**Пономаренко, ІКМ-220Б**

**Лабораторна робота 1**

**Застосування генетичного алгоритму для оптимізації функції**

**Мета роботи** − за наведеним прикладом реалізувати обчислення за генетичним алгоритмом.

**Хід роботи**

**Постановка задачі:** Необхідно 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.

Перш за все, для реалізації програми я використав мову програмування Python та бібліотеку Matplotlib для виводу графіків. Моя програма призначена для розв'язування задач оптимізації, де потрібно максимізувати функцію. В моєму випадку це – f = x+y, де x знаходиться на проміжку [-2,4], а y – [0,4]. Для цього я використовував генетичний алгоритм.

Основні функції програми включають:

* Генерація початкової популяції
* Обчислення фітнес-функції для кожної особини в популяції
* Відбір найкращих особин для наступного покоління
* Схрещування та мутація відібраних особин
* Створення нової популяції
* Повторення циклу до досягнення критерію зупинки

На рис. 1 показана блок-схема роботи алгоритму.

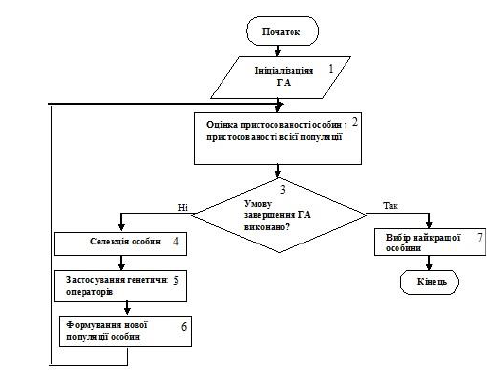


Рисунок 1 - блок-схема роботи генетичного алгоритму

Нижче наведено клас моєї програми на python.

**import** **math**

**import** **random**

**from** **typing** **import** Union, List, Tuple, Callable

**class** **GeneticAlgorithm**:

**def** **\_\_init\_\_**(

self,

func: Callable,

x\_int: Tuple[float, float],

y\_int: Tuple[float, float],

q: float, # точність

N: int, # кількість особин

iters: int, # кількість ітерацій

pc: float = **0.95**, # ймовірність схрещування [0.9, 1]

pm: float = **0.005**, # ймовірність мутації [0, 0.01]

be: float = **0.005** # [0, 0.01]

) -> None:

self.func = func

self.x\_int = x\_int

self.y\_int = y\_int

self.N = N

self.iters = iters

self.pc = pc

self.pm = pm

self.be = be

# Кращий чел

self.best\_individ: Union[int, None] = None

# Кращий результат

self.max\_fitness: Union[int, None] = None

# список пристосованості на ітераціях

self.F\_list: List[float] = []

# тут буду хранить лучшие значения функции на каждой итерации

self.best\_fitness: List[float] = []

# тута будуть зберігатися x, y

self.roots\_list: List[List[float]] = []

a, b = self.x\_int

c, d = self.y\_int

# шукається рішення з точністю q знаків після коми

q = len(str(q).split('.')[**1**])

# Найменше натуральне число L

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

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

# N - число підінтервалів розбиття

Nx = **2** \*\* self.Lx - **1**

Ny = **2** \*\* self.Ly - **1**

# крок квантування інтервалу a, b

self.hx = (b - a) / Nx

self.hy = (d - c) / Ny

# Початкові значення хромосом особин

# Find = []

# ind = []

# x\_num = 4

# y\_num = 3

# x\_step = (b - a) / x\_num

# y\_step = (d - c) / y\_num

# for i in range(1, x\_num):

# for j in range(y\_num-1, 0, -1):

# x = a + i \* x\_step

# y = c + j \* y\_step

# ind.append([x, y])

# Find.append(self.F(x, y))

Find = []

ind = []

**for** i **in** range(**1**, Nx):

**for** j **in** range(Ny-**1**, **0**, -**1**):

x = a + i \* self.hx

y = c + j \* self.hy

ind.append([x, y])

