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/MO-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 остановки 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:

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

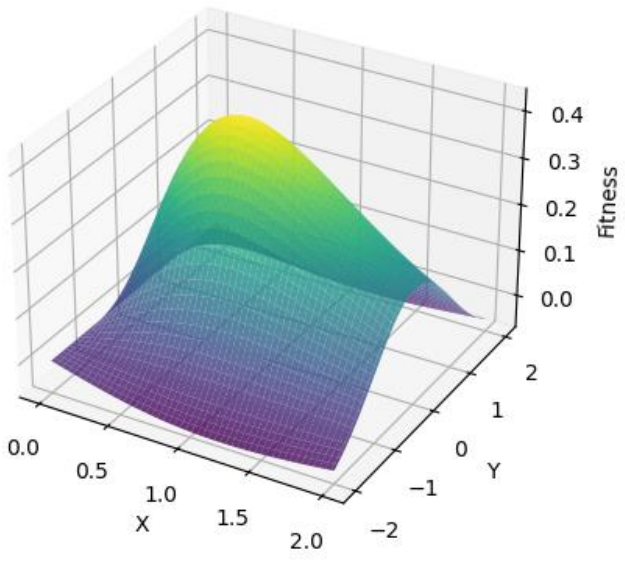


Рис. 1. График функции $f(x, y)$

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

№ поколения	X	Y	FIT	Максимальный результат	Средний результат
0	1.3958	1.0748	0.1142	0.3126	0.217
	1.4901	-0.5033	0.2514		
	1.8722	-0.3888	0.1898		
	1.3477	0.3451	0.3126		
1	1.474	-0.4077	0.2737	0.3331	0.2742
	1.3939	0.0696	0.3331		
	1.8433	-0.3483	0.2003		
	1.5636	0.0430	0.2899		
2	1.3939	0.0696	0.3331	0.3331	0.3068
	1.3939	0.0696	0.3331		
	1.7125	-0.2267	0.2421		
	1.4540	0.0138	0.3189		
3	1.3939	0.1561	0.3277	0.3331	0.3256
	1.3939	0.0696	0.3331		
	1.4520	0.0156	0.3194		
	1.4379	-0.0560	0.3223		
4	1.4059	0.0984	0.3287	0.3287	0.3262
	1.4316	-0.0256	0.3246		
	1.4287	0.072	0.3241		
	1.3979	0.1464	0.3275		
5	1.4089	0.0949	0.3282	0.3459	0.3328
	1.4234	0.0782	0.3252		
	1.3981	0.0686	0.3321		
	1.3239	0.1458	0.3459		

6	1.4158	0.087	0.3268	0.3268	0.3252
	1.4205	0.0815	0.3258		
	1.4371	0.0698	0.3220		
	1.4200	0.0769	0.3261		
7	1.4217	0.0357	0.327	0.3270	0.3256
	1.4336	0.0727	0.3228		
	1.4165	0.0861	0.3266		
	1.4204	0.0816	0.3258		
8	1.4188	0.064	0.327	0.3270	0.3253
	1.421	0.0425	0.327		
	1.4328	0.0702	0.3231		
	1.4306	0.0633	0.3239		
9	1.4313	0.0655	0.3237	0.3237	0.3202
	1.4812	0.0645	0.3108		
	1.4325	0.0692	0.3232		
	1.4324	0.0689	0.3233		
10	1.4324	0.0689	0.3233	0.3233	0.3233
	1.4325	0.0691	0.3232		
	1.4323	0.0687	0.3233		
	1.4324	0.0688	0.3233		

Таблица 1.

Также стоит понимать, что значения за 10 и 100 поколения отличаются, где последнее является более приближенным к действительности:

$$\mathrm{FIT}(1.1913;\ 0.0549) = 0.3829$$

```
Overall Best Solution: (1.1913, 0.0549), Overall Best Fitness: 0.3829
```

Пример расположения точек на начальном поколении приведен на рис. 2, а также для десяти поколений на рис. 3-8.

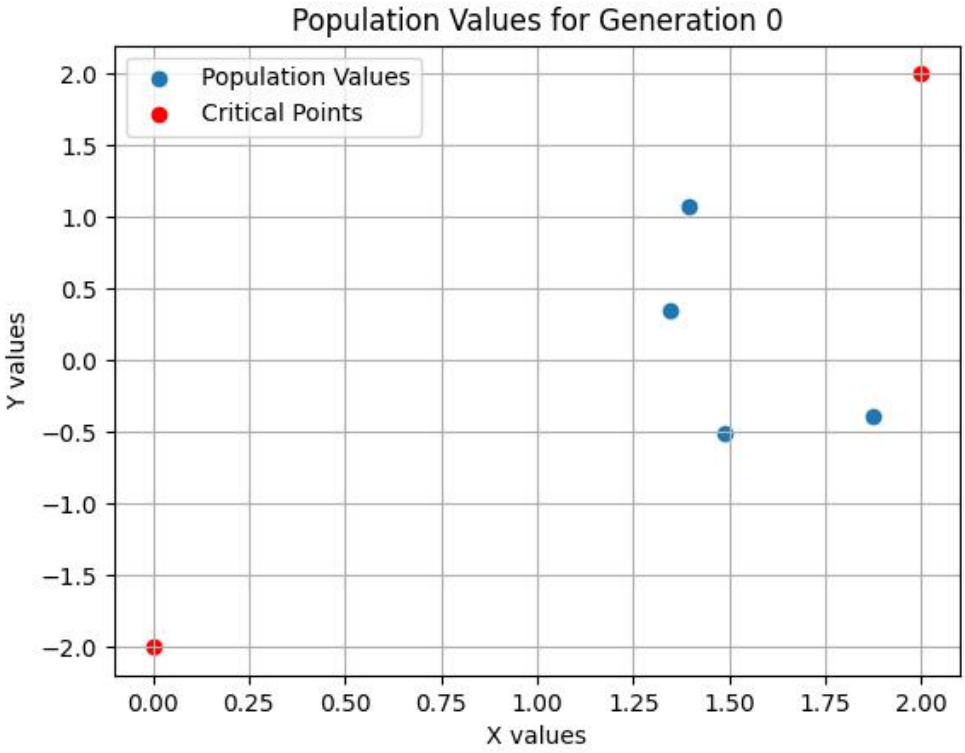


Рис. 2.

