Stochastic Integral Estimation Methods

Kiara Gholizad[1]

[1]Department of Physics, Sharif University of Technology, Tehran, Iran

April 14, 2025

Abstract

In this report, we investigate numerical integration with stochastic methods, with an emphasis on Monte Carlo integration. We consider three different types of problems: the calculation of definite integrals with the use of the Sploosh algorithm, comparison between simple sampling and importance sampling for exponential integrals, and finding a sphere's center of mass with non-uniform density. We implement computational strategies for each problem, checking their efficiency, accuracy, and convergence properties. From our findings, we show that it is possible for stochastic methods to solve deterministic problems effectively, with certain observations made on error estimation and choice of sampling strategy. We also observe that most of our stochastic methods converge with expected $O(1/\sqrt{N})$ convergence. We have made all implementations and code available on GitHub at `https://github.com/KiaraGholizad/Computational-Physics`.

Contents

1 Introduction

Numerical integration is a fundamental problem in computational physicsthat is being applied from quantum mechanics to statistical physics. Most traditional deterministic integration methods like the trapezoidal rule or Simpson's rule are extremely efficient for smooth, well-behavioral functions in small dimensions. But these techniques tend to perform poorly with high-dimensional calculations or irregularly behaving functions.

Stochastic techniques, Monte Carlo integration methods in particular, provide an alternate solution. These techniques involve random sampling to approximate integrals and are especially useful for dealing with complex geometries or high-dimensional space. We examine three problems in this report that demonstrate the strength and range of stochastic integration techniques.

1.1 Theoretical Background

Monte Carlo methods rely on the law of large numbers, i.e., taking a large number of random samples, their mean will converge to the expected value. Using this principle for integration, we can estimate an integral by randomly picking points from the integrator domain and calculating their mean.

As outlined in (1), Monte Carlo integration possesses certain unique strengths:

- Error reduces by $O(1/\sqrt{N})$ with respect to the number of samples, regardless of dimensionality
- It is usually implemented easily
- It easily supports complex geometries.
- Error estimation is incorporated in the approach

Yet, their convergence rate (O(1/sqrtN)) is not faster than that of some deterministic estimators, which converge with O(1/N) or better for smooth functions of small dimension.

The methods we'll explore in this report include:

1. The Sploosh Algorithm: A geometric interpretation of Monte Carlo integration
2. Simple Sampling: Direct random sampling of the integration domain
3. Importance Sampling: an algorithm that concentrates samples in areas that make significant contributions to integration
4. Application to a physical problem: Finding the center of mass of a sphere with non-uniform density

1.2 System Specifications

All calculations carried out in this report were done on MacBook Pro with an M3 Pro processor. This computing system was so high-performance that we could perform simulations with 1,000,000 samples in acceptable time intervals (less than 1 second), which made it possible to carry out complete analyzes of convergence behavior and computational efficiency.

2 The Sploosh Algorithm

2.1 Theoretical Description

An analogy that gave rise to an algorithm called "Sploosh" is that of stones being thrown over a garden fence. Nobody can view a walled garden but wishes to make an estimate of a pond's area inside. By throwing stones indiscriminately over a garden fence and hearing a "sploosh" if a stone splashes in water, or hearing a "thud" if it fall on dry ground, an estimate of the pond area to garden area can be made.

Mathematically, the Sploosh algorithm approximates the integral of a function $f(x)$ between an interval $[a, b]$ by seeing it as computing an area under a curve. The algorithm goes as follows:

Algorithm 1 Sploosh Algorithm for Integration

Select y_{min} and y_{max} to form a bounding rectangle for $f(x)$ in $[a, b]$
Initialize counter $N_{below} = 0$
for $i = 1$ to N **do**
    Generate random $x \in (a, b)$
    Generate random $y \in (y_{min}, y_{max})$
    if $y < f(x)$ **then**
        Increment $N_{below} = N_{below} + 1$
    end if
end for
$p = \frac{N_{below}}{N}$ (proportion of points below the curve)
rectangle_area $= (b - a) \cdot (y_{max} - y_{min})$
below_curve_area $=$ rectangle_area $\cdot\, p$
integral $=$ below_curve_area $- (b - a) \cdot (-y_{min})$
error $=$ rectangle_area $\cdot \sqrt{\frac{p \cdot (1-p)}{N}}$
return integral, error

This method regards an area under a curve as a "pond" inside the rectangular "garden" of $[a, b] \times [y_{min}, y_{max}]$. Random points on this "pond" are proportional to areas, so if you multiply by the rectangular area, you have an estimate of an area under a function.

One key thing that should be kept in mind for functions that go across the x-axis (when both negative and positive values are considered) is handling the negative areas properly. We take care of this in our code by:

1. Setting y_{min} to the minimum value of $f(x)$ in the interval (typically negative)
2. Counting points below the curve regardless of whether they are above or below the x-axis
3. Adjusting the final calculation to account for the shift in the y-coordinate

2.2 Implementation

Implementation of the Sploosh algorithm conforms to the method outlined in the algorithm above. One of the main issues was dealing with functions that cross the x-axis, like our example function of $f(x) = x^3 - 5x$ over interval $[0, 2]$. This function is both negative

and positive, so we must be careful with the bounding rectangle, as well as calculation of the end area.

Our implementation uses direct random sampling from a uniform distribution over a rectangle that encompasses the function completely for the interval of integration. To use for function $f(x) = x^3 - 5x$ over $[0, 2]$, we have vertical limits of our sample rectangle with $y_{min} = -6$ and $y_{max} = 2$.

We seed the random number generator with a fixed value (12) so that our results are reproducible between runs. We estimate errors by employing the binomial distribution formula, since each point is either under or over the curve.

