Maximization of 3D Schwefel function z=x*sin(x)+y*sin(y), where x>=-5,y>=-5 20 chrosomes with 22 binary genes by Fitness Sharing algorithm.

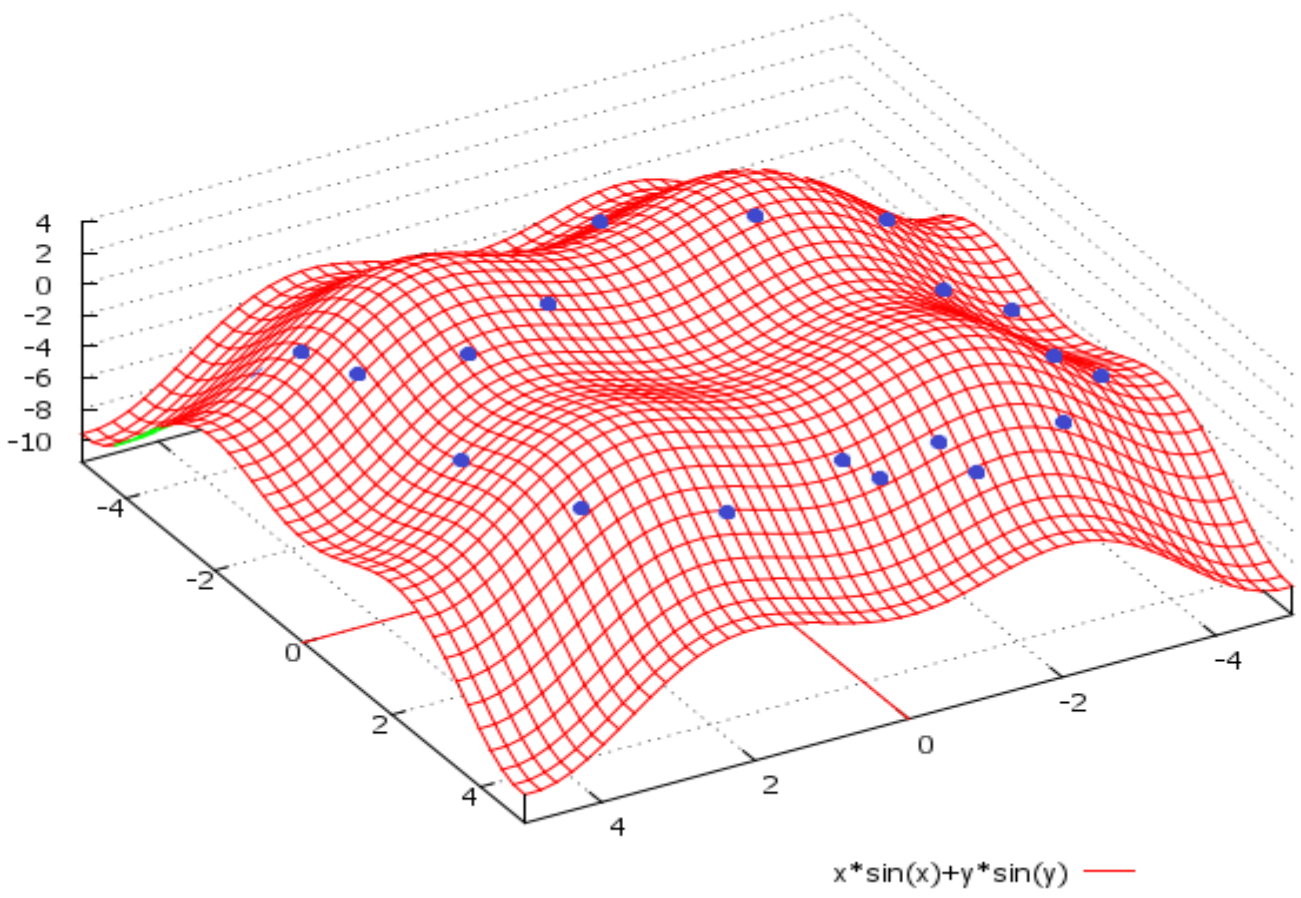
CODE:

```
int _tmain(int argc, _TCHAR* argv[])
{
        srand(time(NULL));
        countChromosome = 20;
        countGenes = 22;

        int generation = [];
        getAverageValue(generation);
        avList = [];
        for (count = 1; 0 < generation; count++;)
                sum = fitnessFunction(i);
        avList.append(sum);
        average = reduce(lambda x, y, x + y, avList) / len(avList);
        min_value = max(avList);
        return  min_value;


        Sfunction(chromosome_one, chromosomeTwo, sigma);
        distance = sqrt((getResX(chromosome_one) - getResX(chromosomeTwo)) * 2 + (getResY(chromosome_one) –
getResY(chromosomeTwo)) * 2)
        if distance > sigma then
                return 0
        else
        return 1 - (distance / sigma);


fitnessFunction(i);
return getResX(i) * sin(abs(getResX(i))) + getResY(i) * sin(abs(getResY(i)));


        shared_fitnessFunction(chromosome, generation);
sum = 0;
radius = 1;
for (i = 0; 0 < countChromosome; i++;)
if generation[it] == chromosome then
        sum += Sfunction(chromosome, generation[it], radius);
return double(fitnessFunction(chromosome)) / sum;


getResX(i);
sum = 0;
```

