# ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

### Кафедра АСУ

### 

Отчёт

о лабораторной работе №2

по курсу: «Решение задачи оптимизации роевыми алгоритмами»

# Вариант «2»

Выполнил:

ст. гр. ИСм-22

Мариничев И.И

Проверила:

Васяева Т.А

# 

# Донецк 2022

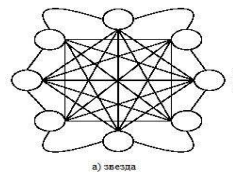
**Ход работы:**

1. Изучить теоретический материал.
2. Разработать роевый алгоритм для нахождения оптимума заданной по варианту функции в лабораторной работе №2. Для четных – локальный роевый алгоритм, для нечетных вариантов – глобальный.
3. Выполнить программную реализацию разработанного алгоритма на языке высокого уровня. Рекомендованный язык Python с применением DEAP. Предусмотреть возможность просмотра процесса поиска решения.
4. Исследовать зависимость числа поколений (генераций), точности нахождения решения от основных параметров роевого алгоритма.
5. Сравнить найденное решение с действительным и с найденным решением в лабораторной работе №2 и №5. Сделать выводы.

*Задание:*

Поскольку мой вариант чётный, для лабораторной работы мною был использован ЛОКАЛЬНЫЙ РОЕВОЙ АЛГОРИТМ.

Поскольку моя задача имеет лишь один глобальный оптимум, сетевой структурой была выбрана «Звезда»:



Каждая точка содержит в себе:

* текущие координаты,
* текущую скорость,
* текущее значение Целевой Функции (далее ЦФ) координаты лучшей ЦФ, найденной точкой,
* координаты лучшей ЦФ соседей.

Схема работы алгоритма, следующая:

1. Генерируется массив точек.
2. Начинается основной цикл(по заданному количеству эпох):
   1. Для каждой точки рассчитывается её следующее месторасположение по 3м вектором:

* Предыдущее направление;
* Направление к лучшей найденной точке;
* Направление к локальной лучшей точке.

И высчитывается значение ЦФ.

* 1. Затем сравнивается текущее значение ЦФ с наилучшим значением, найденным этой точкой и со значением наилучшей ЛОКАЛЬНОЙ точки. Если текущее значение ЦФ лучше, то оно перезаписывается в наилучшее найденное или Локальный оптимум.
  2. После этого в цикле по

(i=0; i<количество\_точек; i++)

{

(j=0; j<количество\_точек; j++)

