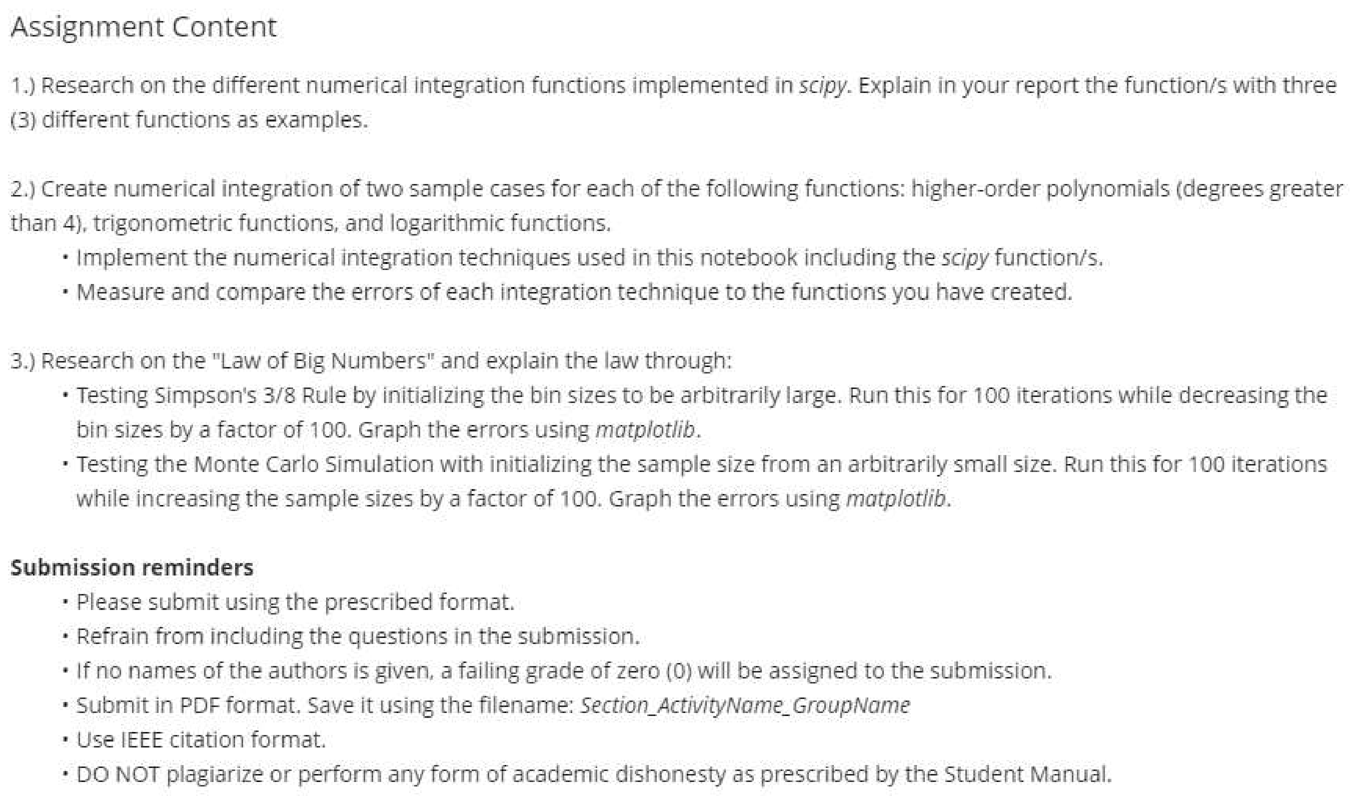
NUMERICAL INTEGRATION

SUBMITTED BY AND CODED BY GROUP *ICHI*:
FREDDIELYN AQUIRO
CHERRYLYN CANOZA
CHRISTOPHER JOAQUIN


Assignment Content

1.) Research on the different numerical integration functions implemented in *scipy*. Explain in your report the function/s with three (3) different functions as examples.

