Hrishikesh Hippalgaonkar
hippalga@usc.edu

EE 511 Simulation Methods for Stochastic Systems
Project #4

https://github.com/hrishi121/Simulation-methods-for-Stochastic-Systems

❖ Monte Carlo Integration:

Monte Carlo methods consist of a broad spectrum of computational algorithms that depend on repeated random sampling to obtain numerical results. Monte Carlo integration is a technique for numerical definite integration using random numbers. The basic idea of Monte Carlo integration is as follows:

Suppose we want to evaluate the following integral:

$$\boldsymbol{I = \int_a^b g(x)dx}$$

This can be expressed as

$$\boldsymbol{I = \frac{(b-a)}{(b-a)}\int_a^b g(x)dx}$$

$$\boldsymbol{I = (b-a)\int_a^b g(x) \times \frac{1}{(b-a)}dx}$$

But we can define a uniform pdf as

$$\boldsymbol{X \sim U(a,b) = \begin{cases} \frac{1}{(b-a)} & x \in (a,b) \\ 0 & elsewhere \end{cases}}$$

Therefore, the definite integral $\boldsymbol{I}$ can be expressed as

$$\boldsymbol{\tilde{I} = (b-a)E_X[g(x)] = (b-a)\left[\frac{1}{n}\sum_{i=1}^{n} g(X_i)\right] \quad X \sim U(a,b)}$$ **- equation 1**

And, according to the Law of Large Numbers,

$$\boldsymbol{\tilde{I}_n \xrightarrow{LLN} I}$$ as # of samples 'n' increases

1. Problem 1

[Pi-Estimation]

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 centered at the origin. This quarter circle inscribes the unit square as shown below:

i.) Use these random samples to estimate the area of the inscribed quarter circle. Use this area estimate to estimate the value of pi. Do k=50 runs of these pi-estimations. Plot the histogram of the k=50 pi estimates (each estimate based on n=100 2-D uniform samples).

ii.) Repeat the experiment with different numbers of uniform samples, n. Plot the sample variance of the pi-estimates for these different values of n. Keep k=50 for all these runs. Comment on the sample variance of your estimates.

Code:

```
import numpy as np
import matplotlib.pyplot as plt

################################ Part (a) ###################################

samples = 100
pi_pred = np.empty([50])
for i in range(0, 50):
    x = np.random.rand(samples, 2)
    count = ( (x[:,0]**2 + x[:,1]**2 ) < 1).sum()
    pi_pred[i] = (count/samples) * 4

print("Average Estimated value of pi: ", np.mean(pi_pred))

plt.figure(1)
plt.hist(pi_pred, histtype = 'bar', facecolor='green', alpha=0.75)
plt.xlabel('Estimated value of Pi')
plt.ylabel('# of times')
plt.title('Estimating the value of π using Monte Carlo simulation')
plt.grid(True)
plt.show()

################################ Part (b) ###################################

n = np.logspace(2, 4)   # Range of uniform samples selected
var_n = np.empty([50])  # Sample variance for different values of n
k = 0
for j in n:
    samples = int(np.floor(j))
    pi_pred = np.empty([50])
    for i in range(0, 50):
        x = np.random.rand(samples, 2)
        count = ( (x[:,0]**2 + x[:,1]**2 ) < 1).sum()
        pi_pred[i] = (count/samples) * 4
    var_n[k] = np.var(pi_pred)
    k = k + 1

t = np.floor(n)
plt.figure(2)
plt.plot(t, var_n, '+g-', alpha=0.75)
plt.xlabel('# of uniform samples selected (n)')
plt.ylabel('Sample variance')
plt.title('Graph of Sample variance of π - estimates for different values of n')
plt.grid(True); plt.show()
```

Result:

