Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет

Методы оптимизации
Лабораторная работа №4

Факультет: ФПМИ
Группа: ПМ-63
Студенты: Кожекин М.В.
Утюганов Д.С.
Вариант: 5

Новосибирск
2019

1. Цель работы

Ознакомиться со статистическими методами поиска при решении задач нелинейного программирования. Изучить методы случайного поиска при определении глобального экстремума функции.

2. Задание

1. Реализовать программу для решения задачи поиска глобального экстремума с использованием метода **простого случайного поиска, 1-3 алгоритмов глобального поиска**.

2. Исследовать метод простого случайного поиска при различных ε и P

ε	P	N	(x, y)	f(x, y)

3. Провести сравнительный анализ 1-3 алгоритмов глобального поиска по точности получаемого решения и числу вычислений целевой функции. Исследование провести при различных значениях числа попыток .

Условие задачи:

Найти **максимум** заданной функции:

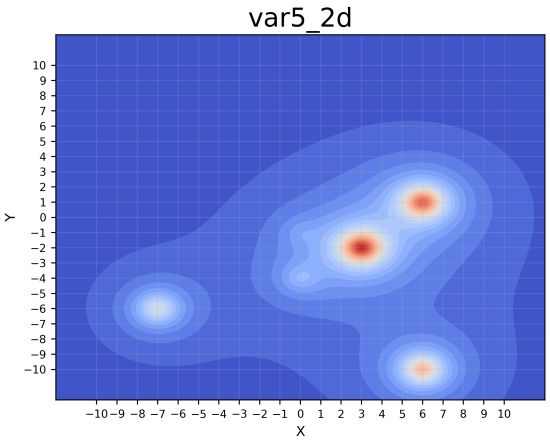
$$f(x,y) = \sum_{i=1}^{6} \frac{C_i}{1 + (x - a_i)^2 + (y - b_i)^2}$$

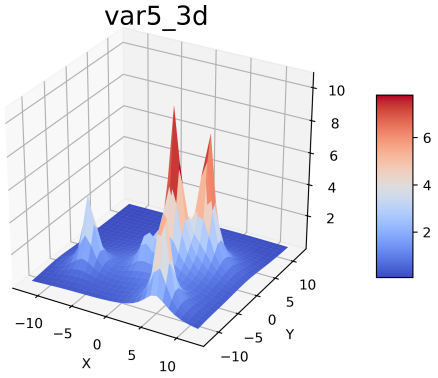
по области $-10 \le x \le 10, -10 \le y \le 10$

