Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №7

з дисципліни «Методи та технології штучного інтелекту»

на тему

«Знаходження мінімуму та максимуму функцій за допомогою генетичних алгоритмів»

Виконала:
студентка групи ІС-12
Павлова Софія

Викладач:
Шимкович В. М.

Київ – 2023

1. Постановка задачі

Мета: Знайти мінімум (мінімізація) і максимум (максимізація) функцій одно- і двох змінних за допомогою генетичних алгоритмів.

Завдання:

1. Мінімізувати функцію однієї змінної.
2. Максимізувати функцію двох змінних.

9.	$y = 0.2 \cdot \sin(3x) \cdot x^2$ $z = \sin\lvert x\rvert \cdot \sin(x + y)$	9.	22.	13.

Рисунок 1 – завдання за варіантом № 22

2. Виконання

Генетичний алгоритм:

Розробимо алгоритм призначений для роботи з функціями однієї та двох змінних.

Можна виділити такі **етапи генетичного алгоритму**:

- Створення початкової популяції;
- Обчислення функції пристосованості для осіб популяції (оцінювання);
- Вибір індивідів із поточної популяції (селекція);
- Схрещення або/та мутація;
- Обчислення функції пристосовуваності для всіх осіб;
- Формування нового покоління.

Розглянемо їх по черзі.

Створення початкової популяції

Створимо випадкову початкову популяцію, де кожен індивід мережі представлений вектором параметрів (змінних), які можуть бути випадковим чином згенеровані в заданих межах.

Лістинг:

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt

# Клас генетичного алгоритму
class Genetic_Algorithm:
    def __init__(self, function, variables, bounds, max_search=True):
        self.function = function
        self.variables = variables
        self.bounds = bounds
        self.max_search = max_search
```

```
    # Створення початкової популяції
    def create_population(self):
        return [[random.uniform(*bound) for bound in self.bounds] for i in range(20)]

    # Головний цикл алгоритму
    def run(self, generations, children):
        # Створення популяції
        population = self.create_population()
```

Обчислення функції пристосованості

Розрахуємо значення функції оптимальності (функція Y або Z) для кожного індивіда популяції.

Здійснимо відбір індивідів на основі значень функції:

- якщо потрібно **максимізувати** – обирається індивід з найвищим значенням;
- якщо потрібно **мінімізувати** – обирається індивід з найнижчим значенням.

Лістинг:

```
# Клас генетичного алгоритму
class Genetic_Algorithm:
    [...]

    # Обчислення значення функції Y для конкретного індивіда
    def calculate_y(self, individual):
        return self.function(*individual)
```

Схрещення та мутація

Згенеруємо нащадків шляхом кросовера (змішування) між обраними батьками. Для деяких з них мутуємо частину їх генів.

Лістинг:

```
# Клас генетичного алгоритму
class Genetic_Algorithm:
    [...]

    # Вибір індивіда з популяції на основі значень функції
    def select(self, population):
        ys = [self.calculate_y(individual) for individual in population]
        # Якщо максимізація (функція Z)
        if self.max_search:
            return max(zip(population, ys), key=lambda x: x[1])[0]
        # Якщо мінімізація (функція Y)
        else:
            return min(zip(population, ys), key=lambda x: x[1])[0]

    # Операція кросовера для створення нащадка
    def crossover(self, parent_1, parent_2):
```

```
        child = []
        # Вибирається випадково половина значень від кожного з батьків
        for i in range(self.variables):
            if random.random() < 0.5:
                child.append(parent_1[i])
            else:
                child.append(parent_2[i])
        return child

    # Операція мутації
    def mutate(self, chromosom):
        # Змінює значення окремих генів індивіда з невеликою ймовірністю
        for i in range(self.variables):
            if random.random() < 0.1:
                chromosom[i] = random.uniform(*self.bounds[i])
        return chromosom

