Project Report

Monte Carlo Approximation

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Name: Probability and Statistics

Course Code : MALB -202

Submitted by

Kartik Mittal (221210056)

Under Supervision of

Dr. Indu Joshi

(TnP Officer)

NATIONAL INSTITUTE OF TECHNOLOGY DELHI

JUNE 2023

ACKNOWLEDGEMENTS

We would like to express our heartful thanks to Dr. Geeta Sikka, HOD of the Computer Science Department for giving permission and for her valuable support.

We are pleased to acknowledge Dr. Indu Joshi, TnP Officer for her invaluable guidance during the course of this project work and conducting a special guest lecture for this topic for better visualisation.

INTRODUCTION

The estimation of mathematical constants has been a fundamental pursuit in the realm of mathematics for centuries. Among these constants, the ratio of a circle's circumference to its diameter, commonly denoted as π (pi), stands as one of the most intriguing and vital figures in mathematics. With a non-repeating, non-terminating decimal representation, π has captured the imagination of mathematicians, scientists, and enthusiasts alike.

This report embarks on a journey to approximate the value of π using the powerful and versatile Monte Carlo simulation method. Monte Carlo methods, named after the Monte Carlo Casino, were first introduced by Stanislaw Ulam and John von Neumann in the 1940s. These methods have since found widespread applications in a wide range of fields, from physics and finance to computational mathematics and statistical analysis.

This report serves to elucidate the theory and application of Monte Carlo approximation techniques in the context of estimating π, with a focus on the Python code used to execute the simulation. As we progress through the report, we will delve into the specifics of the code, present our findings, and analyze the accuracy of our estimate in comparison to the true value of π.

METHODOLOGY

1. Random Point Generation:

Generate a specified number of random points within a unit square. Utilize a random number generator to obtain pairs of (x, y) coordinates, each ranging from -1 to 1, which effectively encompass the entire unit square.

2. Point Evaluation:

For each randomly generated point (x, y), compute its distance from the origin (0, 0) using the Pythagorean theorem: distance = sqrt(x^2 + y^2).

Assess whether the point falls inside or outside the unit circle based on its distance. A point is considered inside the circle if the calculated distance is less than or equal to 1.

3. Accumulation of Data:

Maintain a count of the number of points that fall inside the unit circle. The (x, y) coordinates of points both inside and outside the circle for visualization purposes. Periodically evaluate the estimated value of π as the ratio of the number of points inside the circle to the total number of generated points.

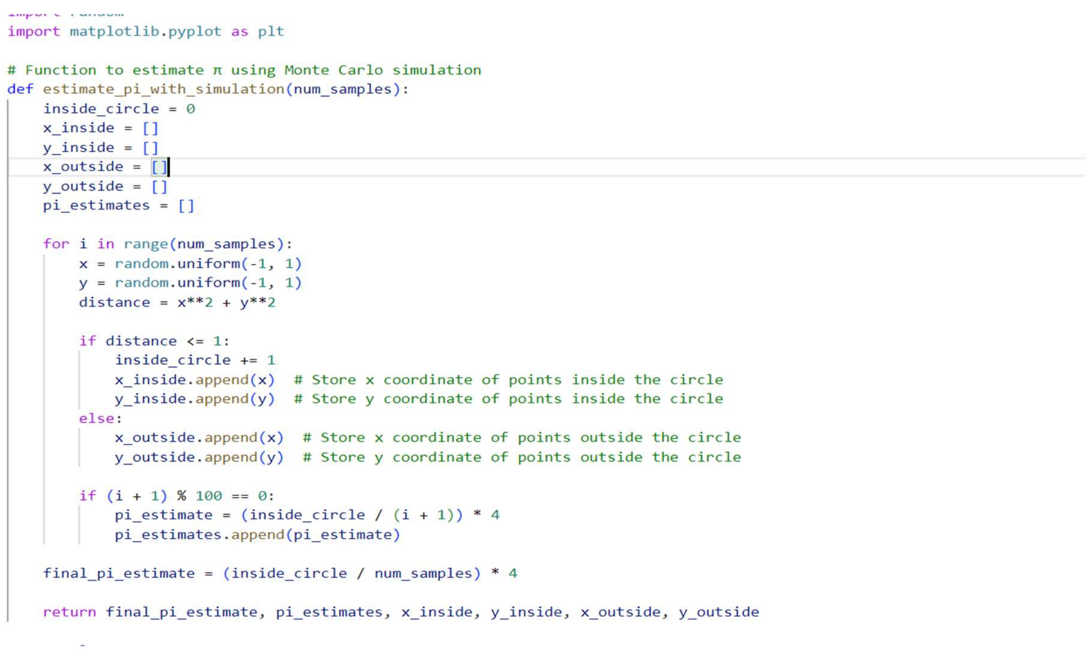
4. Monte Carlo Simulation:

The Monte Carlo simulation involves repeating the above steps for a large number of iterations (Num Samples). As the number of iterations increases, the estimate of π converges to a more accurate value.

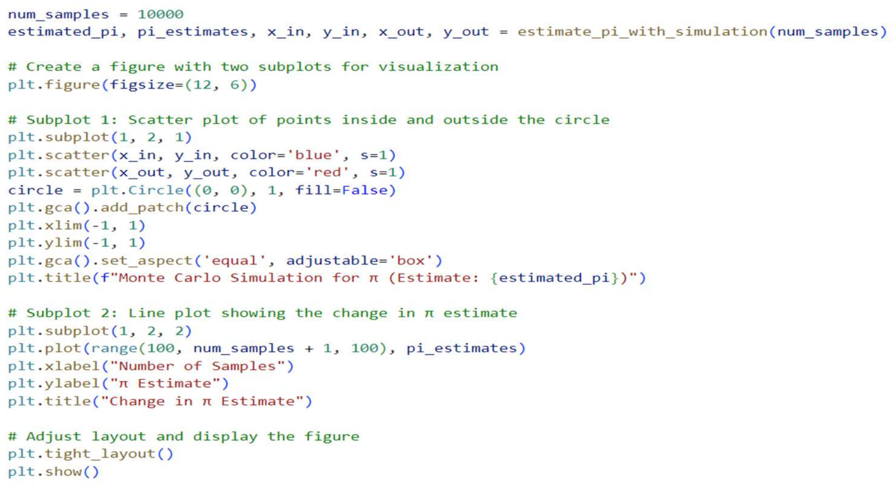
5. Python and matplotlib:

Implement the methodology using the Python programming language. Utilize Python's random number generation capabilities to create random points. Visualize the results using the matplotlib library, enabling the representation of points inside and outside the unit circle, as well as the change in the estimate of π over time.

