Реализация алгоритма Монте-Карло

ссылка на репозиторий:
https://github.com/AnnaAstashkina/A1_set3

id посылки: 293011165

код:

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

double findS(double x1, double y1, double r1, double x2, double y2,
double r2, double x3, double y3, double r3) {
  double leftBorder = std::max(x1 - r1, std::max(x2 - r2, x3 - r3));
  double rightBorder = std::min(x1 + r1, std::min(x2 + r2, x3 + r3));
  double upBorder = std::min(y1 + r1, std::min(y2 + r2, y3 + r3));
  double downBorder = std::max(y1 - r1, std::max(y2 - r2, y3 - r3));
  double srec = (upBorder - downBorder) * (rightBorder - leftBorder);
  std::random_device rand_dev;
  std::mt19937 generator(rand_dev());
  std::uniform_real_distribution<> distrx(leftBorder, rightBorder);
  std::uniform_real_distribution<> distry(downBorder, upBorder);
  int m = 0;
  long long n = 15000000;
  for (int i = 0; i < n; ++i) {
    double x = distrx(generator);
    double y = distry(generator);
    if ((x - x1) * (x - x1) + (y - y1) * (y - y1) <= r1 * r1 && (x -
x2) * (x - x2) + (y - y2) * (y - y2) <= r2 * r2 &&
        (x - x3) * (x - x3) + (y - y3) * (y - y3) <= r3 * r3) {
      m++;
    }
  }
  return srec * m / n;
}
```

