Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №12**

Вариант 15

Преподаватель \_\_\_\_\_\_\_\_\_\_\_ В.В. Тынченко

подпись, дата инициалы, фамилия

Студент КИ15–16Б, 031510065 \_\_\_\_\_\_\_\_\_\_\_ Т.В. Радионов

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2018

# Задача лабораторной работы

Разработать программу, реализующую метод Марквардта.

Найти безусловный экстремум функции, выбранной в соответствии с заданием, с использованием разработанной программы.

# Листинг программы

Point.cs

using System;

namespace ConsoleApp1

{

class Point

{

public double X { get; set; }

public double Y { get; set; }

/// <summary>

/// Конструктор

/// </summary>

public Point(double x, double y)

{

X = x;

Y = y;

}

public Point(Matrix m)

{

if (m.N != 2 && m.M!= 1)

return;

X = m.Element[0, 0];

Y = m.Element[1, 0];

}

/// <summary>

/// Длина

/// </summary>

/// <returns></returns>

public double GetLength()

{

return Math.Sqrt(X \* X + Y \* Y);

}

/// <summary>

/// Сложение

/// </summary>

/// <returns></returns>

public static Point operator +(Point p1, Point p2)

{

return new Point(p1.X + p2.X, p1.Y + p2.Y);

}

/// <summary>

/// Вычитание

/// </summary>

/// <param name="p1"></param>

/// <param name="p2"></param>

/// <returns></returns>

public static Point operator -(Point p1, Point p2)

{

return new Point(p1.X - p2.X, p1.Y - p2.Y);

}

/// <summary>

/// Скалярное произведение

/// </summary>

/// <param name="p1"></param>

/// <param name="p2"></param>

/// <returns></returns>

public static double operator \*(Point p1, Point p2)

{

return p1.X \* p2.X + p1.Y \* p2.Y;

}

/// <summary>

/// Произведение на число

/// </summary>

/// <param name="p1"></param>

/// <param name="c"></param>

/// <returns></returns>

public static Point operator \*(Point p1, double c)

{

return new Point (p1.X \* c, p1.Y \* c);

}

public static Point operator \*(double c, Point p1)

{

return new Point(c \* p1.X, c \* p1.Y);

}

/// <summary>

/// Деление на число

/// </summary>

/// <param name="p1"></param>

/// <param name="p2"></param>

/// <returns></returns>

public static Point operator /(double c, Point p1)

{

return new Point(c / p1.X, c / p1.Y);

}

public static Point operator /(Point p1, double c)

{

return new Point(p1.X / c, p1.Y / c);

}

}

}

Matrix.cs

using System;

using System.Collections.Generic;

namespace ConsoleApp1

{

class Matrix

{

public double[,] Element { get; set; }

public int N { get; set; }

public int M { get; set; }

/// <summary>

/// Конструкторы

/// </summary>

/// <param name="points"></param>

public Matrix(Matrix m)

{

N = m.N;

M = m.M; // для Point

Element = new double[N, M];

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

for (int j = 0; j < M; j++)

Element[i, j] = m.Element[i, j];

}

public Matrix(List<Point> points)

{

N = points.Count;

M = 2; // для Point

Element = new double[N, M];

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

{

Element[i, 0] = points[i].X;

Element[i, 1] = points[i].Y;

}

}

public Matrix(Point[] points)

{

N = points.Length;

M = 2; // для Point

Element = new double[N, M];

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

{

Element[i, 0] = points[i].X;

Element[i, 1] = points[i].Y;

}

}

public Matrix(Point point)

{

N = 1;

M = 2;

Element = new double[N, M];

Element[0, 0] = point.X;

Element[0, 1] = point.Y;

}

public Matrix(double element)

{

N = 1;

M = 1;

Element = new double[N, M];

Element[0, 0] = element;

}

public Matrix(int n = 2, int m = 2)

{

N = n;

M = m;

Element = new double[N, M];

}

/// <summary>

/// Транспонирование

/// </summary>

/// <param name="m1"></param>

/// <returns></returns>

public static Matrix Transpose(Matrix m1) // +

{

Matrix m = new Matrix(m1.M, m1.N);

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

for (int j = 0; j < m.M; j++)

m.Element[i, j] = m1.Element[j, i];

return m;

}

/// <summary>

/// Обратная матрица

/// </summary>

/// <param name="m1"></param>

/// <returns></returns>

public static Matrix Inverse(Matrix m1)

{

if (m1.N != m1.M)

return null;

int n = m1.N;

Matrix m = new Matrix(m1);

