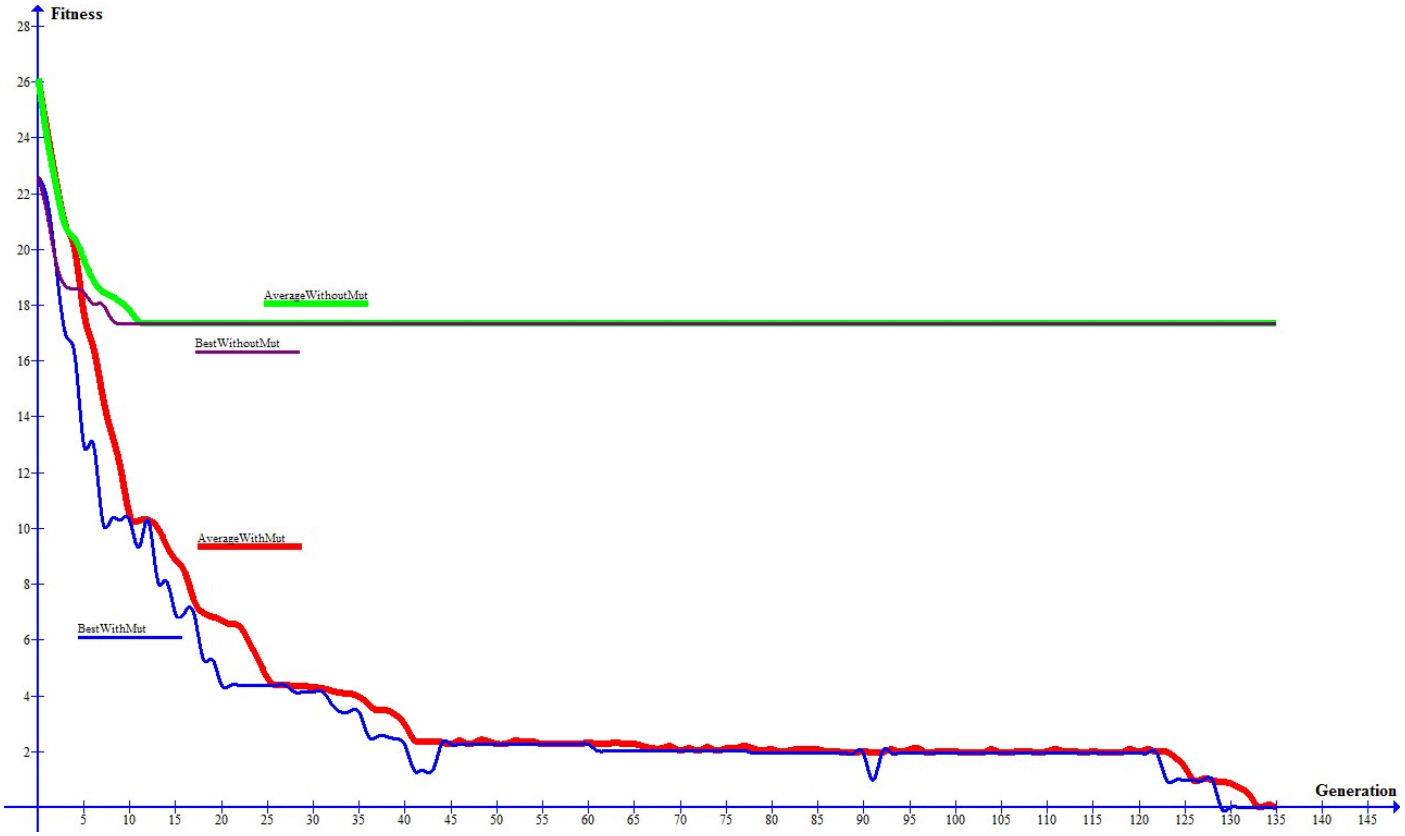
Task 1



Average vs generation graph.

