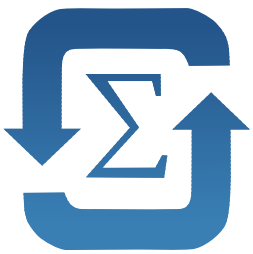
Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики

Лабораторная работа №3
по дисциплине «Методы оптимизации»

Метод штрафных функций



Факультет: ПМИ

Группа: ПМ-63

Студенты: Шепрут И.И.
Крашенинник Н.А.
Пешкичева А.А.

Вариант: 4

Преподаватель: Чимитова Е.В.

Новосибирск
2019

1 Цель работы

Ознакомиться с методами штрафных функций при решении задач нелинейного программирования. Изучить типы штраф¬ных и барьерных функций, их особенности, способы и области применения, влияние штрафных функций на сходимость алгоритмов, зависимость точности решения задачи нелинейного программирования от величины коэффициента штрафа.

2 Задание

• Применяя методы поиска минимума 0-го порядка, реализовать программу для решения задачи нелинейного программирования с использованием метода штрафных функций/барьерных функций.
• Исследовать сходимость метода штрафных функций/барьерных функций в зависимости от:
1. выбора штрафных функций,
2. начальной величины коэффициента штрафа,
3. стратегии изменения коэффициента штрафа,
4. начальной точки,
5. задаваемой точности.

• Сформулировать выводы.

Вариант:

4. $f(x, y) = 2(x - y)^2 + 14(y - 3)^2 \to min$
при ограничении:
а) $y - x \geqslant 0.2$
б) $x = -y$

3 Исследования

Во всех исследованиях, если не указано иное: $\varepsilon = 10^{-7}$; коэффициент увеличения штрафа: 2; начальный коэффициент штрафа: 1; начальное приближение: $(-1, 0)^T$; стратегия изменения штрафа: домножать текущий штраф на коэффициент увеличения штрафа.

3.1 $y - x - 0.2 \geqslant 0$

Функция штрафа для этого случая:

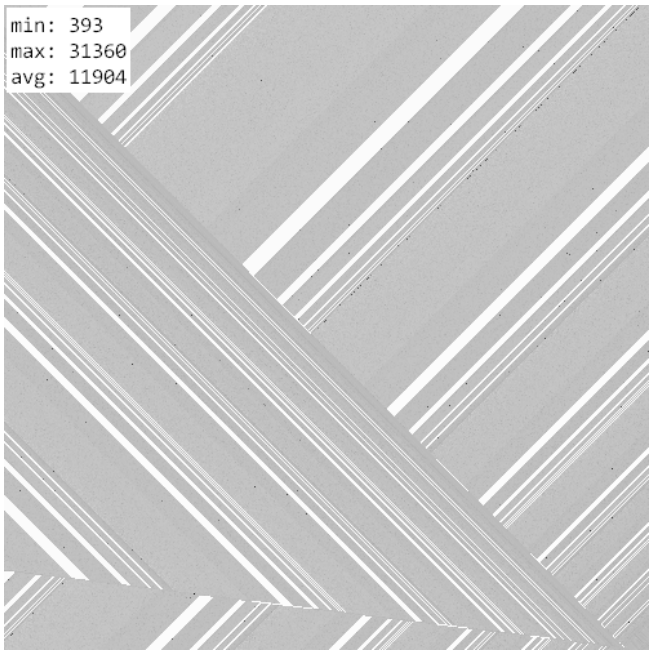
$$G = \begin{cases} 0, & y - x - 0.2 \geqslant 0, \\ -(y - x - 0.2), & \text{иначе} \end{cases}$$

Таблица исследования в зависимости от требуемой стартовой точности:

