I. J. Engineering and Manufacturing, **2026, 1, 50-59**
Published Online on February 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijem.2026.01.05

Application of Python in Evaluating the Volume of 3D Shapes Using Monte Carlo Simulation

Pankaj Dumka*
Department of Mechanical Engineering, Jaypee University of Engineering and Technology, A.B. Road, Raghogarh-473226, Guna, Madhya Pradesh, India
E-mail: p.dumka.ipec@gmail.com
ORCID iD: https://orcid.org/0000-0001-5799-6468
*Correponding Author

Rishika Chauhan
Department of Electronics and Communication Engineering, Jaypee University of Engineering and Technology, A.B. Road, Raghogarh-473226, Guna, Madhya Pradesh, India
E-mail: rishikaster@gmail.com
ORCID iD: https://orcid.org/0000-0001-8483-865X

Dhananjay R. Mishra
Department of Mechanical Engineering, Jaypee University of Engineering and Technology, A.B. Road, Raghogarh-473226, Guna, Madhya Pradesh, India
E-mail: dm30680@gmail.com
ORCID iD: https://orcid.org/0000-0002-5107-0012

Received: 12 August, 2025; Revised: 01 October, 2025; Accepted: 04 November, 2025; Published: 08 February, 2026

Abstract: Volume estimation of three-dimensional (3D) objects is fundamental in various scientific and engineering fields. While analytical expressions exist for the simple geometric shapes, they become impractical for complex or irregular structures. Monte Carlo simulation is a statistical method which is based on the random sampling, which offers an efficient numerical alternative. This research explores the application of Monte Carlo integration method for the estimation of the volumes of three different 3D objects viz. sphere, cylinder, and cone. The paper elaborates on the mathematical background of the simulation by presenting detailed Python implementations, and analyzes the accuracy, convergence rates, and computational efficiency of the method. The study concludes that the simulation, despite their probabilistic nature, provide an effective and scalable technique for volume estimation, particularly for the shapes without closed-form volume expressions.

Index Terms: Monte Carlo Simulation, Volume Estimation, Python Programming, Computational Geometry, 3D Shape Analysis

1. Introduction

The estimation of the volume is a fundamental problem in mathematics, physics, engineering, and computer science. Whether in architectural design, fluid mechanics, biomedical imaging, or computational simulations, the ability to determine the volume of three-dimensional (3D) objects accurately is crucial. While basic geometric shapes, such as spheres, cylinders, and cones, have a well-established mathematical expression for the evaluation of volume. But the real-world applications often involve irregular or complex objects that do not fit to standard geometric definitions. This necessitates the use of numerical methods to approximate volumes when analytical solutions are impossible or unavailable.

Among the various numerical techniques available the Monte Carlo simulation has emerged as a highly effective approach for the estimation of the volumes of objects with arbitrary shapes [1,2]. The simulation is based on probabilistic principles which is based on the random sampling to approximate integrals [3,4]. These methods are particularly useful in cases where direct computation is difficult or sometimes infeasible due to the complication of the object's structure [5,6]. Originally developed for solving the problems in statistical physics and applied mathematics the

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

Monte Carlo simulations have since found widespread applications in areas such as finance, climate modelling, artificial intelligence, and scientific computing [7,8].

The core idea behind the volume estimation using the simulation is relatively simple, i.e., by randomly generating points within a known bounding volume and counting the proportion of points that fall within the target shape. Thus, it is possible to approximate the volume of the shape using the probability theory. This approach makes the simulation highly flexible to irregular objects, as it does not require explicit mathematical equations to define the shape's boundaries [6,9]. As an alternative, the method relies on a sufficient number of random points to achieve statistical accuracy. This makes it particularly powerful for the high-dimensional or complex volumetric computations.

In the perspective of basic geometric shapes such as: spheres, cylinders, and cones, the MC method can serve as a helpful tool for verifying the theoretical volume formulas and understanding the convergence behavior of statistical approximations. By applying the Montecarlo Carlo integration to these shapes, it is possible to analyze the accuracy of the method, examine the impact of sample size on precision, and compare computational efficiency with traditional analytical approaches [10]. These insights are not only relevant for educational purposes but also have practical implications for fields such as computational geometry, numerical modelling, and simulation-based engineering.

