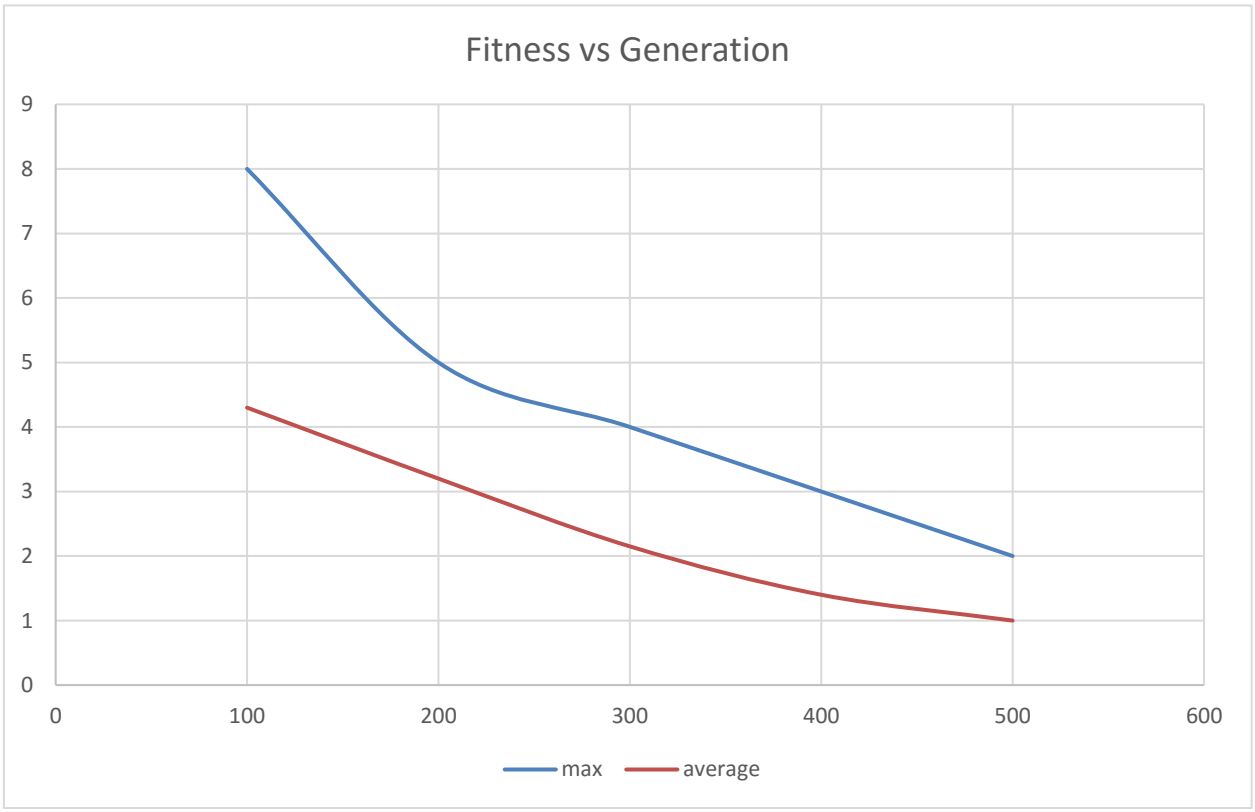
Practice (28/10/2016)
Student –Polina Rachkovskaya

Algorithm:

y1 = (x-2)^2 and y2 = (x-4)^2

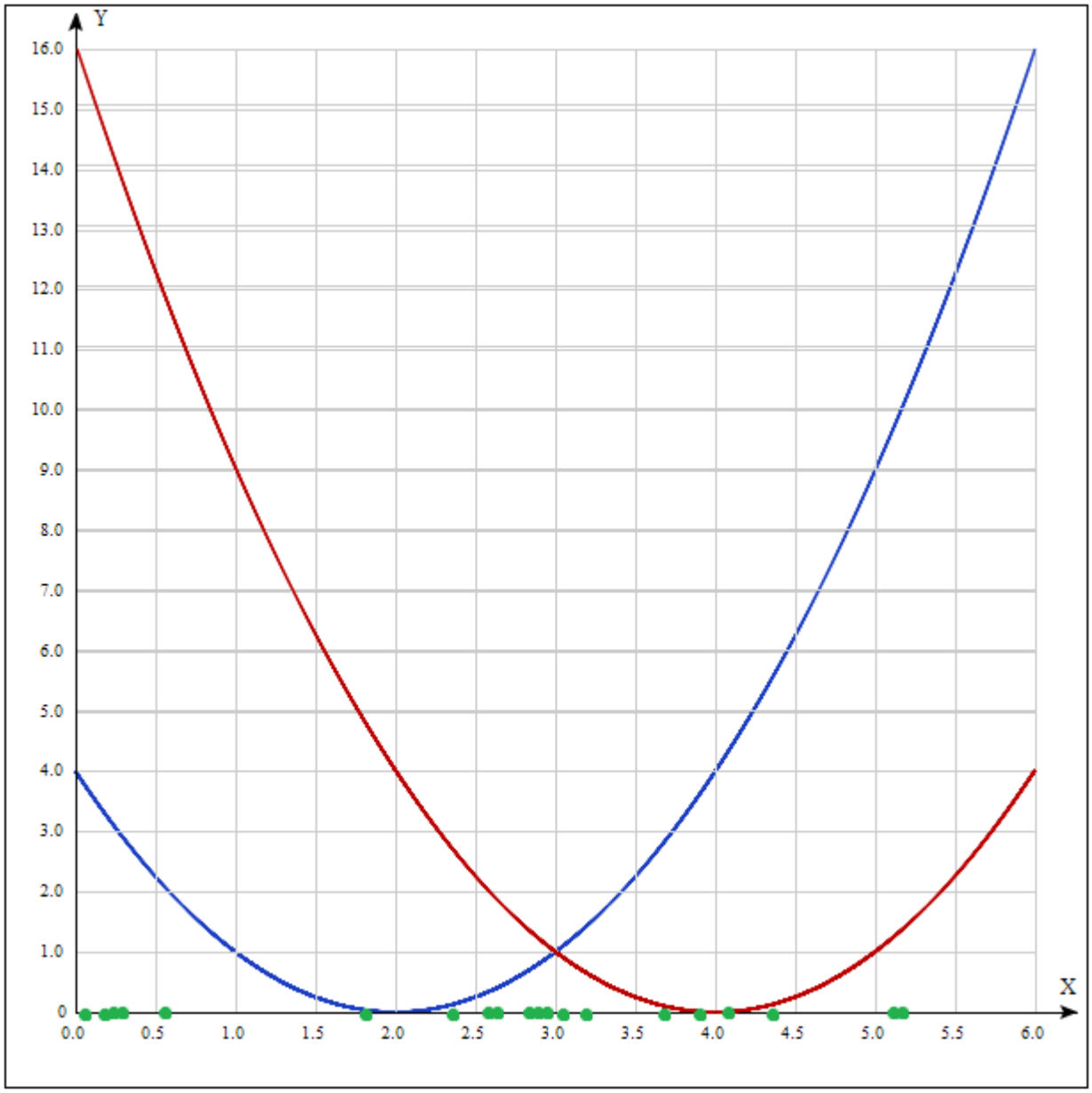
1. Create 20 10-bit binary chromosomes, assuming each chromosome represent x-cordinate ranges from 0 to 6 with (0000· · ·00) and (1111· · ·11) being corresponding to 0 and 6, respectively.
2. Calculate y 1 and y 2 for each of 20 x's represented by these 20 chromosomes.
3. Select individuals uniformly from population.
4. Perform crossover and mutation to create a child.
5. Calculate the rank of the new child.
6. Find the individual in the entire population that is most similar to the child. Replace that individual with the new child if the child's ranking is better, or if the child dominates it.
7. Update the ranking of the population if the child has been inserted.
8. Perform steps 2-6 according to the population size.

Results:



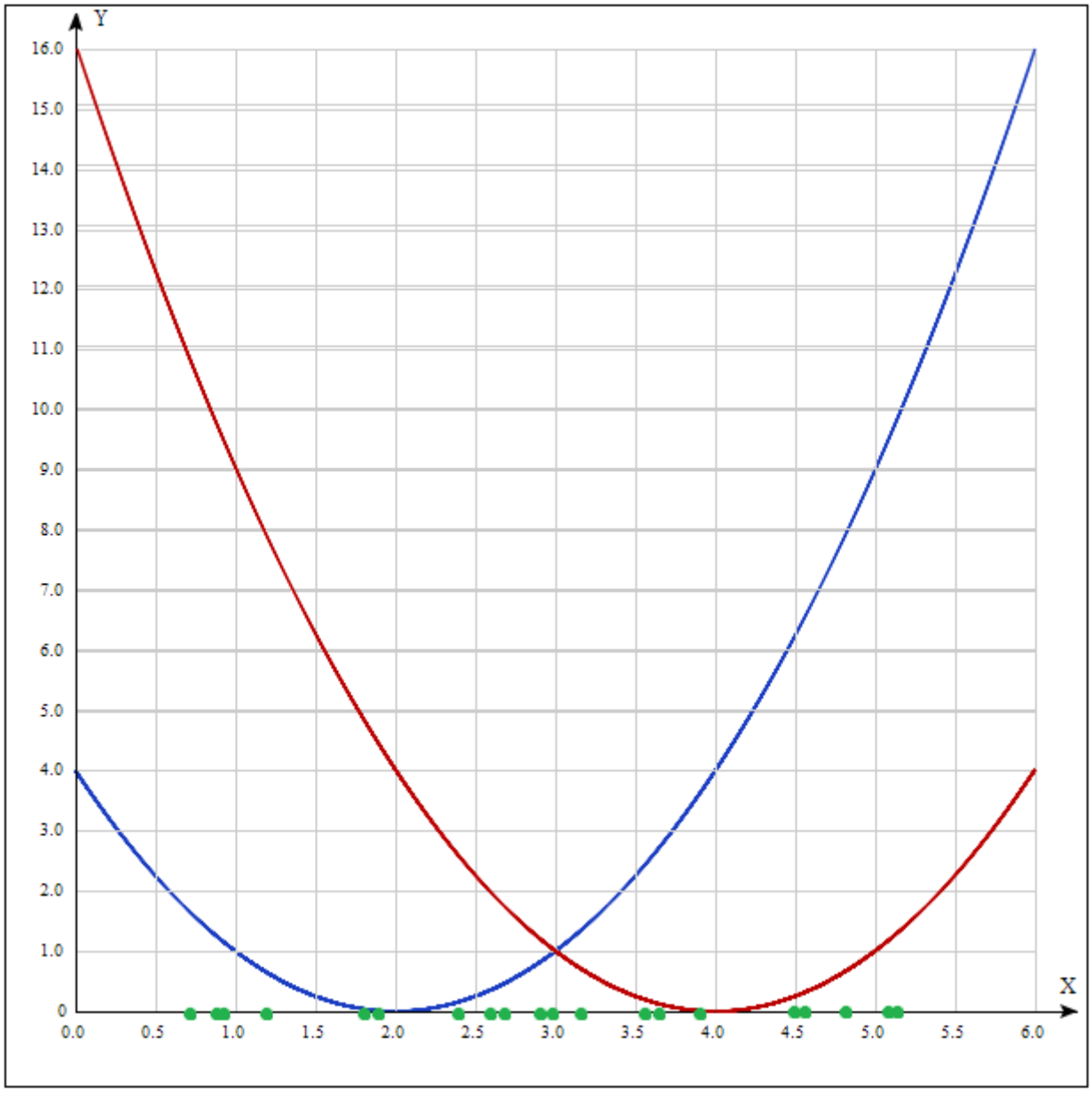
Iteration 1

№	Chromosome	X	Y1	Y2	Rank
1	0111110111	2,950147	0,902779	1,102192	8
2	1001111110	3,741935	3,034339	0,066597	0
3	1101111110	5,243402	10,51965	1,546048	1
4	1000000100	3,026393	1,053483	0,947911	8
5	1011101111	4,404692	5,782544	0,163776	3
6	1101111101	5,237537	10,48164	1,531497	2
7	0000110101	0,31085	2,853226	13,60982	3
8	0000000100	0,02346	3,906709	15,81287	0
9	0111101011	2,879765	0,773987	1,254926	8
10	0110100110	2,475073	0,225695	2,325401	5
11	0000011011	0,158358	3,391646	14,75822	1
12	1010001101	3,829912	3,348578	0,02893	6
13	0111000101	2,656891	0,431506	1,80394	5
14	0000101101	0,26393	3,01394	13,95822	2
15	0111000101	2,656891	0,431506	1,80394	5
16	0111101000	2,86217	0,743337	1,294657	8
17	1000100100	3,214076	1,473981	0,617676	8
18	1011000000	4,129032	4,532778	0,016649	4
19	0001011110	0,55132	2,098675	11,8934	4
20	0100101011	1,753666	0,060681	5,046018	5



