EE 511

PROJECT # 4

BY

PRANAV GUNDEWAR

USC ID: 4463612994

EMAIL: gundewar@usc.edu

Problem 1 Pi- Estimation

Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. Their essential idea is using randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches. Monte Carlo methods are mainly used in three distinct problem classes: optimization, numerical integration, and generating draws from a probability distribution.

Summary:

- Generate n=100 samples of i.i.d 2-dimensional uniform random variables in the unit-square. Count how many of these samples fall within the quarter unit-circle centred at the origin. This quarter circle inscribes the unit square as shown below
- Using area estimate, we are supposed to estimate value of pi. Run the simulation for 50 times
- Plot the sample variance for different values of samples.

Approach:

- Unit circle has radius of 1. Hence, we will generate 100 random points within unit circle radius. If the sum of squares of 2 random points is within the radius, then we accept the sample point else we reject.
- The sample variance has been plotted for different samples values by calculating variance from mean of estimation.

Consider a quadrant inscribed in a unit square. Given that the ratio of their areas is $\pi/4$, the value of π can be approximated using a Monte Carlo method:

1. Draw a square, then inscribe a quadrant within it
2. Uniformly scatter a given number of points over the square
3. Count the number of points inside the quadrant, i.e. having a distance from the origin of less than 1
4. The ratio of the inside-count and the total-sample-count is an estimate of the ratio of the two areas, $\pi/4$. Multiply the result by 4 to estimate π.

Result and Analysis:

There are two important points:

1. If the points are not uniformly distributed, then the approximation will be poor.

2. There are a large number of points. The approximation is generally poor if only a few points are randomly placed in the whole square. On average, the approximation improves as more points are placed.

Uses of Monte Carlo methods require large amounts of random numbers, and it was their use that spurred the development of pseudorandom number generators, which were far quicker to use than the tables of random numbers that had been previously used for statistical sampling.

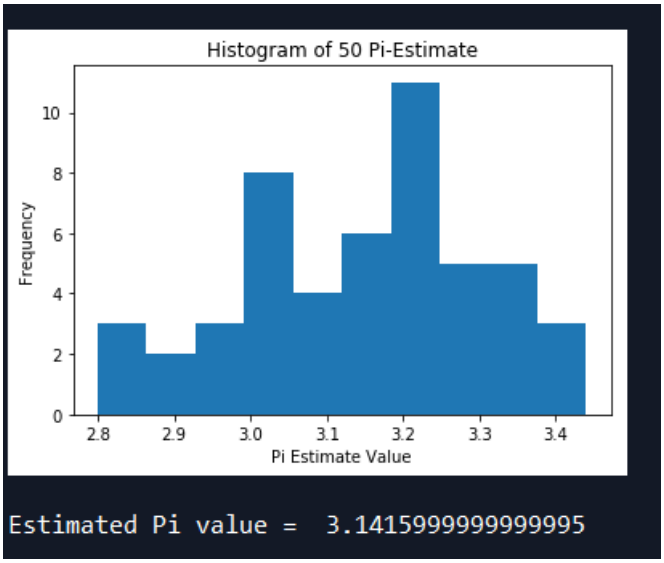

Figure 1. Pi- estimation for 50 estimates

From figure 1, we can observe that estimated value is very close to actual pi-value. Hence, using Monte-Carlo estimation, we have successfully estimated the value over the number of estimates averaged.

From figure 2, we can observe the points which satisfies the equation are inside the unit circle and are plotted in red and points which doesn't satisfy are outside plotted in green. The variance of data is its deviation from mean. I have plotted the equation in red line to make understanding easy.

