Lab 1 Report

Amlan Nayak MS18197

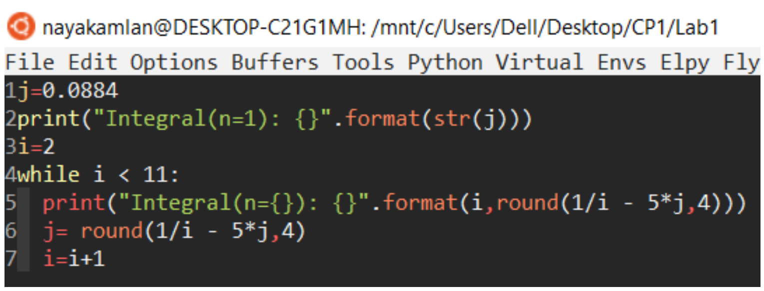
20 Jan 2021

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,...,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




```
j=0.0884
print("Integral(n=1): {}".format(str(j)))
i=2
while i < 11:
  print("Integral(n={}): {}".format(i,round(1/i - 5*j,4)))
  j= round(1/i - 5*j,4)
  i=i+1
```

Figure 1: Intput Code in Python in emacs

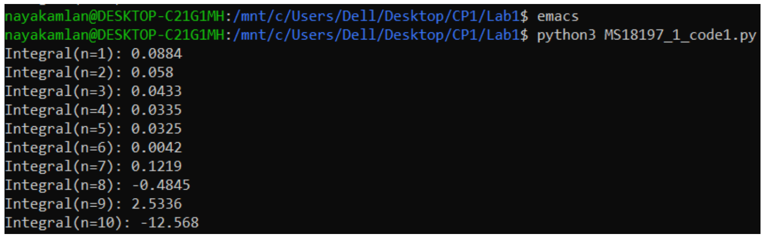


Figure 2: Output

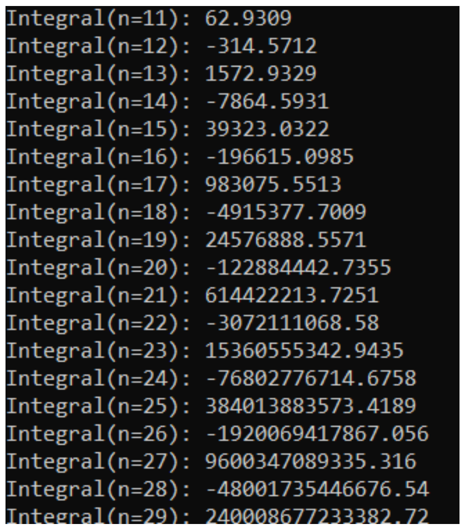


Figure 3: Output beyond n=10

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

```
#include <stdio.h>

int main(void) {

    int value = 1;

    for (int i = 0; i < 37; i++) {

        printf("2^%d = %d\n", i, value);
        value = value * 2;
    }

    return 0;
}
```

Figure 4: Int Code in C++ in emacs

nayakamlan@DESKTOP-C21G1MH: /mnt/c/Users/Dell/De

File Edit Options Buffers Tools C++ Virtual E

```
#include <stdio.h>

int main(void) {

    long int value = 1;

    for (int i = 0; i < 37; i++) {

        printf("2^%d = %ld\n", i, value);
        value = value * 2;
    }

    return 0;
}
```