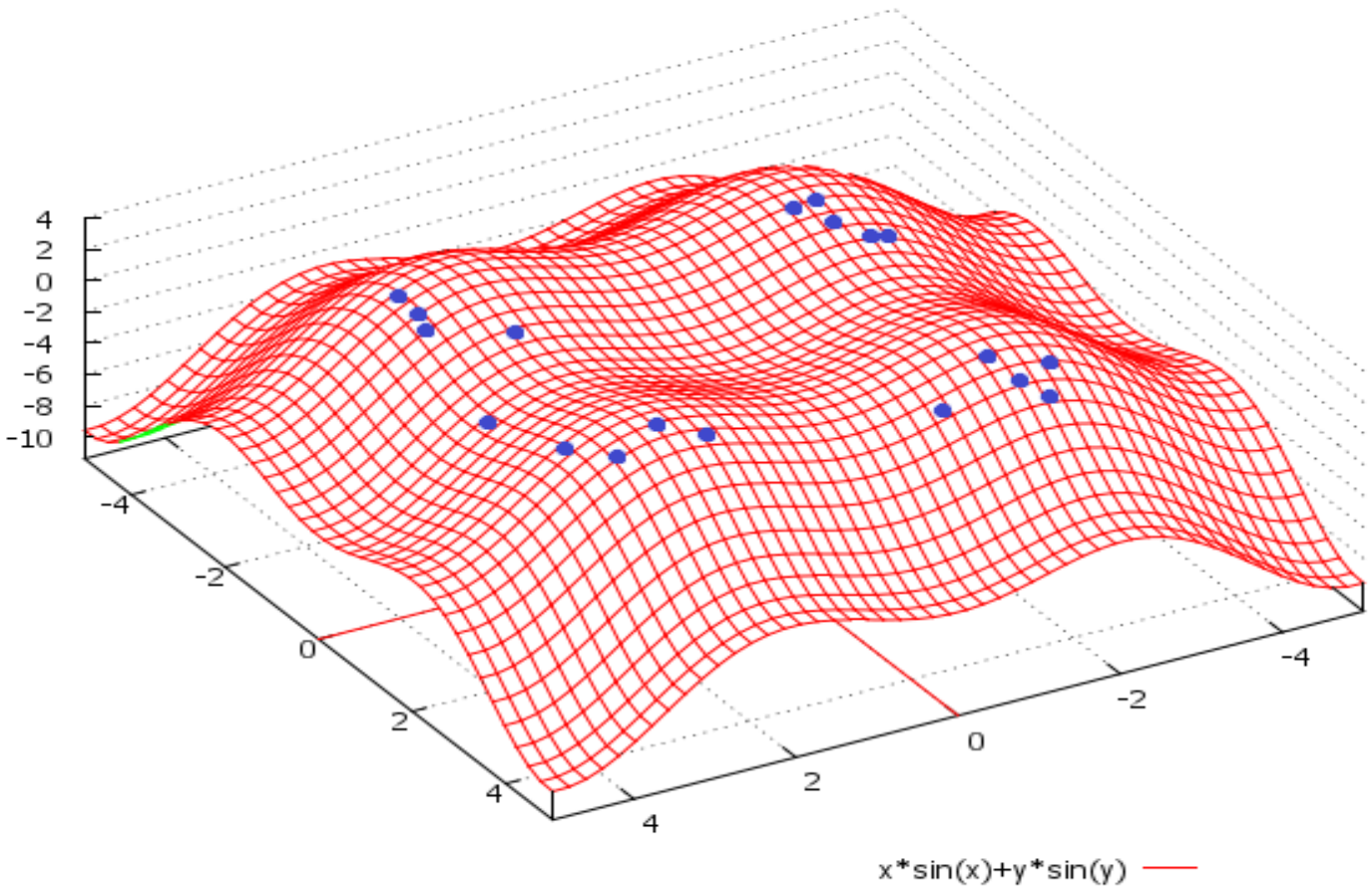
```
for (count = 1; 0 < 11; count++;)
if i[count] == 1 :
sum += 2 * count
if i[0] == 1 then
        sum = sum*(-1);
return float(sum) / 1023 * 5;



getResY(i);
sum = 0;
for (count = 0; 0 < countGenes - 10; count++;)
if i[count] == 1 then
        sum = sum + (2 * count);
if i[11] == 1;
sum = sum*(-1);
return float(sum) / 1023 * 5;

for (i = 1; 0 < 11; i++;)
        chromosome = []

        for (j = 0; j < countGenes; j++;)
                chromosome.append(random.getrandbits(1));
        generation.append(chromosome);

        max_value = getAverageValue(generation);
        cout << max_value;
        count = 0;
        Xs = [];
        MAXs = [];
        while count < 10 then
                Xs.append((map(lambda x, getResX(x), generation)));
        copy_generation = generation[, ];
        generation.sort(key = lambda x, shared_fitnessFunction(x, copy_generation), reverse = False);
        NewGenerat = uniform_crossover(generation);

        new_max = getAverageValue(NewGenerat);
        MAXs.append(new_max);
        if new_max[0] > max_value then
                max_value = new_max[0];
        count = 0;
        else
                count += 1
                generation = NewGenerat;
        cout << Xs[0];
        cout << Xs[2];
        cout << Xs[len(Xs) - 1];
        cout << MAXs;
        cout << "------------------";
```

```
cout << generation[0];



uniform_crossover(generation);
halfChrmsm = countChromosome / 2;
old = generation[0, halfChrmsm]
        NewGenerat = [];
for i in range(0, halfChrmsm) :
        mother = random(0, halfChrmsm);
father = random(0, halfChrmsm);
one = [];
two = [];
for (range = 0; range < countGenes; range++)
        rand_flag = random(1);
if rand_flag == 1  then
        one.extend(old[father][it]);
two.extend(old[mother][it]);
else
        two.extend(old[father][it]);
one.extend(old[mother][it]);
NewGenerat.append(one);
NewGenerat.append(two);
NewGenerat.extend(old);
return NewGenerat;

}
```

