TASK 1

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

$$y = nA + \sum_{i=1}^{n} (x_i^2 - A\cos(2\pi x_i))$$

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

   (1) the graph of fitness vs generation.

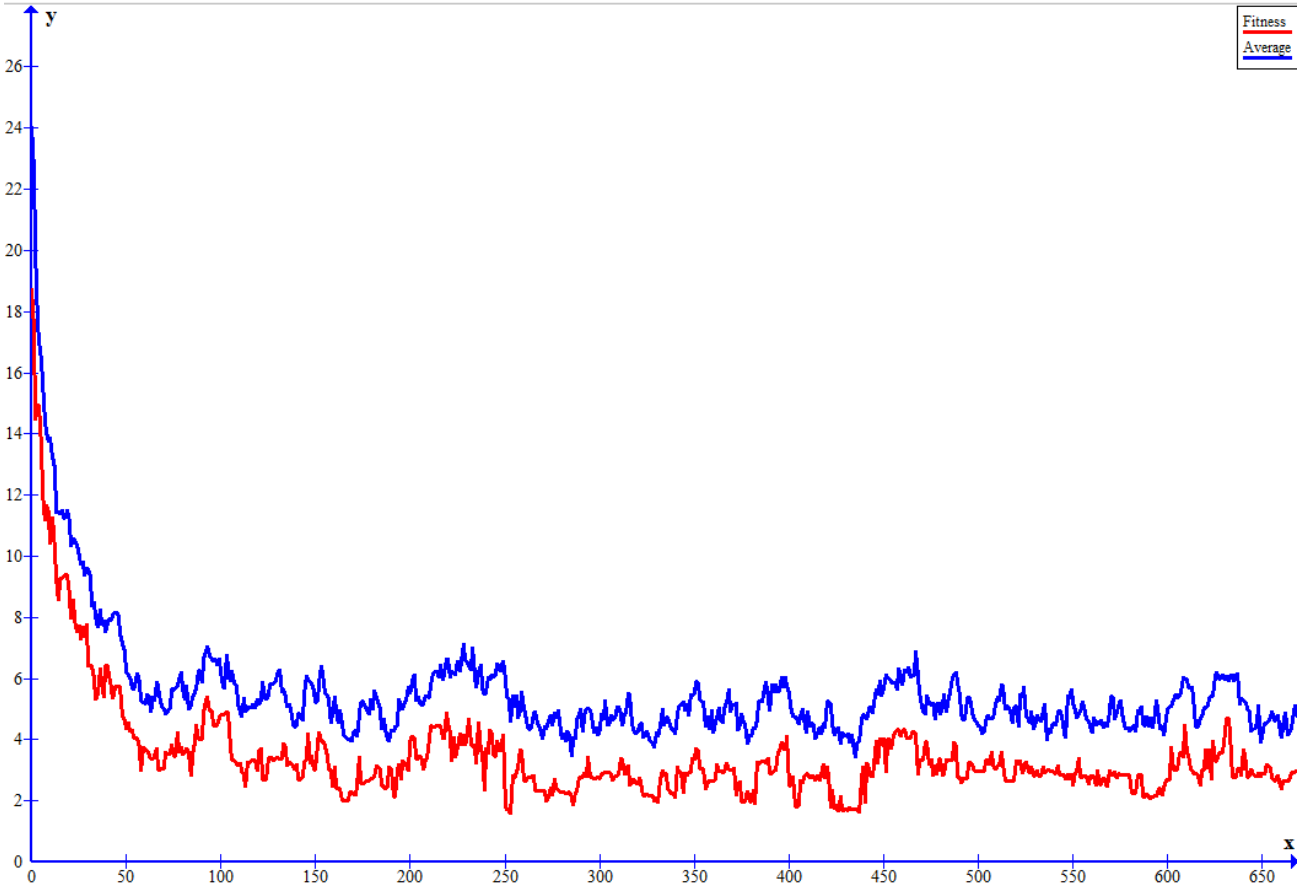
   (2) Create a population of 20 chromosomes at random, with fitness being y.

```
using System;
    using System.Linq;
    namespace SIIT_3._1
    {
        public class Population
        {
            Random rand = new Random();
            const int size = 20;
            const double A = 1;
            double[,] parents = new double[size, size];
            double[,] p = new double[size, size];
            double[,] newPopulation = new double[size / 2, size];
            double[] y = new double[size];
            double m;
            double average;
            int[] keys = new int[size];
            public Population()
            {
                double[] genes = new double[size];
                for (int i = 0; i < size; i++)
                {
                    for (int j = 0; j < size; j++)
                    {
                        genes[j] = ((double)rand.Next(-1000, 1001) / 1000);
                        parents[i, j] = genes[j];
                    }
                    y[i] = Function(i);
                }
            }
            public void Cross()
            {
                double[,] reserv = new double[size, size];
                for (int i = 0; i < size; i++)
                    keys[i] = i;
                Array.Sort(y, keys);
                for (int i = 0; i < size / 2; i++)
                {
                    for (int j = 0; j < size; j++)
                    {
                        newPopulation[i, j] = parents[keys[i], j];
                    }
```

```
                }
                int k = 0;
                for (int i = 0; i < size / 2; i++)
                {
                    int a = rand.Next(0, size / 2);
                    int b = rand.Next(0, size / 2);
                    for (int j = 0; j < size; j++)
                    {
                        p[0, j] = newPopulation[a, j];
                        p[1, j] = newPopulation[b, j];
                        reserv[0, j] = p[0, j];
                    }
                    int c = rand.Next(0, size);
                    for (int j = size - c; j < size; j++)
                    {
                        p[0, j] = p[1, j];
                        p[1, j] = reserv[0, j];
                    }
                    for (int j = 0; j < size; j++)
                    {
                        parents[k, j] = p[0, j];