SOURCE CODE

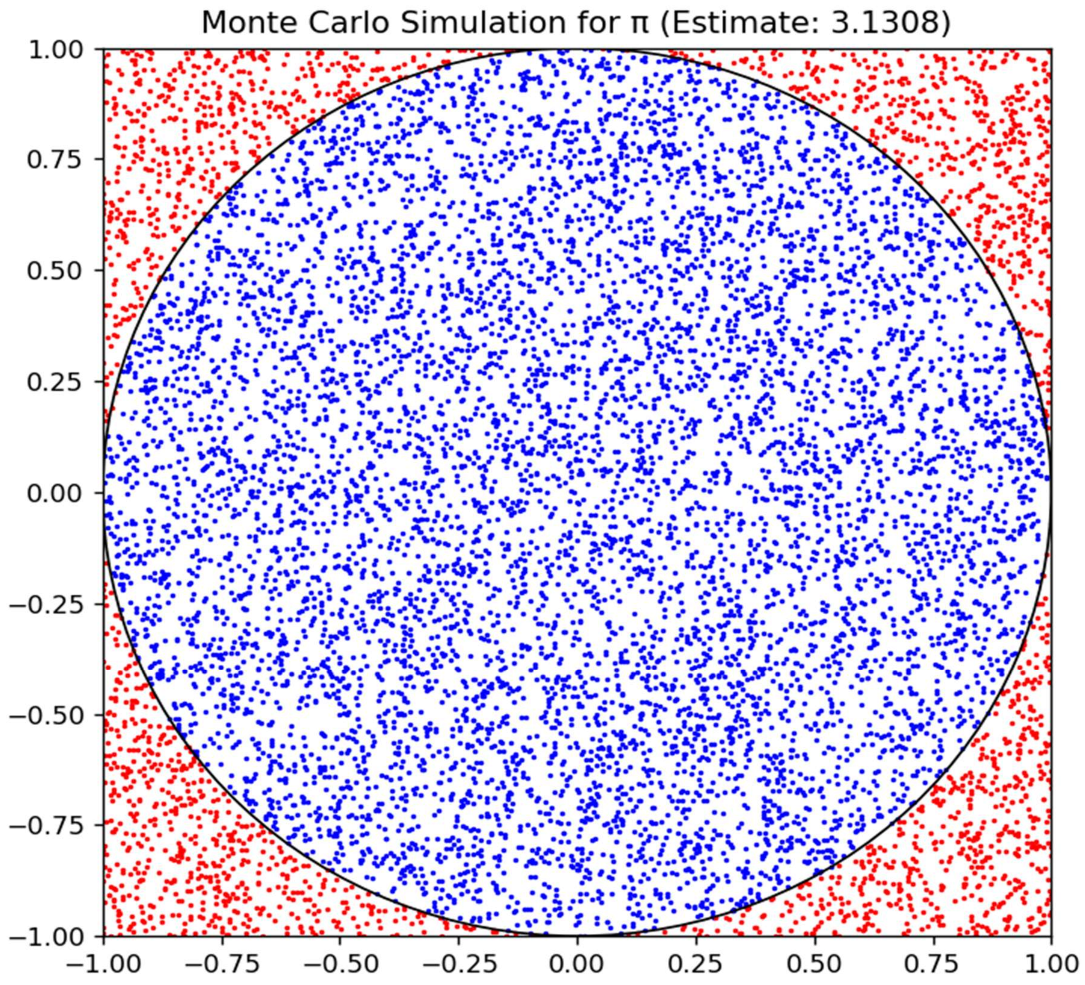


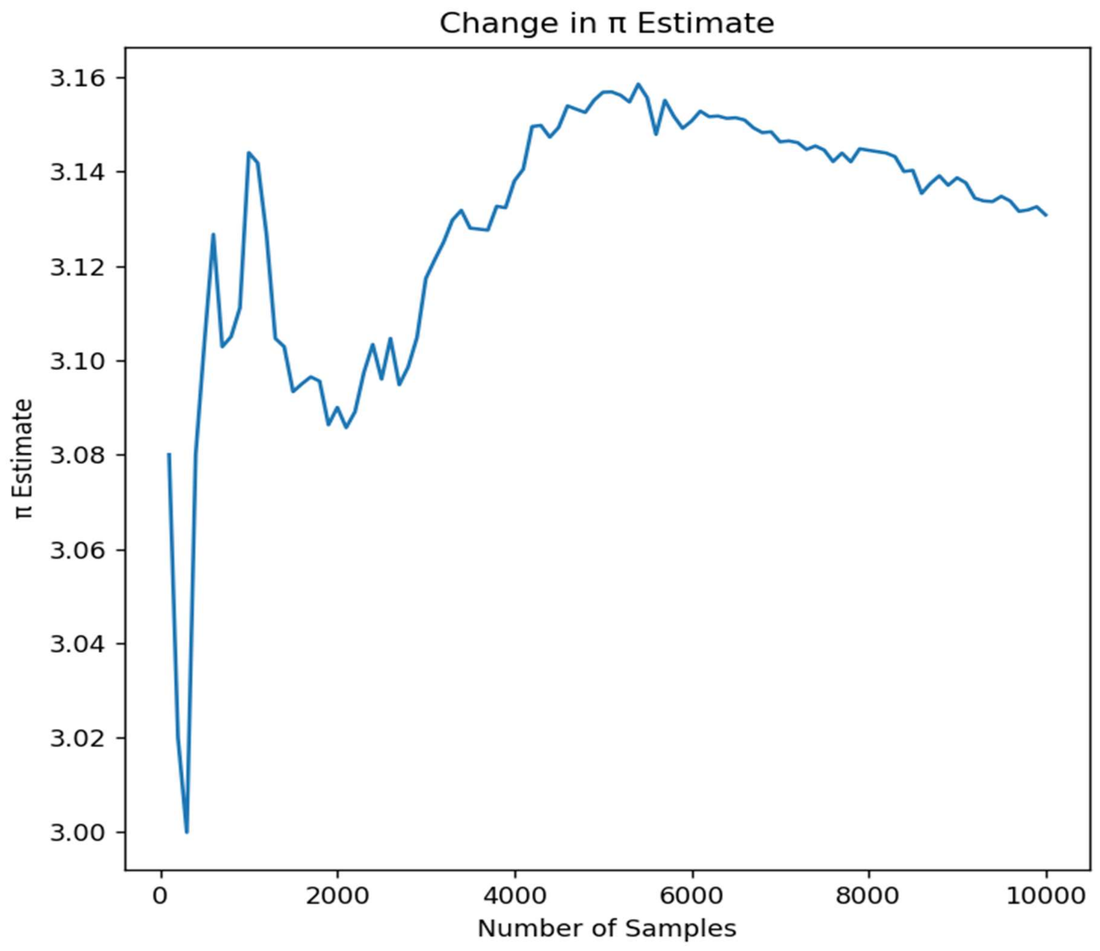
```python
import random
import matplotlib.pyplot as plt

# Function to estimate π using Monte Carlo simulation
def estimate_pi_with_simulation(num_samples):
    inside_circle = 0
    x_inside = []
    y_inside = []
    x_outside = []
    y_outside = []
    pi_estimates = []

    for i in range(num_samples):
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)
        distance = x**2 + y**2

        if distance <= 1:
            inside_circle += 1
            x_inside.append(x)  # Store x coordinate of points inside the circle
            y_inside.append(y)  # Store y coordinate of points inside the circle
        else:
            x_outside.append(x)  # Store x coordinate of points outside the circle
            y_outside.append(y)  # Store y coordinate of points outside the circle

        if (i + 1) % 100 == 0:
            pi_estimate = (inside_circle / (i + 1)) * 4
            pi_estimates.append(pi_estimate)

    final_pi_estimate = (inside_circle / num_samples) * 4

    return final_pi_estimate, pi_estimates, x_inside, y_inside, x_outside, y_outside
```



```python
num_samples = 10000
estimated_pi, pi_estimates, x_in, y_in, x_out, y_out = estimate_pi_with_simulation(num_samples)

# Create a figure with two subplots for visualization
plt.figure(figsize=(12, 6))

# Subplot 1: Scatter plot of points inside and outside the circle
plt.subplot(1, 2, 1)
plt.scatter(x_in, y_in, color='blue', s=1)
plt.scatter(x_out, y_out, color='red', s=1)
circle = plt.Circle((0, 0), 1, fill=False)
plt.gca().add_patch(circle)
plt.xlim(-1, 1)
plt.ylim(-1, 1)
plt.gca().set_aspect('equal', adjustable='box')
plt.title(f"Monte Carlo Simulation for π (Estimate: {estimated_pi})")

# Subplot 2: Line plot showing the change in π estimate
plt.subplot(1, 2, 2)