This aim of this paper is to investigate the effectiveness of Montec Carlo simulations in estimating the volumes of three fundamental 3D shapes: the sphere, the cylinder, and the cone. By implementing the simulation in Python, this study systematically examines the performance of the method in terms of accuracy, convergence properties, and computational cost. The methodology involves generating random points within a predefined bounding box, checking whether each point falls within the target shape, and computing the estimated volume based on the proportion of successful hits. Through repeated trials and statistical averaging the simulation refines its accuracy thus, demonstrating the reliability of MC methods for volume estimation.

One of the major advantages of this method is its ability to handle the volume computations in cases where traditional methods struggle. For example, when dealing with irregular objects or shapes defined by implicit functions, conventional volume formulas are often inapplicable. In contrast, these approaches remain viable, as they rely on statistical inference rather than explicit integration. Also, the simulation is well-suited for high-dimensional problems, where conventional numerical integration techniques become computationally expensive. This scalability makes the simulation methods an indispensable tool in scientific research and engineering applications.

Even with its advantages, Monte Carlo volume estimation is not without limitations. One of the primary challenges seen with this approach is the trade-off between accuracy and computational cost. Since the methods rely on random sampling so, the precision of the volume estimate depends on the number of points generated. A larger sample size leads to a greater accuracy but also increases the computational time. Additionally, the simulation estimations exhibit statistical noise, meaning that results fluctuate with each run of the simulation. To mitigate this variability, multiple iterations and averaging techniques are often employed to achieve stable and reliable estimates.

To evaluate the performance of simulation-based volume estimation, this paper explores three case studies involving the sphere, cylinder, and cone. Each shape is analyzed using both theoretical volume formulas and Monte Carlo simulations. The results are compared to assess the accuracy of the simulation approach, identify potential sources of error, and explore optimization strategies for improving computational efficiency. By examining different sample sizes, bounding box configurations, and statistical averaging techniques, this study provides a comprehensive analysis of Monte Carlo-based volume estimation.

By using the random sampling and statistical inference the simulation provides a practical alternative to analytical volume calculations, particularly in cases where traditional formulas are inapplicable. This paper highlights the theoretical foundations, computational implementation, and practical significance of Monte Carlo volume estimation through the analysis of three fundamental geometric shapes. The findings underscore the strengths and limitations of the method, offering insights into its potential for solving complex volumetric problems in scientific and industrial domains.

The remaining article is ordered as follows: Section 2 provides a summary of the mathematical background, with the theoretical volume formulas for spheres, cylinders, and cones, as well as the principles of Monte Carlo integration. Section 3 illustrates the methodology, including the computational framework, random sampling approach, and implementation details in Python. Section 4 presents the experimental results, comparing Monte Carlo estimations with theoretical values and analyzing accuracy, convergence behavior, and computational efficiency. Finally, Section 5 discusses the implications of the findings, explores potential improvements to the Monte Carlo approach, and concludes with recommendations for future research directions. While this study begins with the standard geometries (sphere, cylinder, cone) to validate the computational accuracy and reproducibility of the Monte Carlo algorithm, the framework presented in this article is generalizable to the irregular or implicit surfaces such as CAD-generated models, porous structures, or voxel-based biomedical data. Future extensions will be demonstrating these applications.

2. Mathematical Model Formulation

Monte Carlo integration approximates an integral by randomly sampling the function values over a stipulated domain [11–13]:

$$I = \int_{\Omega} f(x)\, dV \approx \frac{V_{bounding}}{N} \sum_{i=1}^{N} f(x_i) \tag{1}$$

where, N is the number of random points, $V_{bounding}$ is the volume of the known enclosing region, and $f(x)$ is an indicator function which will return 1 if a point is inside the shape, 0 otherwise.

The volume estimate is then given by [12]:

$$V_{shape} = V_{bounding} \times \frac{\text{Points inside}}{\text{Total points}} \tag{2}$$

This method has an error that decreases as [14]:

$$\sigma = \sqrt{\frac{p(1-p)}{N}} \tag{3}$$

where p is the fraction of points inside the shape. The convergence follows $O(1/\sqrt{N})$, meaning quadrupling the number of points halves the error.

