Lab 9 Report

Amlan Nayak MS18197

1st April 2021

1 The following data was collected for the distance travelled versus time for a rocket. Use numerical differentiation to estimate the rocket's velocity and acceleration at each time

$T, secs$	0	25	50	75	100	125
X, kms	0	32	58	78	92	100

1.1 Algorithm & Discussion

We use Newton's divided difference polynomial interpolation to find the polynomial. The polynomial is given

$$f(x) = f(x_0) + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + ...+$$
$$(x - x_0)(x - x_1).....(x - x_n)f[x_0, x_1, ..., x_n]$$

We differentiate this twice to obtain $f'(x)$ and $f''(x)$.

$$f'(x) = f[x_0, x_1] + [x - x_0 + x - x_1]f[x_0, x_1, x_2] + [(x - x_0)(x - x_1)+$$
$$(x - x_1)(x - x_2) + (x - x_0)(x - x_2)]f[x_0, x_1, x_2, x_3] + ...$$

$$f''(x) = 2f[x_0, x_1, x_2] + 2[x - x_0 + x - x_1 + x - x_2]f[x_0, x_1, x_2, x_3] + ...$$

Here x is time in secs and $f(x)$ is distance travelled till a particular time x. $f'(x)$ gives us the velocity while $f''(x)$ gives us the acceleration.

1.2 Code & Output

```
Time in secs= [0, 25, 50, 75, 100, 125]
Distance in kms= [0, 32, 58, 78, 92, 100]

Divided Difference table:
Difference number 1 : [1.28, 1.04, 0.8, 0.56, 0.32]
Difference number 2 : [-0.0048, -0.0048, -0.0048, -0.0048]
Difference number 3 : [0.0, 0.0, 0.0]
Difference number 4 : [0.0, 0.0]
Difference number 5 : [0.0]

Value of Velocity at:
t=0secs: 1.4
t=25secs: 1.16
t=50secs: 0.92
t=75secs: 0.68
t=100secs: 0.44
t=125secs: 0.2

Value of Velocity at:
t=0secs: -0.0096
t=25secs: -0.0096
t=50secs: -0.0096
t=75secs: -0.0096
t=100secs: -0.0096
t=125secs: -0.0096
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab9$
```

Figure 1: Output of velocity and acceleration

```
import numpy as np
X = [0,25,50,75,100,125]
F =[0,32,58,78,92,100]
Y=F
Divide =[]
print("Time in secs=",X)
print("Distance in kms=",F)
print("\nDivided Difference table:")
for k in range(len(X)-1):
  Y = [np.round((Y[i+1]-Y[i])/(X[i+k+1]-X[i]),decimals=6) for i in range(len(Y)-1)]
  print("Difference number "+str(k+1)+" :", Y)
  Divide.append(Y[0])
def der(x):
  return Divide[0] +  Divide[1]*(2*x - X[0] - X[1])
print("\nValue of Velocity at:")
for k in range(len(X)):
  print("t="+str(X[k])+"secs:", np.round(der(X[k]),decimals=6))
print("\nValue of Velocity at:")
for k in range(len(X)):
  print("t="+str(X[k])+"secs:", 2*Divide[1])
```

Figure 2: Code for finding velocity and acceleration using numerical differentiation

2 Use Trapezoidal , Simpson's (1/3, 3/8) rule, Boole's and Weddle's rule for finding

$$\int_{-2}^{2} x^3 e^x \, dx, n = 12$$

$$\int_{3}^{5} \frac{1}{\sqrt{x^2 - 4}} \, dx, n = 24$$

2.1 Algorithm & Discussion

As discussed in class, we obtain the methods given in the question from the Newton- Cotes quadrature formula.

For Trapezoidal relation, we put n=1 in quadrature formula. Here x_0 refers to the points where we evaluated the functions after dividing the intervals while y_0 refers to the functional value at x_0. The constant gap between successive points is h. We get:

$$\int_{x_0}^{x_0+nh} f(x) \, dx = \frac{h}{2}[y_0 + y_n + 2(y_1 + y_2 + ... + y_{n-1})]$$

For Simpson's one third relation, we put n=2 in quadrature formula and take the curve through a parabola. We get:

$$\int_{x_0}^{x_0+nh} f(x)\,dx = \frac{h}{3}[y_0 + y_n + 4(y_1 + y_3 + ... + y_{n-1}) + 2(y_2 + y_4 + ... + y_{n-2})]$$

