Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ ОПТИМІЗАЦІЇ МУЛЬТИМОДАЛЬНИХ НЕПЕРЕРВНИХ ФУНКЦІЙ
RESEARCH AND SOFTWARE IMPLEMENTATION OF GENETIC ALGORITHMS FOR OPTIMIZATION OF MULTIMODAL CONTINUOUS FUNCTIONS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-214м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

ФРЕНКЕЛЬ Є.О.
(ПРІЗВИЩЕ та ініціали)
Керівник ІЛЬЯШЕНКО М.Б.
(ПРІЗВИЩЕ та ініціали)
Рецензент КОЦУР М.І.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
"____" _____2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ФРЕНКЕЛЯ Євгена Олександровича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація генетичних алгоритмів оптимізації мультимодальних неперервних функцій. Research and Software Implementation of Genetic Algorithms for Optimization of Multimodal Continuous Functions

керівник проєкту (роботи) к.т.н., доцент, ІЛЬЯШЕНКО Матвій Борисович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від "30" вересня 2025 року № 447
2. Строк подання студентом проєкту (роботи) 02 грудня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ІЛЬЯШЕНКО М.Б., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання “ 30 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка структури програми.	3 тиждень	Розділ 3
5	Розробка програми.	4-7 тижні	Розділи 3,4
6	Тестування та експериментальне дослідження	8 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	9-10 тижні	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка) _____ Євген ФРЕНКЕЛЬ
(підпис) (Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Матвій ІЛЬЯШЕНКО
(підпис) (Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 100 с., 4 табл., 23 рис., 3 дод., 14 джерел.

PYTHON, SPYDER, ГЕНЕТИЧНИЙ АЛГОРИТМ, МОВА ПРОГРАМУВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ЦІЛЬОВА ФУНКЦІЯ.

Об’єкт дослідження – процеси обчислень, пов'язані з оптимізацією мультимодальних неперервних функцій за допомогою генетичних алгоритмів.

Предмет дослідження – генетичні методи оптимізації мультимодальних неперервних функцій.

Мета роботи – дослідження та реалізація генетичних алгоритмів оптимізації мультимодальних неперервних функцій.

Матеріали, методи та технічні засоби: мова програмування Python, середовище розробки Spyder.

Результати. Розроблено програмне забезпечення для обчислень, пов'язані з оптимізацією мультимодальних неперервних функцій за допомогою генетичних алгоритмів, за допомогою мови програмування Python та середовища розробки Spyder.

Практична цінність роботи полягає у розробці програмного забезпечення для обчислень, пов'язані з оптимізацією мультимодальних неперервних функцій за допомогою генетичних алгоритмів.

Висновки. Виконано проєктування програмного забезпечення для обчислень, пов'язані з оптимізацією мультимодальних неперервних функцій за допомогою генетичних алгоритмів. Розроблено програмне забезпечення для обчислень, пов'язані з оптимізацією мультимодальних неперервних функцій за допомогою генетичних алгоритмів. Здійснено тестування розробленого програмного забезпечення на основі генетичних алгоритмів.

Галузь використання – оптимізація, моделювання та аналіз даних.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 100 pages, 4 tables, 23 figures, 3 appendixes, 14 sources.

PYTHON, SPYDER, GENETIC ALGORITHM, PROGRAMMING LANGUAGE, SOFTWARE, OBJECTIVE FUNCTION.

The object of research is the process of computations related to the optimization of multimodal continuous functions using genetic algorithms.

The subject of the research is genetic methods for optimizing multimodal continuous functions.

The purpose of this work is research and implementation of genetic algorithms for optimizing multimodal continuous functions.

Materials, methods and technical tools: Python programming language, Spyder development environment.

Results. The software tools for calculations related to the optimization of multimodal continuous functions based on genetic algorithms using the Python programming language and the Spyder development environment have been developed.

The practical value of the work lies in the development of software for calculations related to the optimization of multimodal continuous functions using genetic algorithms.

Conclusions. The software for calculations related to the optimization of multimodal continuous functions using genetic algorithms has been designed. The software for calculations related to the optimization of multimodal continuous functions using genetic algorithms has been developed. The developed software based on genetic algorithms has been tested.

Scope of use – optimization, modeling and data analysis.

ЗМІСТ

С.

Перелік скорочень та умовних познак 8

Вступ 9

1 Аналіз предметної області 11

1.1 Оптимізація мультимодальних неперервних функцій на основі генетичних алгоритмів 11

1.2 Аналіз генетичних алгоритмів для оптимізації мультимодальних неперервних функцій 16

1.3 Висновки за розділом 1 22

2 Матеріали і методи 25

2.1 Вибір мови програмування 25

2.2 Вибір середовища розробки для створення програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів 28

2.3 Еволюційний метод контрольованого витіснення 31

2.4 Висновки за розділом 2 36

3 Опис програми 38

3.1 Структура програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів 38

3.2 Функціонування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів 40

3.3 Особливості реалізації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів 43

3.4 Взаємодія з користувачем програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів 51

3.5 Висновки за розділом 3 54

4 Експлуатація, тестування та експериментальне дослідження програми.....56
4.1 Призначення й умови застосування програми 56
4.2 Характеристики програми для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів 57
4.3 Інструкція по експлуатації програми..... 58
4.3.1 Звернення до програми..... 58
4.3.2 Вхідні й вихідні дані..... 59
4.3.3 Повідомлення..... 59
4.4 Виконання програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів 60
4.5 Тестування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів 62
4.6 Висновки за розділом 4 66
Висновки.....67
Перелік джерел посилання 72
Додаток А Технічне завдання.....74
Додаток Б Фрагмент тексту програми 79
Додаток В Слайди презентації 93

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ГА – генетичний алгоритм;

ПЗ – програмне забезпечення;

ЦФ – цільова функція.

ВСТУП

У сучасних наукових та прикладних дослідженнях часто виникають задачі оптимізації, які характеризуються складними, нелінійними, мультимодальними і неперервними функціями мети. Такі задачі є типовими для інженерії, економіки, машинного навчання, біоінформатики, енергетики та багатьох інших галузей. Класичні методи оптимізації, зокрема градієнтні чи детерміновані пошукові алгоритми, у таких випадках часто виявляються неефективними через велику кількість локальних екстремумів, складність знаходження глобального оптимуму та високу обчислювальну складність. Генетичні алгоритми, як частина еволюційних обчислень, демонструють високу ефективність у розв’язанні задач, де традиційні методи не забезпечують належної точності чи стійкості до локальних мінімумів. Їх гнучкість, здатність працювати з чорними скриньками та простота паралелізації роблять ці алгоритми затребуваними для обробки складних функцій оптимізації [1], [2].

Крім того, розробка ефективного програмного забезпечення, що реалізує генетичні алгоритми для оптимізації мультимодальних неперервних функцій, дозволяє автоматизувати та прискорити розв’язання практичних задач, підвищує продуктивність досліджень і сприяє впровадженню передових обчислювальних технологій у різні галузі діяльності [3]-[5].

Отже, дослідження та програмна реалізація генетичних алгоритмів оптимізації є актуальними як з точки зору розвитку методів обчислювального інтелекту, так і з точки зору задоволення зростаючих потреб у вирішенні складних оптимізаційних задач у різних сферах науки і техніки [4]-[6].

Проте програмне забезпечення, що реалізує генетичні алгоритми оптимізації мультимодальних неперервних функцій, має низку недоліків, які варто враховувати при його застосуванні. Насамперед такі алгоритми відзначаються високою обчислювальною складністю, оскільки для досягнення прийнятного результату потрібна велика кількість ітерацій і постійна робота з

чисельними популяціями рішень. Це безпосередньо впливає на тривалість виконання, адже багаторазова оцінка функції пристосованості значно збільшує загальний час пошуку оптимального розв’язку. Попри здатність уникати локальних екстремумів, генетичні алгоритми не дають гарантії знаходження глобального оптимуму у кожному окремому запуску, що може вимагати повторного виконання з іншими параметрами. Якість результатів суттєво залежить від правильного налаштування параметрів, зокрема розміру популяції, ймовірності мутації та кількості поколінь, що іноді потребує додаткових експериментів чи досвіду користувача. Також через стохастичну природу обчислень результати можуть відрізнятися при кожному повторному запуску, що ускладнює відтворення одержаних рішень. Крім того, існуючі програмні рішення не завжди є універсальними або придатними для широкого спектра практичних задач, що часто зумовлює необхідність додаткової модифікації чи доопрацювання програмного коду для конкретних прикладних застосувань [1]-[6].

Тому актуальним є дослідження генетичних алгоритмів оптимізації мультимодальних неперервних функцій. У дипломній кваліфікаційній роботі магістра розв’язується актуальне завдання дослідження та реалізація генетичних алгоритмів оптимізації мультимодальних неперервних функцій.

Для досягнення поставленої мети у кваліфікаційній роботі магістра необхідно розв’язати такі задачі:

– виконати аналіз генетичних алгоритмів для оптимізації мультимодальних неперервних функцій;

– розробити еволюційний метод контрольованого витіснення;

– здійснити проєктування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів;

– створити програмне забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів;

– виконати тестування розробленого програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі ГА.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Оптимізація мультимодальних неперервних функцій на основі генетичних алгоритмів

У технічних галузях часто виникає завдання знаходження найкращого можливого рішення для функцій, які мають багато локальних екстремумів у багатовимірному просторі змінних. Мова йде про мультимодальну оптимізацію, суть якої полягає у визначенні такого набору керованих параметрів, що забезпечує досягнення глобального екстремуму – мінімального чи максимального значення функції у межах заданих обмежень. Під глобальним оптимумом розуміють такий стан системи, коли цільова функція приймає мінімальне чи максимальне значення у всій області визначення і жодна інша припустима конфігурація не дозволяє отримати кращий результат. Водночас локальний екстремум фіксується тоді, коли для певного набору змінних функція набуває екстремального значення лише у межах обмеженого оточення цієї точки, але не гарантується, що таке значення є найкращим серед усіх можливих варіантів [1]-[3].

Застосування класичних чисельних процедур у багатовимірному випадку зазвичай спирається на локальний пошук, де вибір початкового положення значною мірою визначає фінальний результат. Саме тому такі методи доцільно застосовувати переважно для задач з унімодальною структурою цільової функції, де існує лише один екстремум, і не виникає ризику «застрягти» у випадковій локальній вершині або западині [4], [5].

Існує ціла низка підходів, здатних ефективно розв’язувати задачі пошуку глобального екстремуму функцій, що мають багато локальних максимумів чи мінімумів. Такі функції, що містять кілька точок локального екстремуму, називають мультимодальними або полімодальними. Для знаходження екстремумів у випадку унімодальних функцій зазвичай використовують методи, що базуються на розрахунках похідних або на прямих багатовимірних алгоритмах без використання градієнтів, серед яких

можна згадати методи Хука-Дживса, Пауелла або Нелдера-Міда. Якщо ж у функції багато локальних максимумів чи мінімумів, більшість таких локальних методів у більшості випадків приводить до того, що відшукується лише близький локальний екстремум, тоді як глобальний може залишитися не знайденим [1], [3].

Для полімодальних функцій також застосовують багатовимірні методи, що використовують градієнтні величини – такі як алгоритми Коші, Ньютона чи Левенберга-Марквардта. Вони залежать від обчислення похідних різного порядку, що визначає напрямок пошуку. Ідеальною властивістю подібних алгоритмів є уникнення фіксації у локальних екстремумах та досягнення глобального рішення. Але на практиці в багатьох випадках ці методи виявляються чутливими до початкових умов, залежать від попередньої обробки вихідних даних і можуть демонструвати не найкращі часові показники, окрім того, такі підходи часто створюються під вузькі класи задач [2], [3].

Саме через ці обмеження для пошуку глобального екстремуму у складних просторах доцільно впроваджувати еволюційні схеми. Подібні методи повторюють логіку природного добору та механізми генетичних змін і дозволяють відтворювати обчислювальну модель біологічної еволюції для знаходження оптимальних рішень. За своєю суттю еволюційний підхід полягає у роботі з закодованими представленнями параметрів у вигляді хромосом, що дозволяє адаптувати пошук як до неперервних числових областей, так і до комбінованих дискретних множин [4], [5].

У класичному вигляді еволюційний метод описується як пошук мінімуму чи максимуму заданої цільової функції за допомогою оперування множиною варіантів рішень, представлених у хромосомній формі. Ключовими моментами є визначення того, які змінні підлягають оптимізації, спосіб їх представлення у вигляді генетичного коду, формулювання цільової функції, запуск механізму початкового формування популяції, вибір та налаштування

процедур добору, схрещення і мутації та встановлення умов для завершення пошуку [5], [6].

Варто зважати на те, що методи цієї групи мають високу ефективність саме тоді, коли потрібно віднайти глобальне рішення в умовах великої кількості локальних екстремумів. Саме тому у цьому проєкті робиться акцент на перевірці класичних операторів і пошуку оптимальних параметрів їх використання для складних функцій. При цьому не варто забувати про вже існуючі напрацювання та готові алгоритми, що спеціально адаптовані для полімодального пошуку. Якщо оптимізація здійснюється для аналітичних функцій, особливу увагу слід звернути на роботу з числовими величинами та правильну побудову хромосомних структур, що впливає на результативність генетичних маніпуляцій [6], [7].

Процес вибору хромосом у межах генетичного підходу реалізується через селекційний механізм, що визначає, які особини потраплять під дію тих чи інших операторів еволюції, а також які з них формуватимуть основу нової популяції. Якщо стоїть завдання оптимізувати функцію з багатьма локальними екстремумами, варто застосовувати такі види відбору, де кожна особина оцінюється за значенням пристосованості. При пропорційному методі усім представникам надається вага відповідно до їхнього внеску у сумарну придатність, після чого формується новий склад популяції за принципом випадковості, де ймовірність потрапити далі прямо залежить від того, наскільки добре індивід підходить за критерієм цільової функції. Турнірний підхід передбачає, що з усієї множини вибирається обмежена кількість претендентів, серед яких визначається сильніший, який переходить у наступне покоління. При пороговому методі відсіюються ті, чия пристосованість не перевищує заздалегідь встановленого рівня. У практичній частині перевага надавалась саме турнірній моделі, оскільки пороговий відбір вимагає значних обчислювальних витрат через сортування і не забезпечує належного збереження різноманітності навіть при низькому порозі, а пропорційний варіант може втратити ефективність, якщо показники придатності різних

представників зближуються. З огляду на це, турнірна схема відбору залишається одним із найбільш застосовуваних підходів у практиці еволюційних пошуків [4], [5].

Щодо передачі ознак, ця функція закладається в оператори схрещування, що імітують механізми спадковості. У завданнях знаходження глобального екстремуму у функціях із багатьма локальними максимумами чи мінімумами доцільно використовувати кілька варіантів. При одноточковому кросинговері два батьківських індивіди обмінюються інформацією починаючи з випадково вибраної позиції у хромосомі. Рівномірний підхід дозволяє комбінувати значення генів за кожним окремим локусом із наперед визначеною ймовірністю для кожної пари генів. Якщо застосовується арифметичний спосіб, тоді значення нащадків формуються всередині діапазону між значеннями батьків. На відміну від схрещувань, що не змінюють значень генів, арифметичний спосіб формує нові параметри, що дозволяє прискорити сходження, але може знизити варіативність. Там, де важливо уникнути швидкого зближення рішень і залишити можливість для глибшого пошуку, краще комбінувати одноточковий або рівномірний підхід із більш активною мутацією, адже такі варіанти лише перемішують наявні значення без створення нових. Це дозволяє сповільнити передчасну конвергенцію і дослідити більше зон у межах простору рішень. Цікавим моментом є й те, що для двозмінних функцій результат одноточкового і рівномірного схрещення іноді може збігатися, якщо змінюється лише один ген [5], [6].

Ще одним важливим елементом є мутація, що відповідає за внесення новизни в генофонд і підтримку різноманітності у пошуку. Її суть полягає у випадковому зміненні певних позицій у структурі хромосоми. На відміну від відбору чи схрещування, які скоріше структурують наявні дані, мутаційний механізм створює додаткові варіації й дозволяє заглиблювати дослідження у малодосліджені ділянки простору. Для складних багатовершинних функцій доцільно застосовувати різні варіації мутації, зокрема рівномірний варіант,

коли значення певного гена змінюється на випадкове у межах допустимого діапазону, а також методи, що враховують залежність від поточного кроку пошуку, інтервалів значень і випадкових множників. Власне, саме мутація часто виступає єдиним критично необхідним оператором для підтримки ефективного розширення простору рішень, особливо якщо стоїть завдання уникнути зупинки у локальному екстремумі [6], [7].

Визначено, що у завданнях пошуку глобального екстремуму для мультимодальних неперервних функцій доцільно використовувати генетичні алгоритми через те, що такі алгоритми здатні ефективно працювати з просторами рішень, у яких традиційні методи локального пошуку часто втрачають здатність виходити за межі найближчої області локального оптимуму. Природа мультимодальних функцій передбачає наявність багатьох піків або западин, що можуть вводити в оману градієнтні чи прямі методи, оскільки ті схильні рухатись у напрямку найшвидшого зростання або спадання і не завжди спроможні подолати локальні перешкоди. Генетичні алгоритми запозичують принципи природного добору та випадкових мутацій, що дає змогу не обмежуватися лише одним напрямком руху в пошуковому просторі, а охоплювати його ширше завдяки наявності популяції рішень, які конкурують між собою. Завдяки цьому формується механізм, що поєднує локальну експлуатацію вже знайдених рішень із глобальним дослідженням нових областей. Такий баланс між дослідженням і використанням наявної інформації є ключовим фактором успішного пошуку глобального екстремуму. Крім того, генетичні алгоритми легко адаптуються до функцій, для яких складно або неможливо обчислити похідні, що розширює спектр їхнього використання для різних типів безперервних завдань. Застосування мутації, різних типів селекції та схрещування створює умови для уникнення передчасного збігу рішень до локального максимуму чи мінімуму. Це робить генетичний підхід універсальним і придатним для ситуацій, де інші методи обчислень дають слабкі результати через багатовершинність або складний рельєф цільової функції.

1.2 Аналіз генетичних алгоритмів для оптимізації мультимодальних неперервних функцій

У рамках застосування генетичних алгоритмів головним завданням постає дослідження можливих комбінацій генетичних кодів для пошуку такого індивіда, чия властивість у вигляді фенотипу відповідатиме найкращому значенню за заданим критерієм пристосованості. Такі еволюційні підходи базуються на подібності до природних механізмів добору та мутацій. Зв’язок між біологічною моделлю та завданням пошуку екстремумів функції зі складним рельєфом добре ілюструється поняттям адаптивних поверхонь, де популяція природним чином досягає найближчого підвищення у ландшафті, хоча далі можуть існувати ще кращі піки, до яких прямий шлях не завжди можливий. Для подолання таких бар’єрів доцільно використовувати широкий та масштабний пошук шляхом постійних змін та варіацій, що відкривають доступ до незвіданих ділянок простору [6]-[8].

Відповідно, генетичні алгоритми повинні втілювати такі механізми у своїй структурі, запобігаючи ризику завчасної стабілізації рішення на невдалому локальному максимумі. Існуючі підходи до організації такої динаміки пошуку часто групують за способом протидії швидкому зближенню популяції. Одні методи передбачають розбиття популяції на незалежні підмножини, що майже не перетинаються між собою, інші вводять спеціальні елементи для створення нових ділянок пошуку і активного вивчення малодосліджених фрагментів простору рішень. Окремо виділяються алгоритми, що дозволяють враховувати кілька цільових умов одночасно. Подібну класифікацію можна побачити і у схемі, де візуалізовано основні види методів для багатовершинної оптимізації, серед яких окремо акцентовано увагу на підходах ущільнення, що стали основою реалізації у практичній частині цієї роботи. Існують й інші способи систематизації подібних методів, що, хоча і формулюються під різними назвами, по суті об’єднують схожі рішення з точки зору їхньої функції. Приміром, частина підходів орієнтується

на уникнення швидкої втрати різноманітності у межах популяції і тим самим не дозволяє рішенню зафіксуватись у випадковому локальному максимумі. Інший напрямок методів фокусується на тому, як відновлювати різноманітність, якщо вона втрачена, пропонуючи варіанти повторного запуску, зміни підходу чи циклічного поновлення ключових параметрів для повернення до ширшого пошуку [8]-[10].

