Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра комп’ютерного моделювання процесів і систем

ЗВІТ

з лабораторної роботи №1

“Застосування генетичного алгоритму для оптимізації функції”

з курсу

«Нейронні мережі та методи проектування»

Виконав: студент групи ІКМ-М222к  Черкас Ю.В.

Перевірив: професор, д.т.н.  Успенський В.Б.

Харків 2023р

**Постановка задачі**

Розробити програму, яка за допомогою генетичного алгоритму знаходить максимум функції у заданому діапазоні. Розмір популяції встановити таким, як буде зручно, але не менше, ніж 50 особин. Точність 0.1, кількість кроків (ітерацій) не менше 500. Результати надати у вигляді тексту програми та графіку найкращих значень функції на кожній ітерації. В кінці - координати найкращої точки.

Таблиця 1. Вихідні дані

|  |  |  |  |
| --- | --- | --- | --- |
| Номер  варіанту | Функція , яку максимізуємо | Діапазон [a,b] для змінної x | Діапазон [c,d] для змінної y |
| 15 |  | [0;6] | [-4;0] |

**Виконання**

Фітнес-функцію виберемо рівною цільовій:

import numpy as np

def fitness\_function(ind):

    return 3 \* np.abs(ind.x) - ind.y\*\*2

тобто буде збільшуватися зі збільшенням . Виберемо кількість особин N=50. Кожна особина матиме по 2 хромосоми (x\* і y\*) , що відповідають аргументам цільової функції x та y.

Для зручності створимо допоміжний клас Individual.

class Individual:

    def \_\_init\_\_(self, x, y):

        self.x = x

        self.y = y

Наведемо допоміжні методи, котрі використовуються в подальших розрахунках.

import numpy as np

from numpy.random import randint

from numpy.random import rand

def decode(min, max, bitstring):

    n\_bits = len(bitstring)

    chars = ''.join([str(s) for s in bitstring])

    quanted\_value = int(chars, 2)

    value = min + quanted\_value \* ((max - min)/(2\*\*n\_bits - 1))

    return value

def selectOne(individuals, scores):

    scores\_for\_selection = scores.copy()

    min\_score = np.min(scores\_for\_selection)

    if min\_score < 0:

        scores\_for\_selection = scores\_for\_selection + 2 \* np.abs(min\_score)

    max = sum(scores\_for\_selection)

    selection\_probability = [c/max for c in scores\_for\_selection]

    return individuals[np.random.choice(len(individuals), p=selection\_probability)]

def crossover(p1, p2, crossover\_probability):

    if rand() < crossover\_probability:

        point\_x = randint(1, len(p1.x)-2)

        point\_y = randint(1, len(p1.x)-2)

        c1 = Individual(p1.x[:point\_x] + p2.x[point\_x:], p1.y[:point\_y] + p2.y[point\_y:])

        c2 = Individual(p2.x[:point\_x] + p1.x[point\_x:], p2.y[:point\_y] + p1.y[point\_y:])

        return [c1, c2]

    c1, c2 = Individual(p1.x, p1.y), Individual(p2.x, p2.y)

    return [c1, c2]

def mutation(ind, mutation\_probability):

    for i in range(len(ind.x)):

        if rand() < mutation\_probability:

            ind.x[i] = 1 - ind.x[i]

    for i in range(len(ind.y)):

        if rand() < mutation\_probability:

            ind.y[i] = 1 - ind.y[i]

Кінцева програма знаходження максимуму функції за допомогою генетичного алгоритму на мові Python буде мати вигляд:

import numpy as np

import matplotlib.pyplot as plt

from numpy.random import randint

from helpers import Individual, fitness\_function, decode, selectOne, crossover, mutation

q = 1 # precision (number of digits after the dot)

N\_epochs = 500

N\_individuals = 50

a = 0

b = 6

c = -4

d = 0

crossover\_probability = 0.9

mutation\_probability = 0.01

Lx = np.int(np.ceil(np.log2((b - a) \* 10\*\*q + 1)))

Ly = np.int(np.ceil(np.log2((d - c) \* 10\*\*q + 1)))

# initial population

individuals = [Individual(randint(0, 2, Lx).tolist(), randint(0, 2, Ly).tolist()) for \_ in range(N\_individuals)]

best\_ind, best\_score = 0, fitness\_function(Individual(decode(a, b, individuals[0].x), decode(c, d, individuals[0].y)))

