Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5**

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 Розенброка.

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

# Листинг программы

Point.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Point

{

public double X { get; set; }

public double Y { get; set; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="x"></param>

/// <param name="y"></param>

public Point(double x, double y)

{

X = x;

Y = y;

}

/// <summary>

/// Длина

/// </summary>

/// <returns></returns>

public double GetLength()

{

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

}

/// <summary>

/// Сложение

/// </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>

/// 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);

}

//public static double operator &(Point p1, Point p2)

//{

// return (p1 \* p2) / (p1.GetLength() \* p2.GetLength());

//}

}

}

Program.cs

using System;

using System.Diagnostics;

namespace ConsoleApp1

{

class Program

{

/// <summary>

/// Function

/// </summary>

/// <param name="x1"></param>

/// <param name="x2"></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;

static double Run(Point x, double eps = 0.01, double alpha = 2, double beta = -0.5)

{

// Step 1

int N = 3; // Макс. число неудачных шагов

int k = 0; // Счетчик шагов

int i = 0; // i-ое направление

Point[] d = { new Point(1, 0), new Point(0, 1) }; // Векторы направлений

double[] delta = { 1, 2 }; // Длина шага

int n = delta.Length - 1; // Количество направлений (n = размер массива векторов, если начинать отсчет с 1)

int size = n + 1; // Размер всех массивов в данном пространстве

Point[] y = new Point[size + 1]; // Точки смещения

y[0] = x; // Начальная точка смещения (совпадает с начальной точкой)

Point x\_next = x; // Следующая точка

step2:

while (true)

{

// Step 2

// a

if (Func(y[i] + d[i] \* delta[i]) < Func(y[i]))

{

y[i + 1] = y[i] + d[i] \* delta[i];

delta[i] = delta[i] \* alpha;

}

// b

else

{

y[i + 1] = y[i];

delta[i] = delta[i] \* beta;

}

// Step 3

// a

if (i < n)

{

i++;

continue; // Переход к Step 2

}

// b

else //if (i == n)

{

// b.1

if (Func(y[n + 1]) < Func(y[0]))

{

y[0] = y[n + 1];

i = 0;

continue; // Переход к Step 2

}

// b.2

else

{

// b.2.1

int l = 0; // Счетчик неудачных шагов

if (Func(y[n + 1]) < Func(x))

{

l = 0;

break; // Переход к Step 4

}

// b.2.2

else

{

l++;

if (l < N)

{

bool end = true;

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

{

if (Math.Abs(delta[j]) > eps)

{

y[0] = y[n + 1];

i = 0;

end = false;

break;

}

}

if (end)

{

double result = Func(x);

Console.WriteLine("Результат: {0}", result);

Console.WriteLine("Точность: {0:F10}", Math.Abs(Func(new Point(-0.4166666666, -1)) - result));

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

return result;

}

else

continue; // Переход к Step 2

}

else

{

break; // Переход к Step 4

}

}

}

}

}

// Step 4

x\_next = y[n + 1];

// a

if ((x\_next - x).GetLength() <= eps) // || x[k+1] - x[x] || <= eps

{

x = x\_next;

double result = Func(x);

Console.WriteLine("Результат: {0}", result);

Console.WriteLine("Точность: {0:F10}", Math.Abs(Func(new Point(-0.4166666666, -1)) - result));

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

return result;

}

// b

else

{

double[] lambda = new double[size];

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

{

lambda[j] = (x\_next - x) \* d[j];

}

Point[] a = new Point[size];

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

{

Point sum = lambda[j] \* d[j];

for (int m = j + 1; m < size; m++)

{

sum +=lambda[m] \* d[m];

}

a[j] = sum;

}

Point[] b = new Point[size];

Point[] d\_v = new Point[size];

b[0] = a[0];

d\_v[0] = b[0] / b[0].GetLength();

for (int j = 1; j < size; j++)

{

Point sum = a[j] \* d\_v[0] \* d\_v[0];

for (int m = 1; m < size - 1; m++)

{

sum += a[j] \* d\_v[m] \* d\_v[m];

}

b[j] = a[j] - sum;

d\_v[j] = b[j] / b[j].GetLength();

}

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

{

d[j] = d\_v[j];

}

delta = new double[] { 1, 2 };

k++;

x = y[0];

i = 0;

goto step2; // Переход на шаг 2

}

}

/// <summary>

/// Main method

/// </summary>

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

static void Main(string[] args)

{

Stopwatch sw = new Stopwatch(); // Timer for algorithm's speed

Console.WriteLine("Тест 1");

Console.WriteLine("Берем стандартные параметры");

sw.Reset();

sw.Start();

Run(new Point(0, 0));

sw.Stop();

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

Console.WriteLine("Тест 2");

Console.WriteLine("Уменьшаем точность");

sw.Reset();

sw.Start();

Run(new Point(0, 0), 0.001);

sw.Stop();

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

Console.WriteLine("Тест 3");

