2-Igor Kondrashuk

Minimization of Test Function

High-dimensional Test Function

Problem is minimize or maximize a n-dimensinal function

$y = f(x_1; x_2; x_3; \cdot \cdot \cdot ; x_n)$

We may use chromosomes with n genes of real value, such as

(0:32;−0:51; 0:48; · · · ;−0:93)

For example, minimize

$y = x^2_1 + x^2_1 + x^2_1 + \cdot \cdot \cdot + x^2_{20}$

Rastrigin's Function

Exercise 8 1. Minimize the following y in (i) 20-D, (ii) 3-D and (iii) 2-D cases.

y = nA +Σni=1(x2i− Acos(2_xi))

2. Show the following graphics in each of 3 cases (i), (ii) and (iii).

(1) the graph of _tness vs generation.

(2) Create a population of 20 chromosomes at random, with _tness being y.

Task1

Source code

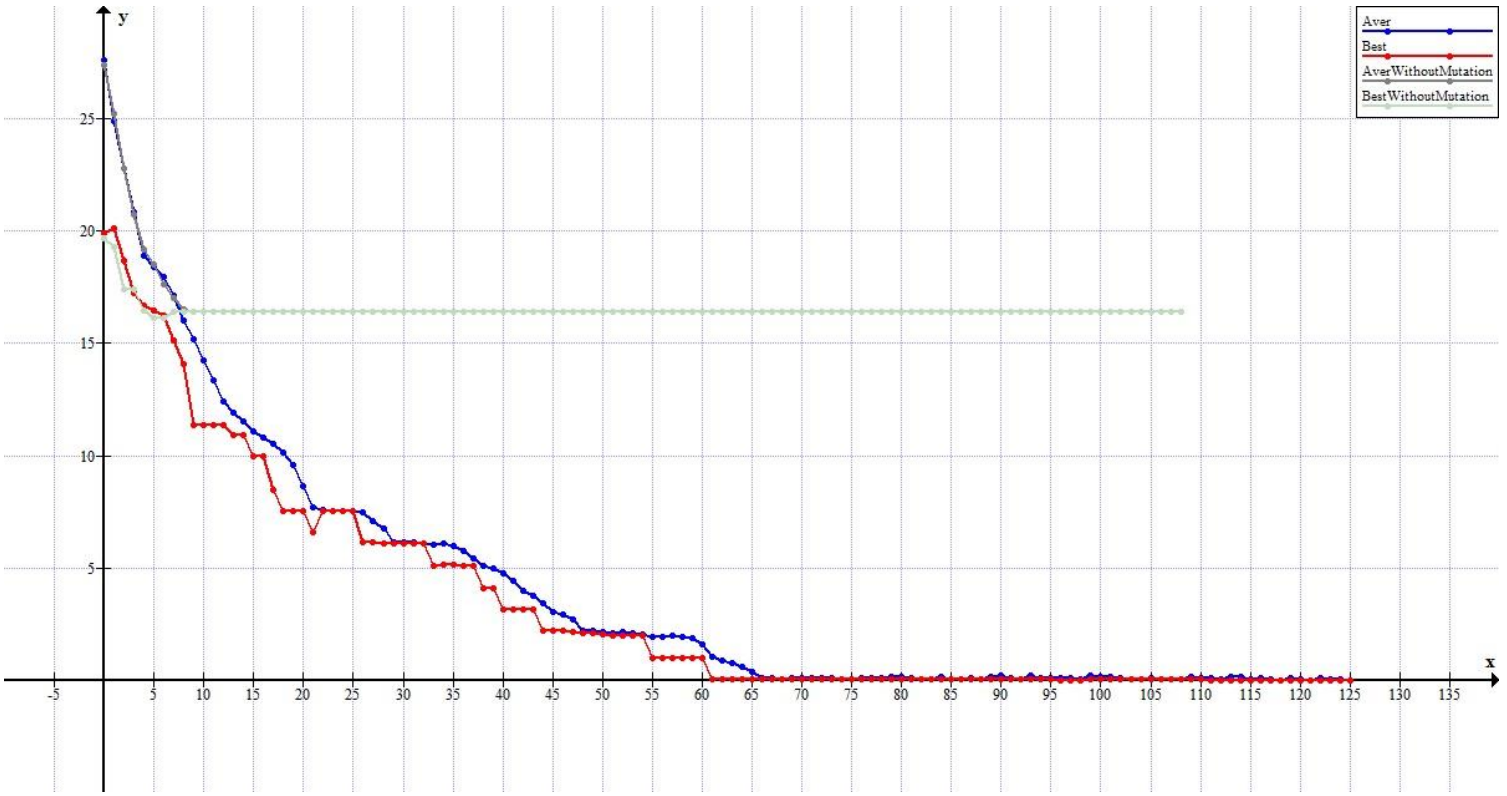
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab3_siit_
{
   class Dot
   {
      public List<double> chromosomes;
      public double fitness;
      public Dot(List<double> chromosomes)
      {
         this.chromosomes = chromosomes;
         fitness = getFitness(chromosomes);
      }
      private double getFitness(List<double> chromosomes)
      {
         double fitness = chromosomes.Count;
         for (int i=0;i<chromosomes.Count;i++)
         {
            double temp = chromosomes[i]* chromosomes[i] - Math.Cos(2 * Math.PI * chromosomes[i]);
            fitness += temp;
         }
         return fitness;
      }
   }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab3_siit_
{
   class Population
   {
      public List<Dot> dots;
      public double theBestFitness;
      public double avarageFitness;
```

