EE 511 SIMULATION METHODS FOR STOCHASTIC SYSTEMS

PROJECT – 4

ABINAYA MANIMARAN

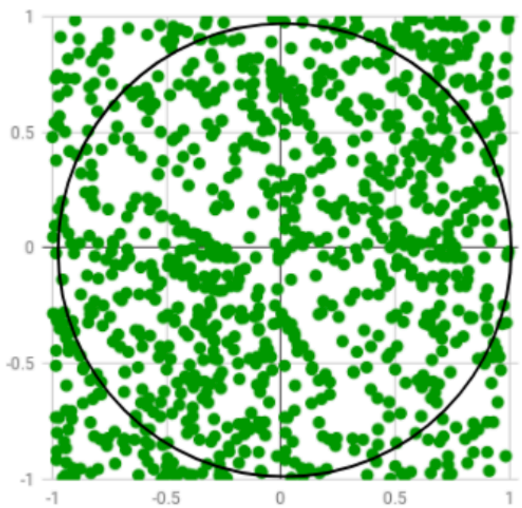
SPRING 2018

04/09/2018

QUESTION 1

Procedure to Estimate value of pi using Monte Carlo:

- The idea is to simulate random (x,y) points in a 2D plane with domain as a square of side length 1 unit
- Imagine the circle inside the square domain with the same diameter and inscribed into the square
- Calculate the number of points lie inside the circle and total number of generated sample points
- The Image below shows how the generated samples can varyingly lie inside the circle and not inside the circle.



- The area of the square is calculated as the formula below.

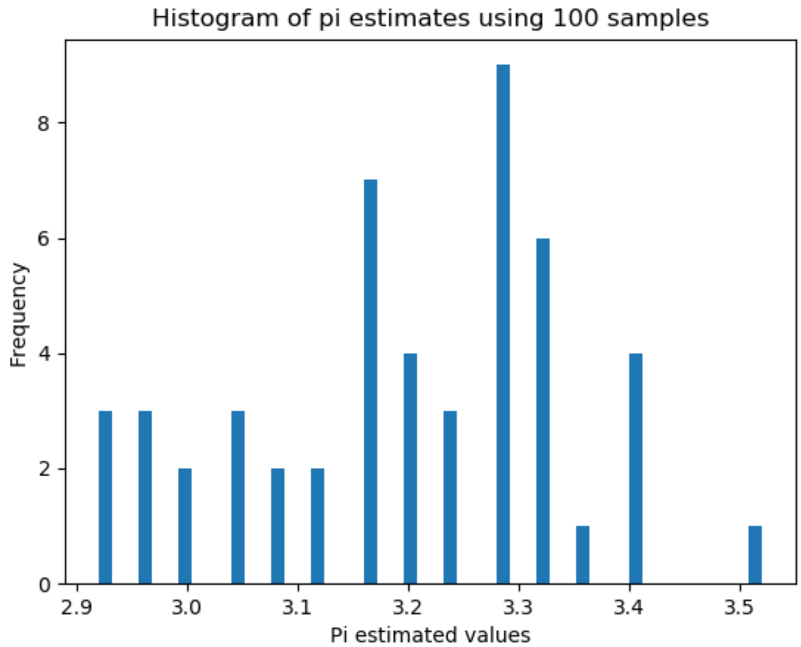
$$\frac{\text{area of the circle}}{\text{area of the square}} = \frac{\text{no. of points generated inside the circle}}{\text{total no. of points generated or no. of points generated inside the square}}$$

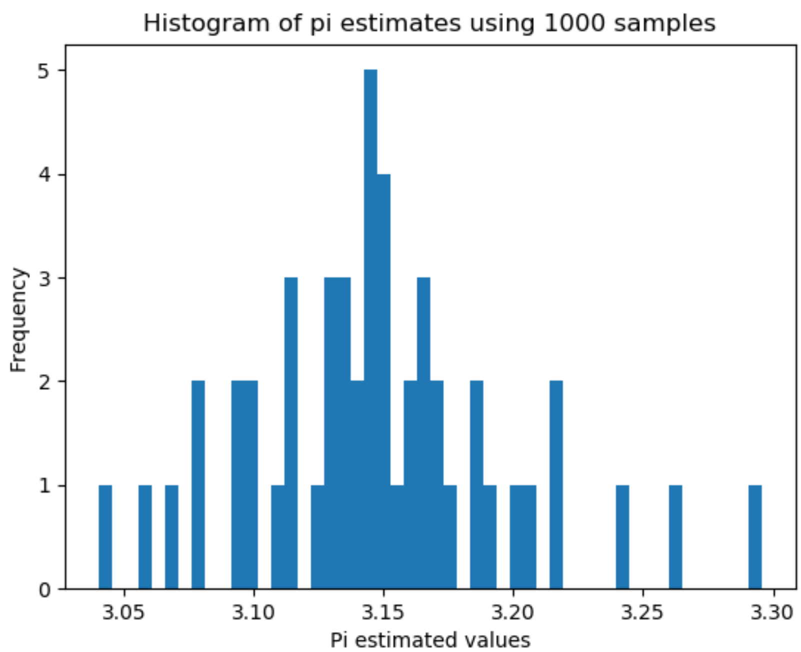
that is,

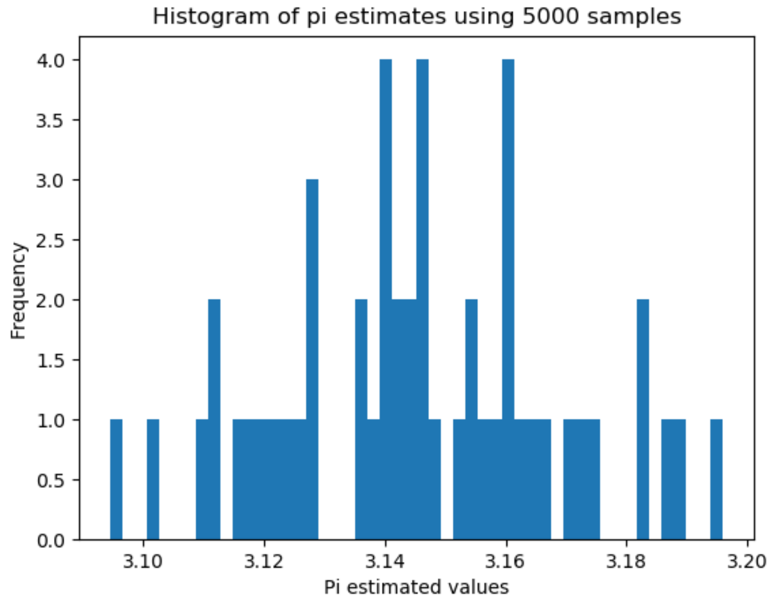
$$\pi = 4 * \frac{\text{no. of points generated inside the circle}}{\text{no. of points generated inside the square}}$$

