Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет

Методы оптимизации
Лабораторная работа №3

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

Новосибирск
2019

1. Цель работы

Ознакомиться с методами штрафных функций при решении задач нелинейного программирования. Изучить типы штрафных 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**.

Исследовать сходимость **метода штрафных функций** в зависимости от

- выбора штрафных функций,
- начальной величины коэффициента штрафа,
- стратегии изменения коэффициента штрафа,
- начальной точки,
- задаваемой точности.

Сформулировать выводы.

Применяя методы поиска минимума 0-го порядка, реализовать программу для решения задачи нелинейного программирования с ограничением типа неравенства **(только пункт а)** с использованием **метода барьерных функций**.

Исследовать сходимость **метода барьерных функций (только пункт а)** в зависимости от

- выбора барьерных функций,
- начальной величины коэффициента штрафа,
- стратегии изменения коэффициента штрафа,
- начального приближения,
- задаваемой точности.

Сформулировать выводы.

Вариант 5

При ограничении:

а) x + y <= -1

б) y = x + 1

Параметры методов в большинстве исследований следующие:

$E = 1e^{-12}$

$r_0 = 10$

$rMult = 2$

$\alpha = 8$

3. Зависимость скорости сходимости метода от заданной точности

3.1. Штрафные функции

$$G_j[g_j(\bar{x})] = \frac{1}{2}\{g_j(\bar{x}) + |g_j(\bar{x})|\}$$

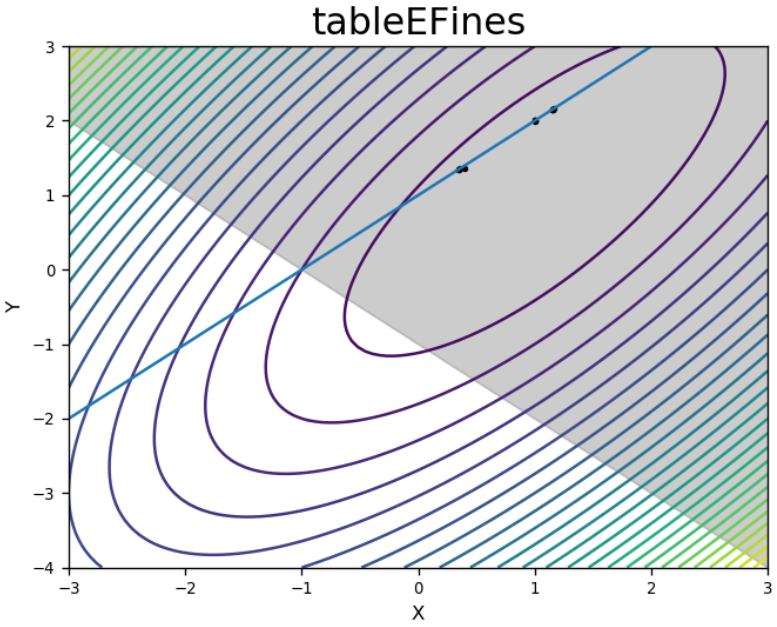
ε	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
$1 \cdot 10^{-3}$	4	271	160	$(9.993447e-01, 1.999343e+00)^T$	4
$1 \cdot 10^{-4}$	5	301	320	$(1.000008e+00, 2.000008e+00)^T$	4
$1 \cdot 10^{-5}$	4	264	160	$(9.999935e-01, 1.999993e+00)^T$	4
$1 \cdot 10^{-6}$	4	263	160	$(1.000001e+00, 2.000001e+00)^T$	4
$1 \cdot 10^{-7}$	4	269	160	$(1.000000e+00, 2.000000e+00)^T$	4

$$G_j[g_j(\bar{x})] = \left[\frac{1}{2}\{g_j(\bar{x}) + |g_j(\bar{x})|\}\right]^2$$

ε	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
$1 \cdot 10^{-3}$	11	767	20,480	$(1.158437e+00, 2.158042e+00)^T$	4.07
$1 \cdot 10^{-4}$	14	984	$1.64 \cdot 10^5$	$(1.155621e+00, 2.155573e+00)^T$	4.07
$1 \cdot 10^{-5}$	17	1,213	$1.31 \cdot 10^6$	$(1.155281e+00, 2.155275e+00)^T$	4.07
$1 \cdot 10^{-6}$	20	1,448	$1.05 \cdot 10^7$	$(1.155233e+00, 2.155232e+00)^T$	4.07
$1 \cdot 10^{-7}$	23	1,698	$8.39 \cdot 10^7$	$(1.155227e+00, 2.155227e+00)^T$	4.07

$$G_j[g_j(\bar{x})] = \left[\frac{1}{2}\{g_j(\bar{x}) + |g_j(\bar{x})|\}\right]^\alpha \text{,если } \alpha \text{ чётное}$$

ε	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
$1 \cdot 10^{-3}$	34	2,143	$1.72 \cdot 10^{11}$	$(3.935642e-01, 1.364665e+00)^T$	4.88
$1 \cdot 10^{-4}$	54	3,429	$1.8 \cdot 10^{17}$	$(3.575230e-01, 1.353514e+00)^T$	5.21
$1 \cdot 10^{-5}$	78	4,945	$3.02 \cdot 10^{24}$	$(3.513336e-01, 1.350961e+00)^T$	5.26
$1 \cdot 10^{-6}$	102	6,468	$5.07 \cdot 10^{31}$	$(3.507720e-01, 1.350737e+00)^T$	5.26
$1 \cdot 10^{-7}$	125	8,037	$4.25 \cdot 10^{38}$	$(3.506919e-01, 1.350688e+00)^T$	5.26



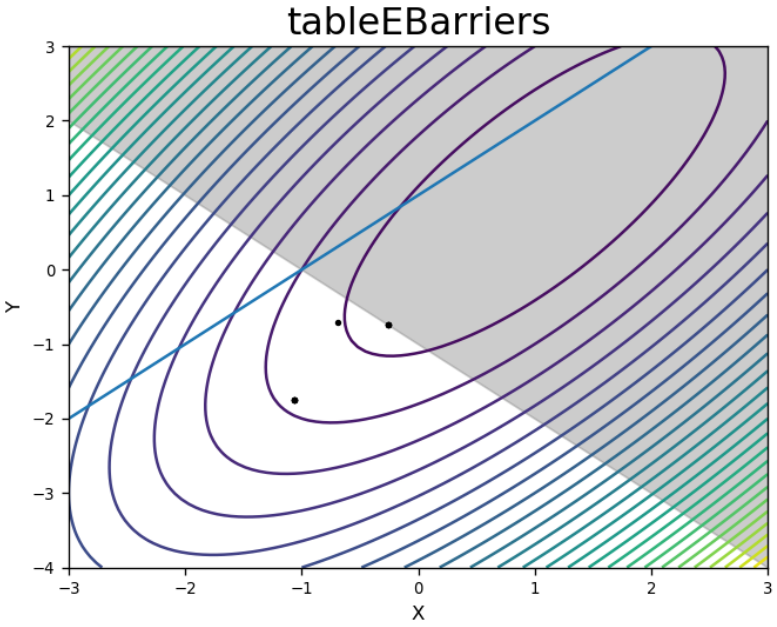
3.2. Барьерные функции

$$G_j[g_j(\bar{x})] = -\frac{1}{g_j(\bar{x})}$$

ε	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
$1 \cdot 10^{-3}$	11	768	$4.88 \cdot 10^{-3}$	$(-6.919326e-01, -6.994372e-01)^T$	8.59
$1 \cdot 10^{-4}$	25	1,912	$2.98 \cdot 10^{-7}$	$(-2.617377e-01, -7.386586e-01)^T$	5.69
$1 \cdot 10^{-5}$	28	2,292	$3.73 \cdot 10^{-8}$	$(-2.603444e-01, -7.397926e-01)^T$	5.68
$1 \cdot 10^{-6}$	31	2,703	$4.66 \cdot 10^{-9}$	$(-2.597752e-01, -7.402737e-01)^T$	5.68
$1 \cdot 10^{-7}$	45	3,477	$2.84 \cdot 10^{-13}$	$(-2.596667e-01, -7.403336e-01)^T$	5.68