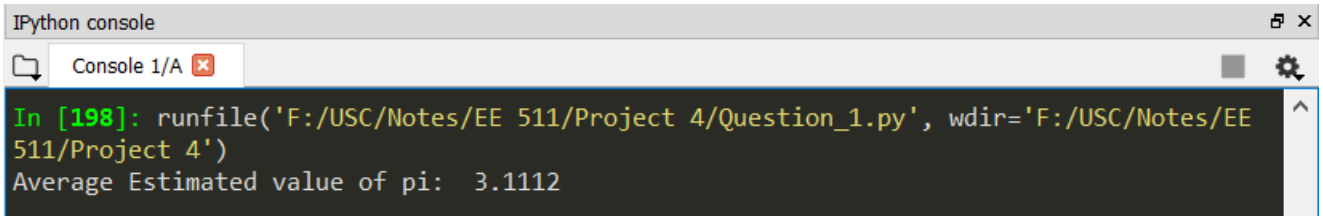

Figure 1a. Estimated value of π using Monte Carlo Estimation and n = 100 samples

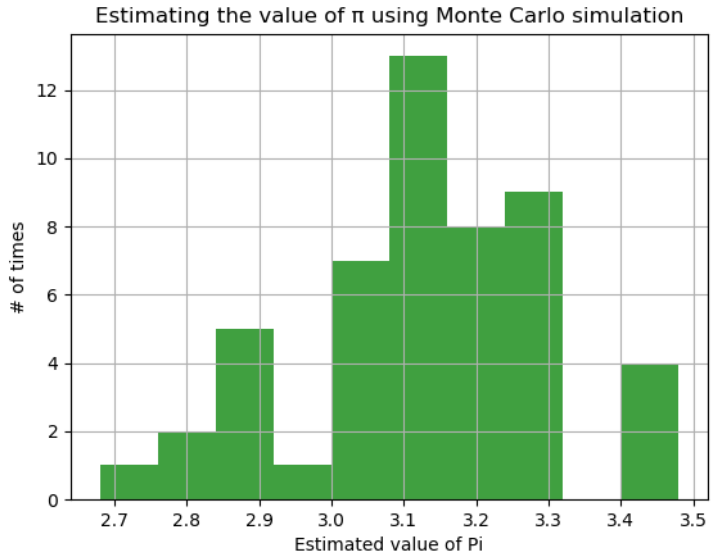

Figure 1b. Histogram of the k = 50 π- estimates

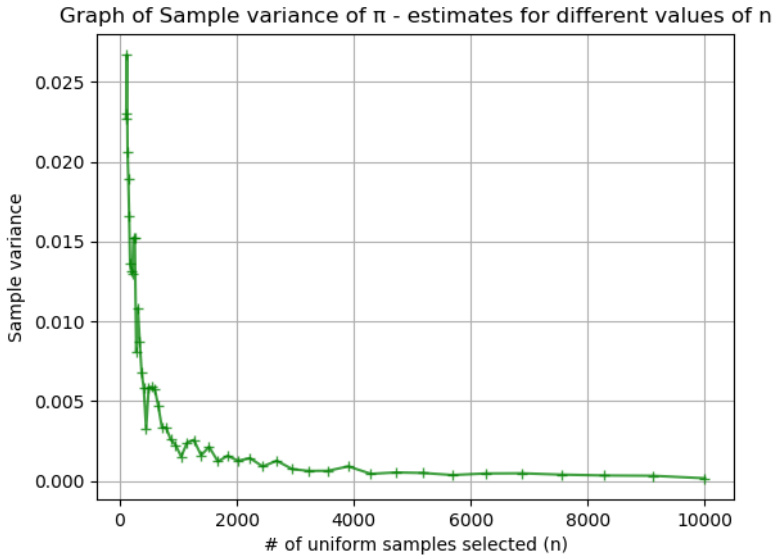

Figure 1c. Sample variance of π- estimates for different values of n

From figure 1a it can be observed that the estimated value of π (3.112) is quite close to actual approximated value of π (3.14159265…..). Since the number of samples used was quite low (n = 100) and no variance reduction techniques were used, the estimate was not as close as the actual approximated value.

In figure 1c it can be observed that, initially as the number of samples n increases, the sample variance steeply decreases up to n = 2000, and after n = 4000 the sample variance remains constant and stays very close to zero. This means that around n = 4000 samples will be needed to closely estimate π, and increasing number of samples after that won't affect the estimated value.