Вариант 5

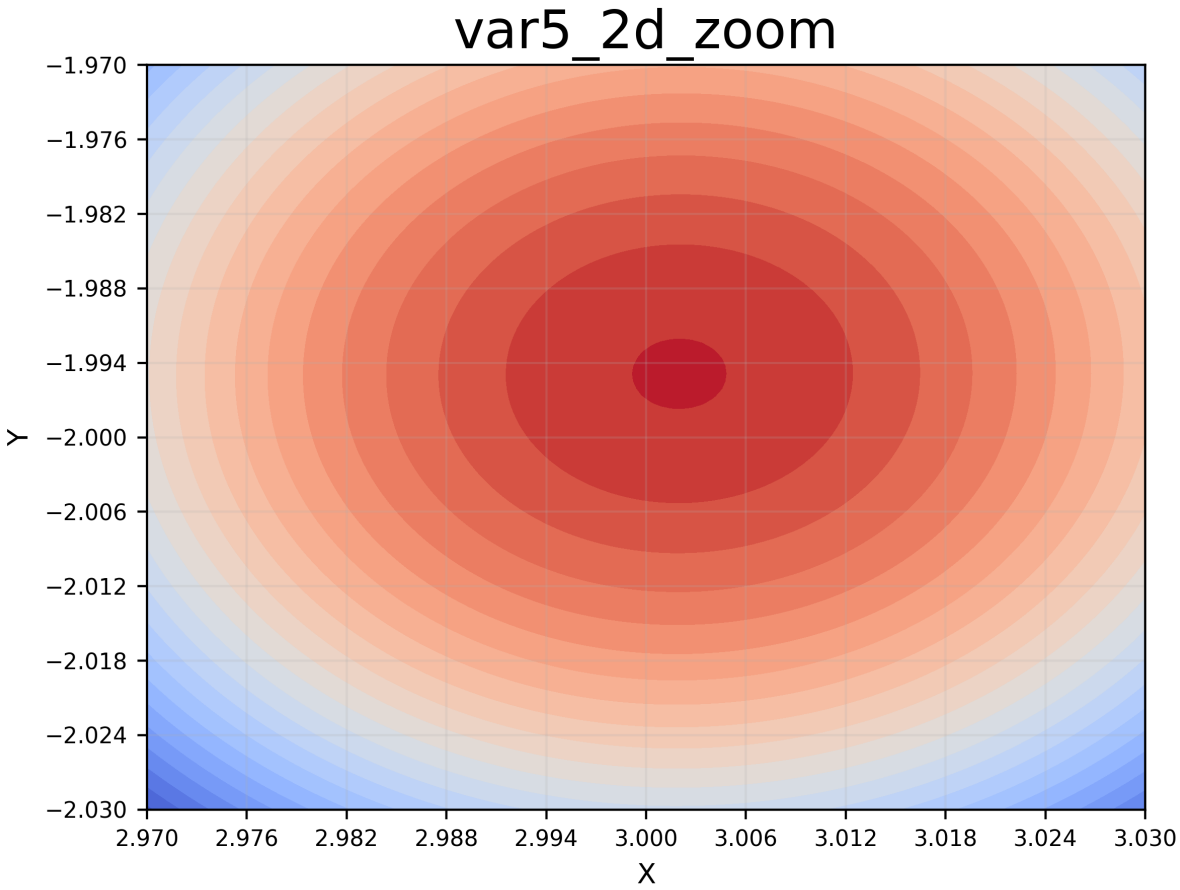
i	1	2	3	4	5	6
C_i	1	2	10	5	7	9
a_i	0	0	3	-7	6	6
b_i	-1	-4	-2	-6	-10	1

Изобразим целевую функцию:





Кажется, что её максимум это точка (3, -2), но программа находит истинный супремум:



3. Зависимость скорости сходимости метода простого случайного поиска при различных ε и P

P \ ε	1	$5e-1$	$1e-1$	$5e-2$	$1e-2$
0.1	42	168	4,214	16,857	$4.21 \cdot 10^5$
0.3	142	570	14,266	57,067	$1.43 \cdot 10^6$
0.5	276	1,108	27,725	$1.11 \cdot 10^5$	$2.77 \cdot 10^6$
0.7	480	1,925	48,158	$1.93 \cdot 10^5$	$4.82 \cdot 10^6$
0.9	919	3,682	92,102	$3.68 \cdot 10^5$	$9.21 \cdot 10^6$

3.1. Вывод

С ростом точности или вероятности растёт и число попыток, необходимых для поиска глобального максимума.

4. Сравнительный анализ 1-3 алгоритмов глобального поиска по точности получаемого решения и числу вычислений целевой функции

m \ алгоритм	алгоритм 1	алгоритм 2	алгоритм 3
10	11553 2.46659	862 2.46659	10145 2.46659
20	27697 2.46659	667 2.46659	26217 0.322755
30	64946 2.46659	388 -2.04217e-06	32609 0.322755
40	31902 2.46659	1669 -4.98634e-06	47148 2.46659

4.1. Вывод

Исходя из результатов мы можем заметить, что второй алгоритм считает быстрее всех. Это обусловлено тем, что мы не запускаем детерминированный алгоритм поиска из тех точек, которые не дадут нам лучшего решения.

