2-lab. Andrei Abramchuk

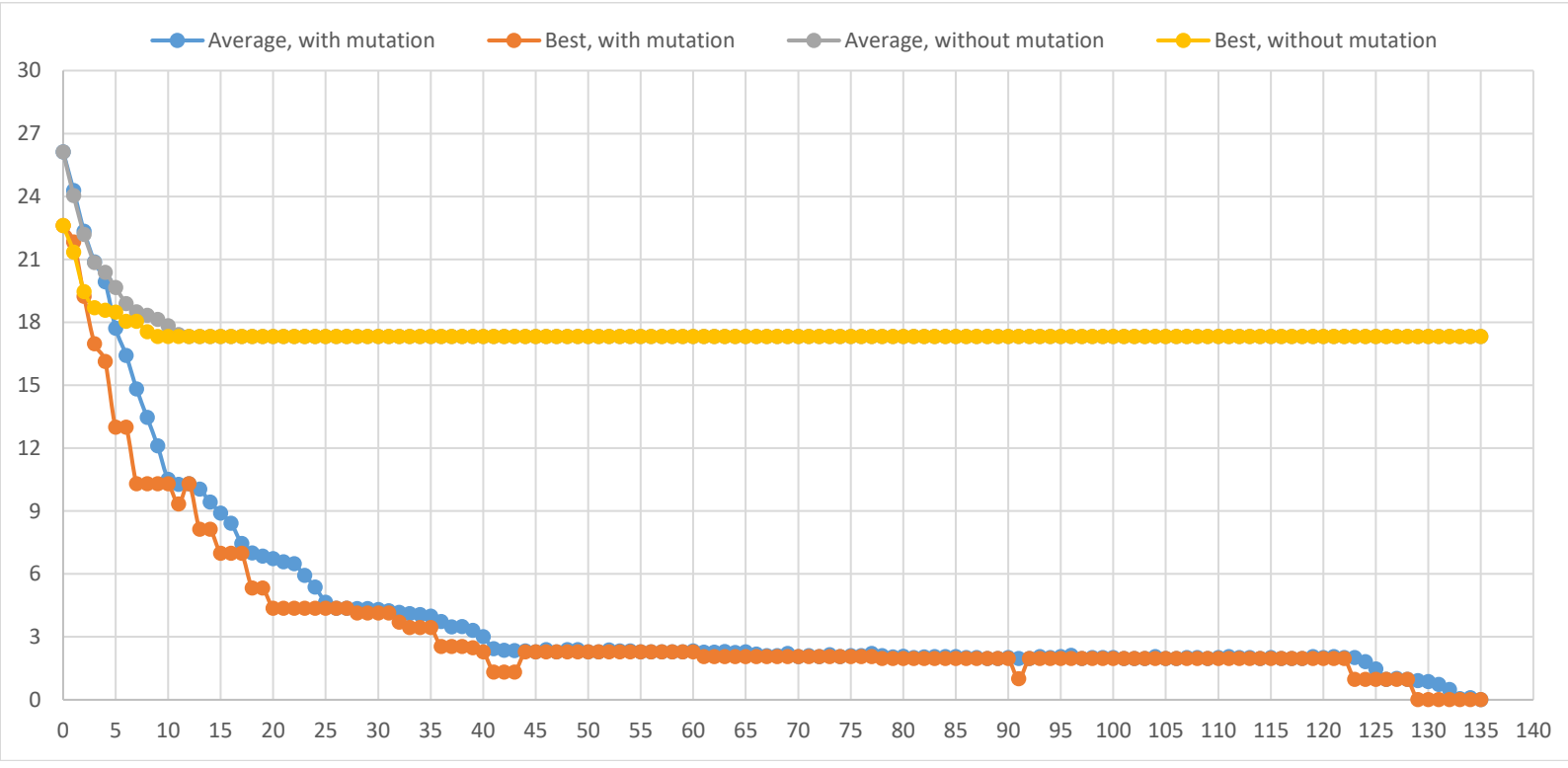
Minimization of Test Functions. High-dimensional Test Function

Task 1. 20-Dimensional Schwefel's function

1. Minimize y in the following way!
$y = nA + \sum^{n}_{i=1} (x2\ i - A\ \cos(2\pi xi))$
(1) Represent each of xi(i = 1, · · · , 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.
2. Show
(1) the graph of fitness vs generation.
(2) Create a population of 20 chromosomes at random, with fitness being y.
(3) Evolve this population till fitness dosen't change.