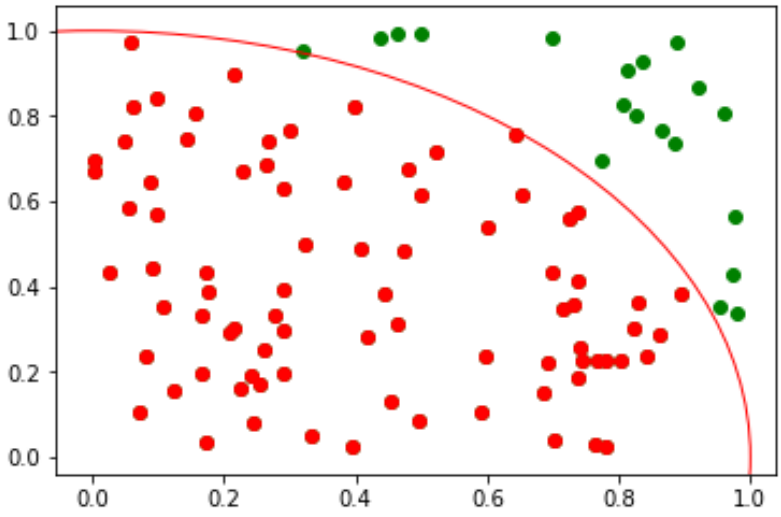

Figure 2. Pi- estimation scatter plot

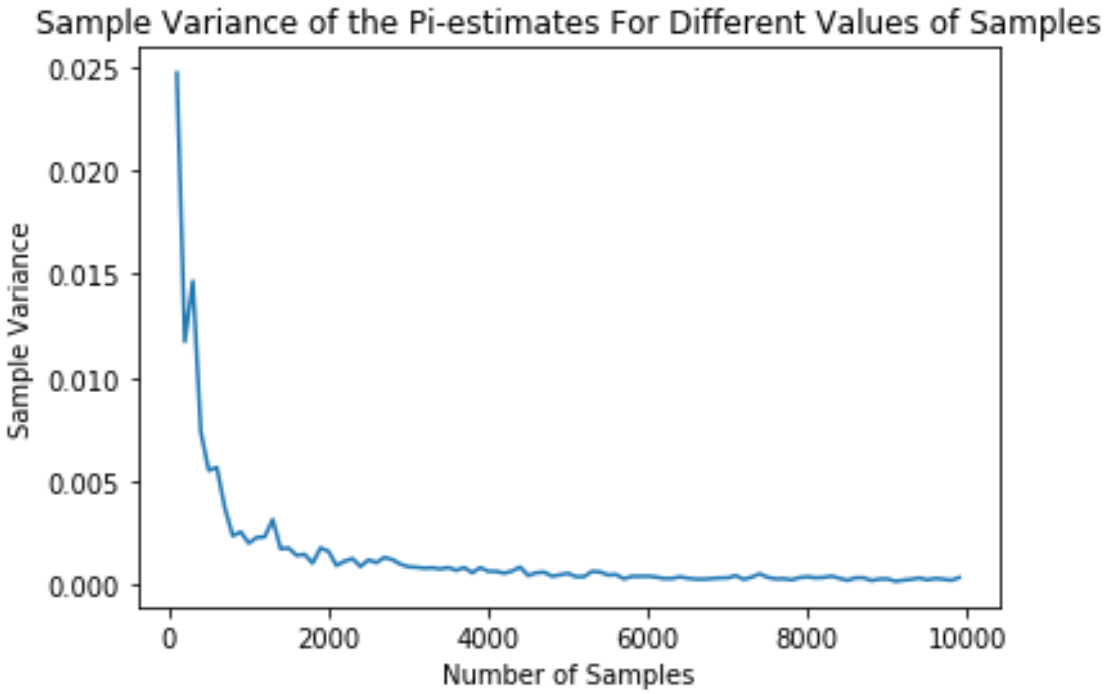

Figure 3. Sample Variance for different sample values

Figure 3 shows sample variance of Pi-estimation for different number of samples. As number of samples increases, the sample variance decreases. The computation increases and time required to run the code also increases as samples increases but the variance will decrease. Thus, the more Monte Carlo sample size, the less estimate variance.

Problem 2 Monte Carlo Integration and Variance Reduction Strategies

Summary:

- Estimate integration using Monte-Carlo estimation for given definite integrals for 1000 samples
- For the same estimation, we have to incorporate importance sampling and stratified sampling by observing function plot
- In the final part, we are asked to test the given integral using own choice of number of samples.