С ростом числа точек m мы замечаем рост точности решения и рост числа вычислений целевой функции.

5. Исходный код программы

head.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#define UNICODE
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
#include <functional>
#include <cmath>
#include <math.h>
#include <random>
#include <algorithm>

using namespace std;


// float || double
typedef double real;
typedef vector <real> vector1D;
typedef vector <vector <real>> matrix2D;



// Умножение на константу
inline bool operator==(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    for (int i = 0; i < a.size(); ++i)
        if (a[i] != b[i])
```

```
            return false;

    return true;
}


// Сложение векторов
inline vector1D operator+(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    vector1D result = a;
    for (int i = 0; i < b.size(); i++)
        result[i] += b[i];
    return result;
}


// Сложение матриц
inline matrix2D operator+(const matrix2D& a, const matrix2D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    matrix2D result = a;
    for (int i = 0; i < b.size(); i++)
        for (int j = 0; j < b.size(); j++)
            result[i][j] += b[i][j];
    return result;
}


// Сложение матриц
inline matrix2D operator/(const matrix2D& a, const real& b) {

    matrix2D result = a;
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < a.size(); j++)
            result[i][j] /= b;
    return result;
}


// Вычитание векторов
inline vector1D operator-(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
    if (a.size() != b.size())
        throw std::exception();
#endif
    vector1D result = a;
    for (int i = 0; i < b.size(); i++)
        result[i] -= b[i];
    return result;
}


inline vector1D operator-(const vector1D& a) {
    vector1D result = a;
    for (int i = 0; i < a.size(); i++)
```

```
93          result[i] = -result[i];
94      return result;
95  }
96
97
98  // Умножение матрицы на вектор
99  inline vector1D operator*(const matrix2D& a, const vector1D& b) {
100     vector1D result = { 0.0, 0.0 };
101     for (int i = 0; i < a.size(); i++)
102         for (int j = 0; j < a.size(); j++)
103             result[i] += a[i][j] * b[j];
104     return result;
105 }
106
107
108 // Умножение на константу
109 inline vector1D operator*(const vector1D& a, double b) {
110     vector1D result = a;
111     for (int i = 0; i < result.size(); i++)
112         result[i] *= b;
113     return result;
114 }
115
116
117 // Умножение на константу
118 inline vector1D operator*(double b, const vector1D& a) {
119     return operator*(a, b);
120 }
121
122
123 // Деление на константу
124 inline vector1D operator/(const vector1D& a, double b) {
125     vector1D result = a;
126     for (int i = 0; i < result.size(); i++)
127         result[i] /= b;
128     return result;
129 }
130
131
132 // Деление на константу
133 inline vector1D operator/(double b, const vector1D& a) {
134     return operator/(a, b);
135 }
136
137
138 // Скалярное произведение
139 inline real operator*(const vector1D& a, const vector1D& b) {
140 #ifdef _DEBUG
141     if (a.size() != b.size())
142         throw std::exception();
143 #endif
144     real sum = 0;
145     for (int i = 0; i < a.size(); i++)
146         sum += a[i] * b[i];
147     return sum;
148 }
149
150
151 // Потоковый вывод вектора
152 inline std::ostream& operator<<(std::ostream& out, const vector1D& v) {
```

```
153
154     for (int i = 0; i < v.size() - 1; ++i)
155         out << v[i] << " ";
156     out << v.back();
157     return out;
158 }
159
160 // Потоковый вывод вектора для TeX
161 inline void printTeXVector(std::ofstream &fout, const vector1D &v) {
162     fout << "$(";
163     for (int i = 0; i < v.size() - 1; ++i)
164         fout << v[i] << ", ";
165     fout << v.back() << ")^T$";
166 }
167
168 // Потоковый вывод матрицы
169 inline std::ostream& operator<<(std::ostream& out, const matrix2D& v) {
170     for (int i = 0; i < v.size() - 1; ++i)
171         out << v[i] << "  ";
172     out << v.back();
173     return out;
174 }
175
176 // Евклидова норма
177 real calcNormE(const vector1D &x) {
178     return sqrt(x*x);
179 }
180
181
182 // Определитель матрицы
183 real det(const matrix2D &m) {
184     return m[0][0] * m[1][1] - m[0][1] * m[1][0];
185 }
```

main.cpp