$$G_j[g_j(\bar{x})] = -ln\Big[-g_j(\bar{x})\Big]$$

ε	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
$1 \cdot 10^{-3}$	1	94	5	$(-1.061042e+00, -1.751568e+00)^T$	14.65
$1 \cdot 10^{-4}$	1	103	5	$(-1.060892e+00, -1.750890e+00)^T$	14.65
$1 \cdot 10^{-5}$	1	113	5	$(-1.060884e+00, -1.750837e+00)^T$	14.65
$1 \cdot 10^{-6}$	1	122	5	$(-1.060879e+00, -1.750834e+00)^T$	14.65
$1 \cdot 10^{-7}$	1	132	5	$(-1.060879e+00, -1.750834e+00)^T$	14.65



3.3. Вывод

При значении малом значении α (1 и 2) заданная точность не оказывает влияние на скорость сходимости метода штрафных функций.

Для первой барьерной функций рост точности ведёт к увеличению числа итераций и вычислений целевой функции.

Вторая же не может найти более точное решение, поэтому её стоит использовать с минимальной точностью.

4. Зависимость скорости сходимости метода от коэффициента изменения штрафа

4.1. Штрафные функции

$$G_j[g_j(\bar{x})] = \frac{1}{2}\{g_j(\bar{x}) + |g_j(\bar{x})|\}$$

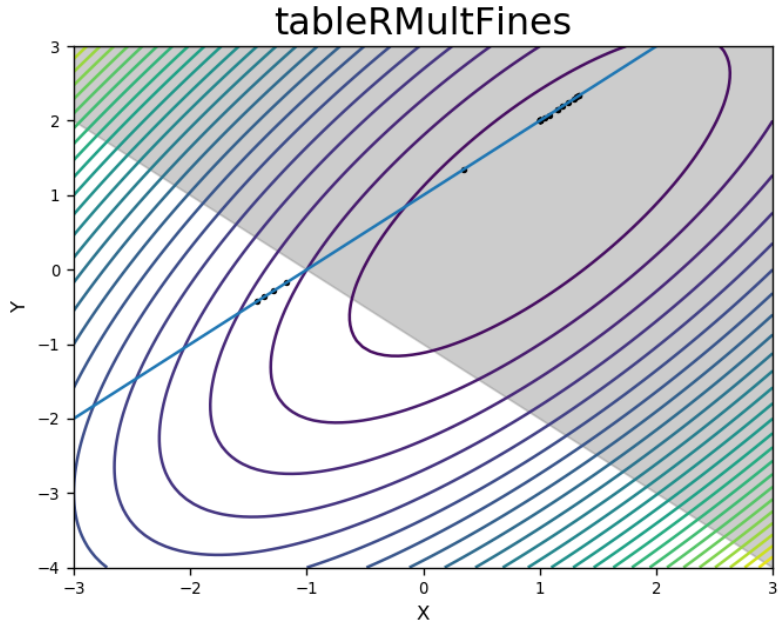
$rMult$	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
2	5	496	320	$(1.000000e+00, 2.000000e+00)^T$	4
3	5	503	2,430	$(1.000000e+00, 2.000000e+00)^T$	4
4	8	685	$6.55 \cdot 10^5$	$(1.000000e+00, 2.000000e+00)^T$	4
5	47	3,234	$7.11 \cdot 10^{33}$	$(9.999861e-01, 1.999986e+00)^T$	4
6	35	2,416	$1.72 \cdot 10^{28}$	$(9.999521e-01, 1.999952e+00)^T$	4
7	37	2,550	$1.86 \cdot 10^{32}$	$(9.999502e-01, 1.999950e+00)^T$	4
8	8	685	$1.68 \cdot 10^8$	$(1.000000e+00, 2.000000e+00)^T$	4
9	26	1,875	$6.46 \cdot 10^{25}$	$(9.999958e-01, 1.999996e+00)^T$	4
10	22	1,587	$1 \cdot 10^{23}$	$(9.999732e-01, 1.999973e+00)^T$	4

$$G_j[g_j(\bar{x})] = \left[\frac{1}{2}\{g_j(\bar{x}) + |g_j(\bar{x})|\}\right]^2$$

$rMult$	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
2	28	3,093	$2.68 \cdot 10^{9}$	$(1.155227e+00, 2.155227e+00)^T$	4.07
3	21	2,381	$1.05 \cdot 10^{11}$	$(1.039906e+00, 2.039906e+00)^T$	4
4	19	2,229	$2.75 \cdot 10^{12}$	$(1.002909e+00, 2.002909e+00)^T$	4
5	19	2,264	$1.91 \cdot 10^{14}$	$(1.000405e+00, 2.000405e+00)^T$	4
6	16	2,069	$2.82 \cdot 10^{13}$	$(1.000008e+00, 2.000008e+00)^T$	4
7	17	1,802	$2.33 \cdot 10^{15}$	$(-1.173890e+00, -1.738896e-01)^T$	18.18
8	12	1,371	$6.87 \cdot 10^{11}$	$(-1.285930e+00, -2.859298e-01)^T$	19.68
9	11	1,261	$3.14 \cdot 10^{11}$	$(-1.364879e+00, -3.648791e-01)^T$	20.78
10	11	1,221	$1 \cdot 10^{12}$	$(-1.425473e+00, -4.254725e-01)^T$	21.65

$$G_j[g_j(\bar{x})] = \left[\frac{1}{2}\{g_j(\bar{x}) + |g_j(\bar{x})|\}\right]^{\alpha}$$,если α чётное

$rMult$	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
2	241	21,121	$3.53 \cdot 10^{73}$	$(3.506846e-01, 1.350685e+00)^T$	5.26
3	158	13,932	$2.43 \cdot 10^{76}$	$(1.078361e+00, 2.078361e+00)^T$	4.02
4	128	11,441	$1.16 \cdot 10^{78}$	$(1.149671e+00, 2.149671e+00)^T$	4.07
5	111	9,999	$3.85 \cdot 10^{78}$	$(1.192329e+00, 2.192329e+00)^T$	4.11
6	100	9,086	$6.53 \cdot 10^{78}$	$(1.247717e+00, 2.247717e+00)^T$	4.18
7	67	6,908	$4.18 \cdot 10^{57}$	$(1.298145e+00, 2.298145e+00)^T$	4.27
8	63	6,541	$7.85 \cdot 10^{57}$	$(1.317640e+00, 2.317640e+00)^T$	4.3
9	61	6,316	$1.62 \cdot 10^{59}$	$(1.336057e+00, 2.336057e+00)^T$	4.34
10	58	6,075	$1 \cdot 10^{59}$	$(1.331247e+00, 2.331247e+00)^T$	4.33



