**Постановка задачи**

Минимизировать многомерные функции  (функция Химмельблау) и  (функция Розенброка) методом сопряженных направлений Пауэлла, использовать для одномерной минимизации метод последовательной квадратичной аппроксимации Пауэлла, условием останова для метода считать , при 0.001 и 0.00001. (100 итераций)

**Основные требования к программной системе:**

* язык программирования С, C++,
* все данные вводятся непосредственно в текст программы;
* необходимо предусмотреть возможность вывода результатов на экран в двух режимах:
	+ вывод только итоговых результатов решения задачи(оценки оптимальной точки, значения функции в оптимальной точке, число вычислений функции),
	+ развернутый вывод – результаты на итерациях многомерного метода обязательно с оценками текущих точек, градиентов, векторов направлений, шагов одномерной оптимизации, результатов проверки критерия останова.

**Пример решения:**

Минимизировать многомерные функции 1 и 2 (функции Химмельблау) методом сопряженных направлений Пауэлла, использовать для одномерной минимизации метод золотого сечения, условием останова для метода считать , при 0.001 и 0.0001

// Optim\_Bulychev.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <cmath>

#include <iostream>

#include <vector>

#include <cmath> // для round

using namespace std;

void preSearch(double x0,double h,double &a,double &b, vector <double> x, vector <double> s);

double goldenRatio(double a, double b,vector <double> x, vector <double> s);

double linF(vector <double> x, vector <double> s, double lambda);

double f(vector <double> x);

double oneDimentionalSearch(vector <double> x, vector <double> s);

vector <double> Pauell (vector <double> x0);

int selectedFunction = 0;

bool showDetails = 0;

int \_tmain(int argc, \_TCHAR\* argv[])

{

 // русский ¤зык

 setlocale(LC\_ALL, "Russian");

 puts ("Выберите функцию Химмельблау (нажмите 1 или 2)");

 scanf("%d", &selectedFunction);

 puts ("Показать подробный отчет? (0 - нет, 1 - да)");

 scanf("%d", &showDetails);

 vector <double> x0;//зададим начальную точку, поскольку обе функции от 2 переменных, обойдемся одной на обе

 x0.push\_back(11);

 x0.push\_back(5);

 x0 = Pauell(x0);

 cout << "Ответ: в точке минимума " << x0[0] << " " << x0[1] << " функция принимает значение " << f(x0);

 return 0;

}

double goldenRatio(double a, double b,vector <double> x, vector <double> s)//поиск минимума методом золотого сечения

{

 double eps=0.001;

 double t = (sqrt(5.0) + 1) / 2;

 double x1 = a + (b - a) / pow(t, 2);

 double x2 = a + (b - a) / t;

 while((b - a) / 2 > eps)

 if(linF(x,s,x1)<=linF(x,s,x2))

 {

 b=x2;

 x2=x1;

 x1=a+b-x1;

 }

 else

 {

 a=x1;

 x1=x2;

 x2=a+b-x1;

 }

 return (((a + b) / 2)) ;

}

void preSearch(double x0,double h,double &a,double &b, vector <double> x, vector <double> s)//поиск отрезка локализации минимума

{

 int k=1;//номер итерации

 if(linF(x,s,x0)>linF(x,s,x0+h))

 {

 a=x0;

 x0=x0+h;

 k=2;

 }

 else if(linF(x,s,x0-h)>=linF(x,s,x0))

 {

 a=x0-h;

 b=x0+h;

 }

 else

 {

 b=x0;

 x0=x0-h;

 h=-h;

 k=2;

 }

 //шаг 4

 while (linF(x,s,x0)>linF(x,s,x0+ pow(2.0,(k-1))\*h))//пока функция на нашем пути убывает, идем увеличивая шаг и двигая границу

 {

 if(h>0)

 a=x0+ pow(2.0,(k-2))\*h;

 else

 b=x0+ pow(2.0,(k-2))\*h;

 k++;

 }

 if(h>0)//перестала убывать на нашем пути

 b=x0+ pow(2.0,(k-1))\*h;

 else

 a=x0+ pow(2.0,(k-1))\*h;

}

double linF(vector <double> x, vector <double> s, double lambda)//линейная функция из метода СН Пауэлла

{

 vector <double> tmp;

 tmp.push\_back(x[0] + s[0]\*lambda);

 tmp.push\_back(x[1] + s[1]\*lambda);

 return f(tmp);

}

double f(vector <double> x)//основная функция

{

 if(selectedFunction==1)

 return 4\* pow((x[0]-5),2) + pow((x[1]-6),2);

 if(selectedFunction==2)

 return pow((pow(x[0],2)+x[1]-11.0),2) + pow((pow(x[1],2)+x[0]-7.0),2);

