ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Университет ИТМО

Отчёт по лабораторной работе № 1

«Расчет геометрической вероятности»

Выполнили работу:

Кузнецова Таисия 466397,

Демьянов Фёдор 471882,

Ячменников Егор 468202

Академическая группа:

№_J3114__

Санкт-Петербург 2025

Введение

Цель: Оценить геометрическую вероятность попадания случайных точек в круг внутри квадрата с помощью метода Монте-Карло, исследовать зависимость точности оценки от количества точек и минимальное количество точек для достижения заданной точности

Задачи:

1. Вычислить истинную геометрическую вероятность
2. Реализовать метод Монте-Карло для оценки вероятности
3. Оценить ошибку
4. Определить количество точек для требуемой точности

Геометрическая вероятность — это вероятность того, что случайное событие происходит в некоторой области, которая имеет геометрическую интерпретацию (например, площадь, длина или объем).

Формула:

$$P = \frac{S_A}{S_B}$$

где:

S_A — площадь интересующей области (куда хотим попасть),

S_B — площадь всей области, по которой равномерно распределены точки.

Метод Монте- Карло - это метод моделирования случайных событий с целью вычисления их характеристики

Оценка ошибки - погрешность метода Монте-Карло (абсолютная ошибка) вычисляется как:

$\varepsilon(n)=|\widehat{p}(n)-p|$

где:

$\widehat{p}(n)$ - полученная оценка вероятности,

p — теоретически точное значение вероятности (если известно).

Ход работы

1. В первую очередь мы выбрали 5 значений радиуса $r \in [\ 0, a]$ за a мы взяли единицу и воспользовались формулой

$$r_k = a - k\Delta r$$

Где:

r_k — радиус на k-м шаге;

a — максимальный (или стартовый) радиус;

Δr — шаг (изменение радиуса на каждом шаге) = $\frac{a}{5}$;

k — номер итерации (шаг, начиная с 0).

Мы выбрали эту формулу, так как считаем, что она лучше остальных подходит для нашей лабораторной работы, поскольку она позволяет равномерно распределить значения, проста в реализации и делает результат более стабильным нежели другие формулы.

Из формулы мы получили такие значения радиусов: 1, 0.8, 0.6, 0.4, 0.2

2. Преобразовали формулу геометрической (истинной) вероятности для нашего случая т.к нам дан квадрат со сторонами $2a$ то $S_A = 4a\text{^}2 = 4$. S_B - площадь круга = πr^2 Отсюда получаем, что наша формула геометрической вероятности:

$$P = \frac{\pi r^2}{4}$$

3. Реализовали эти две формулы в коде, также воспроизвели метод Монте-карло и сравнили полученные результаты.
4. После реализации метода Монте-Карло построили график полученной оценки вероятности и график оценки ошибки (абсолютная ошибка)
5. Для каждого значения r вычислили необходимое количество случайных точек N, необходимых для достижения данной точности εi и посторили график.

Практическая часть

1. Подсчет геометрической вероятности и вероятности с помощью Монте-Карло

```
import numpy as np
import matplotlib.pyplot as plt
import random
from concurrent.futures import ThreadPoolExecutor

#Выбор радиусов
a = 1 # значение из 2a
r = 5 # кол-во  радиусов
radiuses = np.linspace(0.2*a, a,r)
radiuses = np.round(radiuses, 2)

#Находим p (вероятность)
true_probility = ((radiuses**2)*np.pi) / ((a**2)*4)
print(true_probility)
```

```
[0.03141593 0.12566371 0.28274334 0.50265482 0.78539816]
```

Здесь в переменной *true_probility* мы выполнили подсчет вероятности используя формулу приведенную выше, полученные значения соответствуют тем, что мы посчитали в ручную, значит код верно подсчитывает вероятность.

Далее мы реализовали метод Монте-Карло, для генерации случайных точек мы использовали *np.random.uniform* . Для генерации точек мы взяли количество = 1000000. После генерации точек мы подсчитали какая доля попала в круг и из этого получили вероятность.

