SET 3. Task A1

Algorithms. Algorithm Monte Carlo

ID посылки: `292687525`
GitHub: https://github.com/vasyukov1/HSE-FCS-SE-2-year/tree/main/Algorithms/Homework/SET_3/A1

- `a1i.cpp` - общее решение
- `a1_lms.cpp` - решение задачи из ЛМС
- `data_1.txt` - результаты, полученные способом 1
- `data_2.txt` - результаты, полученные способом 2
- `graphs.py` - построение графиков

Общее решение

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <random>
using std::cin;
using std::cout;
using std::vector;
using std::pair;
using std::pow;
using std::sqrt;

bool f(double cx, double cy, double r, double x, double y);
void areaMonteCarlo(vector<double>& a, vector<double>& b, vector<double>& c, bool method);

int main() {
        // Get the parameters of the equation of the circle
    vector<double> a;
    vector<double> b;
    vector<double> c;
    double el;
    for (int i = 0; i < 3; ++i) {
        cin >> el;
        a.push_back(el);
    }
    for (int i = 0; i < 3; ++i) {
        cin >> el;
        b.push_back(el);
    }
    for (int i = 0; i < 3; ++i) {
        cin >> el;
        c.push_back(el);
    }
    // Call the function, which calculate the area
    areaMonteCarlo(a, b, c, true);
    return 0;
}

bool f(double cx, double cy, double r, double x, double y) {
        // The function shows if the point is located in the circle
    return pow((x - cx), 2) + pow((y - cy), 2) <= pow(r, 2);
}

void areaMonteCarlo(vector<double>& a, vector<double>& b, vector<double>& c, bool method) {
        // The function returns the area

    // Find the bounds, where will be choose points
    double x_min, x_max, y_min, y_max;
    if (method) {
        x_min = std::max(std::max(a[0] - a[2], b[0] - b[2]), c[0] - c[2]);
        x_max = std::min(std::min(a[0] + a[2], b[0] + b[2]), c[0] + c[2]);
        y_min = std::max(std::max(a[1] - a[2], b[1] - b[2]), c[1] - c[2]);
        y_max = std::min(std::min(a[1] + a[2], b[1] + b[2]), c[1] + c[2]);
    } else {
        x_min = std::min(std::min(a[0] - a[2], b[0] - b[2]), c[0] - c[2]);
        x_max = std::max(std::max(a[0] + a[2], b[0] + b[2]), c[0] + c[2]);
        y_min = std::min(std::min(a[1] - a[2], b[1] - b[2]), c[1] - c[2]);
        y_max = std::max(std::max(a[1] + a[2], b[1] + b[2]), c[1] + c[2]);
```

```
    }

    // Functions for generation of random numbers
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> distrib_x(x_min, x_max);
    std::uniform_real_distribution<> distrib_y(y_min, y_max);

    // In 'result' will be store the area
    // In 'counter' will be store the number of calculations
    double result = 0;
    int counter = 0;

    // Loop for calculation with different numbers of points
    for (int i = 100; i <= 100000; i += 500) {
            // In 'M' will be store the number of points which are located in area
        double M = 0;

        // Nake points and check them
        for (int j = 0; j < i; ++j) {
                // Generation coordinate x and y for the point
            double x = distrib_x(gen);
            double y = distrib_y(gen);

            // Chech that point is located in circles
            bool circle1 = f(a[0], a[1], a[2], x, y);
            bool circle2 = f(b[0], b[1], b[2], x, y);
            bool circle3 = f(c[0], c[1], c[2], x, y);

            // If the point is located in all circles, it located in the area
            if (circle1 && circle2 && circle3) {
                ++M;
            }
        }

        // S - the square of rectangle, where were taken points
        // S_rec - the square of the area
        double S = (x_max - x_min) * (y_max - y_min);
        double S_rec = (M / i) * S;