{

}

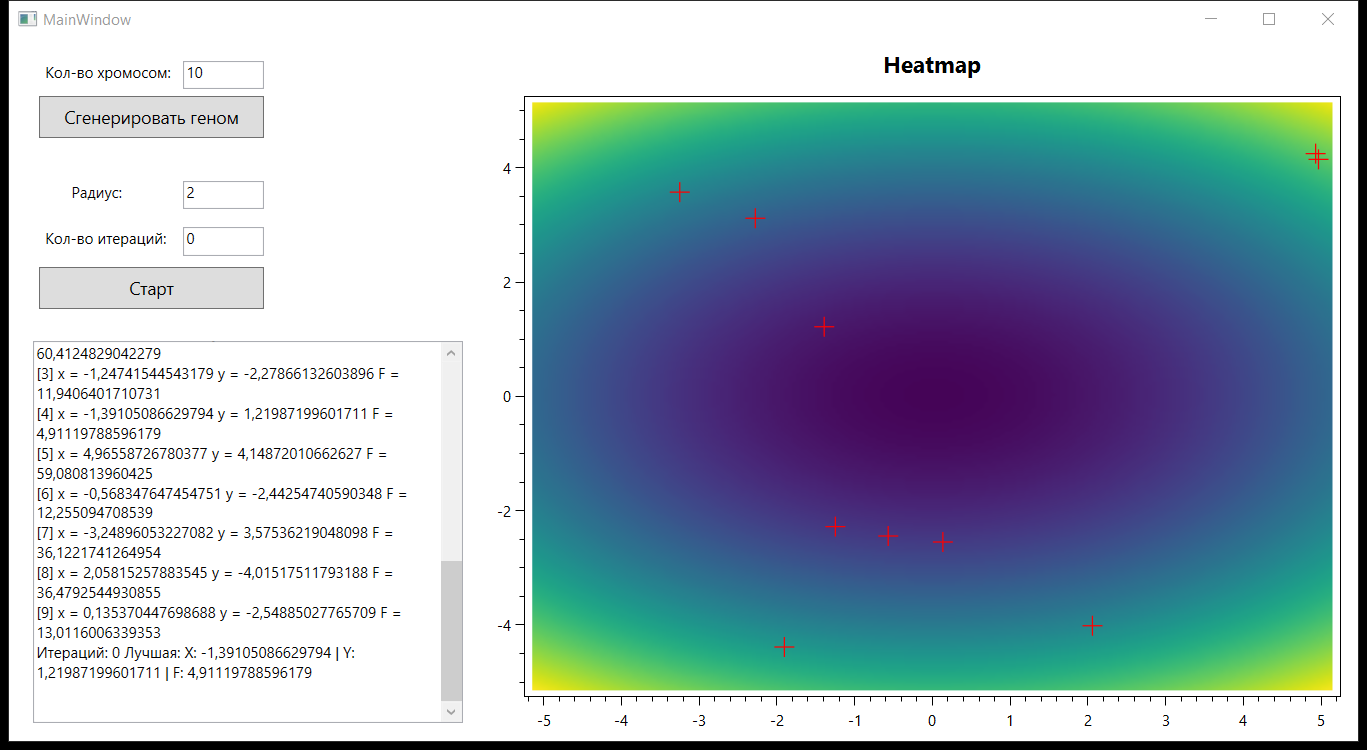
}

Ищутся точки «j», которые находятся в определённом радиусе далее (R) от текущей точки «i». И сравнивается значение наилучших найденных ЦФ и Локального Оптимума этих точек.

* 1. Цикл повторяется заданное количество эпох

**Проверим результат выполнения работы:**

Исходная генерация:



Исходная популяция:

[0] x = -2,27681835995839 y = 3,11881338979994 F = 24,6378957650344

[1] x = -1,90361589281988 y = -4,38541162456638 F = 42,0874237011603

[2] x = 4,92857347956326 y = 4,24980272252569 F = 60,4124829042279

[3] x = -1,24741544543179 y = -2,27866132603896 F = 11,9406401710731

[4] x = -1,39105086629794 y = 1,21987199601711 F = 4,91119788596179

[5] x = 4,96558726780377 y = 4,14872010662627 F = 59,080813960425

[6] x = -0,568347647454751 y = -2,44254740590348 F = 12,255094708539

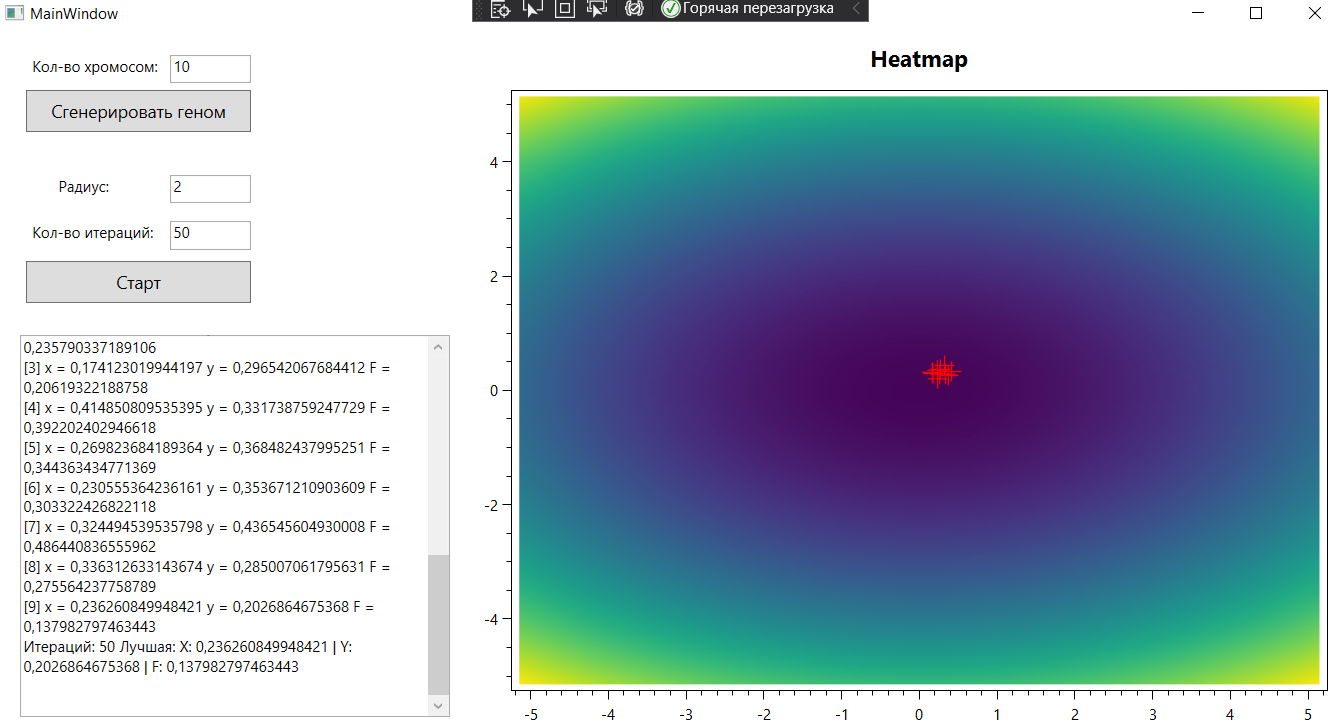
[7] x = -3,24896053227082 y = 3,57536219048098 F = 36,1221741264954

