

Министерство образования Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ
им. Н.Э. БАУМАНА

Факультет: Информатика и системы управления
Кафедра: Информационная безопасность (ИУ8)

Методы оптимизации

Домашнее задание №2 на тему:
«Исследование генетических алгоритмов в задачах
поиска экстремумов»

Вариант 5

Преподаватель:
Коннова Н.С.

Студент:
Девяткин Е.Д.

Группа:
ИУ8-34

Репозиторий работы: <https://github.com/ledibonibell/МО-hw02>

Москва 2023

Цель работы

Изучить основные принципы действия генетических алгоритмов на примере решения задач оптимизации функций двух переменных.

Постановка задачи

Найти максимум функции $f(x, y)$ в области D с помощью простого генетического алгоритма. За исходную популяцию принять 4 случайных точки. Хромосома каждой особи состоит из двух генов: значений координат x, y . В качестве потомков следует выбирать результат скрещивания лучшего решения со вторым и третьим в порядке убывания значений функции приспособленности с последующей случайной мутацией обоих генов. В качестве критерия останова эволюционного процесса задаться номером конечной популяции N : $(10^1 \dots 10^2)$. Визуализировать результаты расчетов.

Ход работы

Заданная функция:

$$f(x, y) = \frac{\sin(x) \sin(y)}{1 + x^2 + y^2}$$

Область:

$$D = (0, 2) \times (-2, 2)$$

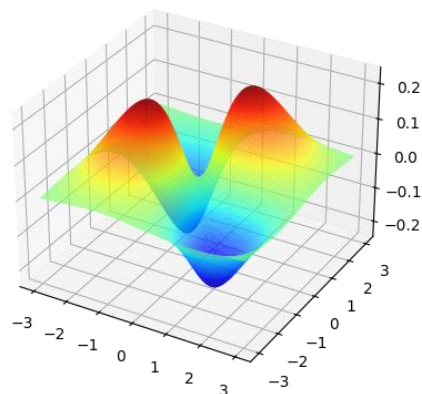
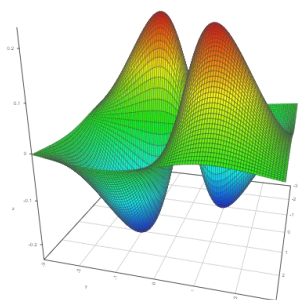
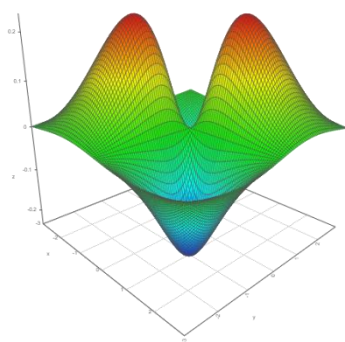


Рис. 1-3. График функции $f(x, y)$

Сгенерированные числа, а также среднее и максимальное значения FIT–функции популяции для поколений $N = 1 \dots 10$ представлены в сводной таблице 1.

№ поколения	X	Y	FIT	Максимальный результат	Средний результат
1	0.5114	0.2446	0.3593	0.3593	0.2568
	0.5017	0.2348	0.3579		
	0.5114	0.2446	0.3593		
	1.6118	-1.9427	-0.0492		
2	0.521	0.2542	0.3606	0.3606	0.2574
	0.5017	0.2348	0.3579		
	0.521	0.2542	0.3606		
	1.6118	-1.9427	-0.0492		
3	0.5305	0.2637	0.3616	0.3616	0.2579
	0.5017	0.2348	0.3579		
	0.5305	0.2637	0.3616		
	1.6118	-1.9427	-0.0492		
4	0.5399	0.2731	0.3624	0.3624	0.2583
	0.5017	0.2348	0.3579		
	0.5399	0.2731	0.3624		
	1.6118	-1.9427	-0.0492		
5	0.5492	0.2824	0.3629	0.3629	0.2586
	0.5017	0.2348	0.3579		
	0.5492	0.2824	0.3629		
	1.6118	-1.9427	-0.0492		
6	0.5585	0.2917	0.3633	0.3633	0.2588
	0.5017	0.2348	0.3579		
	0.5585	0.2917	0.3633		
	1.6118	-1.9427	-0.0492		

