

# Lab 1 Report

Amlan Nayak MS18197

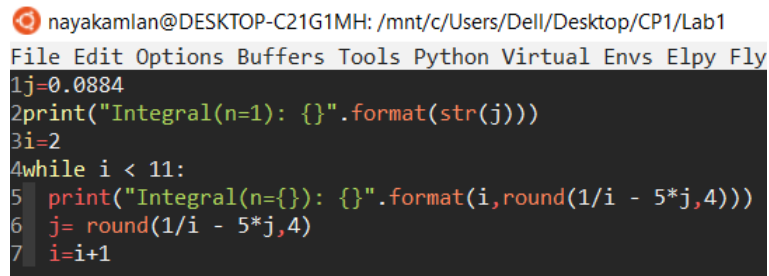
20 Jan 2021

## 1 Write a code to check for induced instability in $I_n = \int_0^1 \frac{x^n}{x+5}$ for $n=1,2,3,\dots,10$

### 1.1 Algorithm and Discussion

A simple loop was used to find the integral values using recurrence relation  $I_n = 1/n - 5I_{n-1}$  for different  $n$ . The values start to oscillate by the end. If the loop is extended beyond  $n=10$ , we find the integral values eventually lead to overflow error.

### 1.2 Code and Output

A screenshot of the Emacs text editor interface. The title bar at the top shows the user 'nayakamlan@DESKTOP-C21G1MH' and the file path '/mnt/c/Users/Dell/Desktop/CP1/Lab1'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Python', 'Virtual', 'Envs', 'Elpy', and 'Fly'. The code is written in a dark-themed window. It starts with a variable 'j' assigned the value 0.0884. Then it prints 'Integral(n=1): {}'.format(str(j)) where 'j' is the variable. Next, 'i' is set to 2. A while loop 'while i < 11:' follows. Inside the loop, it prints 'Integral(n={}): {}'.format(i, round(1/i - 5\*j, 4)) and updates 'j' to 'round(1/i - 5\*j, 4)' and 'i' to 'i+1'.

```
nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/Lab1
File Edit Options Buffers Tools Python Virtual Envs Elpy Fly
1j=0.0884
2print("Integral(n=1): {}".format(str(j)))
3i=2
4while i < 11:
5    print("Integral(n={}): {}".format(i,round(1/i - 5*j,4)))
6    j= round(1/i - 5*j,4)
7    i=i+1
```

Figure 1: Input Code in Python in emacs

```

hayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/Lab1$ emacs
hayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/Lab1$ python3 MS18197_1_code1.py
Integral(n=1): 0.0884
Integral(n=2): 0.058
Integral(n=3): 0.0433
Integral(n=4): 0.0335
Integral(n=5): 0.0325
Integral(n=6): 0.0042
Integral(n=7): 0.1219
Integral(n=8): -0.4845
Integral(n=9): 2.5336
Integral(n=10): -12.568

```

Figure 2: Output

```

Integral(n=11): 62.9309
Integral(n=12): -314.5712
Integral(n=13): 1572.9329
Integral(n=14): -7864.5931
Integral(n=15): 39323.0322
Integral(n=16): -196615.0985
Integral(n=17): 983075.5513
Integral(n=18): -4915377.7009
Integral(n=19): 24576888.5571
Integral(n=20): -122884442.7355
Integral(n=21): 614422213.7251
Integral(n=22): -3072111068.58
Integral(n=23): 15360555342.9435
Integral(n=24): -76802776714.6758
Integral(n=25): 384013883573.4189
Integral(n=26): -1920069417867.056
Integral(n=27): 9600347089335.316
Integral(n=28): -48001735446676.54
Integral(n=29): 240008677233382.72

