**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт №3. Системы управления, информатика и электроэнергетика. Кафедра №304 Вычислительные машины, системы и комплексы

Отчет по лабораторной работе

по учебной дисциплине

теория оптимального планирования и управления

на тему

**«Методы штрафных и барьерных функций»**

Группа М30-207Б

Выполнили студенты:

Кривонос А.А.

Приняли:

Татарникова Е.М.

Давыдкина Е.А.

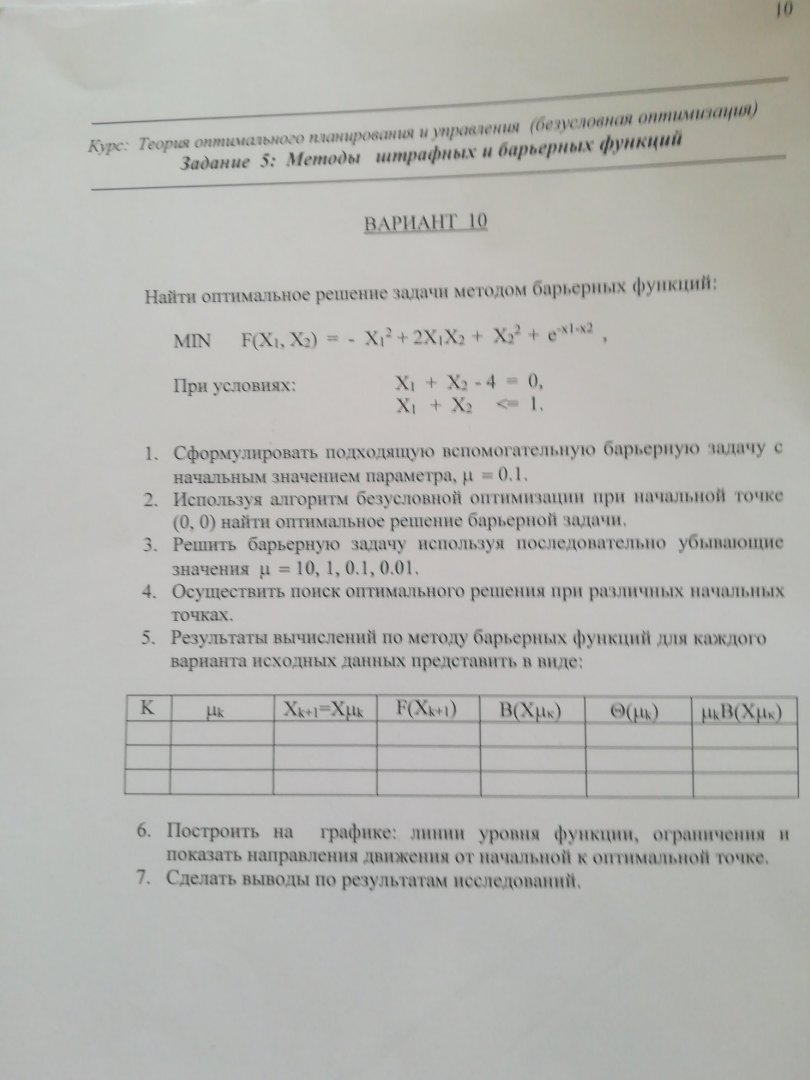
Москва, 2019г.

Содержание**:**

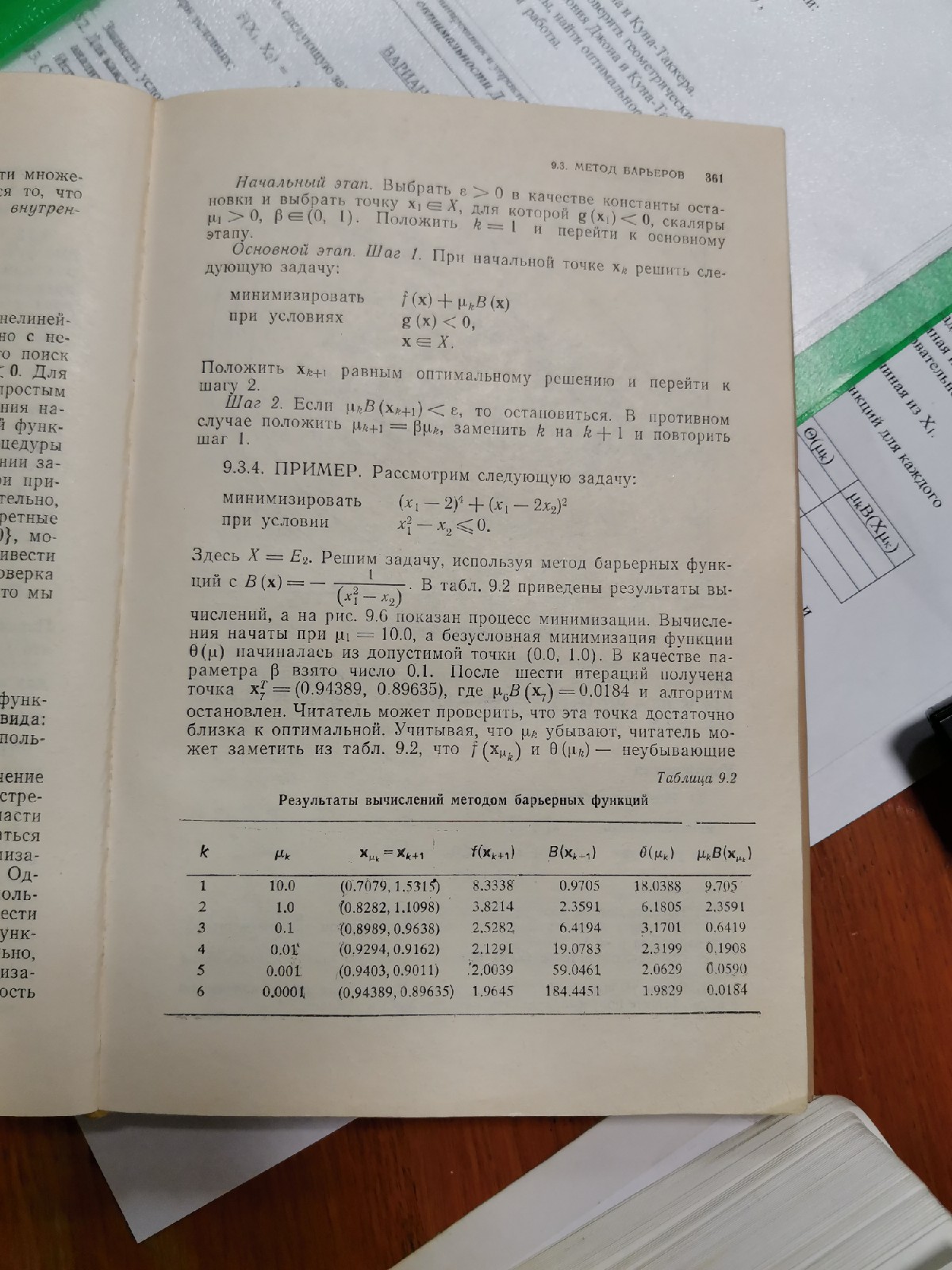
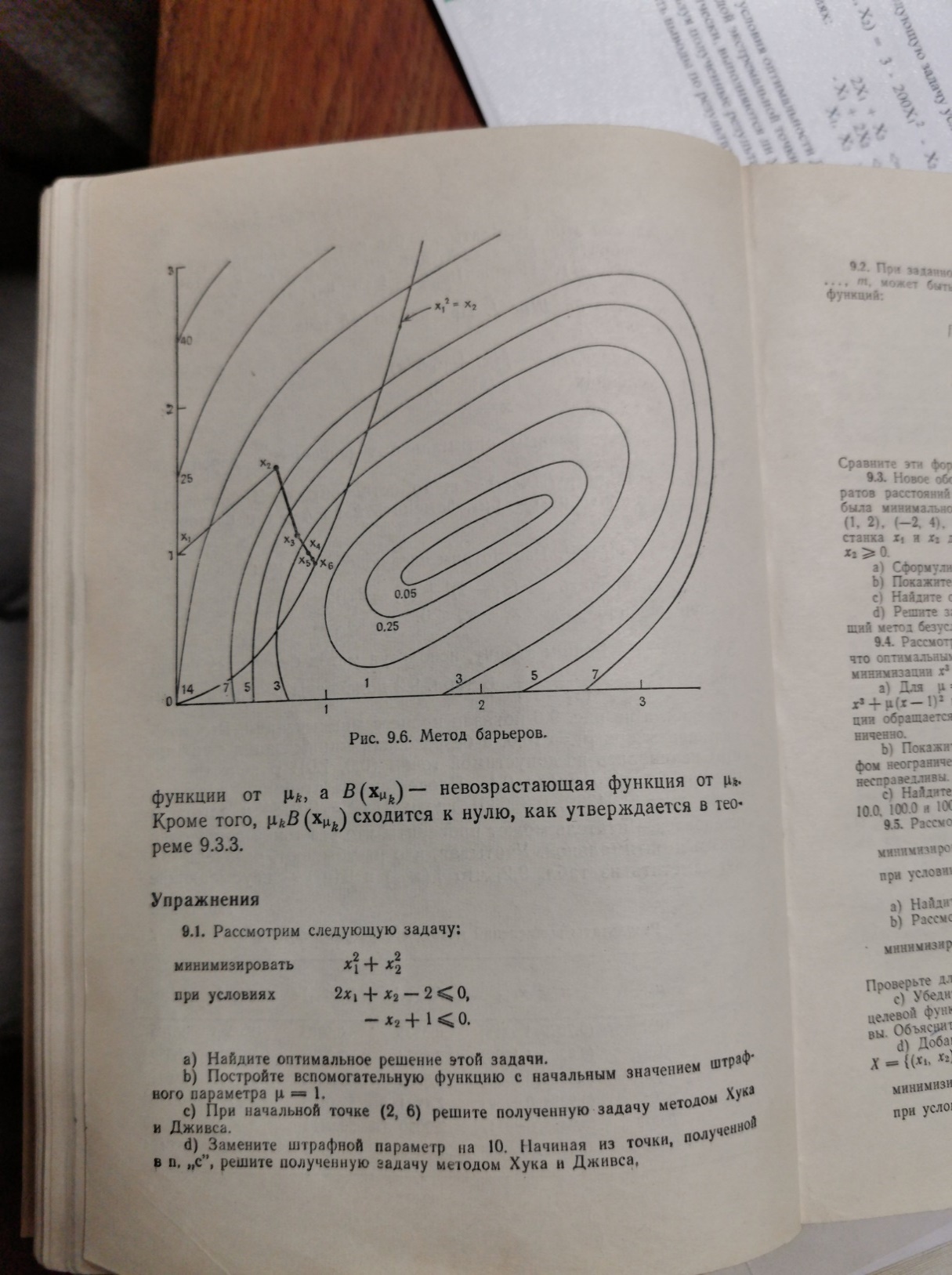
1. Задание………………………………………………………………......... 2
2. Теория………………………………..……………………………………..4
3. Текст программы……….………………………………………………… 7
4. Графики …….……...................................................................................... 9
5. Тест для изначальной функции с некорректными ограничениями....... 14
6. Тест для функции из методички F=(x1-4)4+(x1-2x2)2 ………………....... 16
7. Тест для функции с измененными ограничениями……………………..18
8. Сводная таблица…………………………………………………………..20
9. Вывод………………………………………………………………………21