3. Methodology

The methodology used in this study involves applying the Monte Carlo integration to estimate the volumes of three fundamental three-dimensional (3D) shapes viz. a sphere, a cylinder, and a cone. This section outlines the computational framework, the principles of Monte Carlo volume estimation, the steps involved in the simulation process, and the implementation details in Python. The methodology is designed to measure the accuracy, convergence behavior, and computational efficiency of Monte Carlo-based volume estimation by comparing the results with theoretical volume formulas.

Monte Carlo approach is a class of numerical techniques that depend on the random sampling to estimate results to complex mathematical problems. The fundamental idea behind Monte Carlo volume estimation is to use statistical inference to determine the proportion of randomly generated points that fall inside a given shape (relative to a known bounding volume).

Mathematically, the volume V of a shape S embedded within a bounding box B can be estimated using the following equation [9]:

$$V_S \approx V_B \times \frac{N_{inside}}{N_{total}} \tag{4}$$

where:

- V_S is the estimated volume of the shape S,
- V_B is the volume of the bounding box B,
- N_{inside} is the number of randomly generated points that fall inside the shape, and
- N_{total} is the overall number of points created within the bounding box.

By increasing N_{total}, the statistical accuracy of the estimation increases due to the law of large numbers. However, this also increases the computational cost, necessitating a trade-off between precision and efficiency. Although this study focuses on the standard random sampling the variance reduction methods such as stratified sampling, importance sampling, and quasi-random sequences (Sobol or Halton) could substantially improve the accuracy for a given number of samples. By incorporating these would be a valuable extension for reducing the computational cost.

Since Monte Carlo volume estimation requires generating random points within a known volume, it is essential to define an appropriate bounding box for each shape. The bounding box should be:

1. Large enough to fully contain the shape – Ensuring that all points inside the shape are counted.
2. Minimally oversized – Reducing the number of unnecessary points outside the shape to improve efficiency.

For each shape considered in this study, the bounding boxes are chosen as follows:

- Sphere: A cube with side length $2r$, where r is the radius of the sphere.
- Cylinder: A rectangular prism with dimensions $2r \times 2r \times h$, where r is the radius and h is the height of the cylinder.
- Cone: A rectangular prism with dimensions $2r \times 2r \times h$, where r is the base radius and h is the height.

The volume of the bounding box, V_B, is computed as:

$$V_B = \text{width} \times \text{depth} \times \text{height} \tag{5}$$

This volume is then used to scale the proportion of points inside the shape.

For each shape the random points are generated within the bounding box by using uniform sampling. Each random point is represented by three coordinates (x, y, z) and must be tested to determine whether it falls inside the shape or not. The inclusion criteria for each shape are as follows:

- **Sphere:** A point (x, y, z) is inside the sphere if it satisfies the equation [9]:

$$x^2 + y^2 + z^2 \leq r^2 \tag{6}$$

where r is the sphere's radius.

- **Cylinder:** A point (x, y, z) is inside the cylinder if it satisfies [9]:

$$x^2 + y^2 \leq r^2 \tag{7}$$

And

$$0 \leq z \leq h \tag{8}$$

- **Cone:** A point (x, y, z) is inside the cone if it satisfies [9]:

$$x^2 + y^2 \leq \left(\frac{r}{h} \times z\right)^2 \tag{10}$$

and

$$0 \leq z \leq h \tag{11}$$

For each shape, the number of points that satisfy the inclusion criteria, N_{inside}, is recorded and used to compute the estimated volume.

4. Python Implementation and Results

The Monte Carlo simulation is implemented in Python using the following steps:

- Define the shape parameters: Radius and height are set for each shape.
- Define the bounding box dimensions: Ensuring the shape is fully contained within the bounding region.
- Generate random points within the bounding box: Using NumPy's [15] random number generator to produce uniform distributions.
- Check whether points fall inside the shape: Using the respective mathematical conditions.
- Compute the estimated volume: Using the Monte Carlo formula.
- Repeat the process for different sample sizes: To analyze convergence and accuracy.