    # Головний цикл алгоритму
    def run(self, generations, children):
        [...]

        # Створення поколінь
        for generation in range(generations):
            offspring = []
            for j in range(20):
                # Створення дітей
                child = [None] * children
                for i in range(children):
                    child[i] = self.crossover(random.choice(population),
random.choice(population))
                    # Можлива мутація
                    child[i] = self.mutate(child[i])
                    offspring.append(child[i])
```

Формування нового покоління

Скомбінуємо поточну популяцію та нащадків. Відсортуємо її особин за значенням функції оптимальності та за допомогою елітарного відбору виберемо найкращих.

Лістинг:

```
# Клас генетичного алгоритму
class Genetic_Algorithm:
    [...]

    # Оновлення поколінь
    def update_population(self, current_population, offspring):
        population_size = len(current_population)
        combined_population = current_population + offspring
        sorted_population = sorted(combined_population, key=lambda x:
self.calculate_y(x), reverse=self.max_search)
        return sorted_population[:population_size]

    # Головний цикл алгоритму
    def run(self, generations, children):
        [...]

        # Створення поколінь
```

```
        for generation in range(generations):
            offspring = []
            [...]

            # Оновлення поколінь
            population = self.update_population(population, offspring)
```

Селекція найпристосованішого

На основі функції оптимальності знайдемо найкращого представника популяції. Відобразимо зміну функції оптимальності крізь покоління на графіку.

Лістинг:

```
# Клас генетичного алгоритму
class Genetic_Algorithm:
    [...]

    # Графік функції оптимальності алгоритму
    def plot_fitness(self, fitness_history, generations):
        # Якщо максимізація
        if self.max_search == True:
            text = 'Max Z'
            label = 'Z'
        # Якщо мінімізація
        else:
            text = 'Min Y'
            label = 'Y'

        plt.plot(range(generations), fitness_history, label='Генетичний алгоритм')
        plt.title('Функція оптимальності - ' + text)
        plt.xlabel('Покоління')
        plt.ylabel(label)
        plt.legend()
        plt.show()

    # Головний цикл алгоритму
    def run(self, generations, children):
        [...]

        # Створення поколінь
        # Значення функції оптимальності для кожного покоління
        fitness_history = []
        for generation in range(generations):
            [...]

            # Обчислення функції оптимальності
            best_individual = self.select(population)
            fitness_history.append(self.calculate_y(best_individual))

        # Виведення графіка функції оптимальності
        self.plot_fitness(fitness_history, generations)

        return self.select(population)
```

Використання алгоритму:

Визначимо діапазони зміни значень функцій.

Лістинг:

```
# Функція Y
def y_func(x):
    return 0.2 * np.sin(3 * x) * x**2

# Функція Z
def z_func(x, y):
    return np.sin(np.abs(x)) * np.sin(x + y)

# Головні виклики
if __name__ == "__main__":
    # Діапазон зміни X
    x = np.linspace(-10, 10, 400)
    # Значення функції Y
    y_values = y_func(x)


    # Діапазон зміни Y
    y = np.linspace(-10, 10, 400)
    # Створення простору XYZ
    x_grid, y_grid = np.meshgrid(x, y)
    # Значення функції Z
    z_values = z_func(x_grid, y_grid)
```

Задача 1: Мінімізувати функцію однієї змінної

Знайдемо мінімум функції однієї змінної за допомогою програмних засобів Python та за допомогою генетичного алгоритму. Для генетичного алгоритму використаємо наступні параметри:

Особин у поколінні – 20,

Кількість поколінь – 200,

Дітей у поколінні – 10.

Порівняємо отримані різними методами результати.

Лістинг:

```
# Знаходження min для функції Y
def find_min():
    # Вбудований метод
    y_min = np.min(y_values)

    # Використання генетичного алгоритму
    y_genetic = Genetic_Algorithm(y_func, variables=1, bounds=[(-10, 10)],
max_search=False)
    y_min_genetic = y_genetic.run(200, 10)

    return y_min, y_func(*y_min_genetic)

