Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка інтерактивної експериментальної платформи для досліджень властивостей ЕМ-алгоритму з використанням бібліотеки scikit-learn»

Виконав: студент 4 курсу
групи 4ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Фоменко Д. С.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д. І.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ЗІ Дудатьєв А. В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Фоменку Денису Сергійовичу

1. Тема роботи – розробка інтерактивної експериментальної платформи для досліджень властивостей ЕМ-алгоритму з використанням бібліотеки scikit-learn.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: методи кластеризації – ЕМ-алгоритм, K-Means; набори даних – Gaussian Blobs, Moons, Circles, CSV-файли; параметри алгоритму – комплексна метрика якості; розмірність даних – від 2 до 12 ознак; методи візуалізації – Matplotlib; програмне середовище – Python, PyQt5; середовище розробки – Visual Studio Code; вихідні результати – кластеризовані дані з візуалізацією еліпсів коваріації, значення метрики якості; вхідні дані: вибір типу набору даних, налаштування параметрів кластеризації, вибір метрик якості.

4. Зміст текстової частини: вступ; аналіз стану систем автоматизації та кластеризації даних; порівняльний аналіз аналогів; аналіз методів розв'язання задачі; постановка задач дослідження; структурний аналіз ЕМ-алгоритму для модульної реалізації; моделювання архітектури системи; розробка підсистеми генерації синтетичних даних; розробка підсистеми інтерактивної візуалізації ЕМ-алгоритму; розробка графічної схеми інтерфейсу; аналіз і обґрунтування вибору засобів для реалізації системи; розробка модуля оцінки якості кластеризації; розробка модуля експорту результатів кластеризації в PDF; функціональне тестування системи; розробка інструкції користувача; висновки; список використаних джерел; додатки.

5. Перелік ілюстративної частини: зображення аналогів; блок-схеми алгоритмів роботи застосунку; схема інтерфейсу; графічний інтерфейс застосунку; тестування застосунку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Катєльніков Д.І. к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану систем автоматизації та кластеризації даних	25.03.25 – 07.04.25	вик
2	Розробка підсистеми генерації синтетичних даних	08.04.25 – 20.04.25	вик
3	Розробка підсистеми інтерактивної візуалізації ЕМ-алгоритму	21.04.25 – 03.05.25	вик
4	Розробка модуля оцінки якості кластеризації	04.04.25 – 15.05.025	вик
5	Тестування роботи застосунку	16.05.25 – 24.05.25	вик
6	Оформлення матеріалів до захисту БКР	25.05.25 – 30.05.25	вик

Студент _____ **Д. С. Фоменко**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **Д. І. Катєльніков**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.42:519.24

Фоменко Д. С. Розробка інтерактивної експериментальної платформи для досліджень властивостей ЕМ-алгоритму з використанням бібліотеки scikit-learn: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025 – 66 с.

Бакалаврська кваліфікаційна робота складається з 66 сторінок формату А4, 26 рисунків, 2 таблиць, 4 додатків. Список використаних джерел містить 18 найменувань.

У бакалаврській кваліфікаційній роботі було розроблено методи та інтерактивну платформу для дослідження ЕМ-алгоритму, що забезпечує наочну візуалізацію кластеризації. Проведено аналіз сучасних методів візуалізації алгоритмів машинного навчання. Обґрунтовано необхідність створення інструменту з динамічним відображенням ітерацій.

Розроблено структуру застосунку, що дозволяє модульно реалізовувати візуалізацію даних різної вимірності. Розроблено алгоритми генерації синтетичних наборів даних та обробки CSV-файлів із числовими ознаками. Створено модуль обчислення комплексної метрики якості. Реалізовано графічний інтерфейс на основі PyQt5.

Проведено модульне тестування платформи, яке підтвердило коректність роботи модулів генерації даних, кластеризації та візуалізації. Розроблено алгоритм динамічного відображення еліпсів коваріації та ймовірностей приналежності до кластерів. Створено інструкцію користувача, яка детально описує налаштування платформи та інтерпретацію результатів.

Ключові слова: машинне навчання, ЕМ-алгоритм, кластеризація, інтерактивна візуалізація, scikit-learn, експериментальна платформа.

ABSTRACT

UDC 004.42:519.24

D. S. Fomenko. Development of an interactive experimental platform for studying properties of the EM-algorithm using the scikit-learn library: bachelor's qualification thesis in specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025 – 66 p.

The bachelor's qualification thesis consists of 66 pages in A4 format, 26 figures, 2 tables, 4 appendices. The list of references contains 18 items.

In the bachelor's qualification thesis, methods and an interactive platform for studying the EM-algorithm were developed, providing visualization of clustering. An analysis of modern methods for visualizing machine learning algorithms was conducted. The necessity of creating a tool with dynamic display of iterations was substantiated.

The application structure was developed, allowing modular implementation of data visualization of various dimensions. Algorithms for generating synthetic datasets and processing CSV files with numerical features were developed. A module for calculating complex quality metrics was created. A graphical interface based on PyQt5 was implemented.

Modular testing of the platform was conducted, which confirmed the correctness of the data generation, clustering, and visualization modules. An algorithm for dynamic display of covariance ellipses and cluster membership probabilities was developed. A user manual was created that describes in detail the platform configuration and interpretation of results.

Keywords: machine learning, EM-algorithm, clustering, interactive visualization, scikit-learn, experimental platform.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний етап розвитку методів машинного навчання [1] характеризується стрімким розвитком алгоритмів кластеризації та класифікації даних [2]. Алгоритм максимізації математичного сподівання відзначається особливою результативністю при роботі з неповними даними та прихованими змінними. Зростаюча складність даних вимагає розробки спеціалізованих інструментів для вивчення властивостей алгоритмів та візуалізації їх роботи. Розвиток інтерактивних методів візуалізації створює нові можливості для дослідження математичних основ алгоритмів, що важливо для освітніх та дослідницьких цілей.

EM-алгоритм [3] є одним із найскладніших для розуміння методів машинного навчання, оскільки включає ітеративні кроки оцінки та максимізації, що вимагає глибокого розуміння статистичних концепцій. Для забезпечення наочності його роботи використовують динамічні інтерактивні компоненти, що дозволяє відобразити алгоритмічні особливості методу. Це ускладнює процес розробки, але підвищує точність навчання та дослідження.

У системах машинного навчання цінними є методи, що дозволяють оптимально використовувати обчислювальні ресурси для швидкого аналізу даних. Бібліотека scikit-learn пропонує широкий спектр реалізацій алгоритмів, включаючи EM-алгоритм, та дозволяє працювати з різноманітними наборами даних. Процедури кластеризації та класифікації успішно застосовують для сегментації зображень, розпізнавання образів та обробки природної мови.

Існуючі методи візуалізації та експериментального дослідження EM-алгоритму характеризуються недостатньою інтерактивністю та обмеженими можливостями для глибокого аналізу. Вони мають обмежені можливості інтерактивної зміни параметрів та недостатню наочність відображення проміжних кроків роботи.

Підвищення інтерактивності вивчення алгоритму можна досягти використанням інтерактивних компонентів, що дасть можливість точнішого

розуміння впливу параметрів на результат роботи. При розробці освітніх платформ часто реалізують окремо різні компоненти: генерацію даних, налаштування параметрів, візуалізацію результатів. Поєднання цих компонентів нерідко призводить до ускладнення інтерфейсу або до зниження продуктивності через дублювання обчислень.

Створення інтерактивних освітніх та дослідницьких інструментів висуває високі вимоги до інтерактивності та наочності представлення інформації, оскільки використання простих статичних візуалізацій неприйнятне для глибокого розуміння складних алгоритмів машинного навчання.

Тому актуальними є питання розробки власної інтерактивної експериментальної платформи для візуалізації ЕМ-алгоритму, оскільки існуючі інструменти не задовільняють потреби багатьох освітніх та дослідницьких задач.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення інтерактивності та наочності при вивченні та дослідженні властивостей ЕМ-алгоритму за рахунок розробки інтерактивної експериментальної платформи з використанням бібліотеки scikit-learn. Це дозволить полегшити сприйняття інформації при вивченні ЕМ-алгоритму і дослідженні його особливостей.

Основними задачами дослідження є:

– провести аналіз існуючих методів і засобів візуалізації та експериментального дослідження ЕМ-алгоритму для визначення напрямків підвищення їх інтерактивності та наочності;

– запропонувати нові:

– методи підвищення інтерактивності візуалізації процесу роботи ЕМ-алгоритму;

– методи підвищення наочності та інформативності представлення результатів роботи ЕМ-алгоритму;

– розробити програмні компоненти та інтерактивну експериментальну

платформу на основі запропонованих методів;

– провести експериментальні дослідження розроблених засобів візуалізації та аналізу ЕМ-алгоритму.

Об'єкт дослідження – процес вивчення ЕМ-алгоритму і дослідження його властивостей.

Предмет дослідження – методи та засоби інтерактивної візуалізації та дослідження властивостей ЕМ-алгоритму.

Методи дослідження. В процесі досліджень використовувались: теорія ймовірностей та математична статистика, лінійна алгебра, методи машинного навчання, алгоритми оптимізації, комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод інтерактивної візуалізації ЕМ-алгоритму, який, на відміну від існуючих рішень, забезпечує динамічне відображення проміжних результатів кожної ітерації з можливістю зміни параметрів. Це дозволяє розглядати більше можливих варіантів без потреби повторного запуску алгоритму.

2. Подальшого розвитку отримав метод оцінки якості кластеризації, який, на відміну від існуючих рішень, передбачає використання комбінації зовнішніх та внутрішніх метрик з їх динамічною візуалізацією, що дозволяє різнобічно аналізувати якість кластеризації.

3. Подальшого розвитку отримав метод генерації синтетичних даних для тестування ЕМ-алгоритму, у якому, на відміну від існуючих, використано параметричні моделі з можливістю інтерактивного налаштування, що дозволило розширити спектр експериментальних сценаріїв.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмну платформу для інтерактивного дослідження властивостей ЕМ-алгоритму.