```cpp
int main() {\
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);
  double x1 = 0;
  double y1 = 0;
  double r1 = 0;
  double x2 = 0;
  double y2 = 0;
  double r2 = 0;
  double x3 = 0;
  double y3 = 0;
  double r3 = 0;
  std::cin >> x1 >> y1 >> r1 >> x2 >> y2 >> r2 >> x3 >> y3 >> r3;
  double s = findS(x1, y1, r1, x2, y2, r2, x3, y3, r3);
  std::cout << s << std::endl;
  return 0;
}
```

Графики и анализ

График изменения вычисленной площади в зависимости от количества генерируемых точек n

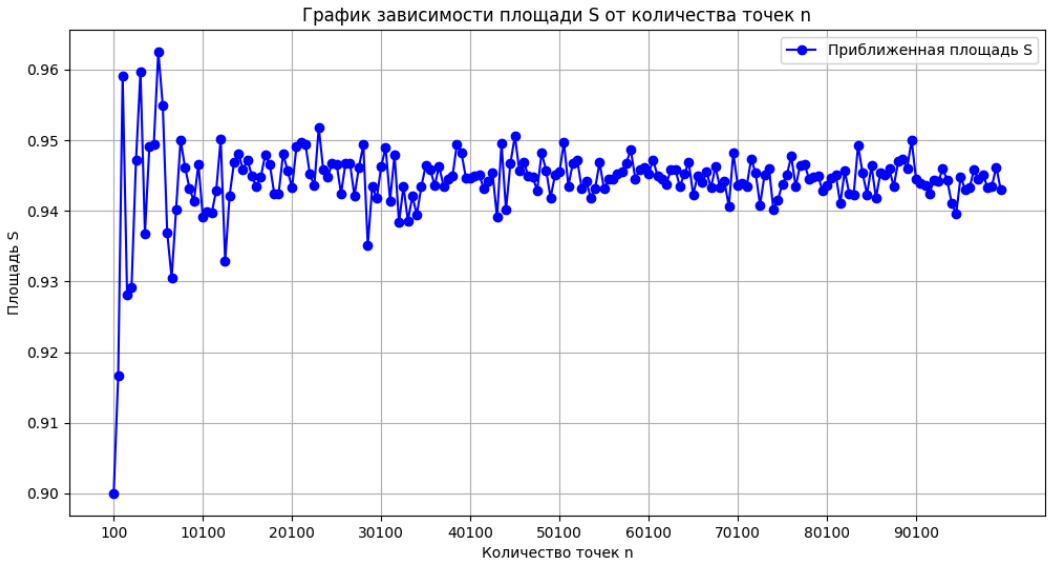

График отклонения (в процентах) от точного значения в зависимости от количества генерируемых точек n

Вывод про количество точек n

Из графиков видно, что чем больше генерируется случайных точек тем более точно производится вычисление площади фигуры. Это можно объяснить тем, что чем больше мы генерируем точек, тем большую площадь рассматриваемого прямоугольника мы покрываем, а площадь области вычисляется, по сути, как доля площади прямоугольника, причем долю мы вычисляем как раз отношением попавших в область точек к общему количеству => при большем количестве точек покрывается большая доля прямоугольника => вычисление доли занимаемой искомой областью становится более точным => площадь вычисляется более точно.

График изменения вычисленной площади в зависимости от k, где k - это то, на сколько рассматриваемый прямоугольник больше минимального (самого точного)

Т.е. если k=0.5, a1 - это самая точная правая граница, a2 - самая неточная (в адекватных пределах) правая граница (самая правая точка кругов), то a=a1+(a2-a1)*k, т.е. a - рассматриваемая граница будет по центру между a1 и a2.

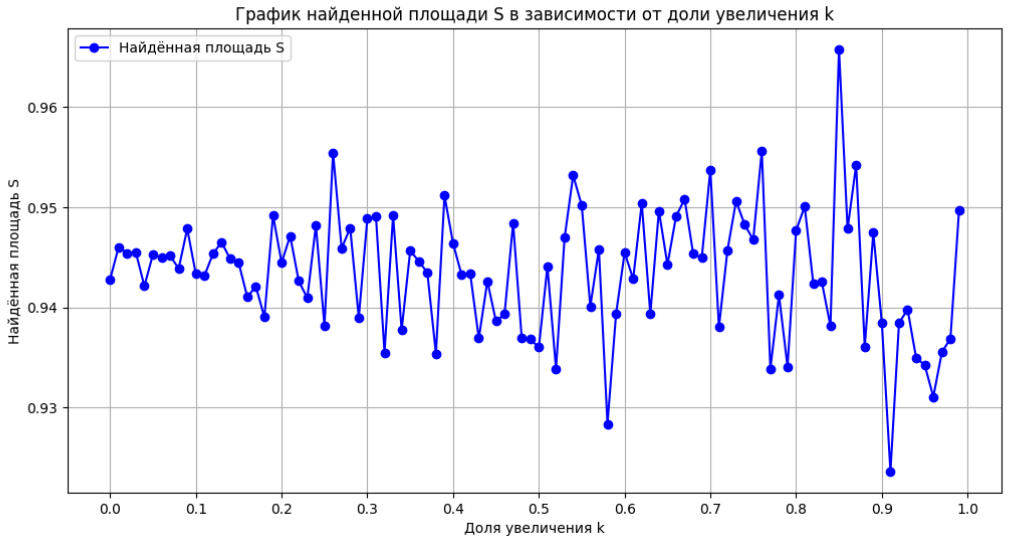

График отклонения (в процентах) от точного значения в зависимости от k, где k - это то, на сколько рассматриваемый прямоугольник больше минимального (самого точного)

Вывод про k

Заметим, что в графиках развитие идет от самого точного прямоугольника до самого неточного!
Можно сделать вывод, что при одинаковом количестве генерируемых точек чем точнее задана область, тем точнее вычисляется площадь области. Это можно объяснить аналогично зависимости от количества генерируемых точек. Если

рассматриваемый прямоугольник имеет меньшую площадь, то фиксированное количество точек покроет его сильнее чем прямоугольник с большей площадью => и вычислении площади искомой области будет более точным.

Код для получения результатов для построения графиков:

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

double findS(double x1, double y1, double r1, double x2, double y2,
double r2, double x3, double y3, double r3,
             long long n, double k) {
  double leftBorderVnesh = std::min(x1 - r1, std::min(x2 - r2, x3 -
r3));
  double rightBorderVnesh = std::max(x1 + r1, std::max(x2 + r2, x3 +
r3));
  double upBorderVnesh = std::max(y1 + r1, std::max(y2 + r2, y3 + r3));
  double downBorderVnesh = std::min(y1 - r1, std::min(y2 - r2, y3 -
r3));
  double leftBorderVnutr = std::max(x1 - r1, std::max(x2 - r2, x3 -
r3));
  double rightBorderVnutr = std::min(x1 + r1, std::min(x2 + r2, x3 +
r3));
  double upBorderVnutr = std::min(y1 + r1, std::min(y2 + r2, y3 + r3));
  double downBorderVnutr = std::max(y1 - r1, std::max(y2 - r2, y3 -
r3));
  double leftBorder = leftBorderVnutr - (leftBorderVnutr -
leftBorderVnesh) * k;
  double rightBorder = rightBorderVnutr + (rightBorderVnesh -
rightBorderVnutr) * k;
  double upBorder = upBorderVnutr + (upBorderVnesh - upBorderVnutr) *
k;
  double downBorder = downBorderVnutr - (downBorderVnutr -
downBorderVnesh) * k;
  double srec = (upBorder - downBorder) * (rightBorder - leftBorder);
```

```
  std::random_device rand_dev;
  std::mt19937 generator(rand_dev());
  std::uniform_real_distribution<> distrx(leftBorder, rightBorder);
  std::uniform_real_distribution<> distry(downBorder, upBorder);

  int m = 0;
  for (long long i = 0; i < n; ++i) {
    double x = distrx(generator);
    double y = distry(generator);
    if ((x - x1) * (x - x1) + (y - y1) * (y - y1) <= r1 * r1 && (x -
x2) * (x - x2) + (y - y2) * (y - y2) <= r2 * r2 &&
        (x - x3) * (x - x3) + (y - y3) * (y - y3) <= r3 * r3) {
      m++;
    }
  }
  return srec * m / n;
}

template <typename T>
void printVector(const std::vector<T>& vec) {
  for (int i = 0; i < vec.size(); ++i) {
    std::cout << vec[i] << " ";
  }
  std::cout << std::endl<<std::endl;
}

int main() {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);

  double x1 = 1;
  double y1 = 1;
  double r1 = 1;
  double x2 = 1.5;
  double y2 = 2;
  double r2 = std::sqrt(5) / 2;
  double x3 = 2;
  double y3 = 1.5;
  double r3 = std::sqrt(5) / 2;

  double exact = 0.25 * M_PI + 1.25 * std::asin(0.8) - 1;

  std::vector<long long> ns;
```

```
  std::vector<double> results1;
  std::vector<double> deviations1;
  std::vector<double> ks;
  std::vector<double> results2;
  std::vector<double> deviations2;

  for (long long n = 100; n <= 100000; n += 500) {
    ns.push_back(n);
    double estimatedArea = findS(x1, y1, r1, x2, y2, r2, x3, y3, r3, n,
0);
    results1.push_back(estimatedArea);
    double deviation = std::abs((estimatedArea - exact) / exact) *
100.0;  // В процентах
    deviations1.push_back(deviation);
   }
  for (double k = 0; k <= 1; k += 0.01) {
    ks.push_back(k);
    double estimatedArea = findS(x1, y1, r1, x2, y2, r2, x3, y3, r3,
100000, k);
    results2.push_back(estimatedArea);
    double deviation = std::abs((estimatedArea - exact) / exact) *
100.0;
    deviations2.push_back(deviation);
   }
  printVector(ns);
  printVector(results1);
  printVector(deviations1);
  printVector(ks);
  printVector(results2);
  printVector(deviations2);
  return 0;
}
```