❖ Monte Carlo variance reduction techniques:

There are several general techniques for variance reduction, known as Monte Carlo swindles since these methods improve the accuracy and convergence rate of Monte Carlo integration without increasing the number of Monte Carlo samples. Some Monte Carlo swindles are:

- Importance Sampling
- Stratified Sampling
- Control Variates
- Antithetic Variates
- Conditioning swindles including Rao-Blackwellization and independent variance decomposition

1. **Importance Sampling:**
   Suppose we want to evaluate the following integral

$$\boldsymbol{I = \int_a^b g(x)dx}$$

We can now introduce another distribution called ***"importance function h(x)"***, whose distribution resembles the function g(x), and rewrite the above integral as

$$\boldsymbol{I = \int_a^b \frac{g(x)}{h(x)} \times h(x)\, dx}$$

And using equation 1 derived in the first section, we can represent the above integral as

$$\boldsymbol{\tilde{I} = (b-a) \times E_{X\sim h}\left(\frac{g(x)}{h(x)}\right)}$$

Previously, the random variable $\boldsymbol{X}$ was chosen from the uniform distribution, but now we are choosing the random variable from the distribution $\boldsymbol{h(x)}$, which closely resemble the distribution of the original function $\boldsymbol{g(x)}$**.** This improves the estimation.

2. **Stratified Sampling:**
   In stratified sampling, we divide the ROI for the integral to be evaluated in regions which have same variance. Furthermore, we choose higher number of samples from those regions which have a higher variance and choose low number of samples from regions which have a lower variance. This improves the estimation.

- **Strengths and weaknesses of stratification and importance sampling**

  In stratified sampling, we choose more samples from that region where the variance of the given function is quite large or the slope of the function is quite steep. This strategy is quite simple to implement. However, the problem with this technique is that if the function has too many areas over which it varies greatly, it can become cumbersome to divide and sample from each region. Also, one needs to first visualize the function or check the gradient of the function to see which areas have the greatest slope; it might not always be easily possible to visualize the function or compute the gradient.

  Importance sampling is more complex than stratified sampling, mainly because it is very difficult to find a probability density function which can closely approximate the given function. Furthermore, it might be possible that the given function cannot be approximated by any pdf at all. But, in some cases there might exist a pdf which might closely approximate the given function, and in such case, importance sampling will achieve better results than stratified sampling.

2. Problem 2

Use n=1000 random samples to obtain Monte Carlo estimates for the definite integrals:

$(\boldsymbol{a})\ \dfrac{\mathbf{1}}{\mathbf{1}+\sinh(2x)\times\ln(x)}\quad x\ in\ [0.8,3]$

$(\boldsymbol{b})\ e^{-(x^2+y^2)}\quad (x,y)in\ [-\pi,\pi]$

Calculate the sample variance of the Monte Carlo estimates using a similar method as in Problem 1. Use the same number of random samples, n=1000, to obtain those Monte Carlo estimates. But this time incorporate stratification and importance sampling in the Monte Carlo estimation procedures. Compare the Monte Carlo estimates and their sample variances. Discuss the quality of the Monte Carlo estimates from each method. Also discuss the strengths and weaknesses of stratification and importance sampling in Monte Carlo estimation. Test your integral estimator on the following function with your own choice of n samples:

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

Code: (Part a)

```
import numpy as np
from math import exp, pi, sinh, log
from scipy.integrate import quad
from scipy.stats import truncnorm
import matplotlib.pyplot as plt

############################## Actual Integration ##############################

area = quad(lambda x: (1 / (1 + sinh(2*x) * log(x)) ), 0.8, 3)

print('Actual value of the definite integral: {0:.3f}'.format(area[0]))

############################## Monte Carlo Estimation ##############################

samples = 1000
y1 = np.zeros([50])
y2 = np.zeros([50])
y3 = np.zeros([50])
count = 0

for i in range(0, 50):
    rv1 = 2.2 * np.random.random_sample([1000]) + 0.8
    for w in range(0, samples):
        temp = pow(1 + np.multiply(np.sinh(2*rv1[w]), np.log(rv1[w]) ), -1)
        y1[i] = y1[i] + temp
        #temp2 = np.divide(temp1, pf)
    y1[i] = (y1[i] / samples) * (2.2)
final1 = np.mean(y1)
print('\n Value of the definite using Monte Carlo estimation:')
print('1. Without variance reduction is: {0:.3f} ; Sample Variance:
{1:.3f}'.format(final1, np.var(y1)))

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

samples = 1000

# parameters for the proposal PDF
lower, upper = 0.8, 3
mu, sigma = 0.45, 0.5

for i in range(0, 50):
    #Generate random samples from the proposal PDF
    x = truncnorm.rvs(
            (lower - mu) / sigma, (upper - mu) / sigma,
            loc = mu, scale = sigma, size = samples)

    pf = truncnorm.pdf(
            x, (lower - mu) / sigma, (upper - mu) / sigma,
            loc = mu, scale = sigma)

    temp1 = pow(1 + np.multiply(np.sinh(2*x), np.log(x) ), -1)
    temp2 = np.divide(temp1, pf)
    y2[i] = np.sum(temp2) / samples

final2 = np.mean(y2)
print('2. Using Importance Sampling: {0:.3f} ; Sample Variance: {1:.3f}
'.format(final2, np.var(y2)))
```

```
############################ Stratified Sampling ##############################


for i in range(0, 50):
    rv1 = 1.2 * np.random.random_sample([550]) + 0.8
    rv2 = np.random.random_sample([450]) + 2
    for w in range(0, 550):
        temp1 = pow(1 + np.multiply(np.sinh(2*rv1[w]), np.log(rv1[w]) ), -1)
        y3[i] = y3[i] + temp1
    for w in range(0, 450):
        temp2 = pow(1 + np.multiply(np.sinh(2*rv2[w]), np.log(rv2[w]) ), -1)
        y3[i] = y3[i] + temp2
    y3[i] = (y3[i] / samples) * (2.2)
final3 = np.mean(y3)
print('3. Using Stratified Sampling: {0:.3f} ; Sample Variance: {1:.3f}'.format(final3,
np.var(y3)))

############################## Plt the function ################################

x1 = np.linspace(0.1, 3)
x2 = np.linspace(0.8, 3)
rv1 = pow(1 + np.multiply(np.sinh(2*x1), np.log(x1) ), -1)
rv2 = pow(1 + np.multiply(np.sinh(2*x2), np.log(x2) ), -1)

plt.figure(1)
plt.plot(x1, rv1, x2, rv2)
plt.legend(('Extended function', 'Actual Function'))
plt.xlabel('x')
plt.ylabel('y')
plt.title('Graph of first function')
plt.grid(True)
plt.show()
```

