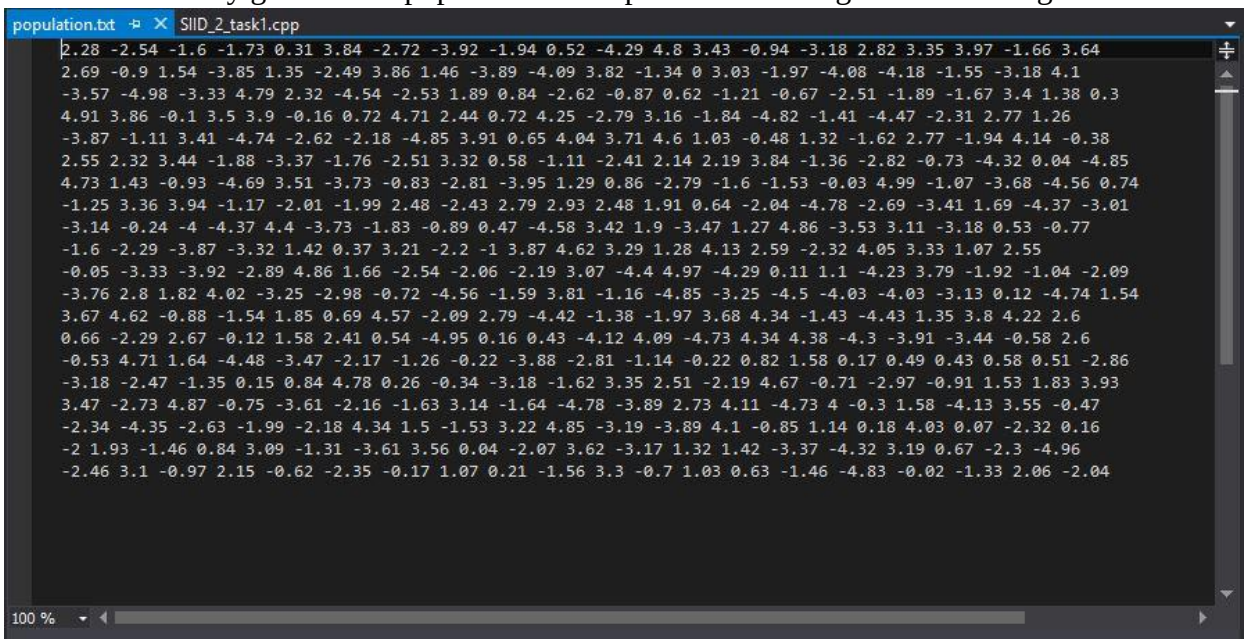
Task 1.

20-Dimensional Schwefel's function.

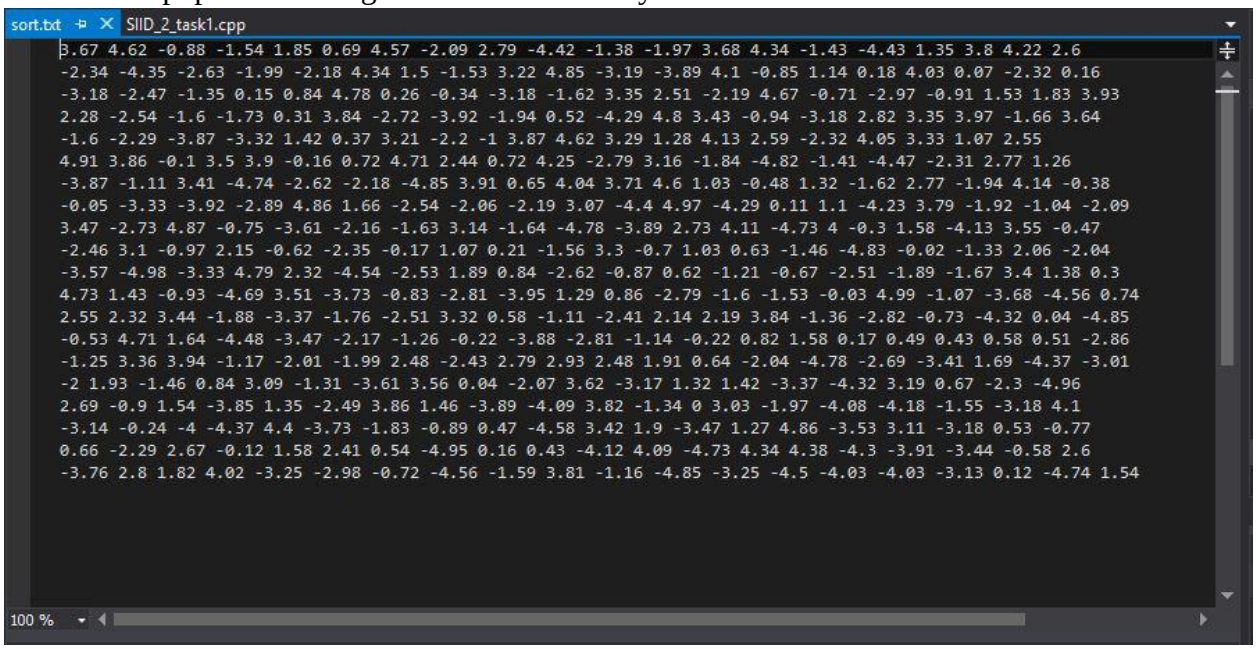
$y = x1 \sin(|x1|) + x2 \sin(|x2|) + \cdots + x20 \sin(|x20|);$

In this task, we must represent each xi by a chromosome with 20 genes.