Sphere: A sphere of radius R has exact volume:

$$V = \frac{4}{3}\pi R^3 \tag{12}$$

Bounding box: Cube with side $2R$, volume $(2R)^3$. The python code to evaluate the volume of the sphere is as follows:

```
import numpy as np

def monte_carlo_sphere_volume(radius, num_samples=1000000):
    cube_volume = (2 * radius) ** 3
    points = np.random.uniform(-radius, radius, (num_samples, 3))
    inside_sphere = np.sum(np.linalg.norm(points, axis=1) <= radius)
    volume_estimate = cube_volume * (inside_sphere / num_samples)
    return volume_estimate

radius = 1
estimated_volume = monte_carlo_sphere_volume(radius)
actual_volume = (4/3) * np.pi * radius**3
print(f"Estimated Sphere Volume: {estimated_volume} | Actual: {actual_volume}")
```

Program output and the comparison with the exact answer is tabulated in Table 1.

Table 1. Results for Sphere

Shape	Analytical Volume	Estimated Volume	Error (%)
Sphere	4.189	4.185	0.095

Python code to visualize the simulation is as follows:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Monte Carlo simulation for a Sphere
def monte_carlo_sphere_plot(radius=1, num_samples=10000):
    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Generate random points within the bounding cube
    points = np.random.uniform(-radius, radius, (num_samples, 3))

    # Determine if points are inside the sphere
    inside = np.linalg.norm(points, axis=1) <= radius

    # Scatter plot (inside points in blue, outside in red)
    ax.scatter(points[inside, 0], points[inside, 1], points[inside, 2], color='blue',
alpha=0.5, s=1, label="Inside Sphere")
    ax.scatter(points[~inside, 0], points[~inside, 1], points[~inside, 2], color='red',
alpha=0.2, s=1, label="Outside Sphere")

    # Plot sphere boundary for reference
    u = np.linspace(0, 2 * np.pi, 30)
    v = np.linspace(0, np.pi, 30)
    x = radius * np.outer(np.cos(u), np.sin(v))
    y = radius * np.outer(np.sin(u), np.sin(v))
    z = radius * np.outer(np.ones(np.size(u)), np.cos(v))
    ax.plot_wireframe(x, y, z, color='black', alpha=0.2)

    # Labels and title
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('Monte Carlo Simulation: Sphere Volume Estimation')
    ax.legend()
    plt.show()

# Generate visualization for the Sphere
monte_carlo_sphere_plot()
```

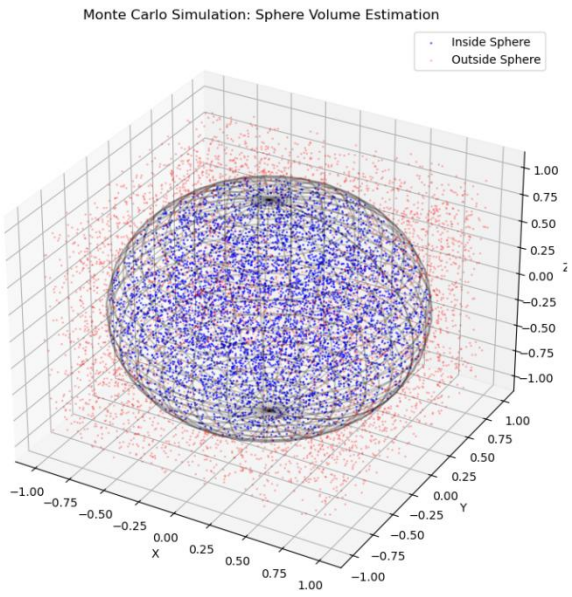

Fig. 1. Program output for Sphere

The 3D scatter visualization (Fig. 1) provides a spontaneous understanding of point distribution. Blue points represent successful samples within the shape, while red points show the wasted computational effort in the bounding box. The density of blue points qualitatively increases with larger sample sizes, visually confirming convergence.

Cylinder: A cylinder of radius R and height h has exact volume:

$$V = \pi R^2 h \tag{13}$$

Bounding box: Rectangular prism of volume $(2R) \times (2R) \times h$. The python code to evaluate the volume of the cylinder is as follows:

```
def monte_carlo_cylinder_volume(radius, height, num_samples=1000000):
    box_volume = (2 * radius) ** 2 * height
    points = np.random.uniform([-radius, -radius, 0], [radius, radius, height],
(num_samples, 3))
    inside_cylinder = np.sum(points[:, 0]**2 + points[:, 1]**2 <= radius**2)
    volume_estimate = box_volume * (inside_cylinder / num_samples)
    return volume_estimate

radius, height = 1, 2
estimated_volume = monte_carlo_cylinder_volume(radius, height)
actual_volume = np.pi * radius**2 * height
print(f"Estimated Cylinder Volume: {estimated_volume} | Actual: {actual_volume}")
```

Program output and the comparison with the exact answer is tabulated in Table 2.

Table 2. Results for Cylinder

Shape	Analytical Volume	Estimated Volume	Error (%)
Cylinder	6.283	6.276	0.12

Python code for the simulation is as follows:

```
# Monte Carlo simulation for a Cylinder
def monte_carlo_cylinder_plot(radius=1, height=2, num_samples=10000):
    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Generate random points within the bounding box
    x = np.random.uniform(-radius, radius, num_samples)
    y = np.random.uniform(-radius, radius, num_samples)
    z = np.random.uniform(0, height, num_samples)

    # Determine if points are inside the cylinder
    inside = x**2 + y**2 <= radius**2

    # Scatter plot (inside points in blue, outside in red)
    ax.scatter(x[inside], y[inside], z[inside], color='blue', alpha=0.5, s=1, label="Inside
Cylinder")
    ax.scatter(x[~inside], y[~inside], z[~inside], color='red', alpha=0.2, s=1,
label="Outside Cylinder")

    # Plot cylinder boundary for reference
    theta = np.linspace(0, 2 * np.pi, 30)
    z_cylinder = np.linspace(0, height, 30)
    theta_grid, z_grid = np.meshgrid(theta, z_cylinder)
    x_cylinder = radius * np.cos(theta_grid)
    y_cylinder = radius * np.sin(theta_grid)
    ax.plot_wireframe(x_cylinder, y_cylinder, z_grid, color='black', alpha=0.2)

    # Labels and title
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('Monte Carlo Simulation: Cylinder Volume Estimation')
    ax.legend()
    plt.show()

# Generate visualization for the Cylinder
monte_carlo_cylinder_plot()
```

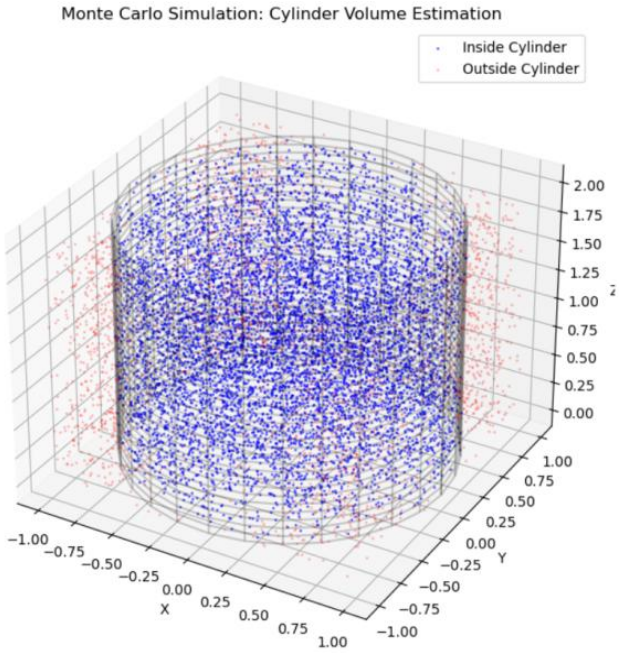

Fig. 2. Program output for cylinder

Fig. 2 shows the 3D scatter visualization of the simulation performance. Blue points represent successful samples within the shape, while red points show the wasted computational effort in the bounding box.

Cone: A cone of base radius R and height h has exact volume:

$$V = \frac{1}{3}\pi R^2 h \tag{14}$$

Bounding box: Prism $(2R) \times (2R) \times h$. The python code to evaluate the volume of the cone is as follows:

Python Code