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

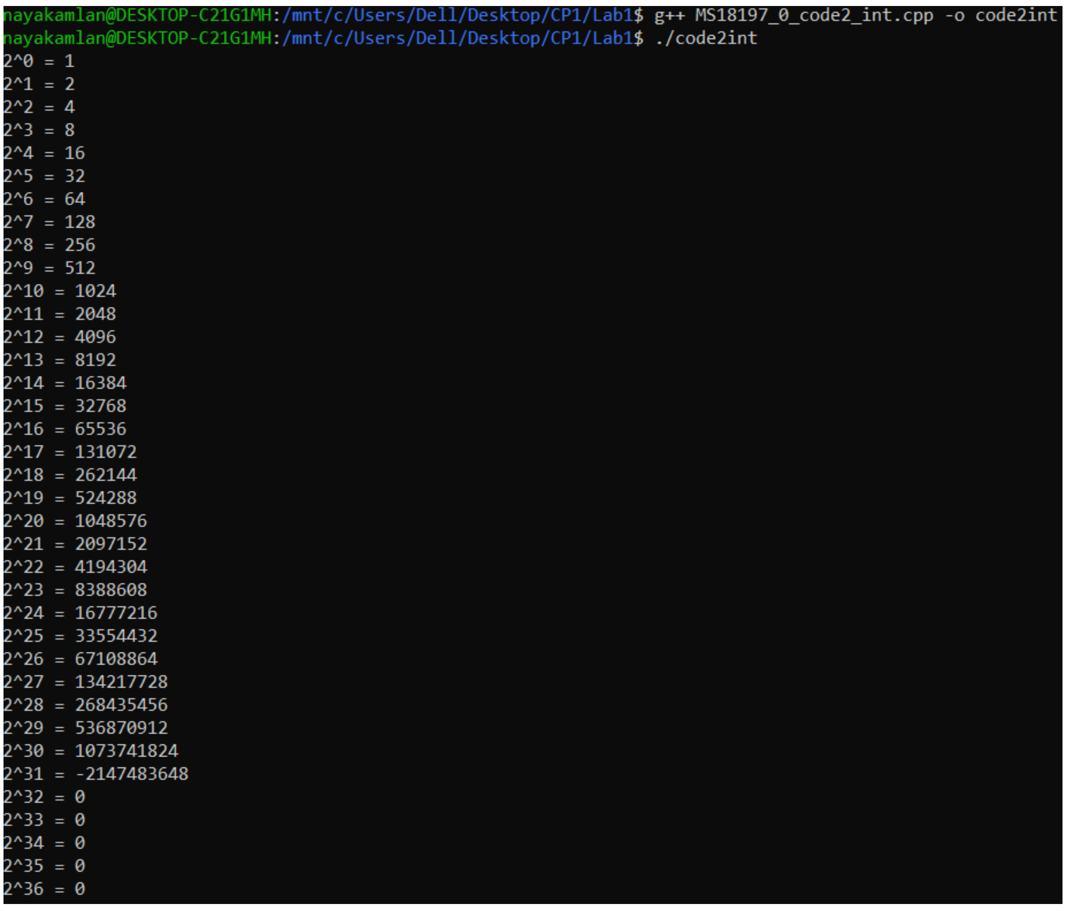
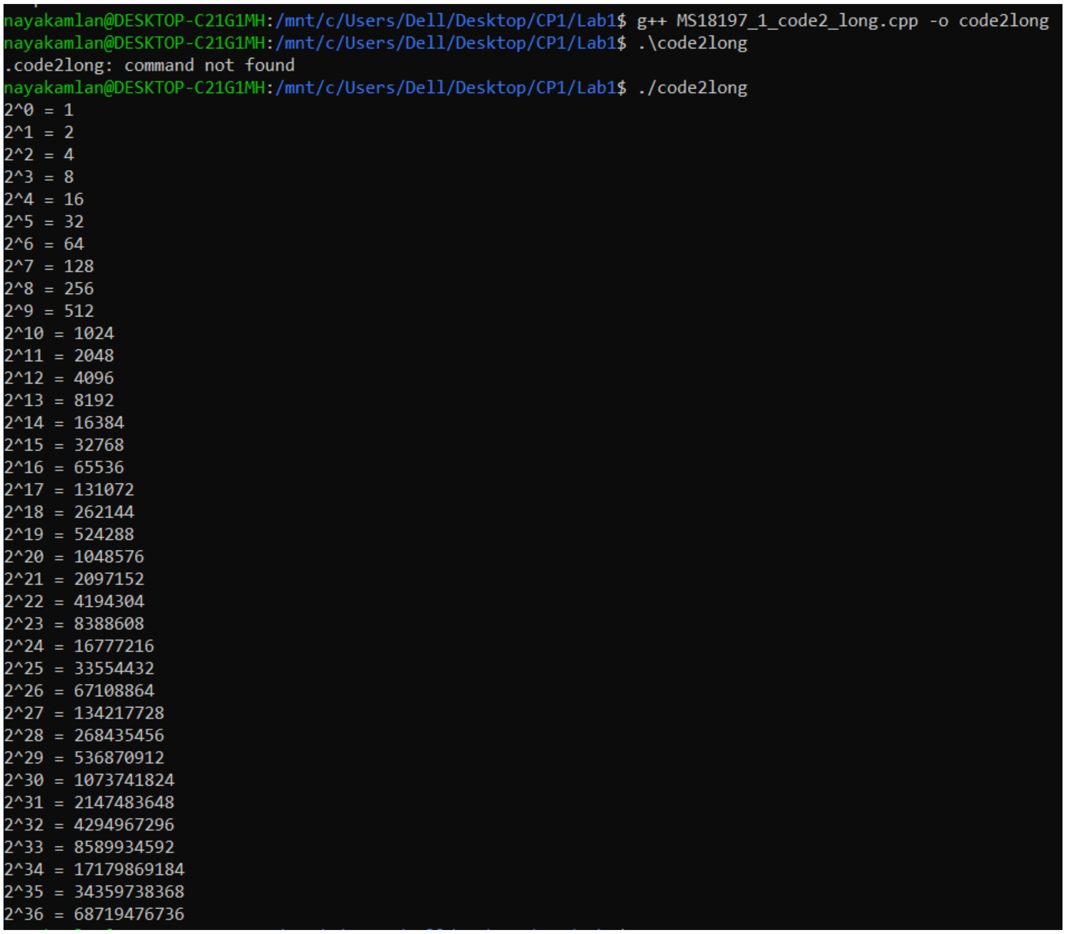


