id посылки: 348645867

гит: https://github.com/Polinushkin/SET3-A1.git

- a1i.cpp - реализация в общем виде
- a1i_2.cpp - реализация с конкретными данными для задачи
- (output.txt - результаты, полученные в результате обоих методов)
- (output.py - для разбиения txt по 2-м csv)
- data_method1.csv - результаты, полученные 1 способом
- data_method2.csv - результаты, полученные 2 способом
- plot.py - для построения графиков

реализация:

в общем виде:

a1i.cpp

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

using namespace std;

#define ll long long
#define ld long double
#define FOR(i, s, f) for(ll (i) = (s); (i) < (f); (i)++)

random_device rd;
mt19937 eng(rd());
uniform_real_distribution<double> distr_double(0, 1);

ld genD(){return distr_double(eng);}

struct Circle{
    ld x, y, r;
    Circle() = default;
    Circle(ld x, ld y, ld r): x(x), y(y), r(r){}
    friend istream& operator>>(istream& in, Circle &A);
    bool is_inside(ld px, ld py){
        return ((x - px) * (x - px) + (y - py) * (y - py)) <= r * r;
    }
};
istream& operator>>(istream &in, Circle &A){
    in >> A.x >> A.y >> A.r;
    return in;
}

```

```cpp
void solve(){
    Circle A, B, C;
    cin >> A >> B >> C;
    ld maxr = max(A.r, max(B.r, C.r));
    ll cntin = 0, cnttest = 10000000;
    FOR(i, 0, cnttest){
        ld genx = genD() * (2 * maxr + 1) - maxr, geny = genD() * (2 * maxr + 1) - maxr;
        if (A.is_inside(genx, geny) && B.is_inside(genx, geny) && C.is_inside(genx, geny)) cntin++;
    }
    cout << (2 * maxr + 1) * (2 * maxr + 1) * cntin / cnttest;
}