2.3 Visualization

In order to better visualize how the Sploosh algorithm operates, we produced a graphical display of random points that have been produced in the bounding rectangle, with points under the curve in green and points over the curve in red:

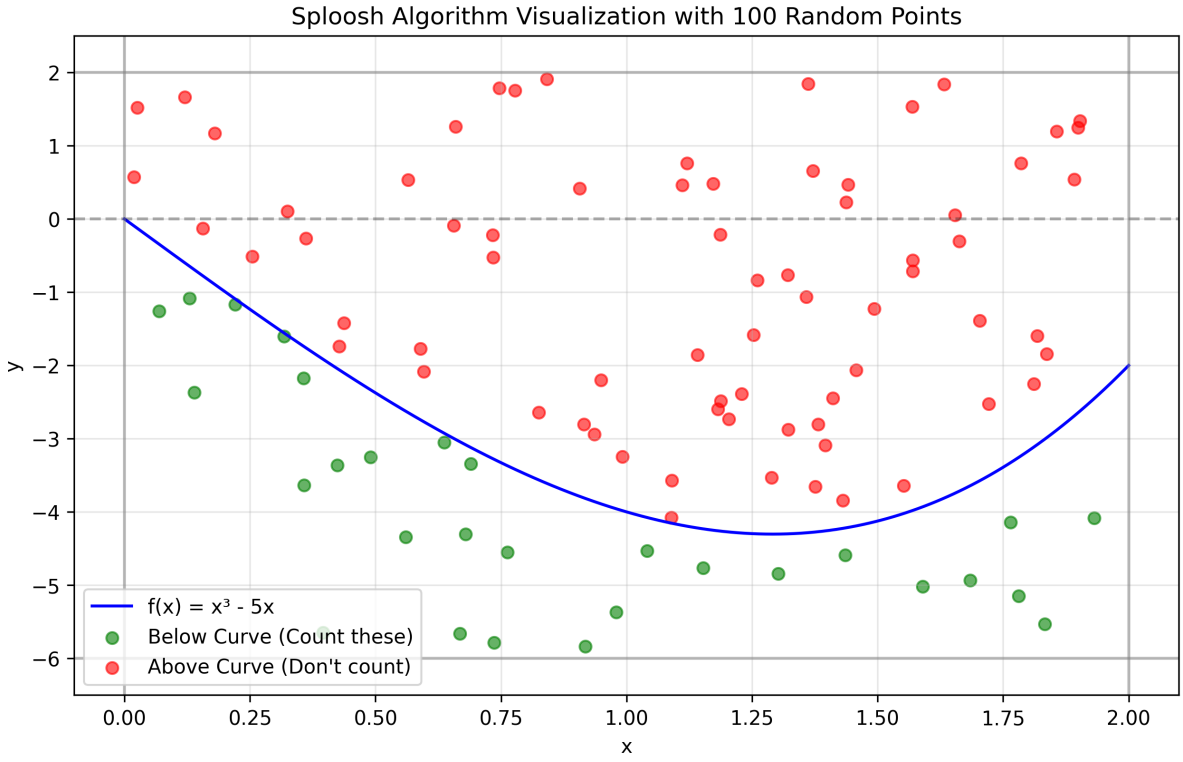

Figure 1: Visualization of the Splooosh algorithm with 100 random points. Green points (i.e., points below the curve) are counted, red points (points above the curve) are not. Function being integrated: $f(x) = x^3 - 5x$ over interval [0, 2].

This example explains that such an algorithm approximates the area under a curve by taking random points and counting those which lie under the curve.

2.4 Results and Analysis

In our test example, we computed the integral $\int_0^2 (x^3 - 5x)dx$. We know that the analytic solution to this integral is -6.

Table 1 displays the outcome of running this integral with various sample sizes using the Sploosh algorithm:

N	Integral	Statistical Error	Actual Error	Time (s)
100	-6.4000	0.7632	0.4000	0.0001
1,000	-5.7440	0.2469	0.2560	0.0007
10,000	-6.0080	0.0774	0.0080	0.0061
100,000	-5.9952	0.0245	0.0048	0.0508
1,000,000	-5.9921	0.0077	0.0079	0.4453

Table 1: Results of the Sploosh algorithm for $\int_0^2 (x^3 - 5x)dx$

Figure 2 illustrates that absolute error in the Sploosh algorithm converges with an increase in number of samples:

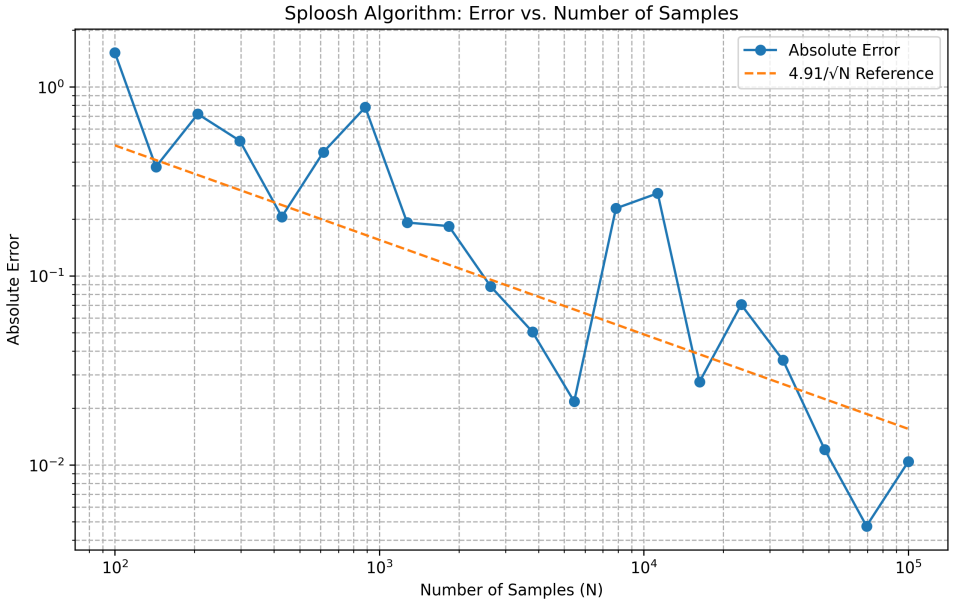

Figure 2: Log-log plot of absolute error vs. number of samples for the Sploosh algorithm against the theoretical convergence rate of $1/\sqrt{N}$. The scaling factor was computed from a calculation of error $\times\sqrt{N}$ average for these points.

As anticipated, the Sploosh algorithm's error drops off at about $1/\sqrt{N}$, which happens to be what is predicted by theoretical rates of convergence for Monte Carlo methods. The error that is returned by the algorithm is a fairly accurate estimate for the error, so we can find some confidence that the error analysis is correct.

