Monte-Carlo and Gauss-Legendre Numerical Integration Methods for 1- to 3-D Functions

Jiayi Lai, Dorothy Ma, Marc-Antoine Nadeau, Haochen Xu
April 12th, 2024

***Abstract*—This study assesses the efficacy of numerical integration methods, particularly Monte-Carlo and Gauss-Legendre Quadrature, for computing integrals across n-dimensional functions. Numerical integration faces challenges, notably with increasing dimensions due to the Curse of Dimensionality, which demands exponentially more computation points. This paper compares Monte-Carlo, known for its robustness in higher dimensions and suitability for parallel computation, with Gauss-Legendre Quadrature, which provides consistent accuracy in lower dimensions but struggles with scalability. The analysis highlights the trade-offs between computational efficiency and accuracy, offering insights into choosing the appropriate method depending on the complexity and dimensionality of the function. Our findings enhance understanding of optimal numerical integration techniques within computational constraints.**

I. INTRODUCTION

INTEGRATION involves the summation of infinite points over a provided interval. Algebraic integration and series may be able to provide a closed form solution but only for a select number of functions and become more difficult with increasing number of dimensions. Numerical methods for integration are applicable to nearly all functions by approximating the integral into a discretized finite sum.

The main problem that lies with numerical integration is its unfavourable scaling to higher dimensions. As the method relies on sampling the function at sufficient and discrete points, the number of required points increases exponentially, as described by the Curse of Dimensionality. For example, let each node be a sampling point. N number of nodes in one dimension would scale to N^D nodes for D dimensions. Considering the functions of interest for numerical integration often involve complex computations, the cost of evaluating the function at N^D points should be avoided.

Mitigating the number of evaluations required lies at the heart of the numerical integration algorithms being developed. Among the popular methods, this paper focuses and compares the efficiency for the Monte-Carlo and Gauss-Legendre Quadrature Methods from one to three dimensions, as well as mentioning potential scaling to more dimensions using Sparse Grids.

II. MONTE-CARLO ALGORITHM

A. Background

The random dot method is a subset of Monte Carlo integration techniques, fundamentally reliant on the probabilistic interpretation of integrals as expected values. For an n-dimensional integral, the method approximates the volume under the surface defined by a function $f(x_1, x_2, ..., x_n)$ over a domain $\mathbf{\Omega}$ in an n-dimensional space.

B. Formula

The first step involves specifying the integration domain $\mathbf{\Omega}$, which is essential for defining the bounds within which random points will be generated. This domain is defined by its lower and upper bounds across each dimension, denoted respectively as $a = (a_1, a_2, ..., a_n)$ and $b = (b_1, b_2, ..., b_n)$, The volume V of this domain is computed through the product of the differences between these bounds across all dimensions, mathematically represented as:

$$V(\mathbf{\Omega}) = \prod_{i=1}^{N} (b_i - a_i) \quad (1)$$

A set of points $\{x_i\}$ is randomly generated within $\mathbf{\Omega}$, where each point represents a vector in the n-dimensional space.ensuring each component $x_{i,j}$, ensuring each component is uniformly sampled between a_j and b_j, the function f is then evaluated at each point x_i, with results summed to:

$$S = \sum_{i=1}^{N} f(x_i) \quad (2)$$

The average value $\overline{f}$ of the function over the sampled points is calculated by dividing the total sum of function evaluations by the number of points N. This average represents the mean value of f across the domain $\mathbf{\Omega}$.

$$\overline{f} = \frac{S}{N} \quad (3)$$

To approximate the integral I over $\mathbf{\Omega}$, the average $\overline{f}$ is scaled by the domain's volume:

$$I = V(\mathbf{\Omega}) * \overline{f} \quad (4)$$

Combine the Equation.1, Equation.2, Equation.3, Equation. 4 can be written to:

$$I = \frac{1}{N} * \prod_{i=1}^{N} (b_i - a_i) * \sum_{i=1}^{N} f(x_i) \quad (5)$$

Where $V(\mathbf{\Omega})$ is the volume of the domain, N is the number of random points $f(x_i)$, and are the function values at these points.