**Задания**

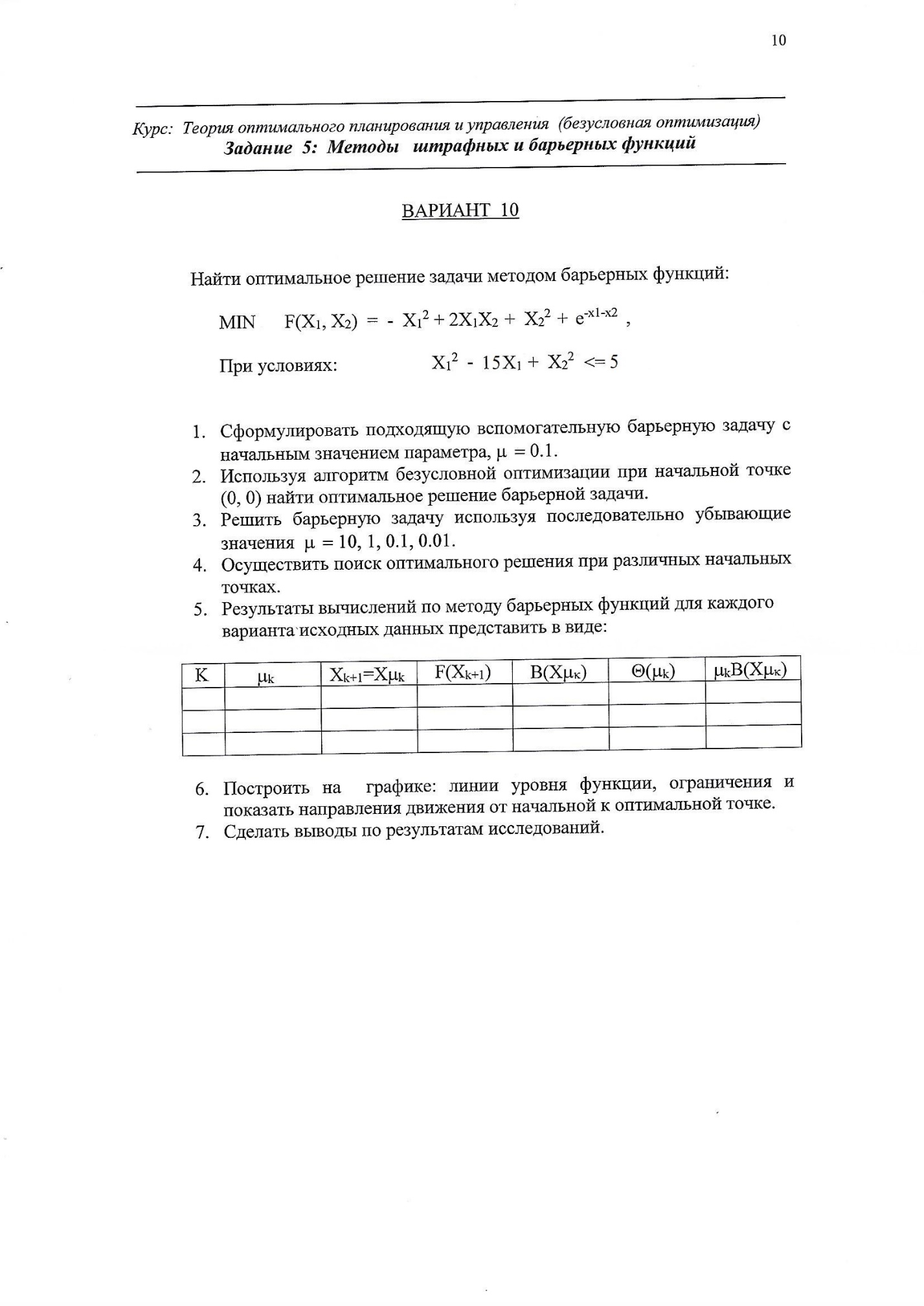
(Изначальное задание)





(Задание и решение из методички для отладки программы)

(Измененное задание с корректными ограничениями)



**Метод барьеров**

Барьерные функции также используются для преобразования задачи с ограничениями в задачу безусловной оптимизации или в последовательность таких задач. Барьерные функции как бы препятствуют выходу из допустимой области. Если оптимальное решение оказывается на границе допустимой области, то процедура приводит к движению изнутри области к границе.

**Алгоритм метода барьерных функций**

Пусть имеется следующая задача:

Минимизировать f(x) при ограничениях g_i(x)\ge0,h_i(x)=0, где функции f,g_i,h_i.

**Начальный этап** Выбрать \epsilon>0 Выбрать начальную точку x^1, параметр штрафа r_1 и число \beta>1. Положить k=1 и перейти к основному этапу.

**Основной этап**. *k-я итерация*.

**Первый шаг**.

При начальной точке x_k и параметре штрафа r_kрешить следующую задачу:

минимизировать

f(x)+r_k\alpha(x)=f(x)+r_k\left[\sum_{i=1}^{m}(max\{0,-g_i(x)\})^p+ \sum_{i=m+1}^{l}|h_i(x)|^p\right] , где

p>0, p - целое.

