PRACTICE 4

Minimization of 2D functions

In these practice we need to find multiple minimums of $y = \sin^6(5\pi x)$

1. 20 random chromosomes with 10 genes each
2. Search for 2 random parents and get their children by uniform crossover
3. Mutate children
4. Choose 2 survivors from 2 parents and 2 children by
   - IF d(p1, c1) + d(p2, c2) > d(p1, c2) + d(p2, c1)
     * IF f(c1) > f(p1) THEN replace p1 with c1
     * IF f(c2) > f(p2) THEN replace p2 with c2
     - ELSE
     * IF f(c2) > f(p1) THEN replace p1 with c2
     * IF f(c1) > f(p2) THEN replace p2 with c1
5. Repeat 2-4 while reach all solutions.

Code

Chromosome.java

```
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class Chromosome {
    private List<Boolean> genes = new ArrayList<>();

    private static Random random = new Random();

    private Double fitness = null;

    public Chromosome() {
        for(int i = 0; i < 10; i++){
            genes.add(random.nextBoolean());
        }
    }

    public Chromosome(List<Boolean> genes) {
        this.genes = genes;
    }

    public Double getFitness() {
        fitness = Math.pow(Math.sin(5.0 * Math.PI * getX()) ,6.0);

        return fitness;
    }

    public void setFitness(Double fitness) {
        this.fitness = fitness;
    }

    public List<Boolean> getGenes() {
        return genes;
    }

    public void setGenes(List<Boolean> genes) {
        this.genes = genes;
    }

    public Double getX(){
        Double sum = 0.0;

        for(int i = 0; i < 10; i++){
            sum += genes.get(i) ? Math.pow(2.0, i) : 0;
        }

        return sum / 1023.0;
    }

    public static List<Chromosome> crossover(Chromosome father, Chromosome mother, PrintWriter printWriter){
```

```
        List<Chromosome> survivers = new ArrayList<>();

        List<Boolean> firstChildGenes = new ArrayList<>();
        List<Boolean> secondChildGenes = new ArrayList<>();

        for(int i = 0; i < 10; i++){
            if(random.nextBoolean()){
                firstChildGenes.add(father.getGenes().get(i));
                secondChildGenes.add(mother.getGenes().get(i));
            }else{
                firstChildGenes.add(mother.getGenes().get(i));
                secondChildGenes.add(father.getGenes().get(i));
            }
        }

        Chromosome firstChild = new Chromosome(firstChildGenes);
        firstChild.mutate();

        Chromosome secondChild = new Chromosome(secondChildGenes);
        secondChild.mutate();

        printWriter.println("" + father.getFitness() + "\t" + firstChild.getFitness());
        printWriter.println("" + mother.getFitness() + "\t" + secondChild.getFitness() + "\n");

        if(getDistance(father, firstChild)+getDistance(mother, secondChild) > getDistance(father, secondChild) +
getDistance(mother, firstChild)){
            survivers.add(father.getFitness() < firstChild.getFitness() ? father : firstChild);
            survivers.add(mother.getFitness() < secondChild.getFitness() ? mother : secondChild);
        }else{
            survivers.add(father.getFitness() < secondChild.getFitness() ? father : secondChild);
            survivers.add(mother.getFitness() < firstChild.getFitness() ? mother : firstChild);
        }

        return survivers;
    }

    private static Double getDistance(Chromosome first, Chromosome second){
        return Math.abs(first.getX() - second.getX());
    }

    public void mutate(){
        for(int i = 0; i< 10; i++){
            if(random.nextDouble() < 0.1){
                Boolean value = !genes.get(i);
                genes.remove(i);
                genes.add(i, value);
            }
        }
    }
}
```

Population.java

```
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

/**
 * Created by Arty on 26.10.2016.
 */
public class Population {
    private List<Chromosome> chromosomes = new ArrayList<>();

    private Random random = new Random();

    public Population() {
        for(int i = 0; i < 20; i++){
            chromosomes.add(new Chromosome());
        }
    }

    public Population(List<Chromosome> chromosomes) {
        this.chromosomes = chromosomes;
    }

    public List<Chromosome> getChromosomes() {
        return chromosomes;
    }

    public void setChromosomes(List<Chromosome> chromosomes) {
        this.chromosomes = chromosomes;
    }

    public Population getNextGeneration(int iteration) throws FileNotFoundException, UnsupportedEncodingException {
        List<Chromosome> nextGenerationChromosomes = new ArrayList<>();

        nextGenerationChromosomes.addAll(chromosomes);

        PrintWriter printWriter = new PrintWriter("result/crossover/" + iteration + ".txt", "UTF-8");
```