C. Algorithm

Algorithm 1 Monte Carlo Integration

```
function MONTE-CARLO-INTEGRATION(f, a, b, N)
    sum ← 0                                   ▷ Initialize the sum
    for i = 1 to N do
        x ← a + (b − a) × rand()              ▷ Random point x in range [a, b]
        fx ← f(x)                             ▷ Evaluate the function f at point x
        sum ← sum + fx                        ▷ Add the evaluation result to sum
    end for
    volume ← b − a                            ▷ Compute the volume of the domain as (b − a)
    average ← sum / N                         ▷ Compute the average of sum by dividing sum by N
    result ← volumes × average                ▷ Scale average to match domain
    return result
end function
```

III. GAUSS-LEGENDRE QUADRATURE

A. Background

Gaussian Quadrature methods of integration involve finding specific nodes of the function and assigning them a weight derived from the basis of orthogonal polynomials. These polynomials are defined as: $p \in \mathbb{R}[x; 2n+1]$ with weight functions $w(x)$, there exists a set of points x_i and weights w_i where $k = 1,\ldots,n$ such that

$$\int_a^b p(x)w(x)dx = \sum_{i=0}^{n} p(x_i)w_i \quad (6.1)$$

From these polynomials, it is possible to approximate a given function $f(x)$ with x_i and w_i such that

$$\int_a^b f(x)w(x)dx \approx \sum_{i=0}^{n} f(x_i)w_i \quad (6.2)$$

We define $g(x) = f(x)/w(x)$ so that we can write

$$\int_{-1}^{1} f(x)dx = \int_{-1}^{1} g(x)w(x)dx \approx \sum_{i=0}^{n} f(x_i)w_i \quad (7)$$

This definition can be expanded to multivariate quadrature. For instance, the 2-dimensional case, we let $g(x,y) = h(x,y)/(w(x)w(y))$ and we obtain

$$\int_{-1}^{1}\int_{-1}^{1} h(x,y)dxdy = \int_{-1}^{1}\int_{-1}^{1} g(x,y)w(x)w(y)\,dx\,dy \approx \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j g(x_i, y_j) \quad (8)$$

Moreover, we can expand this definition to arbitrary domain $[a, b]$, by rescaling the nodes and weights as such:

$$x_{scaled} = (b-a) * x/2 + (a+b)/2$$
$$w_{scaled} = (b-a) * x/2$$

B. Legendre Polynomials

By definition, any orthogonal polynomial satisfies the three-term recurrence relation where α_k and β_k where $k = 1,\ldots,n$

$$u_0 = 1$$
$$u_1 = x - \alpha_1$$
$$u_{k+1} = (x - \alpha_k)u_k - \beta_k u_{k-1}$$

In the case of Legendre polynomials, the coefficients are given by:

$$\alpha_k = 0, \qquad \beta_k = k^2/(4k^2 - 1)$$

Legendre polynomials are also favourable since their weight function on the open interval (-1, 1) is simply:

$$w(x) = 1.$$

C. Golub-Welsch Algorithm

By substituting the u in the recurrence relationship by the orthonormal polynomials, a system of equations with the Jacobi matrix can be obtained:

$$J = \begin{bmatrix} \alpha_1 & \sqrt{\beta_1} & 0 & \dots & 0 \\ \sqrt{\beta_1} & \alpha_2 & \sqrt{\beta_2} & \dots & 0 \\ 0 & \sqrt{\beta_2} & \alpha_3 & \ddots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \dots & & & \sqrt{\beta_{n-1}} \\ 0 & \dots & & \sqrt{\beta_{n-1}} & \alpha_n \end{bmatrix}$$