7	0.5676	0.3008	0.3635	0.3635	0.2589
	0.5017	0.2348	0.3579		
	0.5676	0.3008	0.3635		
	1.6118	-1.9427	-0.0492		
8	0.5766	0.3098	0.3635	0.3635	0.2589
	0.5017	0.2348	0.3579		
	0.5766	0.3098	0.3635		
	1.6118	-1.9427	-0.0492		
9	0.5856	0.3188	0.3633	0.3633	0.2588
	0.5017	0.2348	0.3579		
	0.5856	0.3188	0.3633		
	1.6118	-1.9427	-0.0492		
10	0.5856	0.3188	0.3633	0.3633	0.2588
	0.5017	0.2348	0.3579		
	0.5856	0.3188	0.3633		
	1.6118	-1.9427	-0.0492		

Таблица 1.

Пример расположения точек на первой и десятой итерации приведены на рис. 4. и рис. 5.

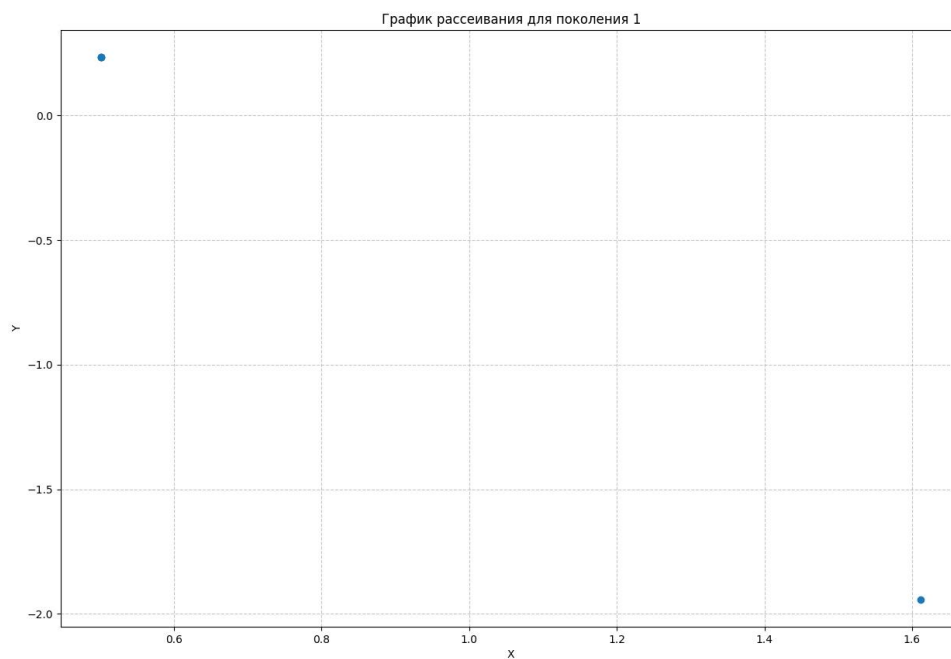


Рис. 4.