Results:

Best fitness with mutation equal 0;
Best fitness without mutation equal 17,325;



Source code:

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.util.*;

public class minClass {

    public ArrayList<ArrayList<Double>> ChrMas2 = new ArrayList<ArrayList<Double>>();
    //public ArrayList<Double> avrageVal = new ArrayList<Double>();
    public Integer generation = 0;
    Double average = 0.0;
```

```
    public static void main(String[] args) {
        minClass obj = new minClass();
        obj.random();
        int f = 1;
        while (f != 0) {
            if (obj.fun2() < 1)
                f = 0;
        }
    }

    public void random() {
        Random rand = new Random();
        for (int a = 0; a < 20; a++) {
            ArrayList<Double> Chr = new ArrayList<Double>();
            for (int b = 0; b < 20; b++) {
                Chr.add(rand.nextDouble() * 2 - 1);
            }
                        ChrMas2.add(Chr);
        }
    }

    public int fun2() {
        //get fitness
        ArrayList<Double> fitnessMas = new ArrayList<Double>();
        Map<Integer, Double> map = new HashMap<>();
        for (int a = 0; a < ChrMas2.size(); a++) {
            Double fitnessChr = 20.0;
            for (int b = 0; b < 20; b++) {
                fitnessChr += Math.pow(ChrMas2.get(a).get(b), 2)-
Math.cos(2*Math.PI*ChrMas2.get(a).get(b));
            }
            fitnessMas.add(fitnessChr);
                        map.put(a, fitnessChr);
        }
        //Sorting
        for (int i = 0; i < 20; i++)
            for (int j = 0; j < 20; j++)
                if (i != j && ((i < j && fitnessMas.get(i) > fitnessMas.get(j)) || (i > j &&
fitnessMas.get(i) < fitnessMas.get(j)))) {
                    for (int k = 0; k < 20; k++) {
                        Double gen = ChrMas2.get(i).get(k);
                        ChrMas2.get(i).set(k, ChrMas2.get(j).get(k));
                        ChrMas2.get(j).set(k, gen);
                    }
                    double fit = fitnessMas.get(i);
                    fitnessMas.set(i, fitnessMas.get(j));
                    fitnessMas.set(j, fit);
                }
       //crossover
        ArrayList<ArrayList<Double>> newGen = new ArrayList<ArrayList<Double>>();
        Random rand = new Random();
        for (int i = 0; i < 10; i++) {
            Double mama = fitnessMas.get(rand.nextInt(10)), papa = fitnessMas.get(rand.nextInt(10));
            int mama_key = getValue(map, mama), papa_key = getValue(map, papa);
            int cros_gen = rand.nextInt(18) + 1;
            ArrayList<Double> firstChild = new ArrayList<Double>();
            for (int j = 0; j < cros_gen; j++)
                firstChild.add(ChrMas2.get(mama_key).get(j));
            for (int j = cros_gen; j < 20; j++)
                firstChild.add(ChrMas2.get(papa_key).get(j));
            newGen.add(firstChild);

        }
        //mutation
        ChrMas2 = new ArrayList<ArrayList<Double>>(newGen);
        for (int i = 0; i < ChrMas2.size(); i++)
            for (int j = 0; j < ChrMas2.get(0).size(); j++)
                if (rand.nextInt(20) == 13) {
                    if (ChrMas2.get(i).get(j) != 1) {
                        ArrayList<Double> t = new ArrayList<Double>(ChrMas2.get(i));
                        t.set(j, 1.1);
                        ChrMas2.set(i, t);
                    } else {
                        ArrayList<Double> t = new ArrayList<Double>(ChrMas2.get(i));
                        t.set(j, 0.0);
                        ChrMas2.set(i, t);
                    }
                }

        average = 0.0;
        for (int i = 0; i < 20; i++)
            average += fitnessMas.get(i);
```

```
        average /= 20;
        System.out.println("Average value: " + average);
        generation++;
        write(fitnessMas.get(0).toString(), average.toString(), generation.toString());
    return end(average);
    }

    public static Integer getValue(Map<Integer, Double> map, Double value) {
        Set<Map.Entry<Integer, Double>> entrySet = map.entrySet();
        Double desiredFitness = value;//что хотим найти
        Integer key = 0;
        for (Map.Entry<Integer, Double> pair : entrySet) {
            if (desiredFitness.equals(pair.getValue())) {
                key = pair.getKey();// нашли наше значение и возвращаем  ключ
            }
        }
        return key;
    }

    public void write(String maxfitness, String average, String generation) {

        try {
            PrintStream printStream1 = new PrintStream(new FileOutputStream("D:\\maxfitness.txt", true),
true);
            PrintStream printStream2 = new PrintStream(new FileOutputStream("D:\\average.txt", true),
true);
            PrintStream printStream3 = new PrintStream(new FileOutputStream("D:\\generation.txt", true),
true);