The Sploosh algorithm precisely approximates the integral with increasing accuracy with a greater number of samples. Using 100,000 samples, it is accurate to 0.08% of the original. It converges, however, fairly slowly with a large number of samples needed to get good accuracy.

In terms of computational performance, the algorithm scales linearly with the number of samples. On our test system, processing 1 million samples took approximately 0.45 seconds, which is efficient for a Monte Carlo method.

3 Simple vs. Importance Sampling

3.1 Theoretical Description

In Problem 2, we are comparing two Monte Carlo methods for approximating the integral $\int_0^2 e^{-x^2} dx$:

1. Simple (Uniform) Sampling: Sampling points are drawn from the integration domain uniformly.
2. Importance Sampling: Points are sampled according to a probability distribution that concentrates samples in regions that contribute most to the integral.

3.1.1 Simple Sampling

In simple sampling, we write the integral as:

$$I = \int_a^b f(x)dx = (b-a) \cdot \frac{1}{b-a} \int_a^b f(x)dx = (b-a) \cdot \langle f \rangle \tag{1}$$

where $\langle f \rangle$ is an average of $f(x)$ over interval $[a, b]$. We approximate this average by taking a sample of N random points in $[a, b]$ and calculating:

$$\langle f \rangle \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{2}$$

The statistical error in this estimate is:

$$\sigma = (b-a) \cdot \frac{\sqrt{\mathrm{Var}(f)}}{\sqrt{N}} \tag{3}$$

where $\mathrm{Var}(f)$ denotes the variance of $f(x)$ over the interval $[a, b]$.

3.1.2 Importance Sampling

In importance sampling, we rewrite the integral as:

$$I = \int_a^b f(x)dx = \int_a^b \frac{f(x)}{g(x)} g(x)dx \tag{4}$$

where $g(x)$ is a probability density function. If we can sample points from the distribution $g(x)$, then we can estimate the integral as:

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{g(x_i)} \tag{5}$$

where the x_i are sampled from the distribution $g(x)$.

The point is that if we make $g(x)$ close to $f(x)$, then the variance of the ratio $f(x)/g(x)$ will be smaller than that for $f(x)$ individually, resulting in better estimates.

For our example, we employ $g(x) = e^{-x}$ as our importance sampling distribution, which can be easily sampled by using the inverse transform method.

3.2 Implementation

3.2.1 Simple Sampling Implementation

The process of using simple sampling is straightforward:

Algorithm 2 Simple Sampling for Integration

1: Generate N random points x_i uniformly in $[a, b]$
2: Calculate function values $f(x_i)$ for each point
3: Compute the mean of function values: $\mu = \frac{1}{N} \sum_{i=1}^{N} f(x_i)$
4: Calculate the variance: $\sigma^2 = \frac{1}{N-1} \sum_{i=1}^{N} (f(x_i) - \mu)^2$
5: Estimate the integral: $I = (b - a) \cdot \mu$
6: Estimate the error: $error = (b - a) \cdot \frac{\sigma}{\sqrt{N}}$
7: **return** $I, error$

3.2.2 Importance Sampling Implementation

For importance sampling we must generate random samples from distribution $g(x) = e^{-x}$. We apply the inverse transform method, which operates in the following way:

Algorithm 3 Generating Samples from Exponential Distribution

1: Generate N uniform random numbers $u_i \in [0, 1]$
2: Transform each u_i to $x_i = -\ln(1 - u_i)$
3: **return** $x_1, x_2, \ldots, x_N$

The method of inverse transform relies on observing that if U is a random variable that is uniform over $[0, 1]$ and if we have a function of distribution, say F, then taking $X = F^{-1}(U)$ will result in X having distribution function F. For an exponential distribution with $\lambda = 1$, we have that $F(x) = 1 - e^{-x}$ for $x \geq 0$ so that $F^{-1}(u) = -\ln(1 - u)$.

After we have simulated samples from an exponential distribution, we can estimate the integral:

Algorithm 4 Importance Sampling for Integration

1: Generate N samples x_i from distribution $g(x) = e^{-x}$ using inverse transform method
2: Filter samples to include only those in $[a, b]$
3: For valid samples, calculate ratios $r_i = \frac{f(x_i)}{g(x_i)}$
4: Compute the mean of ratios: $\mu_r = \frac{1}{N_{valid}} \sum_{i=1}^{N_{valid}} r_i$
5: Calculate the variance: $\sigma_r^2 = \frac{1}{N_{valid}-1} \sum_{i=1}^{N_{valid}} (r_i - \mu_r)^2$
6: Calculate the integral of $g(x)$ over $[a, b]$: $G = \int_a^b g(x)dx = 1 - e^{-b}$ (for $a = 0$)
7: Estimate the integral: $I = G \cdot \mu_r$
8: Estimate the error: $error = G \cdot \frac{\sigma_r}{\sqrt{N_{valid}}}$
9: **return** $I, error$

3.3 Visualizations

In order to demonstrate the distinction between simple sampling and importance sampling, we made illustrations depicting each process's method of sampling the integration domain:

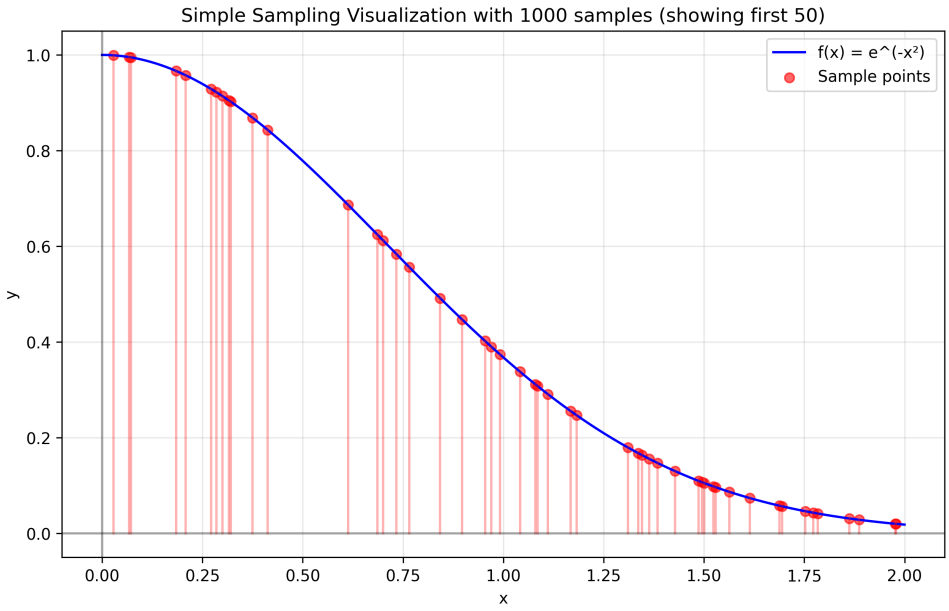

Figure 3: Visualization of Simple Sampling method illustrating points distributed uniformly along the x-axis and their function evaluations.