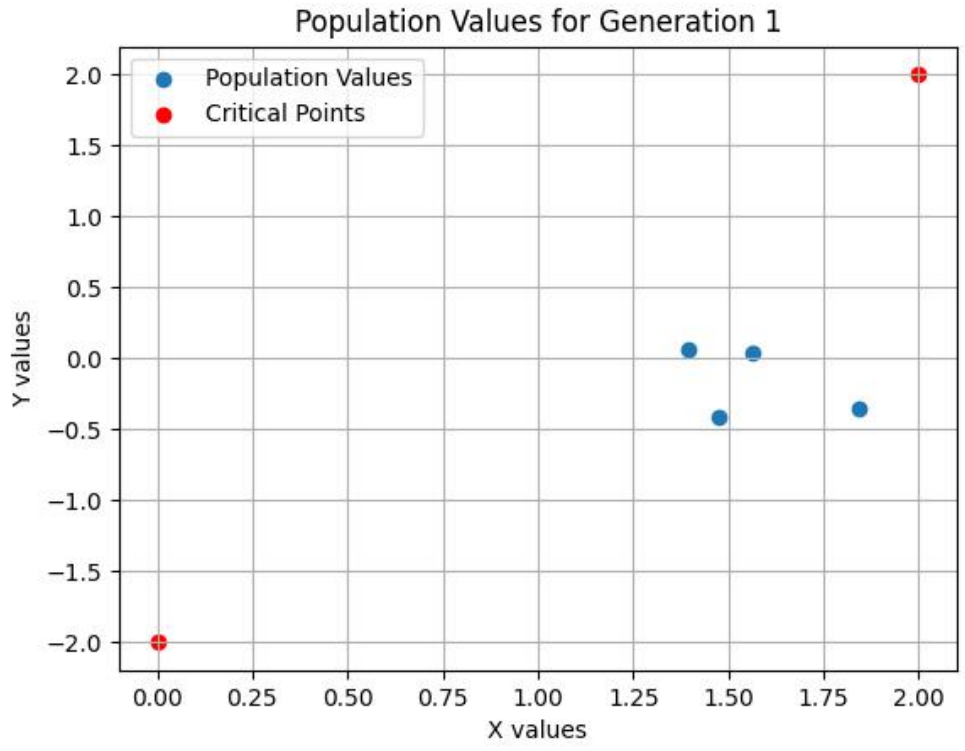


Рис. 3.

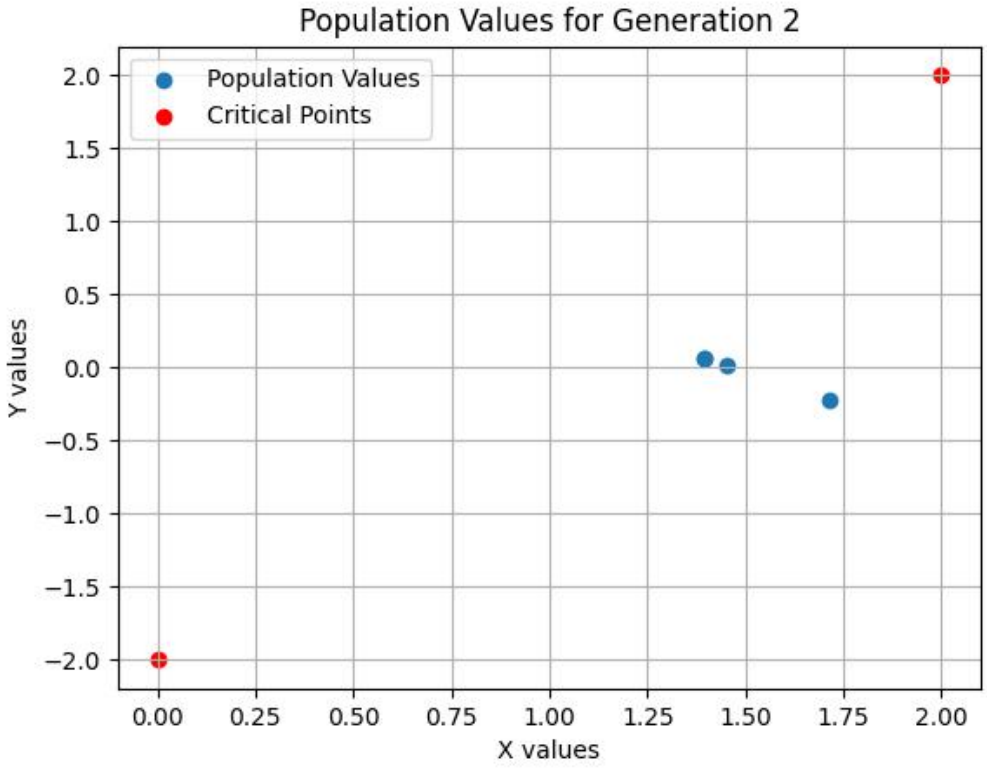

Population Values for Generation 2
Population Values
Critical Points
Y values
X values
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Рис. 4.

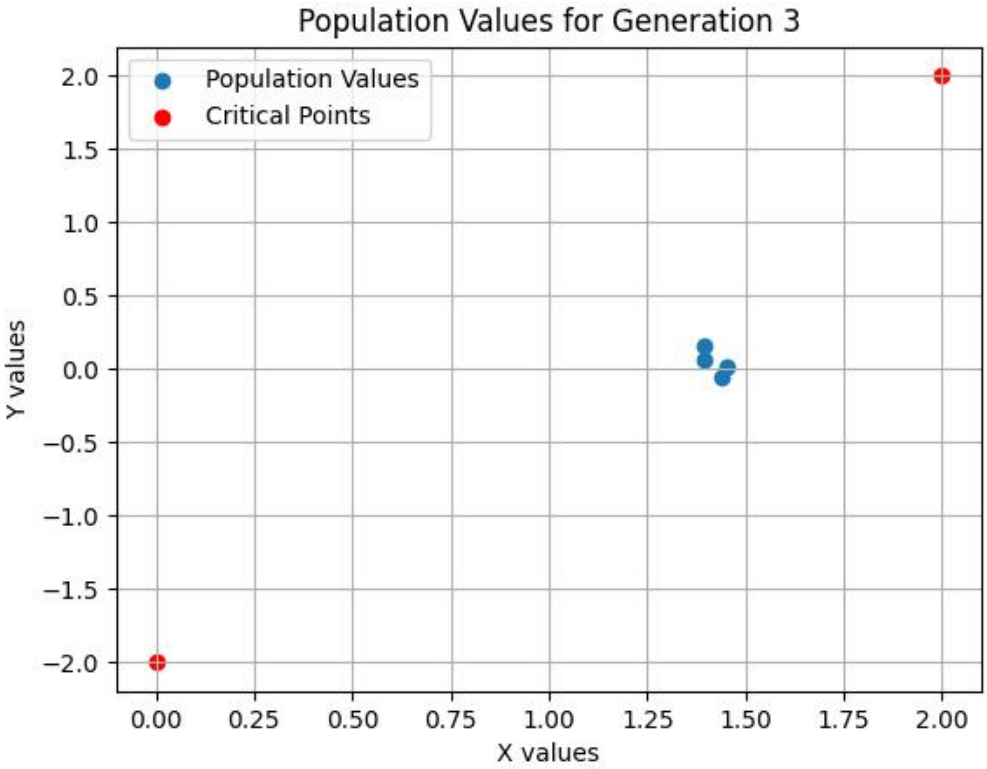

Population Values for Generation 3
Population Values
Critical Points
Y values
X values
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Рис. 5.

