

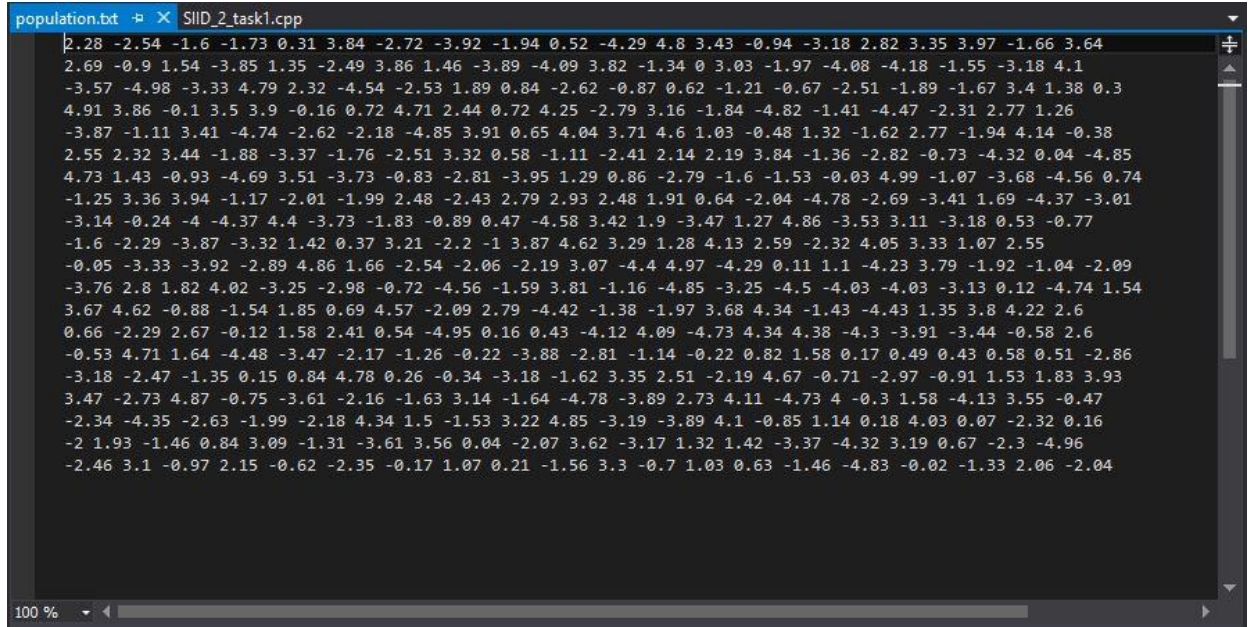
Task 1.

20-Dimensional Schwefel's function.

$$y = x_1 \sin(|x_1|) + x_2 \sin(|x_2|) + \dots + x_{20} \sin(|x_{20}|);$$

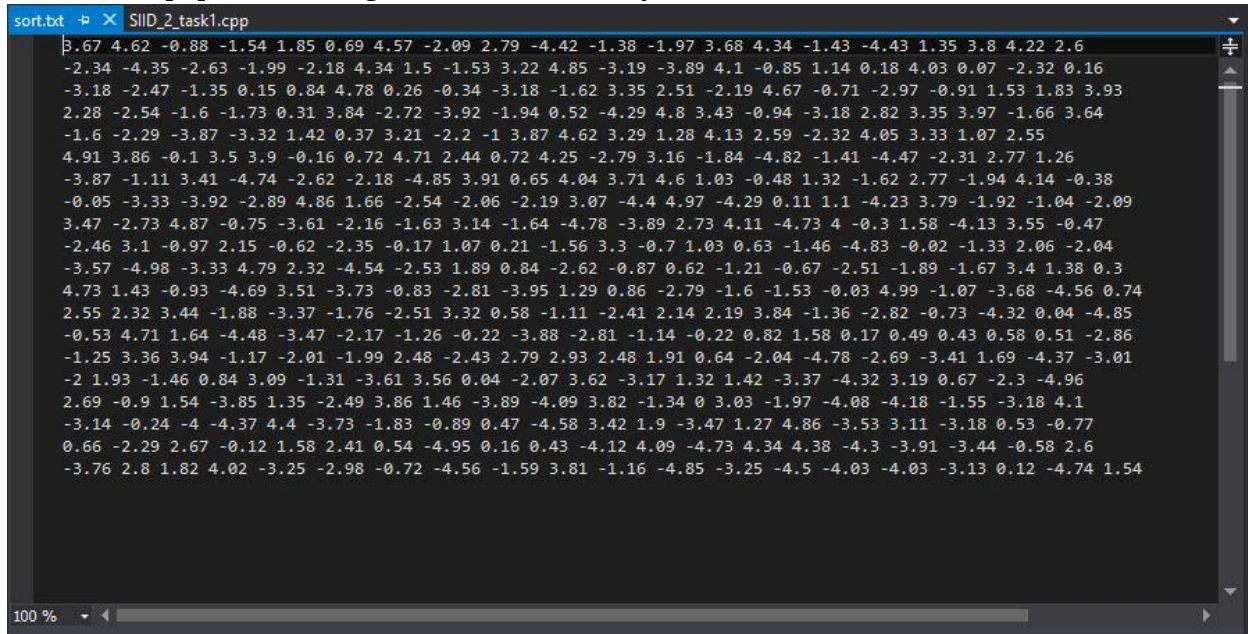
In this task, we must represent each x_i by a chromosome with 20 genes.