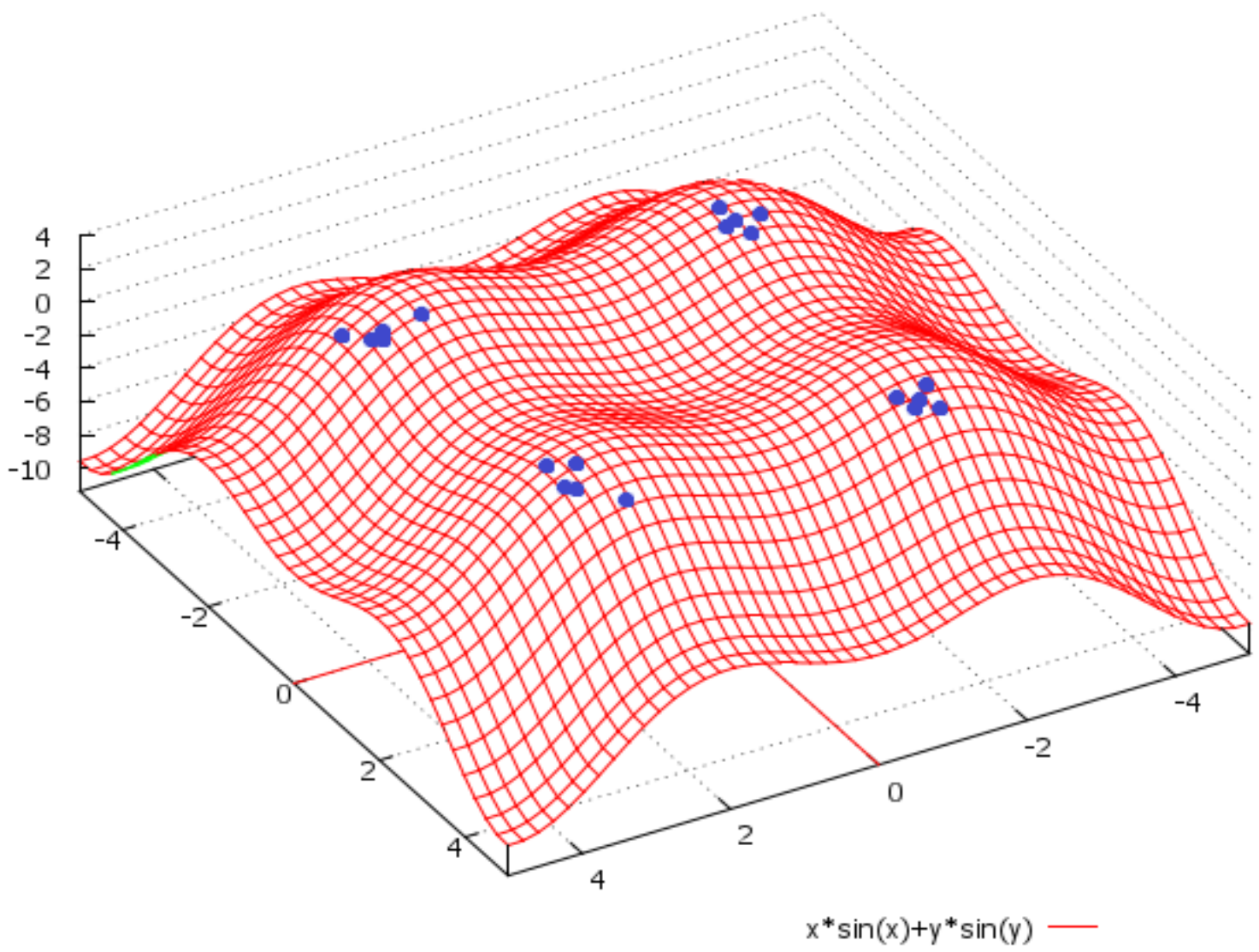
1st iteration.



