Source code on JS:

```
var population = init();
		var sigma = 1.5;
		var alpha = 2;

		function randomInit() {
			return Math.random() > 0.5 ? 1 : 0;
		}

		function sort(population) {
			return population.sort(function(first, second){
				return (first.fitness < second.fitness) - (second.fitness < first.fitness);
			});
		}

		function init(){
			var population = [];

			for(var i = 0; i < 20; i++){
				var chromosome = [];
				for(var j = 0; j < 22; j++) {
					chromosome.push(randomInit());
				}
				population.push(chromosome);
			}
			return population;
		}

		function getX(chromosome) {
			var x = 0;

			for(var i = 1; i < 11; i++) {
```

```
            x += chromosome[i] ? Math.pow(2, i - 1) / 2048 * 5 : 0;
        }
        x *= chromosome[0] ? -1 : 1;

        return x;
}

function getY(chromosome){
        var y = 0;

        for(var i = 11; i < 22; i++){
            y += chromosome[i] ? Math.pow(2, i - 11) / 2048 * 5: 0;
        }
        y *= chromosome[11] ? -1 : 1;

        return y;
}

function getS(d){
        return d < sigma ? 1 - Math.pow(d / sigma, alpha) : 0;
}

function fitness(population){
        for(var i = 0; i < 20; i++) {
            var x = getX(population[i]);
            var y = getY(population[i]);

            population[i].fitness = x * Math.sin(x) + y * Math.sin(y)
        }
        return population;
}
```

```
function sharedFitness(population){
	var memPopulation = population;

	for(var i = 0; i < 20; i++) {
		for(var j = 0; j < 20; j++) {
			var x = getX(population[i]) - getX(population[j]);
			var y = getY(population[i]) - getY(population[j]);
		}
		var d = Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2));
		population[i].fitness += getS(d);
	}
	return population;
}

function uniformCrossover(firstChromosome, secondChromosome) {
	var returnChromosomes = [];
	var returnChromosome1 = [];
	var returnChromosome2 = [];

	for(var i = 0; i < 22; i++){
		Math.random() > 0.5 ? returnChromosome1.push(firstChromosome[i]) :
		returnChromosome1.push(secondChromosome[i]);
		Math.random() > 0.5 ?
returnChromosome2.push(secondChromosome[i]) :
		returnChromosome2.push(firstChromosome[i]);
	}
	returnChromosomes.push(returnChromosome1);
	returnChromosomes.push(returnChromosome2);

	return returnChromosomes;
}