[8] x = 2,05815257883545 y = -4,01517511793188 F = 36,4792544930855

[9] x = 0,135370447698688 y = -2,54885027765709 F = 13,0116006339353

Лучшая: X: -1,39105086629794 | Y: 1,21987199601711 | F: 4,91119788596179

Спустя 50 итераций:



Финальный геном:

[0] x = 0,373302761160707 y = 0,264012074255071 F = 0,278759702195138

[1] x = 0,167143313668543 y = 0,315146064724292 F = 0,226570971526516

[2] x = 0,263937753104897 y = 0,288207563810394 F = 0,235790337189106

[3] x = 0,174123019944197 y = 0,296542067684412 F = 0,20619322188758

[4] x = 0,414850809535395 y = 0,331738759247729 F = 0,392202402946618

[5] x = 0,269823684189364 y = 0,368482437995251 F = 0,344363434771369

[6] x = 0,230555364236161 y = 0,353671210903609 F = 0,303322426822118

[7] x = 0,324494539535798 y = 0,436545604930008 F = 0,486440836555962

[8] x = 0,336312633143674 y = 0,285007061795631 F = 0,275564237758789

[9] x = 0,236260849948421 y = 0,2026864675368 F = 0,137982797463443

Итераций: 50 Лучшая: X: 0,236260849948421 | Y: 0,2026864675368 | F: 0,137982797463443

Известный лучший результат: F(0;0) = 0.

Разница: 0 - 0,137982797463443 = -0,137982797463443;

**СРАВНИМ СО 2Й И 5Й ЛАБОРАТОРНЫМИ:**

Алгоритм работает значительно быстрее, по сравнению со второй лабораторной работой и примерно с той же точностью вычислений.

Однако уступает по точности вычислений алгоритму из 5той лабораторной работы.

**ВЫВОД: для нахождения оптимума в задачах с 1м локальным минимумом, эволюционная стратегия преобладает над генетическим алгоритмом и роевым алгоритмом. Роевой алгоритм, в свою очередь, преобладает над генетическим алгоритмом.**

Листинг программного кода:

Main form:

using Newtonsoft.Json;

using OxyPlot;

using OxyPlot.Axes;

using OxyPlot.Series;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace gen\_2

{

public partial class MainWindow : Window

{

private PlotModel model;

private int sizeMap = 200;

public double startPoint = -5.12, endPoint = 5.12;

private ScatterSeries point = new ScatterSeries();

GeneticAlgorithm geneticAlgorithm = new GeneticAlgorithm();

public MainWindow()

{

InitializeComponent();

initHeatMap();

point.ColorAxisKey = "ColorAxis";

point.MarkerSize = 8;

point.MarkerType = MarkerType.Plus;

}

private void initHeatMap()

{

model = new PlotModel { Title = "Heatmap" };

model.Axes.Add(new LinearColorAxis { Palette = OxyPalettes.Viridis(255) });

var data = new double[sizeMap, sizeMap];

double xValue = startPoint, yValue = startPoint, max = endPoint;

double det = (Math.Abs(xValue) + max) / sizeMap;

for (int y = 0; y < sizeMap; ++y)