Особистий внесок здобувача. Усі наукові результати, викладені у БКР, отримані автором особисто.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Описані у дослідженні положення доповідались на конференції «LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (2025)» та опубліковані в тезах доповіді.

Публікації. За тематикою дослідження опубліковано 1 наукову працю, що доповідалась на конференції «LIII Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2025)» та опубліковано в тезах доповіді [4].

Структура та обсяг БКР. Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків, у які винесено технічне завдання, протокол перевірки на плагіат, лістинг програмного забезпечення й ілюстративний матеріал до захисту роботи.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ

1.1 Аналіз стану систем автоматизації та кластеризації даних

У сучасних умовах стрімкого розвитку методів аналізу даних та машинного навчання алгоритми кластеризації набувають першочергового значення для виявлення прихованих структур у даних [5]. Автоматизація процесів кластеризації відіграє фундаментальну роль у розширенні аналітичних можливостей, забезпечуючи точний розподіл даних на групи відповідно до їхніх внутрішніх характеристик. Розробка програмного забезпечення для дослідження властивостей алгоритмів кластеризації є важливим завданням, що дозволяє створити високопродуктивні інструменти для дослідників та оптимізувати процеси аналізу багатовимірних даних.

Впровадження систем для дослідження властивостей EM-алгоритму забезпечує спрощення процесу взаємодії між дослідником та об'єктом дослідження, що суттєво скорочує час на аналіз результатів кластеризації та підвищує точність інтерпретації отриманих даних. Водночас розробка таких систем становить комплексне завдання, що потребує всебічного аналізу вимог користувачів, а також створення надійних і масштабованих програмних застосунків для обробки різноманітних наборів даних різної структури та розмірності.

Значна кількість науково-дослідних установ та освітніх закладів, що спеціалізуються на вивченні методів кластеризації, застосовують сучасні алгоритми, серед яких EM-алгоритм заслуговує на особливу увагу завдяки його гнучкості та продуктивності при роботі з сумішами розподілів [6]. Унаслідок впровадження таких програмних застосунків користувачі отримують можливість поглиблено вивчати принципи функціонування алгоритму та його поведінку на різних наборах даних, а навчальні заклади підвищують рівень підготовки фахівців у галузі інтелектуального аналізу даних.

Використання спеціалізованого програмного забезпечення для дослідження властивостей ЕМ-алгоритму надає можливість інтеграції різноманітних функціональних можливостей, зокрема поітераційного аналізу виконання алгоритму, обчислення та візуалізації кластерних ваг та інших параметрів. Це не лише спрощує процес дослідження, але й суттєво підвищує якість отриманих результатів, забезпечуючи дослідникам оптимальне та продуктивне середовище для роботи.

Отже, розробка програмного забезпечення для дослідження властивостей ЕМ-алгоритму має важливе значення для сучасної науково-освітньої сфери. Інвестування в якісні інформаційно-технологічні застосунки уможливлює створення інтелектуальних інструментів, що адаптуються до потреб дослідників, підвищують швидкість навчального процесу та вдосконалюють загальний досвід взаємодії з алгоритмами машинного навчання. В умовах, коли аналіз даних і штучний інтелект стають основними рушіями інновацій, необхідним є постійний моніторинг новітніх технологій та тенденцій з метою забезпечення актуальності навчальних та дослідницьких інструментів у сфері інтелектуального аналізу даних.

1.2 Порівняльний аналіз аналогів

З розвитком методів машинного навчання та інтелектуального аналізу даних кластеризація зазнає суттєвих змін, особливо в аспекті ітеративних алгоритмів виявлення прихованих структур у даних. Використання спеціалізованого програмного забезпечення дозволяє дослідникам та студентам аналізувати процес формування кластерів, відстежувати зміну параметрів у реальному часі та глибше розуміти математичні основи алгоритмів.

Особливу увагу привертає ЕМ-алгоритм як один із найкращих підходів до кластеризації на основі моделей сумішей. Сучасні програмні інструменти забезпечують можливість візуалізації кожної ітерації, аналізу результатів та експериментування з параметрами, що робить дослідження більш доступним і наочним.

Наукові та освітні установи активно впроваджують такі інструменти, що сприяє покращенню підготовки фахівців у галузі аналізу даних. Серед існуючих інструментів варто відзначити ELKI, WEKA, MATLAB, KNIME – платформи, які пропонують різні підходи до кластеризації та аналізу результатів.

На рисунку 1.1 зображено програмний пакет ELKI [7], який є середовищем для розробки додатків з виявлення знань у базах даних з підтримкою індексних структур. ELKI представляє собою платформу з відкритим кодом, написану на Java, що спеціалізується на алгоритмах кластеризації та виявлення аномалій у даних. Програма має модульну архітектуру, що дозволяє дослідникам використовувати, розширювати та комбінувати різні алгоритми аналізу даних, включаючи базову реалізацію EM-алгоритму.

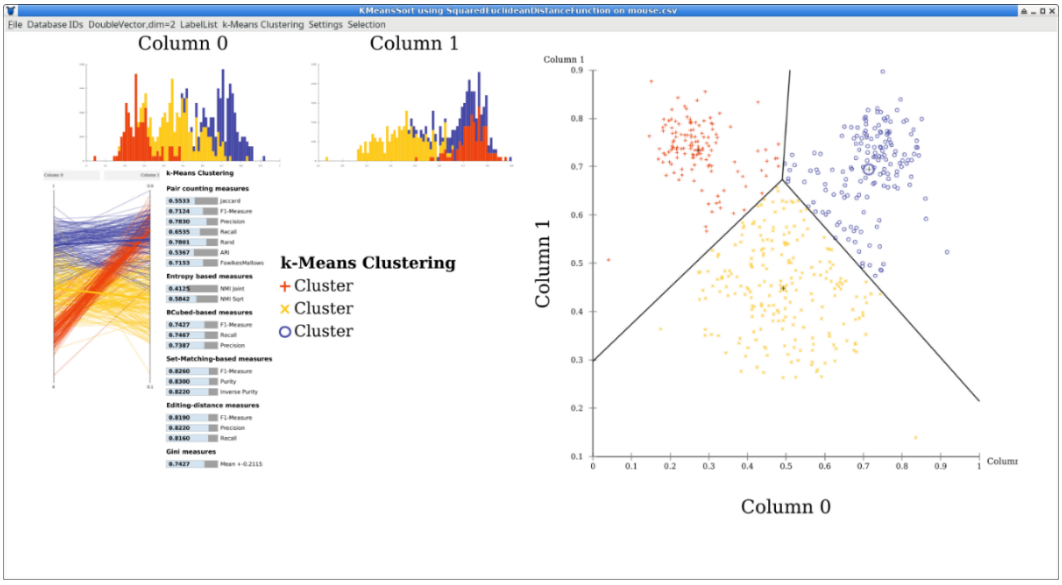


Рисунок 1.1 – Інтерфейс K-means кластерингу в ELKI

ELKI надає користувачам широкий набір інструментів для візуалізації результатів кластеризації у вигляді різноманітних графіків та діаграм, що полегшує інтерпретацію отриманих результатів. Програма містить велику кількість метрик для оцінки якості кластеризації, проте не пропонує спеціалізованих інструментів для дослідження саме EM-алгоритму. ELKI орієнтована переважно на досвідчених користувачів та дослідників у галузі аналізу даних, що обмежує її використання в освітніх цілях для початківців, які

потребують більш інтуїтивного інтерфейсу для вивчення принципів роботи конкретних алгоритмів кластеризації.

На рисунку 1.2 зображено програмне забезпечення WEKA [8], яке є популярним інструментом для машинного навчання та аналізу даних, розробленим в Університеті Вайкато в Новій Зеландії. WEKA надає користувачам графічний інтерфейс для застосування різних алгоритмів машинного навчання, включаючи реалізацію EM-алгоритму в модулі кластеризації. Програма дозволяє завантажувати дані з різних джерел, включаючи формати CSV, ARFF та бази даних, а також виконувати попередню обробку даних перед застосуванням алгоритмів кластеризації.