```
1  #include "head.h"
2  int fCalcCount;
3  real A = -10, B = 10;
4  vector1D xExact = { 3.002, 1.99488 };
5  random_device rd;
6  mt19937_64 e2(2634567);
7  uniform_real_distribution<> dist(A, B);
8
9
10 //-----------------------------------------------------------------------------
11 struct methodResult
12 {
13     real E;
14     int iterationsCount;
15     int fCalcCount;
16     vector1D x0, x, xPrev, S, gradf;
17     matrix2D A;
18     real fx, fxPrev, lambda;
19     void printResult(std::ofstream &fout);
20 };
21
22
23
24 // Вывод результатов в поток
25 void methodResult::printResult(std::ofstream &fout) {
26     fout << iterationsCount << "\t"
```

```
        << fCalcCount << "\t";
    printTeXVector(fout, x);
    fout << "\t" << fx << endl;
}



//------------------------------------------------------------------------------
// Функция для исследования min=0 в точке (1,1)
// x – вектор аргументов функции
inline real f(const vector1D &x) {
    fCalcCount++;
    real C[6] = { 1, 2, 10, 5, 7, 9 };
    real a[6] = { 0, 0, 3, -7, 6, 6 };
    real b[6] = { -1, -4, -2, -6, -10, 1 };

    /*
    1 / (1 + (x - 0)^2 + (y + 1)^2) +
    2 / (1 + (x - 0)^2 + (y + 4 )^2) +
    10 / (1 + (x - 3)^2 + (y + 2)^2) +
    5 / (1 + (x + 7)^2 + (y + 6)^2) +
    7 / (1 + (x - 6)^2 + (y + 10)^2) +
    9 / (1 + (x - 6)^2 + (y - 1)^2)

    1 / (1 + (x - 0)^2 + (y + 1)^2) + 2 / (1 + (x - 0)^2 + (y + 4 )^2) + 10 / (1
     + (x - 3)^2 + (y + 2)^2) + 5 / (1 + (x + 7)^2 + (y + 6)^2) + 7 / (1 + (x -
    6)^2 + (y + 10)^2) + 9 / (1 + (x - 6)^2 + (y - 1)^2)
    */

    real result = 0;
    for (size_t i = 0; i < 6; i++)
        result += C[i] / (1 + pow((x[0] - a[i]), 2) + pow((x[1] - b[i]), 2));
    return -result;
}




// Поиск интервала, содержащего минимум
// f – целевая одномерная функция
// a,b – искомые границы отрезка
// x – начальное приближение
// S – орты направления()
void interval(const function<real(const vector1D &x)> &f, real &a, real &b,
    vector1D &x, vector1D &S)
{
    real lambda0 = 0.0;
    real delta = 1.0e-8;
    real lambda_k_minus_1 = lambda0;
    real f_k_minus_1 = f(x + S * lambda_k_minus_1);
    real lambda_k;
    real f_k;
    real lambda_k_plus_1;
    real f_k_plus_1;
    real h;
    if (f(x + S * lambda0) > f(x + S * (lambda0 + delta)))
    {
        lambda_k = lambda0 + delta;
        h = delta;
```