На рис. 1.1 наведено узагальнену схему класифікації генетичних алгоритмів для оптимізації мультимодальних неперервних функцій.

Для завдань пошуку оптимумів у складних мультимодальних функціях, де простір рішень має багато локальних вершин, застосування генетичних алгоритмів з чітко реалізованими механізмами підтримки та регулювання різноманітності в популяції є виправданим і ефективним підходом. Така аргументація базується насамперед на тому, що класичні еволюційні стратегії без контролю за генетичним різноманіттям швидко схильні до ранньої конвергенції: пошук концентрується лише навколо одного локального екстремуму, часто не досягаючи глобального рішення. Саме тому багатовершинний характер цільової функції вимагає такої структури алгоритму, що дозволяє одночасно досліджувати декілька різних областей простору, не даючи домінувати одній підгрупі особин, яка випадково виявилася ближчою до проміжної вершини [6]-[10].

Керування різноманітністю у популяції забезпечує підтримку балансу між експлуатацією вже знайдених перспективних рішень і дослідженням нових напрямків пошуку. Коли у межах ГА впроваджуються спеціалізовані оператори ущільнення, методи розподілу на острови чи підгрупи, а також селекційні чи мутаційні стратегії, що штучно стимулюють появу нових комбінацій генів, популяція зберігає здатність оновлювати свій генотипічний склад навіть у пізніх поколіннях. Це дозволяє уникнути застою в одній зоні пошуку й поступово наближатися до справжнього глобального екстремуму [5]-[9].

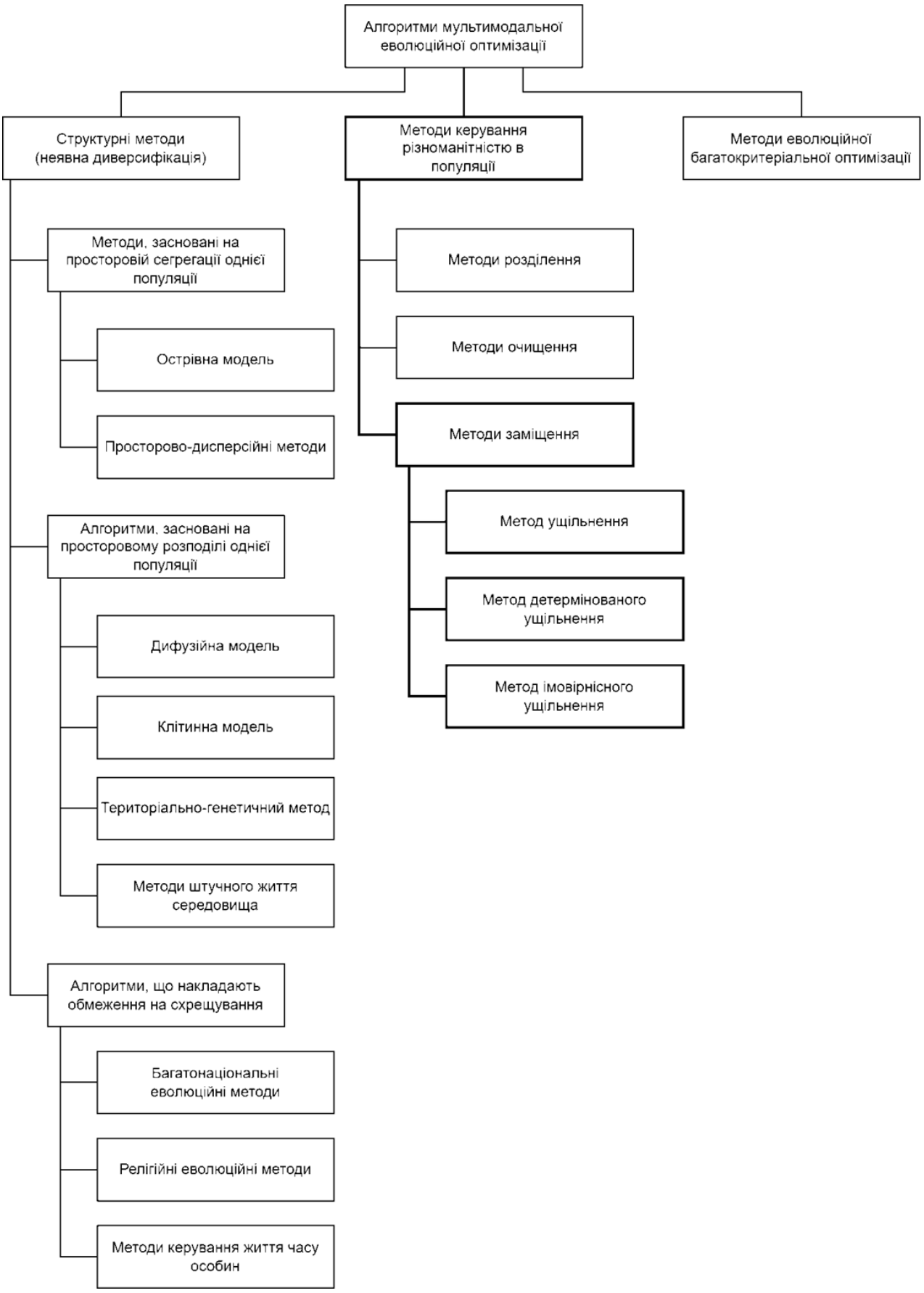


Рисунок 1.1 – Класифікація генетичних алгоритмів для оптимізації мультимодальних неперервних функцій [6]-[9]

Особливо актуально таке рішення у ситуаціях, коли оцінка функції має багато розривів у топографії оптимальних зон і просте збільшення розміру популяції чи кількості поколінь не дає гарантії успіху. Керування різноманітністю допомагає сформувати внутрішню еволюційну конкуренцію між різними «нішами», що конкурують за ресурси та право передавати власні гени у наступні генерації. У підсумку застосування генетичних алгоритмів із вбудованими засобами підтримки різноманітності є одним із небагатьох способів ефективно проходити через складні адаптивні ландшафти, уникаючи замикання на локальних піках і збільшуючи шанси на знаходження справжнього глобального екстремуму [8]-[10].

Коли мова йде про пошук екстремумів у мультимодальних неперервних функціях, найбільшу складність становить велика кількість локальних максимумів чи мінімумів, серед яких потрібно знайти глобальний. Генетичні алгоритми у своїй класичній формі здатні досліджувати такі функції завдяки механізму природного відбору та випадкових комбінацій, однак без спеціального контролю за різноманітністю популяція досить швидко може зійтись до одного локального рішення. Саме тому застосування стратегій заміщення, що регулюють ступінь подібності між особинами та оновлюють популяцію з урахуванням просторової відстані між рішеннями, робить такі алгоритми більш стійкими до ефекту ранньої конвергенції [8]-[10].

Одним з дієвих підходів у цьому напрямку є метод ущільнення. Його суть полягає у тому, що при формуванні нового покоління кожен нащадок порівнюється з батьками з точки зору генетичної або просторової відстані. Якщо нащадок подібний до певного батька, то порівнюється рівень їхньої пристосованості. Якщо потомок демонструє кращі характеристики, він заміщує батьківську особину. Таким чином простір рішень поступово ущільнюється: групи близьких особин відсіюються, що перешкоджає домінуванню лише одного генотипу і стимулює розвиток кількох незалежних ніш [8]-[10].

Метод детермінованого ущільнення діє за схожим принципом, але механізм заміщення строго визначений. Для кожного потомка знаходиться найближчий батько, після чого відбір проводиться за чітким правилом: з пари «батько-нащадок» залишається той, хто має вищу пристосованість. Такий варіант знижує випадковість і підсилює контроль над структурою популяції, особливо ефективно працюючи там, де просторовий поділ рішень має вирішальне значення [8]-[10].

Імовірнісне ущільнення розширює підхід детермінованого тим, що дає можливість заміщення не за жорстким правилом, а із заданою ймовірністю. Це означає, що навіть слабший потомок має шанс залишитись у популяції, якщо його унікальність або положення у генотипічному просторі є перспективними для подальшого пошуку. Такий метод збільшує варіативність і дозволяє уникнути надмірної «консервації» одних рішень, залишаючи простір для більш гнучкого дослідження [8]-[10].

Отже, заміщення у різних формах — від класичного ущільнення до його детермінованих чи імовірнісних варіантів — створює в середині ГА внутрішній механізм балансування між експлуатацією уже знайдених сильних рішень і постійним введенням нового генетичного матеріалу. Для мультимодальних неперервних функцій саме таке поєднання стабільності та різноманітності стає ключовим фактором, який значно підвищує шанси алгоритму знайти глобальний екстремум, не втрачаючи можливості досліджувати весь складний простір рішень [8]-[10].

За результатами проведеного аналізу можна зробити висновок, що у наш час існує досить генетичних алгоритмів для оптимізації мультимодальних неперервних функцій. Проте незважаючи на очевидну ефективність використання генетичних підходів, які цілеспрямовано підтримують різноманітність серед особин для вирішення задач мультимодальної неперервної оптимізації, не можна не враховувати певних обмежень, притаманних класичним схемам заміщення чи ущільнення. По-перше, методи класичного ущільнення, хоч і добре стримують передчасну конвергенцію,

часто мають надмірну жорсткість при виборі пар «батько-дитина», що може призводити до блокування переходу популяції до більш перспективних областей простору рішень, якщо початкове розміщення особин виявилося невдалим. Детерміноване ущільнення, у свою чергу, підсилює цей ефект через однозначні правила заміщення, які можуть ігнорувати локальні можливості для стрибка через долини у багатовершинному ландшафті.

Імовірнісні підходи частково згладжують жорсткість, однак це часто відбувається ціною збільшення флуктуацій пристосованості, що може гальмувати стабільну збіжність або потребує додаткового точного налаштування параметрів імовірнісного відбору. Крім того, традиційні схеми не завжди дозволяють ефективно балансувати інтенсивність витіснення слабких особин і збереження вузьких ніш із перспективним, але ще не найкращим генетичним матеріалом.

Порівняльну характеристику генетичних алгоритмів з керуванням різноманітністю в популяції для оптимізації мультимодальних неперервних функцій наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння генетичних алгоритмів з керуванням різноманітністю в популяції для оптимізації мультимодальних неперервних функцій

Критерій порівняння	Метод ущільнення	Детерміноване ущільнення	Імовірнісне ущільнення
Жорсткість механізму відбору	+–	+	–
Ймовірність збереження слабких рішень	–	–	+
Чутливість до параметрів	+–	+–	+
Простота реалізації	+	+–	+–
Гнучкість налаштування	+–	–	+
Швидкість збіжності	+	+	+–

З огляду на такі суперечності, виникає потреба у вдосконаленні механізму, який одночасно дозволяв би уникати передчасної збіжності до одного максимуму чи мінімуму, зберігав би регіональну різноманітність та не зупиняв би розвиток менш домінуючих, але потенційно цінних напрямів пошуку. Саме тому доречно розробити модифіковану концепцію контрольованого витіснення, яка би поєднувала переваги ущільнення та елементів гнучкого імовірнісного впливу на механізми заміщення, і реалізувати цю ідею у спеціалізованому програмному рішенні. Такий підхід дозволить об’єктивно підвищити якість пошуку глобального екстремуму у складних мультимодальних ландшафтах і водночас знизити залежність результатів від вузько визначених параметрів і налаштувань, роблячи процес оптимізації більш стійким і керованим.

При розробці програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів необхідно забезпечити такі функціональні вимоги:

– підтримка можливості оптимізації мультимодальних неперервних функцій;

– підтримка можливості використання різних генетичних алгоритмів для оптимізації функцій;

– можливість завдання функції для оптимізації;

– можливість графічного відображення отриманих результатів;

– підтримка можливості задавання параметрів генетичних алгоритмів;

– підтримка можливості подання результатів у табличному вигляді.

1.3 Висновки за розділом 1

Визначено, що у завданнях пошуку глобального екстремуму для мультимодальних неперервних функцій доцільно використовувати генетичні алгоритми через те, що такі алгоритми здатні ефективно працювати з просторами рішень, у яких традиційні методи локального пошуку часто

втрачають здатність виходити за межі найближчої області локального оптимуму. Природа мультимодальних функцій передбачає наявність багатьох піків або западин, що можуть вводити в оману градієнтні чи прямі методи, оскільки ті схильні рухатись у напрямку найшвидшого зростання або спадання і не завжди спроможні подолати локальні перешкоди. Генетичні алгоритми запозичують принципи природного добору та випадкових мутацій, що дає змогу не обмежуватися лише одним напрямком руху в пошуковому просторі, а охоплювати його ширше завдяки наявності популяції рішень, які конкурують між собою. Завдяки цьому формується механізм, що поєднує локальну експлуатацію вже знайдених рішень із глобальним дослідженням нових областей. Такий баланс між дослідженням і використанням наявної інформації є ключовим фактором успішного пошуку глобального екстремуму. Крім того, генетичні алгоритми легко адаптуються до функцій, для яких складно або неможливо обчислити похідні, що розширює спектр їхнього використання для різних типів безперервних завдань. Застосування мутації, різних типів селекції та схрещування створює умови для уникнення передчасного збігу рішень до локального максимуму чи мінімуму. Це робить генетичний підхід універсальним і придатним для ситуацій, де інші методи обчислень дають слабкі результати через багатовершинність або складний рельєф цільової функції.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить генетичних алгоритмів для оптимізації мультимодальних неперервних функцій. Проте незважаючи на очевидну ефективність використання генетичних підходів, які цілеспрямовано підтримують різноманітність серед особин для вирішення задач мультимодальної неперервної оптимізації, не можна не враховувати певних обмежень, притаманних класичним схемам заміщення чи ущільнення. По-перше, методи класичного ущільнення, хоч і добре стримують передчасну конвергенцію, часто мають надмірну жорсткість при виборі пар «батько-дитина», що може призводити до блокування переходу популяції до більш перспективних

областей простору рішень, якщо початкове розміщення особин виявилося невдалим. Детерміноване ущільнення, у свою чергу, підсилює цей ефект через однозначні правила заміщення, які можуть ігнорувати локальні можливості для стрибка через долини у багатовершинному ландшафті. Імовірнісні підходи частково згладжують жорсткість, однак це часто відбувається ціною збільшення флуктуацій пристосованості, що може гальмувати стабільну збіжність або потребує додаткового точного налаштування параметрів імовірнісного відбору. Крім того, традиційні схеми не завжди дозволяють ефективно балансувати інтенсивність витіснення слабких особин і збереження вузьких ніш із перспективним, але ще не найкращим генетичним матеріалом.

З огляду на такі суперечності, виникає потреба у вдосконаленні механізму, який одночасно дозволяв би уникати передчасної збіжності до одного максимуму чи мінімуму, зберігав би регіональну різноманітність та не зупиняв би розвиток менш домінуючих, але потенційно цінних напрямів пошуку. Саме тому доречно розробити модифікований еволюційний метод контрольованого витіснення та створити відповідне програмне забезпечення, яке буде поєднувати переваги ущільнення та елементів гнучкого імовірнісного впливу на механізми заміщення, і реалізувати цю ідею у спеціалізованому програмному рішенні. Такий підхід дозволить об’єктивно підвищити якість пошуку глобального екстремуму у складних мультимодальних ландшафтах і водночас знизити залежність результатів від вузько визначених параметрів і налаштувань, роблячи процес оптимізації більш стійким і керованим.

Тому актуальною є розробка еволюційного методу контрольованого витіснення та відповідного програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

Сформульовано функціональні вимоги до програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

При розробці програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів було використано мову програмування Python [11], [12].

Мова Python посідає провідні позиції серед інструментів, які застосовуються у сферах штучного інтелекту, обчислювальної оптимізації та досліджень у галузі машинного навчання. Її активне використання у прикладних і теоретичних розробках сприяє кращій взаємодії отриманих результатів з різноманітними науковими чи промисловими рішеннями й полегшує обмін напрацюваннями всередині професійної спільноти [11], [12].

Завдяки багатому вибору модулів, додаткових пакетів і спеціалізованих фреймворків Python відкриває широкі можливості для повторного використання коду та гнучкої побудови нових прототипів. Це створює зручні умови для комбінування різних методів, включно з інтеграцією генетичних алгоритмів у поєднанні з іншими підходами, наприклад, у зв’язці з нейронними мережами чи різними видами еволюційних стратегій [11], [12].

Інтерактивність середовищ, які підтримують Python, серед яких варто згадати Spyder, Jupyter чи Colab, дозволяє об’єднувати скрипти, графіки, таблиці та коментарі в одному місці. Такий формат підготовки матеріалів значно спрощує ведення спостережень за перебігом дослідів і оформлення проміжних чи фінальних звітів [11], [12].

Програми, написані мовою Python, функціонують без складнощів на різних операційних системах, не потребуючи істотних правок коду. При цьому для завдань, де потрібна підвищена швидкодія, можна поєднувати Python із низькорівневими мовами, як-от C чи C++, а також із CUDA для використання обчислювальних ресурсів графічних процесорів [11], [12].

Широка підтримка міжнародної спільноти розробників і постійний розвиток бібліотек забезпечують стабільний доступ до актуальних рішень і

оновлень. Така відкритість і динаміка розвитку роблять Python міцною основою для створення і підтримки програмних інструментів, призначених для наукових експериментів і прикладних досліджень [11], [12].

Застосування Python як основного інструменту для створення програмних рішень, що реалізують генетичні алгоритми для обробки мультимодальних неперервних функцій, виправдовується низкою практичних переваг. Чіткість і лаконічність синтаксису цієї мови сприяють легкому опануванню та прискорюють етапи написання, перевірки й підтримки коду, що особливо корисно для тих, хто лише розпочинає знайомство з методами обчислювального інтелекту [11], [12].

Завдяки широкому спектру доступних модулів і відкритих бібліотек мова програмування Python дозволяє швидко організовувати обчислення, проводити складні експерименти й адаптувати алгоритми без витрат часу на розробку базових складових з нуля. Підтримка інструментів для побудови графіків і візуального аналізу результатів дає змогу глибше оцінювати поведінку обчислювальних процедур та ефективніше контролювати коректність отриманих даних [11], [12].

Велика кількість прикладів і активна взаємодія користувачів Python створюють сприятливі умови для обміну практичними рішеннями, що зменшує час на пошук відповідей і оптимізує розв’язання стандартних технічних питань. Додатковою перевагою є здатність Python працювати на різних операційних системах, що спрощує розгортання програмних комплексів у різноманітних середовищах [11], [12].

Можливість організації паралельних процесів або масштабування обчислень відкриває перспективи застосування Python для задач великого обсягу, де потрібна висока продуктивність. Сукупність цих властивостей робить вибір Python обґрунтованим і доцільним у контексті дослідження, впровадження та розвитку програмних засобів для оптимізації функцій складної структури [11], [12].

Обґрунтування вибору мови програмування для реалізації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів наведено у таблиці 2.1.

Таблиця 2.1 – Обґрунтування вибору мови програмування для реалізації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

Критерій порівняння мов програмування	Мова програмування		
	Python	C#	Java
Простота синтаксису та легкість навчання	+	+–	–
Наявність готових бібліотек для генетичних алгоритмів	+	+–	+–
Зручність візуалізації результатів	+	+–	–
Зручність для створення ПЗ для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів	+	–	–
Кросплатформність	+	+–	+

Отже, для реалізації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів обрано мову програмування Python, яка має простий та зрозумілий синтаксис, що значно спрощує розробку, тестування й супровід алгоритмів, а також знижує поріг входження для дослідників і студентів. Крім того, Python підтримує велику кількість потужних бібліотек для наукових обчислень, аналізу даних і оптимізації, що дає змогу швидко будувати і експериментувати з різними варіантами генетичних алгоритмів без необхідності реалізовувати всі компоненти з нуля.

