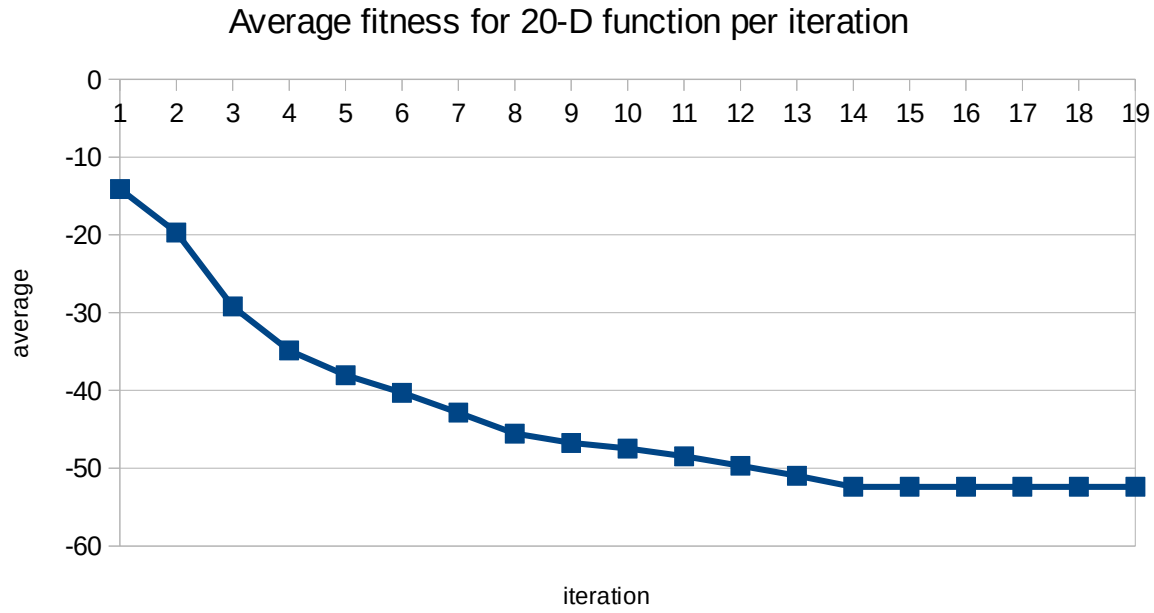
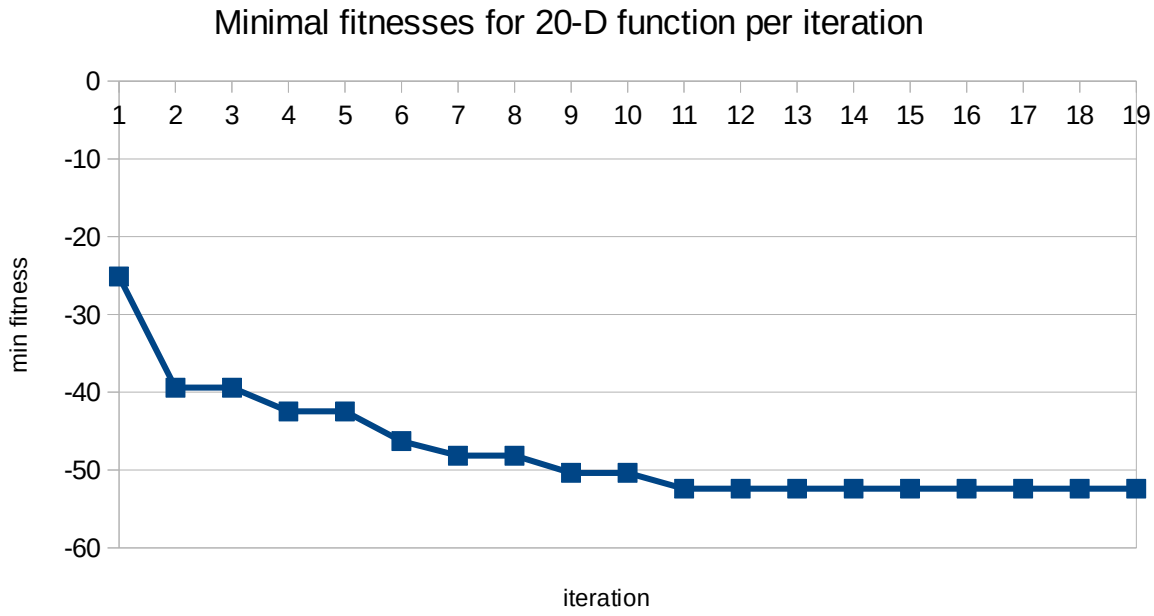
Task 1. 20-Dimensional Schwefel“s function





Source code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import random
import math

def create_population(gens, chromos):
    population = []
    for chrom in xrange(chromos):
        population.append([])
        for gen in xrange(gens):
            population[chrom].append(random.uniform(-5, 5))
    return population
```

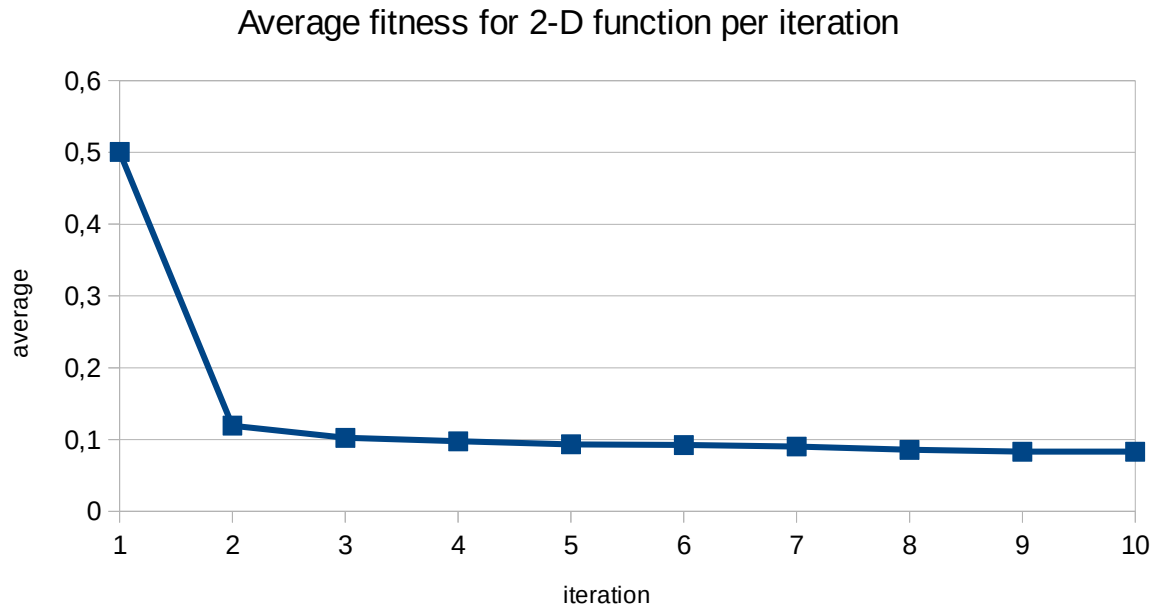
```
def calc_fitness(arr):
    y = 0
    for x in arr:
        y += x * math.sin(x)
    return y

def find_best_chromos(population):
    list = []
    for chrom in population:
        list.append((calc_fitness(chrom), chrom))
    buble_sort(list)
    r = []
    for item in list[len(population)/2:]:
        r.append(item[1])
    return r

def create_childs(pop):
    size = len(pop)