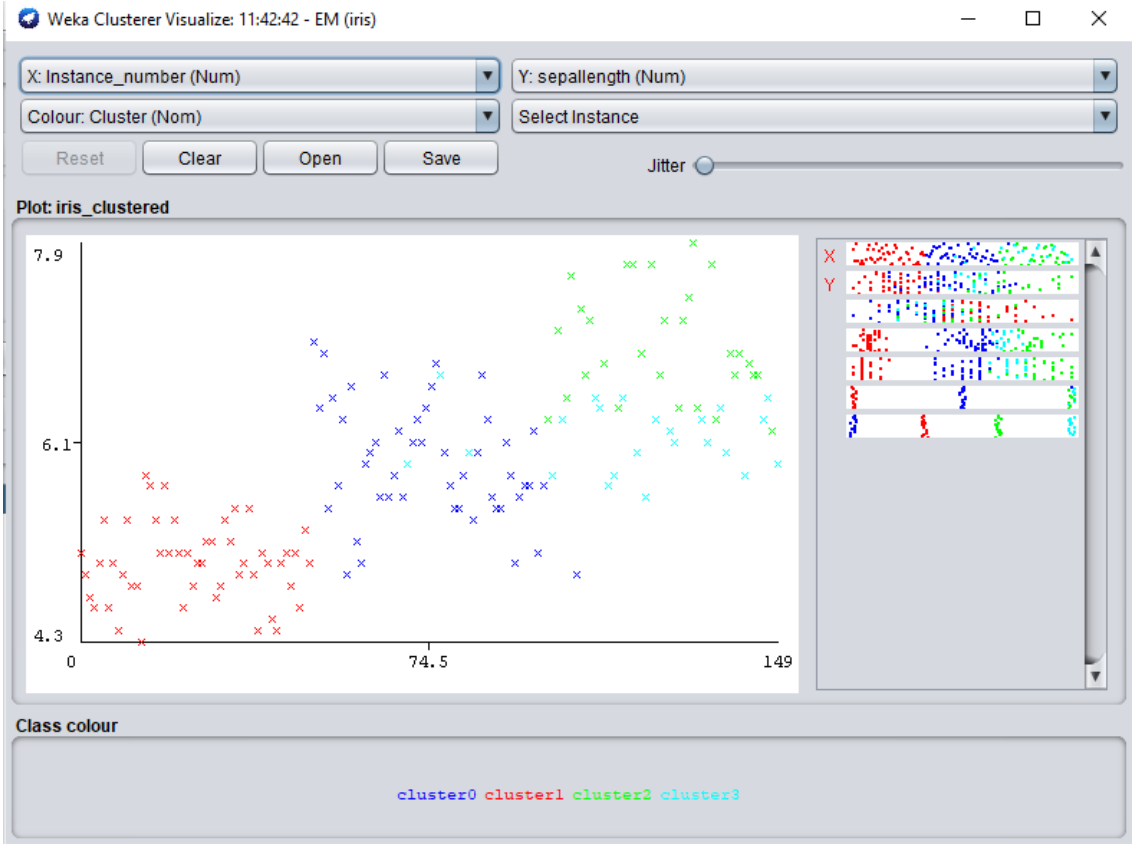


Рисунок 1.2 – Візуалізатор кластерів в Weka

WEKA має достатньо доступний інтерфейс, що робить її популярною у навчальних закладах для ознайомлення студентів з основами аналізу даних та машинного навчання. Програма пропонує базові можливості для налаштування параметрів EM-алгоритму, таких як кількість кластерів та максимальна кількість

ітерацій, проте не дозволяє досліджувати процес виконання алгоритму на рівні окремих ітерацій. WEKA забезпечує візуалізацію результатів кластеризації та можливість експорту результатів для подальшого аналізу, однак не має спеціалізованих функцій для детального дослідження специфічних властивостей EM-алгоритму, таких як відстеження змін Log-Likelihood або аналіз Cluster Weights у процесі виконання.

На рисунку 1.3 зображено середовище MATLAB [9] з пакетом Statistics and Machine Learning Toolbox, яке є потужною комерційною платформою для наукових та інженерних обчислень з розширеними можливостями для статистичного аналізу та машинного навчання. MATLAB пропонує реалізацію EM-алгоритму через функцію fitgmdist, що дозволяє будувати моделі суміші гауссіанів з різними параметрами. Програма забезпечує високу точність обчислень та гнучкість у налаштуванні параметрів алгоритму, а також надає широкі можливості для візуалізації результатів кластеризації.

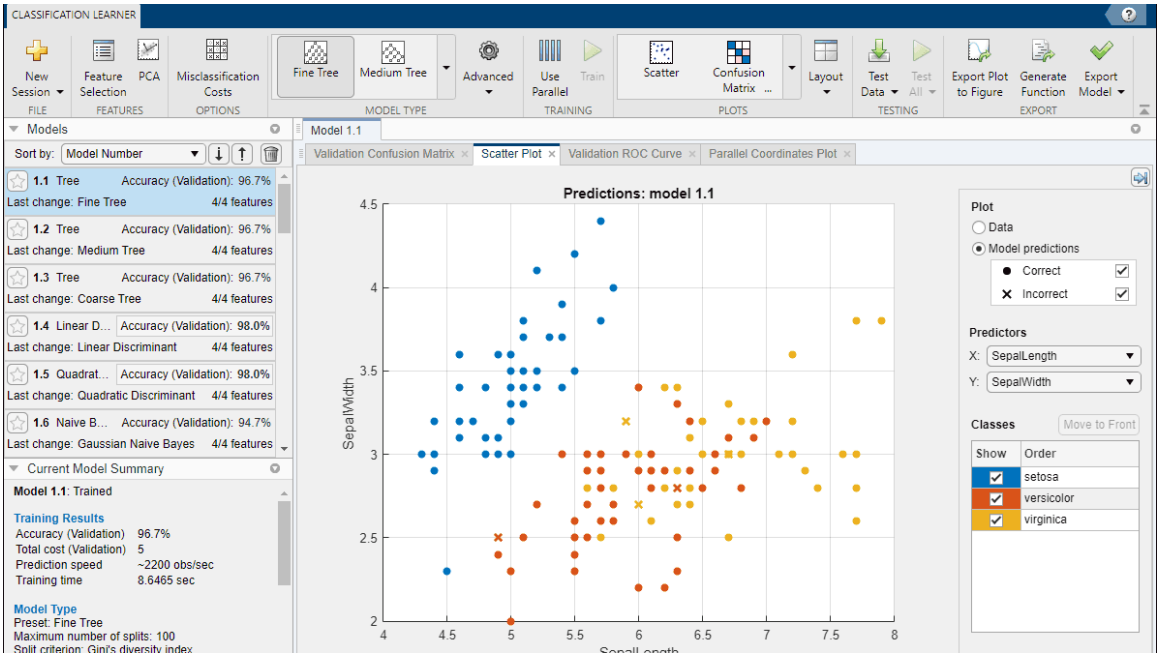


Рисунок 1.3 – Інтерфейс модулю тренування очікуваної моделі в MATLAB

MATLAB дозволяє користувачам програмно керувати процесом кластеризації, що теоретично надає можливість реалізувати поетапне виконання EM-алгоритму, проте такий функціонал не інтегрований у стандартний

інтерфейс і вимагає додаткового програмування на мові MATLAB. Середовище забезпечує розрахунок різних метрик для оцінки якості кластеризації, включаючи Log-Likelihood та BIC, а також дозволяє досліджувати параметри отриманих кластерів. Незважаючи на потужність та гнучкість, MATLAB має високу вартість ліцензії та досить високий поріг входження для початківців, що обмежує його використання в навчальних цілях для широкого кола користувачів.

На рисунку 1.4 зображено програму KNIME Analytics Platform [10], яка є відкритою платформою для аналізу, звітності та інтеграції даних з модульною архітектурою на основі потоків даних. KNIME дозволяє користувачам створювати складні процеси аналізу даних, використовуючи принцип візуального програмування через з'єднання різних функціональних вузлів у єдиний робочий процес. Програма містить вузли для кластеризації, включаючи реалізацію EM-алгоритму, що дозволяє інтегрувати кластерний аналіз у складні аналітичні сценарії.

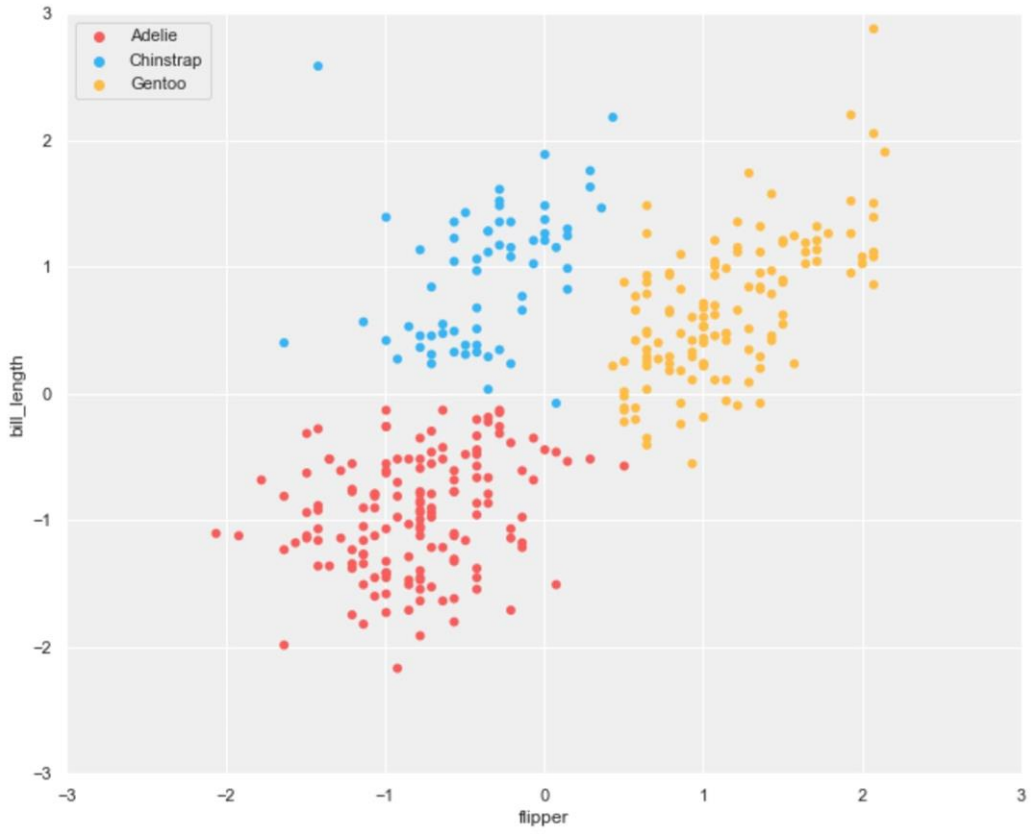


Рисунок 1.4 – Інтерфейс вікна відображення кластерних даних в KNIME

KNIME надає інтуїтивно зрозумілий інтерфейс для налаштування параметрів алгоритмів та візуалізації результатів, що робить її доступною для користувачів з різним рівнем технічної підготовки. Програма забезпечує широкі можливості інтеграції з різними джерелами даних та іншими аналітичними інструментами, що дозволяє розширювати її функціональність за потреби. Проте, незважаючи на гнучкість платформи, KNIME не пропонує спеціалізованих інструментів для детального поітераційного дослідження EM-алгоритму та аналізу його специфічних властивостей, що обмежує її використання для глибокого вивчення принципів роботи цього алгоритму.

