A1. Задача трех кругов

Работу выполнил Девятов Денис БПИ-238

Пункт 1.

ID 293075871 – посылка

https://github.com/BelyLandy/set_3

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

// Проверяет, находится ли точка внутри окружности.
bool isPointInCircle(long double pointX, long double pointY, long double circleCenterX, long double circleCenterY, long double radius) {
    return pow(pointX - circleCenterX, 2) + pow(pointY - circleCenterY, 2) <= pow(radius, 2);
}

// Проверяет, находится ли точка внутри пересечения трех окружностей.
bool isPointInIntersection(long double pointX, long double pointY,
                           long double circle1X, long double circle1Y, long double circle1Radius,
                           long double circle2X, long double circle2Y, long double circle2Radius,
                           long double circle3X, long double circle3Y, long double circle3Radius) {
    return isPointInCircle(pointX, pointY, circle1X, circle1Y, circle1Radius) &&
           isPointInCircle(pointX, pointY, circle2X, circle2Y, circle2Radius) &&
           isPointInCircle(pointX, pointY, circle3X, circle3Y, circle3Radius);
}

int main() {
    // Ввод данных окружностей.
    long double circle1X, circle1Y, circle1Radius;
```

```
    long double circle2X, circle2Y, circle2Radius;
    long double circle3X, circle3Y, circle3Radius;
    std::cin >> circle1X >> circle1Y >> circle1Radius >> circle2X >> circle2Y >> circle2Radius >> circle3X >> circle3Y >> circle3Radius;

    // Определяем границы области для генерации случайных точек.
    long double maxXBoundary = std::max({circle1X + circle1Radius, circle2X + circle2Radius, circle3X + circle3Radius});
    long double minXBoundary = std::min({circle1X - circle1Radius, circle2X - circle2Radius, circle3X - circle3Radius});
    long double maxYBoundary = std::max({circle1Y + circle1Radius, circle2Y + circle2Radius, circle3Y + circle3Radius});
    long double minYBoundary = std::min({circle1Y - circle1Radius, circle2Y - circle2Radius, circle3Y - circle3Radius});

    // Настраиваем генератор случайных чисел.
    std::random_device randomDevice;
    std::mt19937 randomGenerator(randomDevice());
    std::uniform_real_distribution<long double> randomX(minXBoundary, maxXBoundary);
    std::uniform_real_distribution<long double> randomY(minYBoundary, maxYBoundary);

    // Параметры для Монте-Карло.
    size_t pointsInsideIntersection = 0;
    const size_t totalRandomPoints = 5000000;

    // Основной цикл метода Монте-Карло.
    for (size_t i = 0; i < totalRandomPoints; ++i) {
        long double randomPointX = randomX(randomGenerator);
        long double randomPointY = randomY(randomGenerator);

        if (isPointInIntersection(randomPointX, randomPointY,
                                  circle1X, circle1Y, circle1Radius,
                                  circle2X, circle2Y, circle2Radius,
```

```
                    circle3X, circle3Y, circle3Radius)) {
            ++pointsInsideIntersection;
        }
    }

    // Вычисляем площадь пересечения.
    long double intersectionArea = (pointsInsideIntersection * (maxXBoundary - minXBoundary) * (maxYBoundary - minYBoundary)) / totalRandomPoints;
    std::cout << intersectionArea;