```
        public Population(List<Dot> dogs)
        {
            this.dots = dogs;
            theBestFitness = getTheBestFitness(dogs);
            avarageFitness = getAvarageFitness(dogs);
        }
        public double getTheBestFitness(List<Dot> dogs)
        {
            double bestFitness = 9999;
            for (int i = 0; i < dogs.Count; i++)
                if (bestFitness > dogs[i].fitness)
                    bestFitness = dogs[i].fitness;
            return bestFitness;
        }
        public double getAvarageFitness(List<Dot> dogs)
        {
            double avarageFitness = 0;
            for (int i = 0; i < dogs.Count; i++)
                avarageFitness += dogs[i].fitness;
            avarageFitness /= dogs.Count;
            return avarageFitness;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace lab3_siit_
{
    class GA
    {
        Random random;
        List<Population> historyOfPopulation;
        public GA()
        {
            random = new Random();
            List<Dot> dots = new List<Dot>();
            for (int i = 0; i < 20; i++)
            {
                List<double> chromosomes = new List<double>();
                for (int j = 0; j < 20; j++)
                {
                    chromosomes.Add(random.NextDouble() % 2 -1);
                }
                dots.Add(new Dot(chromosomes));
            }
            //Sorting by fitness
            for (int i = 0; i < dots.Count; i++)
            {
                for (int j = dots.Count - 1; j > i; j--)
                {
                    if (dots[j].fitness > dots[j - 1].fitness)
                    {
                        Dot tempDot = dots[j];
                        dots[j] = dots[j - 1];
                        dots[j - 1] = tempDot;
                    }
                }
            }
            Population startPopulation = new Population(dots);
            historyOfPopulation = new List<Population>();
            historyOfPopulation.Add(startPopulation);
            int k = 0;
            while (!isReady(historyOfPopulation))
            {
```

```
        Thread.Sleep(10);
        historyOfPopulation.Add(getNextPupulation(historyOfPopulation[k]));
        k++;
    }
}
private Population getNextPupulation(Population parentPopulation)
{
    List<Dot> childrenPopulationDots = new List<Dot>();

    for (int i = 0; i < 10; i++)
    {
        List<Dot> childrenDots = getChildren(
            parentPopulation.dots[random.Next() % 10 + 10],
            parentPopulation.dots[random.Next() % 10 + 10]
            );
        childrenPopulationDots.AddRange(childrenDots);
    }
    //Sorting by fitness
    for (int i = 0; i < childrenPopulationDots.Count; i++)
    {
        for (int j = childrenPopulationDots.Count - 1; j > i; j--)
        {
            if (childrenPopulationDots[j].fitness > childrenPopulationDots[j - 1].fitness)
            {
                Dot tempDot = childrenPopulationDots[j];
                childrenPopulationDots[j] = childrenPopulationDots[j - 1];
                childrenPopulationDots[j - 1] = tempDot;
            }
        }
    }
    Population childrenPopulation = new Population(childrenPopulationDots);
    return childrenPopulation;
}