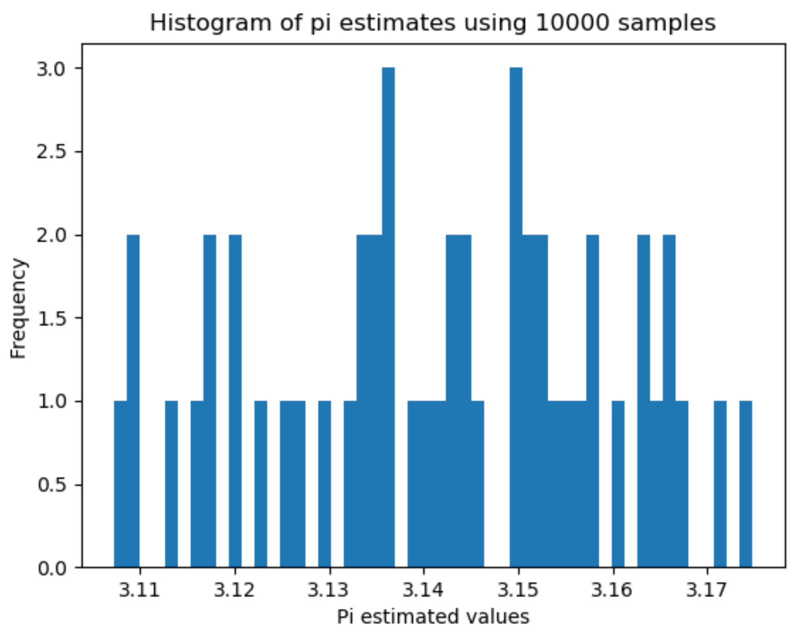
- We also know that the area of the square is 1 unit sq. and that of the circle is $\pi * (\frac{1}{2})^2 = \frac{\pi}{4}$

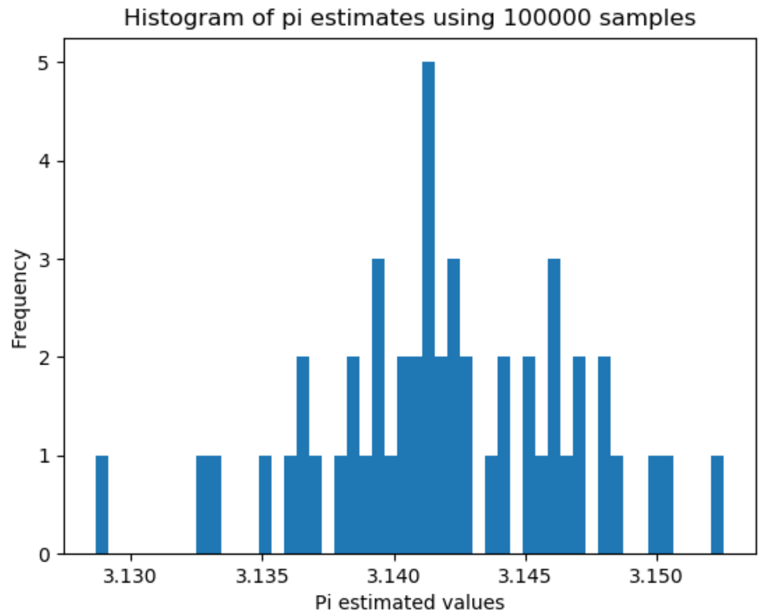
- To check if the generated points lie within the circle or not, we can see $x^2 + y^2 \leqslant 1$

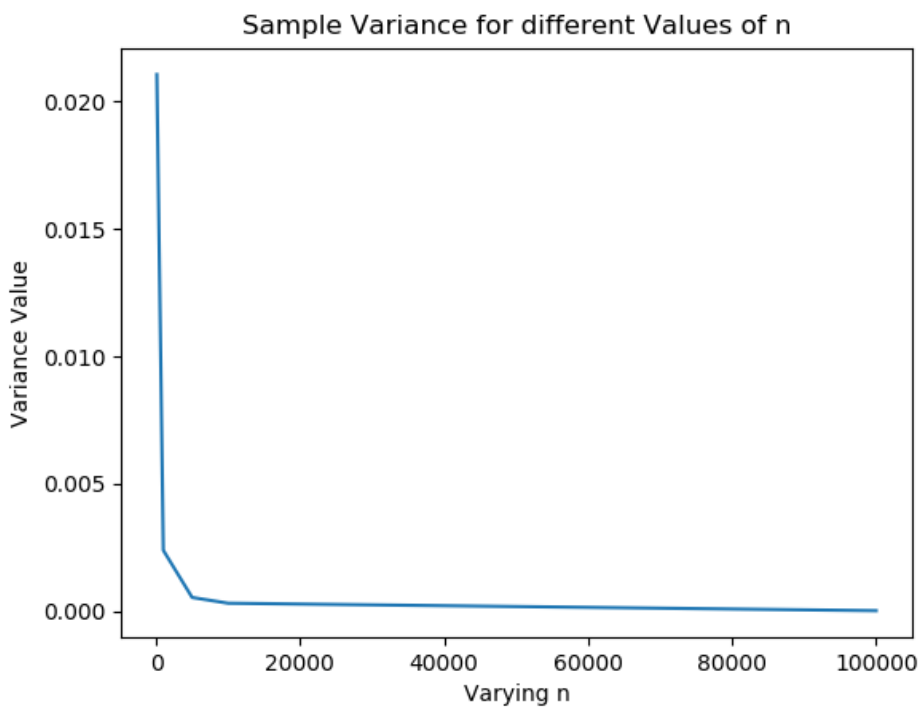

Histogram of pi estimates using 100 samples
Frequency
Pi estimated values
0
2
4
6
8
2.9
3.0
3.1
3.2
3.3
3.4
3.5


Histogram of pi estimates using 1000 samples
Frequency
Pi estimated values
0
1
2
3
4
5
3.05
3.10
3.15
3.20
3.25
3.30


Histogram of pi estimates using 5000 samples
Frequency
Pi estimated values
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3.10
3.12
3.14
3.16
3.18
3.20


Histogram of pi estimates using 10000 samples
Frequency
Pi estimated values
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3.11
3.12
3.13
3.14
3.15
3.16
3.17


Histogram of pi estimates using 100000 samples
Frequency
Pi estimated values
0
1
2
3
4
5
3.130
3.135
3.140
3.145
3.150


Sample Variance for different Values of n
Variance Value
Varying n
0.000
0.005
0.010
0.015
0.020
0
20000
40000
60000
80000
100000