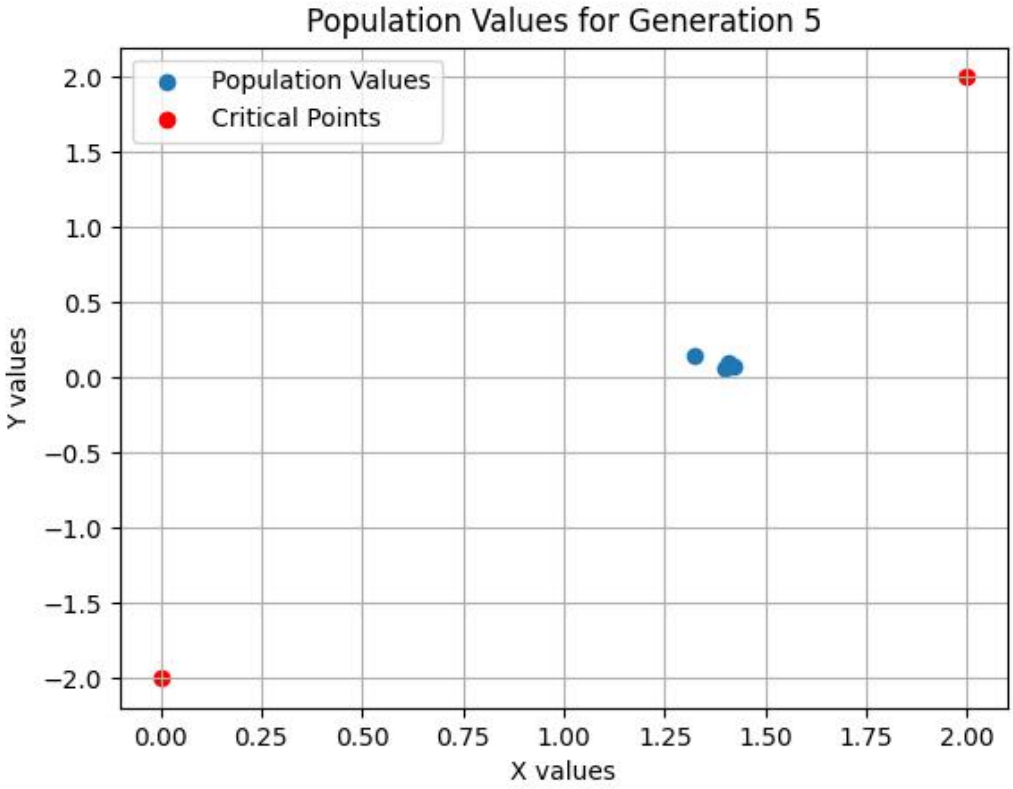

Population Values for Generation 5
Population Values
Critical Points
Y values
X values
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Рис. 6.

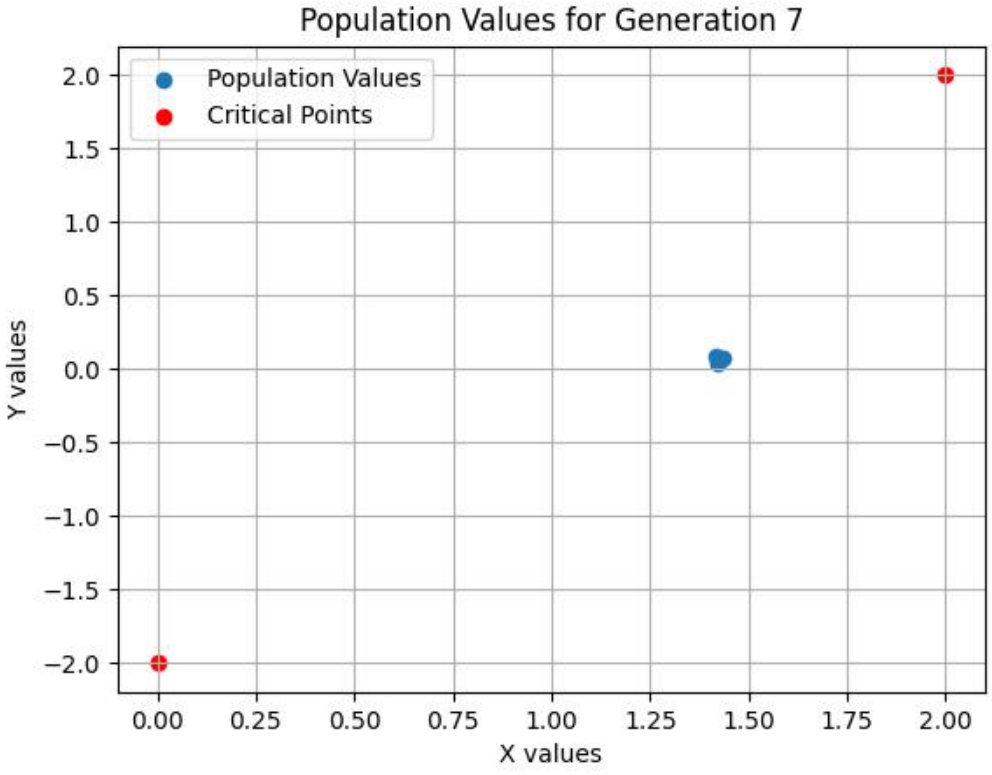

Population Values for Generation 7
Population Values
Critical Points
Y values
X values
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Рис. 7.

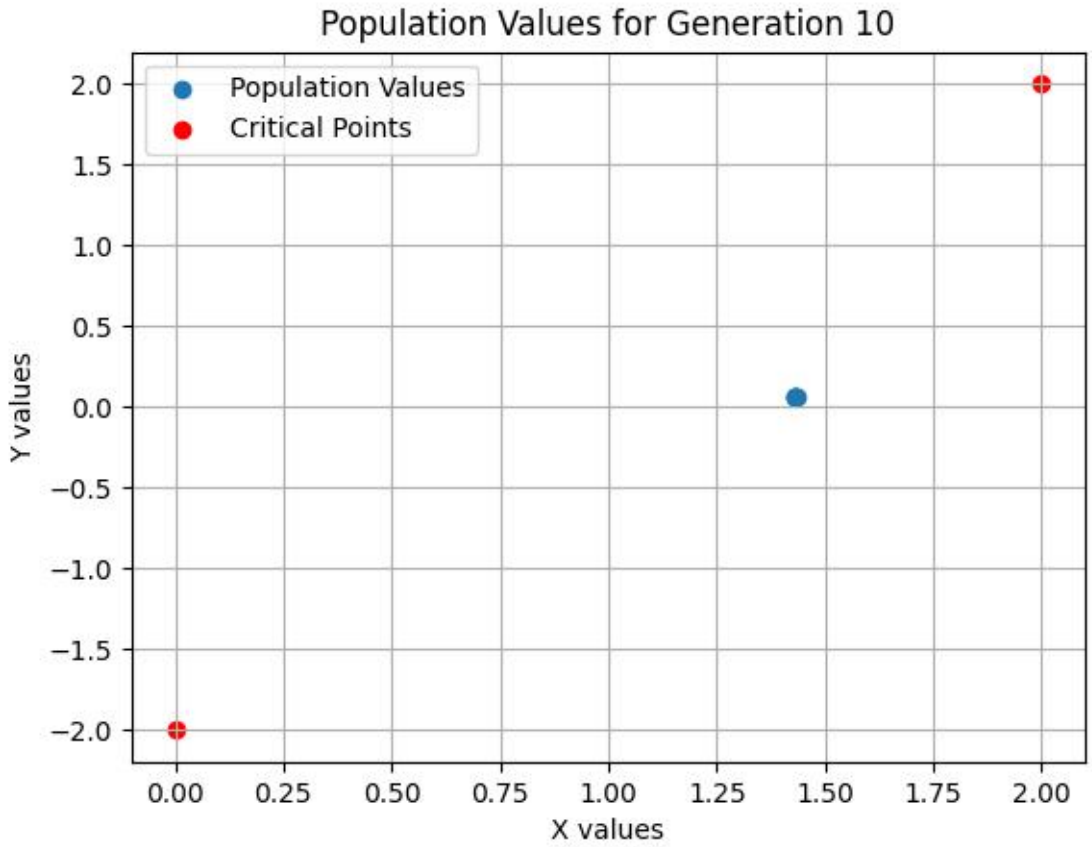


Рис. 8.

