МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ АРХИТЕКТУРНО-СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ» (ННГАСУ)**

Международный институт экономики права и менеджмента.

Кафедра прикладной информатики и статистики

**Курсовая работа**

по дисциплине «Исследование операций и методы оптимизации»

Тема: Метод проекции градиента (случай линейных ограничений)

Выполнил: студент 3 курса группы Пиэ 14.14 Турцев М.А.

Проверил: доцент, кандидат физико-математических Прокопенко Н.Ю.

наук

Нижний Новгород

2016 год

Содержание:

1. Введение…………………………………………………………..3
2. Описание методов………………………………………………..4
   1. Метод наискорейшего спуска………………………………..4
   2. Метод золотого сечения………………………………………5
   3. Метод нахождения проекции градиента…………………….5
   4. Метод нахождения шага

(в случае линейных ограничений и вдоль проекции

градиента функции)…………………………………………..6

* 1. Пошаговый алгоритм метода проекции градиента (в случае линейных ограничений)………………………………………………….6

1. Блок-схемы…………………………………………………………7
   1. Блок-схема метода проекции градиента (случай линейных ограничений)………………………………………………….7

3.2.Блок-схема нахождения проекции вектора градиента……8

3.3 Блок-схема метода нахождения шага shag (в случае линейных ограничений и вдоль проекции градиента функции)…………………..9

3.4 Блок-схема нахождения шага lambda по методу золотого сечения..10

4. Результаты вычислений……………………………………………….11

4.1 Поиск минимума функции аналитическим методом………………11

5. График функции………………………………………………………12

6. Результаты программных вычислений с разными точностями и начальными точками……………………………………………………..13

7. Вывод……………………………………………………………………13

8. Список использованной литературы…………………………………...14

Приложение 1………………………………………………………………15

**1.Введение**

В своей деятельности человеку приходится встречаться явно или неявно с оптимизацией. Это обусловлено с тем, какие задачи встают перед человеком, и насколько они сложны. Задачи, которые решаются, например, в экономике при планировании, в управлении, в проектировании объектов промышленности требуют владения определенным математическим аппаратом, позволяющим осуществить поиск наилучшего варианта с точки зрения намеченной цели. При всем многообразии задач оптимизации дать общие методы их решения может только математика, резкое расширение приложений которой связано с появлением ЭВМ, что привело к математизации не только физики, но и химии, биологии, экономики, психологии, медицины — практически всех наук. Суть математизации состоит в построении математических моделей процессов и явлений и в разработке методов их исследования.

Использование математического аппарата при решении задач оптимизации предполагает формулировку интересующей проблемы на языке математики, придание количественных оценок возможным вариантам вместо слов «лучше», «хуже».

Многие задачи оптимизации сводятся к отысканию наименьшего или наибольшего значения некоторой функции, которую принято называть целевой функцией. В этом случае методы исследования существенно зависят от свойств целевой функции и той информации о ней, которая может считаться доступной до решения задачи и в процессе ее решения.

Наиболее просты с математической точки зрения случаи, когда целевая функция является дифференцируемой функцией. В этом случае для исследования ее свойств (участки возрастания и убывания, точки локального экстремума) может быть использована производная. В последние десятилетия в условиях научно-технического прогресса круг задач оптимизации, поставленных практикой, существенно расширился. Во многих из них значения целевой функции могут получаться в результате численных расчетов или браться из эксперимента. Такие задачи являются более сложными, при их решении нельзя исследовать целевую функцию с помощью производной. Это привело к разработке специальных методов, рассчитанных на широкое применение ЭВМ. Следует также иметь в виду, что сложность решения задачи существенно зависит от размерности целевой функции, т.е. от числа ее аргументов.

Целью моей курсовой работы является изучение метода проекции градиента (в случае линейных ограничений и с нахождением шага методом золотого сечения) и его реализация в программном виде.

Из цели необходимо выявить ряд задач:

1. Ознакомится и подробно рассмотреть метод наискорейшего спуска.
2. Ознакомиться и подробно рассмотреть способ поиска шага методом золотого сечения.
3. Ознакомиться и подробно рассмотреть метод проекции градиента (в случае линейных ограничений).
4. Написать программу, позволяющую решать задачу оптимизации, используя данный алгоритм.