    return 0;
}
```

Пункт 2.

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

// Проверяет, находится ли точка внутри окружности.
bool isPointInCircle(long double pointX, long double pointY, long double circleCenterX, long double circleCenterY, long double radius) {
    return pow(pointX - circleCenterX, 2) + pow(pointY - circleCenterY, 2) <= pow(radius, 2);
}

// Проверяет, находится ли точка внутри пересечения трех окружностей.
bool isPointInIntersection(long double pointX, long double pointY,
                           long double circle1X, long double circle1Y, long double circle1Radius,
                           long double circle2X, long double circle2Y, long double circle2Radius,
                           long double circle3X, long double circle3Y, long double circle3Radius) {
    return isPointInCircle(pointX, pointY, circle1X, circle1Y, circle1Radius) &&
           isPointInCircle(pointX, pointY, circle2X, circle2Y, circle2Radius) &&
```

```
        isPointInCircle(pointX, pointY, circle3X, circle3Y, circle3Radius);
}

long double intersection_area(int n, long double scale = 1) {
  // Ввод данных окружностей.
  long double circle1X = 1, circle1Y = 1, circle1Radius = 1;
  long double circle2X = 1.5, circle2Y = 2, circle2Radius = sqrt(5) / 2;
  long double circle3X = 2, circle3Y = 1.5, circle3Radius = sqrt(5) / 2;

  // Определяем границы области для генерации случайных точек.
  long double maxXBoundary = std::max({circle1X + circle1Radius, circle2X + circle2Radius,
circle3X + circle3Radius});
  long double minXBoundary = std::min({circle1X - circle1Radius, circle2X - circle2Radius,
circle3X - circle3Radius});
  long double maxYBoundary = std::max({circle1Y + circle1Radius, circle2Y + circle2Radius,
circle3Y + circle3Radius});
  long double minYBoundary = std::min({circle1Y - circle1Radius, circle2Y - circle2Radius,
circle3Y - circle3Radius});

  // Настраиваем генератор случайных чисел.
  std::random_device randomDevice;
  std::mt19937 randomGenerator(randomDevice());
  std::uniform_real_distribution<long double> randomX(minXBoundary, maxXBoundary);
  std::uniform_real_distribution<long double> randomY(minYBoundary, maxYBoundary);

  size_t pointsInsideIntersection = 0;
  for(size_t iteration = 0; iteration < n; ++iteration){
    long double randomPointX = randomX(randomGenerator);
    long double randomPointY = randomY(randomGenerator);

    if (isPointInIntersection(randomPointX, randomPointY,
                              circle1X, circle1Y, circle1Radius,
                              circle2X, circle2Y, circle2Radius,
```

```
                        circle3X, circle3Y, circle3Radius)) {
      ++pointsInsideIntersection;
    }
  }

  return (pointsInsideIntersection * (maxXBoundary - minXBoundary) * (maxYBoundary - minYBoundary)) / n * scale * scale;
}

int main() {
  long double right_area = M_PI / 4 + 5 * asin(0.8) / 4 - 1;

  std::cout << "Num, Area, Area_dff\n";

  for (size_t num = 100; num < 100000; num += 500) {
    long double area = intersection_area(num);
    std::cout << num << ", " << area << ", " << std::abs(area - right_area) << "\n";
  }

  std::cout << "S, Area, Area_dff\n";

  for (long double s = 1; s < 10; s += 0.1) {
    long double area = intersection_area(1000000, s);
    std::cout << s << ", " << area << ", " << std::abs(area - right_area) << "\n";
  }

  return 0;
}
```

Графики, которые отображают, как меняется приближенное значение площади в зависимости от указанных параметров алгоритма.

График зависимости площади от числа сгенерированных точек.

С растущим числом точек точность предсказанной площади повышается.