1. I randomly generated a population of 20 parents from 20 genes in the range of -5 to 5.



2. Sort population using the fitness function y.



3. Ten smaller parents from the population by crossing and get 10 children.
4. We get a new population with the selected 10 smaller parents and 10 children.
5. We execute this algorithm twenty iterations.

Source code:

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <math.h>
#include <time.h>

using namespace std;

typedef vector<double> Individual;
typedef vector<Individual> Population;

void makePopulation();
void getPopulation();
int ChromosomeSumm(const Individual& population);
bool ChromosomeSort(const Individual& population1, const Individual& population2);
void print();
void crossing();

Population population;

int main()
{
	srand(time(NULL));

	makePopulation(); //Create a population of 20 chromosomes at random;
	getPopulation();

	sort(population.begin(), population.end(), ChromosomeSort);//sort using a fitness
function y;

	print();

	ofstream file("DataForGraph.txt", ios::out);

	file << "№ | Min y | Average y";
	file << '\n';

	for (int i = 0; i < 20; i++)
	{
		crossing();//get a new population;

		sort(population.begin(), population.end(), ChromosomeSort); //sort using a
fitness function y;

		double Min = 0;
		double Average = 0;

		for (int k = 0; k < population[0].size(); k++) {
			Min +=  population[0][k] * sin(abs(population[0][k]));
		}

		for (int j = 0; j < population.size(); j++) {
			for (int k = 0; k < population[j].size(); k++) {
				Average += population[j][k] * sin(abs(population[j][k]));
			}
		}
		Average = Average / 20;

		file << i + 1 << " | " << Min << " | " << Average;
		file << '\n';
	}
```

```
	file.close();

	system("Pause");
	return 0;
}

void makePopulation() {
	ofstream file("population.txt", ios::out);
	for (int i = 0; i < 20; i++) {
		for (int j = 0; j < 20; j++) {
			file << ((rand() % 1000) - 500)/100.0;
			if (j == 20 - 1) file << '\n';
			else file << ' ';
		}
	}
	file.close();
}

void getPopulation() {
	ifstream file("population.txt");
	for (int i = 0; i < 20; i++) {
		Individual row;
		for (int j = 0; j < 20; j++) {
			double value;
			file >> value;
			row.push_back(value);
		}
		population.push_back(row);
	}
	file.close();
}

int ChromosomeSumm(const Individual& population) {
	double summ = 0;
	for (int i = 0; i < population.size(); i++) {
		summ += population[i] * sin(abs(population[i]));
	}
	return summ;
}

bool ChromosomeSort(const Individual& population1, const Individual& population2) {
	return ChromosomeSumm(population1) < ChromosomeSumm(population2);
}

void print(){
	ofstream file("sort.txt", ios::out);
	for (int i = 0; i < population.size(); i++) {
		int tmp = 0;
		for (int j = 0; j < population[i].size(); j++) {
			file << population[i][j];
			if (j == 20 - 1) file << '\n';
			else file << ' ';
		}
	}
	file.close();
}

void crossing() {
	int CrossingPoint;
	int FirstParent, SecondParent;
	Population resultPopulation;

	for (int i = 0; i < 10; i++){
		CrossingPoint = rand() % 20;
```

```
			FirstParent = rand() % 10;
			SecondParent = rand() % 10;
			if (FirstParent == SecondParent){
				while (FirstParent != SecondParent)
				{
					SecondParent = rand() % 10;
				}
			}

			Individual Child1, Child2;