2.2 Вибір середовища розробки для створення програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

В якості середовища розробки для створення програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів було обрано середовище Spyder [13], [14].

Середовище розробки Spyder спрямовано на потреби розробників, які займаються обробкою складних розрахунків і опрацюванням великих обсягів даних у межах аналітичної чи дослідницької діяльності. Це середовище поєднує у собі зручність редактора коду зі спеціалізованими інструментами, які дають змогу працювати поетапно: писати фрагменти програм, запускати окремі блоки, стежити за змінами у даних під час виконання й фіксувати спостереження, що виникають у процесі обчислень [13], [14].

Наявність функції, що дозволяє переглядати властивості змінних та інших об’єктів, відкриває широкі можливості для контролю стану алгоритмів під час розробки, зокрема коли йдеться про створення чи тестування механізмів еволюції в межах генетичних моделей. Відстеження виконаних операцій і збереження послідовності кроків у середовищі розробки Spyder забезпечують зручність у відновленні ходу роботи або поверненні до попередніх фаз без втрати деталей [13], [14].

Spyder без труднощів працює разом із інструментами для контролю версій, що особливо корисно, коли дослідження виконуються спільно або мають кілька послідовних редакцій. Завдяки можливості підключати додаткові розширення й додатки функціонал можна налаштовувати відповідно до специфіки завдання, поступово розширюючи його межі [13], [14].

Використання платформи Anaconda спрощує організацію та керування необхідними бібліотеками, що гарантує стабільність у налаштуванні середовища для реалізації обчислювальних методів. Сукупність цих рис перетворює Spyder на багатофункціональний інструмент, придатний для

серйозних досліджень, де потрібна реалізація складних алгоритмічних рішень, таких як генетичні методи оптимізації [13], [14].

Використання Spyder як платформи для написання програм, що реалізують генетичні алгоритми для розв’язання задач оптимізації неперервних функцій із багатьма екстремумами, виглядає обґрунтованим через низку властивостей, що роблять це середовище надзвичайно зручним для прикладних експериментів і дослідницьких проєктів. Структура інтерфейсу, що за стилем нагадує знайомі багатьом середовища типу MATLAB, поєднана з відкритістю екосистеми Python, що дозволяє працювати з обчисленнями гнучко та зручно [13], [14].

Завдяки функціоналу, який включає підсвічування коду та автоматичні підказки, організація алгоритмів та їх відлагодження виконуються швидше, а спеціальна консоль допомагає запускати частини скриптів окремо, що особливо цінно для перевірки точності роботи окремих операцій у межах генетичного підходу або в момент оцінювання поведінки популяцій під час еволюції [13], [14].

Можливість переглядати всі ключові дані й параметри безпосередньо під час виконання розрахунків полегшує відстеження проміжних результатів і дозволяє своєчасно реагувати на помилки чи аномалії [13], [14].

Наявність модулів для швидкого створення графіків дозволяє оцінювати динаміку роботи алгоритму буквально під час виконання, не покидаючи середовища [13], [14].

Spyder просто інтегрується з менеджерами пакетів, легко працює з віртуальними середовищами та дає змогу без ускладнень тримати під контролем необхідні залежності навіть тоді, коли проект вимагає специфічних налаштувань чи окремих версій бібліотек. Такий підхід формує сприятливі умови для роботи над складними оптимізаційними завданнями, де важливо поєднувати обчислення, перевірку й наочну оцінку ефективності в одному програмному просторі [13], [14].

Обґрунтування вибору середовища розробки для створення програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів наведено у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору середовища розробки для створення програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

Критерій порівняння середовищ розробки	Середовища розробки		
	Spyder	PyCharm	Jupyter Notebook
Орієнтація на наукові обчислення	+	+–	+–
Зручність інтерактивного тестування коду	+	+–	+
Інтегрована панель змінних	+	–	–
Наявність засобів налагодження	+	+	–
Інтеграція з консоллю Python	+	+	+–

Таким чином, для створення програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів обрано середовище розробки Spyder, яке надає інтегрований редактор коду з підсвічуванням синтаксису та автодоповненням, що значно полегшує написання й налагодження алгоритмів. Крім того, інтерактивна консоль Spyder дозволяє поетапно перевіряти окремі фрагменти коду без необхідності запускати програму повністю. Це особливо важливо при створенні та тестуванні генетичних операторів, аналізі поведінки популяцій або налагодженні функцій пристосованості. Також Spyder легкий у налаштуванні,

підтримує віртуальні середовища Python та дозволяє гнучко керувати залежностями проєкту, що важливо для наукових розробок, які можуть потребувати специфічних версій бібліотек.

2.3 Еволюційний метод контрольованого витіснення

Застосування традиційних генетичних алгоритмів у задачах пошуку глобального екстремуму для мультимодальних неперервних функцій часто стикається з низкою обмежень, що знижують їхню ефективність у складних просторах рішень. Найбільшою проблемою класичних підходів є схильність до передчасної конвергенції, коли значна частина популяції зосереджується навколо локального максимуму або мінімуму, втрачаючи при цьому здатність досліджувати інші перспективні області. Це пояснюється тим, що у більшості панміксійних стратегій відбір батьків відбувається без урахування просторової близькості, а схрещування не враховує потреби підтримувати різноманіття у межах популяції. У результаті вже через кілька поколінь генетичний дрейф поступово знижує розмаїття рішень, що особливо критично для функцій із багатьма локальними екстремумами.

Крім того, відомі модифікації класичних алгоритмів часто потребують складного налаштування великої кількості параметрів для кожної окремої задачі, що робить їх використання менш універсальним і потребує значного обсягу додаткових експериментів для досягнення прийнятних результатів. З цієї причини стандартні генетичні схеми не завжди дозволяють знайти глобальний екстремум, особливо якщо топологія простору має велике число слабко зв’язаних локальних максимумів або мінімумів.

З огляду на такі обмеження у цій роботі передбачено розробити і використати метод контрольованого витіснення, який дає змогу посилити відбір не лише за значенням пристосованості, а й за критерієм структурної схожості. Суть цього підходу полягає у тому, що нові особини не просто додаються до популяції, а проходять перевірку на подібність до вже наявних,

витісняючи менш придатних, якщо це доцільно. Такий механізм підтримує достатній рівень різноманіття всередині популяції, перешкоджає надмірному скупченню рішень у межах одного локального оптимуму та сприяє постійному дослідженню простору у ширшому діапазоні.

Таким чином, створення еволюційного методу контрольованого витіснення обумовлено прагненням подолати обмеження класичних схем і забезпечити більш стабільний і гнучкий пошук глобального екстремуму для складних мультимодальних функцій без необхідності занадто складного налаштування параметрів для кожної нової задачі. Еволюційний метод контрольованого витіснення розглядається як ключовий інструмент для підвищення надійності та точності генетичних алгоритмів у процесі пошуку глобального оптимуму для складних неперервних функцій, що характеризуються наявністю численних локальних екстремумів і високою неоднорідністю простору рішень.

Під поняттям контрольованого витіснення будемо розуміти послідовність дій, коли новоутворені особини не просто додаються у популяцію, а проходять цільову перевірку схожості зі вже існуючими кандидатами. Якщо новий індивід демонструє кращі показники пристосованості й виявляється близьким за структурою чи параметрами до когось із наявних членів популяції, відбувається цілеспрямоване витіснення менш пристосованої особини. Така процедура дозволяє підтримувати баланс: уникати надмірної концентрації рішень у вже вивчених регіонах і водночас стимулювати пошук у тих ділянках простору, які можуть містити нові потенційні глобальні або локальні оптимуми.

Запропонована схема методично поєднує стандартне схрещування та мутацію з елементами детального порівняння кожного нового генотипу зі специфічною підвибіркою батьків чи подібних особин. Якщо нова комбінація має перевагу, вона замінює більш слабкий аналог, таким чином оновлюючи склад популяції.

На рис. 2.1 наведено узагальнену схему еволюційного методу контрольованого витіснення.

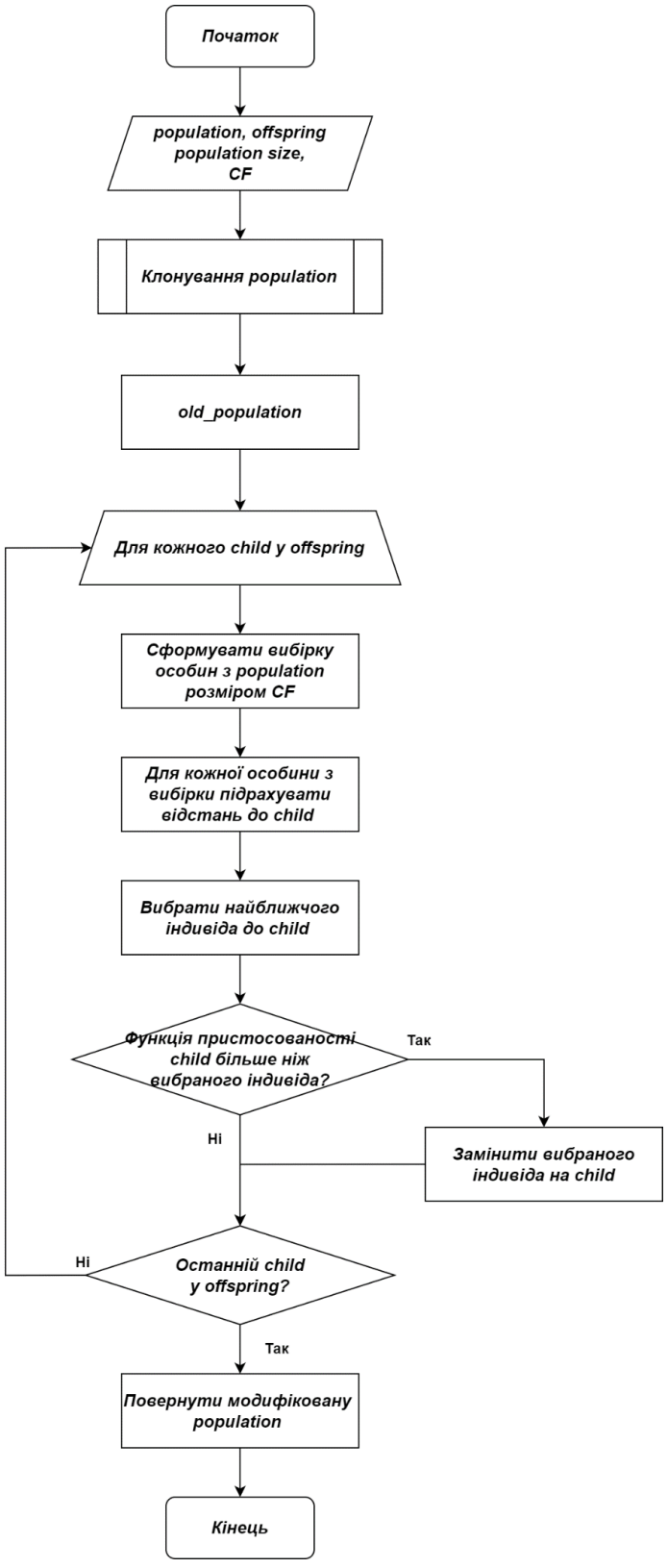


Рисунок 2.1 – Узагальнена схема еволюційного методу контрольованого витіснення

Заплановано застосування детермінованої стратегії порівняння виключно з батьківськими парами для прискорення відбору та додаткове впровадження ймовірнісного механізму, що дозволяє певній частині менш придатних рішень залишатися, аби уникнути одноманітності та забезпечити різноманіття у наступних поколіннях.

Завдяки методові контрольованого витіснення формується динамічне середовище, де нові рішення не просто додаються, а проходять конкурентний відбір за подібністю та якістю, що мінімізує ризик застою алгоритму в межах окремих локальних максимумів. Очікується, що запропонована реалізація продемонструє вищу стійкість і гнучкість порівняно з традиційними панміксійними варіантами, які зазвичай не забезпечують достатнього контролю над структурою популяції. Експериментальна частина проєкту має підтвердити ефективність цієї концепції та обґрунтувати доцільність використання саме підходу контрольованого витіснення для розв’язання широкого кола задач мультимодальної оптимізації.

Запропонований еволюційний метод контрольованого витіснення для оптимізації мультимодальних неперервних функцій організовується як послідовність взаємопов’язаних етапів, що реалізують цілеспрямоване оновлення популяції шляхом порівняння нових рішень із уже наявними та поступового витіснення менш перспективних варіантів. На початковому етапі формується базова популяція та здійснюється генерація першого набору нащадків, що утворюються шляхом традиційних операцій схрещування та мутації.

Далі для кожного нового індивіда проводиться перевірка подібності із певною вибіркою з поточної популяції. Визначається найближчий за структурою або значенням параметрів представник, для чого використовуються критерії відстані у просторі рішень. Після встановлення пари батько-нащадок здійснюється порівняння їх пристосованості. Якщо новоутворений індивід демонструє кращі характеристики, він замінює

подібного батька або іншу близьку за генотипом особину. Якщо ж переваги не виявлено, старий варіант зберігається у складі популяції.

На наступних етапах відбувається перевірка усіх нащадків за тією ж процедурою, допоки не буде оновлено задану частку популяції або завершено весь цикл перегляду. За потреби у метод впроваджується ймовірнісний фактор, що дозволяє залишати частину менш успішних особин для збереження варіативності й запобігання надмірній конвергенції до окремих локальних максимумів чи мінімумів.

Завершальним етапом циклу стає оновлення популяції й підготовка до наступного покоління, що запускається після оцінки результатів та коригування параметрів, якщо це необхідно.

Таким чином, запропоновано еволюційний метод контрольованого витіснення, який поєднує стандартні оператори схрещування та мутації з елементами детального порівняння кожного нового генотипу зі специфічною підвибіркою батьків чи подібних особин, таким чином оновлюючи склад популяції. При цьому застосовується детермінована стратегія порівняння виключно з батьківськими парами для прискорення відбору та додаткове впровадження ймовірнісного механізму, що дозволяє певній частині менш придатних рішень залишатися, аби уникнути одноманітності та забезпечити різноманіття у наступних поколіннях. Це дозволяє забезпечити вищу стійкість і гнучкість популяції порівняно з традиційними генетичними алгоритмами, які зазвичай не забезпечують достатнього контролю над структурою популяції. Завдяки такому підходу еволюційний метод контрольованого витіснення спрямовується на підтримку балансу між експлуатацією вже знайдених рішень і розширеним пошуком у менш досліджених зонах, що є критично важливим для мультимодальних функцій, де велика кількість локальних екстремумів може вводити класичні генетичні алгоритми в оману та знижувати ефективність глобальної оптимізації.

2.4 Висновки за розділом 2

Для реалізації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів обрано мову програмування Python, яка має простий та зрозумілий синтаксис, що значно спрощує розробку, тестування й супровід алгоритмів, а також знижує поріг входження для дослідників і студентів. Крім того, Python підтримує велику кількість потужних бібліотек для наукових обчислень, аналізу даних і оптимізації, що дає змогу швидко будувати і експериментувати з різними варіантами генетичних алгоритмів без необхідності реалізовувати всі компоненти з нуля.

Для створення програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів обрано середовище розробки Spyder, яке надає інтегрований редактор коду з підсвічуванням синтаксису та автодоповненням, що значно полегшує написання й налагодження алгоритмів. Крім того, інтерактивна консоль Spyder дозволяє поетапно перевіряти окремі фрагменти коду без необхідності запускати програму повністю. Це особливо важливо при створенні та тестуванні генетичних операторів, аналізі поведінки популяцій або налагодженні функцій пристосованості.

Запропоновано еволюційний метод контрольованого витіснення, який поєднує стандартні оператори схрещування та мутації з елементами детального порівняння кожного нового генотипу зі специфічною підвибіркою батьків чи подібних особин, таким чином оновлюючи склад популяції. При цьому застосовується детермінована стратегія порівняння виключно з батьківськими парами для прискорення відбору та додаткове впровадження ймовірнісного механізму, що дозволяє певній частині менш придатних рішень залишатися, аби уникнути одноманітності та забезпечити різноманіття у наступних поколіннях. Це дозволяє забезпечити вищу стійкість і гнучкість популяції порівняно з традиційними генетичними алгоритмами, які зазвичай

не забезпечують достатнього контролю над структурою популяції. Завдяки такому підходу еволюційний метод контрольованого витіснення спрямовується на підтримку балансу між експлуатацією вже знайдених рішень і розширеним пошуком у менш досліджених зонах, що є критично важливим для мультимодальних функцій, де велика кількість локальних екстремумів може вводити класичні генетичні алгоритми в оману та знижувати ефективність глобальної оптимізації.

3 ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

Структуру програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів наведено на рис. 3.1.

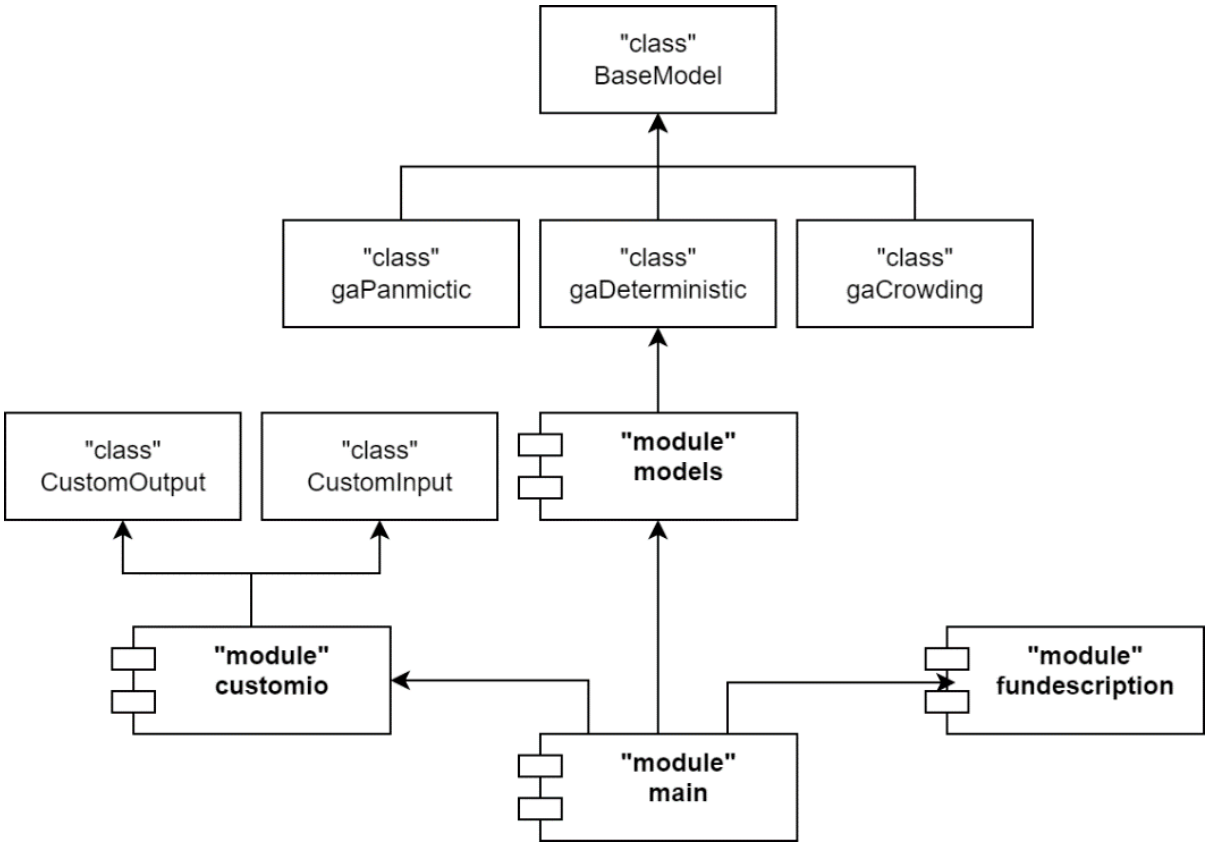


Рисунок 3.1 – Структура програмного забезпечення
для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

У межах реалізації програмного забезпечення, призначеного для виконання задач оптимізації мультимодальних неперервних функцій із використанням генетичних алгоритмів, закладається чітка структура, що поєднує низку взаємопов’язаних складових. Ядром програмної системи виступає модуль models, який акумулює у собі всі реалізовані еволюційні підходи. Серед них передбачено класичний панміксійний алгоритм,

стандартну схему ущільнення та її модифікований варіант, що ґрунтується на детермінованому принципі відбору. Для спрощеного застосування ці методи дублюються у форматі простих функцій, зосереджених у окремій директорії всередині models.