Console.WriteLine("Увеличиваем точность");

sw.Reset();

sw.Start();

Run(new Point(0, 0), 0.1);

sw.Stop();

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

Console.WriteLine("Тест 4");

Console.WriteLine("Увеличиваем коэффициент растяжения");

sw.Reset();

sw.Start();

Run(new Point(0, 0), 0.01, 3);

sw.Stop();

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

Console.WriteLine("Тест 5");

Console.WriteLine("Уменьшаем коэффициент растяжения");

sw.Reset();

sw.Start();

Run(new Point(0, 0), 0.01, 1.1);

sw.Stop();

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

Console.WriteLine("Тест 6");

Console.WriteLine("Увеличиваем коэффициент сжатия");

sw.Reset();

sw.Start();

Run(new Point(0, 0), 0.01, 2, -0.1);

sw.Stop();

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

Console.WriteLine("Тест 7");

Console.WriteLine("Уменьшаем коэффициент сжатия");

sw.Reset();

sw.Start();

Run(new Point(0, 0), 0.01, 2, -0.9);

sw.Stop();

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

Console.ReadKey();

}

}

}

# Скриншоты результата выполнения программы

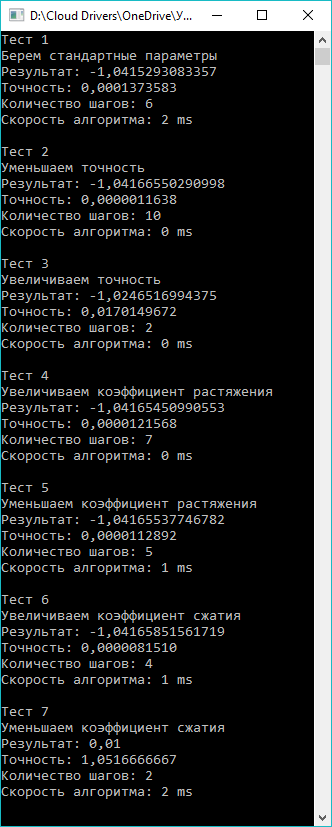


Рисунок 1 – Пример работы программы

# Выводы

В результате проделанной работы была разработана программа, которая выполнила поиск безусловного экстремума (минимума) заданной функции методом Розенброка. В ходе тестов были установлены следующие зависимости при изменениях параметров (таблица 1): при увеличении/уменьшении параметра точности уменьшается/увеличивается количество итераций и понижается/повышается точность результата; изменение коэффициента растяжения в большую сторону повысило точность результата и увеличило количество итераций, уменьшение также повысило точность результата и количество итераций уменьшилось; при увеличении коэффициента сжатия точность повысилась существенно, а количество итераций стало значительно меньше, уменьшение же параметра существенно ухудшило точность, однако и итераций стало еще меньше.

Таблица 2 – Исследование алгоритма путем изменения параметров

|  |  |
| --- | --- |
| Берем стандартные параметры  Результат: -1,0415293083357  Точность: 0,0001373583  Количество шагов: 6  Скорость алгоритма: 3 ms |  |
| Увеличиваем точность  Результат: -1,0246516994375  Точность: 0,0170149672  Количество шагов: 2  Скорость алгоритма: 0 ms | Уменьшаем точность  Результат: -1,04166550290998  Точность: 0,0000011638  Количество шагов: 10  Скорость алгоритма: 0 ms |
| Увеличиваем коэффициент растяжения  Результат: -1,04165450990553  Точность: 0,0000121568  Количество шагов: 7  Скорость алгоритма: 0 ms | Уменьшаем коэффициент растяжения  Результат: -1,04165537746782  Точность: 0,0000112892  Количество шагов: 5  Скорость алгоритма: 1 ms |
| Увеличиваем коэффициент сжатия  Результат: -1,04165851561719  Точность: 0,0000081510  Количество шагов: 4  Скорость алгоритма: 1 ms | Уменьшаем коэффициент сжатия  Результат: 0,01  Точность: 1,0516666667  Количество шагов: 2  Скорость алгоритма: 0 ms |

В качестве сравнения данного алгоритма с другими использованы в каждом методе одинаковые точки старта алгоритма, одинаковые параметры (уникальные параметры взяты по рекомендациям авторов алгоритмов) и сравнивались результаты при изменении параметра точности. Результаты приведены на таблице 2.

Таблица 2 – Сравнение различных алгоритмов при различных значениях параметра точности (0.1 / 0.01 / 0.001)

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Время** | **Точность результата** | **Шаги** |
| Хука-Дживса | 1 мс / 1 мс / 1 мс | 0.0065 / 0.001 / 0.0000016 | 8 / 14 / 20 |
| Нелдера-Мида | 1 мс / 2 мс / 3 мс | 0.026 / 0.0163 / 0.0004 | 7 / 9 / 10 |
| Розенброка | <1 мс / 1 мс / 1 мс | 0.017 / 0.000137 / 0.00000116 | 2 / 6 / 10 |