2-Ilya Babich

Minimization of Test Function

High-dimensional Test Function

Problem is minimize or maximize a n-dimensional function

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

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

(0:32;−0:51; 0:48; · · · ;−0:93)

For example, minimize

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

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

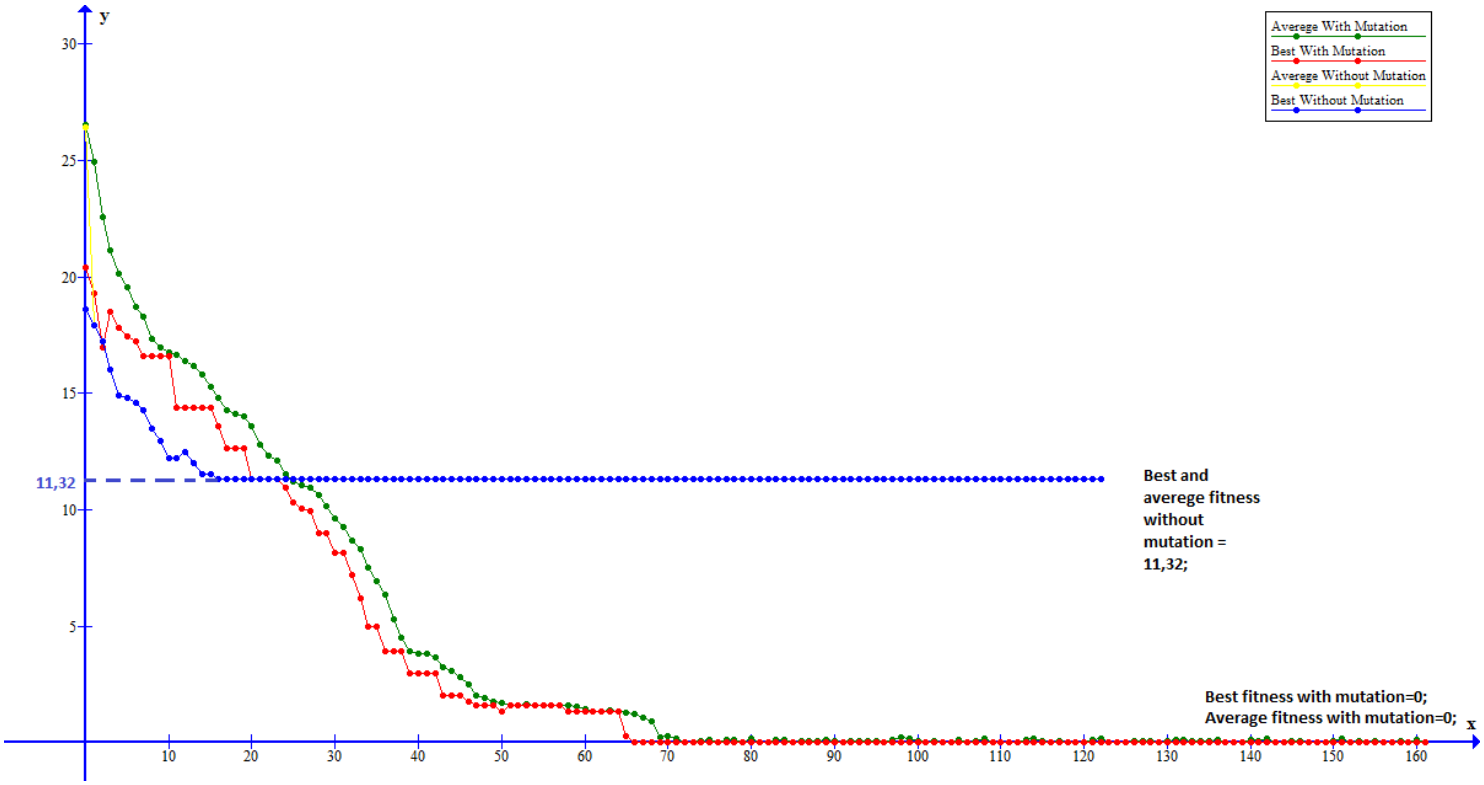
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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

```
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 theBestFit;
        public double avarageFit;
        public Population(List<Dot> dogs)
```

```
        {
            this.dots = dogs;
            theBestFit = getTheBestFit(dogs);
            avarageFit = getAvarageFit(dogs);
        }
        public double getTheBestFit(List<Dot> dogs)
        {
            double bestFit = 9999;
            for (int i = 0; i < dogs.Count; i++)
                if (bestFit > dogs[i].fit)
                    bestFit = dogs[i].fit;
            return bestFit;
        }
        public double getAvarageFit(List<Dot> dogs)
        {
            double avarageFit = 0;
            for (int i = 0; i < dogs.Count; i++)
                avarageFit += dogs[i].fit;
            avarageFit /= dogs.Count;
            return avarageFit;
        }
    }
}
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> histor;