    par1, par2 = random.randint(0, size-1), random.randint(0, size-1)
    cut_point = random.randint(0, len(pop[0]))
    child1 = pop[par1][:cut_point] + pop[par2][cut_point:]
    child2 = pop[par2][:cut_point] + pop[par1][cut_point:]
    return child1, child2

def buble_sort(A):
    for i in range(len(A)):
        for k in range(len(A) - 1, i, -1):
            if A[k][0] > A[k - 1][0]:
                swap(A, k, k – 1)

def swap(A, x, y):
    tmp = A[x]
    A[x] = A[y]
    A[y] = tmp

def get_avg_count(population):
    chromos = len(population)
    average_list = []
    for chrom in xrange(chromos):
        cur_counter = calc_fitness(population[chrom])
        average_list.append(cur_counter)
    return sum(average_list) / len(average_list)

def minimum(population):
    min = population[0][0]
    for chrom in population:
        fintess = calc_fitness(chrom)
        if min > fintess:
            min = fintess
    return min

def main():
    population = create_population(20, 20)
    print(population)
    for i in xrange(50):
        best_chromos = find_best_chromos(population)
        # print("best", best_chromos)
        population = create_new_generation(best_chromos)
        # print("new gen", population)
```

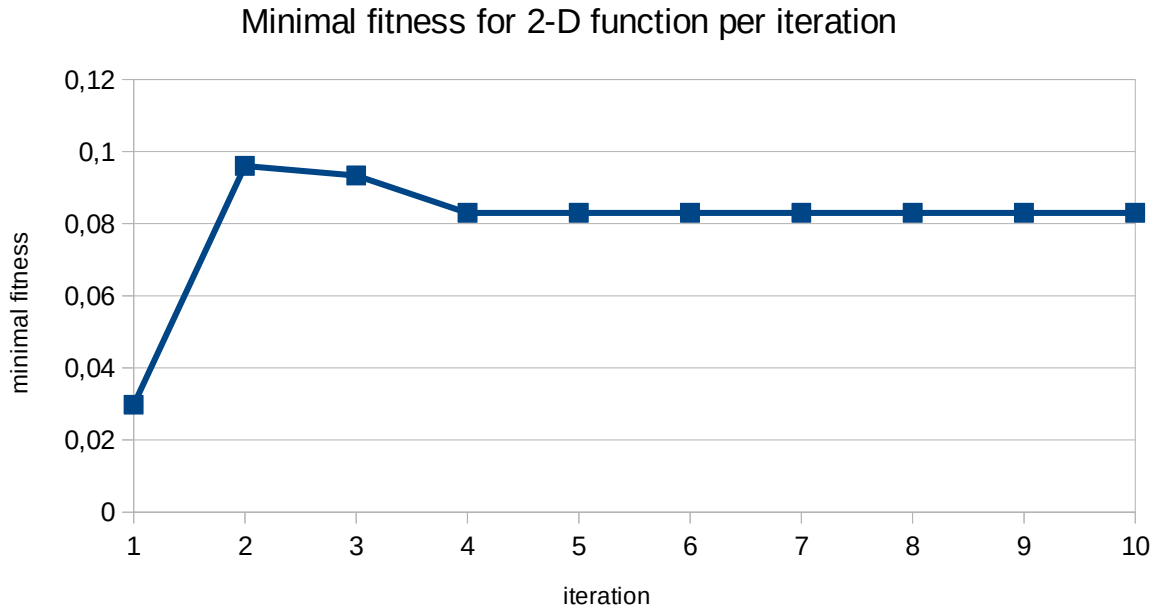
```
        avg = get_avg_count(population)
        min = minimum(population)
        print(str(i) + ";" + str(avg) + ";" + str(min))

def create_new_generation(best):
    new_generation = []
    for iter in xrange(len(best)):
        ch1, ch2 = create_childs(best)
        new_generation.append(ch1)
        new_generation.append(ch2)
    return new_generation
if __name__ == "__main__":
    main()
```

Result in table:

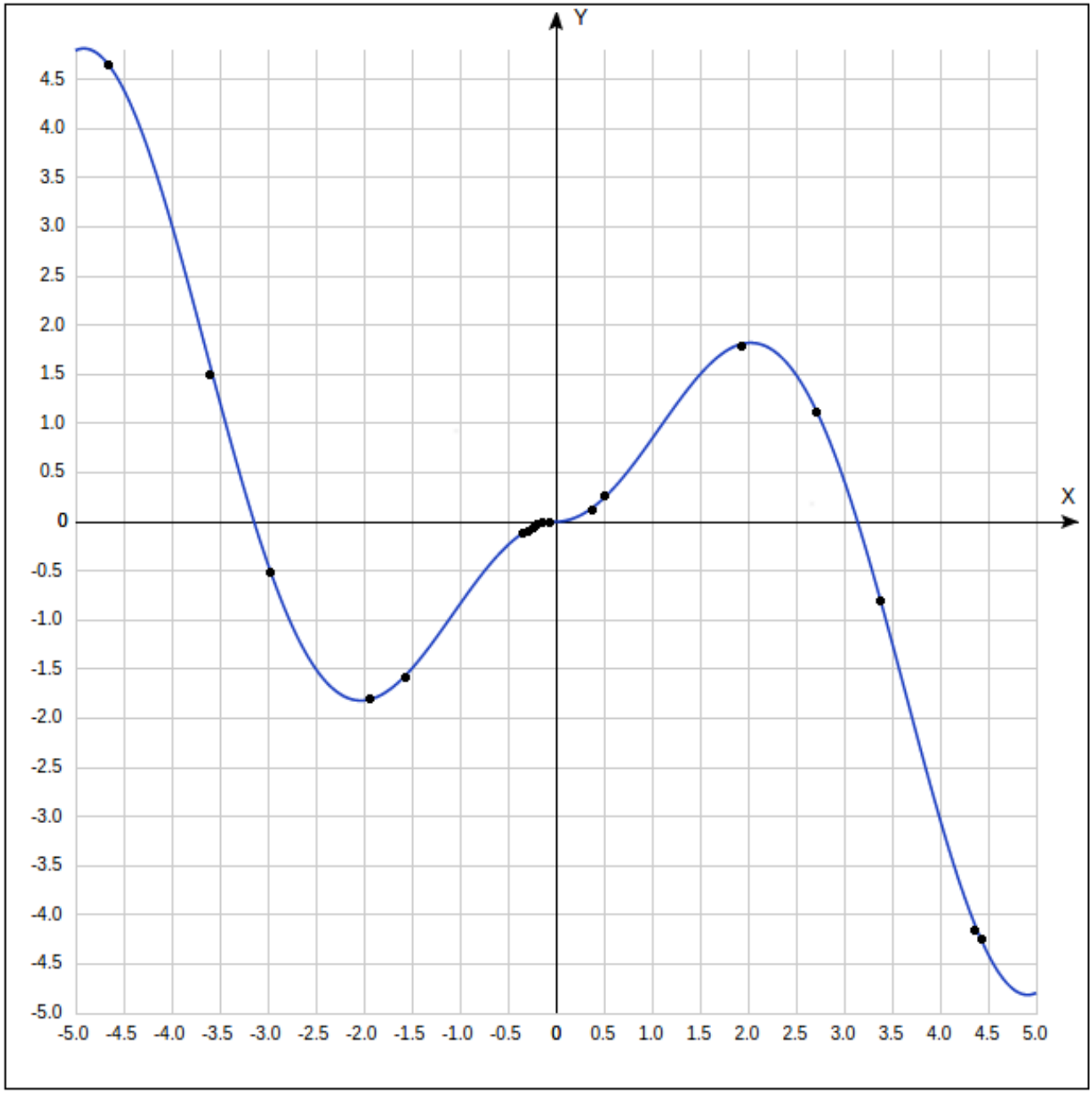
iteration	average	min
0	-14,1016524567	-25,1226466138
1	-19,6957604016	-39,4232909658
2	-29,2065585874	-39,4232909658
3	-34,8704761791	-42,4736244313
