Contemporary Intelligent Intellectual Technology (CIIT)
Kirill Zabrodsky II-11
Task №1 (20-D version of Rastrigin's Function)

The task is to minimise $y = 20A + \sum_{i=1}^{20}\left(x_i^2 - Acos(2\pi x_i)\right)$ in the following way:

(1) Represent each of x_i($i = 1, \cdots, 20$) by a chromosome with 20 genes.

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

(3) Evolve this population till fitness dosen't change.

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--)
                {
```

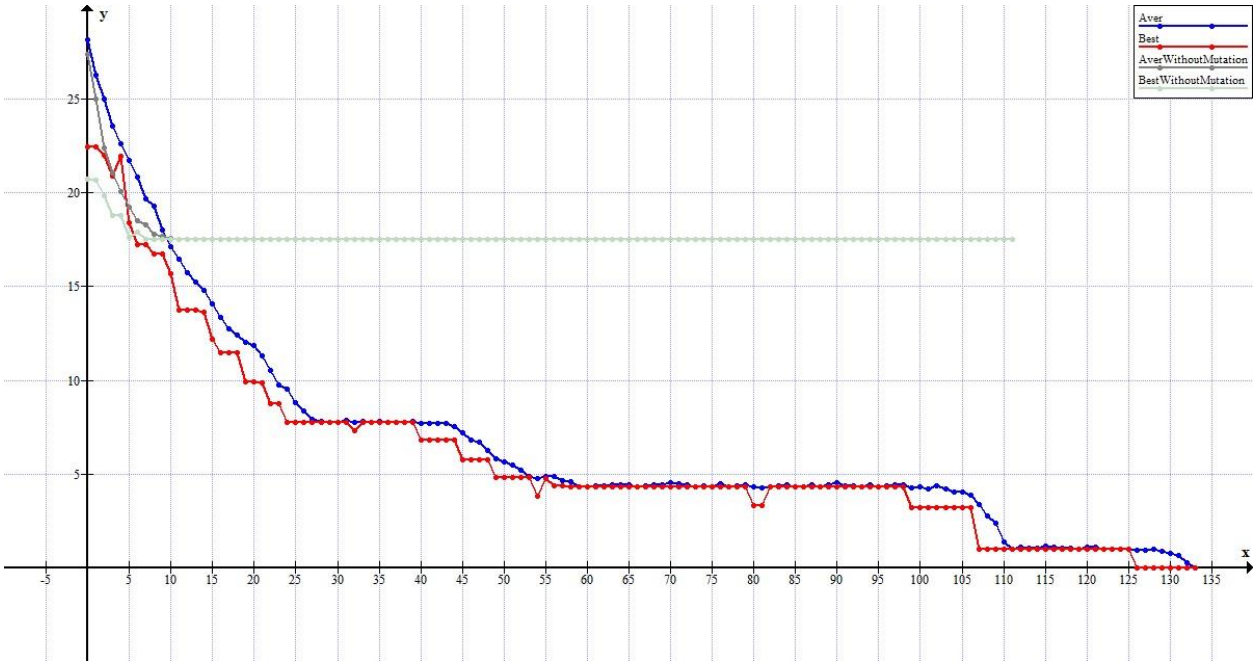
```
                    if (childrenPopulationDots[j].fitness >
childrenPopulationDots[j - 1].fitness)
                    {
