MONTE CARLO SIMULATION

FOR THE AREA UNDER THE CURVE

Group4 (AMS_B)

Lim KimHoung
Kry SengHort
Lay Chhay
Leat Seanglong

TABLE OF CONTENTS

1.INTRODUCTION

Monte Carlo Integration is a powerful computational technique widely used in various fields such as physics, finance, engineering, and computer science. It provides an effective approach to estimate complex integrals by employing random sampling and statistical methods. This literature review aims to explore the fundamental concepts, applications, and advancements in Monte Carlo Integration.

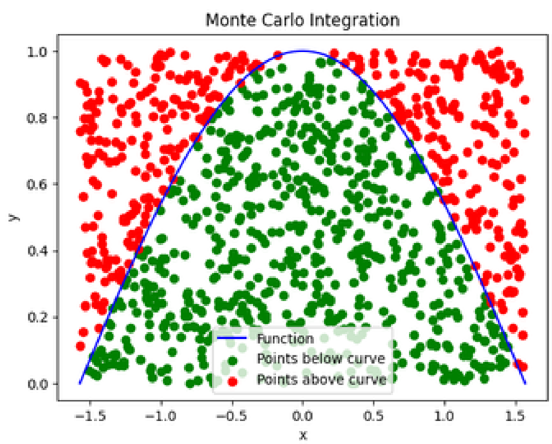
Risk and forecast analysis is part of every decision you make where we constantly face with uncertainty. Although we have a lot of informations, we still can't accurately forecast the future or calculate risk and monte carlo method lets us see all the possible outcome of your decision including the impact of risk in quantitatively, more accurate forcasting and better decision.

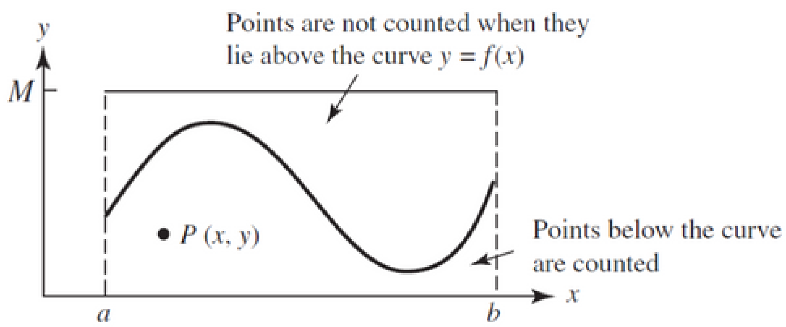
2. THEORETICAL FOUNDATIONS

The foundation of Monte Carlo Integration lies in the law of large numbers and the central limit theorem. By generating random samples from the target distribution, Monte Carlo methods estimate the integral by averaging the function evaluations over these samples. The accuracy of the estimation increases with the number of samples, as the average converges to the true integral value.

Furthermroe, Various sampling techniques are employed in Monte Carlo Integration to generate the necessary random samples. The most commonly used method is simple random sampling, where points are uniformly sampled from the integration domain. Other techniques include stratified sampling, importance sampling, and Latin hypercube sampling, each offering advantages in specific scenarios to reduce variance and improve convergence rates.

On the other hand, Monte Carlo Integration is known for its potential high variance. To address this issue, researchers have developed numerous variance reduction techniques. These techniques aim to modify the sampling process or the function being integrated to achieve more efficient estimations. Some popular approaches include control variates, antithetic variates, and quasi-Monte Carlo methods, which reduce the variance by exploiting the underlying structure of the problem.





3. IMPORTANT KEY OF MONTE CARLO SIMULATION

Monte Carlo Simulation` is a model used to predict the probability of a variety of outcome from random value of uncertain variable.
* It help to explain the impact of risk and uncertainty in prediction and forcasting models.
* It a way to use random samples to parameters to explore behavior of a complex system
* It `requires assigning different values to an uncertainty variable to archieve different result outcome and then averaging the results to obtain an estimate.

4. APPLICATIONS OF MONTE CARLO ALGORITHM

Physics and Engineering`: Monte Carlo Integration finds extensive applications in physics and engineering. It has been employed in computational physics simulations, such as the modeling of particle interactions, Monte Carlo radiation transport, and fluid dynamics. In engineering, Monte Carlo Integration is used for uncertainty quantification, reliability analysis, and optimization problems where complex integrals arise.

