# ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

### Кафедра АСУ

### 

Отчёт

о лабораторной работе №1

по курсу: «Эволюционные методы оптимизации»

# Вариант «2»

Выполнил:

ст. гр. ИСм-22

Мариничев И.И

Проверила:

Васяева Т.А

# 

# Донецк 2022

*Цель работы:* изучение основных принципов генетических алгоритмов, приобретение навыков их программной реализации.

*Ход работы:*

1. Изучить теоретический материал.
2. Ознакомиться с вариантами кодирования хромосомы.
3. Рассмотреть способы выполнения операторов репродукции, кроссинговера и мутации.
4. Разработать простой генетический алгоритм для нахождения оптимума заданной по варианту функции одной переменной (таб. 1.1). Вид экстремума: для четных – минимум, для не четных – максимум.
5. Выполнить программную реализацию разработанного алгоритма на языке высокого уровня. Рекомендованный язык Python с применением DEAP. Предусмотреть возможность просмотра процесса поиска решения.
6. Исследовать зависимость числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма: ‒ число особей в популяции; ‒ вероятность кроссинговера, мутации.
7. Сравнить найденное ГА решение с действительным.

*Задание:*

Поиск минимума.



Для выполнения лабораторной были выбраны следующие методы и параметры:

Для отбора хромосом при репродукции был использован турнирный метод (на каждом шаге итерации – выбор из 3х случайных хромосом).

Партнёр для кроссинговера выбирается по средству инбридинга (выбирается хромосома с минимальной разницей между Y).

При построении функции был использован шаг в 0,01 для оси X и в 0,1 для оси Y.

Максимальное количество итераций установим равным 500. Минимумом функции является (2,23; -2,95)

**Этап 1**

Для проверки работы алгоритма и сбора начальных данных, оставим лишь ограничение по количеству циклов.

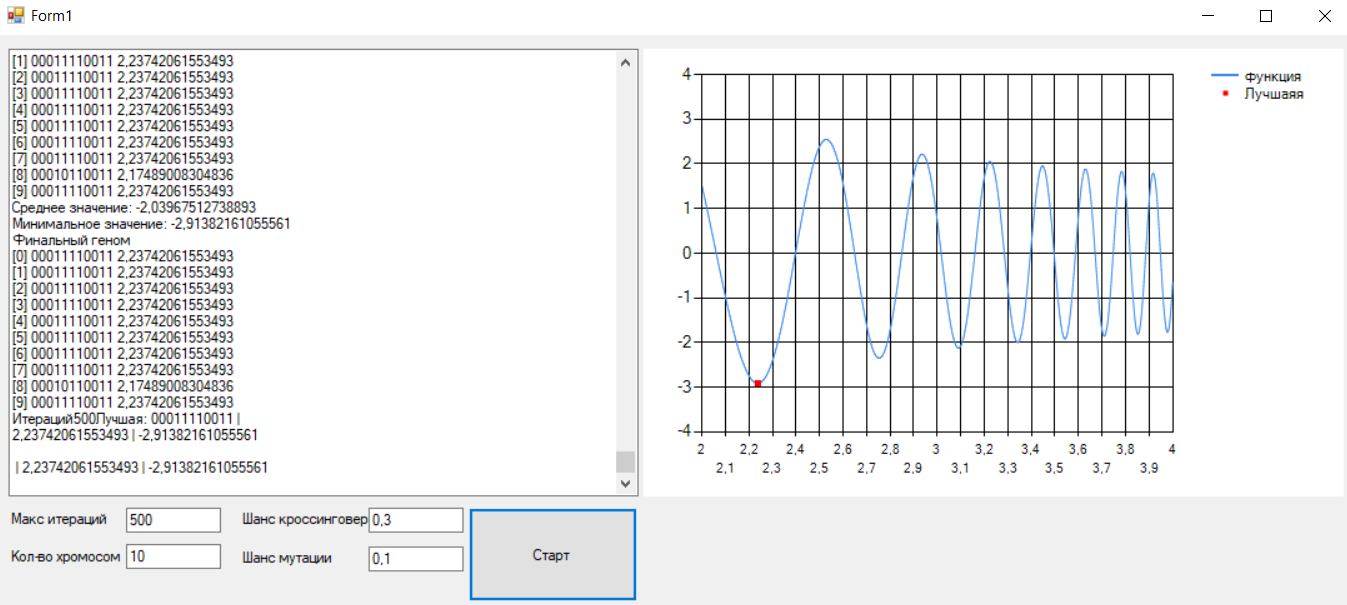
Для каждого эксперимента будем совершать 5 запусков программы и собирать среднее значение результата.

За исходные данные примем:

1. Ограничение циклов = 500;
2. Кол-во хромосом = 10;
3. Шанс кроссинговера = 0,3;
4. Шанс мутации = 0,1.

Результаты:

1й запуск



Пример работы алгоритма функции: ниже представлен ход выполнение 1й итерации из цикла

Исходная популяция

[0] 11001111001 3,61895456765999

[1] 10010010110 3,14704445530044

[2] 01100100010 2,78358573522228

[3] 00011001010 2,19736199316072

[4] 00010100110 2,16218856863703

[5] 00010110010 2,17391304347826

[6] 01010111011 2,68295065950171

[7] 01101101011 2,85490962383977

[8] 00101111101 2,37225207620909

[9] 11011101101 3,73229115779189

Итерация 1

Репродукция:

[0] 01010111011 2,68295065950171

[1] 01100100010 2,78358573522228

[2] 00010110010 2,17391304347826

[3] 00010100110 2,16218856863703

[4] 00011001010 2,19736199316072

[5] 01010111011 2,68295065950171

[6] 00101111101 2,37225207620909

[7] 00010100110 2,16218856863703

[8] 10010010110 3,14704445530044

[9] 10010010110 3,14704445530044

2 Кроссинговер с шансом 0,3

Parent 1 00010110010 2,17391304347826

Parent 2 10010010110 3,14704445530044

K = 9 ,размер = 11

Child 1 000101100\_10

Child 2 100100101\_10

Child 1 2,17391304347826

Child 2 3,14704445530044

3 Кроссинговер с шансом 0,3

Parent 1 00010100110 2,16218856863703

Parent 2 10010010110 3,14704445530044

K = 9 ,размер = 11

Child 1 000101001\_10

Child 2 100100101\_10

Child 1 2,16218856863703

Child 2 3,14704445530044

6 Кроссинговер с шансом 0,3

Parent 1 00101111101 2,37225207620909

Parent 2 10010010110 3,14704445530044

K = 1 ,размер = 11

Child 1 0\_0010010110

Child 2 1\_0101111101

Child 1 2,14655593551539

Child 2 3,37274059599414

7 Кроссинговер с шансом 0,3

Parent 1 00010100110 2,16218856863703

Parent 2 10101111101 3,37274059599414

K = 8 ,размер = 11

Child 1 00010100\_101

Child 2 10101111\_110

Child 1 2,16121152906693

Child 2 3,37371763556424

сокращённая популяция

[0] 00011001010 2,19736199316072

[1] 01100100010 2,78358573522228

[2] 01010111011 2,68295065950171

[3] 01010111011 2,68295065950171

[4] 10010010110 3,14704445530044

[5] 00010100110 2,16218856863703

[6] 00010010110 2,14655593551539

[7] 00010100101 2,16121152906693

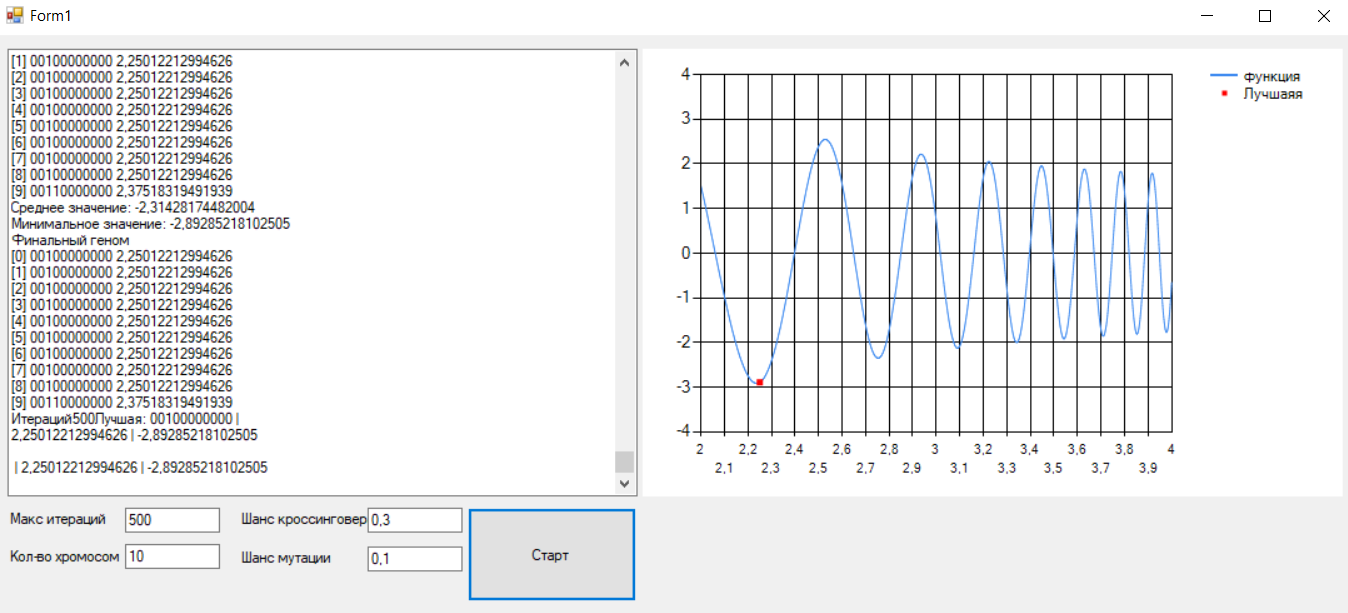
[8] 00010110010 2,17391304347826

[9] 10101111110 3,37371763556424

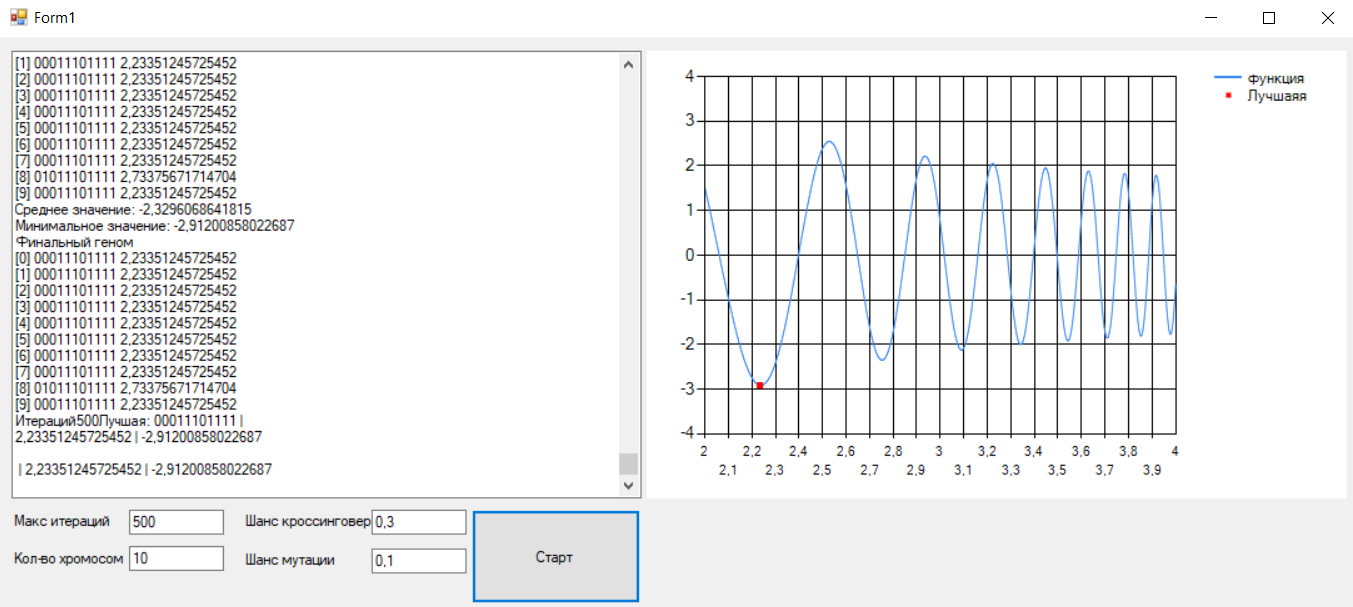
Среднее значение: -0,767044084832254

Минимальное значение: -2,71904484408845

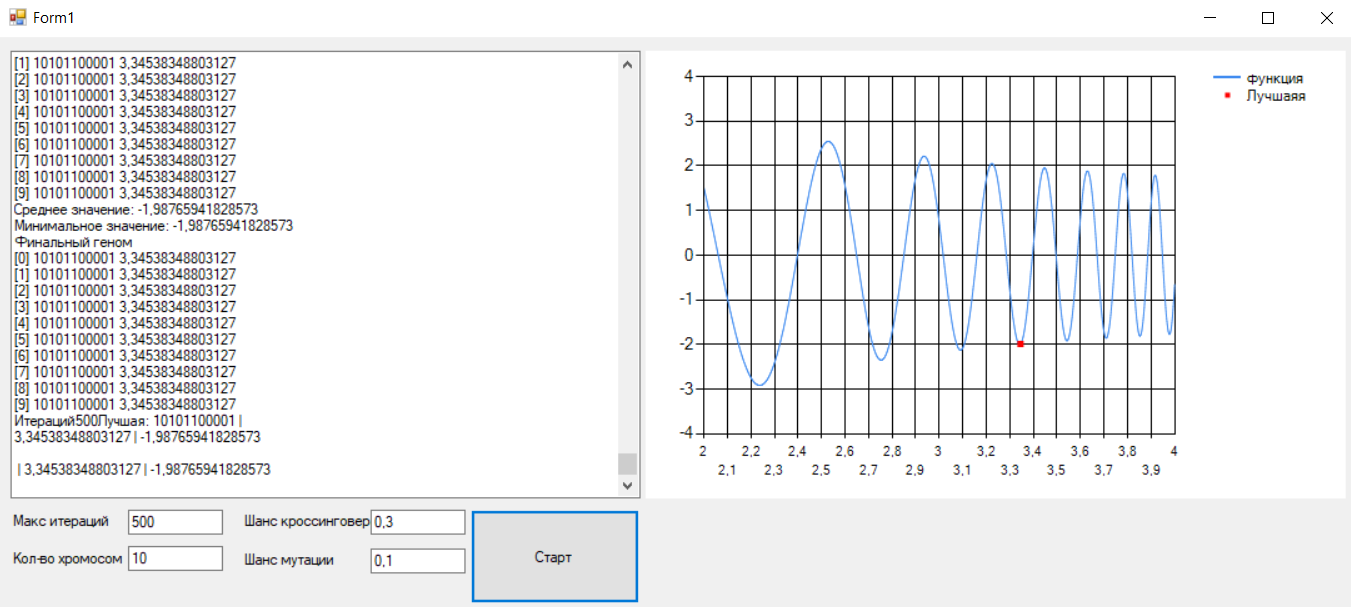
2й запуск



3й запуск



4й запуск



Среднее значение рассчитанного минимума составляет: -2,67.

3 раза алгоритм нашёл глобальный минимум функции и 1 раз локальный.

Погрешность = (2,95 – 2, 67)/ 2,95 = 0,09.

**Этап 2**

Для проведения эксперимента, добавим дополнительное ограничение: алгоритм останавливается, если функция достигла глобального минимума функции с погрешностью в 0,05 единиц (т.е. значение функии от хромосомы будет не менее 2,9).

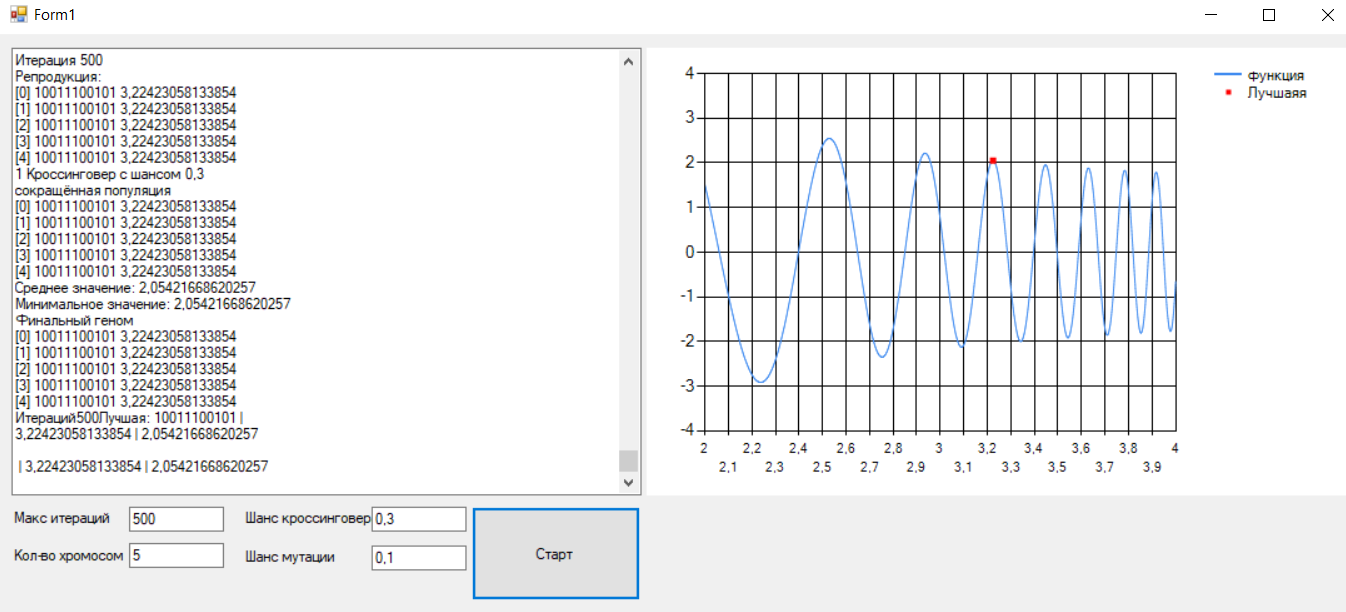
**Проверка влияния количества хромосом в геноме:**

Общее количество точек на графике функции = 200, поэтому, максимальным количеством хромосом в геноме будем считать 20, что составляет 10% от максимального количества точек на графике.

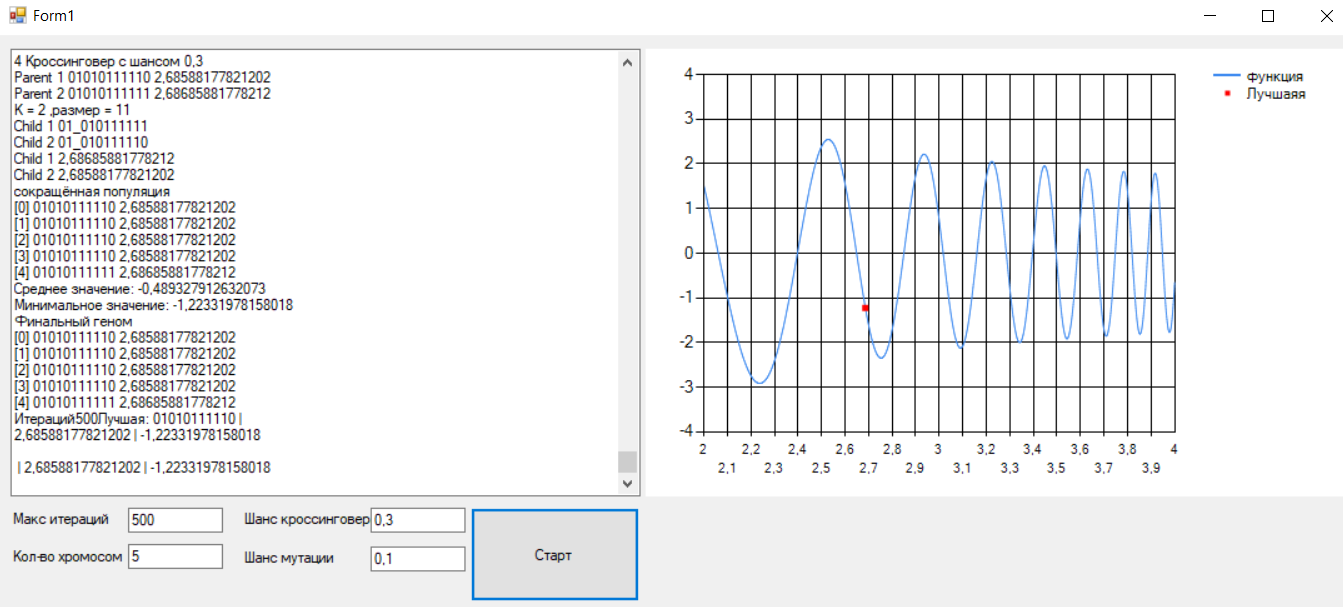
5 хромосом:

1. Ограничение циклов = 500;
2. Кол-во хромосом = 5;
3. Шанс кроссинговера = 0,3;
4. Шанс мутации = 0,1.

