Id_cf: 348171474

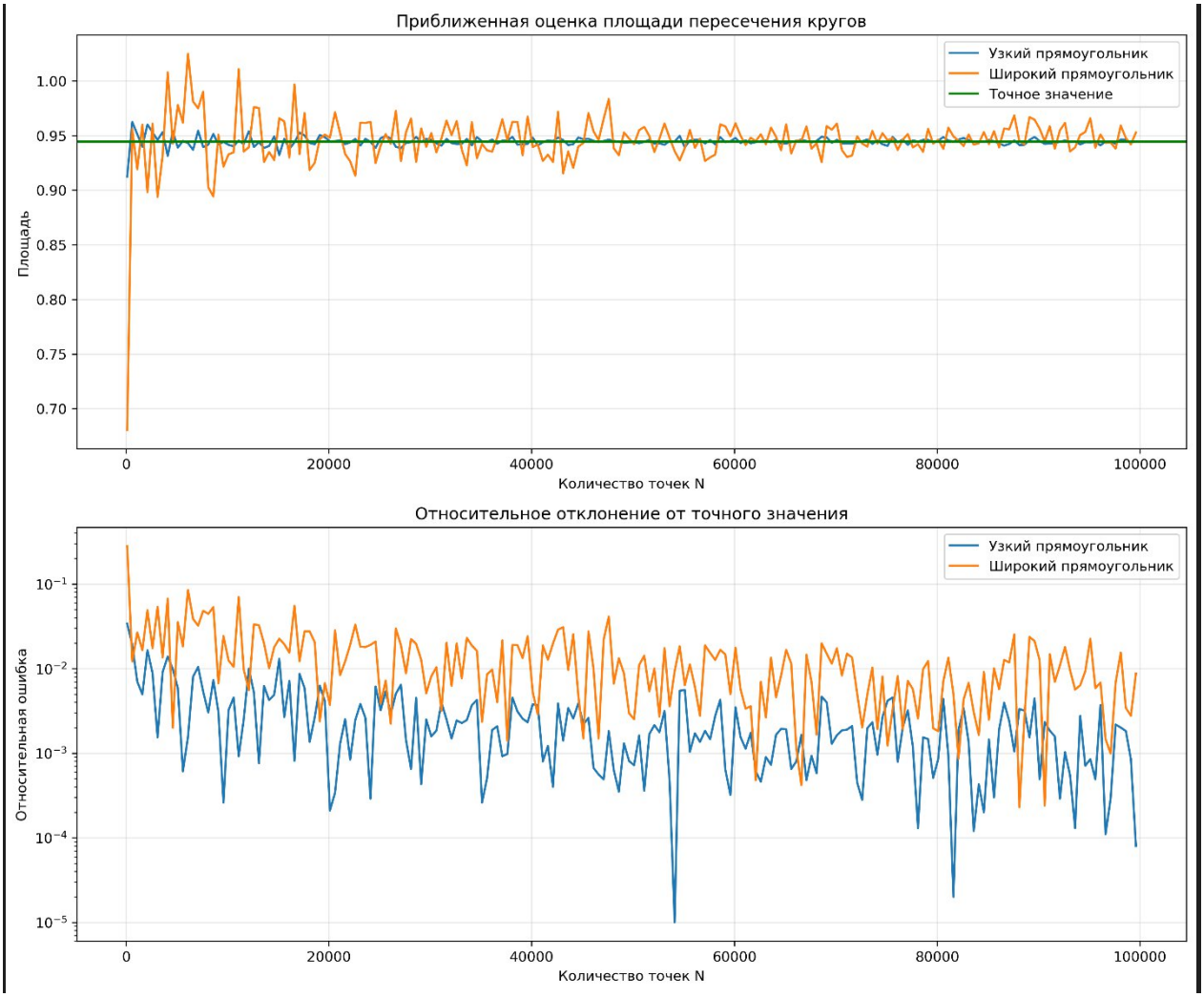
Репо: https://github.com/leolazzz/dsa_hse_set3

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <set>
#include <string>
#include <stack>
#include <cmath>
#include <numeric>
#include <map>
#include <iomanip>

using ll = long long;
using ld = long double;

struct roun {
    ld x, y, r;
};

ld real_area() {
    return 0.5 * acos(0.0) + 1.25 * std::asin(0.8) - 1.0; // 2 * acos(0.0) = pi
}

