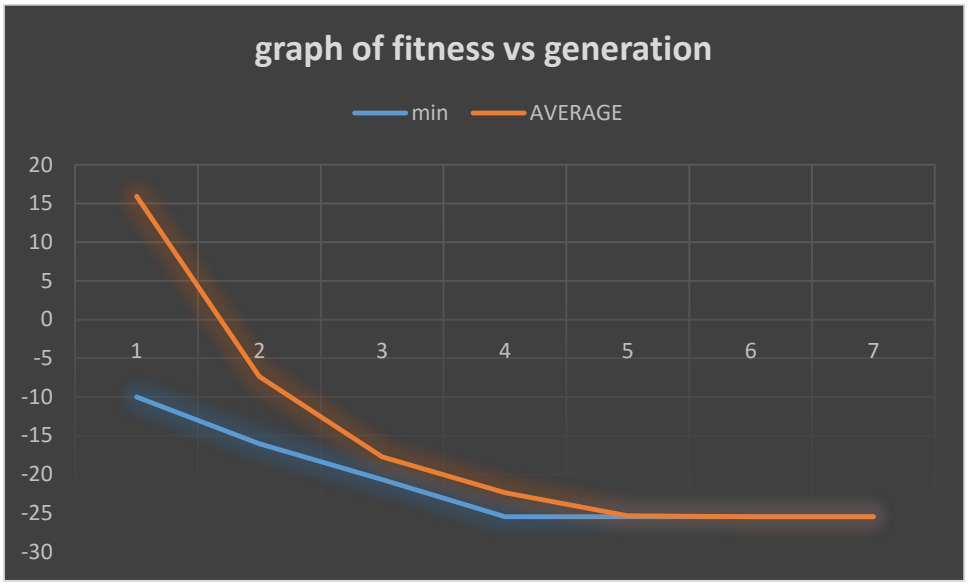
Modern intelligent IT Lab 2 (21.09.2016)
Student – Maxim Tatachka(II-10)

Task 1. 20-Dimensional Schwefel's function:



Code of program:

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

using namespace std;


// fitness
double fitness(const vector<double>& chromosoma) {
	double summ = 0;
	for (double i = 0; i < chromosoma.size() - 1; i++) {
		summ += (chromosoma[i] * sin(abs(chromosoma[i])));
		}

	return summ;
}

bool func(const vector<double>& a, const vector<double>& b) {
	return fitness(a) < fitness(b);
}



int main()
```

```
{
	srand(time(NULL));
	double sizeOfPopulation, rod2Nomer,min1, min, average, min2,average1;
	min = 0;
	min1 = 0;
	min2 = 0;
	average = 0;
	average1 = 0;
	sizeOfPopulation = 20;

	vector<vector<double>> population;

	// started population
	for (double i = 0; i < sizeOfPopulation; i++) {
		vector<double> chromosoma;

		for (double j = 0; j < 20; j++) {
			double rn;
			rn = (rand() % 1000) / 100 - 5;
			chromosoma.push_back(rn);
		}

		population.push_back(chromosoma);
	}

	// cout started population
	cout << "Started population:" << endl;
	for (auto e : population) {

		for (auto l : e)

			average += fitness(e);

		if (min1 > fitness(e)){
			min1 = fitness(e);
		}

	}
	cout << "min = " << min1 << endl;
	cout << "AVERAGE = " << average / 200 << endl;
	average = 0;
	for (double steps = 0; steps < 100; steps++) {
		// choose parents and create child
		vector<vector<double>> potomki;
		for (double i = 0; i < sizeOfPopulation; i++) {
			double rod1Nomer = rand() % (population.size()/2);
			vector<double> roditel_1 = population[rod1Nomer];
			double rod2Nomer = rand() % (population.size()/2);
			vector<double> roditel_2 = population[rod2Nomer];
			// create
			vector<double> potomok_1, potomok_2;



			double chislo = rand() % (roditel_1.size() - 2) + 1;

			potomok_1.insert(potomok_1.end(), roditel_1.begin(),
roditel_1.begin() + chislo);
			potomok_1.insert(potomok_1.end(), roditel_2.begin() +
chislo, roditel_2.end());
			potomok_2.insert(potomok_2.end(), roditel_2.begin(),
roditel_2.begin() + chislo);
			potomok_2.insert(potomok_2.end(), roditel_1.begin() +
chislo, roditel_1.end());
```

```
		potomki.push_back(potomok_1);
		potomki.push_back(potomok_2);