```
    }
    else
    {
        lambda_k = lambda0 - delta;
        h = -delta;
    }
    f_k = f(x + S * lambda_k);
    while (true)
    {
        h *= 2.0;
        lambda_k_plus_1 = lambda_k + h;
        f_k_plus_1 = f(x + S * lambda_k_plus_1);
        if (f_k > f_k_plus_1)
        {
            lambda_k_minus_1 = lambda_k;
            f_k_minus_1 = f_k;
            lambda_k = lambda_k_plus_1;
            f_k = f_k_plus_1;
        }
        else
        {
            a = lambda_k_minus_1;
            b = lambda_k_plus_1;
            if (b < a)
                swap(a, b);
            return;
        }
    }
}



// Вычисление n-го числа Фибоначчи
inline real fib(int n)
{
    real sqrt5 = sqrt(5.0), pow2n = pow(2.0, n);
    return (pow(1.0 + sqrt5, n) / pow2n - pow(1.0 - sqrt5, n) / pow2n) / sqrt5;
}



// Определение коэффициента лямбда методом Фибоначчи
// f - минимизируемая функция
// x - начальное значение
// S - базис
// E - точность
real fibonacci(const function<real(const vector1D &x)> &f, vector1D &x, vector1D
    &S, real E)
{
    real a, b;
    interval(f, a, b, x, S);
    int iter;
    real len = fabs(a - b);
    int n = 0;
    while (fib(n) < (b - a) / E) n++;
    iter = n - 3;
    real lambda1 = a + (fib(n - 2) / fib(n)) * (b - a);
    real f1 = f(x + S * lambda1);
    real lambda2 = a + (fib(n - 1) / fib(n)) * (b - a);
    real f2 = f(x + S * lambda2);
```

```
    for (int k = 0; k < n - 3; k++)
    {
        if (f1 <= f2)
        {
            b = lambda2;
            lambda2 = lambda1;
            f2 = f1;
            lambda1 = a + (fib(n - k - 3) / fib(n - k - 1)) * (b - a);
            f1 = f(x + S * lambda1);
        }
        else
        {
            a = lambda1;
            lambda1 = lambda2;
            f1 = f2;
            lambda2 = a + (fib(n - k - 2) / fib(n - k - 1)) * (b - a);
            f2 = f(x + S * lambda2);
        }
        len = b - a;
    }
    lambda2 = lambda1 + E;
    f2 = f(x + S * lambda2);
    if (f1 <= f2)
        b = lambda1;
    else
        a = lambda1;
    return (a + b) / 2.0;
}



// Метод Розенброка
// f - оптимизируемая функция
// x0 - начальное приближение
// E - точность
// funcname - название функции
methodResult calcByRosenbrock(
    const function<real(const vector1D &x)> &f,
    const vector1D &x0,
    real E,
    const string &funcname)
{
    ofstream fout("report/tableRosenbrock_" + funcname + ".txt");
    ofstream steps("steps/Rosenbrock_" + funcname + ".txt");
    fout << scientific;
    steps << fixed << setprecision(12);
    steps << x0 << endl;

    methodResult result;
    //fCalcCount = 0;
    vector1D xPrev, B, x = x0;
    int maxiter = 1000;
    int iterationsCount = 0, count = 0;
    real lambda1, lambda2;
    matrix2D A(2);
    A[0] = { 1.0, 0.0 };
    A[1] = { 0.0, 1.0 };


    // Начальные ортогональные направления
```

```
    matrix2D S(2);
    S[0] = { 1.0, 0.0 };
    S[1] = { 0.0, 1.0 };

    do {
        xPrev = x;

        // Минимализируем функцию в направлениях S^k_1...S^k_n
        lambda1 = fibonacci(f, x, S[0], E);
        x = x + S[0] * lambda1;
        lambda2 = fibonacci(f, x, S[1], E);
        x = x + S[1] * lambda2;

        // Построение новых ортогональных направлений при
        // сортировке лямбд в порядке убывания по абсолютным значениям
        A[0] = S[0] * lambda1 + S[1] * lambda2;
        if (fabs(lambda1 >= lambda2))
            A[1] = S[1] * lambda2;
        else
            A[1] = S[0] * lambda1;