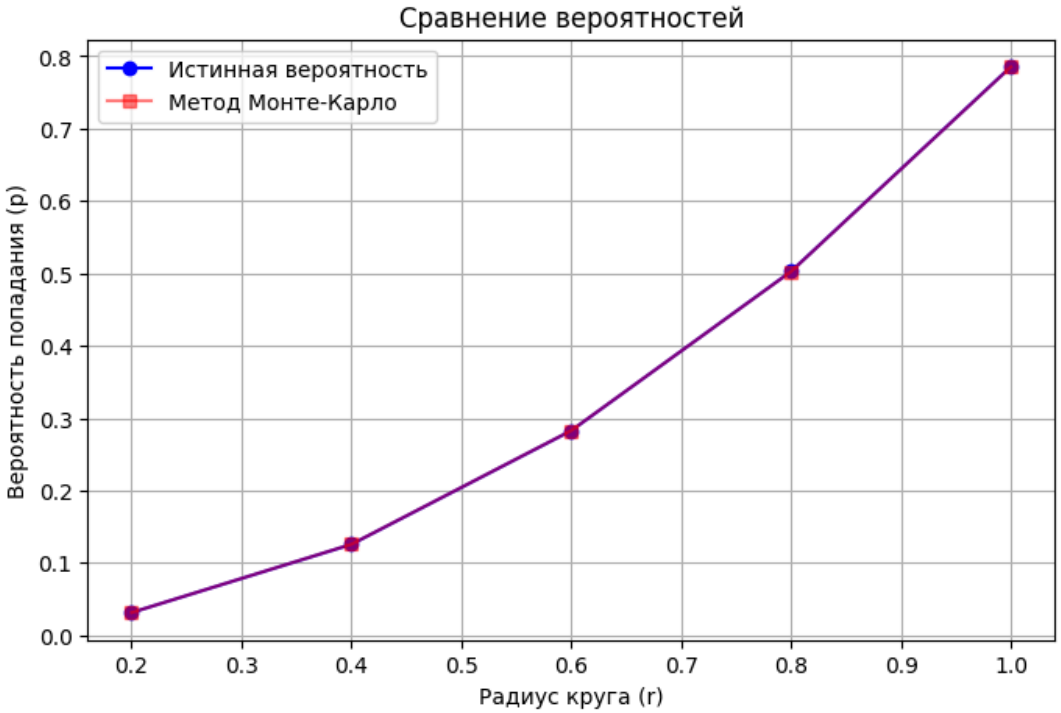
```
# Реализация метода Монте-Карло
def monte_carlo(r, N, a):
    # Генерация случайных точек и проверка попадания в круг
    x = np.random.uniform(-a, a, N)
    y = np.random.uniform(-a, a, N)
    return np.sum(x**2 + y**2 <= r**2) / N

N = 10000000
with ThreadPoolExecutor() as executor:
    Monte_Carlo_probility = list(executor.map(lambda r: monte_carlo(r, N, a), radiuses))

print(Monte_Carlo_probility)
```

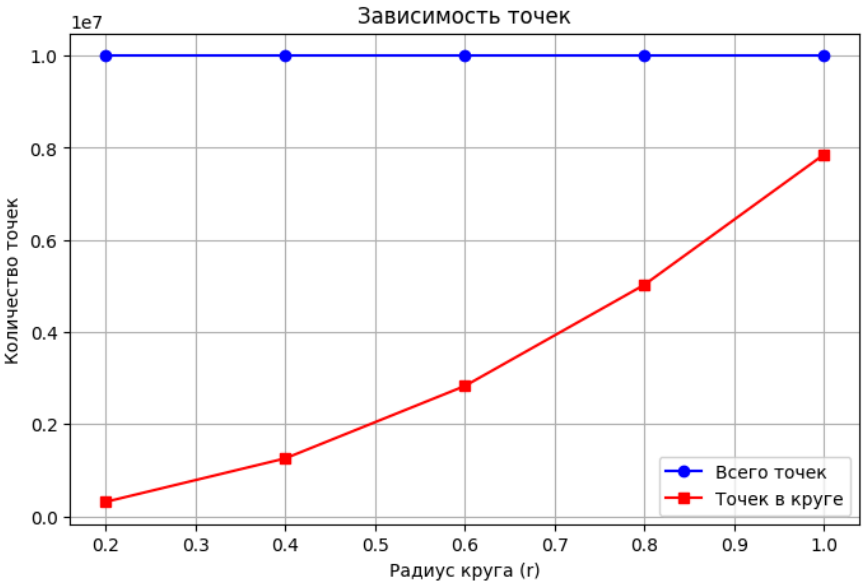
```
[np.float64(0.0314191), np.float64(0.1257747), np.float64(0.2828573), np.float64(0.5025193), np.float64(0.7851919)]
```