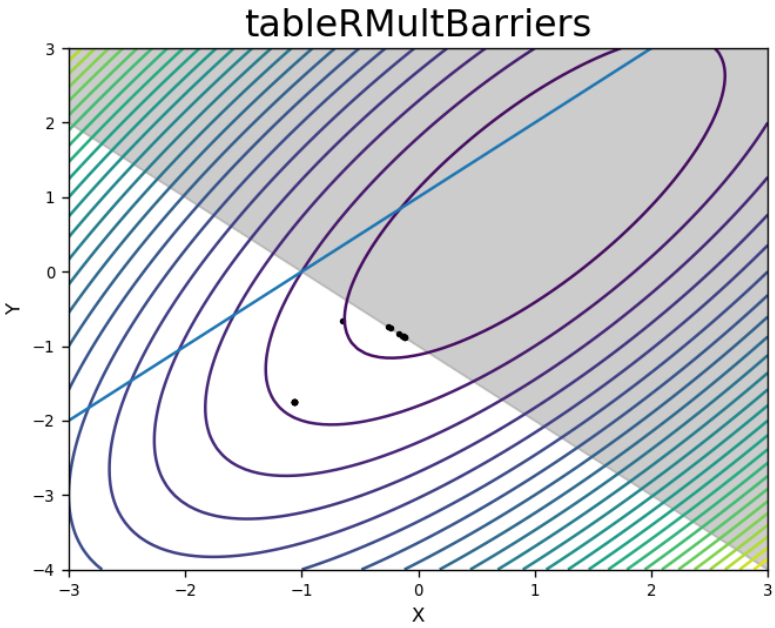
4.2. Барьерные функции

$$G_j[g_j(\bar{x})] = -\frac{1}{g_j(\bar{x})}$$

$rMult$	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
2	58	6,419	$3.47 \cdot 10^{-17}$	$(-2.596589e-01, -7.403411e-01)^T$	5.68
3	18	2,356	$2.58 \cdot 10^{-8}$	$(-6.540634e-01, -6.540635e-01)^T$	8.21
4	43	5,321	$1.29 \cdot 10^{-25}$	$(-2.403403e-01, -7.596596e-01)^T$	5.69
5	36	4,271	$6.87 \cdot 10^{-25}$	$(-1.627271e-01, -8.372728e-01)^T$	5.88
6	30	3,561	$4.52 \cdot 10^{-23}$	$(-1.415051e-01, -8.584949e-01)^T$	5.97
7	27	3,196	$1.52 \cdot 10^{-22}$	$(-1.290716e-01, -8.709284e-01)^T$	6.03
8	25	2,956	$2.65 \cdot 10^{-22}$	$(-1.216684e-01, -8.783316e-01)^T$	6.06
9	24	2,802	$1.25 \cdot 10^{-22}$	$(-1.170250e-01, -8.829749e-01)^T$	6.09
10	23	2,681	$1 \cdot 10^{-22}$	$(-1.139555e-01, -8.860444e-01)^T$	6.11

$$G_j[g_j(\bar{x})] = -ln\Big[-g_j(\bar{x})\Big]$$

$rMult$	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
2	1	180	5	$(-1.060879e+00, -1.750834e+00)^T$	14.65
3	1	180	3.33	$(-1.060879e+00, -1.750834e+00)^T$	14.65
4	1	180	2.5	$(-1.060879e+00, -1.750834e+00)^T$	14.65
5	1	180	2	$(-1.060879e+00, -1.750834e+00)^T$	14.65
6	1	180	1.67	$(-1.060879e+00, -1.750834e+00)^T$	14.65
7	1	180	1.43	$(-1.060879e+00, -1.750834e+00)^T$	14.65
8	1	180	1.25	$(-1.060879e+00, -1.750834e+00)^T$	14.65
9	1	180	1.11	$(-1.060879e+00, -1.750834e+00)^T$	14.65
10	1	180	1	$(-1.060879e+00, -1.750834e+00)^T$	14.65



4.3. Вывод

Для решения методом штрафной функции оптимальным методом является выбор первой функции с коэффициентом штрафа rMult = 2.

С ростом степени α в штрафной функции растёт число итераций и вычислений целевой функции. Для $\alpha >= 2$ оптимален rMult = 3.

Для решения методом барьерной функции оптимален выбор первой функции с параметром rMult = 2 для получения наиболее точного результата. Можно использовать метод с параметром rMult = 10. Так мы получим приближенное решение с вдвое меньшими вычислительными затратами.

5. Зависимость скорости сходимости метода от начального значения штрафа

5.1. Штрафные функции

$$G_j[g_j(\bar{x})] = \frac{1}{2}\{g_j(\bar{x}) + |g_j(\bar{x})|\}$$

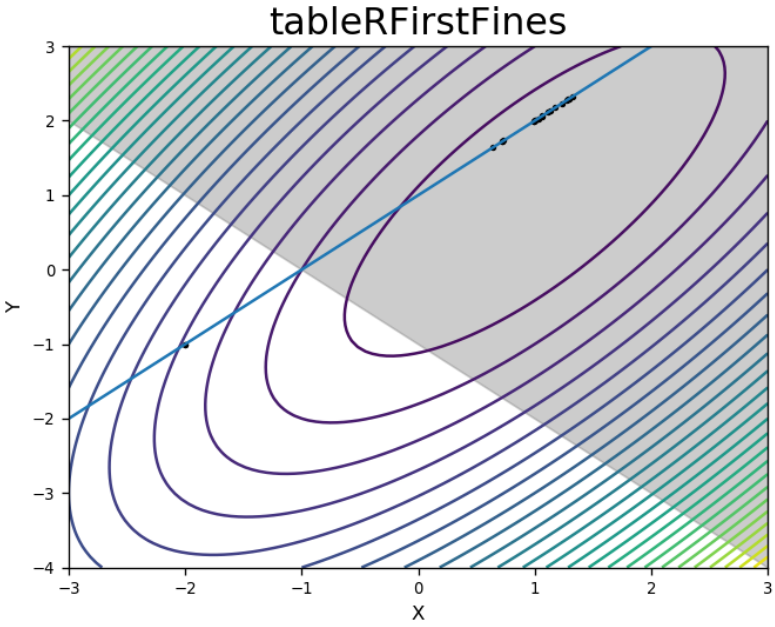
r_0	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
1	6	845	64	$(1.288387e+00, 2.288387e+00)^T$	4.25
2	5	672	64	$(1.132032e+00, 2.132032e+00)^T$	4.05
4	4	496	64	$(6.432039e-01, 1.643204e+00)^T$	4.38
8	5	500	256	$(1.000000e+00, 2.000000e+00)^T$	4
16	5	501	512	$(1.000000e+00, 2.000000e+00)^T$	4
32	1	61	64	$(-2.000000e+00, -1.000000e+00)^T$	31
64	1	61	128	$(-2.000000e+00, -1.000000e+00)^T$	31

$$G_j[g_j(\bar{x})] = \left[\frac{1}{2}\{g_j(\bar{x}) + |g_j(\bar{x})|\}\right]^2$$

r_0	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
1	31	3,561	$2.15 \cdot 10^9$	$(1.320299e+00, 2.320299e+00)^T$	4.31
2	30	3,404	$2.15 \cdot 10^9$	$(1.292411e+00, 2.292411e+00)^T$	4.26
4	29	3,253	$2.15 \cdot 10^9$	$(1.235510e+00, 2.235510e+00)^T$	4.17
8	29	3,180	$4.29 \cdot 10^9$	$(1.173672e+00, 2.173672e+00)^T$	4.09
16	28	3,047	$4.29 \cdot 10^9$	$(1.117858e+00, 2.117858e+00)^T$	4.04
32	27	2,920	$4.29 \cdot 10^9$	$(1.065477e+00, 2.065477e+00)^T$	4.01
64	27	2,791	$8.59 \cdot 10^9$	$(1.027760e+00, 2.027760e+00)^T$	4