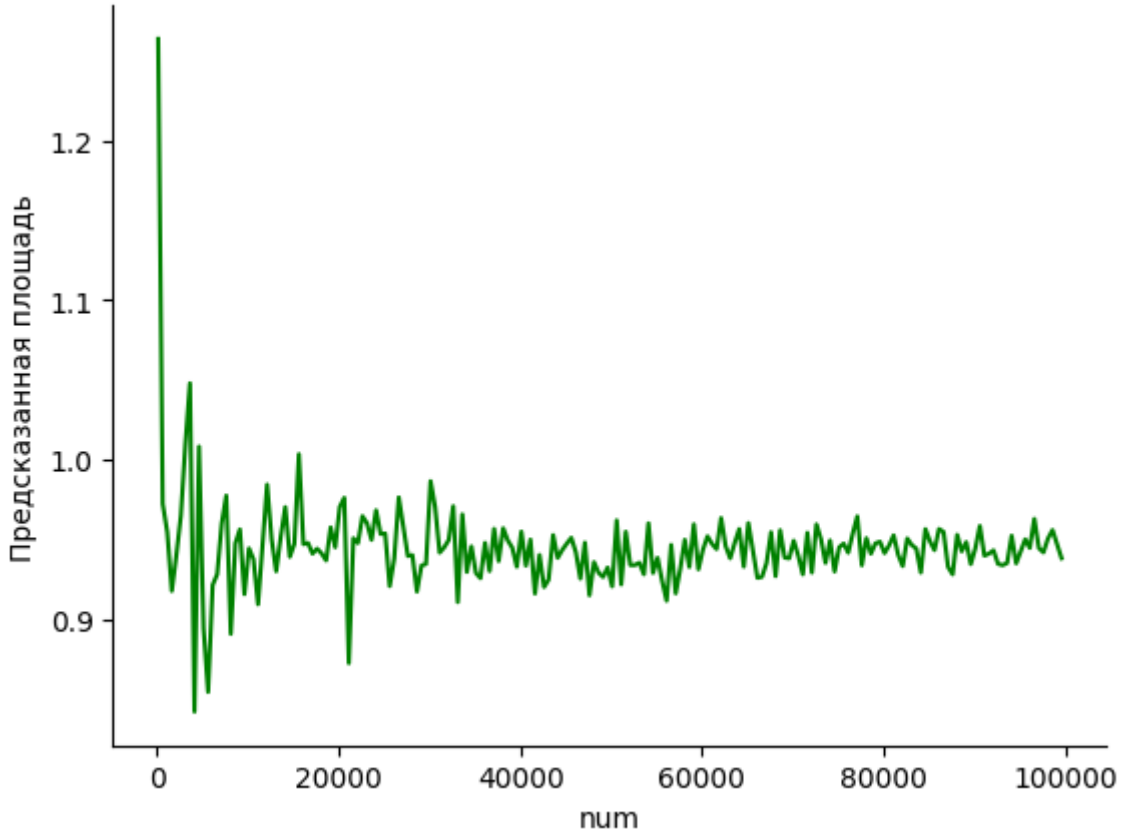
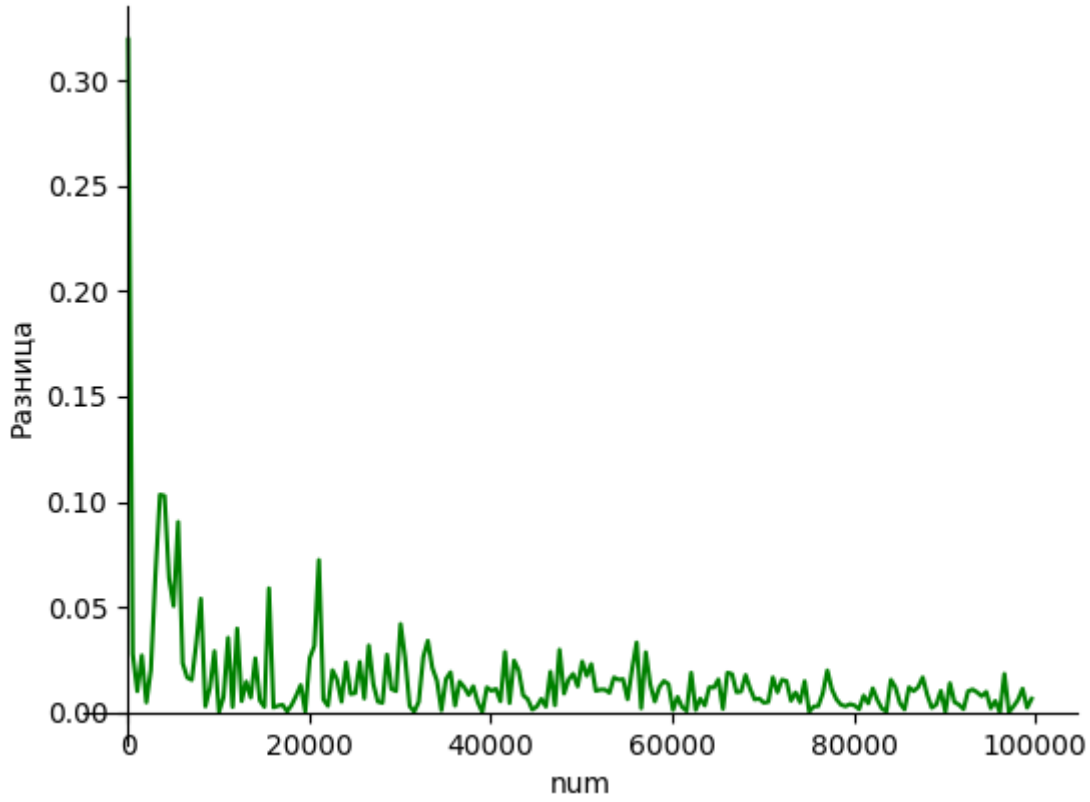