4	-38,039531003	-42,4736244313
5	-40,3064988617	-46,285814026
6	-42,8540607072	-48,1565795975
7	-45,561492543	-48,1565795975
8	-46,7535054189	-50,3749093639
9	-47,4695262449	-50,3749093639
10	-48,4729471689	-52,404269244
11	-49,7001876782	-52,404269244
12	-50,9866503766	-52,404269244
13	-52,404269244	-52,404269244
14	-52,404269244	-52,404269244

Task 2. 2-D version of Schwefels function

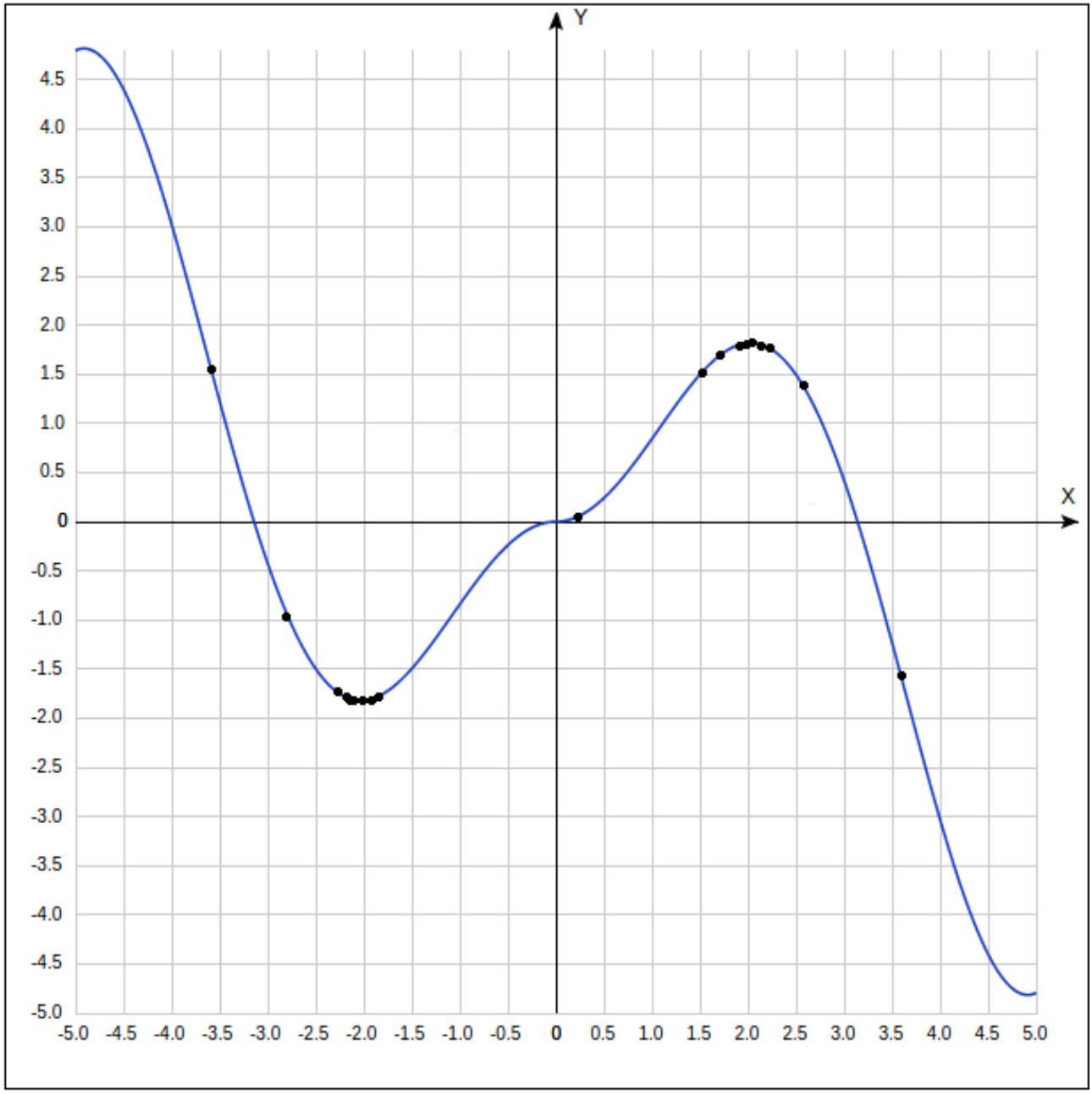




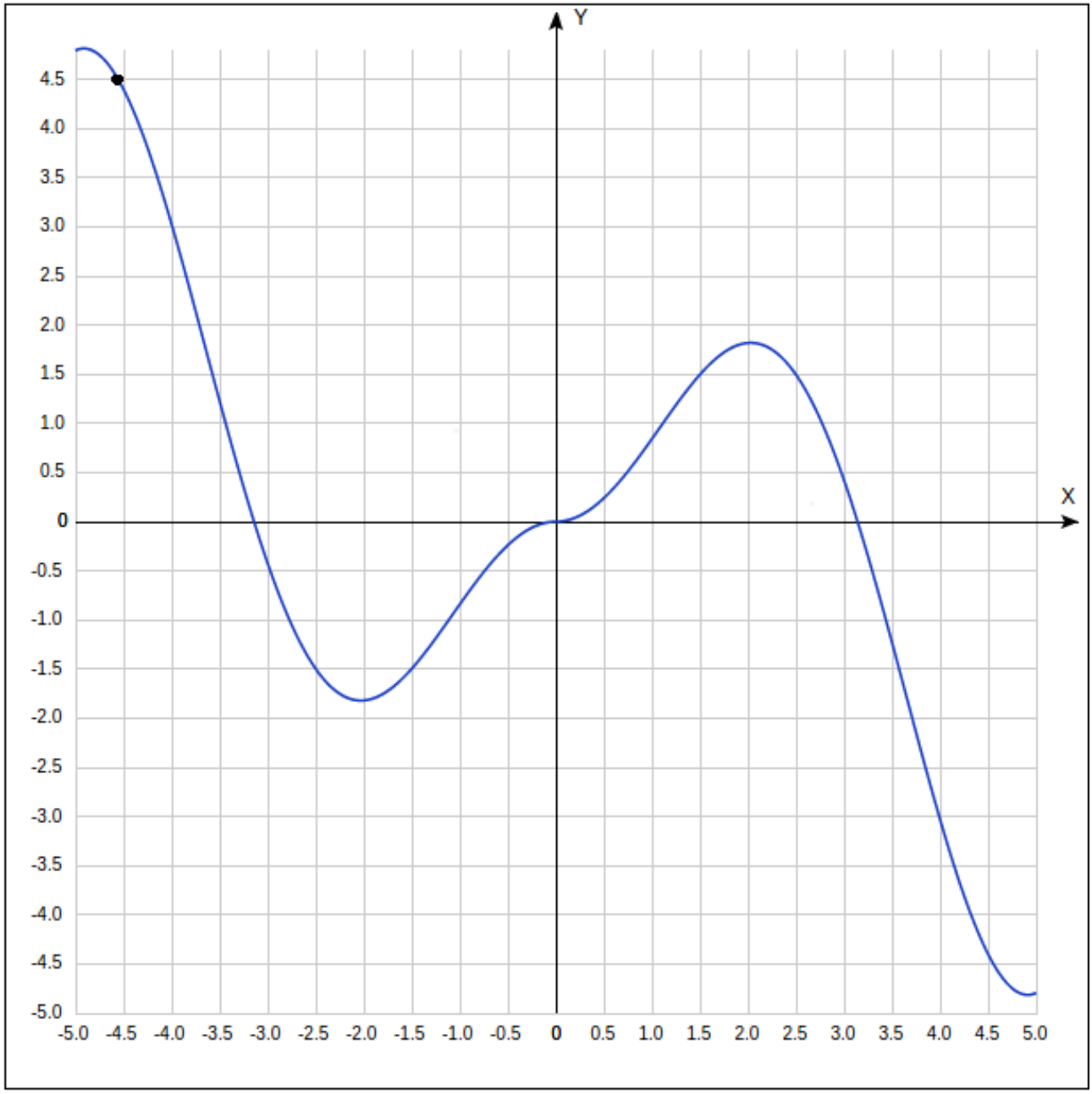
Graphics of First iteration:


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

Intermidiate iteration:



Final iteration:



Source code:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import random
import math
def create_population(gens, chromos):
    population = []
    for chrom in xrange(chromos):
        population.append([])
        for gen in xrange(gens):
            population[chrom].append(random.randint(0, 1))
