Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра ТПИ

МЕТОДЫ ОПТИМИЗАЦИИ

Лабораторная работа № 3

# Решение нелинейных начально-краевых задач

Факультет: ПМИ Преподаватели:

Лемешко Борис Юрьевич

Чимитова Екатерина Владимировна

Группа: ПМ-81

Студенты: Ефремов А.

Ртищева К.

Бортникова А.

Бригада: 2

Новосибирск

2021

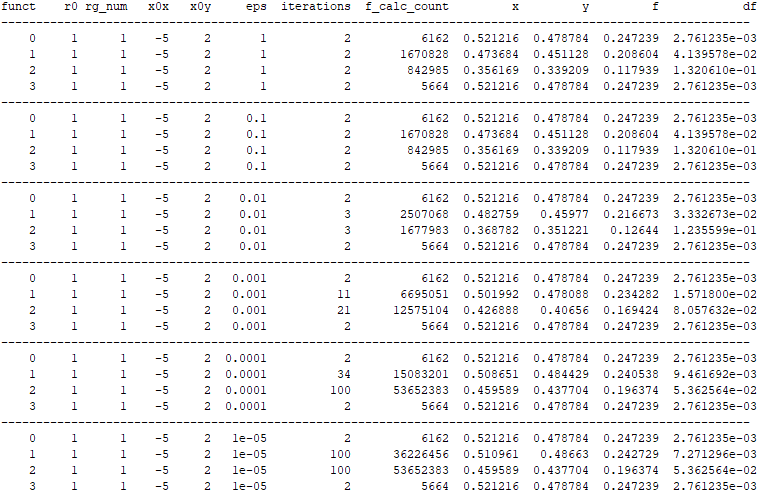
1. **Цель работы**

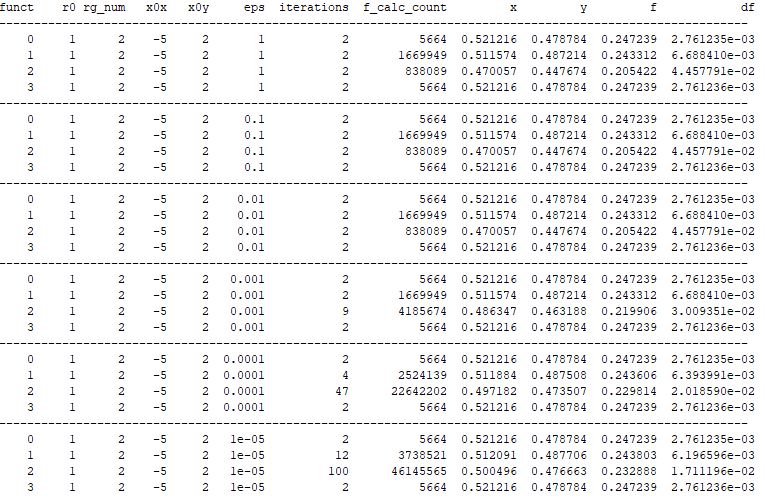
Ознакомиться с методами штрафных функций при решении задач нелинейного программирования. Изучить типы штрафных и барьерных функций, их особенности, способы и области применения, влияние штрафных функций на сходимость алгоритмов, зависимость точности решения задачи нелинейного программирования от величины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

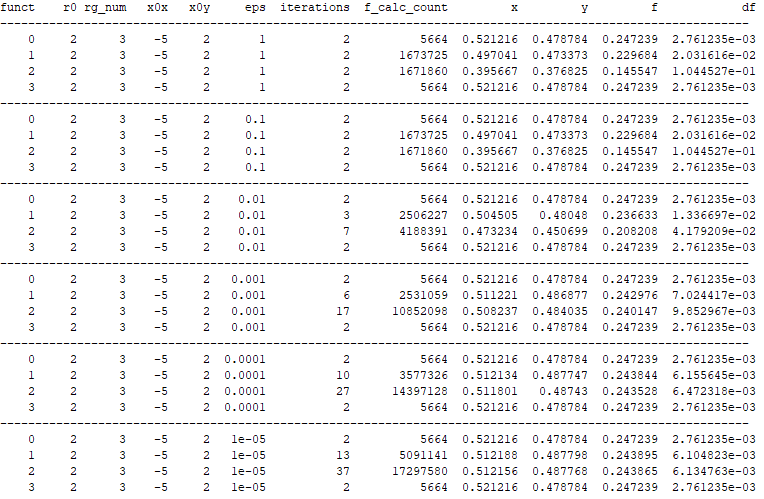
1. **Задание**

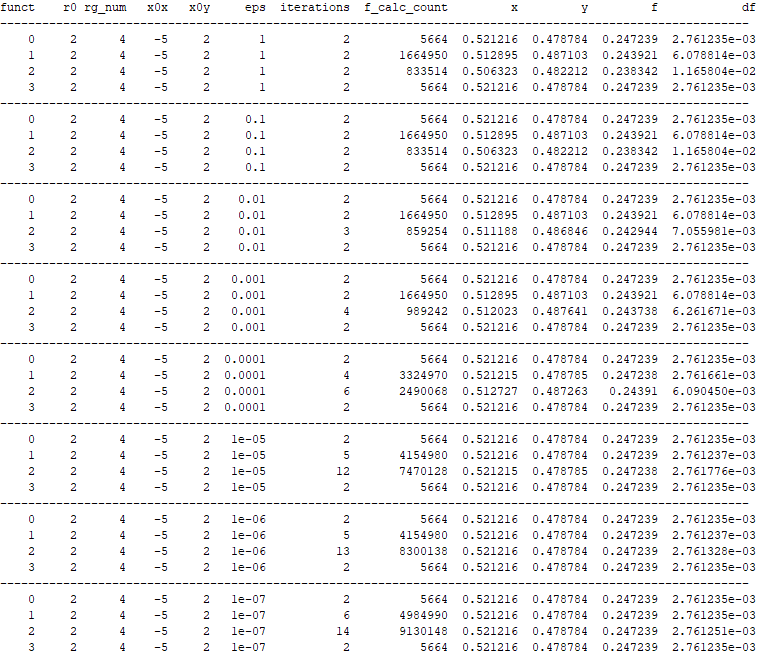
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Применяя методы поиска минимума 0-го порядка, реализовать программу для решения задачи нелинейного программирования с использованием **метода штрафных функций**.  Исследовать сходимость **метода штрафных функций** в зависимости от:   * выбора штрафных 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 задача а)** с использованием **метода барьерных функций**.  Исследовать сходимость **метода барьерных функций** **(только задача а)** в зависимости от:   * выбора барьерных функций, * начальной величины коэффициента штрафа, * стратегии изменения коэффициента штрафа, * начальной точки, * задаваемой точности.   Сформулировать выводы.   |  |  | | --- | --- | | Первая задача (а) | Вторая задача (б) | |  |  | |