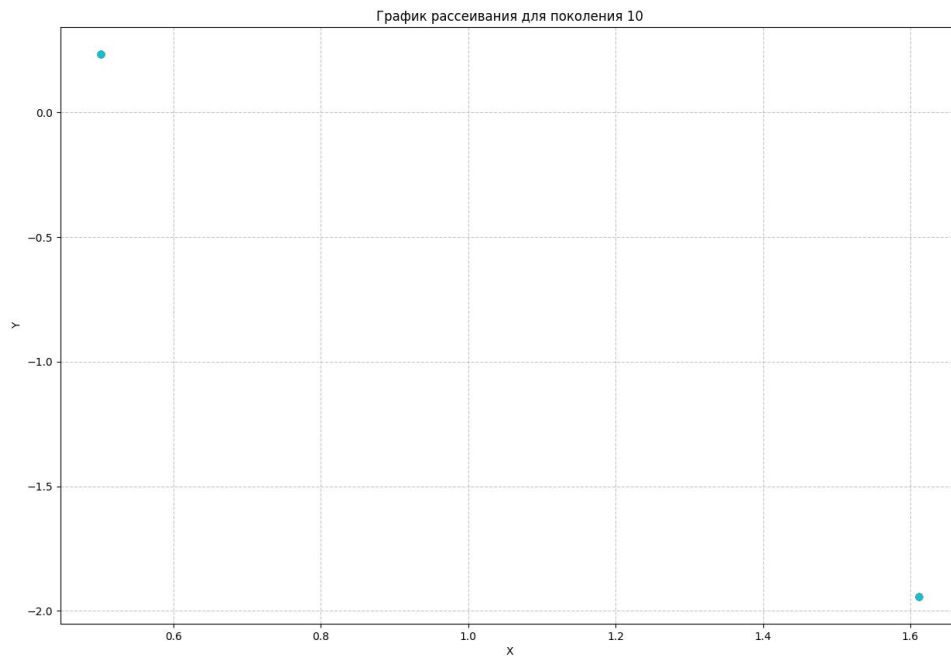


Рис. 5.

Как видно из таблицы 1, а также на рис. 6., уже к 7 - 8 поколениям происходит схождение алгоритма и хромосом всех особей в популяции (в данном случае на рис. 4 и рис. 5 не видны сильные различия лишь из-за малого коэффициента мутации), а результат вычисления приближен к искомому.