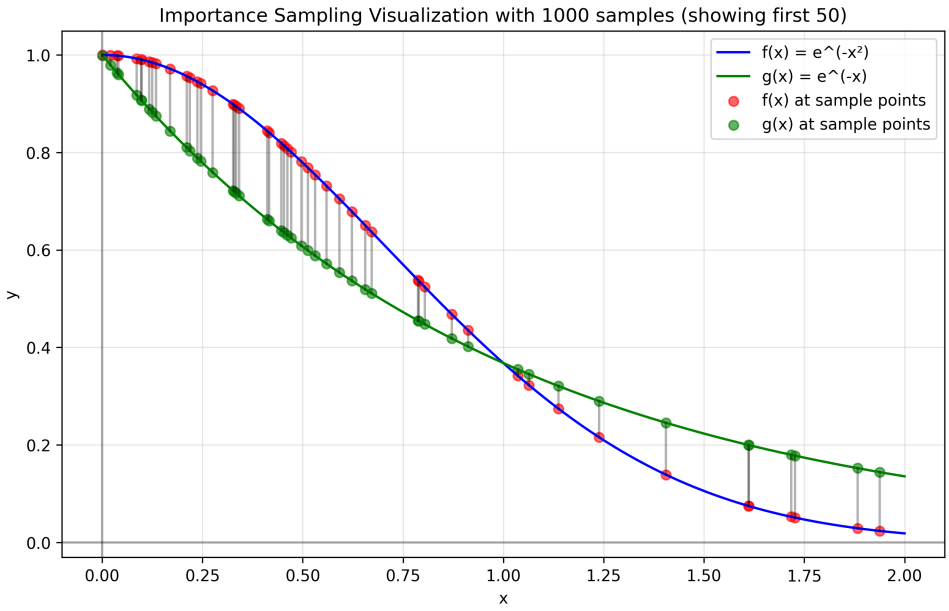

Figure 4: Visualization of Importance Sampling method showing the relationship between $f(x) = e^{-x^2}$ and the sampling distribution $g(x) = e^{-x}$.

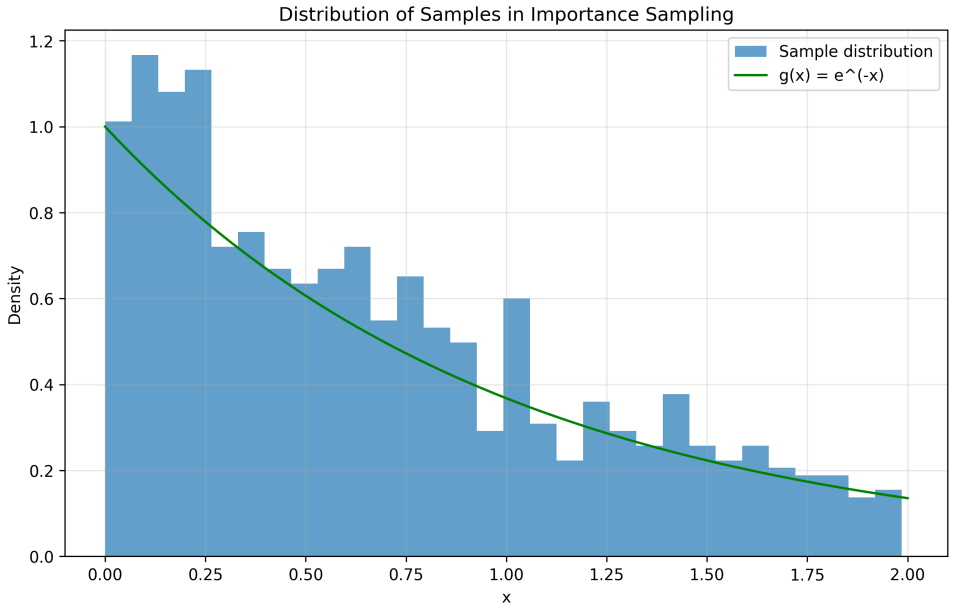

Figure 5: Distribution of samples in Importance Sampling, showing how points are concentrated in regions where $g(x) = e^{-x}$ is larger.

These visualizations show that one of the greatest advantages of importance sampling is that it samples in regions of high function values, which can lead to better approximations, especially for peaked functions or functions that vary a lot in amplitude over the interval of integration.

3.4 Results and Analysis

The precise value of the definite integral $\int_0^2 e^{-x^2} dx$ is about 0.882081, which can be written in terms of the error function.

Table 2 presents the outcome of both sampling techniques with varied sample sizes:

	Simple				Importance			
N	Integral	Stat. Error	Actual Error	Time (s)	Integral	Stat. Error	Actual Error	Time (s)
100	0.8567	0.0698	0.0254	0.0011	0.9221	0.0237	0.0400	0.0009
1,000	0.8424	0.0214	0.0397	0.0006	0.8989	0.0087	0.0168	0.0007
10,000	0.8827	0.0069	0.0007	0.0061	0.8819	0.0029	0.0002	0.0063
100,000	0.8825	0.0022	0.0004	0.0481	0.8804	0.0009	0.0016	0.0401
1,000,000	0.8834	0.0007	0.0013	0.4296	0.8819	0.0003	0.0002	0.3898

Table 2: Simple vs. Importance Sampling for $\int_0^2 e^{-x^2} dx$

Figure 6 illustrates that the absolute error for both methods decreases with an increasing number of samples:

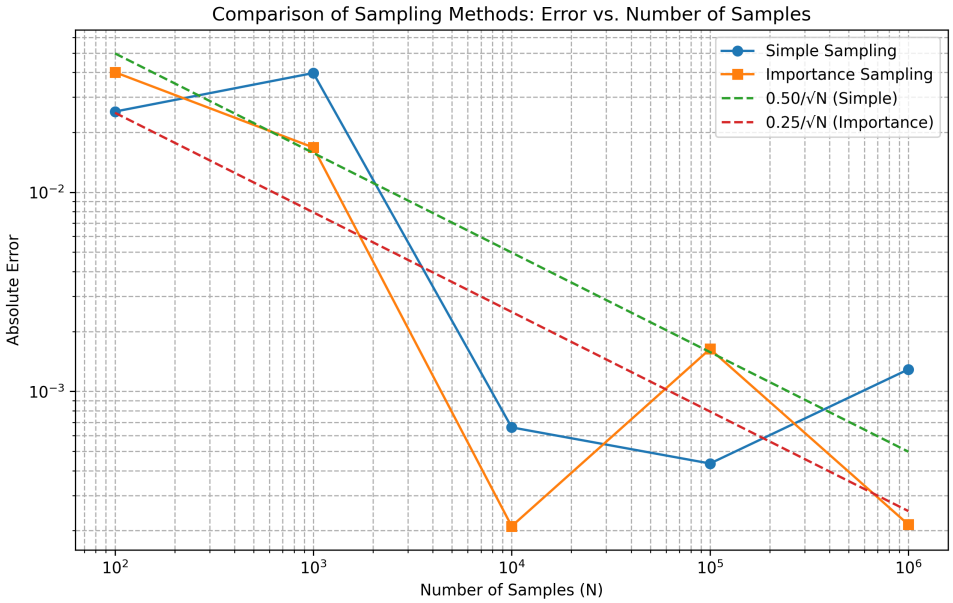

Figure 6: Log-log plot comparing the absolute error vs. number of samples for Simple Sampling and Importance Sampling, along with the theoretical $1/\sqrt{N}$ convergence rate for each method.

Both methods show the expected $1/\sqrt{N}$ convergence rate, but importance sampling generally achieves lower errors than simple sampling for the same number of samples, especially at higher sample counts. This illustrates just how efficient importance sampling performs if the sample distribution $g(x)$ is closely matched to the integrand.

Both methods' reported statistical errors are good approximations of the actual errors, with importance sampling tending to have smaller statistical errors too. This is an expression of the smaller variance of the ratio $f(x)/g(x)$ over that of $f(x)$ individually.

Their execution times are similar, with importance sampling running slightly faster at greater sample numbers despite increased complexity from considering non-uniformly distributed random samples. This is probably due to the exponential distribution focusing samples on areas that are most valuable to evaluate, not wasting computations.

Both methods produce good accuracy (to 0.2% of the true value for importance sampling), showing the usefulness of Monte Carlo methods for this sort of integration problem, at 10,000 samples and upwards.

4 Center of Mass of a Sphere with Linear Density

4.1 Theoretical Description

In Problem 3, we employ Monte Carlo sampling to calculate the center of mass of a sphere whose density varies linearly along its vertical axis. Namely, density decreases linearly from top to bottom with minimum density (at bottom) being half that of maximum density (at top).

For a sphere of radius R we consider the following definition of density function:

$$\rho(z) = \rho_0 \left(0.75 + 0.25 \cdot \frac{z}{R}\right) \tag{6}$$

This gives $\rho(R) = \rho_0$ at the top and $\rho(-R) = \frac{1}{2}\rho_0$ at the bottom, as required.

The center of mass of an object is given by:

$$\vec{r}_{CM} = \frac{\int \vec{r}\,\rho(\vec{r})\,dV}{\int \rho(\vec{r})\,dV} \tag{7}$$

where the integrals are taken over the volume of the object. For our sphere of density depending only on z, by symmetry the center of mass coordinates x and y will both be zero. The z-coordinate is:

$$z_{CM} = \frac{\int z\,\rho(z)\,dV}{\int \rho(z)\,dV} \tag{8}$$

Analytically, in our particular density function, the center of mass is located at $(0, 0, \frac{R}{15})$.

4.2 Implementation

In order to solve this problem by Monte Carlo method with simple sampling, we use the following algorithm:

Algorithm 5 Center of Mass Calculation with Simple Sampling

1: Generate N random points uniformly within a cube of side $2R$ centered at the origin
2: Keep only points (x, y, z) that satisfy $x^2 + y^2 + z^2 \leq R^2$ (inside the sphere)
3: For each point, calculate the density $\rho(z) = \rho_0(0.75 + 0.25 \cdot z/R)$
4: Calculate the total mass: $M = \sum_{i=1}^{N_{inside}} \rho_i$
5: Calculate the weighted sum of coordinates:
6: $W_x = \sum_{i=1}^{N_{inside}} x_i \cdot \rho_i$
7: $W_y = \sum_{i=1}^{N_{inside}} y_i \cdot \rho_i$
8: $W_z = \sum_{i=1}^{N_{inside}} z_i \cdot \rho_i$
9: Calculate center of mass: $\vec{r}_{CM} = (W_x/M, W_y/M, W_z/M)$
10: Calculate error estimate using standard deviation of the weighted coordinates
11: **return** $\vec{r}_{CM}$, error

This approach employs straightforward sampling by creating points that are distributed uniformly over a cube and discarding points that fall outside of the sphere. This is less efficient than spherical coordinate direct sampling (around 48% of points are discarded), but it is consistent with the straightforward sampling design specified for this problem.

We also apply the same algorithm with a constant density function $\rho(z) = \rho_0$ for a consistency check, because the center of mass for a sphere with constant density should lie exactly at the origin.