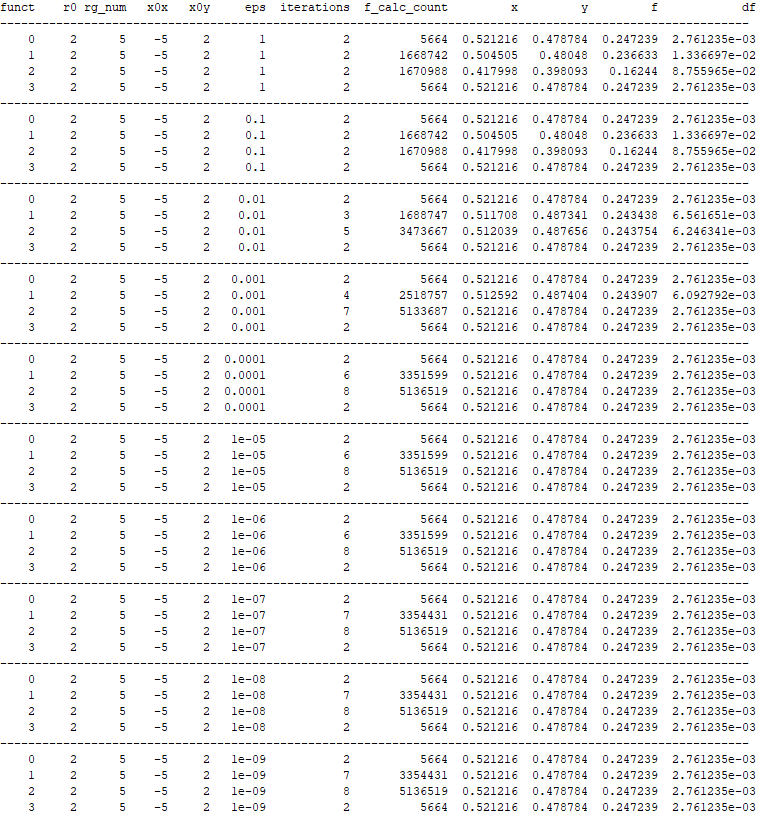
1. **Таблицы с исследованиями и выводы для метода штрафных функций**
   1. **Для функции G**

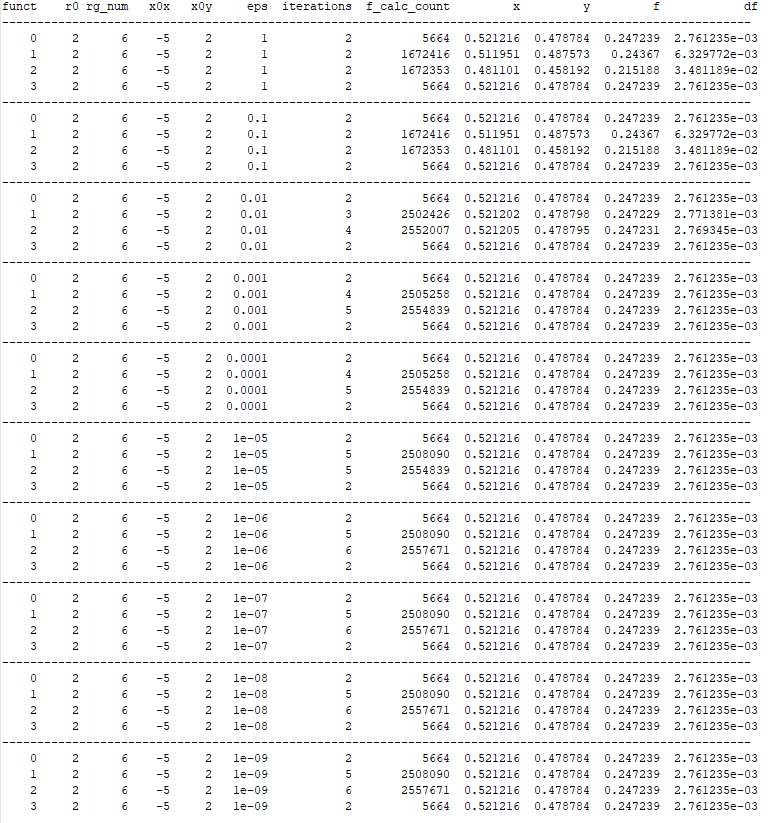








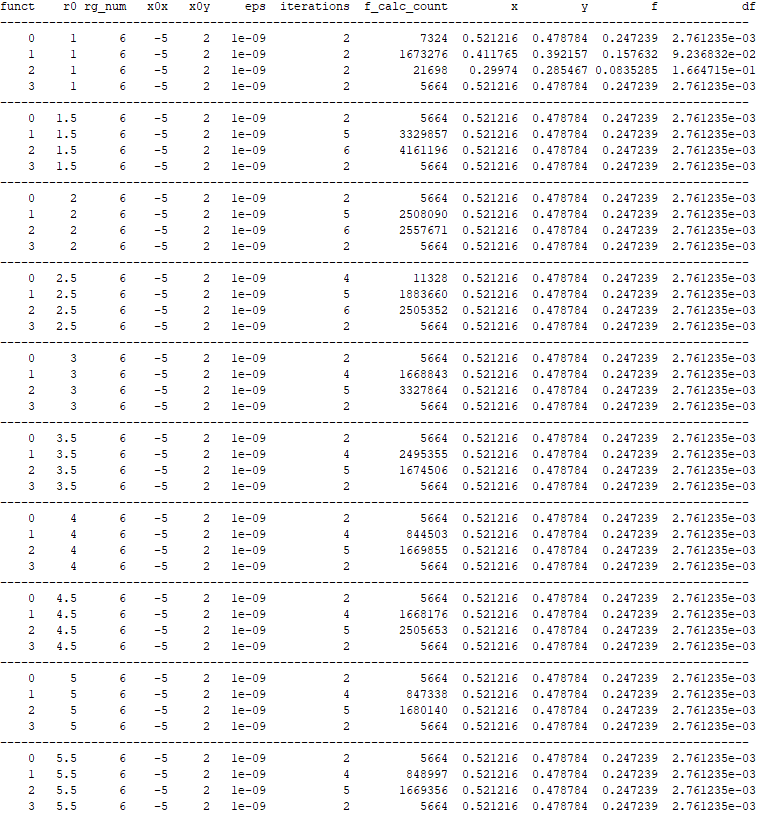




Лучше всего себя показали функции , – они показывают примерно одинаковую и лучшую среди всех функций точность и скорость схождения. Предпочтение стоить отдать , так как в присутствует вычислительно затратная операция деления.