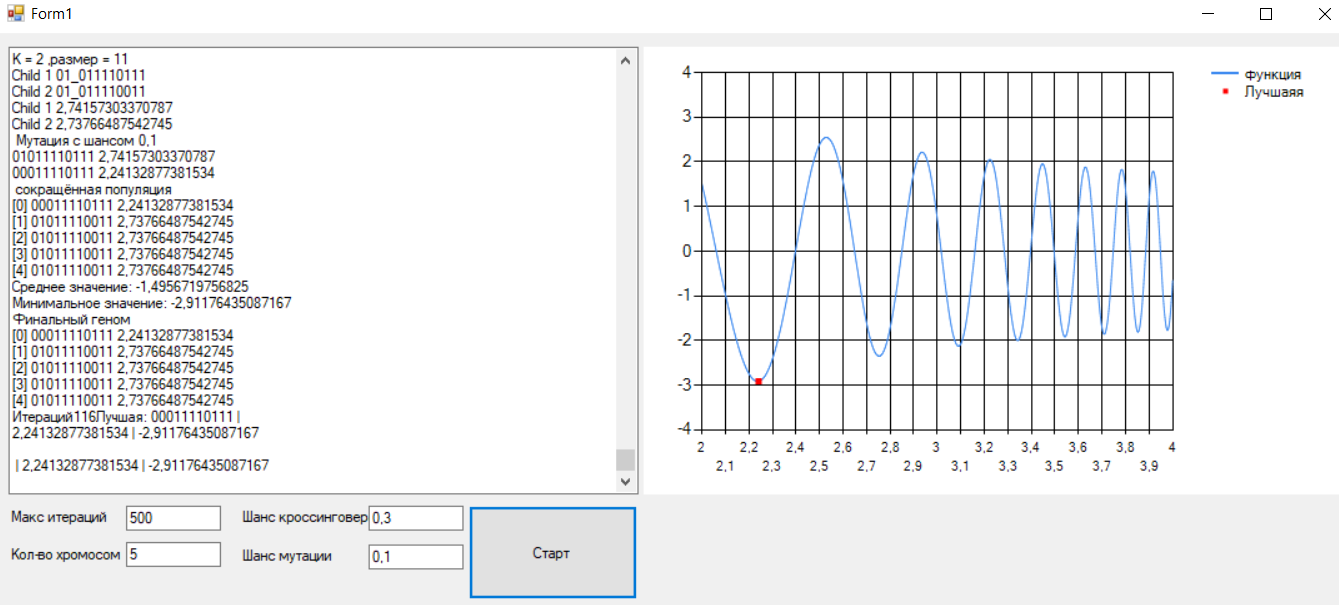
1й запуск



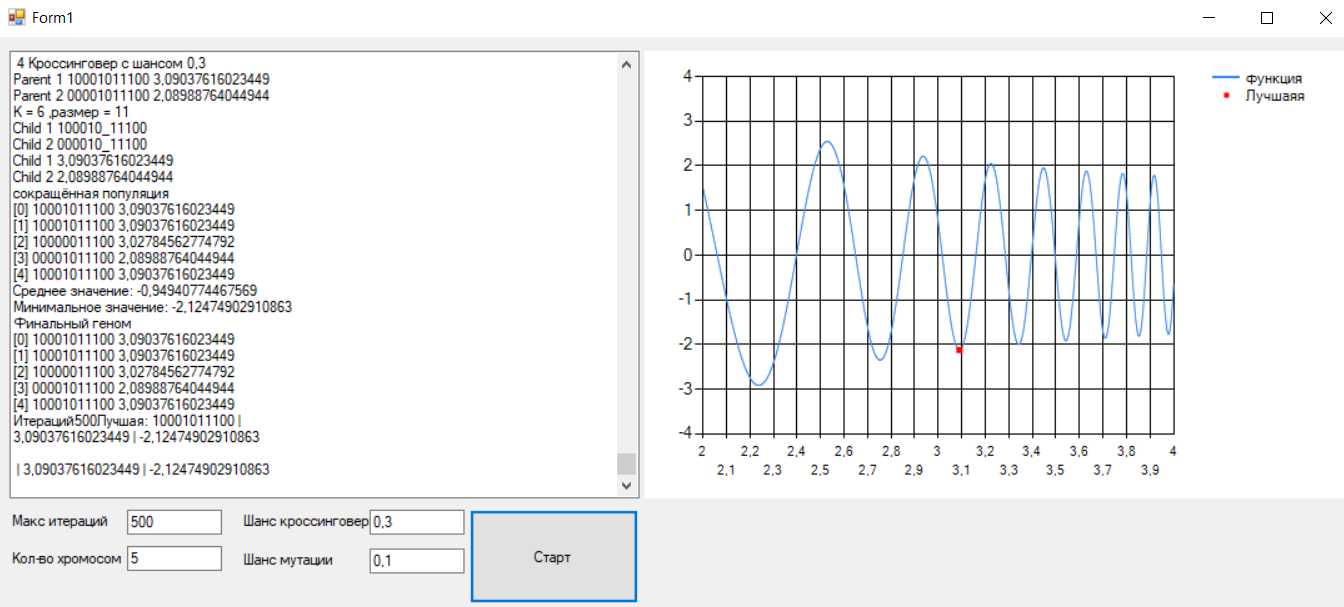
2й запуск



3й запуск



4й запуск

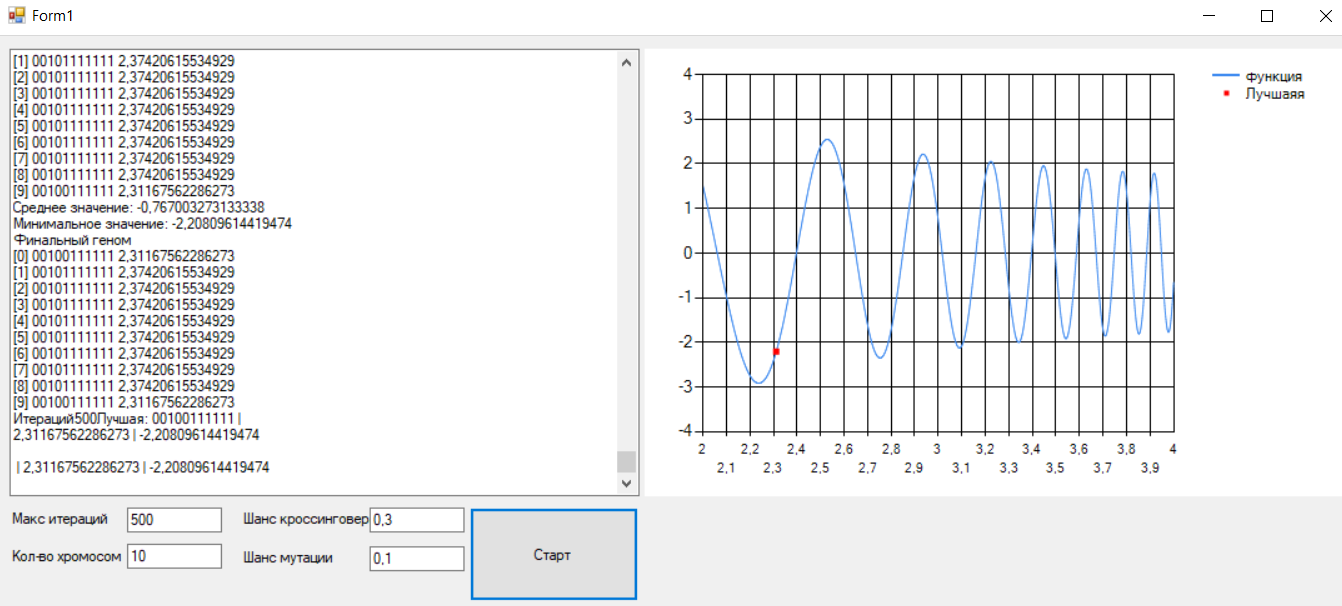


Среднее значение рассчитанного минимума составляет: -1,05.

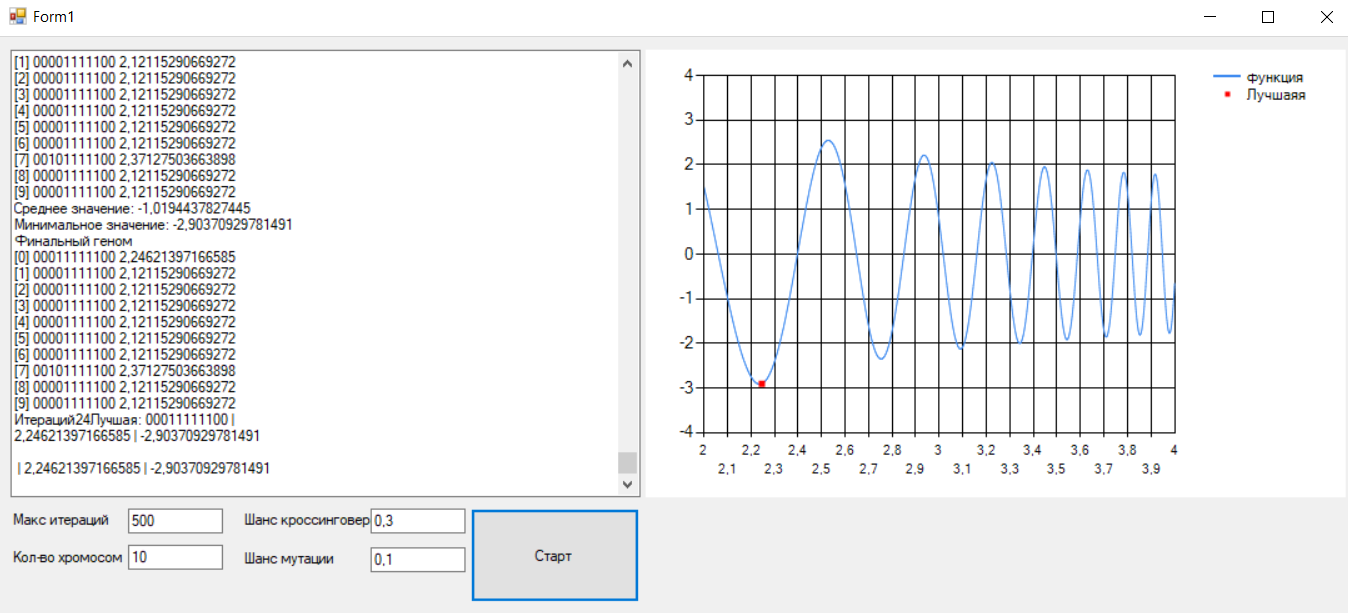
Погрешность = (2,95 – 1,05)/ 2,95 = 0,64.