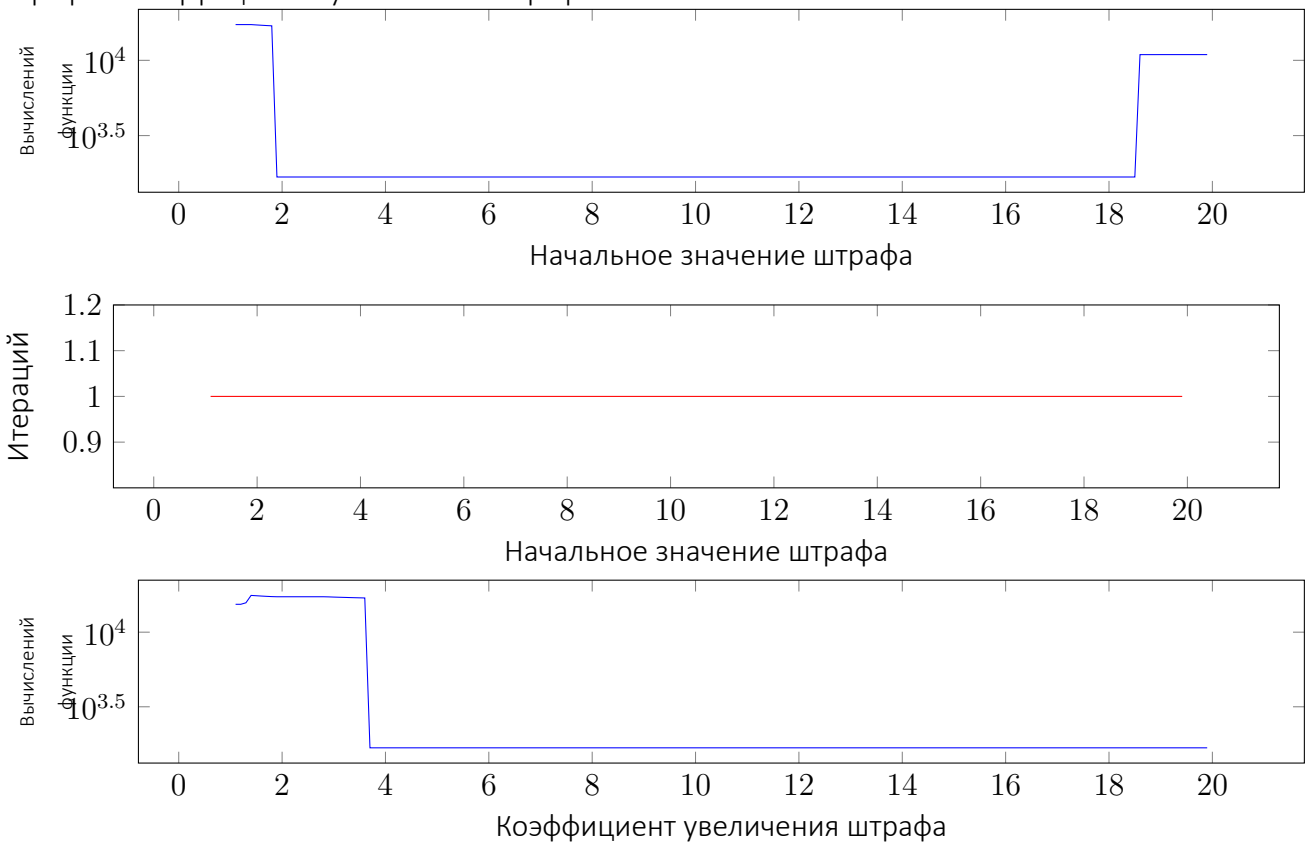
$\varepsilon = 10^i$	Итераций	Вычислений f	Ответ	$G(\mathbf{x})$	$\|\|\nabla f(x, y)\|\|_{L_2}$
-3	1	$17{,}273$	$(2.8, 3)^T$	$3.3 \cdot 10^{-13}$	1.13
-4	1	$17{,}273$	$(2.8, 3)^T$	$3.3 \cdot 10^{-13}$	1.13
-5	1	$17{,}273$	$(2.8, 3)^T$	$3.3 \cdot 10^{-13}$	1.13
-6	1	$17{,}273$	$(2.8, 3)^T$	$3.3 \cdot 10^{-13}$	1.13
-7	1	$17{,}273$	$(2.8, 3)^T$	$3.3 \cdot 10^{-13}$	1.13

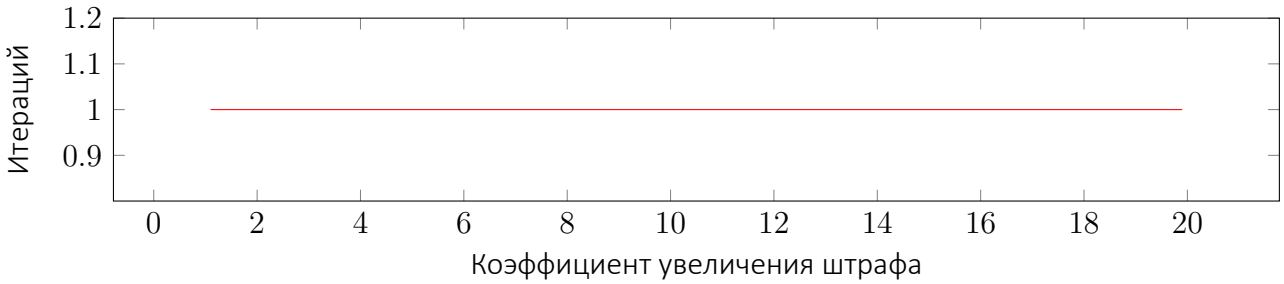
Зависимость числа вычислений функции от начальной точки. Чем темнее пиксель, тем больше требуется вычилений функции. начиная с этой точки; чем светлее, тем меньше. Границы на

изображении: $[-3, 3] \times [-3, 3]$. Сверху слева показана информация о числе вычислений функции для сходимости метода из этой точки.