После чего мы построил график, чтобы сравнить полученные результаты.



Графики накладываются друг на друга. Это значит, что полученные результаты совпадают, что доказывает работоспособность метода Монте-Карло.

Также мы решили посмотреть как размер радиуса влияет на количество попавших в него точек.



Из графика видно, что чем больше радиус, тем больше точек в него попадает, что подтверждает равномерное распределение точек.

2. Посчитали оценочную вероятность для каждого радиуса и построили график

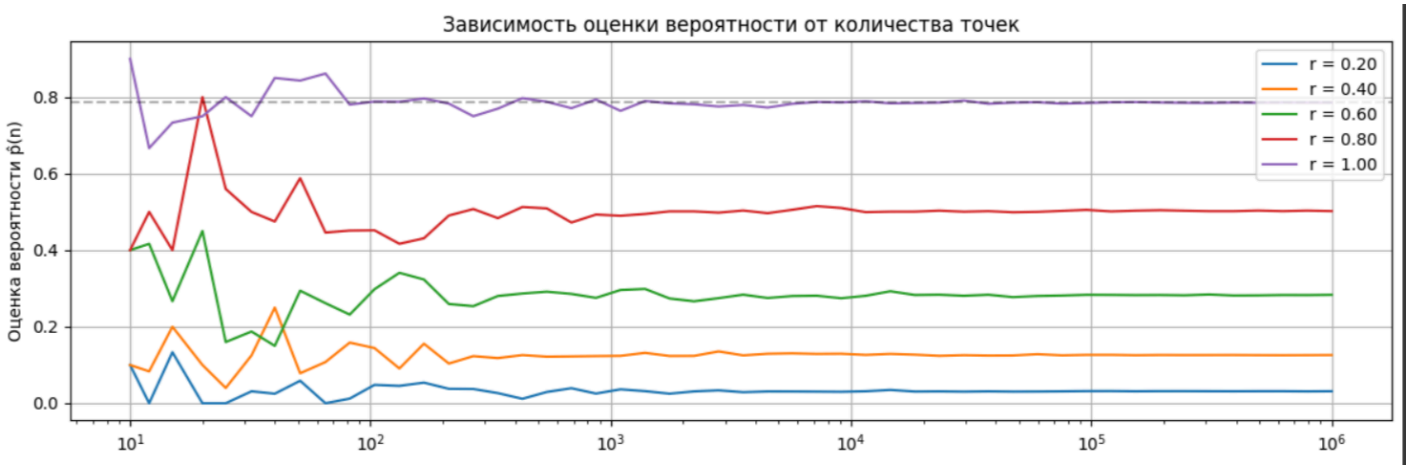
```python
# 6. Анализ зависимости оценки от количества точек для каждого радиуса
n_values = np.logspace(1, 6, 50, dtype=int)

plt.figure(figsize=(12, 8))
for r, true_p in zip(radiuses, true_probility):
    p_hats = []
    for n in n_values:
        x = np.random.uniform(-a, a, n)
        y = np.random.uniform(-a, a, n)
        inside = np.sum(x**2 + y**2 <= r**2)
        p_hats.append(inside / n)

    # График p̂(n)
    plt.subplot(2, 1, 1)
    plt.plot(n_values, p_hats, label=f'r = {r:.2f}')

    # График ε(n)
    errors = np.abs(p_hats - true_p)
    plt.subplot(2, 1, 2)
    plt.plot(n_values, errors, label=f'r = {r:.2f}')

# Настройка графиков
plt.subplot(2, 1, 1)
plt.axhline(true_p, color='black', linestyle='--', alpha=0.3)
plt.xscale('log')
plt.ylabel('Оценка вероятности p̂(n)')
plt.title('Зависимость оценки вероятности от количества точек')
plt.legend()
plt.grid(True)
```



Из него мы видим, что чем больше больше значение радиуса то тем, больше необходимо количества точек для наиболее точной вероятности. То есть с увеличением количества точек, будет увеличиваться и точность вероятности.

