Задача A1

Зарина Рамазанова

SET 3, ноябрь 2025

1 Реализованный алгоритм

Для решения задачи был реализован алгоритм, работоспособность которого проверена на codeforces. Сам код можно увидеть ниже, пояснения и комментарии в коде.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <random>

// фукнция для выяснения, находится ли точка внутри области пересечения кругов
bool isInside(double x, double y,
    double x1, double y1, double r1,
    double x2, double y2, double r2,
    double x3, double y3, double r3) {

    return (x - x1)*(x - x1) + (y - y1)*(y - y1) <= r1 * r1
    && (x - x2)*(x - x2) + (y - y2)*(y - y2) <= r2 * r2
    && (x - x3)*(x - x3) + (y - y3)*(y - y3) <= r3 * r3;
}

int main() {
    std::random_device rd;
    std::mt19937 gen(rd());

    double x1, x2, x3, y1, y2, y3, r1, r2, r3;
    std::cin >> x1 >> y1 >> r1 >> x2 >> y2 >> r2 >> x3 >> y3 >> r3;;

    // находим область, в которой находятся все окружностиширокая( область из условия задачи)
    double a2 = std::max(x1 + r1, std::max(x2 + r2, x3 + r3));
    double a1 = std::max(x1 - r1, std::max(x2 - r2, x3 - r3));
    double b2 = std::max(y1 + r1, std::max(y2 + r2, y3 + r3));
    double b1 = std::max(y1 - r1, std::max(y2 - r2, y3 - r3));

    std::uniform_real_distribution<double> dist1(a1, a2);
    std::uniform_real_distribution<double> dist2(b1, b2);
    int inside = 0;
    int n = 1000000;
    for (int i = 0; i < n; i++) {
        double x = dist1(gen);
        double y = dist2(gen);

        if (isInside(x, y, x1, y1, r1, x2, y2, r2, x3, y3, r3)) {
            inside++;
        }
    }
    // находим площадь широкой области
    double s_rect = (a2 - a1) * (b2 - b1);
    double a = static_cast<double>(inside) / static_cast<double>(n); ;

    double result = s_rect * a;
    std::cout << result;
}
```

2 Эксперимент

Для проведения эксперимента код был модифицирован в соответствии с условием эксперимента. Для удобства данные эксперимента записаны в csv-файл, который можно найти в репозитории с данными по этой задаче.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <random>
#include <fstream>
#include <cmath>

bool isInside(double x, double y,
    double x1, double y1, double r1,
    double x2, double y2, double r2,
    double x3, double y3, double r3) {

    return (x - x1)*(x - x1) + (y - y1)*(y - y1) <= r1*r1
        && (x - x2)*(x - x2) + (y - y2)*(y - y2) <= r2*r2
        && (x - x3)*(x - x3) + (y - y3)*(y - y3) <= r3*r3;
}

double computeExactArea() {
    return 0.25 * std::numbers::pi + 1.25 * asin(0.8) - 1.0;
}

// Широкая область
double wideMonteCarlo(int numPoints) {
    double x1 = 1.0, y1 = 1.0, r1 = 1.0;
    double x2 = 1.5, y2 = 2.0, r2 = sqrt(5.0) / 2.0;
    double x3 = 2.0, y3 = 1.5, r3 = sqrt(5.0) / 2.0;

    double a2 = std::max(x1 + r1, std::max(x2 + r2, x3 + r3));
    double a1 = std::min(x1 - r1, std::min(x2 - r2, x3 - r3));
    double b2 = std::max(y1 + r1, std::max(y2 + r2, y3 + r3));
    double b1 = std::min(y1 - r1, std::min(y2 - r2, y3 - r3));

    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<double> dist_x(a1, a2);
    std::uniform_real_distribution<double> dist_y(b1, b2);

    int inside = 0;
    for (int i = 0; i < numPoints; i++) {
        double x = dist_x(gen);
        double y = dist_y(gen);

        if (isInside(x, y, x1, y1, r1, x2, y2, r2, x3, y3, r3)) {
            inside++;
        }
    }

    double rect_area = (a2 - a1) * (b2 - b1);
    return rect_area * static_cast<double>(inside) / static_cast<double>(numPoints);
}

// Узкая область
double narrowMonteCarlo(int numPoints) {
    double x1 = 1.0, y1 = 1.0, r1 = 1.0;
    double x2 = 1.5, y2 = 2.0, r2 = sqrt(5.0) / 2.0;
    double x3 = 2.0, y3 = 1.5, r3 = sqrt(5.0) / 2.0;

    // Из рисунка видно, что пересечение находится примерно в [0.8, 2.2] на[0.8, 2.2]
    double a1 = 0.8, a2 = 2.2;  // хграницы-
```

```
    double b1 = 0.8, b2 = 2.2;  // угранцы-

    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<double> dist_x(a1, a2);
    std::uniform_real_distribution<double> dist_y(b1, b2);

    int inside = 0;
    for (int i = 0; i < numPoints; i++) {
        double x = dist_x(gen);
        double y = dist_y(gen);

        if (isInside(x, y, x1, y1, r1, x2, y2, r2, x3, y3, r3)) {
            inside++;
        }
    }

    double rect_area = (a2 - a1) * (b2 - b1);
    return rect_area * static_cast<double>(inside) / static_cast<double>(numPoints);
}

int main() {
    // Вычисляем точную площадь
    double exact_area = computeExactArea();

    // Записываем данные экспериментов
    std::ofstream dataFile("experiment_data.csv");
    dataFile << "rectangle_type,num_points,computed_area,relative_error\n";

    for (int n = 100; n <= 100000; n += 500) {
        // Широкая область
        double wide_area = wideMonteCarlo(n);
        double wide_error = std::abs(wide_area - exact_area) / exact_area;

        // Узкая область
        double narrow_area = narrowMonteCarlo(n);
        double narrow_error = std::abs(narrow_area - exact_area) / exact_area;

        dataFile << "wide," << n << "," << wide_area << "," << wide_error << "\n";
        dataFile << "narrow," << n << "," << narrow_area << "," << narrow_error << "\n";
    }

    dataFile.close();

    return 0;
}
```