Boolean isReady(List<Population> historyOfPopulation)
{
    if (historyOfPopulation.Count < 100)
        return false;
    else
    {
        for (int i = historyOfPopulation.Count - 100; i < historyOfPopulation.Count; i++)
        {
            if (historyOfPopulation[historyOfPopulation.Count - 100].avarageFitness != historyOfPopulation[i].avarageFitness)
                return false;
        }
        return true;
    }
}
private List<Dot> getChildren(Dot father, Dot mother)
{
    List<Dot> childrenDots = new List<Dot>();
    int pointCross = random.Next()%20;
    List<double> firstChromosomes = new List<double>();
    List<double> secondChromosomes = new List<double>();
    for (int j = 0; j < 20; j++)
    {
        if(j<pointCross)
        {
            firstChromosomes.Add(father.chromosomes[j]);
            secondChromosomes.Add(mother.chromosomes[j]);
        }
        else
        {
            firstChromosomes.Add(mother.chromosomes[j]);
            secondChromosomes.Add(father.chromosomes[j]);
        }
    }

    childrenDots.Add(new Dot(firstChromosomes));
```

```
            childrenDots.Add(new Dot(secondChromosomes));

            //Mutation
            for (int j = 0; j < childrenDots.Count; j++)
            {
                int prob = random.Next(0, 20);
                if (prob == 7)
                {
                    int number = random.Next(20);
                    childrenDots[j].chromosomes[number] = random.NextDouble() % 2 - 1;
                }
            }
            return childrenDots;
        }
    }
}
```



Best fitness with mutation equal 0;

Best fitness without mutation equal 11,32;

Task2

Rastrigin's Function

Exercise 8 1. Minimize the following y in (i) 20-D, (ii) 3-D and (iii) 2-D cases.

$y = nA + \Sigma ni=1(x2i - A\cos(2\_xi))$

Where A and n equal 1;

2. Show the following graphics in each of 3 cases (i), (ii) and (iii).

(1) the graph of _tness vs generation.

(2) Create a population of 20 chromosomes at random, with _tness being y.

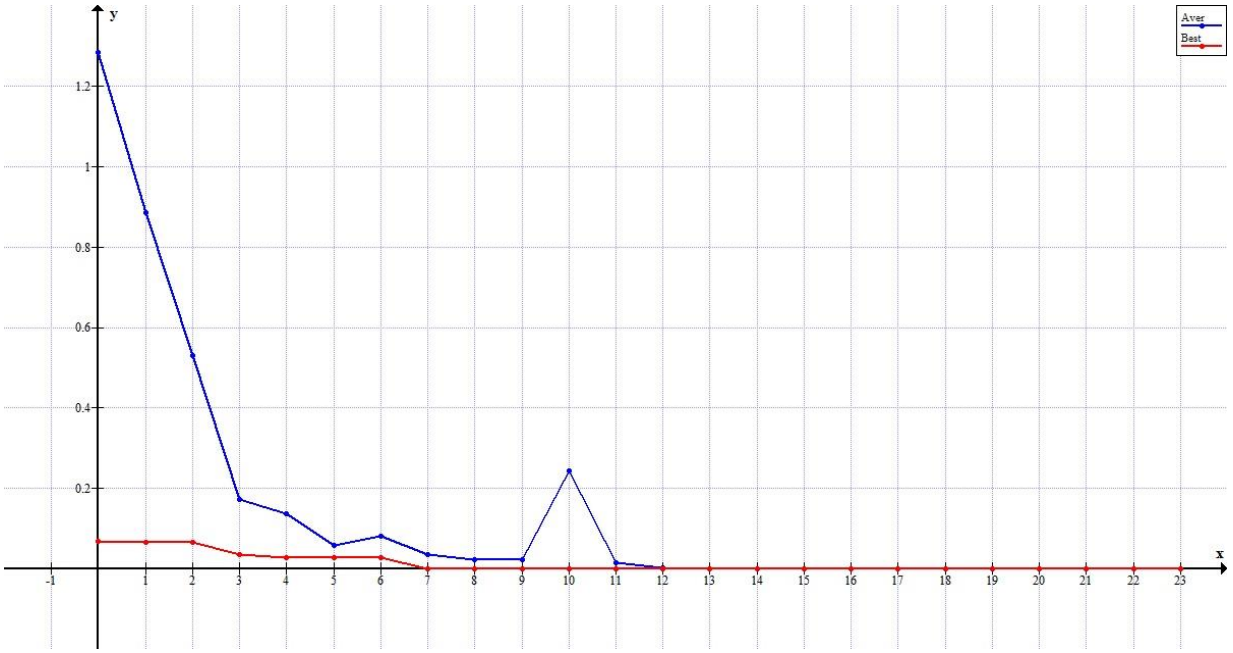
Source code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab3_siit_
```