2.) Create numerical integration of two sample cases for each of the following functions: higher-order polynomials (degrees greater than 4), trigonometric functions, and logarithmic functions.
- Implement the numerical integration techniques used in this notebook including the *scipy* function/s.
- Measure and compare the errors of each integration technique to the functions you have created.

3.) Research on the "Law of Big Numbers" and explain the law through:
- Testing Simpson's 3/8 Rule by initializing the bin sizes to be arbitrarily large. Run this for 100 iterations while decreasing the bin sizes by a factor of 100. Graph the errors using *matplotlib*.
- Testing the Monte Carlo Simulation with initializing the sample size from an arbitrarily small size. Run this for 100 iterations while increasing the sample sizes by a factor of 100. Graph the errors using *matplotlib*.

Submission reminders
- Please submit using the prescribed format.
- Refrain from including the questions in the submission.
- If no names of the authors is given, a failing grade of zero (0) will be assigned to the submission.
- Submit in PDF format. Save it using the filename: *Section_ActivityName_GroupName*
- Use IEEE citation format.
- DO NOT plagiarize or perform any form of academic dishonesty as prescribed by the Student Manual.

```
###First question:
import numpy as np
from scipy import integrate
import matplotlib.pyplot as plt
from matplotlib import style
```

3 DIFFERENT NUMERICAL INTEGRATION FUNCTION USING SCIPY

1. QUAD FUNCTION

This function allows the user to compute the definite integrals of an equation. [1]. *f* symbolizes the equation that it needs to be integrated, *a* serves as the lower limit of integration and *b* represents the upper limit of integration

```
###First question
###Quadpack equation
#Single Integral
f = lambda x: x**2
integrate.quad (f,0,10)
```

```
(333.33333333333326, 3.700743415417188e-12)
```

2. DOUBLE QUAD FUNCTION

One of the quad pack packages that can solve double integral. [2] In *f*, it must have at least two variables (x and y), x serves as the first agrument and y is the second argument. The *a* and *b* are the limits of integration.

```
###Double Integral
f = lambda y, x: x*y**2
integrate.dblquad(f, 0, 2, lambda x: 0, lambda x: 1)
```

```
(0.6666666666666667, 7.401486830834377e-15)
```

3. ROMBERG INTEGRATION FUNCTION

This function is under of scipy integration function which the romberg integration using samples of a function. [3]

```
###Romberg Integration
x = np.arange(10, 14.25, 0.25)
y = np.arange(3, 12)

y = np.sin(np.power(x, 2.5))
integrate.romb(y)
```

```
-0.742561336672229
```

APPLICATION OF DIFFERENT NUMERICAL INTEGRATION FUNCTION IN POLYNOMIALS, LOGARTHMIC AND TRIGONOMTRIC EQUATION

The methods that the researchers used in Trapezoidal function, Simpson's 1/3 and 3/8 function and Monte Carlo Function were found in their proctor's discussion. [4]

POLYNOMIAL EQUATIONS

ScipyFunction

```
###ScipyFunc
###Polynomial Equation 1
def f(x):
  return 3.0*x**4+2*x**2

i,err = integrate.quad(f,-1,1)
print(i)
print(err)
```

```
2.5333333333333337
2.8125649957170637e-14
```

```
###ScipyFunc
###Polynomial Equation 2
def f(x):
  return x**10-6*x**5+2*x**3

i,err = integrate.quad(f,-2.3,2.3)
print(i)
print(err)
```

```
1732.3813780253208
1.9282457170327675e-11
```

Trapeziodal Rule

```
###Trapeziodal Rule
###Polynomial Equation 1
def trapz_rule(func,lb,ub,size):
  h = (ub-lb)/size
  return h*(0.5*(func(lb)+func(ub))+np.sum(func(lb+h*np.arange(1,size))))

f = lambda x: 3.0*x**4+2*x**2
sum = trapz_rule(f, -1,1,1e4)
print(sum)
f_s = 2.5333 #calculated integration
err = abs(f_s-sum)
print(err)
```

```
2.5333334400000003
3.344000000016223e-05
```

```
###Trapeziodal Rule
###Polynomial Equation 2
f = lambda x: x**10-6*x**5+2*x**3
sum = trapz_rule(f, -2.3,2.3,1e4)
print(sum)
f_s = 1732.38 #calculated integration
err = abs(f_s-sum)
print(err)
```

```
1732.3820132317949
0.00201323179476276
```