                        parents[k + 1, j] = p[1, j];
                    }
                    k += 2;
                }
            }
            public void Mutation()
            {
                for (int i = 0; i < size; i++)
                {
                    for (int j = 0; j < size; j++)
                    {
                        int value = rand.Next(0, size);
                        if (value == 0)
                        {
                            parents[i, j] = ((double)rand.Next(-1000, 1001) / 1000);
                        }
                    }
                    y[i] = Function(i);
                }
            }
            public double GetMax()
            {
                m = y.Min();
                return m;
            }
            public double GetAverage()
            {
                average = y.Average();
                return average;
            }
            public double Function(int i)
            {
                double result = 0;
                double sum = 0;
                for (int j = 0; j < size; j++)
                {
                    sum += (Math.Pow(parents[i, j], 2) - (A * Math.Cos(2 * Math.PI *
parents[i, j])));
                }
                result = size * A + sum;
```

```
                return result;
            }
        }
    }
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SIIT_3._1
{
    public class Conditional
    {
        public bool Condit(double max)
        {
            if (max == 0)
            {
                return false;
            }
            else
                return true;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Conditional c = new Conditional();
            Population pop = new Population();
            int i = -1;
            double max = -1;
            double average = 0;
            while (i < 1000)
            {
                i++;
                pop.Cross();
                pop.Mutation();
                max = pop.GetMax();
                average = pop.GetAverage();
                using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"Iteration.txt", true))
                {
                    file.WriteLine(i);
                }
                using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"Minimum.txt", true))
                {
                    file.WriteLine(max);
                }
                using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"Average.txt", true))
                {
                    file.WriteLine(average);
                }
            }
        }
    }
}
```