			for (int j = 0; j < population[i].size(); j++)
			{
				if (j < CrossingPoint)
				{
					Child1.push_back(population[FirstParent][j]);
					Child2.push_back(population[SecondParent][j]);
				}
				else
				{
					Child1.push_back(population[SecondParent][j]);
					Child2.push_back(population[FirstParent][j]);
				}
			}
			resultPopulation.push_back(Child1);
			resultPopulation.push_back(Child2);
		}
		population = resultPopulation;
}
```

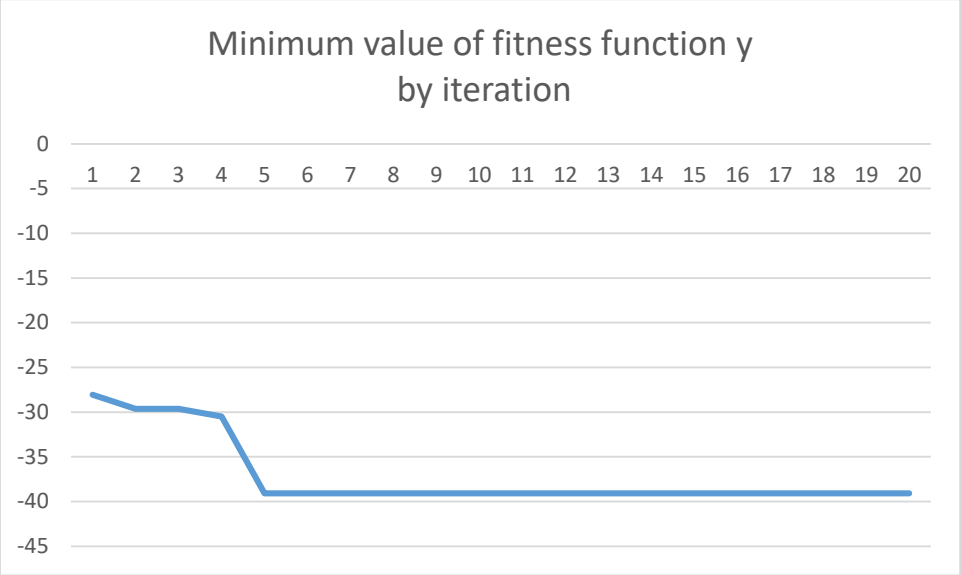
Data for graph:

DataForGraph.txt | SIID_2_task1.cpp

№	Min y	Average y
1	-28.0611	-9.78998
2	-29.6274	-16.5172
3	-29.6274	-19.8377
4	-30.5058	-22.1754
5	-39.0928	-27.1521
6	-39.0928	-31.4315
7	-39.0928	-35.2608
8	-39.0928	-38.8573
9	-39.0928	-39.0928
10	-39.0928	-39.0928
11	-39.0928	-39.0928
12	-39.0928	-39.0928
13	-39.0928	-39.0928
14	-39.0928	-39.0928
15	-39.0928	-39.0928
16	-39.0928	-39.0928
17	-39.0928	-39.0928
18	-39.0928	-39.0928
19	-39.0928	-39.0928
20	-39.0928	-39.0928

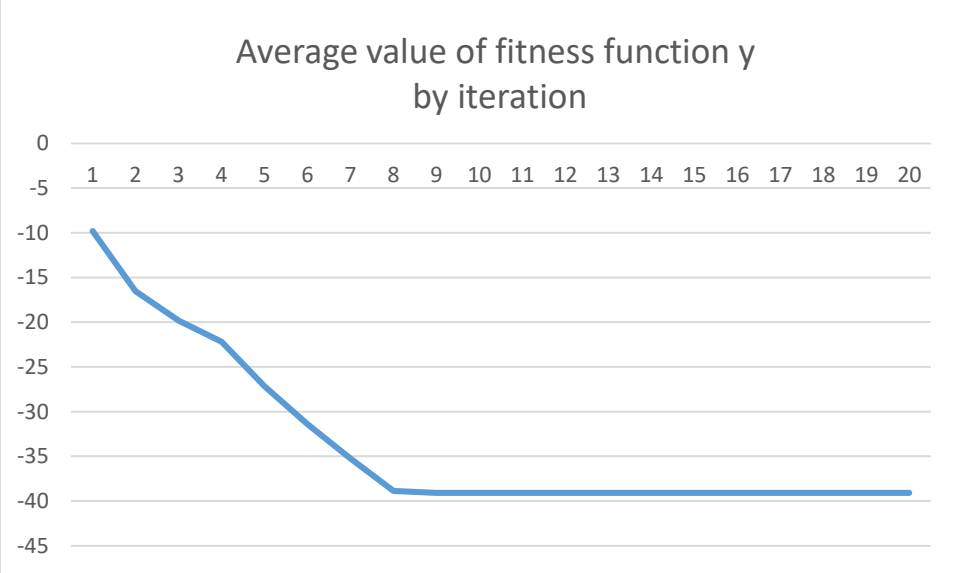
100 %

Graph 1. Minimum value of fitness function y by iteration.



In x line we have current iteration, in y line – minimum value of function.

Graph 2. Average value of fitness function y by iteration.



In x line we have current iteration, in y line – average value of function.

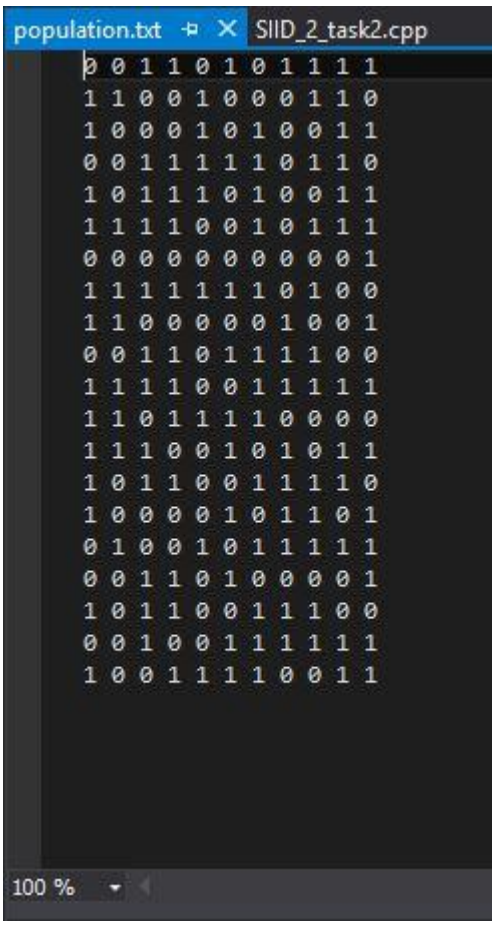
Task 2.

2-D version of Schwefel's function.

$y = x\sin(|x|);$

In this task, we must represent value of x by a 10-bit binary chromosome.

1. I randomly generated a population of 20 parents from 11 genes consisting of 0 and 1.



2. Sort population using the fitness function y:
   the first bit determines the sign of the number(if 1 is a plus, else 0). Remaining ten bits which number is represented in binary form, which we translate into decimal form and divide by 1023. Since our range is from -5 to 5, we multiply our number on 5. Then we sort each parent with respect to the numbers obtained, the less the better.
3. We get a new population with the selected 10 smaller parents and 10 children.
4. We execute this algorithm twenty five iterations.

Source code:

```
#include "stdafx.h"
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <math.h>
#include <time.h>

using namespace std;

typedef vector<double> Individual;
typedef vector<Individual> Population;

void makePopulation();
void getPopulation();
int ChromosomeSumm(const Individual& population);
bool ChromosomeSort(const Individual& population1, const Individual& population2);
void print();
void crossing();
```

```
double x, y;

Population population;

int main()
{
	srand(time(NULL));

	makePopulation();
	getPopulation();

	sort(population.begin(), population.end(), ChromosomeSort);

	ofstream file("DataForGraph.txt", ios::out);

	file << "X | Y";
	file << '\n';

	for (int i = 0; i < 25; i++)
	{

		file << "Iteration N " << i + 1;
		file << '\n';

		crossing();

		sort(population.begin(), population.end(), ChromosomeSort);

		for (int j = 0; j < population.size(); j++) {
			double summ = 0;
			summ = population[j][1] * 1 + population[j][2] * 2 + population[j][3]
* 4 + population[j][4] * 8 + population[j][5] * 16 + population[j][6] * 32 +
population[j][7] * 64 + population[j][8] * 128 + population[j][9] * 256 +
population[j][10] * 512;
			if (population[j][0] == 0){
				summ = summ * -1;
			}
			summ = summ / 1023 * 5;
			x = summ;
			summ = summ * sin(abs(summ));
			y = summ;
			file << x << " | " << y;
			file << '\n';
		}

		double Min = 0;
		double Average = 0;
		double AverageSumm = 0;

		for (int j = 0; j < population.size(); j++) {
			Min = population[0][1] * 1 + population[0][2] * 2 + population[0][3]
* 4 + population[0][4] * 8 + population[0][5] * 16 + population[0][6] * 32 +
population[0][7] * 64 + population[0][8] * 128 + population[0][9] * 256 +
population[0][10] * 512;
			if (population[0][0] == 0){
				Min = Min * -1;
			}
			Min = Min / 1023 * 5;
			Min = Min * sin(abs(Min));
		}

		for (int j = 0; j < population.size(); j++) {
			for (int k = 0; k < population[j].size(); k++) {
				Average = population[j][1] * 1 + population[j][2] * 2 +
population[j][3] * 4 + population[j][4] * 8 + population[j][5] * 16 + population[j][6] *
```

```
32 + population[j][7] * 64 + population[j][8] * 128 + population[j][9] * 256 +
population[j][10] * 512;
                        if (population[j][0] == 0){
                                Average = Average * -1;
                        }
                        Average = Average / 1023 * 5;
                        Average = Average * sin(abs(Average));
                }
                AverageSumm += Average;
        }
        AverageSumm = AverageSumm / 20;

        file << " Minimum: " << Min << " Average: " << AverageSumm;
        file << '\n';
    }

    file.close();

    print();
    system("Pause");
    return 0;
}

void makePopulation() {
    ofstream file("population.txt", ios::out);
    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 11; j++) {
            file << rand() % 2;
            if (j == 11 - 1) file << '\n';
            else file << ' ';
        }
    }
    file.close();
}

void getPopulation() {
    ifstream file("population.txt");
    for (int i = 0; i < 20; i++) {
        Individual row;
        for (int j = 0; j < 11; j++) {
            double value;
            file >> value;
            row.push_back(value);
        }
        population.push_back(row);
    }
    file.close();
}

int ChromosomeSumm(const Individual& population) {
    double summ = 0;
    summ = population[1] * 1 + population[2] * 2 + population[3] * 4 + population[4] *
8 + population[5] * 16 + population[6] * 32 + population[7] * 64 + population[8] * 128 +
population[9] * 256 + population[10] * 512;
    if (population[0] == 0){
        summ = summ * -1;
    }
    summ = summ / 1023 * 5;
    summ = summ * sin(abs(summ));
    return summ;
}

bool ChromosomeSort(const Individual& population1, const Individual& population2) {
    return ChromosomeSumm(population1) < ChromosomeSumm(population2);
}
```

```
void print(){
	ofstream file("sort.txt", ios::out);
	for (int i = 0; i < population.size(); i++) {
		int tmp = 0;
		for (int j = 0; j < population[i].size(); j++) {
			file << population[i][j];
			if (j == 11 - 1) file << '\n';
			else file << ' ';
		}
	}
	file.close();
}

void crossing() {
	int CrossingPoint;
	int FirstParent, SecondParent;
	Population resultPopulation;

	for (int i = 0; i < 10; i++){
		CrossingPoint = rand() % 11;
		FirstParent = rand() % 10;
		SecondParent = rand() % 10;
		if (FirstParent == SecondParent){
			while (FirstParent != SecondParent)
			{
				SecondParent = rand() % 10;
			}
		}