    return population

def calc_fitness(arr):
```

```
    x = convert_x_from_binary_in_range(arr)
    return x * math.sin(x) * 5

def convert_x_from_binary_in_range(arr):
    binary = ""
    for bit in arr:
        binary += str(bit)
    x = int(binary, 2) / 1023.
    if arr[0] == 0:
        x *= -1
    return x

def find_best_chromos(population):
    list = []
    for chrom in population:
        list.append((calc_fitness(chrom), chrom))
    buble_sort(list)
    r = []
    for item in list[len(population)/2:]:
        r.append(item[1])
    return r

def create_childs(pop):
    size = len(pop)
    par1, par2 = random.randint(0, size-1), random.randint(0, size-1)
    cut_point = random.randint(0, len(pop[0]))
    child1 = pop[par1][:cut_point] + pop[par2][cut_point:]
    child2 = pop[par2][:cut_point] + pop[par1][cut_point:]
    return child1, child2

def buble_sort(A):
    for i in range(len(A)):
        for k in range(len(A) - 1, i, -1):
            if A[k][0] > A[k - 1][0]:
                swap(A, k, k – 1)

def swap(A, x, y):
    tmp = A[x]
    A[x] = A[y]
    A[y] = tmp

def get_avg_count(population):
    chromos = len(population)
    average_list = []
    for chrom in xrange(chromos):
        fitness = calc_fitness(population[chrom])
        average_list.append(fitness)
    return sum(average_list) / len(average_list)

def minimum(population):
    min = 9999.
    for chrom in population:
        fintess = calc_fitness(chrom)
        if min > fintess:
            min = fintess
    return min

def create_new_generation(best):
    new_generation = []
    for iter in xrange(len(best)):
```

```
        ch1, ch2 = create_childs(best)
        new_generation.append(ch1)
        new_generation.append(ch2)
    return new_generation

def print_point(population):
    for point in population:
        print(convert_x_from_binary_in_range(point))

def main():
    population = create_population(10, 20)
    print(population)
    for i in xrange(50):
        best_chromos = find_best_chromos(population)
        # print("best", best_chromos)
        population = create_new_generation(best_chromos)
        # print("new gen", population)
        avg = get_avg_count(population)
        min = minimum(population)
        print(str(i) + ";" + str(avg) + ";" + str(min))
        if i in [0, 3, 7]:
            print("iteration:" + str(i))
            print_point(population)

if __name__ == "__main__":
    main()
```

Results in table:

iteration	average	min
0	0,5005124262	0,0297879886
1	0,1193686831	0,0960284459
2	0,1023779864	0,0933508639
3	0,0974393856	0,0830158103
4	0,0932357485	0,0830158103
5	0,0923173585	0,0830158103
6	0,0902503478	0,0830158103
7	0,0855995737	0,0830158103
8	0,0830158103	0,0830158103
9	0,0830158103	0,0830158103