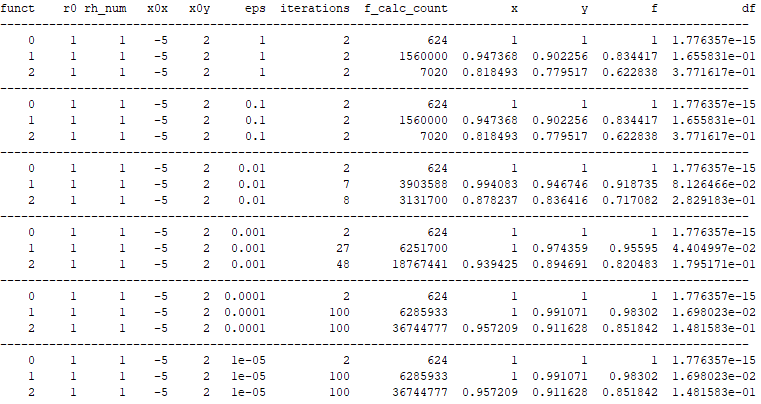
Лучше всего себя показала стратегия изменения коэффициента штрафа - как минимум с точки зрения количества итераций метода штрафных функций.

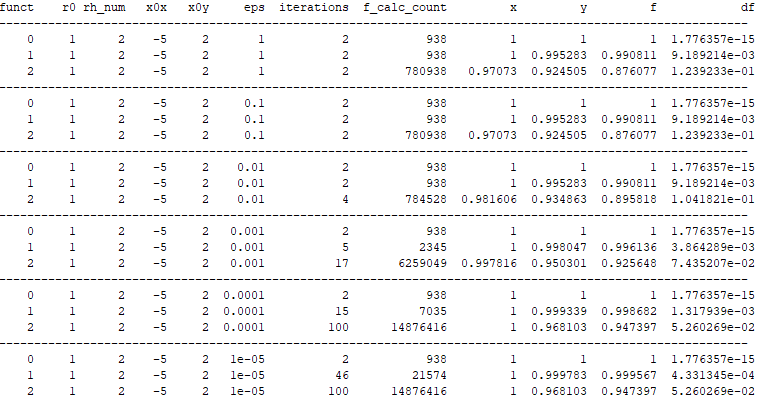
Проведем дополнительные исследования с помощью с варьированием начального коэффициента штрафа.

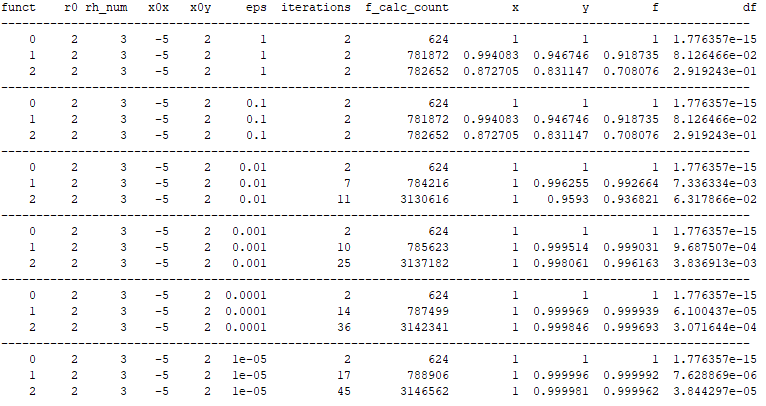


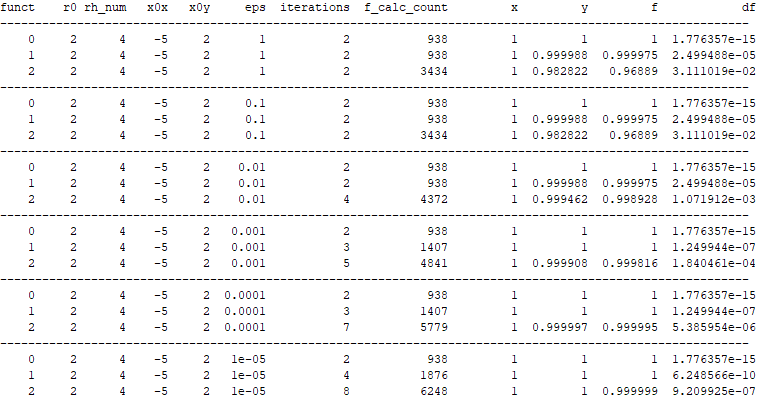
**Вывод:** для данной задачи лучшей функцией является , оптимальной стратегией выбора коэффициента штрафа является степенная функция , оптимальный начальный коэффициент штрафа для этой функции выбирается достаточно свободно в диапазоне от **1.5**.