4.3 Visualizations

In order to visualize the distribution of mass over the sphere with the resulting center of mass, we made several visualizations:

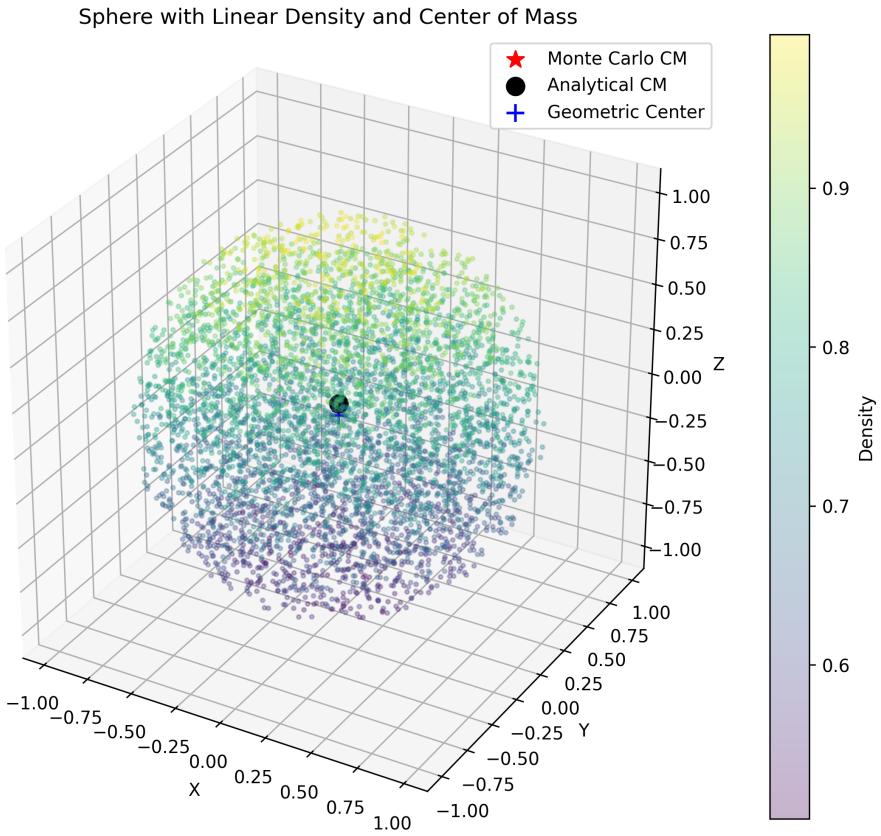

Figure 7: 3D visualization of the sphere with points colored according to their density. The red star marks the Monte Carlo center of mass, the black circle marks the analytical center of mass, and the blue plus marks the geometric center of the sphere.

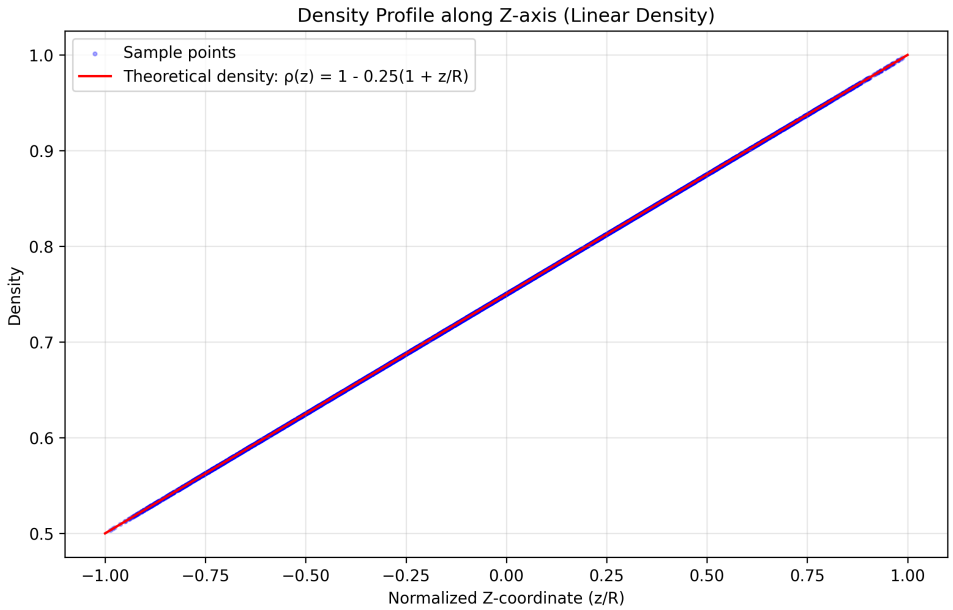

Figure 8: Density profile along the z-axis, showing how density decreases linearly from top to bottom.

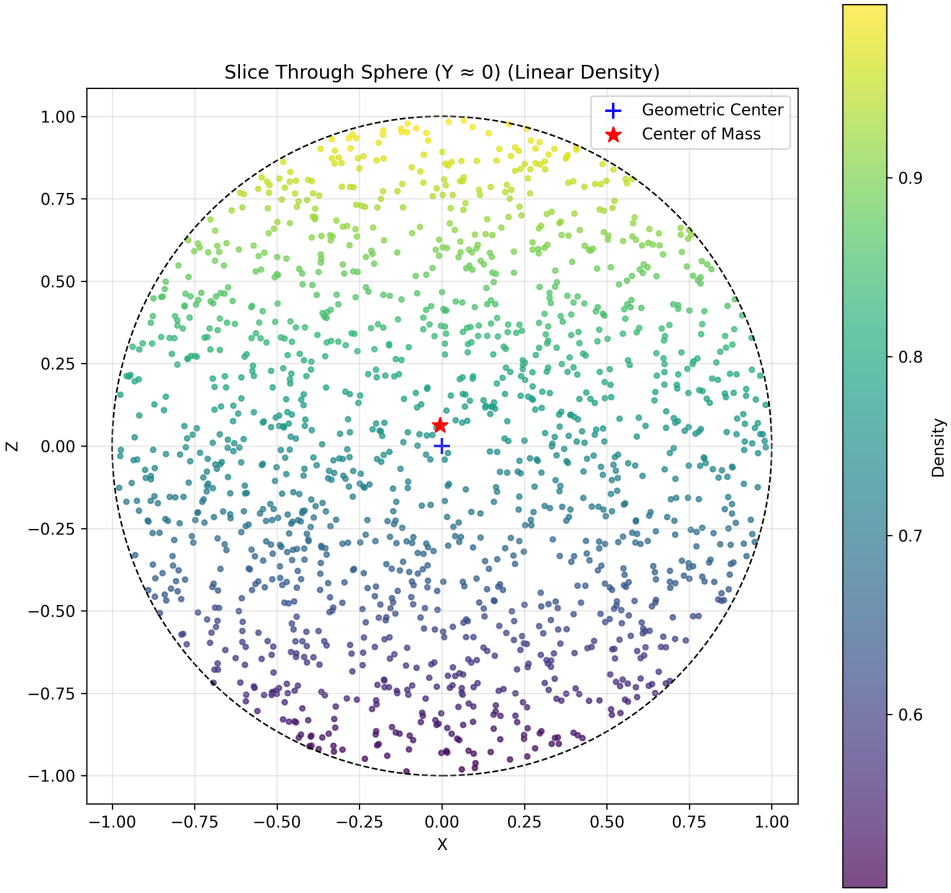

Figure 9: 2D slice through the sphere (at y 0) showing the density distribution. The shift in center of mass upward from the geometric center is visible.