Как видно из таблицы 1, а также на рис. 6., уже к 6 - 7 поколениям происходит схождение алгоритма и хромосом всех особей в популяции, а результат вычисления приближен к искомому (но не достаточно близко!).

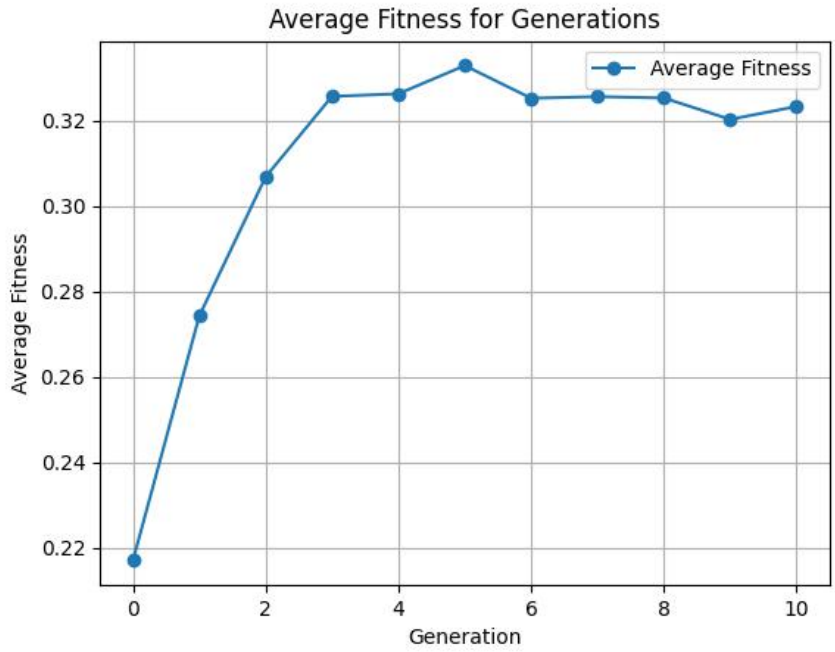


Рис. 9. График зависимости среднего значения FIT от номера популяции

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.

Значения, полученные в ходе работы, оказались близки к действительности, что еще раз доказывает правильность выполнения алгоритма.

Приложение А

Файл ‘Main.py’:

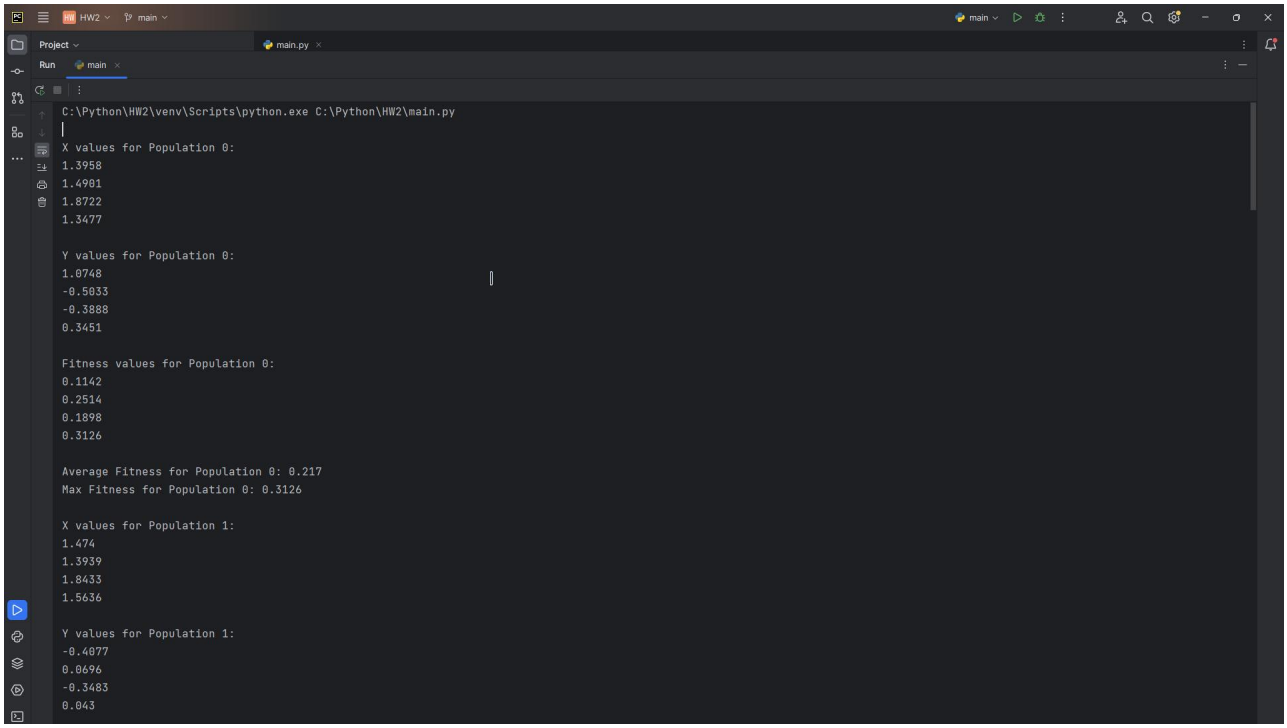
```
import numpy as np
import matplotlib.pyplot as plt

L = 10


def fitness_function(x, y):
    return np.sin(x) * np.cos(y) / (1 + x ** 2 + y ** 2)


def initialize_population(population_size):
    population = []
    for _ in range(population_size):
        x_binary = ''.join(np.random.choice(['0', '1']) for _ in range(L))
        y_binary = ''.join(np.random.choice(['0', '1']) for _ in range(L))

        x = int(x_binary, 2) / (2 ** L - 1)
        y = -2 + int(y_binary, 2) / (2 ** L - 1) * 4

        population.append((x, y))
    return population


def roulette_selection(population, fitness_values):
    total_fitness = sum(fitness_values)
    probabilities = [max(0, fitness / total_fitness) for fitness in fitness_values]
    probabilities /= np.sum(probabilities)
    selected_index = np.random.choice(len(population), p=probabilities)
    return population[selected_index]


def two_point_crossover(parent1, parent2):
    point1 = np.random.randint(1, L)
    point2 = np.random.randint(point1, L)

    x_binary_parent1 = bin(int((parent1[0] * (2 ** L - 1)) + 0.5))[2:].zfill(L)
    x_binary_parent2 = bin(int((parent2[0] * (2 ** L - 1)) + 0.5))[2:].zfill(L)
    y_binary_parent1 = bin(int(((parent1[1] + 2) / 4) * (2 ** L - 1) + 0.5))[2:].zfill(L)
    y_binary_parent2 = bin(int(((parent2[1] + 2) / 4) * (2 ** L - 1) + 0.5))[2:].zfill(L)

    x_child = int(x_binary_parent1[:point1] + x_binary_parent2[point1:point2] +
x_binary_parent1[point2:], 2) / (2 ** L - 1)
    y_child = -2 + int(y_binary_parent1[:point1] + y_binary_parent2[point1:point2] +
y_binary_parent1[point2:], 2) / (2 ** L - 1) * 4
```