Ці застосунки демонструють важливість та потенціал використання інформаційних технологій та машинного навчання для досліджень алгоритмів кластеризації, пропонуючи користувачам зручні застосунки для досліджень властивостей ітераційних алгоритмів, для прикладу EM-алгоритму. Розробка подібного програмного застосунку вимагає глибокого розуміння потреб користувачів, а також технічної експертизи для створення інтуїтивно зрозумілого, надійного, коректно працюючого та інтерактивного програмного забезпечення, здатного покращити точність і доступність для вивчення роботи алгоритму. У результаті аналізу аналогів була створена таблиця 1.1 для порівняння їхнього функціоналу з розроблюваним програмним забезпеченням.

Таблиця 1.1 – Порівняльний аналіз характеристик програмного забезпечення

	ELKI	WEKA	MATLAB	KNIME	Власна розробка
Інтерактивне налаштування синтетичних даних	1	1	1	1	1
Візуалізація проміжних ітерацій EM-алгоритму	0	1	1	0	1
Автоматична обробка пропущених значень у даних	0	1	0	1	1

Продовження таблиці 1.1.

	ELKI	WEKA	MATLAB	KNIME	Власна розробка
Обчислення внутрішніх та зовнішніх метрик якості	1	1	1	1	1
Порівняння з альтернативним алгоритмом кластеризації	1	1	1	1	1
Експорт результатів у PDF з ітераційними даними	0	0	1	1	1
Підтримка 3D-візуалізації кластерів	0	0	1	0	1
Сумарний коефіцієнт	43%	71%	86%	71%	100%

Згідно з даними порівняльного аналізу, наведеними в таблиці 1.1, розробка власного програмного забезпечення для дослідження EM-алгоритму демонструє суттєві переваги порівняно з існуючими системами, такими як ELKI, WEKA, MATLAB та KNIME.

Власна розробка досягає 100% функціональності за всіма функціональними критеріями оцінювання, тоді як аналоги демонструють різноманітні недоліки. MATLAB посідає друге місце з коефіцієнтом функціональності 86%, WEKA та KNIME мають по 71%, а ELKI демонструє лише 43% необхідної функціональності.

Основними особливостями власної реалізації є інтерактивне налаштування синтетичних даних, візуалізація проміжних ітерацій EM-алгоритму, автоматична обробка пропущених значень, комплексне обчислення внутрішніх та зовнішніх метрик якості, можливості 3D-візуалізації кластерів та експорт результатів у PDF з ітераційними даними.

Кількісно власна реалізація перевершує ELKI на 57%, WEKA та KNIME на 29%, а MATLAB на 14% за критеріями оцінювання. Ці метрики підтверджують,

що власна розробка надає дослідникам кращі можливості для глибокого аналізу даних, відстеження ітерацій алгоритму та гнучкого налаштування параметрів кластеризації.

Отже, за результатами порівняльного аналізу можна стверджувати, що розробка власного програмного забезпечення для дослідження EM-алгоритму є не лише обґрунтованою, але й необхідною для забезпечення повноцінних дослідницьких можливостей. Реалізоване програмне забезпечення уможливлює проведення більш детальних та гнучких експериментів завдяки повному набору необхідних функціональних характеристик, що робить його оптимальним вибором для наукових досліджень, навчальних цілей та практичного застосування в галузі аналізу даних та машинного навчання.

1.3 Аналіз методів розв’язання задачі

Розробка програмного забезпечення для аналізу кластеризації з використанням EM-алгоритму вимагає комплексного підходу до вирішення завдань, пов'язаних з визначенням кластерів у даних та їх інтерактивною візуалізацією. Основною задачею, яку повинен реалізувати програмний застосунок, є можливість автоматично визначати центри кластерів та відстежувати проміжні результати виконання алгоритму. Для досягнення цієї мети існують різні методи, кожен з яких має свої переваги та недоліки. Один з основних аспектів розробки програмних засобів для кластеризації – це визначення параметрів кластерів, таких як центри, коваріації та ваги, на основі вхідних даних, що дозволить в подальшому оцінити якість кластеризації. Для цього можна використовувати алгоритми комп'ютерного зору, які базуються на аналізі просторових даних для визначення кластерів. Однак такий підхід має певні недоліки. Він вимагає використання складних обчислень та має низький рівень точності для складних наборів даних. Тому такий підхід не варто застосовувати.

Також можливим методом кластеризації є використання алгоритмів машинного навчання, таких як DBSCAN, для автоматичного визначення

кластерів на основі щільності даних. Цей підхід дозволяє програмному забезпеченню самостійно розпізнавати кластери без необхідності вказувати їхню кількість заздалегідь. Однак цей метод може вимагати значних обчислювальних ресурсів та чутливий до вибору параметрів, таких як радіус околу та мінімальна кількість точок. Тому такий підхід не варто застосовувати. Інший підхід полягає в використанні алгоритму K-Means, який є простим і широко використовуваним методом кластеризації. Алгоритм K-Means базується на ітеративному визначенні центрів кластерів шляхом мінімізації суми квадратів відстаней до центроїдів. Він добре працює з даними сферичної форми, але не здатний обробляти кластери складної форми або різної щільності. Застосування алгоритму K-Means дозволяє отримати швидкі результати, але його обмеження у врахуванні коваріаційних структур даних робить його менш продуктивним для складних наборів даних.

Інший метод кластеризації базується на ієрархічній кластеризації, яка створює ієрархію кластерів шляхом їх послідовного об'єднання або поділу. Цей метод дозволяє отримати детальну структуру даних, але він є обчислювально складним і не підходить для великих наборів даних. Крім того, він не забезпечує інтерактивного відстеження ітерацій, що є важливим для аналізу алгоритмів. Тому використання цього методу є нерезультативним.

Найкращим підходом є використання EM-алгоритму, який дозволяє моделювати дані як суміш гаусівських розподілів. Цей метод точно визначає центри кластерів, коваріації та ваги шляхом ітеративного вдосконалення параметрів моделі. Додатково, інтерактивне відстеження ітерацій EM-алгоритму є важливим етапом у розробці програмних засобів для кластеризації. Цей процес визначає збереження параметрів кожної ітерації, таких як логарифмічна правдоподібність, центри та коваріації, для їх подальшого аналізу та візуалізації.

Для оцінки якості кластеризації можна скористатися комбінацією внутрішніх та зовнішніх метрик. Внутрішні метрики, такі як Silhouette Score, оцінюють компактність і відокремленість кластерів, тоді як зовнішні метрики, такі як Adjusted Rand Index, порівнюють результати з відомими мітками. Цей підхід дозволяє розрахувати якість кластеризації, використовуючи відомі

характеристики даних. Для цього необхідно спочатку визначити метрики, які оцінюють структуру кластерів, та реалізувати їх обчислення в програмному забезпеченні. Розробка власних методів для цього дозволяє зменшити похибку оцінювання і досягнути максимальної точності у результаті.

Під час розробки методів слід враховувати особливості даних, які можуть мати складну структуру, що відрізняється від ідеалізованих моделей, часто використовуваних у математичних алгоритмах. Тому важливим аспектом є обчислення метрик для кожного набору даних, що дозволить врахувати їхні статистичні особливості і забезпечити більш точні результати кластеризації.

Розглянувши переваги та недоліки кожного методу, було вирішено використовувати EM-алгоритм з інтерактивною візуалізацією та оцінкою якості кластеризації за допомогою комбінації внутрішніх та зовнішніх метрик. Цей підхід забезпечує найточніші результати у визначенні параметрів кластерів та їх візуалізації. Також було вирішено розробити та запрограмувати власні методи для обчислення метрик якості та експорту результатів у формати CSV і PDF [11]. Більше того, такий метод розробки дозволить створити унікальну платформу з інтерактивним функціоналом, включаючи підтримку синтетичних і користувацьких наборів даних, 3D-візуалізацію та порівняння з альтернативними алгоритмами, такими як K-Means.

1.4 Постановка задач дослідження

Розробка програмного продукту для аналізу кластеризації з використанням EM-алгоритму потребує використання мови програмування Python у середовищі Visual Studio Code [12], а також бібліотек PyQt5, scikit-learn і Matplotlib [13] для створення інтерактивного графічного інтерфейсу та візуалізації результатів кластеризації. Python – це інтерпретована, високорівнева мова програмування з простим синтаксисом, що широко використовується для розробки додатків у галузі машинного навчання, аналізу даних та візуалізації. Поєднання Python з бібліотеками PyQt5 для створення графічного інтерфейсу, scikit-learn для реалізації алгоритмів кластеризації та Matplotlib для візуалізації дозволить

створити програмні засоби, що забезпечують аналіз і візуалізацію кластерів у двовимірному та тривимірному просторі.

Розробка програмних засобів для аналізу кластеризації включає кілька етапів. По-перше, потрібно створити модуль графічного інтерфейсу користувача, який забезпечить зручну та інтуїтивно зрозумілу взаємодію з системою. Цей модуль відповідатиме за навігацію функціоналом застосунку, налаштування параметрів наборів даних і алгоритмів, а також відображення результатів. Далі буде розроблений модуль для генерації синтетичних наборів даних і обробки користувацьких даних у форматі CSV. Цей модуль використовуватиме бібліотеку scikit-learn для створення синтетичних даних та обробки пропущених значень за допомогою SimpleImputer. Після підготовки даних буде розроблений модуль для виконання EM-алгоритму, який використовуватиме GaussianMixture для визначення центрів кластерів, коваріацій і ваг, а також збереження параметрів кожної ітерації для подальшого аналізу.

Наступним етапом буде модуль для обчислення метрик якості кластеризації, який включатиме внутрішні метрики, такі як Silhouette Score, Log-Likelihood, BIC, AIC, та зовнішні метрики, такі як Adjusted Rand Index, для оцінки точності кластеризації. Також буде розроблено модуль для порівняння результатів EM-алгоритму з альтернативним алгоритмом K-Means, що дозволить користувачам оцінити результативність різних методів кластеризації. Модуль візуалізації відображатиме набори даних, результати кластеризації та проміжні ітерації у вигляді двовимірних або тривимірних діаграм розсіювання з еліпсами або еліпсоїдами, використовуючи Matplotlib. Окремий модуль відповідатиме за експорт результатів у формати CSV і PDF, включаючи дані про ітерації, метрики та візуалізації.