Зависимость числа итераций и вычислений функции в зависимости от начального значения штрафа и коэффициента увеличения штрафа:





3.1.1 Исследования стратегии изменения штрафа

$$g = 100(x - y + 0.2)$$

Для изменения функции штрафа была выбрана следующая функция. Будет исследовано как она влияет на процесс в зависимости от n:

$$G = \left(\frac{g + |g|}{2}\right)^{2n}$$

n	Итераций	Вычислений f	Ответ	$G(\mathbf{x})$	$\|\|\nabla f(x,y)\|\|_{L_2}$
1	1	770	$(2.8, 3)^T$	$3.2 \cdot 10^{-5}$	1.13
2	1	744	$(2.8, 3)^T$	$3.14 \cdot 10^{-4}$	1.12
3	1	726	$(2.8, 3)^T$	$4 \cdot 10^{-4}$	1.11
4	1	734	$(2.8, 3)^T$	$4.02 \cdot 10^{-4}$	1.11
5	1	726	$(2.8, 3)^T$	$3.81 \cdot 10^{-4}$	1.1
6	1	834	$(2.8, 3)^T$	$3.54 \cdot 10^{-4}$	1.1

3.2 $x = -y$

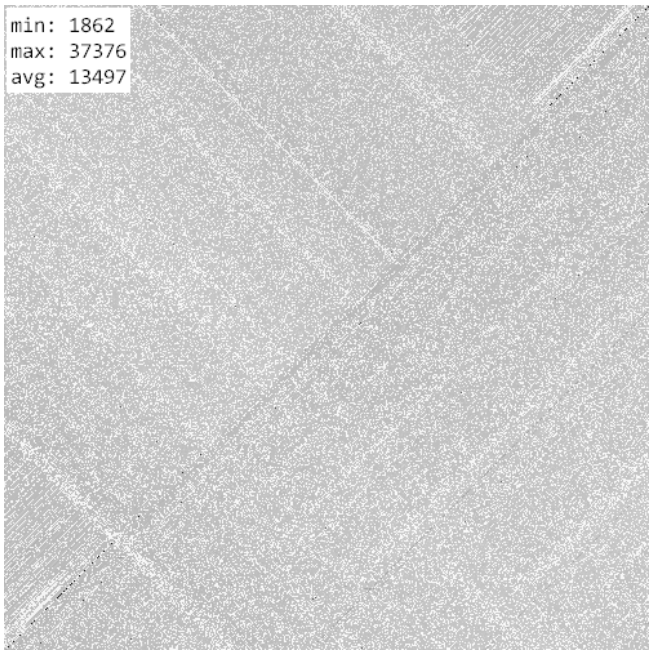
Функция штрафа для этого случая:

$$G = |x + y|$$

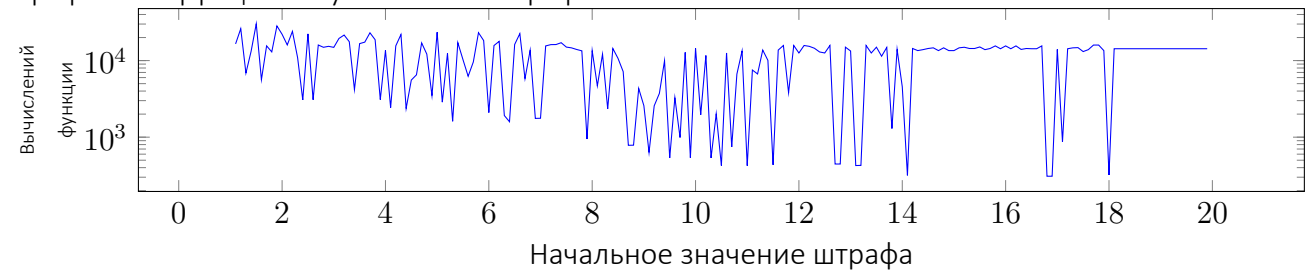
Таблица исследования в зависимости от требуемой стартовой точности:

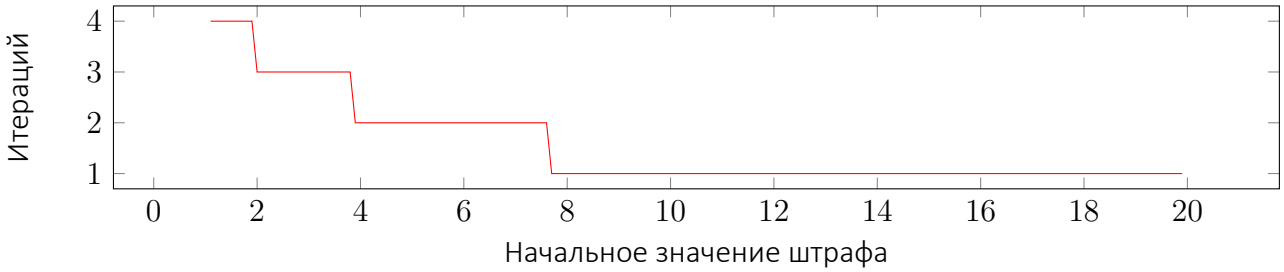
$\varepsilon = 10^i$	Итераций	Вычислений f	Ответ	$G(\mathbf{x})$	$\|\|\nabla f(x,y)\|\|_{L_2}$
-3	4	15,374	$(-1.9, 1.9)^T$	$9.26 \cdot 10^{-9}$	21.77
-4	4	15,374	$(-1.9, 1.9)^T$	$9.26 \cdot 10^{-9}$	21.77
-5	4	15,374	$(-1.9, 1.9)^T$	$9.26 \cdot 10^{-9}$	21.77
-6	4	15,374	$(-1.9, 1.9)^T$	$9.26 \cdot 10^{-9}$	21.77
-7	4	17,993	$(-1.9, 1.9)^T$	$9.26 \cdot 10^{-9}$	21.77

