A1. Задача трех кругов. Реализация алгоритма Монте-Карло

посылка: 293132442

ссылка на репозиторий: https://github.com/matveevaolga/set3-A1

Код реализации:

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

struct Point {
    double x;
    double y;
};

std::uniform_real_distribution<> getDistribution(double x, double y) {
    std::random_device device;
    return std::uniform_real_distribution<>(x, y);
}

Point getRandomPoint(std::uniform_real_distribution<> distribution_x,
std::uniform_real_distribution<> distribution_y, std::mt19937 &generator) {
    Point point;
    point.x = distribution_x(generator);
    point.y = distribution_y(generator);
    return point;
}

bool isPointInCircle(Point point, double radius, Point center) {
    return radius * radius >= (point.x - center.x) * (point.x - center.x) + (point.y -
center.y) * (point.y - center.y);
}

double getSquare(Point p1, Point p2, Point p3, double r1, double r2, double r3,
std::mt19937 &generator) {
    double l_border = std::min(p1.x - r1, std::min(p2.x - r2, p3.x - r3));
    double d_border = std::min(p1.y - r1, std::min(p2.y - r2, p3.y - r3));
    double r_border = std::max(p1.x + r1, std::max(p2.x + r2, p3.x + r3));
    double u_border = std::max(p1.y + r1, std::max(p2.y + r2, p3.y + r3));
    auto distribution_x = getDistribution(l_border, r_border);
    auto distribution_y = getDistribution(d_border, u_border);
    double S = (u_border - d_border) * (r_border - l_border);
    long long N = 10000000;
    int M = 0;
    int ind = 0;
    while (ind < N) {
        Point point = getRandomPoint(distribution_x, distribution_y, generator);
        if (isPointInCircle(point, r1, p1) && isPointInCircle(point, r2, p2) &&
isPointInCircle(point, r3, p3)) {
```

```
            M++;
        }
        ind++;
    }
    return S * M / N;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    Point point1;
    Point point2;
    Point point3;
    double radius1;
    double radius2;
    double radius3;
    std::cin >> point1.x >> point1.y >> radius1;
    std::cin >> point2.x >> point2.y >> radius2;
    std::cin >> point3.x >> point3.y >> radius3;
    std::random_device device;
    std::mt19937 generator(device());
    double square = getSquare(point1, point2, point3, radius1, radius2, radius3,
generator);
    std::cout << square << std::endl;
    return 0;
}
```

Код, генерирующий данные для построения графиков:

```
#include <iostream>
#include <vector>
#include <cmath>
#include <random>
#include <algorithm>
#include <iomanip>
#define _USE_MATH_DEFINES

struct Point {
    double x;
    double y;
};

std::uniform_real_distribution<> getDistribution(double x, double y) {
    std::random_device device;
    return std::uniform_real_distribution<>(x, y);
}

bool isPointInCircle(Point point, double radius, Point center) {
    return radius * radius >= (point.x - center.x) * (point.x - center.x) + (point.y -
center.y) * (point.y - center.y);
}
```

```cpp
Point getRandomPoint(std::uniform_real_distribution<> distribution_x,
std::uniform_real_distribution<> distribution_y, std::mt19937 &generator) {
    Point point;
    point.x = distribution_x(generator);
    point.y = distribution_y(generator);
    return point;
}

void print(const std::vector<long long>& vec) {
    size_t size = vec.size();
    for (size_t ind = 0; ind < size; ind++) {
        if (ind != size - 1) std::cout << vec[ind] << ", ";
        else std::cout << vec[ind];
    }
    std::cout << std::endl << std::endl;
}

void print(const std::vector<double>& vec) {
    size_t size = vec.size();
    for (size_t ind = 0; ind < size; ind++) {
        if (ind != size - 1) std::cout << vec[ind] << ", ";
        else std::cout << vec[ind];
    }
    std::cout << std::endl << std::endl;
}

double getSquare(Point p1, Point p2, Point p3, double r1, double r2, double r3,
std::mt19937 &generator, long long N) {
    double l_border = std::min(p1.x - r1, std::min(p2.x - r2, p3.x - r3));
    double d_border = std::min(p1.y - r1, std::min(p2.y - r2, p3.y - r3));
    double r_border = std::max(p1.x + r1, std::max(p2.x + r2, p3.x + r3));
    double u_border = std::max(p1.y + r1, std::max(p2.y + r2, p3.y + r3));
    double S = (u_border - d_border) * (r_border - l_border);
    auto distribution_x = getDistribution(l_border, r_border);
    auto distribution_y = getDistribution(d_border, u_border);
    long long M = 0;
    long long ind = 0;
    while (ind < N) {
        Point point = getRandomPoint(distribution_x, distribution_y, generator);
        if (isPointInCircle(point, r1, p1) && isPointInCircle(point, r2, p2) &&
isPointInCircle(point, r3, p3)) M++;
        ind++;
    }
    return S * M / N;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    Point p1 = {1.5, 2};
    Point p2 = {1, 1};
    Point p3 = {2, 1.5};
    double r1 = std::sqrt(5) / 2;
```