**2.Описание методов.**

**2.1Метод наискорейшего спуска.**

Метод наискорейшего спуска является вариантом градиентного спуска. Здесь полагают что, но величина шага из находится в результате решения задачи одномерной оптимизации:

то есть на каждой итерации в направление антиградиентасовершается исчерпывающий спуск.

Алгоритм метода:

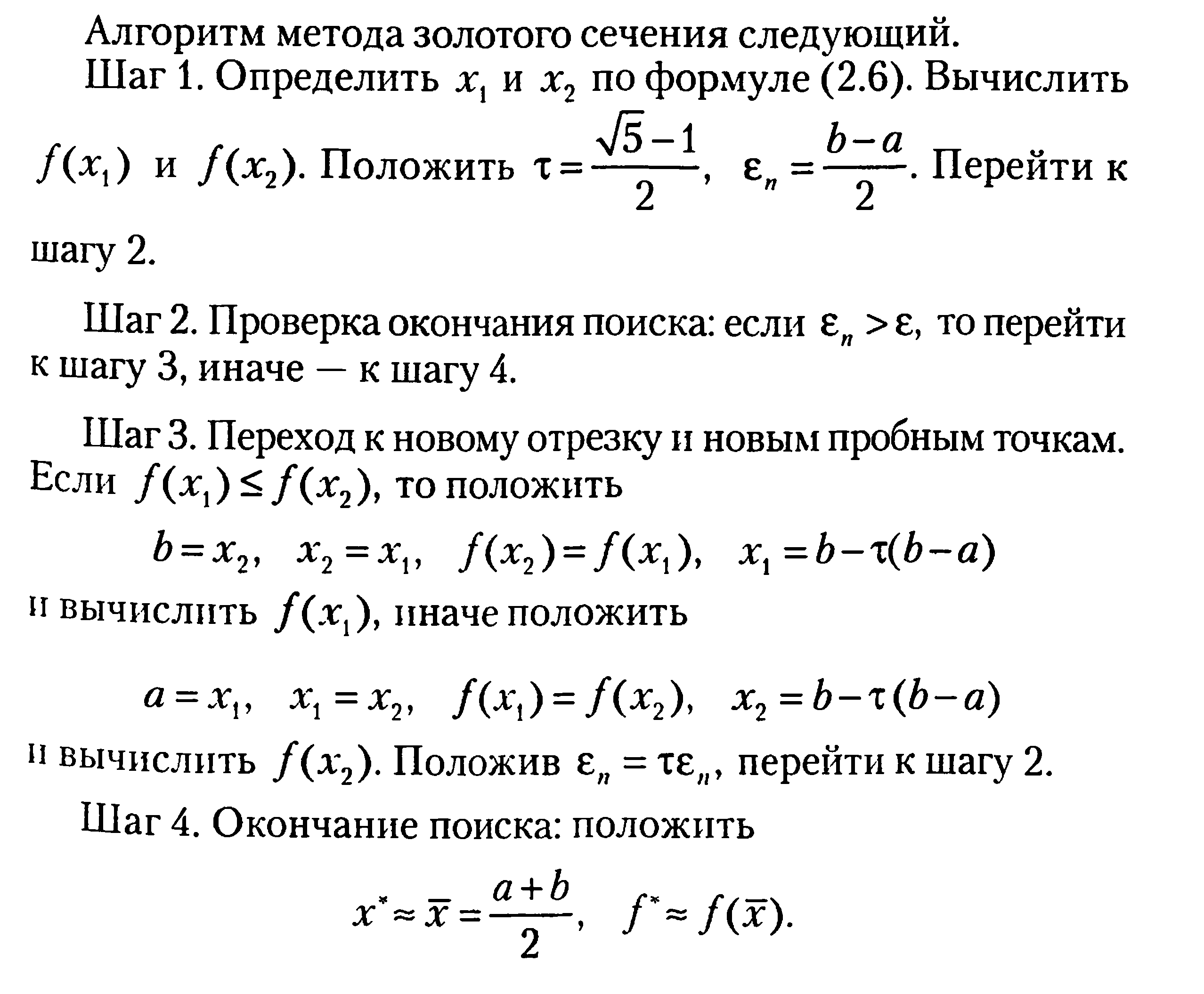
Шаг 1. Задать параметр ε>0, выбрать . Вычислить. Перейти к шагу 2.