Find.append(self.F(x, y))

self.ind = ind

self.Find = Find

# fitness функція

**def** **F**(self, x: float, y: float):

**return** self.func(x, y)

**def** **start**(self):

self.presence()

**for** i **in** range(self.iters):

# print(i, self.Find)

self.selection()

ind = []

**for** pair **in** self.parents:

children = self.crossover(pair[**0**], pair[**1**])

**for** child **in** children:

chromosome1 = self.mutate(child[**0**])

chromosome2 = self.mutate(child[**1**])

ind.append([chromosome1, chromosome2])

self.ch\_ind = ind

self.newPresence()

self.addBestFitness()

# print(i, self.ch\_ind)

**print**(i, self.Y, self.F\_, self.best\_fitness[-**1**])

# Обчислимо пристосованість усієї популяції

**def** **presence**(self) -> None:

self.Y = sum(y **for** y **in** self.Find)

self.F\_ = self.Y / self.N

a, b = self.x\_int

c, d = self.y\_int

# Обчислюємо значення хромосом особин

ind = []

**for** x, y **in** self.ind:

x\_ = math.ceil((x - a) / self.hx)

y\_ = math.ceil((y - c) / self.hy)

ch\_x = bin(x\_)[**2**:].rjust(self.Lx, '0')

ch\_y = bin(y\_)[**2**:].rjust(self.Ly, '0')

ind.append([ch\_y, ch\_x])

self.ch\_ind = ind

# print(self.ch\_ind)

# Виконаємо селекцію за допомогою рулетки

**def** **selection**(self) -> None:

Fmin = min(self.Find)

F\_ind = [val + **2** \* abs(Fmin) **for** val **in** self.Find]

self.Y\_ = sum(F\_ind)

self.F\_\_ = self.Y\_ / self.N

# print(Fmin, Y\_, F\_\_)

Sind = [val / self.Y\_ \* **100** **for** val **in** F\_ind]

S = []

x = **0**

**for** y **in** Sind:

S.append([x, x + y])

x += y

# Генеруємо випадковим чином 6 дійсних чисел з діапазону [0;100]

nums = [random.uniform(**0**, **100**) **for** i **in** range(self.N)]

parents = []

**for** num **in** nums:

i = **0**

**for** a, b **in** S:

**if** num >= a **and** num <= b:

parents.append(self.ch\_ind[i])

**break**

i += **1**

# розіб'ємо особин по парах

random.shuffle(parents)

self.parents = [

[parents[i], parents[i+**1**]]

**for** i **in** range(**0**, len(parents), **2**)

]

# Схрещування

**def** **crossover**(

self,

parent1: str,

parent2: str

) -> Tuple[str, str]:

L = len(parent1)

**if** random.random() <= self.pc:

K = random.randint(**1**, L - **1**)

child1 = parent1[:K] + parent2[K:]

child2 = parent2[:K] + parent1[K:]

**return** child1, child2

**else**:

**return** parent1, parent2

# Мутація

**def** **mutate**(

self,

chromosome: str

) -> str:

mutated\_chromosome = ''

**for** gen **in** chromosome:

**if** random.random() < self.pm:

mutated\_chromosome += '0' **if** gen == '1' **else** '1'

**else**:

mutated\_chromosome = chromosome

**break**

**return** mutated\_chromosome

# Обчислимо пристосованість нових особин, які формують нову популяцію

**def** **newPresence**(self):

a, b = self.x\_int

c, d = self.y\_int

Find = []

ind = []

best\_individ = self.best\_individ

max\_fitness = self.max\_fitness

**for** i **in** self.ch\_ind:

y\_ = int(i[**0**], **2**)

x\_ = int(i[**1**], **2**)

y = c + y\_ \* self.hy

x = a + x\_ \* self.hx

individ = [x, y]

fitness = self.F(x, y)

ind.append(individ)

Find.append(fitness)

