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

Крюкова К.М.

Проверил доцент кафедры «ИС»

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.



Рисунок 5.1 – Вариант задания

**5.3 Ход выполнения работы**

5.3.1 В начале лабораторной работы интеграл, полученный по варианту, был посчитан аналитическим путем:

Затем была написана программа 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 = 2

trial\_counts = [100, 500, 1000, 5000, 10000]

errors = []

def f(x):

return np.exp(x)

# метод монте-карло

def monte\_carlo\_integration(a, b, N):

# генерация случайных точек

random\_points\_x = np.random.uniform(a, b, N)

random\_points\_y = np.random.uniform(0, np.exp(b), 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 = (b - a) \* (np.exp(b) - 0)

# вычисление площади под кривой

integral = S\_rectangle \* m / N

# оценка погрешности

std\_dev = np.std(below\_curve)

error = S\_rectangle \* std\_dev / np.sqrt(N)

return integral, error

# точное значение интеграла

exact\_value = np.exp(2) - np.exp(0)

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}")

# график

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()

# итоговое значение

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. По полученным данным можно сделать вывод, что разница между посчитанным аналитическим решением и программным составляет всего 0,07.

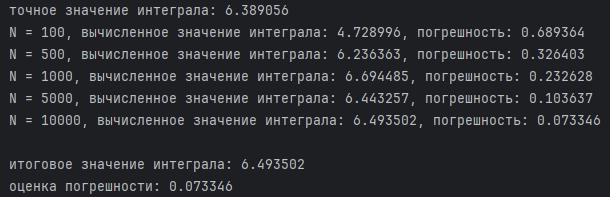


Рисунок 5.1 – Вывод программы

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

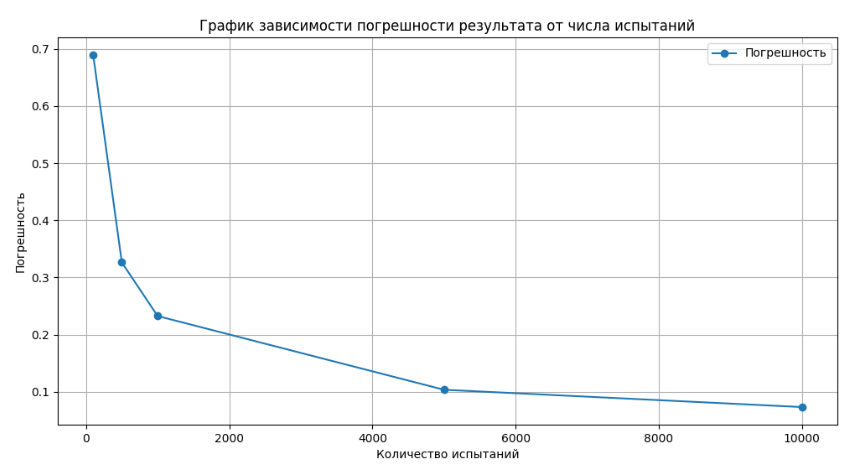


Рисунок 5.2 – График зависимости точности результата от числа испытаний

5.3.2 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.exp(x)

a = 0

b = 2

N = 10000

# определение границ прямоугольной области

x\_min, x\_max = a, b

y\_min, y\_max = 0, np.exp(b)

# 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) = exp(x)')

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.

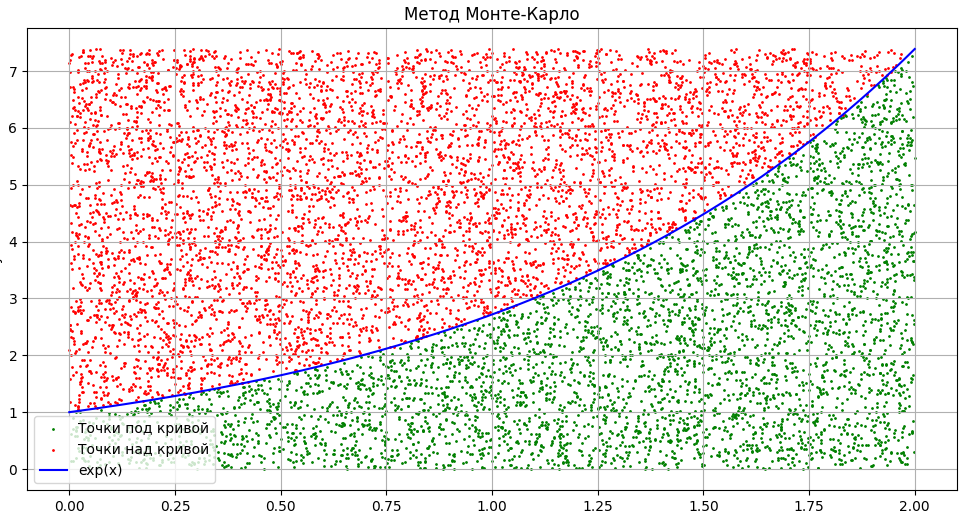


Рисунок 5.3 – Визуализация решения

**Выводы**

В ходе лабораторной работы был исследован метод Монте-Карло, который применяется для моделирования различных физических, экономических и других процессов. Полученный по варианту определенный интеграл был решен аналитическим методом, затем 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. При сравнении ответа, полученного аналитическим путем, и программного решений была выявлена погрешность, которая составила 0,07. 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.