Approach:

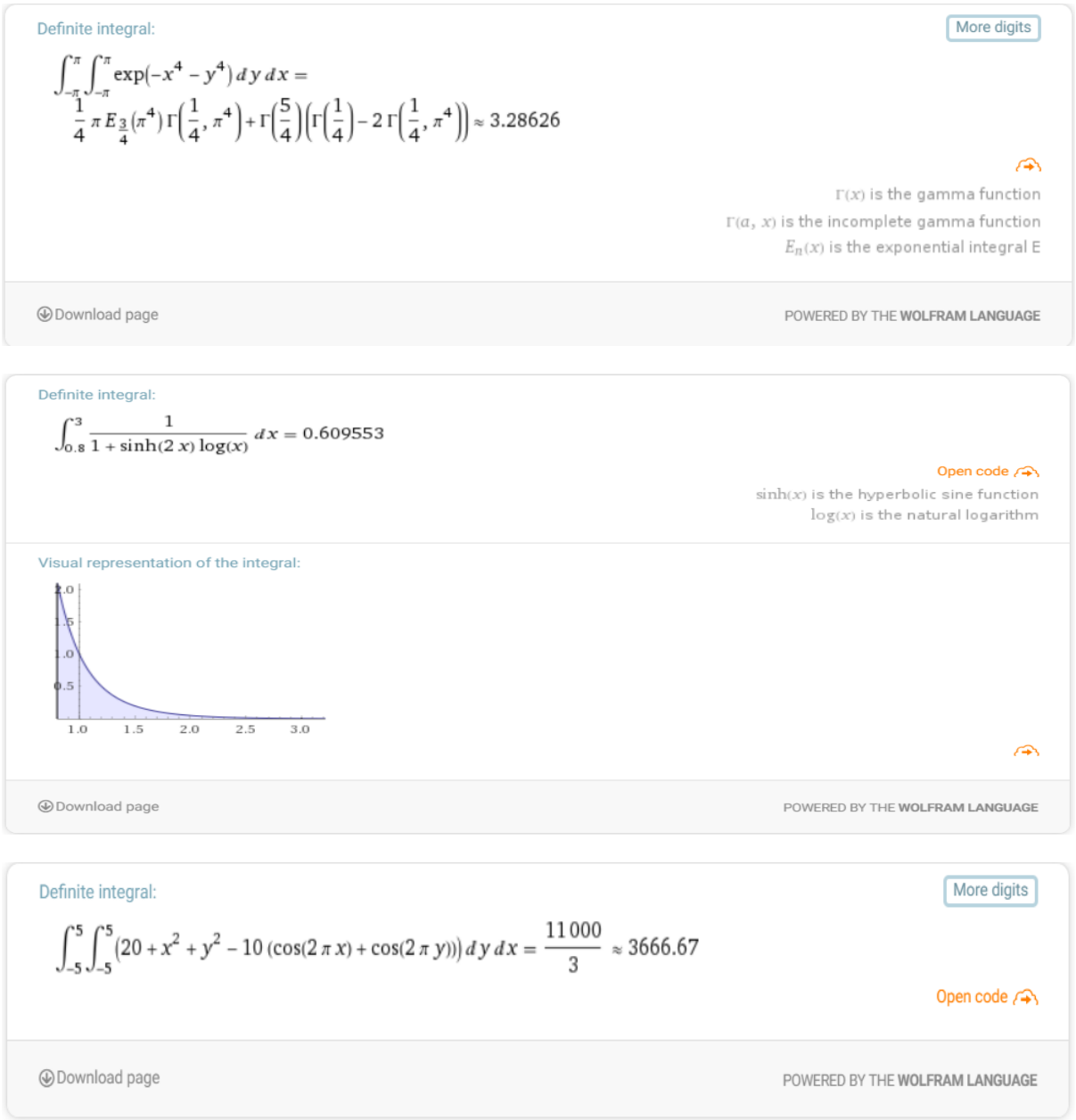

Figure 4. Exact values of given definite integrals

I have used Mathematica to calculate exact values of given definite integrals. The integrals and respective value is shown in figure 4. To estimate the integrals using Monte-Carlo, there 3 types named uniform sampling, stratification and importance sampling. All the types have been explained in detail below.

Stratification-

This technique is used to reduce the variance of the estimated value from mean.

Recursive stratified sampling is a generalization of one-dimensional adaptive quadrature's to multi-dimensional integrals. On each recursion step the integral and the error are estimated using a plain Monte Carlo algorithm. If the error estimate is larger than the required accuracy the integration volume is divided into sub-volumes and the procedure is recursively applied to sub-volumes.

The stratified sampling algorithm concentrates the sampling points in the regions where the variance of the function is largest thus reducing the grand variance and making the sampling more effective, as shown on the illustration.

The basic idea is variance of function over a subinterval of should be lower than the variance over whole interval. Prevent draws from clustering in a particular region of the interval; The procedure is forced to visit each subinterval. The information set used is enlarged.