{

xValue = startPoint;

for (int x = 0; x < sizeMap; ++x)

{

data[x, y] = getFunction(xValue, yValue);

xValue += det;

}

yValue += det;

}

var heatMapSeries = new HeatMapSeries

{

X0 = startPoint,

X1 = endPoint,

Y0 = startPoint,

Y1 = endPoint,

Interpolate = true,

RenderMethod = HeatMapRenderMethod.Bitmap,

Data = data

};

var axis1 = new LinearColorAxis();

axis1.Key = "ColorAxis";

axis1.IsAxisVisible = true;

model.Axes.Add(axis1);

axis1.Palette.Colors.Clear();

axis1.Palette.Colors.Add(OxyColor.FromArgb((byte)255, 255, 0, 0));

model.Series.Add(heatMapSeries);

testPlotView.Model = model;

}

public double getFunction(double x, double y)

{

//return 10 \* 2 + Math.Pow(x, 2) - 10 \* Math.Cos(2 \* 3.14 \* x) + Math.Pow(y, 2) - 10 \* Math.Cos(2 \* 3.14 \* y);

return (x \* x) + (2 \* y \* y);

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

int kolChromosome = Convert.ToInt32(textBlock2.Text);

geneticAlgorithm.CreateGenome(kolChromosome, startPoint, endPoint, startPoint, endPoint);

}

private void Button\_Click\_2(object sender, RoutedEventArgs e)

{

int kolIteration = Convert.ToInt32(textBlock1.Text);

double nrad = Convert.ToDouble(textBlock3.Text);

List<Chromosome> components = geneticAlgorithm.Evolution(kolIteration, nrad);

//double x = components.xVal;

//double y = components.yVal;

initHeatMap();

var random = new Random();

textBox1.Clear();

textBox1.AppendText(geneticAlgorithm.TextAll + Environment.NewLine);

while (model.Series.Count > 1)

{

model.Series.Remove(model.Series.Last());

}

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

{

model.Series.Add(components[i].Point);

}

testPlotView.Model.InvalidatePlot(true);

}

private void TextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

}

private void Button\_Click\_4(object sender, RoutedEventArgs e)

{

}

private void Button\_Click\_5(object sender, RoutedEventArgs e)

{

}

}

}

Class1:

using OxyPlot;

using OxyPlot.Series;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace gen\_2

{

class BestPoint

{

public double bestx;

public double besty;

public double bestfunc;

public BestPoint() { }

}

class Chromosome

{

public double xVal;

public double yVal;

public double funcVal;

public double vtx, vty;

public BestPoint mybest;

public BestPoint localbest;

private ScatterSeries point;

public Chromosome()

{

mybest = new BestPoint();

localbest = new BestPoint();

}

public Chromosome(double xval, double yval, double fun, double nvtx, double nvty)

{

xVal = xval;

yVal = yval;

funcVal = fun;

vtx = nvtx;

vty = nvty;

mybest = new BestPoint();

localbest = new BestPoint();

}

public Chromosome(Chromosome c)

{

xVal = c.xVal;

yVal = c.yVal;

funcVal = c.funcVal;

vtx = c.vtx;

vty = c.vty;

}

public ScatterSeries Point

{

get

{

this.point = new ScatterSeries();

this.point.ColorAxisKey = "ColorAxis";

this.point.MarkerSize = 8;

this.point.MarkerType = MarkerType.Plus;

point.Points.Add(new ScatterPoint(xVal, yVal, double.NaN, 1));

return point;

}

set => point = value;

}

public void ReCreateFull(double xval, double yval, double fun)

{

xVal = xval;

yVal = yval;

funcVal = fun;

}

}

class FitnessFunc

{

public static readonly double minInt = -5.12;

public static readonly double maxInt = 5.12;

public static double Func(double x, double y)

{

//return 10 \* 2 + Math.Pow(x, 2) - 10 \* Math.Cos(2 \* 3.14 \* x) + Math.Pow(y, 2) - 10 \* Math.Cos(2 \* 3.14 \* y);

return (x \* x) + (2 \* y \* y);

}

}

class GeneticAlgorithm

{

public double neihgbourradius;

private Random random = new Random();

private StringBuilder textAll = new StringBuilder();