By the Golub-Welsch algorithm, the eigenvalues of the matrix are the nodes x_i for the Gaussian quadrature, and the weights are derived from the eigenvectors of the matrix. The weights for the Legendre polynomial is given by the multiplication between the first entry of the ith eigenvector and the measure of the weight function:

$$w_i = u_w(\mathbb{R}) v_{i,0}^{\ 2}$$

Since the points are taken from [-1, 1] and scaled to the desired interval, the measure of the weight for Legendre polynomial reduces to

$$u_{wl} = \int_{-\infty}^{\infty} w_l(x)dx = \int_{-1}^{1} 1dx = 2\,.$$

To change the interval of integration, say to [a, b], the x must be rescaled such that the function is still receiving inputs along the interval [-1, 1] to ensure the weights remain correct. This is achieved through a simple change of variables:

$$u = \frac{2x - b - a}{b - a}$$

$$x = \frac{b-a}{2}u + \frac{a+b}{2},\ dx = \frac{b-a}{2}du$$

The dx is taken into consideration with the weight function, since it will be constant with x and affect the weight in each dimension depending on the bounds [a_i, b_i] in the ith dimension.

D. Algorithm

Algorithm 2 Gauss-Legendre Quadrature

```
function GAUSS-LEGENDRE(n, a, b)
    β_i ← sqrt(1 / (4 − 1/i^2)) for i = 1, ..., (n − 1)        ▷ Compute β coefficients
    [V, Λ] ← eig(diag(β, 1) + diag(β, −1))                     ▷ Perform Eigen decomposition
    [x, i] ← sort(diag(Λ))                                     ▷ Extract nodes from Eigen decomposition
    w ← 2 × V(1, i)^2                                          ▷ Compute weights
    x ← ((b − a)/2) × x + (a + b)/2                            ▷ Rescale the nodes
    w ← ((b − a)/2) × w                                        ▷ Rescale the weights
    return x, w                                                ▷ Return nodes and weights
end function
```

Algorithm 3 Integration Using Gauss-Legendre Quadrature

```
function GAUSS-LEGENDRE-INTEGRATION(f, N, a, b)
    [x, w_x] ← Gauss-Legendre(N, a, b)        ▷ Compute nodes and weights
    sum ← 0                                   ▷ Initialize integral value
    for i = 1 to N do
        sum ← sum + f(x_i) × w_x(i)           ▷ Add the evaluation result to sum
    end for
    return sum
end function
```

The Gauss-Legendre Quadrature method is selected for this investigation because it demonstrates the selection of particular nodes and assigning weights to them, while remaining simplistic in its algorithm. This paper will focus on the Gauss-Legendre, as it shifts the focus away from the weight calculation and highlights this discrete weighted summation method.

IV. DISCUSSION

A. Functions for Numerical Experimentation

Six functions were used to test the effectiveness of the numerical integration methods as referenced in this section:

Function 1: $\frac{1}{(2\pi)^{d/2}} exp(0.5 * \sum_{i=1}^{d} - x_i^{\ 2})$

Function 2: $\prod_{i=1}^{d} |4x_i - 2|$

Function 3: $\prod_{i=1}^{d} \frac{1}{(5^{-2} + (x_i - 0.5)^2}$

Function 4: $cos(\pi + \sum_{i=1}^{d} 5x_i)$

Function 5: $\prod_{i=1}^{d} \frac{|4x_i - 2| + 5}{6}$

Function 6: $(1 + \sum_{i=1}^{d} 5x_i)^{(-d+1)}$

B. Monte-Carlo Numerical Experimentation

The absolute percentage error is calculated in comparison to the integral functions provided in MatLab. The percentage error seems to generally decrease as the number of dimensions increases for certain equations, as seen in *Fig. 1*. This may appear counterintuitive at first glance since we expect higher-dimensional integrals to be more complex and, hence, to have larger errors.

The decreasing error with higher dimensions in specific equations might be indicative of the effectiveness of the Monte Carlo method in those cases or the nature of the functions integrated. The

general rule is that the error of a Monte Carlo integration is proportional to $1/\sqrt{N}$, where N is the number of samples.

If the observed trend is consistent across multiple runs and not a result of random chance, it may suggest that variance reduction techniques are being effectively applied or that the equations are particularly well-suited for Monte Carlo methods in higher dimensions.