y
x
Fitness
Average
0
2
4
6
8
10
12
14
16
18
20
22
24
26
0
50
100
150
200
250
300
350
400
450
500
550
600
650

Task 2

```
using System;
using System.Linq;
namespace SIIT_3._1
{
    public class Population
    {
        Random rand = new Random();
        const int size = 20;
        const int size1 = 11;
        const double A = 1;
        double[,] parents = new double[size, size1];
        double[,] p = new double[size, size1];
        double[,] newPopulation = new double[size / 2, size1];
        double[] y = new double[size];
        double[] x = new double[size];
        double m;
        double average;
        double value = 0;
        int[] keys = new int[size];
        public Population()
        {
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size1; j++)
                {
                    parents[i, j] = rand.Next(0, 2);
                }
                y[i] = Function(i);
            }
        }
        public void Cross()
        {
            double[,] reserv = new double[size, size1];
            for (int i = 0; i < size; i++)
                keys[i] = i;
            Array.Sort(y, keys);
            for (int i = 0; i < size / 2; i++)
            {
                for (int j = 0; j < size1; j++)
                {
                    newPopulation[i, j] = parents[keys[i], j];
                }
            }
            int k = 0;
            for (int i = 0; i < size / 2; i++)
            {
                int a = rand.Next(0, size / 2);
                int b = rand.Next(0, size / 2);
                for (int j = 0; j < size1; j++)
                {
                    p[0, j] = newPopulation[a, j];
                    p[1, j] = newPopulation[b, j];
                    reserv[0, j] = p[0, j];
                }
                int c = rand.Next(0, size1);
                for (int j = size1 - c; j < size1; j++)
                {
                    p[0, j] = p[1, j];
```

```
            p[1, j] = reserv[0, j];
        }
        for (int j = 0; j < size1; j++)
        {
            parents[k, j] = p[0, j];
            parents[k + 1, j] = p[1, j];
        }
        k += 2;
    }
}
public void Mutation()
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size1; j++)
        {
            int value = rand.Next(0, size1);
            if (value == 0)
            {
                parents[i, j] = rand.Next(0, 2);
            }
        }
        y[i] = Function(i);
    }
}
public double GetMax()
{
    m = y.Min();
    return m;
}
public double GetAverage()
{
    average = y.Average();
    return average;
}
public double[] GetY()
{
    return y;
}
public double[] GetX()
{
    return x;
}
public double Function(int i)
{
    string str = "";
    value = 0;
    for (int j = 1; j < size1; j++)
    {
        str += Convert.ToString(parents[i, j]);
    }
    value = BinToDec(str) / 1023;
    if (parents[i, 0] == 0)
    {
        value *= -1;
    }
    double result = 0;
    double sum = 0;
    x[i] = value;
    for (int j = 0; j < size; j++)
    {
```

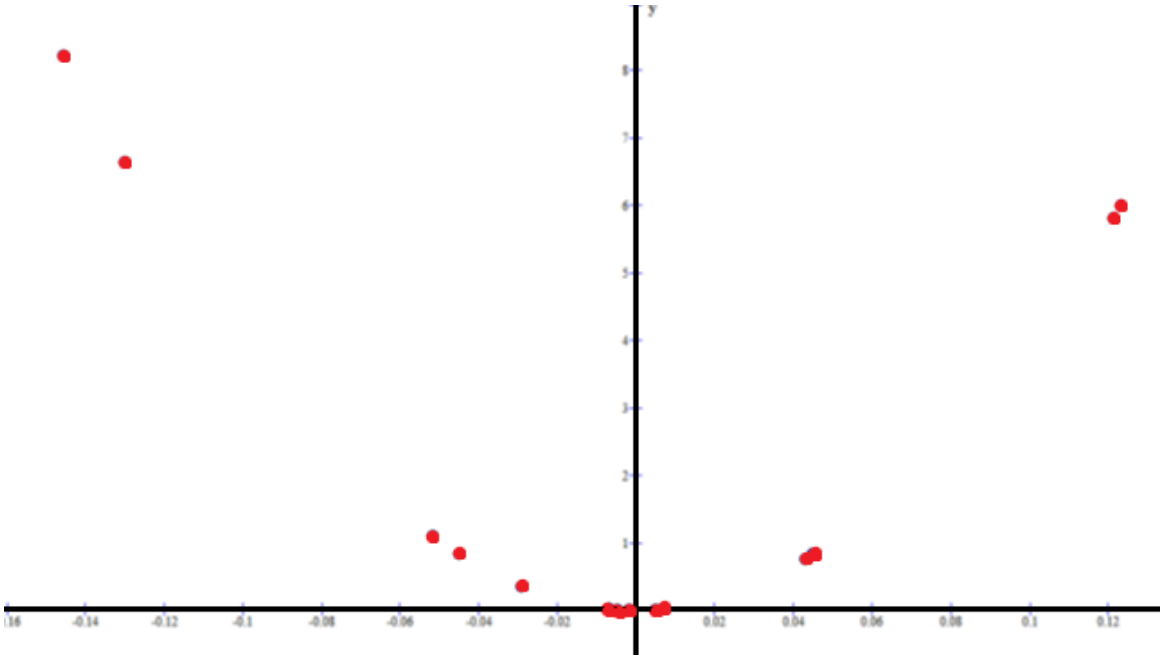
```
                sum += (Math.Pow(value, 2) - (A * Math.Cos(2 * Math.PI * value)));
            }
            result = size * A + sum;
            return result;
        }
        public double BinToDec(string str)
        {
            double res = 0;
            for (int i = 0; i < 8; i++)
            {
                res += double.Parse(str[i].ToString()) * Math.Pow(2, 7 - i);
            }
            return res;
        }
    }
}
```