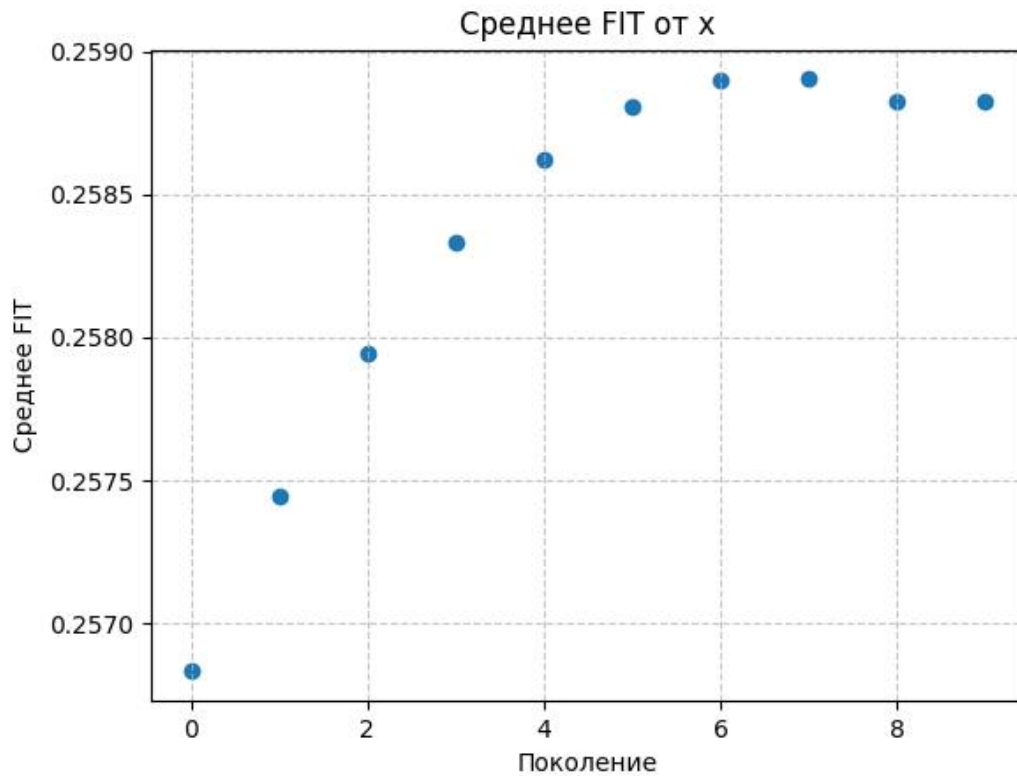


Рис. 6. График зависимости среднего значения FIT от номера популяции

Вывод

В ходе выполнения работы был изучен генетический алгоритм на примере поиска оптимального решения. Было выявлено, что уже к 10 поколению мы приблизились к оптимальному решению (при 100 существующих), что обусловлено использованием элитарного механизма селекции особей для скрещивания. Решением данной проблемы является турнирный метод селекции. Также стоит понимать, что практическое отсутствие разницы на графиках координат первых 10ти итераций обусловлено рандомизацией начальных параметров хромосом и коэффициента мутации.

Приложение А

Файл 'Main.py':

```
import time
from Function import Methods

population_size = 4
num_generations = 100
mutation_rate = 0.25

min_x = 0.0
max_x = 2.0
min_y = -2.0
max_y = 2.0
max_time = 30

population = Methods.initialize_population(population_size, min_x, max_x, min_y,
max_y)
initial_mutation_rate = mutation_rate

start_time = time.perf_counter()
generation_list = []
average_fitness_list = []

for generation in range(num_generations):
    population = Methods.assess_population(population)
    parents = Methods.choose_parents(population)

    Methods.display_results(population, generation)
    current_convergence = generation / num_generations
    mutation_rate = initial_mutation_rate * (1.0 - current_convergence)

    for j in range(0, population_size - 1, 2):
        child1 = Methods.perform_crossover(parents[j], parents[j + 1])
        child1 = Methods.apply_mutation(child1, mutation_rate, min_x, max_x, min_y,
max_y)

        population[j] = child1
        population[j].x = child1.x
        population[j].y = child1.y
        population[j].fitness = Methods.calculate_fitness(population[j].x, population[j].y)

    current_time = time.perf_counter()
    execution_time = current_time - start_time
    if execution_time >= max_time:
        print("Limit of time")
        break
```

```

if Methods.check_population_convergence(population):
    print("Fitness Convergence", generation)
    break

generation_list.append(generation)
average_fitness = sum(p.fitness for p in population) / len(population)
average_fitness_list.append(average_fitness)

Methods.draw_fitness_graph(generation_list[:10], average_fitness_list[:10])
Methods.draw_scatter_plots(population)

best_individual = max(population, key=lambda x: x.fitness)
print("Оптимальное решение: \nx =", round(best_individual.x, 4), ", y =",
      round(best_individual.y, 4), " \nf(x, y) =", round(best_individual.fitness, 4))

```

Файл 'Function.py':

```

import random
import math
import time
import matplotlib.pyplot as plt

class Organism:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.fitness = 0.0

class Methods:
    def initialize_population(population_size, min_x, max_x, min_y, max_y):
        population = []
        seed = int(time.time() * 1000)
        random.seed(seed)
        for _ in range(population_size):
            x = random.uniform(min_x, max_x)
            y = random.uniform(min_y, max_y)
            population.append(Organism(x, y))
        return population

    def calculate_fitness(x, y):
        return (math.sin(x) * math.cos(y)) / (1 + x * x + y * y)

    def assess_population(population):
        for individual in population:
            fitness = (math.sin(individual.x) * math.cos(individual.y)) / (1 + individual.x