Положить x_{k+1} равным оптимальному решению задачи и перейти ко второму шагу.

**Второй шаг**

Если r_k\alpha(x_{k+1})<\epsilon, то остановиться.

В противном случае положить r_{k+1}=\beta r_k. Заменить k на k+1 и перейти к первому шагу.

**Алгоритм метода Нелдера-Мида**

Инициализация. Произвольным образом выбирается n+1 точка x_i = \left(x^{(1)}_i,x^{(2)}_i,\ldots,x^{(n)}_i\right), образующие симплекс n-мерного пространства. В этих точках вычисляются значения функции: f_1=f(x^{(1)}_i), f_2=f(x^{(2)}_i),\ldots, f_{n+1}=f(x^{(n+1)}_i).

1. Сортировка. Из вершин симплекса выбираем три точки: x_h с наибольшим (из выбранных) значением функции f_h, x_g со следующим по величине значением f_g и x_l с наименьшим значением функции f_l. Целью дальнейших манипуляций будет уменьшение по крайней мере f_h.

2. Найдём центр тяжести всех точек, за исключением  x_h: x_c=\frac{1}{n} \sum_{i\neq h} x_i Вычислять f_c=f(x_c) не обязательно.

3. Отражение. Отразим точку x_h относительно x_c с коэффициентом \alpha (при \alpha=1 это будет центральная симметрия, в общем случае — гомотетия), получим точку x_r и вычислим в ней функцию: f_r=f(x_r). Координаты новой точки вычисляются по формуле x_r = (1+\alpha)x_c - \alpha x_h

4. Далее сравниваем значение f_r со значениями f_h, f_g, f_l:

4а. Если f_r<f_l, то производим растяжение. Новая точка x_e = (1-\gamma)x_c + \gamma x_r и значение функции f_e=f(x_e).

Если f_e<f_l, то заменяем точку x_h на x_e и заканчиваем итерацию (на шаг 8).

Если f_e>f_l, то заменяем точку x_h на x_r и заканчиваем итерацию (на шаг 8).

4b. Если f_l < f_r < f_g, то заменяем точку x_h на x_r и переходим на шаг 8.

4с. Если f_h > f_r > f_g, то меняем обозначения x_r, x_h (и соответствующие значения функции) местами и переходим на шаг 5.

4d. Если f_r > f_h, то переходим на шаг 5.

5. Сжатие. Строим точку x_s = \beta x_h + (1-\beta) x_c и вычисляем в ней значение f_s.

6. Если f_s < f_h, то заменяем точку x_h на x_s и переходим на шаг 8.

7. Если f_s > f_h, то производим сжатие симплекса — гомотетию к точке с наименьшим значением x_0: x_i \to x_0 + (x_i-x_0)/2 для всех требуемых точек x_i.

8. Последний шаг — проверка сходимости. Может выполняться по-разному, например, оценкой дисперсии набора точек. Суть проверки заключается в том, чтобы проверить взаимную близость полученных вершин симплекса, что предполагает и близость их к искомому минимуму. Если требуемая точность ещё не достигнута, можно продолжить итерации с шага 1.

**Теорема о переходе**

Пусть 𝑓: 𝐸𝑛→𝐸1 и 𝑔: 𝐸𝑛→𝐸𝑚 непрерывные функции и X- непустое замкнутое множество из 𝐸𝑛. Предположим, что множество {𝑥∈X: g(x)<0} не пусто. Кроме того предположим, что исходная задача минимизации f(x) при условиях g(x)≤0, x∈X имеет оптимальное решение 𝑥̃ , обладающее следующим свойством. Для любой окрестности N точки 𝑥̃ существует точка 𝑥∈𝑋∩𝑁 такая, что g(x)<0. Тогда

{min f(x): g(x)≤0, x∈𝑋}=lim 𝜇→0+ 𝜃(𝜇) = inf 𝜇>0 𝜃(𝜇)

Положим 𝜃(𝜇) = 𝑓(𝑥𝜇)+𝜇𝐵(𝑥𝜇), где 𝑥𝜇∈𝑋 и g(𝑥𝜇)<0. Тогда предел любой сходящейся подпоследовательности является оптимальным решением условной задачи.

**Текст программы**

#include <iostream>

#include <iomanip>

using namespace std;

const int i = 2; //колличество переменных

const int j = i + 1; //колличество вершин

double mu = 10; //коэффициент мю

//НАЧАЛЬНЫЕ ОГРАНИЧЕНИЯ И ВСПОМОГАТЕЛЬНАЯ ФУНКЦИЯ

////первое ограничение(барьер)

//double F1(double x[2]) {

// if (x[0] + x[1] + 4 < 0)

// return -1 / (x[0] + x[1] + 4);

// else

// return 100000000;

//}

////второе ограничение(барьер)

//double F2(double x[2]) {

// if (x[0] + x[1] < 1)

// return -1 / (x[0] + x[1] - 1);

// else

// return 100000000;

//}

////вспомогательная функция

//double F(double x[2]) {

// return (-x[0] \* x[0] + 2 \* x[0] \* x[1] + x[1] \* x[1] + pow(2.8, -x[0] - x[1]) + mu \* (F1(x) + F2(x) + F3(x) + F4(x)));

//}

//

//ИСПРАВЛЕННЫЕ ОГРАНИЧЕНИЕ И ВСПОМОГАТЕЛЬНАЯ ФУНКЦИЯ

double F1(double x[2]) {

if (x[0]\*x[0] - 15\*x[0] + x[1]\*x[1] < 5)

return -1 / (x[0] \* x[0] - 15 \* x[0] + x[1]\*x[1] - 5);

else

return 100000000;

}

//вспомогательная функция

double F(double x[2]) {

return (-pow(x[0], 2) + 2 \* x[0] \* x[1] + pow(x[1], 2) + pow(2.8, (-x[0] - x[1])) + mu \* (F1(x) /\*+ F2(x) + F3(x) + F4(x)\*/));

}

//ОГРАНИЧЕНИЕ И ВСПОМОГАТЕДНАЯ ФУНКЦИЯ ИЗ МЕТОДИЧКИ

//double F(double x[2]) {

// return (pow((x[0] - 2), 4) + pow((x[0] + 2\*x[1]), 2) + mu \* (F1(x) + F3(x) + F4(x)));

//}

//первое ограничение(барьер)

//double F1(double x[2]) {

// if (x[0] \* x[0] + x[1] < 0)

// return -1 / (x[0] \* x[0] + x[1]);

// else

// return 100000000;

//}

//третье ограничение(барьер)

//double F3(double x[2]) {

// if (x[0] > 0)

// return 1 / x[0];

// else

// return 100000000;

//}

////четвертое ограничение(барьер)

//double F4(double x[2]) {

// if (x[1] > 0)

// return 1 / x[1];

// else

// return 1000000000;

//}

//условие остановки алгоритма минимизации

double D(double xh[2], double xg[2], double xl[2], double xc[2])

{

return sqrt(1 / (j + 1) \* ((F(xh)) - F(xc)) \* ((F(xh)) - F(xc)) + ((F(xg)) - F(xc)) \* ((F(xg)) - F(xc)) + ((F(xl)) - F(xc)) \* ((F(xl)) - F(xc)));

}

void Sort(double x[j][i]);

void NeldMid(double x[2]);

int main() {

system("color F0"); //фон белый буквы черные

double Eps = 0.001; //точность

double x[i] = { 6,-6 }; //(-16;-16) - отрицательный тест

double b = 0.1;

int k = 0;

cout << char(218) << setfill(char(196)) << setw(5);

cout << char(194) << setfill(char(196)) << setw(14);

for (int i = 0; i < 5; i++)

{

cout << char(194) << setfill(char(196)) << setw(22);

}

cout << char(191) << endl;

cout << char(179) << " k "

<< char(179) << " mu "

<< char(179) << " X(k+1) = X(mu) "

<< char(179) << " f(X(k+1)) "

<< char(179) << " B(x) "

<< char(179) << " b "

<< char(179) << " mu\*B(x) "

<< char(179) << endl;

while (mu \* (F1(x) /\*+ F2(x) + F3(x) + F4(x)\*/) > Eps)