**10 хромосом**

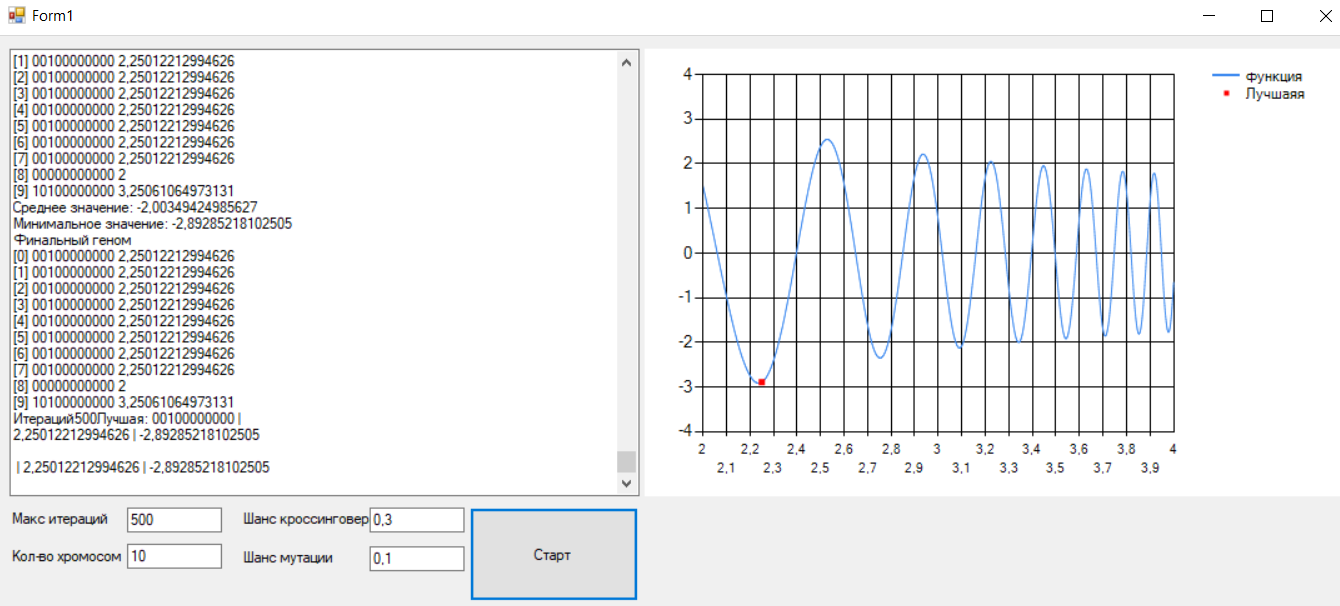
1й запуск



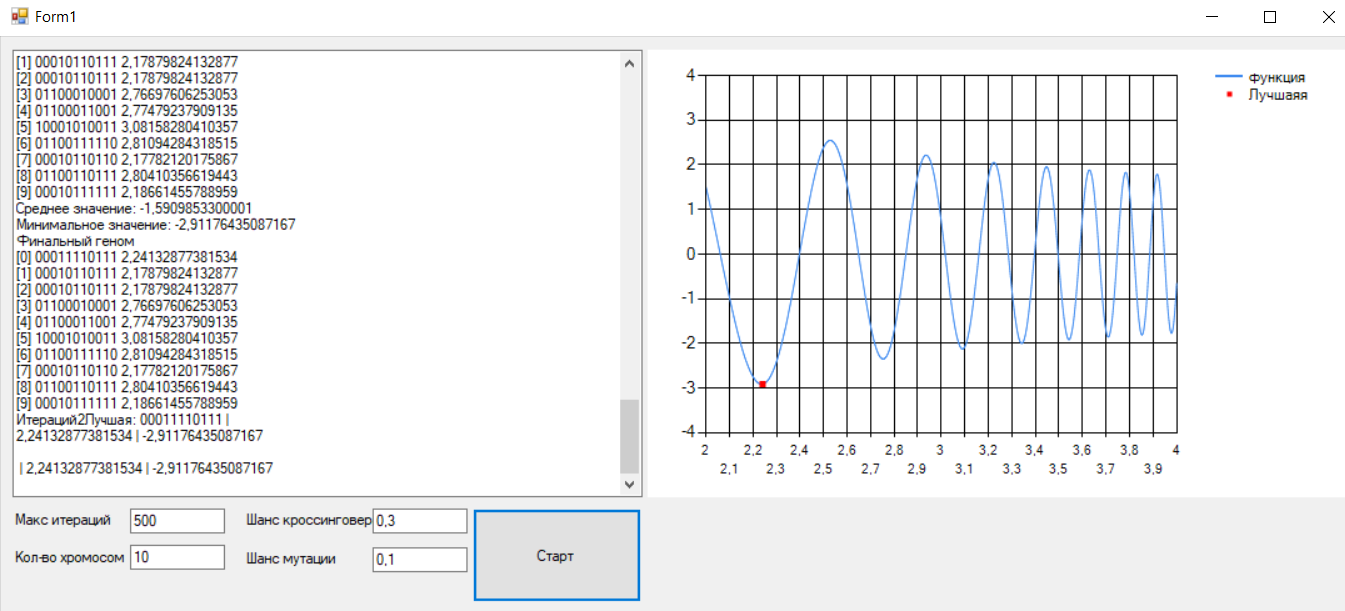
2й запуск



3й запуск



4й запуск

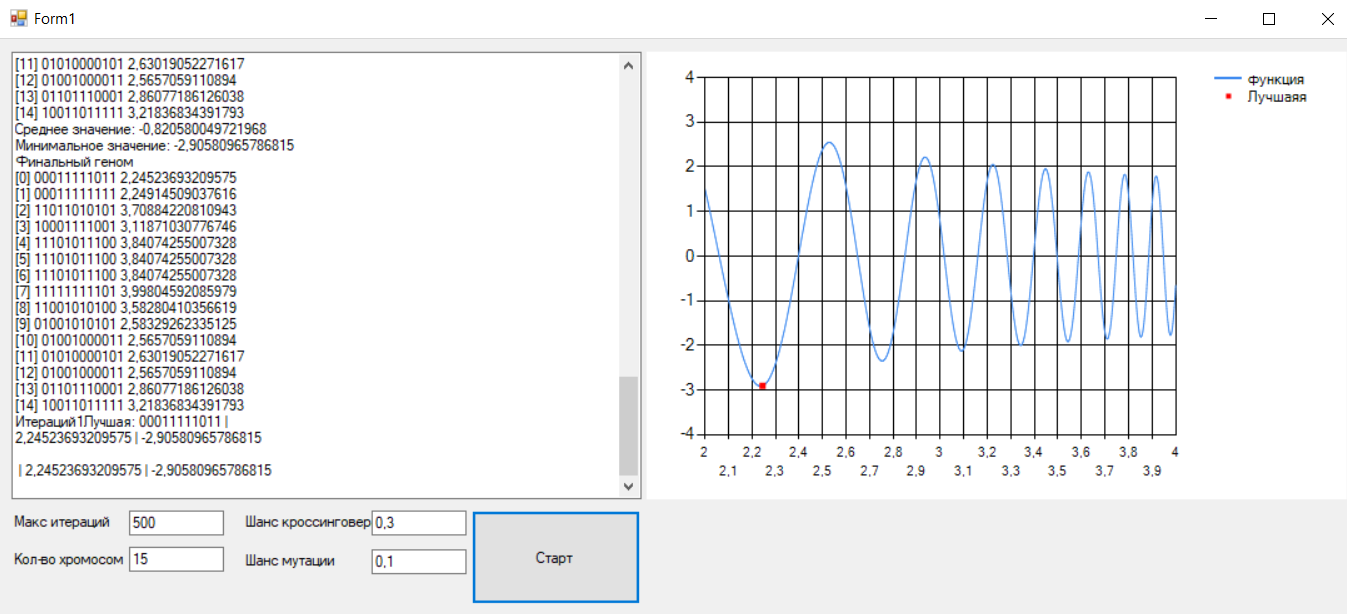


Среднее значение рассчитанного минимума составляет: -2,73.

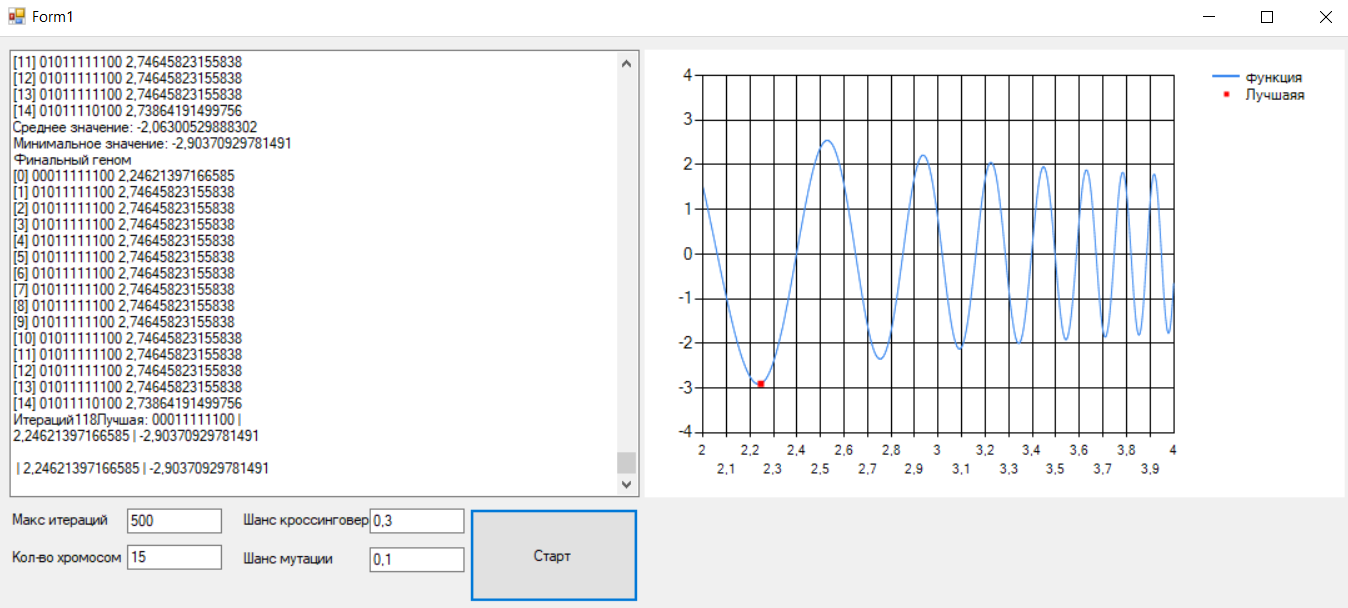
Погрешность = (2,95 – 2,73)/ 2,95 = 0,075.

**15 хромосом**

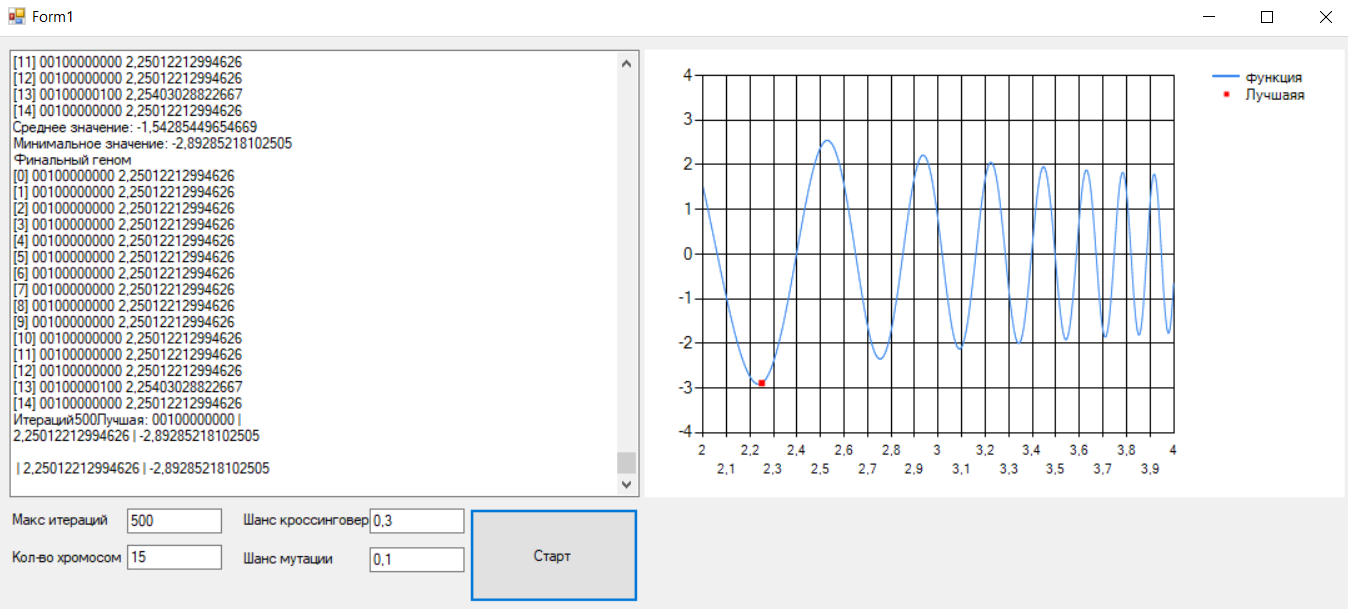
1й запуск



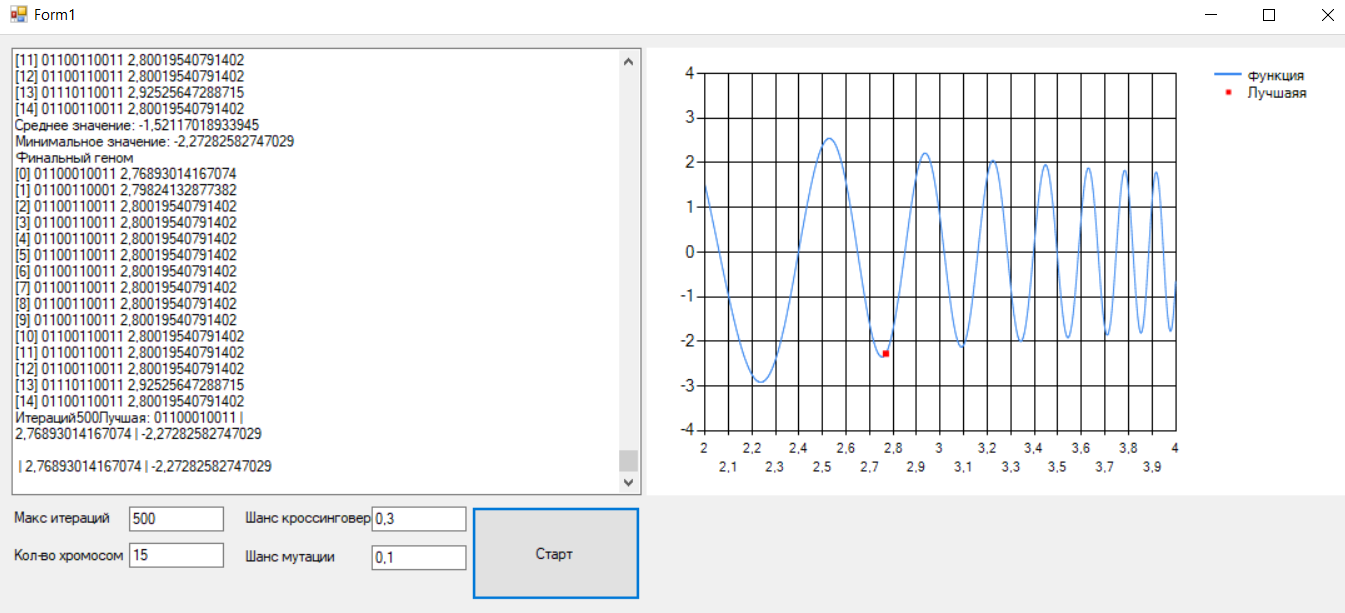
2й запуск



3й запуск



4й запуск

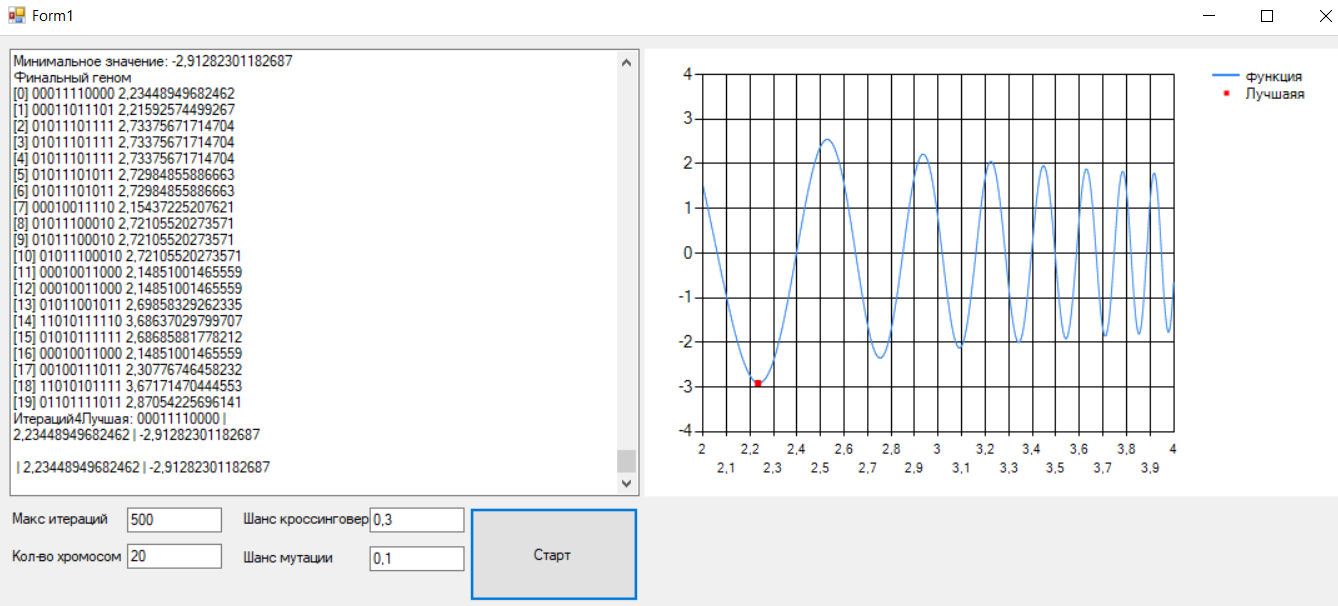


Среднее значение рассчитанного минимума составляет: -2,74.