**if** max\_fitness **is** None **or** max\_fitness < fitness:

best\_individ = individ

max\_fitness = fitness

self.best\_individ = best\_individ

self.max\_fitness = max\_fitness

# Модификация

zipped = list(zip(ind, Find, self.ch\_ind))

zipped\_sorted = sorted(zipped, key=**lambda** x: x[**1**], reverse=True)

ind\_s, Find\_s, ch\_ind\_s = zip(\*zipped\_sorted)

elite = list(ind\_s[:self.N - math.floor(self.N \* self.be)])

elite\_F = list(Find\_s[:self.N - math.floor(self.N \* self.be)])

elite\_chind = list(ch\_ind\_s[:self.N - math.floor(self.N \* self.be)])

new\_ind = elite

new\_chind = elite\_chind

new\_Find = elite\_F

**for** i **in** range(math.floor(self.N \* self.be)):

x = random.uniform(**0**, b)

y = random.uniform(**0**, d)

x\_ = math.ceil((x - a) / self.hx)

y\_ = math.ceil((y - c) / self.hy)

ch\_x = bin(x\_)[**2**:].rjust(self.Lx, '0')

ch\_y = bin(y\_)[**2**:].rjust(self.Ly, '0')

new\_ind.append([x, y])

new\_chind.append([ch\_y, ch\_x])

new\_Find.append(self.F(x, y))

self.ind = new\_ind

self.ch\_ind = new\_chind

self.Find = new\_Find

# self.ind = ind

# self.Find = Find

self.Y = sum(Find)

self.F\_ = self.Y / self.N

# Добавляем лучшее значение функции в массив

**def** **addBestFitness**(self):

self.F\_list.append(self.F\_)

self.best\_fitness.append(self.max\_fitness)

self.roots\_list.append(self.best\_individ)

# отримати найкращє значення функції

**def** **getAnswer**(self):

**return** {

'best\_fitness': max(self.best\_fitness),

'best\_roots': max(self.roots\_list)

}

Після запуску моєї програми, я отримав такі результати (рис. 2-4):

