(3) 3-D version of Schwefel's function

Exercise 7 *1. Minimize*

$$y = x_1 \sin(|x_1|) + x_2 \sin(|x_2|)$$

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 dosen't change.

2. Show

(1) the graph of fitness vs generation.

(2) all 20 points (x, y) in the 1st, an intermediate, and final generation.

Source Code

Chromosome.cs

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

namespace Graph
{
    public class Chromosome : IComparable<Chromosome>
    {
        public List<int> Genes { get; set; }
        public double x1, x2;
        public double Fitness { get; set; }
        Random rnd;
        const int GENES_NUMBER = 22;

        public Chromosome(Random _rnd)
        {
            rnd = _rnd;
            Genes = new List<int>();

            for (int i = 0; i < GENES_NUMBER; i++)
                Genes.Add(rnd.Next(2));

            CalculateFitness();
        }

        public void CalculateFitness()
        {
            int tempX1, tempX2;

            var x1Genes = Genes.Skip(1).Take(10);
            var x2Genes = Genes.Skip(12);
```

```
        string x1String = String.Empty;
        string x2String = String.Empty;

        foreach (int x in x1Genes)
            x1String += x.ToString();

        foreach (int x in x2Genes)
            x2String += x.ToString();

        tempX1 = Convert.ToInt32(x1String, 2);
        tempX2 = Convert.ToInt32(x2String, 2);

        if (Genes.ElementAt(0) == 0)
            tempX1 = -tempX1;

        if (Genes.ElementAt(11) == 0)
            tempX2 = -tempX2;

        x1 = ((double)tempX1 * 5 / 1023);
        x2 = ((double)tempX2 * 5 / 1023);
       // Console.WriteLine($"x1 = {x1}, x2 = {x2}");
            Fitness = x1 * Math.Sin(Math.Abs(x1)) + x2 * Math.Sin(Math.Abs(x2));
    }

    public int CompareTo(Chromosome compareChromosome)
    {
        if (compareChromosome == null)
            return -1;
        else
            return compareChromosome.Fitness.CompareTo(Fitness);
    }

    public Chromosome CreateChild(Chromosome parent2)
    {
        //uniform crossover
        int chooseParent;
        Chromosome child = new Chromosome(rnd);
        child.Genes.Clear();
        child.Fitness = 0.0;
        int i;
        for (i = 0; i < GENES_NUMBER; i++)
        {
            chooseParent = rnd.Next(2);

            if (chooseParent == 1)
                child.Genes.Add(Genes.ElementAt(i));
            else
                child.Genes.Add(parent2.Genes.ElementAt(i));
        }

        int mutation;
        for (i = 0; i < child.Genes.Count; i++)
        {
            mutation = rnd.Next(22);

            if (mutation == 5)
            {
                if (child.Genes.ElementAt(i) == 1)
                {
                    child.Genes.RemoveAt(i);
                    child.Genes.Insert(i, 0);
                }
                else
                {
```

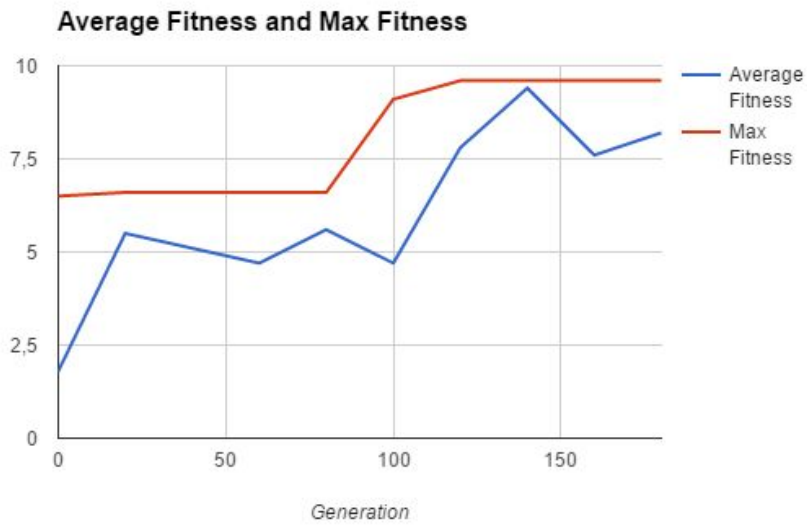
```
                        child.Genes.RemoveAt(i);
                        child.Genes.Insert(i, 1);
                    }
                }
            }

            child.CalculateFitness();