List<Chromosome> genome;

Chromosome best;

public string TextAll { get { return textAll.ToString(); } }

public GeneticAlgorithm() { }

private void poisk(double nrad)

{

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

{

double nrad2 = nrad;

int colp = 0;

change\_nrad:

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

{

double v = Math.Sqrt(Math.Pow(genome[i].xVal - genome[j].xVal, 2) + Math.Pow(genome[i].yVal - genome[j].yVal, 2));

if (v < nrad2)

{

colp++;

if (genome[j].localbest.bestfunc < genome[i].localbest.bestfunc)

{

genome[i].localbest.bestfunc = genome[j].localbest.bestfunc;

genome[i].localbest.bestx = genome[j].localbest.bestx;

genome[i].localbest.besty = genome[j].localbest.besty;

}

}

if (colp == 0)

{

nrad2 += nrad;

goto change\_nrad;

}

}

}

}

public List<Chromosome> Evolution(int iterationsCount, double nrad)

{

double currentnrad = nrad;

textAll.Clear();

double minInterval = FitnessFunc.minInt;

double maxInterval = FitnessFunc.maxInt;

int currentCount = 0;

textAll.Append("Исходная популяция" + Environment.NewLine);

printGenome(genome);

Chromosome best = genome.Find(g => g.mybest.bestfunc == genome.Min(v => v.mybest.bestfunc));

double tmpvtx, tmpvty;

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

{

//if (i % 5 == 0) currentnrad += nrad;

poisk(currentnrad);

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

{

tmpvtx = (0.2 \* genome[j].vtx) + (0.3 \* random.NextDouble() \* (genome[j].mybest.bestx - genome[j].xVal)) + (0.5 \* random.NextDouble() \* (genome[j].localbest.bestx - genome[j].xVal));

tmpvty = (0.2 \* genome[j].vty) + (0.3 \* random.NextDouble() \* (genome[j].mybest.besty - genome[j].yVal)) + (0.5 \* random.NextDouble() \* (genome[j].localbest.besty - genome[j].yVal));

genome[j].xVal += tmpvtx;

genome[j].yVal += tmpvty;

genome[j].funcVal = FitnessFunc.Func(genome[j].xVal, genome[j].yVal);

if(genome[j].funcVal< genome[j].mybest.bestfunc)

{

genome[j].mybest.bestx = genome[j].xVal;

genome[j].mybest.besty = genome[j].yVal;

genome[j].mybest.bestfunc = genome[j].funcVal;

}

if(genome[j].mybest.bestfunc < genome[j].localbest.bestfunc )

{

genome[j].localbest.bestx = genome[j].mybest.bestx;

genome[j].localbest.besty = genome[j].mybest.besty;

genome[j].localbest.bestfunc = genome[j].mybest.bestfunc;

}

}

currentCount++;

}

best = genome.Find(g => g.mybest.bestfunc == genome.Min(v => v.mybest.bestfunc));

textAll.Append("Финальный геном" + Environment.NewLine);

printGenome(genome);

textAll.Append("Итераций: " + currentCount + " Лучшая: " + "X: " + best.xVal + " | " + "Y: " + best.yVal + " | F: " + best.funcVal + Environment.NewLine);

return genome;

}

public void CreateGenome(int count, double xMin, double xMax, double yMin, double yMax)

{

genome = new List<Chromosome>();

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

{

genome.Add(new Chromosome((random.NextDouble() \* (xMax - xMin) + xMin), (random.NextDouble() \* (yMax - yMin) + yMin), 0, 0.5, 0.5));

genome[i].funcVal = FitnessFunc.Func(genome[i].xVal, genome[i].yVal);

genome[i].mybest.bestx = genome[i].xVal;

genome[i].mybest.besty = genome[i].yVal;

genome[i].mybest.bestfunc = genome[i].funcVal;

genome[i].localbest.bestx = genome[i].xVal;

genome[i].localbest.besty = genome[i].yVal;

genome[i].localbest.bestfunc = genome[i].funcVal;

}

}

public void printGenome(List<Chromosome> genome)

{

int i = 0;

foreach (Chromosome c in genome)

{

textAll.Append("[" + i++ + "] " + "x = " + c.xVal + " y = " + c.yVal + " F = " + c.funcVal + Environment.NewLine);

}

}

}

}