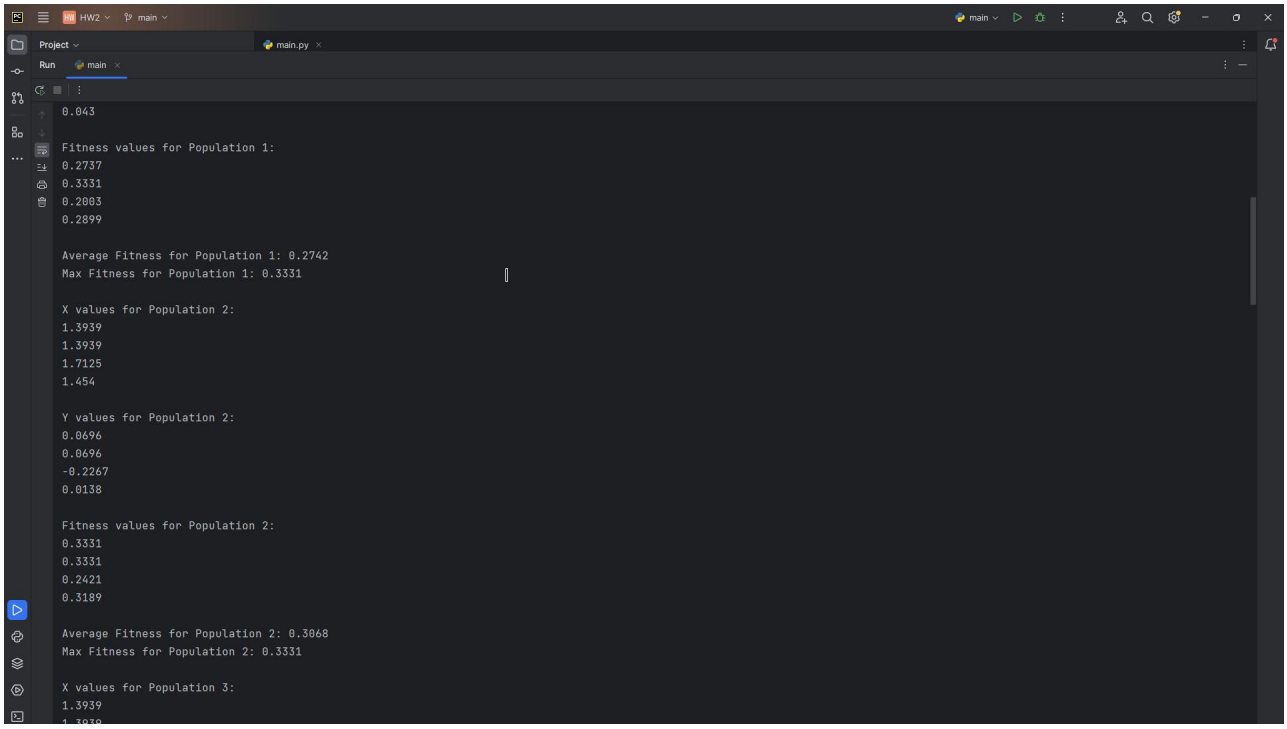
```
    return x_child, y_child


def mutation(child, mutation_rate):
    x_binary = bin(int((child[0] * (2 ** L - 1)) + 0.5))[2:].zfill(L)
    y_binary = bin(int(((child[1] + 2) / 4) * (2 ** L - 1) + 0.5))[2:].zfill(L)

    if np.random.rand() < mutation_rate:
        mutated_bit_x = np.random.randint(L)
        x_binary = x_binary[:mutated_bit_x] + ('0' if x_binary[mutated_bit_x] == '1'
else '1') + x_binary[
                                                                mutated_bit_x + 1:]

    if np.random.rand() < mutation_rate:
        mutated_bit_y = np.random.randint(L)
        y_binary = y_binary[:mutated_bit_y] + ('0' if y_binary[mutated_bit_y] == '1'
else '1') + y_binary[
                                                                mutated_bit_y + 1:]

    child = (int(x_binary, 2) / (2 ** L - 1), -2 + int(y_binary, 2) / (2 ** L - 1) * 4)
    return child


def plot_3d_surface():
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    x_vals = np.linspace(0, 2, 100)
    y_vals = np.linspace(-2, 2, 100)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = fitness_function(X, Y)

    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.8)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Fitness')

    plt.savefig('3D.png')


def plot_average_fitness(generation_values, average_fitness_values):
    plt.figure()
    plt.plot(generation_values[:len(average_fitness_values)], average_fitness_values,
marker='o',
             label='Average Fitness')
    plt.title('Average Fitness for Generations')
```

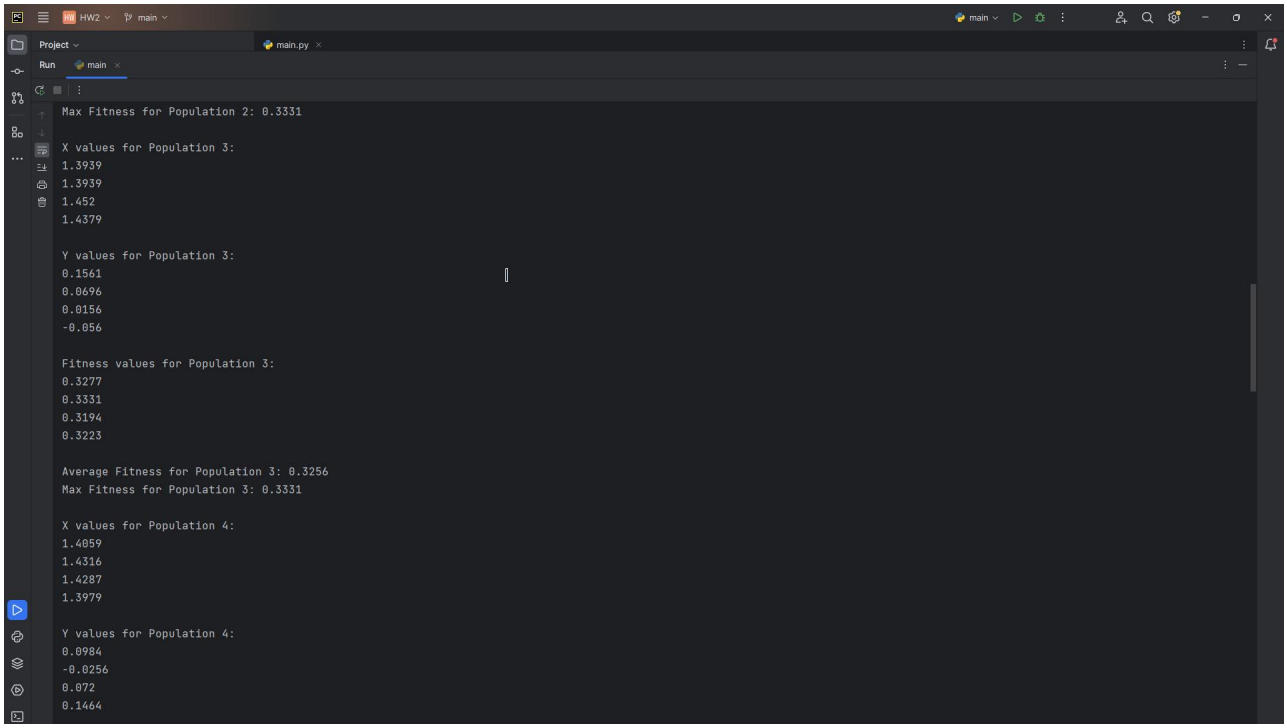
```
    plt.xlabel('Generation')
    plt.ylabel('Average Fitness')
    plt.grid(True)
    plt.legend()
    plt.savefig('average - fitness.png')


def plot_population_values(generation, x_values, y_values):
    plt.figure()
    plt.scatter(x_values, y_values, marker='o', label='Population Values')

    additional_points_x = [0, 2]
    additional_points_y = [-2, 2]
    plt.scatter(additional_points_x, additional_points_y, color='red', marker='o',
label='Critical Points')

    plt.title(f'Population Values for Generation {generation}')
    plt.xlabel('X values')
    plt.ylabel('Y values')
    plt.legend()
    plt.grid(True)
    plt.savefig(f'population - values - {generation}.png')


def genetic_algorithm(population_size, generations, populations_to_display):
    population = initialize_population(population_size)
    best_overall_fitness = -np.inf
    best_overall_individual = None
    average_fitness_values = []

    for generation in range(generations):
        fitness_values = [fitness_function(x, y) for x, y in population]

        if generation in populations_to_display:
            print(f"\nX values for Population {generation}:")
            for i in range(min(10, population_size)):
                print(round(population[i][0], 4))

            print(f"\nY values for Population {generation}:")
            for i in range(min(10, population_size)):
                print(round(population[i][1], 4))

            print(f"\nFitness values for Population {generation}:")
            for i in range(min(10, population_size)):
                print(round(fitness_values[i], 4))

            average_fitness = np.mean(fitness_values)
```