IPython console

Console 1/A

```
In [213]: runfile('/Users/abinaya/USC/Studies/Stochastic-Systems/hw-4/hw_4_prob_1.py', wdir='/Users/abinaya/USC/Studies/Stochastic-Systems/hw-4')
----------- 100
Mean pi Estimate:  3.1943999999999995
Sample Variance:  0.021056639999999988
----------- 1000
Mean pi Estimate:  3.147760000000001
Sample Variance:  0.0023913023999999994
----------- 5000
Mean pi Estimate:  3.1455679999999995
Sample Variance:  0.0005407989759999987
----------- 10000
Mean pi Estimate:  3.141744
Sample Variance:  0.0003156272639999998
----------- 100000
Mean pi Estimate:  3.1419376
Sample Variance:  2.2634122240000142e-05

In [214]:
```

- The above experiment shows that the value of pi estimate is very close to the Actual Pi value when the number of samples are increased.
- The value of pi is estimated to 3.1419376 when n=100000. This is the closest estimate we were able to get by increasing the n value.
- The variance of estimate also decreases drastically as we increase n. We can see that the lowest variance value is for again n= 100000, variance = 0.00002263

Thus, we showed how Monte Carlo Estimate was used in estimation of pi application. We also saw that as number of samples increases, the value estimate becomes closer to the actual value. The variance of estimate also decreases.

QUESTION 2

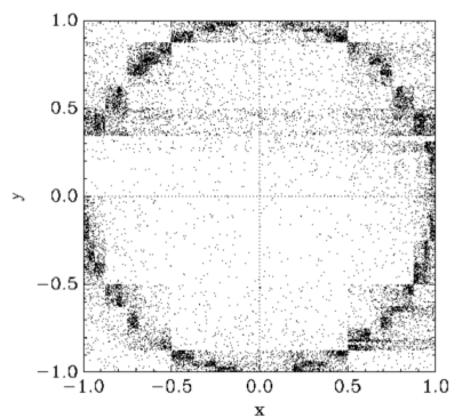
MONTE CARLO INTEGRATION AND VARIANCE REDUCTION STRATEGIES

Procedure to do simple Monte Carlo Integral Estimation:

- Generate Samples from Uniform Distribution between given integral boundary values
- Find the Function value for the generated samples
- Get the mean of the function values (i.e.) Integral value
- Iterate it max_iterations number of times
- Every time append the mean integral value to a list
- At the end get the mean of mean integral values
- Also calculate the variance of mean integral values to find the quality of our integration

Procedure to do simple Monte Carlo Integral Estimation using Stratification:

- Plot the given function between the given boundary values
- Based on the function divide boundary values into various bins
- Also decide allocation of number of samples to each bin based on the function plot. For example, in the given plot below, the number of samples are highly concentrated when the curve has direction changes. Very low number of samples are assigned to the places where information is less for integration



- Generate Samples from Uniform Distribution between given integral boundary values based on our decision.
- Find the Function value for the generated samples between different divided integral locations
- Get the mean of the function values (i.e.) Integral value at different bins
- Get the summation of them to get final integral value
- Iterate it max_iterations number of times
- Every time append the final integral value to a list
- At the end get the mean of final integral values
- Also calculate the variance of final integral values to find the quality of our integration using stratification

Procedure to do simple Monte Carlo Integral Estimation using Importance Sampling:

- Plot the given function between the given boundary values
- Find a pdf, that resembles similar to the plot function we have. This is called as Importance pdf

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

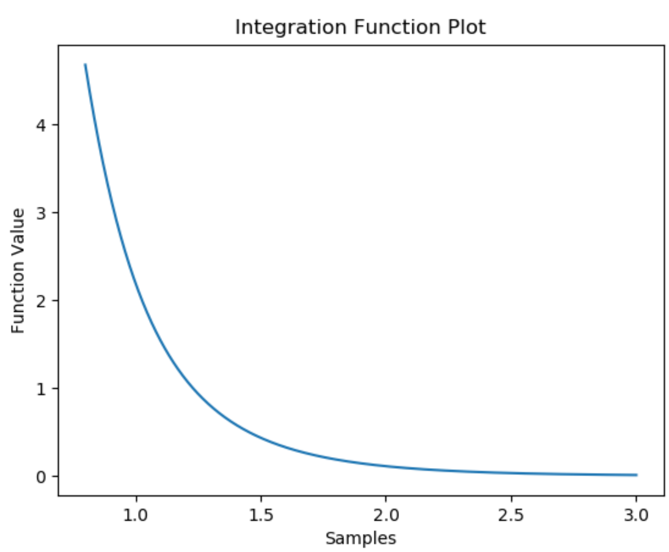
- In the above equation, h(x) is the Integral function that we have to estimate, p(x) is the function that we assume to be Uniform Distribution (from theory behind Monte Carlo) and g(x) is our approximated Importance pdf
- To proceed, generate samples from Importance pdf g(x)
- Based on the generated samples, estimate h(x), p(x) and g(x)
- Calculate the above expression h(x)*p(x) / g(x) within the integral
- Get the average of those values to get Mean Integral Value
- Iterate it max_iterations number of times
- Every time append the mean integral value to a list
- At the end get the mean of mean integral values
- Also calculate the variance of mean integral values to find the quality of our integration

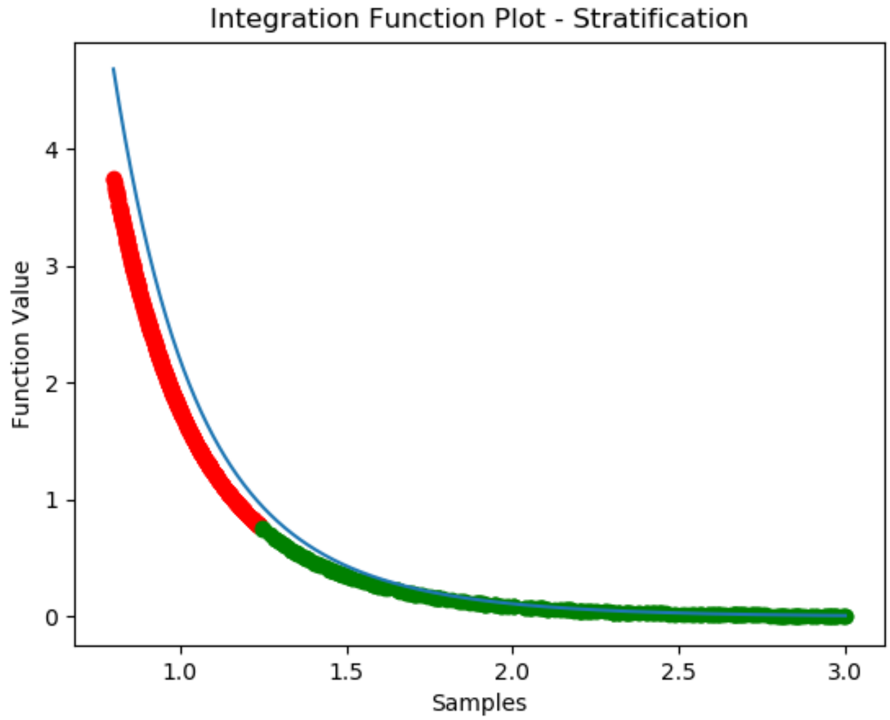
From the above three methods of Monte Carlo Integral Estimation, Stratification and Importance Sampling are Variance reduction methods. The variance of Importance Sampling should be less than any other methods.