```
    double r2 = 1;
    double r3 = r1;
    double correct_S = 0.25 * M_PI + 1.25 * std::asin(0.8) - 1;
    std::vector<long long> vect_of_n;
    std::vector<double> calculated_squares;
    std::vector<double> diffs;
    std::random_device device;
    std::mt19937 generator(device());
    long long N = 100;
    while (N <= 100000) {
        double S = getSquare(p1, p2, p3, r1, r2, r3, generator, N);
        calculated_squares.push_back(S);
        vect_of_n.push_back(N);
        diffs.push_back(100.0 * std::abs((S - correct_S) / correct_S));
        N += 500;
    }
    print(vect_of_n);
    print(calculated_squares);
    print(diffs);
    return 0;
}
```

Данные, полученные с помощью данного кода (используются для построения графиков):

Количество генерируемых точек, входной параметр (n):

100, 600, 1100, 1600, 2100, 2600, 3100, 3600, 4100, 4600, 5100, 5600, 6100, 6600, 7100, 7600, 8100, 8600, 9100, 9600, 10100, 10600, 11100, 11600, 12100, 12600, 13100, 13600, 14100, 14600, 15100, 15600, 16100, 16600, 17100, 17600, 18100, 18600, 19100, 19600, 20100, 20600, 21100, 21600, 22100, 22600, 23100, 23600, 24100, 24600, 25100, 25600, 26100, 26600, 27100, 27600, 28100, 28600, 29100, 29600, 30100, 30600, 31100, 31600, 32100, 32600, 33100, 33600, 34100, 34600, 35100, 35600, 36100, 36600, 37100, 37600, 38100, 38600, 39100, 39600, 40100, 40600, 41100, 41600, 42100, 42600, 43100, 43600, 44100, 44600, 45100, 45600, 46100, 46600, 47100, 47600, 48100, 48600, 49100, 49600, 50100, 50600, 51100, 51600, 52100, 52600, 53100, 53600, 54100, 54600, 55100, 55600, 56100, 56600, 57100, 57600, 58100, 58600, 59100, 59600, 60100, 60600, 61100, 61600, 62100, 62600, 63100, 63600, 64100, 64600, 65100, 65600, 66100, 66600, 67100, 67600, 68100, 68600, 69100, 69600, 70100, 70600, 71100, 71600, 72100, 72600, 73100, 73600, 74100, 74600, 75100, 75600, 76100, 76600, 77100, 77600, 78100, 78600, 79100, 79600, 80100, 80600, 81100, 81600, 82100, 82600, 83100, 83600, 84100, 84600, 85100, 85600, 86100, 86600, 87100, 87600, 88100, 88600, 89100, 89600, 90100, 90600, 91100, 91600, 92100, 92600, 93100, 93600, 94100, 94600, 95100, 95600, 96100, 96600, 97100, 97600, 98100, 98600, 99100, 99600