Simpson's 1/3 Rule

```
###Simpson's 1/3 Rule
######Polynomial Equation 1
def simp_13(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(4*func(lb+h*np.arange(1,divs,2)))+ \
          np.sum(2*func(lb+h*np.arange(2,divs,2)))
  S = (h/3)*A
  return S

h = lambda x: 3.0*x**4+2*x**2
sum = simp_13(h, -1,1,1e4)
print(sum)
h_s = 2.5333
err = abs(h_s-sum)
print(err)
```

```
2.5333333333333354
3.3333333335328064e-05
```

```
###Simpson's 1/3 Rule
######Polynomial Equation 2
def simp_13(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(4*func(lb+h*np.arange(1,divs,2)))+ \
          np.sum(2*func(lb+h*np.arange(2,divs,2)))
  S = (h/3)*A
  return S

h = lambda x: x**10-6*x**5+2*x**3
sum = simp_13(h, -2.3,2.3,1e4)
```

```
print(sum)
h_s = 1732.38
err = abs(h_s-sum)
print(err)
```

```
    1732.3813780254422
    0.001378025442136277
```

Simpson's 3/8 Rule

```
###Simpson's 3/8 Rule
######Polynomial Equation 1
def simp_38(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(3*(func(lb+h*np.arange(1,divs,3))))+ \
          np.sum(3*(func(lb+h*np.arange(2,divs,3))))+ \
          np.sum(2*func(lb+h*np.arange(3,divs,3)))
  S = (3*h/8)*A
  return S

z = lambda x:3.0*x**4+2*x**2
sum = simp_38(z, -1,1,1e4)
print(sum)
z_s = 2.5333
err = abs(z_s-sum)
print(err)
```

```
    2.533083413340001
    0.00021658665999924054
```

```
###Simpson's 3/8 Rule
######Polynomial Equation 2
g = lambda x: x**10-6*x**5+2*x**3
sum = simp_38(g,-2.3,2.3,1e4)
print(sum)
g_s = 1732.38
err = abs(g_s-sum)
print(err)
```

```
    1731.9470406821463
    0.4329593178538289
```

Monte Carlo Rule

```
###Monte Carlo Rule
######Polynomial Equation 1
```

```
a, b = -1, 1
n = 1e3
samples = np.random.uniform(a,b,int(n))
o = lambda x: 3.0*x**4+2*x**2
A = np.sum(o(samples))
S = ((b-a)/n)
S ###Error
```

```
0.002
```

```
###Monte Carlo Rule
######Polynomial Equation 2
a, b = -2.3, 2.3
n = 1e3
samples = np.random.uniform(a,b,int(n))
o = lambda x: x**10-6*x**5+2*x**3
A = np.sum(o(samples))
S = ((b-a)/n)
S ###Error
```

```
0.0046
```

TRIGONOMETRIC EQUATIONS

ScipyFunction

```
###ScipyFunc
###Trigonometric Equation 1
def f(x):
  return np.exp(-x)*np.sin(3.0*x)

i,err = integrate.quad(f,0,2*np.pi)
print(i)
print(err)
```

```
0.29943976718048754
5.05015300411582e-13
```

```
###ScipyFunc
###Trigonometric Equation 2
def f(x):
  return (np.cos(x)**2)*np.sin(x)

i,err = integrate.quad(f,0,np.pi/2)
print(i)
print(err)
```

```
0.3333333333333333
3.700743415417188e-15
```