Code: (Part b)

```
import numpy as np
from math import exp, pi
from scipy.integrate import dblquad
from scipy.stats import multivariate_normal as mvn
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

############################## Actual Integration ##############################

area = dblquad(lambda x, y: exp(-1*(x**4 + y**4)), -pi, pi,
               lambda x: -pi, lambda x: pi)

print('Actual value of the definite integral: {0:.3f}'.format(area[0]))

############################ Monte Carlo Estimation ############################

samples = 1000
y1 = np.zeros([50])
y2 = np.zeros([50])
y3 = np.zeros([50])
rv1 = np.empty([samples, 2])
rv2 = np.empty([samples, 2])
count = 0
```

```
for i in range(0, 50):
    rv1 = 2*pi * np.random.random_sample([1000, 2]) - pi
    for w in range(0, samples):
        temp = exp(-1 * (pow(rv1[w, 0], 4) + pow(rv1[w, 1], 4)))
        y1[i] = y1[i] + temp
        #temp2 = np.divide(temp1, pf)
    y1[i] = (y1[i] / samples) * (4 * pi *pi)
final1 = np.mean(y1)
print('\n Value of the definite using Monte Carlo estimation:')
print('1. Without variance reduction : {0:.3f} ; Sample Variance:
{1:.3f}'.format(final1, np.var(y1)))

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

for i in range(0, 50):
    #Generate random samples from the proposal PDF
    while True:
        if count > samples - 1:
            break
        t1 = mvn.rvs(mean = [0,0], cov = [[1, 0], [0, 1]])
        if (t1[0] <= pi and t1[1] <=pi and t1[0] >= -pi and t1[1] >= -pi):
            rv2[count, 0] = t1[0]
            rv2[count, 1] = t1[1]
            count = count + 1
        else:
            count = count + 1

    pf = mvn.pdf(rv2, mean = [0,0], cov = [[1, 0], [0, 1]])
    temp1 = np.empty([samples])


    for w in range(0, samples):
        temp1[w] = exp(-1 * (pow(rv2[w, 0], 4) + pow(rv2[w, 1], 4)))
    temp2 = np.divide(temp1, pf)
    y2[i] = (np.sum(temp2) / samples)

final2 = np.mean(y2)
print('2. Using Importance Sampling: {0:.3f} ; Sample Variance: {1:.3f}'.format(final2,
np.var(y2)))


############################ Stratified sampling ############################


for i in range(0, 50):
    rv1 = 2*1.73 * np.random.random_sample([1000, 2]) - 1.73
    for w in range(0, samples):
        temp = exp(-1 * (pow(rv1[w, 0], 4) + pow(rv1[w, 1], 4)))
        y3[i] = y3[i] + temp
        #temp2 = np.divide(temp1, pf)
    y3[i] = (y3[i] / samples) * (4 * 1.73 * 1.73)
final3 = np.mean(y3)
print('3. Using stratified Sampling: {0:.3f} ; Sample Variance: {1:.3f}'.format(final3,
np.var(y3)))

############################ Plt the function ################################

t = np.linspace(-pi, pi)
x2, y2 = np.meshgrid(t, t)
```

```
rv3 = np.empty([50, 50])
for i in range(0, 50):
    for j in range(0, 50):
        temp = pow(x2[i, j], 4) + pow(y2[i,j], 4)
        rv3[i, j] = exp(-1*temp)

plt.figure(2)
plt.contourf(x2, y2, rv3)
plt.colorbar()
plt.xticks(); plt.yticks()
plt.xlabel('x') ; plt.xlabel('y')
plt.title('Contour plot of the 2nd function')
plt.show()

plt.figure(3)
ax = plt.axes(projection='3d')
ax.plot_surface(x2, y2, rv3,  cmap='viridis', edgecolor='none')
ax.set_xlabel('x'); ax.set_ylabel('y'); ax.set_zlabel('z');
ax.set_title('Graph of the 2nd function')
plt.show()
```

Code: (Part c)

```
import numpy as np
from math import cos, pi
from scipy.integrate import dblquad

############################## Actual Integration ##############################

area = dblquad(lambda x, y: 20 + x**2 + y**2 - 10*(cos(2*pi*x) + cos(2*pi*y)),
               -5, 5, lambda x: -5, lambda x: 5)

print('Actual value of the definite integral: {0:.3f}'.format(area[0]))

########################### Monte Carlo Estimation ###########################

samples = 1000
y1 = np.zeros([50])
rv = np.empty([samples, 2])

for i in range(0, 50):
    rv = 10 * np.random.random_sample([samples, 2]) - 5
    for w in range(0, samples):
        temp = 20 + rv[w, 0]**2 + rv[w, 1]**2 - 10*(cos(2*pi*rv[w, 0]) + cos(2*pi*rv[w,
0]))
        y1[i] = y1[i] + temp
        #temp2 = np.divide(temp1, pf)
    y1[i] = (y1[i] / samples) * 100
final1 = np.mean(y1)
print('Value of the definite integral using Monte Carlo estimation:
{0:.3f}'.format(final1))
```

Result:

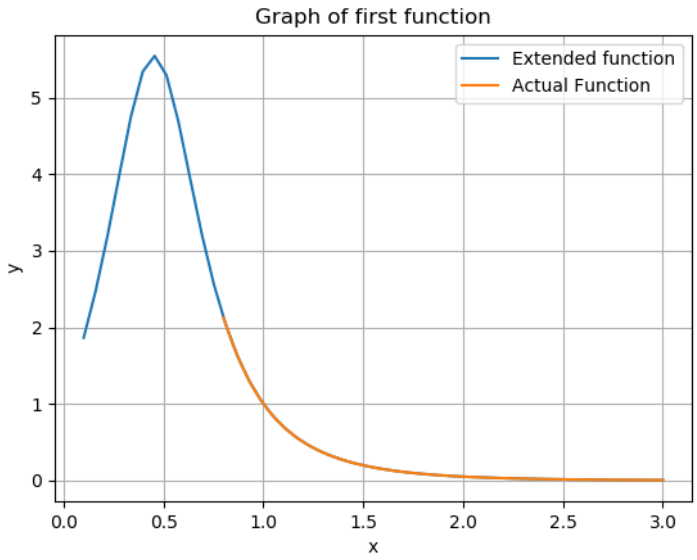

Figure 2a. Graph of the first function `(1 / (1 + sinh(2*x) * log(x))`

```
IPython console
Console 1/A
In [188]: runfile('F:/USC/Notes/EE 511/Project 4/Question_2a.py', wdir='F:/USC/Notes/EE
511/Project 4')
Actual value of the definite integral: 0.610

 Value of the definite using Monte Carlo estimation:
1. Without variance reduction is: 0.607 ; Sample Variance: 0.001
2. Using Importance Sampling: 0.613 ; Sample Variance: 0.002
3. Using Stratified Sampling: 0.614 ; Sample Variance: 0.001
```

