

Kirill Zabrodsky Exsercise 4 NOT END

Demission on reduction from 10D to 2D sphere

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace lab5
{
    public class Class1
    {
        Random rnd = new Random();
        const int size = 30;
        const int size1 = 10;
        const int size2 = 2;
        const int size3 = 100;
        int[] keys = new int[size3];
        List<double[,]> rd2List = new List<double[,]>();
        List<double[,]> array2List = new List<double[,]>();
        double[,] rd30 = new double[size, size1];
        double[,] rd2 = new double[size, size2];
        double[,] array30 = new double[size, size];
        double[,] array2 = new double[size, size];
        List<double[,]> newPopulation = new List<double[,]>();
        List<double[,]> p = new List<double[,]>();
        List<double[,]> parents = new List<double[,]>();
        List<double[,]> DList = new List<double[,]>();
        double[,] D = new double[30, 30];
        List<double> fitnessSum = new List<double>();
        public Class1()
        {
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size1; j++)
                {
                    rd30[i, j] = rnd.Next(0, 2001);
                    if (rd30[i, j] > 1000)
                    {
                        rd30[i, j] /= 1000;
                    }
                    else
                    {
                        rd30[i, j] = rd30[i, j] / 1000 * -1;
                    }
                }
            }
            double[] dif = new double[10];
            double sum = 0;
            double result = 0;
            for (int k = 0; k < size; k++)
            {
                for (int i = 0; i < size; i++)
                {
                    for (int j = 0; j < size1; j++)
                    {
                        dif[j] = Math.Pow((rd30[k, j] - rd30[i, j]), 2);
                    }
                    sum = dif.Sum();
                    result = Math.Sqrt(sum);
                }
            }
        }
    }
}
```

```

        array30[k, i] = result;
    }
}
for (int l = 0; l < size3; l++)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size2; j++)
        {
            rd2[i, j] = rnd.Next(0, 2001);
            if (rd2[i, j] > 1000)
            {
                rd2[i, j] /= 1000;
            }
            else
            {
                rd2[i, j] = rd2[i, j] / 1000 * -1;
            }
        }
    }
}

double[] dif1 = new double[size2];
for (int k = 0; k < size; k++)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size2; j++)
        {
            dif1[j] = Math.Pow((rd2[k, j] - rd2[i, j]), 2);
        }
        sum = dif1.Sum();
        result = Math.Sqrt(sum);
        array2[k, i] = result;
    }
}
double max30 = array30.Cast<double>().Max();
double max2 = array2.Cast<double>().Max();
for (int i = 0; i < size; i++)
{
    for (int j = 0; j < size; j++)
    {
        array30[i, j] = array30[i, j] / max30;
        array2[i, j] = array2[i, j] / max2;
    }
}
rd2List.Add(rd2);
array2List.Add(array2);
}
Fitness();
Cross();
//Mutation();
}
public void Fitness()
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            D[i, j] = array30[i, j] + array2[i, j];
        }
    }
}
}

```

```

        fitnessSum.Add(D.Cast<double>().Sum());
        DList.Add(D);
    }
    public void Cross()
    {
        for(int i = 0; i < size3; i++)
        {
            keys[i] = i;
        }
        Array.Sort(fitnessSum.ToArray(), keys);
        for (int i = 0; i < size3; i++)
        {
            newPopulation.Add(array2List[keys[i]]);
        }
        fitnessSum.Sort();
        for (int i = 0; i < size3; i++)
        {
            int a = rnd.Next(0, size3 / 2);
            int b = rnd.Next(0, size3 / 2);
            p.Add(newPopulation[a]);
            p.Add(newPopulation[b]);
            int c = rnd.Next(0, 2);
            if (c == 0)
            {
                parents.Add(p[0]);
            }
            else
            {
                parents.Add(p[1]);
            }
        }
    }
    public void Mutation()
    {
        for (int i = 0; i < size3; i++)
        {
            for (int j = 0; j < size; j++)
            {
                int value = rnd.Next(0, size);
                if (value == 0)
                {
                    double c = (-1;1)
                    parents[i][size,size] = c;
                }
            }
        }
        fitnessSum.Add(parents.Cast<double>().Sum());
    }
}
}
}

```