Результаты, на основе которых строились графики

k: 0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2, 0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3, 0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4, 0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7, 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79,

0.8, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99

S_k: 0.942813, 0.946034, 0.945422, 0.945511, 0.942136, 0.945295, 0.944942, 0.945174, 0.943876, 0.947853, 0.943359, 0.943166, 0.945347, 0.946526, 0.944844, 0.944442, 0.941058, 0.942097, 0.939022, 0.949212, 0.944469, 0.947142, 0.942635, 0.940944, 0.948193, 0.938172, 0.955445, 0.945902, 0.947938, 0.938982, 0.948864, 0.949123, 0.935488, 0.94918, 0.937743, 0.945664, 0.944626, 0.943476, 0.935399, 0.951177, 0.946385, 0.943238, 0.943369, 0.936916, 0.942624, 0.938621, 0.939333, 0.948414, 0.936969, 0.936824, 0.936018, 0.944088, 0.93389, 0.946964, 0.953178, 0.950186, 0.940101, 0.945812, 0.928391, 0.939324, 0.945483, 0.942845, 0.950426, 0.93936, 0.949649, 0.944272, 0.94908, 0.950819, 0.945353, 0.94495, 0.953737, 0.938028, 0.945738, 0.950614, 0.948277, 0.946757, 0.955641, 0.933871, 0.941236, 0.934091, 0.947695, 0.950073, 0.9424, 0.942535, 0.938148, 0.965743, 0.947846, 0.954228, 0.936071, 0.947446, 0.93843, 0.923629, 0.938434, 0.939815, 0.934953, 0.934262, 0.931087, 0.935531, 0.936913, 0.949704

