МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Методы поиска безусловного экстремума

Отчет по лабораторной работе №2

02.03.02 Фундаментальная информатика   
и информационные технологии

Выполнил: студент 3 курса \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Савин Д.А.

Проверил: к. б. н., доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чупраков П. Г.

Киров

2020 г.

**Цель работы**

Изучение методов поиска безусловного экстремума, применение этих методов на практике (реализация методов для трех различных функций в какой-либо среде программирования, а также проверка при помощи математических пакетов) и сравнение методов по их эффективности.

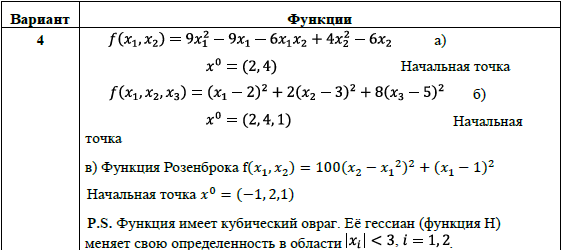
**Задание**

1. Найти локальные минимумы трех функций (по вариантам, см. таблицу 1) с использованием следующих методов: метод нулевого порядка (любой) и метод ненулевого порядка (по вариантам, см. таблицу 2).

2. Дать геометрическую интерпретацию решения для двумерных функций.

3. Проверить решения с помощью математических пакетов.

Функции:



**Листинг программы**

Метод конфигураций:

public class Matrix

{

private double[,] data;

private double[,] getMinor(double[,] matrix, int x, int y) //Метод возвращает матрицу для определения детерминанта без строки x и столбца y

{

int n = (int)Math.Sqrt(matrix.Length);

double[,] minorMatrix = new double[n - 1, n - 1];

for (int i = 0, ln = 0; i < n; i++)

{

for (int j = 0, cm = 0; j < n; j++)

{

if (!((i == y) || (j == x)))

{

minorMatrix[ln, cm] = matrix[i, j];

cm++;

if (cm == n - 1)

{

ln++;

}

}

}

}

return minorMatrix;

}

private double getDeterminant(double[,] matrix) //Метод возвращает определитель указанной матрицы

{

int n = (int)Math.Sqrt(matrix.Length);

if (n == 1) return matrix[0, 0]; //Вернуть единственный элемент матрицы

double result = 0;

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

{

if ((i % 2) == 0)

{

result += matrix[0, i] \* getDeterminant(getMinor(matrix, i, 0));

}

else

{

result -= matrix[0, i] \* getDeterminant(getMinor(matrix, i, 0));

}

}

return result;

}

public double getDeterminant()

{

return getDeterminant(data);

}

public Matrix(int n, int m)

{

data = new double[n, m];

}

public double this[int i, int j]

{

get

{

return data[i, j];

}

set

{

data[i, j] = value;

}

}

public double[,] getData() { return data; }

}

public class Function

{

private double[] x;

public Matrix H;

private int N;

public int getN()

{

return N;

}

public double getX(int i)

{

return x[i];

}

public void Set\_X(int i, double x)

{

this.x[i] = x;

}

public Function(double[] x)

{

this.x = x;

N = x.Length;

H = new Matrix(N, N);

BuildH();

}

public double Func()

{

return 9 \* x[0] \* x[0] - 9 \* x[0] - 6 \* x[0] \* x[1] + 4 \* x[1] \* x[1] - 6 \* x[1];

}

public double GetH()

{

BuildH();

return H.getDeterminant();

}

void BuildH()

{

H = new Matrix(N, N);

H[0, 0] = 9;

H[0, 1] = -3;

H[1, 1] = 4;

H[1, 0] = -3;

}

public Matrix matF()

{

Matrix y = new Matrix(N, 1);

y[0, 0] = 18 \* x[0] - 6 \* x[1] - 9;

y[1, 0] = -6 \* x[0] + 8 \* x[1] - 6;

return y;

}

public Matrix GetMatrix()

{

BuildH();

return H;

}

}

public static void Method0(Function function, double y)

{

int n = function.getN();

int k = 0;

k++;

double f0 = function.Func();

while (y > eps)

{

double f1 = function.Func(), f2 = 0, f3 = 0;

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

{

k++;

function.Set\_X(i, function.getX(i) + y);

f2 = function.Func();

if (f2 > f1)

{

k++;

function.Set\_X(i, function.getX(i) - y \* 2);

f3 = function.Func();

if (f3 > f1 || f2 == f3)

{

function.Set\_X(i, function.getX(i) + y);

}

}

k++;

}

if (f0 == f1)

y = y / 2;

f0 = f1;

}

string s = "Экстремум: " + Math.Round(function.Func(), (int)Math.Round(Math.Log(eps, 0.1))) + " в точке [" + Math.Round(function.getX(0), (int)Math.Round(Math.Log(eps, 0.1)));

for (int i = 1; i < n; i++)

s += " ; " + Math.Round(function.getX(i), (int)Math.Round(Math.Log(eps, 0.1)));

s += "] (Количество итераций: " + k + ")";