        // Update result and counter
        result += S_rec;
        ++counter;
    }
    // Final square if average of all result
    cout << result / counter << '\n';
}
```

Аналогичный код, но с уже имеющимися данными для задачи

```
#include <iostream>
#include <vector>
#include <cmath>
#include <random>
using std::cin;
using std::cout;
using std::vector;
using std::pair;
using std::pow;
using std::sqrt;

bool f(double cx, double cy, double r, double x, double y);
void areaMonteCarlo(vector<double>& a, vector<double>& b, vector<double>& c, bool method);

int main() {
    vector<double> a;
    vector<double> b;
    vector<double> c;
    for (int i = 0; i < 3; ++i) {
        a.push_back(1);
        a.push_back(1);
        a.push_back(1);
    }
    for (int i = 0; i < 3; ++i) {
        b.push_back(1.5);
        b.push_back(2);
```

```
        b.push_back(sqrt(5) / 2);
    }
    for (int i = 0; i < 3; ++i) {
        c.push_back(2);
        c.push_back(1.5);
        c.push_back(sqrt(5) / 2);
    }
    areaMonteCarlo(a, b, c, true);
    return 0;
}

bool f(double cx, double cy, double r, double x, double y) {
    return pow((x - cx), 2) + pow((y - cy), 2) <= pow(r, 2);
}

void areaMonteCarlo(vector<double>& a, vector<double>& b, vector<double>& c, bool method) {
    double x_min, x_max, y_min, y_max;
    if (method) {
        x_min = std::max(std::max(a[0] - a[2], b[0] - b[2]), c[0] - c[2]);
        x_max = std::min(std::min(a[0] + a[2], b[0] + b[2]), c[0] + c[2]);
        y_min = std::max(std::max(a[1] - a[2], b[1] - b[2]), c[1] - c[2]);
        y_max = std::min(std::min(a[1] + a[2], b[1] + b[2]), c[1] + c[2]);
    } else {
        x_min = std::min(std::min(a[0] - a[2], b[0] - b[2]), c[0] - c[2]);
        x_max = std::max(std::max(a[0] + a[2], b[0] + b[2]), c[0] + c[2]);
        y_min = std::min(std::min(a[1] - a[2], b[1] - b[2]), c[1] - c[2]);
        y_max = std::max(std::max(a[1] + a[2], b[1] + b[2]), c[1] + c[2]);
    }

    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> distrib_x(x_min, x_max);
    std::uniform_real_distribution<> distrib_y(y_min, y_max);
    for (int i = 100; i <= 100000; i += 500) {
    double M = 0;

    for (int j = 0; j < i; ++j) {
        double x = distrib_x(gen);
        double y = distrib_y(gen);

        bool circle1 = f(a[0], a[1], a[2], x, y);
        bool circle2 = f(b[0], b[1], b[2], x, y);
        bool circle3 = f(c[0], c[1], c[2], x, y);

        if (circle1 && circle2 && circle3) {
            ++M;
        }
    }

    double S = (x_max - x_min) * (y_max - y_min);
    double S_rec = (M / i) * S;