$$G_j[g_j(\bar{x})] = \left[\frac{1}{2}\{g_j(\bar{x}) + |g_j(\bar{x})|\}\right]^\alpha$$,если α чётное

r_0	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
1	173	17,223	$1.2 \cdot 10^{52}$	$(1.273767e+00, 2.273767e+00)^T$	4.22
2	256	22,123	$2.32 \cdot 10^{77}$	$(7.244551e-01, 1.724455e+00)^T$	4.23
4	255	21,970	$2.32 \cdot 10^{77}$	$(7.242674e-01, 1.724267e+00)^T$	4.23
8	255	21,890	$4.63 \cdot 10^{77}$	$(7.345661e-01, 1.734566e+00)^T$	4.21
16	245	21,489	$9.05 \cdot 10^{74}$	$(1.059100e+00, 2.059100e+00)^T$	4.01
32	245	20,472	$1.81 \cdot 10^{75}$	$(9.954348e-01, 1.995435e+00)^T$	4
64	243	20,178	$9.05 \cdot 10^{74}$	$(9.956446e-01, 1.995645e+00)^T$	4



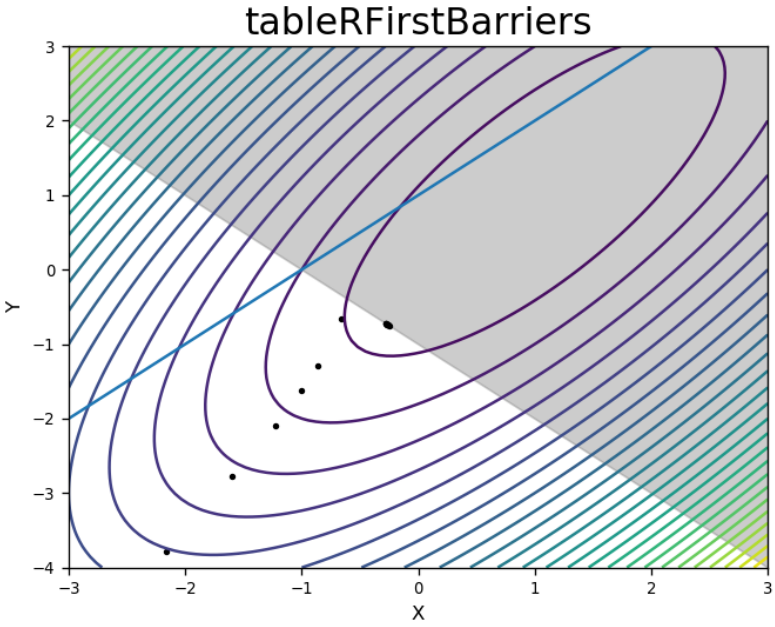
5.2. Барьерные функции

$$G_j[g_j(\bar{x})] = -\frac{1}{g_j(\bar{x})}$$

r_0	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
1	54	5,586	$5.55 \cdot 10^{-17}$	$(-2.764539e-01, -7.235461e-01)^T$	5.69
2	55	5,735	$5.55 \cdot 10^{-17}$	$(-2.765279e-01, -7.234721e-01)^T$	5.69
4	56	5,902	$5.55 \cdot 10^{-17}$	$(-2.765339e-01, -7.234661e-01)^T$	5.69
8	26	3,390	$1.19 \cdot 10^{-7}$	$(-6.597586e-01, -6.597589e-01)^T$	8.26
16	58	6,277	$5.55 \cdot 10^{-17}$	$(-2.461733e-01, -7.538267e-01)^T$	5.69
32	57	6,017	$2.22 \cdot 10^{-16}$	$(-2.651979e-01, -7.348021e-01)^T$	5.68
64	58	6,079	$2.22 \cdot 10^{-16}$	$(-2.628074e-01, -7.371927e-01)^T$	5.68

$$G_j[g_j(\bar{x})] = -ln\Big[-g_j(\bar{x})\Big]$$

r_0	Итераций	Вычислений f	r	$\mathbf{x}$	$f(\mathbf{x})$
1	28	3,668	$3.73 \cdot 10^{-9}$	$(-2.631574e-01, -7.368426e-01)^T$	5.68
2	28	3,647	$7.45 \cdot 10^{-9}$	$(-2.631686e-01, -7.368314e-01)^T$	5.68
4	1	183	2	$(-8.672954e-01, -1.296810e+00)^T$	11.2
8	1	180	4	$(-1.000000e+00, -1.618034e+00)^T$	13.53
16	1	177	8	$(-1.227380e+00, -2.090309e+00)^T$	17.86
32	1	185	16	$(-1.595075e+00, -2.780173e+00)^T$	25.82
64	1	189	32	$(-2.160645e+00, -3.779900e+00)^T$	40.46



5.3. Вывод

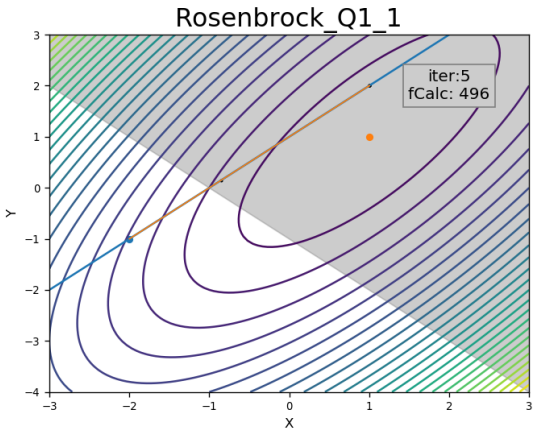
Для решения методом штрафной функции оптимальным методом является выбор первой функции с начальным значением штрафа $\in [8, 16]$.

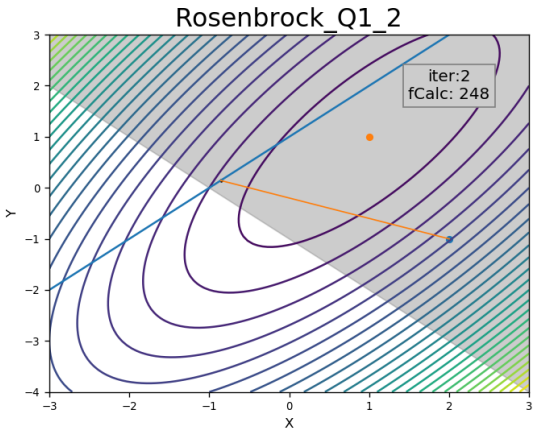
С ростом степени α в штрафной функции растёт число итераций и вычислений целевой функции. Для $\alpha >= 2$ оптимален $r_0 \in [32, 64]$.