function mutation(population){
```

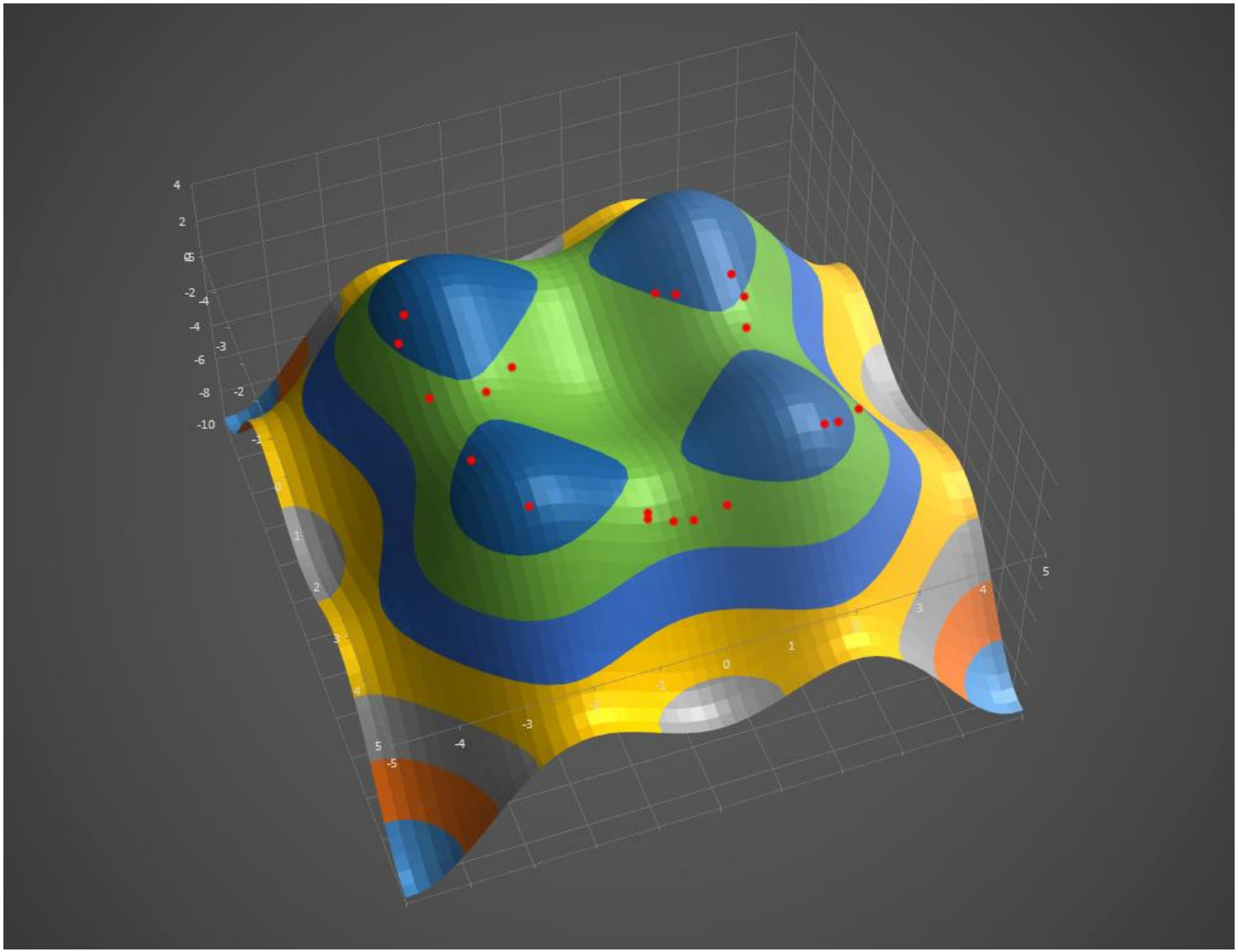
```
                return population.map(function(chromosome) {
                        return chromosome.map(function(gene) {
                                return Math.random() < 0.02 ? (gene + 1) % 2 : gene;
                        });
                });
        }

        function arrayComparison(prev, cur) {
                var flag = true;
                for(var i = 0; i < 20; i++){
                        prev[i].fitness === cur[i].fitness ? flag : flag = false;
                }
                return flag;
        }

        function show(population) {
                var returnStr = '';
                var averageFitness = 0;
                for(var i = 0; i < 20; i++) {
                        averageFitness += population[i].fitness;
                        returnStr += i + '\t' + getX(population[i]).toFixed(3) + '\t ' +
getY(population[i]).toFixed(3) + '\t' + population[i].fitness + '\n';
                }
                returnStr += 'max: ' + population[i].fitness + '\n';
                returnStr += 'aver: ' + (averageFitness / 20) + '\n';

