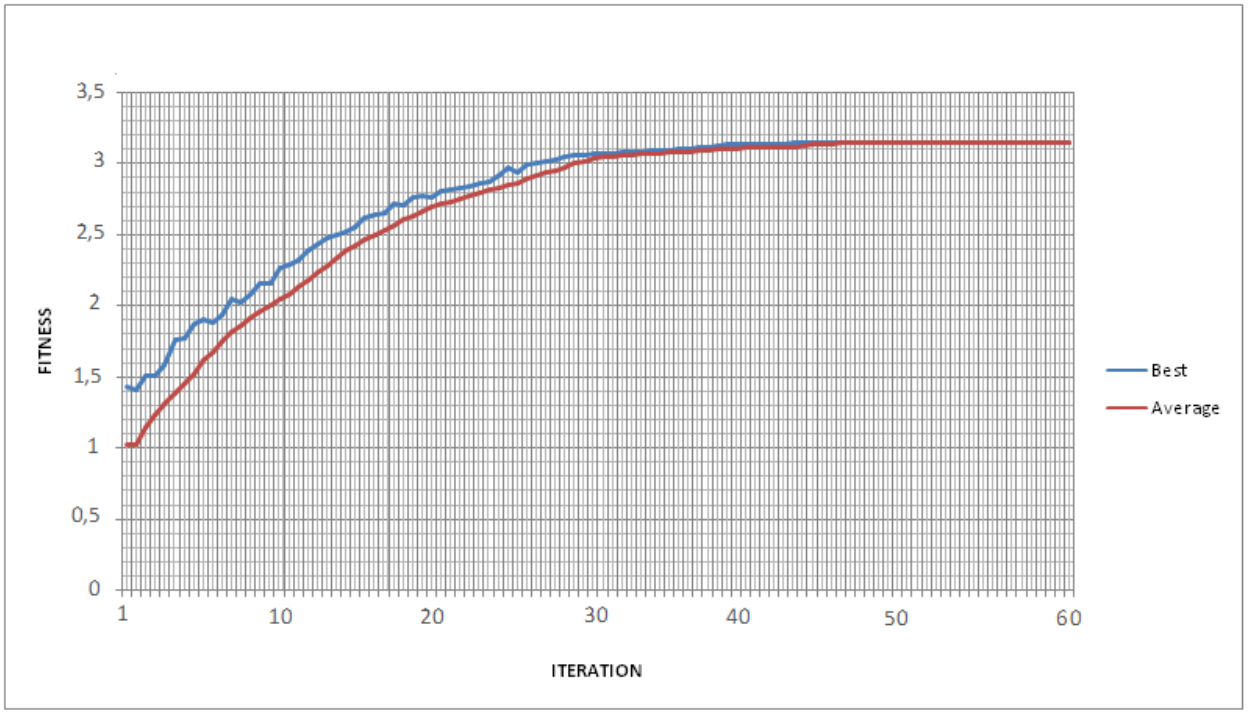
Task 3
Student –Polina Rachkovskaya

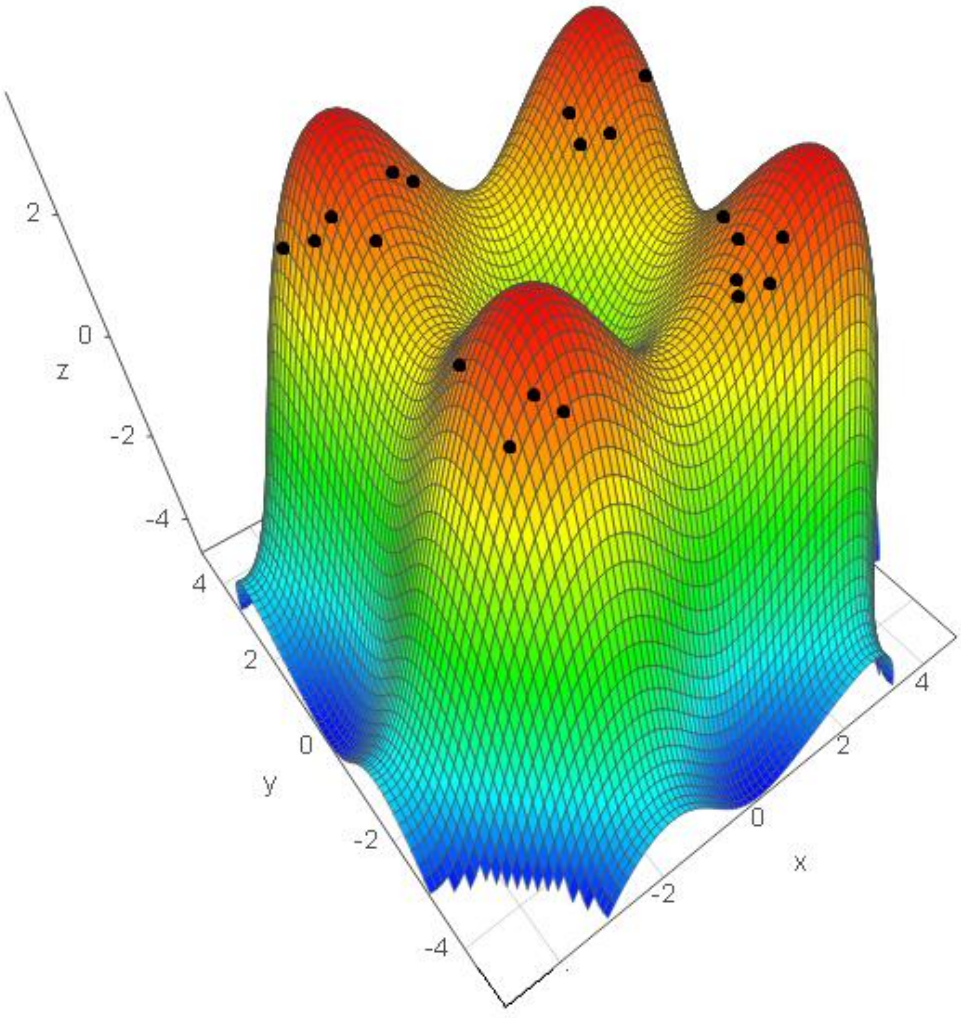
Algorithm:

1. Maximize 3D version of Schwefel's function;
