Статистика для анализа данных

Лабораторная работа 1

Расчет геометрической вероятности

Участники:
Карташов Игорь Олегович, 466135, J3112

27 марта 2025 г.

Содержание

1 Выполнение лабораторной работы

- Ход работы
  - Определение параметров эксперимента
  - Расчёт истинной геометрической вероятности
  - Генерация случайных точек
  - Определение попадания точек в круг
  - Вычисление оценочной вероятности
  - Построение графиков
  - Анализ результатов

2 Основная часть: описание каждого шага, промежуточные результаты и их анализ

2.1 Шаг 1: Задание параметров эксперимента

Описание: Задаются значения параметров:

- a — половина длины стороны квадрата Ω. Пример: $\Omega = [-a, a] \times [-a, a]$.
- r — радиус круга $A(r)$, где $r \in (0, a]$.
- N — общее количество точек, генерируемых для оценки вероятности.

Определены основные параметры, которые будут использованы в последующих расчетах.

2.2 Шаг 2: Расчёт истинной геометрической вероятности

Описание: Истинная вероятность попадания точки в круг вычисляется как отношение площади круга к площади квадрата:

$$p = \frac{\pi r^2}{(2a)^2} = \frac{\pi r^2}{4a^2}.$$

Получено теоретическое значение p, которое служит контрольной точкой для оценки результатов, полученных методом Монте-Карло.

2.3 Шаг 3: Генерация случайных точек в квадрате Ω

Описание: С использованием генератора случайных чисел генерируются N точек с координатами (x, y), равномерно распределёнными по интервалу $[-a, a]$ для каждой координаты. Создан массив случайных точек для дальнейшего анализа.

2.4 Шаг 4: Определение попадания точек в круг $A(r)$

Описание: Для каждой сгенерированной точки проверяется условие попадания в круг:

$$x^2 + y^2 \leq r^2.$$

Если условие выполняется, точка считается попавшей в круг. Получена логическая маска (или массив индикаторов), показывающая, какие точки находятся внутри круга.

2.5 Шаг 5: Вычисление оценочной вероятности $\hat{p}(n)$

Описание: Для различных значений n (например, с шагом 100 от 100 до N) вычисляется оценка вероятности:

$$\hat{p}(n) = \frac{\text{число точек, попавших в круг}}{n}.$$

Получена последовательность оценок $\hat{p}(n)$, что позволяет наблюдать сходимость оценки к истинному значению p.

2.6 Шаг 6: Построение графика $\hat{p}(n)$

Описание: Построен график зависимости оценочной вероятности $\hat{p}(n)$ от числа точек n. На графике также нанесена горизонтальная линия, соответствующая истинному значению p. При увеличении n наблюдается, что оценка $\hat{p}(n)$ сходится к p. Это подтверждает закон больших чисел, поскольку с ростом числа точек флуктуации уменьшаются.