                console.log(returnStr);
        }

        population = fitness(population);
        population = sort(population);
        population = sharedFitness(population);
        population = sort(population);
```

```
		var prevPopulation = [];
		var iteration = 0;
		var statisticArray = [];

		while(true){
			prevPopulation = population;
			var parents = population.slice(0, 10);
			var children = [];
			for(var j = 0; j < 10; j++){
				var tempChild = uniformCrossover(parents[Math.floor(Math.random() *
10)],
	parents[Math.floor(Math.random() * 10)]);
				children.push(tempChild[0]);
				children.push(tempChild[1]);

			}
			children = mutation(children);
			children = fitness(children);

			var newPopulation = [];
			newPopulation = newPopulation.concat(children, parents);
			newPopulation = sharedFitness(newPopulation);
			newPopulation = sort(newPopulation);

			population = newPopulation;
			iteration++;
			statisticArray.push(newPopulation);
			if(arrayComparison(prevPopulation, newPopulation)){
				break;
			}
		}
```

```
var statistic = [];
for(var i = 0; i < statisticArray.length; i += Math.floor(statisticArray.length / 4)){
	statistic.push(statisticArray[i]);
}

statistic.push(statisticArray[statisticArray.length - 1]);

for(var i = 0; i < statistic.length; i++){
	show(statistic[i]);
}

for(var i = 0; i < statisticArray.length;i++){
	var strLog = '';
	var averSum = 0;
	for(var j = 0; j < 20; j++){
		averSum += statisticArray[i][j].fitness;
	}
	var temp = sort(statisticArray[i]);
	strLog += statisticArray[i][0].fitness + '\t ' + averSum / 20;
	console.log(strLog);
}
```

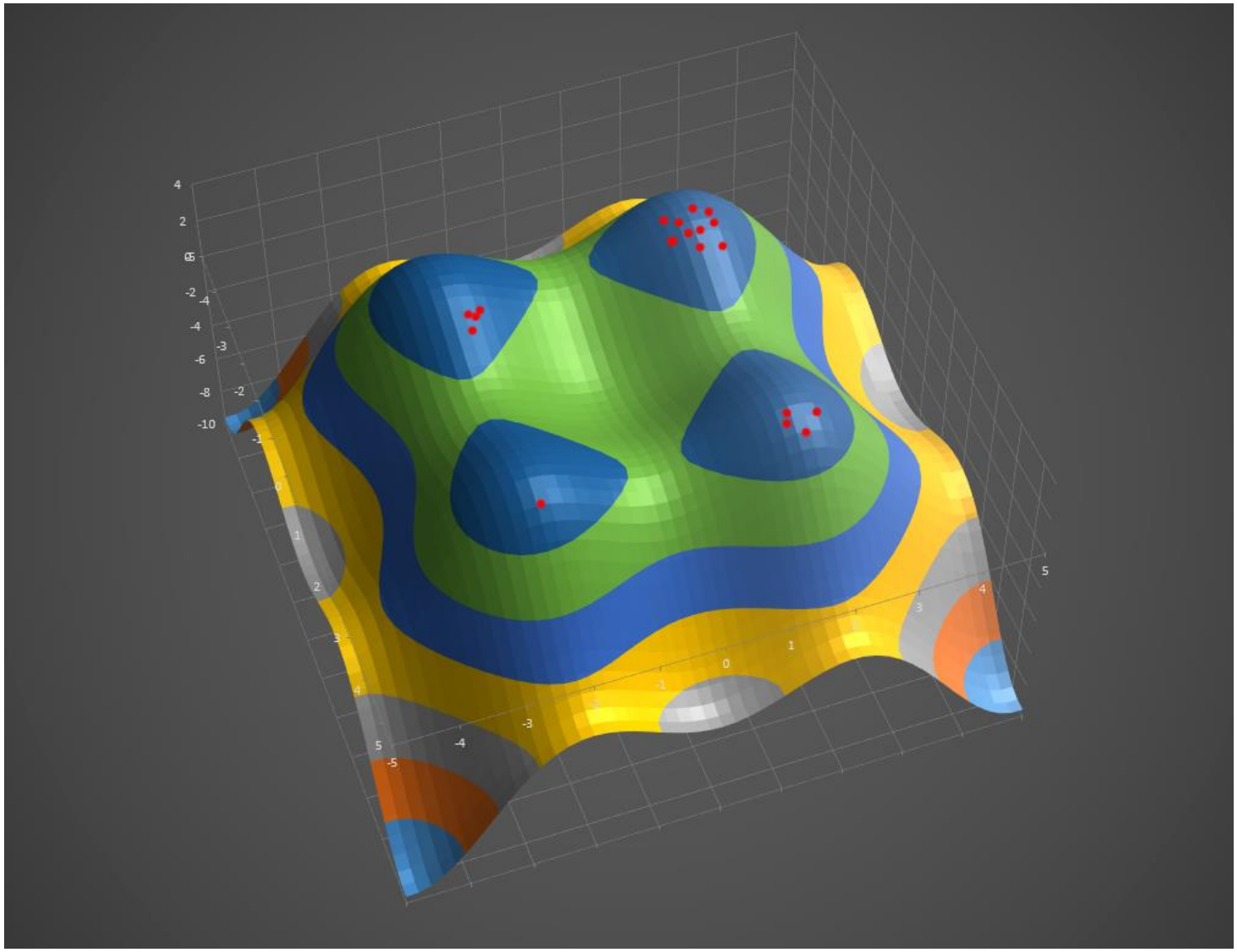
#0

#	x	y	fitness(z)
0	2,158	-2,625	3,09386
1	-1,311	2,563	2,667856
2	-1,121	-2,375	2,655958
3	1,387	2,646	2,620707
4	1,46	2,705	2,594665
5	-0,969	-2,375	2,446098
6	-1,292	2,7	2,394991
7	1,985	0,747	2,324755
8	-1,97	0,742	2,31678
9	-1,89	0,635	2,170811
10	2,249	-0,5	1,991753
11	2,273	-0,5	1,975452
12	2,078	-0,315	1,913998
13	2,151	0,122	1,813897
14	1,414	3,154	1,684596
15	-2,3	-3,152	1,682929
16	-0,508	2,598	1,591241
17	0,867	-2,82	1,55235
18	-0,891	-2,834	1,549986
19	-2,117	-3,235	1,507775



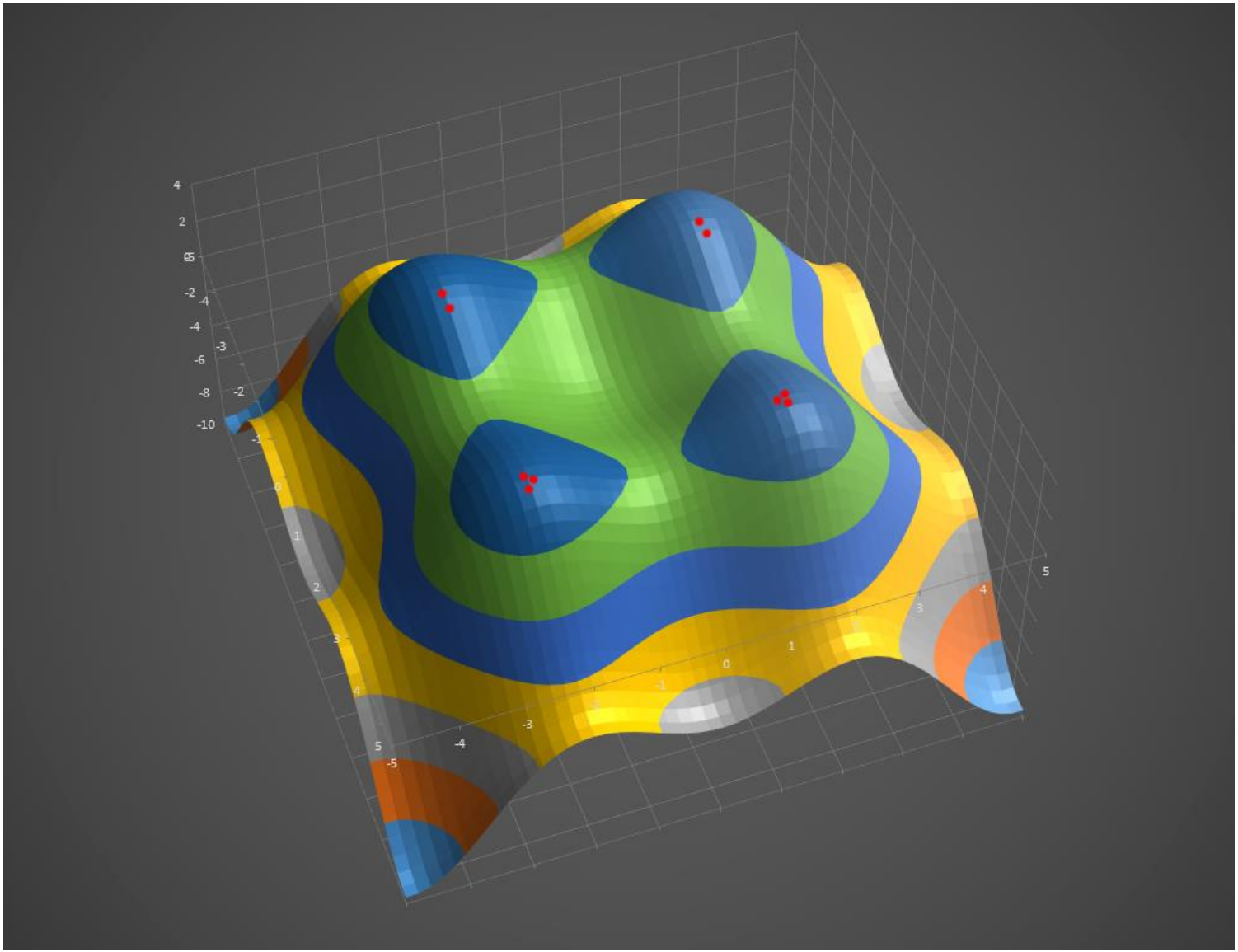
#4

#	x	y	fitness(z)
0	1,985	1,997	3,635485
1	-2,053	1,934	3,626624
