МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ

ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра математичних методів системного аналізу

ЗВІТ

про виконання лабораторної роботи № 1

з дисципліни «Еволюційні методи оптимізації»

з теми «Дослідження генетичного алгоритму

оптимізації багатоекстремальних функцій у дійсному просторі»

Виконала:

Студентка 3 курсу ІПСА

групи КА-24

Спекторовська Лада

Варіант №12

Київ-2024

Мета роботи: Дослідити ефективність генетичного методу при зміні його параметрів або елементів.

Програма реалізована власноруч мовою програмування Python. При виконанні було використано бібліотеки: NumPy, Pandas, Matplotlib, Random, Copy.

Використані параметри та елементи методу:

Параметри, що досліджуються: спосіб відбору батьківських точок для кросоверу або мутації (R), коефіцієнти відбору/кросоверу/мутації (C)

- Розмір популяції: 200.
- Коефіцієнти відбору/кросоверу/мутації: 20% - відбір, 70% - кросовер, 10% - мутація; 20% - відбір, 60% - кросовер, 20% - мутація; 10% - відбір, 80% - кросовер, 10% - мутація; 10% - відбір, 70% - кросовер, 20% - мутація.
- Спосіб генерації початкової популяції: рівномірний випадковий розподіл.
- Спосіб відбору точок для переходу в наступне покоління: елітний відбір
- Спосіб відбору батьківських точок для кросоверу або мутації: ранговий відбір, метод змагання (турніру) та випадковий.
- Спосіб кросоверу: природний рівномірний.
- Спосіб мутації: природна мутація.
- Максимальна кількість поколінь: 50 (100 для тестових запусків).
- Критерій завершення і його параметри: максимальна кількість поколінь – 50 (100 для тестових запусків).

Тестовий запуск для функції Drop-Wave:

- Коефіцієнти відбору/кросоверу/мутації: 20% - відбір, 70% - кросовер, 10% - мутація.
- Спосіб відбору батьківських точок для кросоверу або мутації: випадковий.

Оскільки в отриманій популяції 200 точок, виведемо перші 30:

X	Y	Z
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408

-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408

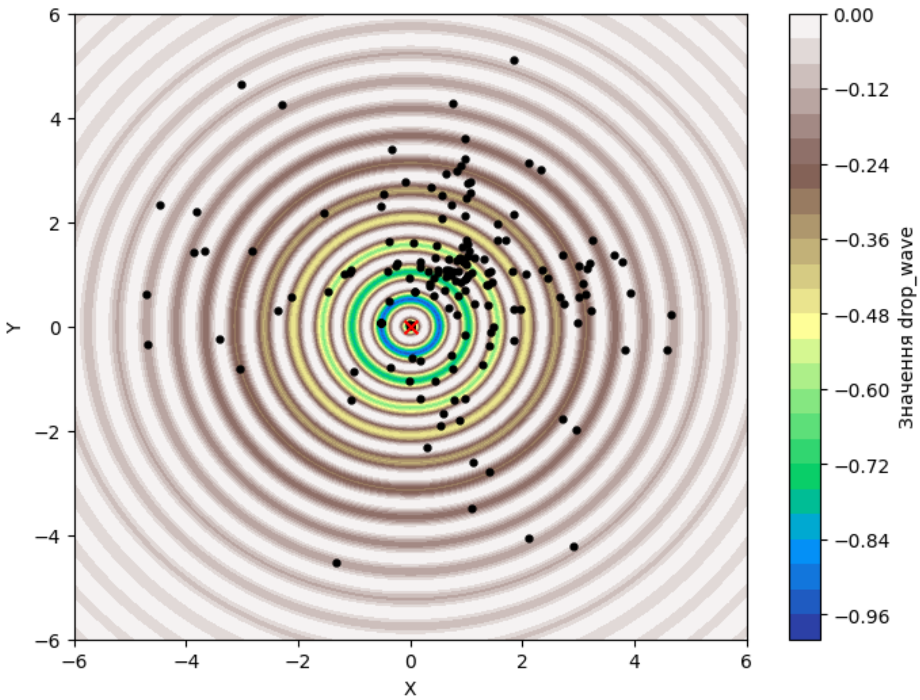

Рисунок №1. Результат тестового запуску для Drop-Wave.