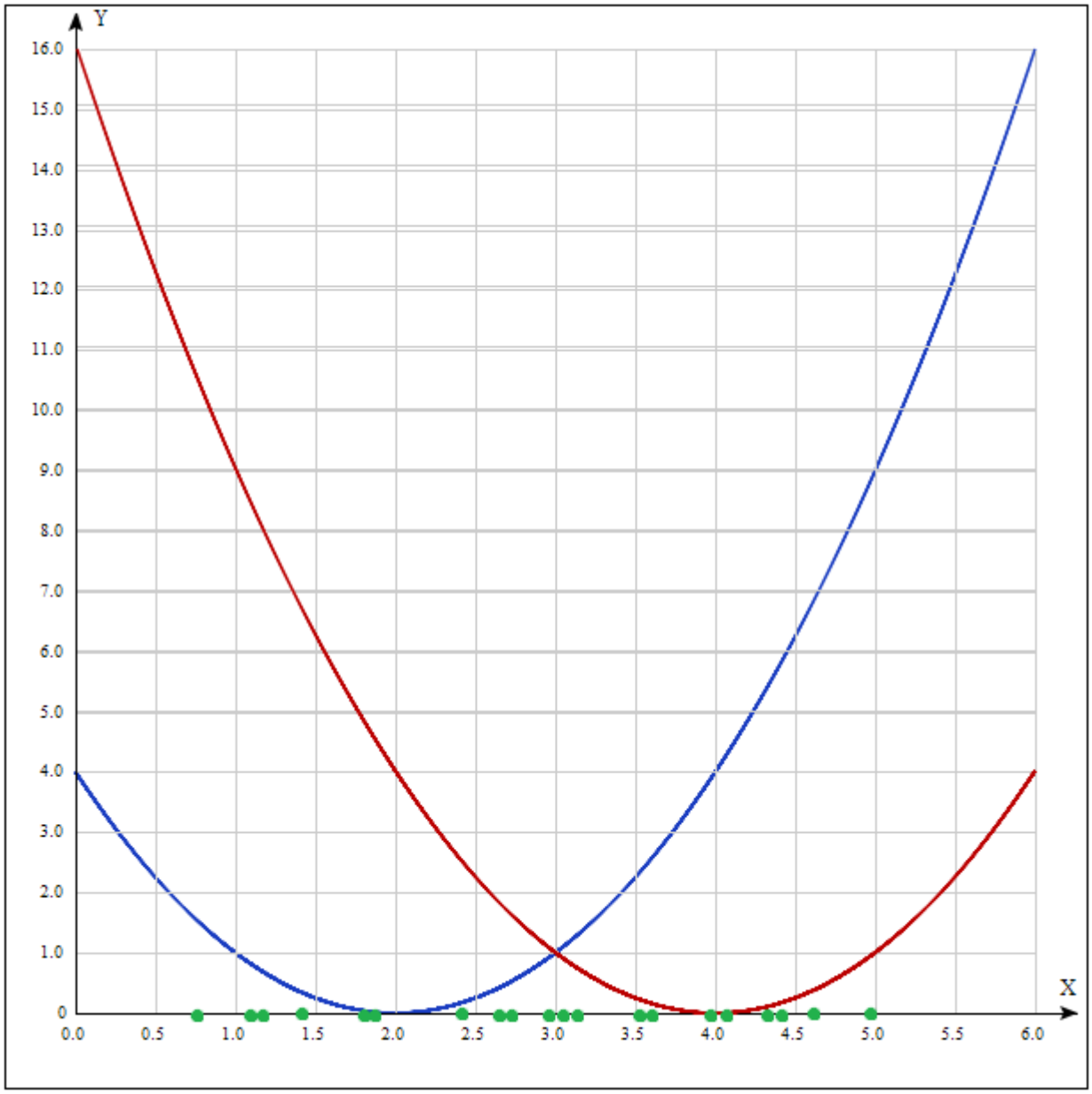
Iteration 125

№	Chromosome	X	Y1	Y2	Rank
1	1100001011	4.567784	6.593515	0.322378671	3
2	0111001000	2.673353	0.453404	1.759992263	5
3	0111000001	2.634562	0.402669	1.864420932	5
4	0111101000	2.864222	0.74688	1.289991665	5
5	1101111110	5.244221	10.52497	1.548085897	1
6	1110010101	5.375664	11.39511	1.892451441	0
7	0011100101	1.345692	0.428119	7.045350959	3
8	0100110000	1.783452	0.046893	4.913085036	4
9	1000100111	3.234525	1.524052	0.585951976	4
10	1001100000	3.567444	2.456881	0.187104693	5
11	1001110101	3.690034	2.856215	0.096078921	4
12	0100111111	1.873324	0.016047	4.522750809	4
13	0110100011	2.456225	0.208141	2.383241251	3
14	1010011100	3.918843	3.681958	0.006586459	5
15	1011111011	4.475355	6.127382	0.225962376	4
16	0001111001	0.712399	1.657917	10.80832099	0
17	1101000001	4.883211	8.312906	0.780061671	2
18	0010011000	0.891336	1.229137	9.663794974	0
19	0010011011	0.910013	1.188072	9.54801966	2
20	0111110000	2.910453	0.828925	1.187112665	5



