Институт информационных технологий

Кафедра «Информационные системы»

ОТЧЕТ

по лабораторной работе № 5

«Метод Монте-Карло»

по дисциплине «Методы системного анализа и проектирования информационных систем»

Выполнил студент группы ИС/б-22-1-о

Донец Н.О.

Проверил доцент

Кудрявченко И.В.

Севастополь

2024

5.1 Цель работы

Углубление теоретических знаний в области системного анализа, ознакомление с методом Монте-Карло.

5.2 Вариант задания

Найти приближенное значение интеграла заданной функции f(x) на отрезке [a, b] по формулам Монте-Карло, произвести оценку погрешности. Вариант показан ан рисунке 5.1.

6	0, 1	$\sin(2x^2 + 1)$

Рисунок 5.1 – Вариант задания

5.3 Ход выполнения работы

Написать программу на языке программирования python для вычисления площади под кривой методом Монте-Карло. Также программа строит график зависимости точности результата от числа испытаний. Код программы показан в листинге 5.1.

Листинг 5.1 – Программа вычисляющая площадь методом Монте-Карло

```
import numpy as np

import matplotlib.pyplot as plt


a = 0

b = 1


trial_counts = [100, 500, 1000, 5000, 10000]

errors = []
```

```
def f(x):
    return np.sin(2*x*x+1)

# метод Монте-Карло
def monte_carlo_integration(a, b, N):
    random_points_x = np.random.uniform(a, b, N)
    function_values = f(random_points_x)
    integral = (b - a) * np.mean(function_values)
    std_dev = np.std(function_values)
    error = (b - a) * std_dev / np.sqrt(N)
    return integral, error

# Точное значение интеграла
from scipy.integrate import quad
exact_value, _ = quad(f, a, b)
print(f"Точное значение интеграла: {exact_value:.6f}")

# Вычисление интеграла и погрешности
for N in trial_counts:
    estimated_value, error = monte_carlo_integration(a, b, N)
    errors.append(error)
    print(f"N = {N}, вычисленное значение интеграла: {estimated_value:.6f}, погрешность: {error:.6f}")

# Построение графика
plt.figure(figsize=(12, 6))
plt.plot(trial_counts, errors, label='Погрешность', marker='o')
plt.xlabel('Количество испытаний')
plt.ylabel('Погрешность')
plt.title('График зависимости погрешности результата от числа испытаний')
plt.legend()
plt.grid(True)
plt.show()

final_estimation = estimated_value
final_error = error
```

```
print("\nИтоговое значение интеграла: {:.6f}".format(final_estimation))

print("Оценка погрешности: {:.6f}".format(final_error))
```

На рисунке 5.1 показан вывод программы, где есть информация о погрешности вычисления методом Монте-Карло при различном количестве испытаний.

```
Точное значение интеграла: 0.831273
N = 100, вычисленное значение интеграла: 0.829305, погрешность: 0.022404
N = 500, вычисленное значение интеграла: 0.826975, погрешность: 0.009501
N = 1000, вычисленное значение интеграла: 0.826278, погрешность: 0.006324
N = 5000, вычисленное значение интеграла: 0.833644, погрешность: 0.002803
N = 10000, вычисленное значение интеграла: 0.830367, погрешность: 0.002005

Итоговое значение интеграла: 0.830367
Оценка погрешности: 0.002005
```

Рисунок 5.1 – Вывод программы

На рисунке 5.2 показано, как точность результата зависит от числа испытаний. Можно сделать вывод, что точность растет вместе с ростом количества испытаний.

Рисунок 5.2 – График зависимости точности результата от числа испытаний

Далее была написана программа, которая визуально показывает решения на графике. Код программы показан в листинге 5.2.