```

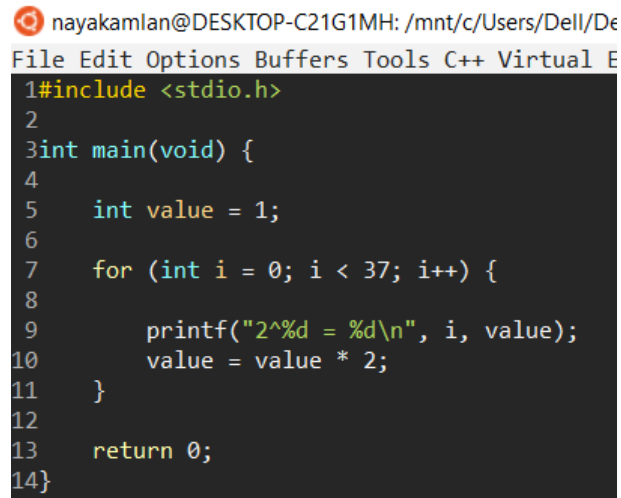
Figure 3: Output beyond n=10

**2 Define a  $=2$  as int and long int, make a loop and multiply it by 2. Run the loop for 36 times and discuss.**

### 2.1 Algorithm and Discussion

A simple C++ code was used to execute the loop. The values from the long int and int outputs clearly show the overflow in the int output at the 31st power of 2.

### 2.2 Code and Output

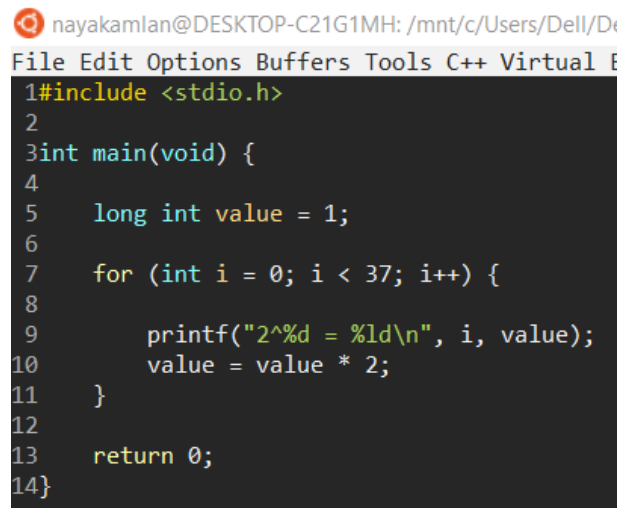


```

nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/De
File Edit Options Buffers Tools C++ Virtual E
1#include <stdio.h>
2
3int main(void) {
4
5    int value = 1;
6
7    for (int i = 0; i < 37; i++) {
8
9        printf("2^%d = %d\n", i, value);
10       value = value * 2;
11    }
12
13    return 0;
14}

```

Figure 4: Int Code in C++ in emacs



```

nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/De
File Edit Options Buffers Tools C++ Virtual E
1#include <stdio.h>
2
3int main(void) {
4
5    long int value = 1;
6
7    for (int i = 0; i < 37; i++) {
8
9        printf("2^%d = %ld\n", i, value);
10       value = value * 2;
11    }
12
13    return 0;
14}

```

Figure 5: Long Int Code in C++ in emacs

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab1$ g++ MS18197_0_code2_int.cpp -o code2int
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab1$ ./code2int
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
2^8 = 256
2^9 = 512
2^10 = 1024
2^11 = 2048
2^12 = 4096
2^13 = 8192
2^14 = 16384
2^15 = 32768
2^16 = 65536
2^17 = 131072
2^18 = 262144
2^19 = 524288
2^20 = 1048576
2^21 = 2097152
2^22 = 4194304
2^23 = 8388608
2^24 = 16777216
2^25 = 33554432
2^26 = 67108864
2^27 = 134217728
2^28 = 268435456
2^29 = 536870912
2^30 = 1073741824
2^31 = -2147483648
2^32 = 0
2^33 = 0
2^34 = 0
2^35 = 0
2^36 = 0
```

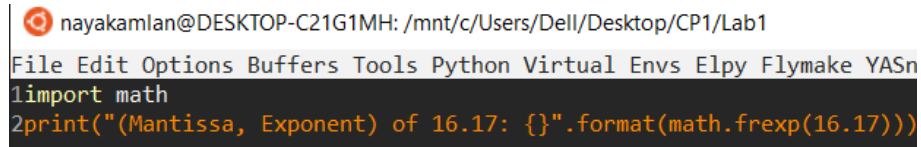
Figure 6: Output for Int

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab1$ g++ MS18197_1_code2_long.cpp -o code2long
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab1$ ./code2long
./code2long: command not found
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab1$ ./code2long
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
2^8 = 256
2^9 = 512
2^10 = 1024
2^11 = 2048
2^12 = 4096
2^13 = 8192
2^14 = 16384
2^15 = 32768
2^16 = 65536
2^17 = 131072
2^18 = 262144
2^19 = 524288
2^20 = 1048576
2^21 = 2097152
2^22 = 4194304
2^23 = 8388608
2^24 = 16777216
2^25 = 33554432
2^26 = 67108864
2^27 = 134217728
2^28 = 268435456
2^29 = 536870912
2^30 = 1073741824
2^31 = 2147483648
2^32 = 4294967296
2^33 = 8589934592
2^34 = 17179869184
2^35 = 34359738368
2^36 = 68719476736
```