Вычисленные площади (s):

1.55554, 0.923603, 0.839639, 1.08159, 0.888881, 0.88621, 0.972214, 0.947908, 0.943759, 0.919376, 0.960776, 0.965269, 0.961057, 0.877938, 0.95989, 0.986285, 0.973414, 0.978996, 0.900633, 0.999557, 0.912533, 0.980468, 0.926668, 0.968861, 0.902311, 0.942121, 0.940301, 0.94076, 0.913605, 0.957564, 0.95161, 0.923603, 0.963156, 0.914818, 0.981879, 0.910898, 0.972214, 0.962805, 0.942182, 0.983622, 0.984789, 0.931154, 0.923373, 0.924053, 0.936141, 0.935218, 0.902349, 0.928134, 0.983509, 0.931112, 0.917599, 0.987784, 0.942786, 0.963442, 0.952482, 0.958124, 0.960796, 0.915784, 0.921765, 0.963017, 0.931839, 0.932181, 0.944704, 0.959599,

0.92436, 0.917937, 0.95694, 0.928233, 0.952256, 0.953668, 0.935929, 0.944358,
0.940435, 0.976729, 0.966186, 0.957475, 0.924751, 0.949797, 0.970224, 0.945699,
0.951848, 0.962156, 0.928216, 0.945104, 0.96136, 0.952587, 0.932964, 0.948577,
0.924815, 0.932758, 0.957555, 0.942578, 0.95471, 0.943005, 0.947031, 0.949951,
0.969182, 0.942607, 0.962115, 0.958885, 0.969885, 0.958956, 0.937777, 0.926052,
0.928548, 0.938759, 0.959763, 0.944462, 0.944, 0.931794, 0.93816, 0.935668, 0.935994,
0.952975, 0.954847, 0.952466, 0.944101, 0.937207, 0.950499, 0.94432, 0.949405,
0.943657, 0.946436, 0.936703, 0.949513, 0.951092, 0.950335, 0.932622, 0.936874,
0.947081, 0.946228, 0.964803, 0.950593, 0.954404, 0.946423, 0.967036, 0.940235,
0.93239, 0.948858, 0.961458, 0.955987, 0.936961, 0.937755, 0.937724, 0.947942,
0.939003, 0.951333, 0.948569, 0.953452, 0.961788, 0.932341, 0.956396, 0.951134,
0.940356, 0.958721, 0.948284, 0.953043, 0.949578, 0.95771, 0.937038, 0.943448,
0.941817, 0.942484, 0.938258, 0.950543, 0.95138, 0.935829, 0.960933, 0.97152, 0.949,
0.954849, 0.944387, 0.938677, 0.94875, 0.946653, 0.935922, 0.934031, 0.947305,
0.944935, 0.958542, 0.928081, 0.948284, 0.931554, 0.941222, 0.96018, 0.945231,
0.942974, 0.928485, 0.953617, 0.935113, 0.925801, 0.947705, 0.946517, 0.946852,
0.941475, 0.940437, 0.950014, 0.933858, 0.956419, 0.946444

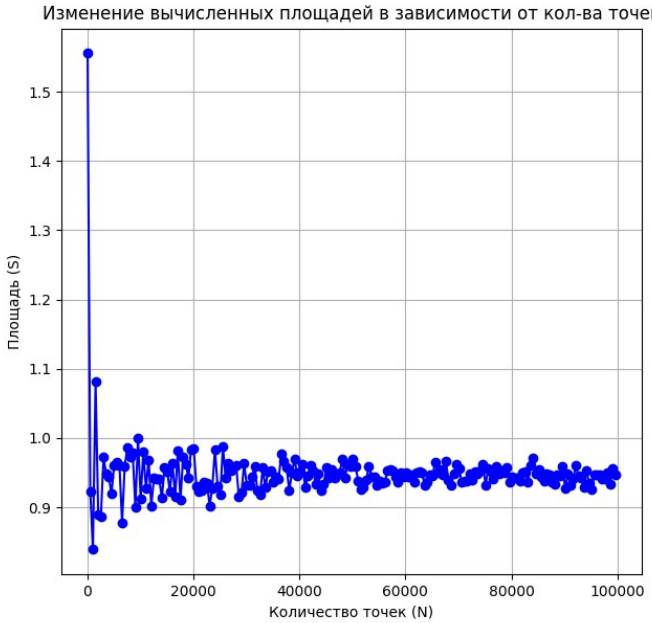
Отклонения вычисленных от точных значений площадей в % (diff):