Console.WriteLine(s);

}

Основная программа:

static void Main(string[] args)

{

Console.WriteLine("Метод конфигурации: ");

double[] a1 = { 2, 4 };

func funct1 = new func();

funct1.SetFunction(new Func1(a1));

Method0(funct1, 0.001, 2);

double[] a2 = { 2, 4, 1 };

func funct2 = new func();

funct1.SetFunction(new Func2(a2));

Method0(funct2, 0.001, 2);

double[] a3 = { 0, 0, 0, 0 };

func funct3 = new func();

funct3.SetFunction(new Func3(a3));

Method0(funct3, 0.001, 2);

}

Метод градиентного спуска:

using System;

using System.Collections.Generic;

namespace Gradient

{

class Program

{

public static int Iterations = 1;

//константа для метода градиентного спуска с постоянным шагом

//начальное значение eps для метода с дроблением шага

const double LAMBDA\_CONSTANT = 0.01;

//максимально возможное число итераций

const int NUMBER\_OF\_ITERATIONS = 100000;

//eps для критерия останова

const double EPS = 0.0001;

public static double f(List<double> x)

{

return 9 \* x[0] \* x[0] - 9 \* x[0] - 6 \* x[0] \* x[1] + 4 \* x[1] \* x[1] - 6 \* x[1];

//return Math.Pow((x[0]-2), 2)+2\*Math.Pow(x[1]-3,2)+8\*Math.Pow(x[2]-5,2);

//return 100\*Math.Pow(x[1]-x[0]\*x[0],2)+Math.Pow(x[0]-1,2);

}

public static List<double> GradientF(List<double> x)

//градиент исследуемой функции

{

List<double> tmp = new List<double>();

tmp.Add(18 \* x[0] - 6 \* x[1] - 9);

tmp.Add(-6 \* x[0] + 8 \* x[1] - 6);

/\*tmp.Add(2 \* (x[0] -2));

tmp.Add(4 \* (x[1] - 3));

tmp.Add(16 \* (x[2] - 5));\*/

/\*tmp.Add(2 \* (200\*x[0]\*x[0]\*x[0] -200\*x[0]\*x[1]+x[0]-1));

tmp.Add(200 \* (x[1] - x[0]\*x[0]));\*/

return tmp;

}

public static List<double> GradientDescent(List<double> x0)

//minimizes N-dimensional function f; x0 - start point

{

List<double> cur\_x = new List<double>(x0);

List<double> old;

List<double> gr=new List<double>();

double s;

double Lambda= LAMBDA\_CONSTANT;

for (Iterations = 1; Iterations <= NUMBER\_OF\_ITERATIONS; Iterations++)

{

//save old value

old = new List<double>(cur\_x);

//compute gradient

gr = GradientF(cur\_x);

//вычисляем новое значение

for (int j = 0; j < old.Count; j++)

{

cur\_x[j] = cur\_x[j] - Lambda \* gr[j];

}

s = Math.Abs(f(cur\_x) - f(old));

if (s < EPS)

{

return cur\_x;

}

}

return cur\_x;

}

static void Main(string[] args)

{

List<double> x = new List<double>();

x.Add(2);

x.Add(4);

//x.Add(1);

//x.Add(-1);

//x.Add(2);

//x.Add(1);

List<double> ans = GradientDescent(x);

Console.WriteLine("" + f(ans));

Console.Write("Point: ");

for (int i = 0; i < ans.Count; i++)

{

Console.WriteLine(ans[i]+" ");

}

Console.WriteLine("Количество итераций: " + Iterations);

}

}

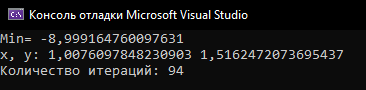
}

**Результаты выполнения программы**

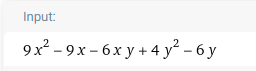
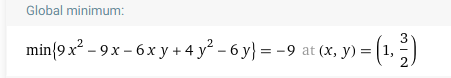
Метод конфигураций:

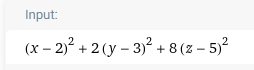


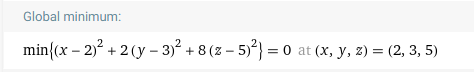
Метод градиентного спуска:

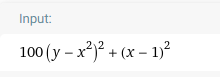


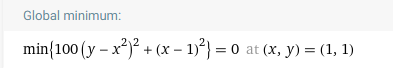
**Проверка вычислений в математическом пакете**









**Выводы**

Были найдены минимумы функций разными способами, методом конфигурации и методом градиентного спуска. Результат проверен математическим пакетом wolframalpha. Метод второго порядка оказался гораздо быстрее. Методы нулевого порядка менее эффективны на сложных функциях, т. к. сдвиг переменной изменяется, постоянно уменьшаясь в какое-то количество раз, а не менялось динамически в зависимости от производный, из-за этого количество итераций очень велико.