Figure 7: Output for Long Int

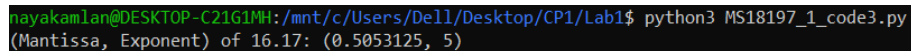
### 3 Decompose 16.17 into its Mantissa and exponent. Use frexp function

#### 3.1 Code and Output



```
nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/Lab1
File Edit Options Buffers Tools Python Virtual Envs Elpy Flymake YASn
1import math
2print("(Mantissa, Exponent) of 16.17: {}".format(math.frexp(16.17)))
```

Figure 8: Code in Python in emacs



```
nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/Lab1$ python3 MS18197_1_code3.py
(Mantissa, Exponent) of 16.17: (0.5053125, 5)
```

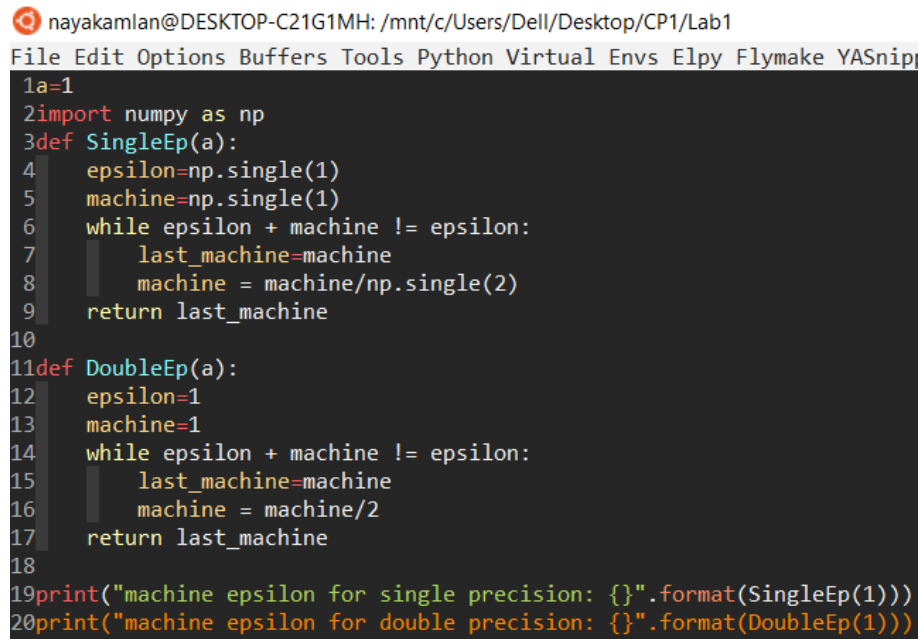
Figure 9: Output of frexp function on 16.17

### 4 Get the machine epsilon for single precision and double precision on your system using C++ or python

#### 4.1 Algorithm

A loop was used to add 1 to a second number whose value is 1 initially. Then I continued to divide the second number by 2 till the loop condition failed. The loop breaks when the python not equal(!=) boolean logic can no longer distinguish between 1 and (1 + second number). That gives us the machine epsilon.