Основою для кожного з еволюційних класів служить BaseModel, який визначає загальний каркас: ініціалізацію ключових параметрів популяції, параметрів мутації, схрещування та інших характеристик, необхідних для роботи алгоритму. Також ця базова структура відповідає за перевірку правильності вхідних даних, що дозволяє уникати помилок ще на етапі запуску, та надає можливість виводити конфігурацію моделі у зручному для редагування форматі.

У складі моделей передбачено спеціалізовані класи для кожного підходу. Так, gaPamnictic відповідає за реалізацію класичного панміксійного сценарію та підтримує два режими запуску: повний і скорочений, що відрізняються рівнем деталізації статистики у процесі пошуку. Власну функцію ідентифікації gaPamnictic перевизначає, аби можна було чітко фіксувати, які саме оператори еволюції активні у конкретному запуску.

Модифікований метод представлений класом gaDeterministic, який успадковує властивості базової моделі, але реалізує еволюційний метод контрольованого витіснення та покращену логіку відбору й заміщення, з урахуванням детерміністичного підходу. Клас дозволяє задавати параметри через конструктор і запускати обчислення як у стандартному режимі, так і у пришвидшеному варіанті. Для реалізації традиційної стратегії контрольованого витіснення передбачений окремий клас gaCrowding, що функціонує за аналогічною схемою і також оперує двома режимами запуску.

Для роботи з тестовими прикладами функцій існує блок fundescription, у якому міститься повний опис цільових функцій. Він охоплює прототипи у вигляді Python-функцій, інформацію про назви, межі простору визначення, відомі оптимальні значення, координати точок оптимумів, а також допустимі відхилення як за значенням пристосованості, так і за просторовою близькістю.

Взаємодія користувача із зовнішніми файлами та параметрами організована через модуль customio, у якому розділено правила обробки вводу та виводу. Блок CustomInput відповідає за завантаження даних про параметри запуску й структуру алгоритмів, визначає шляхи до файлів і правила зчитування інформації. Натомість CustomOutput описує порядок і формат фіксації проміжних і фінальних результатів, кольорову схему та послідовність відображення параметрів у консолі, а також шаблони імен файлів для збереження статистики.

Координацію усіх компонентів забезпечує центральний модуль main, у якому збирається робочий простір проєкту, виконуються імпортування потрібних складових і запускається обчислювальний процес. Саме тут об’єднується логіка окремих частин системи, що дозволяє налаштовувати експерименти, підключати потрібний алгоритм і організовувати цикл оптимізації відповідно до обраного сценарію.

3.2 Функціонування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

Функціонування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів подамо за допомогою схеми, зображеної на рис. 3.2.

Під час запуску створене програмне рішення отримує від користувача заздалегідь визначені алгоритми еволюційного пошуку з заданими конфігураційними параметрами, набір функцій для оптимізації, описаних відповідно до встановленого формату, а також кількість повторів, яку необхідно виконати для статистично коректної оцінки результатів. На етапі виконання всі зазначені вхідні дані проходять попередню перевірку на відповідність, після чого запускається серія обчислювальних експериментів згідно з вибраним сценарієм.

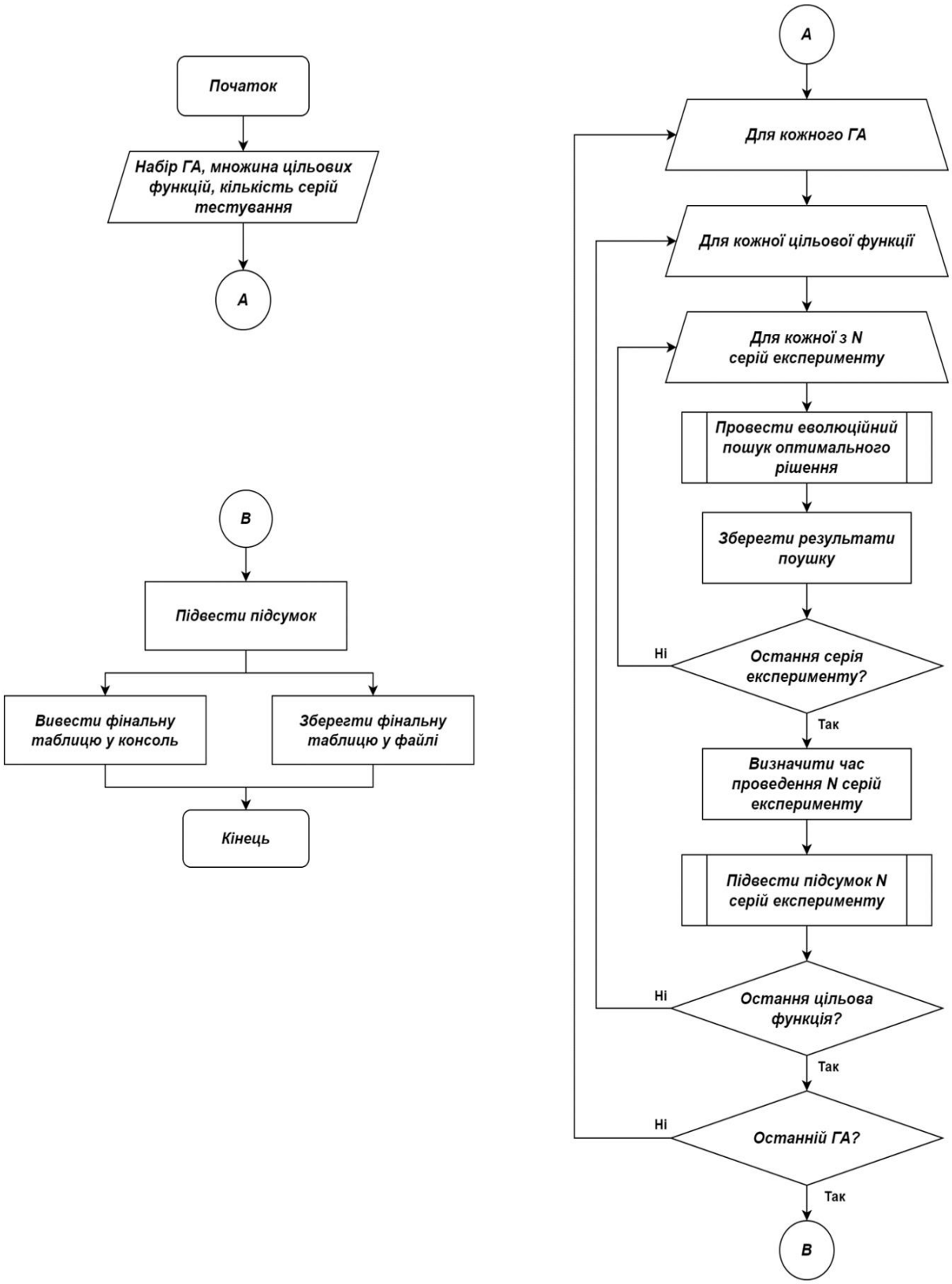


Рисунок 3.2 – Схема функціонування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

По завершенні роботи програмний модуль формує підсумкові дані з ключовими метриками й показниками якості пошуку. Отримана інформація одночасно демонструється у командному рядку, що дозволяє відразу ознайомитися з основними результатами, і дублюється у вихідний файл, який розміщується у заздалегідь визначеній директорії.

Функціонування програмного забезпечення для оптимізації мультимодальних неперервних функцій ґрунтується на організації чіткого взаємозв'язку між усіма внутрішніми компонентами, що забезпечують повний цикл проведення експериментів — від початкового налаштування до фіксації результатів. На першому етапі, після запуску головного модуля, програма звертається до класів, що відповідають за обробку вхідних даних, аби зчитати параметри обраного еволюційного методу, перевірити коректність структури описаних функцій і розподілити їх між серіями тестувань. Це дозволяє одразу виключити некоректні чи неповні конфігурації та запобігти зупинкам виконання під час тривалих обчислень.

Під час самої оптимізації активується один із алгоритмів, реалізованих у вигляді класів, які успадковують спільну модель і дотримуються єдиних стандартів взаємодії. Кожен такий клас управляє створенням популяції, запуском операцій відбору, схрещування та мутації, а також застосуванням додаткових механізмів, таких як запропонований метод контрольованого витіснення, що дозволяє гнучко керувати процесом заміни особин і підтримувати необхідну варіативність у пошуку рішень. Протягом виконання кожного покоління зберігаються проміжні результати: інформація про значення функцій, динаміку зміни пристосованості, склад популяції та розподіл рішень у просторі пошуку.

Ще однією особливістю функціонування цього програмного комплексу є його здатність вести розширений журнал, що дає можливість користувачеві у будь-який момент перевірити, як змінюються ключові параметри і як саме еволюційний пошук реагує на складність топології функції. Кожна серія експериментів зберігається у вигляді окремих файлів, що містять детальну

статистику, графіки розподілу значень і проміжні звіти про ефективність конкретної конфігурації. Таке рішення особливо важливе у науковій роботі, де потрібна можливість відтворити експеримент або провести повторний аналіз з іншими вхідними параметрами.

Варто звернути увагу й на те, що програмне середовище передбачає взаємодію з користувачем у зручному форматі: через консольне повідомлення відображаються не лише фінальні результати, а й підказки про хід процесу, можливі помилки у налаштуваннях чи потенційні конфлікти у версіях бібліотек. У разі необхідності обчислення можна перервати, а після коригування параметрів — поновити з моменту останнього успішного збереження.

Усе це робить запропоноване програмне забезпечення не просто інструментом для запуску генетичних алгоритмів, а повноцінною платформою для глибоких досліджень у сфері оптимізації складних обчислюваних моделей. Гнучкість налаштувань і розширюваність архітектури дозволяють легко інтегрувати нові варіанти еволюційних операторів або додаткові критерії відбору без потреби перебудовувати всю систему. Такий підхід забезпечує високу адаптивність до різних типів задач і відкриває широкі можливості для подальшого розвитку проєкту відповідно до конкретних вимог дослідників або інженерів, що працюють над реальними оптимізаційними кейсами.

3.3 Особливості реалізації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

Особливістю реалізації програмного забезпечення для знаходження оптимальних рішень у складних мультимодальних просторах неперервних функцій на основі генетичних підходів є ретельна побудова його архітектури навколо модульного ядра, де кожен модуль відповідає за конкретний набір

операцій еволюційного циклу. Програма поєднує у собі набір гнучко налаштованих операторів, механізми створення та обслуговування популяції, інструменти для обрахунку пристосованості, а також окремо виділені блоки для спеціалізованих процедур збереження, візуалізації та контролю перебігу оптимізації.

У центрі такої реалізації закладено класи, які визначають як загальні, так і розширені еволюційні методи. Зокрема, окремі об’єкти коду описують функціонування механізмів відбору, схрещування, мутації та процедур витіснення. Розширені можливості надає спеціалізована бібліотека операторів, що дозволяє комбінувати класичні прийоми та модифіковані стратегії, такі як оператори контрольованого витіснення або моделі багатокрокового відбору з додатковими критеріями близькості.

Клас, який описує індивіда, успадковує структуру списку, що дає змогу гнучко працювати з набором генів та зберігати значення пристосованості без потреби у додаткових обгортках. Для швидкого запуску процесу передбачено функції генерації хромосом у межах заданого діапазону та формування повноцінної популяції за одним викликом.

Модуль tools у складі програмного середовища для оптимізації складних мультимодальних неперервних функцій на основі генетичних підходів виконує роль базового сервісного ядра, що відповідає за усі низькорівневі операції із формування, оцінювання та обслуговування популяцій. Головна його мета полягає у забезпеченні всіх необхідних процедур для коректної роботи еволюційних механізмів — від моменту створення окремої хромосоми до обробки статистики поколінь і організації зручного обміну інформацією між окремими етапами пошуку.

Ключовим елементом тут виступає спеціальний клас, який визначає, як виглядає структура кожного індивіда в популяції. Цей клас безпосередньо розширює функціонал списків мови програмування, одночасно додаючи поле для збереження значення пристосованості, що робить подальшу роботу з оцінюванням максимально зручною. Таке рішення спрощує маніпуляції з

генами, дозволяючи зберігати і обробляти як самі дані, так і результати їх аналізу в одному об’єкті.

Важливим компонентом є набір функцій для генерації індивідів та формування повноцінної стартової вибірки. Перший рівень передбачає випадкове формування хромосоми у межах заданих обмежень для кожного гена, тоді як другий — об’єднує такі хромосоми у популяцію потрібного розміру, що економить час і спрощує підготовку до запуску експерименту.

Оцінка рішень реалізована через окремі функції для підрахунку фітнесу як одного індивіда, так і всієї групи. Завдяки цьому модуль здатний швидко обчислювати рівень придатності окремих хромосом і узагальнювати ці дані для всього покоління. Додаткові процедури дозволяють дізнатися про максимум і середній рівень пристосованості у вибірці та визначати найсильніший варіант серед усіх учасників поточного етапу.

Для зручності моніторингу передбачені засоби оперативного виведення статистики. Вони дозволяють не лише зафіксувати ключові метрики кожного покоління, а й гнучко обирати обсяг даних, що відображаються, залежно від цілей — для базового контролю або для поглибленого аналізу ходу еволюції. Крім того, спеціально додано функціонал, що підтримує роботу острівних моделей: можна миттєво отримати і порівняти статистику для декількох незалежних субпопуляцій, які працюють паралельно.

Не менш важливою деталлю є функція клонування індивіда. Вона дозволяє швидко створити копію потрібної особини разом із збереженням її фітнес-значення. Це знижує обчислювальні витрати, оскільки уникнуто повторного підрахунку для незмінних генетичних структур.

Усе це робить модуль tools основою технічного забезпечення всіх ключових операцій еволюційного процесу, підвищуючи зручність програмного засобу, забезпечуючи стабільність роботи і закладаючи фундамент для швидкої модифікації та розширення під різні експериментальні сценарії.

Весь цикл еволюції супроводжується набором функцій, які відповідають за оцінювання як окремих індивідів, так і популяцій загалом. Це дає можливість відстежувати середні та пікові значення пристосованості, а також швидко визначати найбільш придатні особини, що важливо для контролю ходу пошуку та прийняття рішень про подальші мутації чи заміщення.

Модуль select у складі програмного середовища, призначеного для розв'язання задач пошуку оптимумів мультимодальних неперервних функцій, реалізує ключові механізми селекції особин, що дозволяють регулювати напрямок еволюції популяції на кожному циклі роботи алгоритму. Його головне призначення полягає у впровадженні різних стратегій вибору, здатних підтримувати баланс між збереженням найсильніших рішень і стимулюванням генетичного різноманіття.

Основна увага у цьому блоці приділяється кільком поширеним методам відбору, кожен із яких орієнтовано на окремі сценарії пошуку. Зокрема, турнірна схема, що реалізується засобами функції select_tournament, полягає у багаторазовому проведенні міні-змагань всередині популяції. У такому турнірі беруть участь випадково обрані групи особин фіксованого розміру, де кожен міні-турнір завершується тим, що до наступної генерації переходить лише найсильніший претендент. Такий підхід забезпечує помірний селекційний тиск і дозволяє зберігати стійке різноманіття, водночас відсікаючи слабкі рішення.

Ще однією важливою реалізацією є метод, що ґрунтується на ідеї «рулеткового колеса» — select_roulette_wheel. Його суть у пропорційному виборі особин відповідно до їх пристосованості: чим вище значення фітнес-функції, тим більша ймовірність бути обраним для продовження роду. Такий принцип добре підходить для підтримки плавного переходу між поколіннями, стимулюючи послідовне посилення якості рішень.

Окремо у складі модуля реалізовано механізм порогового фільтрування — select_threshold. Цей спосіб дозволяє відсікати усіх представників, рівень

пристосованості яких не перевищує заданого порогового значення. У такий спосіб до наступного кроку потрапляють тільки ті особини, які продемонстрували задовільний результат. Це зручно, коли потрібно різко підвищити якість популяції і посилити селекційний відбір.

Всі ці інструменти разом утворюють гнучкий арсенал засобів, за допомогою яких можна тонко налаштовувати селекційний тиск у залежності від складності завдання, вимог до глибини пошуку і бажаної швидкості збіжності. Завдяки модулю select стає можливим адаптувати поведінку еволюційного ядра до різних класів мультимодальних задач і більш надійно уникати передчасного фіксування у локальних екстремумах.

Модуль crossover у складі програмного комплексу, що орієнтований на оптимізацію мультимодальних неперервних функцій із застосуванням генетичних підходів, виконує центральну роль у створенні нових варіантів рішень шляхом комбінування ознак батьківських особин. Його ключова місія — забезпечити механізми змішування генетичної інформації так, щоб у результаті могли виникати нащадки з потенційно кращими характеристиками пристосованості.

В основі роботи цього модуля закладено кілька реалізацій операторів схрещування, які дозволяють варіювати підхід до об’єднання хромосом залежно від типу завдання та побажань дослідника. Для демонстрації роботи арифметичного способу передбачено функцію test, що у спрощеному вигляді показує, як змінюється результат схрещування за умови використання певного вагового коефіцієнта.

Серед базових процедур передбачено класичний метод одноточкового перетину — crossover_one_point, у якому хромосоми батьків розрізаються в обраному місці, після чого частини обмінюються місцями. Завдяки цьому з двох попередників формується пара нащадків зі змішаними наборами генів.

Ще один популярний варіант — crossover_uniform, що забезпечує рівномірний обмін генами: кожен окремий ген із ймовірністю 50% може бути взятий у одного або іншого батька. Такий підхід ефективний для посилення

різноманіття, особливо коли необхідно уникнути домінування окремих ділянок геному.

Додатково реалізовано crossover_arithmetic, де формування нащадка відбувається через обчислення зваженої комбінації значень генів батьків. Ваговий коефіцієнт, що задається явно або обирається випадково, дозволяє гнучко контролювати частку внеску кожного попередника у кінцевий варіант хромосоми.

Таким чином, модуль crossover надає набір перевірених інструментів для проведення гібридизації особин, відкриваючи широкі можливості для дослідження складних фітнес-ландшафтів та сприяючи ефективному пошуку глобального оптимуму у випадках, коли прості стратегії можуть виявитися недостатніми.

Модуль mutation у складі програмного забезпечення, створеного для вирішення задач багатопікової неперервної оптимізації з використанням генетичних стратегій, забезпечує ключову роль у внесенні варіативності до популяції рішень. Його головне призначення полягає у тому, щоб змінювати окремі гени хромосом таким чином, щоб з’являлися нові комбінації, здатні досліджувати ще не охоплені області простору рішень і тим самим підвищувати шанси знайти глобальний екстремум.

Серед базових складових цього модуля передбачено кілька механізмів реалізації випадкових змін. Так, функція __gene_mutation_gaussian відповідає за обчислення нового значення окремого гена за нормальним розподілом і гарантує, що згенероване значення не виходить за встановлені межі. Аналогічно, __gene_mutation_uniform формує нове значення, обираючи його рівномірно з усього дозволеного інтервалу.

Для роботи з особинами на рівні хромосоми передбачено декілька варіантів мутаційних операторів. Наприклад, mutation_bit_flip виконує інверсію бітів і доцільна переважно для задач, де рішення закодовано у вигляді двійкових послідовностей. Для неперервних представлень більше підходять оператори mutation_gaussian_one_gene та

mutation_uniform_one_gene, які змінюють значення одного з генів відповідно до нормального або рівномірного розподілу й одночасно можуть демонструвати результат мутації у консоль.