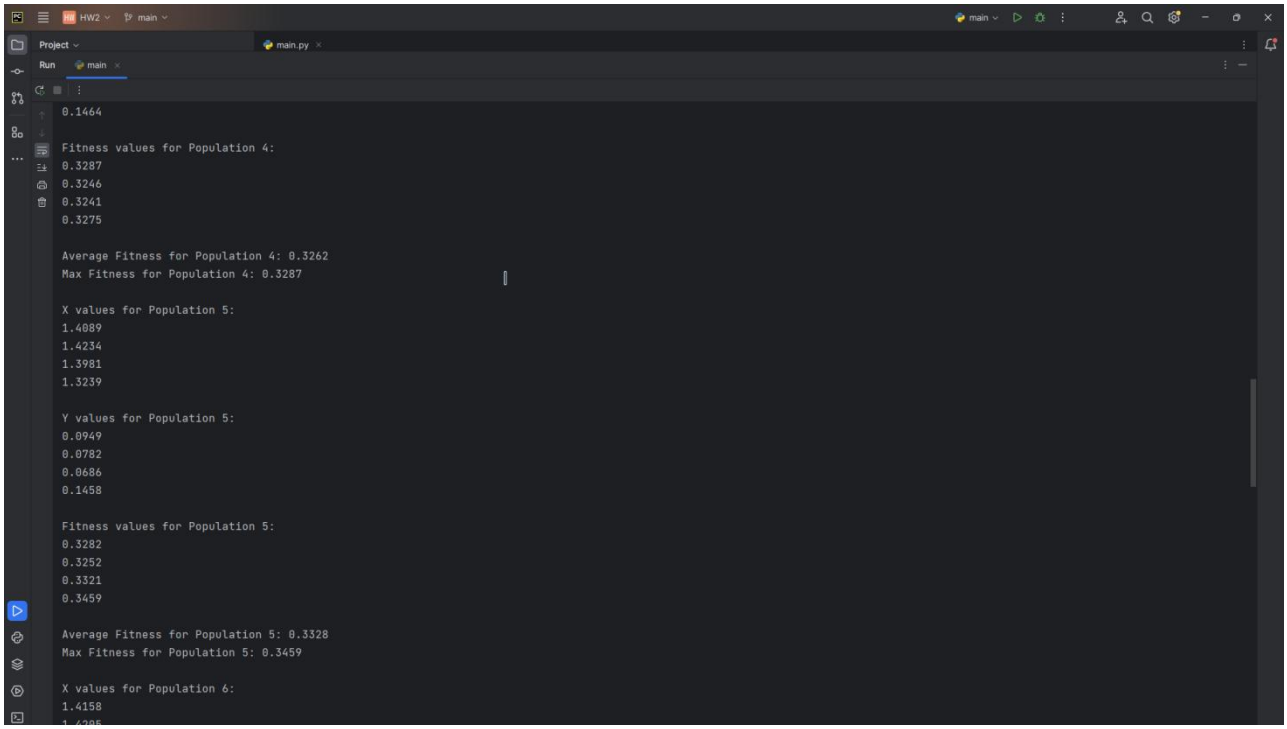
```
            print(f"\nAverage Fitness for Population {generation}:
{round(average_fitness, 4)}")
            print(f"Max Fitness for Population {generation}: {round(max(fitness_values),
4)}")

            average_fitness_values.append(average_fitness)

            x_values = [individual[0] for individual in population[:10]]
            y_values = [individual[1] for individual in population[:10]]
            plot_population_values(generation, x_values, y_values)

        if max(fitness_values) > best_overall_fitness:
            best_overall_fitness = max(fitness_values)
            best_overall_individual = population[np.argmax(fitness_values)]

        parents = [roulette_selection(population, fitness_values) for _ in
range(population_size)]

        children = []
        for i in range(0, population_size, 2):
            parent1 = parents[i]
            parent2 = parents[i + 1]
            child1 = two_point_crossover(parent1, parent2)
            child2 = two_point_crossover(parent2, parent1)
            children.extend([mutation(child1, 0.25), mutation(child2, 0.25)])

        population = children

    print(
        f"\nOverall Best Solution: ({round(best_overall_individual[0], 4)},
{round(best_overall_individual[1], 4)}), Overall Best Fitness:
{round(best_overall_fitness, 4)}")

    plot_average_fitness(range(generations), average_fitness_values)
    plot_3d_surface()


genetic_algorithm(population_size=4, generations=100, populations_to_display=[0,