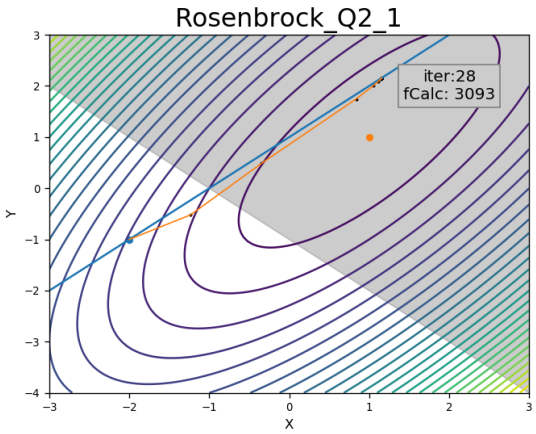
Для решения методом барьерной функции оптимален выбор первой функции с любым параметром или второй с $r_0 \in [1, 2]$

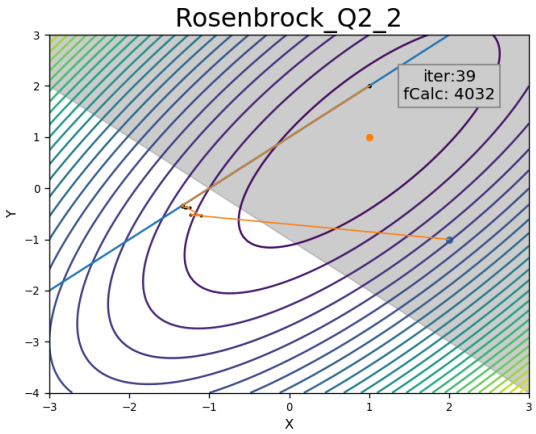
Для второй (логарифмической) функции метода барьеров рост начального значения штрафа ведёт к тому, что метод в первую очередь стремится не найти локальный минимум, а стремится отойти как можно дальше от границы области.

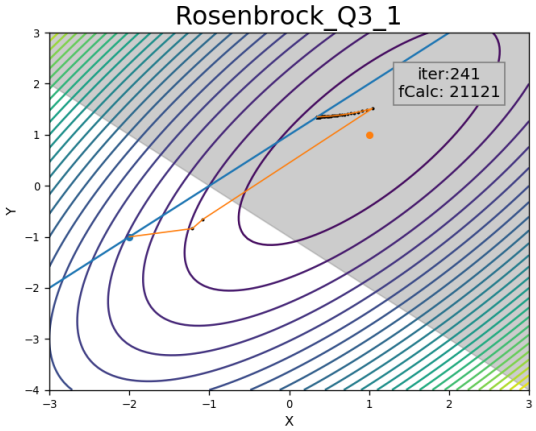
6. Зависимость скорости сходимости метода от начального приближения

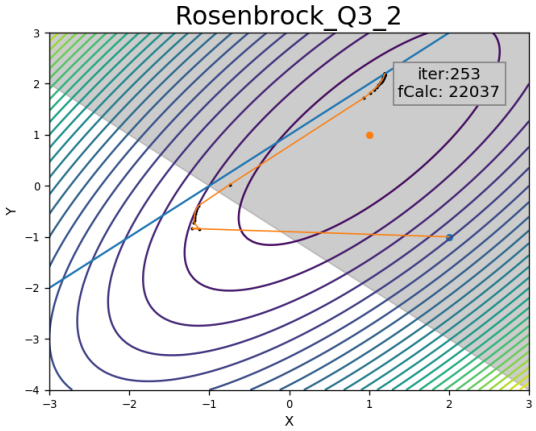


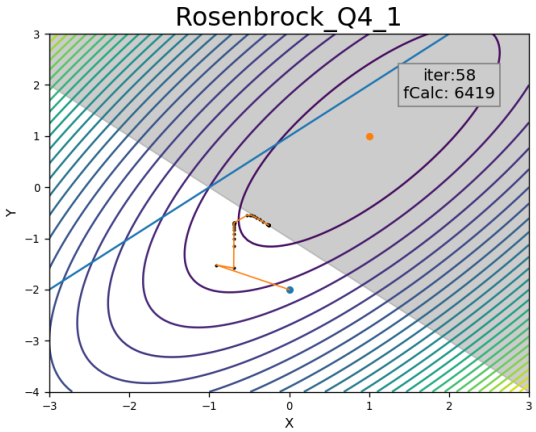


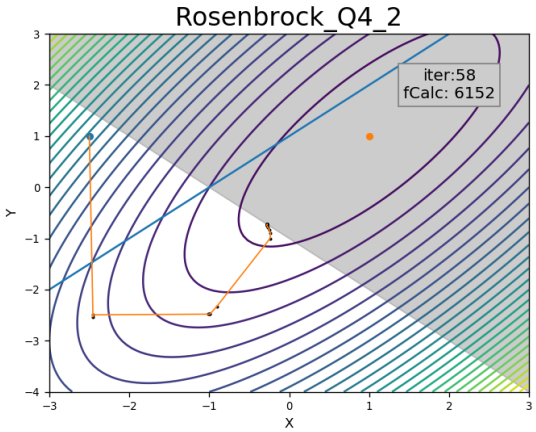

Rosenbrock_Q2_1
iter:28
fCalc: 3093
X
Y

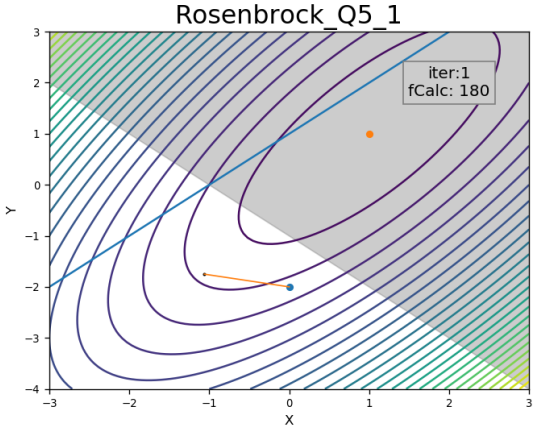

Rosenbrock_Q2_2
iter:39
fCalc: 4032
X
Y

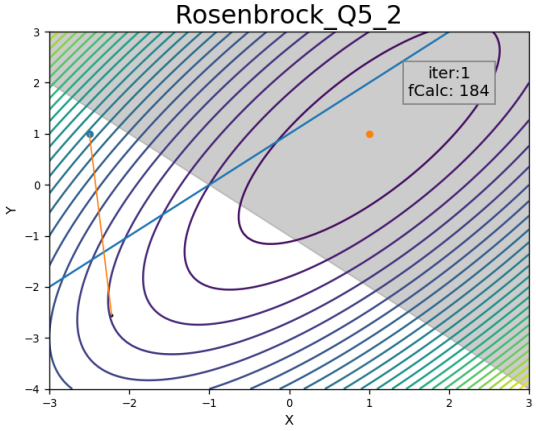

Rosenbrock_Q3_1
iter:241
fCalc: 21121
X
Y


Rosenbrock_Q3_2
iter:253
fCalc: 22037
X
Y


Rosenbrock_Q4_1
iter:58
fCalc: 6419
X
Y


Rosenbrock_Q4_2
iter:58
fCalc: 6152
X
Y


Rosenbrock_Q5_1
iter:1
fCalc: 180
X
Y


Rosenbrock_Q5_2
iter:1
fCalc: 184
X
Y

6.1. Вывод

Начальное приближение никак не влияет на сходимость метода.