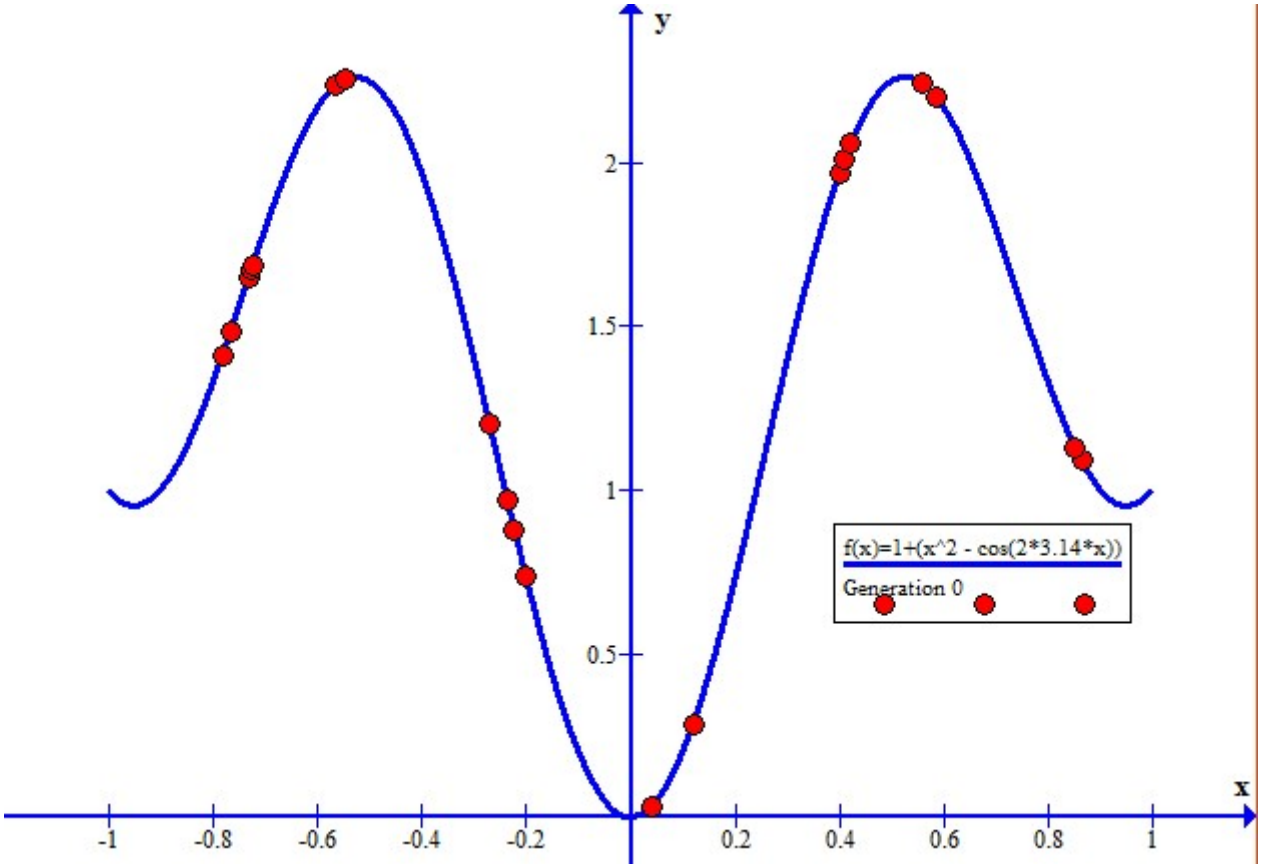
Source code:

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

public class MainTask1
{
   static Random random = new Random(System.currentTimeMillis());

   static class Chromosome
   {
      double[] genes;
      public Chromosome()
      {
         genes = new double[20];
         for(int i = 0; i < 20; i++)
            genes[i] = random.nextDouble() * 2 - 1;
      }
      double getFitness()
      {