{

NeldMid(x);

cout << char(195) << setfill(char(196)) << setw(5);

cout << char(197) << setfill(char(196)) << setw(14);

for (int i = 0; i < 5; i++)

{

cout << char(197) << setfill(char(196)) << setw(22);

}

cout << char(180) << endl;

cout << char(179) << " " << setfill(char(32)) << setw(2) << k << " ";

cout << char(179) << " " << setfill(char(32)) << setw(11) << mu << " ";

cout << char(179) << " (" << setfill(char(32)) << setw(7) << round(x[0] \* 1000) / 1000 << " ; " << left << setfill(char(32)) << setw(7) << round(x[1] \* 1000) / 1000 << right << ") ";

cout << char(179) << " " << setfill(char(32)) << setw(11) << F(x) << " ";

cout << char(179) << " " << setfill(char(32)) << setw(11) << (F1(x) /\*+ F2(x) + F3(x) + F4(x)\*/) << " ";

cout << char(179) << " " << setfill(char(32)) << setw(5) << b << " ";

cout << char(179) << " " << setfill(char(32)) << setw(11) << mu \* (F1(x) /\*+ F2(x) + F3(x) + F4(x)\*/) << " ";

cout << char(179) << endl;

mu = mu \* b;

k++;

}

cout << char(192) << setfill(char(196)) << setw(5);

cout << char(193) << setfill(char(196)) << setw(14);

for (int i = 0; i < 5; i++)

{

cout << char(193) << setfill(char(196)) << setw(22);

}

cout << char(217) << endl;

system("pause");

}

void NeldMid(double x1[2]) {

int n; //индексная переменная

int k = 0; //количество итераций

bool f = 0; //вспомогательный флаг

double Temp[i]; //временная переменная

double Eps = 0.001; //точность

double a = 1; //коэфицент отражения

double b = 0.5; //коэфицент сжатия

double y = 2; //коэфицент рястяжения

//начальный набор точек

double xHigh[2]; //точка с наибольшим значением функции

double xAverage[2]; //средняя точка по велечине значения функции

double xLow[2]; //точка с самым маленьким значением функции

double centerOfWeight[2]; //центр тяжести

double reflection[2]; //отраженный х

double reflectionStretch[2]; //отраженный х c растяжением

double reflectionSqueeze[2]; //отраженный х с жатеем

double x[j][i] = { {x1[0],x1[1]},{ x1[0] + 3,x1[1] + 2 },{ x1[0] - 3,x1[1] + 2 } };

do {

k++;

Sort(x); //сортировка точек

/\*Заполнение массивов\*/

for (n = 0; n < i; n++)

{

centerOfWeight[n] = 0;

reflection[n] = 0;

reflectionStretch[n] = 0;

reflectionSqueeze[n] = 0;

xHigh[n] = x[i][n];

xAverage[n] = x[i - 1][n];

xLow[n] = x[0][n];

}

f = 0;

//поиск центра тяжести

for (n = 0; n < i; n++)

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

centerOfWeight[n] = centerOfWeight[n] + (x[m][n]);

centerOfWeight[0] = centerOfWeight[0] / i; centerOfWeight[1] = centerOfWeight[1] / i;

for (n = 0; n < i; n++)

reflection[n] = centerOfWeight[n] + a \* (centerOfWeight[n] - xHigh[n]);

if (F(reflection) <= F(xLow))

{

for (n = 0; n < i; n++)

reflectionStretch[n] = centerOfWeight[n] + y \* (reflection[n] - centerOfWeight[n]);

double x7 = F(reflectionStretch);

if (F(reflectionStretch) < F(reflection))

for (n = 0; n < i; n++)

xHigh[n] = reflectionStretch[n];

else

for (n = 0; n < i; n++)

xHigh[n] = reflection[n];

}

else if (F(xLow) <= F(reflection) && F(reflection) <= F(xAverage))

{

for (n = 0; n < i; n++)

xHigh[n] = reflection[n];

}

else if (F(xAverage) <= F(reflection) && F(reflection) <= F(xHigh))

{

for (n = 0; n < i; n++)

{

Temp[n] = reflection[n];

reflection[n] = xHigh[n];

xHigh[n] = Temp[n];

f = 1;

}

}

else if (F(xHigh) <= F(reflection))

f = 1;

if (f) {

for (n = 0; n < i; n++)

reflectionSqueeze[n] = centerOfWeight[n] + b \* (xHigh[n] - centerOfWeight[n]);

if (F(reflectionSqueeze) < F(xHigh))

for (n = 0; n < i; n++)

xHigh[n] = reflectionSqueeze[n];

if (F(reflectionSqueeze) >= F(xHigh))

{

for (n = 0; n < i; n++)

{

x[i][n] = xHigh[n];

x[i - 1][n] = xAverage[n];

x[0][n] = xLow[n];

}

Sort(x); //сортировка точек

for (n = 0; n < i; n++)

{

xHigh[n] = x[i][n];

xAverage[n] = x[i - 1][n];

xLow[n] = x[0][n];

}

for (n = 0; n < i; n++)

{

xHigh[n] = (xHigh[n] + xLow[n]) / 2;

xAverage[n] = (xAverage[n] + xLow[n]) / 2;

}

}

}

for (n = 0; n < i; n++)

{

x[i][n] = xHigh[n];

x[i - 1][n] = xAverage[n];

x[0][n] = xLow[n];

}

} while (fabs(D(xHigh, xAverage, xLow, centerOfWeight)) > Eps);

Sort(x); //сортировка точек

for (n = 0; n < i; n++)

x1[n] = xLow[n];

}

void Sort(double x[j][i])

{

double Min; //минимальный элемент

int jMin; //индекс минимального элемента

int iSort; //граница отсортированной области

int k; //индексная переменная

double Temp[i]; //обменная переменная

for (iSort = 0; iSort < j - 1; iSort++)

{

//принимаем первый элемент из неупорядоченных за минимальный

Min = F(x[iSort]); //минимум

jMin = iSort; //его индекс

//ищем минимальный элемент в оставшейся части массива

for (k = iSort + 1; k < j; k++)

{

if (F(x[k]) < Min) //очередной кандидат на минимальный

{

//запоминаем минимальный элемент и его номер

Min = F(x[k]);

jMin = k;

}

}

/\*нашли минимум в неупорядоченной части массива

ставим его на место первого в неупорядоченной части массива

меняем элементы местами\*/

for (k = 0; k < i; k++)

{

Temp[k] = x[iSort][k];

x[iSort][k] = x[jMin][k];

x[jMin][k] = Temp[k];

}

} //for iSort

}

**Тесты**

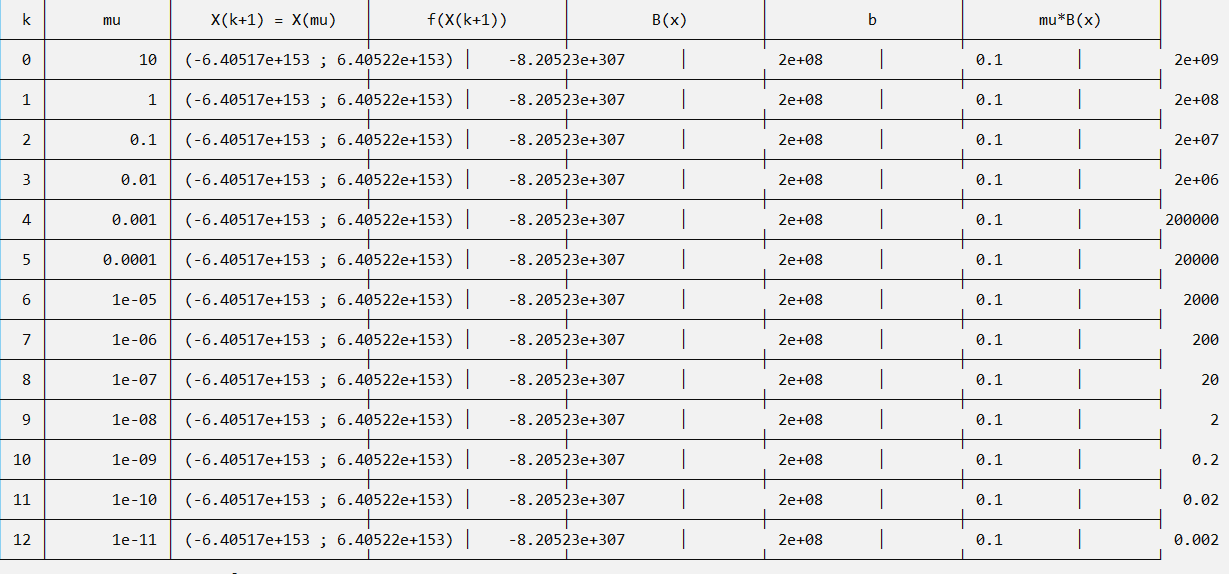
1. Тест для изначальной функции с некорректными ограничениями