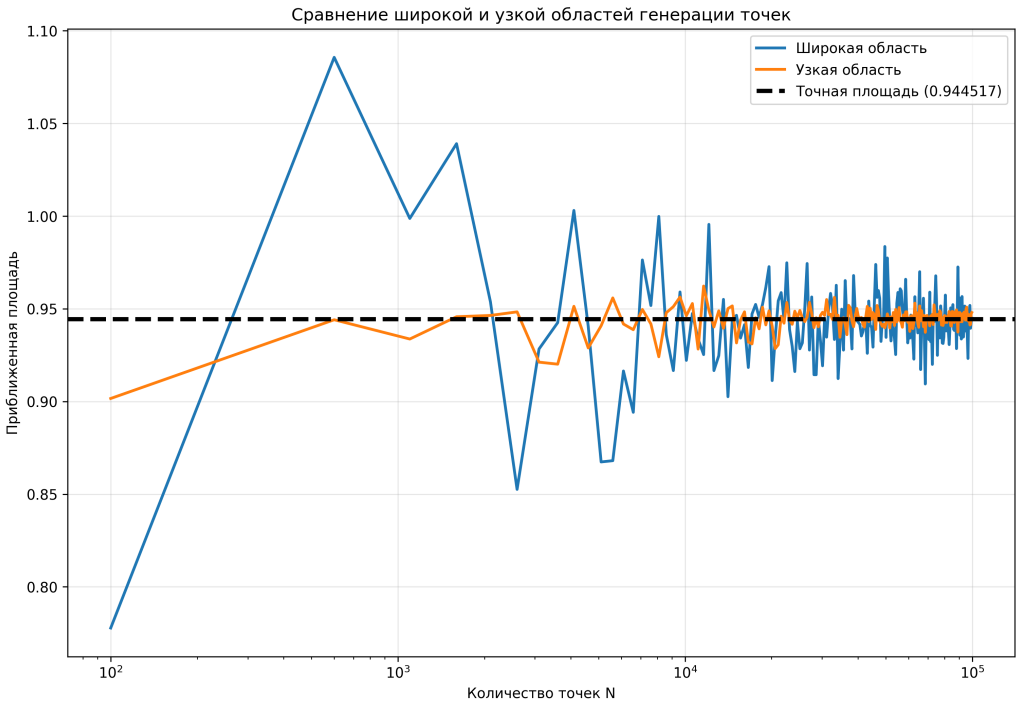
Графики

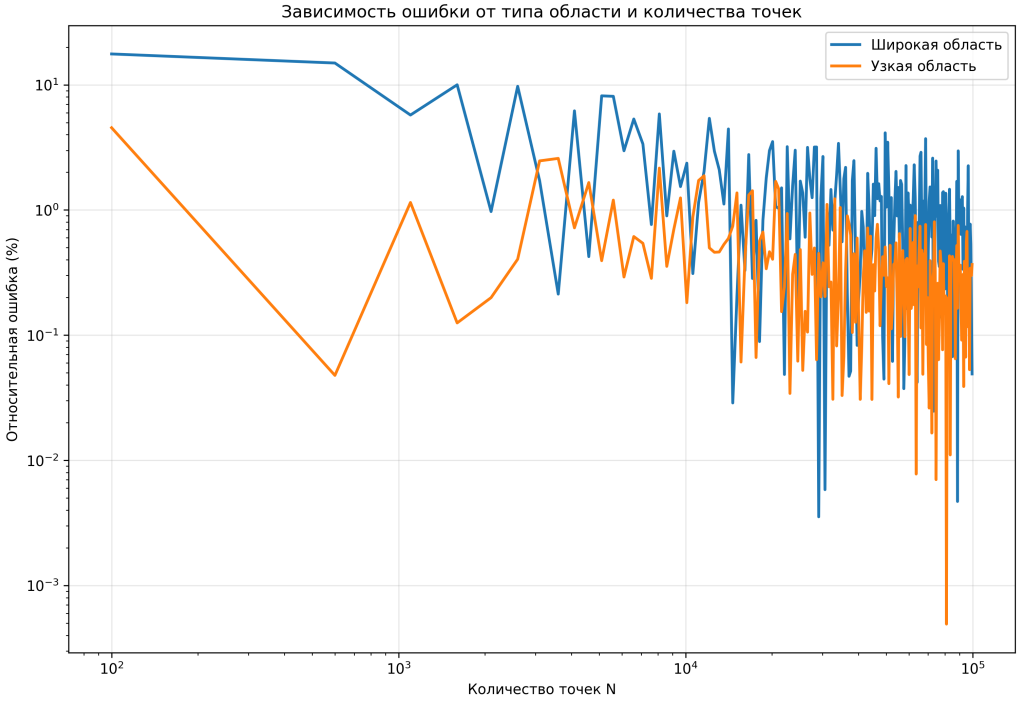
Чтобы наглядно представить результаты эксперимента, был написан следующий код на Python:

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

# Читаем данные
data = pd.read_csv('experiment_data.csv')

# Точная площадь
exact_area = 0.25 * np.pi + 1.25 * np.arcsin(0.8) - 1.0
print(fТочная" площадь: {exact_area:.10f}")

# График 1: Приближенная площадь vs количество точек
plt.figure(figsize=(12, 8))

```

```
for rect_type in data['rectangle_type'].unique():
    rect_data = data[data['rectangle_type'] == rect_type]
    label = Широкая' область' if rect_type == 'wide' else Узкая' область'
    plt.plot(rect_data['num_points'], rect_data['computed_area'],
             label=label, linewidth=2)

plt.axhline(y=exact_area, color='black', linestyle='--',
           label=fТочная' площадь ({exact_area:.6f})', linewidth=3)
plt.xlabelКоличество(' точек N')
plt.ylabelПриближенная(' площадь')
plt.titleСравнение(' широкой и узкой областей генерации точек')
plt.legend()
plt.grid(True, alpha=0.3)
plt.xscale('log')
plt.savefig('area_comparison.png', dpi=300, bbox_inches='tight')
plt.show()

# График 2: Относительная ошибка vs количество точек
plt.figure(figsize=(12, 8))

for rect_type in data['rectangle_type'].unique():
    rect_data = data[data['rectangle_type'] == rect_type]
    label = Широкая' область' if rect_type == 'wide' else Узкая' область'
    plt.plot(rect_data['num_points'], rect_data['relative_error'] * 100,
             label=label, linewidth=2)

plt.xlabelКоличество(' точек N')
plt.ylabelОтносительная(' ошибка (%)')