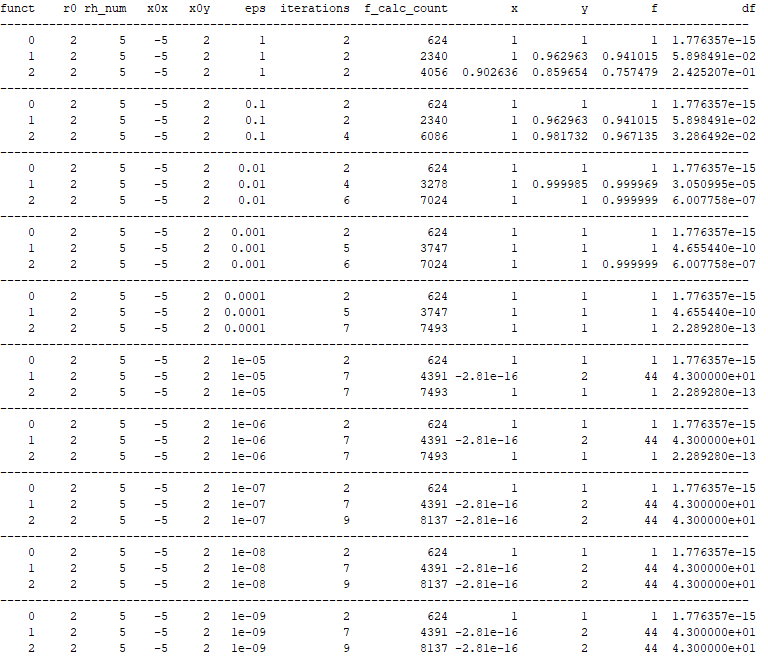
* 1. **Для функции H**

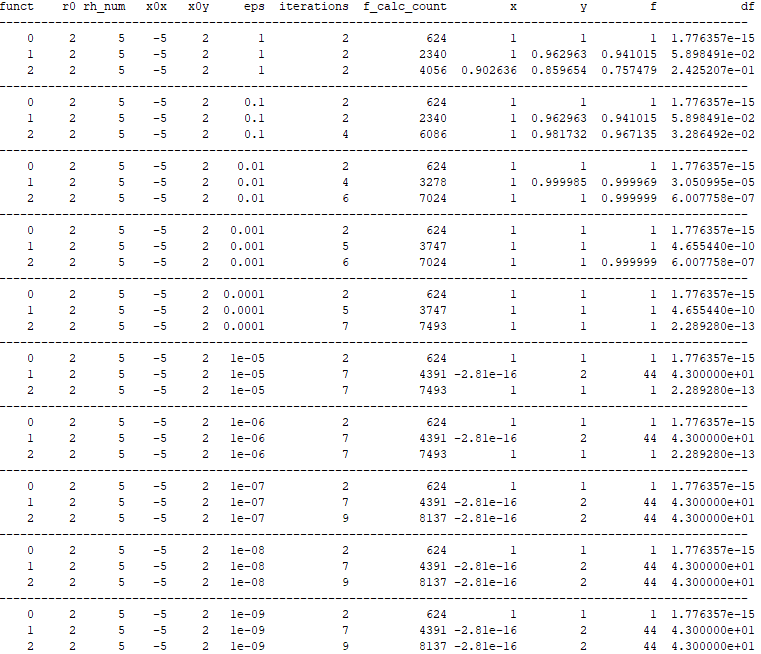








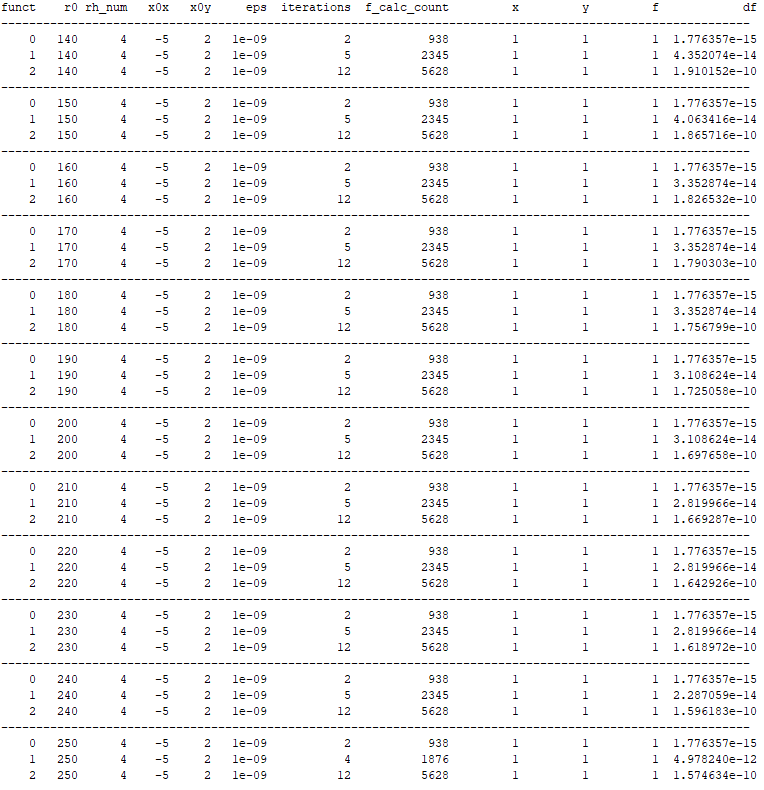




Лучше всего себя показала функция - она ищет правильный экстремум за минимальное и константное число итераций, причем с максимальной точностью. Также стоит отметить, что функция также показывает хороший результат, точность которого можно контролировать параметром **eps**.

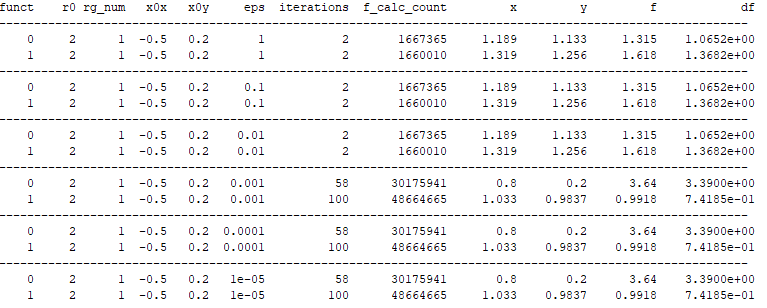
Лучше всего себя показала стратегия изменения коэффициента штрафа . Использование степенных функций нежелательно, так как приводит к нахождению ложной точки минимума.

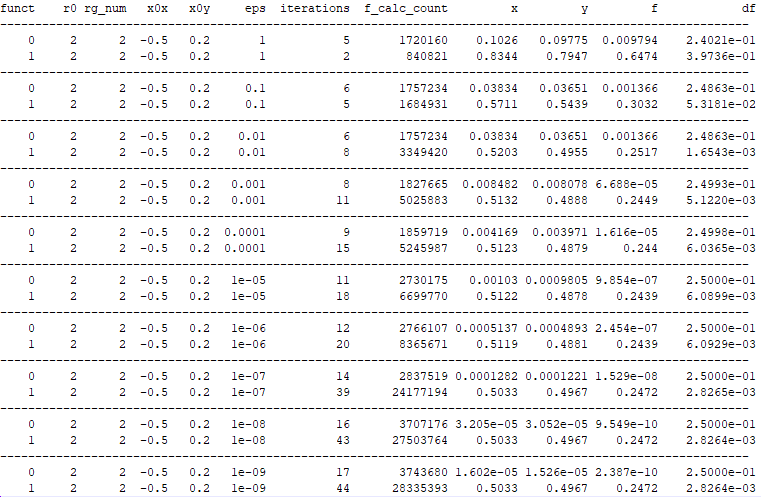
Проведем дополнительные исследования с помощью с варьированием начального коэффициента штрафа.

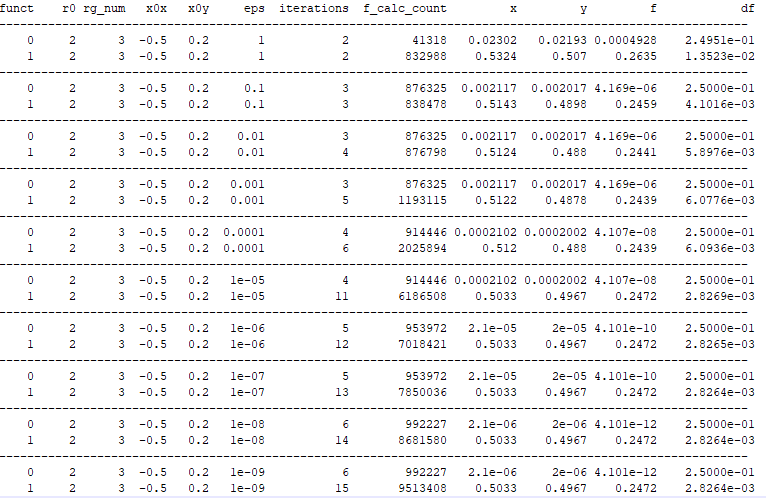


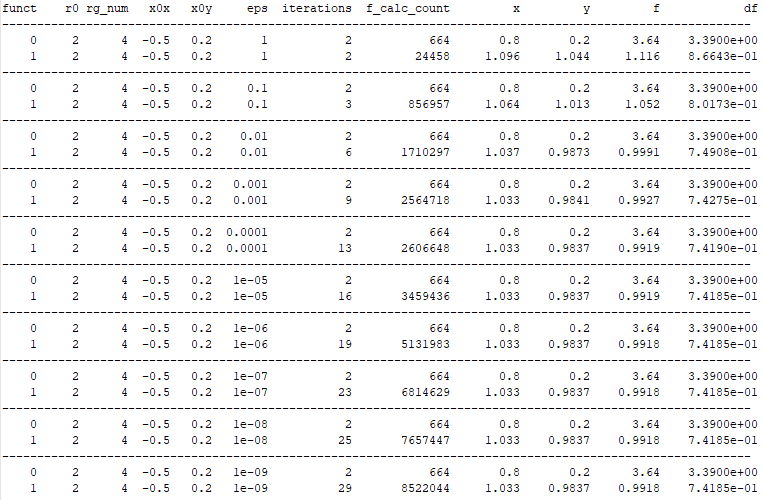
**Вывод:** для данной задачи лучшими функциями являются и , оптимальной стратегией выбора коэффициента штрафа является мультипликативная функция , оптимальный начальный коэффициент штрафа для этой функции примерно равен **240**.