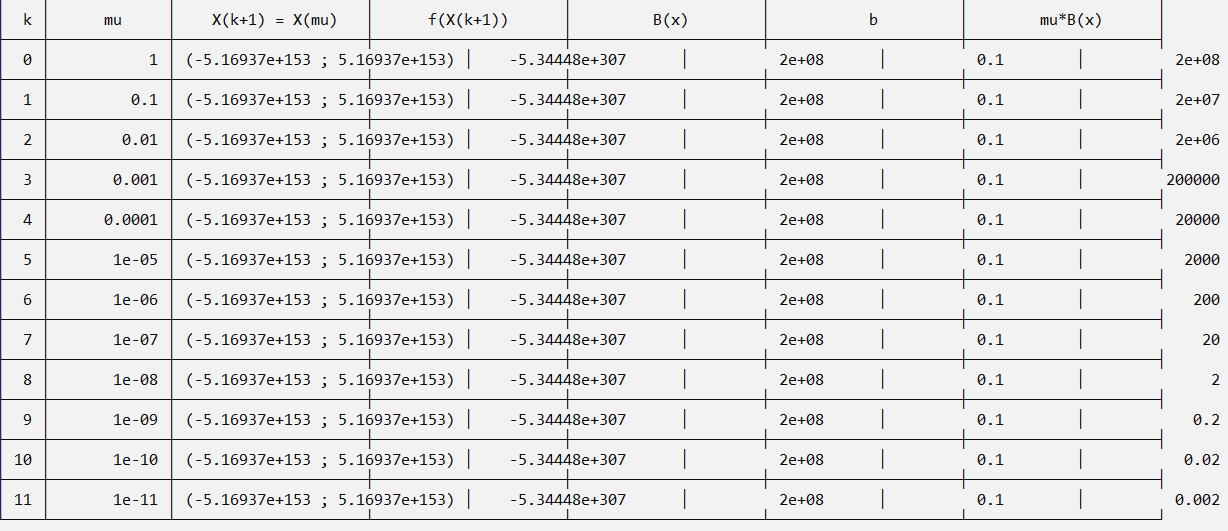
F= -x12+2 x1 x2 + x22 + exp(-x1-x2)

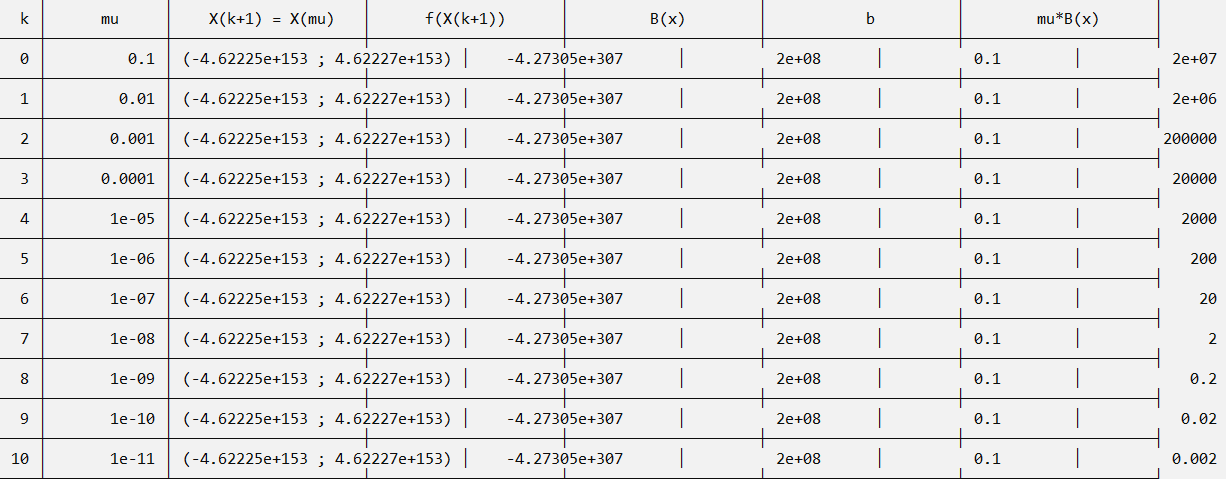
Ограничения: x1 + x2 -4 = 0

x1 + x2 < 1

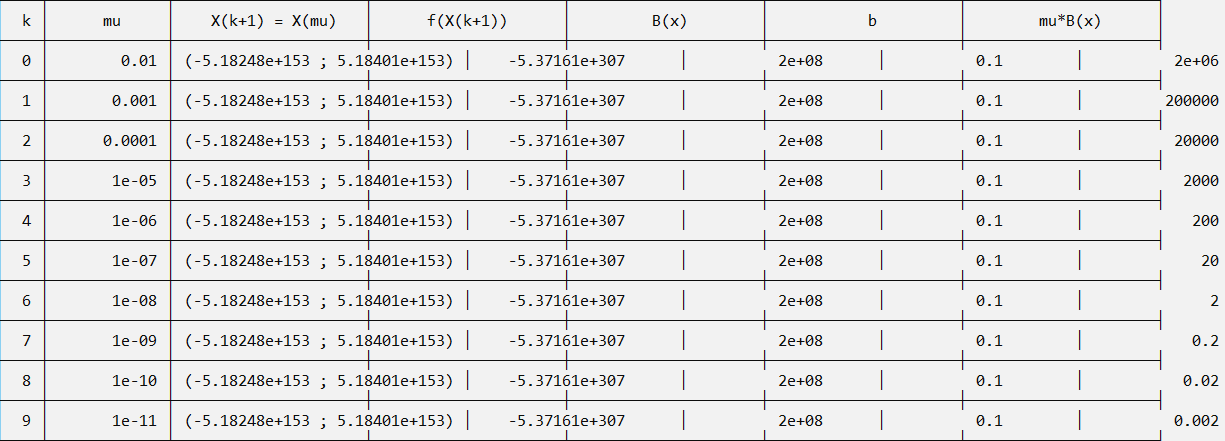
Начальная точка х=(0;0) – точка, данная по заданию, удовлетворяют допустимой области. Коэффициент = 0.1