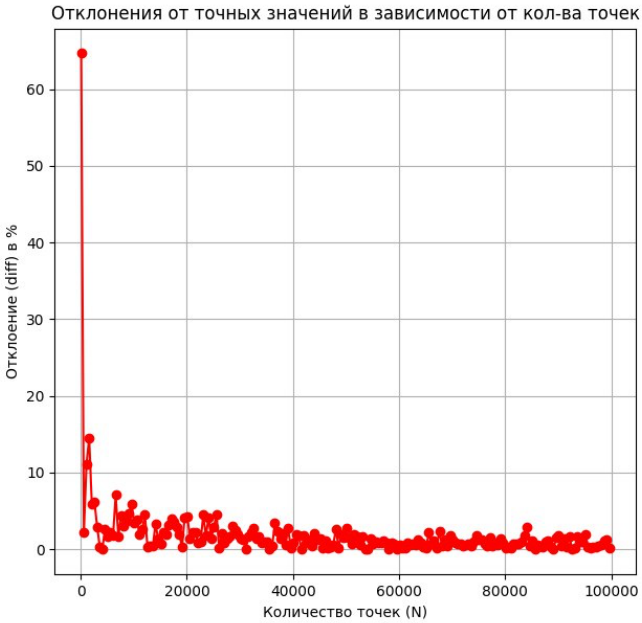
64.6917, 2.21428, 11.1039, 14.5122, 5.89044, 6.17322, 2.93234, 0.359027,
0.0803185, 2.66181, 1.72137, 2.1971, 1.75114, 7.04898, 1.62756, 4.42215, 3.05941,
3.65047, 4.6462, 5.82731, 3.38628, 3.80629, 1.88972, 2.5774, 4.46858, 0.253666,
0.44636, 0.397829, 3.27281, 1.3813, 0.750988, 2.21428, 1.97334, 3.14439, 3.95564,
3.55942, 2.93234, 1.93622, 0.24725, 4.14021, 4.2638, 1.41481, 2.23867, 2.16663,
0.886874, 0.984559, 4.46453, 1.73451, 4.12823, 1.41928, 2.84992, 4.58086, 0.183241,
2.00362, 0.843303, 1.44056, 1.72352, 3.04206, 2.40883, 1.95865, 1.34226, 1.30606,
0.0197803, 1.59682, 2.13412, 2.81419, 1.31527, 1.72413, 0.819355, 0.968886, 0.909299,
0.0168497, 0.432212, 3.41044, 2.29421, 1.37192, 2.09271, 0.559025, 2.72173, 0.12509,
0.776147, 1.86752, 1.72592, 0.0621066, 1.78321, 0.854358, 1.22317, 0.429852, 2.08591,
1.24496, 1.38036, 0.205295, 1.07911, 0.160053, 0.266147, 0.575271, 2.61134, 0.20223,
1.86318, 1.52117, 2.68579, 1.52871, 0.713605, 1.95496, 1.69073, 0.609633, 1.61418,
0.0058453, 0.0547954, 1.34709, 0.673099, 0.936884, 0.902399, 0.895512, 1.09361,
0.841522, 0.0440215, 0.773932, 0.633339, 0.0209248, 0.51745, 0.0910899, 0.203196,
0.827369, 0.528923, 0.696106, 0.61595, 1.25941, 0.809199, 0.271391, 0.181148, 2.14779,
0.64322, 1.04679, 0.20179, 2.38417, 0.453398, 1.28399, 0.459576, 1.79357, 1.21435,
0.800055, 0.715899, 0.719175, 0.362596, 0.583811, 0.721613, 0.428954, 0.945922,
1.8285, 1.28912, 1.25764, 0.700557, 0.440513, 1.50383, 0.39882, 0.902683, 0.535819,
1.39681, 0.79185, 0.113228, 0.285897, 0.215287, 0.662728, 0.637985, 0.726642,
0.919886, 1.73802, 2.8589, 0.474613, 1.09383, 0.0137422, 0.618292, 0.448171, 0.226077,
0.910002, 1.11018, 0.29513, 0.0442225, 1.48485, 1.74018, 0.398791, 1.37252, 0.348914,
1.65825, 0.0755738, 0.163374, 1.69742, 0.963385, 0.995632, 1.98157, 0.337493,
0.211751, 0.247144, 0.322065, 0.431949, 0.581994, 1.12859, 1.26007, 0.204008

Графики:

1) График, который отображает, как меняется S в зависимости от N

2) График, который отображает, как меняется diff в зависимости от N




Код для отображения графиков:

```
import matplotlib.pyplot as plt

# Данные
n_values = [100, 600, 1100, 1600, 2100, 2600, 3100, 3600, 4100, 4600, 5100, 5600,
6100, 6600, 7100,
            7600, 8100, 8600, 9100, 9600, 10100, 10600, 11100, 11600, 12100, 12600,
13100,
            13600, 14100, 14600, 15100, 15600, 16100, 16600, 17100, 17600, 18100,
18600,
            19100, 19600, 20100, 20600, 21100, 21600, 22100, 22600, 23100, 23600,
24100,
            24600, 25100, 25600, 26100, 26600, 27100, 27600, 28100, 28600, 29100,
29600,
            30100, 30600, 31100, 31600, 32100, 32600, 33100, 33600, 34100, 34600,
35100,
            35600, 36100, 36600, 37100, 37600, 38100, 38600, 39100, 39600, 40100,
40600,
            41100, 41600, 42100, 42600, 43100, 43600, 44100, 44600, 45100, 45600,
46100,
            46600, 47100, 47600, 48100, 48600, 49100, 49600, 50100, 50600, 51100,
51600,
            52100, 52600, 53100, 53600, 54100, 54600, 55100, 55600, 56100, 56600,
57100,
            57600, 58100, 58600, 59100, 59600, 60100, 60600, 61100, 61600, 62100,
62600,
            63100, 63600, 64100, 64600, 65100, 65600, 66100, 66600, 67100, 67600,
68100,
            68600, 69100, 69600, 70100, 70600, 71100, 71600, 72100, 72600, 73100,
73600,
            74100, 74600, 75100, 75600, 76100, 76600, 77100, 77600, 78100, 78600,
79100,
            79600, 80100, 80600, 81100, 81600, 82100, 82600, 83100, 83600, 84100,
84600,
```

```
            85100, 85600, 86100, 86600, 87100, 87600, 88100, 88600, 89100, 89600,
90100,
            90600, 91100, 91600, 92100, 92600, 93100, 93600, 94100, 94600, 95100,
95600,
            96100, 96600, 97100, 97600, 98100, 98600, 99100, 99600]

s_values = [1.55554, 0.923603, 0.839639, 1.08159, 0.888881, 0.88621, 0.972214,
0.947908,
            0.943759, 0.919376, 0.960776, 0.965269, 0.961057, 0.877938, 0.95989,
0.986285,
            0.973414, 0.978996, 0.900633, 0.999557, 0.912533, 0.980468, 0.926668,
0.968861,
            0.902311, 0.942121, 0.940301, 0.94076, 0.913605, 0.957564, 0.95161,
0.923603,
            0.963156, 0.914818, 0.981879, 0.910898, 0.972214, 0.962805, 0.942182,
0.983622,
            0.984789, 0.931154, 0.923373, 0.924053, 0.936141, 0.935218, 0.902349,
0.928134,
            0.983509, 0.931112, 0.917599, 0.987784, 0.942786, 0.963442, 0.952482,
0.958124,
            0.960796, 0.915784, 0.921765, 0.963017, 0.931839, 0.932181, 0.944704,
0.959599,
            0.92436, 0.917937, 0.95694, 0.928233, 0.952256, 0.953668, 0.935929,
0.944358,
            0.940435, 0.976729, 0.966186, 0.957475, 0.924751, 0.949797, 0.970224,
0.945699,
            0.951848, 0.962156, 0.928216, 0.945104, 0.96136, 0.952587, 0.932964,
0.948577,
            0.924815, 0.932758, 0.957555, 0.942578, 0.95471, 0.943005, 0.947031,
0.949951,
            0.969182, 0.942607, 0.962115, 0.958885, 0.969885, 0.958956, 0.937777,
0.926052,
            0.928548, 0.938759, 0.959763, 0.944462, 0.944, 0.931794, 0.93816,
0.935668,
            0.935994, 0.952975, 0.954847, 0.952466, 0.944101, 0.937207, 0.950499,
0.94432,
            0.949405, 0.943657, 0.946436, 0.936703, 0.949513, 0.951092, 0.950335,
0.932622,
            0.936874, 0.947081, 0.946228, 0.964803, 0.950593, 0.954404, 0.946423,
0.967036,
            0.940235, 0.93239, 0.948858, 0.961458, 0.955987, 0.936961, 0.937755,
0.937724,
            0.947942, 0.939003, 0.951333, 0.948569, 0.953452, 0.961788, 0.932341,
0.956396,
            0.951134, 0.940356, 0.958721, 0.948284, 0.953043, 0.949578, 0.95771,
0.937038,
            0.943448, 0.941817, 0.942484, 0.938258, 0.950543, 0.95138, 0.935829,
0.960933,
            0.97152, 0.949, 0.954849, 0.944387, 0.938677, 0.94875, 0.946653, 0.935922,
            0.934031, 0.947305, 0.944935, 0.958542, 0.928081, 0.948284, 0.931554,
0.941222,
            0.96018, 0.945231, 0.942974, 0.928485, 0.953617, 0.935113, 0.925801,
0.947705,
```