// Делаем единичную матрицу

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

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

{

if (i == j)

m.Element[i, j] = 1;

else

m.Element[i, j] = 0;

}

// Вычисляем обратную матрицу

double arg;

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

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

{

if (i == j)

continue;

arg = m1.Element[i, j] / m1.Element[j, j];

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

{

m1.Element[i, k] = m1.Element[i, k] - m1.Element[j, k] \* arg;

m.Element[i, k] = m.Element[i, k] - m.Element[j, k] \* arg;

}

}

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

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

{

if (i == j)

{

arg = m1.Element[i, j];

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

{

m1.Element[i, k] = m1.Element[i, k] / arg;

m.Element[i, k] = m.Element[i, k] / arg;

}

}

}

return m;

}

/// <summary>

/// Детерминант матрицы

/// </summary>

/// <param name="m1"></param>

/// <returns></returns>

public static double Determinant(Matrix m1)

{

if (m1.N != m1.M)

return 0;

int n = m1.N;

double result = 0;

if (n == 1)

return m1.Element[0, 0];

else if (n == 2)

return m1.Element[0, 0] \* m1.Element[1, 1] - m1.Element[0, 1] \* m1.Element[1, 0];

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

{

Matrix m = new Matrix(m1.N - 1, m1.M - 1);

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

for (int j = 0; j < m.M; j++)

m.Element[i, j] = (j < k) ? m1.Element[i + 1, j] : m1.Element[i + 1, j + 1];

result += Math.Pow(-1, 2 + k) \* m1.Element[0, k] \* Determinant(m);

}

return result;

}

/// <summary>

/// Умножение

/// </summary>

/// <param name="m1"></param>

/// <param name="m2"></param>

/// <returns></returns>

public static Matrix operator \*(Matrix m1, Matrix m2) // +

{

Matrix m = new Matrix(m1.N, m2.M);

if (m1.N == 1 && m1.M == 1)

{

m = m1.Element[0, 0] \* m2;

return m;

}

else if (m2.N == 1 && m2.M == 1)

{

m = m1 \* m2.Element[0, 0];

return m;

}

else if (m1.M != m2.N)

return null;

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.M; j++)

for (int k = 0; k < m2.N; k++)

m.Element[i, j] += m1.Element[i, k] \* m2.Element[k, j];

return m;

}

public static Point operator \*(Matrix m1, Point p) // +

{

Matrix m2 = new Matrix(p);

m2 = Transpose(m2); // Транспонируем

if (m1.M != m2.N)

return null;

Matrix m = new Matrix(m1.N, m2.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.M; j++)

for (int k = 0; k < m2.N; k++)

m.Element[i, j] += m1.Element[i, k] \* m2.Element[k, j];

p = new Point(m);

return p;

}

public static Point operator \*(Point p, Matrix m2)

{

Matrix m1 = new Matrix(p);

if (m1.M != m2.N)

return null;

Matrix m = new Matrix(m1.N, m2.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.M; j++)

for (int k = 0; k < m2.N; k++)

m.Element[i, j] += m1.Element[i, k] \* m2.Element[k, j];

p = new Point(m);

return p;

}

public static Matrix operator \*(Matrix m1, double c)

{

Matrix m = new Matrix(m1.N, m1.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.N; j++)

m.Element[i, j] = m1.Element[i, j] \* c;

return m;

}

public static Matrix operator \*(double c, Matrix m1)

{

Matrix m = new Matrix(m1.N, m1.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.N; j++)

m.Element[i, j] = c \* m1.Element[i, j];

return m;

}

/// <summary>

/// Деление

/// </summary>

/// <param name="m1"></param>

/// <param name="m2"></param>

/// <returns></returns>

public static Matrix operator /(Matrix m1, Matrix m2) // +

{

Matrix m = new Matrix(m1.N, m2.M);

if (m1.N == 1 && m1.M == 1)

{

m = m1.Element[0, 0] / m2;

return m;

}

else if (m2.N == 1 && m2.M == 1)

{

m = m1 / m2.Element[0, 0];

return m;

}

else if (m1.M != m2.N)

return null;

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.M; j++)

for (int k = 0; k < m2.N; k++)

m.Element[i, j] += m1.Element[i, k] \* (1 / m2.Element[k, j]);

return m;

}

public static Matrix operator /(Matrix m1, double c)

{

Matrix m = new Matrix(m1.N, m1.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.N; j++)

m.Element[i, j] = m1.Element[i, j] / c;

return m;

}

public static Matrix operator /(double c, Matrix m1)