```



```

* individual.x + individual.y * individual.y)
    individual.fitness = fitness
    return population

def choose_parents(population):
    parents = []
    tournament_size = 3
    for _ in range(len(population)):
        best_parent = None
        best_fitness = -float('inf')
        for _ in range(tournament_size):
            random_index = random.randint(0, len(population) - 1)
            candidate = population[random_index]
            distance = math.sqrt((candidate.x - 0.5) ** 2 + (candidate.y + 0.000000005)
** 2)
            distance_weight = math.exp(-0.1 * distance)
            weighted_fitness = candidate.fitness * distance_weight
            if weighted_fitness > best_fitness:
                best_parent = candidate
                best_fitness = weighted_fitness
        parents.append(best_parent)
    return parents

def apply_mutation(individual, mutation_rate, min_x, max_x, min_y, max_y):
    seed = int(time.time() * 1000)
    random.seed(seed)
    mutation_prob = random.uniform(0.0, 1.0)
    mutation_value = random.uniform(-mutation_rate, mutation_rate)
    x = individual.x
    y = individual.y
    if mutation_prob < mutation_rate:
        x += mutation_value
        x = max(min_x, min(x, max_x))
    if mutation_prob < mutation_rate:
        y += mutation_value
        y = max(min_y, min(y, max_y))
    mutated_individual = Organism(x, y)
    mutated_individual.fitness = (math.sin(x) * math.cos(y)) / (1 + x * x + y * y)
    return mutated_individual

def perform_crossover(parent1, parent2):
    seed = int(time.time() * 1000)
    random.seed(seed)
    crossover_prob = random.uniform(0.0, 1.0)
    x = parent1.x if crossover_prob < 0.5 else parent2.x
    y = parent1.y if crossover_prob < 0.5 else parent2.y
    child = Organism(x, y)

```

```

child.fitness = (math.sin(x) * math.cos(y)) / (1 + x * x + y * y)
return child

def check_population_convergence(population):
    fitness_sum = sum(p.fitness for p in population)
    average_fitness = fitness_sum / len(population)
    num_converged = sum(1 for p in population if abs(p.fitness - average_fitness) <
0.0001)
    convergence_ratio = num_converged / len(population)
    return convergence_ratio >= 0.7

def display_results(population, generation):
    sum_fitness = 0
    iterations_to_display = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90,
100]
    if generation in iterations_to_display:
        print(f"Iteration {generation}")
        print("X:")
        for ind in population:
            print(f"{round(ind.x, 4)}")
        print("Y:")
        for ind in population:
            print(f"{round(ind.y, 4)}")
        print("F:")
        for ind in population:
            sum_fitness += ind.fitness
            print(f"{round(ind.fitness, 4)}")
        best_individual = max(population, key=lambda x: x.fitness)
        max_fitness = best_individual.fitness
        print(f"\nmax: {round(max_fitness, 4)}")
        print(f"Среднее: {round(sum_fitness / len(population), 4)}\n")

def draw_fitness_graph(generation_list, average_fitness_list):
    plt.scatter(generation_list, average_fitness_list, marker='o')
    plt.title('Среднее FIT от x')
    plt.xlabel('Поколение')
    plt.ylabel('Среднее FIT')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.savefig('Fitness.png')

def draw_scatter_plots(population):
    plt.figure(figsize=(15, 10))
    plt.subplots_adjust(wspace=0.1, hspace=0.1)

    for i in range(10):
        x_values = [ind.x for ind in population]
        y_values = [ind.y for ind in population]

```

```
plt.scatter(x_values, y_values, marker='o')
plt.title(fГрафик рассеивания для поколения {i + 1})
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, linestyle='--', alpha=0.7)
plt.savefig(fScatter - {i + 1})
```