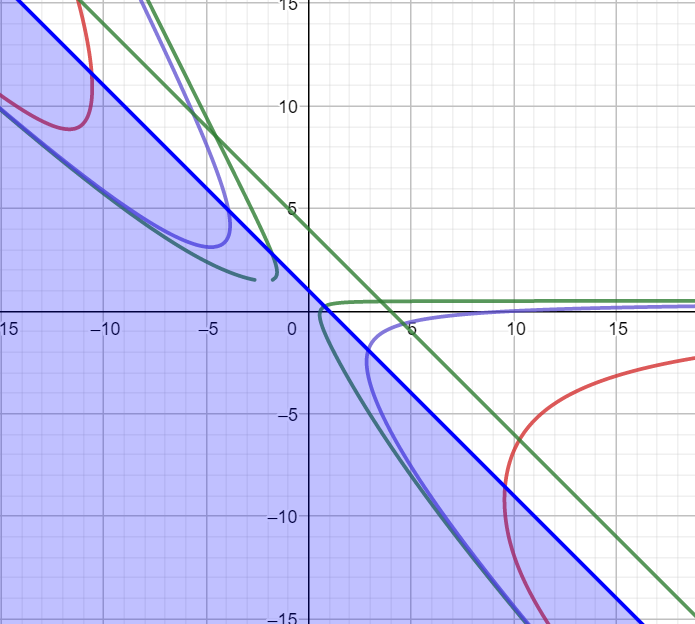
µ = 10

µ = 1

µ = 0.1

µ = 0.01

График



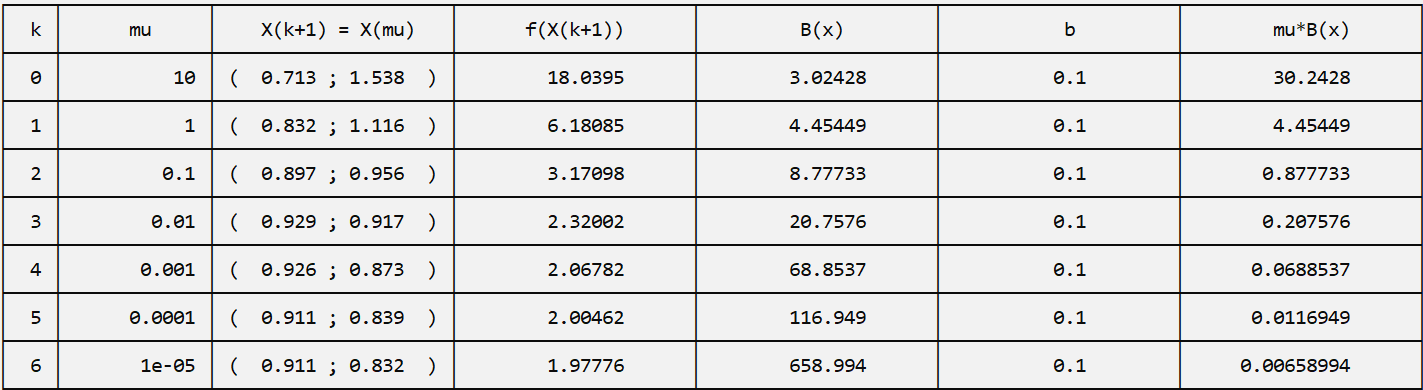
Вывод: как видно из результатов работы программы и графика изначальная задача поставлена некорректно. Линии ограничения параллельные прямые, кроме того, они параллельны ветвям графика функции. То есть линии уровней данной функции двигаются вдоль линий ограничений и никогда не покинут допустимой области.

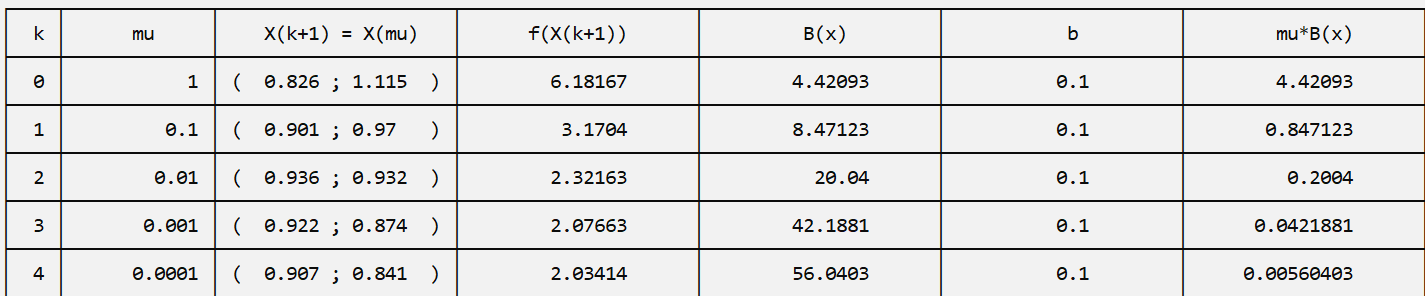
Проверим правильность работы программы на функции, данной для отладки программы и сравним с решением из методички.

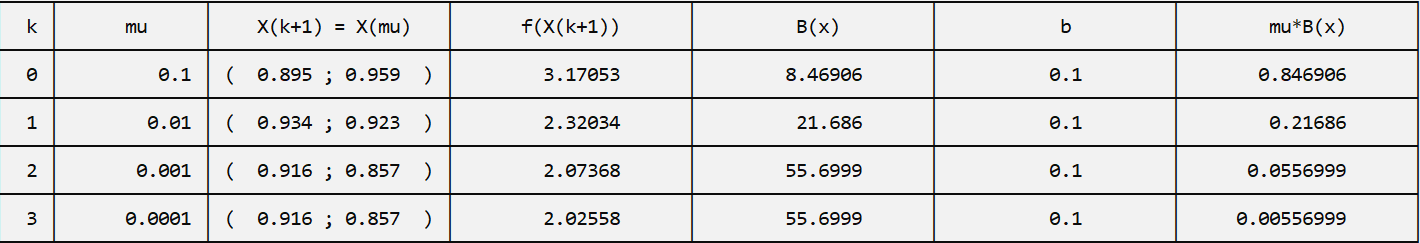
1. Тест для функции из методички F=(x1-4)4+(x1-2x2)2

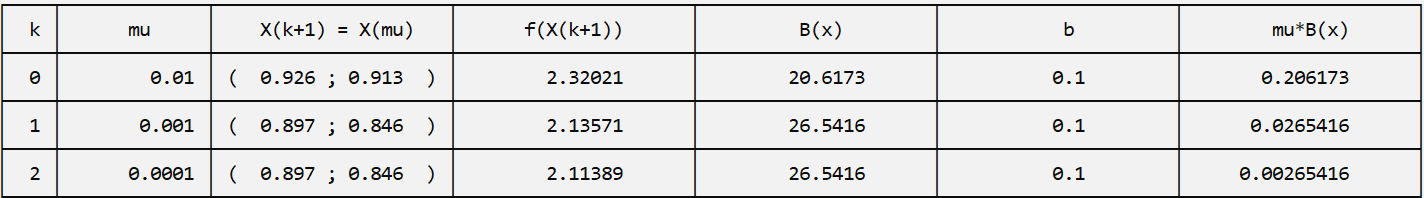
Ограничение: x12-x2<0

Начальная точка x = (1;5) Коэффициент = 0.1

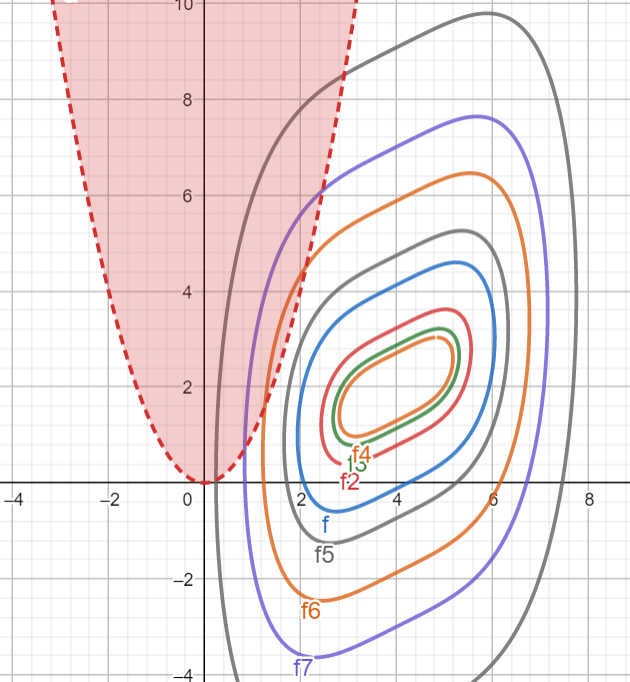
µ = 10

µ=1

µ=0.1

µ=0.01

График



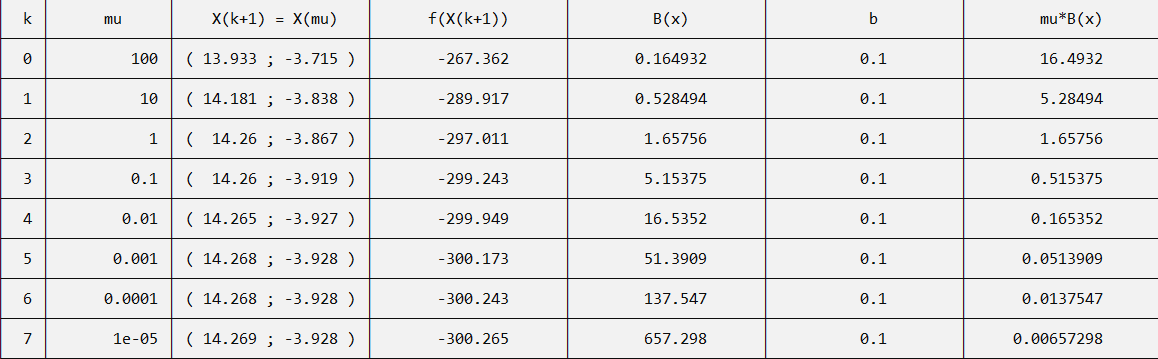
Вывод: программа работает корректно, так как находит точку минимума для функции F=(x1-4)4+(x1-2x2)2, при начальной точкеx = (1;5) из допустимой области. Решение, найденное программой, практически совпадает с решением из методички (F\*=1.977 – по результатам работы программы, F\*=1.96 – значение из методички). Следовательно, ограничения в изначальном задании поставлены некорректно.