#### 4.2 Code and Output

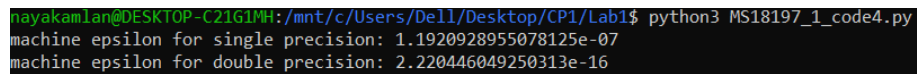


```

nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/Lab1
File Edit Options Buffers Tools Python Virtual Envs Elpy Flymake YASnipe
1a=1
2import numpy as np
3def SingleEp(a):
4    epsilon=np.single(1)
5    machine=np.single(1)
6    while epsilon + machine != epsilon:
7        last_machine=machine
8        machine = machine/np.single(2)
9    return last_machine
10
11def DoubleEp(a):
12    epsilon=1
13    machine=1
14    while epsilon + machine != epsilon:
15        last_machine=machine
16        machine = machine/2
17    return last_machine
18
19print("machine epsilon for single precision: {}".format(SingleEp(1)))
20print("machine epsilon for double precision: {}".format(DoubleEp(1)))

```

Figure 10: Code in Python in emacs



```

nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab1$ python3 MS18197_1_code4.py
machine epsilon for single precision: 1.1920928955078125e-07
machine epsilon for double precision: 2.220446049250313e-16

```

Figure 11: Output

## 5 Fill an array with random integers and now make algorithm to assort them in ascending order.

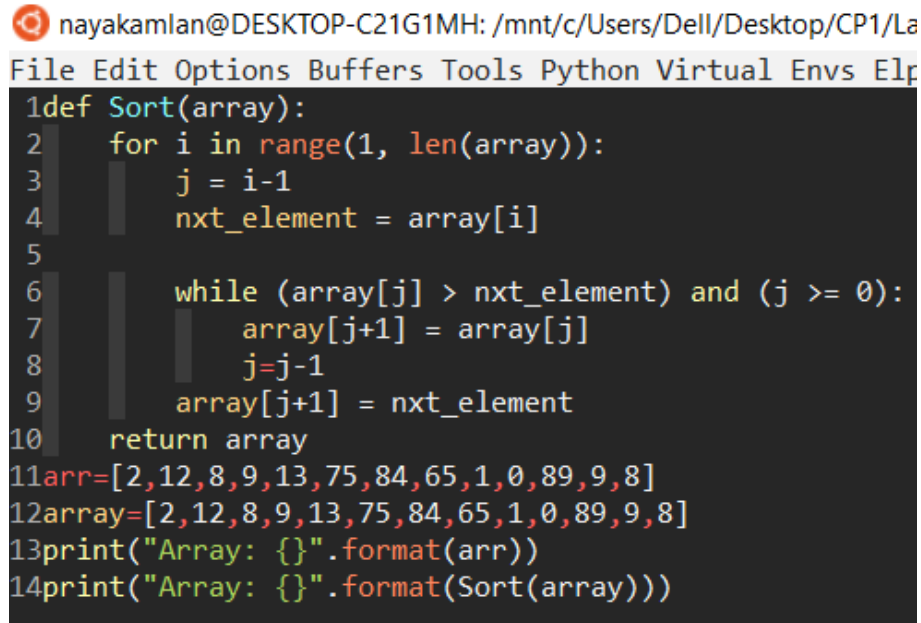
### 5.1 Algorithm

An insertion sort algorithm was used. It comprises of a simple code for small lists, but is not appropriate for bigger lists.

A for loop is used for iterating from 1 to length of the list. In the loop, a value is assigned to an intermediary variable from the list. Inside the for loop, a while is used. All the elements in the list greater than the intermediary variable are moved to the next position from their current position. Then the intermediary variable is compared with the first element of the list. If

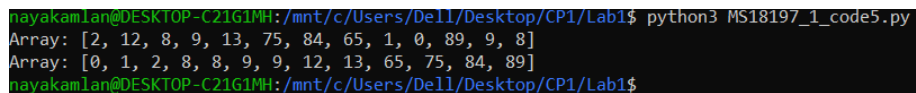
conditions are appropriate, the while loop continues to operate. The same thing happens for each intermediary value from the list through the for loop.