7. 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>

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++)
        result[i] = -result[i];
    return result;
}


// Умножение матрицы на вектор
inline vector1D operator*(const matrix2D& a, const vector1D& b) {
    vector1D result = { 0.0, 0.0 };
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < a.size(); j++)
            result[i] += a[i][j] * b[j];
    return result;
}


// Умножение на константу
inline vector1D operator*(const vector1D& a, double b) {
    vector1D result = a;
    for (int i = 0; i < result.size(); i++)
        result[i] *= b;
    return result;
}
```

```
113
114
115 // Умножение на константу
116 inline vector1D operator*(double b, const vector1D& a) {
117     return operator*(a, b);
118 }
119
120
121 // Деление на константу
122 inline vector1D operator/(const vector1D& a, double b) {
123     vector1D result = a;
124     for (int i = 0; i < result.size(); i++)
125         result[i] /= b;
126     return result;
127 }
128
129
130 // Деление на константу
131 inline vector1D operator/(double b, const vector1D& a) {
132     return operator/(a, b);
133 }
134
135
136 // Скалярное произведение
137 inline real operator*(const vector1D& a, const vector1D& b) {
138 #ifdef _DEBUG
139     if (a.size() != b.size())
140         throw std::exception();
141 #endif
142     real sum = 0;
143     for (int i = 0; i < a.size(); i++)
144         sum += a[i] * b[i];
145     return sum;
146 }
147
148
149 // Потоковый вывод вектора
150 inline std::ostream& operator<<(std::ostream& out, const vector1D& v) {
151
152     for (int i = 0; i < v.size() - 1; ++i)
153         out << v[i] << " ";
154     out << v.back();
155     return out;
156 }
157
158 // Потоковый вывод вектора для TeX
159 inline void printTeXVector(std::ofstream &fout, const vector1D &v) {
160     fout << "$(";
161     for (int i = 0; i < v.size() - 1; ++i)
162         fout << v[i] << ", ";
163     fout << v.back() << ")^T$";
164 }
165
166 // Потоковый вывод матрицы
167 inline std::ostream& operator<<(std::ostream& out, const matrix2D& v) {
168     for (int i = 0; i < v.size() - 1; ++i)
169         out << v[i] << "  ";
170     out << v.back();
171     return out;
172 }
```

```
173
174 // Евклидова норма
175 real calcNormE(const vector1D &x) {
176     return sqrt(x*x);
177 }
178
179
180 // Определитель матрицы
181 real det(const matrix2D &m) {
182     return m[0][0] * m[1][1] - m[0][1] * m[1][0];
183 }
```

main.cpp

```
1  #include "head.h"
2  int fCalcCount, gCalcCount, alpha, rMult;
3  real r;
4
5  //-------------------------------------------------------------------------------
6  struct methodResult
7  {
8      real E;
9      int iterationsCount;
10     int fCalcCount;
11     int r0;
12     vector1D x0, x, xPrev, S, gradf;
13     matrix2D A;
14     real fx, fxPrev, lambda;
15     void printResultE(std::ofstream &fout);
16     void printResultR(std::ofstream &fout);
17     void printResultRFirst(std::ofstream &fout);
18 };
19
20
21 // Вывод результатов в поток
22 void methodResult::printResultE(std::ofstream &fout) {
23     fout << E << "\t"
24         << iterationsCount << "\t"
25         << fCalcCount << "\t"
26         << r << "\t";
27     printTeXVector(fout, x);
28     fout << "\t" << fx << endl;
29 }
30
31
32 // Вывод результатов в поток
33 void methodResult::printResultR(std::ofstream &fout) {
34     fout << rMult << "\t"
35         << iterationsCount << "\t"
36         << fCalcCount << "\t"
37         << r << "\t";
38     printTeXVector(fout, x);
39     fout << "\t" << fx << endl;
40 }
41
42
43 // Вывод результатов в поток
44 void methodResult::printResultRFirst(std::ofstream &fout) {
45     fout << r0 << "\t"
46         << iterationsCount << "\t"
47         << fCalcCount << "\t"
48         << r << "\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++;
    return 4 * pow((x[1] - x[0]), 2) + 3 * pow((x[0] - 1), 2);
}



// Ограничение области б
inline real gFine(const vector1D &x) {
    // y = x + 1
    gCalcCount++;
    return  fabs(x[1] - x[0] - 1) /*+ fabs(x[0] + x[1] + 1)*/;
}

// Ограничение области а
inline real gBarrier(const vector1D &x) {
    // x + y <= -1
    gCalcCount++;
    return x[0] + x[1] + 1;
}



//-------------------------------------------------------------------------------------------
// Штрафная функция G
inline real G1(const vector1D &x) {
    if (gFine(x) > 0)
        return 0.5*(gFine(x) + fabs(gFine(x)));
    else
        return 0;
}
inline real G2(const vector1D &x) {
    if (gFine(x) > 0)
        return pow(0.5*(gFine(x) + fabs(gFine(x))), 2);
    else
        return 0;
}
inline real G3(const vector1D &x) {
    if (gFine(x) > 0)
        return pow(0.5*(gFine(x) + fabs(gFine(x))), alpha);
    else
        return 0;
}
inline real G4(const vector1D &x) {
    if (gBarrier(x) <= 0)
        return -1.0 / gBarrier(x);
    else
        return std::numeric_limits<double>::infinity();
}
inline real G5(const vector1D &x) {
```

```
    if (gBarrier(x) <= 0)
        return -log(-gBarrier(x));
    else
        return std::numeric_limits<double>::infinity();
}


//-----------------------------------------------------------------------------
// Минимизируемая функция Q(x,y)
real Q1(const vector1D &x) { return f(x) + r * G1(x); }
real Q2(const vector1D &x) { return f(x) + r * G2(x); }
real Q3(const vector1D &x) { return f(x) + r * G3(x); }
real Q4(const vector1D &x) { return f(x) + r * G4(x); }
real Q5(const vector1D &x) { return f(x) + r * G5(x); }



// 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 nго– 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)> &fExact,
    const function<real(const vector1D &x)> &f, const function<real(const
    vector1D &x)> &G, const vector1D &x0, real r0, real E, const string &funcname
    , bool isFineNotBarier) {

    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;
    gCalcCount = 0;
    vector1D xPrev, B, x = x0;
    int maxiter = 1000;
    int iterationsCount = 0, count = 0;
    r = r0;
    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;

```

```
284
285         // Ортогонализация ГраммаШмидта
286         S[0] = A[0] / calcNormE(A[0]);
287         B = A[1] - S[1] * A[1] * S[1];
288         if (calcNormE(B) > E)
289             S[1] = B / calcNormE(B);
290         iterationsCount++;
291
292
293         fout << iterationsCount << "\t"
294             << fCalcCount << "\t"
295             << r << "\t"
296             << x << "\t"
297             << fExact(x) << endl;
298
299         steps << x << endl;
300
301         if (isFineNotBarier)
302             r *= rMult;
303         else
304             r /= rMult;
305
306     } while (abs(f(x) - f(xPrev)) > E && abs(calcNormE(x) - calcNormE(xPrev)) >
    E && iterationsCount < maxiter && r*G(x) > E);
307
308     fout.close();
309     steps.close();
310
311     result.E = E;
312     result.iterationsCount = iterationsCount;
313     result.fCalcCount = fCalcCount;
314     result.x0 = x0;
315     result.x = x;
316     result.fx = fExact(x);
317     result.r0 = r0;
318
319     return result;
320 }
321
322
323
324 //-----------------------------------------------------------------------------
325 //-----------------------------------------------------------------------------
326 // Исследования
327 //-----------------------------------------------------------------------------
328 //-----------------------------------------------------------------------------
329
330
331
332
333 // Зависимость скорости сходимости метода от заданной точности
334 void researchE()
335 {
336     cout << "Research E" << endl;
337     ofstream foutF("steps/tableEFines.txt");
338     ofstream foutB("steps/tableEBarriers.txt");
339     ofstream fout1("report/tableE1.txt");
340     ofstream fout2("report/tableE2.txt");
341     ofstream fout3("report/tableE3.txt");
342     ofstream fout4("report/tableE4.txt");
```