Розробка цих програмних модулів потребує глибокого розуміння алгоритмів машинного навчання, графічного програмування та роботи з Python у Visual Studio Code. Для реалізації модулів будуть використані бібліотеки PyQt5 для створення інтерфейсу, scikit-learn для кластеризації та метрик, Matplotlib для візуалізації, а також NumPy [14] і Pandas [15] для обробки даних. Буде проведено

тестування програмних модулів для забезпечення точності обчислень і стабільності роботи інтерфейсу.

Ретельно проаналізувавши переваги та недоліки існуючих програмних засобів для аналізу кластеризації, було визначено наступні завдання, які необхідно виконати для успішної розробки власного програмного забезпечення:

1. Розробити модуль генерації синтетичних наборів даних та їх інтерактивної візуалізації на основі параметрів, заданих користувачем.

2. Розробити модуль обробки користувацьких наборів даних, який забезпечить автоматичну обробку пропущених значень і застосування PCA для високовимірних даних.

3. Розробити алгоритм і програмний модуль для виконання EM-алгоритму, який буде спроможний відстежувати та зберігати параметри кожної ітерації.

4. Розробити модуль графічного інтерфейсу користувача, який буде зручним для дослідників і аналітиків, забезпечуючи швидкий доступ до налаштувань і результатів.

5. Реалізувати програмний продукт з підтримкою двовимірної та тривимірної візуалізації кластерів.

6. Автоматизувати обчислення внутрішніх і зовнішніх метрик якості кластеризації, забезпечуючи високу точність і надійність результатів.

7. Провести тестування програмного забезпечення для перевірки коректності роботи всіх модулів.

Отже, розробка програмних модулів для реалізації програмних засобів аналізу кластеризації з використанням Python у Visual Studio Code та бібліотек PyQt5, scikit-learn і Matplotlib є комплексним завданням, яке включає кілька етапів. Модулі повинні бути ретельно спроектовані, реалізовані та протестовані, щоб забезпечити користувачам зручну та інтерактивну роботу з програмним забезпеченням для аналізу та візуалізації кластерів.

1.5 Висновки

У першому розділі було проведено аналіз існуючих програмних застосунків для кластеризації, зосереджуючись на вивченні таких систем, як ELKI, WEKA, MATLAB та KNIME. Це дослідження дозволило виявити, що, хоча ці інструменти мають певні переваги завдяки стабільності та широкому набору алгоритмів, вони демонструють суттєві недоліки, які обмежують їх продуктивність для інтерактивного аналізу EM-алгоритму. Зокрема, більшість систем не підтримують інтерактивне налаштування синтетичних наборів даних, візуалізацію проміжних ітерацій алгоритму в тривимірному просторі та експорт результатів у PDF з детальними ітераційними даними, що значно знижує гнучкість і зручність дослідницького процесу.

Проведений порівняльний аналіз чітко окреслив потребу в створенні власної програми для дослідження властивостей EM-алгоритму з використанням Python у середовищі Visual Studio Code з бібліотеками PyQt5, scikit-learn і Matplotlib. Така програма не лише об'єднає найкращі функції існуючих систем, але й впровадить інноваційні можливості: інтерактивне налаштування синтетичних і користувацьких наборів даних, детальне відстеження еволюції кластерів з візуалізацією проміжних ітерацій, обчислення внутрішніх і зовнішніх метрик якості, порівняння з альтернативним алгоритмом K-Means, а також експорт результатів у CSV і PDF. Ці функції, реалізовані через зручний графічний інтерфейс, розроблений для наукових і навчальних потреб, забезпечать гнучкість і інтерактивність у аналізі кластеризації.

2 РОЗРОБКА МОДУЛЬНОЇ АРХІТЕКТУРИ СИСТЕМИ

2.1 Структурний аналіз EM-алгоритму для модульної реалізації

EM-алгоритм призначений для знаходження оцінок максимальної правдоподібності параметрів у статистичних моделях, де дані включають приховані змінні. У задачах кластеризації, таких як моделювання суміші гаусівських розподілів, приховані змінні представляють належність точок даних до певних кластерів. Алгоритм ітеративно опрацьовує логарифмічну правдоподібність даних, чергуючи два основних етапи: E-крок, або крок очікування, який обчислює очікувані значення прихованих змінних, та M-крок, або крок максимізації, який оновлює параметри моделі. Такий підхід дозволяє системі поступово вдосконалювати оцінки параметрів, таких як середні, коваріації та ваги кластерів, забезпечуючи точне групування даних.

Модульна структура системи дослідження EM-алгоритму є важливою для реалізації в програмному забезпеченні, оскільки вона дозволяє розділити обробку даних, обчислення параметрів і оцінку результатів на незалежні компоненти. Це сприяє комфортному управлінню потоком даних і полегшує інтеграцію з іншими функціональними модулями, такими як генерація даних, візуалізація та експорт результатів. У контексті програмного забезпечення для кластеризації, модульність забезпечує можливість обробки як синтетичних, так і користувацьких наборів даних, а також порівняння з альтернативними методами групування.

Послідовність EM-алгоритму складається з ініціалізації параметрів, ітеративного виконання E- та M-кроків, а також перевірки збіжності. На етапі ініціалізації система встановлює початкові значення параметрів моделі, таких як середні, коваріації та ваги компонент суміші. Ці параметри можуть бути визначені випадковим чином або за допомогою попереднього аналізу даних, наприклад, методу K-Means, для забезпечення швидшої збіжності. Модуль ініціалізації відповідає за підготовку даних і параметрів. E-крок відповідає за обчислення очікуваних значень прихованих змінних, відомих як

відповідальності, які відображають імовірність належності кожної точки даних до певного кластера. Для набору даних {x1, x2, . . . , xN } з K компонентами суміші відповідальність γ_{nk} обчислюється за формулою:

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}, \quad (2.1)$$

де π_k – вага k-ї компоненти, $\mathcal{N}(x_n|\mu_k, \Sigma_k)$ – гаусівська густина ймовірності для точки x_n з середнім μ_k і коваріацією Σ_k, μ_k – середнє k-ї компоненти, Σ_k – коваріаційна матриця k-ї компоненти, $\mathcal{N}$ – кількість точок даних, K – кількість кластерів.

Ця формула реалізована в модулі обчислення відповідальностей, який обробляє вхідні дані та поточні параметри моделі, зберігаючи результати для використання на M-кроці. Відповідальності є ключовими елементами для оцінки внеску кожної точки даних у параметри кластерів.

M-крок оновлює параметри моделі, максимізуючи логарифмічну правдоподібність на основі обчислених відповідальностей. Оновлення ваги π_k, середнього μ_k і коваріації Σ_k для k-ї компоненти виконується за формулами:

$$\pi_k = \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk}, \quad (2.2)$$

де π_k – нова вага k-ї компоненти, γ_{nk} – відповідальність точки x_n для k-ї компоненти, N – кількість точок даних.

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}}, \quad (2.3)$$

де μ_k – нове середнє k-ї компоненти, x_n – точка даних, γ_{nk} – відповідальність.

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma_{nk}(x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma_{nk}}, \quad (2.4)$$

де Σ_k – нова коваріаційна матриця k-ї компоненти, $(x_n - \mu_k)(x_n - \mu_k)^T$ – зовнішній добуток вектора відхилення, γ_{nk} – відповідальність.

Ці оновлення виконуються в модулі оновлення параметрів, який обробляє відповідальності та дані для обчислення нових значень. Модульність дозволяє ізолювати обчислення для кожної компоненти, полегшуючи паралельну обробку великих наборів даних.

Логарифмічна правдоподібність даних, яка оцінює якість моделі, обчислюється за формулою:

$$\ln p\,(X|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln\left(\sum_{k=1}^{K} \pi_{k\mathcal{N}}(x_n|\mu_k, \Sigma_k)\right), \quad (2.5)$$

де X – набір даних, π – вектор ваг, μ – вектор середніх, Σ – набір коваріаційних матриць, $\mathcal{N}(x_n|\mu_k, \Sigma_k)$ – гаусівська густина.

Ця формула використовується в модулі оцінки збіжності для порівняння правдоподібності між ітераціями. Зміна правдоподібності менша за заданий поріг, наприклад 10^{-6}, вказує на збіжність алгоритму. Перевірка збіжності виконується після кожного циклу E- та M-кроків. Модуль збіжності порівнює поточну правдоподібність із попередньою, припиняючи ітерації, якщо різниця є незначною, або повертаючись до E-кроку для наступної ітерації. Цей модуль також зберігає історію параметрів і правдоподібності для подальшого аналізу, що є важливим для відстеження прогресу алгоритму.

Модульна реалізація забезпечує гнучкість, дозволяючи обробляти різноманітні набори даних і інтегруватися з іншими аналітичними функціями. Детальна послідовність E- та M-кроків, підкріплена математичними формулами, забезпечує чітке розуміння роботи алгоритму, тоді як модульна структура полегшує його впровадження в програмне забезпечення, подібне до описаного.

2.2 Моделювання архітектури системи

Розробка програмного забезпечення для аналізу даних спирається на принципи програмної інженерії, які надають пріоритет модульності управління потоком даних та надійним можливостям обробки. Модульна архітектура, у якій окремі компоненти відповідають за конкретні функції, такі як введення даних, їх перетворення та генерація результатів, забезпечує масштабованість і легкість підтримки. Такі принципи особливо актуальні для програм, які обробляють складні набори даних, генерують аналітичні результати та створюють візуальні або файлові виходи, оскільки вони забезпечують надійну роботу та розширюваність. Ці концепції формують основу описаної системи, яка інтегрує обробку даних, агрегацію результатів і управління виведенням для підтримки аналітичних завдань.