1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

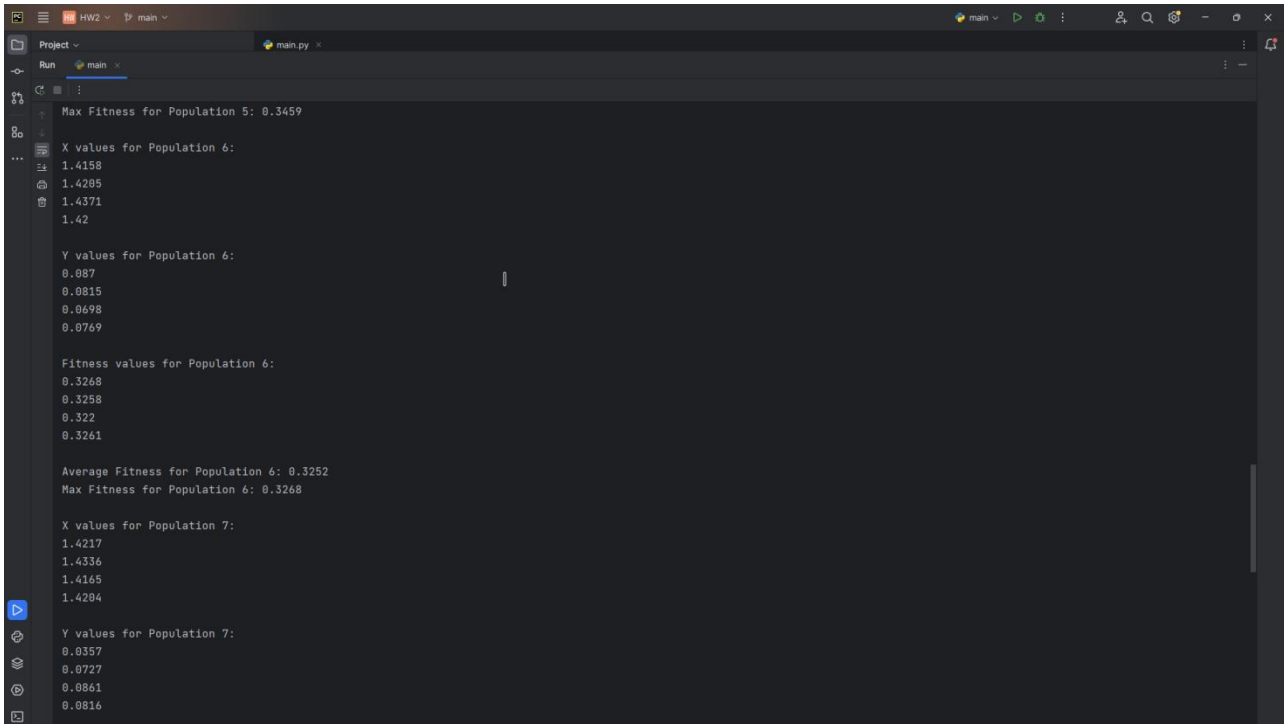
Приложение Б

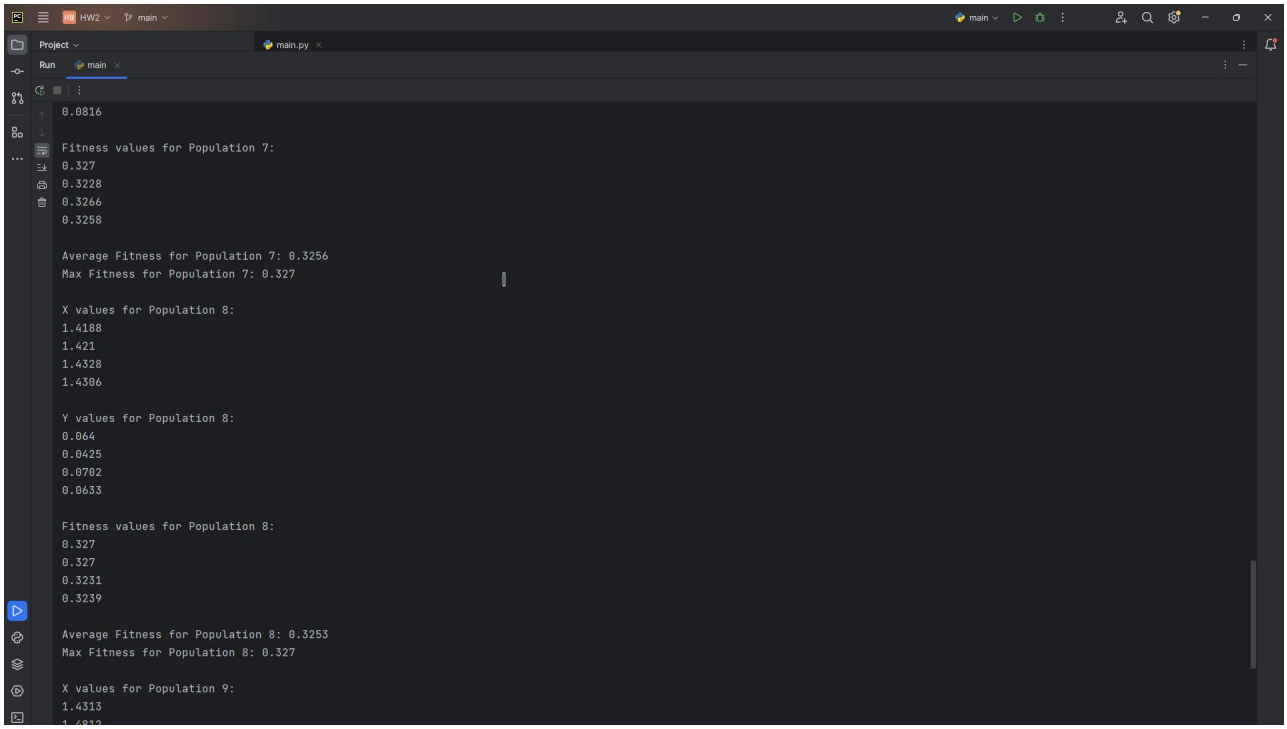


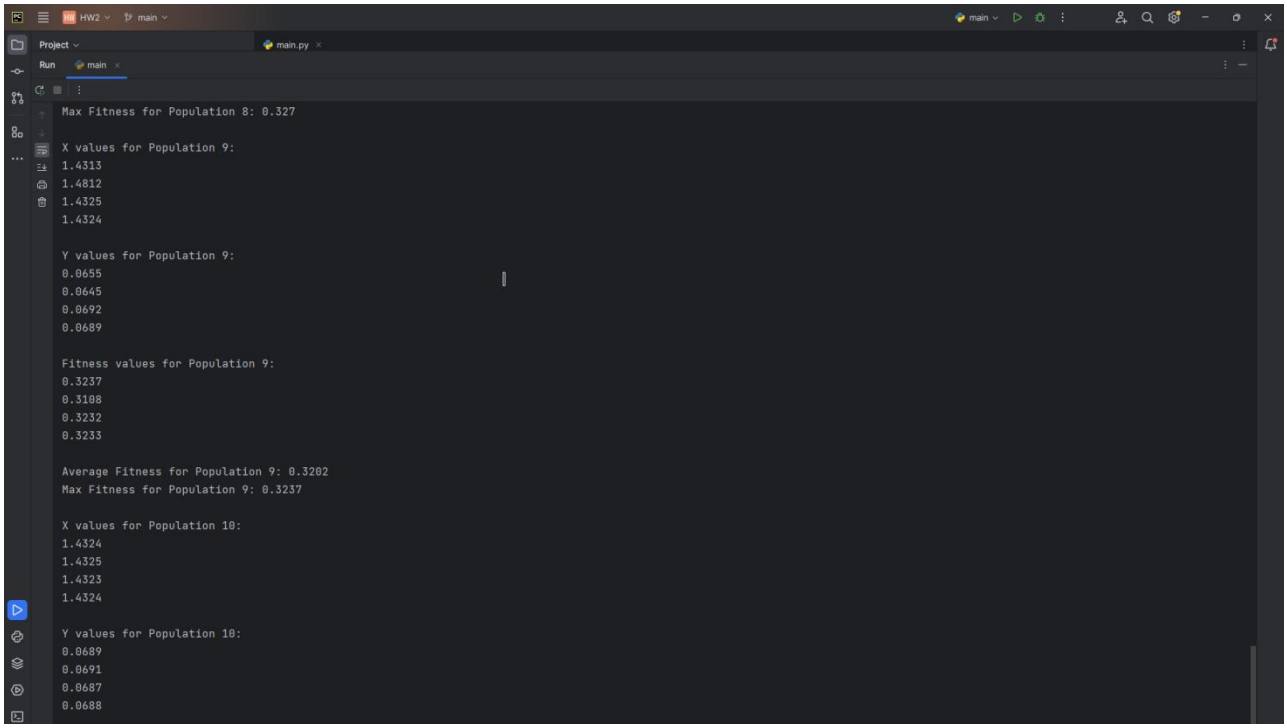


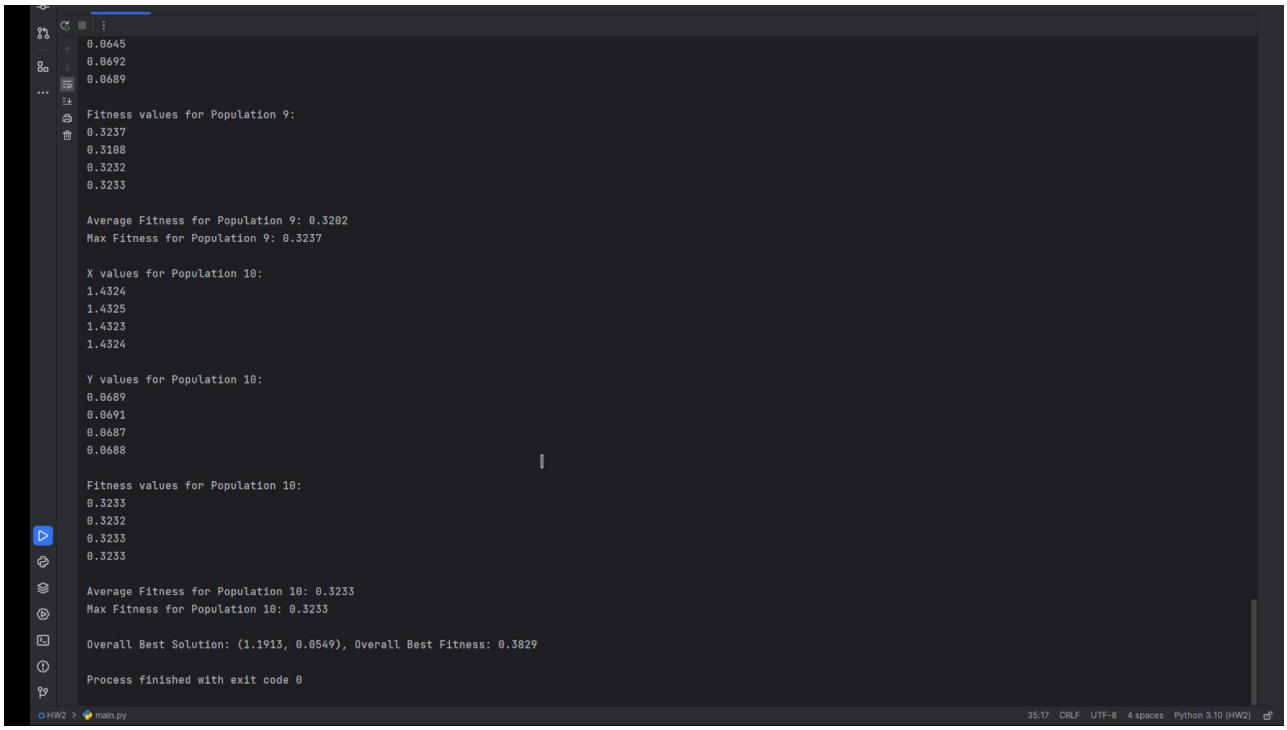

HW2
main
Project
main.py
Run
main
Max Fitness for Population 2: 0.3331
X values for Population 3:
1.3939
1.3939
1.452
1.4379
Y values for Population 3:
0.1561
0.0696
0.0156
-0.056
Fitness values for Population 3:
0.3277
0.3331
0.3194
0.3223
Average Fitness for Population 3: 0.3256
Max Fitness for Population 3: 0.3331
X values for Population 4:
1.4059
1.4316
1.4287
1.3979
Y values for Population 4:
0.0984
-0.0256
0.072
0.1464


HW2
main
Project
main.py
Run
main
0.1464
Fitness values for Population 4:
0.3287
0.3246
0.3241
0.3275
Average Fitness for Population 4: 0.3262
Max Fitness for Population 4: 0.3287
X values for Population 5:
1.4089
1.4234
1.3981
1.3239
Y values for Population 5:
0.0949
0.0782
0.0686
0.1458
Fitness values for Population 5:
0.3282
0.3252
0.3321
0.3459
Average Fitness for Population 5: 0.3328
Max Fitness for Population 5: 0.3459
X values for Population 6:
1.4158


HW2
main
Project
main.py
Run
main
Max Fitness for Population 5: 0.3459
X values for Population 6:
1.4158
1.4205
1.4371
1.42
Y values for Population 6:
0.087
0.0815
0.0698
0.0769
Fitness values for Population 6:
0.3268
0.3258
0.322
0.3261
Average Fitness for Population 6: 0.3252
Max Fitness for Population 6: 0.3268