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



```
population.txt  X  SIID_2_task1.cpp
2.28 -2.54 -1.6 -1.73 0.31 3.84 -2.72 -3.92 -1.94 0.52 -4.29 4.8 3.43 -0.94 -3.18 2.82 3.35 3.97 -1.66 3.64
2.69 -0.9 1.54 -3.85 1.35 -2.49 3.86 1.46 -3.89 -4.09 3.82 -1.34 0 3.03 -1.97 -4.08 -4.18 -1.55 -3.18 4.1
-3.57 -4.98 -3.33 4.79 2.32 -4.54 -2.53 1.89 0.84 -2.62 -0.87 0.62 -1.21 -0.67 -2.51 -1.89 -1.67 3.4 1.38 0.3
4.91 3.86 -0.1 3.5 3.9 -0.16 0.72 4.71 2.44 0.72 4.25 -2.79 3.16 -1.84 -4.82 -1.41 -4.47 -2.31 2.77 1.26
-3.87 -1.11 3.41 -4.74 -2.62 -2.18 -4.85 3.91 0.65 4.04 3.71 4.6 1.03 -0.48 1.32 -1.62 2.77 -1.94 4.14 -0.38
2.55 2.32 3.44 -1.88 -3.37 -1.76 -2.51 3.32 0.58 -1.11 -2.41 2.14 2.19 3.84 -1.36 -2.82 -0.73 -4.32 0.04 -4.85
4.73 1.43 -0.93 -4.69 3.51 -3.73 -0.83 -2.81 -3.95 1.29 0.86 -2.79 -1.6 -1.53 -0.03 4.99 -1.07 -3.68 -4.56 0.74
-1.25 3.36 3.94 -1.17 -2.01 -1.99 2.48 -2.43 2.79 2.93 2.48 1.91 0.64 -2.04 -4.78 -2.69 -3.41 1.69 -4.37 -3.01
-3.14 -0.24 -4 -4.37 4.4 -3.73 -1.83 -0.89 0.47 -4.58 3.42 1.9 -3.47 1.27 4.86 -3.53 3.11 -3.18 0.53 -0.77
-1.6 -2.29 -3.87 -3.32 1.42 0.37 3.21 -2.2 -1 3.87 4.62 3.29 1.28 4.13 2.59 -2.32 4.05 3.33 1.07 2.55
-0.05 -3.33 -3.92 -2.89 4.86 1.66 -2.54 -2.06 -2.19 3.07 -4.4 4.97 -4.29 0.11 1.1 -4.23 3.79 -1.92 -1.04 -2.09
-3.76 2.8 1.82 4.02 -3.25 -2.98 -0.72 -4.56 -1.59 3.81 -1.16 -4.85 -3.25 -4.5 -4.03 -4.03 -3.13 0.12 -4.74 1.54
3.67 4.62 -0.88 -1.54 1.85 0.69 4.57 -2.09 2.79 -4.42 -1.38 -1.97 3.68 4.34 -1.43 -4.43 1.35 3.8 4.22 2.6
0.66 -2.29 2.67 -0.12 1.58 2.41 0.54 -4.95 0.16 0.43 -4.12 4.09 -4.73 4.34 4.38 -4.3 -3.91 -3.44 -0.58 2.6
-0.53 4.71 1.64 -4.48 -3.47 -2.17 -1.26 -0.22 -3.88 -2.81 -1.14 -0.22 0.82 1.58 0.17 0.49 0.43 0.58 0.51 -2.86
-3.18 -2.47 -1.35 0.15 0.84 4.78 0.26 -0.34 -3.18 -1.62 3.35 2.51 -2.19 4.67 -0.71 -2.97 -0.91 1.53 1.83 3.93
3.47 -2.73 4.87 -0.75 -3.61 -2.16 -1.63 3.14 -1.64 -4.78 -3.89 2.73 4.11 -4.73 4 -0.3 1.58 -4.13 3.55 -0.47
-2.34 -4.35 -2.63 -1.99 -2.18 4.34 1.5 -1.53 3.22 4.85 -3.19 -3.89 4.1 -0.85 1.14 0.18 4.03 0.07 -2.32 0.16
-2 1.93 -1.46 0.84 3.09 -1.31 -3.61 3.56 0.04 -2.07 3.62 -3.17 1.32 1.42 -3.37 -4.32 3.19 0.67 -2.3 -4.96
-2.46 3.1 -0.97 2.15 -0.62 -2.35 -0.17 1.07 0.21 -1.56 3.3 -0.7 1.03 0.63 -1.46 -4.83 -0.02 -1.33 2.06 -2.04
```