		Individual Child1, Child2;

		for (int j = 0; j < population[i].size(); j++)
		{
			if (j < CrossingPoint)
			{
				Child1.push_back(population[FirstParent][j]);
				Child2.push_back(population[SecondParent][j]);
			}
			else
			{
				Child1.push_back(population[SecondParent][j]);
				Child2.push_back(population[FirstParent][j]);
			}
		}
		resultPopulation.push_back(Child1);
		resultPopulation.push_back(Child2);
	}
	population = resultPopulation;
}
```

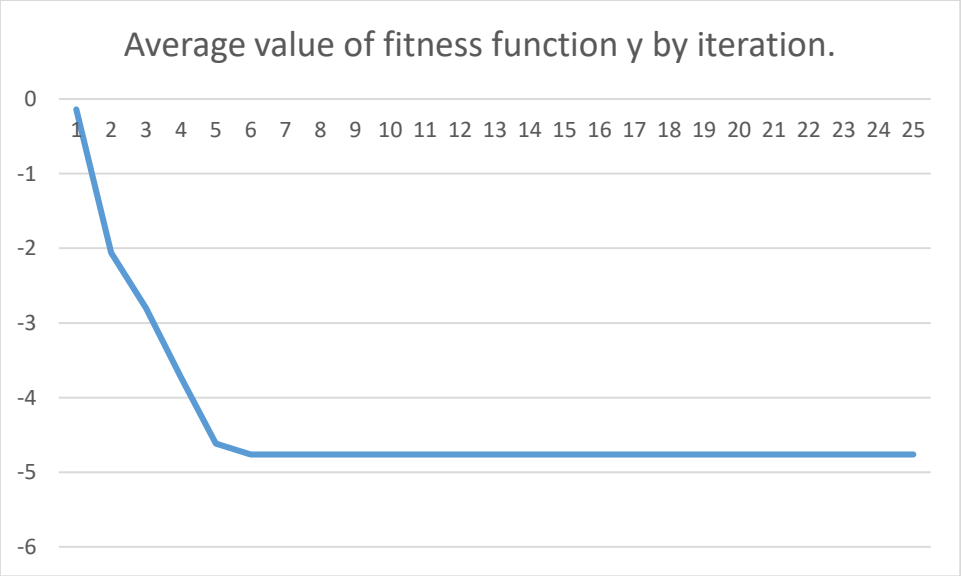
Data for graph:

DataForGraphXY.txt | SIID_2_task2.cpp*

Min	Average
-4.76229	-0.138569
-4.76229	-2.0601
-4.76229	-2.80361
-4.76229	-3.7275
-4.76229	-4.6165
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229
-4.76229	-4.76229

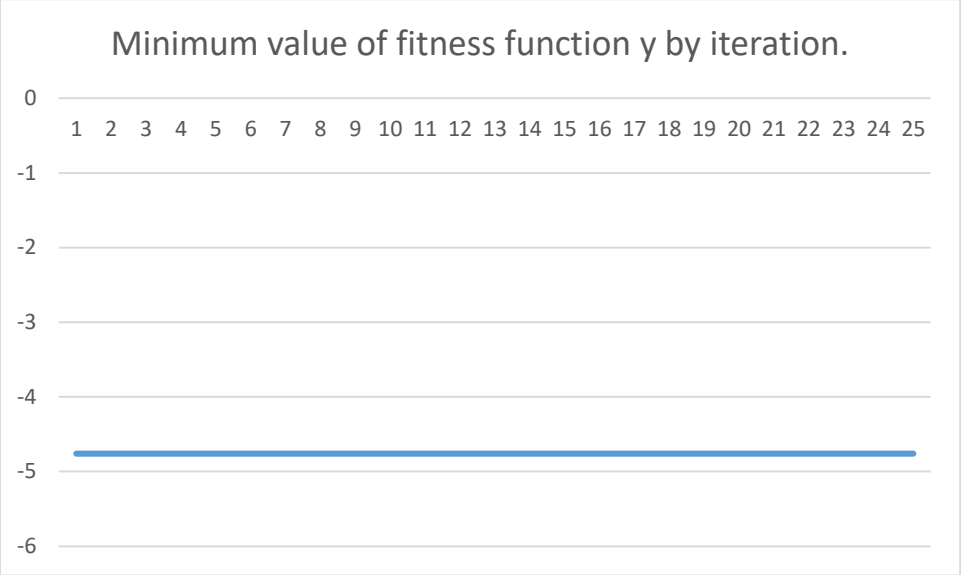
100 %

Graph 1. Average value of fitness function y by iteration.



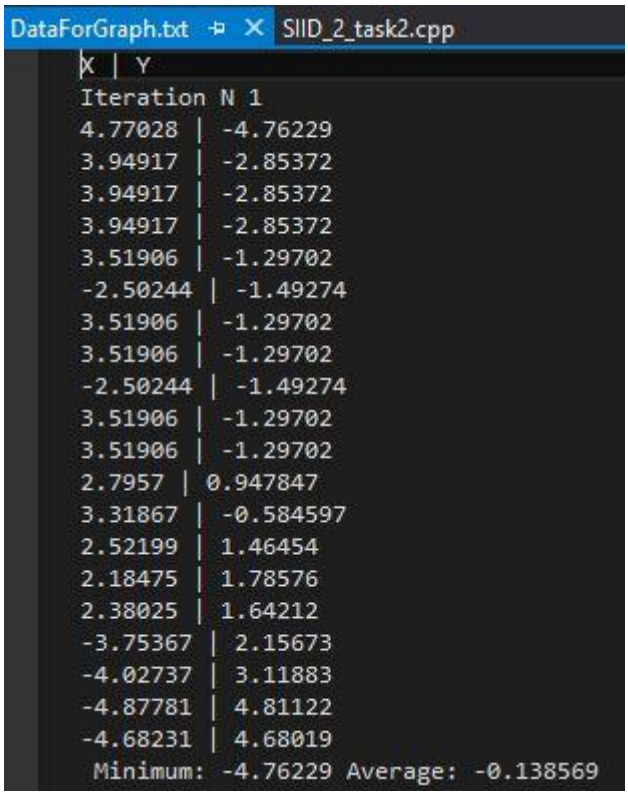
In x line we have current iteration, in y line – average value of function.