```
    ofstream fout5("report/tableE5.txt");
    fout1 << scientific << "E\titer\tfCalc\tr\tx\tfx" << endl;
    fout2 << scientific << "E\titer\tfCalc\tr\tx\tfx" << endl;
    fout3 << scientific << "E\titer\tfCalc\tr\tx\tfx" << endl;
    fout4 << scientific << "E\titer\tfCalc\tr\tx\tfx" << endl;
    fout5 << scientific << "E\titer\tfCalc\tr\tx\tfx" << endl;

    methodResult result;
    real r0 = 10;
    rMult = 2;
    alpha = 8;
    vector1D x0Fine = { -2, -1 };
    vector1D x0Barrier = { 0, -2 };

    for (double E = 1e-3; E >= 1e-7; E /= 10)
    {
        result = calcByRosenbrock(f, Q1, G1, x0Fine, r0, E, "Q1", true);
        result.printResultE(fout1);
        foutF << result.x << endl;
        result = calcByRosenbrock(f, Q2, G2, x0Fine, r0, E, "Q2", true);
        result.printResultE(fout2);
        foutF << result.x << endl;
        result = calcByRosenbrock(f, Q3, G3, x0Fine, r0, E, "Q3", true);
        result.printResultE(fout3);
        foutF << result.x << endl;

        result = calcByRosenbrock(f, Q4, G4, x0Barrier, r0, E, "Q4", false);
        result.printResultE(fout4);
        foutB << result.x << endl;
        result = calcByRosenbrock(f, Q5, G5, x0Barrier, r0, E, "Q5", false);
        result.printResultE(fout5);
        foutB << result.x << endl;
    }

    // Визуализация
    string runVisualisation = "python plot.py "
        + to_string(x0Fine[0]) + " " + to_string(x0Fine[1]) + " "
        + to_string(x0Barrier[0]) + " " + to_string(x0Barrier[1]) + " " +
    to_string(1);
    system(runVisualisation.c_str());

    foutF.close();
    foutB.close();
    fout1.close();
    fout2.close();
    fout3.close();
    fout4.close();
    fout5.close();
}



// Зависимость скорости сходимости метода от стратегии изменения коэффициента штрафа
void researchRMult()
{
    cout << "Research r^mult" << endl;
    ofstream foutF("steps/tableRMultFines.txt");
    ofstream foutB("steps/tableRMultBarriers.txt");
    ofstream fout1("report/tableRMult1.txt");
    ofstream fout2("report/tableRMult2.txt");
```

```
    ofstream fout3("report/tableRMult3.txt");
    ofstream fout4("report/tableRMult4.txt");
    ofstream fout5("report/tableRMult5.txt");
    fout1 << scientific << "rMult\titer\tfCalc\tr\tx\tfx" << endl;
    fout2 << scientific << "rMult\titer\tfCalc\tr\tx\tfx" << endl;
    fout3 << scientific << "rMult\titer\tfCalc\tr\tx\tfx" << endl;
    fout4 << scientific << "rMult\titer\tfCalc\tr\tx\tfx" << endl;
    fout5 << scientific << "rMult\titer\tfCalc\tr\tx\tfx" << endl;

    methodResult result;
    real E = 1e-12;
    real r0 = 10;
    alpha = 8;
    vector1D x0Fine = { -2, -1 };
    vector1D x0Barrier = { 0, -2 };
    for (size_t i = 2; i <= 10; i += 1)
    {
        rMult = i;
        result = calcByRosenbrock(f, Q1, G1, x0Fine, r0, E, "Q1", true);
        result.printResultR(fout1);
        foutF << result.x << endl;
        result = calcByRosenbrock(f, Q2, G2, x0Fine, r0, E, "Q2", true);
        result.printResultR(fout2);
        foutF << result.x << endl;
        result = calcByRosenbrock(f, Q3, G3, x0Fine, r0, E, "Q3", true);
        result.printResultR(fout3);
        foutF << result.x << endl;


        result = calcByRosenbrock(f, Q4, G4, x0Barrier, r0, E, "Q4", false);
        result.printResultR(fout4);
        foutB << result.x << endl;
        result = calcByRosenbrock(f, Q5, G5, x0Barrier, r0, E, "Q5", false);
        result.printResultR(fout5);
        foutB << result.x << endl;
    }

    // Визуализация
    string runVisualisation = "python plot.py "
        + to_string(x0Fine[0]) + " " + to_string(x0Fine[1]) + " "
        + to_string(x0Barrier[0]) + " " + to_string(x0Barrier[1]) + " " +
   to_string(2);
    system(runVisualisation.c_str());

    foutF.close();
    foutB.close();
    fout1.close();
    fout2.close();
    fout3.close();
    fout4.close();
    fout5.close();
}



// Зависимость скорости сходимости метода от стратегии изменения коэффициента штрафа
void researchRFirst()
{
    cout << "Research r_0" << endl;
    ofstream foutF("steps/tableRFirstFines.txt");
```

```
    ofstream foutB("steps/tableRFirstBarriers.txt");
    ofstream fout1("report/tableRFirst1.txt");
    ofstream fout2("report/tableRFirst2.txt");
    ofstream fout3("report/tableRFirst3.txt");
    ofstream fout4("report/tableRFirst4.txt");
    ofstream fout5("report/tableRFirst5.txt");
    fout1 << scientific << "rFirst\titer\tfCalc\tr\tx\tfx" << endl;
    fout2 << scientific << "rFirst\titer\tfCalc\tr\tx\tfx" << endl;
    fout3 << scientific << "rFirst\titer\tfCalc\tr\tx\tfx" << endl;
    fout4 << scientific << "rFirst\titer\tfCalc\tr\tx\tfx" << endl;
    fout5 << scientific << "rFirst\titer\tfCalc\tr\tx\tfx" << endl;

    methodResult result;
    real E = 1e-12;
    rMult = 2;
    alpha = 8;
    vector1D x0Fine = { -2, -1 };
    vector1D x0Barrier = { 0, -2 };

    for (real r0 = 1; r0 <= 64; r0 *= 2)
    {
        result = calcByRosenbrock(f, Q1, G1, x0Fine, r0, E, "Q1", true);
        result.printResultRFirst(fout1);
        foutF << result.x << endl;
        result = calcByRosenbrock(f, Q2, G2, x0Fine, r0, E, "Q2", true);
        result.printResultRFirst(fout2);
        foutF << result.x << endl;
        result = calcByRosenbrock(f, Q3, G3, x0Fine, r0, E, "Q3", true);
        result.printResultRFirst(fout3);
        foutF << result.x << endl;

        result = calcByRosenbrock(f, Q4, G4, x0Barrier, r0, E, "Q4", false);
        result.printResultRFirst(fout4);
        foutB << result.x << endl;
        result = calcByRosenbrock(f, Q5, G5, x0Barrier, r0, E, "Q5", false);
        result.printResultRFirst(fout5);
        foutB << result.x << endl;
    }


    // Визуализация
    string runVisualisation = "python plot.py "
        + to_string(x0Fine[0]) + " " + to_string(x0Fine[1]) + " "
        + to_string(x0Barrier[0]) + " " + to_string(x0Barrier[1]) + " " +
   to_string(3);
    system(runVisualisation.c_str());

    foutF.close();
    foutB.close();
    fout1.close();
    fout2.close();
    fout3.close();
    fout4.close();
    fout5.close();
}



// Зависимость скорости сходимости метода от начального приближения
void researchXFirst()
```