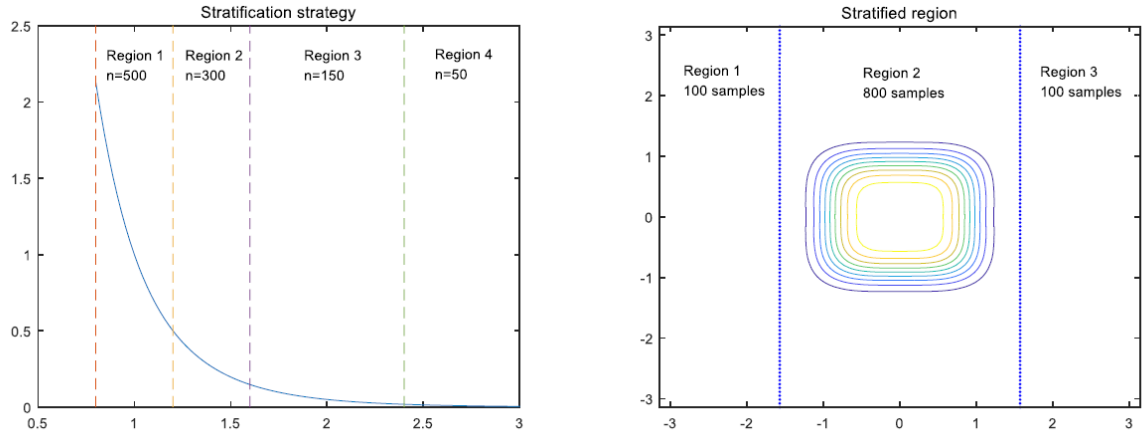

Figure 5. Stratification idea

Importance Sampling-

- By drawing numbers for a uniform distribution in crude Monte Carlo methods, information is spread all over the interval we are sampling over.
- A simple transformation of the problem may exist for which Monte Carlo can generate a far better result in terms of variance.
- Suppose a function g(x) exists such that h(x) = f(x)/g(x) is almost constant over the domain of integration. Restate the problem

$$\int f(x)dx = \int \frac{f(x)}{g(x)} g(x)dx = \int h(x)g(x)dx$$

- We can now easily integrate f by instead sampling h(x), but not by drawing numbers from a uniform density function, but rather from a nonuniform density g(x)dx.

The VEGAS algorithm takes advantage of the information stored during the sampling, and uses it and importance sampling to efficiently estimate the integral *I*. It samples points from the probability distribution described by the function |*f*| so that the points are concentrated in the regions that make the largest contribution to the integral.

By observing 2 integrands, I choose truncated normal gaussian for first equation and bivariate normal for the second as both PDF closely matches plot of integrals.

Hence, I choose truncated Gaussian from 0.8-3 for first equation and mean = [0,0] and covariance matrix I = [0.5 0; 0 0.5]