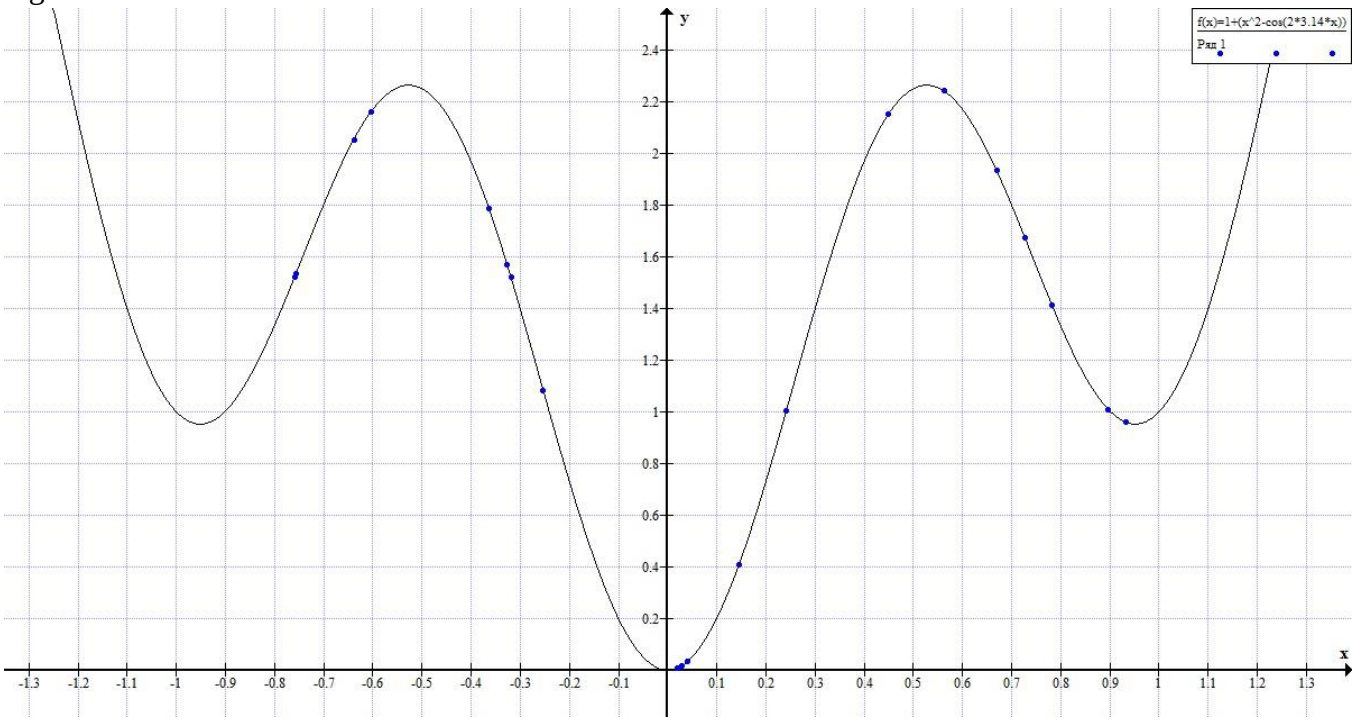
```
    class Dot
    {
        public List<double> chromosomes;
        public double fitness;
        public Dot(List<double> chromosomes)
        {
            this.chromosomes = chromosomes;
            fitness = getFitness(chromosomes);
        }
        private double getFitness(List<double> chromosomes)
        {
            double fitness = chromosomes.Count;
            double x=0;
            for (int i = 1; i < chromosomes.Count; i++)
                if (chromosomes[i] == 1)
                    x += Math.Pow(2, i - 1);
            x /= 1023;
            if (chromosomes[0] == 0)
                x *= -1;
            return 1 + (x * x - Math.Cos(Math.PI * 2 * x));
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab3_siit_
{
    class Population
    {
        public List<Dot> dots;
        public double theBestFitness;
        public double avarageFitness;
        public Population(List<Dot> dogs)
        {
            this.dots = dogs;
            theBestFitness = getTheBestFitness(dogs);
            avarageFitness = getAvarageFitness(dogs);
        }
        public double getTheBestFitness(List<Dot> dogs)
        {
            double bestFitness = 9999;
            for (int i = 0; i < dogs.Count; i++)
                if (bestFitness > dogs[i].fitness)
                    bestFitness = dogs[i].fitness;
            return bestFitness;
        }
        public double getAvarageFitness(List<Dot> dogs)
        {
            double avarageFitness = 0;
            for (int i = 0; i < dogs.Count; i++)
                avarageFitness += dogs[i].fitness;
            avarageFitness /= dogs.Count;
            return avarageFitness;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace lab3_siit_
{
    class GA
    {
        Random random;
        List<Population> historyOfPopulation;
        public GA()
        {
            random = new Random();
            List<Dot> dots = new List<Dot>();
            for (int i = 0; i < 20; i++)
            {
                List<double> chromosomes = new List<double>();
                for (int j = 0; j < 11; j++)
                {