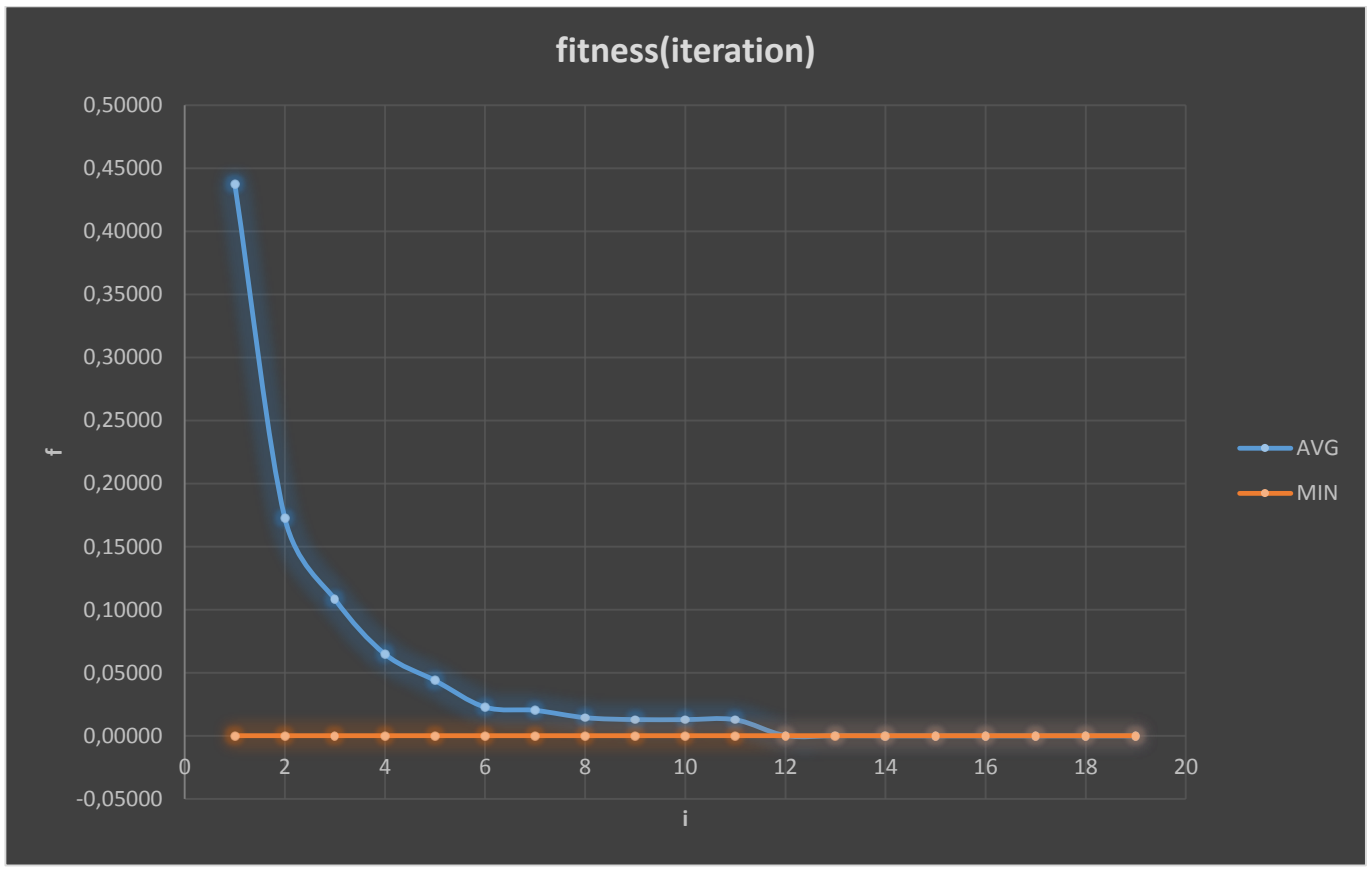
```
        for(int i = 0; i < 10; i++){
            Chromosome p1;
            Chromosome p2;

            do{
                p1 = nextGenerationChromosomes.get(random.nextInt(20));
                p2 = nextGenerationChromosomes.get(random.nextInt(20));
            }
            while (p1 == p2);

            List<Chromosome> survivers = Chromosome.crossover(p1, p2, printWriter);

            nextGenerationChromosomes.remove(p1);
            nextGenerationChromosomes.remove(p2);

            nextGenerationChromosomes.addAll(survivers);
        }

        printWriter.close();
        return new Population(nextGenerationChromosomes);
    }

    public Double getMinimumFitness(){
        int minIndex = 0;

        for(int i = 0; i < 20; i++){
            minIndex = chromosomes.get(i).getFitness() < chromosomes.get(minIndex).getFitness() ? i : minIndex;
        }
        return chromosomes.get(minIndex).getFitness();
    }

    public Double getAverageFitness(){
        double sum = 0.0;

        for(int i = 0; i < 20; i++){
            sum += chromosomes.get(i).getFitness();
        }

        return sum / 20.0;
    }
}
```