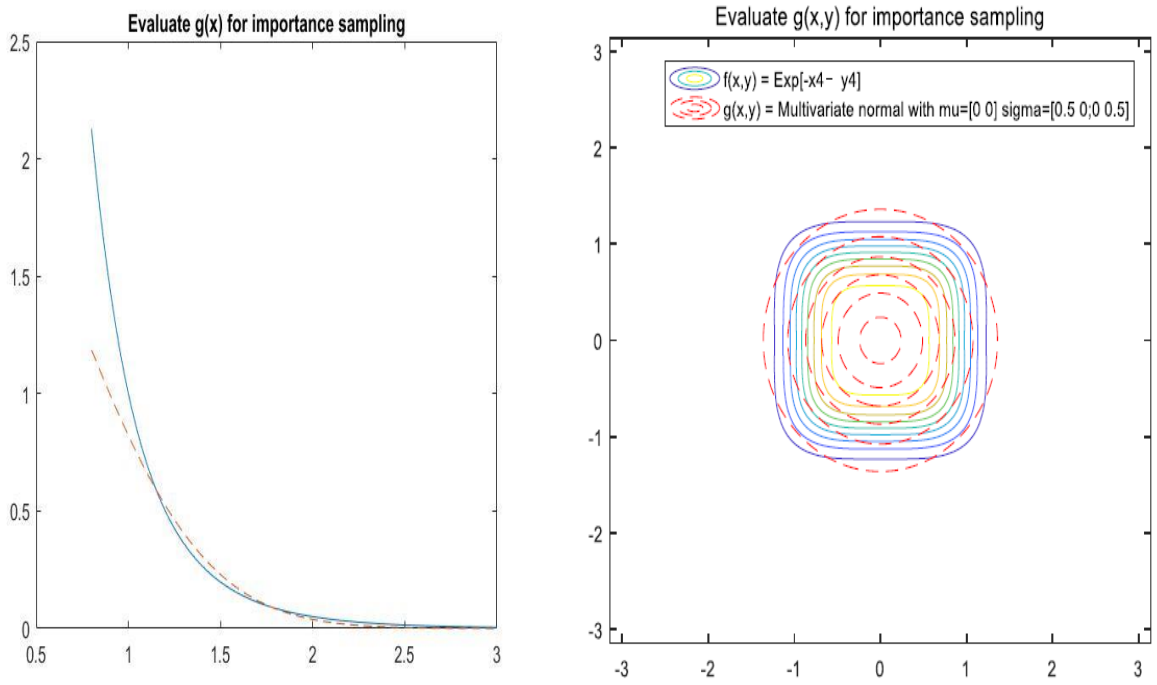

Figure 6. Importance Sampling

Figure 6 shows basic idea about estimated PDF and original PDF and gives us insight about importance sampling. Estimating the correct distribution is important.

I have estimated given all three integrals using discussed method shown below in figure 7.

Result and Analysis:

```
Different Monte Carlo Estimation for function 1/(1+sinh(2*x)*log(x)

Monte Carlo estimation using Uniform Sampling
Estimation = 0.612007 Variance = 0.000899
Monte Carlo estimation using Stratified Sampling
Estimation = 0.613276 Variance = 0.000217
Monte Carlo estimation using Importance Sampling
Estimation = 0.610901 Variance = 0.000127
```

```
Different Monte Carlo Estimation for function e^(-x^4-y^4)

Monte Carlo estimation using Uniform Sampling
Estimation = 3.274825 Variance = 0.091577
Monte Carlo estimation using Stratified Sampling
Estimation = 3.313443 Variance = 0.070298
Monte Carlo estimation using Importance Sampling
Estimation = 3.284664 Variance = 0.008341
```

```
Different Monte Carlo Estimation for function 20+x^2+y^2-10(cos(2*pi*x)+cos(2*pi*y))

Monte Carlo estimation using Uniform Sampling
Estimation = 3666.335053 Variance = 0.134407
```

Figure 7. Monte Carlo Estimation

From figure, uniform sampling has highest variance but it decreases as we use stratification. Importance sampling has lowest variance hence used widely everywhere. Hence, quality of importance sampling is highest followed by stratification and last comes uniform.

The strengths and weakness

Uniform sampling is the simple way to estimate integral. The variance could be reduced by increasing number of samples. But the weakness is that the variance is the biggest among the three methods. To get higher precision, we need a very big n and a lot of time of calculation.

Stratification sampling can reduce variance significantly. But we need to analysis the integrand and draw its plot to find proper regions. This method depends on the integrand. In multi-dimensional space, it may very difficult to observe and stratify the integrand.

Importance sampling can reduce variance further than stratification. But this method depends on whether we can find a proper pdf which is similar to the integrand and easy to generate random samples.

Test estimate integration-

The last function is very complex and hard to find a proper pdf or stratified region. I use uniform stratification sampling method which uses uniform PDF to calculate random numbers between -5 to 5