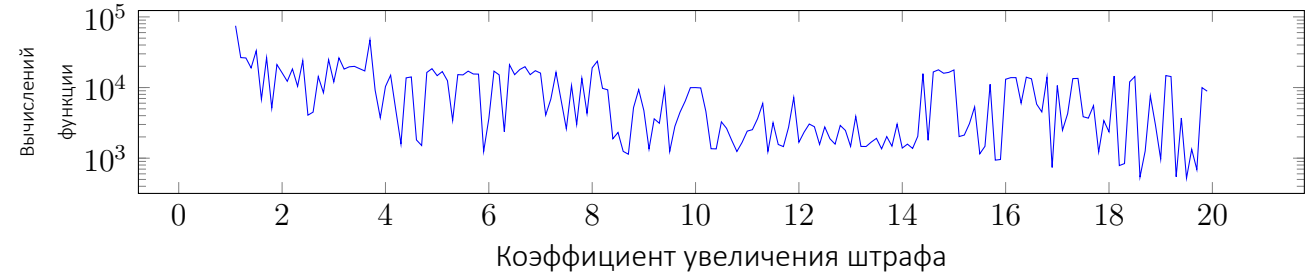
Зависимость числа вычислений функции от начальной точки. Чем темнее пиксель, тем больше требуется вычилений функции. начиная с этой точки; чем светлее, тем меньше. Границы на изображении: $[-3, 3] \times [-3, 3]$. Сверху слева показана информация о числе вычислений функции для сходимости метода из этой точки.

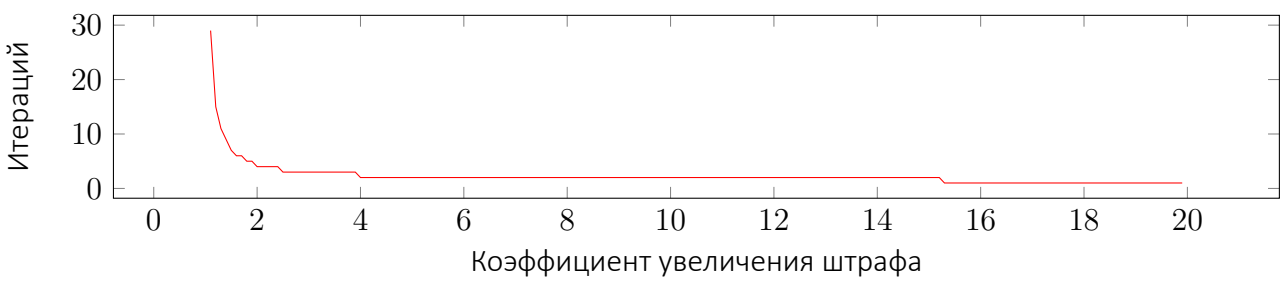


Зависимость числа итераций и вычислений функции в зависимости от начального значения штрафа и коэффициента увеличения штрафа:









3.3 Барьерная функция

Коэффициент увеличения штрафа: 0.5 (чтобы штраф уменьшался).

$$g = 100(x - y + 0.2)$$

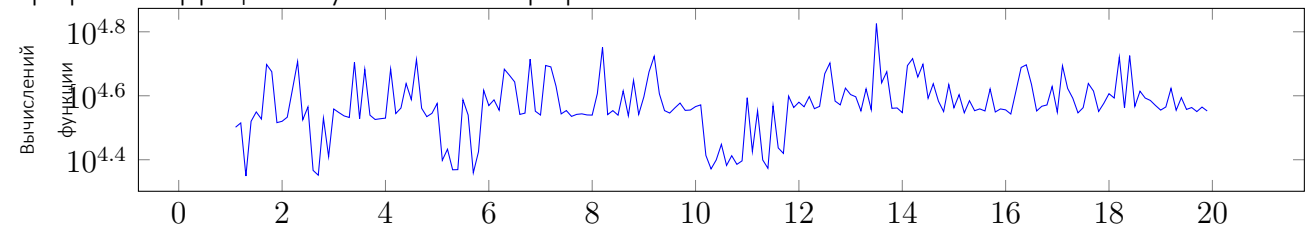
Функция штрафа для этого случая:

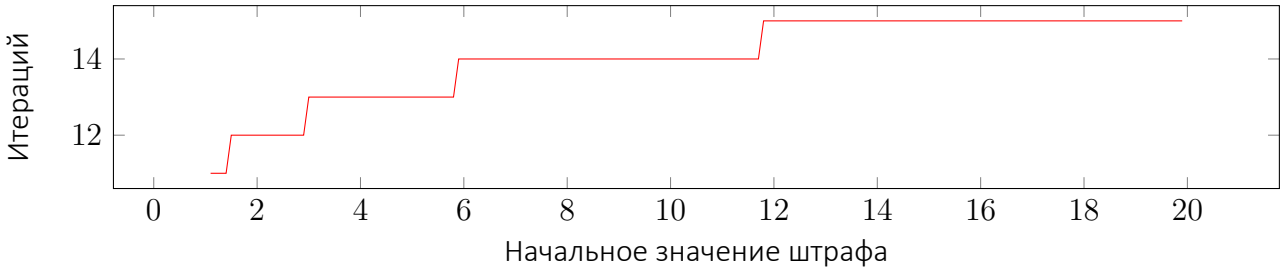
$$G = \begin{cases} -\ln(-g), & g < 0, \\ \infty, & \text{иначе} \end{cases}$$

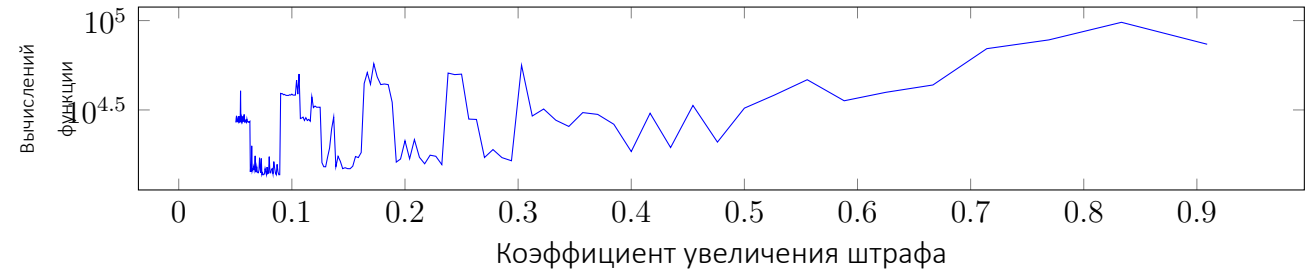
Таблица исследования в зависимости от требуемой стартовой точности:

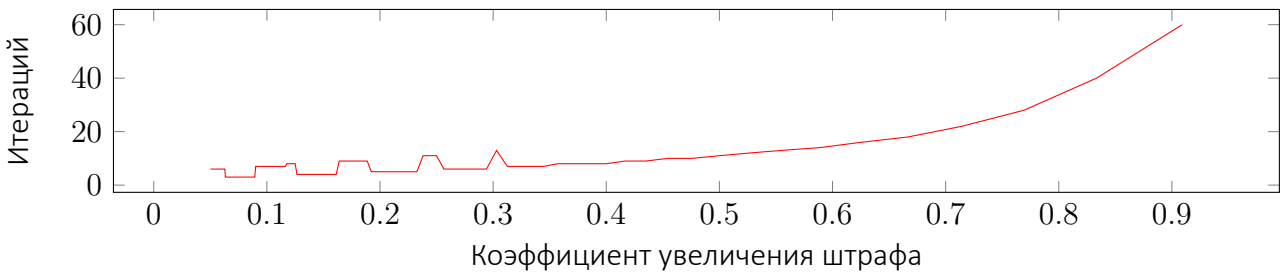
$\varepsilon = 10^i$	Итераций	Вычислений f	Ответ	$G(\mathbf{x})$	$\|\|\nabla f(x,y)\|\|_{L_2}$
-3	1	774	$(2.8, 3)^T$	$3.69 \cdot 10^{-5}$	1.13
-4	1	774	$(2.8, 3)^T$	$3.69 \cdot 10^{-5}$	1.13
-5	4	3,271	$(2.8, 3)^T$	$5.91 \cdot 10^{-6}$	1.13
-6	7	6,488	$(2.8, 3)^T$	$9.01 \cdot 10^{-7}$	1.13
-7	11	32,364	$(2.8, 3)^T$	$6.99 \cdot 10^{-8}$	0.98

Зависимость числа итераций и вычислений функции в зависимости от начального значения штрафа и коэффициента увеличения штрафа:









4 Выводы

1. *Об объеме вычислений в зависимости от требуемой точности:*
   • Для простых штрафных функий требуемая точность практически не влияет на число вычислений.
   • Для барьерных функций требуемая точность значительно влияет на число итераций. Чем больше требуется точность, тем больше итераций.
2. *Об объеме вычислений в зависимости от начального приближения:* по обоим изображениям видно, что нет никакой закономерности. Хотя, возможно, потому что исследуется на квадратичной функции.
3. *Об объеме вычислений в зависимости от начальной величины штрафа и коэффициента изменения штрафа:* делается на основе построенных графиков
   • Для штрафных функций оптимальное начальное значение — 10, коэффициент увеличения штрафа — 10.
   • Для барьерных функций оптимальное начальное значение — 0.1, коэффициент увеличения штрафа — 0.1.
4. *Об объеме вычислений в зависимости от выбора штарфных функций:* функция $G = \left(\frac{g+|g|}{2}\right)^{2n}$ практически никак не повлияла на сходимость и объем вычислений.

