Monte Carlo Integration Report

1. Introduction

This report explores the calculation of definite integrals using two Monte Carlo (MC) methods: the simple MC and geometric MC approaches. We examine the accuracy of these methods by calculating four integrals and plotting the error (ε) as a function of the total MC statistics, N_{tot}. The integrals are as follows:

1. $I_1 = \int_{-1}^{3} x\, e^{-x^2}\, dx$
2. $I_2 = \int_{0}^{2} x\, \sin(x^2)\, dx$
3. $I_3 = \int_{0.5}^{2.5} \frac{x}{1+x^2}\, dx$
4. $I_4 = \int_{0}^{1} x^2\, e^{-x^3}\, dx$

2. Methods

We use two different Monte Carlo approaches for numerical integration:

- **Simple Monte Carlo Method:** This approach uses random sampling of points within the interval of integration, averaging the function's value at these points, and multiplying by the interval length.
- **Geometric Monte Carlo Method:** Similar to the simple MC, but uses the absolute values of the function at the random points, which can sometimes improve accuracy in certain integral types.

3. Error Calculation

The error (ε) for each integral I is defined as:

$$\epsilon = |I_{\text{exact}} - I_{\text{MC}}|$$

where I_{exact} is the result obtained from the exact integration using SciPy's quad function, and I_{MC} is the result from the Monte Carlo integration. This error is evaluated across several values of N_{tot}: 100, 1000, 10000, 100000, and 1000000.

4. Results and Discussion

This code calculates the exact integrals, performs Monte Carlo integration with both methods, and generates the plot for error analysis.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

# Exact integrals
def I1_exact():
    return quad(lambda x: x * np.exp(-x**2), -1, 3)[0]

def I2_exact():
    return quad(lambda x: x * np.sin(x**2), 0, 2)[0]

def I3_exact():
    return quad(lambda x: x / (1 + x**2), 0.5, 2.5)[0]

def I4_exact():
    return quad(lambda x: x**2 * np.exp(-x**3), 0, 1)[0]

# Simple Monte Carlo method for integration
def simple_monte_carlo(f, a, b, N):
    random_points = np.random.uniform(a, b, N)
    return (b - a) * np.mean(f(random_points))

# Geometric Monte Carlo method for integration
def geometric_monte_carlo(f, a, b, N):
    random_points = np.random.uniform(a, b, N)
    weights = np.abs(f(random_points))
    return (b - a) * np.sum(weights) / N

# Functions for integrals
f1 = lambda x: x * np.exp(-x**2)
f2 = lambda x: x * np.sin(x**2)
f3 = lambda x: x / (1 + x**2)
f4 = lambda x: x**2 * np.exp(-x**3)

# Exact results
I_exact = [I1_exact(), I2_exact(), I3_exact(), I4_exact()]

# MC parameters
N_tot_values = [100, 1000, 10000, 100000, 1000000]

# Errors arrays for simple and geometric MC
eps_simple = np.zeros((4, len(N_tot_values)))
eps_geometric = np.zeros((4, len(N_tot_values)))

# Calculate the errors for each integral and N_tot
functions = [f1, f2, f3, f4]
I_exact = [I1_exact(), I2_exact(), I3_exact(), I4_exact()]

for i, (f, I) in enumerate(zip(functions, I_exact)):
    for j, N_tot in enumerate(N_tot_values):
        I_mc_simple = simple_monte_carlo(f, -1, 3, N_tot) if i == 0 else simple_monte_carlo(f, 0, 2.5, N_tot)
        I_mc_geometric = geometric_monte_carlo(f, -1, 3, N_tot) if i == 0 else geometric_monte_carlo(f, 0, 2.5, N_tot)

        # Calculate absolute errors
        eps_simple[i, j] = np.abs(I - I_mc_simple)
        eps_geometric[i, j] = np.abs(I - I_mc_geometric)

# Plot the errors for all integrals
plt.figure(figsize=(10, 8))

for i in range(4):
    plt.plot(N_tot_values, eps_simple[i], label=f'I{i+1} Simple MC', linestyle='--')
    plt.plot(N_tot_values, eps_geometric[i], label=f'I{i+1} Geometric MC')

plt.xscale('log')
plt.yscale('log')
plt.xlabel('N_tot')
plt.ylabel('epsilon (|I_exact - I_MC|)')
plt.legend()
plt.title('Error (epsilon) as function of N_tot for Simple and Geometric Monte Carlo methods')
plt.grid(True)

# Save the plot as a .jpg file
plt.savefig("monte_carlo_errors_plot_3.jpg")

plt.show()

```

In the following plot, we show the dependence of the error ε as a function of N_{tot} for both MC methods. Each curve represents the error for one of the integrals, with separate lines for the simple and geometric approaches.