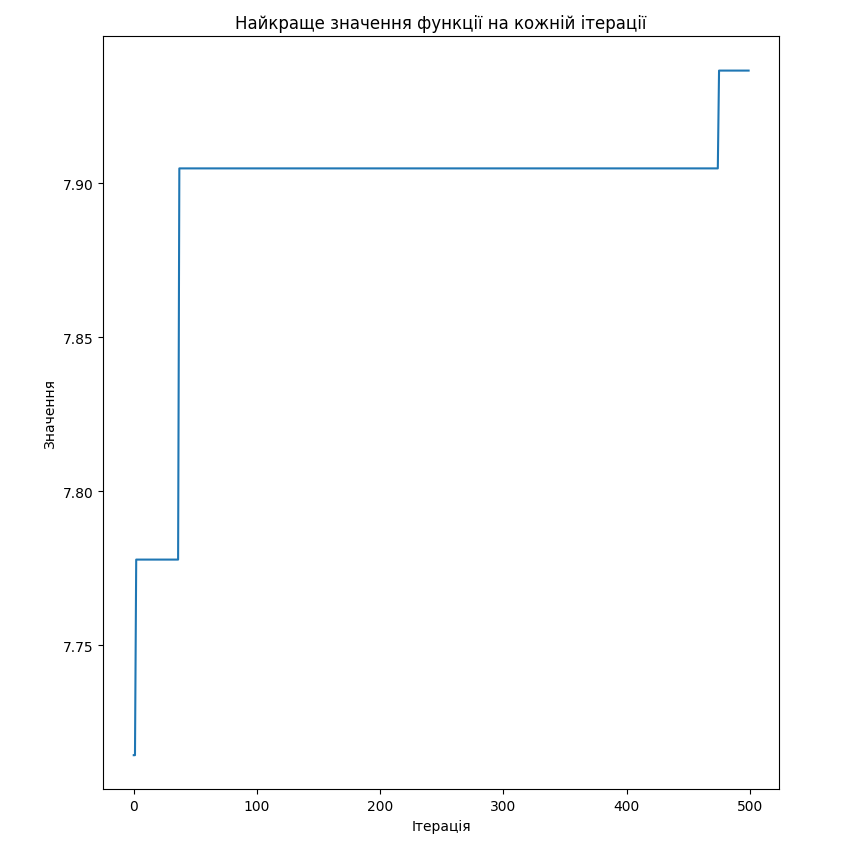


Рисунок 2 - графік найкращого значення функції на кожній ітерації

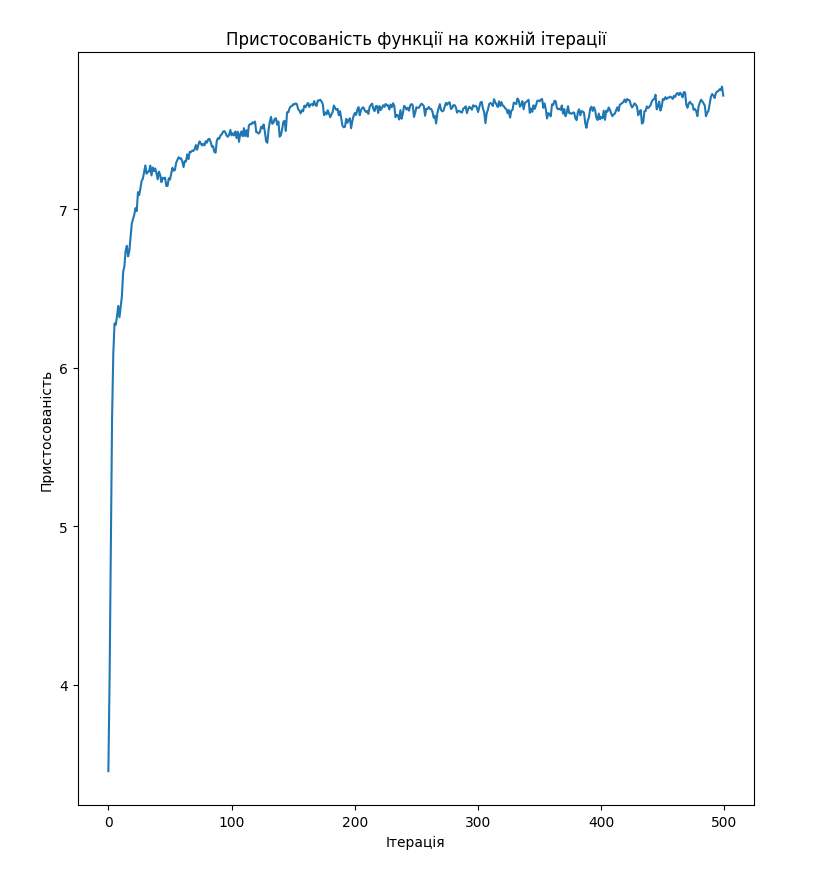


Рисунок 3 - графік пристосованості функції на кожній ітерації

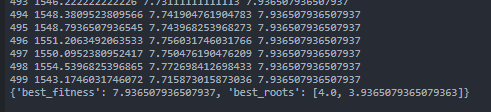


Рисунок 4 - отримані максимальне значення функції та її корені

Правильні значення: f(x, y) = 8.0 x = 4.0 y = 4.0

Похибки між отриманими числами та реальними:εf = εy ≈ 0.0635 εx = 0.0

**Висновки**

Генетичний алгоритм є ефективним методом розв'язання задач оптимізації, особливо тих, які мають багато локальних максимумів та складні для оптимізації.

Ефективна реалізація генетичного алгоритму включає в себе правильний вибір параметрів та функції цілі, а також налагодження процесу генерації популяції, відбору, схрещування та мутації.

Моя програма показала високу швидкість виконання та добрі результати оптимізації хоча функція в принципі була не важка. Отримані мною результати мають трохи меншу похибку ніж була задана (0.1).

Отже, реалізація програми для обчислення за генетичним алгоритмом була успішною та показала, що генетичний алгоритм є ефективним методом розв'язання задач оптимізації. Дана програма може бути використана для розв'язання різноманітних задач оптимізації в наукових дослідженнях та промислових додатках.