PART 1:

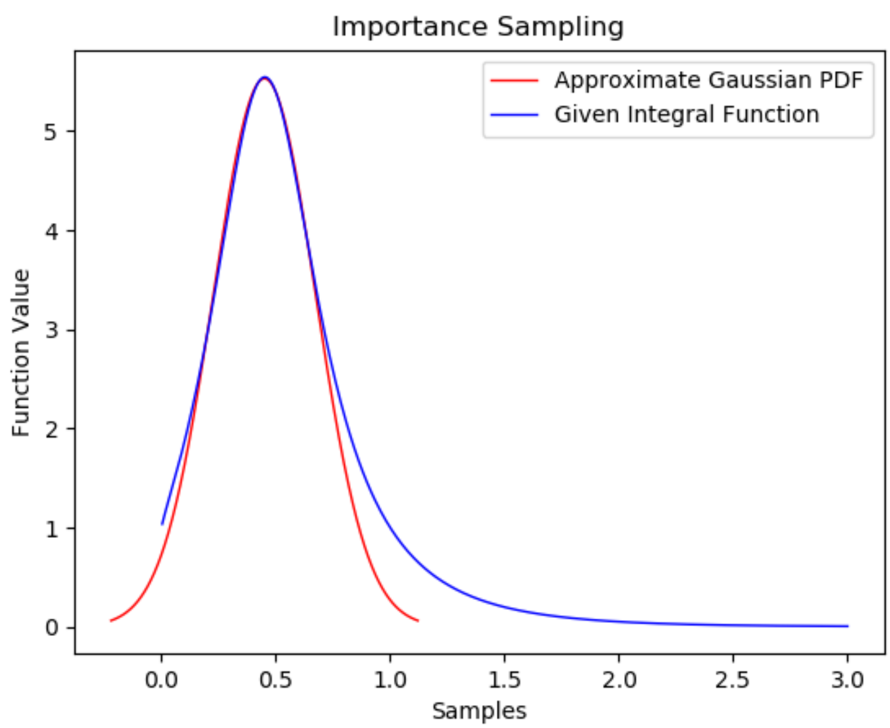
Given Definite Integral:

$[1+\sinh(2x)\ln(x)]^{-1}$ for x in [0.8, 3]

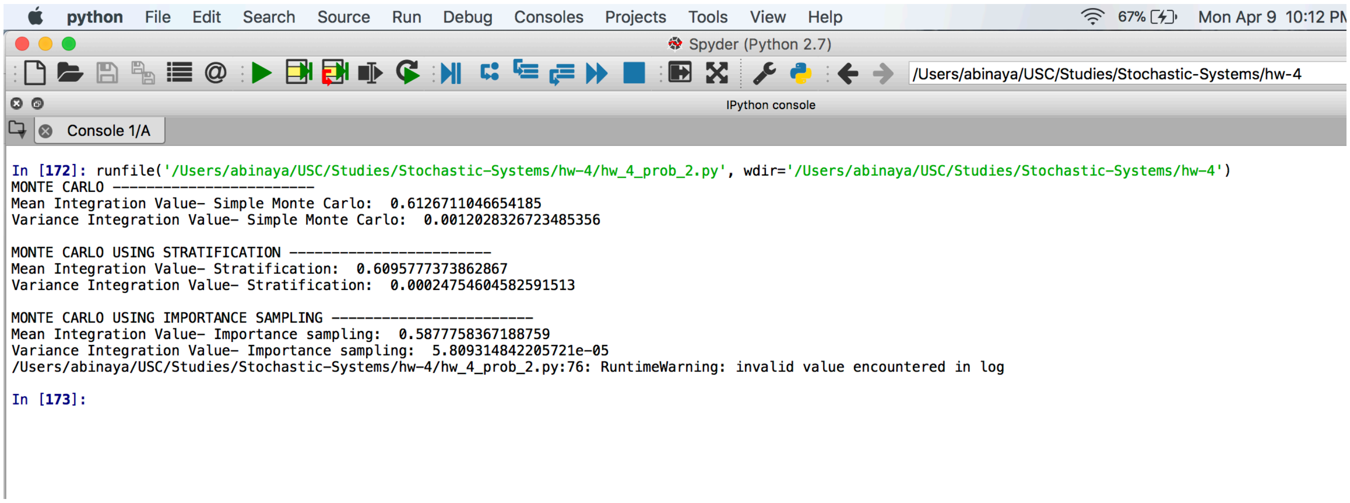




- For Stratification, 30% of samples were generated between (0.8, 1.24) and 70% samples were generated between (1.24, 3)



- For Importance Sampling, the approximation PDF considered to be a Gaussian PDF with Mean = 0.5 and Standard Deviation = 0.7



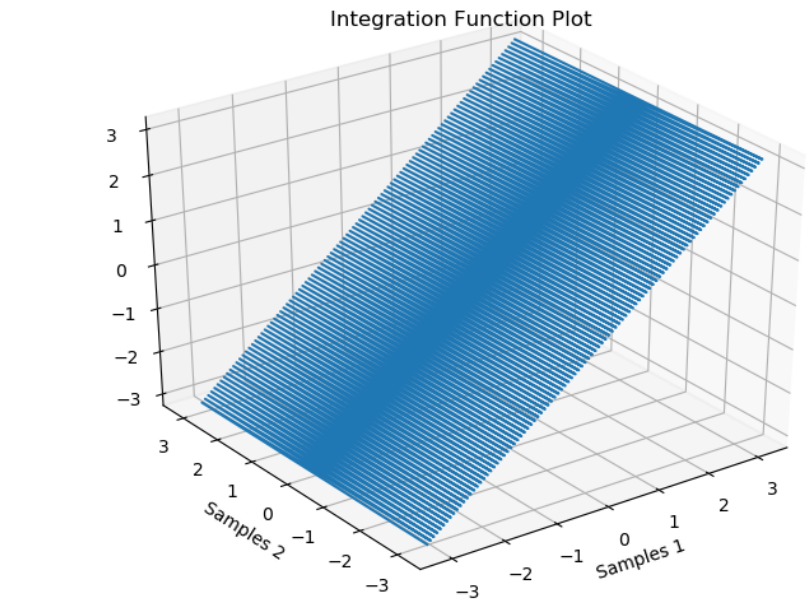
	Integral Estimate	Variance
Monte Carlo - Simple	0.61267	0.0012
Using Stratification	0.60957	0.00024
Using Importance Sampling Gaussian – Mean = 0.5 Standard Dev = 0.7	0.58775	0.000058