            return child;
        }
    }
}
```

Program.cs

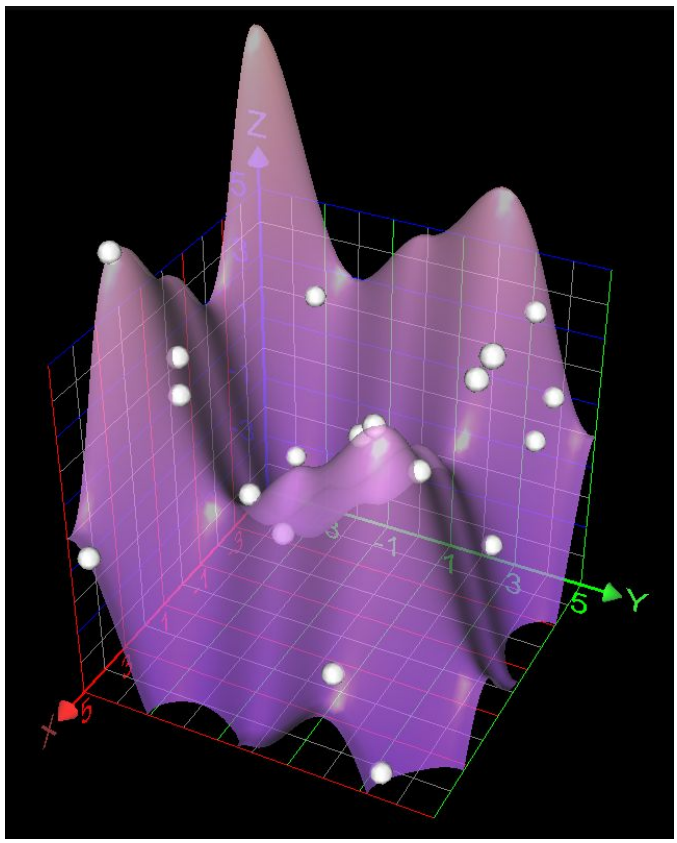
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Graph
{
    class Program
    {
        static void Main(string[] args)
        {
            const int CHROMOSOMES_NUMBER = 20;
            Random rnd = new Random();
            int i;
            var chromosomes = new List<Chromosome>();
            var childChromosomes = new List<Chromosome>();
            childChromosomes.Clear();

            double maxFitness, averageFitness;
            int fitnessChange = 0;

            for (i = 0; i < CHROMOSOMES_NUMBER; i++)
                chromosomes.Add(new Chromosome(rnd));

            maxFitness = chromosomes.Max(x => x.Fitness);

            while(fitnessChange != 500)
            {
                chromosomes.Sort();

                for (i = 0; i < CHROMOSOMES_NUMBER; i++)
                {

childChromosomes.Add(chromosomes.ElementAt(rnd.Next(CHROMOSOMES_NUMBER / 2))
                        .CreateChild(chromosomes.ElementAt(rnd.Next(CHROMOSOMES_NUMBER
/ 2))));
                }

                chromosomes.Clear();
                chromosomes.AddRange(childChromosomes);
                childChromosomes.Clear();

                fitnessChange++;

                if (fitnessChange % 10 == 0)
                {
```

```
                    Console.WriteLine("Max Fitness = " + chromosomes.Max(x =>
x.Fitness) + " Average Fitness = " + chromosomes.Average(x => x.Fitness));

                }
            }
        }
    }
}
```