S_k_pr: 0.180482, 0.160628, 0.0957654, 0.105264, 0.252136, 0.0823406, 0.0450019, 0.0695522, 0.067882, 0.353174, 0.122612, 0.143018, 0.0878924, 0.212727, 0.034593, 0.0079473, 0.366212, 0.256183, 0.581821, 0.497047, 0.00505038, 0.277898, 0.199263, 0.378343, 0.389227, 0.671741, 1.15695, 0.146649, 0.36222, 0.586054, 0.460212, 0.487686, 0.955949, 0.493693, 0.717256, 0.121459, 0.0114931, 0.110191, 0.965406, 0.705081, 0.197784, 0.135413, 0.121593, 0.80481, 0.200434, 0.62427, 0.548904, 0.412576, 0.799209, 0.814538, 0.899836, 0.0454679, 1.1251, 0.259029, 0.916927, 0.60018, 0.467566, 0.137097, 1.70739, 0.549783, 0.102215, 0.177023, 0.625615, 0.546052, 0.543288, 0.0259089, 0.483088, 0.667167, 0.0885382, 0.045773, 0.97617, 0.687016, 0.129203, 0.645544, 0.398033, 0.23715, 1.17768, 1.12717, 0.347425, 1.10386, 0.336478, 0.588247, 0.224109, 0.209867, 0.674293, 2.24731, 0.35245, 1.02808, 0.894251, 0.310105, 0.644519, 2.21155, 0.644078, 0.497872, 1.01256, 1.08578, 1.42189, 0.951379, 0.805102, 0.549124

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

S_n: 0.9, 0.916667, 0.959091, 0.928125, 0.929167, 0.947115, 0.959677, 0.936806, 0.949085, 0.949457, 0.9625, 0.954911, 0.936885, 0.930492, 0.940141, 0.95, 0.946142, 0.943169, 0.941346, 0.946615, 0.939109, 0.939858, 0.939752, 0.942888, 0.950103, 0.932937, 0.942176, 0.946875, 0.948138, 0.945805, 0.947185, 0.944952, 0.943401, 0.944804, 0.947953, 0.946591, 0.942472, 0.942473, 0.948037, 0.945727, 0.943284, 0.94909, 0.949763, 0.949479, 0.945192, 0.943584, 0.951786, 0.945869, 0.944813, 0.946697, 0.946614, 0.942432, 0.946695, 0.946758, 0.942113, 0.94615, 0.949333, 0.935096, 0.943428, 0.941765, 0.946221, 0.948979, 0.941439, 0.947943, 0.938357, 0.943482, 0.938595, 0.942039, 0.939443, 0.943425, 0.946368, 0.945892, 0.943594, 0.946277, 0.943497, 0.944515, 0.944882, 0.949449, 0.94821, 0.944697, 0.944701, 0.94492, 0.945043, 0.943209, 0.944151, 0.945335, 0.939066, 0.949513, 0.940136, 0.946749, 0.950554, 0.945696, 0.946909, 0.944903, 0.944772, 0.94291, 0.948207, 0.94573, 0.941853, 0.945035, 0.945509, 0.949679, 0.943518, 0.946778, 0.947217, 0.943108, 0.944209, 0.941884, 0.943184, 0.946818, 0.943103, 0.944537, 0.944452, 0.945186, 0.945578, 0.946766, 0.948623, 0.944561, 0.945812, 0.946162, 0.945258, 0.947174, 0.944988, 0.944562, 0.9438, 0.945847, 0.94582, 0.943455, 0.945164, 0.946865, 0.942281, 0.94497, 0.943986, 0.945495, 0.943312, 0.946209, 0.943337, 0.944206, 0.94063, 0.948204, 0.943545, 0.943892, 0.94346, 0.947294, 0.945371, 0.94084, 0.945075, 0.946043, 0.940182, 0.941555, 0.943775, 0.94504, 0.9477, 0.943424, 0.946498, 0.946553, 0.94451, 0.944784, 0.945006, 0.942871, 0.943649, 0.944665, 0.945052, 0.941054, 0.94563, 0.942479, 0.942253, 0.949282, 0.945318, 0.942332, 0.946387, 0.941852, 0.94534, 0.945035, 0.94591, 0.943422, 0.94702, 0.947319, 0.946044, 0.95, 0.944478, 0.94386, 0.943661, 0.94244, 0.944408, 0.944141, 0.945959, 0.944298, 0.941047, 0.939588, 0.944808, 0.943031, 0.943249, 0.945807, 0.944477, 0.945146, 0.943336, 0.943484, 0.946203, 0.942959