5 Код

5.1 Заголовочные файлы

FILE **methods3.h**

```h
#pragma once

#include "../2/methods.h"

struct BarrierResult
{
    ExitType exit;  /// Причина выхода из большой функции
    double k;       /// Коэфициент штрафа

    Vector answer;  /// Итоговый ответ
    int iterations; /// Число итераций больших задач
    int fCount;     /// Число вычислений функции при всех решениях задач
};

MethodResult optimizeHookeJeeves(const Function& f1, const ArgMinFunction& argmin, const Vector& x0,
↪  const double& eps);

Function sumWeight(Function f1, Function f2, double w1, double w2);
Function makeRestriction(int n, Function g);

BarrierResult optimizeWithRestriction(
    const Optimizator& optimizer,
    const Function& f,
    const Function& restriction,
    const ArgMinFunction& argmin,
    const Vector& x0,
    const double& eps,
    const double& penaltyExponent = 2,
    const double& startPenaltyCoef = 1
);
```

FILE **visualize.h**

```h
#pragma once

#include <vector>
#include <string>

#include "../2/visualize/find_borders.h"
#include "methods3.h"
```

```
void visualizeStartPoint(
    const FindBorders& brd,
    const ArgMinFunction& argmin,
    const Function& f,
    const Function& restriction,
    const double& eps,
    const std::string& filename
);
```

5.2 Файлы исходного кода

FILE **make_tables.cpp**

```
#include "methods3.h"
#include "visualize.h"
#include "../2/visualize/find_borders.h"

//-----------------------------------------------------------------------------
double restriction1(const Vector& v) {
    const double& x = v(0);
    const double& y = v(1);
    if (y - x >= 0.2)
        return 0;
    else
        return -(y-x-0.2);
}

//-----------------------------------------------------------------------------
double restriction2(const Vector& v) {
    const double& x = v(0);
    const double& y = v(1);
    return std::fabs(x+y);
}

//-----------------------------------------------------------------------------
double g(const Vector& v) {
    const double& x = v(0);
    const double& y = v(1);
    return 100*(x-y+0.2);
}

//-----------------------------------------------------------------------------
double restriction3(const Vector& v) {
    if (g(v) <= 0)
        //return -1.0 / g(v);
        return -1e-5*log(-g(v));
    else
        return std::numeric_limits<double>::infinity();
}

//-----------------------------------------------------------------------------
double f(const Vector& v) {
    const double& x = v(0);
    const double& y = v(1);
    return 2 * pow(x - y, 2) + 14 * pow(y - 3, 2);
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
void makeFirstTable(
    const ArgMinFunction& argmin,
    const Function& f,
    const Function& restriction,
    const Vector& x0,
    const std::string& file,
    double ex = 2
) {
    std::ofstream fout(file + ".txt");
    fout << std::setprecision(10);

    fout
        << "x0 = " << x0 << ", "
        << "startPenaltyCoef = " << 1 << ", "
        << "penaltyExponent = " << ex << std::endl;

    fout << "10^i\titer\tfCount\tanswer\trestriction_value\tgrad.norm()" << std::endl;
    for (int i = 3; i <= 7; ++i) {
        double eps = pow(10.0, -double(i));
```

```
        auto result = optimizeWithRestriction(optimizeHookeJeeves, f, restriction, argmin, x0, eps, ex
);
        fout
            << -i << "\t"
            << result.iterations << "\t"
            << result.fCount << "\t"
            << result.answer << "\t"
            << restriction(result.answer) * result.k << "\t"
            << grad(f, result.answer).norm() << std::endl;
    }
    fout.close();
}

//-----------------------------------------------------------------------------
void makeSecondTable(
    const ArgMinFunction& argmin,
    const Function& f,
    const Function& restriction,
    const Vector& x0,
    const std::string& file,
    double eps,
    double exmul = 1
) {
    std::ofstream fout(file + ".txt");
    fout << std::setprecision(10);

    fout
        << "x0 = " << x0 << ", "
        << "startPenaltyCoef = " << 1 << ", "
        << "eps = " << eps << std::endl;

    fout << "penaltyExponent\titer\tfCount\trestriction_value\tgrad.norm()" << std::endl;
    for (double i = 1.1; i < 20; i += 0.1) {
        auto result = optimizeWithRestriction(optimizeHookeJeeves, f, restriction, argmin, x0, eps,
        ↪  pow(i, exmul), 1);
        fout
            << pow(i, exmul) << "\t"
            << result.iterations << "\t"
            << result.fCount << "\t"
            << restriction(result.answer) * result.k << "\t"
            << grad(f, result.answer).norm() << std::endl;
    }

    fout.close();
}

//-----------------------------------------------------------------------------
void makeThirdTable(
    const ArgMinFunction& argmin,
    const Function& f,
    const Function& restriction,
    const Vector& x0,
    const std::string& file,
    double eps,
    double ex = 2
) {
    std::ofstream fout(file + ".txt");
    fout << std::setprecision(10);

    fout
        << "x0 = " << x0 << ", "
        << "eps = " << eps << ", "
        << "penaltyExporent = " << ex << std::endl;

    fout << "startPenaltyCoef\titer\tfCount\trestriction_value\tgrad.norm()" << std::endl;
    for (double i = 1.1; i < 20; i += 0.1) {
        auto result = optimizeWithRestriction(optimizeHookeJeeves, f, restriction, argmin, x0, eps,
        ↪  ex, i);
        fout
            << i << "\t"
            << result.iterations << "\t"
            << result.fCount << "\t"
            << restriction(result.answer) * result.k << "\t"
            << grad(f, result.answer).norm() << std::endl;
    }

    fout.close();
}

//-----------------------------------------------------------------------------
void makeFourthTable(
    const ArgMinFunction& argmin,
    const Function& f,
    const Function& g,
    const Vector& x0,
    const std::string& file,
    double eps
) {
```

```
    std::ofstream fout(file + ".txt");
    fout << std::setprecision(10);

    fout
        << "x0 = " << x0 << ", "
        << "eps = " << eps << ", "
        << "startPenaltyCoef = " << 1 << ", "
        << "penaltyExponent = " << 0.5 << std::endl;

    fout << "n\titer\tfCount\tanswer\trestriction_value\tgrad.norm()" << std::endl;
    for (int i = 1; i < 7; ++i) {
        auto restriction = makeRestriction(i, g);
        auto result = optimizeWithRestriction(optimizeHookeJeeves, f, restriction, argmin, x0, eps,
        ↪  0.5);
        fout
            << i << "\t"
            << result.iterations << "\t"
            << result.fCount << "\t"
            << result.answer << "\t"
            << restriction(result.answer) * result.k << "\t"
            << grad(f, result.answer).norm() << std::endl;
    }

    fout.close();
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

int main() {
    Vector x0(2);
    x0 << -1, 0;

    auto argmin = bindArgmin(optimizeGoldenRatio);

    makeFirstTable(argmin, f, restriction1, x0, "table_eps_1");
    makeSecondTable(argmin, f, restriction1, x0, "table_exp_1", 1e-7);
    makeThirdTable(argmin, f, restriction1, x0, "table_start_1", 1e-7);

    makeFirstTable(argmin, f, restriction2, x0, "table_eps_2");
    makeSecondTable(argmin, f, restriction2, x0, "table_exp_2", 1e-7);
    makeThirdTable(argmin, f, restriction2, x0, "table_start_2", 1e-7);

    makeFirstTable(argmin, f, restriction3, x0, "table_eps_3", 0.5);
    makeSecondTable(argmin, f, restriction3, x0, "table_exp_3", 1e-7, -1);
    makeThirdTable(argmin, f, restriction3, x0, "table_start_3", 1e-7, 0.5);

    makeFourthTable(argmin, f, g, x0, "table_fine_g", 0.001);

    FindBorders brd(500, 0, false);
    brd.process({-3, -3});
    brd.process({3, 3});
    brd.finish();
    visualizeStartPoint(brd, argmin, f, restriction1, 0.001, "1");
    visualizeStartPoint(brd, argmin, f, restriction2, 0.001, "2");
}
```