}

// mutation
double ver_mutazii = 0.001; // 0.001
for (double i = 0; i < potomki.size(); i++) {
		double rn = rand() % 100 + 1;
		if (rn < ver_mutazii) {
			double pos = rand() % (potomki[0].size() - 4) + 2;
			double buff = potomki[i][pos - 1];
			potomki[i][pos - 1] = potomki[i][pos + 1];
			potomki[i][pos + 1] = buff;
		}
}

cout << "Iteration: " << steps + 1 <<endl;
for (auto e : population) {
		for (auto l : e)

			average += fitness(e);

		if (min > fitness(e)){
			min = fitness(e);
		}
}

cout << "min = " << min << endl;
cout << "AVERAGE = " << average / 200 << endl;


if ((min == min2) && (average == average1))
{
		cout << "Finished population:" << endl;
		cout << "min = " << min2 << endl;
		cout << "AVERAGE = " << average / 200 << endl;
		break;
}

average1 = average;
min2 = min;
average = 0;
min = 0;


// select new pop
vector<vector<double>> vse;
vse.insert(vse.end(), population.begin(), population.end());
vse.insert(vse.end(), potomki.begin(), potomki.end());
sort(vse.begin(), vse.end(), func);

population.clear();
for (double i = 0; i < (sizeOfPopulation / 2); i++) {
		population.push_back(vse.at(i));
}
```

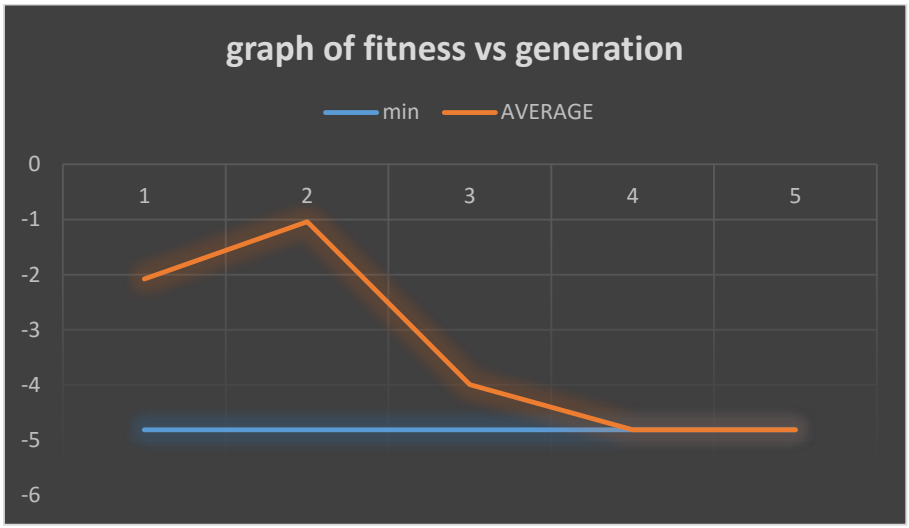
```
        }

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

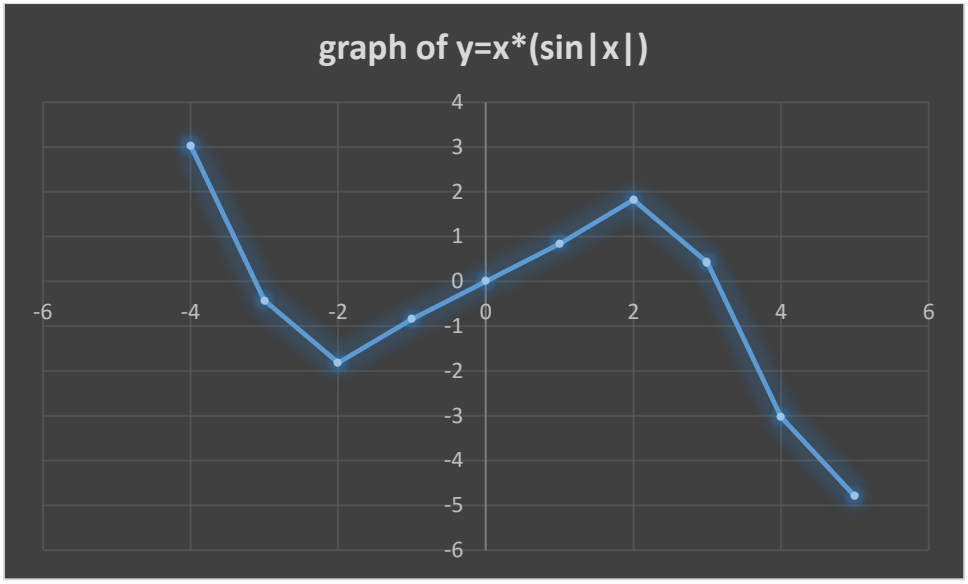
Work of program:

```
Started population:
min = -10.0265
AVERAGE = 15.9285
Iteration: 1
min = -10.0265
AVERAGE = 15.9285
Iteration: 2
min = -16.0633
AVERAGE = -7.3541
Iteration: 3
min = -20.694
AVERAGE = -17.7617
Iteration: 4
min = -25.4886
AVERAGE = -22.3841
Iteration: 5
min = -25.4886
AVERAGE = -25.3491
Iteration: 6
min = -25.4886
AVERAGE = -25.4886
Iteration: 7
min = -25.4886
AVERAGE = -25.4886
Finished population:
min = -25.4886
AVERAGE = -25.4886
```