Щоб забезпечити елемент адаптивності до різних фаз пошуку, модуль включає процедури mutation_random_one_gene та mutation_non_uniform_one_gene. Ці функції враховують прогрес еволюційного процесу: поточний номер покоління й загальну кількість ітерацій. Такий підхід дозволяє з часом коригувати силу мутації — наприклад, зменшувати діапазон змін для кращої локальної оптимізації на пізніх стадіях або навпаки, підвищувати варіативність на початку пошуку.

Завдяки поєднанню різних механізмів, модуль mutation виступає запорукою генетичного різноманіття, підтримуючи життєздатність популяції та запобігаючи її передчасному «злипненню» у локальних мінімумів. Це робить його невід’ємним елементом будь-якого повноцінного еволюційного середовища.

Мутації підтримуються як бітові, так і неперервні. Реалізовано варіанти із нормальним, рівномірним та змішаним законами зміни генів, а також методи з регульованою ступеневою зміною значень залежно від поточного етапу оптимізації. Це дозволяє не лише варіювати інтенсивність мутацій, але й адаптувати їх до складності топології конкретної функції.

Модуль витіснення, який відповідає за контроль збереження різноманіття у популяції. Для цього застосовується метод вимірювання схожості між батьками та нащадками, а у разі переваги дитини за пристосованістю здійснюється заміщення. При цьому може бути задіяна як проста схема обрання найближчого сусіда, так і більш складний варіант із детермінованою або імовірнісною логікою витіснення.

Модуль витіснення crowding у складі програмного комплексу, орієнтованого на вирішення завдань багатовершинної неперервної оптимізації, відповідає за реалізацію стратегії контрольованого витіснення для підтримки розмаїття рішень протягом усього еволюційного процесу. Його головна роль

— регулювати співіснування подібних особин у популяції, не дозволяючи їм надмірно скупчуватися у вже опрацьованих областях простору рішень.

Ключова функція manhattan_distance дає змогу визначати ступінь близькості між двома особинами за допомогою так званої Манхеттенської метрики, відомої також як відстань Хеммінга у випадках роботи з двома генами. Це обчислення лежить в основі механізму порівняння батьків і нащадків.

Функція crowding_simple відповідає за базову реалізацію алгоритму витіснення. Вона бере початкову групу батьків та сформоване потомство, аналізує відстані між ними й визначає, чи слід здійснити заміну. Якщо дитина перевершує найближчого за схожістю батька за критерієм пристосованості, вона займає його місце у популяції. У результаті формується оновлена вибірка особин, більш збалансована з точки зору якості та розподілу у просторі рішень.

Для реалізації чіткого сценарію заміни передбачена допоміжна процедура _p_function_deterministic, яка визначає, за якими правилами відбувається детерміноване витіснення. В перспективі цей компонент можна доповнити імовірнісними моделями для гнучкішого регулювання відбору.

Також у складі модуля є функція crowding_deterministic, що використовується для попарного порівняння батьків і дітей. Вона дозволяє зіставити найближчі пари та відібрати з них ті особини, що демонструють кращі показники пристосованості. Такий підхід забезпечує своєчасний «тиск» на популяцію, запобігаючи швидкій втраті альтернативних рішень і сприяючи більш глибокому обстеженню простору пошуку.

Завдяки комплексу цих засобів модуль crowding виконує роль механізму тонкого управління різноманітністю всередині поколінь, що є критично важливим для успішної роботи генетичних алгоритмів у складних мультимодальних середовищах.

Важливим також є модуль інформування. Він забезпечує виведення усіх ключових показників роботи алгоритму, як у реальному часі у консоль,

так і у вигляді окремих звітів чи графіків. Завдяки цьому користувач має змогу не лише аналізувати кінцевий результат, а й спостерігати за перебігом пошуку та вчасно коригувати параметри.

Таким чином, загальна структура програмного комплексу об’єднує усі необхідні інструменти для виконання повного циклу оптимізації — від генерації стартової популяції до глибокого аналізу одержаних рішень і подальшого дослідження їх якості. Усе це робить систему придатною для гнучкого використання у наукових розробках і практичних експериментах, де важлива як точність пошуку, так і можливість легко розширювати чи комбінувати підходи відповідно до потреб користувача.

3.4 Взаємодія з користувачем програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

При розробці засобів для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів було враховано вимоги технічного завдання:

– підтримка можливості оптимізації мультимодальних неперервних функцій;

– підтримка можливості використання різних генетичних алгоритмів для оптимізації функцій;

– можливість завдання функції для оптимізації;

– можливість графічного відображення отриманих результатів;

– підтримка можливості задавання параметрів генетичних алгоритмів;

– підтримка можливості подання результатів у табличному вигляді.

Було прийнято рішення про використання консольного способу взаємодії.

У табл. 3.1 наведено визначення основних функцій та приклади їх використання.

Таблиця 3.1 – Визначення основних функцій та приклади їх використання

Визначення функції	Опис функції
fitness_stat, population_stat = gaCrowding(himmelblau_d['fun'], himmelblau_d['low_up'])	ініціювання процесу пошуку рішення за допомогою алгоритму з використанням принципів контрольованого витіснення, де одразу передається обрана цільова функція та її обмеження. Після завершення виконання формується звіт, у якому міститься інформація про хід пошуку, включаючи дані про найкращого знайденого кандидата разом із його генетичним складом та обчисленим значенням критерію якості.
fitness_on_gen_graph(fitness_stat)	дозволяє відобразити, як змінювалося наближення до розв’язку протягом усіх кроків еволюційної процедури. Для побудови графіка залучаються дані, накопичені протягом усього пошуку.
fitness_dynamic(himmelblau_d['fun'], population_stat, himmelblau_d['low_up'])	забезпечує побудову наочних інтерактивних візуалізацій, які демонструють розвиток еволюційного процесу крок за кроком. Окрім самої цільової функції й меж її визначення, за потреби можуть додатково задаватися координати очікуваного екстремуму та параметри масштабування контурної поверхні для кращого відображення структури простору рішень.

Продовження таблиці 3.1

Визначення функції	Опис функції
draw_2arguments_graph(himmelblau_d['fun'], himmelblau_d['low_up'])	реалізує побудову просторової моделі поверхні цільової функції, що залежить від двох аргументів. Завдяки цьому забезпечується можливість графічно оцінити складність топології завдання та візуально спостерігати поведінку функції в заданих межах.
FUNCTIONS = [fun6_d, himmelblau_d, bukin_d, schwefel_d, griewank_d, trefethen_d, tableholder_d, rosenbrockmode_d, newfunction02_d] GENALGOS = [gaPanmictic()] fun_test(FUNCTIONS, GENALGOS, 50)	організовує серію тестів, у ході яких послідовно запускається класичний панміксійний генетичний алгоритм для цілої групи наперед заданих тестових завдань. Запланована кількість повторень дозволяє оцінити стабільність і точність алгоритму, а результат зберігається у вигляді порівняльної таблиці разом із ключовими показниками та з фіксацією витраченого часу. Уся отримана інформація автоматично архівується у файл за заздалегідь визначеним шляхом для подальшого аналізу або звітування.

Для запуску програмного засобу, який виконує оптимізацію неперервних мультимодальних функцій з використанням генетичних алгоритмів, обов’язково задається повний опис цільової функції, що планується мінімізувати чи максимізувати. Передбачено, що користувач чітко фіксує посилання на обраний фрагмент коду, у якому реалізовано обчислення значень функції, зазначає зрозумілу назву для подальшої ідентифікації, а

також визначає межі, у яких будуть шукатися рішення. Крім того, обов’язково задаються координати оптимальних точок, значення глобального екстремуму та рівень точності, після досягнення якого можна вважати, що розв’язання проведено без помилок. Для забезпечення контролю правильності пошуку встановлюється ще й максимально припустима Евклідова відстань, яка відображає, наскільки близько до теоретичного рішення наблизився алгоритм.

Щодо налаштувань самого еволюційного механізму, користувач має вказати повний перелік параметрів, що регулюють поведінку відбору, схрещування та мутацій. Серед таких даних обов’язково зазначається посилання на фітнес-функцію, діапазон допустимих значень, кількість поколінь, які можуть бути сформовані протягом роботи алгоритму, а також імовірність утворення нових особин шляхом схрещування і частота мутаційних змін. Необхідно визначити обсяг популяції та довжину хромосоми, що задає кількість генів у кожному індивіді. Для гнучкого налаштування додатково задаються параметри, які впливають на вид відбору — наприклад, поріг при використанні порогового механізму, імовірність зміни окремого гену під час рівномірного схрещування, коефіцієнт, що впливає на рівень арифметичного комбінування батьківських генів, а також параметри для моделювання нерівномірної мутації. Не менш важливим є фактор, що визначає інтенсивність роботи механізму контрольованого витіснення, адже саме він дозволяє уникати надмірного скупчення рішень у певних ділянках простору. Для підвищення інформативності можна окремо увімкнути чи вимкнути режим виведення проміжних результатів у консоль, що полегшує відстеження процесу на кожному етапі експерименту.

3.5 Висновки за розділом 3

Розроблено програмне забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

Запропоновано структуру програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів. Ядром програмної системи виступає модуль models, який акумулює у собі всі реалізовані еволюційні підходи. Координацію усіх компонентів забезпечує центральний модуль main, у якому збирається робочий простір проєкту, виконуються імпортування потрібних складових і запускається обчислювальний процес. Саме тут об’єднується логіка окремих частин системи, що дозволяє налаштовувати експерименти, підключати потрібний алгоритм і організовувати цикл оптимізації відповідно до обраного сценарію.

Описано функціонування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів. Визначено, що функціонування програмного забезпечення для оптимізації мультимодальних неперервних функцій ґрунтується на організації чіткого взаємозв’язку між усіма внутрішніми компонентами, що забезпечують повний цикл проведення експериментів, від початкового налаштування до фіксації результатів.

Описано особливості реалізації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів. Описано спосіб взаємодії користувача з засобами для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програма призначена для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів. Програма написана на мові Python, яка має простий та зрозумілий синтаксис, що значно спрощує розробку, тестування й супровід алгоритмів, а також знижує поріг входження для дослідників і студентів. В якості середовища розробки для створення програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів обрано Spyder, яке надає інтегрований редактор коду з підсвічуванням синтаксису та автодоповненням, що значно полегшує написання й налагодження алгоритмів.

Програмне забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів забезпечує виконання таких функцій:

– підтримка можливості оптимізації мультимодальних неперервних функцій;

– підтримка можливості використання різних генетичних алгоритмів для оптимізації функцій;

– можливість завдання функції для оптимізації;

– можливість графічного відображення отриманих результатів;

– підтримка можливості задавання параметрів генетичних алгоритмів;

– підтримка можливості подання результатів у табличному вигляді.

Для експлуатації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів необхідно таке апаратне та програмне забезпечення. Система розрахована на запуск у середовищі персонального комп’ютера під управлінням операційної системи Windows версії не нижче 10. Передбачається, що обчислювальні процеси виконуються на процесорі, тактова частота якого складає не менше

одного гігагерца, а краще, якщо вона буде більшою для забезпечення стабільної швидкодії при обробці великих популяцій та численних поколінь. Оперативної пам’яті повинно вистачати для одночасної роботи ядра програми, графічних модулів і службових процесів — рекомендований мінімум складає два гігабайти у випадку використання 64-бітної архітектури. На накопичувачі має бути зарезервовано достатньо місця, щонайменше тридцять два гігабайти, що дозволяє розмістити не лише файли самого додатка, а й проміжні та фінальні дані експериментів, а також звіти чи графічні матеріали. Для коректної побудови графіків та відображення інтерактивних візуалізацій потрібен графічний модуль із підтримкою обробки даних на основі бібліотек DirectX, причому не нижче дев’ятої версії. Щоб користувач міг повноцінно працювати з інтерфейсом програми та мати зручний перегляд графіків і таблиць, монітор має забезпечувати мінімальну роздільну здатність екрана від 1024 на 768 пікселів, проте для кращої деталізації та зручності можуть застосовуватись дисплеї з вищою роздільністю. Усі ці вимоги визначають лише базовий рівень, що у разі використання більш складних моделей чи великих обчислювальних серій може бути адаптований у бік більш потужних характеристик.

Функціональні характеристики розробленого програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів наведено у технічному завданні (додаток А). Фрагмент тексту програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів наведено у додатку Б.

4.2 Характеристики програми для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

Програмне забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів розроблено як автономний десктопний інструмент, орієнтований на роботу у середовищі

операційної системи для персональних комп’ютерів. Реалізація побудована на модульній архітектурі, що дозволяє окремо налаштовувати та використовувати еволюційні оператори, методи відбору, різновиди схрещувань і варіанти мутацій. Програма підтримує гнучке конфігурування параметрів популяції, кількості поколінь, ймовірностей генетичних перетворень, що робить її придатною як для базового навчання принципам еволюційних алгоритмів, так і для практичного експериментування з різними стратегіями пошуку глобального оптимуму.

Інтерфейс передбачає інтегровані візуалізатори, які дозволяють відстежувати динаміку еволюційного процесу, фіксувати зміни показників пристосованості та аналізувати структуру популяцій у різні моменти виконання. Передбачено можливість збереження результатів у файли, що спрощує подальший аналіз та звітність.

Програмний код організовано так, що кожен компонент можна модернізувати або замінювати, розширюючи функціонал без необхідності суттєвої перебудови всієї системи.

Передбачено просту інтеграцію нових тестових функцій та модифікацій існуючих алгоритмів. Важливою рисою залишається підтримка обробки великої кількості експериментів у серіях, що дозволяє оцінювати стабільність роботи алгоритму та порівнювати різні методики за однакових умов.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Для запуску програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів необхідно в обраному каталозі запустити відповідний ру-файл.

4.3.2 Вхідні й вихідні дані

Вхідними даними до програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів є інформація про функцію, що необхідно оптимізувати та параметри генетичного пошуку (розмір популяції, ймовірності мутації та схрещування, максимальна кількість ітерацій та ін.).

Вихідними даними програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів є результати оптимізації функції (значення цільової функції), побудовані графіки процесу генетичного пошуку та ін.

4.3.3 Повідомлення

Під час роботи програми, що виконує оптимізацію неперервних мультимодальних функцій із застосуванням генетичних алгоритмів, користувач може отримати різноманітні сповіщення, які сигналізують про помилки у вихідних даних або неправильне налаштування параметрів. Серед таких повідомлень може бути інформація про те, що значення порогової величини для відбору вийшло за межі допустимого діапазону, а отже подальша робота обраного селекційного механізму є неможливою. Якщо під час розрахунку мутаційного оператора виявляється некоректно передана кількість ітерацій, що не може бути більшою за нуль, алгоритм повідомить про неможливість обчислення логарифмічного виразу, наслідком чого стає отримання нескінченного результату. У разі, коли користувач не надає всі обов’язкові для запуску дані чи параметри еволюційної стратегії, система негайно сповістить про їх відсутність і запропонує внести уточнення. Якщо у шляху до файлу джерела або призначення вказано невірне ім’я чи розташування, програма нагадає про те, що потрібний файл знайти не вдалося. Також може бути виведено попередження про недопустимість від’ємної

ємності списку найкращих рішень для реалізації стратегії збереження елітних особин. Повідомлення про помилку виводиться і тоді, коли кількість циклів перевірки задана від'ємною величиною, що суперечить логіці багаторазового запуску.

4.4 Виконання програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

У процесі запуску програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів користувач вводить вхідні параметри задає цільову функцію, що необхідно оптимізувати, та параметри генетичного пошуку (розмір популяції, ймовірності мутації та схрещування, максимальна кількість ітерацій та ін.).

У процесі генетичної оптимізації виводиться інформація про ефективність пошуку. На рис. 4.1 наведено приклад графіку розподілу індивідів першої популяції у процесі пошуку.

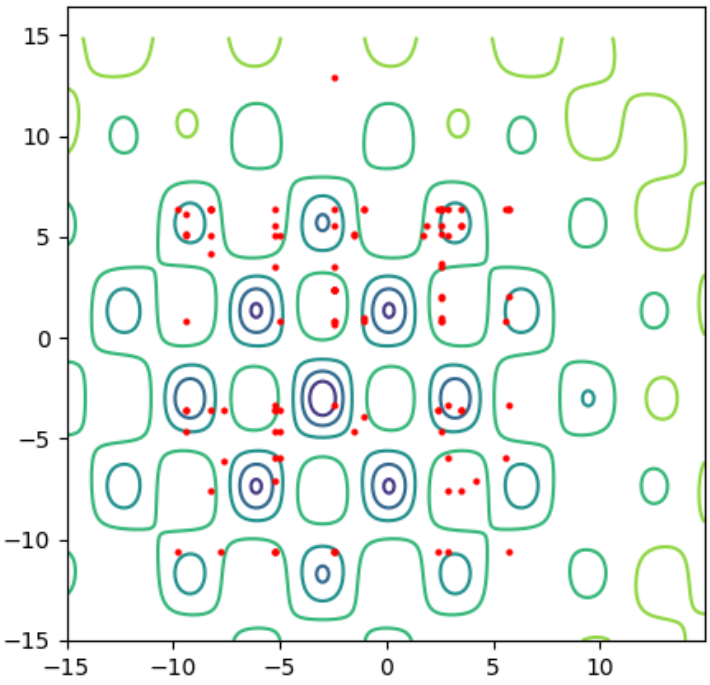


Рисунок 4.1 – Приклад графіку розподілу індивідів першої популяції у процесі пошуку

Крім того виводяться графіки середнього та оптимального значення цільової функції (функції пристосованості) на кожній ітерації (рис. 4.2), що дає можливість аналізу ефективності процесу генетичного пошуку та налаштування його параметрів.