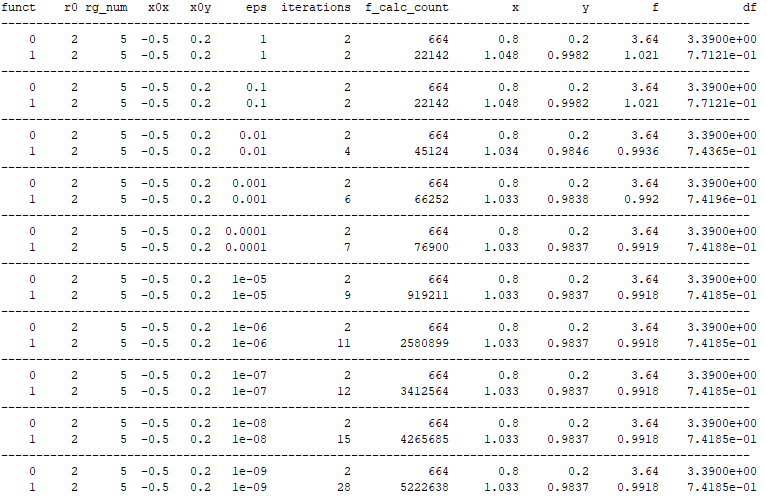
1. **Таблицы с исследованиями и выводы для метода барьерных функций**







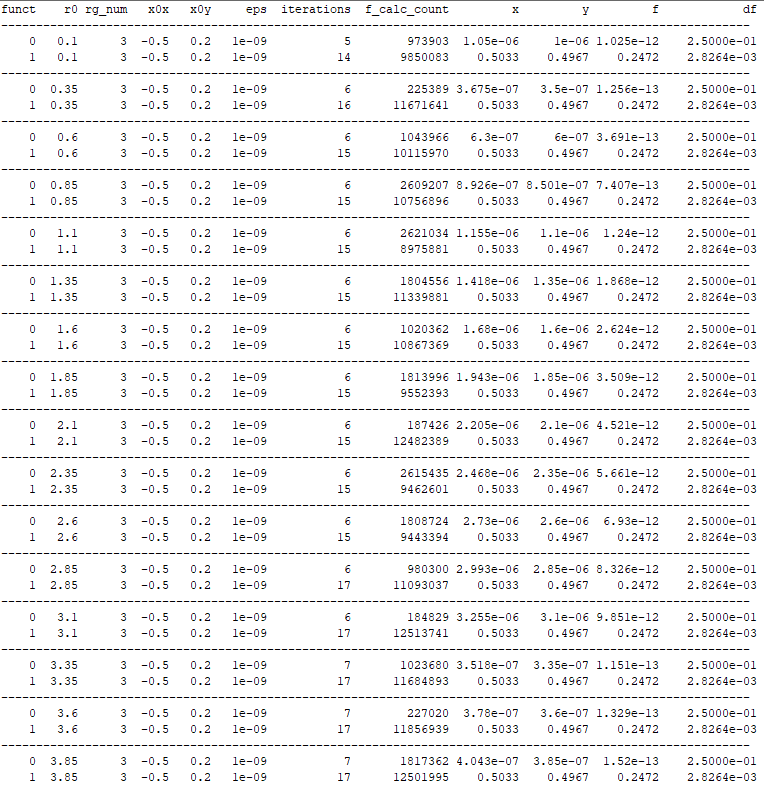




Лучше всего себя показывает .

Лучше всего себя показали мультипликативные стратегии изменения коэффициента штрафа, а именно . Использование степенных функций нежелательно, так как приводит к нахождению ложной точки минимума.

Проведем дополнительные исследования с помощью с варьированием начального коэффициента штрафа.



**Вывод:** для данной задачи лучшей функцией является , оптимальной стратегией выбора коэффициента штрафа является мультипликативная функция , оптимальный начальный коэффициент штрафа для этой функции выбирается достаточно свободно в диапазоне от **0.1**.

1. **Общие выводы**

Большое число вычисления функций объясняется необходимостью для каждой итерации метода штрафных или барьерных функций вызывать функцию метода Гаусса, который в свою очередь для каждой своей итерации два раза вызывает функцию одномерного поиска по компонентам методом золотого сечения, который в свою очередь вызывает функцию поиска отрезка с минимумом.

Высокая погрешность решения объясняется несовершенством метода Гаусса, работоспособность которого зависит от выбора начальной точки, а также вариацией параметра delta в функции поиска отрезка с минимумом.

1. **Текст программы**

**main.cpp**

#include <iostream>

#include <fstream>

#include "Gauss.h"

#include "PenaltyMethodData.h"

#include "PenaltyMethod.h"

#include "BarrierMethodData.h"

#include "BarrierMethod.h"

#include "Test.h"

using namespace std;

void PenaltyGTest(const string& file\_name)

{

// Объект для хранения тестовых функций

Test test = Test(0);

// Объект с данными для метода штрафных функций

PenaltyMethodData penalty\_method\_data = PenaltyMethodData(test);

// Объект метода штрафных функций

PenaltyMethod penalty\_method = PenaltyMethod(penalty\_method\_data);

// Поток вывода

ofstream fout(file\_name);

int rg\_num = 6;

vector<double> x0 = { -5, 2 };

fout << setw(5) << "funct" << setw(6) << "r0" << setw(7) << "rg\_num";

fout << setw(6) << "x0x" << setw(6) << "x0y";

fout << setw(8) << "eps" << setw(12) << "iterations" << setw(14) << "f\_calc\_count";

fout << setw(10) << "x" << setw(10) << "y" << setw(10) << "f";

fout << setw(14) << "df" << endl;

for(double i = 1; i < 10; i+=0.5)

{

double r0 = i;

fout << "----------------------------------------------------------";

fout << "-------------------------------------------------" << endl;

double f\_eps = pow(10, -9);

for(int funct\_n = 0; funct\_n < 4; funct\_n++)

{

penalty\_method.barrier\_method\_data.funct\_n = funct\_n;

vector<double> x = penalty\_method.FindExtremum(x0, r0, rg\_num, 0, f\_eps);