Шаг 2. Вычислить и проверить условие достижения точности: . Если оно выполнено, вычисления завершить, полагая Иначе перейти к шагу 3.

Шаг 3. Решить задачу одномерной оптимизации

длято есть найти . Положить и перейти к шагу 2.[[1]](#footnote-1)

**2.2 Метод золотого сечения**

****

**2.3 Метод нахождения проекции градиента:**

Существует 3 случая, когда нужно искать проекцию вектора градиента:

1. Проекция градиента равна антиградиенту функции в случае, когда .
2. Проекция градиента равна максимуму из в случае, если равен своей левой границе ограничения c.
3. Проекция градиента равна минимуму из в случае, если

равен своей правой границе ограничения d.

**2.4. Метод нахождения шага (в случае линейных ограничений и вдоль проекции градиента функции):**

Если проекция градиента не равна 0, то:

1. Ищем шаг 1 по формуле:
2. Ищем шаг 2 по формуле:

Искомый шаг shag равен минимуму из и .

**2.5.Пошаговый алгоритм метода проекции градиента (в случае линейных ограничений):**

1. Если начальное приближение не принадлежит границам, то выводим сообщение о том, что начальная точка находится вне границ, и останавливаемся. Иначе переходим к шагу 2
2. Запоминаем координаты точки. Переходим к шагу 3.
3. Вычисляем градиент функции
4. Вычисляем проекцию градиента согласно алгоритму метода нахождения проекции градиента.
5. Проверяем условие: если проекция точки по осям не равна 0, то переходим к шагу 6, иначе останавливаемся и выводим результат.
6. Вычисляем шаг вдоль проекции градиента shag
7. Вычисляем шаг по методу золотого сечения в отрезке от shag до 0.
8. Вычисляем следующее приближение с шагом, вычисленным по методу золотого сечения.
9. Проверяем критерий остановки: если , то останавливаемся и выводим результат. Иначе переходим на шаг 2.

**3. Блок-схемы**

**3.1 Блок-схема метода проекции градиента (случай линейных ограничений)**

Вычисляем

Ввод , , c, dd, q, w

Вычисляем

||

Вычисляем shag

Вычисляем lambda в [0;shag]

Вывод X

да

нет

да

нет

**3.2Блок-схема нахождения проекции вектора градиента:**

(

, c, d

нет

да

нет

да

да

да

да

нет

нет

Вывод

**3.3 Блок-схема метода нахождения шага shag (в случае линейных ограничений и вдоль проекции градиента функции)**

,,

да

нет

Вывод shag

shag=

shag=

**3.4 Блок-схема нахождения шага lambda по методу золотого сечения:**

Ввод a=0,b=shag,

b=x2

a=x1

|b-a|≤

A<B

x1=a+0,382\*(b-a)

x2=a+0,618\*(b-a)

A=f(x1); B=f(x2)

нет

да

|b-a|≤

да

нет

да

нет

x2=x1;

x1=a+0,618\*(b-a)

A=f(x1)

lambda=(a+b)/2

x1=x2;

x2=a+0,618\*(b-a);

B=f(x2)

Вывод lambda

**4.Результаты вычислений:**

Для примера возьмем функцию:

**4.1Поиск минимума функции аналитическим методом.**

Найдём градиент данной функции, для этого вычислим частные производные по x и y:

Градиент функции будет равен:

Далее составим систему уравнений и найдем стационарные точки:

Получаем точки А(1; 1) и B(0,0) подозрительные на экстремум.

Проверим точку А(1; 1) . Для этого необходимо построить матрицу вторых частных производных:

Следовательно, точка A(1;1) – точка минимума функции

Проверим точку B(0; 0) . Для этого необходимо построить матрицу вторых частных производных:

Следовательно, точка B(0;0) – точка локального максимума функции.