Trapezoidal Rule

```
###Trapeziodal Rule
###Trigonometric Equation 1
def trapz_rule(func,lb,ub,size):
  h = (ub-lb)/size
  return h*(0.5*(func(lb)+func(ub))+np.sum(func(lb+h*np.arange(1,size))))

f = lambda x: np.exp(-x)*np.sin(3.0*x)
sum = trapz_rule(f, 0,2*np.pi,1e4)
print(sum)
f_s = 0.2944 #calculated integration
err = abs(f_s-sum)
print(err)
```

```
0.29943966866874894
0.005039668668748942
```

```
###Trapeziodal Rule
###Trigonometric Equation 2
def trapz_rule(func,lb,ub,size):
  h = (ub-lb)/size
  return h*(0.5*(func(lb)+func(ub))+np.sum(func(lb+h*np.arange(1,size))))

f = lambda x: (np.cos(x)**2)*np.sin(x)
sum = trapz_rule(f, 0,np.pi/2,1e4)
print(sum)
f_s = 0.33333 #calculated integration
err = abs(f_s-sum)
print(err)
```

```
0.33333333127716575
3.3312771657301177e-06
```

Simpson's 1/3 Rule

```
###Simpson's 1/3 Rule
######Trigonometric Equation 1
def simp_13(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(4*func(lb+h*np.arange(1,divs,2)))+ \
          np.sum(2*func(lb+h*np.arange(2,divs,2)))
  S = (h/3)*A
  return S

h = lambda x: np.exp(-x)*np.sin(3.0*x)
sum = simp_13(h,0,2*np.pi,1e4)
print(sum)
h_s = 0.2944
err = abs(h_s-sum)
print(err)
```

```
0.2994397671805031
0.005039767180503085
```

```
###Simpson's 1/3 Rule
######Trigonometric Equation 2
def simp_13(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(4*func(lb+h*np.arange(1,divs,2)))+ \
          np.sum(2*func(lb+h*np.arange(2,divs,2)))
  S = (h/3)*A
  return S

h = lambda x: (np.cos(x)**2)*np.sin(x)
sum = simp_13(h,0,np.pi/2,1e4)
print(sum)
h_s = 0.33333
err = abs(h_s-sum)
print(err)
```

```
0.33333333333333337
3.3333333333551707e-06
```

Simpson's 3/8 Rule

```
###Simpson's 3/8 Rule
######Trigonometric Equation 1
def simp_38(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(3*(func(lb+h*np.arange(1,divs,3))))+ \
          np.sum(3*(func(lb+h*np.arange(2,divs,3))))+ \
          np.sum(2*func(lb+h*np.arange(3,divs,3)))
  S = (3*h/8)*A
  return S

z = lambda x:np.exp(-x)*np.sin(3.0*x)
sum = simp_38(z,0,2*np.pi,1e4)
print(sum)
z_s = 0.2944
```

```
err = abs(z_s-sum)
print(err)
```

```
0.29943976745692863
0.005039767456928634
```

```
###Simpson's 3/8 Rule
######Trigonometric Equation 2
def simp_38(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(3*(func(lb+h*np.arange(1,divs,3))))+ \
          np.sum(3*(func(lb+h*np.arange(2,divs,3))))+ \
          np.sum(2*func(lb+h*np.arange(3,divs,3)))
  S = (3*h/8)*A
  return S

z = lambda x:(np.cos(x)**2)*np.sin(x)
sum = simp_38(z,0,np.pi/2,1e4)
print(sum)
z_s = 0.33333
err = abs(z_s-sum)
print(err)
```

```
0.33333333333349485
3.3333334948371096e-06
```

Monte Carlo Rule

```
###Monte Carlo Rule
######Trigonometric Equation 1
a, b = 0, 2*np.pi
n = 1e3
samples = np.random.uniform(a,b,int(n))
o = lambda x: np.exp(-x)*np.sin(3.0*x)
A = np.sum(o(samples))
S = ((b-a)/n)
S ###Error
```

```
0.006283185307179587
```

```
###Monte Carlo Rule
######Trigonometric Equation 2
a, b = 0, np.pi/2
n = 1e3
samples = np.random.uniform(a,b,int(n))
o = lambda x: (np.cos(x)**2)*np.sin(x)
A = np.sum(o(samples))
S = ((b-a)/n)
```

```
S ###Error
```

```
    0.0015707963267948967
```