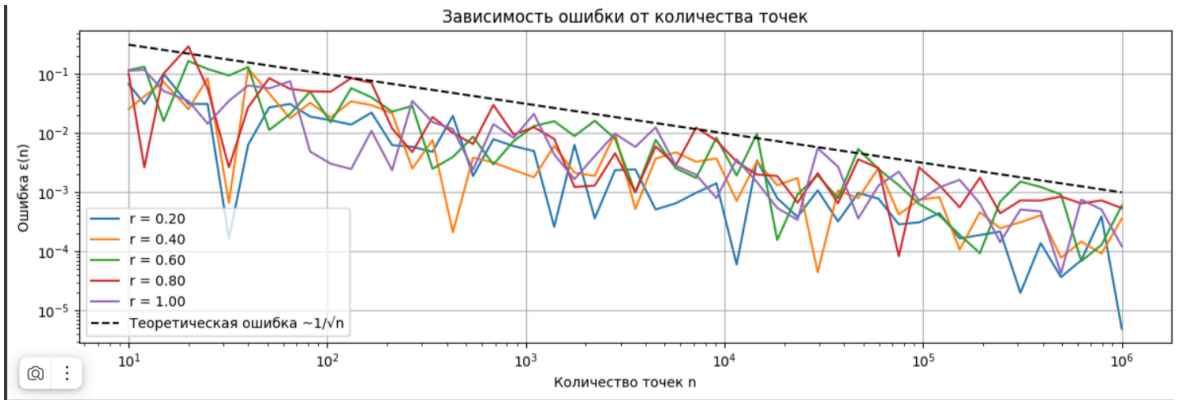
3. Далее мы высчитали оценку ошибки и построили график для неё

```
    # График ε(n)
    errors = np.abs(p_hats - true_p)
    plt.subplot(2, 1, 2)
    plt.plot(n_values, errors, label=f'r = {r:.2f}')

# Настройка графиков
plt.subplot(2, 1, 1)
plt.axhline(true_p, color='black', linestyle='--', alpha=0.3)
plt.xscale('log')
plt.ylabel('Оценка вероятности p̂(n)')
plt.title('Зависимость оценки вероятности от количества точек')
plt.legend()
plt.grid(True)

plt.subplot(2, 1, 2)
plt.plot(n_values, 1/np.sqrt(n_values), 'k--', label='Теоретическая ошибка ~1/√n')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Количество точек n')
plt.ylabel('Ошибка ε(n)')
plt.title('Зависимость ошибки от количества точек')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



Из него мы видим, что с увеличением количества точек, значение ошибки уменьшается, маленькое значение N приводит к довольно большой величине ошибки, из-за чего результат будет сильно различаться с настоящими значениями. Также можно увидеть, что чем меньше радиус тем больше необходимо точек, для получения точной оценки.

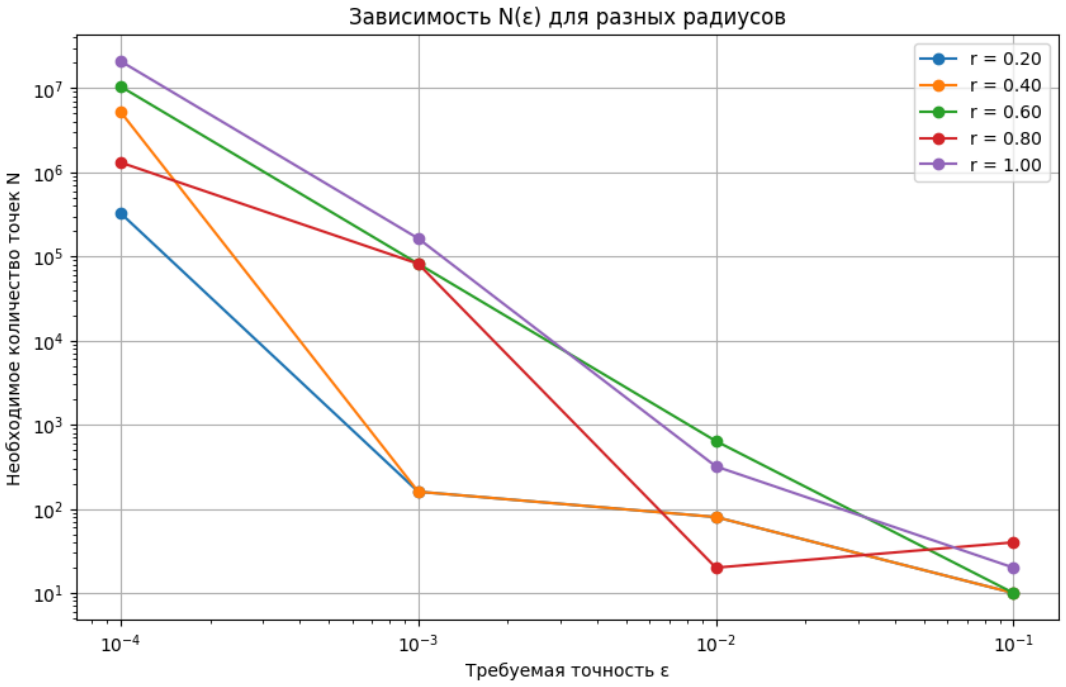
4. Для значения r вычислили необходимое количество случайных точек N, необходимых для достижения точности $\varepsilon \in [10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}]$ и построили график