```
{
    cout << "Research x_0" << endl;
    methodResult result;
    real E = 1e-12;
    real r0 = 10;
    rMult = 2;
    alpha = 8;
    vector1D x0Fine = { -2, -1 };
    vector1D x0Barrier = { 0, -2 };

    ofstream fout1("steps/Rosenbrock_Q_1.txt");
    result = calcByRosenbrock(f, Q1, G1, x0Fine, r0, E, "Q1_1", true);
    fout1 << result.iterationsCount << " " << result.fCalcCount << endl;
    result = calcByRosenbrock(f, Q2, G2, x0Fine, r0, E, "Q2_1", true);
    fout1 << result.iterationsCount << " " << result.fCalcCount << endl;
    result = calcByRosenbrock(f, Q3, G3, x0Fine, r0, E, "Q3_1", true);
    fout1 << result.iterationsCount << " " << result.fCalcCount << endl;
    result = calcByRosenbrock(f, Q4, G4, x0Barrier, r0, E, "Q4_1", false);
    fout1 << result.iterationsCount << " " << result.fCalcCount << endl;
    result = calcByRosenbrock(f, Q5, G5, x0Barrier, r0, E, "Q5_1", false);
    fout1 << result.iterationsCount << " " << result.fCalcCount << endl;

    // Визуализация
    string runVisualisation = "python plot.py "
        + to_string(x0Fine[0]) + " " + to_string(x0Fine[1]) + " "
        + to_string(x0Barrier[0]) + " " + to_string(x0Barrier[1]) + " " +
   to_string(4);
    system(runVisualisation.c_str());


    ofstream fout2("steps/Rosenbrock_Q_2.txt");
    x0Fine = { 2, -1 };
    x0Barrier = { -2.5, 1 };
    result = calcByRosenbrock(f, Q1, G1, x0Fine, r0, E, "Q1_2", true);
    fout2 << result.iterationsCount << " " << result.fCalcCount << endl;
    result = calcByRosenbrock(f, Q2, G2, x0Fine, r0, E, "Q2_2", true);
    fout2 << result.iterationsCount << " " << result.fCalcCount << endl;
    result = calcByRosenbrock(f, Q3, G3, x0Fine, r0, E, "Q3_2", true);
    fout2 << result.iterationsCount << " " << result.fCalcCount << endl;
    result = calcByRosenbrock(f, Q4, G4, x0Barrier, r0, E, "Q4_2", false);
    fout2 << result.iterationsCount << " " << result.fCalcCount << endl;
    result = calcByRosenbrock(f, Q5, G5, x0Barrier, r0, E, "Q5_2", false);
    fout2 << result.iterationsCount << " " << result.fCalcCount << endl;

    // Визуализация
    runVisualisation = "python plot.py "
        + to_string(x0Fine[0]) + " " + to_string(x0Fine[1]) + " "
        + to_string(x0Barrier[0]) + " " + to_string(x0Barrier[1]) + " " +
   to_string(5);
    system(runVisualisation.c_str());
    fout2.close();
}



void main()
{
    researchE();
    researchRMult();
    researchRFirst();
```

```
    researchXFirst();
}
```

plot.py

```
import pylab
import numpy
import sys
import matplotlib.pyplot as plt
import matplotlib.lines as lines
import matplotlib.cm as cm

DPI = 120

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])


# Целевая функция
def f(x, y):
    return 4*(y-x)**2 + 3*(x-1)**2


# Рассчёт значений целевой функции на сетке
def buildIsoLines(f):
    x = numpy.linspace (-3, 3, 100)
    y = numpy.linspace (-4, 3, 100)
    xgrid, ygrid = numpy.meshgrid(x, y)
    zgrid = f(xgrid, ygrid)
```

```
    return xgrid, ygrid, zgrid


# Закраска областей, где накладывается штраф или барьер уходит в бесконечность
def drawFines():
    x1 = numpy.linspace(-3, 3, 100)
    y1 = -1 - x1
    yBorder = 3
    plt.fill_between(x1,y1,yBorder, color='grey', alpha=0.4)
    x2 = numpy.linspace(-3, 2, 100)
    y2 = x2 + 1
    plt.plot(x2, y2)


# Отрисовка сходимости метода
def drawMethodConvergence(name, f, index, x0, y0, xExp, yExp):
    global text
    inputSteps(name + '.txt')
    drawFines()
    x, y, z = buildIsoLines(f)
    cs = pylab.contour(x, y, z, 25)
    plt.plot(xList, yList, linewidth=1)
    for i in range(len(xList)):
        plt.scatter(xList[i], yList[i], s=2, color='black')
    plt.title(name, fontsize=19)
    plt.xlabel('X', fontsize=10)
    plt.ylabel('Y', fontsize=10)
    plt.tick_params(axis='both', labelsize=8)
    plt.scatter(x0, y0, s=20)
    plt.scatter(xExp, yExp, s=20)
    plt.text(2, 2, text[index], size=12,
         ha="center", va="center",
         bbox=dict(boxstyle="square",
                   ec=(.5, .5, .5),
                   fc=(.8, .8, 0.8),
                   )
         )
    plt.savefig(folder + name + '.png', dpi=DPI)
    plt.clf()


# Отрисовка точек в исследовании
def drawPointsAtResearch(name, f):
    inputSteps(name + '.txt')
    drawFines()
    x, y, z = buildIsoLines(f)
    cs = pylab.contour(x, y, z, 25)
    plt.title(name, fontsize=19)
    plt.xlabel('X', fontsize=10)
    plt.ylabel('Y', fontsize=10)
    plt.tick_params(axis='both', labelsize=8)
    for i in range(len(xList)):
        plt.scatter(xList[i], yList[i], s=5, color='black')
    plt.savefig(folder + name + '.png', dpi=DPI)
    plt.clf()



if __name__ == '__main__':
    # парсим начальное приближение
```

```
    params = sys.argv[1:6]
    x0Fine = float(params[0])
    y0Fine = float(params[1])
    x0Barrier = float(params[2])
    y0Barrier = float(params[3])
    mode = int(params[4])
    xExp = 1
    yExp = 1


    if mode == 1:
        drawPointsAtResearch('tableEFines', f)
        drawPointsAtResearch('tableEBarriers', f)

    if mode == 2:
        drawPointsAtResearch('tableRMultFines', f)
        drawPointsAtResearch('tableRMultBarriers', f)

    if mode == 3:
        drawPointsAtResearch('tableRFirstFines', f)
        drawPointsAtResearch('tableRFirstBarriers', f)

    if mode == 4:
        inputMethodResults('Rosenbrock_Q_1.txt')
        drawMethodConvergence('Rosenbrock_Q1_1', f, 0, x0Fine, y0Fine, xExp,
    yExp)
        drawMethodConvergence('Rosenbrock_Q2_1', f, 1, x0Fine, y0Fine, xExp,
    yExp)
        drawMethodConvergence('Rosenbrock_Q3_1', f, 2, x0Fine, y0Fine, xExp,
    yExp)
        drawMethodConvergence('Rosenbrock_Q4_1', f, 3, x0Barrier, y0Barrier,
    xExp, yExp)
        drawMethodConvergence('Rosenbrock_Q5_1', f, 4, x0Barrier, y0Barrier,
    xExp, yExp)

    if mode == 5:
        inputMethodResults('Rosenbrock_Q_2.txt')
        drawMethodConvergence('Rosenbrock_Q1_2', f, 0, x0Fine, y0Fine, xExp,
    yExp)
        drawMethodConvergence('Rosenbrock_Q2_2', f, 1, x0Fine, y0Fine, xExp,
    yExp)
        drawMethodConvergence('Rosenbrock_Q3_2', f, 2, x0Fine, y0Fine, xExp,
    yExp)
        drawMethodConvergence('Rosenbrock_Q4_2', f, 3, x0Barrier, y0Barrier,
    xExp, yExp)
        drawMethodConvergence('Rosenbrock_Q5_2', f, 4, x0Barrier, y0Barrier,
    xExp, yExp)
```