График разницы между предсказанным и истинным значением.

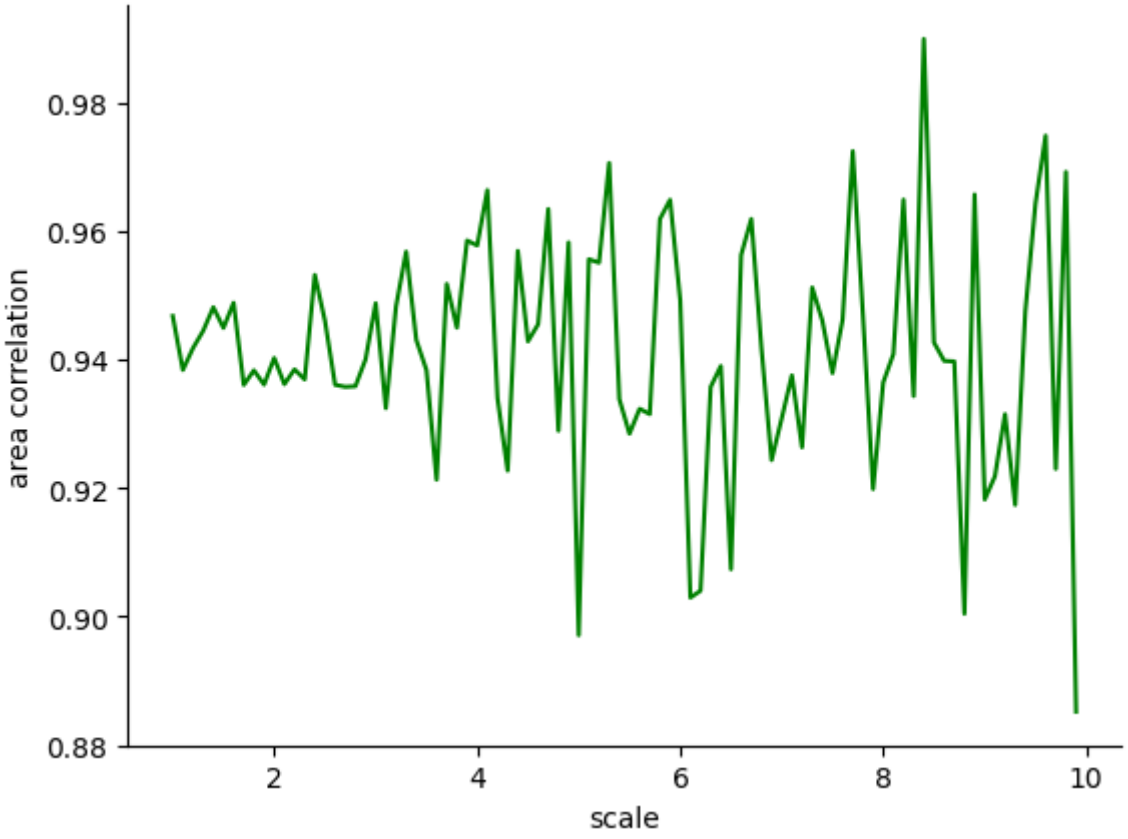
Здесь мы видим, что с растущим числом точек точность повышается => разница уменьшается.