For Simpson's three eight relation, we put n=3 in quadrature formula and take the curve through a 3rd order polynomial. We get:

$$\int_{x_0}^{x_0+nh} f(x)\,dx = \frac{3h}{8}[y_0 + y_n + 3(y_1 + y_2 + y_4 + y_5 + y_7.... + y_{n-1}) + 2(y_3 + y_6 + y_9.... + y_{n-3})]$$

For Boole's relation, we put n=4 in quadrature formula. We get:

$$\int_{x_0}^{x_0+nh} f(x)\,dx = \frac{2h}{45}[7(y_0 + y_n) + 32(y_1 + y_3 + y_5 + ... + y_{n-1}) + 12(y_2 + y_6 + y_{10} + ... + y_{n-2}) +$$
$$14(y_4 + y_8 + ... + y_{n-4})]$$

For Weddle's relation, we put n=6 in quadrature formula. We get:

$$\int_{x_0}^{x_0+nh} f(x)\,dx = \frac{3h}{10}[(y_0 + y_n) + 5(y_1 + y_7 + y_{13} + ... + y_{n-5}) + (y_2 + y_8 + y_{14}... + y_{n-4}) +$$
$$6(y_3 + y_9 + ... + y_{n-3}) + (y_4 + y_{10} + ... + y_{n-2}) + 5(y_5 + y_1 1 + y_{17} + ... + y_{n-1})]$$

2.2 Code & Output

```
import numpy as np
print("f1(x) = exp(x)*x^3")
print("f2(x) = 1/sqrt(x^2 - 4)")
def f1(x):
  return np.power(x,3)*np.exp(x)
def f2(x):
  return 1/np.sqrt(np.power(x,2) - 4)
print("\nUsing Trapezoidal rule:")

X = np.linspace(-2,2,num=12)
h = X[1] - X[0]
values = [0.5*h*(f1(X[i])+ f1(X[i+1])) for i in range(len(X)-1)]
print(" Integral of f1 from (-2,2):", np.round(np.sum(values),decimals=5))
X = np.linspace(3,5,num=24)
h = X[1] - X[0]
values = [0.5*h*(f2(X[i])+ f2(X[i+1])) for i in range(len(X)-1)]
print(" Integral of f2 from (3,5):",np.round(np.sum(values),decimals=5))

print("\nUsing Simpson 1/3 rule:")
X = np.linspace(-2,2,num=12)
h = (X[1]-X[0])/2
Y = [f1(i) for i in X]
Y_half= [f1((X[i]+X[i+1])/2) for i in range(len(X)-1)]
value = (h /3) * (Y[0]+Y[-1] + 2*np.sum(Y[1:-1]) + 4*np.sum(Y_half) )
print(" Integral of f1 from (-2,2):", np.round(value,decimals=5))
X = np.linspace(3,5,num=24)
h = (X[1] - X[0])/2
Y = [f2(i) for i in X]
Y_half= [f2((X[i]+X[i+1])/2) for i in range(len(X)-1)]
value = (h /3) * (Y[0]+Y[-1] + 2*np.sum(Y[1:-1]) + 4*np.sum(Y_half) )
print(" Integral of f2 from (3,5):", np.round(value,decimals=5))

```

Figure 3: Code in emacs part1

```
import numpy as np
print("f1(x) = exp(x)*x^3")
print("f2(x) = 1/sqrt(x^2 - 4)")
def f1(x):
  return np.power(x,3)*np.exp(x)
def f2(x):
  return 1/np.sqrt(np.power(x,2) - 4)
print("\nUsing Trapezoidal rule:")

X = np.linspace(-2,2,num=12)
h = X[1] - X[0]
values = [0.5*h*(f1(X[i])+ f1(X[i+1])) for i in range(len(X)-1)]
print(" Integral of f1 from (-2,2):", np.round(np.sum(values),decimals=5))
X = np.linspace(3,5,num=24)
h = X[1] - X[0]
values = [0.5*h*(f2(X[i])+ f2(X[i+1])) for i in range(len(X)-1)]
print(" Integral of f2 from (3,5):",np.round(np.sum(values),decimals=5))

print("\nUsing Simpson 1/3 rule:")
X = np.linspace(-2,2,num=12)
h = (X[1]-X[0])/2
Y = [f1(i) for i in X]
Y_half= [f1((X[i]+X[i+1])/2) for i in range(len(X)-1)]
value = (h /3) * (Y[0]+Y[-1] + 2*np.sum(Y[1:-1]) + 4*np.sum(Y_half) )
print(" Integral of f1 from (-2,2):", np.round(value,decimals=5))
X = np.linspace(3,5,num=24)
h = (X[1] - X[0])/2
Y = [f2(i) for i in X]
Y_half= [f2((X[i]+X[i+1])/2) for i in range(len(X)-1)]
value = (h /3) * (Y[0]+Y[-1] + 2*np.sum(Y[1:-1]) + 4*np.sum(Y_half) )
print(" Integral of f2 from (3,5):", np.round(value,decimals=5))