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    cout.precision(64);
    solve();
    return 0;
}

```

с конкретными данными:

a1i_2.cpp output.py plot.py 2

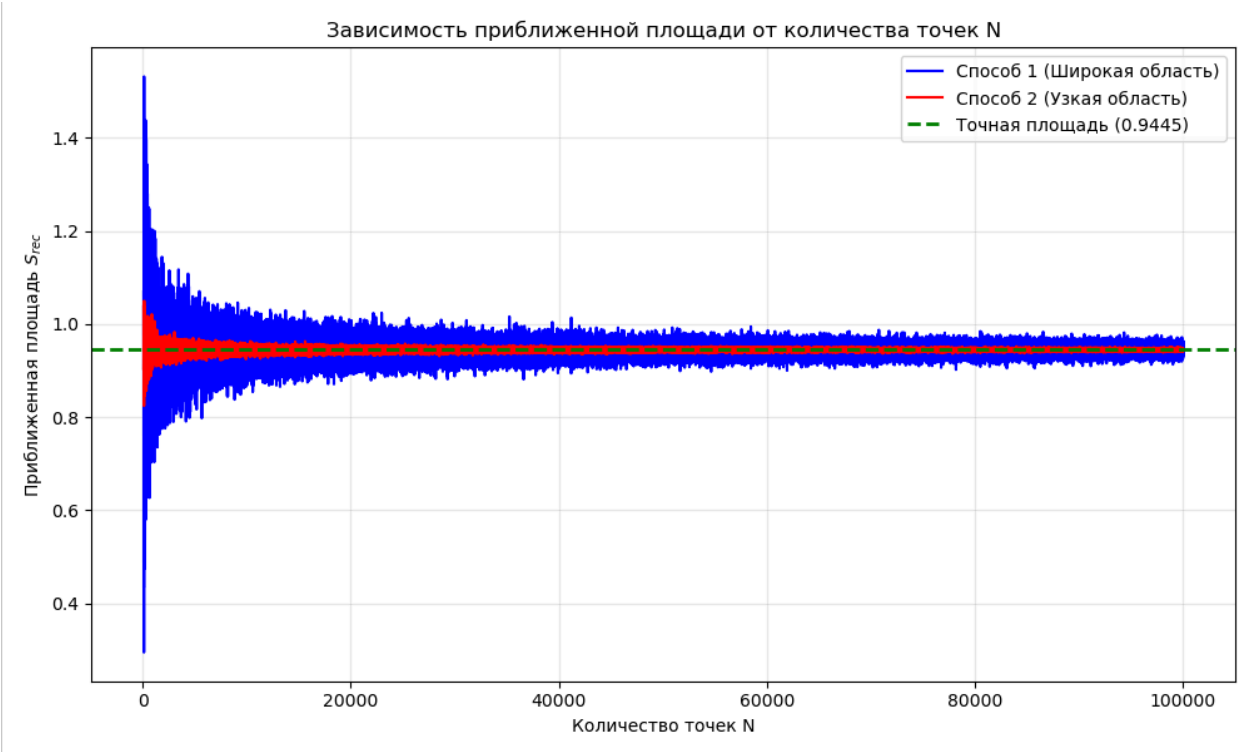
a1i_2.cpp

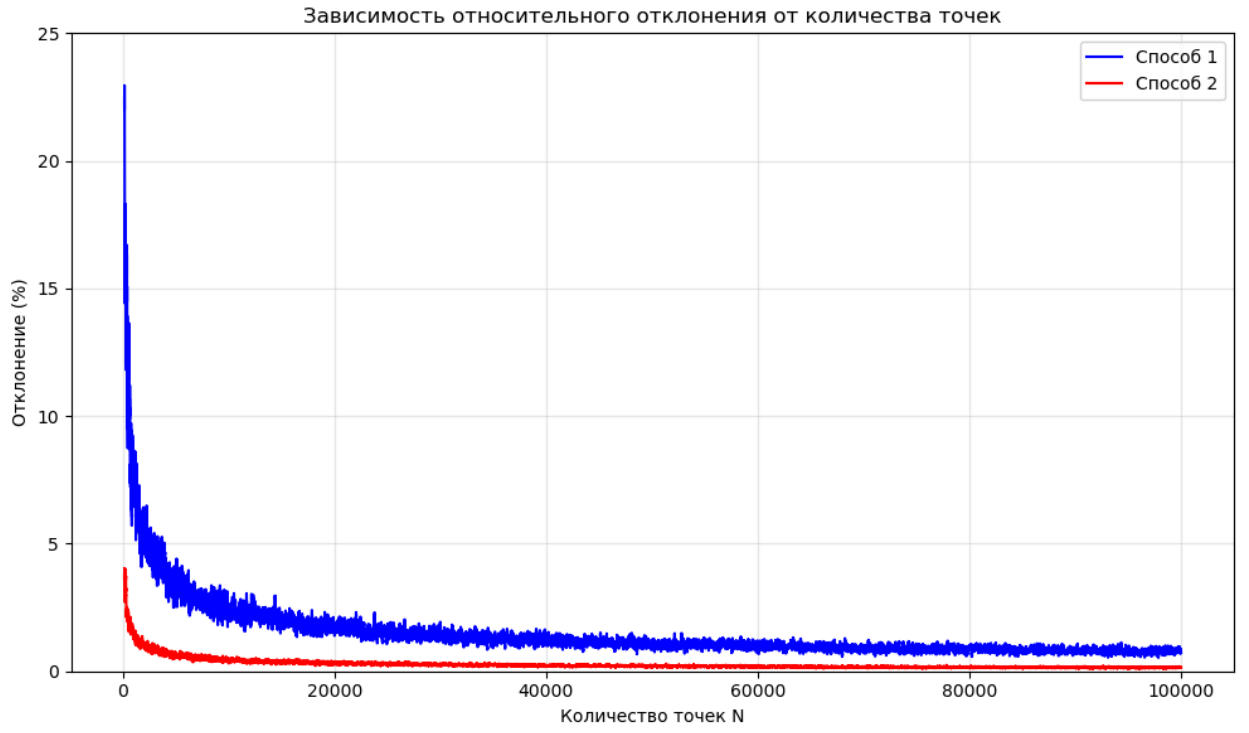
```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>
#include <cmath>

using namespace std;