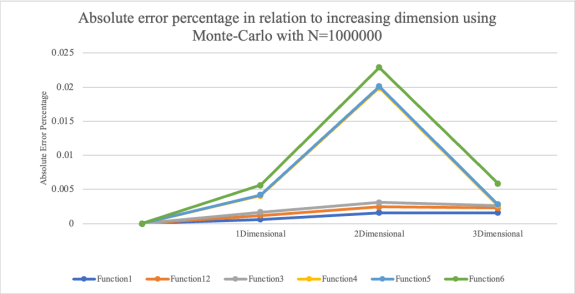

Fig. 1. Absolute error percentage for 1-3 dimensions using Monte-Carlo, N=1000000.

Based on *Fig. 2*. The simulation time generally increases with the number of dimensions, which is expected due to the increased complexity and number of computations required.

The increase in computation time with dimensionality highlights the need for efficient computation. Parallel computing can indeed help manage this increase by distributing the workload.

The data shows that the computational demand is manageable for the 1D and 2D cases across all equations, but begins to grow more significantly in the 3D cases. This growth underscores the need for parallel processing, especially for more complex or higher-dimensional problems.

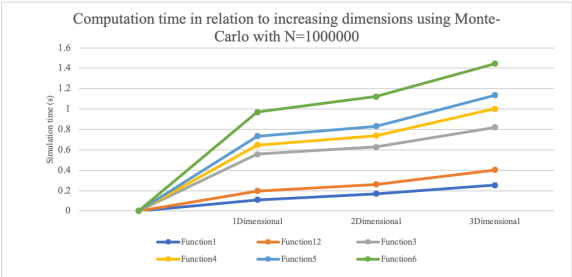

Fig. 2. Computation time for 1-3 dimensions using Monte-Carlo, N=1000000.

Key advantages of Monte Carlo are its robustness to high-dimensional problems, parallelizability, and applicability to complex domains. Unlike many traditional numerical methods, Monte Carlo integration does not suffer from the exponential increase in computational complexity as the number of dimensions increases, making it highly suitable for multi-dimensional integrals. Additionally, Each sample can be evaluated independently, which is ideal for parallel processing. This capability allows the method to efficiently utilize modern multi-core and distributed computing resources, making it scalable and effective for large-scale problems. Furthermore, the Monte Carlo method can easily handle integrals over complicated geometries where traditional methods would struggle or require complex transformations.

There are also some disadvantages of Monte Carlo in multidimensional integration, namely the slow convergence rate, sample size dependence, and inherent randomness. The method's convergence rate is proportional to $1/\sqrt{N}$, meaning that a significant increase in the number of samples is required for a modest improvement in accuracy, which can be computationally expensive. Secondly, the accuracy of Monte Carlo integration heavily relies on the number of samples; therefore, to achieve high precision, especially in complex or high-dimensional spaces, a very large number of samples may be necessary. Lastly, the stochastic nature of the method means that the results can exhibit variability between different runs with the same number of samples, which can be a disadvantage when consistency and deterministic results are required.

C. Numerical Experimentation using Gauss-Legendre Quadrature

The absolute percent error of the integration using Gauss-Legendre is shown in *Fig. 3*. The error appears to have significant dependence on the function being evaluated itself, as the error fluctuates between orders of 1 to 10 magnitude for the 6 functions examined. This could be due to the nodes being more numerically stable for certain functions as they are being evaluated. In this instance, Function 4 has the cosine, which would have bounded the function outputs between -1 and 1, and reduce the numerical difference errors when summing. For all functions except 4, the error increases with increasing dimensions, as expected with this systematic

summation approach. More dimensions result in more points, and more required arithmetics that could propagate errors.

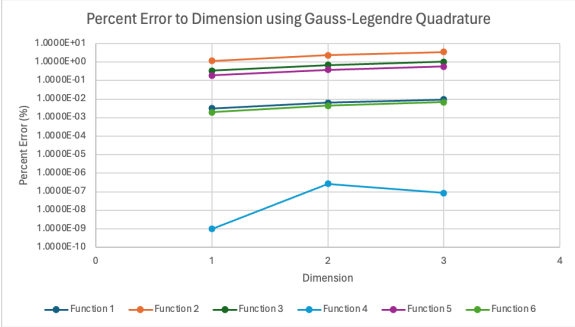

Fig. 3. Absolute error percentage for 1-3 dimensions using Gauss-Legendre, N = 8^3 = 512.