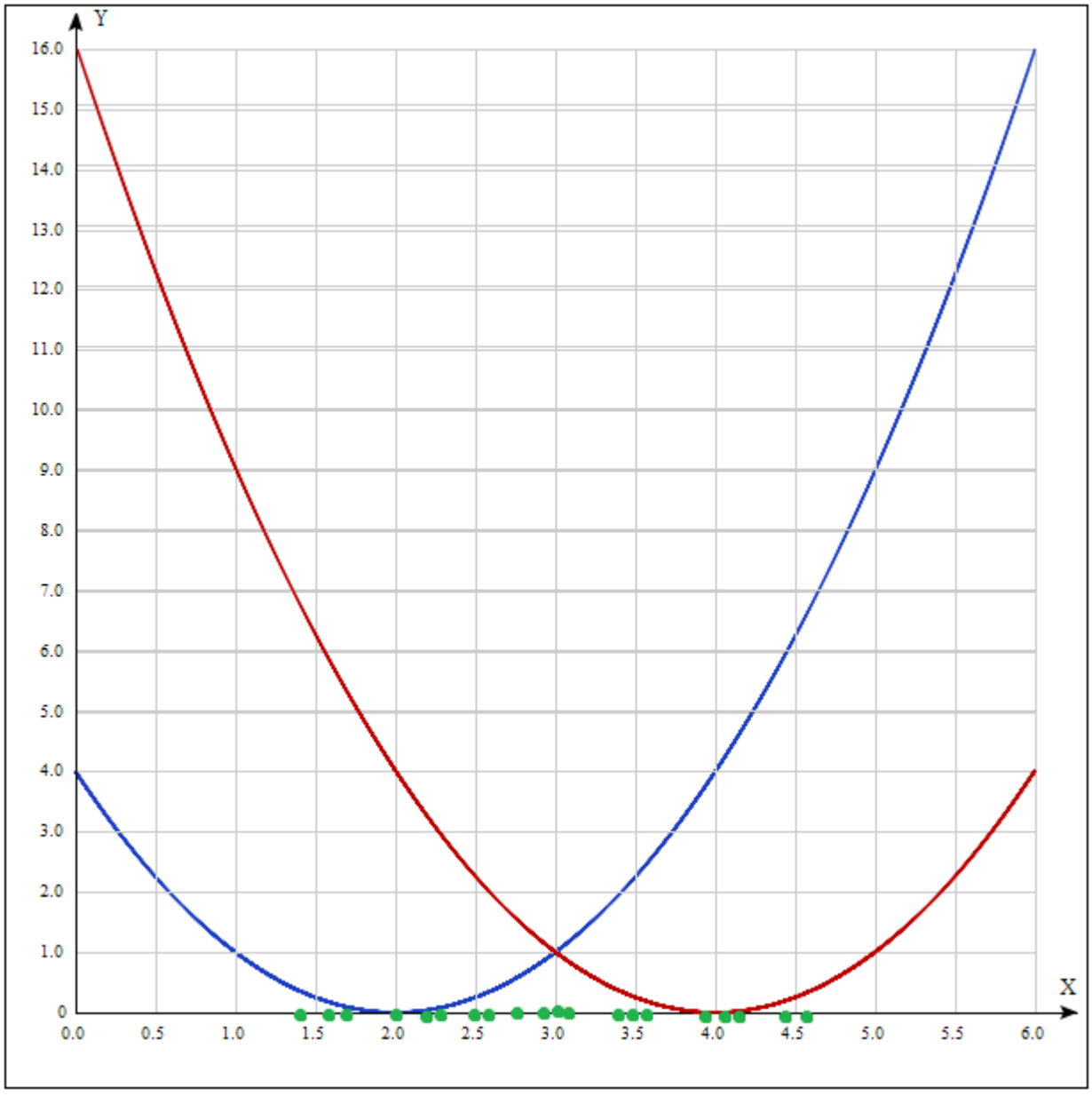
Iteration 250

№	Chromosome	X	Y1	Y2	Rank
1	1100110000	4.787668	7.771093	0.620420878	0
2	0100110001	1.786554	0.045559	4.899343195	4
3	1011010110	4.255532	5.087425	0.065296603	3
4	0110100011	2.456789	0.208656	2.381500191	4
5	0001111101	0.734578	1.601293	10.66298084	0
6	1101000111	4.922345	8.5401	0.850720299	0
7	0111110110	2.944333	0.891765	1.114432815	1
8	0011000011	1.145678	0.729866	8.14715408	1
9	1000101011	3.255432	1.57611	0.554381507	3
10	1011110100	4.435556	5.931933	0.189709029	3
11	0011010011	1.238635	0.579677	7.625136663	2
12	1011111011	4.476889	6.134979	0.227423118	1
13	1001010100	3.496784	2.240362	0.253226343	1
14	0011111101	1.486755	0.26342	6.31640043	3
15	1010011010	3.903332	3.622673	0.009344702	3
16	0111001011	2.689644	0.475609	1.717032847	3
17	1000011000	3.144432	1.309725	0.731996603	2
18	0111010010	2.733422	0.537908	1.60421983	3
19	0101000000	1.874432	0.015767	4.518039323	5
20	1001010111	3.514433	2.293507	0.235775311	1