2. Sort population using the fitness function y .



```
sort.txt  X  SIID_2_task1.cpp
3.67 4.62 -0.88 -1.54 1.85 0.69 4.57 -2.09 2.79 -4.42 -1.38 -1.97 3.68 4.34 -1.43 -4.43 1.35 3.8 4.22 2.6
-2.34 -4.35 -2.63 -1.99 -2.18 4.34 1.5 -1.53 3.22 4.85 -3.19 -3.89 4.1 -0.85 1.14 0.18 4.03 0.07 -2.32 0.16
-3.18 -2.47 -1.35 0.15 0.84 4.78 0.26 -0.34 -3.18 -1.62 3.35 2.51 -2.19 4.67 -0.71 -2.97 -0.91 1.53 1.83 3.93
2.28 -2.54 -1.6 -1.73 0.31 3.84 -2.72 -3.92 -1.94 0.52 -4.29 4.8 3.43 -0.94 -3.18 2.82 3.35 3.97 -1.66 3.64
-1.6 -2.29 -3.87 -3.32 1.42 0.37 3.21 -2.2 -1 3.87 4.62 3.29 1.28 4.13 2.59 -2.32 4.05 3.33 1.07 2.55
4.91 3.86 -0.1 3.5 3.9 -0.16 0.72 4.71 2.44 0.72 4.25 -2.79 3.16 -1.84 -4.82 -1.41 -4.47 -2.31 2.77 1.26
-3.87 -1.11 3.41 -4.74 -2.62 -2.18 -4.85 3.91 0.65 4.04 3.71 4.6 1.03 -0.48 1.32 -1.62 2.77 -1.94 4.14 -0.38
-0.05 -3.33 -3.92 -2.89 4.86 1.66 -2.54 -2.06 -2.19 3.07 -4.4 4.97 -4.29 0.11 1.1 -4.23 3.79 -1.92 -1.04 -2.09
3.47 -2.73 4.87 -0.75 -3.61 -2.16 -1.63 3.14 -1.64 -4.78 -3.89 2.73 4.11 -4.73 4 -0.3 1.58 -4.13 3.55 -0.47
-2.46 3.1 -0.97 2.15 -0.62 -2.35 -0.17 1.07 0.21 -1.56 3.3 -0.7 1.03 0.63 -1.46 -4.83 -0.02 -1.33 2.06 -2.04
-3.57 -4.98 -3.33 4.79 2.32 -4.54 -2.53 1.89 0.84 -2.62 -0.87 0.62 -1.21 -0.67 -2.51 -1.89 -1.67 3.4 1.38 0.3
4.73 1.43 -0.93 -4.69 3.51 -3.73 -0.83 -2.81 -3.95 1.29 0.86 -2.79 -1.6 -1.53 -0.03 4.99 -1.07 -3.68 -4.56 0.74
2.55 2.32 3.44 -1.88 -3.37 -1.76 -2.51 3.32 0.58 -1.11 -2.41 2.14 2.19 3.84 -1.36 -2.82 -0.73 -4.32 0.04 -4.85
-0.53 4.71 1.64 -4.48 -3.47 -2.17 -1.26 -0.22 -3.88 -2.81 -1.14 -0.22 0.82 1.58 0.17 0.49 0.43 0.58 0.51 -2.86