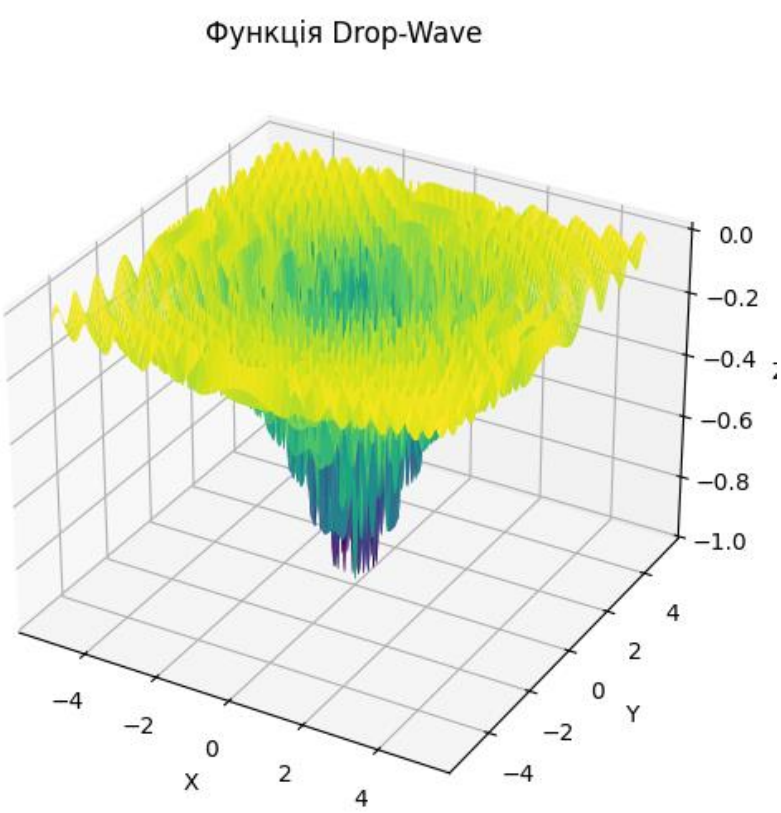

Рисунок №2. Функція Drop-Wave.

Тестовий запуск для функції Швефеля:

- Коефіцієнти відбору/кросоверу/мутації: 20% - відбір, 70% - кросовер, 10% - мутація.
- Спосіб відбору батьківських точок для кросоверу або мутації: ранговий.

Оскільки в отриманій популяції 200 точок, виведемо перші 30:

X	Y	Z
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837

423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
423,0922705	421,9061553	0,680284837
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408

-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408
-0,003500844	0,019197088	-0,986260408

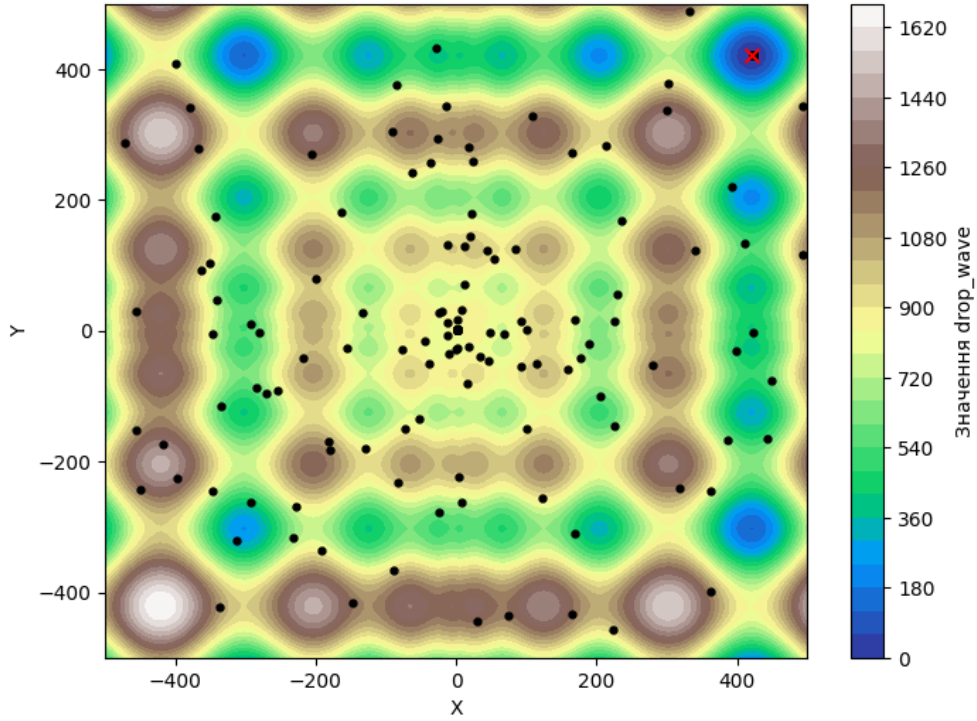

Рисунок №3. Результат тестового запуску для функції Швефеля.

