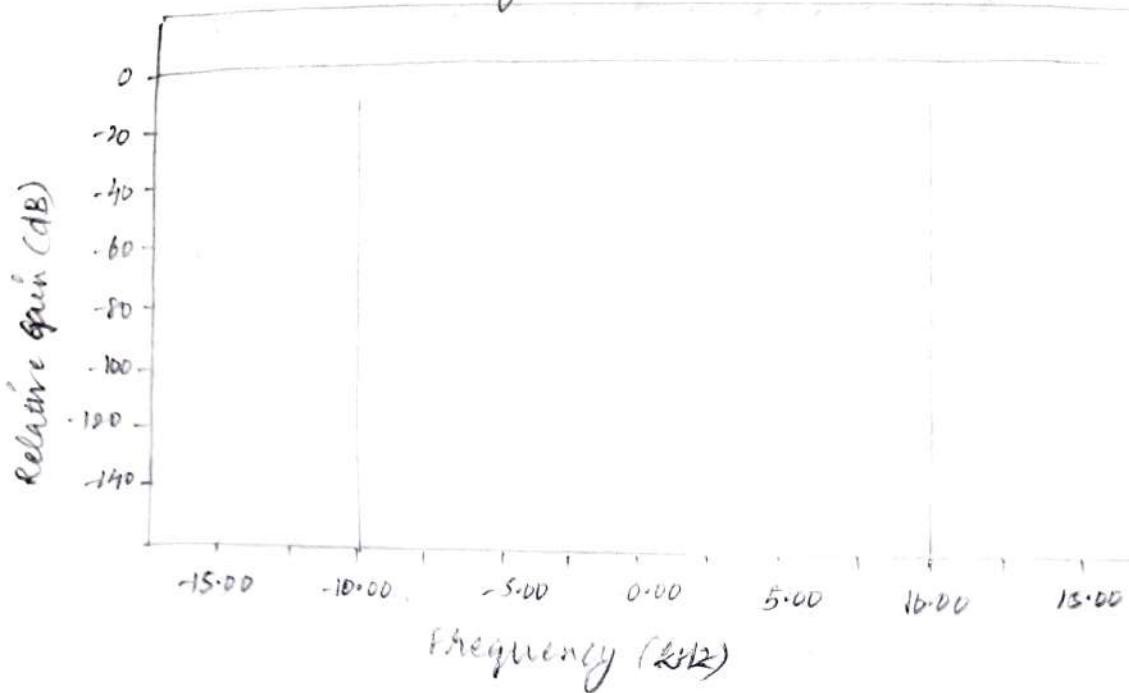
modulated signal w(t) in Time domain



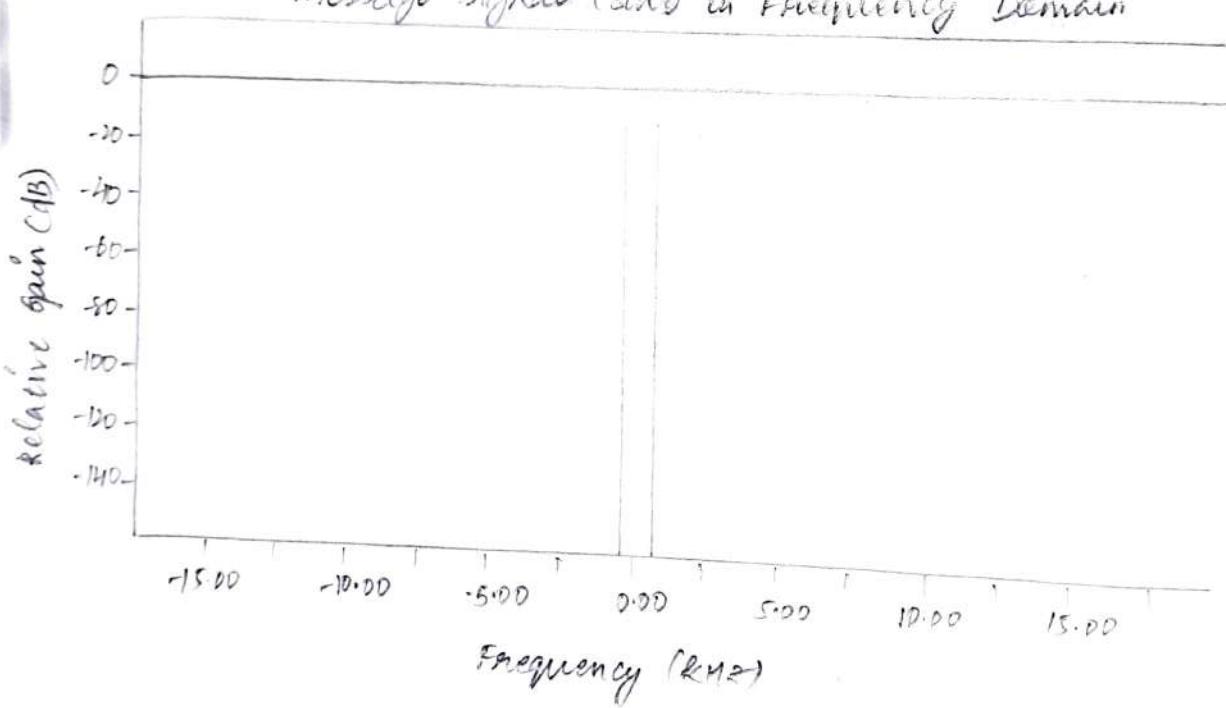
Demodulated Signal in Time domain

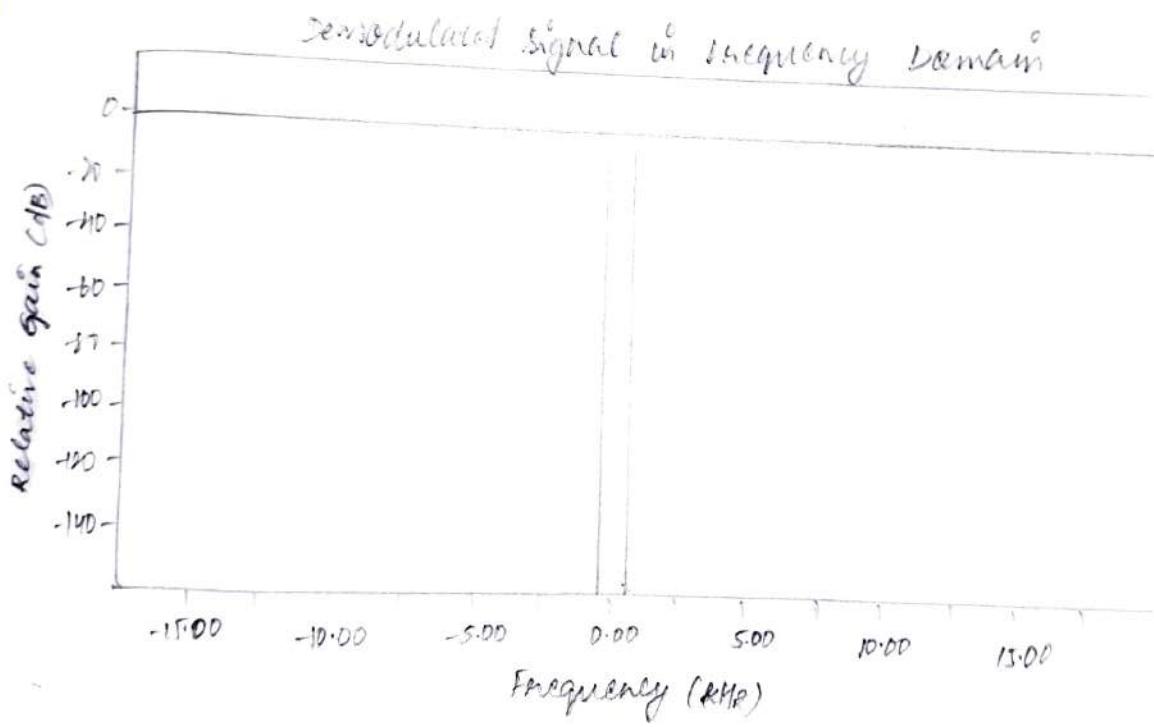
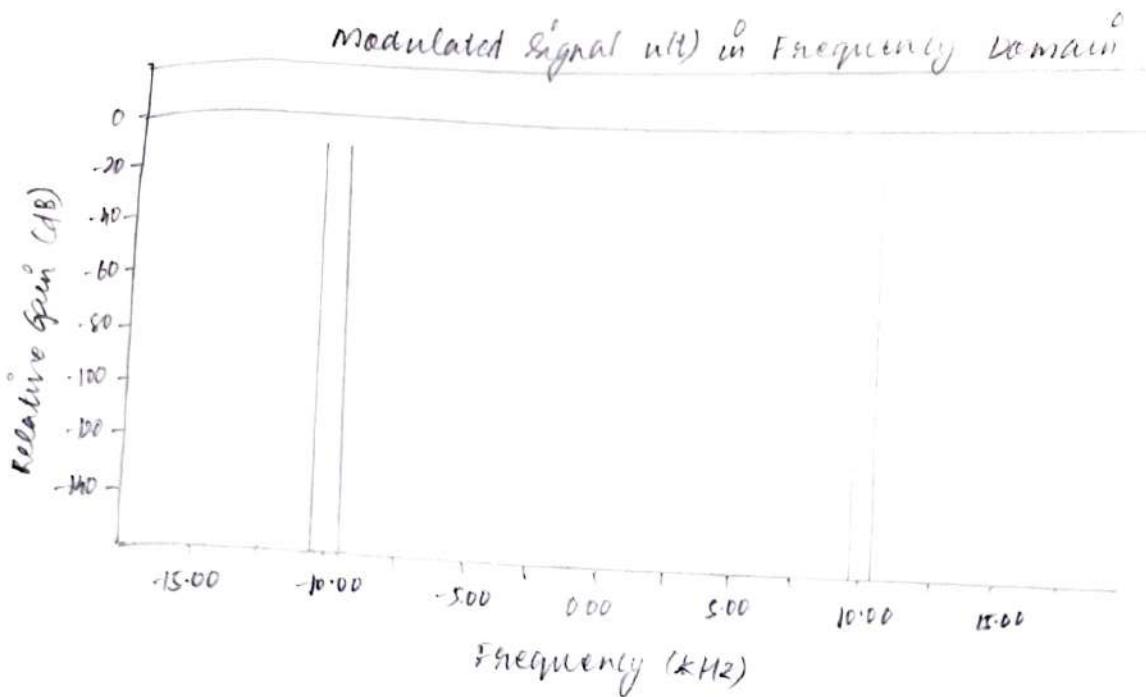


carrier signal (cosine) in Frequency domain



Message Signal (sin) in Frequency domain





#### \* RESULT:

Thus, designed and analyzed a simple communication system using Software Defined Radio (SDR) on GNU Radio companion. All the simulation results were verified successfully.