        public GA()
        {
            random = new Random();
            List<Dot> dots = new List<Dot>();
            for (int i = 0; i < 20; i++)
            {
                List<double> chromo = new List<double>();
                for (int j = 0; j < 20; j++)
                {
                    chromo.Add(random.NextDouble() % 2 -1);
                }
                dots.Add(new Dot(chromo));
            }
            //Sorting by fit
            for (int i = 0; i < dots.Count; i++)
            {
                for (int j = dots.Count - 1; j > i; j--)
                {
                    if (dots[j].fit > dots[j - 1].fit)
                    {
                        Dot tempDot = dots[j];
                        dots[j] = dots[j - 1];
                        dots[j - 1] = tempDot;
                    }
                }
            }
            Population startPopulation = new Population(dots);
            histor = new List<Population>();
            histor.Add(startPopulation);
            int k = 0;
            while (!isReady(histor))
            {
                Thread.Sleep(10);
                histor.Add(getNextPupulation(histor[k]));
                k++;
            }
        }
        private Population getNextPupulation(Population parent)
```

```
{
    List<Dot> childrenPopulationDots = new List<Dot>();

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

Boolean isReady(List<Population> histor)
{
    if (histor.Count < 100)
        return false;
    else
    {
        for (int i = histor.Count - 100; i < histor.Count; i++)
        {
            if (histor[histor.Count - 100].avarageFit != histor[i].avarageFit)
                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> firstChromo = new List<double>();
    List<double> secondChromo = new List<double>();
    for (int j = 0; j < 20; j++)
    {
        if(j<pointCross)
        {
            firstChromo.Add(father.chromo[j]);
            secondChromo.Add(mother.chromo[j]);
        }
        else
        {
            firstChromo.Add(mother.chromo[j]);
            secondChromo.Add(father.chromo[j]);
        }
    }

    childrenDots.Add(new Dot(firstChromo));
    childrenDots.Add(new Dot(secondChromo));

    //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].chromo[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 +Σni=1(x2i− Acos(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.

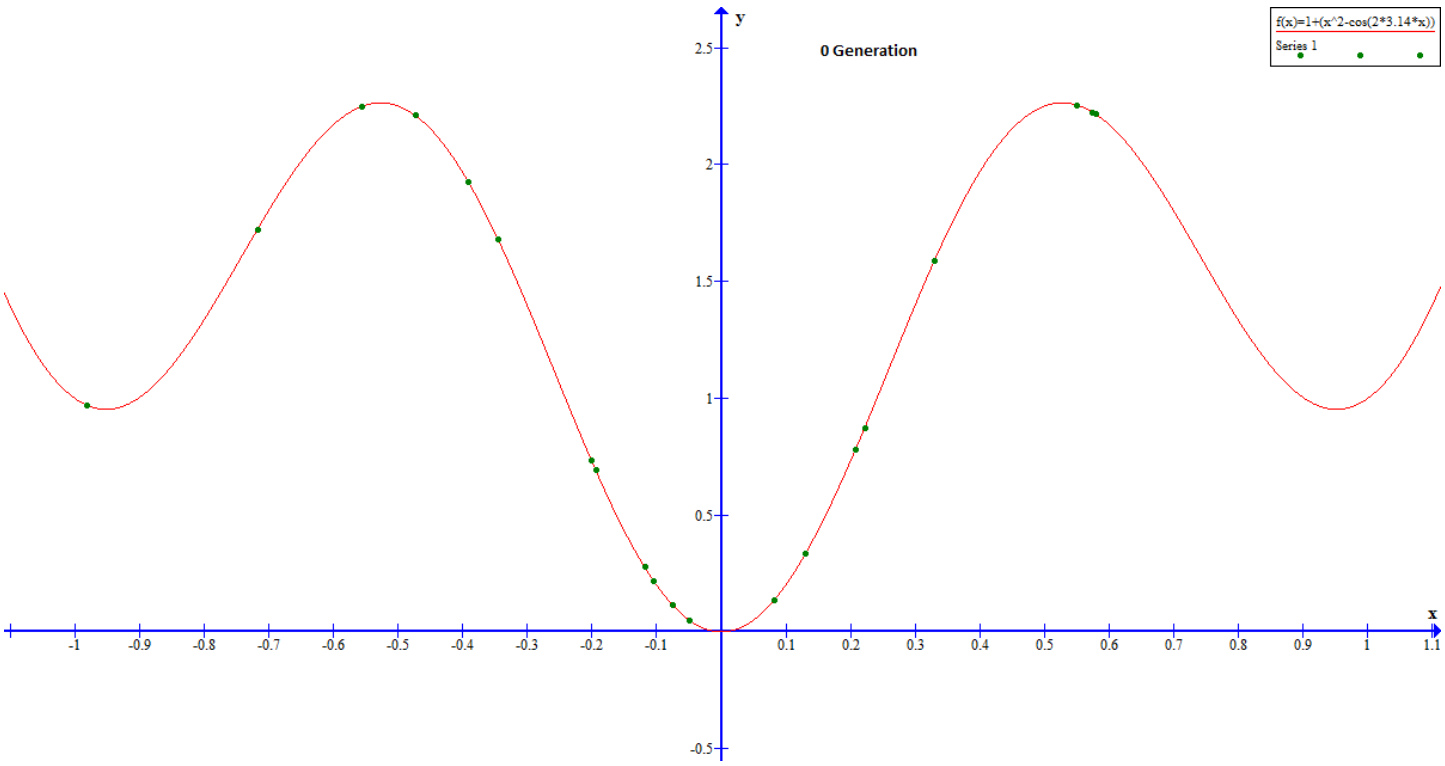
Source code

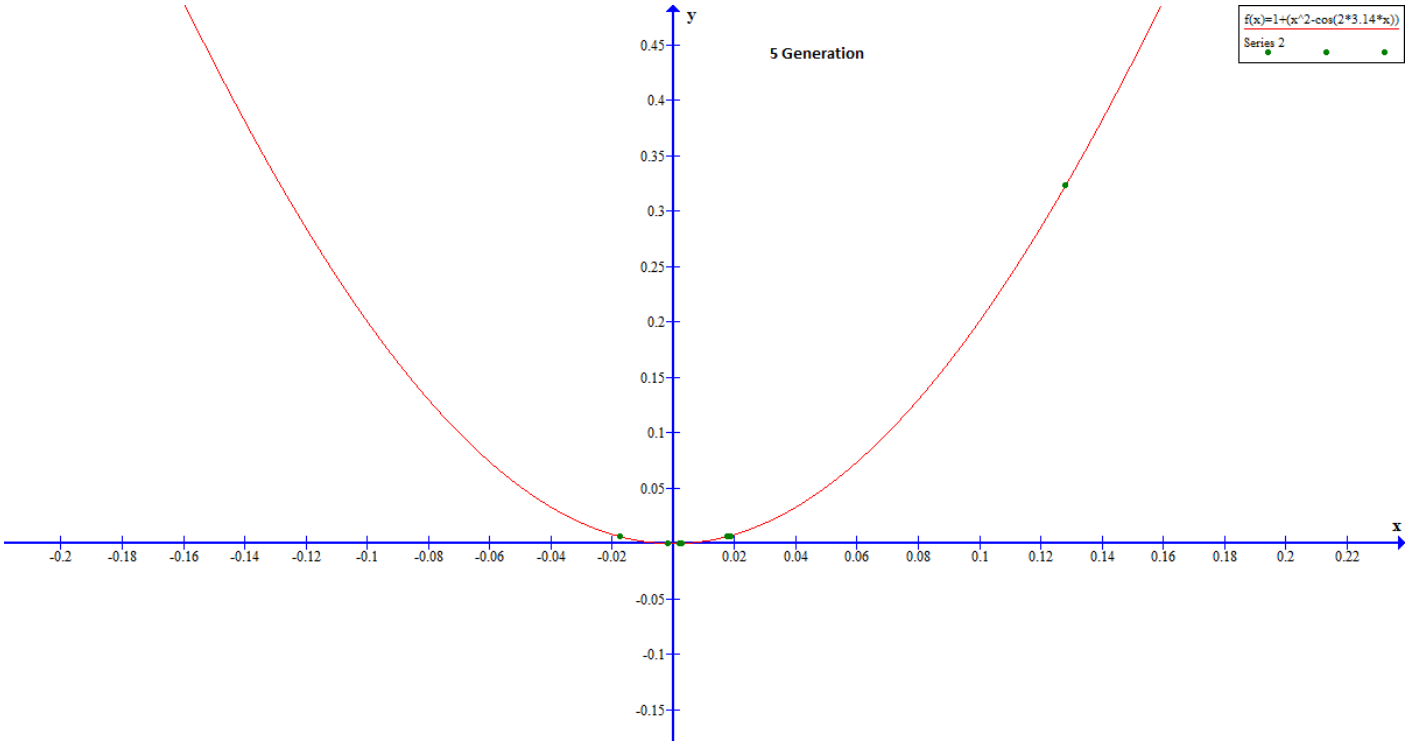
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab3_siit_
    class Dot
    {
        public List<double> chrom;
        public double fitness;
        public Dot(List<double> chrom)
        {
            this.chrom = chrom;
            fitness = getFitness(chrom);
        }
        private double getFitness(List<double> chrom)
        {
            double fitness = chrom.Count;
```