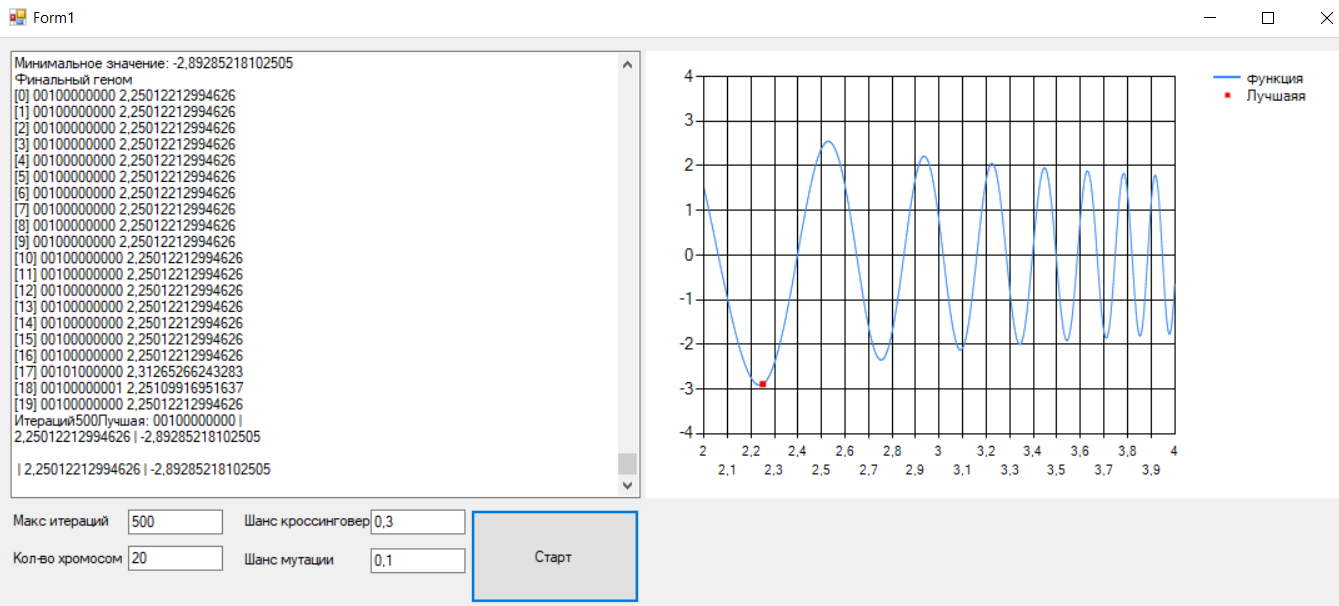
Погрешность = (2,95 – 2,74)/ 2,95 = 0,071.

**20ть хромосом**

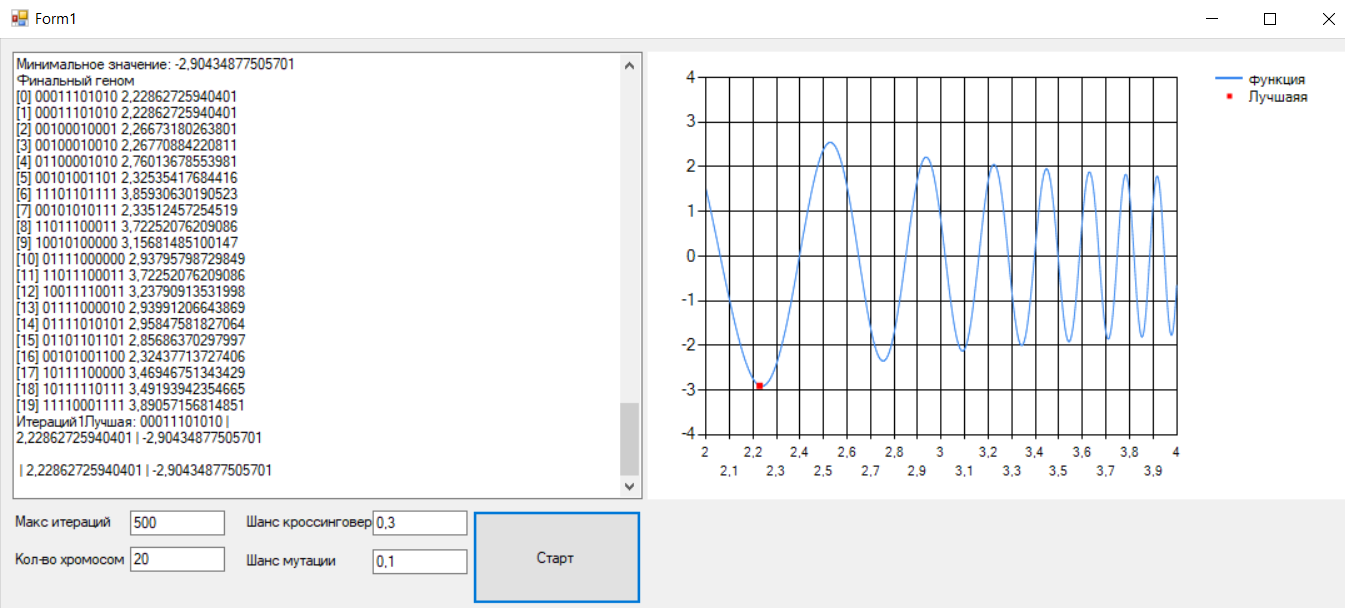
1й запуск

****

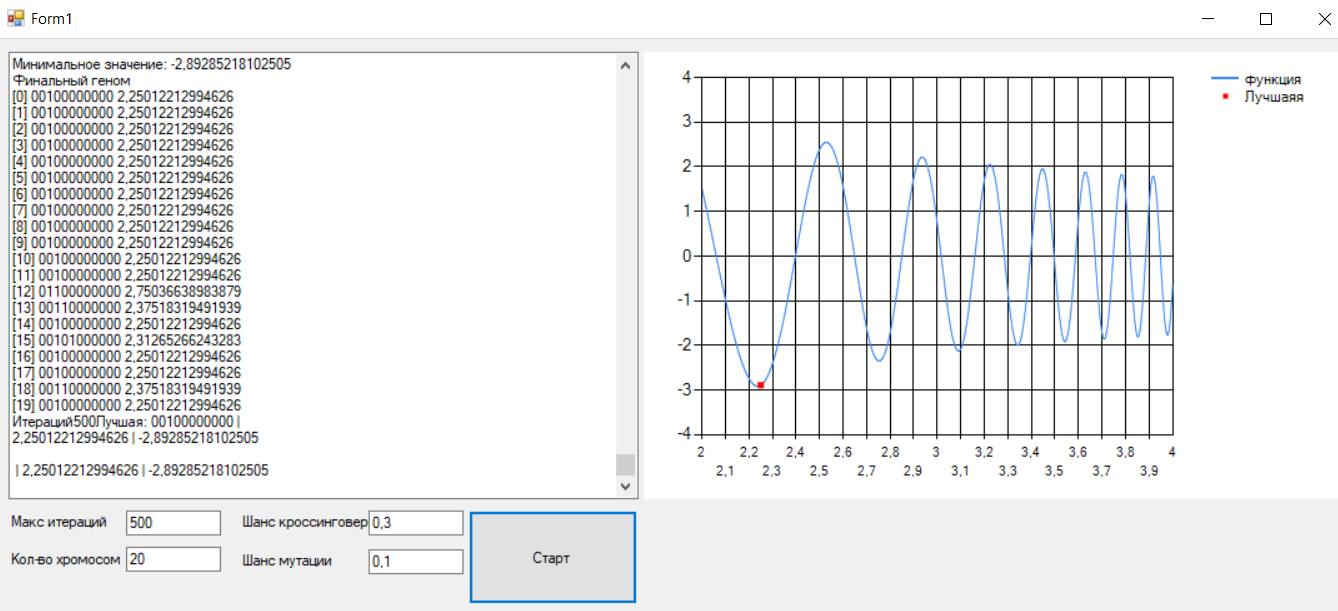
2й запуск



3й запуск



4й запуск



Среднее значение рассчитанного минимума составляет: -2,9.

Погрешность = (2,95 – 2,89)/ 2,95 = 0,02.

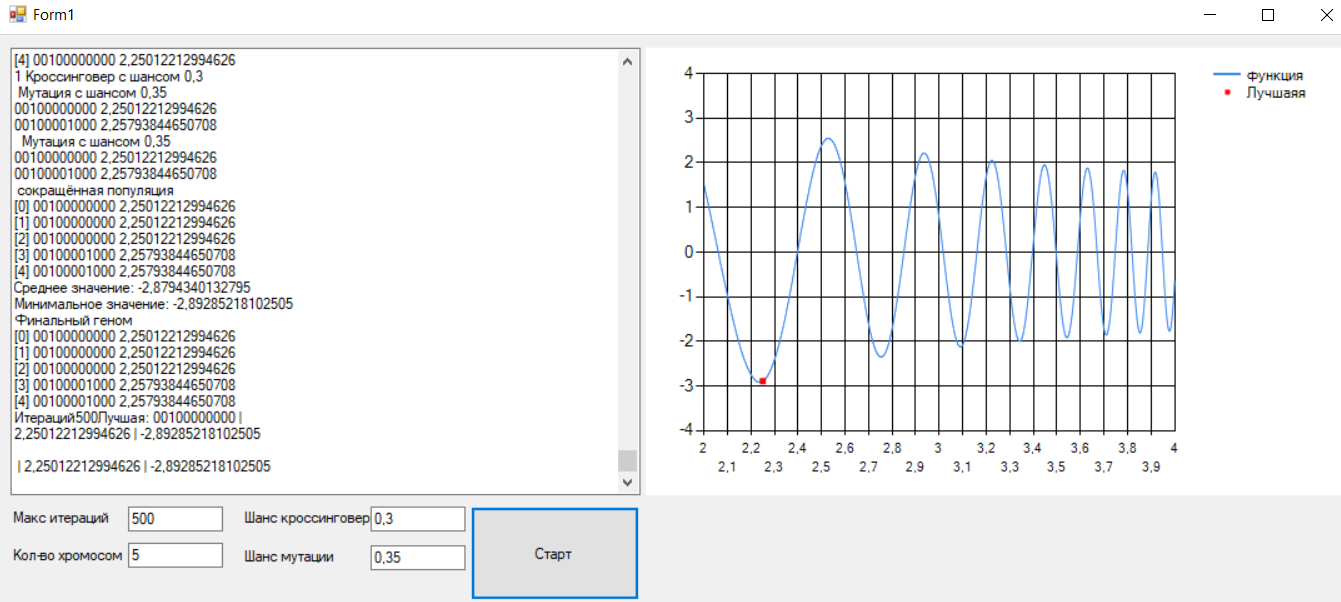
Вывод: увеличение кол-ва хромосом положительно влияет на результаты вычислений.