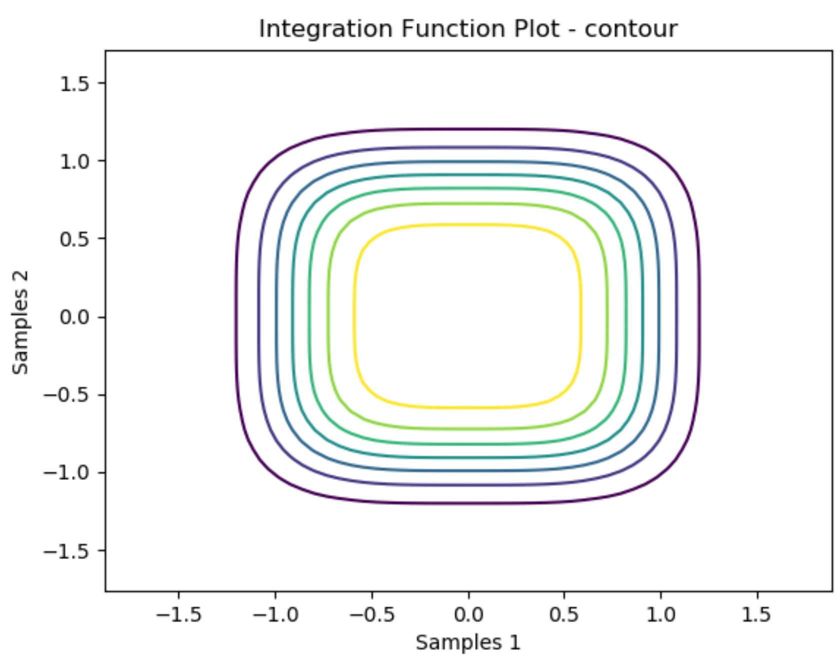
- We can see from the above results; the variance has drastically reduced for Importance Sampling. While there is a compromise on the Integral Value
- For Stratification, we are getting a good Integral Value. The variance has also reduced when compared to the simple Monte Carlo Integration. But it is not as less as Importance Sampling method.
- Depending upon the application we can choose the best method.

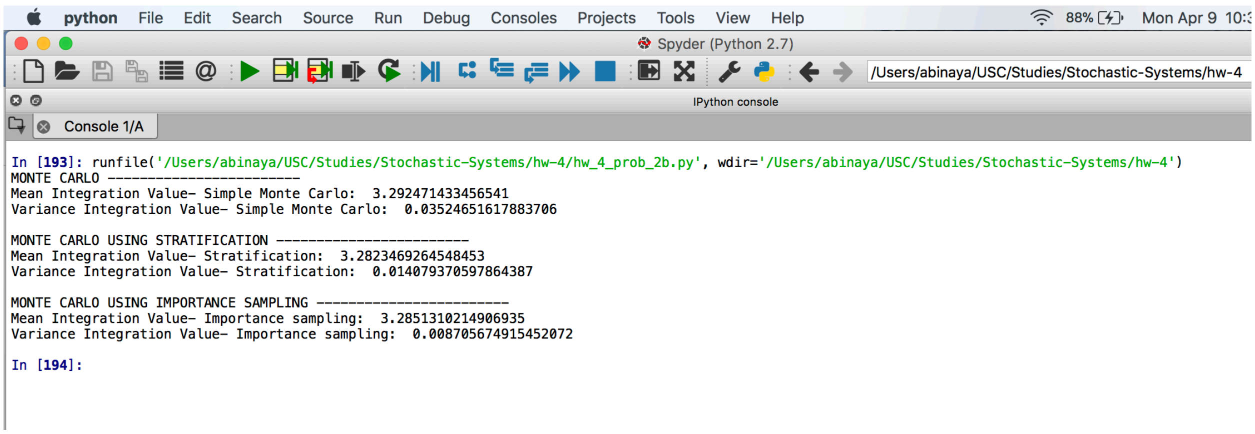
PART 2:

Given Definite Integral:

$$\text{Exp}[-x^4 - y^4] \qquad \textbf{for (x, y) in [-pi, pi]}$$







```
In [193]: runfile('/Users/abinaya/USC/Studies/Stochastic-Systems/hw-4/hw_4_prob_2b.py', wdir='/Users/abinaya/USC/Studies/Stochastic-Systems/hw-4')
MONTE CARLO ------------------------
Mean Integration Value- Simple Monte Carlo:  3.292471433456541
Variance Integration Value- Simple Monte Carlo:  0.03524651617883706

MONTE CARLO USING STRATIFICATION ------------------------
Mean Integration Value- Stratification:  3.2823469264548453
Variance Integration Value- Stratification:  0.014079370597864387

MONTE CARLO USING IMPORTANCE SAMPLING ------------------------
Mean Integration Value- Importance sampling:  3.2851310214906935
Variance Integration Value- Importance sampling:  0.008705674915452072

In [194]:
```

- For Stratification, 80% of samples were generated between (-1.73, 1.73) values of x and y and 20% samples were generated outside
- For Importance Sampling, the approximation PDF considered to be a Gaussian PDF with Mean = 0 and Standard Deviation = 0.5
- We can see from the above results; the variance has drastically reduced for Importance Sampling. While there is a compromise on the Integral Value
- For Stratification, we are getting a good Integral Value. The variance has also reduced when compared to the simple Monte Carlo Integration. But it is not as less as Importance Sampling method.
- Depending upon the application we can choose the best method.

	Integral Estimate	Variance
Monte Carlo - Simple	3.2924	0.03
Using Stratification	3.2823	0.01
Using Importance Sampling Gaussian – Mean = 0.5 Standard Dev = 0.7	3.2851	0.008

Pros and Cons of both the methods:

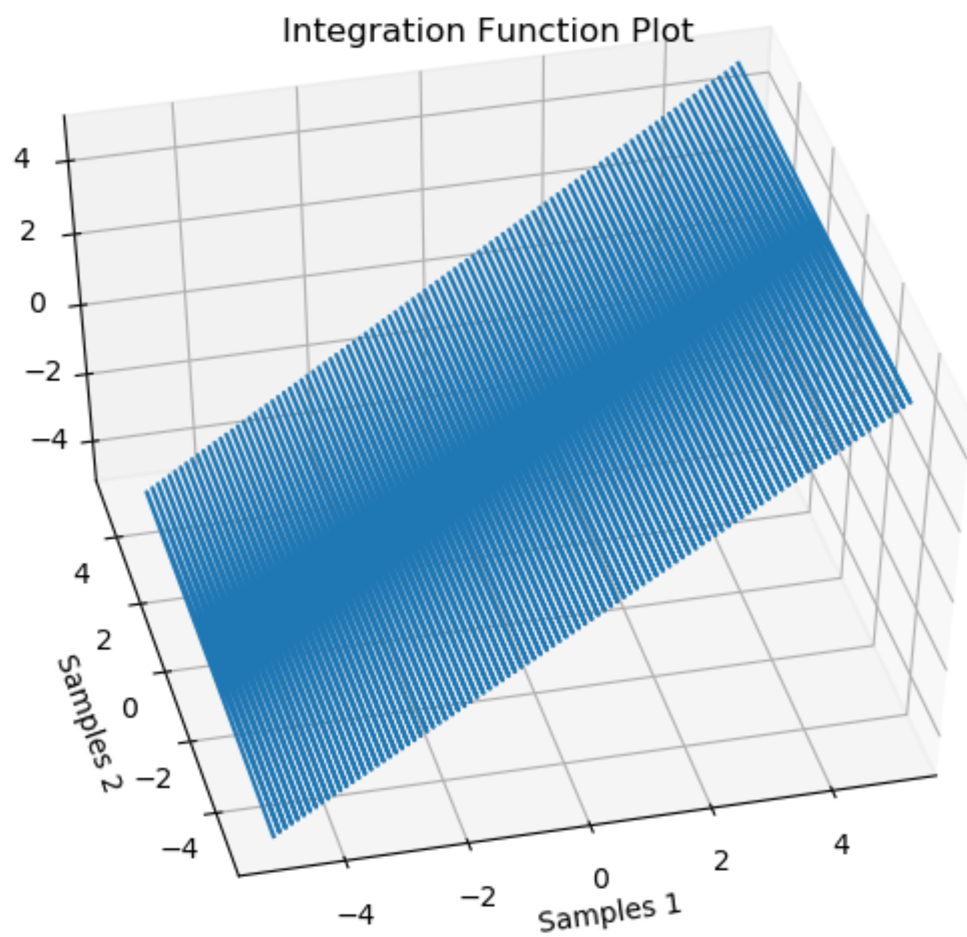
	Pros	Cons
Stratification	Reduces Selection Bias, Can be used when Accurate results needs to be got, because it ensures subgroups within the population receives proper attention	Several Conditions need to be met for it to be used properly, Should Accurately sort each member of the population
Importance Sampling	Can bring enormous gains in reducing the variance of estimate when compared to any other Monte Carlo extension methods	The very low variance can itself be a drawback since it reduces the accuracy of estimation

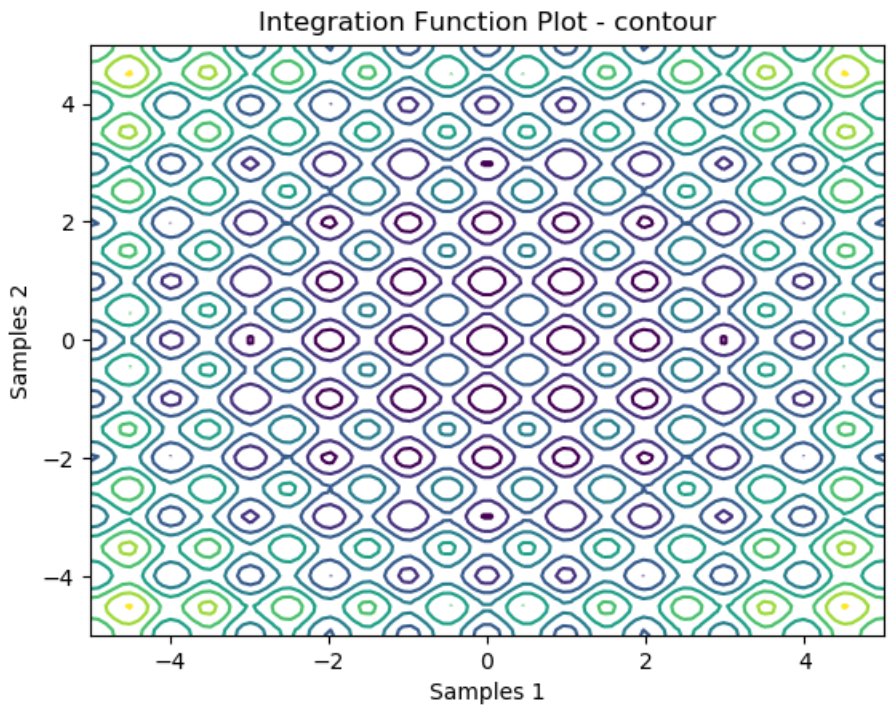
PART 3:

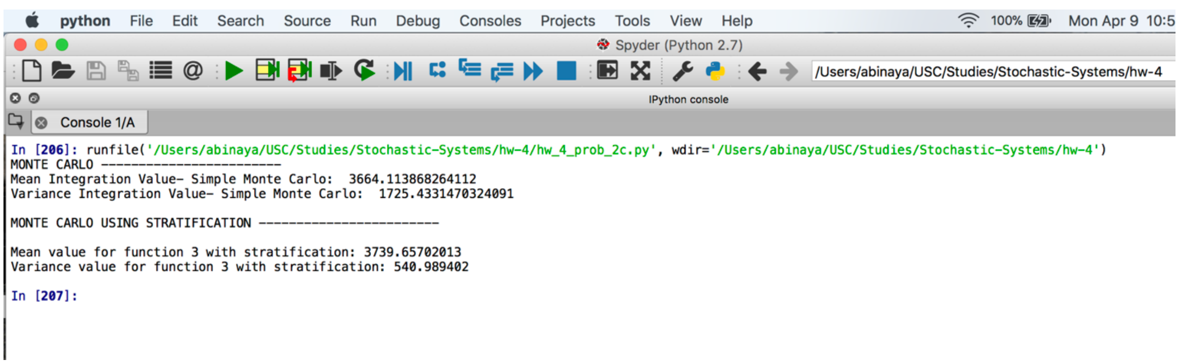
Given Definite Integral:

$$20+x^2+y^2-10(cos[2\pi\times x]+cos[2\pi\times y])$$ **for (x, y) in [-5,5]**

- I have chosen Stratification for part 3 since it is considered to be the best method.
- Though the variance is not as less as Importance Sampling, the Integral Value for Stratification method is the best one close to the actual Integral Value.
- From the below results also, the variance has reduced using Stratification. Also, the Integral Estimate is better and close to the Actual Estimate.







	Integral Estimate	Variance
Monte Carlo - Simple	3664.113	1725.433
Using Stratification	3739.65	540.98

- I chose Stratification because I did not want to compromise on the accuracy of Integral Estimation. Using Stratification, the variance reduced than Simple Monte Carlo estimation. The accuracy of the estimate has also increased. Choosing the method depends upon the application as mentioned earlier.