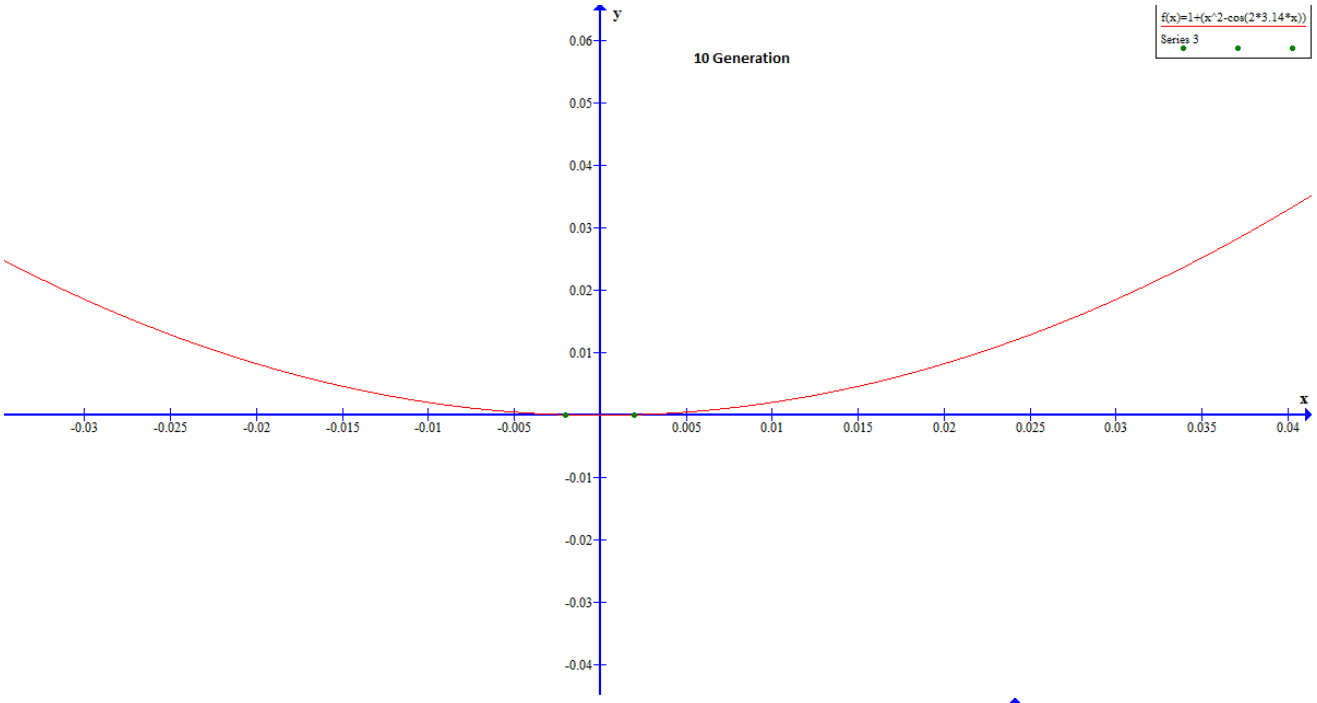
```
            double x=0;
            for (int i = 1; i < chrom.Count; i++)
                if (chrom[i] == 1)
                    x += Math.Pow(2, i - 1);
            x /= 1023;
            if (chrom[0] == 0)
                x *= -1;
            return 1 + (x * x - Math.Cos(Math.PI * 2 * x));
        }
    }
}


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> history;
        public GA()
        {
            random = new Random();
            List<Dot> dots = new List<Dot>();
            for (int i = 0; i < 20; i++)
            {
                List<double> chrom = new List<double>();
                for (int j = 0; j < 11; j++)
                {
                    chrom.Add(random.Next() % 2);
                }
                dots.Add(new Dot(chrom));
```