         double y = 20;
         for(int i = 0; i < 20; i++)
            y += genes[i] * genes[i] - Math.cos(2 * Math.PI * genes[i]);
         return y;
      }
   }
```

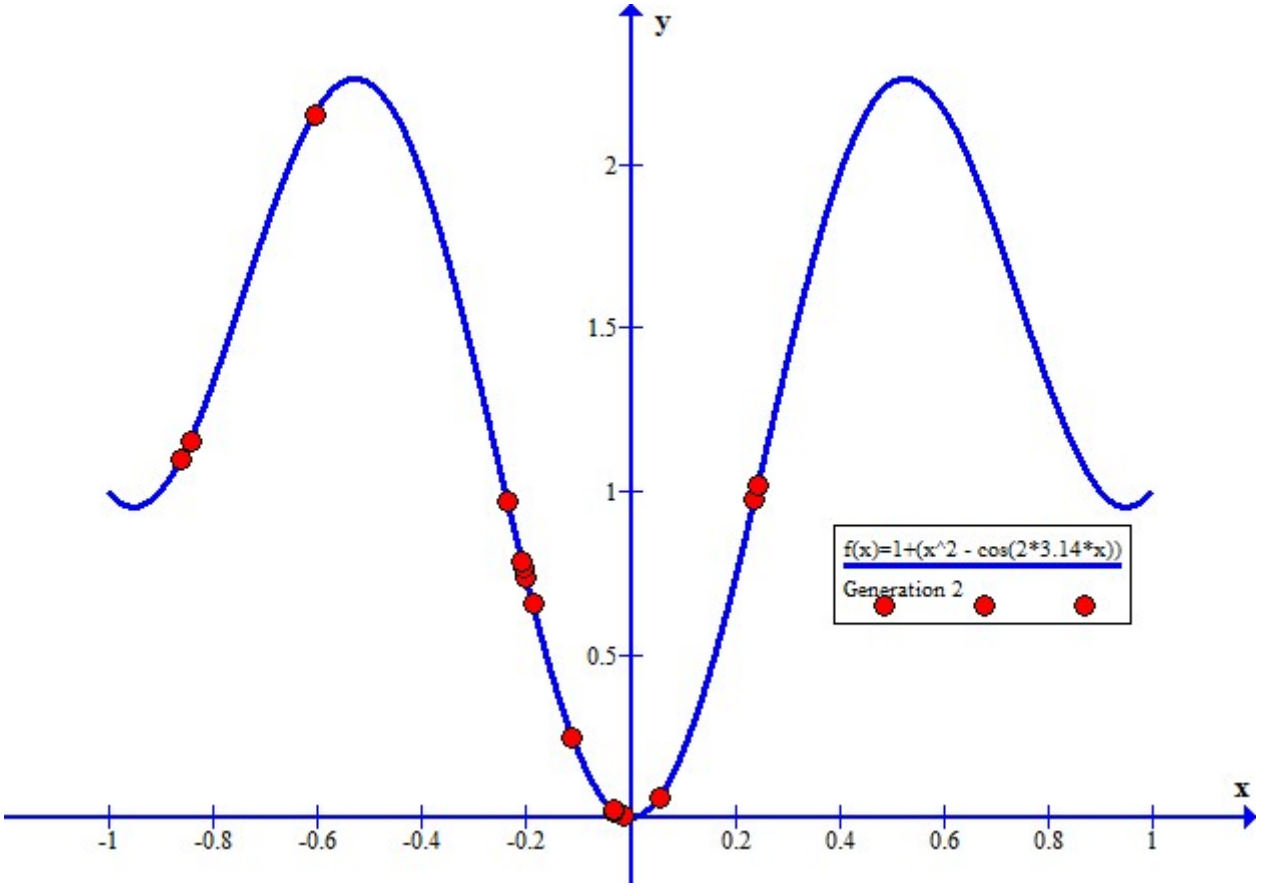
```
static class Generation
{
   Chromosome[] chromosomes;
   public Generation()
   {
      chromosomes = new Chromosome[20];
      for(int i = 0; i < 20; i++)
         chromosomes[i] = new Chromosome();
   }
   double getAverageFitness()