Приложение Б

```
C:\Python\HW2\venv\Scripts\python.exe C:\Python\HW2\main.py
Iteration 1
X:
0.5114
0.5017
0.5114
1.6118
Y:
0.2446
0.2348
0.2446
-1.9427
F:
0.3593
0.3579
0.3593
-0.0492

max: 0.3593
Среднее: 0.2568

Iteration 2
X:
0.521
0.5017
0.521
1.6118
Y:
0.2542
0.2348
0.2542
-1.9427
F:
0.3606
0.3579
0.3606
-0.0492

max: 0.3606
Среднее: 0.2574

Iteration 3
X:
0.5305
0.5017
0.5305
1.6118
Y:
0.2637
0.2348
0.2637
-1.9427
F:
0.3616
0.3579
0.3616
-0.0492

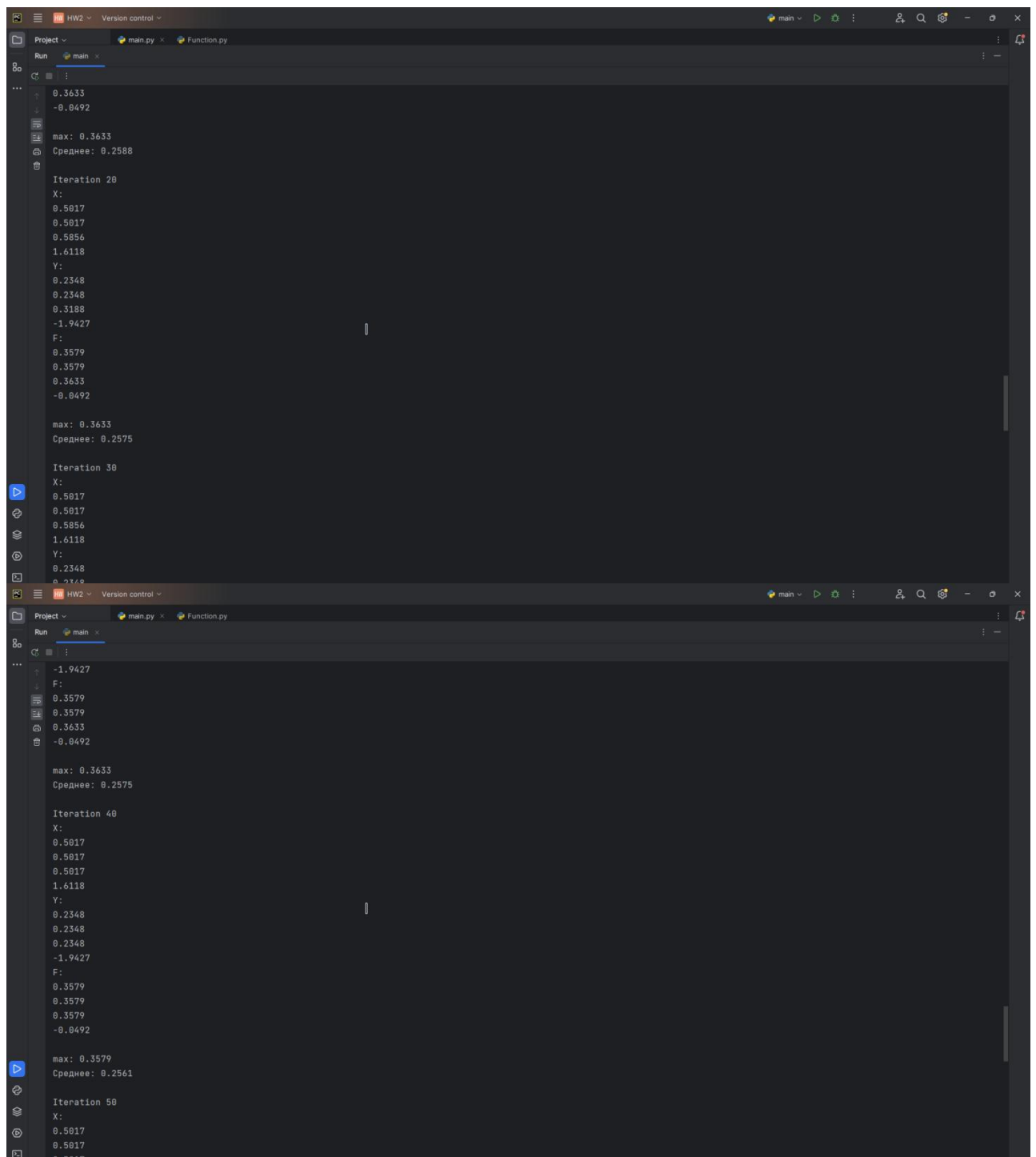
max: 0.3616
Среднее: 0.2579

Iteration 4
X:
0.5399
0.5017
0.5399
1.6118
Y:
0.2731
0.2348
0.2731
-1.9427
F:
0.3616
0.3579
0.3616
-0.0492

max: 0.3616
Среднее: 0.2579
```

```
HW2 - Version control
Project
main.py x Function.py
Run main
F:
0.3624
0.3579
0.3624
-0.0492
max: 0.3624
Среднее: 0.2583
Iteration 5
X:
0.5492
0.5017
0.5492
1.6118
Y:
0.2824
0.2348
0.2824
-1.9427
F:
0.3629
0.3579
0.3629
-0.0492
max: 0.3629
Среднее: 0.2586
Iteration 6
X:
0.5585
0.5017
0.5585
1.6118
Y:
0.2917
0.2348
0.2917
-1.9427
F:
0.3633
0.3579
0.3633
-0.0492
max: 0.3633
Среднее: 0.2588
Iteration 7
X:
0.5676
0.5017
0.5676
1.6118
Y:
0.3008
0.2348
0.3008
-1.9427
F:
0.3635
0.3579
0.3635
-0.0492
max: 0.3635
Среднее: 0.2589
```

```
HW2 - Version control
Project
main.py x Function.py
Run
0.3635
-0.0492