{

Matrix m = new Matrix(m1.N, m1.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.N; j++)

m.Element[i, j] = c / m1.Element[i, j];

return m;

}

/// <summary>

/// Вычитание

/// </summary>

/// <param name="m1"></param>

/// <param name="m2"></param>

/// <returns></returns>

public static Matrix operator -(Matrix m1, Matrix m2) // +

{

Matrix m = new Matrix(m1.N, m2.M);

if (m1.N == 1 && m1.M == 1)

{

m = m1.Element[0, 0] - m2;

return m;

}

else if (m2.N == 1 && m2.M == 1)

{

m = m1 - m2.Element[0, 0];

return m;

}

else if (m1.N != m2.N && m1.M != m2.M)

return null;

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] - m2.Element[i, j];

return m;

}

public static Matrix operator -(Matrix m1, Point p)

{

Matrix m2 = new Matrix(p);

if (m1.N != m2.N && m1.M != m2.M)

return null;

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] - m2.Element[i, j];

return m;

}

public static Matrix operator -(Point p, Matrix m2)

{

Matrix m1 = new Matrix(p);

if (m1.N != m2.N && m1.M != m2.M)

return null;

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] - m2.Element[i, j];

return m;

}

public static Matrix operator -(Matrix m1, double c)

{

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.M; j++)

m.Element[i, j] = m1.Element[i, j] - c;

return m;

}

public static Matrix operator -(double c, Matrix m1)

{

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.M; j++)

m.Element[i, j] = c - m1.Element[i, j];

return m;

}

/// <summary>

/// Сложение

/// </summary>

/// <param name="m1"></param>

/// <param name="m2"></param>

/// <returns></returns>

public static Matrix operator +(Matrix m1, Matrix m2)

{

Matrix m = new Matrix(m1.N, m2.M);

if (m1.N == 1 && m1.M == 1)

{

m = m1.Element[0, 0] + m2;

return m;

}

else if (m2.N == 1 && m2.M == 1)

{

m = m1 + m2.Element[0, 0];

return m;

}

else if (m1.N != m2.N && m1.M != m2.M)

return null;

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] + m2.Element[i, j];

return m;

}

public static Matrix operator +(Matrix m1, Point p)

{

Matrix m2 = new Matrix(p);

if (m1.N != m2.N && m1.M != m2.M)

return null;

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] + m2.Element[i, j];

return m;

}

public static Matrix operator +(Point p, Matrix m2)

{

Matrix m1 = new Matrix(p);

if (m1.N != m2.N && m1.M != m2.M)

return null;

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] + m2.Element[i, j];

return m;

}

public static Matrix operator +(Matrix m1, double c)

{

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.M; j++)

m.Element[i, j] = m1.Element[i, j] + c;

return m;

}

public static Matrix operator +(double c, Matrix m1)

{

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.M; j++)

m.Element[i, j] = c + m1.Element[i, j];

return m;

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Diagnostics;

namespace ConsoleApp1

{

class Program

{

/// <summary>

/// Функция

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static double Func(Point p) => Math.Pow(p.Y, 2) + 6 \* Math.Pow(p.X, 2) + 5 \* p.X + 2 \* p.Y + 1;

//2 \* Math.Pow(p.X, 2) + p.X \* p.Y + Math.Pow(p.Y, 2);

/// <summary>

/// Первая производная F'x

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static double Derrivate\_dx(Point p)

{

double dx = 0.001;

return (Func(new Point(p.X + dx, p.Y)) - Func(p)) / dx;

}

/// <summary>

/// Первая производная F'y

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static double Derrivate\_dy(Point p)

{

double dy = 0.001;

return (Func(new Point(p.X, p.Y + dy)) - Func(p)) / dy;

}

/// <summary>

/// Вторая производная F''x

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static double Derrivate2\_dxdx(Point p)

{

double dx = 0.001;

return (Derrivate\_dx(new Point(p.X + dx, p.Y)) - Derrivate\_dx(p)) / dx;

}

/// <summary>

/// Смешанная производная F''xy

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static double Derrivate2\_dxdy(Point p)

{

double dx = 0.001;

return (Derrivate\_dy(new Point(p.X + dx, p.Y)) - Derrivate\_dy(p)) / dx;

}

/// <summary>

/// Вторая производная F''y

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static double Derrivate2\_dydy(Point p)

{

double dy = 0.001;

return (Derrivate\_dy(new Point(p.X, p.Y + dy)) - Derrivate\_dy(p)) / dy;