   {
      double counter = 0;
      for(Chromosome i : chromosomes)
         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);

   /*Mutation block*/
```

```
        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() <= 124)
            return false;
        for(int i = 0; i < 124; i++)
            if(fitnessHistory.get(fitnessHistory.size() - i - 1) != tmp)
                return false;
        return true;
    }


    public static void main(String[] args) throws InterruptedException
    {
        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(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(isLastGenerationUnchanged())
                break;

        }

        Thread.sleep(1000);
    }
}
```

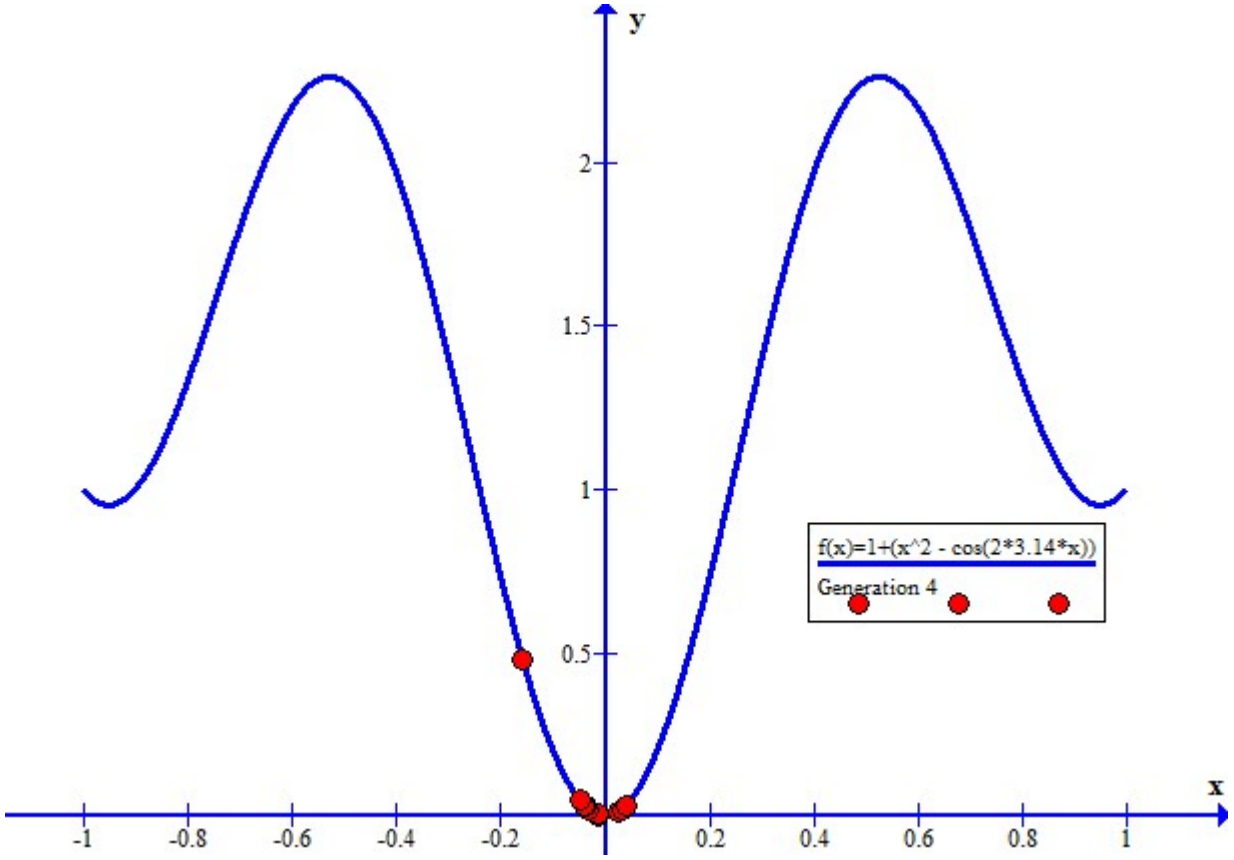
Task 2



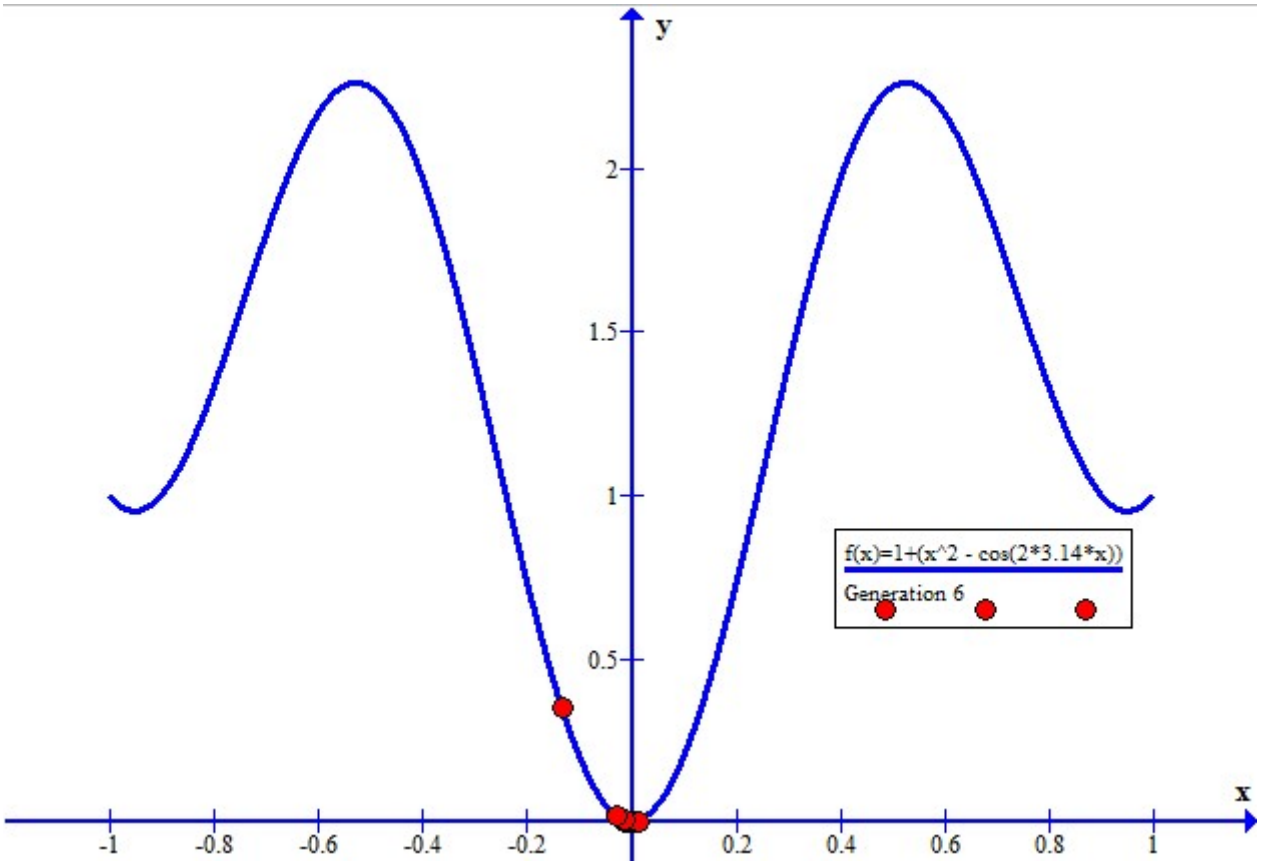
Generation 0 graph.



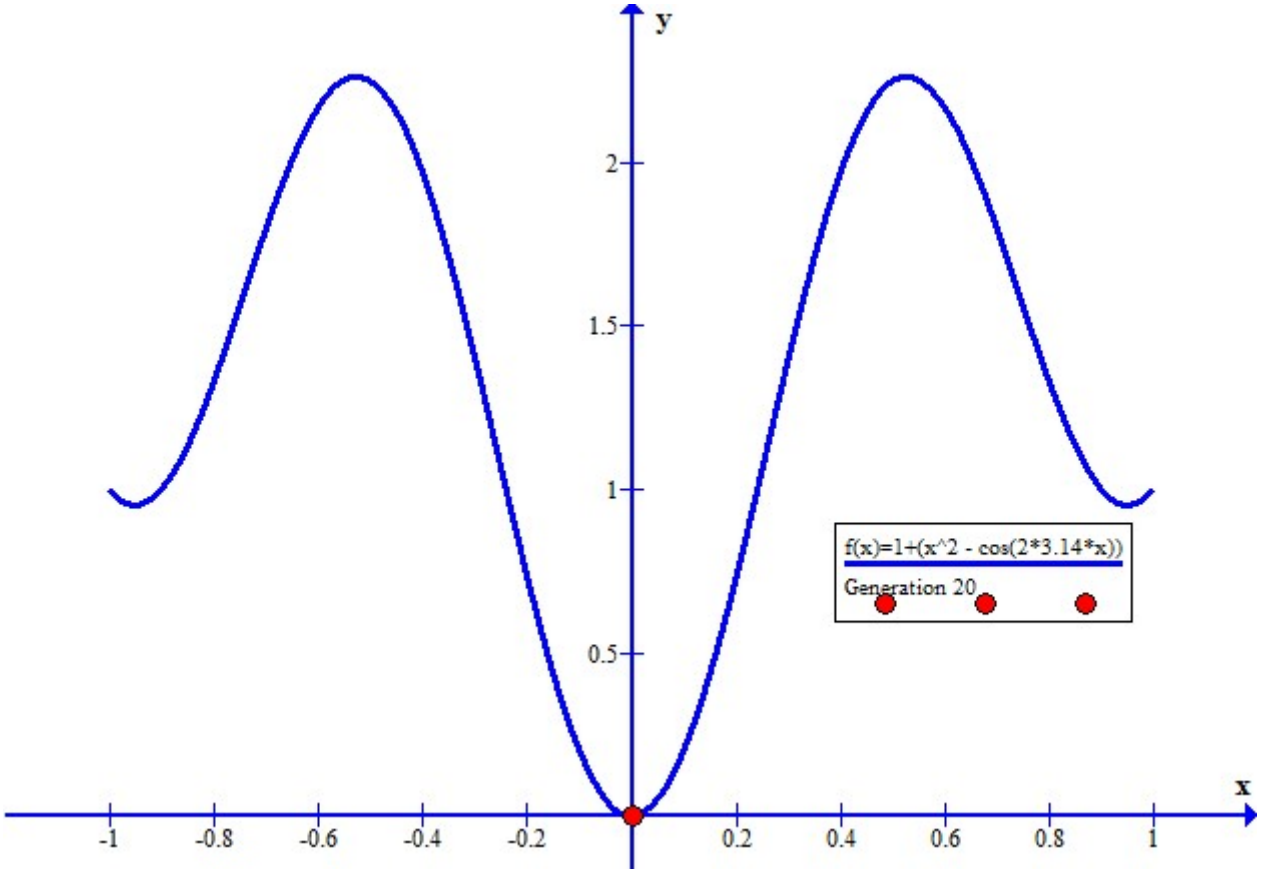
Generation 2 graph.