```
           0.946517, 0.946852, 0.941475, 0.940437, 0.950014, 0.933858, 0.956419,
0.946444]

diff_values = [64.6917, 2.21428, 11.1039, 14.5122, 5.89044, 6.17322, 2.93234,
0.359027,
               0.0803185, 2.66181, 1.72137, 2.1971, 1.75114, 7.04898, 1.62756,
4.42215,
               3.05941, 3.65047, 4.6462, 5.82731, 3.38628, 3.80629, 1.88972, 2.5774,
               4.46858, 0.253666, 0.44636, 0.397829, 3.27281, 1.3813, 0.750988,
2.21428,
               1.97334, 3.14439, 3.95564, 3.55942, 2.93234, 1.93622, 0.24725, 4.14021,
               4.2638, 1.41481, 2.23867, 2.16663, 0.886874, 0.984559, 4.46453,
1.73451,
               4.12823, 1.41928, 2.84992, 4.58086, 0.183241, 2.00362, 0.843303,
1.44056,
               1.72352, 3.04206, 2.40883, 1.95865, 1.34226, 1.30606, 0.0197803,
1.59682,
               2.13412, 2.81419, 1.31527, 1.72413, 0.819355, 0.968886, 0.909299,
0.0168497,
               0.432212, 3.41044, 2.29421, 1.37192, 2.09271, 0.559025, 2.72173,
0.12509,
               0.776147, 1.86752, 1.72592, 0.0621066, 1.78321, 0.854358, 1.22317,
0.429852,
               2.08591, 1.24496, 1.38036, 0.205295, 1.07911, 0.160053, 0.266147,
0.575271,
               2.61134, 0.20223, 1.86318, 1.52117, 2.68579, 1.52871, 0.713605,
1.95496,
               1.69073, 0.609633, 1.61418, 0.0058453, 0.0547954, 1.34709, 0.673099,
               0.936884, 0.902399, 0.895512, 1.09361, 0.841522, 0.0440215, 0.773932,
               0.633339, 0.0209248, 0.51745, 0.0910899, 0.203196, 0.827369, 0.528923,
               0.696106, 0.61595, 1.25941, 0.809199, 0.271391, 0.181148, 2.14779,
0.64322,
               1.04679, 0.20179, 2.38417, 0.453398, 1.28399, 0.459576, 1.79357,
1.21435,
               0.800055, 0.715899, 0.719175, 0.362596, 0.583811, 0.721613, 0.428954,
               0.945922, 1.8285, 1.28912, 1.25764, 0.700557, 0.440513, 1.50383,
0.39882,
               0.902683, 0.535819, 1.39681, 0.79185, 0.113228, 0.285897, 0.215287,
               0.662728, 0.637985, 0.726642, 0.919886, 1.73802, 2.8589, 0.474613,
1.09383,
               0.0137422, 0.618292, 0.448171, 0.226077, 0.910002, 1.11018, 0.29513,
               0.0442225, 1.48485, 1.74018, 0.398791, 1.37252, 0.348914, 1.65825,
               0.0755738, 0.163374, 1.69742, 0.963385, 0.995632, 1.98157, 0.337493,
               0.211751, 0.247144, 0.322065, 0.431949, 0.581994, 1.12859, 1.26007,
0.204008]

# Создание графика для S в зависимости от N
plt.figure(figsize=(12, 6))
# График S
plt.subplot(1, 2, 1)
plt.plot(n_values, s_values, marker='o', linestyle='-', color='b')
plt.title('Изменение вычисленных площадей в зависимости от кол-ва точек')
plt.xlabel('Количество точек (N)')
```

```
plt.ylabel('Площадь (S)')
plt.grid()
# График diff в зависимости от N
plt.subplot(1, 2, 2)
plt.plot(n_values, diff_values, marker='o', linestyle='-', color='r')
plt.title('Отклонения от точных значений в зависимости от кол-ва точек')
plt.xlabel('Количество точек (N)')
plt.ylabel('Отклоение (diff) в %')
plt.grid()
# Отображение графиков
plt.tight_layout()
plt.show()
```

Содержательные выводы:

Графики показывают, что с увеличением числа случайных точек, используемых для вычисления, точность оценки площади фигуры растет. Это происходит потому, что при большем количестве точек они более равномерно распределяются по области. Таким образом, когда больше точек попадает внутрь кругов, вероятность точного вычисления площади увеличивается. Также увеличение числа точек помогает точнее определить долю точек, находящихся внутри фигуры, что улучшает соотношение между количеством точек внутри и общим количеством.