The computational time also increases with increasing dimension, with a similar trend across all functions, depicted in *Fig. 4*. The computation time increases exponentially with dimension, as expected with the algorithm. The number of nodes is fixed in each dimension, and would increase exponentially by the Curse of Dimensionality. In this instance, only 8 nodes were used up to 3 dimensions, resulting in manageably low computation times.

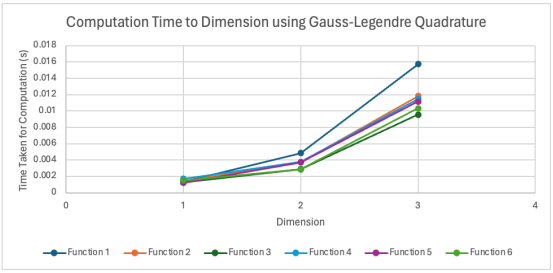

Fig. 4. Computation time for 1-3 dimensions using Gauss-Legendre Quadrature, N = 8.

The use of the Legendre polynomials means that the integration is effective as long as the function of interest can be reasonably interpolated by a polynomial at the nodes of interest. In fact, it can integrate all $2n + 1$ degree polynomials exactly, when given n points [D]. Furthermore, the Jacobi matrix by the Golub-Welsch algorithm is tri-diagonal, whose eigenvalues can be found in $\Theta(n^2)$ as opposed to $\Theta(n^3)$ [E]. The algorithm is simple and has high accuracy at and low computation times for low dimensionality, as seen in the numerical experimentations.

While the selection of nodes using the Gauss-Legendre polynomial is able to reduce the number of points at which the function is evaluated, the limiting factor of this method is still the Curse of Dimensionality, as observed in *Fig. 4*. Thus, this method is limited to low dimensions.

C. Comparison

Based on the individual computations, it already appears that the Gauss-Legendre algorithm is computationally less expensive and similarly accurate for the 6 functions of interest from 1 to 3 dimensions. For a more direct comparison, functions 4 and 3 were chosen since their computation times were approximately in the middle for both algorithms from the previous section.

From *Fig. 5* and *Fig. 6*, the percent error for the Gauss-Legendre integration is much lower than that of Monte-Carlo. In fact, the error for Gauss-Legendre appears to be at its minimum at 1e-7, as it does not decrease even with increasing nodes. This indicates that the number of nodes above 1000 and 10000 is excessive for integrating functions 4 and 3 respectively using the Gauss-Legendre algorithm, important for optimizing computation efficiency. Meanwhile, the Monte-Carlo method sees errors 5 orders of magnitude greater, even with 10e8 points, demonstrating that the Gauss-Legendre is much more accurate than Monte-Carlo for a smaller number of nodes.

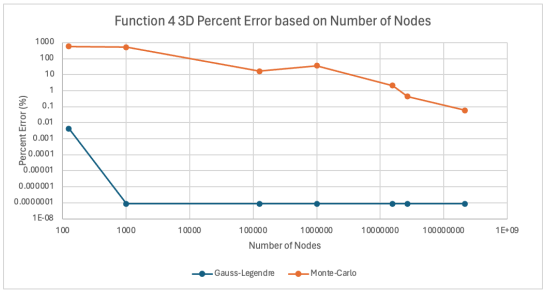

Fig. 5. Percent error of Function 4 in 3 dimensions using the two algorithms relative to increasing nodes.

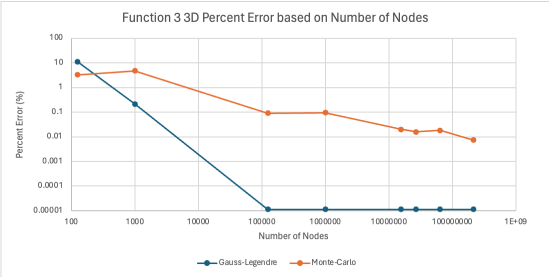

Fig. 6. Percent error of Function 3 in 3 dimensions using the two algorithms relative to increasing nodes.