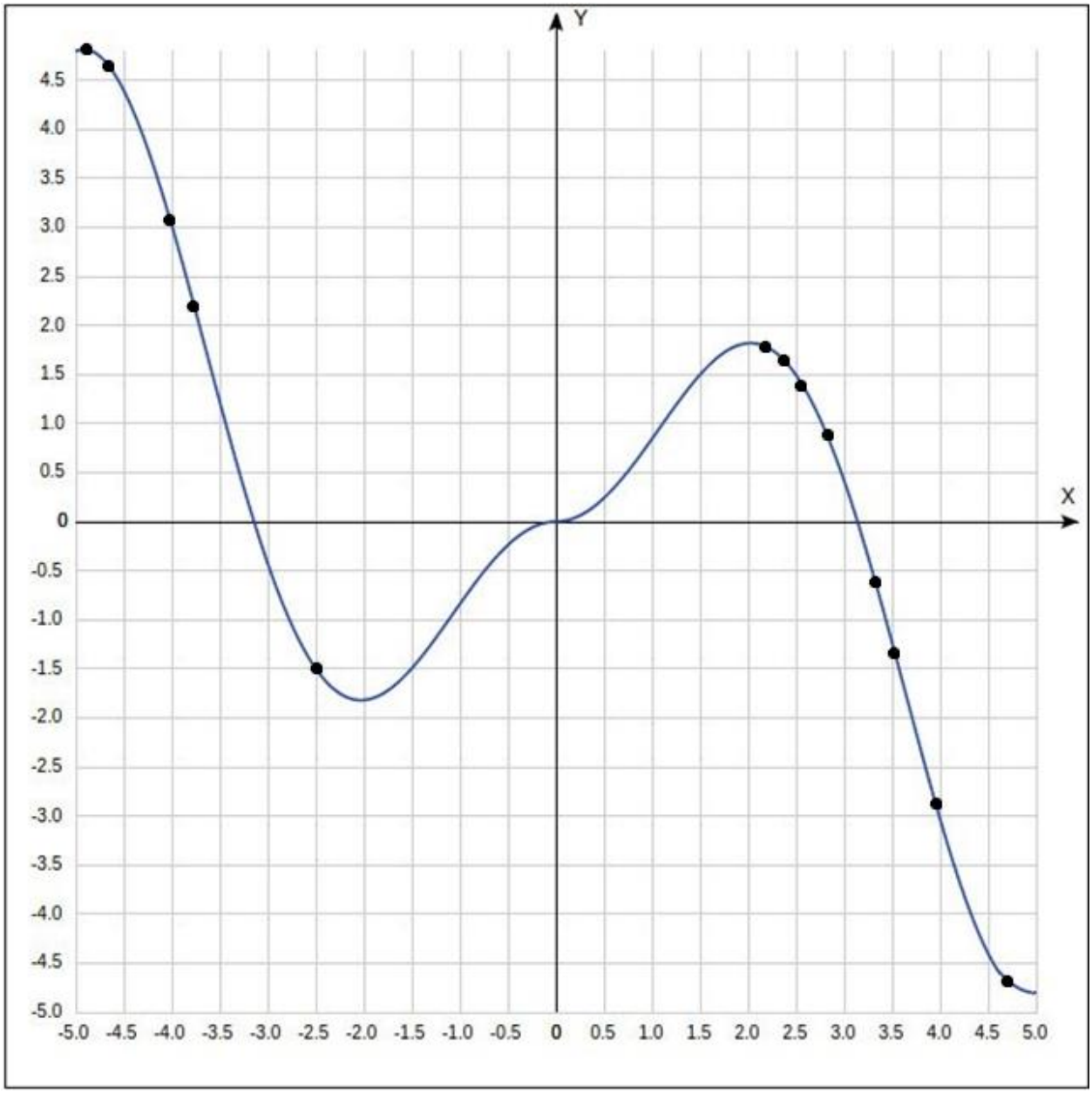
Graph 2. Minimum value of fitness function y by iteration.



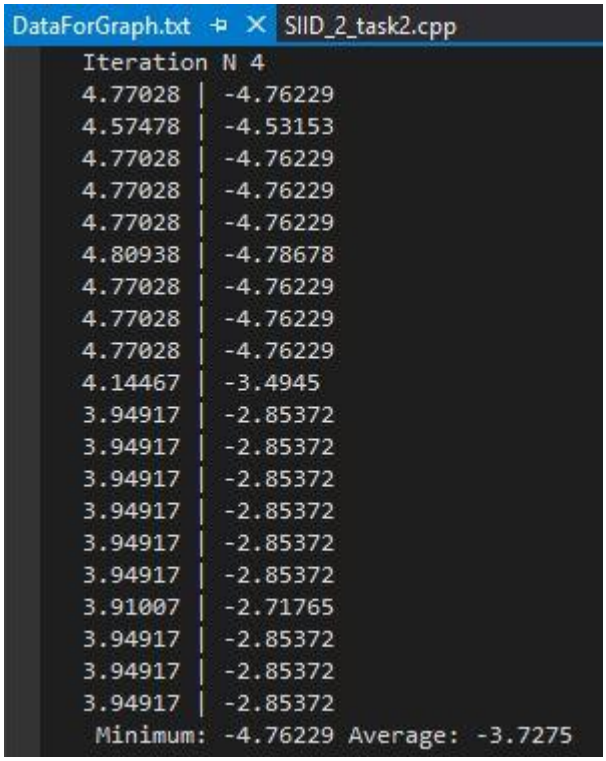
In x line we have current iteration, in y line – minimum value of function.

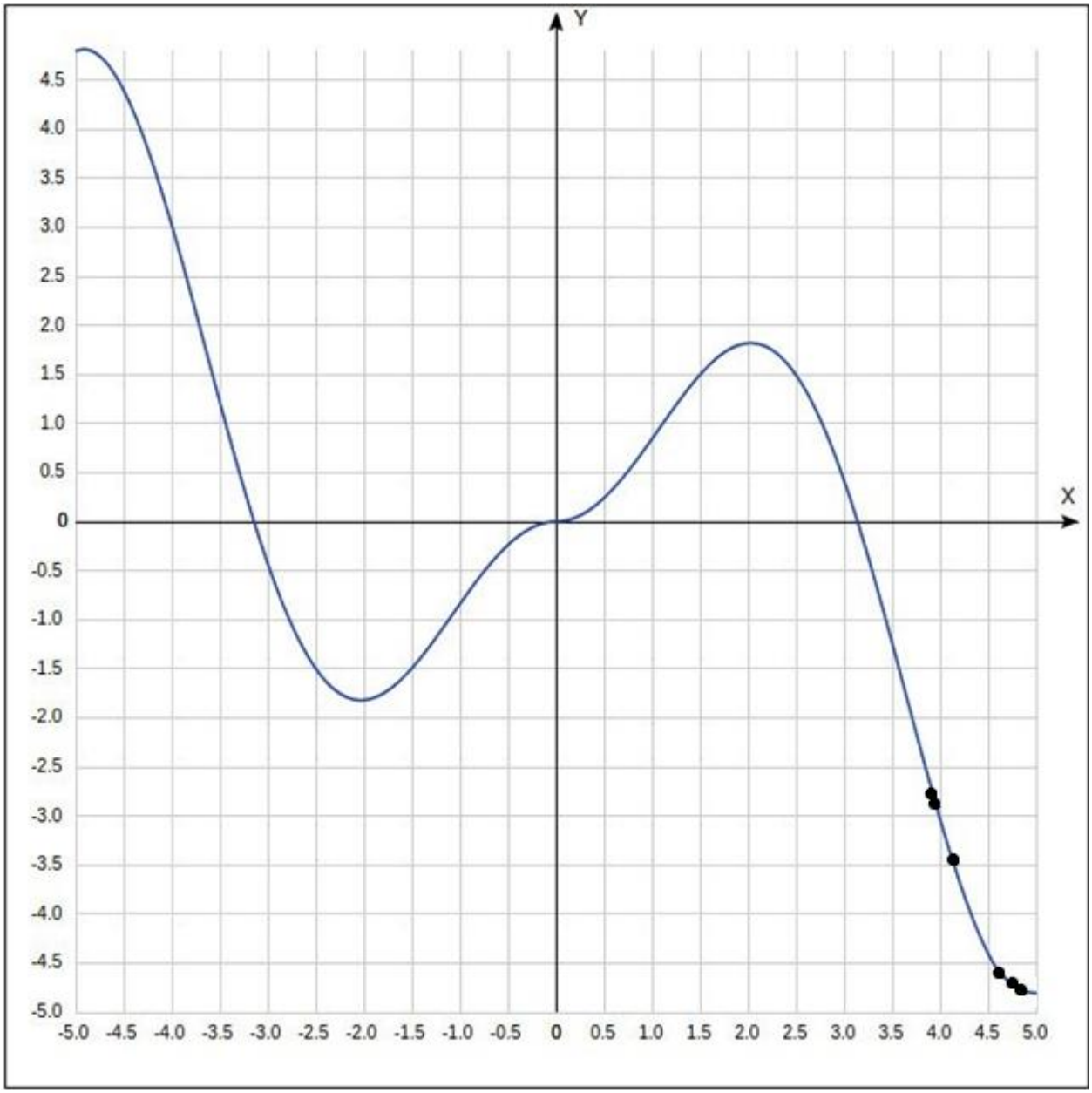
Graph 3. Dots on diagram on 1 iteration.



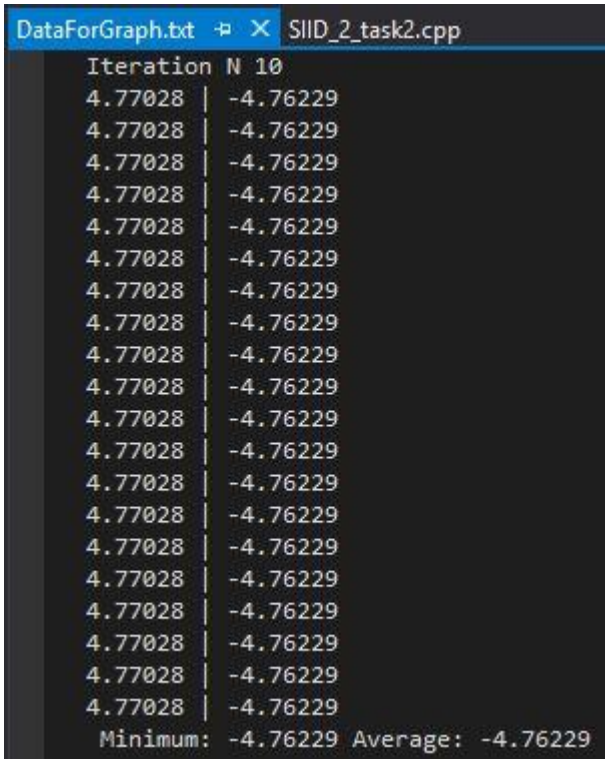

Y
X
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
-3.5
-4.0
-4.5
-5.0