Code:

```
"""

@author: Pranav Gundewar

Project #4: Investigations on Monte Carlo Methods

Q1- Pi- Estimation

"""

#Importing Libraries

import numpy as np

import matplotlib.pyplot as plt

from scipy import optimize

import random

from math import sqrt

############################################## PART A
########################################


Pi_Array = np.zeros((50,1))

colors = ['g', 'b', 'y', 'c', 'm']

for i in range(0,50):

    cnt = 0

    a = np.random.uniform(0,1,size=(100,2))

    label = np.ones(100)

    for j in range(0,100):

        if a[j][0] ** 2 + a[j][1] ** 2 <= 1:

            cnt +=1

            label[j] = 2

    Pi_Array[i] = cnt*4/100
```

```
b = np.mean(Pi_Array)

plt.figure(1)

circle1 = plt.Circle((0, 0), 1, color='r',fill=False)

fig, ax = plt.subplots()

ax.add_artist(circle1)

plt.legend([]['In', 'Out'])

plt.scatter(a[:,0],a[:,1], c = 'g')

for j in range(len(a)):

    if label[j] == 2:

        plt.scatter(a[j, 0], a[j, 1], c='r')


plt.figure(2)

circle1 = plt.Circle((0, 0), 1, color='r',fill=False)

fig, ax = plt.subplots()

ax.add_artist(circle1)

plt.hist(Pi_Array)

plt.title("Estimated Pi value = {:.4f}".format(b))

plt.xlabel("Pi Estimate Value")

plt.ylabel("Frequency")

plt.show()


print('\nEstimated Pi value = ',np.mean(Pi_Array))


############################################### PART B
########################################

Pi_array = np.zeros((50,1))

N = list(range(100,10000,100))

Var_array = []

#Var_array = np.zeros((50,1))

for nums in N:

    for k in range(0,50):
```

```
        Pi = 0
        a = np.random.uniform(0,1,size=(nums,2))
        for i in range(0,nums):
            if a[i][0] ** 2 + a[i][1] ** 2 <= 1:
                Pi += 1
        Pi_array[k] = Pi/(nums/4.0)

    Est_var = 0
    mean = np.mean(Pi_array)
    for unit in Pi_array:
        Est_var += (unit - mean)**2
    Var_array.append(Est_var/49.0)

plt.plot(N,Var_array)
plt.title("Sample Variance of the Pi-estimates For Different Values of
Samples")
plt.xlabel("Number of Samples")
plt.ylabel("Sample Variance")
plt.show()



"""
@author: Pranav Gundewar
Project #4: Investigations on Monte Carlo Methods
Q2- Monte Carlo Estimation and Varinace Reduction Stratergies
"""
# Importing required libraries
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

```
from scipy.stats import truncnorm

import statistics

from scipy.stats import multivariate_normal as mvn


#####################################                PART                A
#################################

print('\nDifferent     Monte     Carlo     Estimation     for     function
1/(1+sinh(2*x)*log(x)')

# Define function

def integrand1(x):

    return 1/(1+np.sinh(2*x)*np.log(x))


# N draws

N= 1000

# Define limites for integrals

a1 = 0.8;

b1 = 3;


# Plot the function to get better idea about its estimation

x=np.linspace(a1,b1,1000)

plt.plot(x,integrand1(x))

plt.xlabel('x')

plt.ylabel('f(x)')

plt.title('Function Graph: 1/(1+sinh(2*x)*log(x)')

plt.grid()

plt.show()

###############################              Uniform               Sampling
############################

def estimate1():

    x  =  np.random.uniform(low=a1,  high=b1,  size=N)  #  N  values  uniformly
drawn from a to b

    z =integrand1(x)    # CALCULATE THE f(x)
```

```
    V = b1-a1
    # Monte-Carlo Estimation
    I = V * np.sum(z) / N;
    return I

exactval=0.609553   # f(x) value calculated using Mathematica
I = np.empty(50)
for i in range(50):
    I[i] = estimate1()
var = statistics.variance(I)
print("\nMonte Carlo estimation using Uniform Sampling")
print("Estimation = {:.6f}".format(np.mean(I)), "Variance = {:.6f}".format(var))

############################## Stratified Sampling ##########################

def strfsample1(sv,n):
    # Take more number of samples where function value is varying the most
    x1 = np.random.uniform(low=a1, high=sv, size=n)
    # Calculate the integral value for that section where value is varying the most
    V = sv - a1
    z = integrand1(x1)
    I1 = V * np.sum(z) / n
    # Take rest of the samples from section of function where value is not changing by much
    x2 = np.random.uniform(low=sv, high=b1, size=N-n)
    V = b1 - sv
    z = integrand1(x2)
    I2 = V * np.sum(z) / (N-n)
    # Monte-Carlo Estimation
```

```
    I = I1+ I2

    return I

I = np.empty(50)

for i in range(50):

    I[i] = strfsample1(1.5,800);

var = statistics.variance(I)

print("Monte Carlo estimation using Stratified Sampling")

print("Estimation      =      {:.6f}".format(np.mean(I)),"Variance      =
{:.6f}".format(var))



################################            Importance            Sampling
##########################



def impsample1():

    x = truncnorm.rvs(a1, b1, loc=0, size=N)

    p = truncnorm.pdf(x, a1, b1, loc=0)

    z = np.empty(N)

    for i in range(N):

        z[i] = integrand1(x[i])

        z[i] /= p[i]

    return np.mean(z)