2	-1,975	1,938	3,624776
3	2,063	1,919	3,62193
4	1,99	1,919	3,621486
5	-1,907	1,953	3,612258
6	-1,907	1,953	3,612258
7	-1,907	2,109	3,610921
8	1,821	2,114	3,574091
9	-2,224	-1,899	3,563877
10	-2,263	-1,958	3,555075
11	-2,253	-1,895	3,544593
12	-2,327	-1,938	3,501978
13	2,253	-1,797	3,499625
14	-2,327	-1,895	3,48918
15	-2,371	2,061	3,470271
16	-2,38	2,041	3,461497
17	-2,38	2,061	3,460325
18	-2,38	2,061	3,460325
19	2,38	-2,09	3,45659



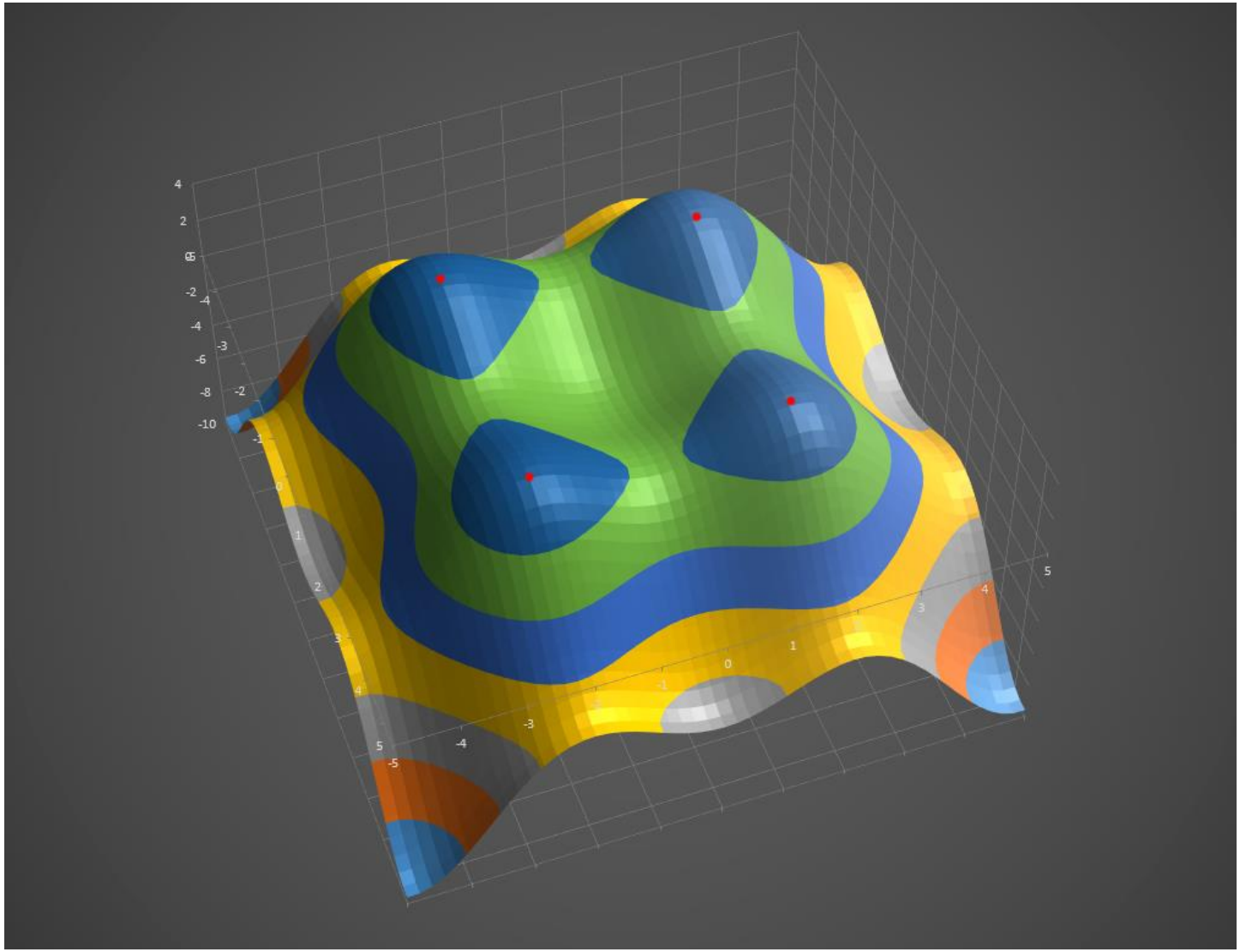
#8

#	x	y	fitness(z)
0	-2,024	2,017	3,639181
1	2,004	2,017	3,638414
2	2,004	-2,017	3,638414
3	-2,004	-1,997	3,637266
4	2,004	-1,997	3,637266
5	2,004	-1,997	3,637266
6	1,99	-2,017	3,637173
7	1,99	-2,017	3,637173
8	-1,985	-2,026	3,636825
9	1,985	2,017	3,636633
10	1,985	2,017	3,636633
11	1,985	2,017	3,636633
12	1,985	2,017	3,636633
13	1,985	2,017	3,636633
14	1,985	2,017	3,636633
15	1,985	2,017	3,636633
16	1,985	2,017	3,636633
17	-1,985	2,017	3,636633
18	1,985	2,017	3,636633
19	-1,985	2,017	3,636633



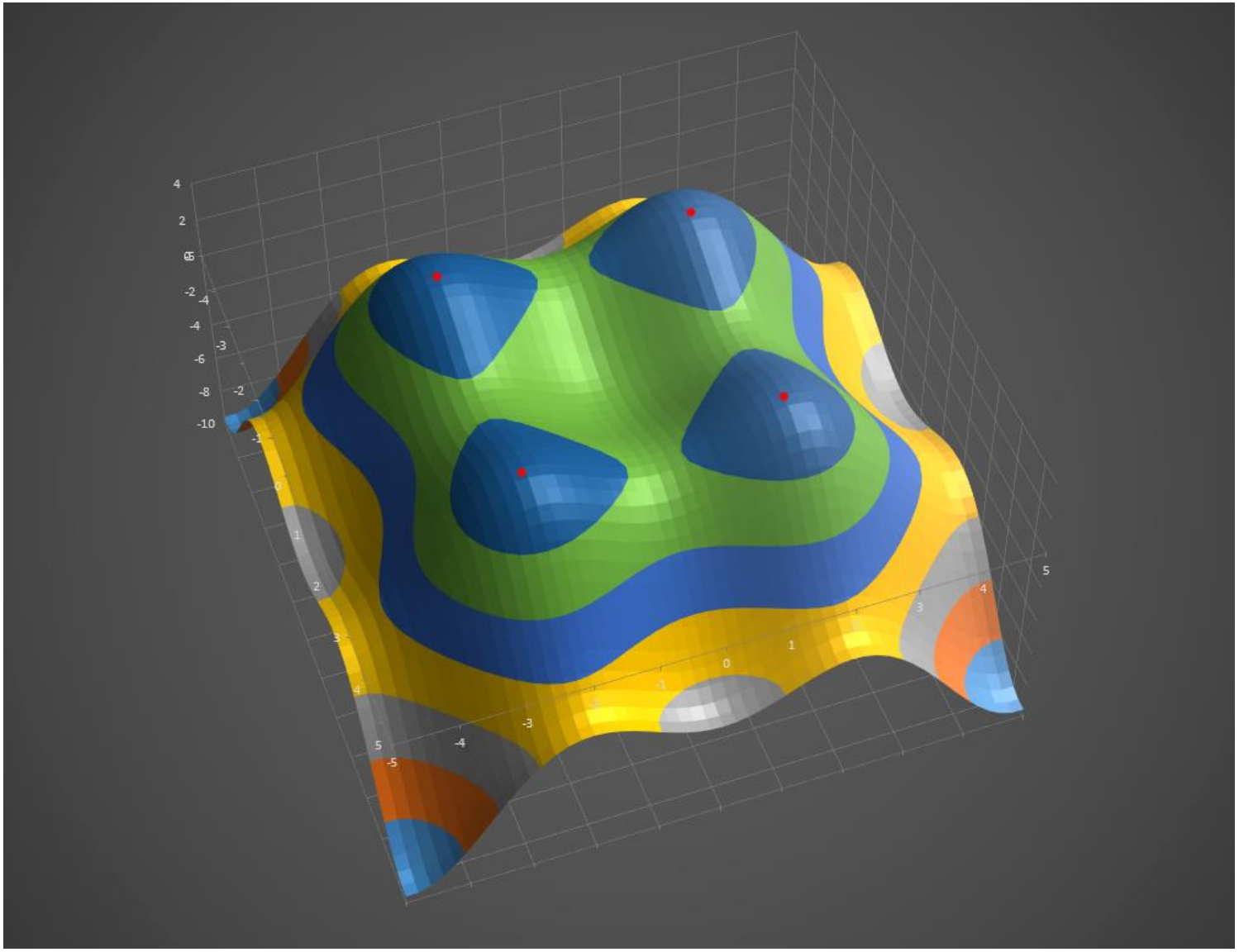
#12

#	x	y	fitness(z)
0	2,029	2,017	3,639212
1	-2,024	2,017	3,639181
2	-2,024	2,017	3,639181
3	-2,024	2,017	3,639181