// Блок вывода

fout << setw(5) << penalty\_method.barrier\_method\_data.funct\_n << setw(6) << r0 << setw(7) << rg\_num;

fout << setw(6) << x0[0] << setw(6) << x0[1];

fout << setw(8) << f\_eps << setw(12) << penalty\_method.iterations\_count;

fout << setw(14) << penalty\_method.f\_calc\_cout;

fout << setw(10) << x[0] << setw(10) << x[1] << setw(10) << penalty\_method\_data.test.f(x) << scientific;

fout << setw(14) << abs(penalty\_method\_data.test.f(x) - penalty\_method\_data.test.f\_min1) << defaultfloat << endl;

}

}

fout.close();

}

void PenaltyHTest(const string& file\_name)

{

// Объект для хранения тестовых функций

Test test = Test(0);

// Объект с данными для метода штрафных функций

PenaltyMethodData penalty\_method\_data = PenaltyMethodData(test);

// Объект метода штрафных функций

PenaltyMethod penalty\_method = PenaltyMethod(penalty\_method\_data);

// Поток вывода

ofstream fout(file\_name);

int rh\_num = 4;

vector<double> x0 = { -5, 2 };

fout << setw(5) << "funct" << setw(6) << "r0" << setw(7) << "rh\_num";

fout << setw(6) << "x0x" << setw(6) << "x0y";

fout << setw(8) << "eps" << setw(12) << "iterations" << setw(14) << "f\_calc\_count";

fout << setw(10) << "x" << setw(10) << "y" << setw(10) << "f";

fout << setw(14) << "df" << endl;

for(int i = 140; i < 260; i += 10)

{

double r0 = i;

fout << "----------------------------------------------------------";

fout << "-------------------------------------------------" << endl;

double f\_eps = pow(10, -9);

for(int funct\_n = 0; funct\_n < 3; funct\_n++)

{

penalty\_method.barrier\_method\_data.funct\_n = funct\_n;

vector<double> x = penalty\_method.FindExtremum(x0, r0, 0, rh\_num, f\_eps);

// Блок вывода

fout << setw(5) << penalty\_method.barrier\_method\_data.funct\_n << setw(6) << r0 << setw(7) << rh\_num;

fout << setw(6) << x0[0] << setw(6) << x0[1];

fout << setw(8) << f\_eps << setw(12) << penalty\_method.iterations\_count;

fout << setw(14) << penalty\_method.f\_calc\_cout;

fout << setw(10) << x[0] << setw(10) << x[1] << setw(10) << penalty\_method\_data.test.f(x) << scientific;

fout << setw(14) << abs(penalty\_method\_data.test.f(x) - penalty\_method\_data.test.f\_min2) << defaultfloat << endl;

}

}

fout.close();

}

void BarrierGTest(const string& file\_name)

{

// Объект для хранения тестовых функций

Test test = Test(0);

// Объект с данными для метода барьерных функций

BarrierMethodData barrier\_method\_data = BarrierMethodData(test);

// Объект метода штрафных функций

BarrierMethod penalty\_method = BarrierMethod(barrier\_method\_data);

// Поток вывода

ofstream fout(file\_name);

int rg\_num = 3;

vector<double> x0 = { -0.5, 0.2 };

fout << setw(5) << "funct" << setw(6) << "r0" << setw(7) << "rg\_num";

fout << setw(6) << "x0x" << setw(6) << "x0y";

fout << setw(8) << "eps" << setw(12) << "iterations" << setw(14) << "f\_calc\_count";

fout << setw(10) << "x" << setw(10) << "y" << setw(10) << "f";

fout << setw(14) << "df" << endl;

fout << setprecision(4);

for(double i = 0.1; i < 5; i += 0.25)

{

double r0 = i;

fout << "----------------------------------------------------------";

fout << "-------------------------------------------------" << endl;

double f\_eps = pow(10, -9);

for(int funct\_n = 0; funct\_n < 2; funct\_n++)

{

penalty\_method.barrier\_method\_data.funct\_n = funct\_n;

vector<double> x = penalty\_method.FindExtremum(x0, r0, rg\_num, f\_eps);

// Блок вывода

fout << setw(5) << penalty\_method.barrier\_method\_data.funct\_n << setw(6) << r0 << setw(7) << rg\_num;

fout << setw(6) << x0[0] << setw(6) << x0[1];

fout << setw(8) << f\_eps << setw(12) << penalty\_method.iterations\_count;

fout << setw(14) << penalty\_method.f\_calc\_cout;

fout << setw(10) << x[0] << setw(10) << x[1] << setw(10) << barrier\_method\_data.test.f(x) << scientific;

fout << setw(14) << abs(barrier\_method\_data.test.f(x) - barrier\_method\_data.test.f\_min1) << defaultfloat << endl;

}

}

fout.close();

}

int main()

{

PenaltyGTest("results/penalty\_g\_test.txt");

//PenaltyHTest("results/penalty\_h\_test.txt");

//BarrierGTest("results/barrier\_g\_test.txt");

cout << "Done!";

}

**vector.h**

#pragma once

#include <vector>

#include <iomanip>

#include <fstream>

using namespace std;

// Умножение вектора на число

vector<double> operator \* (const double& val, vector<double> vec)

{

const size\_t size = vec.size();

for (size\_t i = 0; i < size; ++i)

vec[i] \*= val;

return vec;

}

// Деление вектора на число

vector<double> operator / (const double& val, vector<double> vec)

{

const size\_t size = vec.size();

for(size\_t i = 0; i < size; ++i)

vec[i] /= val;

return vec;

}

vector<double>& operator \*= (vector<double>& vec, const double& val)

{

const size\_t size = vec.size();

for(size\_t i = 0; i < size; ++i)

vec[i] \*= val;

return vec;

}

// Сложение векторов

vector<double> operator + (vector<double> vec1, const vector<double>& vec2)

{

const size\_t size = vec1.size();

for (size\_t i = 0; i < size; ++i)

vec1[i] += vec2[i];

return vec1;

}

// Вычитание векторов

vector<double> operator - (vector<double>vec1, const vector<double>& vec2)

{

const size\_t size = vec1.size();

for (size\_t i = 0; i < size; ++i)

vec1[i] -= vec2[i];

return vec1;

}

// Скалярное произведение векторов

double operator \* (const vector<double>& vec1, const vector<double>& vec2)

{

const size\_t size = vec1.size();

double res = 0;

for(size\_t i = 0; i < size; ++i)

res += vec1[i] \* vec2[i];

return res;

}

// Норма вектора

double Norm(const vector<double>& vec)

{

const size\_t size = vec.size();

double res = 0;

for(int i = 0; i < size; i++)

res += vec[i] \* vec[i];

return sqrt(res);

}

**Function.h**

#pragma once

#include <vector>

class Function

{

public:

virtual double GetValue(const std::vector<double>&) const = 0;

};

**Test.h**

#pragma once

#include <vector>

using namespace std;

// Класс с информацией о тестовых функциях и коэффициентах r

class Test

{

public:

int test\_n;

Test() : test\_n(0) {}

Test(int t\_test\_n) : test\_n(t\_test\_n) {}

double f(const vector<double>& x) const

{

switch(test\_n)