Програма, реалізована на мові Python, використовує модульну архітектуру для обробки та аналізу даних, застосовуючи бібліотеки, такі як NumPy, Pandas, Matplotlib і scikit-learn. Система починає роботу з прийому вхідних даних, які можуть складатися з синтетично згенерованих наборів даних або табличних даних, імпортованих із файлів CSV. Спеціалізований модуль обробки даних опрацьовує ці входи, усуваючи невідповідності, такі як пропущені значення, шляхом заповнення їх статистичними середніми та, за потреби, зменшуючи розмірність даних для полегшення подальших операцій. Оброблені дані зберігаються в пам’яті у вигляді структурованого числового масиву, який слугує основою для всіх наступних процесів.

Після підготовки даних система спрямовує їх до обчислювального модуля, відповідального за організацію даних у групи. Цей модуль створює результати, які включають призначення груп і пов’язані з ними ймовірності, що зберігаються для подальшого використання. Оброблені результати передаються до модуля візуалізації, який відтворює дані у вигляді діаграм розсіювання у двовимірному або тривимірному просторі. Ці візуалізації відображають згруповану структуру даних, дозволяючи системі представити організацію, досягнуту під час обробки.

Блок-схему загальної роботи системи зображено на рисунку 2.1.

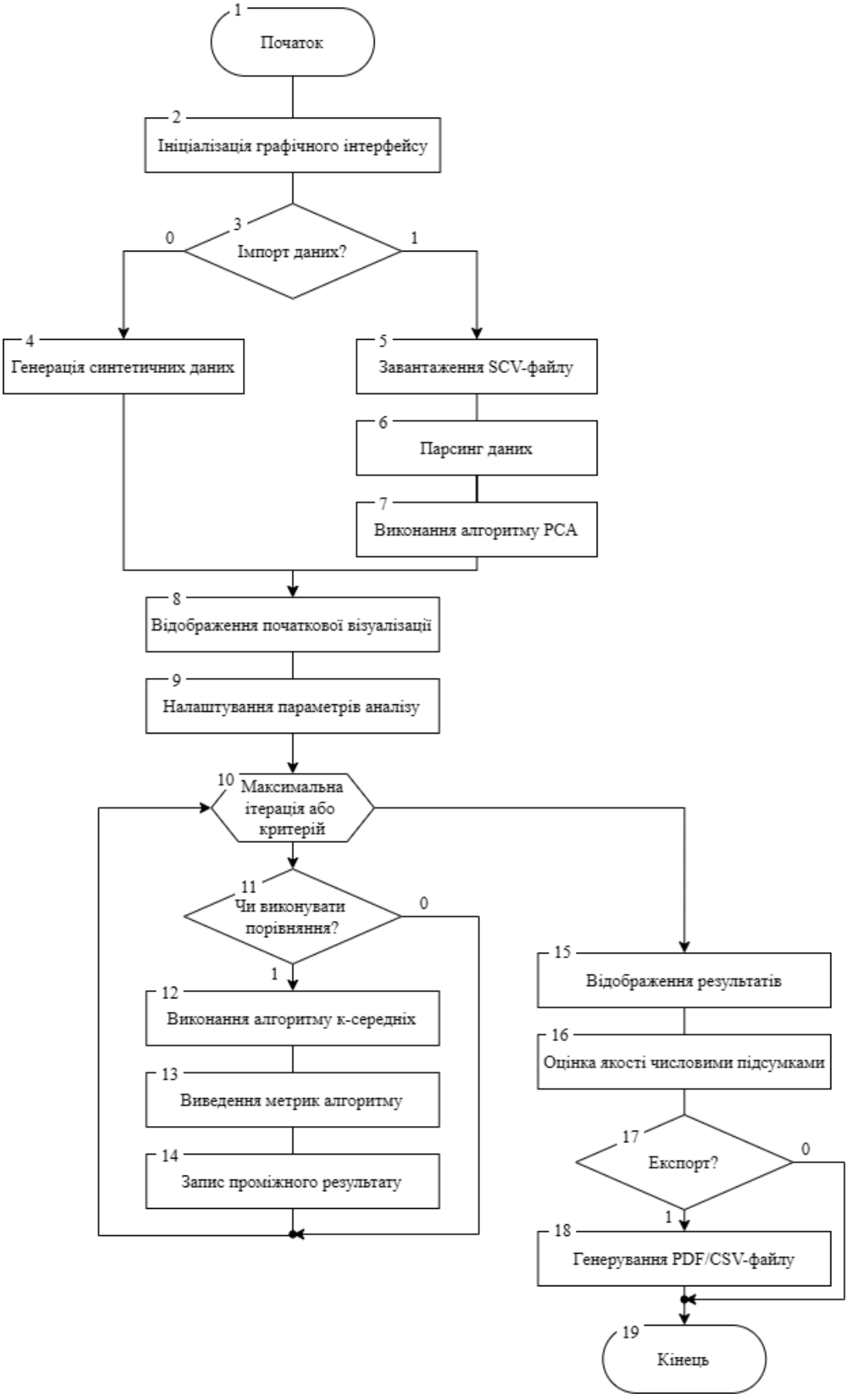


Рисунок 2.1 – Блок-схема загальної роботи системи

Після початкової обробки система агрегує результати для всебічного аналізу. Обчислювальний модуль зберігає проміжні результати, дозволяючи системі зберігати історію процесу групування. Ці проміжні результати доступні для детального аналізу, при цьому модуль візуалізації генерує додаткові діаграми для ілюстрації прогресу організації даних на різних етапах. Система також створює числові підсумки, які кількісно оцінюють якість групування, зберігаючи ці метрики разом із візуальними результатами для інтегрованого аналізу.

Для підтримки порівняльного аналізу система може виконати альтернативний метод групування, створюючи паралельні результати, які зберігаються та візуалізуються разом із основними результатами. Для збереження результатів аналізу модуль експорту перетворює оброблені дані, візуалізації та підсумки у вихідні формати. Система генерує CSV-файли, що містять дані та результати, а також PDF-документи, які об’єднують візуалізації ітерацій та числові підсумки. Механізм обробки подій забезпечує виконання операцій обробки даних, збереження результатів, візуалізації та експорту, у відповідь на системні тригери, підтримуючи цілісний потік даних.

2.3 Розробка підсистеми генерації синтетичних даних

Генерація синтетичних даних є дуже важливою підсистемою в програмному забезпеченні для аналізу даних, зокрема в задачах кластеризації, де контрольовані набори даних необхідні для тестування алгоритмів. Підсистема генерації синтетичних даних забезпечує створення різноманітних наборів даних із заданими характеристиками, що дозволяє досліджувати поведінку алгоритмів у різних сценаріях.

Алгоритм генерації синтетичних даних складається з послідовних етапів, які детально зображені на основній блок-схемі на рисунку 2.2. Процес розпочинається з ініціалізації модуля, що включає завантаження бібліотек і підготовку конфігурації. Далі визначається тип набору даних, що впливає на алгоритм генерації. Параметри, такі як кількість зразків, кластерів і рівень шуму,

проходять валідацію для забезпечення їх коректності. Після цього викликається відповідна функція генерації, яка створює числовий масив даних і мітки кластерів. Згенеровані дані перевіряються на відповідність розмірності та формату, зберігаються в пам’яті та позначаються як синтетичні для подальшої обробки.

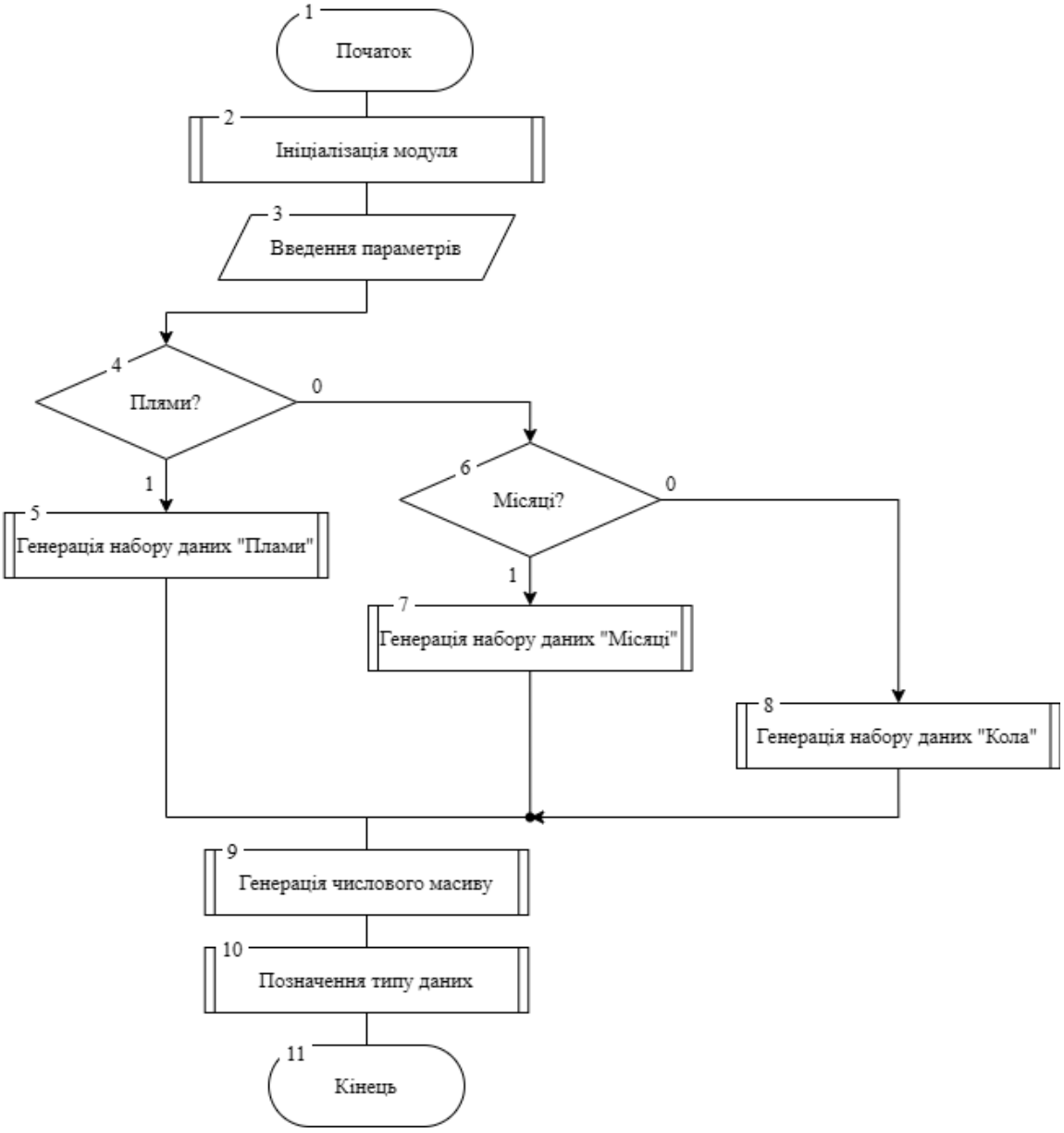


Рисунок 2.2 – Блок-схема роботи підсистеми генерації синтетичних даних

Генерація синтетичних даних базується на математичних моделях, реалізованих у функціях make_blobs, make_moons і make_circles.