**Проверка влияния на результаты оператора мутации:**

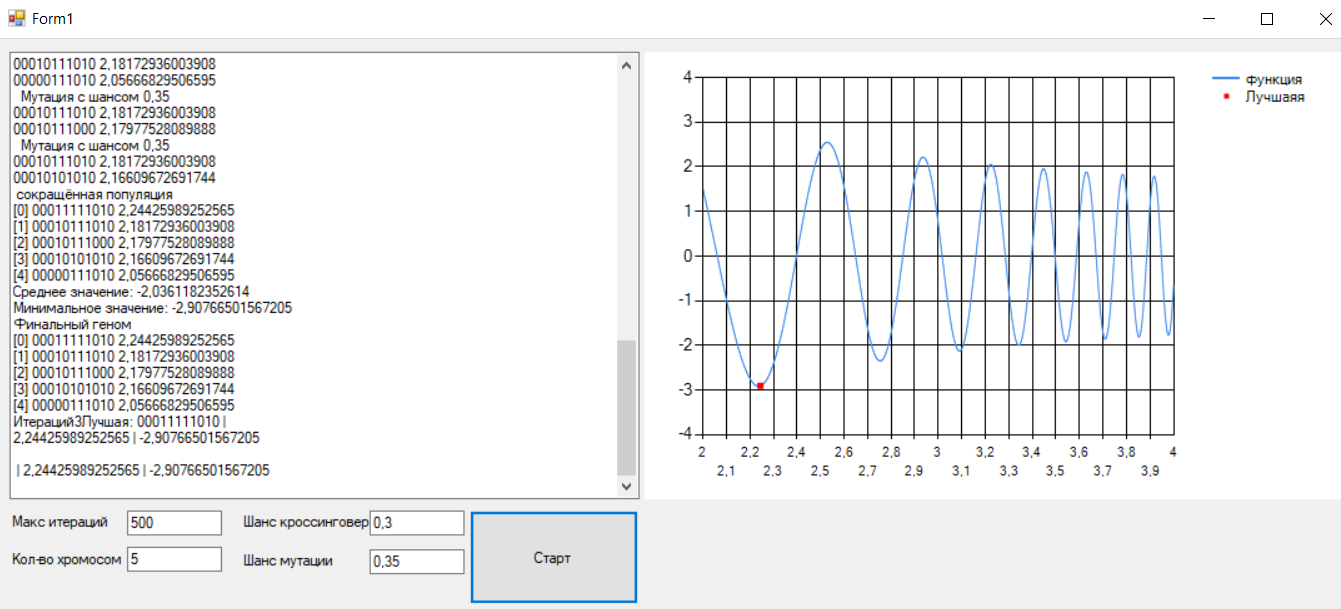
Проверим, увеличится ли точности вычисление при повышении шанса мутации. Для этого возьмём из опробованных количеств хромосом, давшее наихудший результат. В нашем случае, это 5.

1. Ограничение циклов = 500;
2. Кол-во хромосом = 5;
3. Шанс кроссинговера = 0,3;
4. Шанс мутации = 0,35.

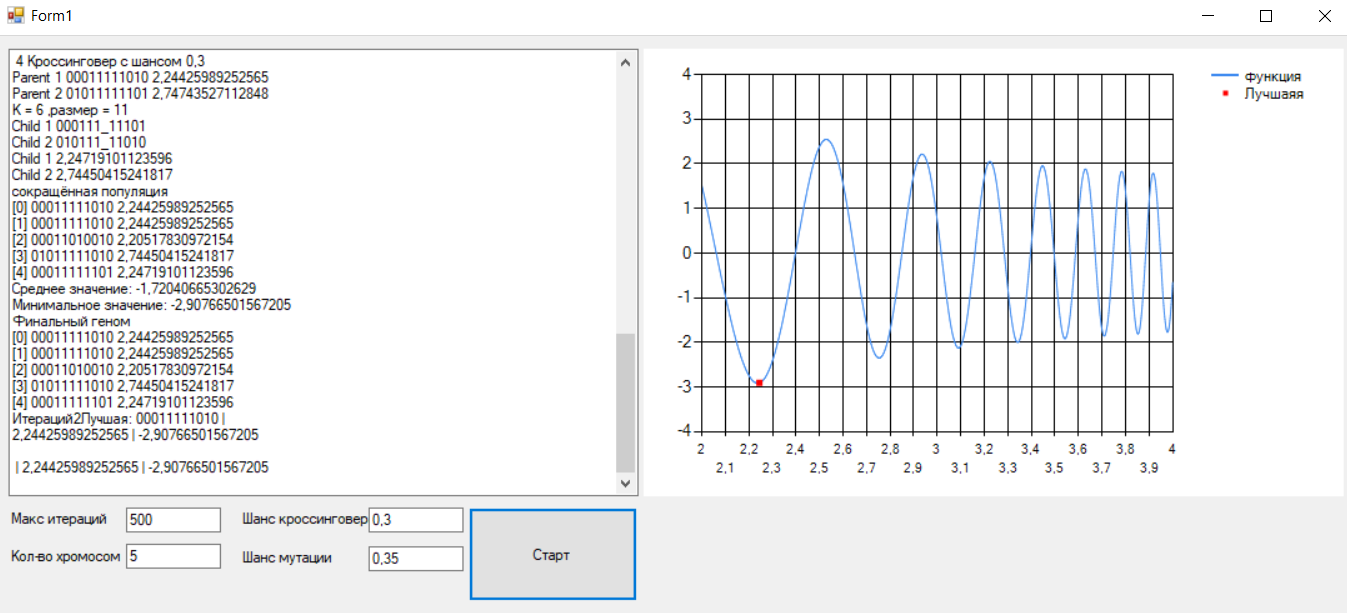
1й запуск:



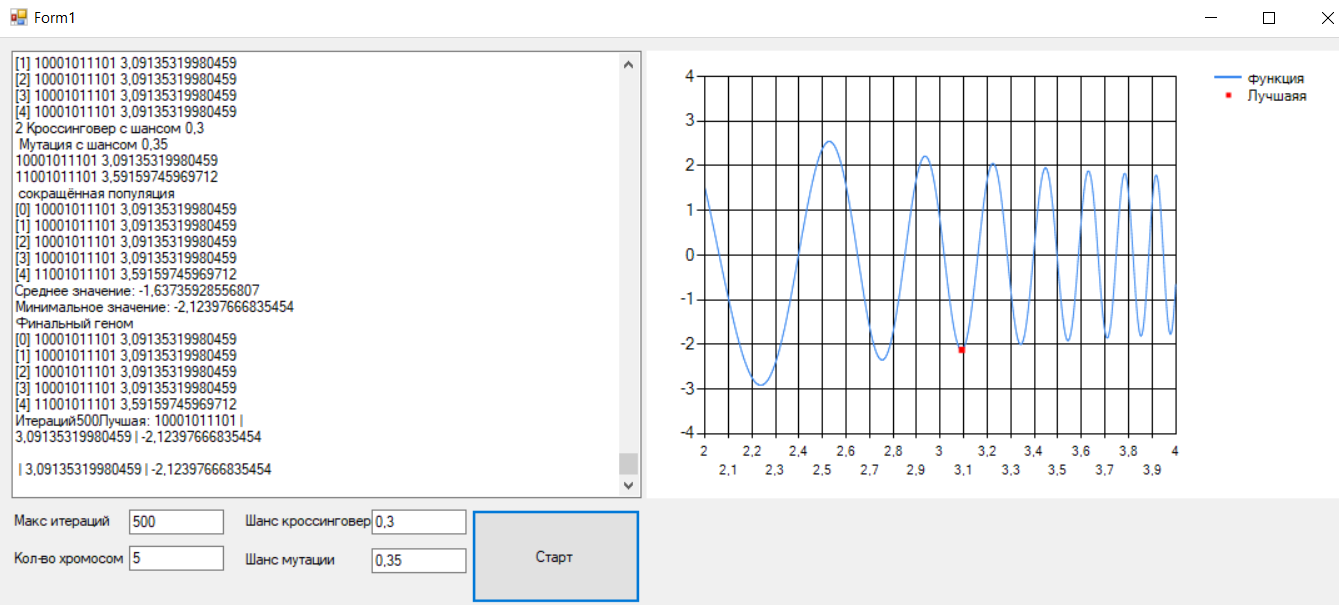
2й запуск:



3й запуск:



4й запуск:



Среднее значение рассчитанного минимума составляет: -2,71.

Погрешность = (2,95 – 2,71)/ 2,95 = 0,08.

Было: 0,64.

Стало: 0,08.

Вывод: повышение шанса мутации положительно повлияло на результаты вычисления при низком количестве хромосом.

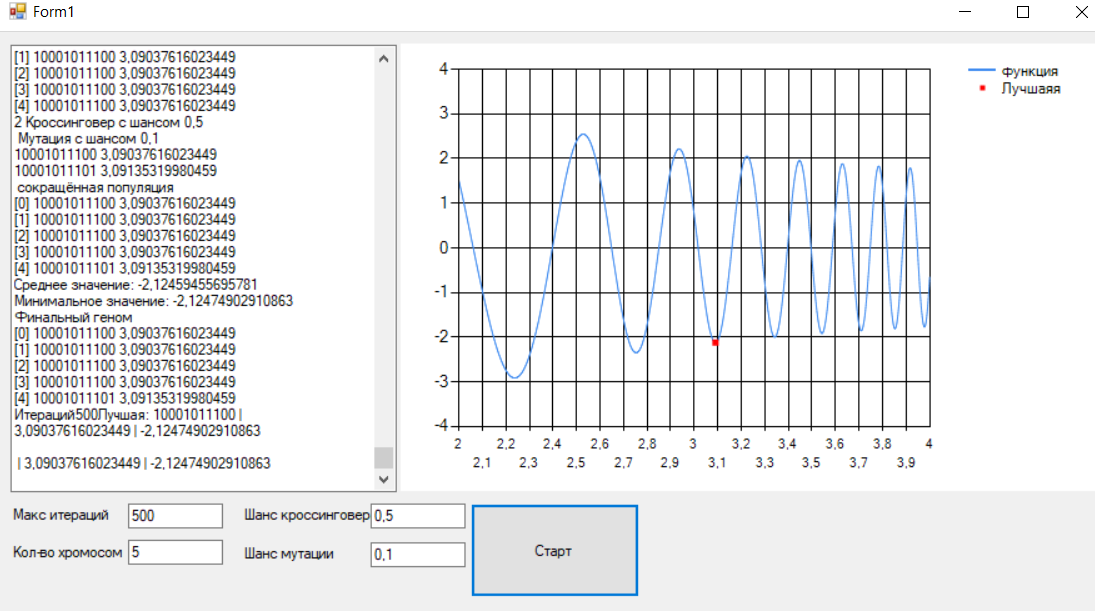
**Проверка влияния на результат оператора кроссинговера:**

Для проверки влияния на результаты увеличение оператора кроссинговера, возьмём следующие его значения: 0,5; 0,9.

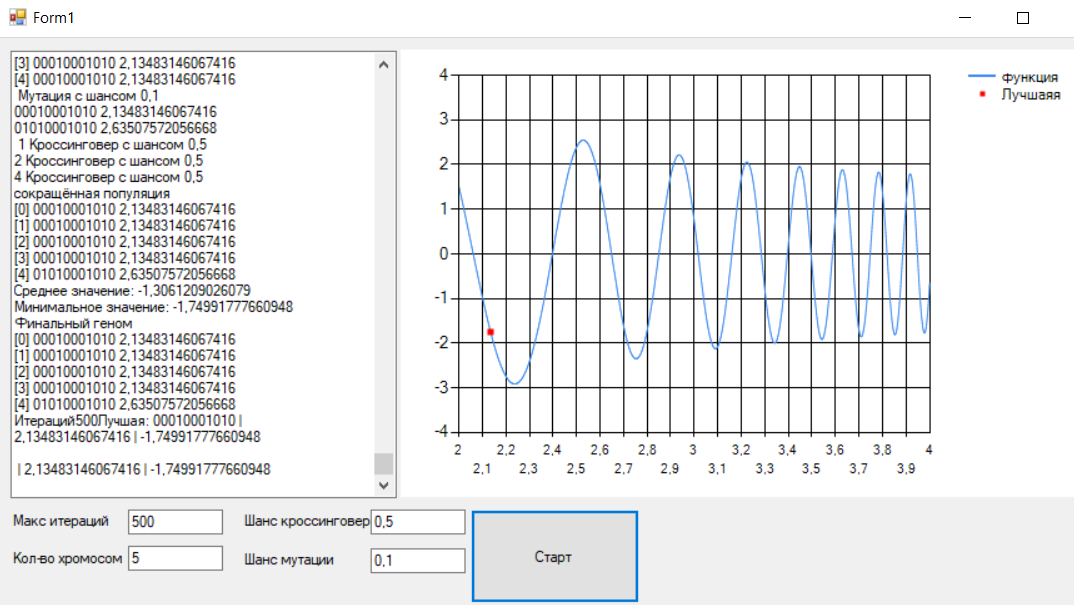
Эксперимент будем проводить на варианте с 5ю хромосомами, который дал наихудший вариант.