{

case 0: return 10 \* (x[1] - x[0]) \* (x[1] - x[0]) + x[1] \* x[1];

case 1: return (x[0] - 1) \* (x[0] - 1) + (x[1] - 3) \* (x[1] - 3);

case 2: return x[0] \* x[0] + x[1] \* x[1];

case 3: return 5 \* (x[1] + x[0]) \* (x[1] + x[0]) + (x[0] - 2) \* (x[0] - 2);

}

}

double h(const vector<double>& x) const

{

return x[0] + x[1] - 2;

}

double g(const vector<double>& x) const

{

return 1 - x[0] - x[1];

}

// Точные значения минимума задач и точек

// в которых эти значения достигаются

double f\_min1 = 0.25;

double x\_min1 = 0.5;

double y\_min1 = 0.5;

double f\_min2 = 1;

double x\_min2 = 1;

double y\_min2 = 1;

};

**Gauss.h**

#include "Function.h"

using namespace std;

/// <summary>

/// Класс для поиска экстремума функции многомерного параметра

/// методом Гаусса

/// </summary>

class Gauss

{

public:

size\_t size = 0; // Размерность вектора

vector<double> prev; // Приближение на текущем шаге

vector<double> curr; // Новое приближение,

double delta = 10;

int max\_iter\_count = 5000; // Максимальное количество итераций

///<param name = "t\_size">- размерность вектора функции</param>

Gauss(const size\_t& t\_size);

/// <summary>

/// Поиск экстремума функции методом Гаусса

/// </summary>

/// <param name="funct">- объект класса-наследника класса Function с информацией о функции</param>

/// <param name="x0">- вектор начального приближения</param>

/// <param name="f\_eps">- точость по изменению функции</param>

/// <param name="xs\_eps">- точность по компонентам</param>

/// <param name="fout">- поток вывода</param>

/// <returns>Количество итераций</returns>

int FindExtremum(const Function& funct,

const vector<double>& x0,

const double& f\_eps, const double& xs\_eps);

const double SQRT5 = sqrt(5);

const double PI = 3.141592653589793238462;

int f\_calc\_count = 0; // Количество вычислений функции

/// <summary>

/// Поиск отрезка с минимумом функции для метода Гаусса

/// </summary>

/// <param name="funct">- объект класса-наследника класса Function с информацией о функции</param>

/// <param name="x">- вектор компонент</param>

/// <param name="comp\_n">- номер компоненты</param>

/// <returns>Пара чисел - отрезок с минимумом функции</returns>

pair<double, double> FindSegmentWithMin(const Function& funct,

vector<double> x,

const int& comp\_n);

/// <summary>

/// Поиск аргумента минимума функции для метода Гаусса

/// </summary>

/// <param name="funct">- объект класса-наследника класса Function с информацией о функции</param>

/// <param name="x">- вектор компонент</param>

/// <param name="comp\_n">- номер компоненты</param>

/// <param name="eps">- точность поиска</param>

/// <returns>Аргумент минимума функции</returns>

double FindMinArgGolden(const Function& funct,

vector<double> x,

const int& comp\_n,

const double& eps);

};

**Gauss.cpp**

#include <vector>

#include <fstream>

#include "Gauss.h"

Gauss::Gauss(const size\_t& t\_size) : size(t\_size)

{

prev = vector<double>(t\_size);

curr = vector<double>(t\_size);

}

int Gauss::FindExtremum(const Function& funct,

const vector<double>& x0,

const double& f\_eps, const double& xs\_eps)

{

prev = x0;

curr = x0;

bool result\_fit = false;

int iter\_count = 0;

do

{

for(size\_t i = 0; i < size; i++)

curr[i] = FindMinArgGolden(funct, curr, i, 1e-15);

iter\_count++;

// Расчет изменения решения на текущей итерации

if(abs(funct.GetValue(prev) - funct.GetValue(curr)) < f\_eps)

result\_fit = true;

for(int i = 0; i < size && result\_fit; i++)

if(abs(prev[i] - curr[i]) > xs\_eps)

result\_fit = false;

prev = curr;

} while(iter\_count < max\_iter\_count && !result\_fit);

return iter\_count;

}

pair<double, double> Gauss::FindSegmentWithMin(const Function& funct,

vector<double> x,

const int& comp\_n)

{

pair<double, double> result;

double x0 = 0;

double xk, xk1, xk\_1, h = 1;

x[comp\_n] = x0;

double f = funct.GetValue(x);

f\_calc\_count += 1;

x[comp\_n] = x0 + delta;

if(f == funct.GetValue(x))

{

result.first = x0;

result.second = x0 + delta;

f\_calc\_count = 2;

return result;

}

else

{

x[comp\_n] = x0 - delta;

if(f == funct.GetValue(x))

{

result.first = x0 - delta;

result.second = x0;

f\_calc\_count = 3;

return result;

}

else

{

x[comp\_n] = x0 + delta;

if(f > funct.GetValue(x))

{

xk = x0 + delta;

h = delta;

f\_calc\_count++;

}

else

{

x[comp\_n] = x0 - delta;

if(f > funct.GetValue(x))

{

xk = x0 - delta;

h = -delta;

f\_calc\_count += 2;

}

else

{

result.first = x0 - delta;

result.second = x0 + delta;

f\_calc\_count += 2;

return result;

}

}

xk\_1 = x0;

bool exit = false;

do

{

h \*= 2;

xk1 = xk + h;

x[comp\_n] = xk;

double f1 = funct.GetValue(x);

x[comp\_n] = xk1;

double f2 = funct.GetValue(x);

if(f1 > f2)

{

xk\_1 = xk;

xk = xk1;

}

else

exit = true;

f\_calc\_count += 2;

} while(!exit);

result.first = xk\_1;

result.second = xk;

}

}

return result;

}

double Gauss::FindMinArgGolden(const Function& funct,

vector<double> x,

const int& comp\_n,

const double& eps)

{

pair<double, double> segment = FindSegmentWithMin(funct, x, comp\_n);

double a = segment.first, b = segment.second;

double x1 = a + (3 - SQRT5) / 2 \* (b - a);

double x2 = a + (SQRT5 - 1) / 2 \* (b - a);

x[comp\_n] = x1;

double f1 = funct.GetValue(x);

f\_calc\_count += 1;

x[comp\_n] = x2;

double f2 = funct.GetValue(x);

f\_calc\_count += 1;

double a1, b1;

int iter\_count = 0;

for(; abs(b - a) > eps; iter\_count++)

{

a1 = a, b1 = b;

if(f1 < f2)

{

b = x2;

x2 = x1;

x1 = a + (3 - SQRT5) / 2 \* (b - a);

f2 = f1;

x[comp\_n] = x1;

f1 = funct.GetValue(x);

f\_calc\_count += 1;

}

else

{

a = x1;

x1 = x2;

x2 = a + (SQRT5 - 1) / 2 \* (b - a);

f1 = f2;

x[comp\_n] = x2;

f2 = funct.GetValue(x);

f\_calc\_count += 1;

}

}

return a;

}

**PenaltyMethodData.cpp**

#pragma once

#include<vector>

#include "Test.h"