Функція make_blobs генерує ізотропічні кластери за допомогою

гаусівського розподілу. Кожна точка x_n для k-ї компоненти генерується за формулою:

$$x_n \sim \mathcal{N}(\mu_k, \sigma_k^2 I), \tag{2.6}$$

де μ_k – центр k-ї компоненти, σ_k– стандартне відхилення, що визначається параметром шуму, I – одинична матриця, $\mathcal{N}$ – гаусівський розподіл.

Ця формула описує створення точок навколо центрів кластерів із ізотропною дисперсією, де шум σ_k контролює розсіювання точок. На рисунку 2.3 зображено приклад згенерованого набору даних.

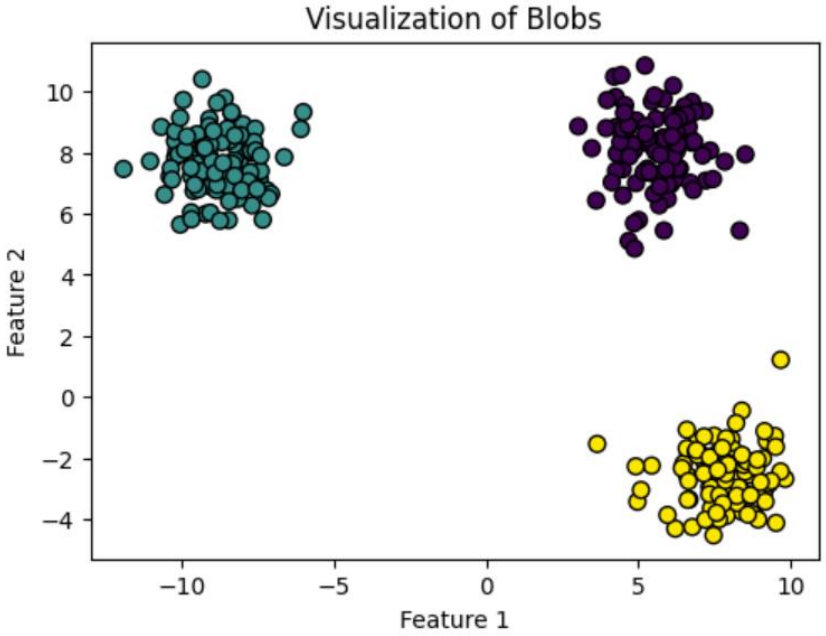


Рисунок 2.3 – Візуалізація згенерованого набору даних «Blobs»

Генерація включає вибір центрів, встановлення рівня шуму, генерацію точок, призначення міток і повернення даних.

Функція make_moons створює дві півмісяцеві форми за допомогою нелінійного перетворення. Точки генеруються за формулою:

$$x_n = \begin{bmatrix} \cos(\theta_n) \\ \sin(\theta_n) \end{bmatrix} + \epsilon_n, \quad \theta_n \in [0, \pi], \tag{2.7}$$

де θ_n – кут для півмісяця, $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ – гаусівський шум, σ – параметр шуму.

Другий півмісяць генерується шляхом зміщення та відображення першого, створюючи нелінійно відокремлені кластери. Параметр шуму додає випадкові відхилення до точок. На рисунку 2.4 зображено приклад згенерованого набору даних «Місяці» з різними значеннями заданого шуму.

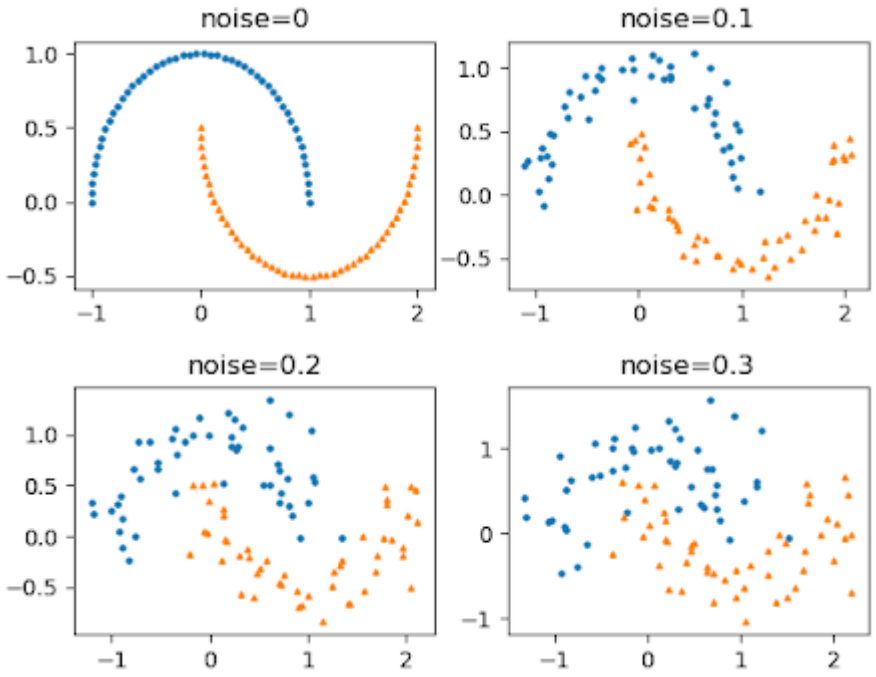


Рисунок 2.4 – Приклад згенерованого набору даних «Місяці»

Це охоплює генерацію кутів, застосування нелінійного перетворення, додавання шуму, призначення міток і повернення даних.

Функція make_circles створює концентричні кільця за допомогою радіального розподілу. Точки для кільця генеруються за формулою:

$$x_n = r \begin{bmatrix} \cos(\theta_n) \\ \sin(\theta_n) \end{bmatrix} + \epsilon_n, \tag{2.8}$$

де r – радіус кільця, що залежить від параметра factor, $\theta_n \in [0,2\pi]$ – кут, $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ – гаусівський шум, σ – параметр шуму.

Параметр factor визначає відношення радіусів внутрішнього та зовнішнього кілець, створюючи нелінійно відокремлені кластери. Приклад згенерованого набору даних «Кільця» з різними значеннями заданого шуму зображено на рисунку 2.5.

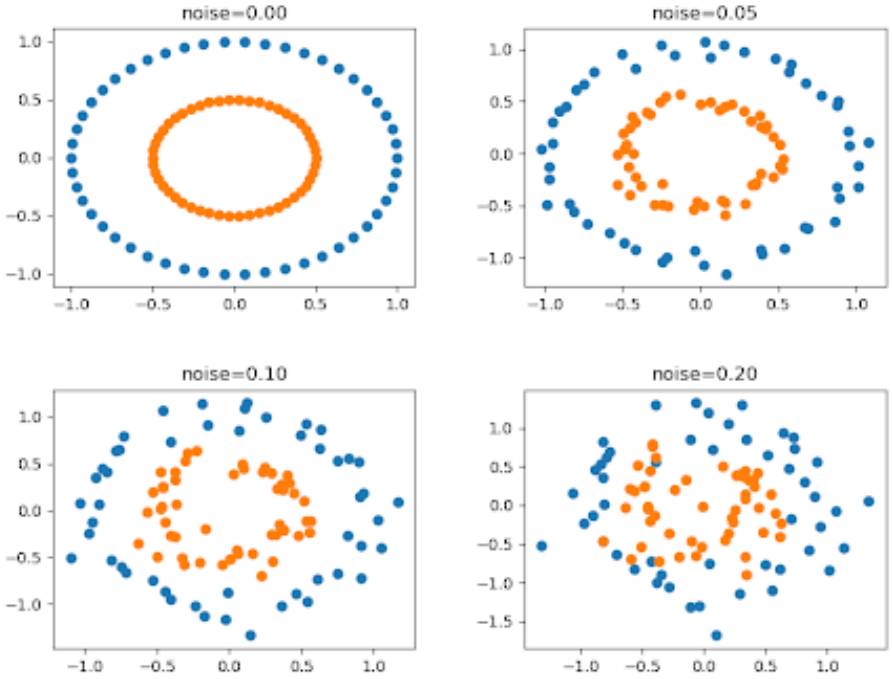


Рисунок 2.5 – Приклад згенерованого набору даних «Кільця»

Підсистема генерації синтетичних даних інтегрується з іншими модулями програмного забезпечення, такими як обробка даних і кластеризація. Модуль оцінки якості забезпечує коректність параметрів і даних перед передачею до EM-алгоритму. Згенеровані дані передаються до модуля обробки для попередньої підготовки, наприклад, зменшення розмірності, а також до модуля візуалізації для створення діаграм розсіювання. Модуль експорту дозволяє зберігати синтетичні дані у файлах CSV для подальшого використання. Детальна алгоритмічна послідовність, підкріплена математичними формулами, дозволяє створювати різноманітні набори даних із заданими характеристиками. Основна блок-схема та математичні формули деталізують процеси генерації, забезпечуючи чітке розуміння технічної реалізації.