```
def monte_carlo_cone_volume(radius, height, num_samples=1000000):
    box_volume = (2 * radius) ** 2 * height
    points = np.random.uniform([-radius, -radius, 0], [radius, radius, height],
(num_samples, 3))
    inside_cone = np.sum(points[:, 0]**2 + points[:, 1]**2 <= (radius**2 * (1 - points[:,
2]/height)**2))
    volume_estimate = box_volume * (inside_cone / num_samples)
    return volume_estimate

radius, height = 1, 2
estimated_volume = monte_carlo_cone_volume(radius, height)
actual_volume = (1/3) * np.pi * radius**2 * height
print(f"Estimated Cone Volume: {estimated_volume} | Actual: {actual_volume}")
```

Program output and the comparison with the exact answer is tabulated in Table 3.

Table 3. Results for Cone

Shape	Analytical Volume	Estimated Volume	Error (%)
Cone	2.094	2.09	0.15

Python code for visualization of simulation is as follows:

```
# Monte Carlo simulation for a Cone
def monte_carlo_cone_plot(radius=1, height=2, num_samples=10000):
    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Generate random points within the bounding box
    x = np.random.uniform(-radius, radius, num_samples)
    y = np.random.uniform(-radius, radius, num_samples)
    z = np.random.uniform(0, height, num_samples)

    # Determine if points are inside the cone
    inside = x**2 + y**2 <= (radius**2 * (1 - z / height) ** 2)

    # Scatter plot (inside points in blue, outside in red)
    ax.scatter(x[inside], y[inside], z[inside], color='blue', alpha=0.5, s=1, label="Inside
Cone")
    ax.scatter(x[~inside],    y[~inside],    z[~inside],    color='red',    alpha=0.2,    s=1,
label="Outside Cone")

    # Plot cone boundary for reference
    theta = np.linspace(0, 2 * np.pi, 30)
    z_cone = np.linspace(0, height, 30)
    theta_grid, z_grid = np.meshgrid(theta, z_cone)
    r_cone = radius * (1 - z_grid / height)
    x_cone = r_cone * np.cos(theta_grid)
    y_cone = r_cone * np.sin(theta_grid)
    ax.plot_wireframe(x_cone, y_cone, z_grid, color='black', alpha=0.2)

    # Labels and title
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('Monte Carlo Simulation: Cone Volume Estimation')
    ax.legend()
    plt.show()