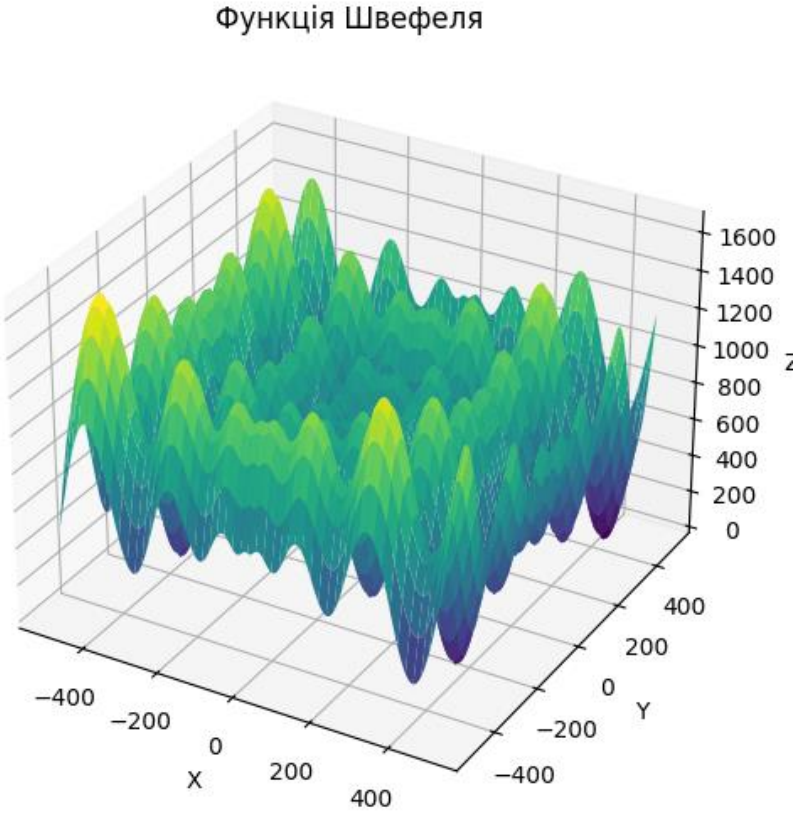

Рисунок №4. Функція Швефеля

Експерименти:

Надалі проведемо 200 експериментів для кожної функції. Ефективність проведених досліджень буде вимірюватись у наступних критеріях:

- Здатність знайти глобальний мінімум (або стабільність). Обчислюємо скільки зі 200 випадків метод потрапляє в епсілон окіл потрібної точки. Для функції Швефеля епсілон взято 10 для функції Drop-Wave - 0.6.
- Точність визначення глобального мінімуму. Обчислюємо середню відстань від найкращої точки методу на 200 експериментах, тобто чим менше ця відстань тим краще вправляється метод.
- Кількість підрахунків функції.

Розглянемо результати:

Target function	Selection rate	Cross rate	Mutation rate	Selection method	Stability	AvgDistance	Function counts
schwefel	0,20	0,70	0,10	ranked	105	10,3163324	10200
schwefel	0,20	0,70	0,10	tournament	13	239,9461313	10200
schwefel	0,20	0,70	0,10	random	108	10,36629951	10200
schwefel	0,20	0,60	0,20	ranked	88	25,28155711	10200
schwefel	0,20	0,60	0,20	tournament	10	263,5153407	10200
schwefel	0,20	0,60	0,20	random	83	15,35010972	10200
schwefel	0,10	0,80	0,10	ranked	125	9,203267906	10200
schwefel	0,10	0,80	0,10	tournament	13	241,663958	10200
schwefel	0,10	0,80	0,10	random	90	15,32071677	10200
schwefel	0,10	0,70	0,20	ranked	106	10,47871053	10200
schwefel	0,10	0,70	0,20	tournament	12	250,0934605	10200
schwefel	0,10	0,70	0,20	random	99	18,21107896	10200
drop_wave	0,20	0,70	0,10	ranked	200	0,218300274	10200
drop_wave	0,20	0,70	0,10	tournament	185	0,523990727	10200
drop_wave	0,20	0,70	0,10	random	200	0,248328462	10200
drop_wave	0,20	0,60	0,20	ranked	200	0,275044474	10200
drop_wave	0,20	0,60	0,20	tournament	190	0,536724353	10200
drop_wave	0,20	0,60	0,20	random	200	0,263903387	10200
drop_wave	0,10	0,80	0,10	ranked	200	0,198941241	10200
drop_wave	0,10	0,80	0,10	tournament	189	0,527847157	10200
drop_wave	0,10	0,80	0,10	random	200	0,218894277	10200
drop_wave	0,10	0,70	0,20	ranked	200	0,208822158	10200
drop_wave	0,10	0,70	0,20	tournament	189	0,527558167	10200
drop_wave	0,10	0,70	0,20	random	200	0,238080512	10200

Таблиця №3. Результати експериментів.

Візуалізуємо зміни стабільності та точності для обох функцій. Зауважимо, що group1, group2 та group3 це коефіцієнти відбору, кросоверу, мутації, тобто:

- group1: 10% - відбір, 70% - кросовер, 20% - мутація;
- group2: 10% - відбір, 80% - кросовер, 10% - мутація;
- group3: 20% - відбір, 60% - кросовер, 20% - мутація;
- group4: 20% - відбір, 70% - кросовер, 10% - мутація.