```

Figure 4: Code in emacs part2

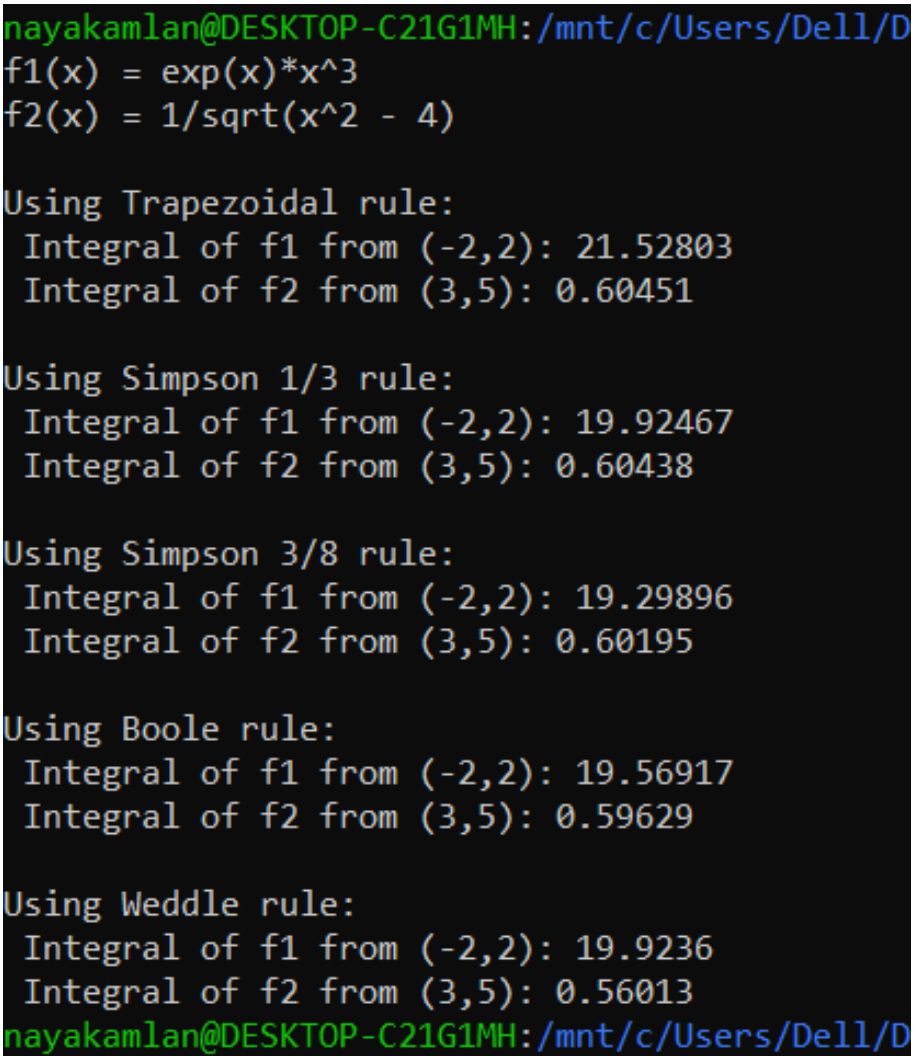

Figure 5: Output

3 A car laps a race track in 84 seconds. The speed of the car at each 6-second interval is determined by using a radar gun and is given from the beginning of the lap in km/second, by The entries in the following table. How long is the track ?

Time in Secs	0	6	12	18	24	30	36	42	48	54	60	66	72	78	84
Speed in Kms/s	124	134	148	156	147	133	121	109	99	85	78	89	104	116	123

3.1 Algorithm & Discussion

We use the simple trapezoidal method to find the area under the speed time curve, which gives us the length of the track.

We do the same with Weddle's rule.