        // Ортогонализация ГраммаШмидта
        S[0] = A[0] / calcNormE(A[0]);
        B = A[1] - S[1] * A[1] * S[1];
        if (calcNormE(B) > E)
            S[1] = B / calcNormE(B);
        iterationsCount++;


        fout << iterationsCount << "\t"
            << fCalcCount << "\t"
            << x << "\t"
            << f(x) << endl;

        steps << x << endl;

    } while (abs(f(x) - f(xPrev)) > E && abs(calcNormE(x) - calcNormE(xPrev)) >
    E && iterationsCount < maxiter);

    fout.close();
    steps.close();

    result.E = E;
    result.iterationsCount = iterationsCount;
    result.fCalcCount = fCalcCount;
    result.x0 = x0;
    result.x = x;
    result.fx = f(x);

    return result;
}



//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
// Исследования
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
```

```
262
263
264 // Метод простого случайного поиска
265 int simpleRandomSearch(real E, real P, bool doVisualisation) {
266
267     cout << "Simple random search method" << endl;
268     real V = pow((B – A), 2);
269     real V_E = E * E;
270     real P_E = V_E / V;
271     real N = log(1.0 – P) / log(1.0 – P_E);
272     cout << N << endl;
273
274
275     vector1D x1, x2, minX = { dist(e2), dist(e2) };
276     real fx1, fx2, fxMin = f(minX);
277
278     ofstream fout("steps/simpleRandomSearch.txt");
279     for (size_t i = 0; i < N; i++)
280     {
281         x1 = { dist(e2), dist(e2) };
282         x2 = { dist(e2), dist(e2) };
283         fx1 = f(x1);
284         fx2 = f(x2);
285         if (doVisualisation) {
286             if (fx1 < fxMin && fx1 < fx2) {
287                 fxMin = fx1;
288                 minX = x1;
289                 fout << minX << endl;
290             }
291             else if (fx2 < fxMin && fx2 < fx1) {
292                 fxMin = fx2;
293                 minX = x2;
294                 fout << minX << endl;
295             }
296         }
297         else {
298             if (fx1 < fxMin && fx1 < fx2) {
299                 fxMin = fx1;
300                 minX = x1;
301             }
302             else if (fx2 < fxMin && fx2 < fx1) {
303                 fxMin = fx2;
304                 minX = x2;
305             }
306         }
307
308
309     }
310     //cout << minX;
311     fout << minX << "\t";
312     // Визуализация
313     if (doVisualisation) {
314         string runVisualisation = "python plot.py";
315         system(runVisualisation.c_str());
316     }
317     fout.close();
318     return N;
319 }
320
321
```

```
322 void table1() {
323
324     vector<real> Es = { 1, 5e-1, 1e-1, 5e-2, 1e-2 };
325     ofstream fout("report/table1.txt");
326     fout << "a\t$1$\t$5e-1$\t$1e-1$\t$5e-2$\t$1e-2$" << endl;
327     for (real P = 0.1; P <= 1; P += 0.2)
328     {
329         fout << "$" << P << "$\t";
330         for (auto E : Es)
331         {
332             fout << simpleRandomSearch(E, P, false) << "\t";
333         }
334         fout << endl;
335     }
336     fout.close();
337 }
338
339
340
341 // ый1- алгоритм
342 pair <int, double> alg1(int m) {
343
344     fCalcCount = 0;
345     methodResult res;
346     vector1D x, minX = { dist(e2), dist(e2) };
347     real fxMin = f(minX);
348     real E = 1e-12;
349     int count = 0;
350     while (count < m) {
351         x = { dist(e2), dist(e2) };
352         res = calcByRosenbrock(f, x, E, "alg1");
353         if (res.fx < fxMin) {
354             minX = res.x;
355             fxMin = res.fx;
356             count = 0;
357         }
358         else count++;
359     }
360     return make_pair(fCalcCount, calcNormE(res.x) - calcNormE(xExact));
361 }
362
363
364
365 // ой2- алгоритм
366 pair <int, double> alg2(int m) {
367
368     fCalcCount = 0;
369     methodResult res;
370     vector1D x, minX = { dist(e2), dist(e2) };
371     real fxMin = f(minX), fx;
372     real E = 1e-12;
373     int count = 0;
374     while (count < m) {
375         do {
376             x = { dist(e2), dist(e2) };
377             fx = f(x);
378             count++;
379         } while (fx > fxMin && count < m);
380
381         if (count < m) {
```