                        Dot tempDot = childrenPopulationDots[j];
                        childrenPopulationDots[j] =
childrenPopulationDots[j - 1];
                        childrenPopulationDots[j - 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;
            }
        }
}
```

The graph of fitness vs generation:



Best fitness with mutation is 0;
Best fitness without mutation is

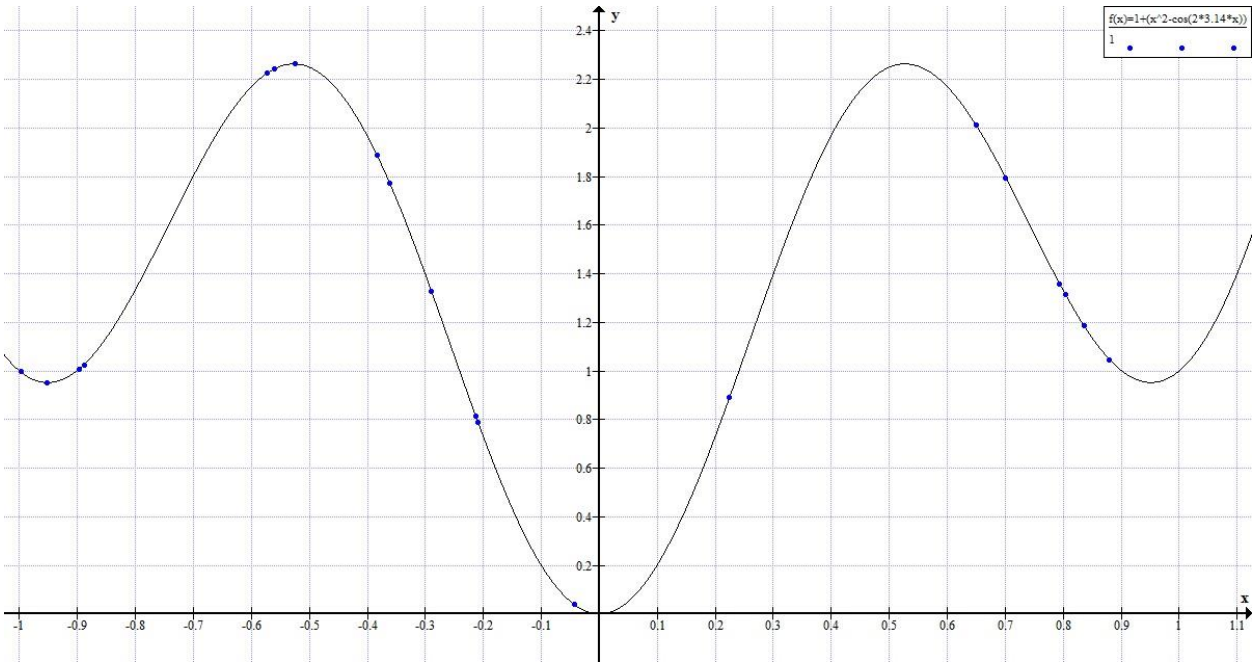
Task #2 (3-D Rastrigin's Function)

The task is to minimise $y = 20A + \sum_{i=1}^{20}\left(x_i^2 - Acos(2\pi x_i)\right)$ in the following way:

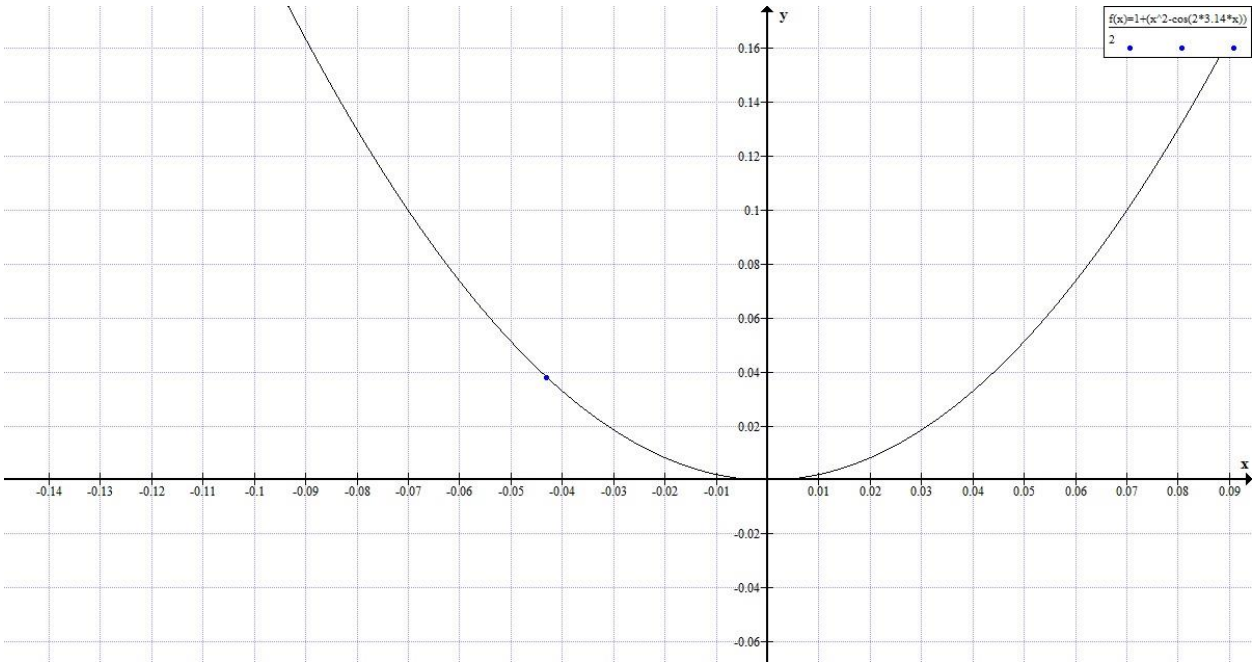
(1) Represent value of x by a 10-bit binary chromosome.