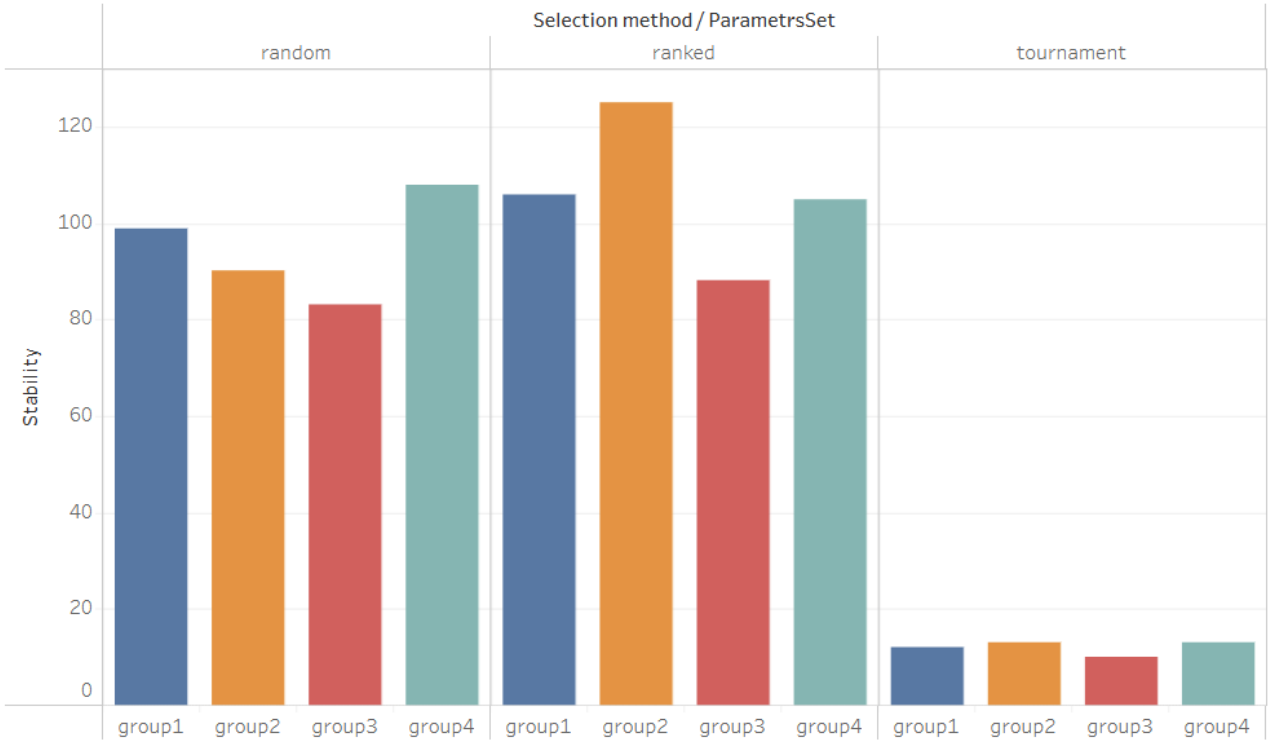

Рисунок №5. Функція Швефеля. Візуалізація змін стабільності.

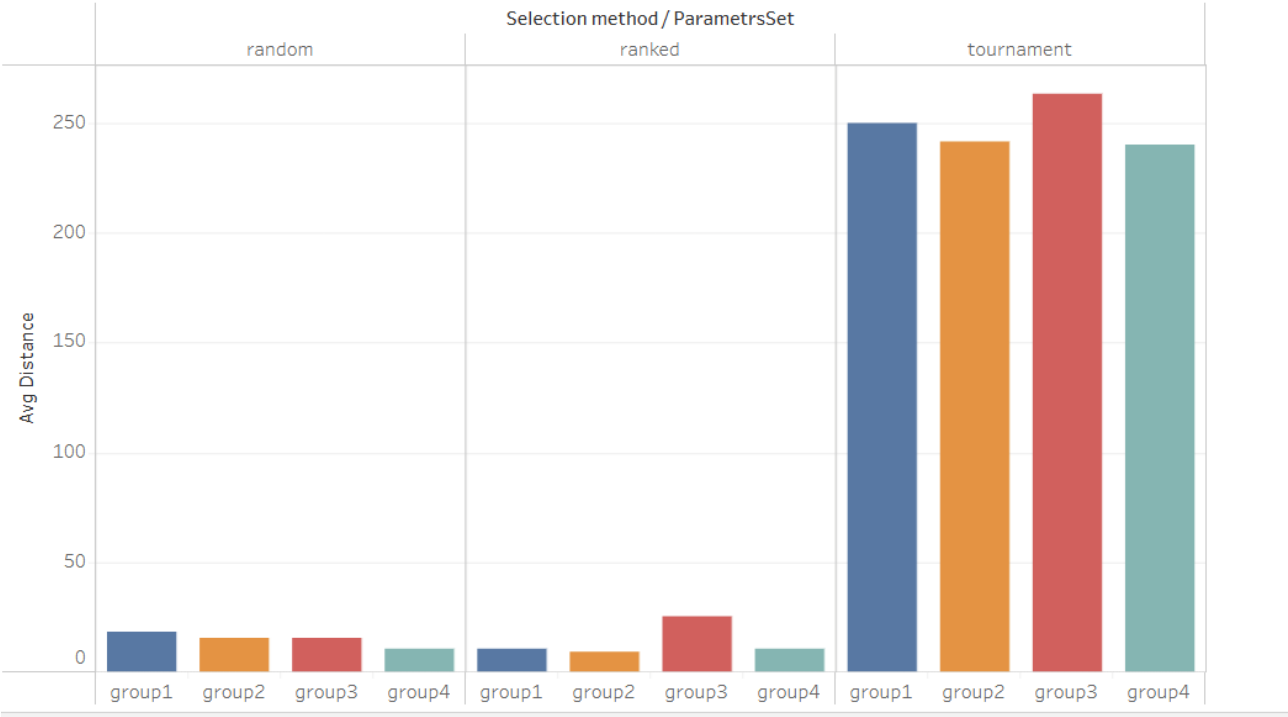

Рисунок №6. Функція Швефеля. Візуалізація змін точності.