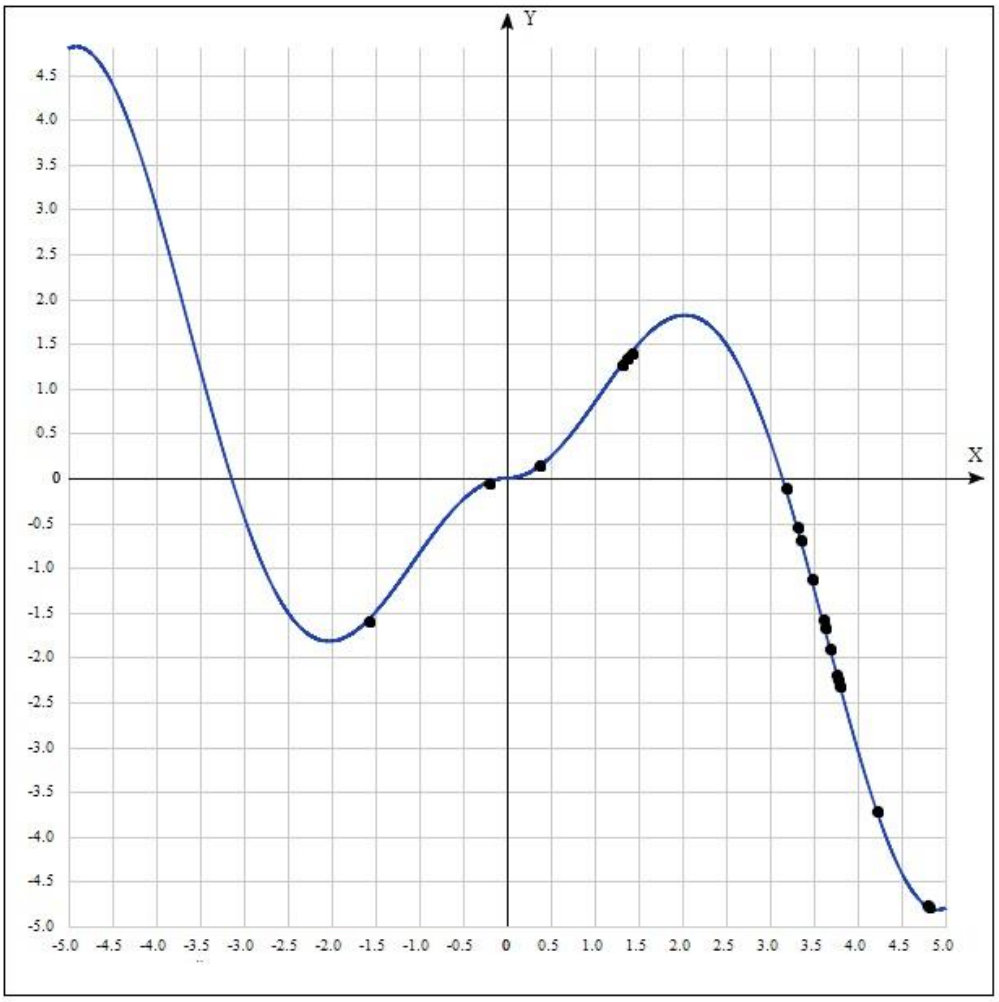
Task 2. 2-D version of Schwefel's function:

Function= xsin(|x|) Interval =-5;5

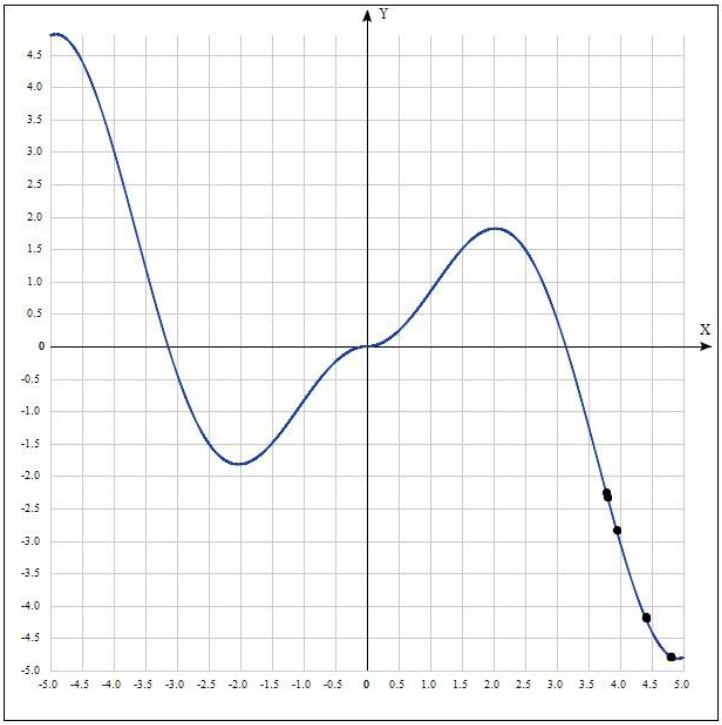



graph of y=x*(sin|x|)
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6
-4
-2
0
2
4
6

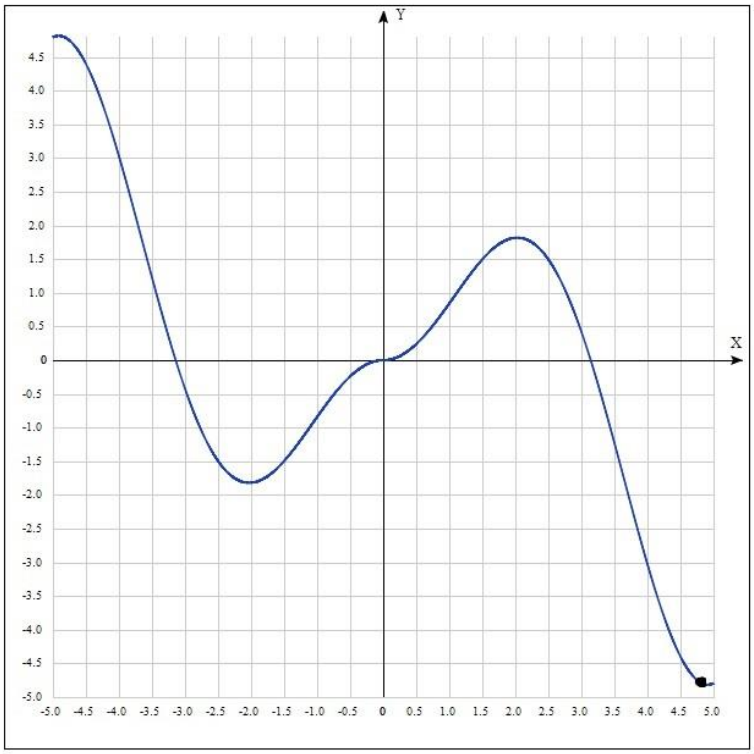
Start:


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

Average:



Final:



Code of program:

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

using namespace std;


// fitness
double fitness(const vector<double>& chromosoma) {
	double summ = 0;
	for (double i = 0; i < chromosoma.size() - 1; i++) {
		summ = (chromosoma[i] * sin(abs(chromosoma[i])));

		}

	return summ;
}

bool func(const vector<double>& a, const vector<double>& b) {
	return fitness(a) < fitness(b);
}



int main()
{
	srand(time(NULL));
	double sizeOfPopulation, rod2Nomer, min1, min, average, min2, average1,
mass[10],summ1,x,mel;
	x = 0;
	mel = 9;
	summ1 = 0;
	min = 0;
	min1 = 0;
	min2 = 0;
	average = 0;
	average1 = 0;
	sizeOfPopulation = 20;

	vector<vector<double>> population;