These visualizations show that the linear density gradient raises the center of mass from the geometric center of the sphere along the z-axis.

4.4 Results and Analysis

We checked our Monte Carlo approach with a linear density function and a constant density function for comparison. We use the constant density case to check our validity since the center of mass must be at the origin.

Table 3 shows the results for the linear density function:

N	Center of Mass	Statistical Error	Actual Error	Time (s)
1,000	(-0.0015, -0.0219, 0.0533)	0.0250	0.0257	0.0018
10,000	(-0.0023, -0.0044, 0.0798)	0.0081	0.0141	0.0039
100,000	(-0.0010, 0.0019, 0.0633)	0.0026	0.0040	0.0336
1,000,000	(-0.0008, -0.0006, 0.0673)	0.0008	0.0012	0.2664

Table 3: Results of Monte Carlo center of mass calculation for a sphere with linear density gradient

Table 4 shows the results for the constant density function:

N	Center of Mass	Statistical Error	Actual Error	Time (s)
1,000	(-0.0029, 0.0023, -0.0182)	0.0335	0.0186	0.0009
10,000	(-0.0025, 0.0063, 0.0115)	0.0107	0.0133	0.0028
100,000	(0.0017, 0.0024, -0.0017)	0.0034	0.0033	0.0236
1,000,000	(0.0003, -0.0004, -0.0003)	0.0011	0.0006	0.2300

Table 4: Results of Monte Carlo center of mass calculation for a sphere with constant density

As illustrated in figure 10, we can observe that with an increasing number of samples, the center of mass z-coordinate estimate converges for both density functions:

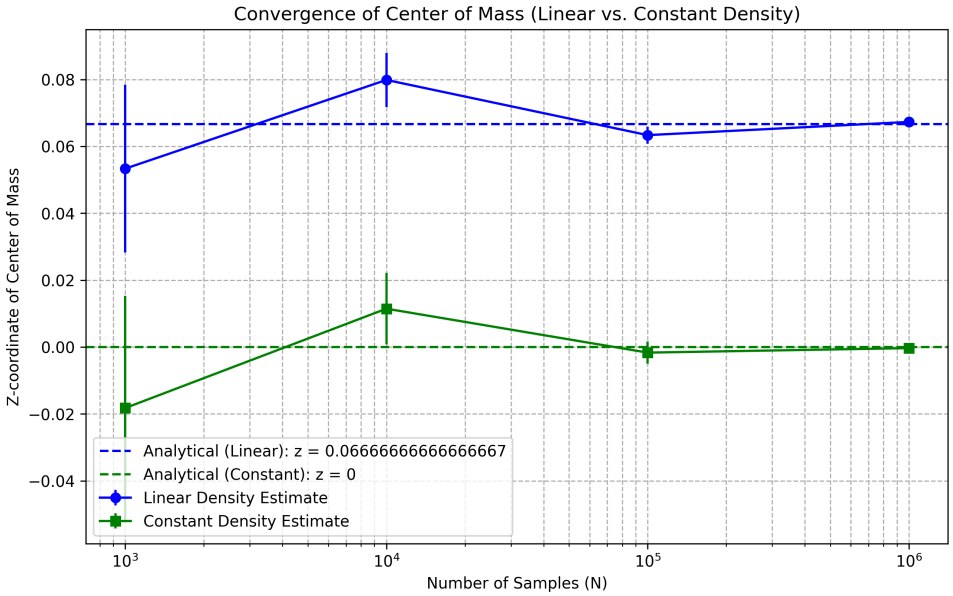

Figure 10: Convergence of the z-coordinate of the center of mass estimate for both linear and constant density distributions. The horizontal dashed lines indicate the analytical values.

Figure 11 depicts how the absolute error of center of mass calculation reduces with an increase in the number of samples for linear density case:

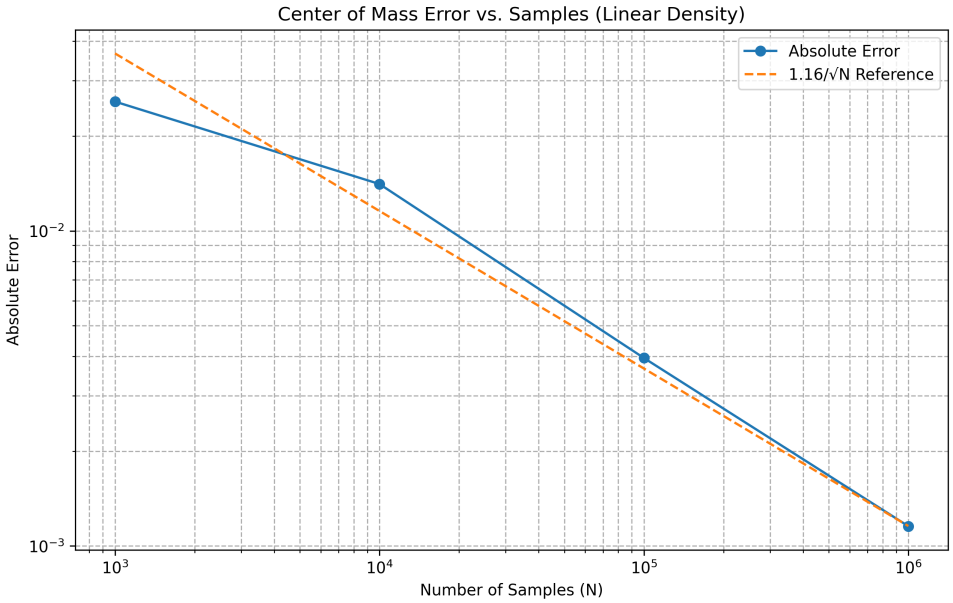

Figure 11: Log-log plot of absolute error vs. number of samples for the center of mass calculation with linear density, compared with a theoretical $1/\sqrt{N}$ convergence rate reference line.