Main.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;

public class Main {

    public static void main(String[] args) throws FileNotFoundException, UnsupportedEncodingException {
        new File("E:\\A_WORK\\5cource\\siit\\lab04\\result").mkdir();
        new File("E:\\A_WORK\\5cource\\siit\\lab04\\result\\crossover").mkdir();

        PrintWriter printMax = new PrintWriter("result/max.txt", "UTF-8");
        PrintWriter printAvg = new PrintWriter("result/avg.txt", "UTF-8");

        List<Population> populations = new ArrayList<>();
        Population currentPopulation = new Population();

        int iteration = 0;
//        while (!isTenPrevEquals(populations)) {
        while (iteration != 25) {

            //if (iteration % 100 == 0) System.out.println(iteration);

            printAvg.println(currentPopulation.getAverageFitness());
            printMax.println(currentPopulation.getMinimumFitness());

            populations.add(currentPopulation);
            currentPopulation = currentPopulation.getNextGeneration(iteration);

            iteration++;
        }

        printAvg.close();
        printMax.close();

        for (int i = 0; i <= iteration; i += (int) (iteration * 0.25)) {
            if(i > iteration * 0.76) i = iteration - 1;
            PrintWriter printIterationX = new PrintWriter("result/" + i + "X.txt", "UTF-8");
            PrintWriter printIterationY = new PrintWriter("result/" + i + "Y.txt", "UTF-8");

            Population p = populations.get(i);