Графики, которые отображают, как меняется величина относительного отклонения приближенного значения площади от ее точной оценки в зависимости от указанных параметров алгоритма.

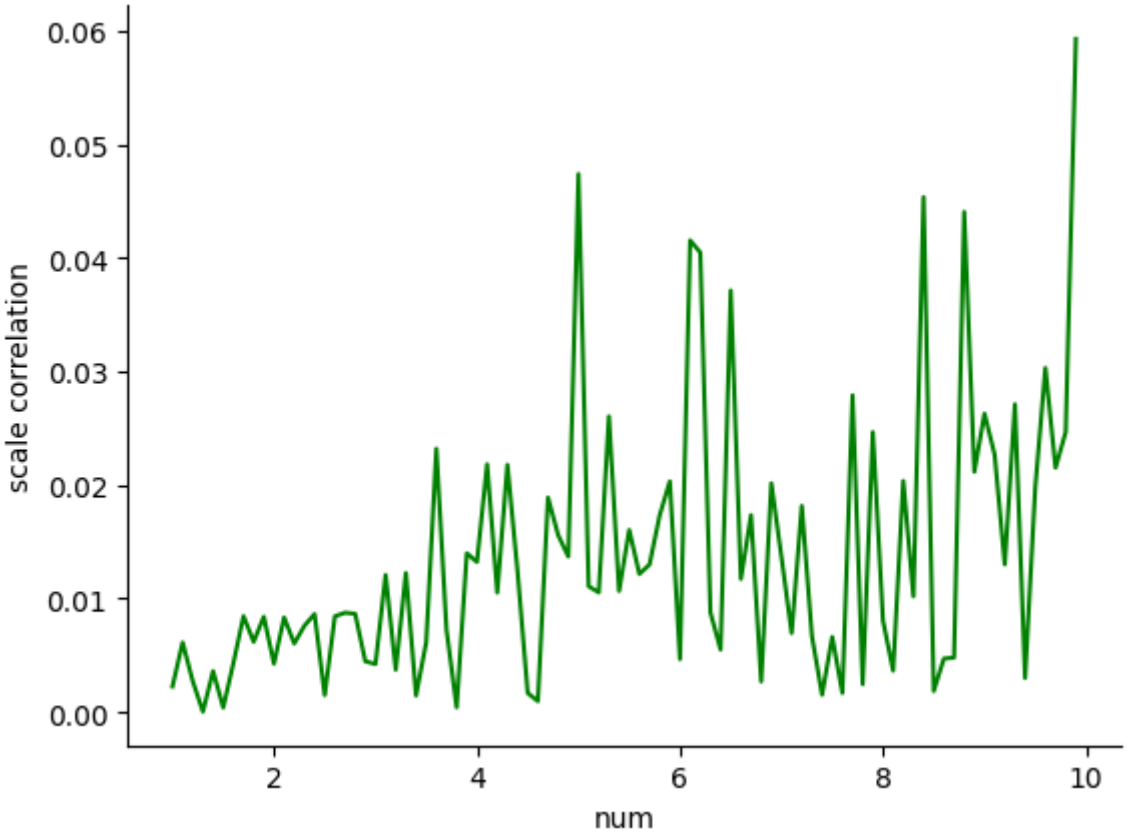
График зависимости площади от масштаба.

По этому графику мы видим, что при увеличении scale падает вероятность нахождения точки в пересечении, и в случае одинакового кол-ва точек точность площ. ухудшается.



Разница между предсказанным и прав. значениями

С уменьшением точности измерений разница с истинным значением значительно возрастает.



Таким образом, проведя анализ результатов, полученных из наших данных, для улучшения точности алгоритмов вычисления площади методом Монте-Карло необходимо увеличить количество генерируемых точек и выбрать минимально возможный масштаб.