# **MTH 251 Probability and Statistics Project: Monte Carlo Integration**

Authors:

Alden Dimarco,

Dillon Girvin

Date: April 25th, 2024

TA: Cameron Bundy

Lecture Section 2

Professor: Richard O'Shaughnessy

[GitHub Project](https://github.com/DillonGirvin/MTH251MonteCarloIntegration-DillonGirvin/tree/main)

**Abstract:**

This project attempts to accurately approximate the Rosenbrock Integral within the bounds [-5, 5] for both x and y in two-dimensional space. To accomplish this, the project makes use of Monte Carlo Integration. To use this estimation technique, several python programs were constructed. IntegralEstimates0.py and IntegralEstimates1.py estimate several one-dimensional distributions, IntegralEstimates0.py estimating using uniform distributions, and IntegralEstimates1.py estimating using normal distributions. The Rosenbrock Integral was estimated by Rosenbrock.py using a uniform distribution in two-dimensional space. The python code used the scipy and numpy libraries to make the required distributions and approximate the desired integrals. The exact code used is provided in the GitHub project linked on the cover page of this report. While the findings of IntegralEstimates0 and 1 correlate with the expected output, Rosenbrock.py is not functional, and gives an incorrect result. More details on this problem are included in the Results section.

**Project Overview:**

The Monte Carlo integration technique can estimate the integral of a multidimensional function over a multidimensional region by rewriting the integral as the expected value of the original function over a sampling distribution. The Central Limit Theorem states that as the number of samples in the distribution approaches infinity, the sample mean becomes a more accurate estimate of the true mean. The number of samples used in this project are sufficiently large enough to use the sample mean in place of the true mean. Succinctly, the Monte Carlo method allows for approximation of both the value and error of a multidimensional integral by evaluating the function over a set of random numbers from a sampling distribution.

However, due to the large number of samples needed for the Central Limit Theorem to hold true, doing the estimation by hand is wildly inefficient and impractical. Therefore, Python was used to create samples from the required distributions and estimate the integral. Within the project, two different sampling distributions were used: A uniform distribution, and a standard normal distribution. The distribution used to approximate the integrals in IntegralEstimates0.py and Rosenbrock.py was the uniform distribution, while IntegralEstimates1.py used the standard normal distribution.

**Results:**

*IntegralEstimates0.py:*

IntegralEstimates0.py evaluated several different integrals using a uniform distribution. The exact functions used can be seen in either the project specifications document provided in MyCourses, or the ReadMe on the GitHub linked on the cover page.

Figure 1 – IntegralEstimates0.py output:

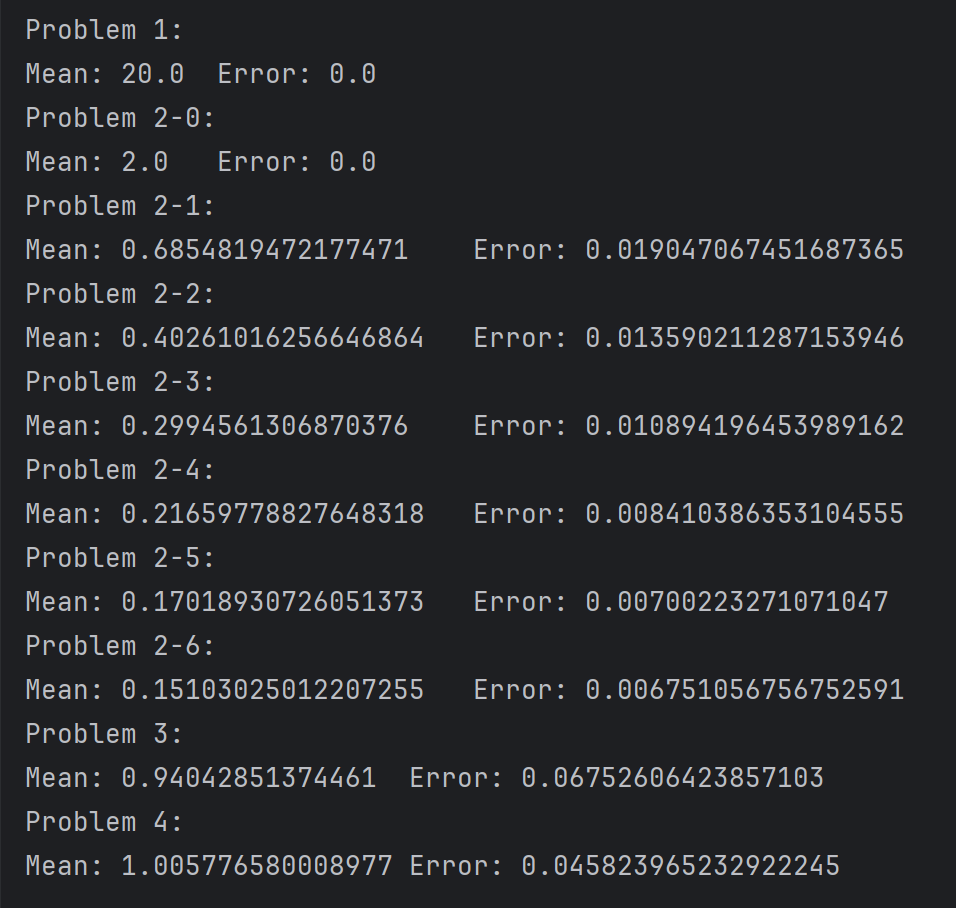


Figure 1 shows the output for IntegralEstimates0.py. It should be noted that both problem 1 and 2-0 have no error. The function used in Problem 1 was f(x) = 1. Because the function was constant throughout the entire range, the value of the function is not dependent on the randomly selected samples within the uniform distribution. Similarly, problem 2-0 is also a constant function that does not depend on the samples selected. The l value within the Legendre function used corresponds to the degree of the function. Since for problem 2-0, the l value used was 0, it is also a constant function and as such has an error of zero for the same reason as problem 1.

*IntegralEstimates1.py:*

IntegralEstimates1.py evaluated the same functions as IntegralEstimates0.py, apart from the Legendre in IntegralEstimates0.py’s problem 2. The difference between the programs is that IntegralEstimates1.py used a standard normal sampling distribution instead of a uniform sampling distribution. It should be noted that problems 2 and 3 for IntegralEstimates1.py correspond to the functions used in problems 3 and 4 in IntegralEstimates0.py respectively.

Figure 2 – IntegralEstimates1.py output:

A computer error message

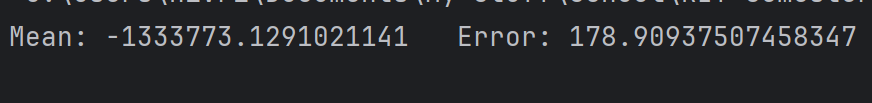
Description automatically generated

Figure 2 shows the output for IntegralEstimates1.py. In Figure 1, the problem approximating the normal distribution (Problem 3) has an error while the same function in Figure 2 (Problem 2) does not. The reason for this is that the uniform sampling distribution assigns equal probabilities for all x values, while the normal sampling distribution assigns probabilities normally. Since the function being evaluated is the normal distribution, if the random samples used are assigned normally, there will be no error. This means that the closer the shape of the sampling distribution is to the function being approximated, the more accurate the result will be. This can be seen in Figure 2 as the approximated values are far less accurate for Problems 1 and 3 than they were in Figure 1. To accurately approximate these functions using the normal sampling distribution, far more samples will be required, dramatically impacting computation time. As such, it is important to use a sampling distribution that matches the function that is being approximated.

*Rosenbrock.py*

Rosenbrock.py uses the knowledge gained from IntegralEstimates0 and 1 to attempt to approximate the Rosenbrock Integral. A uniform sampling distribution was used, with 108 samples. The Rosenbrock Function used in the program was provided in the Project Specifications document in MyCourses, and the sample code was referenced to determine how to use the method created in IntegralEstimates0 and 1 for higher dimensions. However, the output is completely incorrect. The code was reviewed and verified multiple times, but the reason for this issue has still not yet been identified. The program is available on the GitHub page linked on the cover page of this report for reference to the exact code used to attempt the approximation. Regardless of the fact the output is incorrect, the output of the program will be provided below for documentation purposes.

Figure 3 – Incorrect Rosenbrock.py output:



Initially the problem was assumed to be that the function given in MyCourses was blindly followed, so the function was re-written to use the normal python “\*\*” operator while referencing the Rosenbrock Integral itself instead. When this resulted in the same problem, the function was rolled back to correspond directly to what was provided in MyCourses. Since the error of the function was high, a larger number of samples was used than was deemed necessary, resulting in longer computation times. Unfortunately, despite this change, the approximation continued to converge to an incorrect value.

The code used in Rosenbrock.py was created while directly referencing the sample code on MyCourses, and as no documentation was provided on how the sample code functions, it was difficult to identify exactly what the sample code was doing. Through research of the [numpy](https://numpy.org/doc/) and [scipy](https://docs.scipy.org/doc/scipy/) library documentation, every line of the code was inspected to determine its function, and comments and pydocs were created to help explain what the code did. These comments and pydocs will be viewable on the GitHub page linked on the cover page of this report. If the Rosenbrock.py program is run on another device, it is suggested to temporarily reduce the number of samples to reduce computation time while troubleshooting.

**Conclusion**

Monte Carlo Integration is a powerful method to estimate multi-dimensional integrals that are difficult to complete by hand. It is crucial to consider the sampling distribution used for approximation. Choosing a sampling distribution that does not fit the function well results in a higher number of samples required for accurate approximation. While the project was able to successfully use Python to approximate several one-dimensional integrals, the higher dimensional Rosenbrock Integral was far more troublesome to compute. Several troubleshooting techniques were used, but unfortunately all were ineffective, and the cause of the incorrect output is still unknown.