LOGARITHMIC EQUATIONS

ScipyFunction

```
###ScipyFunc
###logarithmic Equation 1
def f(x):
  return np.log(1+x**2)

i,err = integrate.quad(f,-1.7,1.7)
print(i)
print(err)
```

```
    1.974880174087571
    7.596196078835799e-13
```

```
###ScipyFunc
###logarithmic Equation 2
def f(x):
  return np.log(1+np.exp(x))

i,err = integrate.quad(f,-5,5)
print(i)
print(err)
```

```
    14.131480805093181
    4.820608159612078e-13
```

Trapezoidal Rule

```
###Trapeziodal Rule
###Logarithmic Equation 1
def trapz_rule(func,lb,ub,size):
  h = (ub-lb)/size
  return h*(0.5*(func(lb)+func(ub))+np.sum(func(lb+h*np.arange(1,size))))

f = lambda x: np.log(1+x**2)
sum = trapz_rule(f, -1.7,1.7,1e4)
print(sum)
f_s = 1.97488 #calculated integration
err = abs(f_s-sum)
print(err)
```

```
1.9748801909273306
1.9092733061221168e-07
```

```
###Trapeziodal Rule
###Logarithmic Equation 2
def trapz_rule(func,lb,ub,size):
  h = (ub-lb)/size
  return h*(0.5*(func(lb)+func(ub))+np.sum(func(lb+h*np.arange(1,size))))

f = lambda x: np.log(1+np.exp(x))
sum = trapz_rule(f, -5,5,1e4)
print(sum)
f_s = 14.1315 #calculated integration
err = abs(f_s-sum)
print(err)
```

```
14.13148088731104
1.9112688960376545e-05
```

Simpson 1/3 Rule

```
###Simpson's 1/3 Rule
######Logarithmic Equation 1
def simp_13(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(4*func(lb+h*np.arange(1,divs,2)))+ \
          np.sum(2*func(lb+h*np.arange(2,divs,2)))
  S = (h/3)*A
  return S

h = lambda x: np.log(1+x**2)
sum = simp_13(h,-1.7,1.7,1e4)
print(sum)
h_s = 1.97488
err = abs(h_s-sum)
print(err)
```

```
1.9748801740875706
1.7408757058134938e-07
```

```
###Simpson's 1/3 Rule
######Logarithmic Equation 2
def simp_13(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(4*func(lb+h*np.arange(1,divs,2)))+ \
          np.sum(2*func(lb+h*np.arange(2,divs,2)))
  S = (h/3)*A
  return S

h = lambda x: np.log(1+np.exp(x))
sum = simp_13(h,-5,5,1e4)
print(sum)
h_s = 14.1315
err = abs(h_s-sum)
print(err)
```

```
14.131480805093181
1.919490681956404e-05
```

Simpson's 3/8 Rule

```
###Simpson's 3/8 Rule
###Logarithmic Equation 1
def simp_38(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(3*(func(lb+h*np.arange(1,divs,3))))+ \
          np.sum(3*(func(lb+h*np.arange(2,divs,3))))+ \
          np.sum(2*func(lb+h*np.arange(3,divs,3)))
  S = (3*h/8)*A
  return S

z = lambda x:np.log(1+x**2)
sum = simp_38(z,-1.7,1.7,1e4)
print(sum)
z_s = 1.97488
err = abs(z_s-sum)
print(err)
```

```
1.9747647219387874
0.00011527806121258699
```

```
###Simpson's 3/8 Rule
###Logarithmic Equation 2
def simp_38(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
          np.sum(3*(func(lb+h*np.arange(1,divs,3))))+ \
          np.sum(3*(func(lb+h*np.arange(2,divs,3))))+ \
          np.sum(2*func(lb+h*np.arange(3,divs,3)))
  S = (3*h/8)*A
  return S

z = lambda x:np.log(1+np.exp(x))
sum = simp_38(z,-5,5,1e4)
```

```
print(sum)
z_s = 14.1315
err = abs(z_s-sum)
print(err)
```

```
14.130229250419594
0.001270749580406516
```