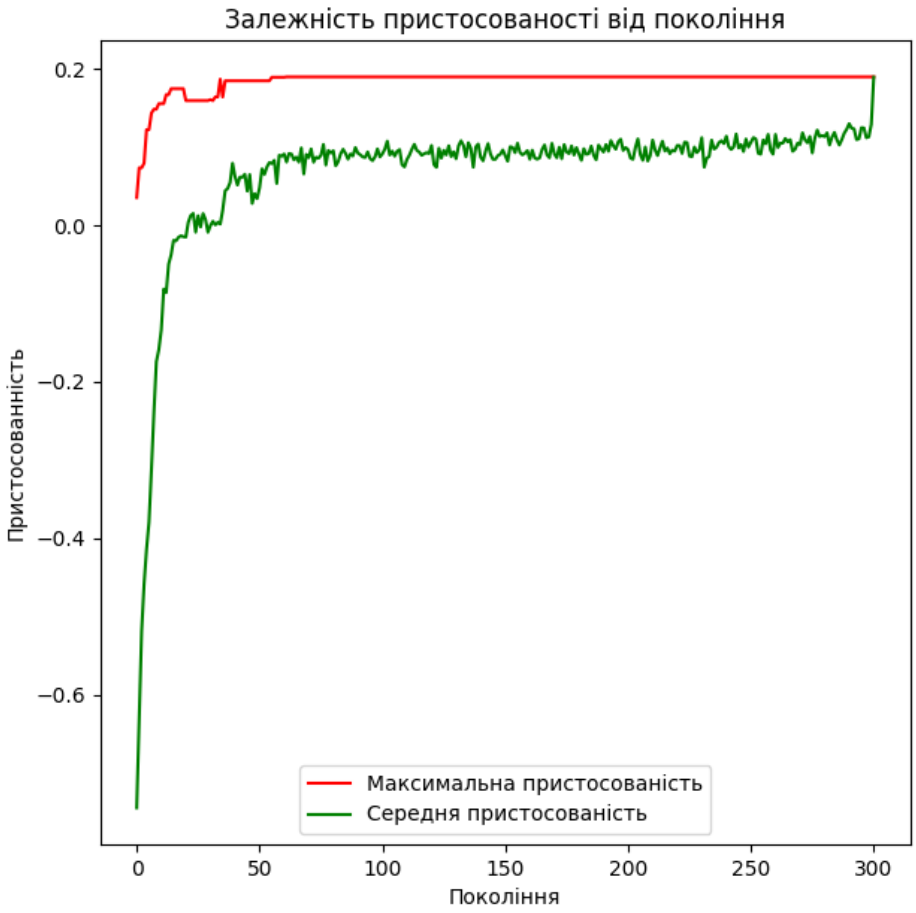


Рисунок 4.2 – Графік залежності значення цільової функції в залежності від номеру ітерації

По закінченню процесу генетичного пошуку виводяться результати оптимізації цільових функцій, подані у табличному вигляді (рис. 4.3).

```
K = 0.9
func6 done. Ellapsed time: 14.344062566757202 s.
func_matyas done. Ellapsed time: 7.746044397354126 s.
func_himmelblau done. Ellapsed time: 8.217491626739502 s.
func_booth done. Ellapsed time: 8.060330152511597 s.
+-----------------+----------------------+----------------------+-----------------+---------------------+--------+
|     Fun name    |     Max fitness      |     Mean fitness     | Found solutions | Euclidean solutions |  Time  |
+-----------------+----------------------+----------------------+-----------------+---------------------+--------+
|      func6      | -0.18766915901164347 | -0.2826841769024185  |        6        |          10         | 14.344 |
|   func_matyas   | -0.10814305852471374 | -0.1850331234286794  |        11       |          6          | 7.746  |
| func_himmelblau | -1.1229553240863008  |  -3.573311690216121  |        1        |          30         | 8.217  |
|    func_booth   | -3.6890351261434557  | -6.0288670952688195  |        0        |          6          |  8.06  |
+-----------------+----------------------+----------------------+-----------------+---------------------+--------+
K = 0.95
func6 done. Ellapsed time: 14.236966133117676 s.
func_matyas done. Ellapsed time: 7.789100885391235 s.
func_himmelblau done. Ellapsed time: 8.162423372268677 s.
func_booth done. Ellapsed time: 8.065345048904419 s.
+-----------------+----------------------+----------------------+-----------------+---------------------+--------+
|     Fun name    |     Max fitness      |     Mean fitness     | Found solutions | Euclidean solutions |  Time  |
+-----------------+----------------------+----------------------+-----------------+---------------------+--------+
|      func6      | -0.2680015089863188  | -0.43657801110047784 |        0        |          0          | 14.237 |
|   func_matyas   | -0.19748843208334205 | -0.8978280134999826  |        5        |          4          | 7.789  |
| func_himmelblau |  -4.386214266189097  | -18.628142679060915  |        0        |          14         | 8.162  |
|    func_booth   | -10.698256893185782  | -25.265294042899477  |        0        |          1          | 8.065  |
+-----------------+----------------------+----------------------+-----------------+---------------------+--------+
```

Рисунок 4.3 – Результати оптимізації цільових функцій, подані у табличному вигляді

4.5 Тестування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

Виконано тестування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

За допомогою розроблених програмних засобів можна досліджувати та модифікувати генетичні алгоритми, а також обирати параметри пошуку.

Під час виконання дослідження здійснювалося оцінювання характеру взаємозалежності між окремими налаштуваннями алгоритму та отриманою середньою величиною пристосованості найсильнішого рішення, що обчислювалася на основі кількох десятків повторів для кожної конфігурації. Для кількісної перевірки зв’язків використовувався стандартний показник попарної кореляції, який дозволив виявити ступінь впливу кожного параметра. Візуальний аналіз результатів (рис. 4.4) свідчить про те, що найбільшу силу зв’язку з кінцевим показником ефективності демонструють такі змінні, як допустима межа кількості поколінь та чисельність популяції, для яких простежується виразний прямолінійний характер залежності із досить

високими значеннями коефіцієнтів. Для величини, що відповідає ймовірності застосування оператора схрещування, зафіксовано яскраво виражену залежність, але вже у вигляді криволінійної взаємодії.

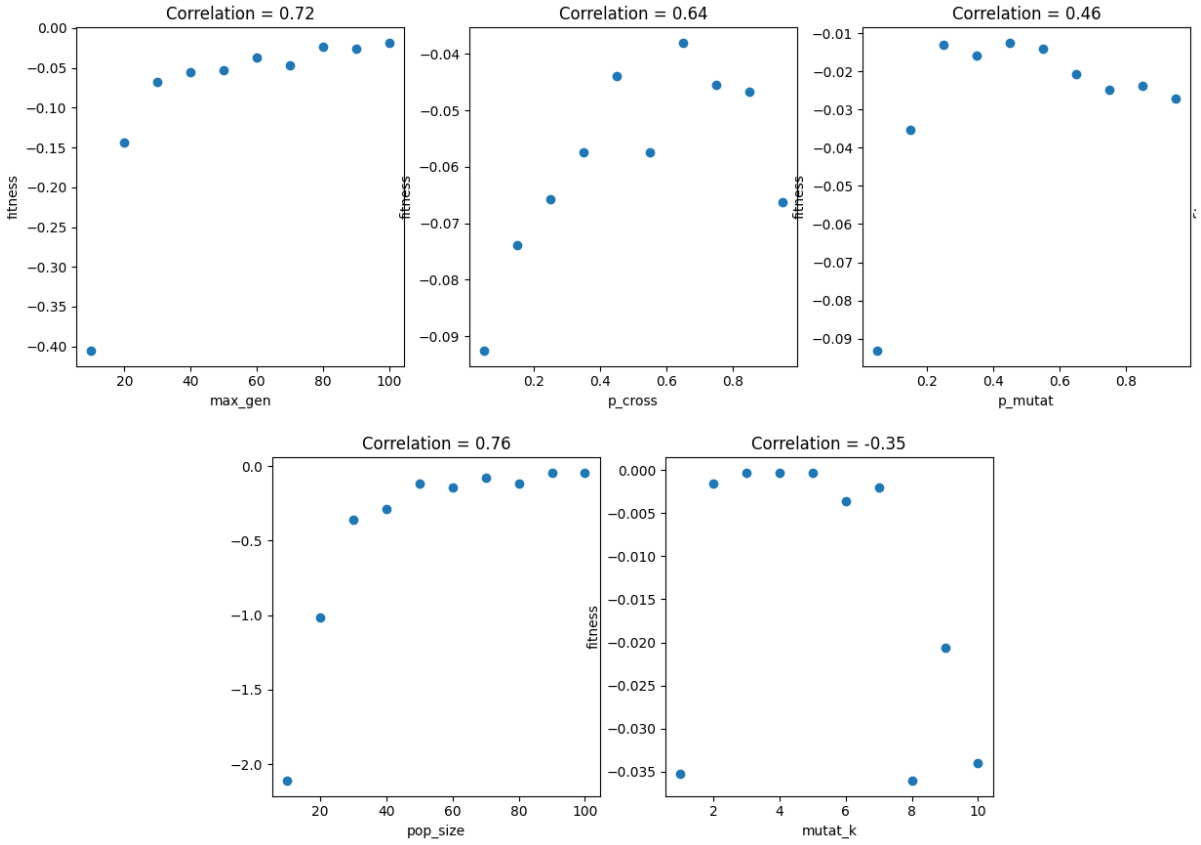


Рисунок 4.4 – Результати дослідження залежності значення цільової функції fitness від параметрів генетичного пошуку

У випадку із параметром, який регулює вірогідність мутаційних змін, спостерігається менш помітний, проте все ж лінійний взаємозв'язок. Що стосується коефіцієнта, котрий визначає нерівномірність змін під час мутації, то його вплив має складний характер і важко піддається точному опису за допомогою звичайних математичних моделей. Загальний висновок підкреслює необхідність ретельного добору чисельності популяції для підвищення стабільності роботи вдосконаленої версії алгоритму.

Протягом дослідження була окремо проаналізована робота механізму елітизму, де ключову увагу приділено пошуку доцільної кількості особин, що

підлягають збереженню у наступній популяції. Задля цього здійснювалося кілька десятків окремих запусків із фіксованими параметрами для ряду цільових функцій, що мають нульову величину оптимального значення, аби спростити порівняння отриманих результатів. На основі зібраних даних встановлено, що певна чисельність елітних хромосом (30) дозволяє утримувати найбільш успішні рішення, одночасно не перевантажуючи популяцію зайвими кандидатами.

Графічна інтерпретація (рис. 4.5) цих спроб продемонструвала, що при визначеній кількості збережених рішень результативність алгоритму підвищується, тоді як надто низький обсяг сховища може призвести до втрати перспективних варіантів у процесі відбору.

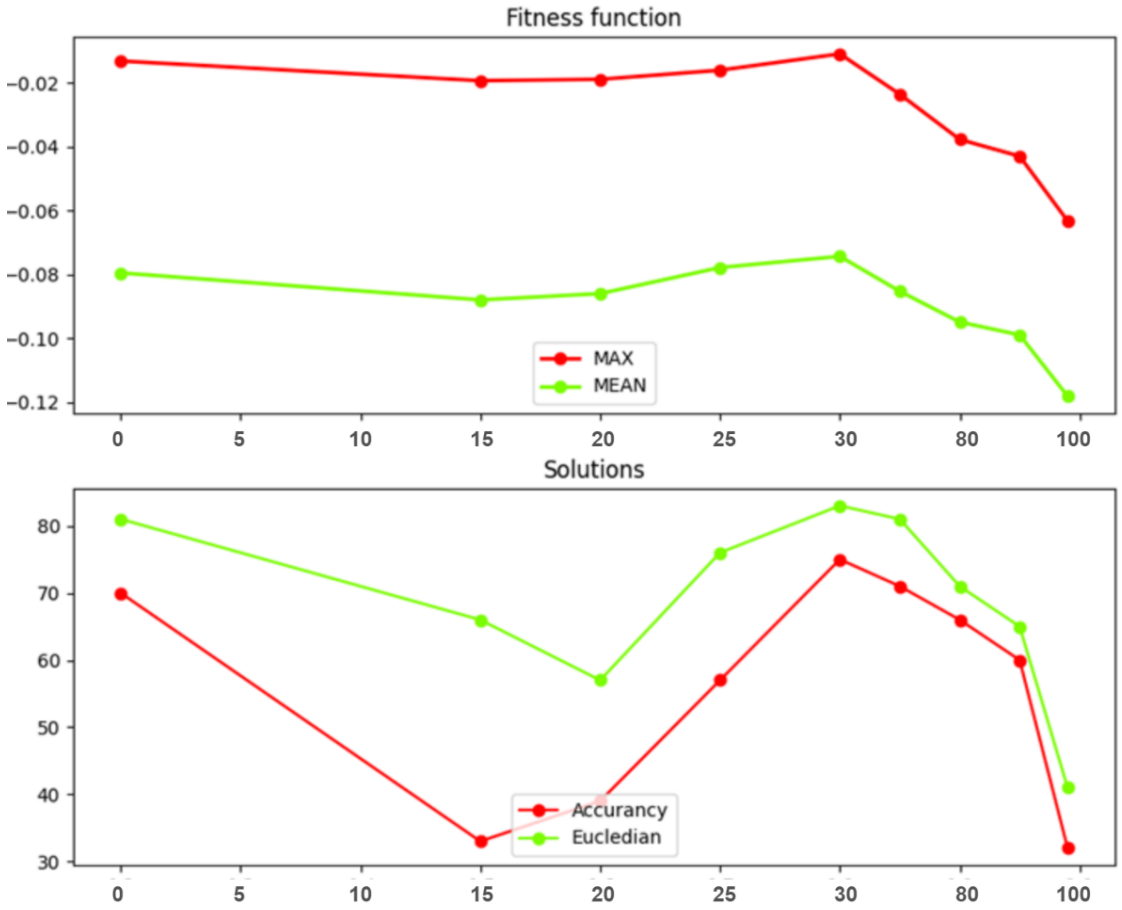


Рисунок 4.5 – Результати дослідження вибору кількості елітних індивідів для генетичної оптимізації мультимодальних функцій

У протилежному випадку, коли таких елітних екземплярів стає забагато, у подальше покоління просочуються неперспективні хромосоми, що нівелює користь самого механізму. Таким чином, для збереження балансу між стабільністю та глибиною пошуку обрано таке обмеження, яке забезпечує оптимальне співвідношення між кількістю еліти та загальною розмірністю популяції.

У продовження дослідження окремо розглядалася ступінь впливу механізму збереження найкращих рішень на загальну різноманітність у межах популяції. Щоб перевірити, наскільки внесення елітизму змінює картину пошуку, було здійснено порівняння між класичним панміксійним алгоритмом, стандартною схемою ущільнення без будь-якого збереження кращих та модифікованим варіантом, де частина індивідів переноситься далі без змін. Аналіз отриманих графіків (рис. 4.6) підтвердив, що впровадження постійної частки елітних учасників майже не спотворює варіативність популяції, впливаючи на неї мінімально можливим чином.

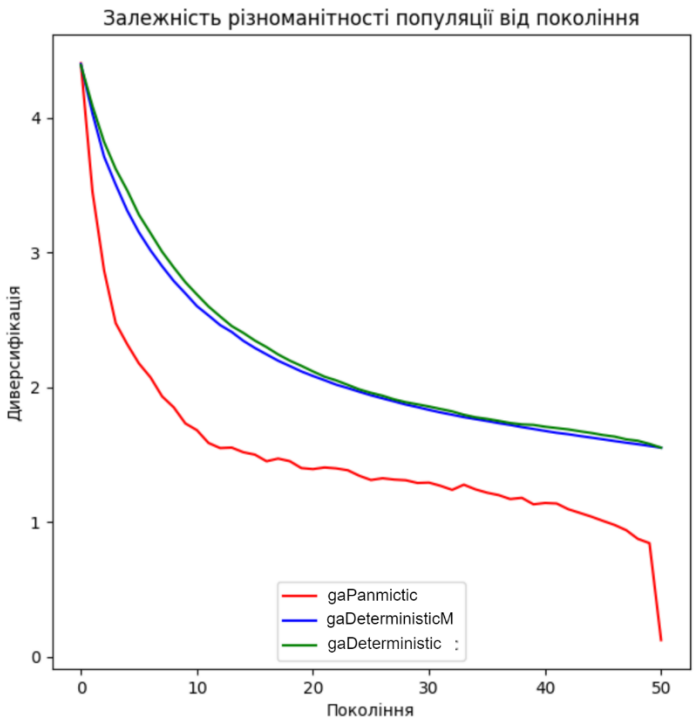


Рисунок 4.6 – Графік залежності різноманітності популяції індивідів від номеру ітерації пошуку

Такий вплив пояснюється тим, що стабільно повторювані особини, які гарантовано переходять у наступні покоління, дещо зменшують розкид рішень, однак ця зміна залишається практично непомітною для загального процесу еволюції. З огляду на цей результат, для поліпшеної моделі ущільнення остаточно визначено конфігурацію ключових параметрів, де поєднується значний обсяг популяції, велика кількість ітерацій, висока ймовірність мутації та мала частка схрещування. Також закладено стабільну кількість кращих особин, що дозволяє гарантувати збереження найбільш перспективних варіантів без ризику втрати корисних рішень у процесі відбору та подальшого відновлення поколінь.

Виявлено, що програма для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів функціонує правильно та злагоджено. Вона реалізує всі функціональні вимоги та успішно виконує свою основну задачу підтримки процесу оптимізації цільових функцій.

Розроблене програмне забезпечення дозволяє забезпечувати автоматизацію процесів, пов’язаних з підтримкою оптимізації функцій при розв’язанні практичних завдань аналізу даних, прогнозування, діагностування.

4.6 Висновки за розділом 4

Описано програмне забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів. Виконано тестування розробленого програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов’язаних з підтримкою оптимізації функцій при розв’язанні практичних завдань аналізу даних, прогнозування, діагностування.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процеси обчислень, пов'язані з оптимізацією мультимодальних неперервних функцій за допомогою генетичних алгоритмів.

Визначено, що у завданнях пошуку глобального екстремуму для мультимодальних неперервних функцій доцільно використовувати генетичні алгоритми через те, що такі алгоритми здатні ефективно працювати з просторами рішень, у яких традиційні методи локального пошуку часто втрачають здатність виходити за межі найближчої області локального оптимуму. Природа мультимодальних функцій передбачає наявність багатьох піків або западин, що можуть вводити в оману градієнтні чи прямі методи, оскільки ті схильні рухатись у напрямку найшвидшого зростання або спадання і не завжди спроможні подолати локальні перешкоди. Генетичні алгоритми запозичують принципи природного добору та випадкових мутацій, що дає змогу не обмежуватися лише одним напрямком руху в пошуковому просторі, а охоплювати його ширше завдяки наявності популяції рішень, які конкурують між собою. Завдяки цьому формується механізм, що поєднує локальну експлуатацію вже знайдених рішень із глобальним дослідженням нових областей. Такий баланс між дослідженням і використанням наявної інформації є ключовим фактором успішного пошуку глобального екстремуму. Крім того, генетичні алгоритми легко адаптуються до функцій, для яких складно або неможливо обчислити похідні, що розширює спектр їхнього використання для різних типів безперервних завдань. Застосування мутації, різних типів селекції та схрещування створює умови для уникнення передчасного збігу рішень до локального максимуму чи мінімуму. Це робить генетичний підхід універсальним і придатним для ситуацій, де інші методи обчислень дають слабкі результати через багатовершинність або складний рельєф цільової функції.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить генетичних алгоритмів для оптимізації мультимодальних неперервних функцій. Проте незважаючи на очевидну ефективність використання генетичних підходів, які цілеспрямовано підтримують різноманітність серед особин для вирішення задач мультимодальної неперервної оптимізації, не можна не враховувати певних обмежень, притаманних класичним схемам заміщення чи ущільнення. По-перше, методи класичного ущільнення, хоч і добре стримують передчасну конвергенцію, часто мають надмірну жорсткість при виборі пар «батько-дитина», що може призводити до блокування переходу популяції до більш перспективних областей простору рішень, якщо початкове розміщення особин виявилося невдалим. Детерміноване ущільнення, у свою чергу, підсилює цей ефект через однозначні правила заміщення, які можуть ігнорувати локальні можливості для стрибка через долини у багатовершинному ландшафті. Імовірнісні підходи частково згладжують жорсткість, однак це часто відбувається ціною збільшення флуктуацій пристосованості, що може гальмувати стабільну збіжність або потребує додаткового точного налаштування параметрів імовірнісного відбору. Крім того, традиційні схеми не завжди дозволяють ефективно балансувати інтенсивність витіснення слабких особин і збереження вузьких ніш із перспективним, але ще не найкращим генетичним матеріалом.

З огляду на такі суперечності, виникає потреба у вдосконаленні механізму, який одночасно дозволяв би уникати передчасної збіжності до одного максимуму чи мінімуму, зберігав би регіональну різноманітність та не зупиняв би розвиток менш домінуючих, але потенційно цінних напрямів пошуку. Саме тому доречно розробити модифікований еволюційний метод контрольованого витіснення та створити відповідне програмне забезпечення, яке буде поєднувати переваги ущільнення та елементів гнучкого імовірнісного впливу на механізми заміщення, і реалізувати цю ідею у спеціалізованому програмному рішенні. Такий підхід дозволить об’єктивно підвищити якість пошуку глобального екстремуму у складних мультимодальних ландшафтах і

водночас знизити залежність результатів від вузько визначених параметрів і налаштувань, роблячи процес оптимізації більш стійким і керованим.

Тому актуальною є розробка еволюційного методу контрольованого витіснення та відповідного програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

Сформульовано функціональні вимоги до програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

Для реалізації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів обрано мову програмування Python, яка має простий та зрозумілий синтаксис, що значно спрощує розробку, тестування й супровід алгоритмів, а також знижує поріг входження для дослідників і студентів. Крім того, Python підтримує велику кількість потужних бібліотек для наукових обчислень, аналізу даних і оптимізації, що дає змогу швидко будувати і експериментувати з різними варіантами генетичних алгоритмів без необхідності реалізовувати всі компоненти з нуля.

Для створення програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів обрано середовище розробки Spyder, яке надає інтегрований редактор коду з підсвічуванням синтаксису та автодоповненням, що значно полегшує написання й налагодження алгоритмів. Крім того, інтерактивна консоль Spyder дозволяє поетапно перевіряти окремі фрагменти коду без необхідності запускати програму повністю. Це особливо важливо при створенні та тестуванні генетичних операторів, аналізі поведінки популяцій або налагодженні функцій пристосованості.

