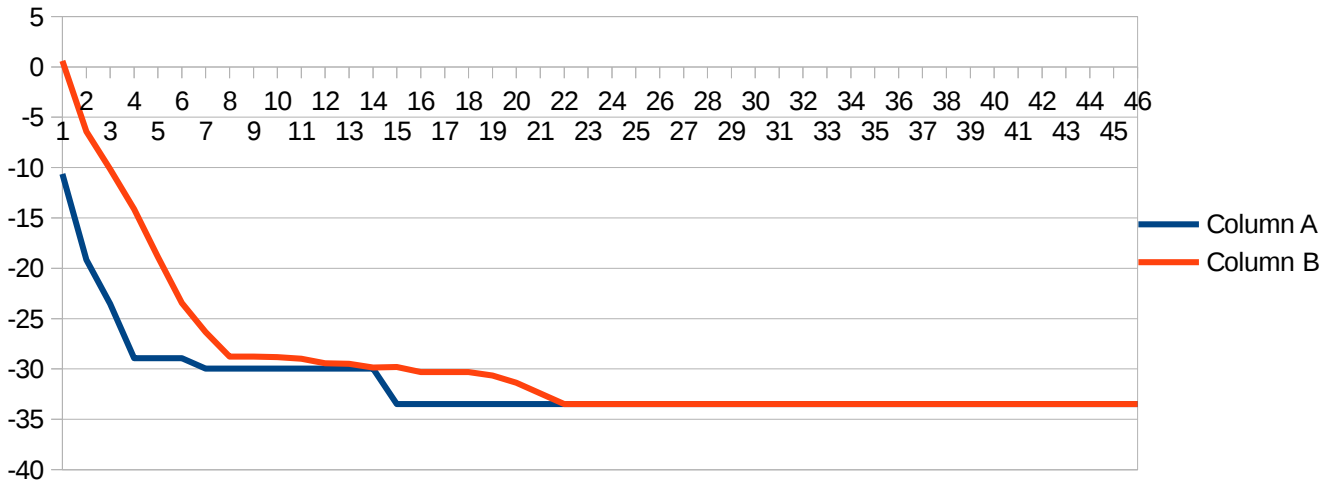
TASK1:

Comments for the code your can see in previous work. This code have small changes.

```java
public class Main {
    public static Double[][] generation = new Double[20][20];
    public static Double[][] good = new Double[10][20];
    public static Double[][] childes = new Double[10][20];
    public static Integer childCount = 0;
    public static Integer checkCount = 0;
    public static Double prevMax = 0.0;
    public static void main(String[] args) {
        createGenerate();
        while(true) {
            Arrays.sort(generation, new Comparator<Double[]>() {
                @Override
                public int compare(Double[] o1, Double[] o2) {
                    Double int1 = fitness(o1);
                    Double int2 = fitness(o2);
                    return (int)(int1 - int2) * 100;
                }
            });
            Double max = fitness(generation[0]);
            if(prevMax.equals(max))
               checkCount ++;
            else
                checkCount = 0;
            prevMax = max;
            System.out.printf("%.3f %.3f", max,average());
            System.out.println();
            if(checkCount > 30) {
                return;
            }
            setGood();
            Random random = new Random();
            childCount = 0;
            for (int i = 0; i < 5; i++) {
                createChildes(good[random.nextInt(10)], good[random.nextInt(10)]);
            }
            createNewGeneration();
        }
    }
    public static void createGenerate() {
        Random random = new Random();
        for(int i=0;i<20;i++) {
            for (int j = 0; j < 20; j++) {
                generation[i][j] = random.nextInt(1000) / 100.0 - 5;
            }
        }
    }
    public static Double fitness(Double[] obj) {
        Double sum = 0.0;
        for (int i=0; i<obj.length;i++) {
            sum += obj[i] * Math.sin(Math.abs(obj[i]));
        }
        return sum;
    }
    public static void setGood() {
        for(int i =0; i< 10; i++) {
            good[i] = generation[i];
        }
```

```
    }
    public static void createChildes(Double[] parent1, Double[] parent2) {
            Random random = new Random();
        Integer delimiter = random.nextInt(20);
        Double[] child1 = new Double[20];
        Double[] child2 = new Double[20];
        for(int i = 0;i<20;i++) {
            if(i<=delimiter)
                child1[i] = parent1[i];
            else
                child1[i] = parent2[i];
        }
        for(int i = 0;i<20;i++) {
            if(i<=delimiter)
                child2[i] = parent2[i];
            else
                child2[i] = parent1[i];
        }
        childes[childCount] = child1;
        childCount++;
        childes[childCount] = child2;
        childCount++;
    }
    public static void createNewGeneration() {
        for (int i=0;i<20;i++) {
            if(i<10)
            generation[i] = good[i];
            else
                generation[i] = childes[i-10];
        }
    }
    public static Double average() {
        Double s = 0.0;
        for (int i=0;i<20;i++) {
            s+= fitness(generation[i]);
        }
        Double average = s / 100.0;
        return average;
    }
}
```

On the graph, we can see, how minimization of function changed after each step.