                    chromosomes.Add(random.Next() % 2);
                }
                dots.Add(new Dot(chromosomes));
            }
            //Sorting by fitness
            for (int i = 0; i < dots.Count; i++)
            {
                for (int j = dots.Count - 1; j > i; j--)
                {
                    if (dots[j].fitness > dots[j - 1].fitness)
                    {
                        Dot tempDot = dots[j];
                        dots[j] = dots[j - 1];
                        dots[j - 1] = tempDot;
                    }
                }
            }
            Population startPopulation = new Population(dots);
            historyOfPopulation = new List<Population>();
            historyOfPopulation.Add(startPopulation);
            int k = 0;
            while (!isReady(historyOfPopulation))
            {
                Thread.Sleep(10);
                historyOfPopulation.Add(getNextPupulation(historyOfPopulation[k]));
                k++;
            }
        }
        private Population getNextPupulation(Population parentPopulation)
        {
            List<Dot> childrenPopulationDots = new List<Dot>();

            for (int i = 0; i < 10; i++)
            {
                List<Dot> childrenDots = getChildren(
                    parentPopulation.dots[random.Next() % 10 + 10],
                    parentPopulation.dots[random.Next() % 10 + 10]
                    );
                childrenPopulationDots.AddRange(childrenDots);
            }
            //Sorting by fitness
            for (int i = 0; i < childrenPopulationDots.Count; i++)
            {
                for (int j = childrenPopulationDots.Count - 1; j > i; j--)
                {
                    if (childrenPopulationDots[j].fitness > childrenPopulationDots[j - 1].fitness)
                    {
                        Dot tempDot = childrenPopulationDots[j];
                        childrenPopulationDots[j] = childrenPopulationDots[j - 1];
                        childrenPopulationDots[j - 1] = tempDot;
                    }
                }
            }
            Population childrenPopulation = new Population(childrenPopulationDots);
```