-1.25 3.36 3.94 -1.17 -2.01 -1.99 2.48 -2.43 2.79 2.93 2.48 1.91 0.64 -2.04 -4.78 -2.69 -3.41 1.69 -4.37 -3.01
-2 1.93 -1.46 0.84 3.09 -1.31 -3.61 3.56 0.04 -2.07 3.62 -3.17 1.32 1.42 -3.37 -4.32 3.19 0.67 -2.3 -4.96
2.69 -0.9 1.54 -3.85 1.35 -2.49 3.86 1.46 -3.89 -4.09 3.82 -1.34 0 3.03 -1.97 -4.08 -4.18 -1.55 -3.18 4.1
-3.14 -0.24 -4 -4.37 4.4 -3.73 -1.83 -0.89 0.47 -4.58 3.42 1.9 -3.47 1.27 4.86 -3.53 3.11 -3.18 0.53 -0.77
0.66 -2.29 2.67 -0.12 1.58 2.41 0.54 -4.95 0.16 0.43 -4.12 4.09 -4.73 4.34 4.38 -4.3 -3.91 -3.44 -0.58 2.6
-3.76 2.8 1.82 4.02 -3.25 -2.98 -0.72 -4.56 -1.59 3.81 -1.16 -4.85 -3.25 -4.5 -4.03 -4.03 -3.13 0.12 -4.74 1.54
```

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;

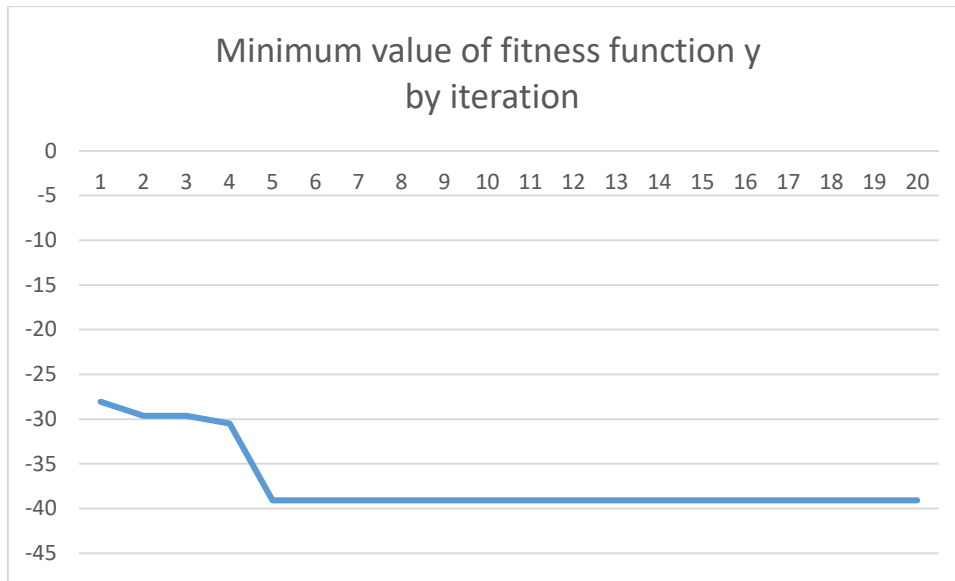
for (int j = 0; j < population[i].size(); j++)
{
    if (j < CrossingPoint)
    {
        Child1.push_back(population[FirstParent][j]);
        Child2.push_back(population[SecondParent][j]);
    }
    else
    {
        Child1.push_back(population[SecondParent][j]);
        Child2.push_back(population[FirstParent][j]);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

```

Data for graph:

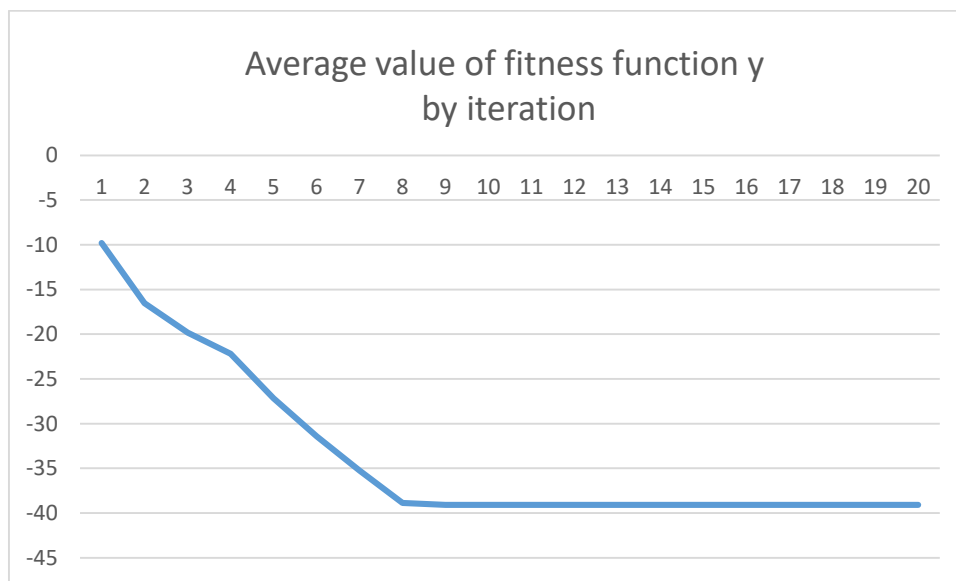
#	Min y	Average y
1	-28.0611	-9.78998
2	-29.6274	-16.5172
3	-29.6274	-19.8377
4	-30.5058	-22.1754
5	-39.0928	-27.1521
6	-39.0928	-31.4315
7	-39.0928	-35.2608
8	-39.0928	-38.8573
9	-39.0928	-39.0928
10	-39.0928	-39.0928
11	-39.0928	-39.0928
12	-39.0928	-39.0928
13	-39.0928	-39.0928
14	-39.0928	-39.0928
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

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.