            for (Chromosome c : p.getChromosomes()) {
                printIterationX.println(c.getX());
                printIterationY.println(c.getFitness());
```

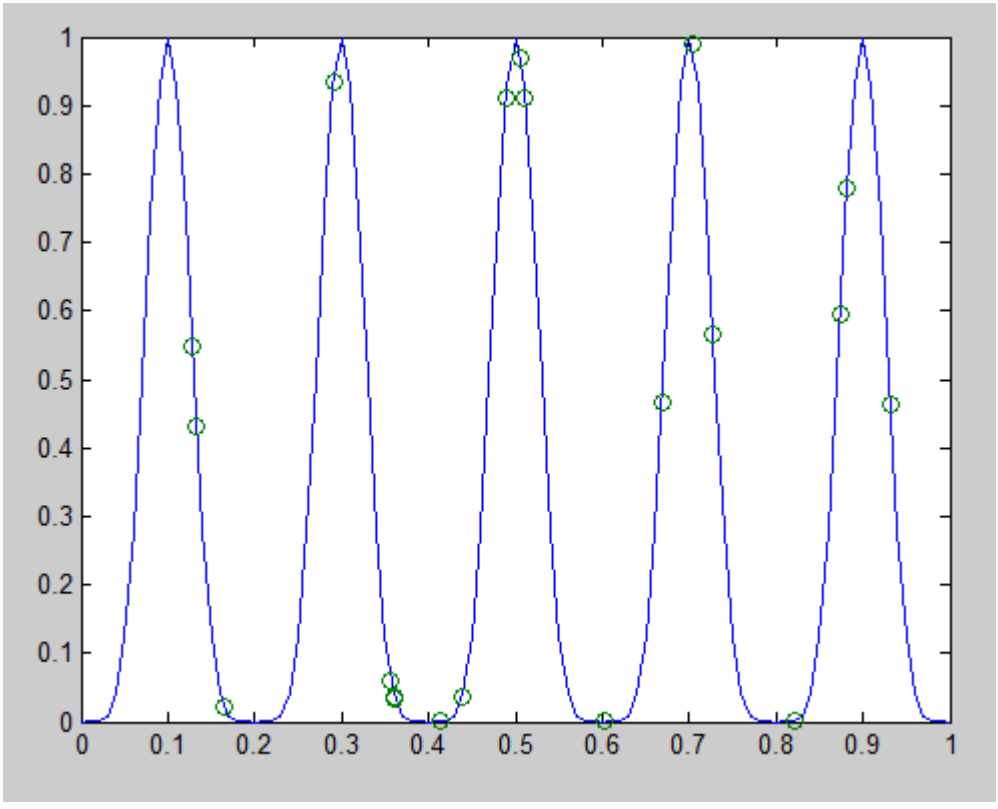
```
            }