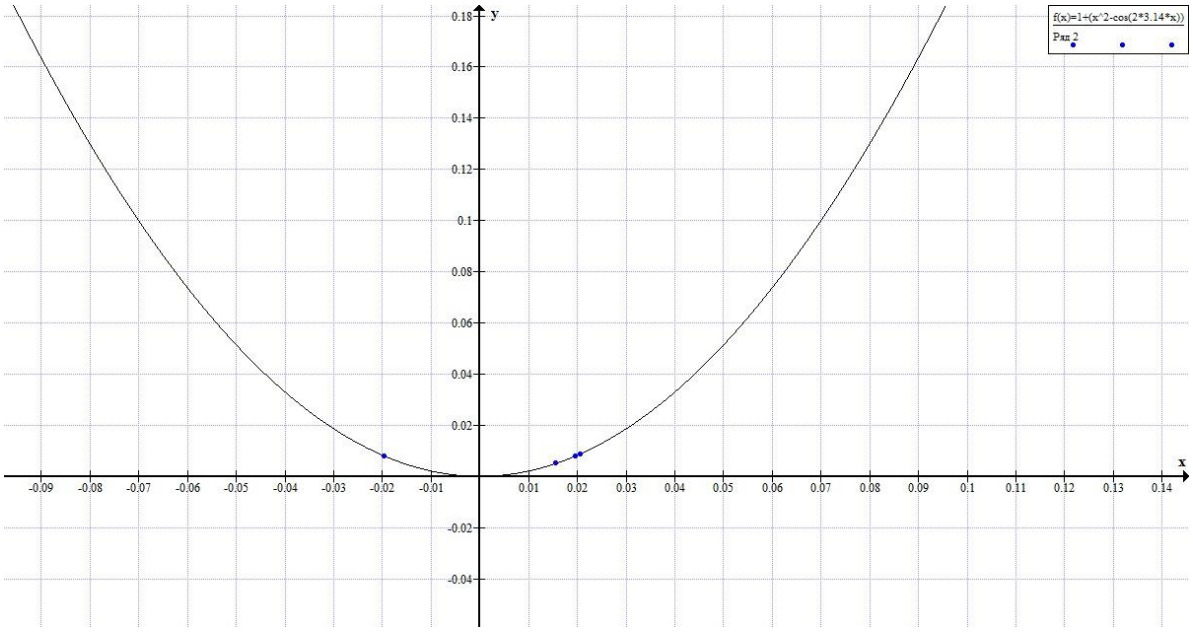
```
        return childrenPopulation;
    }

    Boolean isReady(List<Population> historyOfPopulation)
    {
        if (historyOfPopulation.Count < 100)
            return false;
        else
        {
            for (int i = historyOfPopulation.Count - 100; i < historyOfPopulation.Count; i++)
            {
                if (historyOfPopulation[historyOfPopulation.Count - 100].avarageFitness != historyOfPopulation[i].avarageFitness)
                    return false;
            }
            return true;
        }
    }
    private List<Dot> getChildren(Dot father, Dot mother)
    {
        List<Dot> childrenDots = new List<Dot>();
        int pointCross = random.Next()%20;
        List<double> firstChromosomes = new List<double>();
        List<double> secondChromosomes = new List<double>();
        for (int j = 0; j < 11; j++)
        {
            if(j<pointCross)
            {
                firstChromosomes.Add(father.chromosomes[j]);
                secondChromosomes.Add(mother.chromosomes[j]);
            }
            else
            {
                firstChromosomes.Add(mother.chromosomes[j]);
                secondChromosomes.Add(father.chromosomes[j]);
            }
        }

        //Mutation
        for (int j = 0; j < childrenDots.Count; j++)
        {
            int prob = random.Next(0, 20);
            if (prob == 7)
            {
                int number = random.Next(20);
                childrenDots[j].chromosomes[number] = random.NextDouble() % 2 - 1;
            }
        }

        childrenDots.Add(new Dot(firstChromosomes));
        childrenDots.Add(new Dot(secondChromosomes));
        return childrenDots;
    }
  }
}
```