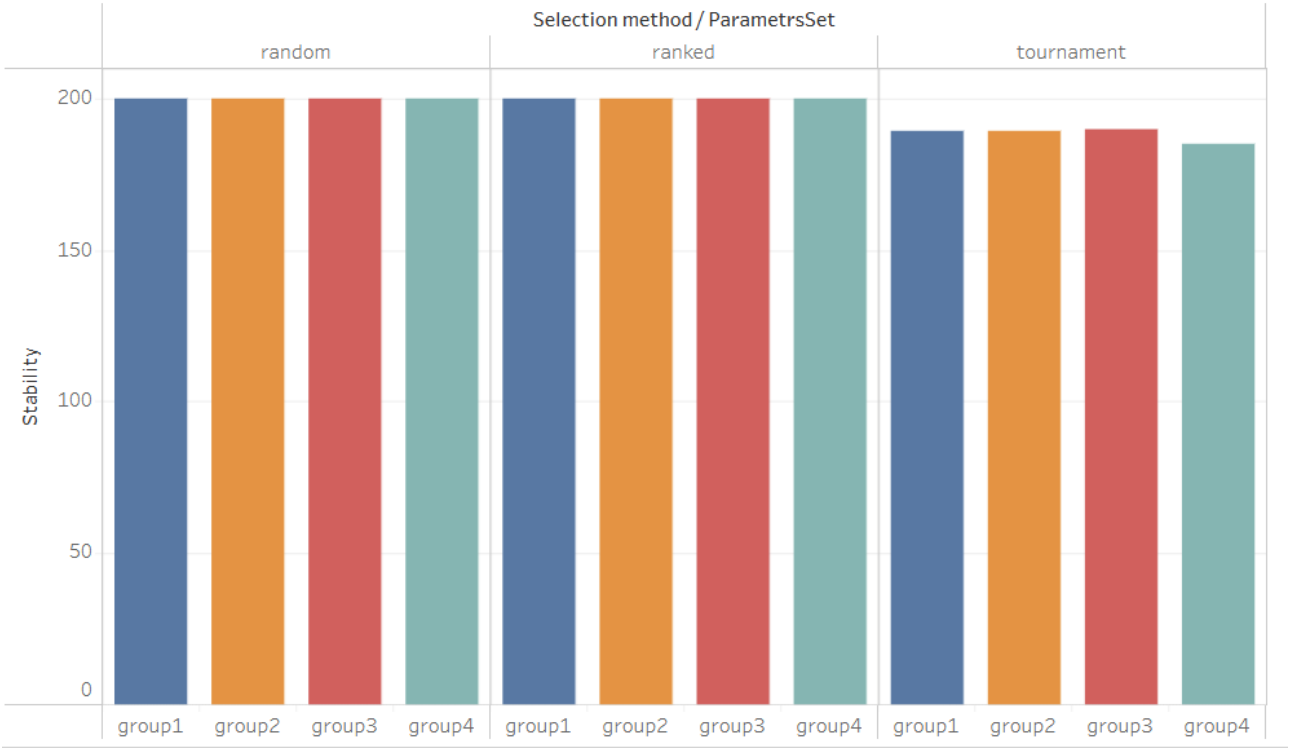

Рисунок №7. Функція Drop-Wave. Візуалізація змін стабільності.