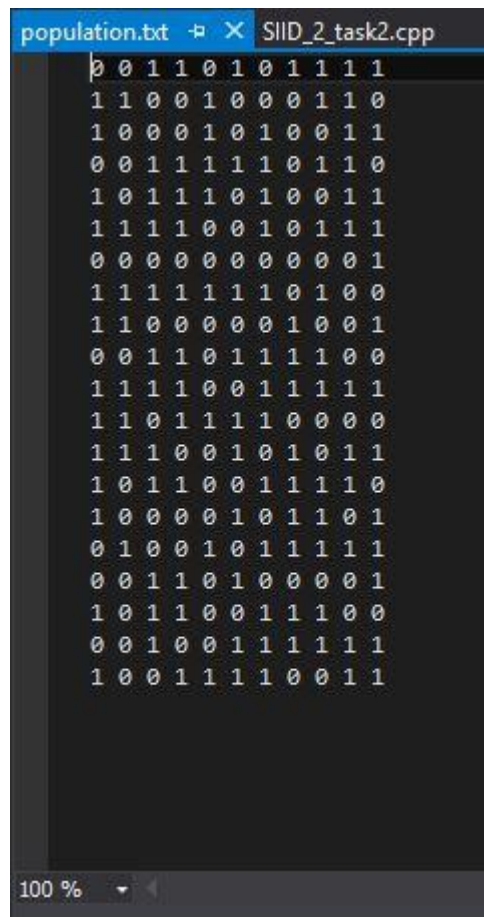
Task 2.

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.



```
population.txt  SIID_2_task2.cpp
0 0 1 1 0 1 0 1 1 1 1
1 1 0 0 1 0 0 0 1 1 0
1 0 0 0 1 0 1 0 0 1 1
0 0 1 1 1 1 1 0 1 1 0
1 0 1 1 1 0 1 0 0 1 1
1 1 1 1 0 0 1 0 1 1 1
0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 0 1 0
1 1 0 0 0 0 0 1 0 0 1
0 0 1 1 0 1 1 1 1 0 0
1 1 1 1 0 0 1 1 1 1 1
1 1 0 1 1 1 1 0 0 0 0
1 1 1 0 0 1 0 1 0 1 1
1 0 1 1 0 0 1 1 1 1 0
1 0 0 0 0 1 0 1 1 0 1
0 1 0 0 1 0 1 1 1 1 1