Новизна роботи полягає у тому, що запропоновано еволюційний метод контрольованого витіснення, який поєднує стандартні оператори схрещування та мутації з елементами детального порівняння кожного нового генотипу зі специфічною підвибіркою батьків чи подібних особин, таким чином

оновлюючи склад популяції. При цьому застосовується детермінована стратегія порівняння виключно з батьківськими парами для прискорення відбору та додаткове впровадження ймовірнісного механізму, що дозволяє певній частині менш придатних рішень залишатися, аби уникнути одноманітності та забезпечити різноманіття у наступних поколіннях. Це дозволяє забезпечити вищу стійкість і гнучкість популяції порівняно з традиційними генетичними алгоритмами, які зазвичай не забезпечують достатнього контролю над структурою популяції. Завдяки такому підходу еволюційний метод контрольованого витіснення спрямовується на підтримку балансу між експлуатацією вже знайдених рішень і розширеним пошуком у менш досліджених зонах, що є критично важливим для мультимодальних функцій, де велика кількість локальних екстремумів може вводити класичні генетичні алгоритми в оману та знижувати ефективність глобальної оптимізації.

Практична цінність роботи полягає у тому, що розроблено програмне забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

Запропоновано структуру програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів. Ядром програмної системи виступає модуль models, який акумулює у собі всі реалізовані еволюційні підходи. Координацію усіх компонентів забезпечує центральний модуль main, у якому збирається робочий простір проєкту, виконуються імпортування потрібних складових і запускається обчислювальний процес. Саме тут об’єднується логіка окремих частин системи, що дозволяє налаштовувати експерименти, підключати потрібний алгоритм і організовувати цикл оптимізації відповідно до обраного сценарію.

Описано функціонування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів. Визначено, що функціонування програмного забезпечення для оптимізації мультимодальних неперервних функцій ґрунтується на організації чіткого

взаємозв’язку між усіма внутрішніми компонентами, що забезпечують повний цикл проведення експериментів, від початкового налаштування до фіксації результатів.

Описано особливості реалізації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів. Описано спосіб взаємодії користувача з засобами для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

Виконано тестування розробленого програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов’язаних з підтримкою оптимізації функцій при розв’язанні практичних завдань аналізу даних, прогнозування, діагностування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. An Introduction to Genetic Algorithms: The Concept of Biological Evolution in Optimization [Electronic resource]. – Access mode: https://towardsdatascience.com/an-introduction-to-genetic-algorithms-the-concept-of-biological-evolution-in-optimization-fc96e78fa6db/.

2. Genetic Algorithms [Electronic resource]. – Access mode: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_introduction.htm.

3. Genetic algorithm [Electronic resource]. – Access mode: https://optimization.cbe.cornell.edu/index.php?title=Genetic_algorithm.

4. Genetic Algorithm: Complete Guide With Python Implementation [Electronic resource]. – Access mode: https://www.datacamp.com/tutorial/genetic-algorithm-python.

5. What Are Genetic Algorithms? Working, Applications, and Examples [Electronic resource]. – Access mode: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-are-genetic-algorithms/.

6. Complete Guide to Genetic Algorithms – From Theory to Implementation [Electronic resource]. – Access mode: https://scienceofbiogenetics.com/articles/complete-guide-to-genetic-algorithms-from-theory-to-implementation.

7. Genetic Algorithm A Literature Review [Electronic resource]. – Access mode: https://ieeexplore.ieee.org/document/8862255.

8. A Complete Guide to Genetic Algorithm Advantages, Limitations & More [Electronic resource]. – Access mode: https://medium.com/@byanalytixlabs/a-complete-guide-to-genetic-algorithm-advantages-limitations-more-738e87427dbb.

9. Understanding the Working Principle of Genetic Algorithm [Electronic resource]. – Access mode: https://scienceofbiogenetics.com/articles/understanding-

the-working-principle-of-genetic-algorithm-unleashing-the-power-of-evolutionary-computation.

10. Genetic Algorithms — Intuitively and Exhaustively Explained [Electronic resource]. – Access mode: https://iaee.substack.com/p/genetic-algorithms-intuitively-and.

11. Python [Electronic resource]. – Access mode: https://www.python.org/.

12. Learn Python [Electronic resource]. – Access mode: https://www.freecodecamp.org/news/learn-python-free-python-courses-for-beginners/.

13. Spyder [Electronic resource]. – Access mode: https://pypi.org/project/spyder/.

14. Spyder: The Scientific Python Development Environment — Documentation [Electronic resource]. – Access mode: https://docs.spyder-ide.org/3/index.html.

ДОДАТОК А

Технічне завдання

Вступ

Програмне забезпечення може використовуватися для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та програмна реалізація генетичних алгоритмів оптимізації мультимодальних неперервних функцій», затверджене наказом Національного університету «Запорізька політехніка» № 447 від 30 вересня 2025 р.

А.2 Призначення розробки

Програмний продукт призначений для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів.

А.3 Основні вимоги до програми, що розробляється

А.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів забезпечує виконання таких функцій:

– підтримка можливості оптимізації мультимодальних неперервних функцій;

– підтримка можливості використання різних генетичних алгоритмів для оптимізації функцій;

– можливість завдання функції для оптимізації;

– можливість графічного відображення отриманих результатів;

– підтримка можливості задавання параметрів генетичних алгоритмів;

– підтримка можливості подання результатів у табличному вигляді.

А.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів повинен бути зручним для користувачів. Повинна бути забезпечена можливість роботи в візуальному режимі.

А.3.3 Вимоги до надійності

Програмне забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів повинно забезпечити надійне функціонування.

А.3.4 Умови експлуатації

Для експлуатації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів необхідна наявність персонального комп’ютера.

А.3.5 Вимоги до складу та параметрів технічний засобів

Для експлуатації програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів необхідно таке апаратне та програмне забезпечення. Система розрахована на запуск у середовищі персонального комп’ютера під управлінням операційної

системи Windows версії не нижче 10. Передбачається, що обчислювальні процеси виконуються на процесорі, тактова частота якого складає не менше одного гігагерца, а краще, якщо вона буде більшою для забезпечення стабільної швидкодії при обробці великих популяцій та численних поколінь. Оперативної пам’яті повинно вистачати для одночасної роботи ядра програми, графічних модулів і службових процесів — рекомендований мінімум складає два гігабайти у випадку використання 64-бітної архітектури. На накопичувачі має бути зарезервовано достатньо місця, щонайменше тридцять два гігабайти, що дозволяє розмістити не лише файли самого додатка, а й проміжні та фінальні дані експериментів, а також звіти чи графічні матеріали. Для коректної побудови графіків та відображення інтерактивних візуалізацій потрібен графічний модуль із підтримкою обробки даних на основі бібліотек DirectX, причому не нижче дев’ятої версії. Щоб користувач міг повноцінно працювати з інтерфейсом програми та мати зручний перегляд графіків і таблиць, монітор має забезпечувати мінімальну роздільну здатність екрана від 1024 на 768 пікселів, проте для кращої деталізації та зручності можуть застосовуватись дисплеї з вищою роздільністю. Усі ці вимоги визначають лише базовий рівень, що у разі використання більш складних моделей чи великих обчислювальних серій може бути адаптований у бік більш потужних характеристик.

А.3.6 Вимоги до маркування і пакування

Програма для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів може бути записана на будь-якому носії інформації.

На пакуванні повинна бути назва програми – «Програмне забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів».

А.4 Вхідні дані до роботи

Вхідними даними до програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів є інформація про функцію, що необхідно оптимізувати та параметри генетичного пошуку (розмір популяції, ймовірності мутації та схрещування, максимальна кількість ітерацій та ін.).

Вихідними даними програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів є результати оптимізації функції (значення цільової функції), побудовані графіки процесу генетичного пошуку та ін.

ДОДАТОК Б

Фрагмент тексту програми

```
class gaVytisning(BsMdl):

    def __init__(self,
                 fitnsfunc=None,
                 lwuprng=None,
                 mxgeners=50,
                 pcross=.8,
                 pmut=.8,
                 genertnsz=1000,
                 arithmk=.8,
                 knonn=.5,
                 Vytisningfctr=4,
                 isprnt=True,
                 isftstatlght=False,
                 isppstclct=True):

        super().__init__(fitnsfunc, lwuprng, mxgeners,
                         pcross, pmut, genertnsz,
                         arithmk, knonn, isprnt, isftstatlght,
                         isppstclct)

        self.Vytisningfctr=Vytisningfctr


    def gtnm(self):
        return f'gaVytisning | px:{self.pcross} | pm:{self.pmut}
| '\
            f'ps:{self.genertnsz} | mg:{self.mxgeners} | ' \
            f'sel:turn | x:unfrm | m:non-un |
cf:{self.Vytisningfctr} | '\
            f'k:self.knonn'


    def invkg(self):
        if not self._is_valid():
            return

        genertn = instrmnt.create_genertn(self.lwuprng,
self.genertnsz)
        instrmnt.count_genertn_fitn_2(self.fitnsfunc, genertn,
wght=-1.0)

        genrtnlichlk = 0
        fitnstt = []
        genertnstt = []

        def genertnStat(genertn):
```

```
            fitnstt.append(instrmnt.print_genrtn_info(
                genertn, genrtnlichlk, self.isprnt,
self.isftstatlght))
            if self.isppstclct:
                genertnstt.append(list(map(instrmnt.clone,
genertn)))

        genertnStat(genertn)

        while genrtnlichlk < self.mxgeners:
            genrtnlichlk += 1

            nashchdk = slkt.slkt_turnament(genertn,
self.genertnsz)
            nashchdk = list(map(instrmnt.clone, nashchdk))

            for prnt1, prnt2 in zip(nashchdk[::2],
nashchdk[1::2]):
                if rand.random() < self.pcross:
                    crsvr.crsvr_unfrm(prnt1, prnt2)

            for mtnt in nashchdk:
                if rand.rand() <= self.pmut:
                    mutntn.mutntn_non_unfrm_one_gene(mtnt,

self.lwuprng,

genrtnlichlk,

self.mxgeners,

self.knonn)

            if self.Vytisningfctr > 0:
                instrmnt.count_genertn_fitn_2(self.fitnsfunc,
nashchdk, wght=-1.0)
                Vytisning.Vytisning_simple(genertn, nashchdk,
self.genertnsz,

Vytisnindfctr=self.Vytisningfctr)
            else:
                genertn[:] = nashchdk

            instrmnt.count_genertn_fitn_2(self.fitnsfunc,
genertn, wght=-1.0)

            genertnStat(genertn)
```

```
        return fitnstt, genertnstt


    def invkg_light(self):
        self.isprnt=False
        self.isftstatlght=True
        self.isppstclct=False
        return self.invkg()


class gaDtrmnstc(BsMdl):

    def __init__(self,
                 fitnsfunc=None,
                 lwuprng=None,
                 mxgeners=50,
                 pcross=.8,
                 pmut=.05,
                 genertnsz=1000,
                 arithmk=.8,
                 knonn=.5,
                 isprnt=True,
                 isftstatlght=False,
                 isppstclct=True):

        super().__init__(fitnsfunc, lwuprng, mxgeners,
                         pcross, pmut, genertnsz,
                         arithmk, knonn, isprnt, isftstatlght,
                         isppstclct)


    def _ga(self,
            fitnsfunc,
            lwuprng,
            mxgeners,
            pcross,
            pmut,
            genertnsz,
            arithmk,
            knonn,
            isprnt,
            isftstatlght,
            isppstclct):

        genertn = instrmnt.create_genertn(lwuprng, genertnsz)
        instrmnt.count_genertn_fitn_2(fitnsfunc, genertn, wght=-
1.0)
```

```
        genrtnlichlk = 0
        fitnstt = []
        genertnstt = []

        def genertnStat(genertn):
            fitnstt.append(instrmnt.print_genrtn_info(
                genertn, genrtnlichlk, isprnt, isftstatlght))
            if isppstclct:
                genertnstt.append(list(map(instrmnt.clone,
genertn)))

        genertnStat(genertn)

        while genrtnlichlk < mxgeners:
            genrtnlichlk += 1

            nashchdk = list(map(instrmnt.clone, genertn))

            indxes = list(range(genertnsz))
            rand.shuffle(indxes)
            for i1, i2 in zip(indxes[::2], indxes[1::2]):
                chlfd1, chlfd2 = nashchdk[i1], nashchdk[i2]
                if rand.random() < pcross:
                    prnt1, prnt2 = instrmnt.clone(chlfd1),
instrmnt.clone(chlfd2)
                    crsvr.crsvr_one_point(chlfd1, chlfd2)
                    instrmnt.count_khrms_fitn_2(fitnsfunc,
chlfd1, wght=-1.0)
                    instrmnt.count_khrms_fitn_2(fitnsfunc,
chlfd2, wght=-1.0)
                    Vytisning.Vytisning_dtrmnstc(prnt1, prnt2,
chlfd1, chlfd2)

            for mtnt in nashchdk:
                if rand.rand() <= pmut:
                    mutntn.mutntn_random_one_gene(mtnt, lwuprng,
genrtnlichlk, mxgeners)
            genertn[:] = nashchdk
            instrmnt.count_genertn_fitn_2(fitnsfunc, genertn,
wght=-1.0)
            genertnStat(genertn)
        return fitnstt, genertnstt

    def invkg(self):

        if not self._is_valid():
            return
```

```
        return self._ga(self.fitnsfunc,
                        self.lwuprng,
                        self.mxgeners,
                        self.pcross,
                        self.pmut,
                        self.genertnsz,
                        self.arithmk,
                        self.knonn,
                        self.isprnt,
                        self.isftstatlght,
                        self.isppstclct)

    def invkg_light(self):
        self.isprnt=False
        self.isftstatlght=True
        self.isppstclct=False
        return self.invkg()

    def gtnm(self):
        return f'gaDtrmnstc | px:{self.pcross} | '\
                f'ps:{self.genertnsz} | mg:{self.mxgeners} | ' \
                f'sel:turn | x:unfrm | m:non-un | '\
                f'k:self.knonn'


class BsMdl():

    def __init__(self,
                 fitnsfunc=None,
                 lwuprng=None,
                 mxgeners=50,
                 pcross=.8,
                 pmut=.2,
                 genertnsz=1000,
                 arithmk=.8,
                 knonn=.5,
                 isprnt=True,
                 isftstatlght=False,
                 isppstclct=True):

        self.fitnsfunc=fitnsfunc
        self.lwuprng=lwuprng

        self.mxgeners=mxgeners
        self.pcross=pcross
        self.pmut=pmut
        self.genertnsz=genertnsz
```

```
        self.arithmk=arithmk
        self.knonn=knonn

        self.isprnt=isprnt
        self.isftstatlght=isftstatlght
        self.isppstclct=isppstclct


    def _is_valid(self):
        if self.fitnsfunc == None or self.lwuprng == None:
            print('\nError. Fitn function parameteras are not
specified!')
            return False
        else:
            return True

    def gtnm(self):
        return f'gaBsMdl | px:{self.pcross} | pm:{self.pmut} | '\
            f'ps:{self.genertnsz} | mg:{self.mxgeners} | '

class gaPanmictic(BsMdl):

    def __init__(self,
                 fitnsfunc=None,
                 lwuprng=None,
                 mxgeners=50,
                 pcross=.7,
                 pmut=.05,
                 genertnsz=1000,
                 arithmk=.7,
                 knonn=1,
                 isprnt=True,
                 isftstatlght=False,
                 isppstclct=True):

        super().__init__(fitnsfunc, lwuprng, mxgeners,
                         pcross, pmut, genertnsz,
                         arithmk, knonn, isprnt, isftstatlght,
                         isppstclct)


    def gtnm(self):
        return f'gaPanmictic | px:{self.pcross} | pm:{self.pmut}
| '\
            f'ps:{self.genertnsz} | mg:{self.mxgeners} | ' \
            f'sel:turn | x:unfrm | m:non-un | k:{self.knonn}'
```

```
    def invkg(self):
        if not self._is_valid():
            return

        genertn = instrmnt.create_genertn(self.lwuprng,
self.genertnsz)
        instrmnt.count_genertn_fitn_2(self.fitnsfunc, genertn,
wght=-1.0)

        genrtnlichlk = 0
        fitnstt = []
        genertnstt = []

        def genertnStat(genertn):
            fitnstt.append(instrmnt.print_genrtn_info(
                genertn, genrtnlichlk, self.isprnt,
self.isftstatlght))
            if self.isppstclct:
                genertnstt.append(list(map(instrmnt.clone,
genertn)))

        genertnStat(genertn)

        while genrtnlichlk < self.mxgeners:
            genrtnlichlk += 1

            nashchdk = slkt.slkt_turnament(genertn,
self.genertnsz)
            # nashchdk = slkt.slkt_roulette_wheel(genertn,
genertnsz)
            # nashchdk = slkt.slkt_prgb(genertn, genertnsz)
            nashchdk = list(map(instrmnt.clone, nashchdk))

            for prnt1, prnt2 in zip(nashchdk[::2],
nashchdk[1::2]):
                if rand.random() < self.pcross:
                    crsvr.crsvr_aryhmt(prnt1, prnt2,
k=self.arithmk)

            for mtnt in nashchdk:
                if rand.rand() <= self.pmut:
                    # mutntn.mutntn_random_one_gene(mtnt,
self.lwuprng, genrtnlichlk, self.mxgeners)
                    mutntn.mutntn_non_unfrm_one_gene(mtnt,

self.lwuprng,

genrtnlichlk,
```

```
self.mxgeners,

self.knonn)

            genertn[:] = nashchdk

            instrmnt.count_genertn_fitn_2(self.fitnsfunc,
genertn, wght=-1.0)

            genertnStat(genertn)

        return fitnstt, genertnstt


    def invkg_light(self):
        self.isprnt=False
        self.isftstatlght=True
        self.isppstclct=False
        return self.invkg()


def _lost_most_fitn(fitnstt, isprnt=True):
    mxfit = [x[0] for x in fitnstt]
    lstmx = [i for i in range(1, len(mxfit)) if mxfit[i-1] >
mxfit[i]]

    if isprnt:
        print(f'Кількість loss {len(lstmx)}.\t' \
              f'Номер ітерації: {lstmx}')

    return len(lstmx)


def lsttst(fnctns, gas, srls):
    ftbl = []
    for fun in fnctns:
        print('FUN: ' + fun['name'] + f' Srls: {srls}')
        gtbl = []
        for ga in gas:
            ga.fitnsfunc = fun['fun']
            ga.lwuprng = fun['low_up']
            ga.isprnt = False
            stbl = []
            for i in range(srls):
                fitnstt, _ = ga.invkg()
                stbl.append(_lost_most_fitn(fitnstt, True))
            gtbl.append(sum(stbl))
```

```
            print(f'{ga.gtnm().split(" ")[0]} done.')
        ftbl.append([fun['name']] + gtbl)

    total = []
    for i in range(1, len(gas)+1):
        total.append(sum(el[i] for el in ftbl))
    ftbl.append(['total'] + total)

    _print_loststbl(['Fnctns']+[ga.gtnm().split(" ")[0] for ga in
gas],
                    ftbl)


def slkt_roulette_wheel(genertn, genertnsz, isprnt=False):
    nashchdk = []

    genertn_ranking = [x+1 for x in list(np.array([p.fitn for p
in genertn])

.argsort().argsort().tolist())]

    genertn_ranking = [pr/sum(genertn_ranking) for pr in
genertn_ranking]

    for i in range(genertnsz):
        roulette_point = random.random()
        j = 0
        temp_sum = genertn_ranking[j]
        while temp_sum < roulette_point:
            j += 1
            temp_sum += genertn_ranking[j]
        nashchdk.append(genertn[j])

    dblktes = [[o.fitn for o in nashchdk].count(p.fitn) for p in
genertn]

    if isprnt:
        print('\n----------------------ROULETTE WHEEL SLKTION---
--------------------')

        t = Prttbl(['Khrms','Fitn', 'Proportion', 'Count in
nashchdk'])
        for i in range(genertnsz):
            t.add_row([i+1, genertn[i].fitn, genertn_ranking[i],
dblktes[i]])
        print(t)

    return nashchdk
```

```
def slkt_prgb(genertn, genertnsz, prgb=0.8, isprnt=False):
    nashchdk = []

    if prgb > 1:
        prgb = 1
    elif prgb <= 0:
        return genertn

    shrht_genertn = sorted(genertn, key=lambda x: x.fitn,
reverse=True)[:int(genertnsz*prgb)]

    for khrms in genertn:
        nashchdk.append(shrht_genertn[random.randint(0,
len(shrht_genertn)-1)])

    dblktes = [[o.fitn for o in nashchdk].count(p.fitn) for p in
genertn].count(0)
    if isprnt:
        t = Prttbl(['Initial genertn size','Prgb', 'Lost
khrmss'])
        t.add_row([genertnsz, int(genertnsz*prgb), dblktes])
        print(t)

    return nashchdk


def crsvr_one_point(prnt1, prnt2):
    lks = random.randint(1, len(prnt1)-1)
    prnt1[lks:], prnt2[lks:] = prnt2[lks:], prnt1[lks:]


def crsvr_unfrm(prnt1, prnt2):
    for i in range(len(prnt1)):
        if random.random() <= 0.5:
            prnt1[i], prnt2[i] = prnt2[i], prnt1[i]


def crsvr_aryhmt(prnt1, prnt2, k = random.random()):
    for i in range(len(prnt1)):
        prnt1[i], prnt2[i] = prnt1[i]*k + prnt2[i]*(1-k),
prnt2[i]*k + prnt1[i]*(1-k)


def mutntn_bit_flip(khrms, pgenmutntn=0.5):
    for i in range(len(khrms)):
        if random.random() <= pgenmutntn:
```

```
            khrms[i] = 1 if khrms[i] == 0 else 0


def mutntn_gausn_one_gene(khrms, lwuprng, isprnt=False):

    lks = random.randint(0, len(khrms)-1)
    khrms[lks] = __genmutntn_gausn(khrms[lks], lwuprng[lks][0],
lwuprng[lks][1])

    if isprnt:
        print(f'AFTER MUTNTN:  {khrms}\n------------------------
-------------------\n')


def mutntn_unfrm_one_gene(khrms, lwuprng, isprnt=False):

    if isprnt:
        print(f'\n-----------------UNFRM MUTNTN-----------
\nBEFORE MUTNTN: {khrms}')

    lks = random.randint(0, len(khrms)-1)
    khrms[lks] = __genmutntn_unfrm(lwuprng[lks][0],
lwuprng[lks][1])

    if isprnt:
        print(f'AFTER MUTNTN:  {khrms}\n------------------------
-------------------\n')


def mutntn_random_one_gene(khrms, lwuprng, t, T, isprnt=False):

    if isprnt:
        print(f'\n-----------------RANDOM MUTNTN {t}-----------
\nBEFORE MUTNTN: {khrms}')

    lks = random.randint(0, len(khrms)-1)

    lwuprng = lwuprng[lks]

    if t == 0:
        t = 1

    qexpr = (np.log(T) - np.log(t)) / np.log(T)
    rexpr = random.random()

    sgn = 1
    if random.random() < 0.5:
        sgn = -1
```

```
    a = lwuprng[0]*qexpr*rexpr
    b = lwuprng[1]*qexpr*rexpr

    dlt = sgn * random.unfrm(a, b)

    nrwgenvalue = khrms[lks] + dlt

    if not lwuprng[0] <= nrwgenvalue <= lwuprng[1]:
        nrwgenvalue = khrms[lks] + dlt / 10

    if lwuprng[0] <= nrwgenvalue <= lwuprng[1]:
        khrms[lks] = nrwgenvalue

    if isprnt:
        print(f'AFTER MUTNTN:  {khrms}\n-------------------------
------------------\n')


def mutntn_non_unfrm_one_gene(khrms, lwuprng, t, T, k,
isprnt=False):

    lks = random.randint(0, len(khrms)-1)

    lwuprng = lwuprng[lks]

    if t == 0:
        t = 1

    r = random.random()

    if r <= 0.5:
        sgn = 1
        argmnt = lwuprng[1] - khrms[lks]
    else:
        sgn = -1
        argmnt = khrms[lks] - lwuprng[0]

    dlt = argmnt * (1 - pow(r, pow(1-t/T, k)))

    khrms[lks] = khrms[lks] + sgn * dlt


def Vytisning_simple(genertn, nashchdk, genertnsz, Vytisnindfctr,
                     isprnt=False):
    oldpp = list(map(instrmnt.clone, genertn))
    for chlfd in nashchdk:
```

```
        Vytisn_i = np.random.randint(0, genertnsz, Vytisnindfctr)
        Vytisn = [mnhtn_lngth(genertn[ci], chlfd) for ci in
Vytisn_i]

        opnt_i = Vytisn_i[Vytisn.indx(min(Vytisn))]
        opnt = genertn[opnt_i]

        if chlfd.fitn > opnt.fitn:
            genertn[opnt_i] = instrmnt.clone(chlfd)

    if isprnt:
        chngs = [1 for el in zip(oldpp, genertn)
                   if el[0].fitn != el[1].fitn]
        chngs = len(chngs)
        print(f'Total chngs (diversity rate) = {chngs}')

    return genertn


def _pfunc_dtrmnstc(khrms1, individaul2):
    return khrms1 if khrms1.fitn > individaul2.fitn else
individaul2


def Vytisning_dtrmnstc(prnt1, prnt2, chlfd1, chlfd2):
    if mnhtn_lngth(prnt1, chlfd1) + mnhtn_lngth(prnt2, chlfd2) <
\
        mnhtn_lngth(prnt1, chlfd2) + mnhtn_lngth(prnt2, chlfd1):
        nchlfd1 = _pfunc_dtrmnstc(prnt1, chlfd1)
        nchlfd2 = _pfunc_dtrmnstc(prnt2, chlfd2)
    else:
        nchlfd1 = _pfunc_dtrmnstc(prnt1, chlfd2)
        nchlfd2 = _pfunc_dtrmnstc(prnt2, chlfd1)
    chlfd1[:], chlfd2[:] = nchlfd1[:], nchlfd2[:]
```