```
            res = calcByRosenbrock(f, x, E, "alg2");
            if (res.fx < fxMin) {
                minX = res.x;
                fxMin = res.fx;
                count = 0;
            }
        }
    }
    return make_pair(fCalcCount, calcNormE(res.x) - calcNormE(xExact));
}


// ий3- алгоритм
pair <int, double> alg3(int m) {

    fCalcCount = 0;
    methodResult res;
    vector1D x = { dist(e2), dist(e2) }, minX = x;
    real fxMin = f(minX);
    real E = 1e-12;
    int count = 0;
    while (count < m) {
        res = calcByRosenbrock(f, x, E, "alg3");
        x = { dist(e2), dist(e2) };
        int step = 1;
        vector1D xTmp = res.x + (res.x - x) * step;
        real fxTmp = f(xTmp);
        while (fxTmp >= res.fx &&
            A < xTmp[0] && xTmp[0] < B &&
            A < xTmp[1] && xTmp[1] < B)
        {
            xTmp = res.x + (res.x - x) * step;
            fxTmp = f(xTmp);
            step++;
            count++;
        }

        if (res.fx < fxMin) {
            minX = res.x;
            fxMin = res.fx;
            count = 0;
        }
    }
    return make_pair(fCalcCount, calcNormE(res.x) - calcNormE(xExact));
}


void table2() {
    pair <int, real> result;
    ofstream fout("report/table2.txt");
    fout << "a\алгоритмt$1$\алгоритмt$2$\алгоритмt$3$" << endl;
    for (size_t m = 10; m < 50; m += 10)
    {
        fout << m << "\t";
        result = alg1(m);
        fout << result.first << " " << result.second << "\t";
        result = alg2(m);
        fout << result.first << " " << result.second << "\t";
        result = alg3(m);
        fout << result.first << " " << result.second << endl;
```

```
    }
    fout.close();
}

void main()
{
    table1();
    table2();
}
```

plot.py

```
import pylab
import numpy
import sys
import matplotlib.pyplot as plt
import matplotlib.lines as lines
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

DPI = 400

xList = []
yList = []
data = []
text = []

folder = 'pics/'
steps = 'steps/'

# Считывание итогов работы метода
def inputMethodResults(filename):
    global text
    global data
    text = []
    data = []
    with open(steps + filename, 'r') as f:
        for line in f: # read rest of lines
            data.append([ int(x) for x in line.split()])
    for i in range(len(data)):
        text.append('iter:'+str(data[i][0])+'\nfCalc: '+str(data[i][1]))


# Считывание хода метода
def inputSteps(filename):
    global xList
    global yList
    global data
    xList = []
    yList = []
    data = []
    with open(steps + filename, 'r') as f:
        for line in f: # read rest of lines
            data.append([ float(x) for x in line.split()])
    for i in range(len(data)):
        xList.append(data[i][0])
        yList.append(data[i][1])


```