4	2,024	-2,017	3,639181
5	2,024	-2,017	3,639181
6	-2,024	-2,017	3,639181
7	2,024	-2,017	3,639181
8	-2,024	-2,017	3,639181
9	2,024	-2,017	3,639181
10	-2,024	2,017	3,639181
11	-2,024	2,017	3,639181
12	2,024	2,017	3,639181
13	-2,024	2,017	3,639181
14	-2,024	2,017	3,639181
15	-2,024	2,017	3,639181
16	-2,024	2,017	3,639181
17	-2,024	2,017	3,639181
18	-2,024	2,017	3,639181
19	-2,024	2,017	3,639181



#16

#	x	y	fitness(z)
0	-2,024	2,026	3,639372
1	2,024	2,026	3,639372
2	2,024	2,026	3,639372
3	2,029	-2,019	3,639284
4	2,029	-2,019	3,639284
5	2,029	2,019	3,639284
6	2,029	2,019	3,639284
7	-2,024	2,019	3,639253
8	-2,024	2,019	3,639253
9	-2,024	-2,019	3,639253
10	-2,024	-2,019	3,639253
11	-2,024	-2,019	3,639253
12	2,024	-2,019	3,639253
13	2,024	-2,019	3,639253
14	-2,024	-2,019	3,639253
15	2,024	-2,019	3,639253
16	2,024	-2,019	3,639253
17	2,024	-2,019	3,639253
18	-2,024	-2,019	3,639253
19	2,029	2,017	3,639212



maximum	average
3,0939	2,1275
3,6355	2,6559
3,6355	3,1573
3,6355	3,3757
3,6355	3,5502
3,6355	3,6215
3,6384	3,6316
3,6384	3,6359
3,6392	3,6371
3,6392	3,6375
3,6392	3,6380
3,6392	3,6390
3,6392	3,6392
3,6393	3,6392
3,6394	3,6392
3,6394	3,6393