CODES

PROBLEM 1:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Sun Apr  8 00:28:47 2018

@author: abinaya
"""

import numpy as np
import matplotlib.pyplot as plt
plt.close('all')

n_list = [100, 1000, 5000, 10000, 100000]
dim = 2
k = 50

sample_variance_list = []

for n in n_list:
    print "-----------",n
    pi_estimate_list = []

    for iteration in range(0,k):
        uniform_trials= np.random.uniform(size = [n,dim])
        area = np.sum(uniform_trials**2, axis=1)
        no_points_inside = sum(area<=1)
        pi_estimate = 4 * (float(no_points_inside)/float(n))
        pi_estimate_list.append(pi_estimate)

    plt.figure()
    plt.hist(pi_estimate_list, bins=50)
    plt.title('Histogram of pi estimates using '+str(n)+' samples')
    plt.xlabel('Pi estimated values')
    plt.ylabel('Frequency')

    sample_variance_list.append(np.var(pi_estimate_list))
    print "Mean pi Estimate: ", np.mean(pi_estimate_list)
    print "Sample Variance: ", np.var(pi_estimate_list)

plt.figure()
```

```
plt.plot(n_list, sample_variance_list)
plt.title('Sample Variance for different Values of n')
plt.xlabel('Varying n')
plt.ylabel('Variance Value')
```

PROBLEM 2 – A:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Sun Apr  8 21:49:30 2018

@author: abinaya
"""

import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.stats import norm, uniform

def function_f(x):
    z = (3 - 0.8) * (1/ (1 + (np.sinh(2*x) * np.log(x))))
    return z

def function_p(x):
    z = uniform.pdf(x, loc=0.8, scale=3)
    return z

def function_g(x, mu, std):
    z = norm.pdf(x, loc=mu, scale=std)
    return z

print "MONTE CARLO ------------------------"

max_iterations = 50
monte_carlo_estimates = []
for i in range(0,max_iterations):
    test_x1 = np.random.uniform(0.8,3,size=1000)
    #test_z1 = map(lambda t: function_f(t),test_x1)
    test_z1 = (3 - 0.8) * (1/ (1 + (np.sinh(2*test_x1) * np.log(test_x1))))
    monte_carlo_estimates.append(np.mean(test_z1))

print "Mean Integration Value- Simple Monte Carlo: ", np.mean(monte_carlo_estimates)
```

```
print "Variance Integration Value- Simple Monte Carlo: ", np.var(monte_carlo_estimates)

'''
test_x1 = np.linspace(0.8,3,1000)
test_z1 = (3 - 0.8) * (1/ (1 + (np.sinh(2*test_x1) * np.log(test_x1))))
plt.figure()
plt.plot(test_x1,test_z1)
plt.xlabel('Samples')
plt.ylabel('Function Value')
plt.title('Integration Function Plot - Stratification')
'''

print "\nMONTE CARLO USING STRATIFICATION ------------------------"

n=1000
n1=700
n2=300
value1 = []
value2 = []

for i in range (0,50):
    X1_1 = np.random.uniform(0.8, 1.24, n1)
    Fnc1_1 = (1.24-0.8) * pow((1 + (np.sinh(2*X1_1)*np.log(X1_1))),-1)
    value1_1 = (np.sum(Fnc1_1))/ n1

    X1_2 = np.random.uniform(1.24, 3, n2)
    Fnc1_2 = (3-1.24)*pow((1 + (np.sinh(2*X1_2)*np.log(X1_2))),-1)
    value1_2 = (np.sum(Fnc1_2))/ n2

    value2.append(value1_1 + value1_2)
variance_stratified_Fnc1 = np.var(value2)
Mean_stratified_Fnc1 = np.mean(value2)
print  "Mean Integration Value- Stratification: ", Mean_stratified_Fnc1
print  "Variance Integration Value- Stratification: ", variance_stratified_Fnc1

'''
plt.scatter(X1_1,Fnc1_1, color='r')
plt.scatter(X1_2,Fnc1_2, color='g')
'''

print "\nMONTE CARLO USING IMPORTANCE SAMPLING ------------------------"

importance_sampling_estimates = []
mu = 0.5
```

```
std = 0.7

for i in range(0,max_iterations):
    test_x3 = np.random.normal(loc=mu, scale=std, size=1000)

    fx3 =  (3 - 0.8) * (1/ (1 + (np.sinh(2*test_x3) * np.log(test_x3))))
    px3 = uniform.pdf(test_x3, loc=0.8, scale=3)
    gx3 = norm.pdf(test_x3, loc=mu, scale=std)

    z3 = (np.array(fx3)) * (np.array(px3) / np.array(gx3))
    z3 = z3[~np.isnan(z3)]
    importance_sampling_estimates.append(np.mean(z3))

print  "Mean Integration Value- Importance sampling: ",
np.mean(importance_sampling_estimates)
print  "Variance Integration Value- Importance sampling: ",
np.var(importance_sampling_estimates)
```

PROBLEM 2 – B:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Apr  9 20:27:26 2018

@author: abinaya
"""

import numpy as np
import matplotlib.pyplot as plt
import math
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import norm, uniform
from scipy.stats import multivariate_normal

def function_f(x,y):
    z = math.pow((math.pi*2),2) * (np.exp((-1 * math.pow(x,4)) + (-1 * math.pow(y,4))))
    return z

def function_p(x):
    z1 = uniform.pdf(x[0], loc=-math.pi, scale=math.pi)
    z2 = uniform.pdf(x[1], loc=-math.pi, scale=math.pi)
    return z1*z2
```

```
def function_g(x, mu, std):
    z = multivariate_normal.pdf(x, mean=mu, cov=std)
    return z


test_x2 = np.linspace(-math.pi, math.pi,100)
test_y2 = np.linspace(-math.pi, math.pi,100)
xx, yy = np.meshgrid(test_x2, test_y2)
test_xy = np.hstack( (xx.reshape(xx.shape[0]*xx.shape[1], 1, order='F'),
yy.reshape(yy.shape[0]*yy.shape[1], 1, order='F')))
test_z2 = map(lambda t:function_f(t[0],t[1]), test_xy)

fig = plt.figure()
ax = Axes3D(fig)
num_pts = np.shape(test_xy)[0]
xs_plot = np.reshape(test_xy[:,0], [num_pts])
ys_plot = np.reshape(test_xy[:,1], [num_pts])
zs_plot = np.reshape(test_z2, [num_pts])
ax.plot(xs=xs_plot, ys=ys_plot, zs=xs_plot)
plt.xlabel('Samples 1')
plt.ylabel('Samples 2')
plt.zlabel('Function Value')
plt.title('Integration Function Plot')

zz =  zs_plot.reshape(100,100)
plt.figure()
plt.contour(xx, yy, zz)
plt.xlabel('Samples 1')
plt.ylabel('Samples 2')
plt.title('Integration Function Plot - contour')


print "MONTE CARLO ------------------------"

max_iterations = 50

monte_carlo_estimates = []
for i in range(0,max_iterations):
    test_x1 = np.random.uniform(-math.pi, math.pi,1000)
    test_y1 = np.random.uniform(-math.pi, math.pi,1000)
    xx, yy = np.meshgrid(test_x1, test_y1)
    test_xy = np.hstack((xx.reshape(xx.shape[0]*xx.shape[1], 1, order='F'),
yy.reshape(yy.shape[0]*yy.shape[1], 1, order='F')))
```