## 5.2 Code and Output



```
nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/La
File Edit Options Buffers Tools Python Virtual Envs Elp
1def Sort(array):
2    for i in range(1, len(array)):
3        j = i-1
4        nxt_element = array[i]
5
6        while (array[j] > nxt_element) and (j >= 0):
7            array[j+1] = array[j]
8            j=j-1
9            array[j+1] = nxt_element
10    return array
11arr=[2,12,8,9,13,75,84,65,1,0,89,9,8]
12array=[2,12,8,9,13,75,84,65,1,0,89,9,8]
13print("Array: {}".format(arr))
14print("Array: {}".format(Sort(array)))
```

Figure 12: Code in Python in emacs



```
nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/Lab1$ python3 MS18197_1_code5.py
Array: [2, 12, 8, 9, 13, 75, 84, 65, 1, 0, 89, 9, 8]
Array: [0, 1, 2, 8, 8, 9, 9, 12, 13, 65, 75, 84, 89]
nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/Lab1$
```

Figure 13: Output

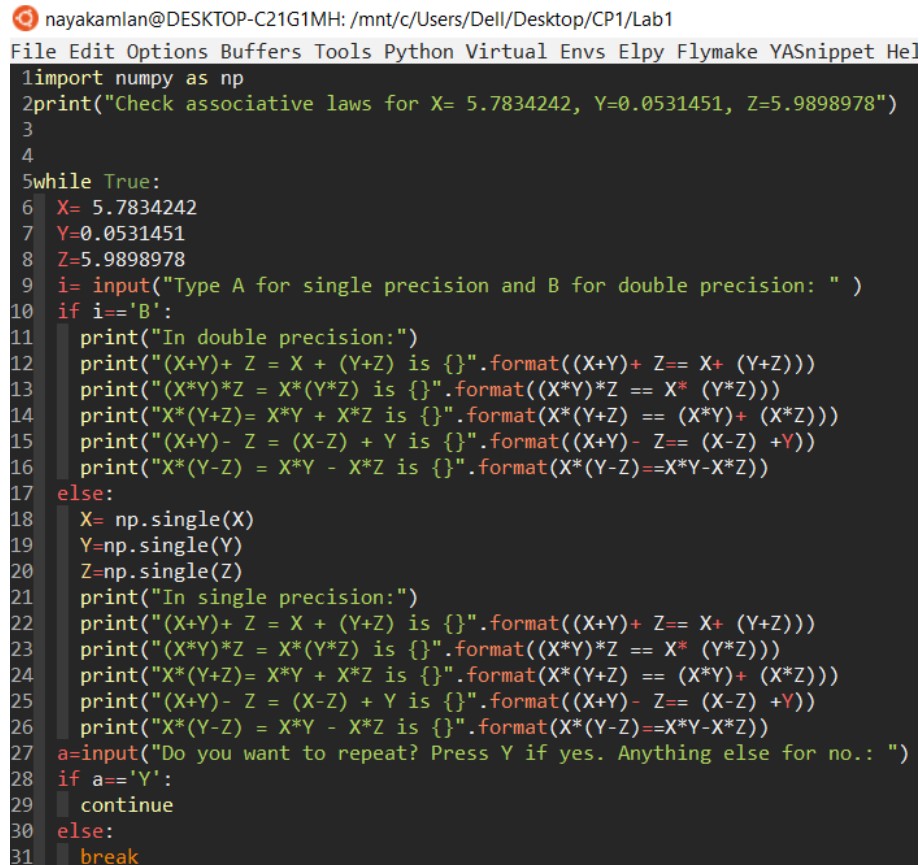
## 6 Check for associate law with $X= 5.7834242$ , $Y=0.0531451$ , $Z=5.9898978$

### 6.1 Algorithm

The default float in python is double precision. Hence, Numpy's single and double precision was used to check for the laws.



