T3-B
Roll.No:49

MUHAMMED HISHAM V.

SCIENTIFIC COMPUTING LAB

PRELIMINARY LAB REPORT

Lab 2: Familiarization of Scientific Computing.

Q1) Write a short note on different arithmetic functions such as abs, sin(), real(), imag(), complex() and sinc() in python.

i) abs( ) in Python.

The abs() function in python is used to return the absolute value of a number.

Syntax:
abs(number)
number : Can either be an integer, a floating point number or a complex number

ii) sin( ) in Python

In Python, math module contains a number of mathematical operations, which can be performed with ease using the module. math.sin() function returns the sine of value passed as argument. The value passed in this function should be in radians.

Syntax:
math.sin(x)
x : value to be passed to sin()
Returns: Returns the sine of value passed as argument

iii) real( ) in Python

The real part of a complex number can be accessed using the function real().

Syntax:
a.real()
a : The complex number of which real part is to be found.

iv) imag( ) in Python

The imaginary part of a complex number can be accessed using the function imag().

Syntax:

a.imag()
a : The complex number of which imaginary part is to be found.

v) complex( ) in Python
Python converts the real numbers x and y into complex using the function complex(x,y).
It takes two arguments where first one is the real part of the complex number and the second one is the imaginary part of the complex number. It returns the complex number.

Syntax :
z = complex(x,y)
Where
x : real part
y : imaginary part
z : complex number returned

vi) sinc( )
It is a function from NumPy module. Therefore to use it we should import NumPy module. This mathematical function in python helps user to calculate sinc function for all x(being the array elements). It takes an array of numbers and returns their sinc values. The input should be in radians.

Syntax :
arr = numpy.sinc(array)
arr : list of sinc values
array : Input list of numbers in radian.

Q2) Write a short note on vectorized computation in python.

In the context of high-level languages like Python, Matlab, and R, the term vectorization describes the use of optimized, pre-compiled code written in a low-level language (eg. C) to perform mathematical operations over a sequence of data. This is done in place of an explicit iteration written in the native language code (eg. a “for-loop” written in Python).

Vectorization is used to speed up the Python code without using loop. Using such a function can help in minimizing the running time of code efficiently. Various operations are being performed over vector such as dot product of vectors which is also known as scalar product as it produces single output, outer products which results in square matrix of dimension equal to length X length of the vectors, Element wise multiplication which products the element of same indexes and dimension of the matrix remain unchanged.

Time complexity in the execution of any algorithm is very crucial deciding whether an application is reliable or not. To run a large algorithm in as much as optimal time possible is very important when it comes to real-time application of output. To do so, Python has some standard mathematical functions for fast operations on entire arrays of data without having to write loops. One of such library which contains such function is NumPy.

NumPy provides highly-optimized functions for performing mathematical operations on arrays of numbers. Performing extensive iterations (eg.via 'for-loops') in Python to perform repeated mathematical computations should nearly always be replaced by the use of vectorized functions on arrays. This informs the entire design paradigm of NumPy.

Q3) Write the algorithm/ flowchart for the experiments.

Experiment-1

Aim: To compute the factorial of an integer using a function.

Algorithm:

```
1 : Start
2 : Define function factorial(n)
    i.   If n==1 then return 1
    ii.  else
    iii. f = n * factorial(n – 1)
    iv.  Return f
3 : Input a number and store it in 'n'
4 : Print factorial(n)
5 : Stop
```

Experiment-2

Aim: To compute the sum of first 'N' Fibonacci numbers using function.

Algorithm:

```
1 : Start
2 : Define function fsum(n)
    • a =0, b=1
    • if n==1 return 0
    • elif n==2 return 1
    • else repeat steps i – iv for n != 0
        i.   c = a + b
        ii.  sum = sum + c
        iii. a = b
        iv.  b = c
        v.   n = n-1
    • return sum
3: Input an integer and store it in variable 'n'
4: Print fsum(n)
5:Stop
```

Experiment-3

Aim: Algorithm to Represent a complex number in python using built in function, display it's real part, imaginary part and absolute value using built in functions.

Algorithm:

```
1 : Start
2 : Import module 'cmath'
3 : Input 'x' as real part and 'y' as imaginary part.
4 : z = complex(x, y) #representing as a complex number.
5 : Print the real part using z.real()
6 : Print the imaginary part using z.imag()
7 : Stop
```

Experiment-4

Aim: To Plot a sine wave in python with frequency 100 Hz and sampling rate fs = 10000 Hz.

Algorithm:

```
1 : Start
```

```
2 : Import libraries numpy as np and matplotlib.pyplot as plot
3 : Assign sampling frequency fs = 10000 and message frequency fm = 100
4 : Discrete frequency, w = 2 * np.pi * fm/fs
5 : Get x values of the sine wave, time = np.arange(0, 100, 0.1)
5 : Plot a sine wave using time and amplitude obtained for the sine wave.
6 : Give a title,x-axis label and y-axis label for the sine wave plot
7 : Display the sine wave using plot.show().
8 : Stop.
```

Experiment-5

Aim: To plot a sinc function in python.