        cout << i << ' ' << S_rec << '\n';
    }
}
```

В файле `a1_data_1.txt` хранятся результаты вычисления площади первым способом, когда границы берутся по наибольшим координатам окружностей.

В файле `a1_data_2.txt` хранятся результаты вычисления площади первым способом, когда границы берутся по наименьшим координатам окружностей.

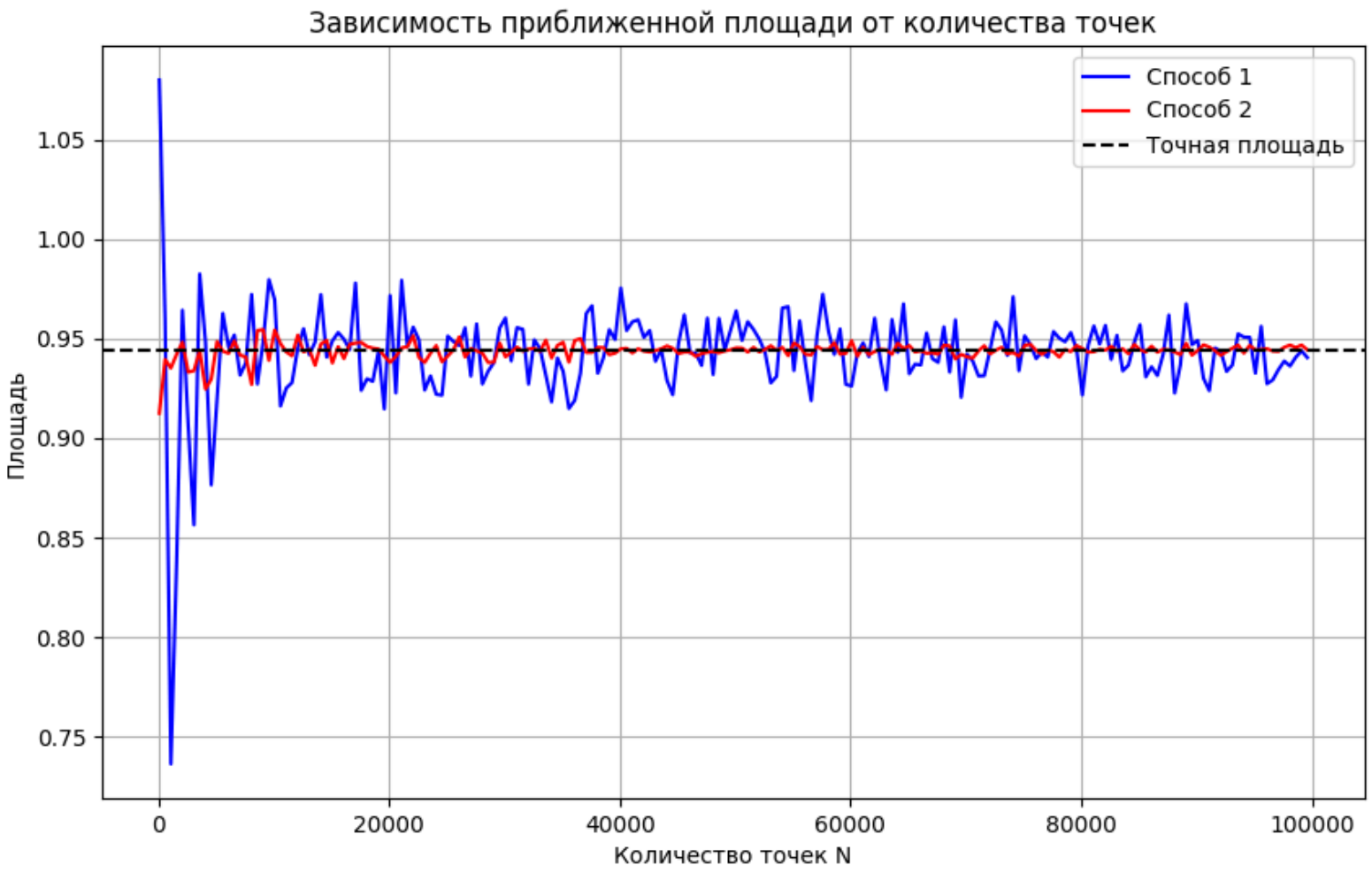

График 1

Зависимость приближённой площади от количества точек.