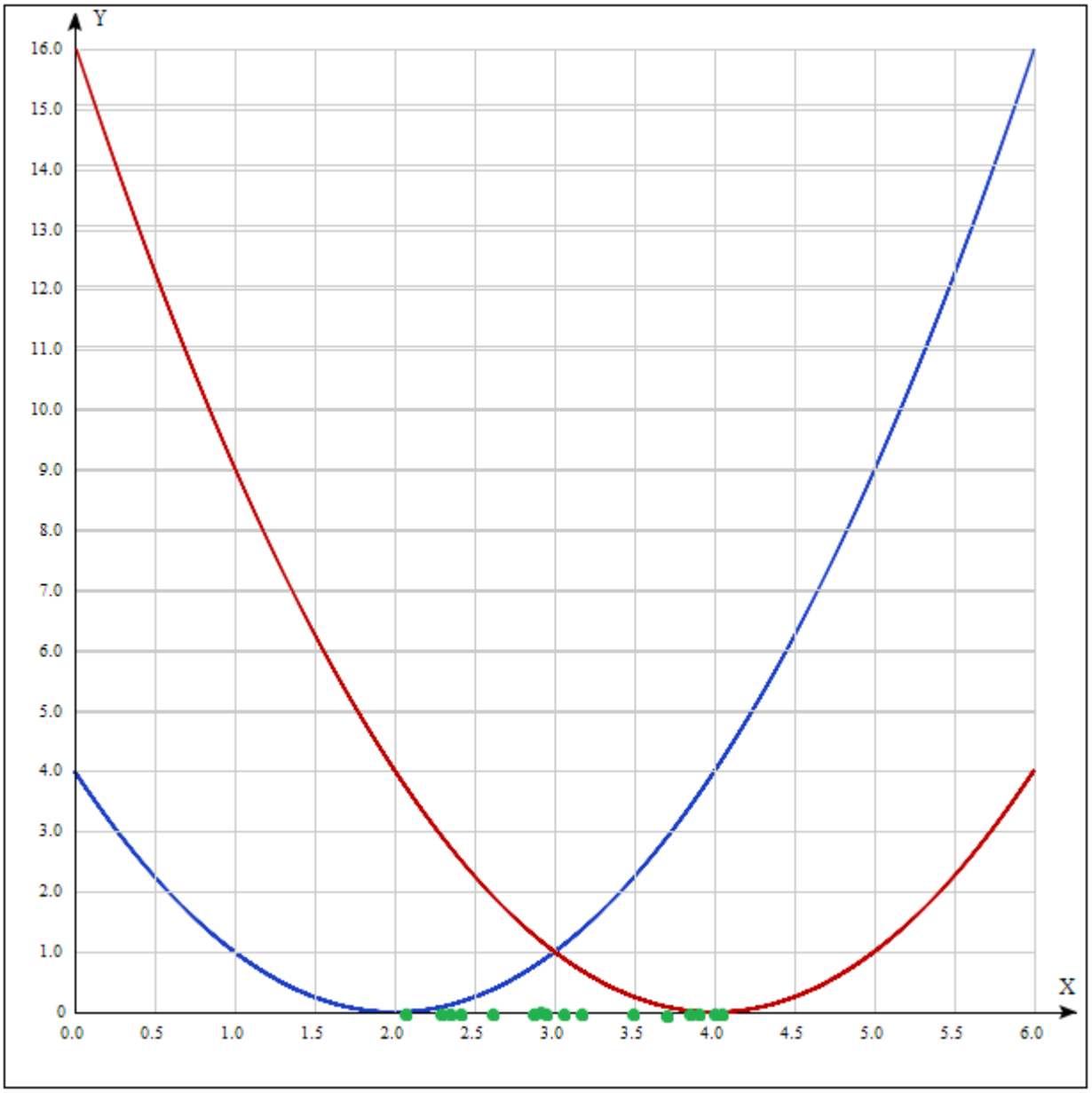
Iteration 375

№	Chromosome	X	Y1	Y2	Rank
1	1011111001	4.463433	6.068502	0.214770145	1
2	1000000101	3.032211	1.06546	0.936615549	1
3	1000100000	3.189224	1.414254	0.657357722	1
4	1001011001	3.524452	2.323954	0.2261459	1
5	0011110101	1.435678	0.318459	6.57574732	0
6	1100001110	4.588224	6.698903	0.346007474	0
7	0100001100	1.573889	0.181571	5.886014584	0
8	1011010110	4.255368	5.086685	0.065212815	2
9	0100011001	1.645632	0.125577	5.543048679	2
10	0101010101	2.000123	1.51E-08	3.999508015	0
11	0101101110	2.144326	0.02083	3.443525994	3
12	0101111111	2.245821	0.060428	3.077143964	3
13	1001100101	3.596622	2.549202	0.162713811	2
14	1001010011	3.489211	2.217749	0.260905403	1
15	0110101001	2.495523	0.245543	2.263451044	2
16	0110101110	2.522368	0.272868	2.183396327	2
17	0111010101	2.752219	0.565833	1.556957424	1
18	1010100010	3.954112	3.818554	0.002105709	3
19	0111111000	2.955312	0.912621	1.091373017	1
20	1010110011	4.051331	4.207959	0.002634872	3