**5. График функции**

**6.Результаты программных вычислений с разными точностями и начальными точками.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Точка :** | **Точность** | **Число итераций N** | **Точка экстремума** | **Отклонение**  **X-** |
| 1 | (5; 5) | 0.0001 | 2 | (1; 1) | (0.0; 0.0) |
| 2 | (9; 9) | 0.001 | 10 | (1; 1) | (0.0;  0.0) |
| 3 | (4; 7) | 0.02 | 5 | (0,998753;  1,00034) | (0.001247;  0.00034) |
| 4 | (0; 4) | 0.001 | 5 | (0.99181; 0,994283) | (0.00819; 0.005717) |
| 5 | (0; 1) | 0.000001 | 3 | (1; 1) | (0.0; 0.0) |
| 6 | (4; 9) | 0.001 | 11 | (0.999383; 1.00283) | (0.000617; -0,00283) |
| 7 | (8; 8) | 0.001 | 5 | (0.999999; 0.999999) | (0.000001; 0.000001) |

**7.Вывод.**

Градиентные методы сходятся к минимуму с высокой скоростью (со скоростью геометрической прогрессии) для гладких выпуклых функций.

Однако на практике, как правило, минимизируемые функции имеют плохо обусловленные матрицы Гессе. Значения таких функций вдоль некоторых направлений изменяются гораздо быстрее (не редко на несколько порядков), чем в других направлениях. Их поверхности уровня в простейшем случае сильно вытягиваются, а в более сложных случаях изгибаются и представляют собой овраги. Функции, обладающие такими свойствами, называют овражными*.* Направление антиградиента этих функций существенно отклоняется от направления в точку минимума, что приводит к замедлению скорости сходимости.

Скорость сходимости градиентных методов также существенно зависит от того, как точно вычислен градиент. Потеря точности, а это обычно происходит в окрестности точек минимума или в овражной ситуации, может вообще нарушить сходимость процесса градиентного спуска. Вследствие перечисленных причин градиентные методы зачастую используются в комбинации с другими, более эффективными методами на начальной стадии решения задачи. В этом случае точка находится далеко от точки минимума, и шаги в направлении антиградиента позволяют достичь существенного убывания функции.

**8.Список использованной литературы:**

1. Методы оптимизации : учеб.пособие для вузов / В. А. Гончаров. — М.: Издательство Юрайт ; ИД Юрайт, 2014. — 191 с. — Серия : Бакалавр. Базовый курс.
2. М.Базара, К.Шетти. Нелинейное программирование. Теория и алгоритмы: Пер. с англ. – М.:Мир,1982.583 с

**Приложение 1.**

Программный код, выполненный на языке С++.

//Программа демонстрирует поиск минимума функции нескольких переменных методом проекции градиента

#include <iostream>

#include <cmath>

#include<iomanip>

using namespace std;

//Структура вектор

//Содержит количество переменных исходной функции

struct vector

{

double x, y;

};

//Исходная функция

double fx(vector x)

{

return pow(x.x, 3) + pow(x.y, 3) - 3 \* x.x\*x.y;

}

//Градиент исходной функции

//Также для нахождения градиента можно использовать численные методы

vector gradient(vector x)

{

vector grad;

grad.x = 3 \* pow(x.x, 2) - 3 \* x.y;

grad.y = 3 \* pow(x.y, 2) - 3 \* x.x;

return grad;

}

//Вычисление одномерной функции для нахождения шага методом золотого сечения

double MakeSimplefx(double x, vector grad, vector xj)

{

vector buffer;

buffer.x = xj.x - x\*grad.x;

buffer.y = xj.y - x\*grad.y;

return fx(buffer);

}

//Метод золотого сечения для нахождения шага (lambda)

double GoldenSelection(double a, double b, double eps, vector gradient, vector x)

{

//const double fi = 1.6180339887;

double x1, x2;

double y1, y2;

x1 = a + ((b - a)\*0.382);

x2 = a + ((b - a)\*0.618);

y1 = MakeSimplefx(x1, gradient, x);

y2 = MakeSimplefx(x2, gradient, x);

while (std::abs(b - a)>eps)

{

if (y1 <= y2)