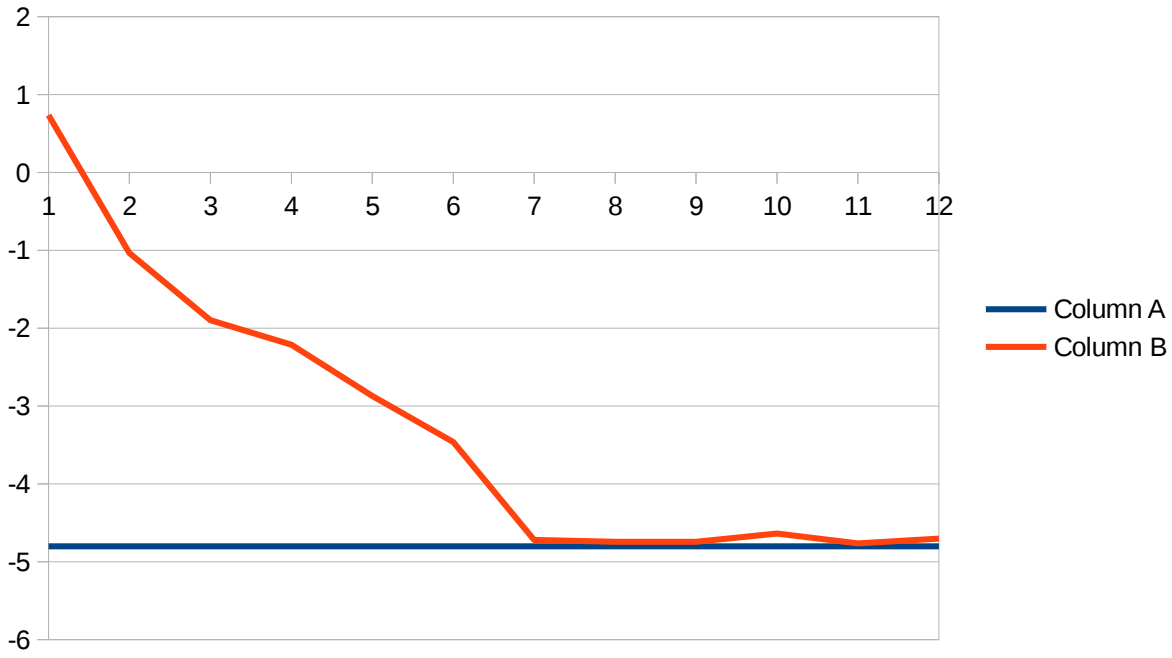
TASK2:

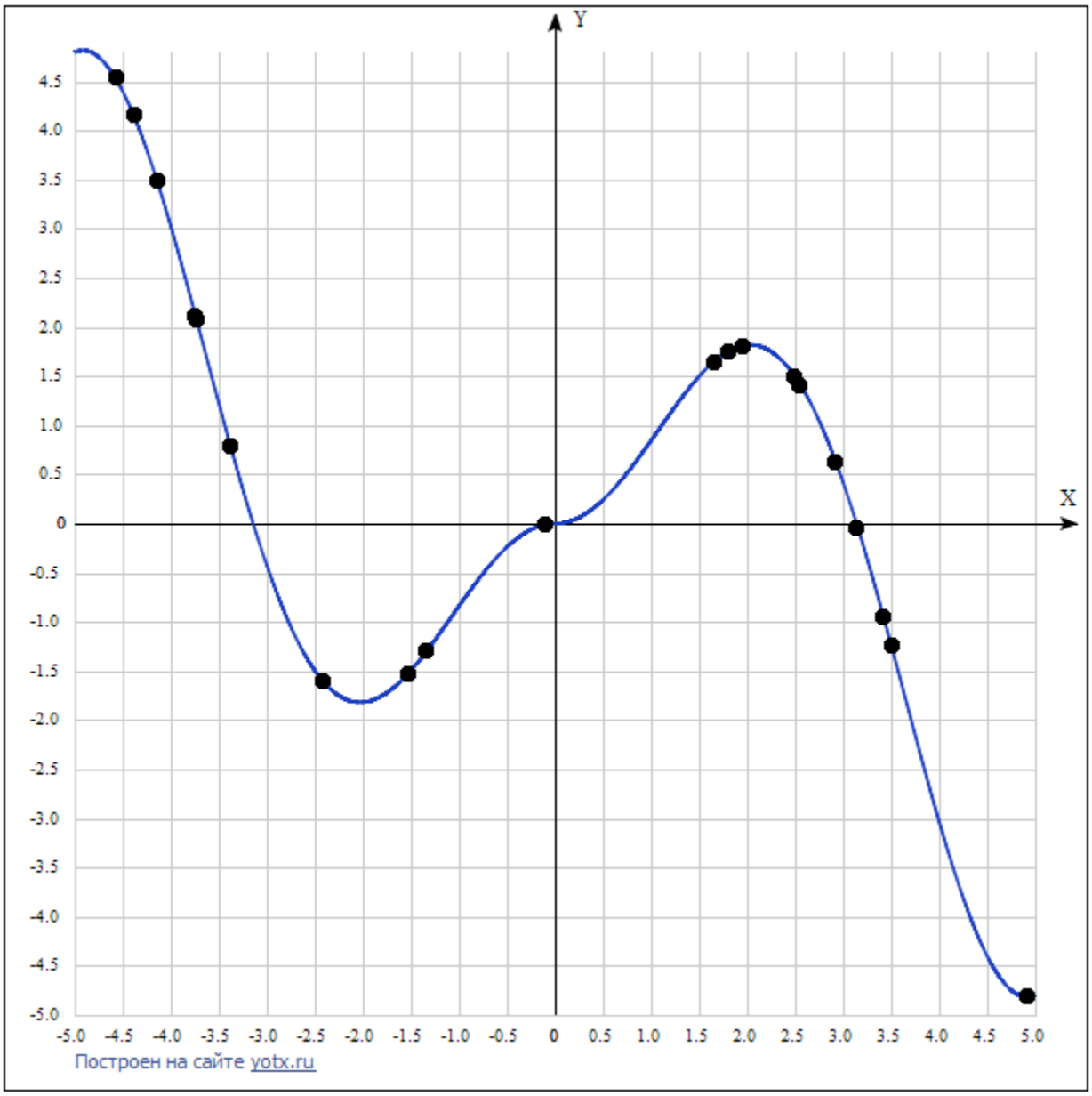
```
public class Main {
    public static Integer N = 11;
    public static Integer[][] generation = new Integer[20][N];
    public static Integer[][] good = new Integer[10][N];
    public static Integer[][] childes = new Integer[10][N];
    public static Integer childCount = 0;
    public static Integer checkCount = 0;
    public static Double prevMax = 0.0;
    public static Double prevAverage = 0.0;
    public static void main(String[] args) {
        createGenerate();
        while(true) {
            Arrays.sort(generation, new Comparator<Integer[]>() {
                @Override
                public int compare(Integer[] o1, Integer[] o2) {
                    Double int1 = fitness(o1);
                    Double int2 = fitness(o2);
                    return (int)(int1 - int2) * 100;
                }
            });
            Double max = fitness(generation[0]);
            Double average = average();
            if(prevMax.equals(max)) {
                checkCount ++;
            }
            else
                checkCount = 0;
            prevMax = max;
            System.out.printf("%.3f %.3f", max,average());
            for(int i=0;i<20;i++) {
                System.out.printf("%10.3f",calcX(generation[i]));
            }
            System.out.println();
            if(checkCount > 10) {
                return;
            }
            setGood();
            Random random = new Random();
            childCount = 0;
            for (int i = 0; i < 5; i++) {
                createChildes(good[random.nextInt(10)], good[random.nextInt(10)]);
            }
            createNewGeneration();
        }
    }
    public static void createGenerate() {
        Random random = new Random();
        for(int i=0;i<20;i++) {
            for (int j = 0; j < N; j++) {
               generation[i][j] = random.nextInt(2);
            }
        }
    }
    public static Double fitness(Integer[] obj) {
        Double x = calcX(obj);
        return x * Math.sin(Math.abs(x));
    }
```

```
    public static Double calcX(Integer[] obj) {
        String str = "";
        for (int i = 1; i < obj.length; i++) {
            str += obj[i].toString();
        }
        if(obj[0] == 1)
            return Integer.parseInt(str,2)/ 1023.0 * 5;
        else
            return -Integer.parseInt(str,2)/ 1023.0 * 5;
    }
    public static void setGood() {
        for(int i =0; i< 10; i++) {
            good[i] = generation[i];
        }
    }
    public static void createChildes(Integer[] parent1, Integer[] parent2) {
        Random random = new Random();
        Integer delimiter = random.nextInt(N);
        Integer[] child1 = new Integer[N];
        Integer[] child2 = new Integer[N];
        for(int i = 0;i<N;i++) {
            if(i<=delimiter)
                child1[i] = parent1[i];
            else
                child1[i] = parent2[i];
        }
        for(int i = 0;i<N;i++) {
            if(i<=delimiter)
                child2[i] = parent2[i];
            else
                child2[i] = parent1[i];
        }
        childes[childCount] = child1;
        childCount++;
        childes[childCount] = child2;
        childCount++;
    }
    public static void createNewGeneration() {
        for (int i=0;i<20;i++) {
            if(i<10)
                generation[i] = good[i];
            else
                generation[i] = childes[i-10];
        }
    }
    public static Double average() {
        Double s = 0.0;
        for (int i=0;i<20;i++) {
            s+= fitness(generation[i]);
        }
        Double average = s / 20;
        return average;
    }
}
```