Figure 2b. Estimate of the definite integral of the first function

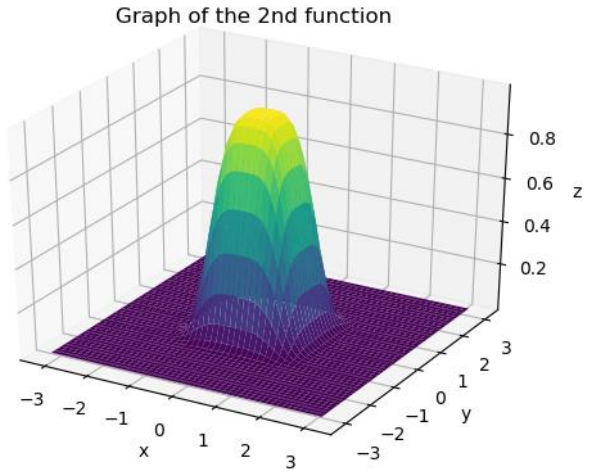

Figure 2c. Graph of the second function `exp(-1*(x^4 + y^4))`

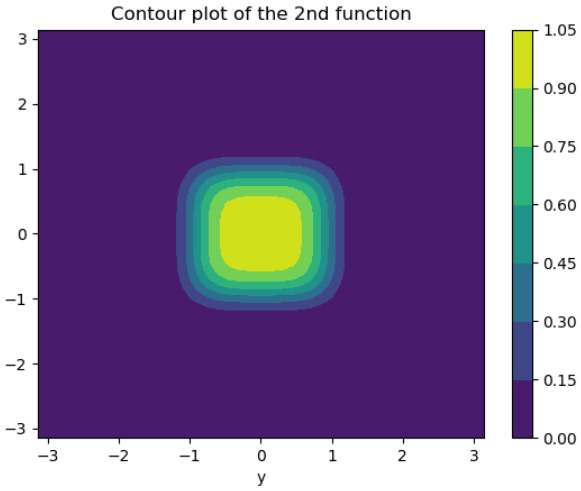

Figure 2d. Contour plot of the second function

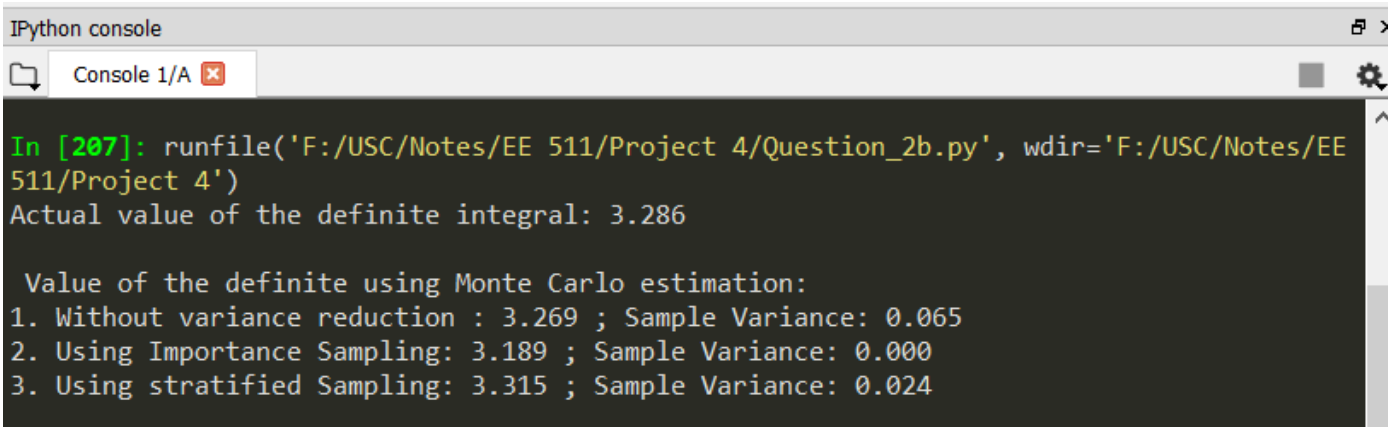

Figure 2e. Estimate of the definite integral of the second function

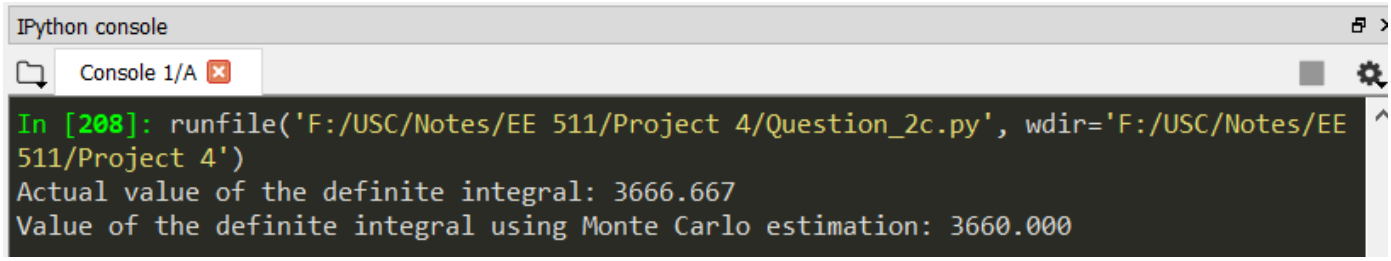

Figure 2f. Estimate of the definite integral of the third function

It can be observed in figure 2a that the graph of the function is like a gaussian curve. Thus, for importance sampling, I have used a gaussian pdf with mean around 0.45 and standard deviation 0.5. Furthermore, for stratified sampling, I have divided the region of function in two regions, one (0.8, 1.5) and another (0.15, 0.3). Since the function is steep in the first region, I have used more samples in this region and less samples in the next region.

In figure 2b it can be observed that both the Monte Carlo estimates, one with and other without variance reduction, are quite close to actual integration value.

It can be observed in figure 2c that the graph of the function is like a bi-variate gaussian curve. Thus, for importance sampling, I have used a bivariate gaussian pdf with mean 0 and an identity covariance matrix. Furthermore, for stratified sampling, I have divided the region of function in two regions, one ($-1.73 < x, y < 1.73$) and other is the remaining area over which the function is defined. Since the function is almost zero in the remaining area, I have drawn almost all samples from the first region and very few samples from the second region.

In figure 2e it can be observed that both the Monte Carlo estimates, one with and other without variance reduction, are quite close to actual integration value.

Similarly, it can be observed in figure 2f. that the estimated value of the definite integral of the third function is almost close to actual value.