МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-наукового інституту атомної і теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ЗВІТ

ДО ВИКОНАННЯ ПРАКТИЧНОГО ЗАВДАННЯ №3

з дисципліни

«МЕТОДОЛОГІЯ РОЗРОБКИ ІНТЕЛЕКТУАЛЬНИХ КОМП'ЮТЕРНИХ ПРОГРАМ »

Тема: «Оптимізація функції із застосуванням генетичних алгоритмів»

Варіант 16

КИЇВ 2025

Мета: Написати програму оптимізації функції з використанням генетичного алгоритму

Опис дії алгоритму

Алгоритм імітує природний добір для знаходження максимуму та мінімуму функції $f(x) = \cos(x^2)/x$ на інтервалі [0.001, 5]. Кожна особина (хромосома) представляє значення x у бінарному вигляді. Алгоритм починає з випадкової популяції, оцінює пристосованість (значення функції), а потім повторює цикл: відбір кращих, схрещування пар, мутації та створення нової популяції.

На кожній ітерації оцінюється похибка відносно істинного екстремуму (знайденого шляхом перебору). Алгоритм поступово зближується до оптимального значення, що візуалізується графіком збіжності. Результатом є знайдені значення x, при яких функція досягає максимуму та мінімуму, а також середня квадратична похибка (MSE).

Лістинг програми

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


# Кодування дійсного числа у бінарний рядок заданої довжини
def encode(x, a, b, n_bits):
    max_int = 2**n_bits - 1
    ratio = (x - a) / (b - a)
    int_val = int(ratio * max_int)
    return format(int_val, f'0{n_bits}b')


# Декодування бінарного рядка у дійсне число з врахуванням меж [a, b]
def decode(chromosome, a, b):
    max_int = 2**len(chromosome) - 1
    int_val = int(chromosome, 2)
    return a + (b - a) * int_val / max_int


# Оцінка пристосованості (fitness): значення функції або його мінус (для мінімізації)
def fitness(chromosome, func, a, b, optimize_for):
    x = decode(chromosome, a, b)
```

```
    val = func(x)
    return val if optimize_for == 'max' else -val

# Мутація хромосоми: з ймовірністю mutation_rate змінюємо кожен біт
def mutate(chromosome, mutation_rate):
    result = []
    for bit in chromosome:
        if random.random() < mutation_rate:
            result.append('1' if bit == '0' else '0')  # інвертуємо біт
        else:
            result.append(bit)
    return ''.join(result)

# Одноточковий кросовер: обмінюємо частини хромосом
def crossover(p1, p2):
    point = random.randint(1, len(p1)-1)
    return p1[:point] + p2[point:], p2[:point] + p1[point:]

# Обчислення справжнього екстремуму функції перебором з малим кроком
def calc_true_extreme(func, a, b, precision, optimize_for):
    xs = []
    x = a
    step = precision / 10
    while x < b:
        xs.append(x)
        x += step
    ys = [func(x) for x in xs]
    if optimize_for == 'max':
        idx = max(range(len(ys)), key=ys.__getitem__)
    else:
        idx = min(range(len(ys)), key=ys.__getitem__)
    return xs[idx], ys[idx]
```

```
# Основна функція генетичного алгоритму
def genetic_algorithm(func, a, b, precision, pop_size=50, epochs=100,
                      mutation_rate=0.01, optimize_for='max'):

    # Визначення кількості біт для кодування
    n_bits = math.ceil(math.log2((b - a) / precision))

    # Початкова популяція з випадкових особин
    population = [encode(random.uniform(a, b), a, b, n_bits) for _ in range(pop_size)]

    # Справжнє значення максимуму або мінімуму
    true_x, true_y = calc_true_extreme(func, a, b, precision, optimize_for)
    error_history = []

    # Основний цикл еволюції
    for _ in range(epochs):
        # Оцінка популяції
        scored = [(ind, fitness(ind, func, a, b, optimize_for)) for ind in population]
        scored.sort(key=lambda x: x[1], reverse=True)

        # Збереження поточної похибки (квадрат різниці від істинного значення)
        best_fitness = scored[0][1]
        error = ((best_fitness if optimize_for=='max' else -best_fitness) - true_y )**2
        error_history.append(error)

        # Турнірна селекція (вибір кращого з двох випадкових)
        def tournament():
            i1, i2 = random.sample(range(pop_size), 2)
            return population[i1] if fitness(population[i1], func, a, b, optimize_for) >
fitness(population[i2], func, a, b, optimize_for) else population[i2]

        new_population = []
        # Створення нової популяції шляхом схрещування і мутації
```

```
        while len(new_population) < pop_size:
            p1 = tournament()
            p2 = tournament()
            c1, c2 = crossover(p1, p2)
            c1 = mutate(c1, mutation_rate)
            c2 = mutate(c2, mutation_rate)
            new_population.extend([c1, c2])

        population = new_population[:pop_size]