X values for Population 7:
1.4217
1.4336
1.4165
1.4204
Y values for Population 7:
0.0357
0.0727
0.0861
0.0816


HW2
main
Project
main.py
Run
main
0.0816
Fitness values for Population 7:
0.327
0.3228
0.3266
0.3258
Average Fitness for Population 7: 0.3256
Max Fitness for Population 7: 0.327
X values for Population 8:
1.4188
1.421
1.4328
1.4306
Y values for Population 8:
0.064
0.0425
0.0702
0.0633
Fitness values for Population 8:
0.327
0.327
0.3231
0.3239
Average Fitness for Population 8: 0.3253
Max Fitness for Population 8: 0.327
X values for Population 9:
1.4313


Max Fitness for Population 8: 0.327
X values for Population 9:
1.4313
1.4812
1.4325
1.4324
Y values for Population 9:
0.0655
0.0645
0.0692
0.0689
Fitness values for Population 9:
0.3237
0.3108
0.3232
0.3233
Average Fitness for Population 9: 0.3202
Max Fitness for Population 9: 0.3237
X values for Population 10:
1.4324
1.4325
1.4323
1.4324
Y values for Population 10:
0.0689
0.0691
0.0687
0.0688


0.0645
0.0692
0.0689
Fitness values for Population 9:
0.3237
0.3108
0.3232
0.3233
Average Fitness for Population 9: 0.3202
Max Fitness for Population 9: 0.3237
X values for Population 10:
1.4324
1.4325
1.4323
1.4324
Y values for Population 10:
0.0689
0.0691
0.0687
0.0688
Fitness values for Population 10:
0.3233
0.3232
0.3233
0.3233
Average Fitness for Population 10: 0.3233
Max Fitness for Population 10: 0.3233
Overall Best Solution: (1.1913, 0.0549), Overall Best Fitness: 0.3829
Process finished with exit code 0