# Generate visualization for the Cone
monte_carlo_cone_plot()
```

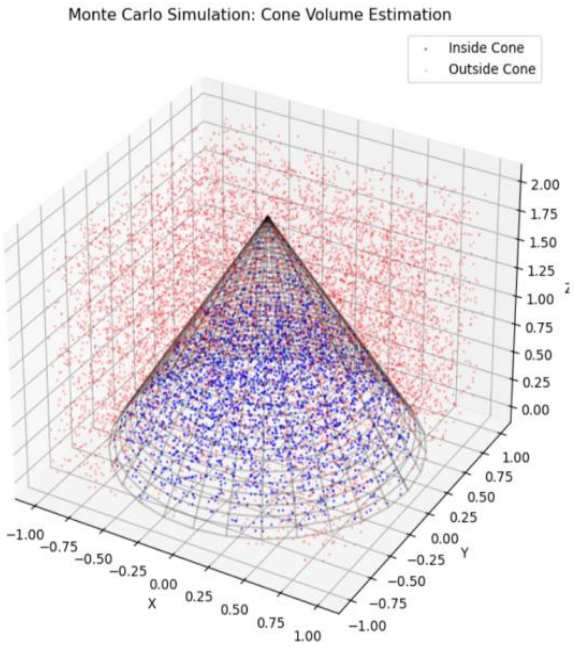

Fig. 3. Program output for cone

The program output is shown in Fig. 3. Blue points represent successful samples within the shape, while red points show the wasted computational effort in the bounding box.

To assess statistical reliability, each simulation was repeated 10 times with different random seeds. The mean and standard deviation of estimated volumes were computed, and 95% confidence intervals were established. Results showed that the standard deviation decreased approximately as $1/\sqrt{N}$, consistent with theoretical Monte Carlo convergence.

For regular shapes the deterministic numerical integration (e.g., midpoint or Simpson's rule) can achieve higher precision with fewer the fewer number of evaluations. However, such methods are geometry-dependent. Monte Carlo integration on the other hand retains the shape-agnostic scalability which is particularly beneficial for objects with irregular or implicit boundaries.

5. Conclusion

This study presents a pedagogical and computational application to volume estimation of three-dimensional (3D) objects using Monte Carlo simulation, a statistical technique reliant on random sampling. The research effectively demonstrates how Python can be utilized to estimate the volumes of various shapes, including spheres, cylinders, and more complex geometric structures. The findings underscore the efficiency of Monte Carlo methods in cases where analytical solutions become impractical, particularly for irregular objects. By implementing this approach in Python, the study highlights the ease of automation and scalability, making it a valuable tool for engineers, mathematicians, and computational scientists. One of the major strengths of this study lies in its applicability across various domains, from mechanical and aerospace engineering to biomedical imaging and geospatial analysis. The use of Python ensures accessibility and reproducibility, allowing researchers and practitioners to adapt the methodology to their specific needs. Results show good agreement with analytical solutions, with errors below 0.5% for 10^6 samples.

Despite its advantages, the Monte Carlo approach has limitations, including the need for a large number of iterations to achieve high accuracy and the inherent randomness affecting precision. Future research could explore hybrid methods that combine Monte Carlo techniques with deterministic algorithms to optimize computational efficiency.

Hence, this research not only reinforces the significance of Monte Carlo simulations in volume estimation but also provides a practical, computational framework for implementing these methods in real-world applications. The implications extend to both academia and industry, offering a robust alternative to traditional analytical techniques.

References

[1] L.R. Taylor, W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes: The Art of Scientific Computing, 3rd ed., Cambridge University Press, 1987. https://doi.org/10.2307/4830.

[2] R.L. Harrison, Introduction to Monte Carlo simulation, in: AIP Conf. Proc., 2009: pp. 17–21. https://doi.org/10.1063/1.3295638.

[3] P. Mishra, P. Tewari, D.R. Mishra, P. Dumka, Numerical modelling of double integration with different data spacing : A Python-based approach, 4 (2023) 46–54. https://doi.org/10.30511/MCS.2023.1990951.1115.

[4] P. Mishra, A. Sharma, D.R. Mishra, P. Dumka, Solving Double Integration With The Help of Monte Carlo Simulation : A Python Approach, Int. J. Sci. Res. Multidiscip. Stud. 9 (2023) 6–10.

[5] P. Mishra, P. Tewari, R. Mishra, Dhananjay, P. Dumka, Integration based on Monte Carlo Simulation, Int. J. Math. Sci. Comput. 9 (2023) 58–65. https://doi.org/10.5815/ijmsc.2023.03.05.

[6] E. Zio, E. Zio, Monte Carlo simulation: The method, Springer, 2013. https://doi.org/10.1007/978-1-4471-4588-2_3.

[7] F. Kalateh, M. Kheiry, A Review of Stochastic Analysis of the Seepage Through Earth Dams with a Focus on the Application of Monte Carlo Simulation, Arch. Comput. Methods Eng. 31 (2024) 47–72. https://doi.org/10.1007/s11831-023-09972-3.