```python
# Расчет N(ε) для разных значений точности
eps_targets = [10**(-i) for i in range(1, 5)]  # ε от 0.1 до 0.0001

plt.figure(figsize=(10, 6))
for r, true_p in zip(radiuses, true_probility):
    n_required = []
    for eps_target in eps_targets:
        n = 10  # Начальное значение
        while True:
            x = np.random.uniform(-a, a, n)
            y = np.random.uniform(-a, a, n)
            p_hat = np.sum(x**2 + y**2 <= r**2) / n
            if np.abs(p_hat - true_p) <= eps_target:
                n_required.append(n)
                break
            n *= 2 # Каждый раз увеличиваем значение в 2 раза после каждой итерации

    plt.plot(eps_targets, n_required, 'o-', label=f'r = {r:.2f}')

plt.xscale('log')
plt.yscale('log')
plt.xlabel('Требуемая точность ε')
plt.ylabel('Необходимое количество точек N')
plt.title('Зависимость N(ε) для разных радиусов')
plt.legend()
plt.grid(True)
plt.show()
```



По графику видно,что чем меньше значение требуемой точности, тем больше необходимо значение N. Также можно заметить, что чем меньше точность, тем меньше разница количества точек для радиусов, так когда точность 10^{-1} значения N для радиуса 0.4 и 0.6 практически одинаково

Вывод.

В ходе выполнения лабораторной работы мы можем сделать выводы:

1. Метод Монте-Карло является мощным методом для оценки вероятностей.Преимущество этого метода заключается в его универсальности и возможности работать с различными геометрическими фигурами, но его точность зависит от количества случайных точек.
2. Чем больше случайных точек мы генерируем, тем точнее становится наша оценка вероятности. Для достижения более высокой точности требуется экспоненциально больше точек, что подтверждается логарифмической зависимостью на графике. Это демонстрирует важность правильного выбора количества точек в зависимости от требуемой точности. Для достижения высокой точности с помощью метода Монте-Карло потребуется значительное увеличение числа точек
3. С увеличением радиуса и числа точек могут возникать числовые погрешности, особенно когда почти все точки попадают в круг. Это подтверждает важность учета числовых ошибок.

Приложение

Код: Матстат1.ipynb