- **`Financial and Computational Applications`** : In the realm of finance, Monte Carlo Integration plays a crucial role in option pricing, risk assessment, and portfolio optimization. By simulating thousands or millions of potential market scenarios, Monte Carlo methods provide reliable estimates for pricing derivatives and evaluating investment strategies. Additionally, in computational science, Monte Carlo Integration is utilized in areas like image reconstruction, numerical integration, and stochastic optimization.

- **`Advancements and Hybrid Approaches`**: Researchers continue to advance Monte Carlo Integration techniques to overcome computational challenges and improve accuracy. Hybrid approaches that combine Monte Carlo methods with other numerical integration techniques, such as deterministic quadrature methods or Markov chain Monte Carlo methods, have gained attention. These approaches aim to capitalize on the strengths of different methods to achieve more efficient and accurate integration results.

5. DRAWBACK OF MONTE CARLO ALGORITHM

- `**Slow Convergence**`: Monte Carlo methods typically rely on generating a large number of random samples to obtain accurate results. As a result, the convergence rate can be slow, especially for high-dimensional problems. The computational time required to achieve a desired level of precision can be significant, making Monte Carlo methods inefficient in certain scenarios.

- `**High Variance**`: Monte Carlo methods are known for their potential high variance, particularly when the integrand function exhibits large fluctuations or has a complex behavior. This high variance can lead to imprecise estimations and the need for a larger number of samples to reduce the uncertainty.

- **`Difficulties in Rare Event Estimation`:** Monte Carlo methods can encounter challenges when estimating rare events with extremely low probabilities. Due to the random nature of the sampling process, it may take an extensive number of samples to capture such rare events accurately. This issue is often referred to as the "rare event problem."

- **`Dependence on Randomness`:** Monte Carlo methods heavily rely on the generation of random numbers to produce the samples for estimation. The quality of the random number generator used can impact the accuracy and reliability of the results. Moreover, the randomness introduced can make the Monte Carlo estimates less reproducible.

- **`Curse of Dimensionality`:** Monte Carlo methods can suffer from the curse of dimensionality, especially when dealing with high-dimensional integration problems. As the dimensionality increases, the number of samples required to achieve a certain level of accuracy grows exponentially, which poses a computational challenge.

- **`Need for Special Techniques`:** In some cases, Monte Carlo methods may require additional techniques to improve efficiency or accuracy. Variance reduction techniques such as importance sampling, control variates, or stratified sampling may need to be applied to mitigate high variance issues. Implementing and optimizing these techniques can add complexity to the Monte Carlo process.

Despite these drawbacks, Monte Carlo methods remain valuable and widely used due to their versatility, applicability to complex problems, and ability to handle problems with limited analytical solutions. Researchers continue to develop and refine Monte Carlo techniques, as well as hybrid approaches, to address these limitations and enhance the efficiency and accuracy of Monte Carlo simulations.