0 0 1 1 0 1 0 0 0 0 1
1 0 1 1 0 0 1 1 1 0 0
0 0 1 0 0 1 1 1 1 1 1
1 0 0 1 1 1 1 0 0 1 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;

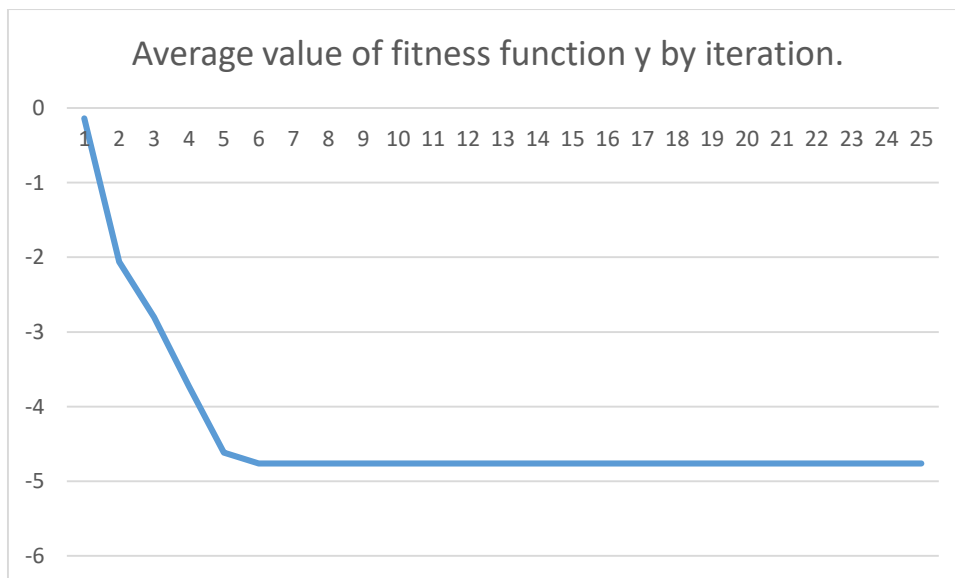
        for (int j = 0; j < population[i].size(); j++)
        {
            if (j < CrossingPoint)
            {
                Child1.push_back(population[FirstParent][j]);
                Child2.push_back(population[SecondParent][j]);
            }
            else
            {