Figure 6: Output for Int



```
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 YASnip
a=1
import numpy as np
def SingleEp(a):
    epsilon=np.single(1)
    machine=np.single(1)
    while epsilon + machine != epsilon:
        last_machine=machine
        machine = machine/np.single(2)
    return last_machine

def DoubleEp(a):
    epsilon=1
    machine=1
    while epsilon + machine != epsilon:
        last_machine=machine
        machine = machine/2
    return last_machine

print("machine epsilon for single precision: {}".format(SingleEp(1)))
print("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
def Sort(array):
    for i in range(1, len(array)):
        j = i-1
        nxt_element = array[i]

        while (array[j] > nxt_element) and (j >= 0):
            array[j+1] = array[j]
            j=j-1
        array[j+1] = nxt_element
    return array
arr=[2,12,8,9,13,75,84,65,1,0,89,9,8]
array=[2,12,8,9,13,75,84,65,1,0,89,9,8]
print("Array: {}".format(arr))
print("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

```
import numpy as np
print("Check associative laws for X= 5.7834242, Y=0.0531451, Z=5.9898978")


while True:
  X= 5.7834242
  Y=0.0531451
  Z=5.9898978
  i= input("Type A for single precision and B for double precision: " )
  if i=='B':
    print("In double precision:")
    print("(X+Y)+ Z = X + (Y+Z) is {}".format((X+Y)+ Z== X+ (Y+Z)))
    print("(X*Y)*Z = X*(Y*Z) is {}".format((X*Y)*Z == X* (Y*Z)))
    print("X*(Y+Z)= X*Y + X*Z is {}".format(X*(Y+Z) == (X*Y)+ (X*Z)))
    print("(X+Y)- Z = (X-Z) + Y is {}".format((X+Y)- Z== (X-Z) +Y))
    print("X*(Y-Z) = X*Y - X*Z is {}".format(X*(Y-Z)==X*Y-X*Z))
  else:
    X= np.single(X)
    Y=np.single(Y)
    Z=np.single(Z)
    print("In single precision:")
    print("(X+Y)+ Z = X + (Y+Z) is {}".format((X+Y)+ Z== X+ (Y+Z)))
    print("(X*Y)*Z = X*(Y*Z) is {}".format((X*Y)*Z == X* (Y*Z)))
    print("X*(Y+Z)= X*Y + X*Z is {}".format(X*(Y+Z) == (X*Y)+ (X*Z)))
    print("(X+Y)- Z = (X-Z) + Y is {}".format((X+Y)- Z== (X-Z) +Y))
    print("X*(Y-Z) = X*Y - X*Z is {}".format(X*(Y-Z)==X*Y-X*Z))
  a=input("Do you want to repeat? Press Y if yes. Anything else for no.: ")
  if a=='Y':
    continue
  else:
    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
import numpy as np
x= np.pi/3
a=np.pi/4
taylor= np.sin(a)+np.cos(a)*(x-a)- (np.sin(a)*((x-a)**2))/2 - (np.cos(a)*((x-a)**3))/6
print("Value of sin(x) at x=pi/3 is {}".format(np.sin(x)))
print("Taylor series expansion of sin(x) about pi/4 is {}".format(taylor))
print("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