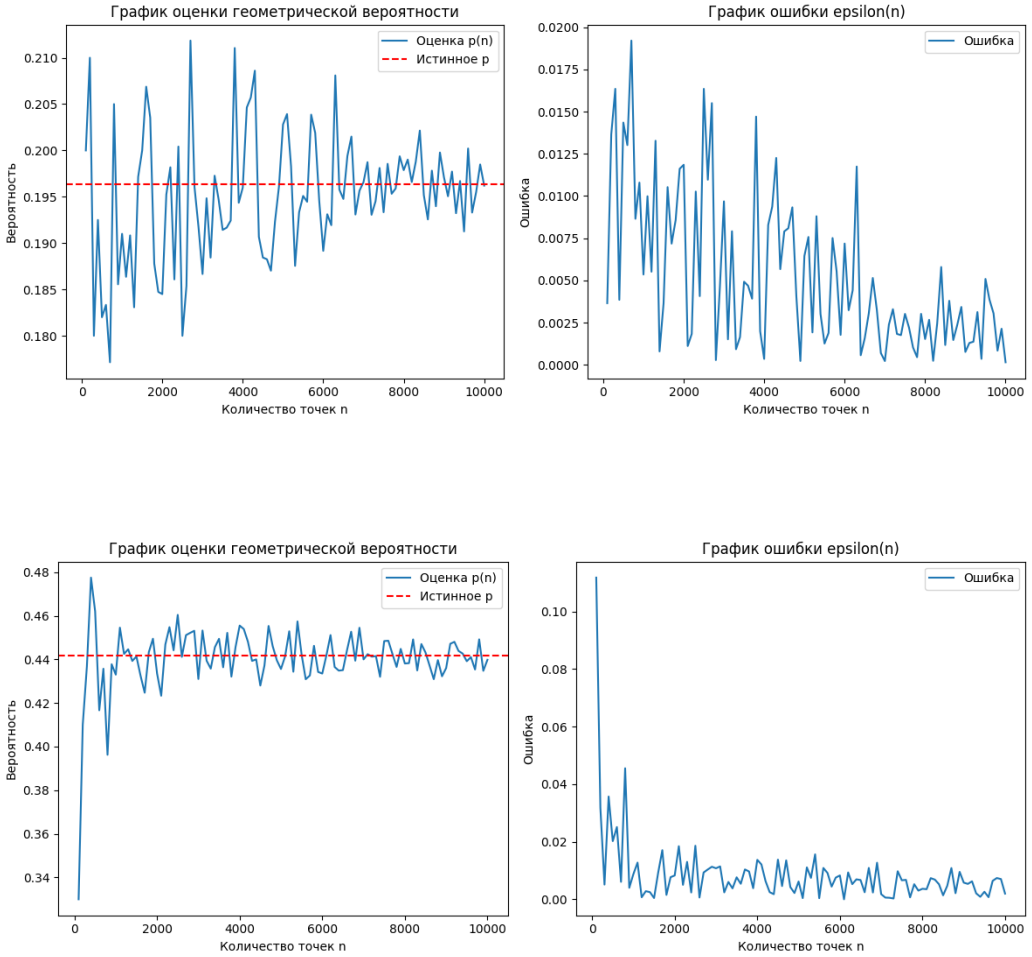
2.7 Шаг 7: Вычисление и построение графика ошибки $\varepsilon(n)$

Описание: Ошибка оценки определяется как абсолютная разность между оценкой и истинным значением:

$$\varepsilon(n) = \left|\hat{p}(n) - p\right|.$$

Для каждого значения n вычисляется соответствующая ошибка. **Промежуточный результат:** Получена последовательность значений ошибки $\varepsilon(n)$, отражающая улучшение точности оценки с увеличением числа точек. **Анализ:** График ошибки показывает, что при росте n значение $\varepsilon(n)$ уменьшается, что соответствует теоретическому соотношению $\varepsilon(n) \propto \frac{1}{\sqrt{n}}$.

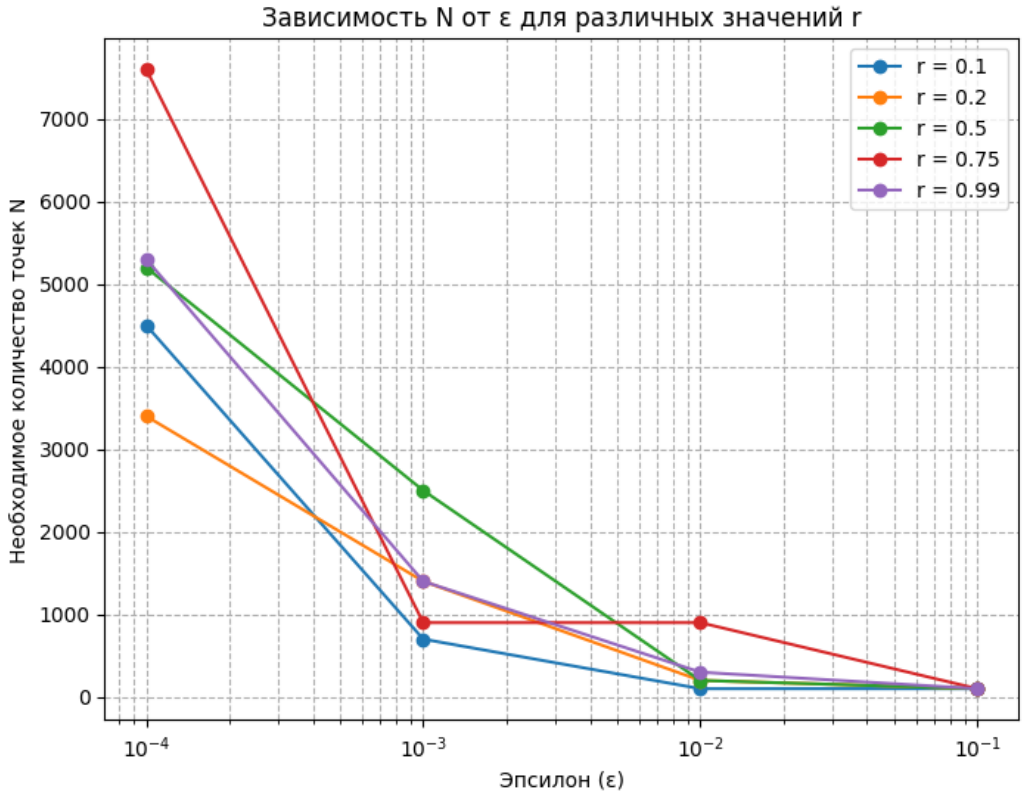
Пример для радиуса 0.5 и 0.75:



2.8 Шаг 8: Определение необходимого количества точек $N(\varepsilon)$

Зафиксировав последовательность значений точности ε, для каждого значения определяется минимальное n, при котором достигается условие

$\varepsilon(n) \leq \varepsilon$. Полученная зависимость $N(\varepsilon)$ показывает, сколько точек необходимо для достижения заданной точности. **Анализ:** Такой анализ позволяет оценить эффективность метода и определить оптимальное число точек для практических вычислений.



3 Пример кода на Python

Ниже приведён пример реализации расчета геометрической вероятности с использованием метода Монте-Карло:

Листинг 1: Пример кода для расчета геометрической вероятности

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

a = 1.0
r_list = [0.1, 0.2, 0.5, 0.75, 0.99]
N = 10000

for r in r_list:
    print("␣", r)
    #
    p_true = (np.pi * r**2) / (4 * a**2)

    x = np.random.uniform(-a, a, N)
    y = np.random.uniform(-a, a, N)
    inside = (x**2 + y**2) <= r**2
    p_estimate = np.sum(inside) / N

    print("␣:", p_true)
    print("␣:", p_estimate)

    points = np.arange(100, N+1, 100)
    p_estimates = []
    errors = []

    #         n
    for n in points:
        x = np.random.uniform(-a, a, n)
        y = np.random.uniform(-a, a, n)
        inside = (x**2 + y**2) <= r**2
        p_est = np.sum(inside) / n
        p_estimates.append(p_est)
        errors.append(abs(p_est - p_true))

    #    2   2
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

    #
    ax1.plot(points, p_estimates, label="␣p(n)")
    ax1.axhline(y=p_true, color='r', linestyle='--', label="␣p")
    ax1.set_xlabel("␣␣n")
    ax1.set_ylabel("")
    ax1.set_title("␣␣␣")
```

```
    ax1.legend()

    #
    ax2.plot(points, errors, label="")
    ax2.set_xlabel("␣␣n")
    ax2.set_ylabel("")
    ax2.set_title("␣␣epsilon(n)")
    ax2.legend()

    plt.tight_layout()
    plt.show()


epsilons = [10**(-i) for i in range(1, 5)]

results = {} #   ,   r    N  epsilons

#
for r in r_list:
    p_true = (np.pi * r**2) / (4 * a**2)
    N_list = []

    #   eps    ,  |p_est - p_true| < eps
    for eps in epsilons:
        count = 100 #   100
        while True:
            x = np.random.uniform(-a, a, count)
            y = np.random.uniform(-a, a, count)
            inside = (x**2 + y**2) <= r**2
            p_est = np.sum(inside) / count #

            if abs(p_est - p_true) < eps:
                N_list.append(count)
                break
            else:
                count += 100 #
                if count > 1e6: #
                    N_list.append(count)
                    break
    results[r] = N_list

plt.figure(figsize=(8, 6))
for r in r_list:
    plt.plot(epsilons, results[r], marker='o', label=f"r␣=␣{r}")

plt.xlabel("␣()")
plt.ylabel("␣␣␣N")
plt.title("␣N␣␣␣␣␣␣r")
plt.xscale('log') #     x
```

```
plt.legend()
plt.grid(True, which="both", linestyle='--')
plt.show()
```

4 Заключение

В ходе работы я релиазовал метод Монте-Карло для оценки геометрической вероятности попадания точки в круг $A(r)$, вписанный в квадрат Ω. Полученные результаты показывают, что:

- Оценка $\hat{p}(n)$ сходится к теоретически вычисленному значению p при увеличении числа точек.
- Ошибка $\varepsilon(n)$ уменьшается с ростом выборки, что подтверждает закон больших чисел.
- Анализ зависимости $N(\varepsilon)$ позволяет определить оптимальное число точек, необходимое для достижения заданной точности.

Таким образом, проведённый эксперимент демонстрирует корректность теоретических выкладок и эффективность применения метода Монте-Карло для расчёта геометрической вероятности.