6. MONTE CARLO ALGORITHM

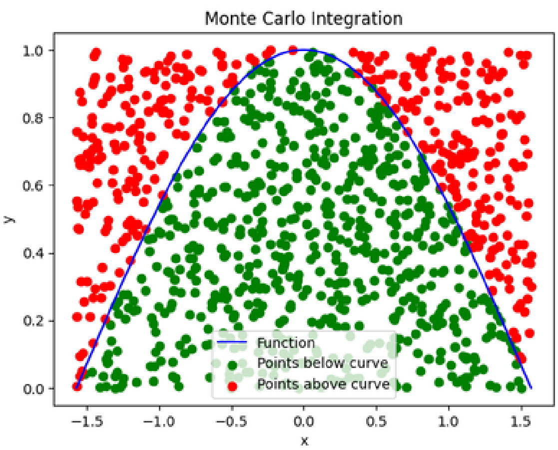
Step of Algorithm performance

- $Input: \ total\ number\ n\ of\ random\ points\ to\ be\ generated\ in\ the\ simulation$
- $Output: \ Area\ =\ approximate\ area\ under\ specified\ curve\ y\ = f(x)\ over\ the\ given\ interval\ a\ <= x\ <= b\ where\ 0 <= f(x) < M$
- $Step1: \ Initialize: counter = 0$
- $Step2: \ For\ i = 1, 2, 3, .., n$
- $Step3: \ Calculate\ random\ corrdinates\ x_i\ and\ y_i\ that\ satisfied\ a <= x_i <= b\ and\ 0 <= y_i < M.$
- $Step4: \ Calculate\ f(x_i)\ for\ the\ random\ x_i\ coordinate.$
- $Step5: \ If\ y_i <= f(x_i), then\ increment\ the\ counter\ by\ 1.\ Otherwise,\ leave\ counter\ as\ is,$
- $Step6: \ Calculate\ Area$

$$Area = \frac{M(b-a)counter}{n}$$

CODE IMPLEMENTATION

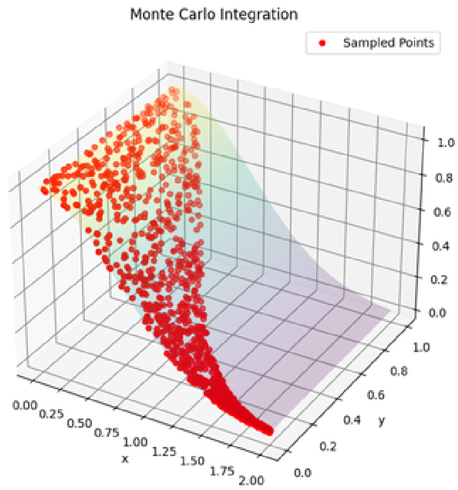
```python
import numpy as np
import matplotlib.pyplot as plt

a = -(np.pi)/2
b = (np.pi)/2
M = 1
N = 1000

def f(x):
    return np.cos(x)

# Generate random points
x = np.random.uniform(a, b, N)
y = np.random.uniform(0, M, N)

# Compute the mask of points below the curve
mask = y <= f(x)

# Compute the estimated area
counter = np.sum(mask)
area = (M - 0) * (b - a) * counter / N

# Plot the function and the random points
fig, ax = plt.subplots()
x_vals = np.linspace(a, b, 1000)
y_vals = f(x_vals)
ax.plot(x_vals, y_vals, 'b-', label='Function')
ax.scatter(x[mask], y[mask], c='g', label='Points below curve')
ax.scatter(x[~mask], y[~mask], c='r', label='Points above curve')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Monte Carlo Integration')
ax.legend()
plt.show()

print(f'The estimated area is: {area}')
```




```
The estimated area is: 2.038893632179776
```

CODE IMPLEMENTATION IN OTHER WAY

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

def simple_monte_carlo_integration3D(f, a, b, n):
    x_values = []
    y_values = []
    integral_values = []

    for _ in range(n):
        x = np.random.uniform(a, b)  # Randomly sample x
        x_values.append(x)

        integral = f(x)
        integral_values.append(integral)

        y = np.random.uniform(0, integral)  # Randomly sample y within the integral range
        y_values.append(y)

    average = np.mean(integral_values)
    integral = (b - a) * average

 #   print(f'Approximation area by integral = {integral}')

    return x_values, y_values, integral_values



if __name__ == '__main__':

    def f(x):
        return np.exp(-x**2)

    a = 0.0  # Lower limit
    b = 2.0  # Upper limit
    n = 1000  # Number of samples

    x_vals, y_vals, integral_vals = simple_monte_carlo_integration3D(f=f, a=a, b=b, n=n)

    # Generate a grid of x and y values for the function plot
    x_grid = np.linspace(a, b, 1000)
    y_grid = np.linspace(0, max(integral_vals), 1000)
    X, Y = np.meshgrid(x_grid, y_grid)
    Z = f(X)

    # Create a 3D plot
    fig = plt.figure(figsize=(7,8))
    ax = fig.add_subplot(111, projection='3d')

    # Plot the function surface
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.2)

    # Plot the sampled points
    ax.scatter(x_vals, y_vals, integral_vals, color='red', label='Sampled Points')

    # Set plot labels and title
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    ax.set_title('Monte Carlo Integration')