    # Пошук найкращого результату в фінальній популяції
    best_chromosome = max(population, key=lambda ch: fitness(ch, func, a, b, optimize_for))
    best_x = decode(best_chromosome, a, b)
    best_y = func(best_x)

    return best_x, best_y, error_history, (true_x, true_y)

# Побудова графіку функції з відмітками знайдених максимуму і мінімуму
def plot_function_with_extrema(func, a, b, precision, max_res, min_res):
    x_vals = []
    x = a
    step = precision
    while x < b:
        x_vals.append(x)
        x += step
    y_vals = [func(x) for x in x_vals]

    plt.figure(figsize=(10, 6))
    plt.plot(x_vals, y_vals, label='f(x)', linewidth=2)

    plt.scatter(max_res[0], max_res[1], color='red', s=80, label='Max', zorder=5)
    plt.annotate(f'Max\nx={max_res[0]:.3f}\ny={max_res[1]:.3f}',
                 (max_res[0], max_res[1]), textcoords="offset points", xytext=(0, 10),
```

```
            ha='center', color='red')

    plt.scatter(min_res[0], min_res[1], color='blue', s=80, label='Min', zorder=5)
    plt.annotate(f'Min\nx={min_res[0]:.3f}\ny={min_res[1]:.3f}',
            (min_res[0], min_res[1]), textcoords="offset points", xytext=(0, -25),
            ha='center', color='blue')

    plt.title("Function with Found Maximum and Minimum")
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# Побудова графіку історії похибки (квадрат помилки на кожній ітерації)
def plot_error_history(error_history, label):
    plt.figure(figsize=(8, 5))
    plt.plot(error_history, label=label, color='orange')
    plt.xlabel("Epoch")
    plt.ylabel("Squared Error")
    plt.title(f"Convergence to True Optimum ({label})")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

# Цільова функція для оптимізації
def target_function(x):
    return math.cos(x**2) / x

# Параметри
precision = 0.001
```

```
a, b = precision, 5
pop_size = 30
epochs = 100
mutation_rate = 0.02

# Запуск алгоритму для максимуму
max_x, max_y, max_error_hist, true_max = genetic_algorithm(target_function, a, b, precision,
                                                           pop_size=pop_size,
                                                           epochs=epochs,
                                                           mutation_rate=mutation_rate,
                                                           optimize_for='max')

# Запуск алгоритму для мінімуму
min_x, min_y, min_error_hist, true_min = genetic_algorithm(target_function, a, b, precision,
                                                           pop_size=pop_size,
                                                           epochs=epochs,
                                                           mutation_rate=mutation_rate,
                                                           optimize_for='min')

# Виведення результатів
print(f"Max: x={max_x:.4f}, f(x)={max_y}")
print(f"Min: x={min_x:.4f}, f(x)={min_y}")

# Побудова графіків похибок
plot_error_history(max_error_hist, "Maximization")
plot_error_history(min_error_hist, "Minimization")

# Обчислення середньої квадратичної похибки
final_mean_error = (max_error_hist[-1] + min_error_hist[-1]) / 2
print(f"Final Mean Squared Error: {final_mean_error:.20f}")

# Побудова функції з позначенням максимуму і мінімуму
plot_function_with_extrema(target_function, a, b, precision,
```

max_res=(max_x, max_y),

min_res=(min_x, min_y))

Тестування

Протестуємо наший алгоритм на різних параметрах. Почнемо з кількості епох. На таблиці нижче наведено залежність значення середньо-квадратичної помилки від кількості епох(розмір популяції становить 30):

Кількість епох	Середньо-квадратична помилка
10	151070.137
20	71820.21
30	0.00000000633093327689
50	0.00000000633093327689
100	0.00000000000453725742

Як ми бачимо точність результату стрімко збільшується зі збільшенням кількості епох приблизно до 30 епох. Далі зміна є незначною

Тепер перевіримо як залежить значення помилка від розміру популяції(кількість епох = 30):

Розмір популяції	Середньо-квадратична помилка
10	283639.34176409780047833920
20	0.00000000633093327689
30	0.00000000000007525815
40	0.00000000000002546521

Як ми бачимо точність результату стрімко збільшується зі збільшенням розміру популяції приблизно до 20 . Далі зміна є незначною

Висновок

Генетичний алгоритм ефективно знаходить наближені максимуми і мінімум цільової функції за допомогою еволюційних операцій — відбору, схрещування та мутації. Використання турнірного відбору дозволяє підтримувати баланс між дослідженням і експлуатацією, що сприяє поступовому покращенню якості рішень. Алгоритм демонструє стабільну збіжність і може бути застосований для оптимізації складних функцій із заданою точністю.