Синее - первый способ.

Красное - второй способ.

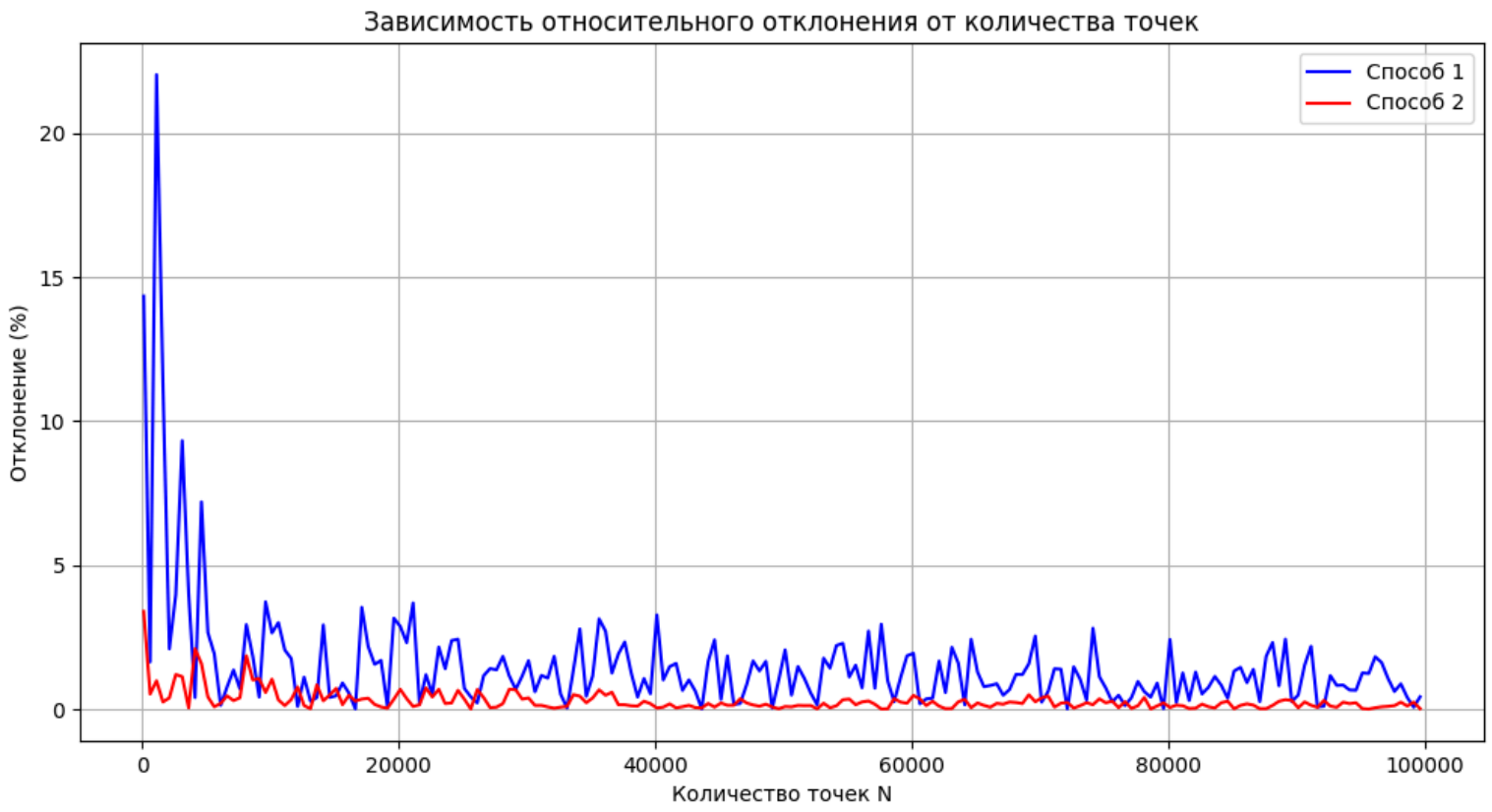

График 2

Зависимость относительного отклонения от количества точек.

Синее - первый способ.

Красное - второй способ.

Чем больше количество точек используется для нахождения площади, тем точнее значение, потому что покрывается большее пространство. Также, чем меньше область выбора точек, тем точнее значение, так как вероятность выбора точки, находящейся в области, выше.