2. Create a population of 20 chromosome with 22 binary genes;
3. Evolve this population till fitness of chromosomes doesn't change, using fitness sharing algorithm;

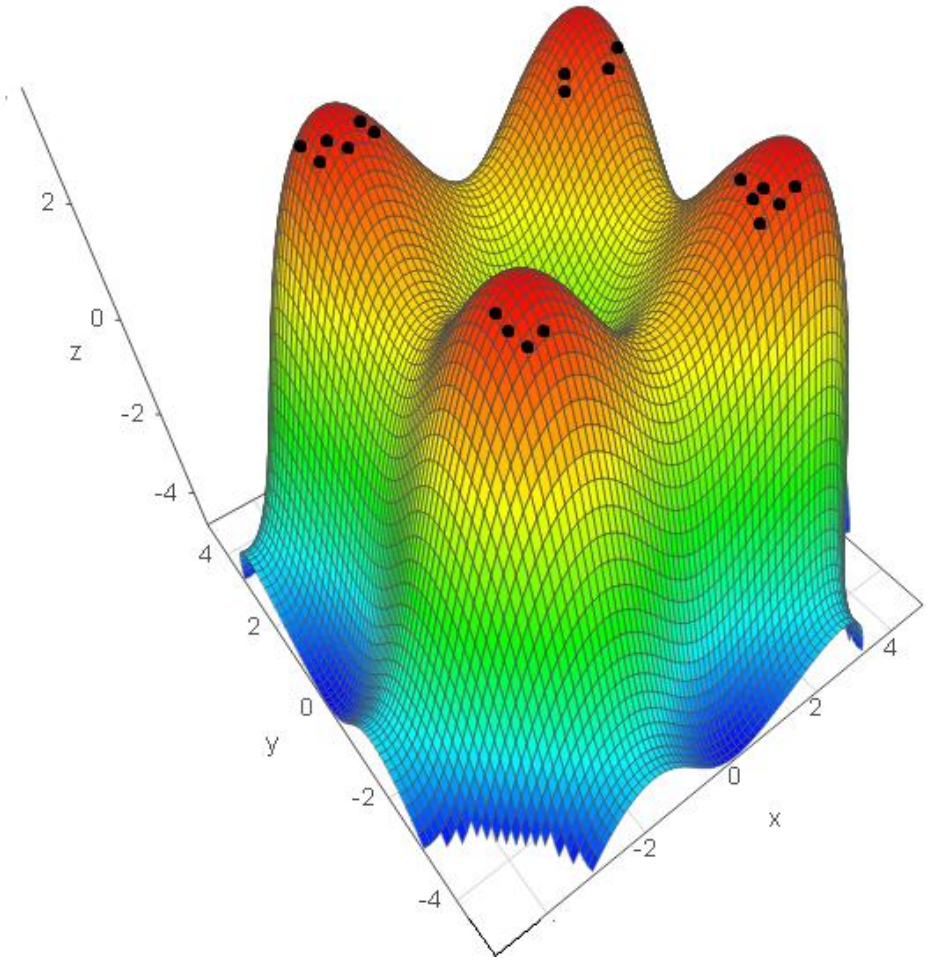
Results:



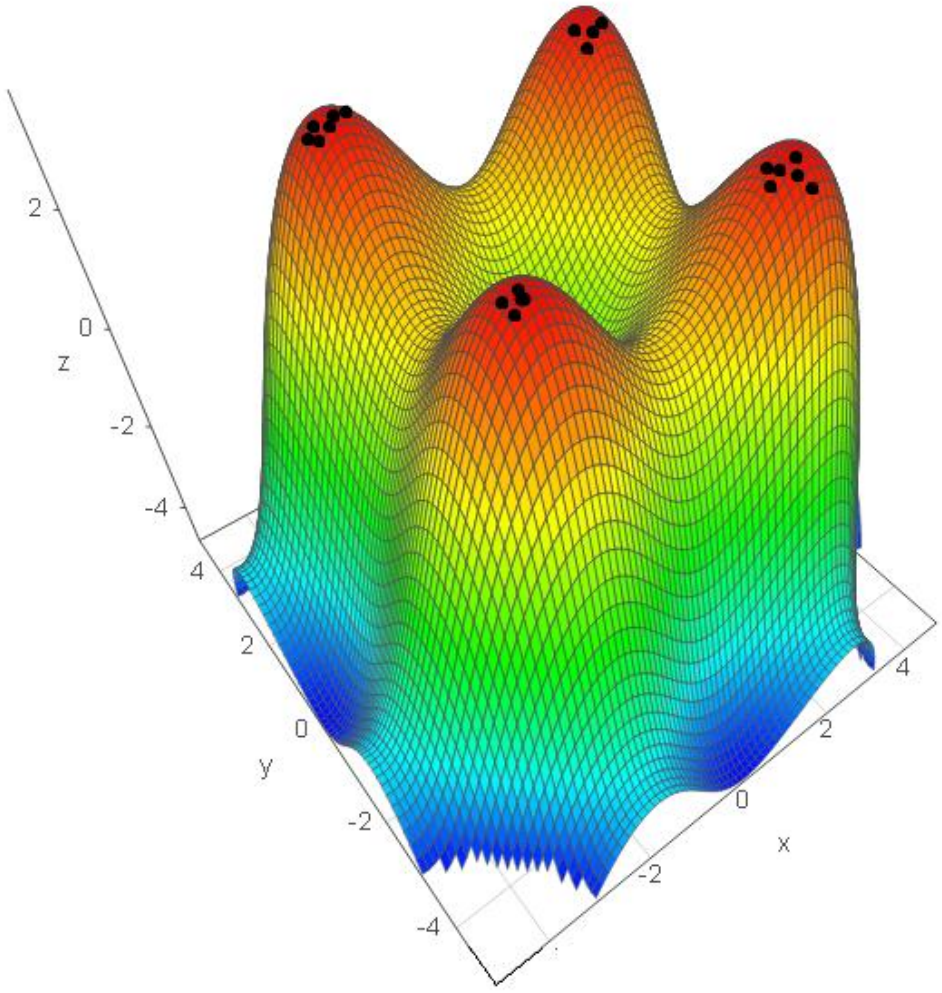
Iteration 1:



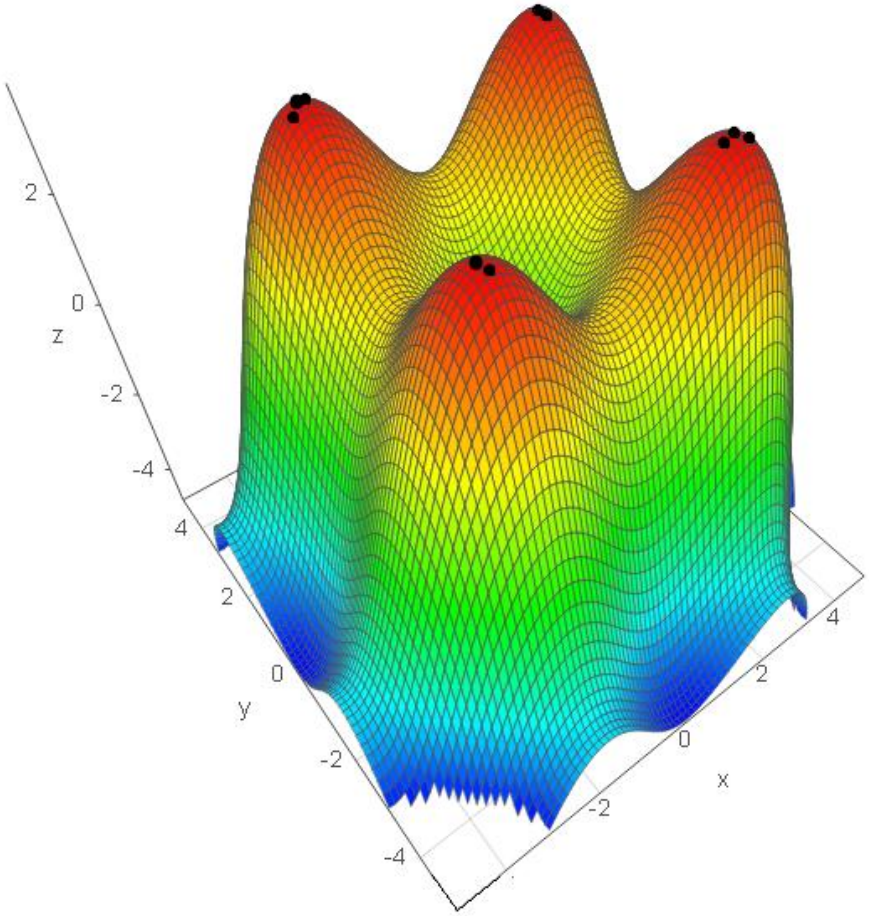
Iteration 15:



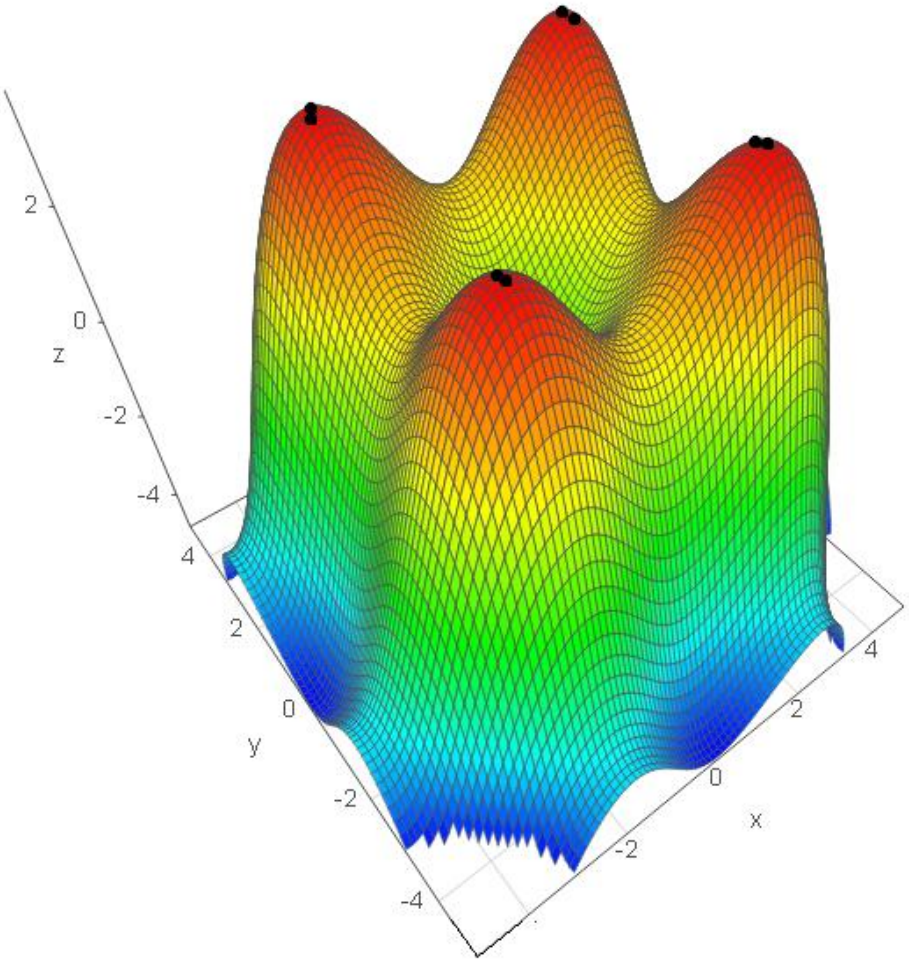
Iteration 30:



Iteration 45:



Iteration 60:



Code:

```
def get_average_value(generation):
av_list = [ ]
for i in generation:
sum = fitness_function(i)
av_list.append(sum)
average = reduce(lambda x, y: x + y, av_list) / len(av_list)
max_value = max(av_list)
return (average, max_value)
def fitness_function(i):
sum = 0
for j in i:
sum += j * sin(abs(j))
return sum
def main():
generation = [ ]
for i in range(0, 20):
hromosome = [ ]
for j in range(0, 20):
hromosome.append(random.uniform(-5, 5))
generation.append(hromosome)
max_value = get_average_value(generation)[1]
print(max_value)
count = 0
while count < 10:
generation.sort(key=lambda x: fitness_function(x), reverse=False)
fathers = generation[0:10]
new_generation = []
for i in range(0, 10):
mother = random.randrange(0, 10)
father = random.randrange(0, 10)
rand_index = random.randrange(0, 20)
first = fathers[mother][0:rand_index]
first.extend(fathers[father][rand_index:20])
second = fathers[father][0:rand_index]
second.extend(fathers[mother][rand_index:20])
new_generation.append(first)
new_generation.append(second)
new_generation.extend(fathers)
new_max = get_average_value(new_generation)
print(new_max)
if new_max[1] < max_value:
max_value = new_max[1]
count = 0
else:
count += 1
generation = new_generation
if __name__ == '__main__':
main()
function sharedFitness(generation){
var mem generation = generation;
for(var i = 0; i < 20; i++) {
for(var j = 0; j < 20; j++) {
var x = getX(generation [i]) -getX(generation [j]);
var y = getY(generation [i]) -getY(generation [j]);}
var d = Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2));
generation [i].fitness += getS(d);}
return generation;}
function mutation(population){
return population.map(function(chromosome) {
return chromosome.map(function(gene) {
return Math.random() < 0.02 ? (gene + 1) % 2 : gene;});
});}
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;}
```