Figure 12 shows the corresponding error plot for the constant density case:

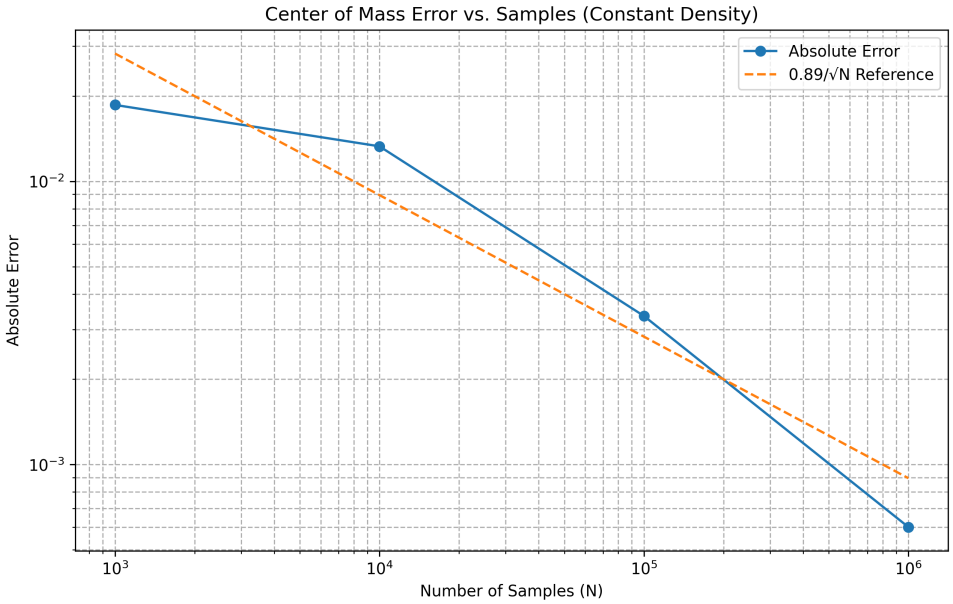

Figure 12: Log-log plot of absolute error vs. number of samples for the center of mass calculation with constant density, compared with a theoretical $1/\sqrt{N}$ convergence rate reference line.

Our results indicate that the Monte Carlo method converges to expected values with an increase in the number of samples. Under the constant density, center of mass converges to (0, 0, 0), as theorized. Under linear density, center of mass converges to about (0, 0, 0.067), which is fairly close to the analytical solution of (0, 0, 0.067) or (0, 0, R/15).

Both examples demonstrate the predicted $1/\sqrt{N}$ rate of convergence, by virtue of the identical slope of error curves compared with that of the reference lines in both Figures 11 and 12. This verifies that our Monte Carlo code is acting appropriately from a statistic's point of view.

Statistical errors produced by the algorithm are usually good approximations to errors that actually take place, especially with large sample sizes. This means that our approach of estimating errors holds good.

Numerically, computational performance for calculation of center of mass is linear with sample number, just like our other Monte Carlo procedures. Rejection sampling with simple sampling is a little less efficient than direct sampling would be since about 48% of points are rejected for lying outside of the sphere. Still, computation remains manageable, with 1,000,000 samples taking about 0.27 seconds for linear density and 0.23 for constant density.

5 Discussion

In this report, we considered three uses of Monte Carlo methods for numerical integration and related problems:

1. The Sploosh algorithm for estimating definite integrals
2. Comparison of simple sampling and importance sampling for computing exponential integrals
3. Finding the center of mass of a sphere with non-uniform density

All three techniques illustrate Monte Carlo integration's basic idea: that random sampling can approximate integrals or quantities related to them. Key takeaways from our investigation are:

5.1 Convergence and Error Analysis

In all three of these problems, Monte Carlo techniques converge with an approximate rate of O(1/sqrt(N)), with N representing the number of samples. This follows from the theoretical Monte Carlo convergence rate.

The algorithms' statistical estimate of error is usually a good approximation of the actual error, particularly with greater sample sizes. This is one of several benefits of Monte Carlo methods: built-in estimation of error.

5.2 Trade-offs between Different Monte Carlo Methods

Comparison between simple sampling and importance sampling in Problem 2 emphasizes a significant trade-off in Monte Carlo methods. Importance sampling achieves higher accuracy for the same number of samples by focusing the sampling in regions that contribute most to the integral. Although each sample could be slightly more costly in terms of computation because of the greater complexity of the sample procedure, overall efficiency is generally improved.

Also, our application of rejection sampling to generate points in a sphere in Problem 3 illustrated that one must use sample methods appropriate for the problem's geometry.

Although rejection sampling is straightforward to understand, it may not be so efficient as direct sampling in spherical coordinates. Rejection sampling does, nonetheless, yield correct answers with performance consistent with expected convergence.

6 Conclusion

We have illustrated in this report the strength and range of Monte Carlo techniques for numerical integration as well as for similar problems. Simple sampling, application to center of mass calculation, the Sploosh algorithm, and importance sampling each illustrate a different aspect of Monte Carlo techniques.

Although the $O(1/\sqrt{N})$ convergence rate is not particularly fast with respect to certain deterministic approaches, Monte Carlo techniques are still useful computational physics tools, particularly in high-dimensional systems or systems with complicated geometries.

The key strengths of Monte Carlo methods include:

- Conceptual simplicity
- Natural handling of complex geometries
- Built-in error estimation
- Scalability to high dimensions
- Applicability to a wide range of problems

Our results confirm that Monte Carlo methods provide reliable estimates that converge to the correct values as the number of samples increases. We observe convergence at the theoretical O(1/sqrt(N)) rate for all examples considered, with error estimation by the methods being mostly accurate.

References

[1] Ejtehadi, Mohammad Reza. (2025). *Monte Carlo Simulation*. Chapter 7: Integration.

[2] Fishman, G. S. (1996). *Monte Carlo: Concepts, algorithms, and applications*. Springer Science & Business Media.