3.2 Code and Output

```
1import numpy as np
2X = np.linspace(0,84,num=15)
3Y = [124,134,148,156,147,133,121,109,99,85,78,89,104,116,123]
4
5print("Time in secs:",X)
6print("Speed in  km/secs:",Y)
7h = X[1] - X[0]
8
9print("\nUsing Trapezoidal rule:")
10value = 0.5*h*(Y[0]+Y[-1] + 2*np.sum(Y[1:-1:1]))
11print(" Length of track in kms:", np.sum(value))
12
13print("\nUsing Simpson 1/3 rule:")
14value = (h/3)*(Y[0]+Y[-1] + 4*np.sum(Y[1:-1:2])+2*np.sum(Y[2:-2:2]))
15print(" Length of track in kms:", np.sum(value))
16
17print("\nUsing Simpson 3/8 rule:")
18value = (3*h/8)*(Y[0]+Y[-1]  + 2*np.sum(Y[3:-1:3]))
19for i in range(3,len(Y),3):
20  Y[i] = 0
21value = value +(3*h/8)*3*np.sum(Y[1:-1])
22print(" Length of track in kms:", np.sum(value))
```

Figure 6: Code in emacs

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab9$ pyth
on3 MS18197_9_code3.py
Time in secs: [ 0.  6. 12. 18. 24. 30. 36. 42. 48. 54. 60. 66. 72.
78. 84.]
Speed in  km/secs: [124, 134, 148, 156, 147, 133, 121, 109, 99, 85,
 78, 89, 104, 116, 123]

Using Trapezoidal rule:
 Length of track in kms: 9855.0

Using Simpson 1/3 rule:
 Length of track in kms: 9858.0

Using Simpson 3/8 rule:
 Length of track in kms: 9760.5
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab9$
```

Figure 7: Code in emacs

4 Use Romberg integration to compute $\int_0^{48} \sqrt{1 + cos^2(x)}\, dx$

4.1 Algorithm & Discussion

Here, we work very similar to the table structuring of the divided difference method. Here h is initial difference between the two points we have to calculate the integral in. We then successively divide the interval in powers of 2 and find the corresponding integral values using the trapezoidal method. I^0 represents precisely that.

$Interval$	I^0	I^1	I^2	I^3
h	$I^0(h)$			
		$I^1(h)$		
$h/2$	$I^0(h/2)$		$I^2(h)$	
		$I^1(h/2)$		$I^3(h)$
$h/4$	$I^0(h/4)$		$I^2(h/2)$	
		$I^1(h/4)$		
$h/8$	$I^0(h/8)$			

I^1, I^2 and others are calculated by the following relation which we find for each h :

$$I^m(h) = \frac{4^m I^{m-1}(h/2) - I^{m-1}(h)}{4^m - 1}$$

$$m = 1, 2, ...$$

4.2 Code & Output

```
import numpy as np
def f(x):
  return np.sqrt(1+np.power(np.cos(x),2))
N = [np.power(2,k) for k in range(1,11)]
I= []
for n in N:
  X= np.linspace(0,48,n)
  F = np.array([[f(i) for i in X]],dtype=float)
  h= X[1]-X[0]
  I.append((h/2)*(F[0,0]+F[0,-1] + 2*(np.sum(F[0,1:-1]))))
Y = I
for j in range(len(I)-1):
  Y = [((4**(j+1))*Y[m+1] - Y[m])/(4**(j+1) - 1) for m in range(0,len(Y)-1)]
print("Using the Romberg method")
print("Value of integral of f(x) = sqrt((cos(x))^2+1) over (0,48) is:", Y)
```

Figure 8: Code in emacs

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab9$ python3 MS18197_9_co
Using the Romberg method
Value of integral of f(x) = sqrt((cos(x))^2+1) over (0,48) is: [58.47046924546879]
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab9$
```

Figure 9: Output

5 Evaluate given integral by Gauss one point, two point and three point formula and compare with Romberg integration.

$$\int_0^1 \frac{1}{1+x}\,dx$$

5.1 Algorithm & Discussion

Gauss point methods involves changing the variables of the integral such that it resembles $\int_{-1}^{1} f(x)\,dx$.