FILE **methods3.cpp**

```
#include "methods3.h"

//-----------------------------------------------------------------------------
MethodResult optimizeHookeJeeves(const Function& f1, const ArgMinFunction& argmin, const Vector& x0,
↪  const double& eps) {
    MethodResult result;
    auto f = setFunctionToCountCalls(&result.fCount, f1);
    result.iterations = 0;

    double argmineps = 1e-7;

    // prepare calculation:
    Vector x = x0;
    Vector x1 = x0;
    Vector s = x0;            // direction to 1D minimization

    double f0 = f(x0), flast = f0;        // start f value
    double f1val = 0;            // value in next finding point
    //debug(x0);

    Vector zero = Vector::Zero(x0.size());
    Matrix zeroM = Matrix::Zero(x0.size(), x0.size());
    result.steps.push_back({ x, f1(x), zero, 0, grad(f1, x), zeroM });

    // optimization:
    while (true) {
```

```
        // examining search
        bool succesfulStep = false;
        double dx = 1e-2;        // start dx value

        int localIterations = 0;
        do {
            for (int i = 0; i < x1.size(); i++) {
                double fp = 0;            // value in x + dx point
                double fm = 0;            // value in x - dx point

                double temp = x1[i];
                x1[i] += dx;
                fp = f(x1);

                if (fp > f0) {
                    x1[i] = temp - dx;
                    fm = f(x1);

                    if (fm > f0) x1[i] = temp;    // x1[i] not changed
                    else f0 = fm;
                }
                else f0 = fp;
            }

            if ((x1 - x).norm() < 1e-13) dx /= 2;
            else succesfulStep = true;

            localIterations++;
            if (localIterations > 100) break;

        } while (!succesfulStep);

        // minimization in finded direction
        //debug(x1);
        //debug(x);
        s = x1 - x;        // direction
        //debug(s);

        auto optimizeFunc = [f, s, x](double lambda) -> double {
            return f(x + lambda * s);
        };

        double lambda = argmin(optimizeFunc, argmineps);
        x1 = x + lambda * s; //debug(x1);

        f1val = f(x1);

        result.steps.push_back({ x1, f1(x1), s, lambda, grad(f1, x1), zeroM });
        result.iterations++;

        // check value exit:
        //if ( < eps) break;
            //debug(x1);
        double sub = (x - x1).norm();
        double fsub = fabs(f1val - flast);
        double gg = fsub / sub;
        if (gg < eps || result.iterations > 100) break;
        else {    // prepare next iteration:
            f0 = f1val;
            flast = f0;
            x = x1;
        }

        // check step exit:
                // there shoulb be check step exit.
    }

    result.answer = x1;
    return result;
}

//---------------------------------------------------------------------------
Function sumWeight(Function f1, Function f2, double w1, double w2) {
    return [=] (const Vector& x) -> double {
        return w1*f1(x) + w2*f2(x);
    };
}

//---------------------------------------------------------------------------
Function makeRestriction(int n, Function g) {
    return [=] (const Vector& x) -> double {
        double gv = g(x);
        return pow((gv + std::fabs(gv))/2.0, 2*n);
    };
}

//---------------------------------------------------------------------------
BarrierResult optimizeWithRestriction(
    const Optimizator& optimizer,
```

```
    const Function& f,
    const Function& restriction,
    const ArgMinFunction& argmin,
    const Vector& x0,
    const double& eps,
    const double& penaltyExponent,
    const double& startPenaltyCoef
) {
    double k = startPenaltyCoef;
    MethodResult res;
    int fCount = 0;
    for (int i = 0; i < 60; ++i) {
        k *= penaltyExponent;
        res = optimizer(sumWeight(f, restriction, 1, k), argmin, x0, eps);
        fCount += res.fCount;
        if (k * restriction(res.answer) < eps)
            return {EXIT_RESIDUAL, k, res.answer, i+1, fCount};
    }
    return {EXIT_ITERATIONS, k, res.answer, 60, fCount};
}
```

