1 Introduction

Monte Carlo integration serves as a powerful numerical method for estimating integrals by using random numbers. It is widely used because of its simplicity and directness. The method is based on the principle of using random sampling to approximate the integral value by averaging the function values at randomly generated points within the integration domain. By generating a large number of random samples and averaging their function values, Monte Carlo integration can provide a reliable estimate for the value of the integral. However, as Russel Caflisch notes "its convergence rate, $O(N^{-1/2})$, is independent of dimension, which shows Monte Carlo to be very robust but also slow" (Caflisch 1998, p. 1).

The Python script we wrote implements Monte Carlo integration and provides a comprehensive exploration of its behavior with varying sample sizes. The goal is to gain a deeper understanding of the convergence trend and to assess the effectiveness of the Quasi-Monte Carlo method as a variance reduction technique. Moreover, we apply the `nquad()` function from the `scipy.integrate` module to obtain a numerical solution to a challenging 6-dimensional integral problem.

2 Findings

2.1 Monte Carlo Integration

In our case, we implemented a Python script to estimate a 6-dimensional integral using Monte

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \sin\left(\frac{\pi}{2} \exp\left(\frac{x_1 + x_2 + x_3}{3}\right) \exp\left(\frac{x_4 + x_5 + x_6}{3}\right)\right) \mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}x_3 \mathrm{d}x_4 \mathrm{d}x_5 \mathrm{d}x_6$$

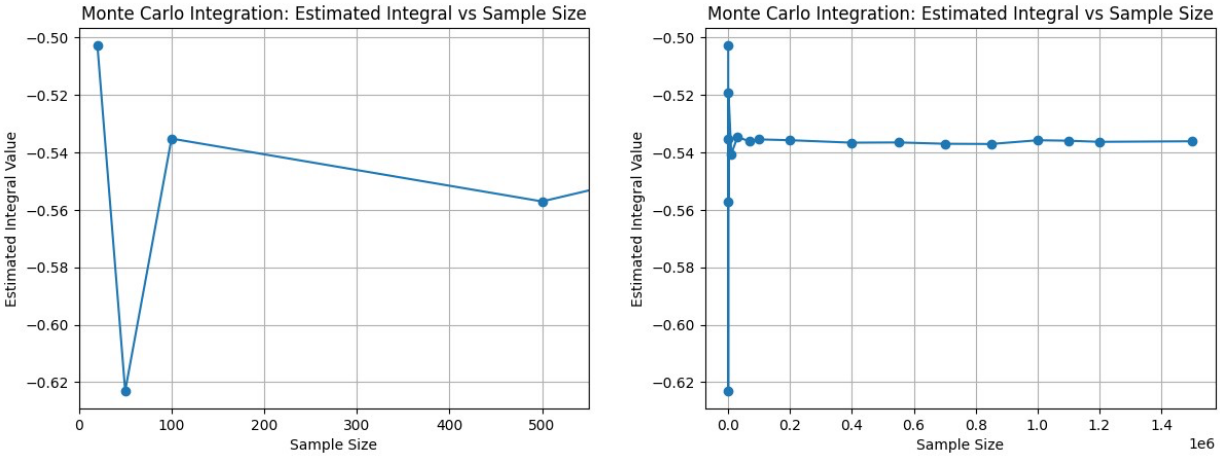
Carlo integration.

We generate random samples within the integration domain, which, in our case, is [0, 1] for each dimension. The function values at these sampled points were averaged to estimate the integral value. To obtain the final estimated integral value, the average is multiplied by the volume of the integration domain, which is equal to 1 raised to the power of the number of dimensions (in this case, 6). For our first number of samples we use 1000000. Our code computes the estimated integral value at -0.5362565028744494, which, as we later see, is a good approximation.

2.2 Different Sample Sizes

To understand the behavior of the estimate with varying sample sizes, we ran the Monte Carlo integration script for different sample sizes and recorded the estimated integral values. The

purpose of this experiment was to analyze the convergence behavior of the estimate. We plot our estimates against different numbers of samples to see the convergence trend better.