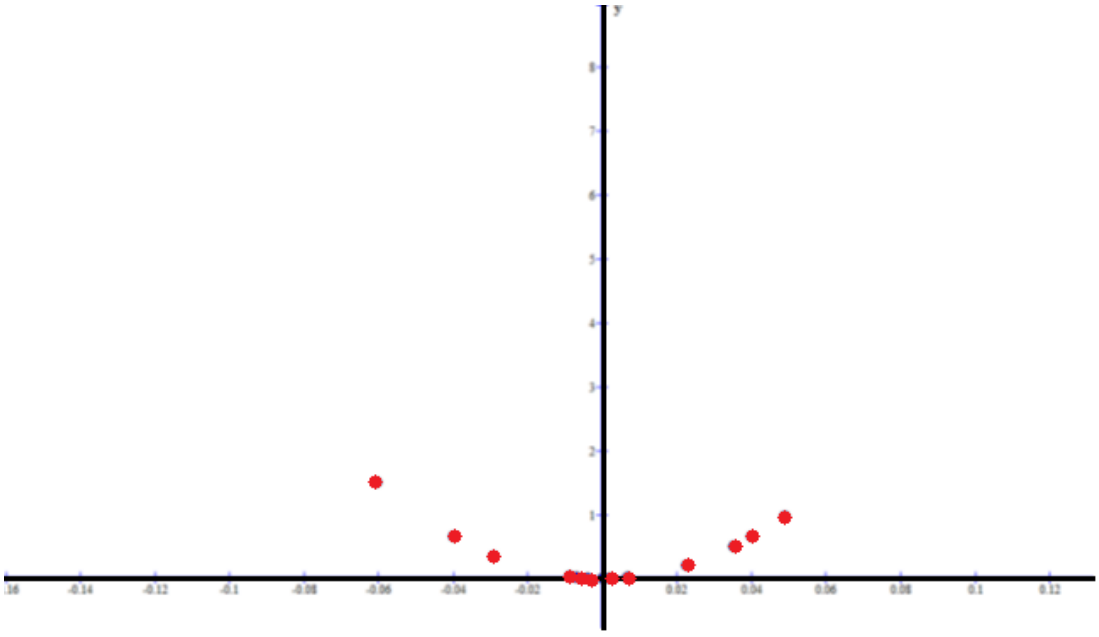
```
using System;
namespace SIIT_3._1
{
    public class Conditional
    {
        public bool Condit(double max)
        {
            if (max == 0)
            {
                return false;
            }
            else
                return true;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Conditional c = new Conditional();
            Population pop = new Population();
            int i = -1;
            double max = -1;
            double average = 0;
            double[] y = new double[20];
            double[] x = new double[20];
            while (i < 10000)
            {
                i++;
                pop.Cross();
                pop.Mutation();
                y = pop.GetY();
                x = pop.GetX();
                max = pop.GetMax();
                average = pop.GetAverage();
                if (i == 9999)
                {
                    for (int j = 0; j < 20; j++)
                    {
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"X.txt", true))
                        {
                            file.WriteLine(x[j]);
                        }
```

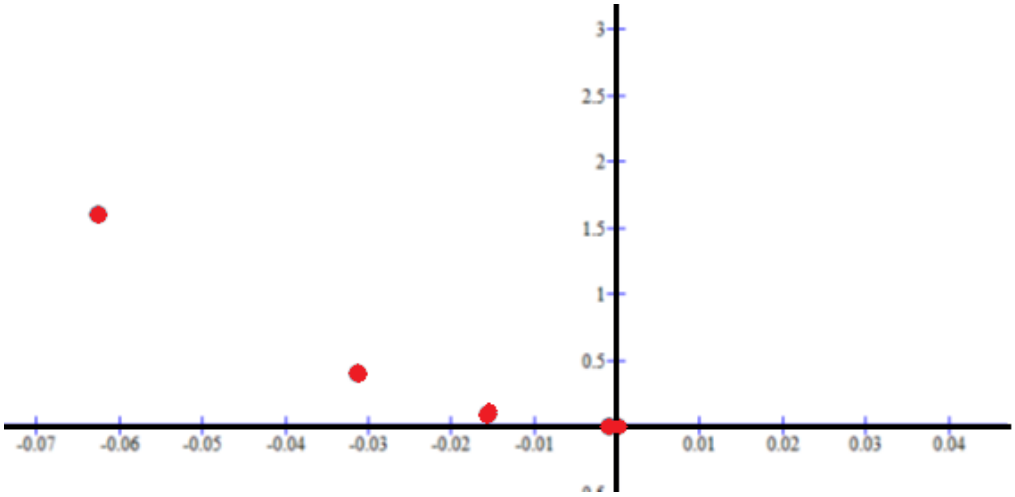
```
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"Y.txt", true))
                        {
                            file.WriteLine(y[j]);
                        }
                    }
                }
            }
        }
    }
}
```