#define ll long long
#define ld long double
#define FOR(i, s, f) for(ll (i) = (s); (i) < (f); (i)++)

random_device rd;
mt19937 eng(rd());

struct Circle{
    ld x, y, r;
    Circle() = default;
    Circle(ld x, ld y, ld r): x(x), y(y), r(r){}
    bool is_inside(ld px, ld py){
        return ((x - px) * (x - px) + (y - py) * (y - py)) <= r * r;
    }
};

void areaMonteCarlo(const vector<Circle>& circles, bool method, int startN, int endN, int stepN) {
    ld sqrt5_half = sqrt(5.0L) / 2.0L;
    Circle A(1.0L, 1.0L, 1.0L);
    Circle B(1.5L, 2.0L, sqrt5_half);
    Circle C(2.0L, 1.5L, sqrt5_half);
    ld x_min, x_max, y_min, y_max;

```

```
    if (method) { // Способ 1
        x_min = min({A.x - A.r, B.x - B.r, C.x - C.r});
        x_max = max({A.x + A.r, B.x + B.r, C.x + C.r});
        y_min = min({A.y - A.r, B.y - B.r, C.y - C.r});
        y_max = max({A.y + A.r, B.y + B.r, C.y + C.r});
        cout << "# Method 1\n";
    } else { // Способ 2
        x_min = max({A.x - A.r, B.x - B.r, C.x - C.r});
        x_max = min({A.x + A.r, B.x + B.r, C.x + C.r});
        y_min = max({A.y - A.r, B.y - B.r, C.y - C.r});
        y_max = min({A.y + A.r, B.y + B.r, C.y + C.r});
        if (x_min >= x_max || y_min >= y_max) {
            cerr << "Invalid bounds for Method 2\n";
            cerr << "x_min: " << x_min << ", x_max: " << x_max << "\n";
            cerr << "y_min: " << y_min << ", y_max: " << y_max << "\n";
            return;
        }
        cout << "# Method 2\n";
    }
    uniform_real_distribution<ld> distr_x(x_min, x_max);
    uniform_real_distribution<ld> distr_y(y_min, y_max);
```

```
    FOR(N, startN, endN + 1) {
        // Сброс счётчиков
        ll cntin = 0;
        FOR(i, 0, N) {
            ld genx = distr_x(eng);
            ld geny = distr_y(eng);
            if (A.is_inside(genx, geny) && B.is_inside(genx, geny) && C.is_inside(genx, geny)) {
                cntin++;
            }
        }
        ld S_rect = (x_max - x_min) * (y_max - y_min);
        ld S_rec = (static_cast<ld>(cntin) / static_cast<ld>(N)) * S_rect;
        cout << N << " " << S_rec << "\n";
    }
    cout << endl;
}

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    cout.precision(17);
    int startN = 100;
    int endN = 100000;
    int stepN = 500;
    areaMonteCarlo({}, true, startN, endN, stepN);
    areaMonteCarlo({}, false, startN, endN, stepN);
    return 0;
}
```

на графиках:


Зависимость приближенной площади от количества точек N
Способ 1 (Широкая область)
Способ 2 (Узкая область)
Точная площадь (0.9445)
Приближенная площадь S_{rec}
1.4
1.2
1.0
0.8
0.6
0.4
0
20000
40000
60000
80000
100000
Количество точек N


Зависимость относительного отклонения от количества точек
Способ 1
Способ 2
Отклонение (%)
25
20
15
10
5
0
0
20000
40000
60000
80000
100000
Количество точек N

Анализ графиков:

- Оба метода соответствуют теории метода Монте-Карло и показывают сходимость, при увеличении N приближенное значение площади S_rec колеблется вокруг точного значения S_exact, и эти колебания уменьшаются
- Способ 2 показывает меньшую относительную ошибку по сравнению со способом 1 при одинаковом N, как видно на графике
- Средняя относительная ошибка способа 2 - всего 0.28%, что гораздо меньше, чем у способа 1 - 1.5%. И стандартное отклонение ошибки для способа 2 - 0.38% тоже меньше, чем для способа 1 - 2.04%, что указывает на большую стабильность результатов (посчитано при построении графика)

Вывод:

Оба способа позволяют получить приближенную площадь пересечения трех кругов, которая сходится к точному значению S_exact ~ 0.9445 при увеличении числа точек N. Но способ 2 более эффективен. Поскольку область генерации точек ближе к реальной области пересечения, вероятность того, что случайно выбранная точка попадет внутрь целевой области, выше. Это приводит к более точной оценке с тем же количеством точек N. Т. е. чем меньше выбранная для генерации точек область и чем больше количество выбранных N, тем точнее полученное значение.