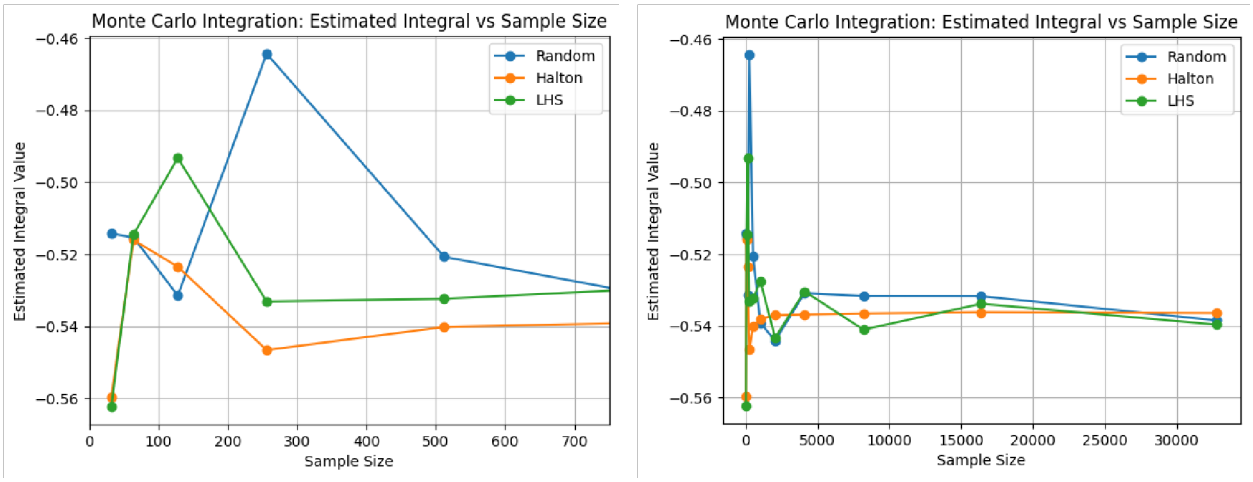
As the sample size increases, our estimate tends to improve in terms of accuracy and precision. This improvement occurs, because with a larger number of samples, the random points are more evenly distributed within the integration domain, leading to a better approximation of the integral. Thus, the larger the Monte Carlo sample, the smaller the error around our Monte Carlo estimate. The law of large numbers suggests that as the sample size approaches infinity, the average of the function values at the sampled points converges to the true value of the integral.

Therefore, increasing the number of random samples in Monte Carlo integration generally leads to a better estimate. However, generating a very large number of samples can become computationally expensive.

2.3 Variance Reduction Technique (Quasi-Monte-Carlo)

In traditional Monte Carlo integration, random samples are generated uniformly within the integration domain. However, this approach may lead to clustering or uneven coverage of the domain, especially in higher dimensions. Quasi-Monte Carlo is a variance reduction technique, which aims to minimize the variance of the estimated integral, leading to more accurate results and improved convergence properties. “The points in a quasi-random sequence are correlated to provide greater uniformity” (Caflisch 1998, p. 1). Examples of such sequences include the Halton sequence, Sobol sequence, and Latin Hypercube Sampling (LHS). Compared to the convergence rate of traditional Monte Carlo methods, the quasi-Monte Carlo method offers a faster convergence rate of approximately $O((logN)^k N^{-1})$.

To compare the estimates obtained using random sampling and quasi-random sequences, we modified the Monte Carlo integration script to include the Halton sequence as well as Latin Hypercube Sampling (LHS) as a sampling method.



As evident from the graphs, both the Halton sequence and LHS demonstrate faster convergence and improved accuracy. However, between the two, utilizing the Halton sequence appears to be the better choice in this case.

2.4 Application of `nquad()` from `scipy.integrate`

As an alternative approach to numerical integration, we explore the application of the `nquad()` function from the `scipy.integrate` module. By defining the integrand function and specifying the integration limits for each dimension as the range [0, 1], we utilize the `nquad()` function to compute the integral value and estimate the error. The numerical solution we get is 0.5363665358782651, which is very similar to the value we got in section 2.1. However, because of the complexity of the integrand function, this approach takes significantly longer than Monte Carlo integration. By using the `time` module, we measured the execution time of the code and obtained a value of 280.837 seconds. In comparison, it took 7.762 seconds to run the Monte Carlo integration using a sample size of 1000000.

3 Conclusion

Monte Carlo integration offers a powerful tool for estimating integrals, with larger sample sizes leading to more accurate results. Additionally, the incorporation of variance reduction techniques, such as the Quasi-Monte Carlo method, especially the Halton sequence, significantly improves the accuracy and convergence speed of the estimates compared to random sampling.

This stark difference in execution time highlights the computational efficiency of the Monte Carlo integration method compared to the `nquad()` function for the specific integrand function and higher-dimensional problem considered in this report. While the `nquad()` function provides an alternative approach to numerical integration, it may not be the most suitable choice when efficiency is a critical factor. It is computationally demanding, particularly for complex integrand

functions and higher dimensions. Therefore, careful consideration must be given to the computational resources and time required when using this method.

The findings emphasize the importance of carefully considering computational resources and time requirements when utilizing numerical integration methods, especially for higher-dimensional integrals. The choice between Monte Carlo integration, Quasi-Monte Carlo methods, or other numerical integration techniques should be based on the specific requirements of the problem at hand.