}

double oneDimentionalSearch(vector <double> x, vector <double> s)//возвращает аргумент минимизации одномерной функции linF при заданных х и s

{

 double a=6,b=4;

 preSearch(1,0.1,a,b,x,s);

 return goldenRatio(a,b,x,s);

}

vector <double> Pauell (vector <double> x0)//метод сопряженных направлений пауэлла

{

 double e = 0.0001;//погрешность

 int count = 0;//счетчик

 vector <double> res;//ответ будет здесь

 vector<vector <double>> x;//вектор точек

 vector <double> s1;//вектора направлений

 vector <double> s2;

 s1.push\_back(1);

 s1.push\_back(0);

 s2.push\_back(0);

 s2.push\_back(1);

 x.push\_back(x0);//добавим начальную точку в вектор всех точек

 if (showDetails) printf ("Начальная точка %.4f %.4f \n", x[0][0], x[0][1]);

 double lambda = 0;

 while(count < 100 && (x.size()==1 || abs(f(x[x.size()-1]) - f(x[x.size()-2]))>e))//критерий останова: если мы уже никуда не движемся, то достаточно итераций, 100 итераций всегда достаточно

 {

 if (showDetails) printf ("Итерация %d \n", count+1);

 if (showDetails) printf ("--Шаг в направлении %.4f %.4f \n", s2[0],s2[1]);

 lambda =oneDimentionalSearch(x[count\*3], s2);//идем из точки по направлению, ищем минимум и принимаем минимум за следующую начальную точку

 x0[0]= x[count\*3][0] + lambda\*s2[0];

 x0[1]= x[count\*3][1] + lambda\*s2[1];

 x.push\_back(x0);

 if (showDetails) printf ("--Новая точка %.4f %.4f. ", x0[0],x0[1]);

 if (showDetails) printf ("--Значение функции в ней %.4f \n", f(x0));

 if (showDetails) printf ("--Шаг в направлении %.4f %.4f \n", s1[0],s1[1]);

 lambda =oneDimentionalSearch(x[count\*3+1], s1);//идем из точки по направлению, ищем минимум и принимаем минимум за следующую начальную точку

 x0[0]= x[count\*3+1][0] + lambda\*s1[0];

 x0[1]= x[count\*3+1][1] + lambda\*s1[1];

 x.push\_back(x0);

 if (showDetails) printf ("--Новая точка %.4f %.4f. ", x0[0],x0[1]);

 if (showDetails) printf ("--Значение функции в ней %.4f \n", f(x0));

 if (showDetails) printf ("--Шаг в направлении %.4f %.4f \n", s2[0],s2[1]);

 lambda =oneDimentionalSearch(x[count\*3+2], s2);//идем из точки по направлению, ищем минимум и принимаем минимум за следующую начальную точку

 x0[0]= x[count\*3+2][0] + lambda\*s2[0];

 x0[1]= x[count\*3+2][1] + lambda\*s2[1];

 x.push\_back(x0);

 if (showDetails) printf ("--Новая точка %.4f %.4f. ", x0[0],x0[1]);

 if (showDetails) printf ("--Значение функции в ней %.4f \n", f(x0));

 s1[0]=s2[0];//заменяем направления по правилу

 s1[1]=s2[1];

 s2[0]= x[count\*3+3][0]-x[count\*3+1][0];

 s2[1]= x[count\*3+3][1]-x[count\*3+1][1];

 count++;

 }

 lambda =oneDimentionalSearch(x[x.size()-1], s2);//делаем поиск по сопряженному направлению

 res.push\_back(x[x.size()-1][0]+lambda\*s2[0]);

 res.push\_back(x[x.size()-1][1]+lambda\*s2[1]);

 return res;

}

**Численные исследования**

Тестирование проходило на функциях

1.функция Химмельблау №1 

2.функция Химмельблау №2 

с 2 значениями критерия остановки на 4 точках

**Сводная таблица**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Ф1 Е=0.001 | Ф1 Е=0.0001 | Ф2 Е=0.001 | Ф2 Е=0.0001 |
| X1 (3,-2) | 4.9983 5.9992 | 4.9997 5.9997 | 3.5841 -1.8479 | 3.5845 -1.8481 |
| X2 (2,3) | 4.9955 5.9992 | 4.9994 5.9995 | 2.9981 2.0000 | 2.9999 2.0000 |
| X3 (5,8) | 4.9994 5.9989 | 4.9999 5.9999 | 2.9985 2.0002 | 2.9996 2.0000 |
| X4 (40, 17) | 4.9993 5.9983 | 4.9998 5.9997 | 3.5848 -1.8493 | 3.5844 -1.8481 |