```
51
52
53  # Целевая функция
54  # Вариант 5
55  def f(x, y):
56      C = [1, 2, 10, 5, 7, 9]
57      a = [0, 0, 3, −7, 6, 6]
58      b = [−1, −4, −2, −6, −10, 1]
59      result = 0
60      for i in range(6):
61          result += C[i] / (1 + (x−a[i])**2 + (y−b[i])**2)
62      return result
63
64
65  # Рассчёт значений целевой функции на сетке
66  def buildIsoLines(f):
67      x = numpy.linspace (−12, 12, 100)
68      y = numpy.linspace (−12, 12, 100)
69      xgrid, ygrid = numpy.meshgrid(x, y)
70      zgrid = f(xgrid, ygrid)
71      return xgrid, ygrid, zgrid
72
73
74  # Рассчёт значений целевой функции на сетке
75  def buildIsoLines2(f):
76      x = numpy.linspace (2.97, 3.03, 100)
77      y = numpy.linspace (−2.03, −1.97, 100)
78      xgrid, ygrid = numpy.meshgrid(x, y)
79      zgrid = f(xgrid, ygrid)
80      return xgrid, ygrid, zgrid
81
82
83
84
85  def draw2D():
86      name = 'var5_2d'
87      fig = plt.figure()
88      ax = fig.add_subplot(111)
89      x, y, z = buildIsoLines(f)
90      cs = pylab.contourf(x, y, z, 25, cmap=cm.coolwarm)
91      plt.title(name, fontsize=19)
92      plt.xlabel('X', fontsize=10)
93      plt.ylabel('Y', fontsize=10)
94      ax.set_xticks(numpy.linspace(−10, 10, 21))
95      ax.set_yticks(numpy.linspace(−10, 10, 21))
96      plt.tick_params(axis='both', labelsize=8)
97      plt.grid(alpha=0.2)
98      plt.savefig(folder + name + '.png', dpi=DPI)
99      plt.clf()
100
101
102 def draw2DZoom():
103     name = 'var5_2d_zoom'
104     fig = plt.figure()
105     ax = fig.add_subplot(111)
106     x, y, z = buildIsoLines2(f)
107     cs = pylab.contourf(x, y, z, 25, cmap=cm.coolwarm)
108     plt.title(name, fontsize=19)
109     plt.xlabel('X', fontsize=10)
110     plt.ylabel('Y', fontsize=10)
```

```
    ax.set_xticks(numpy.linspace (2.97, 3.03, 11))
    ax.set_yticks(numpy.linspace (-2.03, -1.97, 11))
    plt.tick_params(axis='both', labelsize=8)
    plt.grid(alpha=0.2)
    plt.savefig(folder + name + '.png', dpi=DPI)
    plt.clf()


def draw3D():
    name = 'var5_3d'
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    X = numpy.linspace (-12, 12, 30)
    Y = numpy.linspace (-12, 12, 30)
    X, Y = numpy.meshgrid(X, Y)
    Z = f(X, Y)
    surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm,
            linewidth=0, antialiased=True)
    fig.colorbar(surf, shrink=0.5, aspect=5)
    plt.title(name, fontsize=19)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.savefig(folder + name + '.png', dpi=DPI)


def simpleRandomSearch():
    name = 'simpleRandomSearch'
    inputSteps(name + '.txt')
    fig = plt.figure()
    ax = fig.add_subplot(111)
    x, y, z = buildIsoLines(f)
    cs = pylab.contourf(x, y, z, 25, cmap=cm.coolwarm)
    plt.title(name, fontsize=19)
    plt.xlabel('X', fontsize=10)
    plt.ylabel('Y', fontsize=10)
    ax.set_xticks(numpy.linspace(-10, 10, 21))
    ax.set_yticks(numpy.linspace(-10, 10, 21))
    plt.tick_params(axis='both', labelsize=8)
    plt.grid(alpha=0.2)
    plt.plot(xList, yList, linewidth=0.5, color='grey')
    for i in range(len(xList)):
        plt.scatter(xList[i], yList[i], s=2, color='black')
    plt.savefig(folder + name + '.png', dpi=DPI)
    plt.clf()


if __name__ == '__main__':
    #simpleRandomSearch()
    draw2D()
    draw2DZoom()
    draw3D()
```