            printIterationX.close();
            printIterationY.close();
        }


    }
}
```

Results



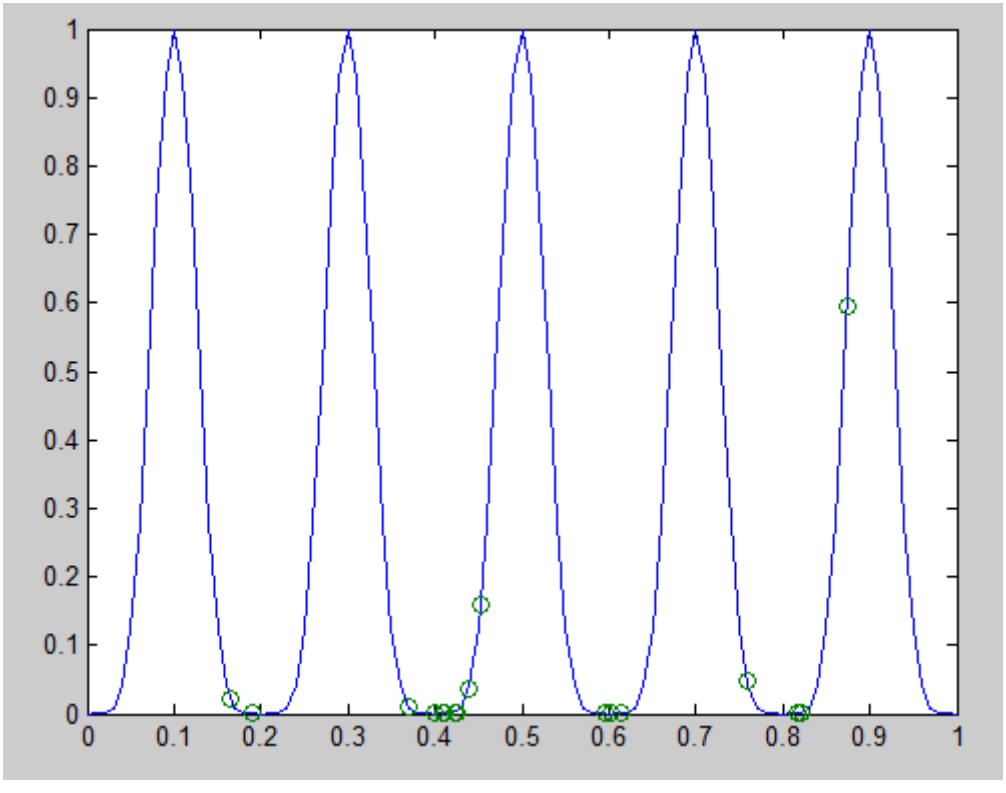
We get such graphic of minimums because even on the first iteration we can get random value with can be almost a solution. See the Iteration 1 graphic.

Iteration 1



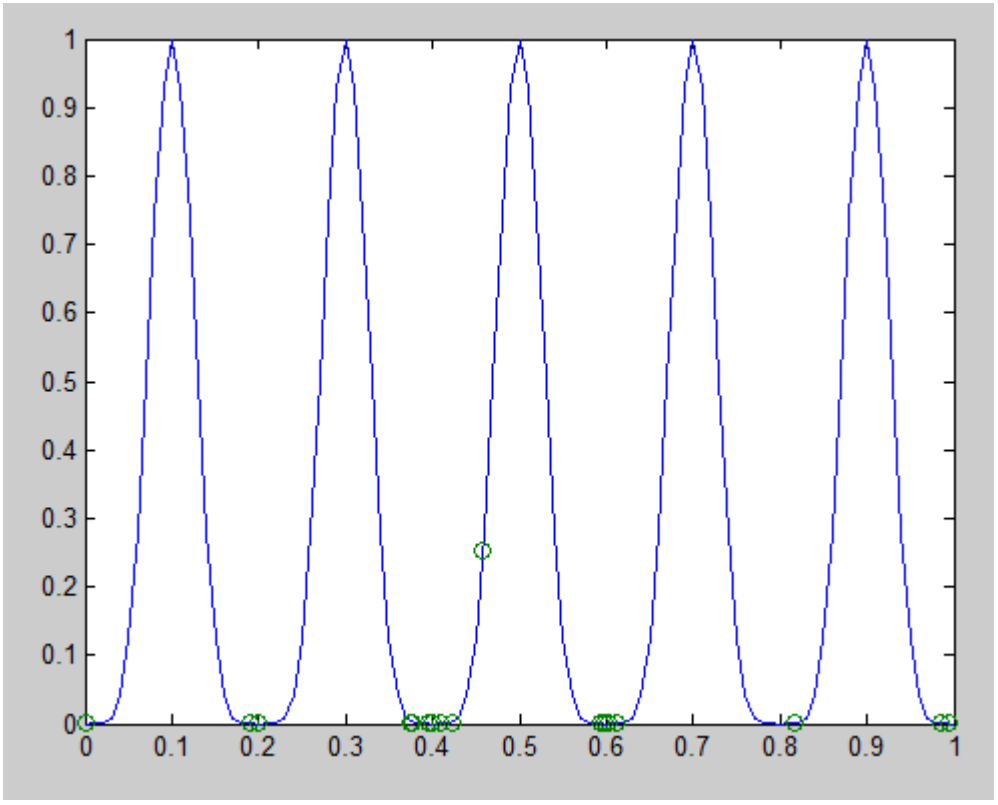
P1	0,035686168	C1	0,262303
P2	0,430882212	C2	0,031221
P1	0,03295298	C1	0,950042
P2	0,910253697	C2	0,009593
P1	0,547441766	C1	0,009593
P2	0,009592795	C2	1,00E-05
P1	0,001191897	C1	0,160728
P2	0,035686168	C2	0,867853
P1	0,462861697	C1	0,973987
P2	0,467482243	C2	0,490751
P1	0,462861697	C1	1,13E-05
P2	8,65E-05	C2	7,20E-04
P1	0,970509984	C1	7,20E-04
P2	0,03295298	C2	0,623639
P1	0,035686168	C1	0,834752
P2	0,932774234	C2	0,090309
P1	0,778166669	C1	0,921874
P2	0,001191897	C2	2,57E-04
P1	0,989293111	C1	0,661377
P2	0,022893698	C2	3,91E-11

Iteration 5



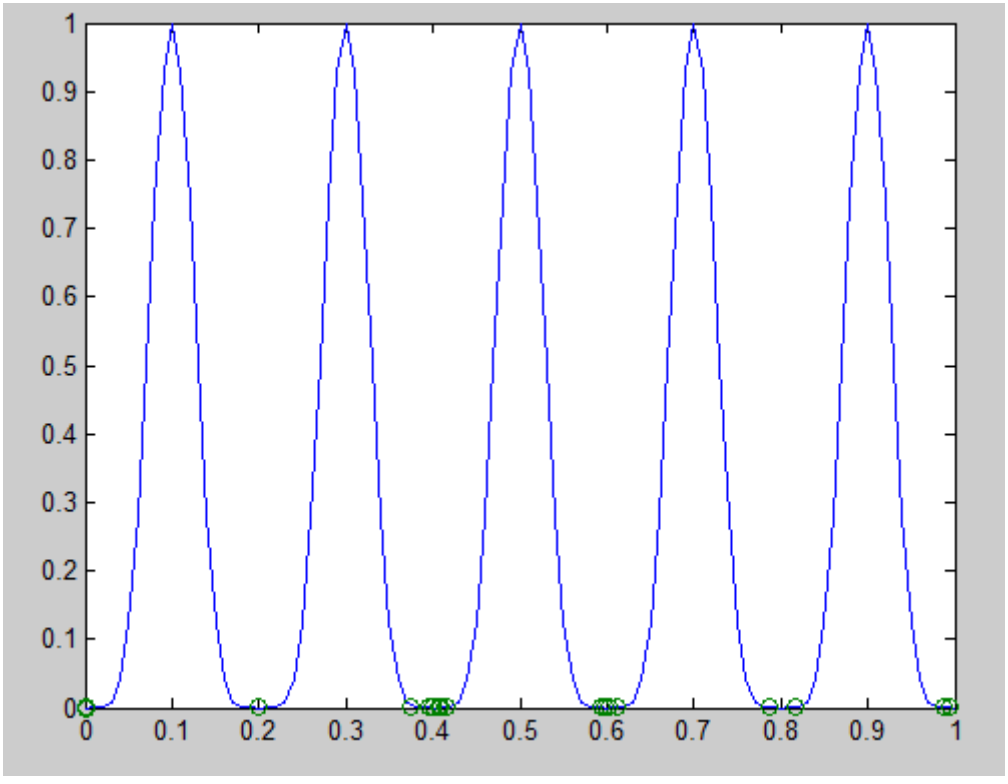
P1	3,91E-11	C1	8,39E-16
P2	0,158029	C2	0,144972
P1	0,009593	C1	0,96091