#include "Function.h"

using namespace std;

class PenaltyMethodData : public Function

{

public:

Test test;

int funct\_n = 0;

double rg = 0;

double rh = 0;

PenaltyMethodData(const Test& t\_test, int t\_funct\_n) :

test(t\_test), funct\_n(t\_funct\_n) { }

PenaltyMethodData(const Test& t\_test) : test(t\_test) { }

double G(const vector<double>& x) const

{

if(test.g(x) <= 0)

return 0;

else

{

switch(funct\_n)

{

case 0: return 0.5 \* (test.g(x) + abs(test.g(x)));

case 1: return pow(0.5 \* (test.g(x) + abs(test.g(x))), 2);

case 2: return pow(0.5 \* (test.g(x) + abs(test.g(x))), 4);

case 3: return 1.0 / test.g(x);

default: return 0;

}

}

}

double H(const vector<double>& x) const

{

switch(funct\_n)

{

case 0: return abs(test.h(x));

case 1: return pow(test.h(x), 2);

case 2: return pow(test.h(x), 4);

default: return 0;

}

}

double GetValue(const vector<double>& x) const override

{

return test.f(x) + rg \* G(x) + rh \* H(x);

}

};

**PenaltyMethod.cpp**

#pragma once

#include <iomanip>

#include "PenaltyMethodData.h"

using namespace std;

class PenaltyMethod

{

public:

PenaltyMethodData barrier\_method\_data;

int max\_iter\_count = 100; // Максимальное количество итераций

int iterations\_count = 0;

int f\_calc\_cout = 0;

double CalcNewR(const int& num, const double& r)

{

switch(num)

{

case 0: return 0.0;

case 1: return r + 1.0;

case 2: return r + 100;

case 3: return r \* 2;

case 4: return r \* 200;

case 5: return r \* r;

case 6: return r \* r \* r;

}

}

PenaltyMethod(const PenaltyMethodData& t\_penalty\_method\_data):

barrier\_method\_data(t\_penalty\_method\_data) { }

/// <summary>

/// Поиск экстремума методом штрафных функций

/// </summary>

/// <param name="x0">- начальная точка</param>

/// <param name="r0">- начальное значение коэффициента штрафа</param>

/// <param name="rg\_num">- номер функции для вычисления параметра штрафа для функции G</param>

/// <param name="rh\_num">- номер функции для вычисления параметра штрафа для функции H</param>

/// <param name="f\_eps">- точость по изменению функции</param>

/// <returns>Точка с найденным экстремумом</returns>

vector<double> FindExtremum(const vector<double>& x0,

const double& r0,

const int& rg\_num,

const int& rh\_num,

const double& f\_eps)

{

barrier\_method\_data.rg = r0;

barrier\_method\_data.rh = r0;

// Значения функции на предыдущей и текущей итерациях

double prev\_f = barrier\_method\_data.test.f(x0);

double curr\_f;

// Найденная точка

vector<double> x(2);

f\_calc\_cout = 0;

iterations\_count = 0;

bool result\_fit = false;

do

{

Gauss gauss = Gauss(2);

// Изменение коэффициентов штрафа

barrier\_method\_data.rg = CalcNewR(rg\_num, barrier\_method\_data.rg);

barrier\_method\_data.rh = CalcNewR(rh\_num, barrier\_method\_data.rh);

// Поиск экстремума методом Гаусса

gauss.FindExtremum(barrier\_method\_data, x0, 1e-20, 1e-20);

// Полученная точка

x = gauss.curr;

// Функция в этой точке

curr\_f = barrier\_method\_data.test.f(gauss.curr);

iterations\_count++;

f\_calc\_cout += gauss.f\_calc\_count;

if(abs(prev\_f - curr\_f) < f\_eps)

result\_fit = true;

prev\_f = curr\_f;

} while(!result\_fit && iterations\_count < max\_iter\_count);

return x;

}

};

**BarrierMethodData.cpp**

#pragma once

#include<vector>

#include "Test.h"

#include "Function.h"

using namespace std;

class BarrierMethodData : public Function

{

public:

Test test;

int funct\_n = 0;

double rg = 0;

BarrierMethodData(const Test& t\_test, int t\_funct\_n) :

test(t\_test), funct\_n(t\_funct\_n) { }

BarrierMethodData(const Test& t\_test) : test(t\_test) { }

double G(const vector<double>& x) const

{

switch(funct\_n)

{

case 0: return - 1.0 / test.g(x);

case 1: return -1.0 \* log(-1.0 \* test.g(x));

default: return 0;

}

}

double GetValue(const vector<double>& x) const override

{

return test.f(x) + rg \* G(x);

}

};

**BarrierMethod.cpp**

#pragma once

#include <iomanip>

#include "BarrierMethodData.h"

using namespace std;

class BarrierMethod

{

public:

BarrierMethodData barrier\_method\_data;

int max\_iter\_count = 100; // Максимальное количество итераций

int iterations\_count = 0;

int f\_calc\_cout = 0;

double CalcNewR(const int& num, const double& rg)

{

switch(num)

{

case 0: return 0.0;

case 1: return rg - 0.01;

case 2: return rg / 2;

case 3: return rg / 10;

case 4: return pow(rg, 0.5);

case 5: return pow(rg, 0.25);

}

}

BarrierMethod(const BarrierMethodData& t\_barrier\_method\_data) :

barrier\_method\_data(t\_barrier\_method\_data) { }

/// <summary>

/// Поиск экстремума методом барьерных функций

/// </summary>

/// <param name="x0">- начальная точка</param>

/// <param name="r0">- начальное значение коэффициента штрафа</param>

/// <param name="rg\_num">- номер функции для вычисления параметра штрафа для функции G</param>

/// <param name="f\_eps">- точость по изменению функции</param>

/// <returns>Точка с найденным экстремумом</returns>

vector<double> FindExtremum(const vector<double>& x0,

const double& r0,

const int& rg\_num,

const double& f\_eps)

{

barrier\_method\_data.rg = r0;

// Значения функции на предыдущей и текущей итерациях

double prev\_f = barrier\_method\_data.test.f(x0);

double curr\_f;

// Найденная точка

vector<double> x(2);

f\_calc\_cout = 0;

iterations\_count = 0;

bool result\_fit = false;

do

{

Gauss gauss = Gauss(2);

// Изменение коэффициентов штрафа

barrier\_method\_data.rg = CalcNewR(rg\_num, barrier\_method\_data.rg);

// Поиск экстремума методом Гаусса

gauss.FindExtremum(barrier\_method\_data, x0, 1e-20, 1e-20);

// Полученная точка

x = gauss.curr;

// Функция в этой точке

curr\_f = barrier\_method\_data.test.f(gauss.curr);

iterations\_count++;

f\_calc\_cout += gauss.f\_calc\_count;

if(abs(prev\_f - curr\_f) < f\_eps)

result\_fit = true;

prev\_f = curr\_f;

} while(!result\_fit && iterations\_count < max\_iter\_count);

return x;

}

};