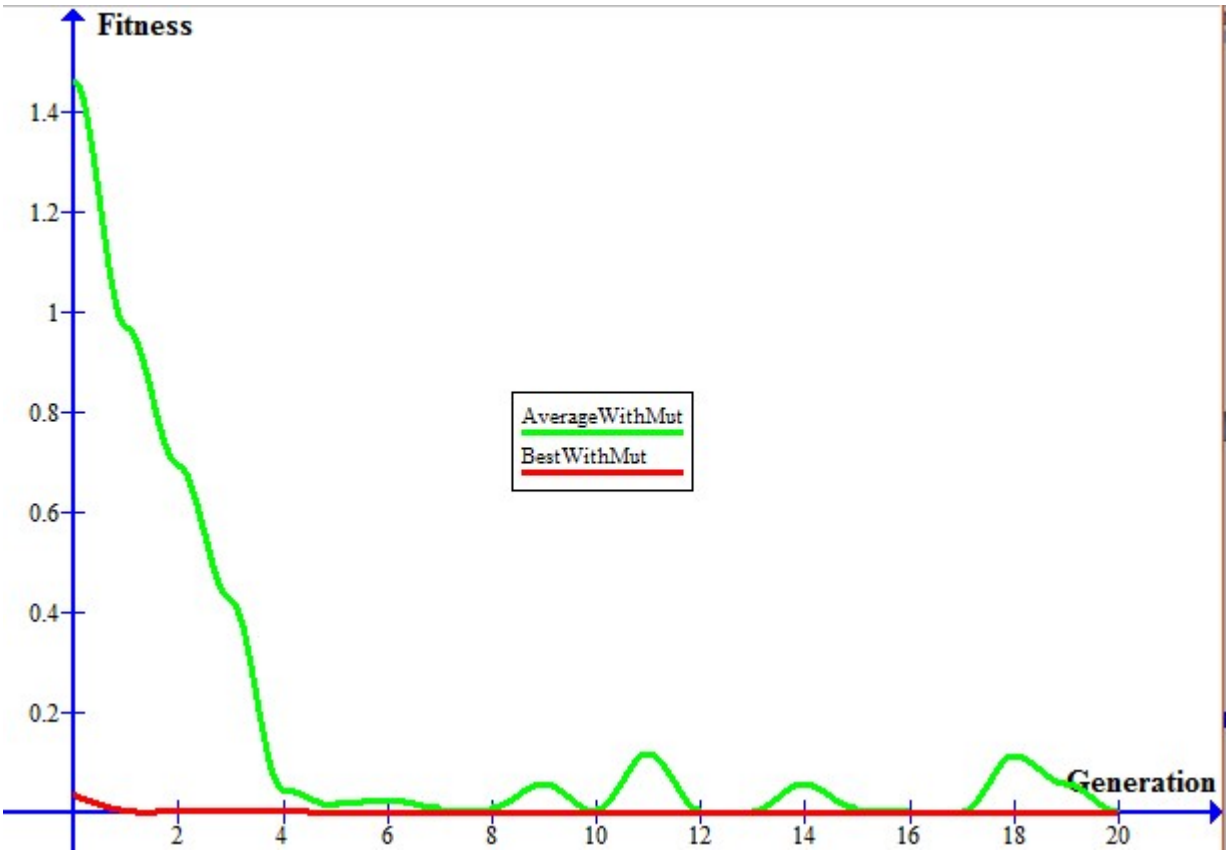
Generation 4 graph.



Generation 4 graph.



Generation 20 graph.



Average vs generation graph.

Source code:

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

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

   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(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(isLastGenerationUnchanged())
            break;

      }
   }
}
```