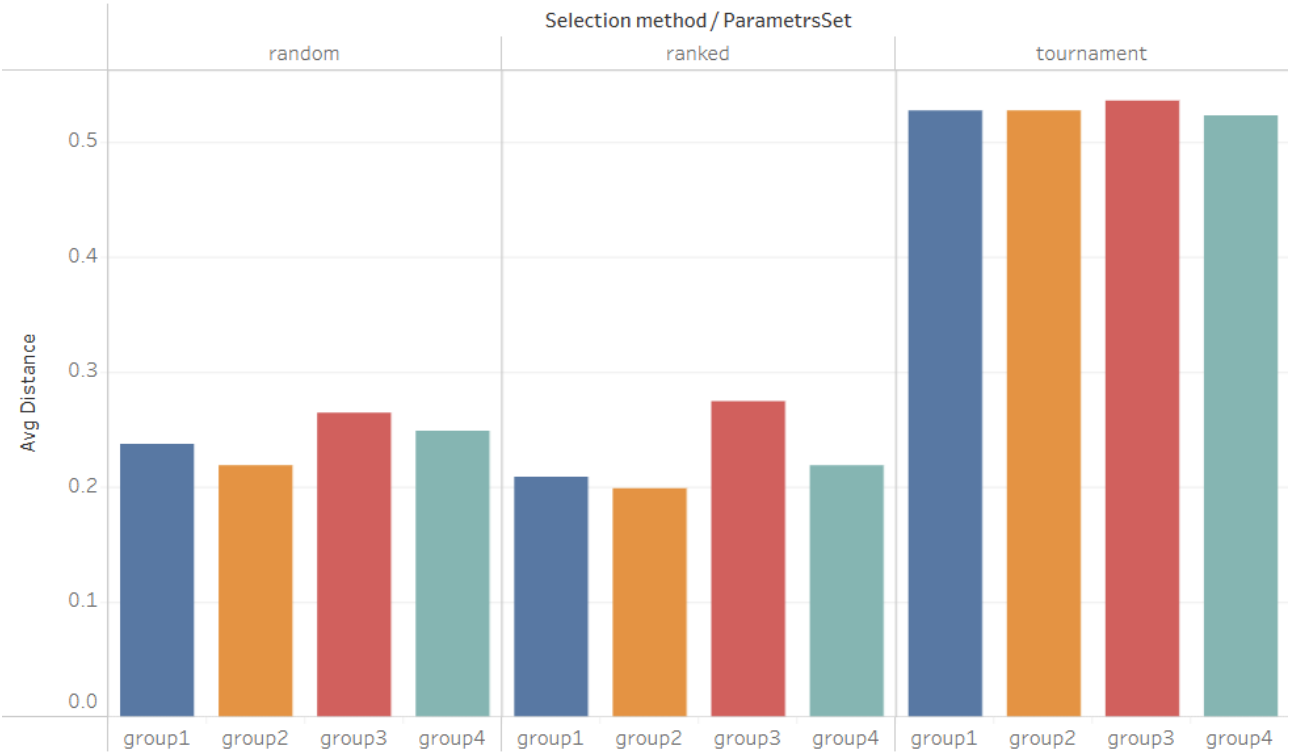

Рисунок №8. Функція Drop-Wave. Візуалізація змін точності.

Аналіз впливу методів відбору та параметрів на точність і стабільність

1. Порівняння методів відбору

Отримані результати показують значні відмінності у продуктивності трьох методів відбору: рангового (ranked), турнірного (tournament) і випадкового (random).

- Для функції schwefel, яка є складною та має багато локальних мінімумів, турнірний відбір показав найгірші результати. Його стабільність у більшості випадків нижча за 15, а середня відстань до оптимального рішення (AvgDistance) перевищує 240, що свідчить про погану збіжність алгоритму.
- Ранговий і випадковий відбір для schwefel показують значно кращі результати. Вони мають вищу стабільність (≈ 100-125) та значно менші значення AvgDistance (≈ 9-15), що означає кращу точність.
- Для функції drop_wave, яка є гладкою та має чітко виражений глобальний мінімум, усі методи працюють добре. Проте турнірний відбір все ж має вищі значення AvgDistance (≈ 0,52) порівняно з ранговим та випадковим (~0,2-0,27), що вказує на його меншу ефективність.

2. Вплив параметрів відбору (Selection rate, Cross rate, Mutation rate)

Аналіз впливу параметрів показує:

- Різні значення selection rate (0,10 та 0,20) суттєво не впливають на точність або стабільність для обох функцій.
- Вищий mutation rate (0,20) призводить до погіршення стабільності у випадку турнірного відбору, особливо для schwefel, де стабільність падає до 10-12.
- Вплив cross rate (0,60 та 0,80) мінімальний, але при високому значенні (0,80) спостерігається більша нестабільність для schwefel при турнірному відборі.

Гіпотеза. Чому турнірний відбір працює гірше?

Основна проблема турнірного відбору полягає у його механізмі: локальна конкуренція призводить до вибору лише найкращих особин з невеликої підгрупи, що підвищує ризик ранньої збіжності до локальних мінімумів.

- Функція schwefel містить багато локальних мінімумів, тому турнірний відбір часто потрапляє в них, що пояснює високі значення AvgDistance і низьку стабільність.

- Функція drop_wave має чіткий глобальний мінімум, тому турнірний відбір працює краще, хоча все ще поступається ранговому та випадковому методам.

Висновки:

- **Ранговий і випадковий відбір кращі за турнірний**, особливо для таких складних функцій, як в нашому випадку.
- **Турнірний відбір схильний до ранньої збіжності** та погано працює для складних функцій з багатьма локальними мінімумами.
- **Зміна параметрів (selection rate, cross rate, mutation rate) має мінімальний вплив**, але високий mutation rate (0,20) може знижувати стабільність.
- Для складних оптимізаційних задач рекомендується використовувати **ранговий або випадковий метод відбору**, оскільки вони забезпечують кращу генетичну різноманітність та точність результатів.

Додаток А

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

import random

import copy


# Функція Швефеля

def schwefel(x, y):

    return 837.9658 - (x * np.sin(np.sqrt(abs(x))) + y * np.sin(np.sqrt(abs(y))))


# Функція Drop-Wave

def drop_wave(x, y):

    numerator = 1 + np.cos(12 * np.sqrt(x**2 + y**2))

    denominator = 0.5 * (x**2 + y**2) + 2

    return -numerator / denominator
```

```
min_schwefel=(420.9687,420.9687,0)
min_drop_wave=(0,0,-1)
gen=40
bounds_schwefel=[-500,500]
bounds_drop_wave=[-5.12,5.12]
temp = [0,0,0]
d=1
n_f=0

# Функція для відображення графіків
def plot_function(func, bounds, title):
    x = np.linspace(bounds[0], bounds[1], 400)
    y = np.linspace(bounds[0], bounds[1], 400)
    X, Y = np.meshgrid(x, y)
    Z = func(X, Y)

    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(X, Y, Z, cmap='viridis')

    ax.set_title(title)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    plt.show()
```