I = np.empty(50)

for i in range(50):

    I[i] = impsample1()

var = statistics.variance(I)

print("Monte Carlo estimation using Importance Sampling")

print("Estimation      =      {:.6f}".format(np.mean(I)),"Variance      =
{:.6f}".format(var))
```

```
#####################################                PART                B
#################################

#Define function

def integrand2(x,y):

    return np.exp(-x**4-y**4)


#Define limites for integrals

a2 = -1*np.pi

b2 = np.pi

# use N draws

N= 1000

exactval=3.28626

# Plot the function to get better idea about its estimation

x=np.linspace(a2,b2,1000)

y=np.linspace(a2,b2,1000)

plt.plot(x,integrand2(x,y))

plt.xlabel('x')

plt.ylabel('f(x)')

plt.title('Function Graph: exp(-x^4-y^4)')

plt.grid()

plt.show()


print('\nDifferent Monte Carlo Estimation for function e^(-x^4-y^4)\n')


################################              Uniform              Sampling
############################


def estimate2():

    x = np.random.uniform(low=a2, high=b2, size=N) # N values uniformly
drawn from a to b

    y = np.random.uniform(low=a2, high=b2, size=N) # N values uniformly
drawn from a to b
```

```
    z =integrand2(x,y)    # CALCULATE THE f(x)

    V = b2-a2

    I = V * V * np.sum(z)/ N;

    return I


I = np.empty(50)

for i in range(50):

    I[i] = estimate2()

var = statistics.variance(I)

print("Monte Carlo estimation using Uniform Sampling")

print("Estimation      =      {:.6f}".format(np.mean(I)),"Variance      =
{:.6f}".format(var))


################################            Stratified            Sampling
##########################

def strfsample2():

    x1 = np.random.uniform(low=-1.1, high=1.1, size=1000)

    y1 = np.random.uniform(low=-1.1, high=1.1, size=1000)

    V = 2.2

    z = integrand2(x1,y1)

    I1 = V * V * np.sum(z) / 1000

    return I1


I = np.empty(50)

for i in range(50):

    I[i] = estimate2()

var = statistics.variance(I)

print("Monte Carlo estimation using Stratified Sampling")

print("Estimation      =      {:.6f}".format(np.mean(I)),"Variance      =
{:.6f}".format(var))

#
```

```
################################            Importance            Sampling
########################

#

def impsample2():

    count = 0

    X = np.empty((N,2))

    while count < N:

        x = mvn.rvs(mean = [0, 0] ,cov = [[1, 0], [0, 1]])

        if x[0] > a2 and x[0] < b2 and x[1] > a2 and x[1] < b2:

            X[count,0] = x[0]

            X[count,1] = x[1]

        count +=1

    p = mvn.pdf(X,mean = [0, 0] ,cov = [[1, 0], [0, 1]])

    I = integrand2(X[:,0],X[:,1])

    q = np.divide(I,p)

    return np.mean(q)


I = np.empty(50)

for i in range(50):

    I[i] = impsample2()

var = statistics.variance(I)

print("Monte Carlo estimation using Importance Sampling")

print("Estimation      =      {:.6f}".format(np.mean(I)),"Variance      =
{:.6f}".format(var))


####################################                PART                C
################################

print('\nDifferent  Monte  Carlo  Estimation  for  function  20+x^2+y^2-
10(cos(2*pi*x)+cos(2*pi*y))\n')

# Define function

def integrand3(x,y):

    return 20 + x**2 + y**2 - 10*(np.cos(2*np.pi*x)+np.cos(2*np.pi*y))
```

```
#Define limites for integrals

a3 = -5

b3 = 5

# use N draws

N= 1000

exactval=3666.67

def estimate3():

    x = np.random.uniform(low=a3, high=b3, size=N) # N values uniformly
drawn from a to b

    y = np.random.uniform(low=a3, high=b3, size=N) # N values uniformly
drawn from a to b

    z =integrand3(x,y)   # CALCULATE THE f(x)

    V = b3-a3

    I = V * V * np.sum(z)/ N;

    return I


I = np.empty(50)

for i in range(50):

    I[i] = estimate3()

var = np.var(I)

print("Monte Carlo estimation using Uniform Sampling")

print("Estimation      =      {:.6f}".format(np.mean(I)),"Variance      =
{:.6f}".format(var/10000))
```