# Графік функції Y
def plot_2d(x, y_values):
    plt.plot(x, y_values)
    plt.title('Функція Y')
    plt.grid()
    plt.show()

# Головні виклики
if __name__ == "__main__":
    [...]

    # Знаходження min та max функцій
    y_min, y_min_genetic = find_min()

    # Створення таблиці порівняння результатів
    data = {
        'Min Y': [y_min_genetic, y_min, abs(y_min_genetic - y_min)]
    }
    df = pd.DataFrame(data, index=['Генетичний алгоритм', 'Реальне значення', 'Похибка'])
    # Виведення таблиці
    print('\nРезультат алгоритму:')
    print(df)

    # Виведення графіків
    plot_2d(x, y_values)
```

Результат:

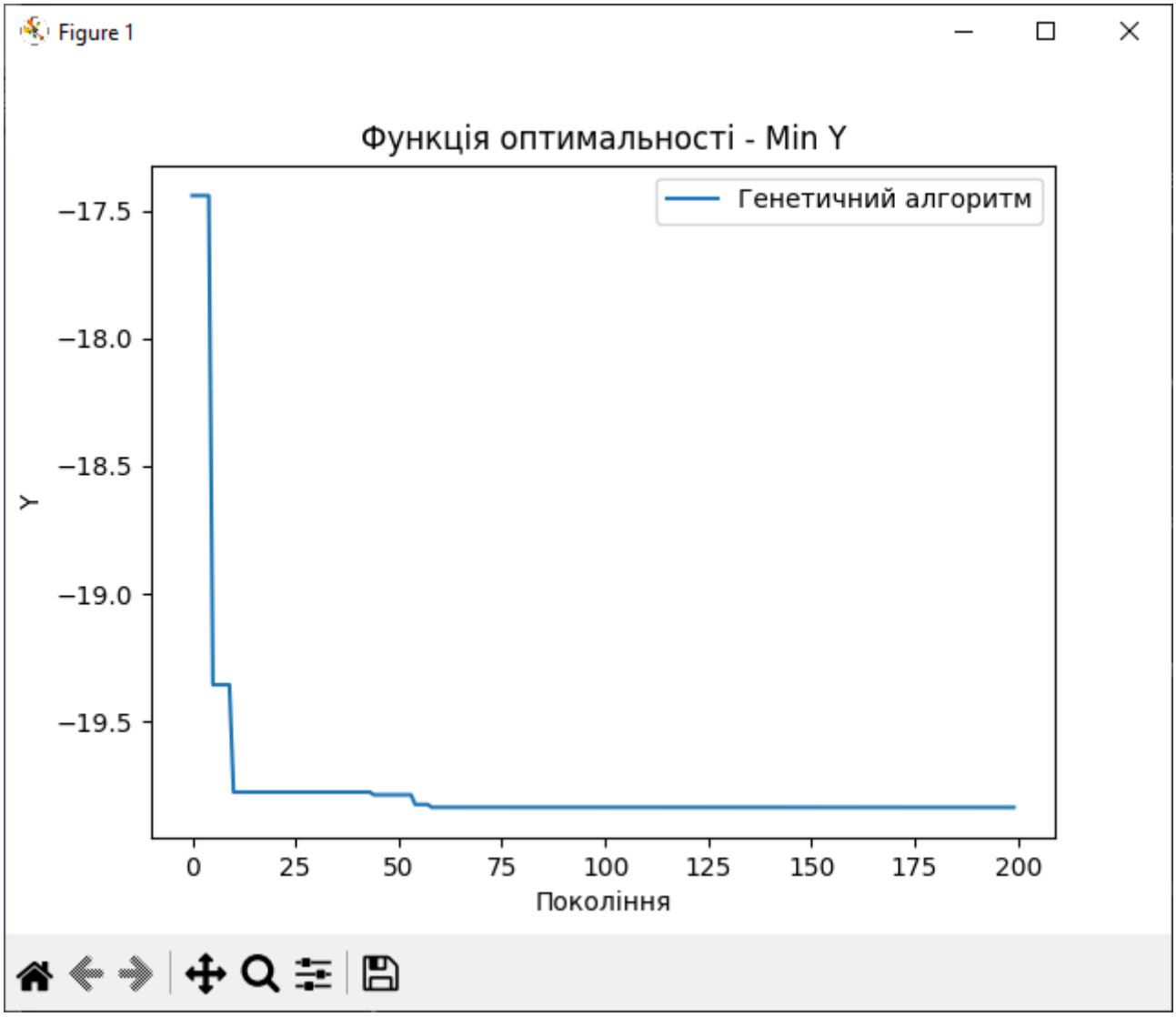

Рисунок 2 – Значення функції оптимальності **для мінімізації Y** крізь покоління

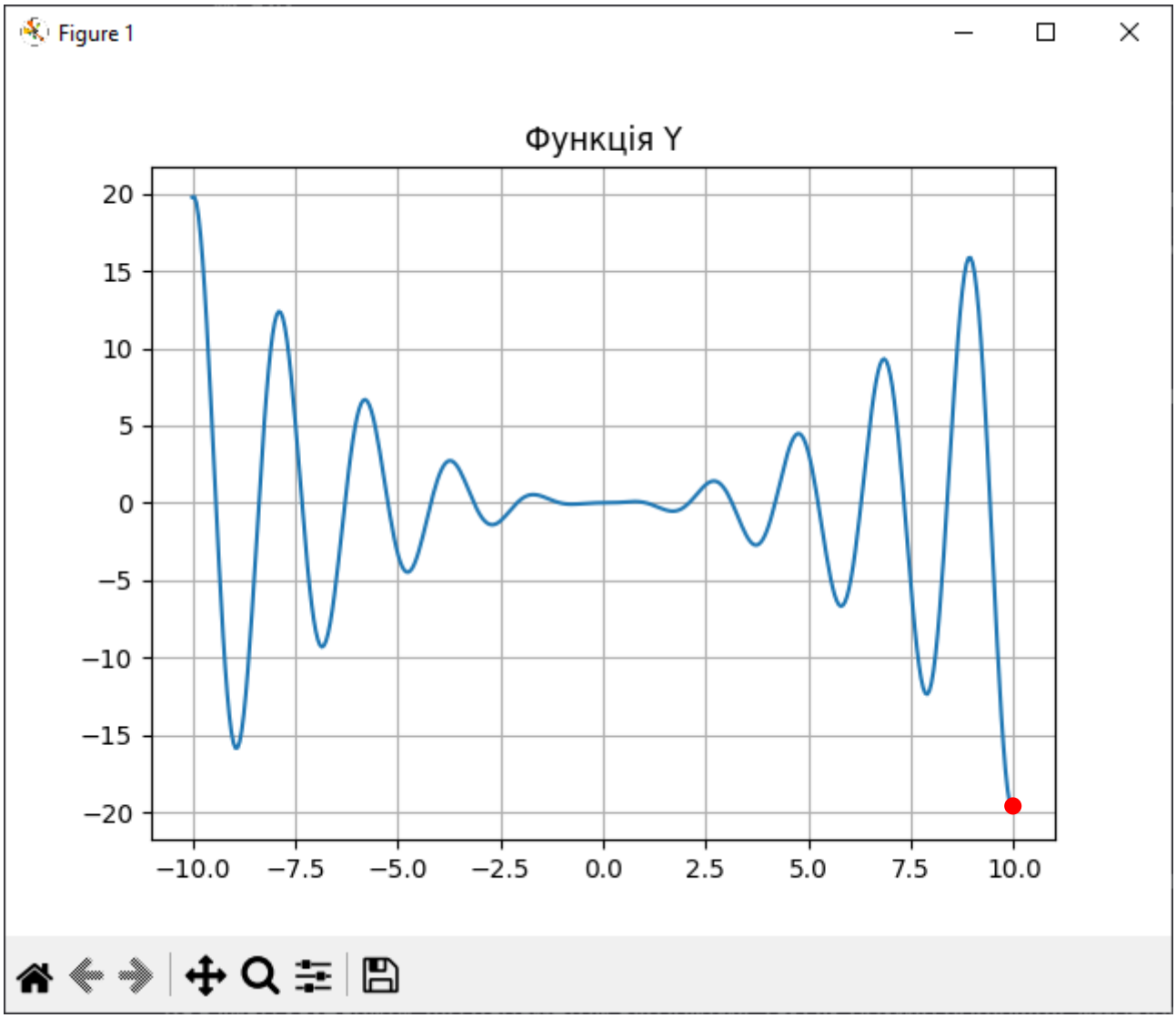

Рисунок 3 – Мінімум функції Y

Задача 2: Максимізувати функцію двох змінних

Знайдемо максимум функції двох змінних за допомогою програмних засобів Python та за допомогою генетичного алгоритму. Для генетичного алгоритму використаємо наступні параметри:

Особин у поколінні – 20,

Кількість поколінь – 50,

Дітей у поколінні – 10.

Порівняємо отримані різними методами результати.

Лістинг:

```
# Знаходження max для функції Z
def find_max():
    # Вбудований метод
    z_max = np.max(z_values)

    # Використання генетичного алгоритму
    z_genetic = Genetic_Algorithm(z_func, variables=2, bounds=[(-10, 10), (-10, 10)])
    z_max_genetic = z_genetic.run(50, 10)

    return z_max, z_func(*z_max_genetic)

# Графік функції Z
def plot_3d(x_grid, y_grid, z_values):
    ax = plt.subplot(projection='3d')
    ax.plot_surface(x_grid, y_grid, z_values, cmap='viridis', edgecolor='none')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('z_func(x, y)')
    ax.view_init(azim=30, elev=10)
    plt.title('Функція Z')
    plt.tight_layout()
    plt.show()

# Головні виклики
if __name__ == "__main__":
    [...]

    # Знаходження min та max функцій
    z_max, z_max_genetic = find_max()

    # Створення таблиці порівняння результатів
    data = {
        'Max Z': [z_max_genetic, z_max, abs(z_max_genetic - z_max)]
    }
    df = pd.DataFrame(data, index=['Генетичний алгоритм', 'Реальне значення', 'Похибка'])
    # Виведення таблиці
    print('\nРезультат алгоритму:')
    print(df)

    # Виведення графіків
    plot_3d(x_grid, y_grid, z_values)
```

Результат:

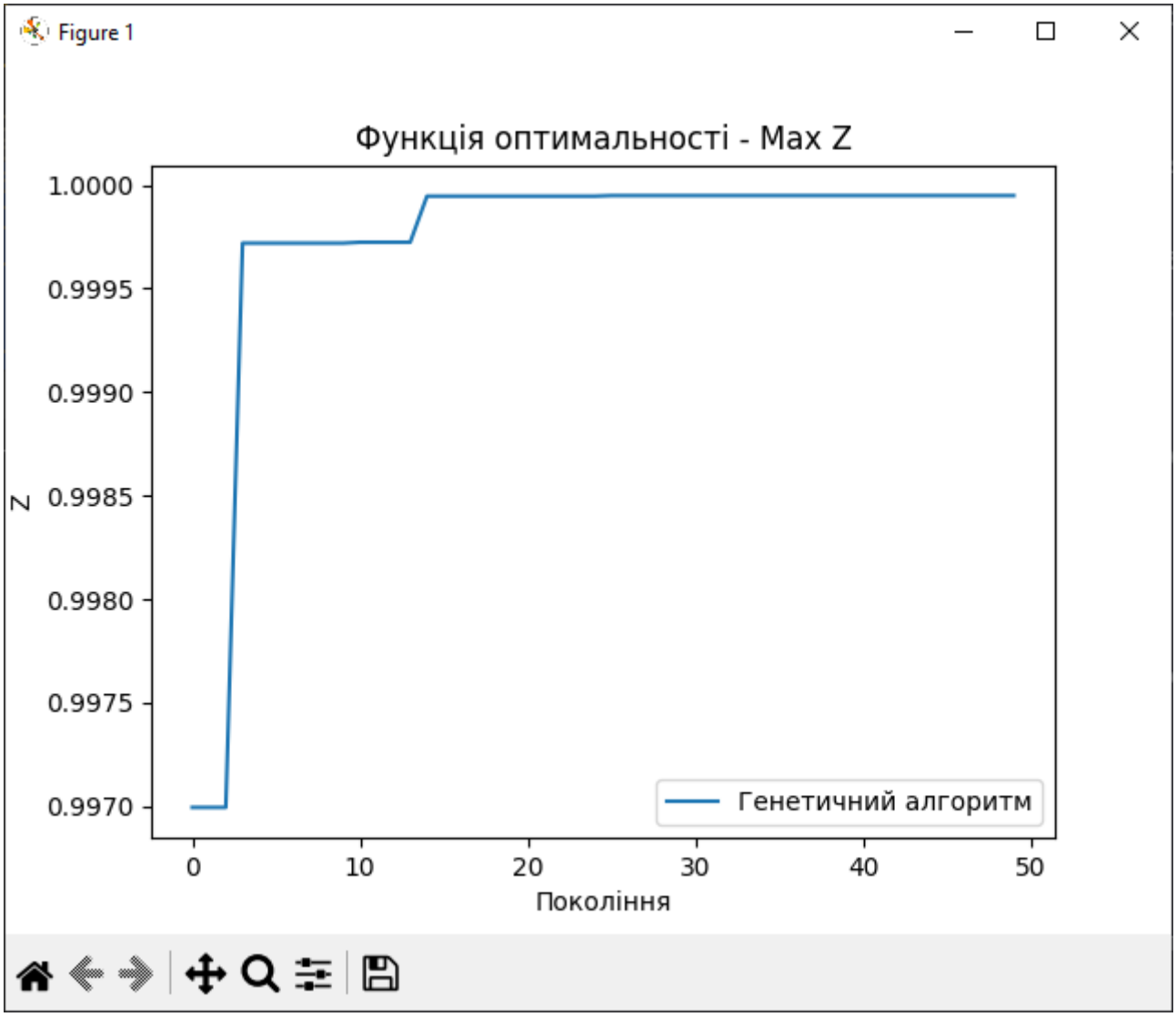

Рисунок 4 – Значення функції оптимальності **для максимізації Z** крізь покоління

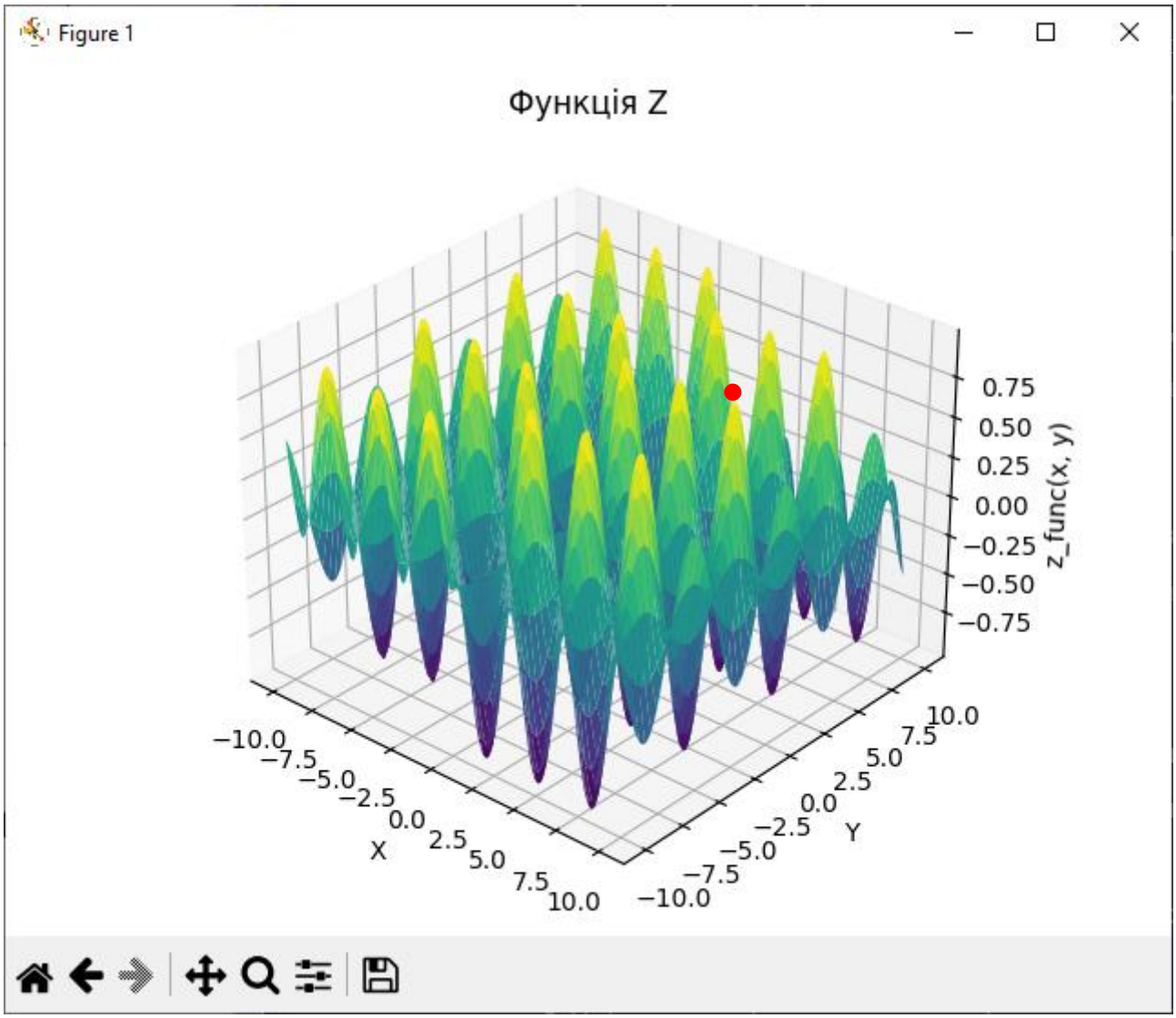

Рисунок 5 – Максимум функції Z

Порівняння результатів

Перевіримо працездатність розробленого генетичного алгоритму знайшовши абсолютну похибку.

```
Результат алгоритму:
                         Min Y     Max Z
Генетичний алгоритм -19.837932  0.999949
Реальне значення    -19.799801  0.999967
Похибка               0.038131  0.000017
```

Рисунок 6 – Таблиця порівняння

Бачимо, що значення значення генетичного алгоритму доволі близькі до реальних. Отже **розроблений генетичний алгоритм працює справно**.

Висновок:

Під час виконання лабораторної роботи було розроблено генетичний алгоритм для роботи з функціями однієї та двох змінних.

Розроблений алгоритм гнучкий до набору параметрів і успішно виконує оптимізацію за критеріями $min()$ та $max()$ функцій однієї та двох змінних.