}

/// <summary>

/// Смешанная производная F''yx

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static double Derrivate2\_dydx(Point p)

{

double dy = 0.001;

return (Derrivate\_dx(new Point(p.X, p.Y + dy)) - Derrivate\_dx(p)) / dy;

}

/// <summary>

/// Создать матрицу Гессе

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static Matrix Create\_H(Point p)

{

Matrix m = new Matrix();

m.Element[0, 0] = Derrivate2\_dxdx(p);

m.Element[0, 1] = Derrivate2\_dxdy(p);

m.Element[1, 0] = Derrivate2\_dydx(p);

m.Element[1, 1] = Derrivate2\_dydy(p);

return m;

}

/// <summary>

/// Градиент функции в точке

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static Point GradFunc(Point p)

{

double derX = Derrivate\_dx(p);

double derY = Derrivate\_dy(p);

return new Point(derX, derY);

}

/// <summary>

/// Вывести результат в консоль

/// </summary>

/// <param name="resultPoint"></param>

/// <param name="steps"></param>

static void PrintResult(Point resultPoint, int steps)

{

double result = Func(resultPoint);

Console.WriteLine("X = " + resultPoint.X + "; Y = " + resultPoint.Y);

Console.WriteLine("F(x,y) = {0:F10}", result);

Console.WriteLine("Точность: {0:F10}", Math.Abs(Func(new Point(-0.4164, -1)) - result));

Console.WriteLine("Количество шагов: {0}", steps);

}

/// <summary>

/// Метод Марквардта

/// </summary>

/// <param name="x0"></param>

/// <param name="eps"></param>

static void Run(Point x0, double eps1 = 0.1, double eps2 = 0.15, int M = 10)

{

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

x.Add(x0);

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

mu.Add(20);

Matrix E = new Matrix(new Point[] { new Point (1, 0), new Point (0, 1) });

List<Point> d = new List<Point>();

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

// Шаг 1

Point gradF = GradFunc(x0); // !!! ненужная операция

Matrix H = Create\_H(x0); // !!! ненужная операция

// Шаг 2

int k = 0;

mu[k] = mu[0];

while (true)

{

// Шаг 3

gradF = GradFunc(x[k]);

// Шаг 4

if (gradF.GetLength() < eps1)

{

// а

PrintResult(x[k], k);

return;

}

// б

// Шаг 5

// а

if (k >= M)

{

PrintResult(x[k], k);

return;

}

// б

// Шаг 6

H = Create\_H(x[k]);

// Шаг 7

step7:

Matrix X = H + mu[k] \* E;

// Шаг 8

X = Matrix.Inverse(X);

// Шаг 9

d.Add(null);

d[k] = -1 \* X \* gradF;

// Шаг 10

x.Add(null);

x[k + 1] = x[k] + d[k]; //!!! можно было и так записать вместо того, что предлагает алгоритм

// Шаг 11

if (Func(x[k + 1]) < Func(x[k]))

{

// а

// Шаг 12

mu.Add(0);

mu[k + 1] = mu[k] / 2; //!!! в алгоритме увеличивают k и затем пишут mu[k+1], а нужно

// увеличить k в конце

k++;

// goto Шаг 3

}

else

{

// б

// Шаг 13

mu[k] = 2 \* mu[k];

goto step7;

}

}

}

/// <summary>

/// Главная функция

/// </summary>

/// <param name="args"></param>

static void Main(string[] args)

{

Stopwatch sw = new Stopwatch();

//Console.WriteLine("Тест 0");

//Console.WriteLine("Берем стандартные параметры");

//sw.Reset();

//sw.Start();

//Run(new Point(0.5, 1));

//sw.Stop();

Console.WriteLine("Тест 1");

Console.WriteLine("Берем стандартные параметры");

sw.Reset();

sw.Start();

Run(new Point(10, 10));

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Тест 2");

Console.WriteLine("Увеличим eps1");

sw.Reset();

sw.Start();

Run(new Point(10, 10), 0.5);

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Тест 3");

Console.WriteLine("Уменьшим eps1");

sw.Reset();

sw.Start();

Run(new Point(10, 10), 0.01);

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Тест 4");

Console.WriteLine("Уменьшим M");

sw.Reset();

sw.Start();

Run(new Point(10, 10), 0.1, 4);

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Тест 5");

Console.WriteLine("Увеличим M");

sw.Reset();

sw.Start();

Run(new Point(10, 10), 0.1, 20);

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.ReadKey();