Monte Carlo Rule

```
###Monte Carlo Rule
###Logarithmic Equation 1
a, b = -1.7, 1.7
n = 1e3
samples = np.random.uniform(a,b,int(n))
o = lambda x: np.log(1+x**2)
A = np.sum(o(samples))
S = ((b-a)/n)
S ###Error
```

```
0.0034
```

```
###Monte Carlo Rule
###Logarithmic Equation 2
a, b = -5, 5
n = 1e3
samples = np.random.uniform(a,b,int(n))
o = lambda x: np.log(1+np.exp(x))
A = np.sum(o(samples))
S = ((b-a)/n)
S ###Error
```

```
0.01
```

LAW OF BIG NUMBERS

Simpson's 3/8 Rule

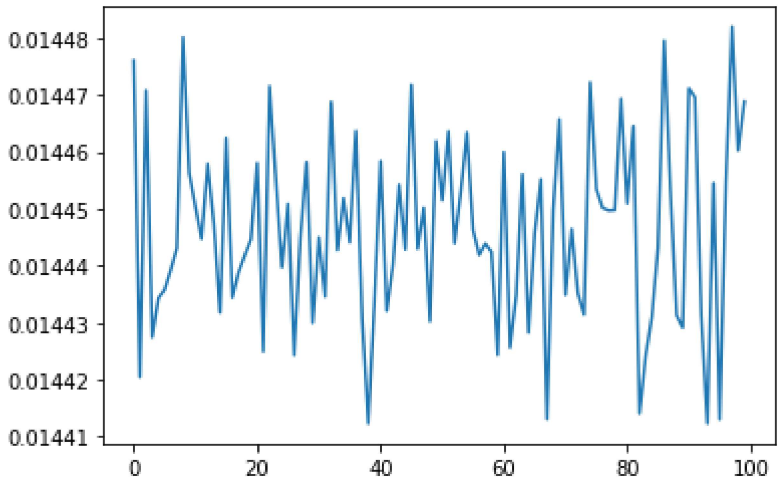
```
###Simpson's 3/8 Rule

def simp_38(func,lb,ub,divs):
  h = (ub-lb)/divs
  A = (func(lb)+func(ub))+ \
            np.sum(3*(func(lb+h*np.arange(1,divs,3))))+ \
            np.sum(3*(func(lb+h*np.arange(2,divs,3))))+ \
            np.sum(2*func(lb+h*np.arange(3,divs,3)))
```

```
          np.sum(2*func(a+h*np.arange(3,divs,3)))
  S = (3*h/8)*A
  return S

z = lambda x: np.random.uniform(a,b,int(n))
sum = simp_38(z,0,1,1e4)
print(sum)
##z_s = 1.97488
##err = abs(z_s-sum)
##print(err)
fig= plt.figure()
plt.plot(sum)
plt.show()
```

```
[0.014476   0.01442048 0.01447076 0.01442744 0.0144343  0.01443575
 0.01443924 0.01444309 0.01448009 0.01445629 0.01445042 0.0144447
 0.01445788 0.0144473  0.01443178 0.01446241 0.01443441 0.01443886
 0.01444179 0.01444459 0.01445802 0.01442488 0.0144715  0.01445558
 0.01443972 0.01445086 0.0144243  0.01444497 0.01445816 0.01442996
 0.01444499 0.01443455 0.01446873 0.01444272 0.0144519  0.01444409
 0.01446365 0.01443087 0.0144123  0.01443379 0.01445838 0.01443208
 0.0144401  0.01445424 0.01444284 0.01447173 0.01444295 0.01445019
 0.01443022 0.01446189 0.0144515  0.01446359 0.01444394 0.01445288
 0.0144634  0.01444626 0.01444187 0.0144439  0.01444238 0.01442435
 0.01445995 0.01442561 0.01443475 0.01445608 0.01442826 0.01444567
 0.01445517 0.01441308 0.01444981 0.01446572 0.01443486 0.01444646
 0.01443513 0.01443142 0.01447218 0.01445342 0.01445017 0.01444971
 0.01444979 0.01446932 0.01445095 0.01446452 0.01441405 0.01442423
 0.01443099 0.01444318 0.01447951 0.01445352 0.0144313  0.01442911
 0.01447112 0.01446955 0.01443143 0.01441232 0.01445452 0.01441304
 0.01445459 0.01448196 0.01446025 0.0144688 ]
```