Algorithm:

```
1 : Start
2 : Import libraries numpy as np and matplotlib.pyplot as plt
3 : Define x axis using linespace() function in numpy as x = np.linespace(-2 * np.pi,
2 * np.pi , 500)
4 : Define y axis using sinc() function as y = np.sinc(x)
5 : Initialize the figure with figure( ) function in matplotlib.pyplot
6 : Plot the figure with plot( ) function
7 : Display the figure with show( ) function.
8 : Stop.
```

Experiment-6

Experiment-6.a.1

Aim : Add given two matrices using **non-vectorized** method.

Algorithm:

```
1 : Start
2 : Initialize x and y with two given matrices.
3 :  i = 0,j = 0,z=[[]]
4 : while i < len(x) repeat steps 4 - 7
5 : while j < len(x[0]) repeat steps 5 - 6
6 : z[i][j] = x[i][j] + y[i][j]
7 : j = j + 1
```

```
8 : i = i + 1
9: Print z
10:Stop
```

Experiment-6.a.2

Aim : Add given two matrices using **vectorized** method.

Algorithm:

```
1 : Start
2 : Import library numpy
3 : Initialize x and y using numpy.array() method
4 : Add x and y using numpy.add() method with x and y as arguments.
5 : Print numpy.add(x, y)
6 : Stop
```

Experiment-6.b.1

Aim : Add '2' to each element in the first row of a matrix using **non-vectorized** method.

Algorithm:

```
1 : Start
2 : Initialize y with the given matrix and i = 0
3 : while i < len(y) repeat steps 4 to 7
4 : while j < len(y[0]) repeat steps 5 to 6
5 : y[i][j] = y[i][j] + 2
6 : j = j +1
7 : i = i + 1
8 : Print y
9 : Stop
```

Experiment-6.b.2

Aim : Add '2' to each element in the first row of a matrix using **vectorized** method.

Algorithm:

```
1 : Start
2 : Import library numpy
```

```
3 : Initialize y using numpy.array() method
4 : y = y + 2
5 : Print y
6 : Stop
```

Experiment-6.c.1

Aim : Find the sum of all elements in a matrix using **non-vectorized** method.

Algorithm:

```
1 : Start
2 : Initialize x  with the given matrix.
3 :  i = 0,j = 0,sum=0
4 : while i < len(x) repeat steps 4 - 7
5 : while j < len(x[0]) repeat steps 5 - 6
6 : sum = sum + x[i][j]
7 : j = j + 1
8 : i = i + 1
9: Print sum
10: Stop
```

Experiment-6.c.2

Aim : Find the sum of all elements in a matrix using **vectorized** method.

Algorithm:

```
1 : Start
2 : Import library numpy
3 : Initialize y using numpy.array() method
4 : sum = numpy.sum(x)
5 : Print sum
6 : Stop
```

Experiment-6.d.1

Aim : Find a 1D array by summing over the rows within each column of a matrix using **non-vectorized** method.

Algorithm:

```
1 : Start
2 : Initialize y with the given matrix.
3 :  i = 0,j = 0
4 : while i < len(x) repeat steps 4 – 10
5 : sum = 0
6 : while j < len(x[0]) repeat steps 6 - 8
7 : sum = sum + y[j][i]
8 : j = j + 1
9 : i = i + 1
10 : Insert sum into a list l
11:Print l
12 : Stop
```

Experiment-6.d.2

Aim : Find a 1D array by summing over the rows within each column of a matrix using **vectorized** method.

Algorithm:

```
1 : Start
2 : Import library numpy
3 : Initialize y using numpy.array() method
4 : n = numpy.sum(y, axis=0)
5 : Print n
6 : Stop
```

Experiment-6.e.1

Aim : Find element by element multiplication of two matrices using **non-vectorized** method.

Algorithm:

```
1 : Start
2 : Initialize x and y with two given matrices.
3 :  i = 0,j = 0,z=[[]]
4 : while i < len(x) repeat steps 4 - 7
5 : while j < len(x[0]) repeat steps 5 - 6
```

```
6 : z[i][j] = x[i][j] * y[i][j]
7 : j = j + 1
8 : i = i + 1
9: Print z
10:Stop
```

Experiment-6.e.2

Aim : Find element by element multiplication of two matrices using **vectorized** method.

Algorithm:

```
1 : Start
2 : Import library numpy
3 : Initialize x and y using numpy.array() method
4 : Multiply element-wise x and y using numpy.multiply() method with x and y as arguments.
5 : Print numpy.multiply(x, y)
6 : Stop
```