S_n_pr: 4.71322, 2.94865, 1.54298, 1.73551, 1.62522, 0.275082, 1.60508, 0.816463, 0.483652, 0.522948, 1.90392, 1.10041, 0.808026, 1.48486, 0.463342, 0.580489, 0.172023, 0.14278, 0.33573, 0.22206, 0.572597, 0.493236, 0.504484, 0.172496, 0.591426, 1.22609, 0.247916, 0.249632, 0.383382, 0.136325, 0.282498, 0.0460275, 0.118215, 0.0303892, 0.363787, 0.219554, 0.216493, 0.216414, 0.37262, 0.128092, 0.130607, 0.484122, 0.5554, 0.525346, 0.071478, 0.0987928, 0.76955, 0.143085, 0.0313485, 0.230802, 0.22195, 0.220805, 0.230617, 0.237193, 0.254589, 0.172911, 0.509843, 0.997444, 0.115334, 0.291364, 0.180383, 0.472365, 0.32591, 0.362709, 0.652237, 0.109642, 0.626989, 0.262409, 0.537245, 0.11565, 0.195903, 0.145542, 0.0977222, 0.186353, 0.107979, 0.000270852, 0.0386127, 0.522203, 0.390944, 0.0190345, 0.0194345, 0.0426424, 0.0556256, 0.138489, 0.0387875, 0.0865332, 0.577127, 0.528887, 0.463849, 0.236279, 0.639177, 0.124835, 0.25322, 0.0408937, 0.0269531, 0.170195, 0.390641, 0.128454, 0.28203, 0.054853, 0.105006, 0.546487, 0.105829, 0.239373, 0.285829, 0.149158, 0.0326247, 0.278752, 0.141159, 0.243572, 0.149678, 0.00208409, 0.00691509, 0.0707585, 0.112306, 0.238144, 0.434707, 0.00459434, 0.137107, 0.174134, 0.0784229, 0.281298, 0.049818, 0.00471166, 0.0758974, 0.140755, 0.137948, 0.112438, 0.0684604, 0.248607, 0.236743, 0.0478897, 0.0561982, 0.103578, 0.127575, 0.179153, 0.124951, 0.0329953, 0.411603, 0.390341, 0.102936, 0.0662288, 0.111938, 0.293992, 0.0903982, 0.389296, 0.0590835, 0.161523, 0.458965, 0.313623, 0.0785819, 0.0553189, 0.33702, 0.115779, 0.209723, 0.215523, 0.000735044, 0.0282186, 0.0517868, 0.174331, 0.0919646, 0.015651, 0.0566658, 0.36667, 0.117853, 0.215811, 0.239686, 0.504502, 0.0847934, 0.231401,

0.197923, 0.282213, 0.0870853, 0.0547853, 0.14745, 0.115973, 0.265029, 0.296684, 0.161626, 0.580489, 0.00411094, 0.0695393, 0.0906679, 0.219925, 0.0115333, 0.0397788, 0.152613, 0.0232545, 0.367429, 0.521901, 0.0308, 0.157364, 0.134245, 0.136606, 0.00421834, 0.0665756, 0.12507, 0.109412, 0.178519, 0.164936