As derived in class notes, Gauss one point rule says:

$$\int_{-1}^{1} f(x)\,dx = 2f(0)$$

Similarly Gauss two point rule says:

$$\int_{-1}^{1} f(x)\,dx = f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}})$$

Gauss three point rule says:

$$\int_{-1}^{1} f(x)\,dx = \frac{1}{9}[5f(-\sqrt{3/5}) + 8f(0) + 5f(\sqrt{3/5})]$$

In our case, we get the transformed integral:

$$\int_{-1}^{1} g(t)\,dt = \int_{-1}^{1} \frac{1}{t+3}\,dt$$

5.2 Code & Output

```
import numpy as np

def f(x):
  return 1/(1+x)
N = [np.power(2,k) for k in range(1,10)]
I= []
for n in N:
  X= np.linspace(0,1,n)
  F = np.array([[f(i) for i in X]],dtype=float)
  h= X[1]-X[0]
  I.append((h/2)*(F[0,0]+F[0,-1] + 2*(np.sum(F[0,1:-1]))))
Y = I
for j in range(len(I)-1):
  Y = [((4**(j+1))*Y[m+1] - Y[m])/(4**(j+1) - 1) for m in range(0,len(Y)-1)]
print("Value of integral of f(x) = 1/(1+x) over (0,1) is:", Y)


def g(t):
  return 1/(3+t)
print("\nChanging variables in f(x) gives us g(t)= 1/(3+t) over (-1,1)")
print("f(x) and g(t)= 1/(3+t) are equivalent")
print("\nValue of integral through gauss one point method:", 2*(g(0)))
m= 1/np.sqrt(3)
print("Value of integral through gauss two point method:", g(m)+  g(-m) )
m= np.sqrt(3/5)
val = (1/9)*(5*g(-m)+ 8*g(0)+5*g(m))
print("Value of integral through gauss three point method:", val )
```

Figure 10: Code in emacs

```
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab9$ python3
Value of integral of f(x) = 1/(1+x) over (0,1) is: [0.6931471799926368]

Changing variables in f(x) gives us g(t)= 1/(3+t) over (-1,1)
f(x) and g(t)= 1/(3+t) are equivalent

Value of integral through gauss one point method: 0.6666666666666666
Value of integral through gauss two point method: 0.6923076923076923
Value of integral through gauss three point method: 0.6931216931216931
nayakamlan@DESKTOP-C21G1MH:/mnt/c/Users/Dell/Desktop/CP1/Lab9$
```

Figure 11: Output

There is enormous time saving in Gauss three point method when compared to Romberg method due to less computation cost involved. Gauss three point method also gives an accurate result upto 4 decimals when compared to Romberg method.

6 Use Monte Carlo integration to solve the following with $10^3, 10^5, 10^6$ and 10^7 sample size.

$$\int_3^5 \frac{1}{\sqrt{x^2-4}}\,dx$$

$$\int_0^{48} \sqrt{1+cos^2(x)}\,dx$$

6.1 Algorithm & Discussion

Monte Carlo Integration is used normally to solve complicated integrals and higher dimension integrals.

We can approximate numerical integration of a integral using MC integration as:

$$\int_a^b f(x)\,dx = (b-a)\frac{1}{N}\sum_i^N f(x_i)$$

Here x_i is a randomly chosen point in the range of (a, b). Most of the code was written in numpy arrays for maximum efficiency in calculations. Here N is a large number of the order 10^6&10^7

Another method of calculating this would taking N points with $x \in (a, b)$ and $y \in (f_{min}, f_{max})$. We then calculated N_t points where $y < f(x)$. We can then calculated the integral using Monte Carlo integration:

$$\int_a^b f(x)\,dx = f_{max}(b-a)\frac{N_t}{N}$$

This works on the principle of proportional areas.

6.2 Code & Output

```
import numpy as np
import time
f = lambda x: 1/(x**2-4)**(0.5)
print("For f(x)= 1/sqrt(x^2 - 4) in x=(3,5)\n")
for N in [10**i for i in range(1,8)]:
  t1=time.time()
  Y= np.zeros((N,2))
  Y[:,0] = np.random.uniform(low=3, high=5, size=N)
  Y[:,1] = f(Y[:,0])
  nt = np.mean(Y[:,1]*(2))
  t2=time.time()
  print("\nMC Integration Value with N="+str(N)+":",np.round(nt,decimals=8))
  print("Time taken in secs:",np.round(t2-t1,decimals=8))
f = lambda x: np.sqrt(1+np.power(np.cos(x),2))
print("\nFor f(x)= sqrt(cos(x)^2 + 1) in x=(0,48)\n")
for N in [10**i for i in range(1,8)]:
  t1=time.time()
  Y= np.zeros((N,2))
  Y[:,0] = np.random.uniform(low=0, high=48, size=N)
  Y[:,1] = f(Y[:,0])
  nt = np.mean(Y[:,1]*(48))
  t2=time.time()
  print("\nMC Integration Value with N="+str(N)+":",np.round(nt,decimals=8))
  print("Time taken in secs:",np.round(t2-t1,decimals=8))
```