-5.0 -4.5 -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

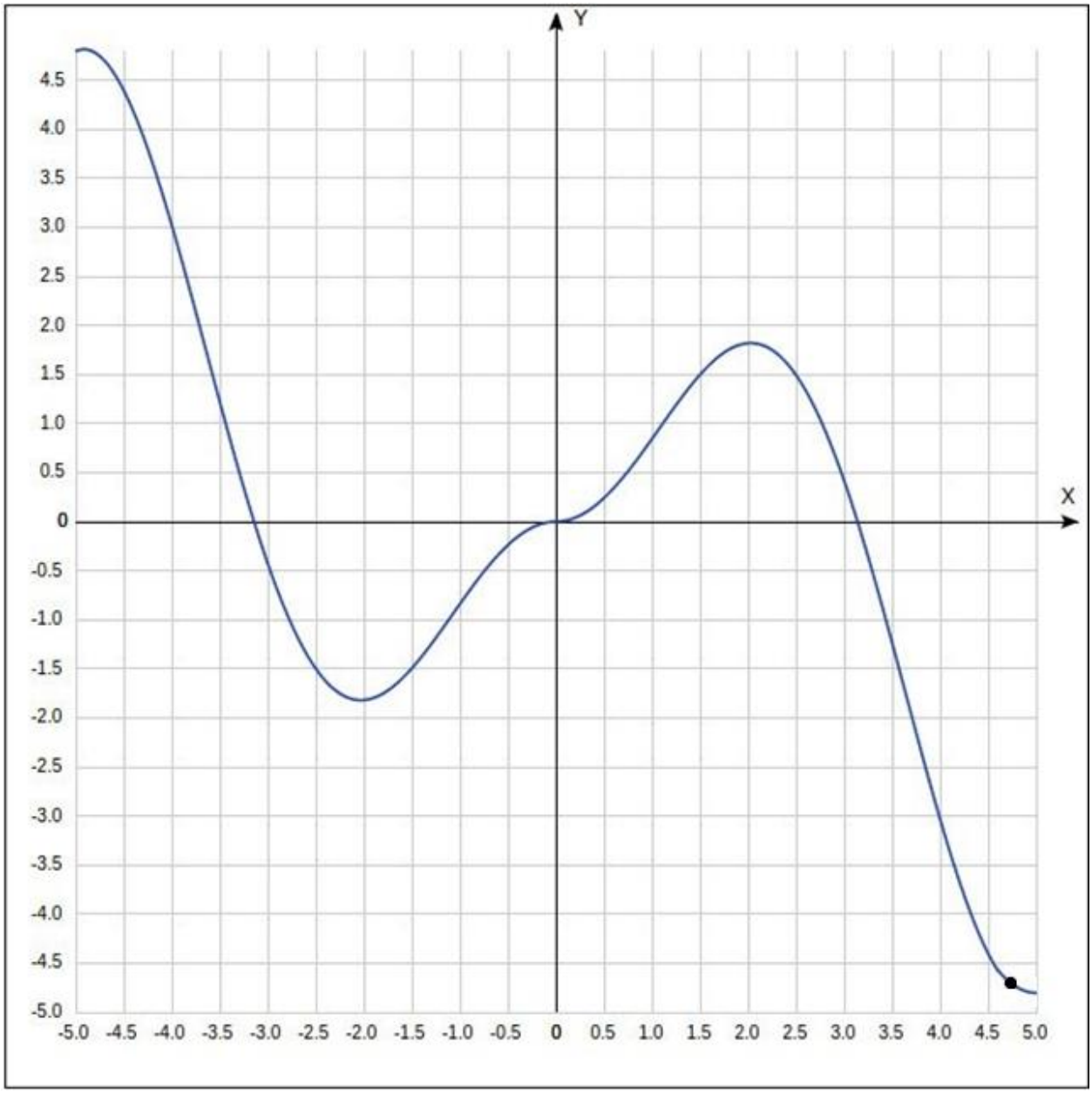
Graph 4. Dots on diagram on 4 iteration.




Y
X
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
-3.5
-4.0
-4.5
-5.0
-5.0 -4.5 -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

Graph 4. Dots on diagram on 10 iteration.




Y
X
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
-3.5
-4.0
-4.5
-5.0
-5.0 -4.5 -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0