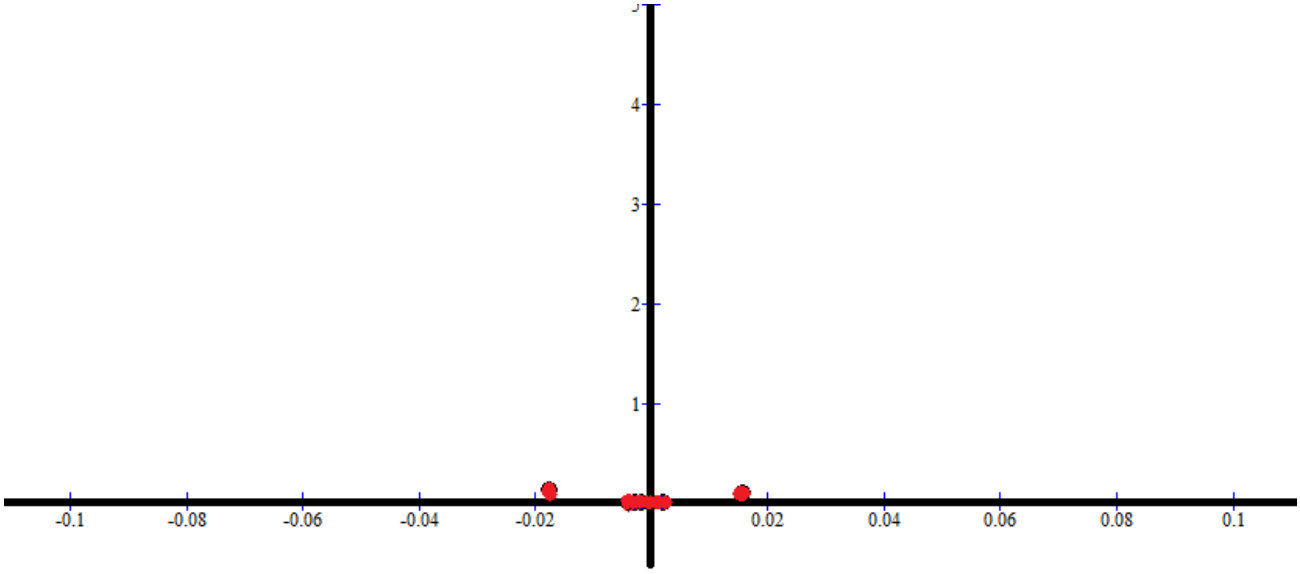
Results:

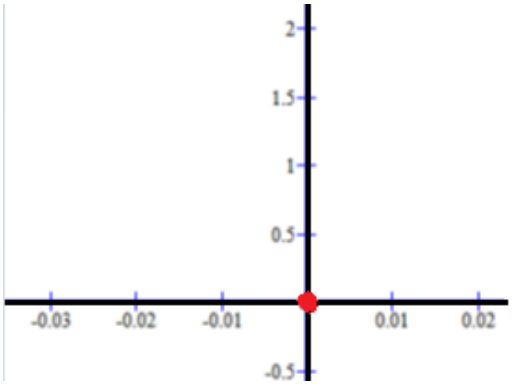
Red point - population






3
2.5
2
1.5
1
0.5
-0.07
-0.06
-0.05
-0.04
-0.03
-0.02
-0.01
0.01
0.02
0.03
0.04


4
3
2
1
-0.1
-0.08
-0.06
-0.04
-0.02
0.02
0.04
0.06
0.08
0.1


2
1.5
1
0.5
-0.5
-0.03
-0.02
-0.01
0.01
0.02