plt.titleЗависимость(' ошибки от типа области и количества точек')
plt.legend()
plt.grid(True, alpha=0.3)
plt.xscale('log')
plt.yscale('log')
plt.savefig('error_comparison.png', dpi=300, bbox_inches='tight')
plt.show()
```

Были получены следующие графики:





3 Вывод

В ходе выполнения работы была исследована эффективность метода Монте-Карло для приближённого вычисления площади пересечения трёх кругов.Сравнивалось два подхода: использования широкой и узкой областей для генерации случайных точек. Анализ показал, что выбор области существенно влияет на точность и скорость сходимости метода. При использовании узкой области, плотно ограничивающей целевую фигуру, удалось достичь значительно более высокой точности вычислений при том же количестве точек. Это объясняется тем, что в узкой области большая доля точек попадает в пересечение кругов, что повышает эффективность использования вычислительных ресурсов. Напротив, при генерации точек в широкой области большая их часть оказывается вне целевой фигуры, что приводит к увеличению погрешности и замедлению сходимости.

Эксперименты также подтвердили, что с увеличением количества точек точность метода растёт для обоих подходов, однако узкая область демонстрирует более стабильную и быструю сходимость к точному значению площади. Таким образом, можно сделать вывод о том, что предварительный анализ геометрии задачи и оптимизация области генерации точек являются важными этапами, позволяющими повысить эффективность метода Монте-Карло без дополнительных вычислительных затрат.

4 Прочие данные

Номер посылки на codeforces: 347473237
Ссылка на публичный репозиторий с материалами: `github.com/sunflowerthu/ADS_A1_SET3`