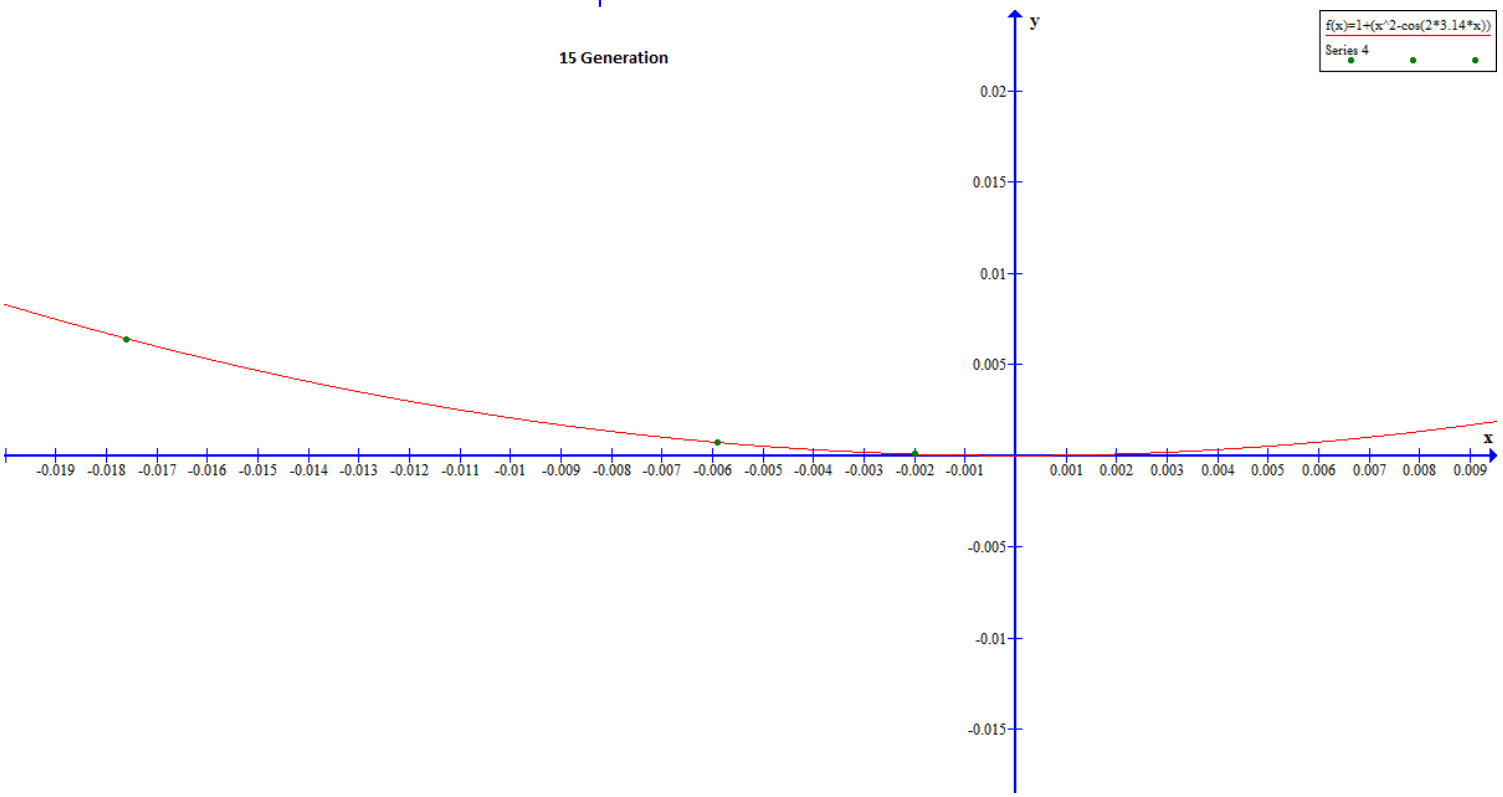
```
        }
        //Sorting by fitness
        for (int i = 0; i < dots.Count; i++)
        {
            for (int j = dots.Count - 1; j > i; j--)
            {
                if (dots[j].fitness > dots[j - 1].fitness)
                {
                    Dot tempDot = dots[j];
                    dots[j] = dots[j - 1];
                    dots[j - 1] = tempDot;
                }
            }
        }
        Population start = new Population(dots);
        history = new List<Population>();
        history.Add(start);
        int k = 0;
        while (!isReady(history))
        {
            Thread.Sleep(10);
            history.Add(getNextPupulation(history[k]));
            k++;
        }
    }
    private Population getNextPupulation(Population parentPopulation)
    {
        List<Dot> childrenDots = 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]
                );
            childrenDots.AddRange(childrenDots);
        }
        //Sorting by fitness
        for (int i = 0; i < childrenDots.Count; i++)
        {
            for (int j = childrenDots.Count - 1; j > i; j--)
            {
                if (childrenDots[j].fitness > childrenDots[j - 1].fitness)
                {
                    Dot tempDot = childrenDots[j];
                    childrenDots[j] = childrenDots[j - 1];
                    childrenDots[j - 1] = tempDot;
                }
            }
        }
        Population childrenPopulation = new Population(childrenDots);
        return childrenPopulation;
    }

    Boolean isReady(List<Population> history)
    {
        if (history.Count < 100)
            return false;
        else
        {
            for (int i = history.Count - 100; i < history.Count; i++)
            {
                if (history[history.Count - 100].avarageFitness != history[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> firstChrom = new List<double>();
        List<double> secondChrom = new List<double>();
```