FILE **visualize.cpp**

```
#include <sstream>
#include <iostream>

#include <twg/image/image_drawing.h>

#include "visualize.h"

using namespace twg;
using namespace std;

void visualizeStartPoint(
    const FindBorders& brd,
    const ArgMinFunction& argmin,
    const Function& f,
    const Function& restriction,
    const double& eps,
    const string& filename) {
    ImageDrawing_aa img(brd.getCalculatedSize());

    double min1, max1, average1 = 0;
    vector<vector<int>> values(img.width(), vector<int>(img.height(), 0));
    for (int i = 0; i < img.width(); i++) {
        cout << "\r" << 100 * double(i)/img.width() << "%      ";
        for (int j = 0; j < img.height(); j++) {
            vec2 pos(i, j);
            pos = brd.fromImg(pos);
            Vector x(2);
            x << pos.x, pos.y;

            auto result = optimizeWithRestriction(optimizeHookeJeeves, f, restriction, argmin, x, eps);
            double value = result.fCount;
            values[i][j] = value;
            if (i == 0 && j == 0) {
                min1 = value;
                max1 = value;
            }
            if (value > max1) max1 = value;
            if (value < min1) min1 = value;

            average1 += value;
        }
    }

    average1 /= img.width() * img.height();

    for (int i = 0; i < img.width(); i++) {
        for (int j = 0; j < img.height(); j++) {
            const int count = 255;
            double value = values[i][j];
            value = (value - min1)/(max1-min1);
            Color clr = getColorBetween(int(value * count) / double(count), White, Black);
            if (max1 == min1)
                img[Point_i(i, j)] = White;
            else
                img[Point_i(i, j)] = clr;
        }
    }

    {
        ImageDrawing_win imgw(&img);
        wstringstream sout;
        sout << "min: " << int(min1) << endl;
        sout << "max: " << int(max1) << endl;
```

```
        sout << ”avg: ” << int(average1) << endl;
        imgw.setTextStyle(TextStyle(20, L”Consolas”, TEXT_NONE));
        imgw.setPen(Pen(1, Black));

        auto textsize = imgw.getTextSize(sout.str());
        Point_i start(5, 5);
        Point_i offset(3, 3);

        start -= offset;
        textsize += 2 * offset;

        Polygon_d poly;
        poly.array.push_back(start);
        poly.array.push_back(start + Point_d(textsize.x, 0));
        poly.array.push_back(start + textsize);
        poly.array.push_back(start + Point_d(0, textsize.y));
        imgw.setBrush(setAlpha(White, 192));
        imgw.drawPolygon(poly);

        imgw.drawText(Point_d(5, 5), sout.str());
        for (int i = 0; i < img.width(); i++) {
            for (int j = 0; j < img.height(); j++) {
                Color& clr = img[Point_i(i, j)];
                if (getAlpha(clr) == 0)
                    clr = setAlpha(clr, 255);
            }
        }
    }

    twg::saveToPng(&img, wstring(filename.begin(), filename.end()) + L”.png”);
}
```