**Шанс = 0,5**

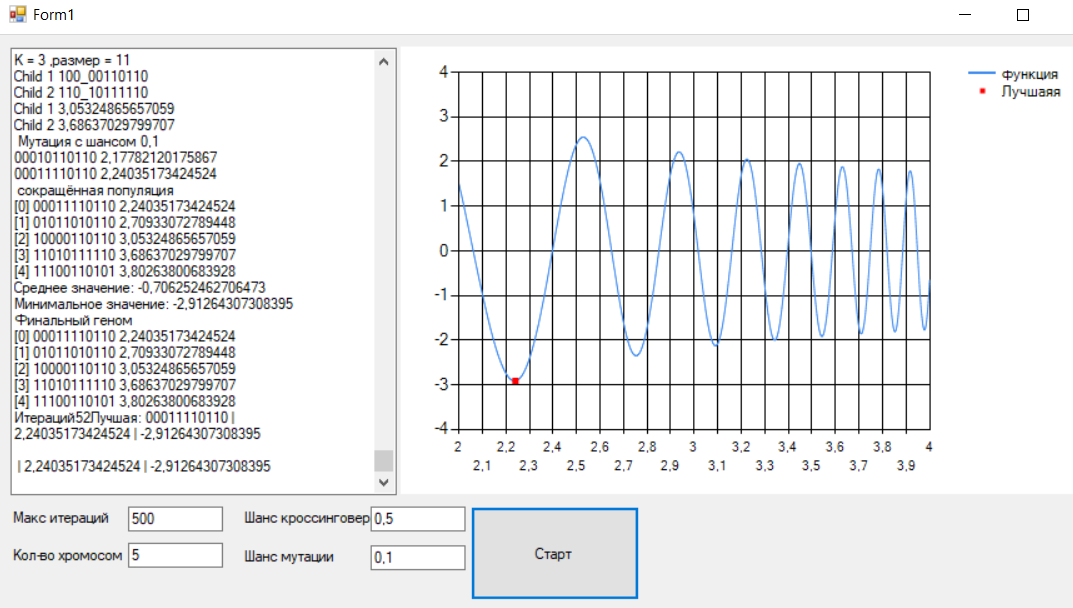
1й запуск:



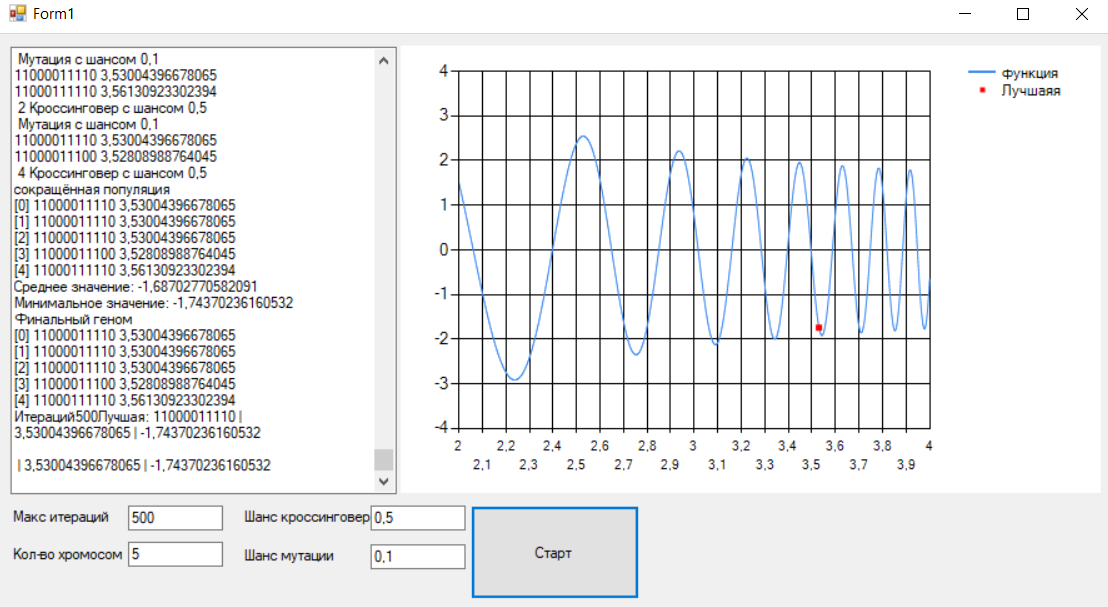
2й запуск:



3й запуск:



4й запуск:



Среднее значение рассчитанного минимума составляет: -2,71.

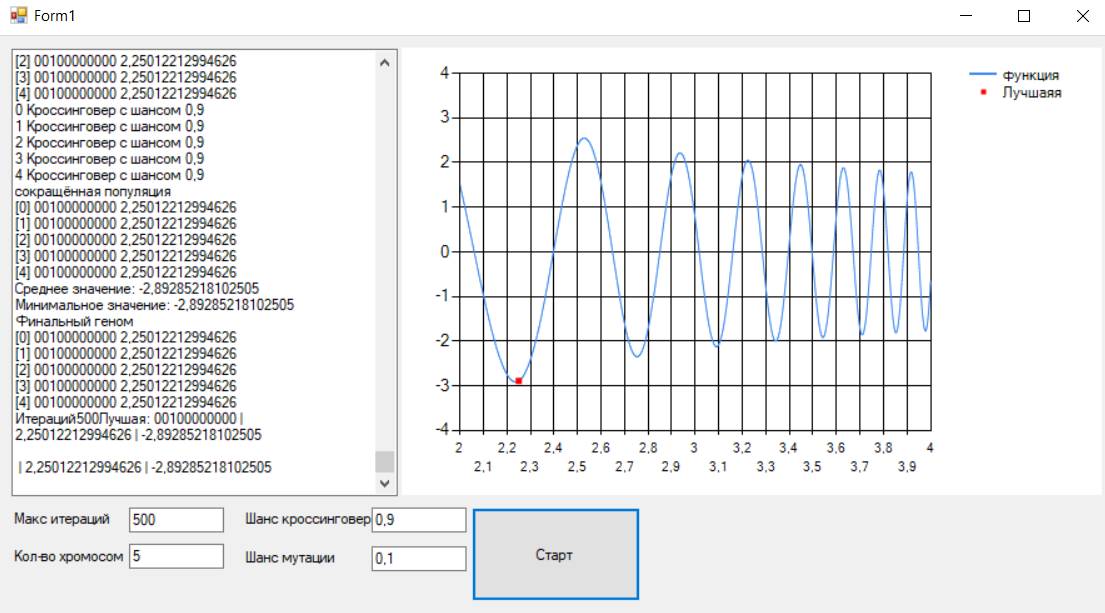
Погрешность = (2,95 – 2,13)/ 2,95 = 0,28.

Было: 0,64.

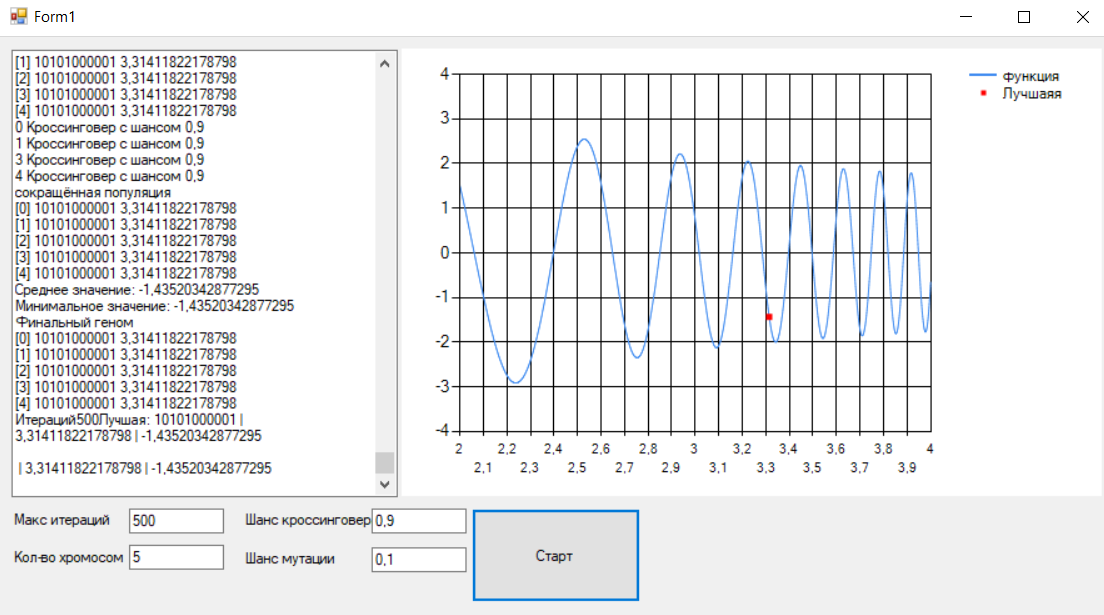
Стало: 0,28.

**Шанс 0,9**

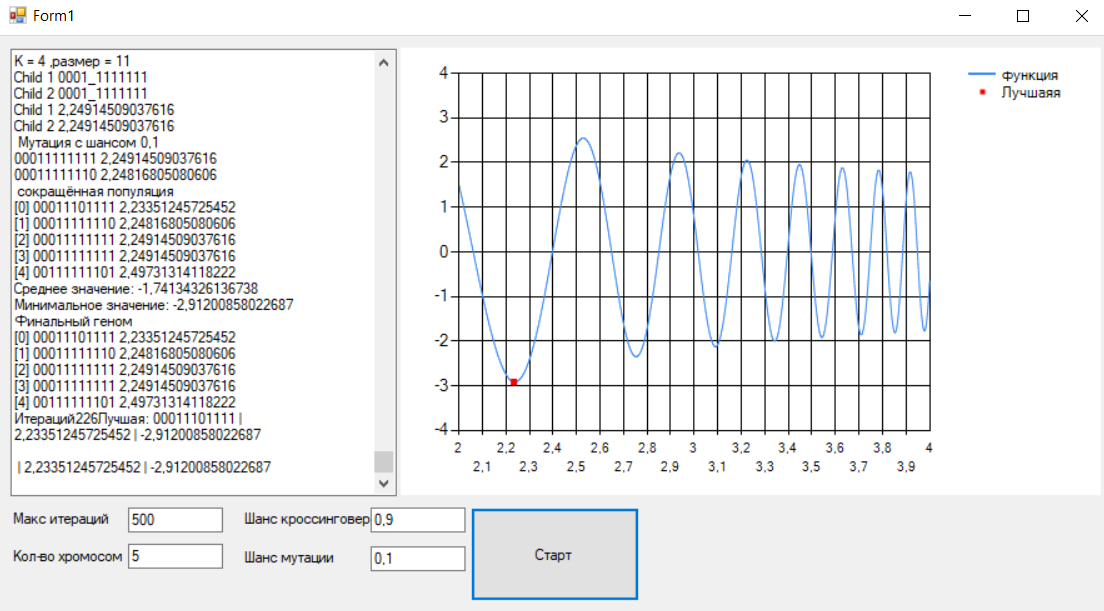
1й запуск:



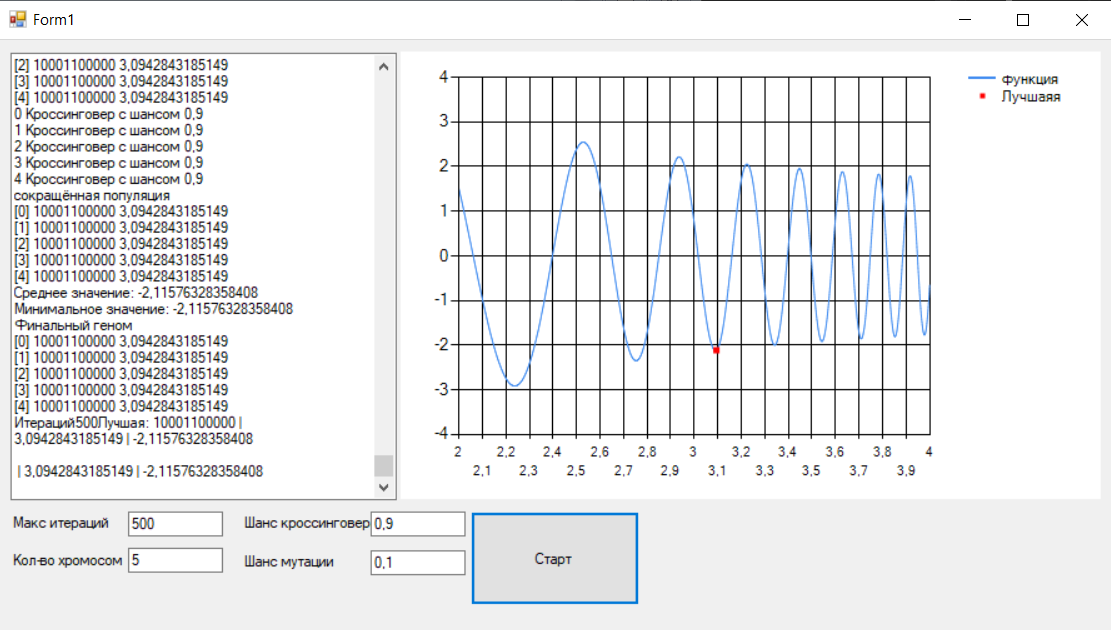
2й запуск:



3й запуск:



4й запуск:



Среднее значение рассчитанного минимума составляет: -2,33.