```
# Початкова популяція
def initialize_population(pop_size, bounds):
    population=[]
    for i in range(pop_size):
            temp[0] = random.uniform(bounds[0], bounds[1])
            temp[1] = random.uniform(bounds[0], bounds[1])
            population.append(copy.deepcopy(temp))
    return population

# Фітнес-функція
def fitness(individual, func):
    global n_f
    n_f += 1
    if func == 'schwefel':
        return schwefel(individual[0], individual[1])
    elif func == 'drop_wave':
        return drop_wave(individual[0], individual[1])

# Ранговий відбір
def ranked_selection(pop_size, count=1):
    def select_one():
        Roulette_size = int(pop_size * (pop_size + 1) / 2)
        ball = random.randint(1, Roulette_size)
        choice = pop_size
        check = pop_size
        while check < ball:
            choice -= 1
            check += choice
```

```
        return pop_size - choice

    if count == 1:
        return select_one()
    else:
        a = select_one()
        b = select_one()
        while b == a:
            b = select_one()
        return [a, b]

# Турнірний відбір
def tournament_selection(population, tournament_size=3, count=1):
    population = list(population)
    def select_one():
        selected = random.sample(population, tournament_size)
        return np.argmin([ind[2] for ind in selected])

    if count == 1:
        return select_one()
    else:
        a = select_one()
        b = select_one()
        while b == a:
            b = select_one()
        return [a, b]

#  Рандомний відбір
```

```
def random_selection(population, count=1):
    if count == 1:
        return random.randint(0, len(population) - 1)
    else:
        return random.sample(range(len(population)), 2)


def crossover(p1,p2,func,bounds):
    t=[0,0,0]
    betta_1=random.uniform(-1*d,1+d)
    betta_2=random.uniform(-1*d,1+d)
    t[0]=betta_1*p1[0]+(1-betta_1*p2[0])
    t[1]=betta_2*p1[1]+(1-betta_2*p2[1])
    if np.abs(t[0])>np.abs(bounds[0]):
        t[0]=random.uniform(bounds[0],bounds[1])
    if np.abs(t[1])>np.abs(bounds[0]):
        t[1]=random.uniform(bounds[0],bounds[1])
    return t


def mutation(p, bounds, func):
    t=[0,0,0]
    mutating_element=np.random.normal(loc=0, scale=(bounds[1]-bounds[0])/6,
size=2)
    t[0]=p[0]+mutating_element[0]
    t[1]=p[1]+mutating_element[1]
    if np.abs(t[0])>np.abs(bounds[0]):
        t[0]=random.uniform(bounds[0],bounds[1])
    if np.abs(t[1])>np.abs(bounds[0]):
        t[1]=random.uniform(bounds[0],bounds[1])
    return p
```

```
def euclidean_distance(p1, p2):
    return np.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)
def evaluate_minimum(best, true_minimum, tolerance=0.3):
    distance = euclidean_distance(best, true_minimum)
    found_minimum = int(distance <= tolerance)
    return distance, found_minimum


# Основна функція генетичного алгоритму
def genetic_algorithm(func, bounds, pop_size=50, generations=100, cross_rate=0.7,
mutation_rate=0.1, selection_method="ranked"):
    global n_f
    n_f = 0
    population = np.array(initialize_population(pop_size, bounds))

    for ind in population:
        ind[2] = fitness(ind, func)

    population = population[population[:,2].argsort()]
    n_cross=int(np.floor(pop_size*cross_rate))
    n_mutate= int(pop_size*mutation_rate)
    n_selected = pop_size-n_mutate-n_cross

    for _ in range(generations):
        new_population = []

        # selected
```

```
    for k in range(n_selected):
        new_population.append(population[k])
    k=0
    #cross
    for k in range(n_cross):
        if selection_method == 'ranked':
            points = ranked_selection(pop_size,2)
        elif selection_method == 'tournament':
            points = tournament_selection(population, 3, 2)
        elif selection_method == 'random':
            points = random_selection(population,2)

        new_population.append(crossover(population[points[0]],
population[points[1]], func, bounds))
    k=0
    for k in range(n_mutate):
        if selection_method == 'ranked':
            point = ranked_selection(pop_size)
        elif selection_method == 'tournament':
            point = tournament_selection(population)
        elif selection_method == 'random':
            point = random_selection(population)

        new_population.append(mutation(population[point], bounds, func))

    population=np.array(copy.deepcopy(new_population))

    for ind in population:
        ind[2] = fitness(ind, func)
```

```
        population = population[population[:,2].argsort()]


    #best_solution = population[0]
    #dist  = euclidean_distance(best_solution, optimal_solution)
    return population, n_f
test,serv=genetic_algorithm("drop_wave", bounds_drop_wave, pop_size=200,
cross_rate=0.7, mutation_rate=0.1, selection_method="random")
print(test)
print(serv)
drop_wave_test_x=[]
drop_wave_test_y=[]
drop_wave_test_z=[]
for i in test:
    drop_wave_test_x.append(i[0])
    drop_wave_test_y.append(i[1])
    drop_wave_test_z.append(i[2])
temp_exp={
    "X": drop_wave_test_x,
    "Y": drop_wave_test_y,
    "Z": drop_wave_test_z
}
export_test=pd.DataFrame(temp_exp)
export_test.to_excel("output1.xlsx", index=False)
x = np.linspace(-6, 6, 500)  # Діапазон X
y = np.linspace(-6, 6, 500)  # Діапазон Y
X, Y = np.meshgrid(x, y)     # Створюємо координатну сітку
Z = drop_wave(X, Y)              # Обчислюємо значення функції