                Child1.push_back(population[SecondParent][j]);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

```

Data for graph:

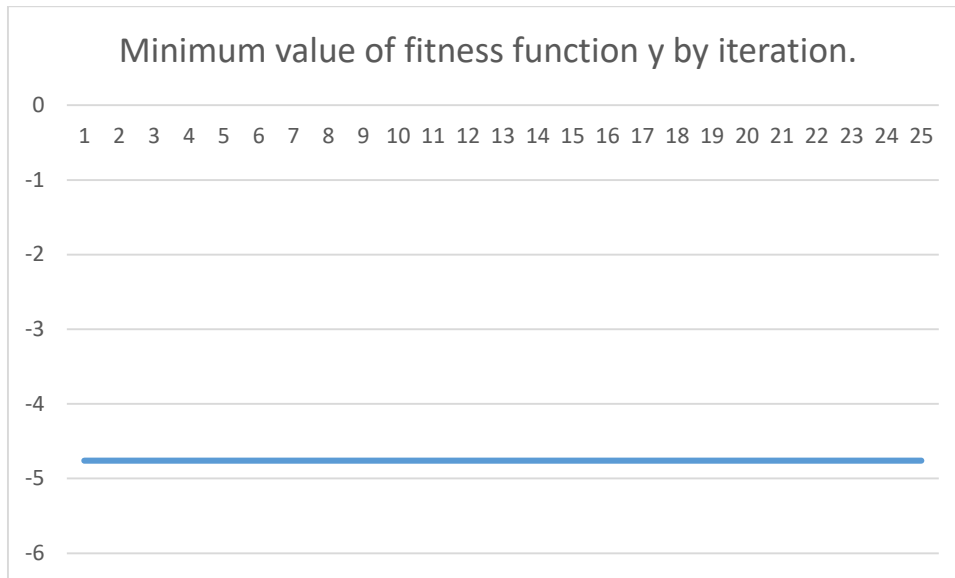
[illegible]

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



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

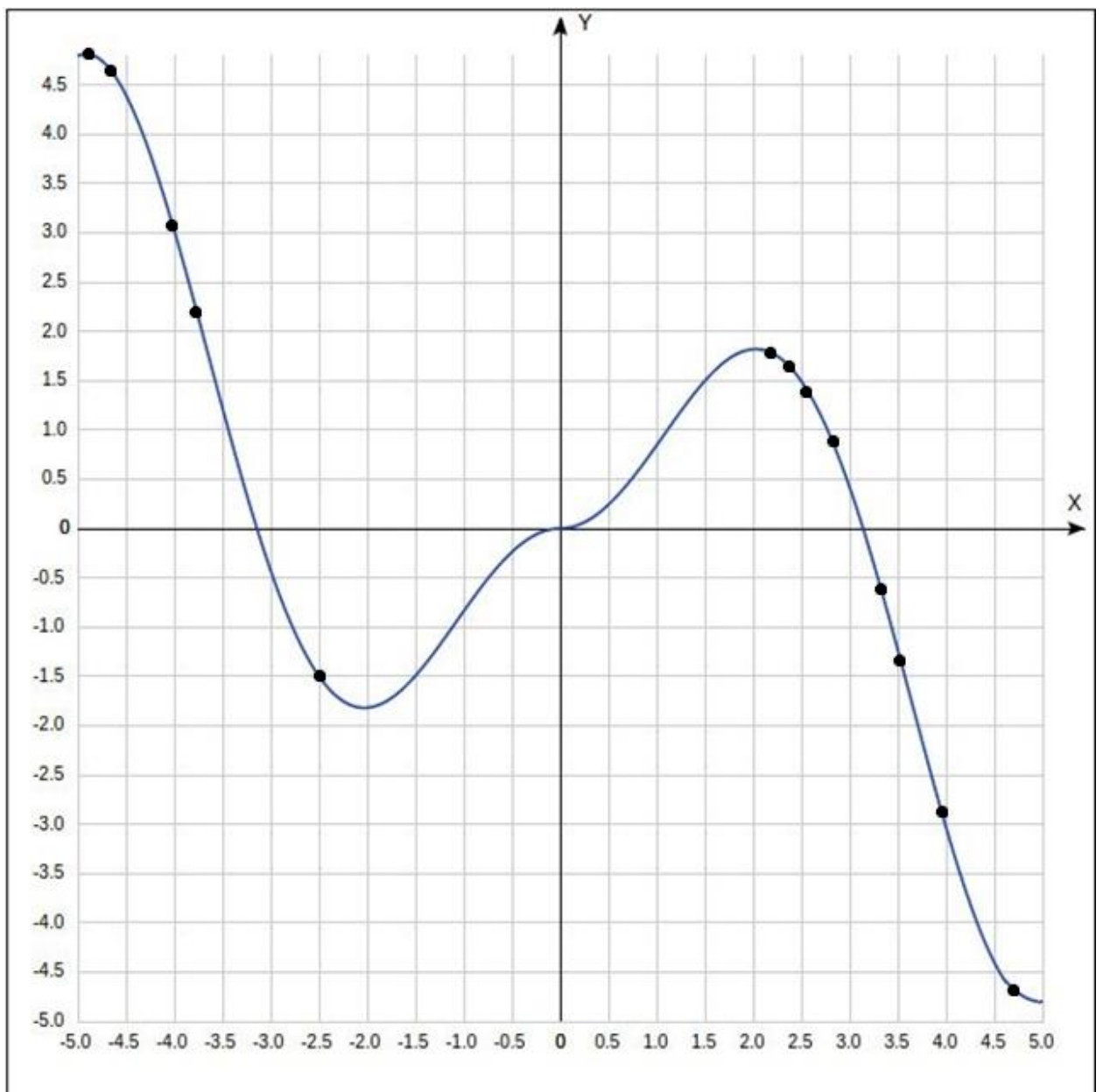
Graph 2. Minimum value of fitness function y by iteration.



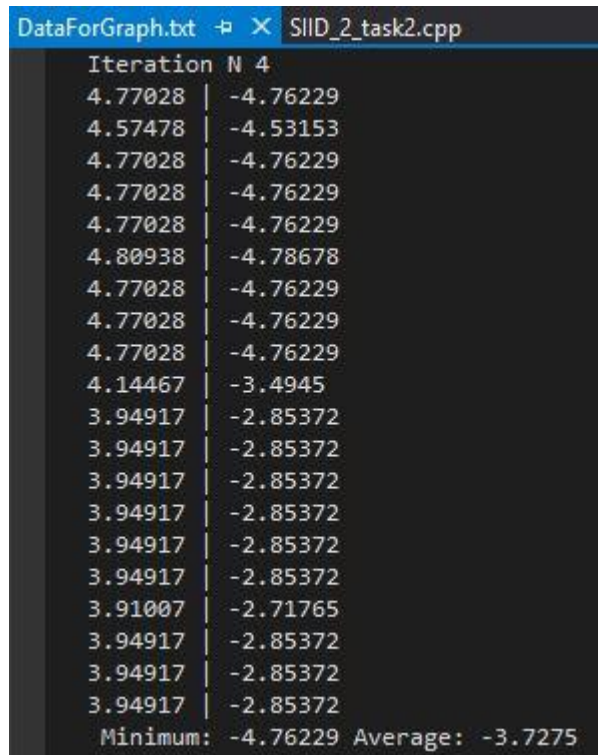
In x line we have current iteration, in y line – minimum value of function.

Graph 3. Dots on diagram on 1 iteration.