## 6.2 Code and Output



```
nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/Lab1
File Edit Options Buffers Tools Python Virtual Envs Elpy Flymake YASnippet Hel
1import numpy as np
2print("Check associative laws for X= 5.7834242, Y=0.0531451, Z=5.9898978")
3
4
5while True:
6    X= 5.7834242
7    Y=0.0531451
8    Z=5.9898978
9    i= input("Type A for single precision and B for double precision: " )
10    if i=='B':
11        print("In double precision:")
12        print("(X+Y)+ Z = X + (Y+Z) is {}".format((X+Y)+ Z== X+ (Y+Z)))
13        print("(X*Y)*Z = X*(Y*Z) is {}".format((X*Y)*Z == X* (Y*Z)))
14        print("X*(Y+Z)= X*Y + X*Z is {}".format(X*(Y+Z) == (X*Y)+ (X*Z)))
15        print("(X+Y)- Z = (X-Z) + Y is {}".format((X+Y)- Z== (X-Z) +Y))
16        print("X*(Y-Z) = X*Y - X*Z is {}".format(X*(Y-Z)==X*Y-X*Z))
17    else:
18        X= np.single(X)
19        Y=np.single(Y)
20        Z=np.single(Z)
21        print("In single precision:")
22        print("(X+Y)+ Z = X + (Y+Z) is {}".format((X+Y)+ Z== X+ (Y+Z)))
23        print("(X*Y)*Z = X*(Y*Z) is {}".format((X*Y)*Z == X* (Y*Z)))
24        print("X*(Y+Z)= X*Y + X*Z is {}".format(X*(Y+Z) == (X*Y)+ (X*Z)))
25        print("(X+Y)- Z = (X-Z) + Y is {}".format((X+Y)- Z== (X-Z) +Y))
26        print("X*(Y-Z) = X*Y - X*Z is {}".format(X*(Y-Z)==X*Y-X*Z))
27    a=input("Do you want to repeat? Press Y if yes. Anything else for no.: ")
28    if a=='Y':
29        continue
30    else:
31        break
```

Figure 14: Code in Python in emacs

```

nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab1$ python3 MS18197_1_code6.py
Check associative laws for X= 5.7834242, Y=0.0531451, Z=5.9898978
Type A for single precision and B for double precision: A
In single precision:
(X+Y)+ Z = X + (Y+Z) is True
(X*Y)*Z = X*(Y*Z) is False
X*(Y+Z)= X*Y + X*Z is True
(X+Y)- Z = (X-Z) + Y is False
X*(Y-Z) = X*Y - X*Z is False
Do you want to repeat? Press Y if yes. Anything else for no.: Y
Type A for single precision and B for double precision: B
In double precision:
(X+Y)+ Z = X + (Y+Z) is True
(X*Y)*Z = X*(Y*Z) is False
X*(Y+Z)= X*Y + X*Z is True
(X+Y)- Z = (X-Z) + Y is False
X*(Y-Z) = X*Y - X*Z is True
Do you want to repeat? Press Y if yes. Anything else for no.: n
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab1$

```

Figure 15: Output

- 7 Compare the Taylor series of  $f(x) = \sin x$  at  $x = \pi/3$  with base point at  $\pi/4$  with original function by keep adding the next term till fourth order.

### 7.1 Algorithm

The taylor expansion of  $\sin x$  at  $x = \pi/3$  with base point at  $\pi/4$  till 4th term was found out and compared with the actual value of function at  $\pi/3$ . The error percentage was found out.

```

nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/Desktop/CP1/Lab1
File Edit Options Buffers Tools Python Virtual Envs Elpy Flymake YASnippet Help
1import numpy as np
2x= np.pi/3
3a=np.pi/4
4taylor= np.sin(a)+np.cos(a)*(x-a)- (np.sin(a)*((x-a)**2))/2 - (np.cos(a)*((x-a)**3))/6
5print("Value of sin(x) at x=pi/3 is {}".format(np.sin(x)))
6print("Taylor series expansion of sin(x) about pi/4 is {}".format(taylor))
7print("Difference between both is {} %".format(abs((-taylor + np.sin(np.pi/3))/np.sin(np.pi/3))*100))

```

Figure 16: Code in emacs

```

nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab1$ python3 MS18197_1_code7.py
Value of sin(x) at x=pi/3 is 0.8660254037844386
Taylor series expansion of sin(x) about pi/4 is 0.8658800807233364
Difference between both is 0.016780461689357797 %

```

Figure 17: Output