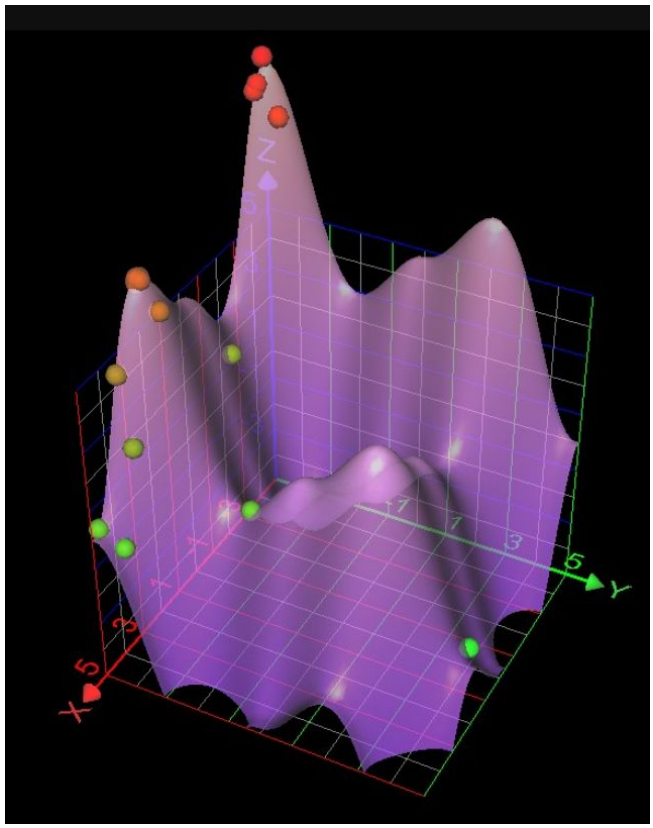
Results



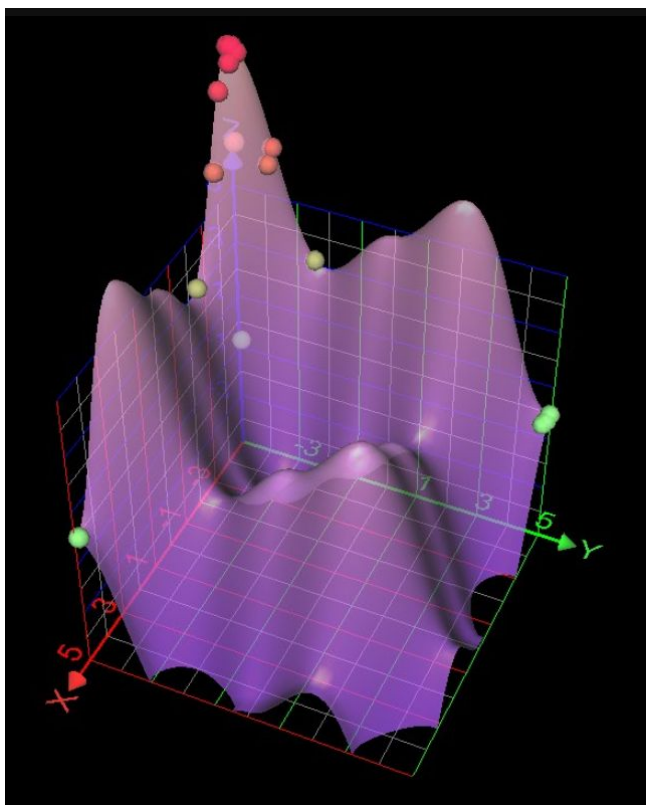
1-st generation



100 generation



199 generation



Console(Program Output)