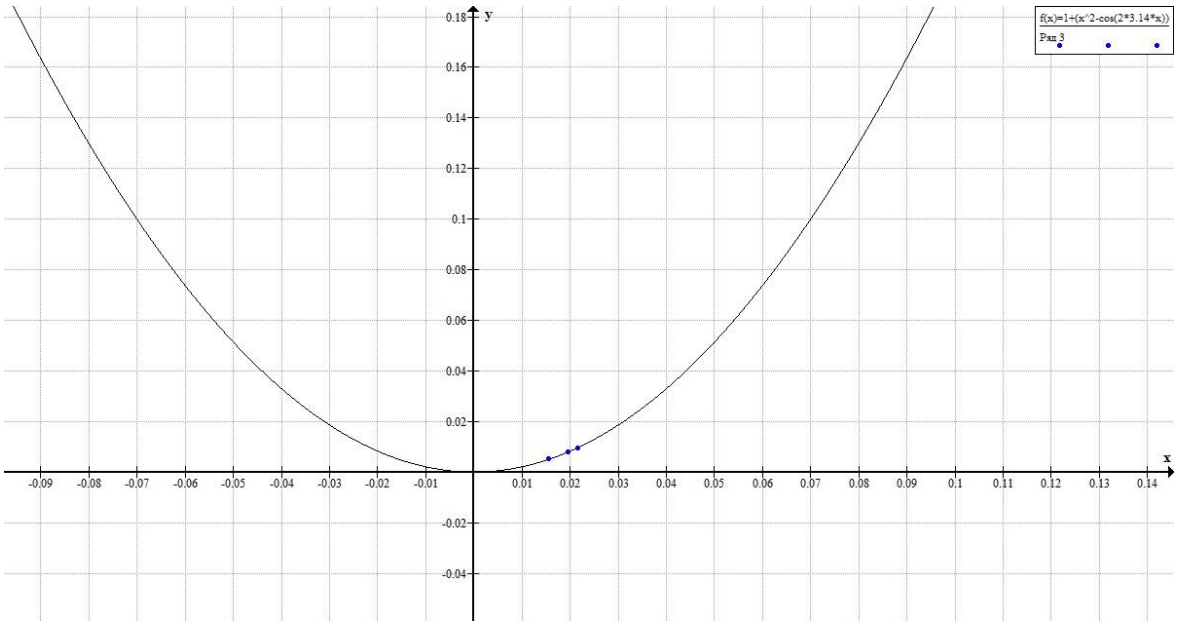
1 generation



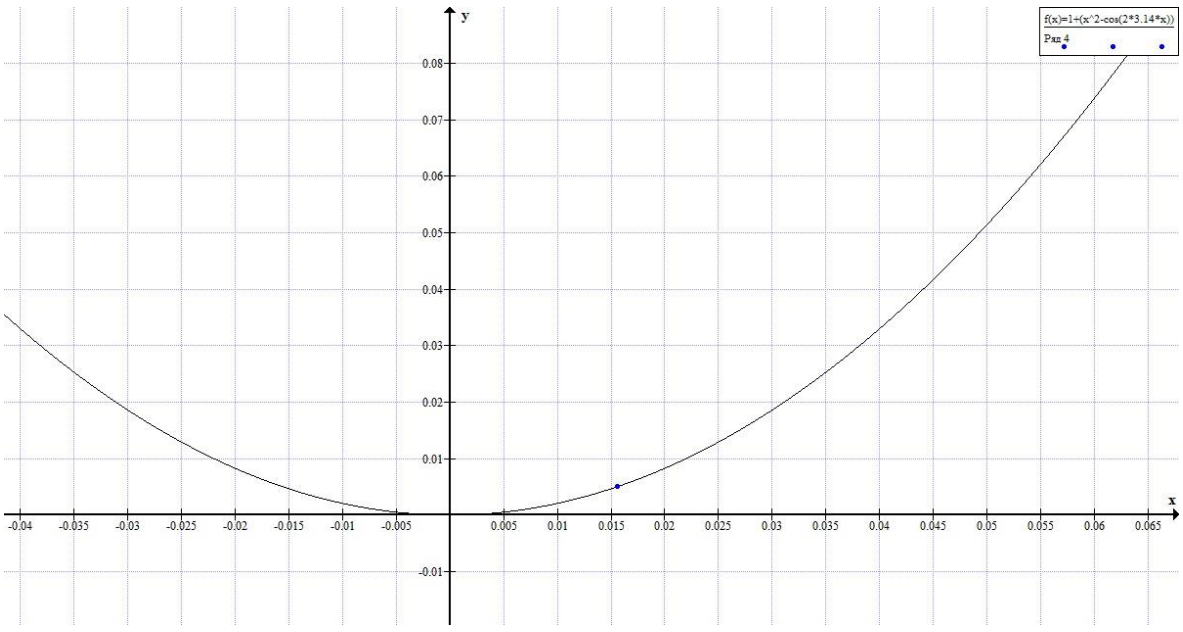
5 generation



10 generation



15 generation



20 generation