P2	0,001192	C2	0,034757
P1	3,91E-11	C1	0,365014
P2	0,5951	C2	0,998805
P1	0,002083	C1	0,399642
P2	1,40E-08	C2	0,022231
P1	0,001192	C1	0,62838
P2	0,002636	C2	0,811196
P1	1,13E-05	C1	0,993223
P2	0,001628	C2	0,004096
P1	0,001192	C1	0,024264
P2	0,002636	C2	0,84235
P1	1,02E-04	C1	0,011032
P2	0,001628	C2	0,675384
P1	0,022894	C1	0,002887
P2	0,035686	C2	0,090309
P1	1,40E-08	C1	4,95E-07
P2	0,001192	C2	0,003926

Iteration 9



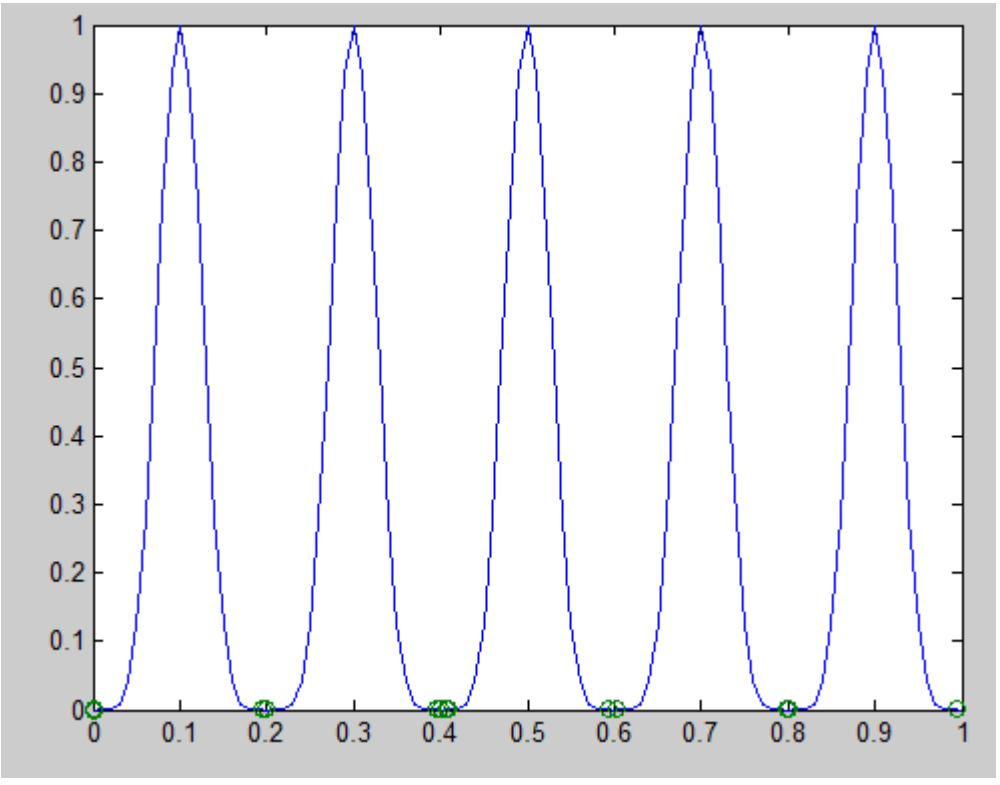
P1	1,00E-05	C1	0,675384
P2	2,20E-10	C2	2,20E-10
P1	4,95E-07	C1	4,95E-07
P2	4,95E-07	C2	0,023571
P1	6,30E-09	C1	0,984484
P2	1,40E-08	C2	2,51E-05
P1	0,002887	C1	1,92E-04
P2	1,40E-08	C2	0,023571
P1	8,38E-10	C1	0,901082
P2	2,04E-07	C2	1,53E-04
P1	3,91E-11	C1	0,020951
P2	2,04E-07	C2	0,486077
P1	4,95E-07	C1	0,786584
P2	2,57E-07	C2	0,904182
P1	1,40E-08	C1	0,012219
P2	2,57E-07	C2	0,303752
P1	2,57E-07	C1	0,533188
P2	2,51E-05	C2	0,891533
P1	2,20E-10	C1	0,395248
P2	8,38E-10	C2	4,04E-09

Iteration 13



P1	4,59E-05	C1	0,412931
P2	0	C2	0,127901