Iteration 500

№	Chromosome	X	Y1	Y2	Rank
1	0111110111	2.950147	0.902779	1.102192104	0
2	1001111110	3.741935	3.034339	0.066597294	0
3	1000000100	3.026393	1.053483	0.947910665	0
4	0111101011	2.879765	0.773987	1.254925568	0
5	0110100110	2.475073	0.225695	2.325401398	0
6	0111000101	2.656891	0.431506	1.803940455	0
7	0111101000	2.86217	0.743337	1.294656909	0
8	1000100100	3.214076	1.473981	0.617676147	0
9	1010001101	3.829912	3.348578	0.02892992	0
10	1010101101	4.017595	4.070691	0.000309595	0
11	1010101010	4.005865	4.023495	0.000003439	2
12	0110100001	2.445748	0.198691	2.415699899	0
13	1010100101	3.970674	3.883558	0.000859986	0
14	0110100001	2.445748	0.198691	2.415699899	0
15	0111110111	2.950147	0.902779	1.102192104	0
16	1010101101	4.017595	4.070691	0.000309595	0
17	1010001101	3.829912	3.348578	0.02892992	0
18	0110100001	2.445748	0.198691	2.415699899	0
19	0101011010	2.005865	3.976541	0.0000341	2
20	1001011100	3.542522	2.379374	0.209286126	0