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

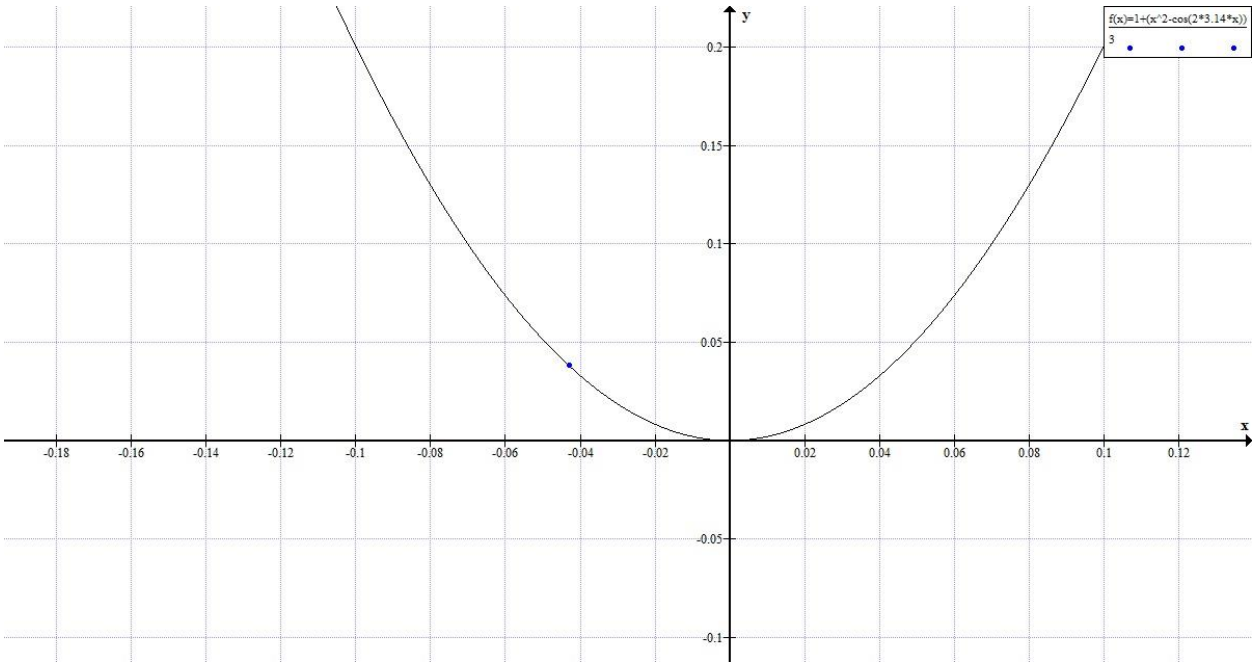
(3) Evolve this population till fitness doesn't change.