# Будуємо графік
```

```
plt.figure(figsize=(8, 6))
contour = plt.contourf(X, Y, Z, levels=30, cmap='terrain')  # 'terrain' - кольори, схожі на фізичну карту
plt.colorbar(contour, label="Значення drop_wave")  # Додаємо кольорову шкалу
plt.xlabel("X")
plt.ylabel("Y")
for i in test:
    plt.scatter(i[0],i[1], color="black", marker=".", s=50)
plt.scatter(0, 0, color="Red", marker="x", s=50)
# Відображення функцій


plot_function(drop_wave, bounds_drop_wave, "Функція Drop-Wave")
test,serv=genetic_algorithm("schwefel", bounds_schwefel, pop_size=200, cross_rate=0.7, mutation_rate=0.1, selection_method="ranked")
print(test)
print(serv)
schwefel_test_x=[]
schwefel_test_y=[]
schwefel_test_z=[]
for i in test:
    schwefel_test_x.append(i[0])
    schwefel_test_y.append(i[1])
    schwefel_test_z.append(i[2])
temp_exp={
    "X": schwefel_test_x,
    "Y": schwefel_test_y,
    "Z": schwefel_test_z
}
export_test=pd.DataFrame(temp_exp)
```

```
export_test.to_excel("output.xlsx", index=False)
x = np.linspace(-500, 500, 1000)  # Діапазон X
y = np.linspace(-500, 500, 1000)  # Діапазон Y
X, Y = np.meshgrid(x, y)     # Створюємо координатну сітку
Z = schwefel(X, Y)              # Обчислюємо значення функції

# Будуємо графік
plt.figure(figsize=(8, 6))
contour = plt.contourf(X, Y, Z, levels=30, cmap='terrain')  # 'terrain' - кольори, схожі на фізичну карту
plt.colorbar(contour, label="Значення schwefel")  # Додаємо кольорову шкалу
plt.xlabel("X")
plt.ylabel("Y")
for i in test:
    plt.scatter(i[0],i[1], color="black", marker=".", s=50)
plt.scatter(420.9687, 420.9687, color="Red", marker="x", s=50)
plot_function(schwefel, bounds_schwefel, "Функція Швефеля")
functions=["schwefel", "drop_wave"]
selection_methods = ["ranked", "tournament", "random"]
cross_mutation_rates=[[0.7, 0.1],[0.6,0.2],[0.8,0.1],[0.7, 0.2]]
def grid_test():
    temp_counter=[]
    temp_pop=[]
    temp_funcs=[]
    for func in functions:
        for rate in cross_mutation_rates:
            for method in selection_methods:
                if func=="schwefel":
                    bounds=bounds_schwefel
```

```
            else:
                bounds=bounds_drop_wave
            x, y=genetic_algorithm(func, bounds, pop_size=200, generations=50,
cross_rate=rate[0], mutation_rate=rate[1], selection_method=method)
            temp_pop.append(x)
            temp_counter.append(y)
            temp_funcs.append(func)
    return temp_pop, temp_counter, temp_funcs
service_stability=[]
service_accuracy=[]
service_functions=[]
service_counter=[]
service_cross_rates=[]
service_mut_rates=[]
service_sel_rates=[]
service_method=[]

for func in functions:
        for rate in cross_mutation_rates:
            for method in selection_methods:
                service_functions.append(func)
                service_sel_rates.append(1-rate[0]-rate[1])
                service_cross_rates.append(rate[0])
                service_mut_rates.append(rate[1])
                service_method.append(method)
                service_stability.append(np.float64(0))
                service_accuracy.append(np.float64(0))
                service_counter.append(np.float64(0))
dict={
```

```
    "Target function": service_functions,
    "Selection rate": service_sel_rates,
    "Cross rate": service_cross_rates,
    "Mutation rate": service_mut_rates,
    "Selection method":service_method,
    "Stability": service_stability,
    "AvgDistance":service_accuracy,
    "Function counts": service_counter
    }
results_df=pd.DataFrame(dict)
print(results_df)
dist=0
for i in range(200):
    pop,count,funcs =grid_test()
    print(i)
    for j in range(len(pop)):
        if funcs[j]=="schwefel":
            dist=euclidean_distance(pop[j][0],min_schwefel)
            #print(dist)
            if dist<=10:
                results_df.at[j,'Stability']+=1

            results_df.at[j,'AvgDistance']=results_df.at[j,'AvgDistance']+dist
            results_df.at[j,'Function counts']=count[j]
        if funcs[j]=="drop_wave":
            dist=euclidean_distance(pop[j][0],min_drop_wave)
            if dist<=0.6:
                results_df.at[j,'Stability']+=1
```

```
            results_df.at[j,'AvgDistance']=results_df.at[j,'AvgDistance']+dist
            results_df.at[j,'Function counts']=count[j]
for i in range(24):
    results_df.at[i,"AvgDistance"]=results_df.at[i,"AvgDistance"]/200
 results_df.to_excel("rezal.xlsx", index=False)
```