Погрешность = (2,95 – 2,33)/ 2,95 = 0,21.

Было: 0,28.

Стало: 0,21.

Вывод: увеличение шанса кроссинговера положительно влияет на результат.

**Листинг программного кода:**

Form1.cs  
using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

namespace Лаба\_4\_ИИ

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

GeneticAlgorithm geneticAlgorithm = new GeneticAlgorithm();

private void button1\_Click(object sender, EventArgs e)

{

int kolChromosome = Convert.ToInt32(textBox3.Text);

int kolIteration = Convert.ToInt32(textBox2.Text);

double VerMyta = Convert.ToDouble(textBox6.Text);

double VerCrosengove = Convert.ToDouble(textBox5.Text);

Chromosome components = geneticAlgorithm.Evolution(kolChromosome, kolIteration, VerCrosengove, VerMyta);

double x = components.decVal;

double y = FitnessFunc.Func(x);//components.funcVal;

var random = new Random();

textBox1.Clear();

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

textBox1.AppendText(" | " + x + " | " + y + Environment.NewLine);

chart1.Series.Clear();

chart1.ChartAreas.Clear();

ChartArea chart = new ChartArea("Math functions");

chart.AxisX.Minimum = 2;

chart.AxisX.Maximum = 4;

chart.AxisY.Minimum = -4;

chart.AxisY.Maximum = 4;

chart.AxisY.Interval = 1;

chart.AxisX.Interval = 0.1;

chart1.ChartAreas.Add(chart);

Series mySeriesOfPoint = new Series("функция");

mySeriesOfPoint.ChartType = SeriesChartType.Line;

mySeriesOfPoint.ChartArea = "Math functions";

Series Finn = new Series("Лучшаяя");

Finn.ChartType = SeriesChartType.Point;

Finn.Color = Color.Red;

Finn.ChartArea = "Math functions";

for (double xx = FitnessFunc.minInt; xx <= FitnessFunc.maxInt; xx += 0.001)

{

mySeriesOfPoint.Points.AddXY(xx, FitnessFunc.Func(xx));

}

Finn.Points.AddXY(x, y);

//Добавляем созданный набор точек в Chart

chart1.Series.Add(mySeriesOfPoint);

chart1.Series.Add(Finn);

}

private void button2\_Click(object sender, EventArgs e)

{

double i = getChromosomeValue(textBox7.Text, 0, 20);

textBox8.Text = Convert.ToString(i);

}

public static double getChromosomeValue(string s, double minInterval, double maxInterval)

{

int sum = 0;

double c, znach = 0;

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

{

c = char.GetNumericValue(s[i]);

znach = Math.Pow(2, s.Count() - i - 1);

sum += (int)(c \* znach);

}

return minInterval + sum \* ((maxInterval - minInterval) / (Math.Pow(2, s.Count()) - 1));

}

}

}

Class1:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Лаба\_4\_ИИ

{

class Chromosome

{

public string binVal;

public double decVal;

public double funcVal;

public double p;

public Chromosome(string bin)

{

binVal = bin;

}

public Chromosome(string bin, double dec, double fun)

{

binVal = bin;

decVal = dec;

funcVal = fun;

}

public void ReCreate(string bin)

{

binVal = bin;

decVal = 0;

funcVal = 0;

}

}

class FitnessFunc

{

public static readonly double minInt = 2;

public static readonly double maxInt = 4;

public static double Func(double x)

{

return (Math.Cos(Math.Exp(x)) / Math.Sin(Math.Log10(x)));

}

}

class GeneticAlgorithm

{

private Random random = new Random();

private StringBuilder textAll = new StringBuilder();

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

public GeneticAlgorithm() { }

public Chromosome Evolution(int chromosomesCount, int iterationsCount, double crossingoverChance, double mutationChance)

{

textAll.Clear();

double minInterval = FitnessFunc.minInt;

double maxInterval = FitnessFunc.maxInt;

int intervalsCount = (int)(maxInterval - minInterval) \* 1000;

int chromosomesSize = getChromosomeSize(intervalsCount);

int currentCount = 0;

List<Chromosome> genome = CreateGenome(chromosomesSize, chromosomesCount);

foreach (Chromosome c in genome)

{

c.decVal = getChromosomeValue(c.binVal, minInterval, maxInterval);

}

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

printGenome(genome);

while (currentCount < iterationsCount)

{

textAll.Append("Итерация " + (currentCount + 1) + Environment.NewLine);

Reproduction(genome, minInterval, maxInterval, chromosomesCount);

printGenome(genome);

foreach (Chromosome chromosome in genome)

{

if (random.NextDouble() <= crossingoverChance)

{

textAll.Append(genome.IndexOf(chromosome) + " Кроссинговер с шансом " + crossingoverChance + Environment.NewLine);

Crossingover(genome, chromosome, minInterval, maxInterval);

}

string binary = chromosome.binVal;

if (random.NextDouble() <= mutationChance)

{

textAll.Append(" Мутация с шансом " + mutationChance + Environment.NewLine);

textAll.Append(chromosome.binVal + " " + chromosome.decVal + Environment.NewLine);

chromosome.binVal = Mutation(binary);

chromosome.decVal = getChromosomeValue(chromosome.binVal, minInterval, maxInterval);

chromosome.funcVal = FitnessFunc.Func(chromosome.decVal);

textAll.Append(chromosome.binVal + " " + chromosome.decVal + Environment.NewLine + " ");

}

}

Chromosome cr;

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

{

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

{

if (genome[i].funcVal < genome[j].funcVal)

{

cr = genome[i];

genome[i] = genome[j];

genome[j] = cr;

}

}

}

if (genome.Count > chromosomesCount)

{

for (int i = genome.Count - 1; i > chromosomesCount - 1; i--)

{

genome.RemoveAt(i);

}

}

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

printGenome(genome);

textAll.Append("Среднее значение: " + genome.Average(g => g.funcVal) + Environment.NewLine + "Минимальное значение: " + genome.Min(g => g.funcVal) + Environment.NewLine);

currentCount++;

}

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

printGenome(genome);

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

Random rand = new Random();

double ran, d;

//5,1857390269396 | 43,1771087167325

textAll.Append("Лучшая: " + best.binVal + " | " + Environment.NewLine + best.decVal + " | " + best.funcVal + Environment.NewLine);

//textAll.Append("Лучшая: " + best.binVal + " | " + nom1 + ran + " | " + nom2+ran);

return best;

}

private string Mutation(string chromosome)

{

StringBuilder newChromosome = new StringBuilder(chromosome);

int i = random.Next(chromosome.Count());

newChromosome[i] = (newChromosome[i] == '0') ? '1' : '0';

return newChromosome.ToString();

}

private void Reproduction(List<Chromosome> genome, double minInterval, double maxInterval, int chromosomesCount)

{

double fValuesSum = 0, a = 0, b = 0;

foreach (Chromosome c in genome)

{

double funcVal = FitnessFunc.Func(getChromosomeValue(c.binVal, minInterval, maxInterval));

c.funcVal = funcVal;

fValuesSum += funcVal;

}

double sum;

List<Chromosome> newGenome = new List<Chromosome>();

textAll.Append("Репродукция:" + Environment.NewLine);

Chromosome cr;

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

{

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

{

if (genome[i].funcVal < genome[j].funcVal)

{

cr = genome[i];

genome[i] = genome[j];

genome[j] = cr;

}

}

}

Random rnd = new Random();

double[] mass22 = new double[genome.Count];

int[] mass11 = new int[5];

int[] rez = new int[genome.Count];

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

{

for (int j = 0; j < mass11.Length; j++)

{

mass11[j] = rnd.Next(0, genome.Count);

}

mass22[i] = genome[mass11[0]].funcVal;

rez[i] = mass11[0];

for (int j = 1; j < mass11.Length; j++)

{

if (mass22[i] > mass22[mass11[j]])

{

mass22[j] = mass22[mass11[j]];

rez[i] = mass11[j];

}

}

}

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

{

{

;

newGenome.Add(new Chromosome(genome[rez[i]].binVal, genome[rez[i]].decVal, genome[rez[i]].funcVal));

}

}

genome.Clear();

genome.AddRange(newGenome);

}

private void Crossingover(List<Chromosome> genome, Chromosome parent1, double minInterval, double maxInterval)

{

Chromosome parent2 = getParents(genome, parent1);

if (parent2 != null)

{

textAll.Append("Parent 1 " + parent1.binVal + " " + parent1.decVal + Environment.NewLine);

textAll.Append("Parent 2 " + parent2.binVal + " " + parent2.decVal + Environment.NewLine);

GetChildBySinglePointCrossingover(parent1, parent2, minInterval, maxInterval);

}

}

private string CreateBinaryModel(int size)

{

StringBuilder value = new StringBuilder();

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

value.Append(random.NextDouble() <= 0.5 ? '1' : '0');

return value.ToString();

}

private List<Chromosome> CreateGenome(int size, int count)

{

List<Chromosome> genome = new List<Chromosome>();

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

genome.Add(new Chromosome(CreateBinaryModel(size)));

return genome;

}

private Chromosome getParents(List<Chromosome> genome, Chromosome parent1)

{

bool contains = false;

foreach (Chromosome c in genome)

if (c.Equals(parent1))

{

contains = true;

break;

}

if (contains)

{

Chromosome parent2;

int[] mass = new int[genome.Count];

int count = 0;

double tmp1, tmp2;

bool b = false;

tmp1 = Math.Abs(parent1.funcVal - genome[0].funcVal);

for (int i = 1; i < genome.Count; i++)

{

parent2 = genome[i];

tmp2 = Math.Abs(parent1.funcVal - parent2.funcVal);

if (tmp2 > 0 && tmp1 < tmp2)

{

b = true;

tmp1 = parent2.funcVal;

count = i;

}

}

if (b)

{

return genome[count];

}

else return null;

}

else

return null;

}

private int checkbin(string str1, string str2)

{

int count = 0;

for (int i = 0; i < str1.Length; i++)

{

if (str1[i] != str2[i])

count++;

}

return count;

}

private void GetChildBySinglePointCrossingover(Chromosome parent1, Chromosome parent2, double minInterval, double maxInterval)

{

int k = random.Next(parent1.binVal.Length);

textAll.Append("K = " + k + " ,размер = " + parent1.binVal.Count() + Environment.NewLine);

StringBuilder child1 = new StringBuilder();

StringBuilder child2 = new StringBuilder();

child1.Append(parent1.binVal.Substring(0, k));

child1.Append(parent2.binVal.Substring(k, parent2.binVal.Length - k));

child2.Append(parent2.binVal.Substring(0, k));

child2.Append(parent1.binVal.Substring(k, parent1.binVal.Length - k));

textAll.Append("Child 1 " + parent1.binVal.Substring(0, k) + "\_" + parent2.binVal.Substring(k, parent2.binVal.Length - k) + Environment.NewLine);

textAll.Append("Child 2 " + parent2.binVal.Substring(0, k) + "\_" + parent1.binVal.Substring(k, parent1.binVal.Length - k) + Environment.NewLine);

parent1.ReCreate(child1.ToString());

parent2.ReCreate(child2.ToString());

parent1.decVal = getChromosomeValue(child1.ToString(), minInterval, maxInterval);

parent2.decVal = getChromosomeValue(child2.ToString(), minInterval, maxInterval);

textAll.Append("Child 1 " + parent1.decVal + Environment.NewLine);

textAll.Append("Child 2 " + parent2.decVal + Environment.NewLine);

}

public void printGenome(List<Chromosome> genome)

{

int i = 0;

foreach (Chromosome c in genome)

{

textAll.Append("[" + i++ + "] " + c.binVal + " " + c.decVal + Environment.NewLine);

}

}

public static int getChromosomeSize(int num)

{

int size = 0;

while (num > 0)

{

num >>= 1;

size++;

}

return size;

}

public static double getChromosomeValue(string s, double minInterval, double maxInterval)

{

int sum = 0;

double c, znach = 0;

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

{

c = char.GetNumericValue(s[i]);

znach = Math.Pow(2, s.Count() - i - 1);

sum += (int)(c \* znach);

}

//sum = sum;

return minInterval + sum \* ((maxInterval - minInterval) / (Math.Pow(2, s.Count()) - 1));

}

}

}