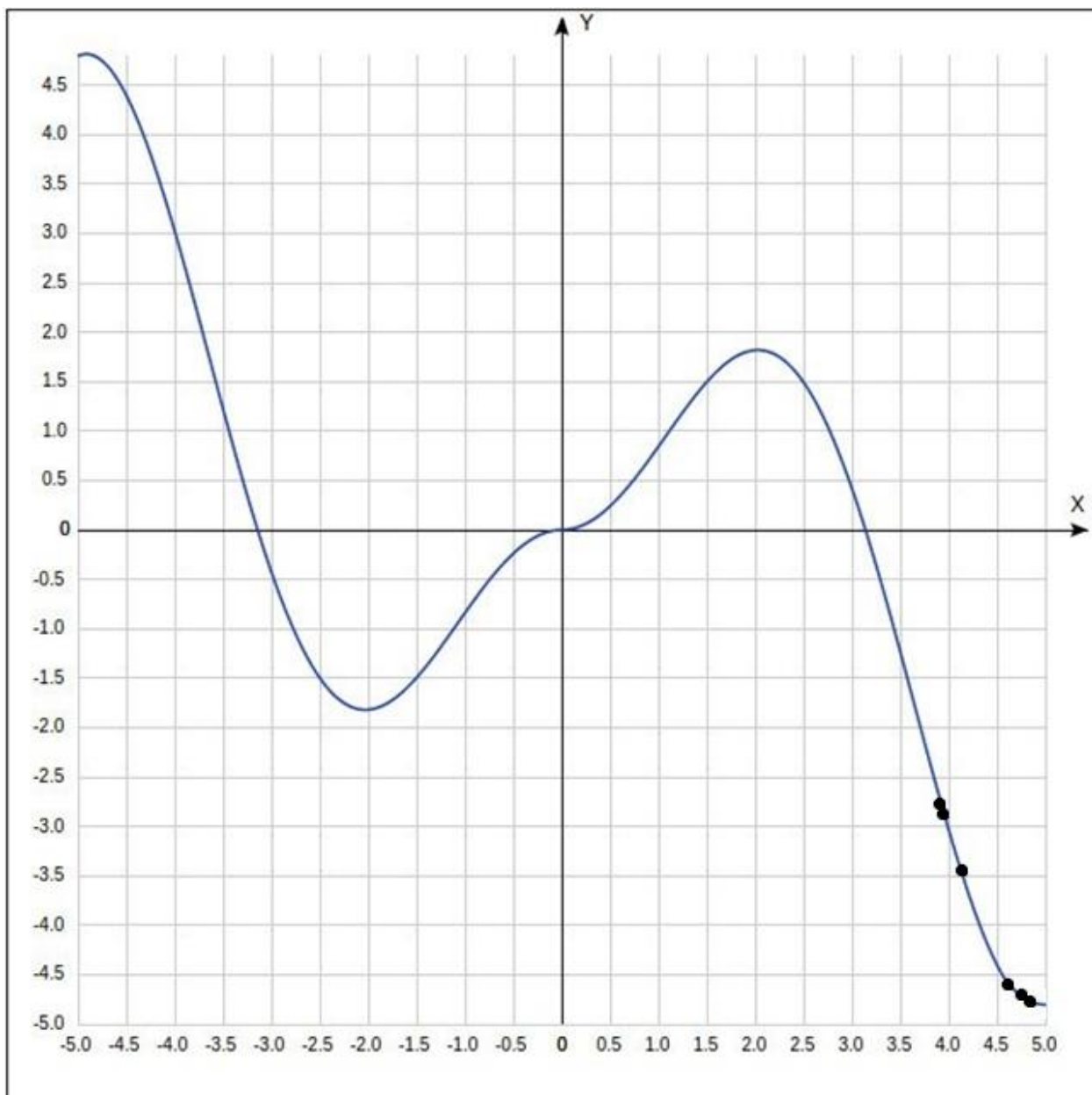
DataForGraph.txt	
X	Y
Iteration N 1	
4.77028	-4.76229
3.94917	-2.85372
3.94917	-2.85372
3.94917	-2.85372
3.51906	-1.29702
-2.50244	-1.49274
3.51906	-1.29702
3.51906	-1.29702
-2.50244	-1.49274
3.51906	-1.29702
3.51906	-1.29702
2.7957	0.947847
3.31867	-0.584597
2.52199	1.46454
2.18475	1.78576
2.38025	1.64212
-3.75367	2.15673
-4.02737	3.11883
-4.87781	4.81122
-4.68231	4.68019
Minimum: -4.76229 Average: -0.138569	



Graph 4. Dots on diagram on 4 iteration.



```
Iteration N 4
4.77028 | -4.76229
4.57478 | -4.53153
4.77028 | -4.76229
4.77028 | -4.76229
4.77028 | -4.76229
4.80938 | -4.78678
4.77028 | -4.76229
4.77028 | -4.76229
4.77028 | -4.76229
4.14467 | -3.4945
3.94917 | -2.85372
3.94917 | -2.85372
3.94917 | -2.85372
3.94917 | -2.85372
3.94917 | -2.85372
3.94917 | -2.85372
3.91007 | -2.71765
3.94917 | -2.85372
3.94917 | -2.85372
3.94917 | -2.85372
Minimum: -4.76229 Average: -3.7275
```



Graph 4. Dots on diagram on 10 iteration.

[illegible]