Figure 12: Code for MC integration in emacs

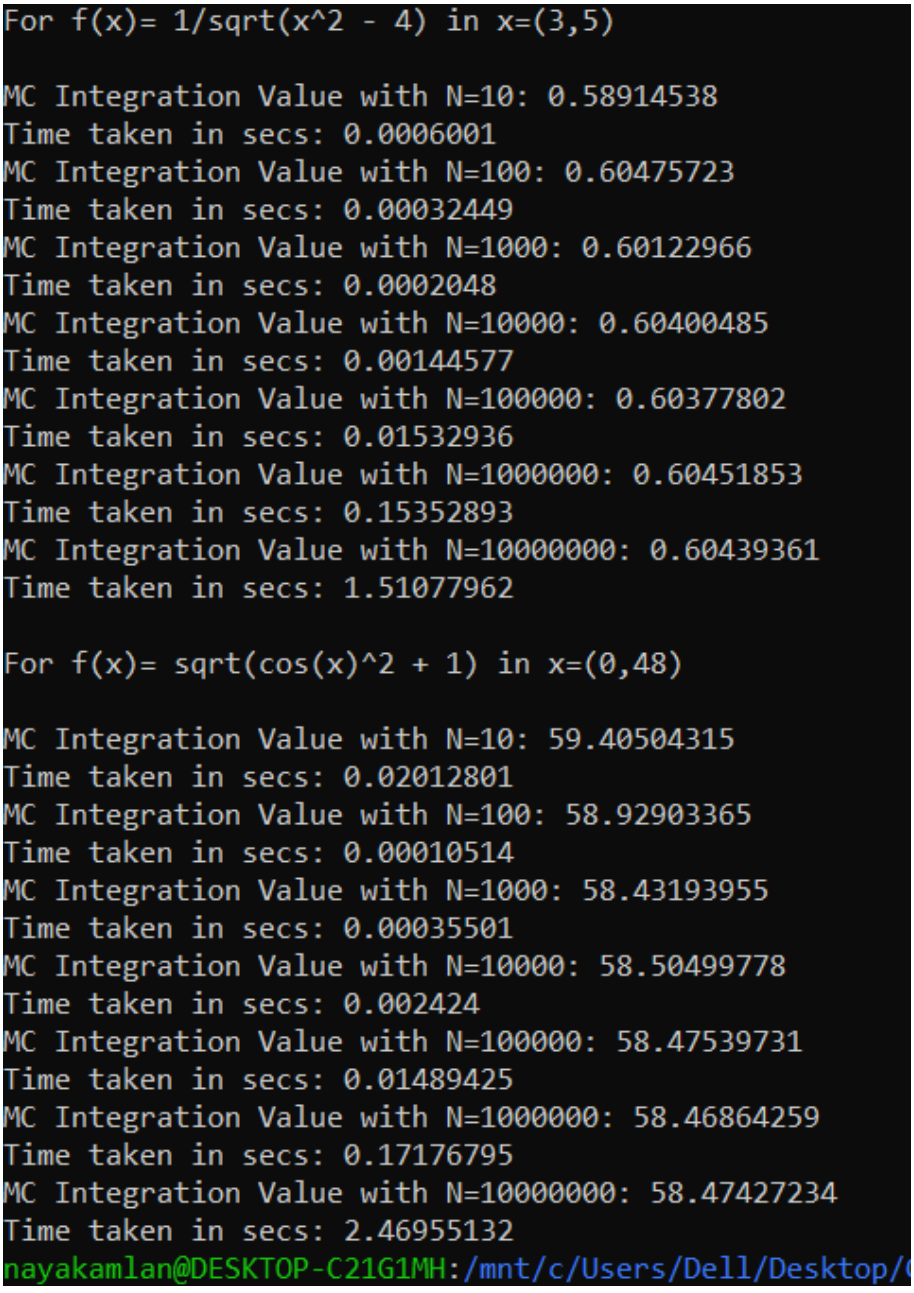

Figure 13: Output

7 Use Monte Carlo integration to solve the following

$$\int_1^4 \int_1^4 \frac{1}{\sqrt{x+(y+1)^2}}\, dx\, dy$$

7.1 Algorithm & Discussion

For two dimensions, We can approximate numerical integration of a integral using MC integration as:

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x,y)\, dx\, dy = (b_1 - a_1)(b_2 - a_2)\frac{1}{N}\sum_{i}^{N} f(x_i, y_i)$$

Here x_i and y_i are randomly chosen values in the suitable range of corresponding variable. Most of the code was written in numpy arrays for maximum efficiency in calculations.