```
    test_z1 = math.pow((math.pi*2),2) * (np.exp((-1 * pow(test_xy[:,0],4)) + (-1 *
pow(test_xy[:,1],4))))
    monte_carlo_estimates.append(np.mean(test_z1) )

print "Mean Integration Value- Simple Monte Carlo: ", np.mean(monte_carlo_estimates)
print "Variance Integration Value- Simple Monte Carlo: ", np.var(monte_carlo_estimates)

print "\nMONTE CARLO USING STRATIFICATION ------------------------"
value3 = []
value4 = []
n = 1000

for i in range (0,50):
    X2_1 = np.random.uniform(-1.73, 1.73, n)
    Y2_1 = np.random.uniform(-1.73, 1.73, n)
    Fnc2_1 = pow(math.e, (-pow(X2_1,4)-pow(Y2_1,4)))
    value3_1 = ((1.73+1.73)**2) * np.sum(Fnc2_1)/ n
    value4.append(value3_1)
variance_stratified_Fnc2 = np.var(value4)
Mean_stratified_Fnc2 = np.mean(value4)
print  "Mean Integration Value- Stratification: ", Mean_stratified_Fnc2
print  "Variance Integration Value- Stratification: ", variance_stratified_Fnc2

print "\nMONTE CARLO USING IMPORTANCE SAMPLING ------------------------"

importance_sampling_estimates = []
mu = 0
std = 0.5

mu_mat = np.array([mu, mu])

for i in range(0,max_iterations):
    #print "--- iteration ", i
    test_x3 = np.random.normal(loc=mu,size=1000)
    test_y3 = np.random.normal(loc=mu,size=1000)

    xx, yy = np.meshgrid(test_x3, test_y3)
    test_xy = np.hstack((xx.reshape(xx.shape[0]*xx.shape[1], 1, order='F'),
yy.reshape(yy.shape[0]*yy.shape[1], 1, order='F')))

    fx3 = math.pow((math.pi*2),2) * (np.exp((-1 * pow(test_xy[:,0],4)) + (-1 *
pow(test_xy[:,1],4))))
    z1 = uniform.pdf(test_xy[:,0], loc=-math.pi, scale=math.pi+math.pi)
    z2 = uniform.pdf(test_xy[:,1], loc=-math.pi, scale=math.pi+math.pi)
```

```
    px3 = z1*z2
    gx3 = multivariate_normal.pdf(test_xy, mean=mu_mat)


    z3 = (np.array(fx3)) * (np.array(px3) / np.array(gx3))
    z3 = z3[~np.isnan(z3)]
    importance_sampling_estimates.append(np.mean(z3))

print  "Mean Integration Value- Importance sampling: ",
np.mean(importance_sampling_estimates)
print  "Variance Integration Value- Importance sampling: ",
np.var(importance_sampling_estimates)
```

PROBLEM 2 – :C

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Apr  9 22:52:13 2018

@author: abinaya
"""
import numpy as np
import matplotlib.pyplot as plt
import math

print "MONTE CARLO ------------------------"

last_fnc=[]
stratified_lastfunction=[]
## Function 3
for i in range (0,50):
    x3= np.random.uniform(-5,5,(1000,2))
    function3=20 + pow(x3[:,0],2) + pow(x3[:,1],2) - 10 * (np.cos(2 * math.pi * x3[:,0] ) + np.cos(2
* math.pi * x3[:,1] ))
    last_fnc.append((np.sum(function3))/1000 * (5 -(-5)) * (5 -(-5)))

var_lastFcn=np.var(last_fnc)
mean_lastFcn=np.mean(last_fnc)

print "Mean Integration Value- Simple Monte Carlo: ", mean_lastFcn
print "Variance Integration Value- Simple Monte Carlo: ", var_lastFcn
```

```
print "\nMONTE CARLO USING STRATIFICATION ------------------------"

n1=50
n2=50
n3=800
n4=50
n5=50
for i in range (0,50):

   x3_1= np.random.uniform(-5,5,n1)
   y3_1= np.random.uniform(-5,-2.5,n1)
   last_fnc_1=20 + pow(x3_1,2) + pow(y3_1,2) - 10 * (np.cos(2 * math.pi * x3_1 ) + np.cos(2 *
math.pi * y3_1 ))
   fcn3_1=(np.sum(last_fnc_1))/n1 * 10 * 2.5

   x3_2= np.random.uniform(-5,5,n2)
   y3_2= np.random.uniform(-2.5,2.5,n2)
   last_fnc_2=20 + pow(x3_2,2) + pow(y3_2,2) - 10 * (np.cos(2 * math.pi * x3_2 ) + np.cos(2 *
math.pi * y3_2 ))
   fcn3_2=(np.sum(last_fnc_2))/n2 * 10 * 5

   x3_3= np.random.uniform(-5,5,n3)
   y3_3= np.random.uniform(2.5,5,n3)
   last_fnc_3=20 + pow(x3_3,2) + pow(y3_3,2) - 10 * (np.cos(2 * math.pi * x3_3 ) + np.cos(2 *
math.pi * y3_3 ))
   fcn3_3=(np.sum(last_fnc_3))/n3 * 10 * 2.5

   y3_4= np.random.uniform(-5,5,n4)
   x3_4= np.random.uniform(-5,-2.5,n4)
   last_fnc_4=20 + pow(x3_4,2) + pow(y3_4,2) - 10 * (np.cos(2 * math.pi * x3_4 ) + np.cos(2 *
math.pi * y3_4 ))
   fcn3_4=(np.sum(last_fnc_4))/n4 * 10 * 2.5

   y3_5= np.random.uniform(-5,5,n5)
   x3_5= np.random.uniform(2.5,5,n5)
   last_fnc_5=20 + pow(x3_5,2) + pow(y3_5,2) - 10 * (np.cos(2 * math.pi * x3_5 ) + np.cos(2 *
math.pi * y3_5 ))
   fcn3_5=(np.sum(last_fnc_5))/n5 * 10 * 2.5

   stratified_lastfunction.append(fcn3_1+fcn3_2+fcn3_3+fcn3_4+fcn3_5)

var_stratlastfunc=np.var(stratified_lastfunction)
mean2_stratlastfunc=np.mean(stratified_lastfunction)
```

```
'''
print  "Mean Integration Value- Stratification: ", mean2_stratlastfunc
print  "Variance Integration Value- Stratification: ", var_stratlastfunc
'''

#print('\nMean value for function 3 with stratification: ',mean2_stratlastfunc)
#print('Variance value for function 3 with stratification',var_stratlastfunc)
```