P1	0,002887	C1	0,509539
P2	4,95E-07	C2	1,28E-06
P1	8,38E-10	C1	0,57126
P2	1,13E-05	C2	0,003301
P1	2,20E-10	C1	0,756698
P2	2,04E-07	C2	7,63E-04
P1	2,04E-07	C1	0,998805
P2	1,28E-06	C2	0,007716
P1	2,20E-10	C1	0,927416
P2	1,28E-06	C2	0,670726
P1	0	C1	2,02E-08
P2	2,51E-05	C2	0,013064
P1	6,30E-09	C1	0,730217
P2	9,42E-05	C2	6,11E-13
P1	2,91E-06	C1	0,666057
P2	8,39E-16	C2	0,090309
P1	4,59E-05	C1	0,834752
P2	1,13E-05	C2	0,69849

Iteration 18



P1	1,13E-05	C1	0,028753
P2	1,13E-05	C2	0,003301
P1	4,95E-07	C1	0,006385
P2	4,01E-07	C2	0,968693
P1	8,38E-10	C1	8,92E-07
P2	4,01E-07	C2	0,030381
P1	2,57E-07	C1	2,57E-07
P2	1,28E-06	C2	0,481412
P1	2,57E-07	C1	1,80E-06
P2	1,13E-05	C2	0,034757
P1	4,95E-07	C1	1,07E-06
P2	8,39E-16	C2	2,84E-08
P1	8,39E-16	C1	0,001628
P2	8,39E-16	C2	8,39E-16
P1	8,39E-16	C1	0,001396
P2	4,01E-07	C2	4,69E-04
P1	2,57E-07	C1	1,80E-06
P2	1,80E-06	C2	2,57E-07
P1	8,38E-10	C1	0,395248
P2	2,91E-06	C2	0,001628

We can say that not all runs of program we can get all the solutions. Usually algorithm finds 4/6 solutions.