            try {
                //Записываем текст
                printStream1.println(maxfitness);
                printStream2.println(average);
                printStream3.println(generation);
            } finally {
                //После чего мы должны закрыть файл, Иначе файл не запишется
                printStream1.close();
                printStream2.close();
                printStream3.close();
            }
        } catch(IOException e) {
            throw new RuntimeException(e);
        }
    }

    public int end(Double average) {
        int sch = 0;
        if (average == 0.0)
        return 1;
        else return  0;
    }
}
```

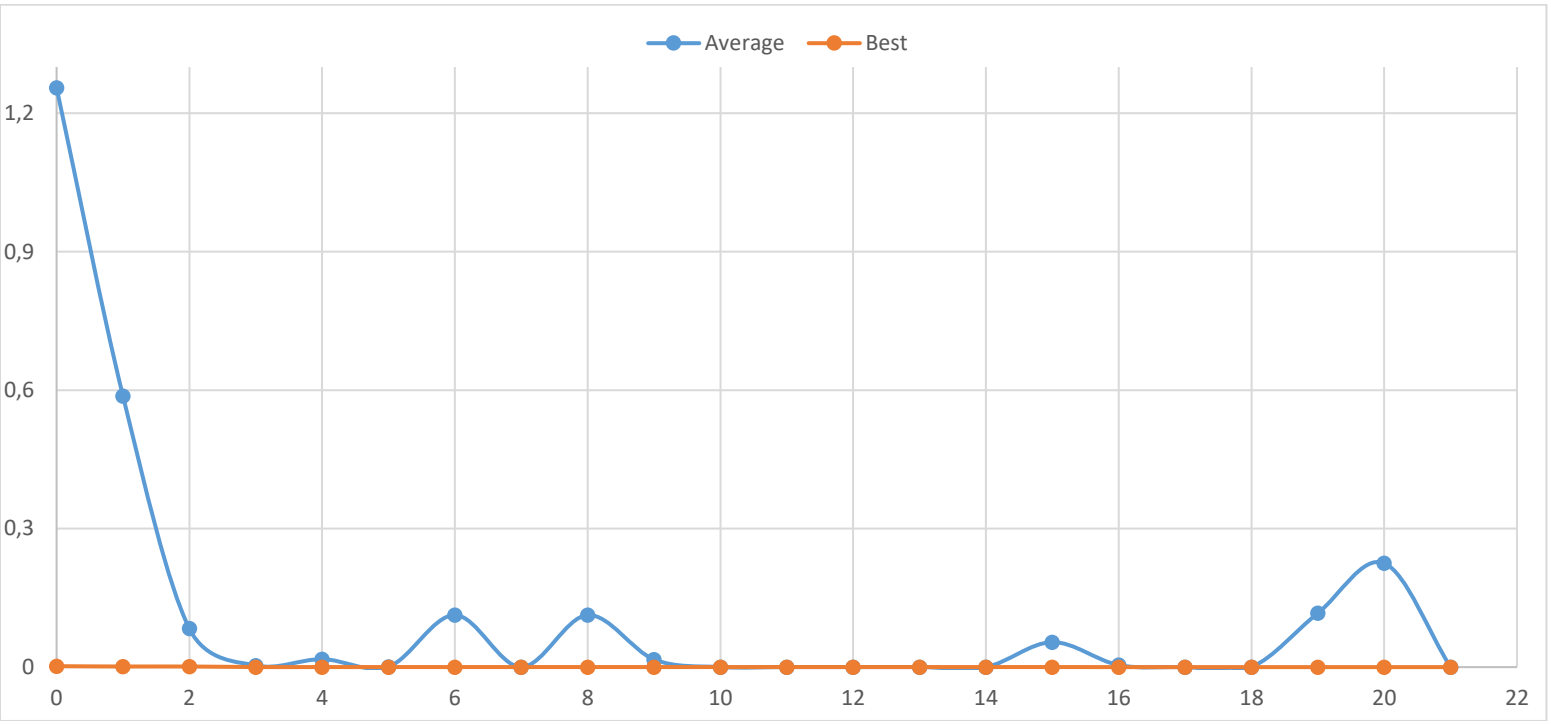
Task 2. 2-D version of Schwefel's function

1. Minimize in the following way!

$y = nA + \sum^{n}_{i=1} (x2\ i - A \cos(2\pi xi))$

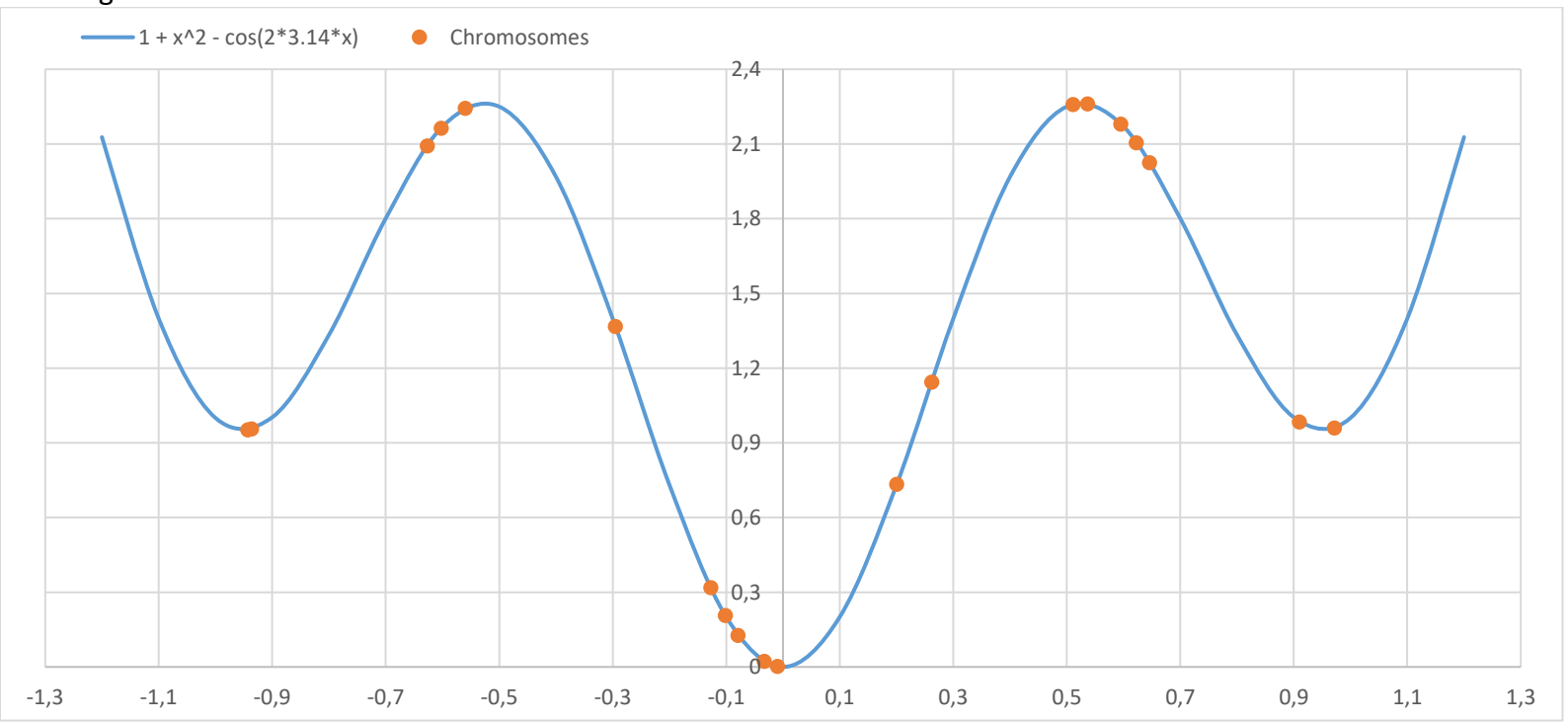
(1) Represent value of x by a 10-bit binary chromosome.