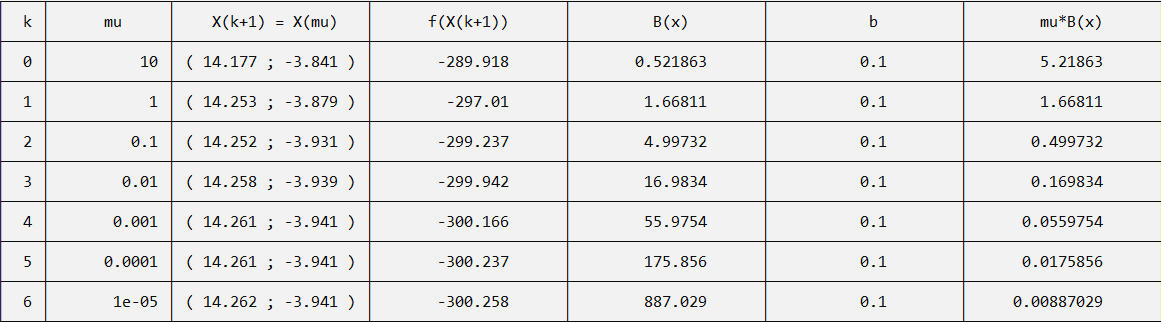
1. Тест для функции с измененными ограничениями

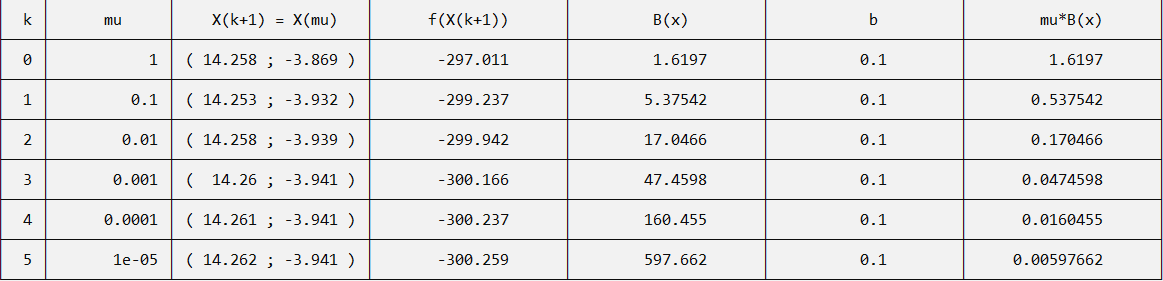
F=-x12+2 x1 x2 + x22 + exp(-x1-x2)

Ограничение: x12-15 \*х1 +x2<=5

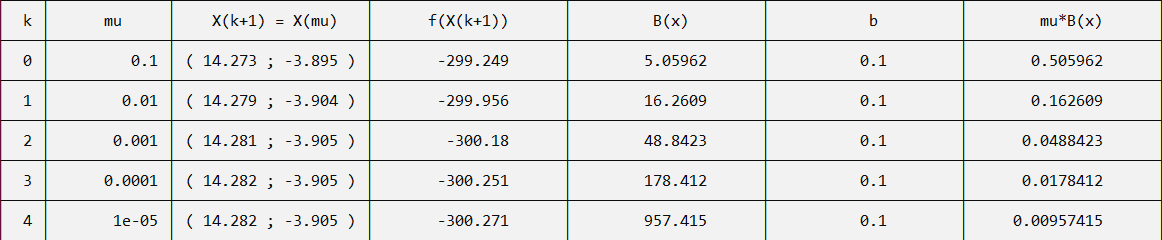
Начальная точка x = (10;5) – точка из допустимой области. Коэффициент = 0.1