From *Fig. 7* and *Fig. 8*, the Gauss-Legendre algorithm appears faster for every point in both functions. It is also significantly faster for fewer points, which, from the error analysis, was sufficient to provide an accurate estimate of the integral. However, as the number of nodes increases, the computation time increases exponentially, and the trend suggests that Monte-Carlo will be more efficient at a higher number of nodes.

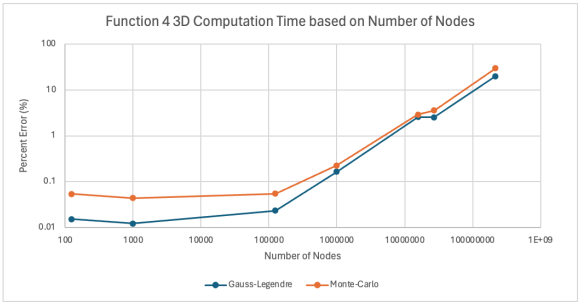

Fig. 7. Computation time of Function 4 in 3 dimensions using the two algorithms relative to increasing nodes.

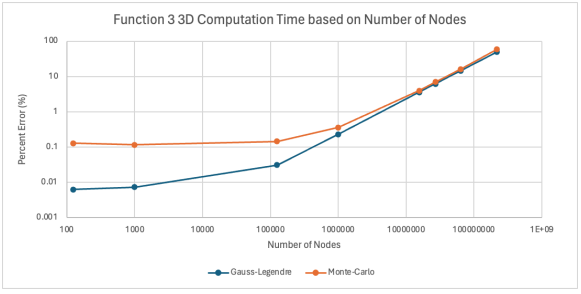

Fig. 8. Computation time of Function 3 in 3 dimensions using the two algorithms relative to increasing nodes.

Since the number of nodes grows exponentially with dimension, this implies that the Gauss-Legendre is more accurate for lower dimensions. Likewise, since Monte-Carlo is not exponentially dependent on dimension, increasing the dimensions would give it more nodes and therefore accuracy, while not exponentially increasing the computational cost like with Gauss-Legendre.

V. CONCLUSION

This investigation demonstrates that quadrature integration methods like Gauss-Legendre is favourable over Monte-Carlo for its simplistic and accuracy at small sample sizes. For the functions investigated in this paper, the Gauss-Legendre was significantly more computationally efficient and accurate. However, with increasing dimensions, the Curse of Dimensionality necessitates an increase in the number of nodes, making Monte-Carlo the optimal choice for multi-dimensional numerical integration, in particular, for those above 3 dimensions. Further optimizations for quadrature methods could be to use a subset of the N^D nodes to minimize function evaluations. Meanwhile Monte-Carlo methods could implement parallel evaluations to optimize computation power.

VI. REFERENCES

[A] P. L. Bonate, "A Brief Introduction to Monte Carlo Simulation," *Clinical Pharmacokinetics*, vol. 40, no. 1, pp. 15-22, 2001, doi: 10.2165/00003088-200140010-00002.

[B] H. Al-Sharif, “Application of Monte Carlo Integrals,” B.S. Capstone paper. Dept. Math., Georgia College and State Univ., Milledgeville, 2018.

[C] J. Geweke, “Monte Carlo Simulation and Numerical Integration.” *Handbook of computational economics*, vol. 1, pp. 731-800, 1996, doi: 10.21034/sr.192

[D] G. H. Golub and J. H. Welsch, "Calculation of Gauss Quadrature Rules," *Mathematics of Computation*, vol. 23, no. 106, pp. 221-s10, 1969, doi: 10.2307/2004418.

[E] I. S. Dhillon, "A New O(n2) Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem," Ph. D. dissertation, Comput. Sci., University of California, Berkeley, 1997.

Frist A. Jiayi Lai, research report of Monte-Carlo method.

Second B. Dorothy Ma, research report report of Gauss-Legendre Quadrature method.

Third C. Marc-Antoine Nadeau, code part of Gauss-Legendre Quadrature method.

Forth D. Haochen Xu, code part of Monte-Carlo method.