	// started population
	for (double i = 0; i < sizeOfPopulation; i++) {
		vector<double> chromosoma;

		for (double j = 0; j < 20; j++) {
			for (int k = 0; k < 10; k++){
				mass[k] = (rand() % 2);

			}



			for (int d = 0; d < 10; d++){
				summ1 += (mass[d] * (pow(2, mel)));
				mel--;
			}
```

```
                mel = 9;



                x = ((summ1 / 1023) * 5);

                summ1 = 0;
                if (mass[0] == 0){
                      x = -x;
                      chromosoma.push_back(x);


                }
                else chromosoma.push_back(x);

                x = 0;
          }


          population.push_back(chromosoma);
    }

    // cout started population
    cout << "Started population:" << endl;
    for (auto e : population) {



          for (auto l : e)

                average += fitness(e);

          if (min1 > fitness(e)){
                min1 = fitness(e);
          }

    }
    cout << "min = " << min1 << endl;
    cout << "AVERAGE = " << average / 200 << endl;
    average = 0;
    for (double steps = 0; steps < 100; steps++) {
          // choose parents and create child
          vector<vector<double>> potomki;
          for (double i = 0; i < sizeOfPopulation; i++) {
                double rod1Nomer = rand() % (population.size()/2);
                vector<double> roditel_1 = population[rod1Nomer];
                double rod2Nomer = rand() % (population.size()/2);
                vector<double> roditel_2 = population[rod2Nomer];
                // create
                vector<double> potomok_1, potomok_2;



                double chislo = rand() % (roditel_1.size() - 2) + 1;

                potomok_1.insert(potomok_1.end(), roditel_1.begin(),
roditel_1.begin() + chislo);
                potomok_1.insert(potomok_1.end(), roditel_2.begin() +
chislo, roditel_2.end());
                potomok_2.insert(potomok_2.end(), roditel_2.begin(),
roditel_2.begin() + chislo);
                potomok_2.insert(potomok_2.end(), roditel_1.begin() +
chislo, roditel_1.end());
                potomki.push_back(potomok_1);
```

```
		potomki.push_back(potomok_2);




	}

	// mutation
	double ver_mutazii = 0.001; // 0.001
	for (double i = 0; i < potomki.size(); i++) {
		double rn = rand() % 100 + 1;
		if (rn < ver_mutazii) {
			double pos = rand() % (potomki[0].size() - 4) + 2;
			double buff = potomki[i][pos - 1];
			potomki[i][pos - 1] = potomki[i][pos + 1];
			potomki[i][pos + 1] = buff;
		}
	}

	cout << "Iteration: " << steps + 1 <<endl;
	for (auto e : population) {

		cout << fitness(e) << endl;


		for (auto l : e)

			average += fitness(e);

		if (min > fitness(e)){
			min = fitness(e);
		}
	}

	cout << "min = " << min << endl;
	cout << "AVERAGE = " << average / 400 << endl;


	if ((min == min2) && (average == average1))
	{
		cout << "Finished population:" << endl;
		cout << "min = " << min2 << endl;
		cout << "AVERAGE = " << average / 400 << endl;
		break;
	}

	average1 = average;
	min2 = min;
	average = 0;
	min = 0;


	// select new pop
	vector<vector<double>> vse;
	vse.insert(vse.end(), population.begin(), population.end());
	vse.insert(vse.end(), potomki.begin(), potomki.end());
	sort(vse.begin(), vse.end(), func);

	population.clear();
	for (double i = 0; i < (sizeOfPopulation); i++) {
		population.push_back(vse.at(i));
	}
```

```
        }

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

Work of program:

```
Started population:
min = -4.81437
AVERAGE = -2.59911
Iteration: 1
-4.81305
1.39678
-1.97796
-1.19053
-1.63804
-1.78576
-2.31657
-1.60496
-2.28116
-4.23967
-0.582126
-0.118019
1.42003
-2.8874
1.43517
1.33997
-4.81437
0.308082
-1.59764
-0.0438454
min = -4.81437
AVERAGE = -1.29955
```

```
Iteration: 2
-4.81437
-4.81305
-4.81305
-4.81305
-4.23967
-4.23967
-4.23967
-4.23967
-4.23967
-2.8874
-2.31657
-2.31657
-2.31657
-2.31657
-2.31657
-2.31657
-2.28116
-2.28116
-2.28116
-2.28116
min = -4.81437
AVERAGE = -3.31817
```

```
Iteration: 5
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
-4.81437
min = -4.81437
AVERAGE = -4.81437
Finished population:
min = -4.81437
AVERAGE = -4.81437
```