(2) Create a population of 20 chromosomes at random, with fitness being y.
(3) Evolve this population till fitness dosn't change.
2. Show
(1) the graph of fitness vs generation.
(2) all 20 points (x, y) in the 1st, an intermediate, and final generation.

Results:

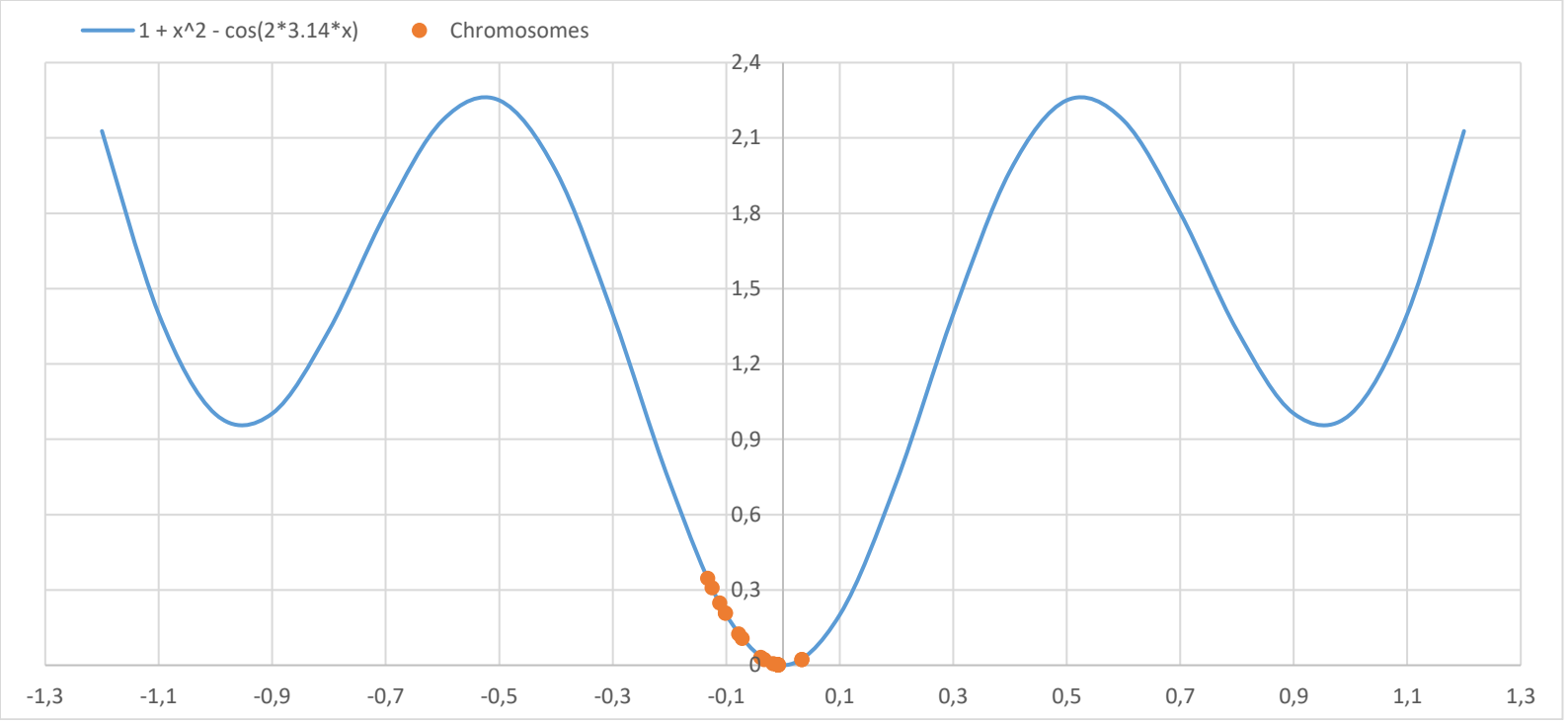