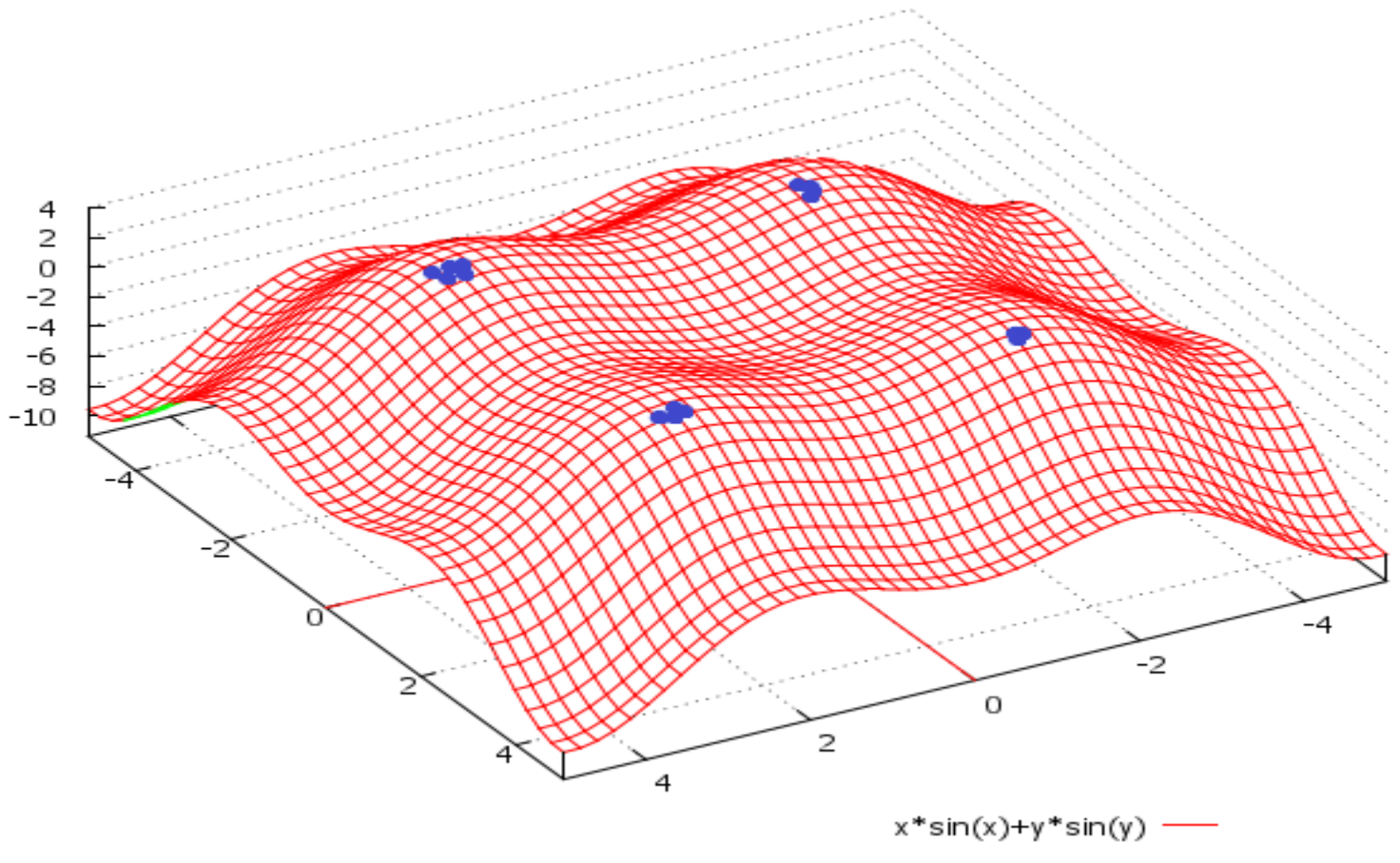
6st iteration.



12st iteration.



19st iteration.



26st iteration.