}

}

}

# Скриншоты результата выполнения программы

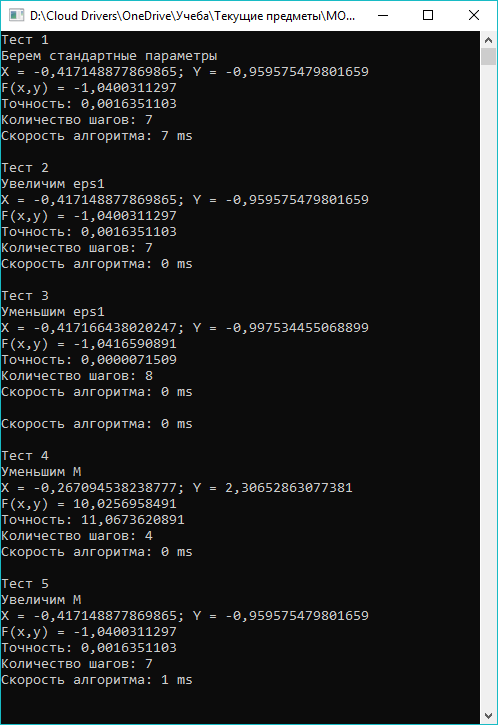


Рисунок 1 – Пример работы программы

# Выводы

В результате проделанной работы была разработана программа, которая выполнила поиск безусловного экстремума (минимума) заданной функции методом Марквардта. В ходе тестов были установлены следующие зависимости при изменениях параметров (таблица 1): при уменьшении значения eps1 точность значительно возрастает, а при увеличении изначального значения результат не изменился; параметр М при увеличении влияния на алгоритм не оказал, однако его уменьшение привело к значительной потере точности и неверному результату в целом (алгоритм не сделал достаточно шагов для поиска минимума).

Таблица 1 – Исследование алгоритма путем изменения параметров

|  |  |
| --- | --- |
| Тест 1  Берем стандартные параметры  X = -0,417148877869865; Y = -0,959575479801659  F(x,y) = -1,0400311297  Точность: 0,0016351103  Количество шагов: 7  Скорость алгоритма: 7 ms | Тест 2  Увеличим eps1  X = -0,417148877869865; Y = -0,959575479801659  F(x,y) = -1,0400311297  Точность: 0,0016351103  Количество шагов: 7  Скорость алгоритма: 0 ms |
| Тест 3  Уменьшим eps1  X = -0,417166438020247; Y = -0,997534455068899  F(x,y) = -1,0416590891  Точность: 0,0000071509  Количество шагов: 8  Скорость алгоритма: 0 ms | Тест 4  Уменьшим M  X = -0,267094538238777; Y = 2,30652863077381  F(x,y) = 10,0256958491  Точность: 11,0673620891  Количество шагов: 4  Скорость алгоритма: 0 ms |
| Тест 5  Увеличим M  X = -0,417148877869865; Y = -0,959575479801659  F(x,y) = -1,0400311297  Точность: 0,0016351103  Количество шагов: 7  Скорость алгоритма: 1 ms |  |

Также можно сравнить данный алгоритм с методами первого порядка (у них схожие входные параметры и виды функций), а также можно сравнить с методами нулевого порядка с общей функцией, для которой выполняется поиск минимума и сравнить их. Результаты приведены в таблице 2:

Таблица 2 – Сравнение различных алгоритмов при значениях x0 = (10,10) eps1=0.1, eps2 = 0.15, M = 10 для методов первого порядка и одинаковой исследуемой функции для методов нулевого порядка с eps = 0.1

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Время** | **Точность результата** | **Шаги** |
| Хука-Дживса | 1 мс | 0.0065 | 8 |
| Нелдера-Мида | 1 мс | 0.026 | 7 |
| Розенброка | 1 мс | 0.017 | 2 |
| Покоординатного спуска | 3 мс | 0.0011 | 8 |
| Флетчера-Ривса | 3 мс | 0.00053 | 2 |
| Дэвидона-Флетчера-Пауэлла | 6 мс | 0.00549 | 3 |
| Ньютона | 6 мс | 0.000001323 | 1 |
| Ньютона-Рафсона | 6 мс | 0,0000065610 | 1 |
| Марквардта | 7 мс | 0,0016351103 | 7 |

Как видно из таблицы 2, наилучший результат у метода Ньютона. Хоть точность результата метода Марквардта не является наилучшей (среди методов второго порядка худшая), однако здесь не используется метод одномерной минимизации, который может повлиять на результат алгоритма, который используется в остальных методах второго порядка и не только. Время, затраченное на вычисление, соизмеримо с остальными методами второго порядка.