    # Set the rotation angle of the z-axis label
    ax.zaxis.set_rotate_label(True)  # Disable automatic rotation
    ax.zaxis.set_label_coords(0.5, 0.1)  # Set the coordinates of the label

    # Customize gridlines
    ax.xaxis._axinfo["grid"]['color'] = 'gray'
    ax.yaxis._axinfo["grid"]['color'] = 'gray'
    ax.zaxis._axinfo["grid"]['color'] = 'gray'

    # Display a legend
    ax.legend()
    plt.show()
```



MONTE CARLO VOLUMN ALGORITHM

Monte Carlo Volume Algorithm

Input Total number n of random points to be generated in the simulation.

Output VOLUME = approximate volume enclosed by the specified function, $z = f(x, y)$ in the first octant, $x > 0$, $y > 0$, $z > 0$.

Step 1 Initialize: COUNTER = 0.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3–5.

Step 3 Calculate random coordinates x_i, y_i, z_i that satisfy $0 \le x_i \le 1$, $0 \le y_i \le 1$, $0 \le z_i \le 1$. (In general, $a \le x_i \le b$, $c \le y_i \le d$, $0 \le z_i \le M$.)

Step 4 Calculate $f(x_i, y_i)$ for the random coordinate (x_i, y_i).

Step 5 If random $z_i \le f(x_i, y_i)$, then increment the COUNTER by 1. Otherwise, leave COUNTER as is.

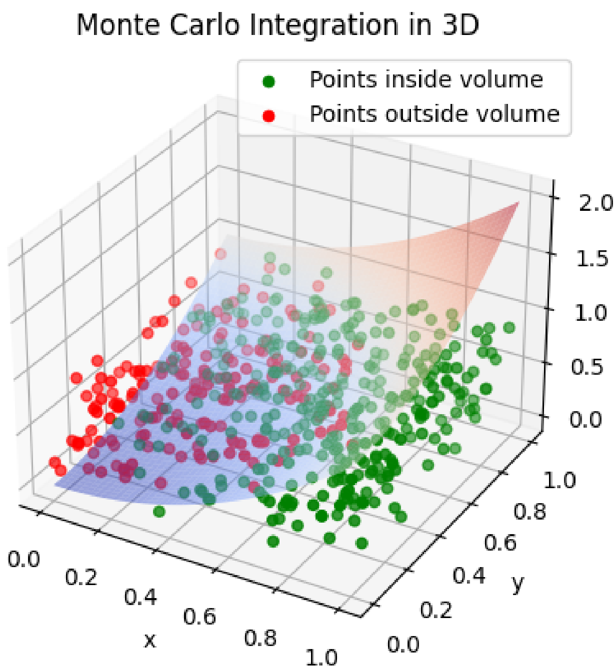
Step 6 Calculate VOLUME $= M(d - c)(b - a)$COUNTER$/n$.

Step 7 OUTPUT (VOLUME)

STOP

CODE IMPLEMENTATION

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

def f(x, y):
    return (x**2 + y**2)

a = 0
b = 1
c = 0
d = 1
M = 1
N = np.array([500, 1000, 5000, 10000])

for k in range(len(N)):
    # Generate random points
    x = np.random.uniform(a, b, N[k])
    y = np.random.uniform(c, d, N[k])
    z = np.random.uniform(0, M, N[k])

    # Compute the mask of points inside the volume
    mask = z <= f(x, y)

    # Compute the estimated volume
    counter = np.sum(mask)
    volume = M * (d - c) * (b - a) * counter / N[k]

    # Plot the function and the random points
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    x_vals = np.linspace(a, b, 100)
    y_vals = np.linspace(c, d, 100)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)

    ax.plot_surface(X, Y, Z, cmap='coolwarm', alpha=0.5)
    ax.scatter(x[mask], y[mask], z[mask], c='g', marker='o', label='Points inside volume')
    ax.scatter(x[~mask], y[~mask], z[~mask], c='r', marker='o', label='Points outside volume')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    ax.set_title('Monte Carlo Integration in 3D')
    ax.legend()
    plt.show()

    print(f'The estimated volume with {N[k]} points is: {volume}')
```



The estimated volume is: 0.646