7.2 Code & Output

```
1import numpy as np
2import time
3f = lambda x,y: 1/np.sqrt(x + np.power(y+1,2))
4print("For f(x,y)= 1/sqrt( x + (y+1)^2 ) in x,y=(1,4)\n")
5for N in [10**i for i in range(0,8)]:
6  t1=time.time()
7  Y= np.zeros((N,3))
8  Y[:,0] = np.random.uniform(low=1, high=4, size=N)
9  Y[:,1] = np.random.uniform(low=1, high=4, size=N)
10 Y[:,2] = f(Y[:,0],Y[:,1])
11 nt = np.mean(Y[:,2]*9)
12 t2=time.time()
13 print("\nMC Integration Value with N="+str(N)+":",np.round(nt,decimals=8))
14 print("Time taken in secs:",np.round(t2-t1,decimals=8))
```

Figure 14: Code for MC integration in emacs

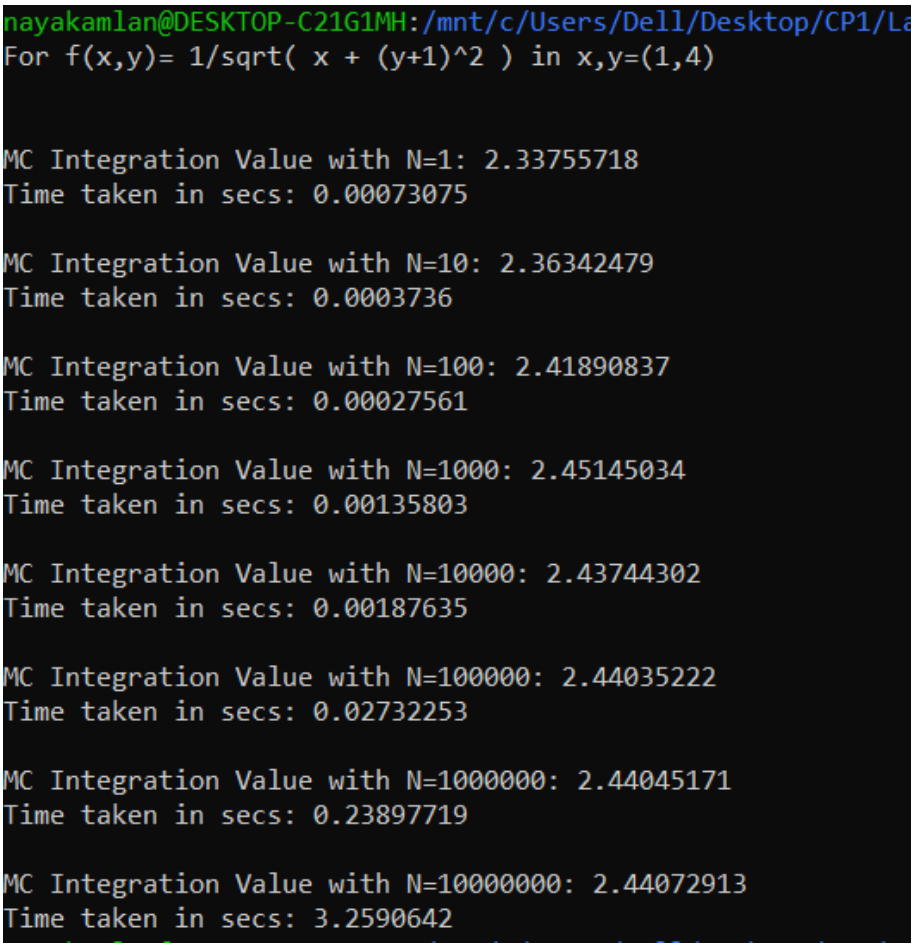

Figure 15: Output