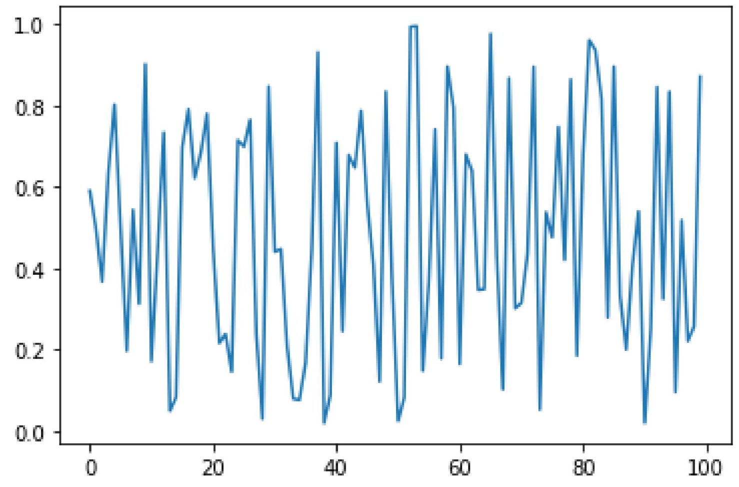


Monte Carlo Rule

```
###Monte Carlo
a, b = 0, 1
n = 1e2
```

```
samples = np.random.uniform(a,b,int(n))
print(samples.size)
print(samples)
fig= plt.figure()
plt.plot(samples)
plt.show()
```

```
100
[0.58991887 0.49699168 0.36683007 0.63495482 0.80180426 0.49818271
 0.19615215 0.54313221 0.31179739 0.90149532 0.17105774 0.44225619
 0.73342001 0.04933632 0.08243803 0.69635223 0.7902021  0.62082606
 0.68351138 0.77951083 0.44967219 0.21527999 0.23774947 0.145562
 0.71453682 0.69842459 0.76462202 0.23946318 0.02940464 0.84662154
 0.44013341 0.44649288 0.20805997 0.07890746 0.07601545 0.16686421
 0.44464874 0.93038544 0.01908493 0.08723751 0.70802472 0.2448365
 0.67787001 0.64673416 0.78665727 0.56140032 0.4069359  0.12097407
 0.83385248 0.3779061  0.02443915 0.08204543 0.99298672 0.99473367
 0.14762389 0.36823816 0.741175   0.17754875 0.89526226 0.79405214
 0.16403092 0.67935642 0.63846731 0.34657758 0.34846981 0.97605591
 0.43078852 0.10182686 0.86707724 0.30102307 0.31431165 0.43557776
 0.89487995 0.05219523 0.53730497 0.47592993 0.74732425 0.42009182
 0.8641214  0.18468493 0.67552923 0.96021699 0.93613746 0.81528491
 0.27812388 0.89525631 0.33270628 0.19955146 0.4047135  0.53925486
 0.01952671 0.25141816 0.84559317 0.3243056  0.83328823 0.09539812
 0.51750057 0.22038428 0.25613406 0.87025293]
```



Reference:

[1] “scipy.integrate.quad — SciPy v1.6.3 Reference Guide.” https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html#scipy.integrate.quad (accessed May 16, 2021).
[2] “scipy.integrate.dblquad — SciPy v1.6.3 Reference Guide.” https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.dblquad.html#scipy.integrate.dblquad (accessed May 16, 2021).
[3] “scipy.integrate.romb — SciPy v1.6.3 Reference Guide.” https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.romb.html#scipy.integrate.romb (accessed May 16, 2021).

[4] “numeth2021/NuMeth_6_Numerical_Integration.ipynb at main · dyjdlopez/numeth2021 · GitHub.” https://github.com/dyjdlopez/numeth2021/blob/main/Week%2015%20-%20Numerical%20Integration/NuMeth_6_Numerical_Integration.ipynb (accessed May 16, 2021).