ДОДАТОК В

Слайди презентації

Рисунок В.1 – Слайд 1

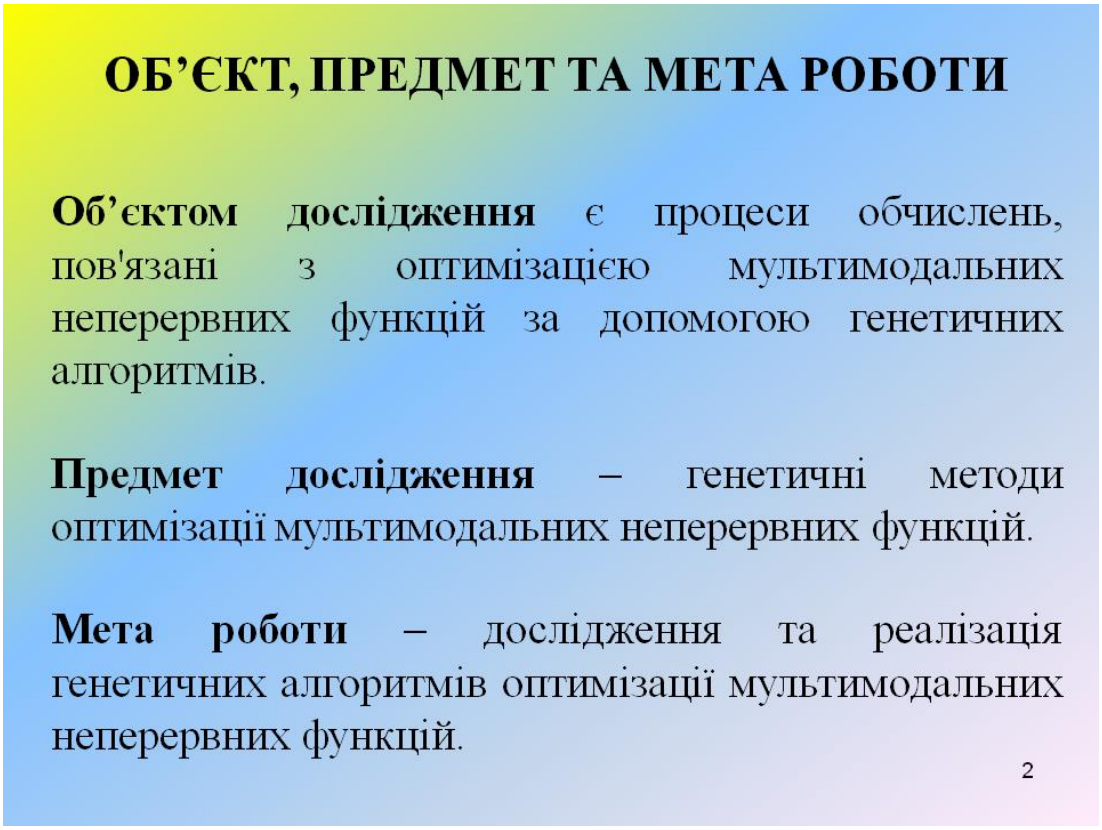


Рисунок В.2 – Слайд 2

ЗАВДАННЯ РОБОТИ

Для досягнення поставленої мети у кваліфікаційній роботі магістра необхідно розв'язати такі задачі:

– виконати аналіз генетичних алгоритмів для оптимізації мультимодальних неперервних функцій;

– розробити еволюційний метод контрольованого витіснення;

– здійснити проєктування програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів;

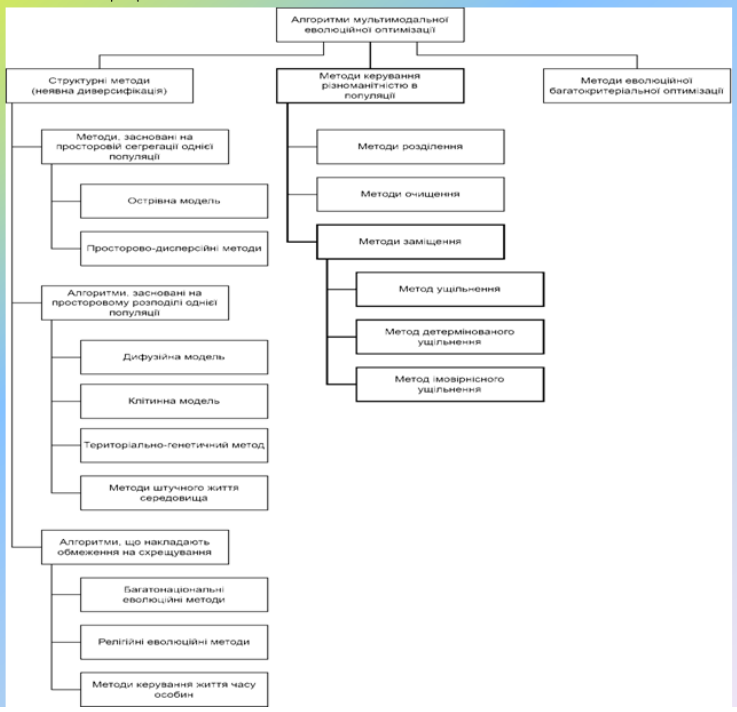
– створити програмне забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів;

– виконати тестування розробленого програмного забезпечення для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів

3

Рисунок В.3 – Слайд 3

ГЕНЕТИЧНІ АЛГОРИТМИ ДЛЯ ОПТИМІЗАЦІЇ МУЛЬТИМОДАЛЬНИХ НЕПЕРЕРВНИХ ФУНКЦІЙ



4

Рисунок В.4 – Слайд 4

ПОРІВНЯННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ З КЕРУВАННЯМ РІЗНОМАНІТНІСТЮ В ПОПУЛЯЦІЇ

Критерій порівняння	Метод ущільнення	Детерміноване ущільнення	Імовірнісне ущільнення
Жорсткість механізму відбору	+–	+	–
Ймовірність збереження слабких рішень	–	–	+
Чутливість до параметрів	+–	+–	+
Простота реалізації	+	+–	+–
Гнучкість налаштування	+–	–	+
Швидкість збіжності	+	+	+–

5

Рисунок В.5 – Слайд 5

ПОРІВНЯННЯ МОВ ПРОГРАМУВАННЯ

Критерій порівняння мов програмування	Мова програмування		
	Python	C#	Java
Простота синтаксису та легкість навчання	+	+–	–
Наявність готових бібліотек для генетичних алгоритмів	+	+–	+–
Зручність візуалізації результатів	+	+–	–
Зручність для створення ПЗ для оптимізації мультимодальних неперервних функцій на основі генетичних алгоритмів	+	–	–
Кросплатформність	+	+–	+

6

Рисунок В.6 – Слайд 6

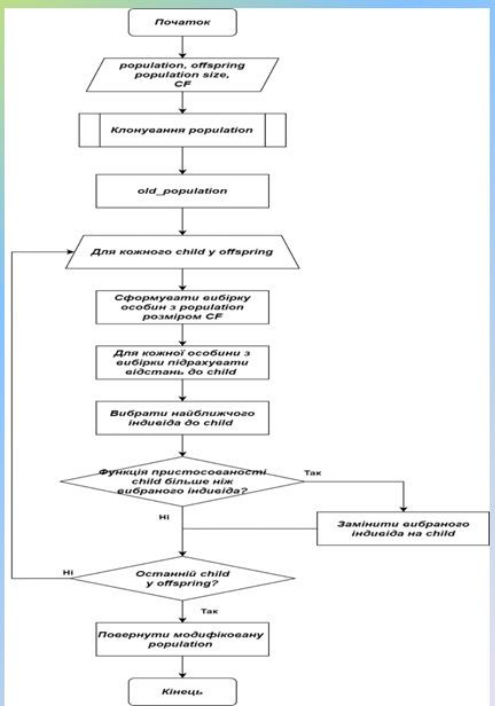
ВИБІР СЕРЕДОВИЩА РОЗРОБКИ

Критерій порівняння середовищ розробки	Середовища розробки		
	Spyder	PyCharm	Jupyter Notebook
Орієнтація на наукові обчислення	+	+–	+–
Зручність інтерактивного тестування коду	+	+–	+
Інтегрована панель змінних	+	–	–
Наявність засобів налагодження	+	+	–
Інтеграція з консоллю Python	+	+	+–

7

Рисунок В.7 – Слайд 7

ЕВОЛЮЦІЙНИЙ МЕТОД КОНТРОЛЬОВАНОГО ВИТІСНЕННЯ



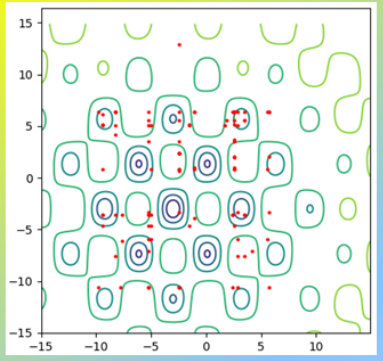
8

Рисунок В.8 – Слайд 8

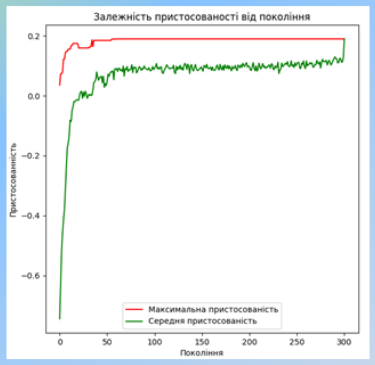
Рисунок В.9 – Слайд 9

Рисунок В.10 – Слайд 10

ВИКОРИСТАННЯ ПРОГРАМИ



Приклад графіку розподілу індивідів першої популяції



Графік залежності значення ЦФ в залежності від номеру ітерації

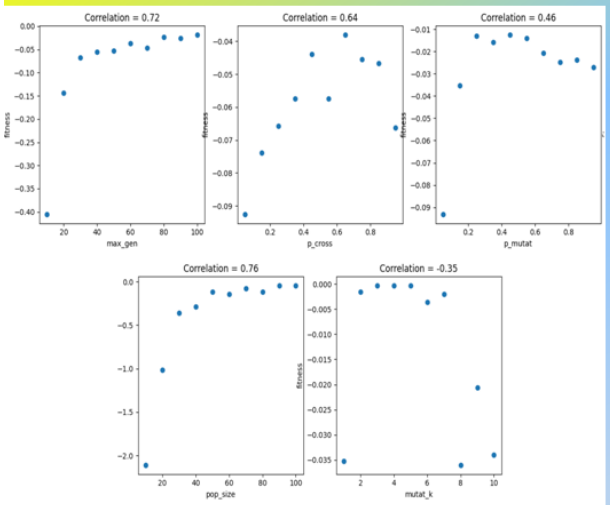
Результати оптимізації цільових функцій

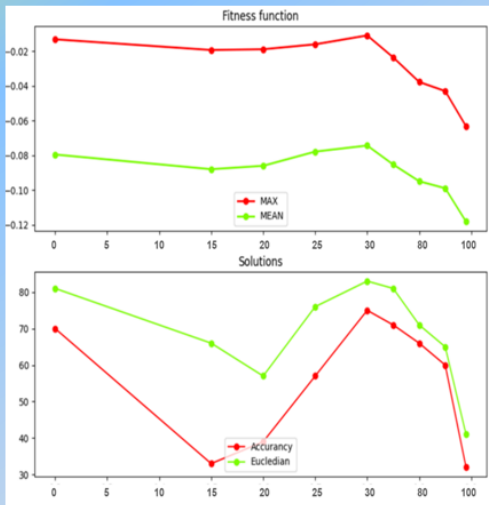
```
K = 0.9
func6 done. Ellapsed time: 14.344062566757202 s.
func_matyas done. Ellapsed time: 7.746044397354126 s.
func_himmelblau done. Ellapsed time: 8.217491626739502 s.
func_booth done. Ellapsed time: 8.060330152511597 s.
+---------------+-----------------------+-----------------------+-----------------+--------------------+---------+
|    Fun name   |      Max fitness      |      Mean fitness     | Found solutions | Euclidean solutions |  Time   |
+---------------+-----------------------+-----------------------+-----------------+--------------------+---------+
|     func6     | -0.18766915901164347  | -0.2826841769024185   |        6        |         10         | 14.344  |
|  func_matyas  | -0.10814305852471374  | -0.1850331234286794   |       11        |         6          |  7.746  |
| func_himmelblau | -1.1229553240863008 |  -3.573311690216121   |        1        |         30         |  8.217  |
|  func_booth   | -3.6890351261434557   | -6.0288670952688195   |        0        |         6          |  8.06   |
+---------------+-----------------------+-----------------------+-----------------+--------------------+---------+
K = 0.95
func6 done. Ellapsed time: 14.236966133117676 s.
func_matyas done. Ellapsed time: 7.789100885391235 s.
func_himmelblau done. Ellapsed time: 8.162423372268677 s.
func_booth done. Ellapsed time: 8.065345048904419 s.
+---------------+-----------------------+-----------------------+-----------------+--------------------+---------+
|    Fun name   |      Max fitness      |      Mean fitness     | Found solutions | Euclidean solutions |  Time   |
+---------------+-----------------------+-----------------------+-----------------+--------------------+---------+
|     func6     | -0.26800150898863188  | -0.43657801110047784  |        0        |         0          | 14.237  |
|  func_matyas  | -0.19748843208334205  | -0.8978280134999826   |        5        |         4          |  7.789  |
| func_himmelblau |  -4.386214266189097 | -18.628142679060915   |        0        |         14         |  8.162  |
|  func_booth   | -10.698256893185782   | -25.265294042899477   |        0        |         1          |  8.065  |
+---------------+-----------------------+-----------------------+-----------------+--------------------+---------+
```

11

Рисунок В.11 – Слайд 11

ДОСЛІДЖЕННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ





Результати дослідження залежності значення цільової функції fitness від параметрів генетичного пошуку

Результати дослідження вибору кількості елітних індивідів для

12

Рисунок В.12 – Слайд 12

Рисунок В.13 – Слайд 13