Листинг 5.2 – Программа, визуально отображающая решения

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

def f(x):
    return np.sin(2 * x**2 + 1)

a = 0
b = 1
N = 10000

# определение границ прямоугольной области
x_min, x_max = a, b
y_min, y_max = 0, 1  # максимальное значение sin(x) равно 1

# генерация случайных точек
random_points_x = np.random.uniform(x_min, x_max, N)
random_points_y = np.random.uniform(y_min, y_max, N)

# вычисление значений функции в случайных точках
function_values = f(random_points_x)

# подсчет точек под кривой
below_curve = random_points_y <= function_values
m = np.sum(below_curve)

# площадь области
S_rectangle = (x_max - x_min) * (y_max - y_min)

# вычисление площади под кривой
S_curve = S_rectangle * m / N

# оценка погрешности
std_dev = np.std(below_curve)
```

```
error = S_rectangle * std_dev / np.sqrt(N)


# построение графика
plt.figure(figsize=(12, 6))


# отображение точек
plt.scatter(random_points_x[below_curve], random_points_y[below_curve], color='green', s=1, label='Точки под кривой')
plt.scatter(random_points_x[~below_curve], random_points_y[~below_curve], color='red', s=1, label='Точки над кривой')


# отображение функции
x_values = np.linspace(a, b, 500)
y_values = f(x_values)
plt.plot(x_values, y_values, color='blue', label='f(x) = sin(2x^2 + 1)')


plt.xlabel('x')
plt.ylabel('y')
plt.title('Метод Монте-Карло: точки и кривая')
plt.legend()
plt.grid(True)


plt.show()
```

На рисунке 5.3 показан вывод программы, где визуально отображены все точки, которые входят и не входят в решение.

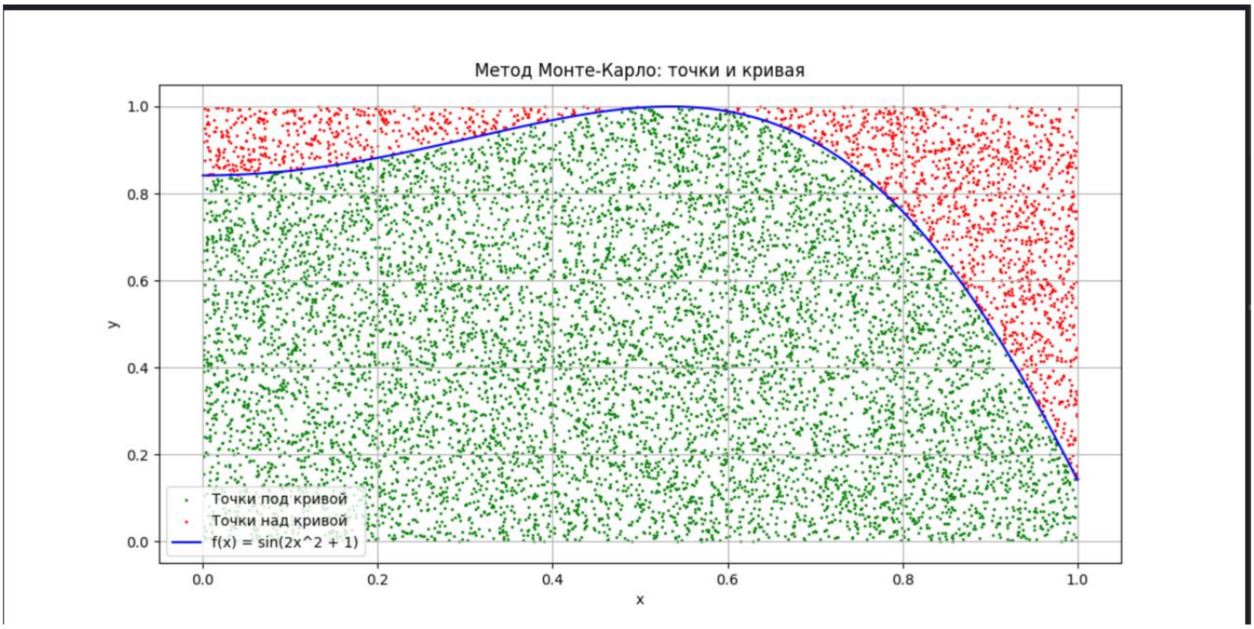

Рисунок 5.3 – Визуализация решения

Выводы

В ходе лабораторной работы был исследован метод Монте-Карло, который применяется для моделирования различных физических, экономических и других процессов. Для определенного интеграла, полученного по варианту, была написана программа на языке программирования python, которая вычисляет площадь под кривой методом Монте-Карло, помимо этого она строит график зависимости точности результата от числа испытаний. Также в качестве дополнительного задания была написана еще одна программа на языке python, которая визуально отображает результаты решения. В конце выполнения лабораторной работы был написан отчет.