plt.plot(range(100, num_samples + 1, 100), pi_estimates)
plt.xlabel("Number of Samples")
plt.ylabel("π Estimate")
plt.title("Change in π Estimate")

# Adjust layout and display the figure
plt.tight_layout()
plt.show()
```

OUTPUT


Monte Carlo Simulation for π (Estimate: 3.1308)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


Change in π Estimate
π Estimate
Number of Samples
3.16
3.14
3.12
3.10
3.08
3.06
3.04
3.02
3.00
0
2000
4000
6000
8000
10000

CONCLUSION

The Monte Carlo simulation employed in this study has provided valuable insights into the estimation oπ, a fundamental mathematical constant of great significance. This conclusion encapsulates the key findings and offers a comprehensive assessment of the Monte Carlo approximation method.

1. Versatility of Monte Carlo Method:

The Monte Carlo method showcases its versatility in approximating complex mathematical constants,

offering a unique alternative to traditional analytical methods.The randomness inherent in the method provides a robust approach for tackling a wide range of problems.

2. Future Directions:

Further research can explore variations and improvements in the Monte Carlo simulation method to enhance accuracy and efficiency.The application of Monte Carlo techniques to other mathematical constants and scientific problems remains a promising area of study.

In conclusion, the Monte Carlo approximation for π stands as an effective and flexible tool for estimating mathematical constants and solving a wide range of computational problems. This study has demonstrated the potential of the method, emphasizing its reliability, versatility, and practical applicability.

The Python code used in this study, combined with the matplotlib library, provides a user-friendly and efficient platform for executing Monte Carlo simulations, making it accessible to a broader audience of researchers and practitioners.