Max Fitness = 6,55364726508597 Average Fitness = 1,80084368359127

Generation: 0

x1 = -2,26295210166178, x2 = 3,31378299120235, y = -2,30996935830601

x1 = 4,41348973607038, x2 = 3,30400782013685, y = -4,75206578711701

x1 = -3,81720430107527, x2 = 3,75855327468231, y = 0,212639004736834

x1 = 1,38318670576735, x2 = -4,76050830889541, y = 6,1139137193534

x1 = 4,3108504398827, x2 = 1,94525904203324, y = -2,15750779098347

x1 = -2,88856304985337, x2 = -2,86901270772239, y = -1,49550485024844

x1 = 2,32160312805474, x2 = -4,80449657869013, y = 6,48154462116646

x1 = 1,68132942326491, x2 = 1,70576735092864, y = 3,36132279031548

x1 = 1,66666666666667, x2 = 1,33431085043988, y = 2,95618675412439

x1 = 1,68132942326491, x2 = -4,21309872922776, y = 5,3698436432657

x1 = 1,96480938416422, x2 = -1,56402737047898, y = 0,250266090542297

x1 = -4,40860215053763, x2 = 3,96871945259042, y = 1,28579976560189

x1 = -4,13000977517107, x2 = -2,86901270772239, y = 2,67681712781273

x1 = 4,9169110459433, x2 = -4,47702834799609, y = -0,460835541972179

x1 = 1,4613880742913, x2 = -3,75366568914956, y = 3,60937664784761

x1 = 1,95014662756598, x2 = -3,455522971652, y = 2,87856466729243

x1 = 1,70087976539589, x2 = 3,79765395894428, y = -0,630061907129475

x1 = -4,41348973607038, x2 = 3,3088954056696, y = 3,66679250780999

x1 = 2,24340175953079, x2 = -4,99022482893451, y = 6,55364726508597

x1 = 1,63734115347019, x2 = 2,86901270772239, y = 2,4061043026269

Max Fitness = 6,63389663515081 Average Fitness = 5,50931240021012

Max Fitness = 6,63395468243388 Average Fitness = 5,16021347046184

Max Fitness = 6,63089195159296 Average Fitness = 4,75083165118428

Max Fitness = 6,63100966133728 Average Fitness = 5,66986749434948

Generation: 99

x1 = 2,01857282502444, x2 = -4,95112414467253, y = 6,63026544408641

x1 = 1,97947214076246, x2 = -1,81818181818182, y = 0,0536293585529051

x1 = 2,01857282502444, x2 = -4,32551319648094, y = 5,82539042772362

x1 = 1,97947214076246, x2 = -4,96089931573803, y = 6,62495832715747

x1 = 2,01857282502444, x2 = -4,32551319648094, y = 5,82539042772362

x1 = -4,48191593352884, x2 = -4,30596285434995, y = 8,31860424286707

x1 = 3,6217008797654, x2 = -4,37438905180841, y = 2,45411144014244

x1 = -4,78983382209189, x2 = -4,94623655913979, y = 9,58708703273621

x1 = 2,01368523949169, x2 = -4,96089931573803, y = 6,62789959704446

x1 = -1,35386119257087, x2 = -4,22287390029325, y = 2,40481468833051

x1 = 0,762463343108504, x2 = 4,32062561094819, y = -3,46664581183725

x1 = 4,63831867057674, x2 = -5, y = 0,169020747254683

x1 = 3,26490713587488, x2 = -4,95112414467253, y = 4,40910890276325

x1 = 1,90127077223851, x2 = -4,95112414467253, y = 6,60908978356089

x1 = -4,48191593352884, x2 = -4,30596285434995, y = 8,31860424286707

x1 = -4,48191593352884, x2 = -4,32062561094819, y = 8,35669005182577

x1 = -4,48191593352884, x2 = -4,95601173020528, y = 9,17307010444862

x1 = 4,48680351906158, x2 = -4,32551319648094, y = -0,367298198133858

x1 = -4,40371456500489, x2 = -5, y = 8,99020367928081

x1 = 2,01857282502444, x2 = -4,86803519061584, y = 6,62875406379597

Max Fitness = 9,17546435492153 Average Fitness = 4,78903944577828

Max Fitness = 9,62457874664465 Average Fitness = 7,86021149985483

Max Fitness = 9,62544743405881 Average Fitness = 9,43082777842302

Max Fitness = 9,62842721322818 Average Fitness = 7,67265038424611

Max Fitness = 9,6286251914388 Average Fitness = 8,24191346393028

Generation: 199

x1 = -4,64320625610948, x2 = 4,86314760508309, y = -0,175888077382415

x1 = -4,60410557184751, x2 = -4,79472140762463, y = 9,35561945189926

x1 = -4,9169110459433, x2 = -2,42913000977517, y = 3,22651284293542

x1 = -4,88269794721408, x2 = -4,93157380254154, y = 9,62564326437134

x1 = -4,86314760508309, x2 = -4,82893450635386, y = 9,60416323435308

x1 = -4,87292277614858, x2 = -4,93646138807429, y = 9,62332022902308

x1 = -4,92179863147605, x2 = -4,86314760508309, y = 9,62226306870307

x1 = -4,92179863147605, x2 = -4,99022482893451, y = 9,61313208350927

x1 = -2,40469208211144, x2 = -4,9266862170088, y = 3,19805052855792

x1 = 4,91202346041056, x2 = -4,9266862170088, y = -0,000472782214211875

x1 = -4,92179863147605, x2 = 4,97067448680352, y = 0,0084822387304655

x1 = -4,56011730205279, x2 = -3,68035190615836, y = 6,39563571144366

x1 = -4,86314760508309, x2 = -4,8533724340176, y = 9,61320564049235

x1 = -4,87781036168133, x2 = -4,70674486803519, y = 9,51789353948068

x1 = -4,72629521016618, x2 = 4,94623655913979, y = -0,0857718099624991

x1 = -4,88269794721408, x2 = -4,86314760508309, y = 9,62004425474327

x1 = -3,60215053763441, x2 = -4,86314760508309, y = 6,40895534781053

x1 = -4,24731182795699, x2 = -4,95112414467253, y = 8,60689158424226

x1 = -4,88269794721408, x2 = -4,93157380254154, y = 9,62564326437134

x1 = -4,86314760508309, x2 = -3,6950146627566, y = 6,75009227805428