max: 0.3635
Среднее: 0.2589
Iteration 8
X:
0.5766
0.5017
0.5766
1.6118
Y:
0.3098
0.2348
0.3098
-1.9427
F:
0.3635
0.3579
0.3635
-0.0492
max: 0.3635
Среднее: 0.2589
Iteration 9
X:
0.5856
0.5017
0.5856
1.6118
Y:
0.3188
0.2348
0.3188
-1.9427
F:
0.3633
0.3579
0.3633
-0.0492
max: 0.3633
Среднее: 0.2588
Iteration 10
X:
0.5856
0.5017
0.5856
1.6118
Y:
0.3188
0.2348
0.3188
-1.9427
F:
0.3633
0.3579
0.3633
-0.0492
max: 0.3633
Среднее: 0.2588
```



```
Project - Version control
main.py x Function.py
Run main x
Y:
0.2348
0.2348
0.2348
-1.9427
F:
0.3579
0.3579
0.3579
-0.0492
max: 0.3579
Среднее: 0.2561
Iteration 60
X:
0.5017
0.5017
0.5017
1.6118
Y:
0.2348
0.2348
0.2348
-1.9427
F:
0.3579
0.3579
0.3579
-0.0492
max: 0.3579
Среднее: 0.2561
Iteration 70
```



```
HW2 - Version control
Project - main.py x Function.py
Run - main x
0.5017
0.5017
0.5017
1.6118
Y:
0.2348
0.2348
0.2348
-1.9427
F:
0.3579
0.3579
0.3579
-0.0492

max: 0.3579
Среднее: 0.2561

Iteration 80
X:
0.8143
0.5017
0.5482
1.6118
Y:
0.5475
0.2348
0.2814
-1.9427
F:
0.3164
0.3579
0.3629
-0.0492

max: 0.3629
Среднее: 0.247

Iteration 90
X:
0.8143
0.5017
0.5017
1.6118
Y:
0.5475
0.2348
0.2348
-1.9427
F:
0.3164
0.3579
0.3579
-0.0492

max: 0.3579
Среднее: 0.2457

Оптимальное решение:
x = 0.5017 , y = 0.2348
f(x, y) = 0.3579
```