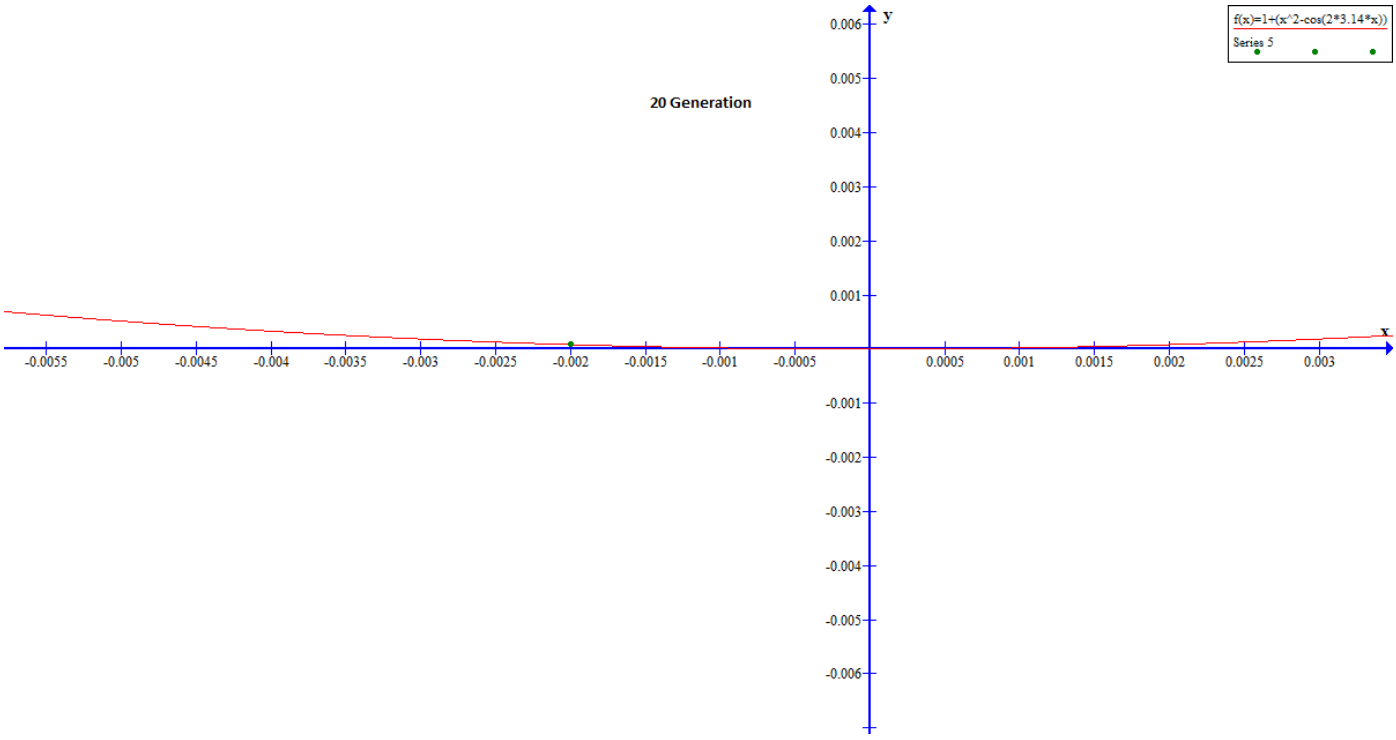
```
            for (int j = 0; j < 11; j++)
            {
                if(j<pointCross)
                {
                    firstChrom.Add(father.chrom[j]);
                    secondChrom.Add(mother.chrom[j]);
                }
                else
                {
                    firstChrom.Add(mother.chrom[j]);
                    secondChrom.Add(father.chrom[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].chrom[number] = random.NextDouble() % 2 - 1;
                }
            }

            childrenDots.Add(new Dot(firstChrom));
            childrenDots.Add(new Dot(secondChrom));
            return childrenDots;
        }
    }
}
```