Code:

```
public static final int GENE_LENGTH = 10;
private int[] genes;private double x,y1,y2;
private int rank;
public Individual(boolean initialize) {
genes = new int[GENE_LENGTH];
if (initialize) {
generateIndividual();this.x = calculateX();
this.y1 = calculateY1();this.y2 = calculateY2();
this.rank = 0; }}
public Individual(int[] genes) {
this.genes = genes; this.x = calculateX();this.y1 = calculateY1();
this.y2 = calculateY2();this.rank = 0; }
public double getX() {return x; }
public double getY1() {return y1; }
public double getY2() {return y2; }
public int getRank() {return rank; }
public int updateRank() {return ++this.rank; }
public void resetRank() {this.rank = 0; }
public void generateIndividual() {
for (int i = 0; i < GENE_LENGTH; ++i) {
genes[i] = ThreadLocalRandom.current().nextInt(0, 2);} }
public double calculateX() {
return Integer.parseInt(Arrays.toString(genes).replaceAll("[,\\[\\] ]",
""), 2) / 170.5;     }
public double calculateY1() {return Math.pow(x - 2, 2); }
public double calculateY2() {return Math.pow(x - 4, 2); }
public int[] getGenesBeforeCutPoint(int cutPoint) {
int[] genes = new int[cutPoint];
System.arraycopy(this.genes, 0, genes, 0, cutPoint);
return genes; }
public int[] getGenesAfterCutPoint(int cutPoint) {
int[] genes = new int[GENE_LENGTH - cutPoint];
System.arraycopy(this.genes, cutPoint, genes, 0, genes.length);
return genes; }
public static final int POPULATION_SIZE = 20;
private Individual[] individuals;
public Population(boolean initialize) { individuals = new
Individual[POPULATION_SIZE];
if (initialize) {
for (int i = 0; i < POPULATION_SIZE; ++i) {
individuals[i] = new Individual(true); }}}
public Population(Individual[] individuals) {
this.individuals = new Individual[POPULATION_SIZE];
System.arraycopy(individuals, 0, this.individuals, 0,
individuals.length); }
public Individual getIndividual(int index) {return individuals[index];}
public void replaceIndividual(Individual individual, Individual child) {
for (int i = 0; i < individuals.length; ++i) {
if (individuals[i].equals(individual)) {
individuals[i] = child; break; }}}
public Individual[] getAllIndividuals() {return individuals; }
public void calculateRankForAllIndividuals() {resetRank();
for (int i = 0; i < POPULATION_SIZE; ++i) {
for (int j = 0; j < POPULATION_SIZE; ++j) {
 if (i != j && (individuals[i].getY1() < individuals[j].getY1() &&
individuals[i].getY2() < individuals[j].getY2())) {
individuals[i].updateRank();}}}}
private void resetRank() {
```

```
for (Individual individual: individuals) {
individual.resetRank();}}
private Population population;
public GeneticAlgorithm(Population population) {this.population =
population; }
public Population run() {Population nextGeneration = new
Population(population.getAllIndividuals());
nextGeneration.calculateRankForAllIndividuals();
for (int i = 0; i < Population.POPULATION_SIZE; ++i) {
Individual[] parents = chooseParents(nextGeneration);
int cutPoint = ThreadLocalRandom.current().nextInt(0,
Individual.GENE_LENGTH);
Individual[] children = crossover(parents, cutPoint);
Individual child = getBetterIndividualFromTwoChildren(children);
calculateIndividualRank(child);
Individual individual = findIndividualThatMostSimilarToChild(child);
if (childBetterThanParent(individual, child)) {
nextGeneration.replaceIndividual(individual, child);
nextGeneration.calculateRankForAllIndividuals();}}
population = nextGeneration; return population; }
private Individual[] chooseParents(Population fittestIndividuals) {
return new Individual[]
{fittestIndividuals.getIndividual(ThreadLocalRandom.current().nextInt(0,
Population.POPULATION_SIZE)),
fittestIndividuals.getIndividual(ThreadLocalRandom.current().nextInt(0,
Population.POPULATION_SIZE))}; }
private Individual[] crossover(Individual[] parents, int curPoint) {
Individual[] descendants = new Individual[2];
int[] firstDescendantGenes =
concat(parents[0].getGenesBeforeCutPoint(curPoint),
parents[1].getGenesAfterCutPoint(curPoint));
int[] secondIndividualGenes =
concat(parents[0].getGenesAfterCutPoint(curPoint),
parents[1].getGenesBeforeCutPoint(curPoint));
descendants[0] = new Individual(firstDescendantGenes);
descendants[1] = new Individual(secondIndividualGenes);
return descendants;}
private Individual getBetterIndividualFromTwoChildren(Individual[]
children) {return calculateIndividualRank(children[0]) >
calculateIndividualRank(children[1]) ? children[0] : children[1]; }
private Individual findIndividualThatMostSimilarToChild(Individual
child) {Individual individual = population.getIndividual(0);
double minX = Math.abs(population.getIndividual(0).getX() -
child.getX());for (int i = 1; i < Population.POPULATION_SIZE; ++i) {
double tempMinX = Math.abs(population.getIndividual(i).getX() -
child.getX());if (tempMinX < minX) {minX = tempMinX; individual =
population.getIndividual(i); }}return individual; }
private boolean childBetterThanParent(Individual individual, Individual
child) {return child.getRank() > individual.getRank() || (child.getY1()
< individual.getY1() && child.getY2() < individual.getY2());}
private int calculateIndividualRank(Individual individual) {         for
(int i = 0; i < Population.POPULATION_SIZE; ++i) {              if
(individual.getY1() < population.getIndividual(i).getY1() &&
individual.getY2() < population.getIndividual(i).getY2()) {
individual.updateRank();}}return individual.getRank();}
private int[] concat(int[] genes1, int[] genes2) {int[] genes = new
int[Individual.GENE_LENGTH];
System.arraycopy(genes1, 0, genes, 0, genes1.length);
System.arraycopy(genes2, 0, genes, genes1.length, genes2.length);
return genes; }}
```