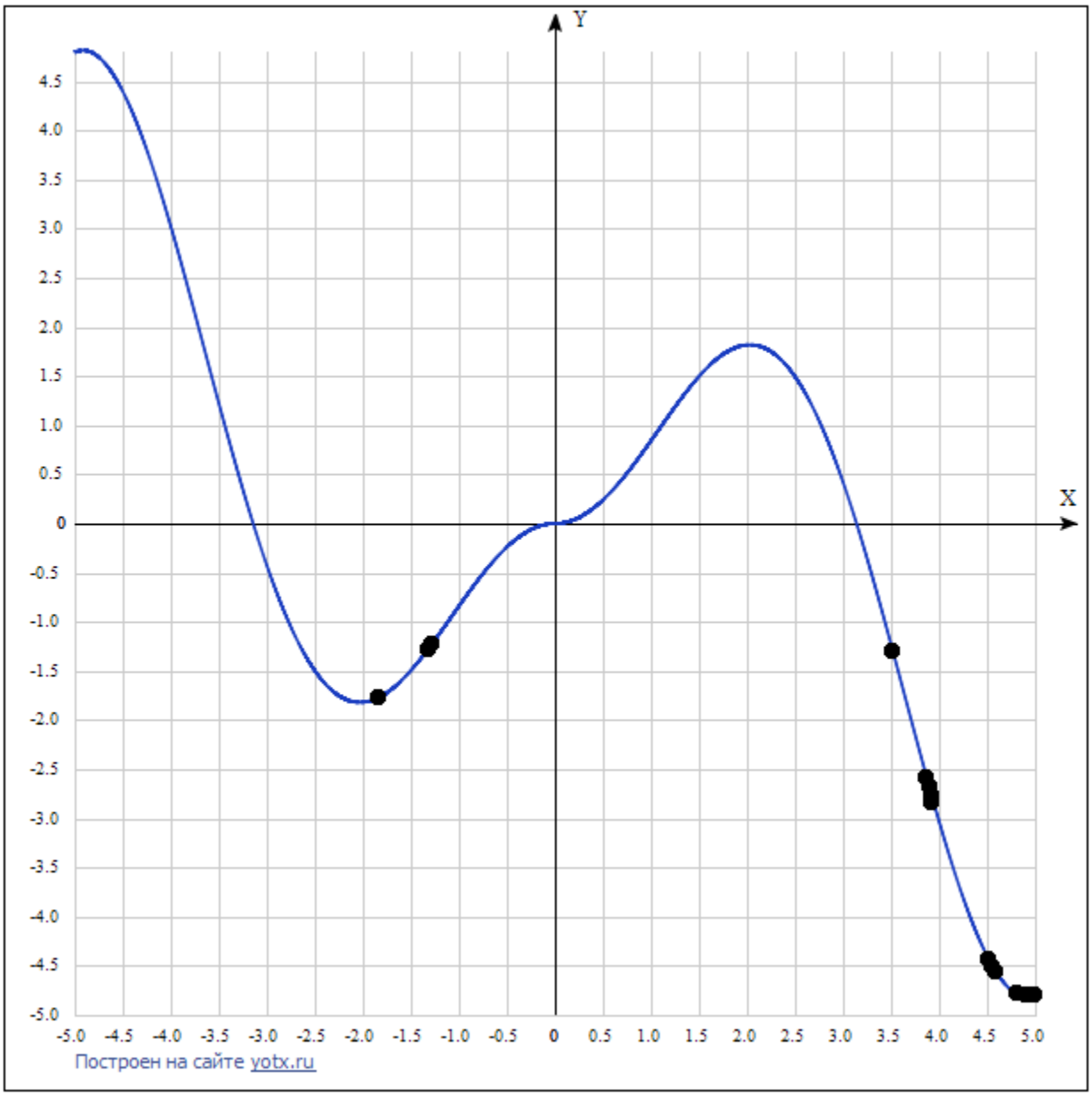
Red line – average values

Blue line – minimum values

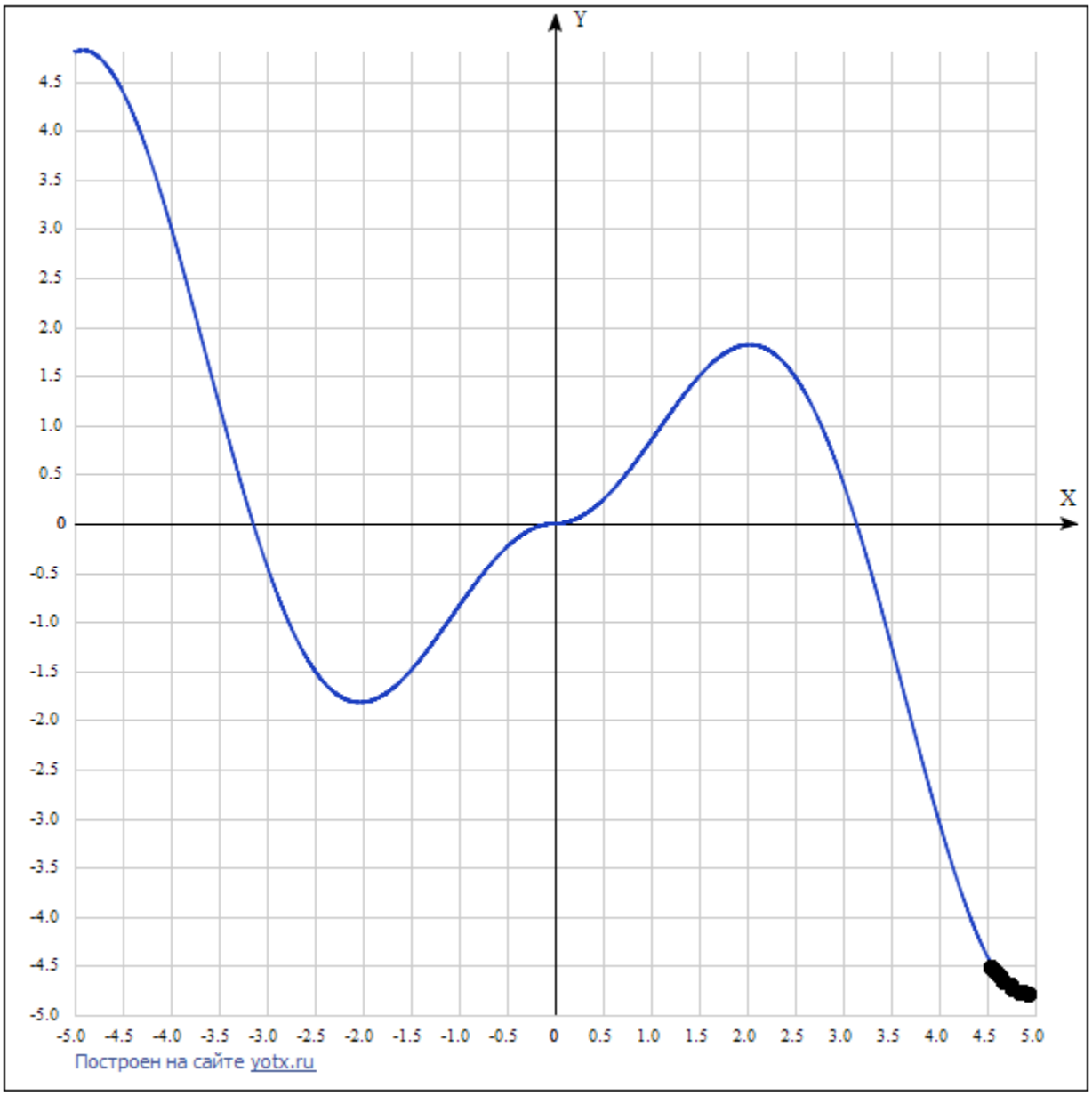




First graph, where we create first population for function.



Second graph, middle population(between first and last). We can see, that already some chromosomes stay near minimum of function.



Third graph(Last population from iterations). We can see, almost all chromosomes stay on minimum of function.