The best values are within [0; 0.002]. Therefore, the schedule merges with the horizontal x-axis.

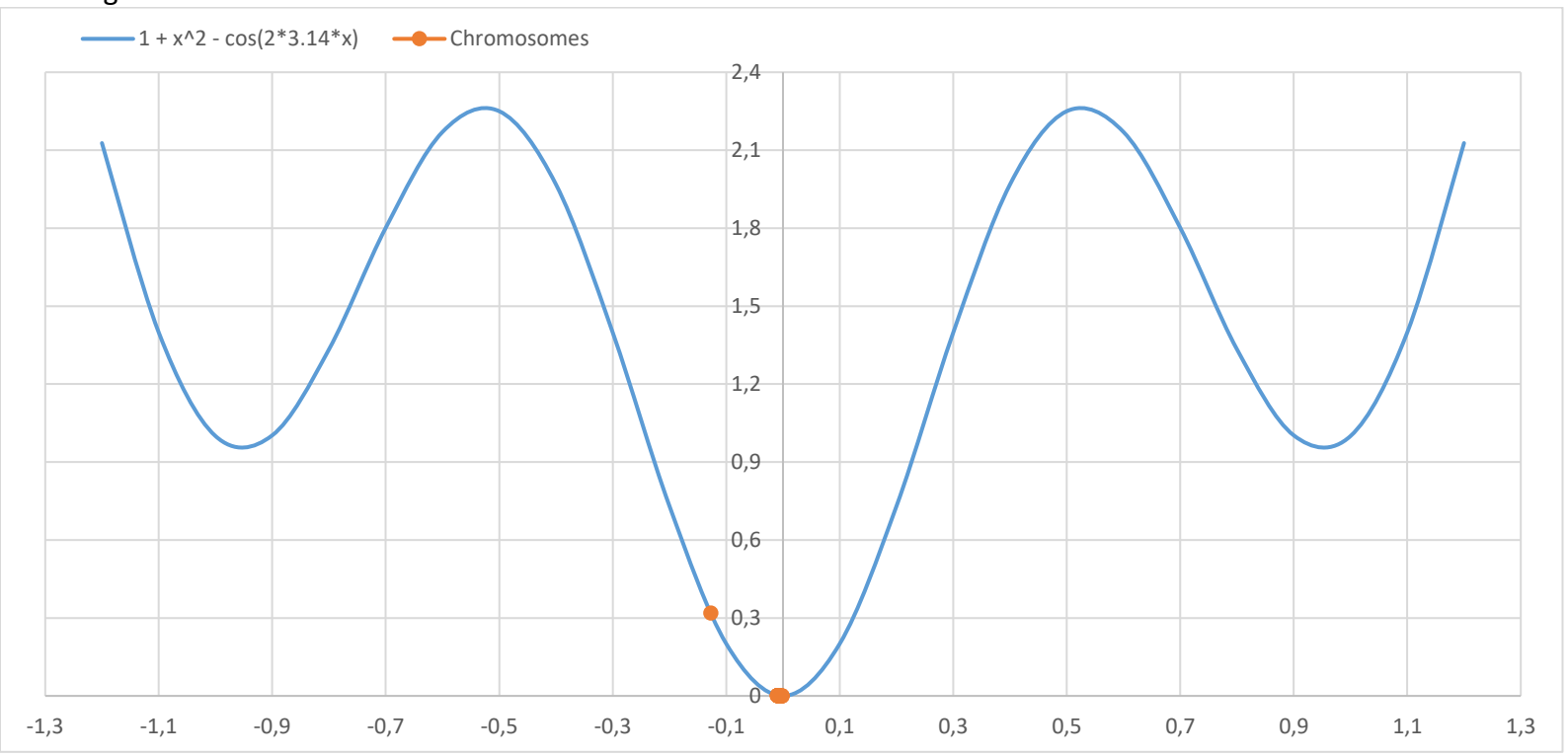
1-st generation:



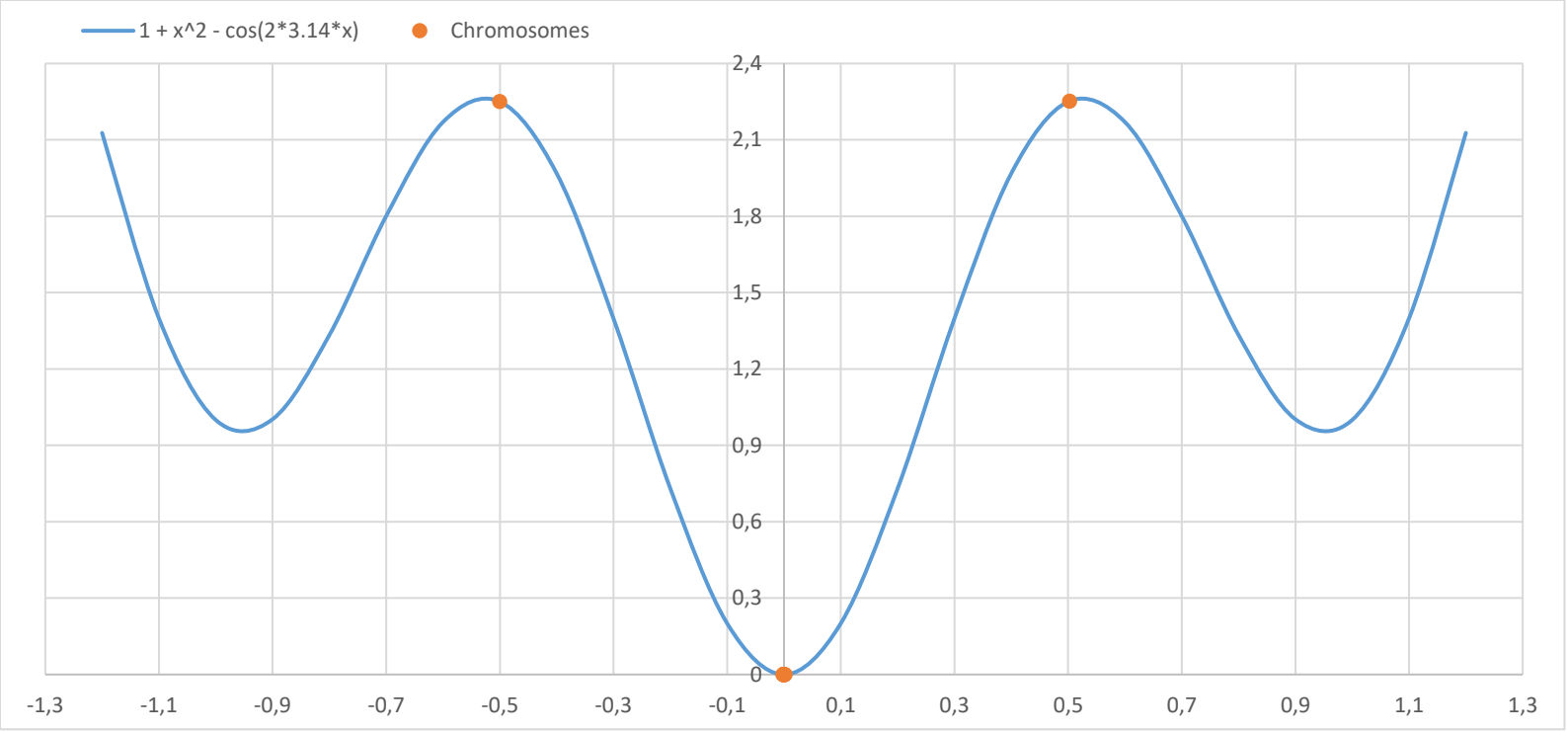
3-rd generation:



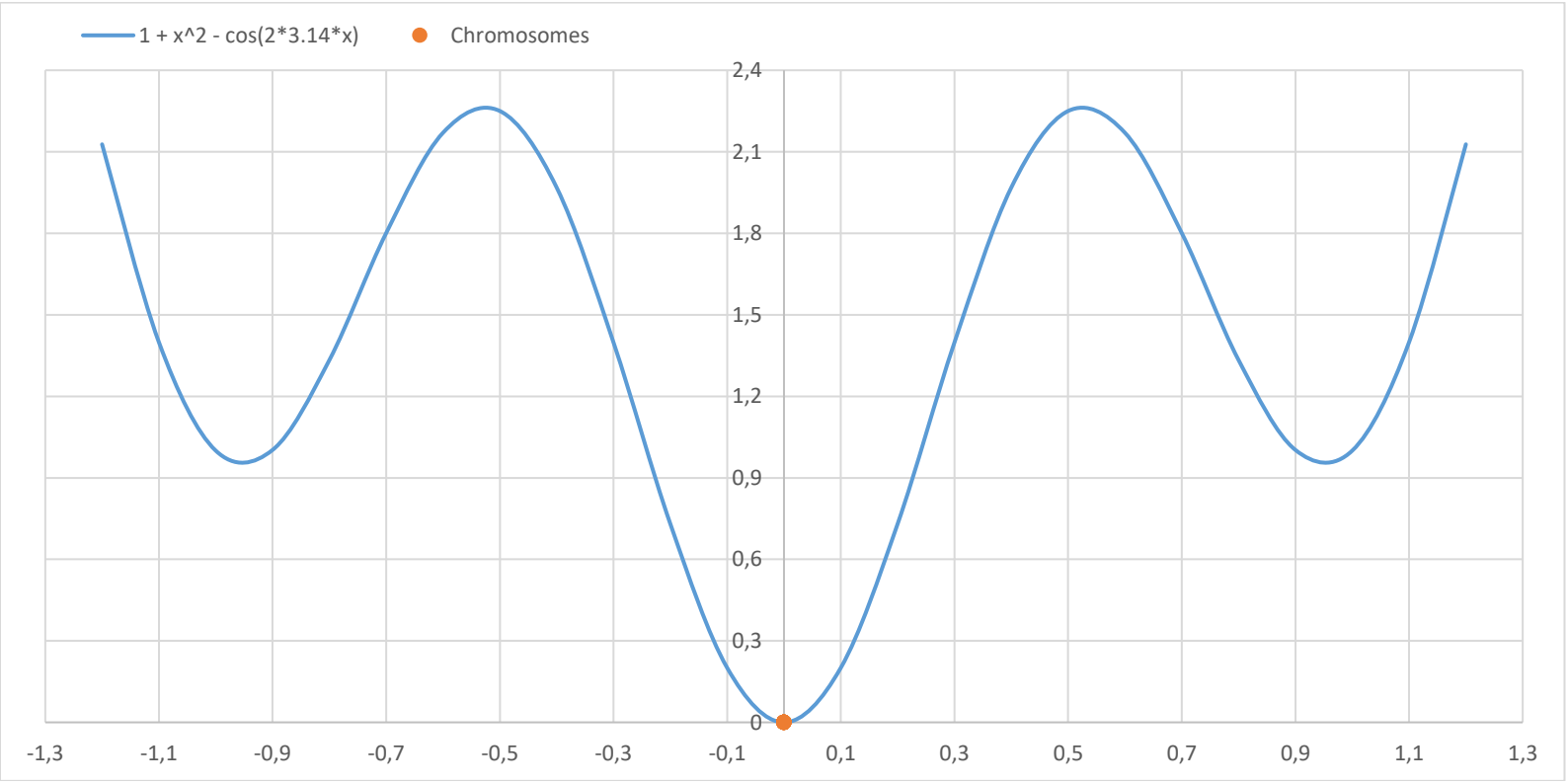
5-th generation:



21-st generation:



22-nd generation:



Source code:

```
package com.abrams;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;

public class MainTask2
{
   static Random random = new Random(/*System.currentTimeMillis()*/2210812);

   static class Chromosome
   {
      double[] genes;
      public Chromosome()
      {
         genes = new double[11];
         for(int i = 0; i < 11; i++)
```

```
            genes[i] = random.nextInt(2);
      }
      double getFitness()
      {
         double x = 0;
         for(int i = 1; i < 11; i++)
            if(genes[i] == 1)
               x += Math.pow(2, i - 1);
         x /= 1023;
         if(genes[0] == 0)
            x *= -1;
         return 1 + (x * x - Math.cos(Math.PI * 2 * x));
      }
   }

   static class Generation
   {
      Chromosome[] chromosomes;
      public Generation()
      {
         chromosomes = new Chromosome[20];
         for(int i = 0; i < 20; i++)
            chromosomes[i] = new Chromosome();
      }
      double getAverageFitness()
      {
         double counter = 0;
         for(Chromosome i : chromosomes)
            counter += i.getFitness();
         return counter / 20;
      }
      void sort()
      {
         Arrays.sort(chromosomes, (o1, o2) ->
         {
            double o1fit = o1.getFitness();
            double o2fit = o2.getFitness();
            if(o1fit < o2fit)
               return -1;
            else if(o1fit > o2fit)
               return 1;
            else
               return 0;
         });
      }
      Chromosome[] get2RandomChromosomes()
      {
         Chromosome[] random2 = new Chromosome[2];
         random2[0] = chromosomes[random.nextInt(10)];
         random2[1] = chromosomes[random.nextInt(10)];
         return random2;
      }

      Chromosome getBestChromosome()
      {
         return chromosomes[0];
      }
   }

   static Chromosome crossover(Chromosome first, Chromosome second)
   {
      int crossoverPoint = random.nextInt(first.genes.length);
      Chromosome child = new Chromosome();
      System.arraycopy(first.genes, 0, child.genes, 0, crossoverPoint);
      System.arraycopy(second.genes, crossoverPoint, child.genes, crossoverPoint,
      second.genes.length - crossoverPoint);

      if(random.nextInt(20) == 13)
      {
         int pos = random.nextInt(child.genes.length);
         if(child.genes[pos] == 0)
            child.genes[pos] = 1;
         else
            child.genes[pos] = 0;
      }
      return child;
   }

   static ArrayList<Double> fitnessHistory;
   static boolean isLastGenerationUnchanged()
   {
      double tmp = fitnessHistory.get(fitnessHistory.size() - 1);
```

```
      if(tmp == 0.0)
         return true;
      if(fitnessHistory.size() <= 100)
         return false;
      for(int i = 0; i < 100; i++)
         if(fitnessHistory.get(fitnessHistory.size() - i - 1) != tmp)
            return false;
      return true;
   }


   public static void main(String[] args)
   {
      ArrayList<Generation> generations = new ArrayList<>();
      generations.add(new Generation());
      generations.get(0).sort();
      fitnessHistory = new ArrayList<>();
      fitnessHistory.add(generations.get(0).getAverageFitness());
      System.out.println(String.format("%d\t\t%.4f\t\t%.4f", 0, generations.get(0).getAverageFitness(),
                              generations.get(0).getBestChromosome().getFitness()));
      for(Chromosome j : generations.get(0).chromosomes)
      {
         double x = 0;
         for(int i = 1; i < 11; i++)
            if(j.genes[i] == 1)
               x += Math.pow(2, i - 1);
         x /= 1023;
         if(j.genes[0] == 0)
            x *= -1;
         System.out.println(String.format("%.4f\t%.4f", x, j.getFitness()));
      }

      for(int i = 1; ; i++)
      {
         Generation newGeneration = new Generation();
         for(int j = 0; j < newGeneration.chromosomes.length; j++)
         {
            Chromosome[] random2 = generations.get(generations.size() - 1).get2RandomChromosomes();
            newGeneration.chromosomes[j] = crossover(random2[0], random2[1]);
         }
         newGeneration.sort();
         generations.add(newGeneration);
         generations.remove(0);
         fitnessHistory.add(newGeneration.getAverageFitness());
         System.out.println(String.format("%d\t\t%.4f\t\t%.4f", i, newGeneration.getAverageFitness(),
                                 newGeneration.getBestChromosome().getFitness()));
         if(i == 2 || i == 4 || i == 6 || i == 20)
            for(Chromosome j : newGeneration.chromosomes)
            {
               double x = 0;
               for(int z = 1; z < 11; z++)
                  if(j.genes[z] == 1)
                     x += Math.pow(2, z - 1);
               x /= 1023;
               if(j.genes[0] == 0)
                  x *= -1;
               System.out.println(String.format("%.4f\t%.4f", x, j.getFitness()));
            }
         if(isLastGenerationUnchanged())
            break;

      }
   }
}
```