//narrow
ld solve1(roun a, roun b, roun c, ll n) {
    ld minx = std::max(std::max(a.x - a.r, b.x - b.r), c.x - c.r);
    ld maxx = std::min(std::min(a.x + a.r, b.x + b.r), c.x + c.r);
    ld miny = std::max(std::max(a.y - a.r, b.y - b.r), c.y - c.r);
    ld maxy = std::min(std::min(a.y + a.r, b.y + b.r), c.y + c.r);
    ld s = (maxx - minx) * (maxy - miny);
    ll k = 0;
    for (ll i = 0; i < n; ++i) {
        ld dx = minx + (maxx - minx) * (ld)rand() / (ld)RAND_MAX;
        ld dy = miny + (maxy - miny) * (ld)rand() / (ld)RAND_MAX;
        bool fl = 1;
        ld qx = dx - a.x;
        ld qy = dy - a.y;
        if (qx * qx + qy * qy > a.r * a.r) {
            fl = 0;
        }
        qx = dx - b.x;
        qy = dy - b.y;
        if (qx * qx + qy * qy > b.r * b.r) {
            fl = 0;
        }
        qx = dx - c.x;
        qy = dy - c.y;
        if (qx * qx + qy * qy > c.r * c.r) {
            fl = 0;
        }
        if (fl) {
            k++;
        }
    }
    return s * (ld)k / (ld)n;
}

//width
ld solve2(roun a, roun b, roun c, ll n) {
    ld minx = std::min(std::min(a.x - a.r, b.x - b.r), c.x - c.r);
```

```
    ld maxx = std::max(std::max(a.x + a.r, b.x + b.r), c.x + c.r);
    ld miny = std::min(std::min(a.y - a.r, b.y - b.r), c.y - c.r);
    ld maxy = std::max(std::max(a.y + a.r, b.y + b.r), c.y + c.r);
    ld s = (maxx - minx) * (maxy - miny);
    ll k = 0;
    for (ll i = 0; i < n; ++i) {
        ld dx = minx + (maxx - minx) * (ld)rand() / (ld)RAND_MAX;
        ld dy = miny + (maxy - miny) * (ld)rand() / (ld)RAND_MAX;
        bool fl = 1;
        ld qx = dx - a.x;
        ld qy = dy - a.y;
        if (qx * qx + qy * qy > a.r * a.r) {
            fl = 0;
        }
        qx = dx - b.x;
        qy = dy - b.y;
        if (qx * qx + qy * qy > b.r * b.r) {
            fl = 0;
        }
        qx = dx - c.x;
        qy = dy - c.y;
        if (qx * qx + qy * qy > c.r * c.r) {
            fl = 0;
        }
        if (fl) {
            k++;
        }
    }
    return s * (ld)k / (ld)n;
}
int main() {
    srand(1);
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(NULL);
    roun a;
    a.x = 1.0;
    a.y = 1.0;
    a.r = 1.0;
    roun b;
    b.x = 1.5;
    b.y = 2.0;
    b.r = std::sqrt(5.0)/ 2.0;
    roun c;
    c.x = 2.0;
    c.y = 1.5;
    c.r = std::sqrt(5.0) / 2.0;
    ld real = real_area();
    std::cout << std::fixed << std::setprecision(5);
    std::cout << "n,area_n,area_w,disp_n,disp_w\n";
    for (ll i = 100; i <= 100000; i += 500) {
        ld s1 = solve1(a, b, c, i);
        ld s2 = solve2(a, b, c, i);
        std::cout << i << "," << s1 << "," << s2 << "," << std::abs(s1 - real) / real << "," << std::abs(s2 - real) / real
<< '\n';
    }
}
```

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

df = pd.read_csv('data.csv')
```

```
_, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

ax1.plot(df['n'], df['area_n'], label='Узкий прямоугольник')
ax1.plot(df['n'], df['area_w'], label='Широкий прямоугольник')

ax1.axhline(y=0.25 * np.pi + 1.25 * np.arcsin(0.8) - 1.0, color='g', label='Точное значение')

ax1.set_xlabel('Количество точек N')
ax1.set_ylabel('Площадь')
ax1.set_title('Приближенная оценка площади пересечения кругов')
ax1.legend()
ax1.grid(True, alpha=0.3)

ax2.plot(df['n'], df['disp_n'], label='Узкий прямоугольник')
ax2.plot(df['n'], df['disp_w'], label='Широкий прямоугольник')

ax2.set_xlabel('Количество точек N')
ax2.set_ylabel('Относительная ошибка')
ax2.set_title('Относительное отклонение от точного значения')
ax2.legend()
ax2.grid(True, alpha=0.3)
ax2.set_yscale('log')

plt.tight_layout()
plt.savefig('monte_carlo_results.png', dpi=300, bbox_inches='tight')
plt.show()
```



Вывод:

Метод Монте-Карло с узкой областью генерации точек показывает лучшую точность по сравнению с широкой областью. При одинаковом количестве точек относительная ошибка для узкой области меньше. С увеличением количества точек ошибка уменьшается. Узкая область быстрее сходится к точному значению и дает более стабильные результаты. Использование оптимально подобранной ограничивающей области существенно повышает эффективность метода Монте-Карло.