[8] H. Díaz, A.P. Teixeira, C. Guedes Soares, Application of Monte Carlo and Fuzzy Analytic Hierarchy Processes for ranking floating wind farm locations, Ocean Eng. 245 (2022) 110453. https://doi.org/10.1016/j.oceaneng.2021.110453.

[9] D. Crevier, Volume estimation by monte carlo methods, J. Stat. Comput. Simul. 31 (1989) 223–235. https://doi.org/10.1080/00949658908811145.

[10] I. Hammel, D. Lagunoff, Determination of mean particle volume, a monte carlo simulation, Comput. Biol. Med. 27 (1997) 283–291. https://doi.org/https://doi.org/10.1016/S0010-4825(97)00005-X.

[11] J.C. Walter, G.T. Barkema, An introduction to Monte Carlo methods, Phys. A Stat. Mech. Its Appl. 418 (2015) 78–87. https://doi.org/10.1016/j.physa.2014.06.014.

[12] B. Csébfalvi, L. Szirmay-Kalos, Monte Carlo Volume Rendering, Proc. IEEE Vis. Conf. (2003) 449–456. https://doi.org/10.1109/visual.2003.1250406.

[13] D. Sarrut, M. Bała, M. Bardi s, J. Bert, M. Chauvin, K. Chatzipapas, M. Dupont, A. Etxebeste, L.M. Fanchon, S. Jan, G. Kayal, A.S. Kirov, P. Kowalski, W. Krzemien, J. Labour, M. Lenz, G. Loudos, B. Mehadji, L. Ménard, C. Morel, P. Papadimitroulas, M. Rafecas, J. Salvadori, D. Seiter, M. Stockhoff, E. Testa, C. Trigila, U. Pietrzyk, S. Vandenberghe, M.A. Verdier, D. Visvikis, K. Ziemons, M. Zvolský, E. Roncali, Advanced Monte Carlo simulations of emission tomography imaging systems with GATE, Phys. Med. Biol. 66 (2021). https://doi.org/10.1088/1361-6560/abf276.

[14] R.L. Harrison, Introduction to Monte Carlo simulation, AIP Conf. Proc. 1204 (2009) 17–21. https://doi.org/10.1063/1.3295638.

[15] S. Van Der Walt, S.C. Colbert, G. Varoquaux, The NumPy array: A structure for efficient numerical computation, Comput. Sci. Eng. 13 (2011) 22–30. https://doi.org/10.1109/MCSE.2011.37.

Authors' Profiles

Pankaj Dumka has completed his B. Tech. (Mechanical Engineering) from Inderprastha Engineering College, Ghaziabad in 2007, M. Tech. from Indian Institute of Technology, Kanpur in "Fluids and Thermal Sciences" in 2010, and PhD from Jaypee University of Engineering and Technology, Guna in 2021. He has been working with the Jaypee University of Engineering and Technology as an assistant professor in the Department of Mechanical Engineering since 2011. He has published more than 40 research articles in reputed SCI and SCOPUS indexed Journals. His area of interest includes Thermodynamics, Fluid Dynamics, Computational Fluid Dynamics, Numerical Computations, Python programming, and Solar water desalination.

Rishika Chauhan is an Assistant Professor in the Department of Electronics and Communication Engineering at Jaypee University of Engineering and Technology, Guna. Her research focuses on wireless communication, neural networks, deep learning, and error-control coding. She has published numerous research papers in reputed international journals and conferences, with her work receiving over 200 citations. Dr. Chauhan's recent studies emphasize the integration of intelligent computing techniques in communication system design.

Dhananjay R. Mishra has obtained his bachelor's degree (Mechanical Engineering) from Amravati University (M.S.), Master's degree (Production Engineering) in 2007 from BIT, Durg of Pt. Ravi Shankar Shukla University, Raipur (C.G.), and Doctor of Philosophy in Mechanical Engineering from National Institute of Technology Raipur, Raipur, on August 2016. He is working as an Assistant Professor (SG) in the Mechanical Engineering Department of Jaypee University of Engineering and Technology, Guna (M.P.). He has published more than 125 research articles in peer-reviewed international, national journals, and conferences. His area of interest includes Solar thermal Applications, Renewable Energy, Python Programming, Numerical Computations, Internal combustion engines, and Solar Water Desalination.

How to cite this paper: Pankaj Dumka, Rishika Chauhan, Dhananjay R. Mishra, "Application of Python in Evaluating the Volume of 3D Shapes Using Monte Carlo Simulation", International Journal of Engineering and Manufacturing (IJEM), Vol.16, No.1, pp. 50-59, 2026. DOI:10.5815/ijem.2026.01.05