F\_max\_per\_epoch, F\_avg\_per\_epoch = [], []

for epoch in range(N\_epochs):

    decoded = [Individual(decode(a, b, ind.x), decode(c, d, ind.y)) for ind in individuals]

    scores = [fitness\_function(d) for d in decoded]

    F\_max = 0

    F\_sum = 0

    for i in range(N\_individuals):

        F\_sum += scores[i]

        if scores[i] > best\_score:

            best\_ind, best\_score = individuals[i], scores[i]

            print(f'{epoch} epoch: new best f({decoded[i].x}, {decoded[i].y}) = {scores[i]}')

        if scores[i] > F\_max:

            F\_max = scores[i]

    F\_avg = F\_sum / N\_individuals

    F\_avg\_per\_epoch.append(F\_avg)

    F\_max\_per\_epoch.append(F\_max)

    # select parents

    selected = [selectOne(individuals, scores) for \_ in range(N\_individuals)]

    # create the next generation

    children = list()

    for i in range(0, N\_individuals, 2):

        # get selected parents in pairs

        p1, p2 = selected[i], selected[i+1]

        # crossover and mutation

        for child in crossover(p1, p2, crossover\_probability):

            # mutation

            mutation(child, mutation\_probability)

            # store for next generation

            children.append(child)

    # replace population

    individuals = children

best\_decoded = Individual(decode(a, b, best\_ind.x), decode(c, d, best\_ind.y))

print(f'Best case: f({best\_decoded.x}, {best\_decoded.y})) = {best\_score}')

x = np.arange(0, 500, 1)

plt.plot(x, F\_max\_per\_epoch, 'r')

plt.plot(x, F\_avg\_per\_epoch, 'b')

plt.legend([r'Fmax - максимальна пристосованість у популяції', 'Favg - середня пристосованість по популяції'])

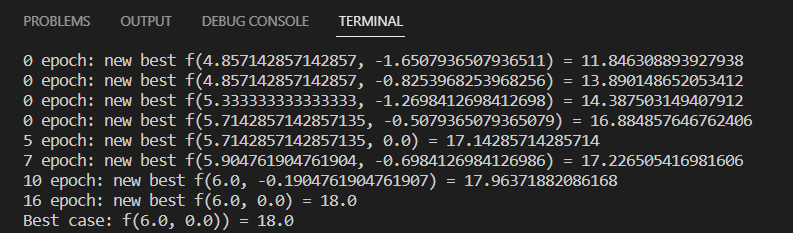
plt.grid(True)

plt.ylabel(r'F')

plt.xlabel('Ітерація')

plt.show()

Нижче наведений результат роботи програми, котрий був виведений в термінал:



На рисунку 1 зображені графіки максимальної пристосованості у популяції та середньої пристосованості по популяції для кожної ітерації роботи програми.

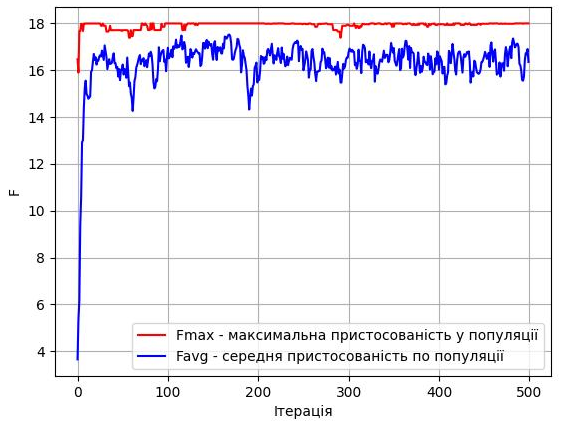


Рисунок 1 – Графік значень середньої та максимальної пристосованості в залежності від ітерації алгоритму

З рисунку 1 та з даних виведених в термінал бачимо, що максимум фітнес-функції набуває на 16 ітерації в точці , , в котрій він має величину .

**Висновки**

На даній лабораторній роботі ми дослідили роботу генетичного алгоритму на прикладі знаходження максимуму функції в заданих діапазонах змін аргументів. Реалізували даних алгоритм на мові Python та проаналізували його роботу при різних початкових умовах та налаштуваннях.

Враховуючи простоту функції, генетичний алгоритм вже на 16-й ітерації знайшов максимум функції: , , . По формулі досліджуваної функції () та по аналізованих діапазонах (, ), ми бачимо, що отримані результати відповідають дійсності, а отже, генетичний алгоритм доказав коректність своєї роботи.