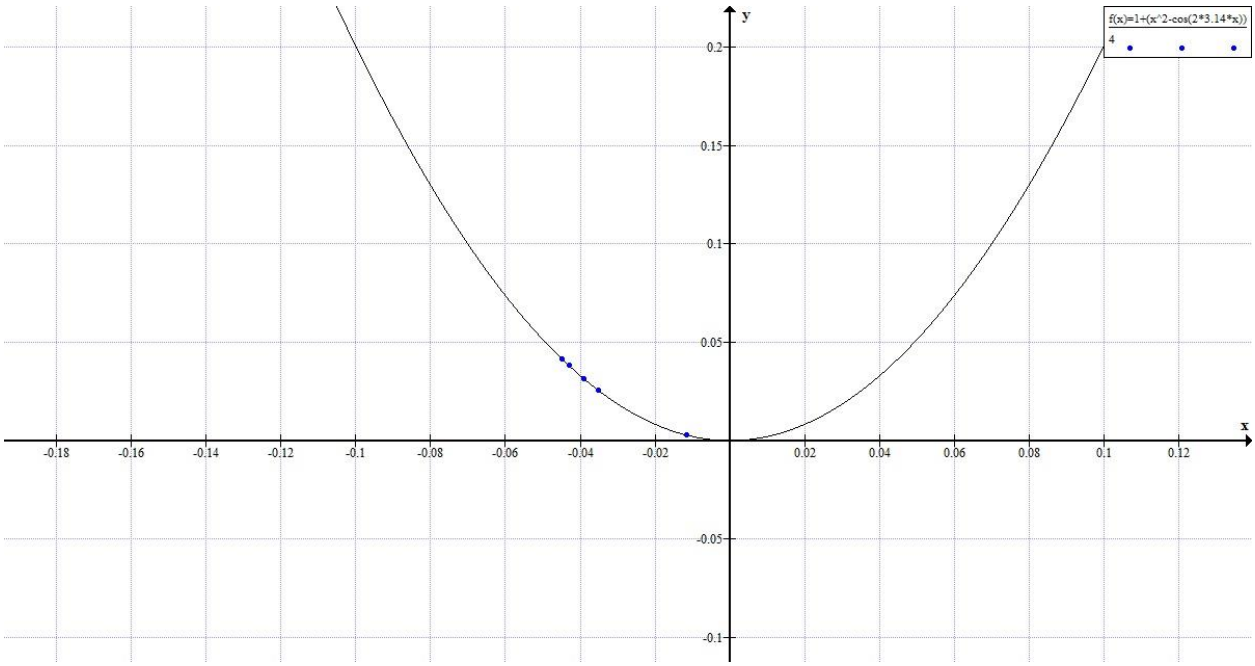
0 generation



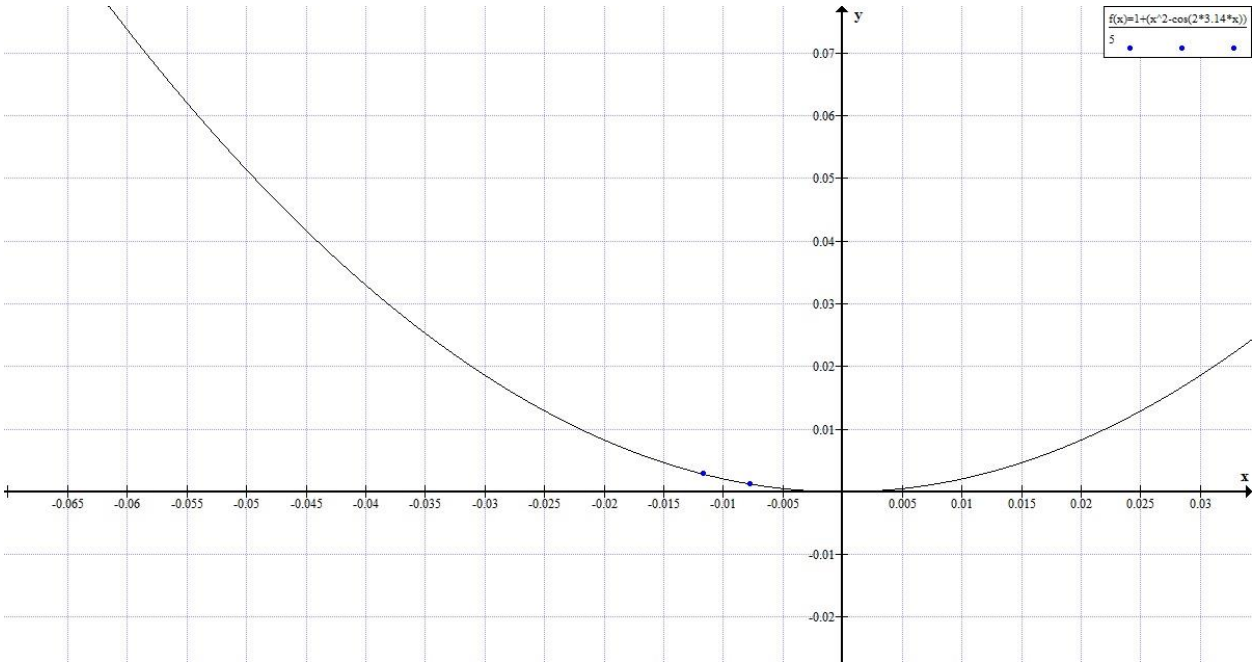
5th generation



10th generation



15th generation



20th generation