{

b = x2;

x2 = x1;

x1 = a + ((b - a)\*0.382);

y2 = y1;

y1 = MakeSimplefx(x1, gradient, x);

}

else

{

a = x1;

x1 = x2;

x2 = a + ((b - a)\*0.618);

y1 = y2;

y2 = MakeSimplefx(x2, gradient, x);

}

}

return (a + b) / 2;

}

//Функция вычисления нового приближения

vector Calculate(vector x, vector gradient, double lambda)

{

vector buffer;

buffer.x = x.x - lambda\*gradient.x;

buffer.y = x.y - lambda\*gradient.y;

return buffer;

}

// Проекция вектора градиента

vector Proection(vector x, vector current, vector gradient, double c, double d, double q, double w)

{

// c - левая граница для x1; d-правая граница для x1;

// q - левая граница для x2; w-правая граница для x2;

vector proection;

proection.x = 0;

proection.y = 0;

if ((current.x > c) && (current.x < d))

{

proection.x = -gradient.x;

}

if ((current.y > q) && (current.y < w))

{

proection.y = -gradient.y;

}

if (current.y == q){

if (-gradient.y > 0)

{

proection.y = -gradient.y;

}

else proection.y = 0;

}

if (current.y == w){

if (-gradient.y > 0)

{

proection.y = 0;

}

else proection.y = -gradient.y;

}

if (current.x == c)

{

if (-gradient.x > 0)

{

proection.x = -gradient.x;

}

else proection.x = 0;

}

if (current.x == d)

{

if (-gradient.x > 0)

{

proection.x = 0;

}

else proection.x = -gradient.x;

}

return proection;//возвращаем результат

}

double Step(vector proection, vector x, double c, double d, double q, double w)

{

double step1, step2, step;

if (proection.x != 0)

{

step1 = (c - x.x) / proection.x;

step2 = (d - x.x) / proection.x;

if (step1 < step2)

{

step = step1;

}

else step = step2;

}

if (proection.y != 0)

{

step1 = (q - x.y) / proection.y;

step2 = (w - x.y) / proection.y;

if (step1 < step2)

{

step = step1;

}

else step = step2;

}

return step;

}

//Метод наискорейшего спуска

vector GradDown(vector x, double eps, double c, double d, double q, double w)

{

vector current = x;

vector last;

double itterations = 0;

do

{

//double c, d, q, w;

last = current; //Запоминаем предыдущее значение

vector grad = gradient(current); //Вычисляем градиент

vector pr = Proection(x, current, gradient(current), c, d, q, w);// находим прекцию вектора градиента

if ((pr.x != 0) && (pr.y != 0))

{

double shag = Step(pr, x, c, d, q, w);// находим шаг вычислений для того, чтоб не выйти за ограничения по x и по y

double lambda = GoldenSelection(0, shag, eps, pr, current); //Находим шаг вычислений методом золотого сечения

current = Calculate(current, pr, lambda); //Вычисляем новое приближение

itterations = itterations + 1;

}

else cout << "Начальное приближене не лежит в заданной области ограничений";

} while (abs(fx(current) - fx(last))>eps);//Проверяем условие

cout << "Разница между предыдущим и последующим значением функции: " << abs(fx(current) - fx(last)) << endl;

cout << "Число итераций= " << itterations << endl;

return current; //Возвращаем результат

}

//Тело главной функции

int main()

{

setlocale(LC\_ALL, "Russian");

vector x;

double eps;

double c, d, q, w;

cout << "Введите через пробел начальное приближение x и y (например: -1 1): ";

cin >> x.x >> x.y;

cout << "\nВведите точность вычислений (например 0.000001): ";

cin >> eps;

cout << endl;

cout << "\nВведите через пробел ограничения для x (например -5 5): ";

cin >> c >> d;

cout << std::endl;

cout << "\nВведите через пробел ограничения для y (например -5 5): ";

cin >> q >> w;

cout << endl;

vector result = GradDown(x, eps, c, d, q, w);

std::cout << "\nРезультат: x = " << result.x << " y = " << result.y;

cin.get();

cin.get();

return 0;

};

1. [↑](#footnote-ref-1)