µ = 100

µ = 10

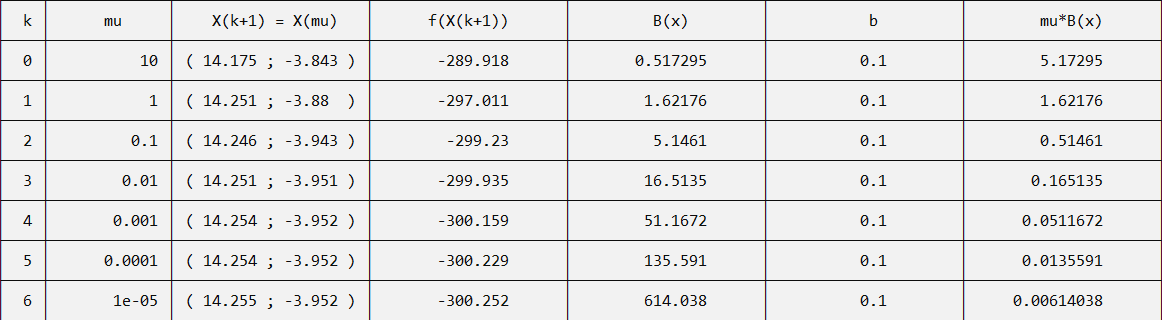
µ=1

µ=0.1

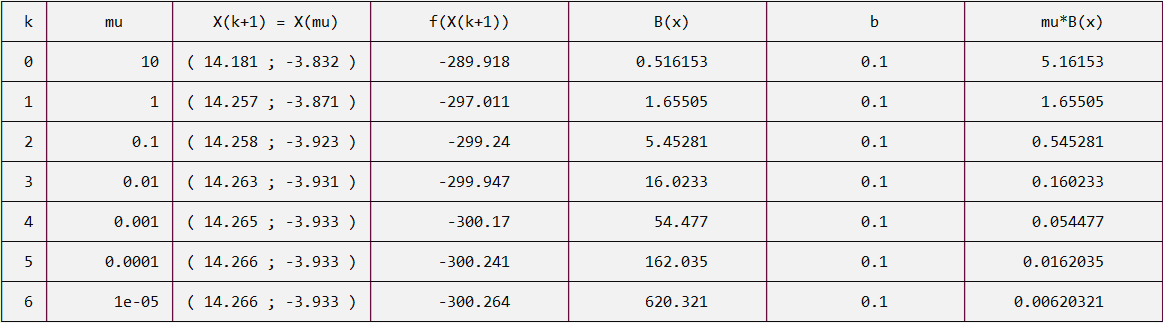


Возьмём другие точки из допустимой области при µ = 10 и = 0.1

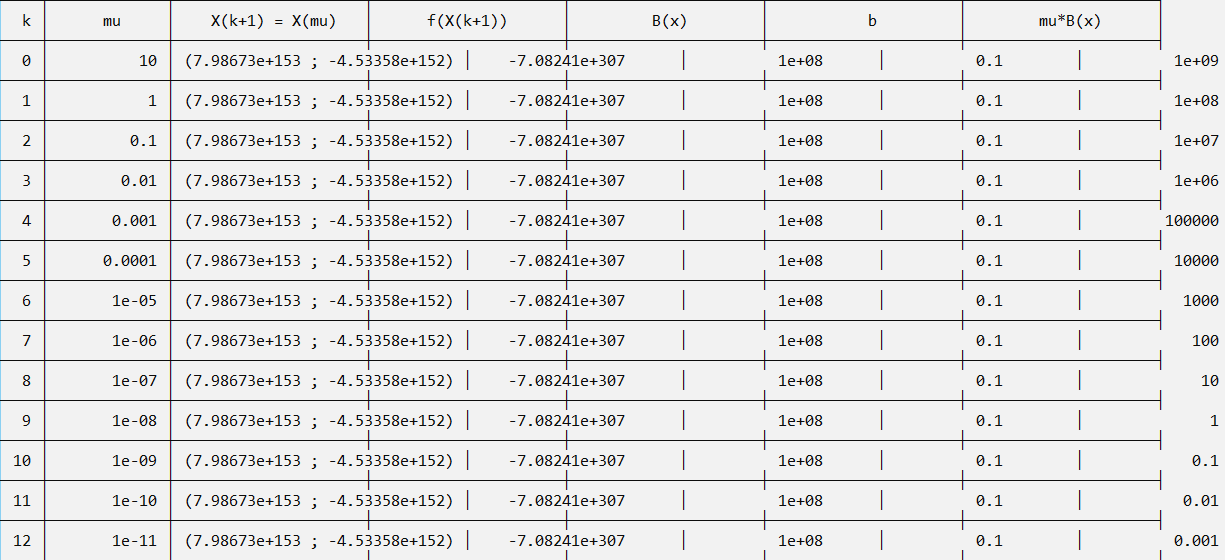
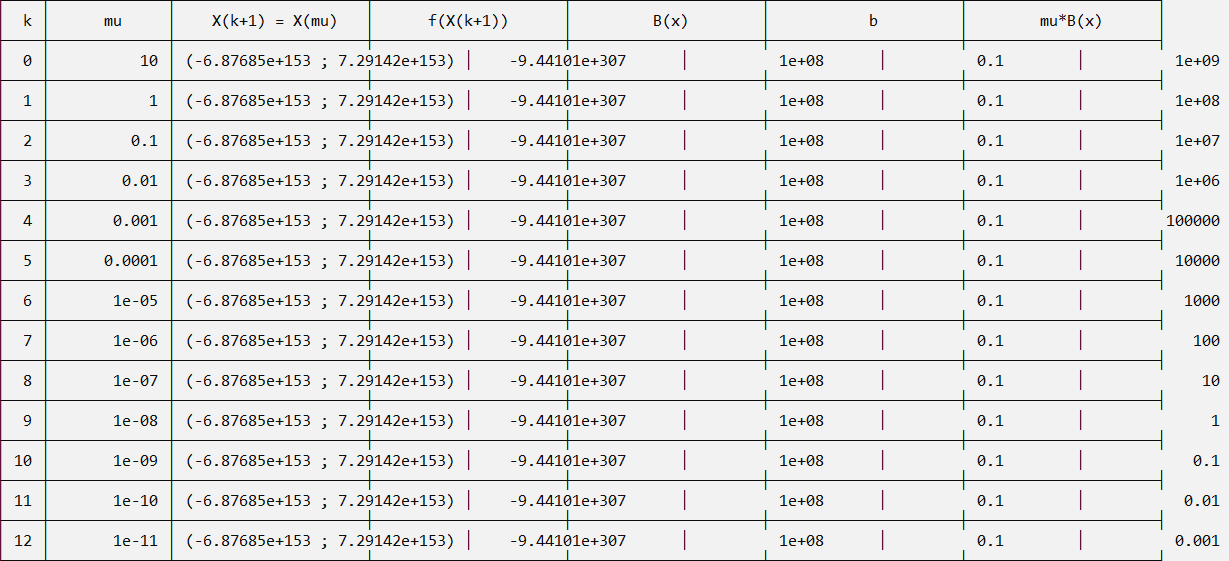
1. Х = (0;0)

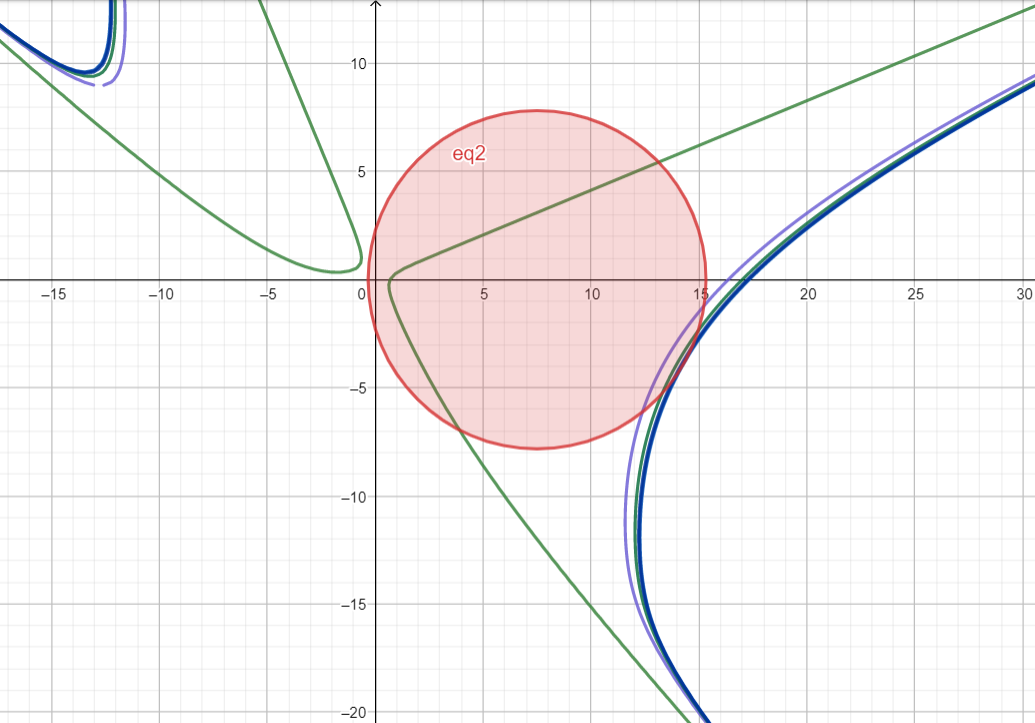


1. Х = (6;-5)



Возьмём точку из недопустимой области

1. Х = (-16;-16)
2. Х = (10;0)

График

**Сводная таблица**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №  п/п | Х0 |  | k | X\* | F(x\*) |
| 1. | (10;5) | 100 | 7 | (14.269;-3.928) | -300.265 |
| 2. | (10;5) | 10 | 6 | (14.262;-3.941) | -300.258 |
| 3. | (10;5) | 1 | 5 | (14.262;-3.941) | -300.259 |
| 4. | (10;5) | 0.1 | 4 | (14.282;-3.905) | -300.271 |
| 5. | (0;0) | 10 | 6 | (14.255;-3.952) | -300.252 |
| 6. | (6;-5) | 10 | 6 | (14.266;-3.933) | -300.264 |
| 7. | (-16;-16) | 10 | 12 | Огромные значения  координат | Огромное отрицательное  значение функции |
| 8. | (10;0) | 10 | 12 |

**Вывод**

В ходе проведения лабораторной работы, мы выяснили, что изначальная задача была поставлена некорректно. Линии ограничения - параллельные прямые, кроме того, они параллельны ветвям графика функции. То есть линии уровней данной функции двигаются вдоль линий ограничений и никогда не покинут допустимую область. Затем проверили правильность работы программы на задаче из методички. Программа находит точку минимума для функции F=(x1-4)4+(x1-2x2)2, при начальной точкеx = (1;5) из допустимой области. Данное решение практически совпадает с решением из методички (F\*=1.977 – по результатам работы программы, F\*=1.96 – значение из методички). После мы изменили начальные условия, а именно поменяли ограничение и провели несколько тестов. В результате этого мы выяснили, что:

1. Количество итераций зависит от начального µ (чем больше µ, тем больше итераций). Например, возьмем строки 1 и 4 из сводной таблицы: при одинаковом значении начальной точки (10;5) но разных µ (100 и 0.1) программа находит точку минимума за разное количество итераций: 7 и 4 соответственно.
2. µ практически не влияет на точность ответа (значения минимума функции в тестах с 1-го по 4-ый различается в сотых)
3. Метод барьеров работает только с точками из допустимой области, в отличии от метода штрафных функций. Тест 7 и 8 из сводной таблицы показывает, что если точка будет взята из недопустимой области, то программа будет выдавать некорректные значения.