5 Generation
f(x)=1+(x^2-cos(2*3.14*x))
Series 2
y
x
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
-0.05
-0.1
-0.15
-0.2
-0.18
-0.16
-0.14
-0.12
-0.1
-0.08
-0.06
-0.04
-0.02
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
0.22


10 Generation
f(x)=1+(x^2-cos(2*3.14*x))
Series 3
y
x
0.06
0.05
0.04
0.03
0.02
0.01
-0.01
-0.02
-0.03
-0.04
-0.03
-0.025
-0.02
-0.015
-0.01
-0.005
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04


15 Generation
f(x)=1+(x^2-cos(2*3.14*x))
Series 4
y
x
0.02
0.015
0.01
0.005
-0.005
-0.01
-0.015
-0.019
-0.018
-0.017
-0.016
-0.015
-0.014
-0.013
-0.012
-0.011
-0.01
-0.009
-0.008
-0.007
-0.006
-0.005
-0.004
-0.003
-0.002
-0.001
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009


20 Generation
f(x)=1+(x^2-cos(2*3.14*x))
Series 5
y
x
0.006
0.005
0.004
0.003
0.002
0.001
-0.001
-0.002
-0.003
-0.004
-0.005
-0.006
-0.0055
-0.005
-0.0045
-0.004
-0.0035
-0.003
-0.0025
-0.002
-0.0015
-0.001
-0.0005
0.0005
0.001
0.0015
0.002
0.0025
0.003