2.4 Розробка підсистеми інтерактивної візуалізації ЕМ-алгоритму

Інтерактивна візуалізація є ключовим компонентом програмного забезпечення для аналізу даних, дозволяючи користувачам досліджувати результати кластеризації. У контексті розробленого програмного забезпечення, підсистема інтерактивної візуалізації забезпечує відображення даних, призначень кластерів і параметрів моделі у вигляді діаграм розсіювання та гаусівських еліпсів, а також підтримує динамічне переглядання історії ітерацій ЕМ-алгоритму.

Підсистема інтерактивної візуалізації функціонує як модуль, який інтегрується з обчислювальним ядром програмного забезпечення, особливо з модулем ЕМ-алгоритму, для відображення результатів кластеризації. Вона використовує бібліотеку Matplotlib, вбудовану в графічний інтерфейс, для створення діаграм розсіювання, що показують точки даних із кольоровим кодуванням за кластерами, та гаусівських еліпсів, що відображають розподіли компонент моделі. Модуль підтримує інтерактивність, дозволяючи переглядати проміжні ітерації ЕМ-алгоритму, що допомагає аналізувати прогрес кластеризації. Модуль отримує оброблені дані та параметри моделі, такі як середні та коваріації, і генерує графічні представлення, які можуть бути збережені або експортовані. Валідація даних і параметрів перед візуалізацією гарантує коректність відображення, що є важливим для точного аналізу.

Процес розпочинається з ініціалізації графічного полотна, що включає завантаження бібліотек і підготовку контейнера для графіків. Далі отримуються оброблені дані та параметри моделі, які проходять валідацію для перевірки їх формату та цілісності. Алгоритм ініціалізує графік, створюючи діаграму розсіювання для відображення точок даних із призначеннями кластерів. Гаусівські еліпси генеруються на основі середніх і коваріацій компонент моделі, додаючи візуальне представлення розподілів. Інтерактивність забезпечується обробкою подій, таких як вибір ітерації, що оновлює графік відповідно до параметрів обраної ітерації.

Зображення блок-схеми роботи підсистеми зображено на рисунку 2.6.

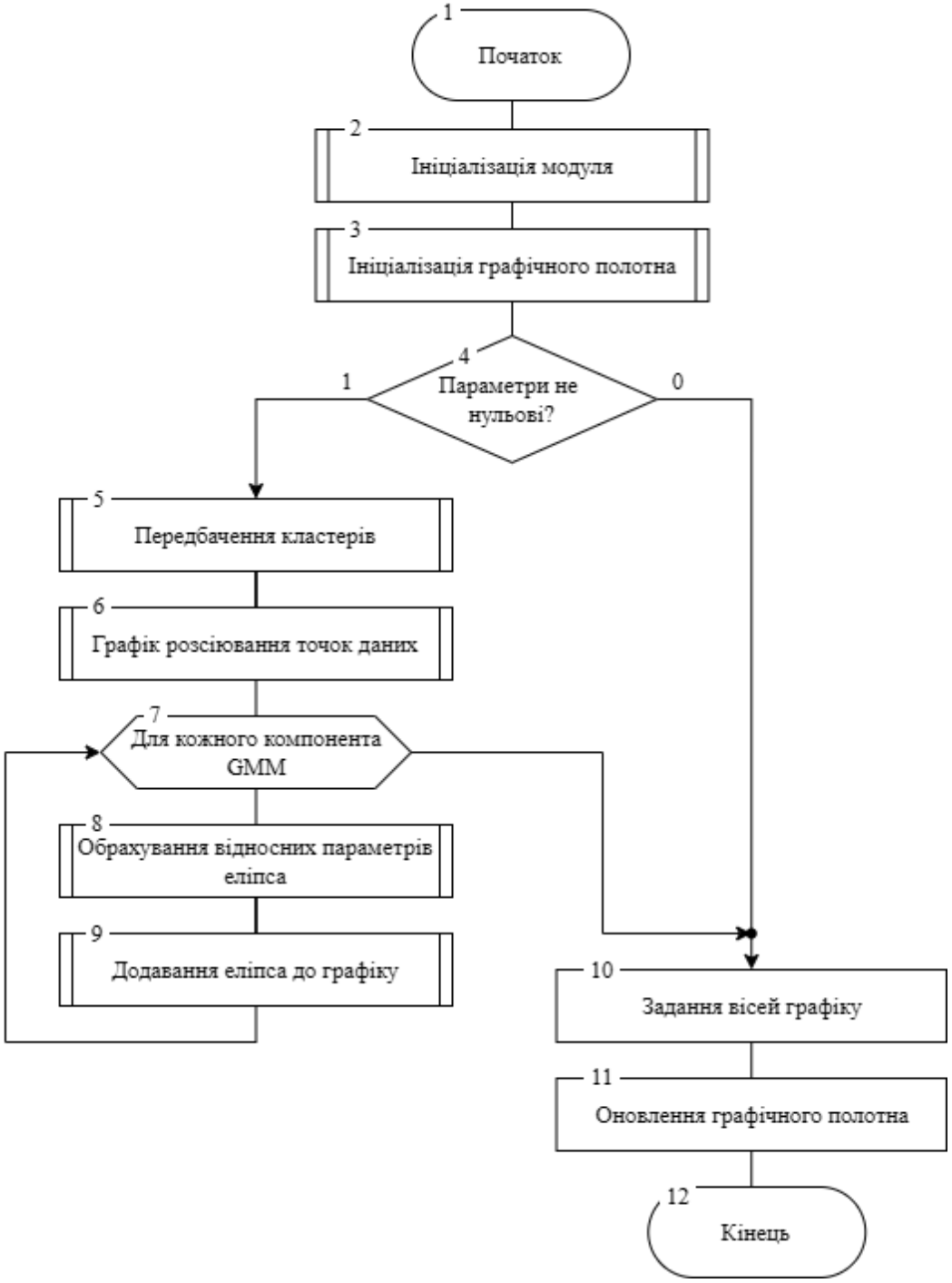


Рисунок 2.6 – Блок-схема роботи підсистеми інтерактивної візуалізації ЕМ-алгоритму

Інтерактивність забезпечується обробкою подій, таких як вибір ітерації, що оновлює графік відповідно до параметрів обраної ітерації. Згенеровані графіки перевіряються на коректність і зберігаються для відображення або експорту. Кожен компонент візуалізації має власну логіку, що деталізується в формулах для діаграми розсіювання, еліпсів і навігації ітераціями.

Візуалізація результатів ЕМ-алгоритму базується на математичних моделях, що описують положення точок і параметри компонент моделі.

Діаграма розсіювання відображає точки даних x_n у двовимірному просторі, де кожна точка має координати та мітку кластера. Координати точки задаються вектором:

$$\mathrm{x_n} = \begin{bmatrix} \mathrm{x_{n1}} \\ \mathrm{x_{n2}} \end{bmatrix}, \tag{2.9}$$

де x_{n1} – координата по осі X, x_{n2} – координата по осі Y, $n = 1, \ldots, N, N$ – кількість точок.

Мітки кластерів y_n визначаються ЕМ-алгоритмом на основі максимальної відповідальності γ_{nk}. Ця формула забезпечує позиціонування точок на графіку, де колір точки відповідає її кластеру.

Гаусівські еліпси відображають розподіли компонент моделі, використовуючи середні μ_k і коваріаційні матриці Σ_k. Еліпс для k-ї компоненти задається рівнянням:

$$(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) = c, \tag{2.10}$$

де $\mu_k = \begin{bmatrix} \mu_{k1} \\ \mu_{k2} \end{bmatrix}$ – середнє k-ї компоненти, Σ_k – коваріаційна матриця, c – константа, зазвичай 1 або 2 для 1–2 стандартних відхилень, x – точка на еліпсі.

Еліпс обчислюється через власні значення та вектори Σ_k, що визначають його форму та орієнтацію. Ця формула дозволяє візуалізувати форму кластера.

Навігація ітераціями дозволяє відображати параметри моделі для кожної ітерації ЕМ-алгоритму. Параметри оновлюються за формулами:

$$\mu_k^{(t+1)} = \frac{\sum_{n=1}^{N} \gamma_{nk}^{(t)} x_n}{\sum_{n=1}^{N} \gamma_{nk}^{(t)}}, \tag{2.11}$$

де $\mu_k^{(t+1)}$ – середнє k-ї компоненти на ітерації $t+1$, $\gamma_{\mathrm{nk}}^{(\mathrm{t})}$– відповідальність на ітерації t, x_n – точка даних, N – кількість точок.

Ця формула описує оновлення середніх, які використовуються для оновлення діаграми розсіювання та еліпсів для обраної ітерації.

Підсистема інтерактивної візуалізації інтегрується з модулями обробки даних і кластеризації. Модуль валідації перевіряє коректність даних і параметрів перед візуалізацією. Модуль рендерингу генерує графіки, використовуючи Matplotlib, і вбудовує їх у графічний інтерфейс через FigureCanvasQTAgg. Модуль інтерактивності обробляє події, такі як вибір ітерації, для динамічного оновлення графіків. Модуль експорту дозволяє зберігати графіки у форматі PDF.

Підсистема інтерактивної візуалізації є важливим компонентом програмного забезпечення для кластеризації, забезпечуючи наочне представлення результатів EM-алгоритму. Алгоритмічна послідовність, підкріплена математичними формулами, дозволяє створювати діаграми розсіювання, гаусівські еліпси та підтримувати інтерактивність.

2.5 Розробка графічної схеми інтерфейсу

Для створення естетичного та функціонального дизайну системи слід приділити увагу проектуванню інтерфейсу. Для цього використовуватимуться графічні схеми, які допоможуть візуалізувати структуру та вигляд, забезпечуючи зручність та естетичний вигляд користувацького середовища. Графічний дизайн відіграє ключову роль у створенні зручного та привабливого інтерфейсу, що сприяє зручності взаємодії користувача з системою, тому особлива увага приділяється кожній деталі та елементу вікна програми.

При розробці графічної схеми інтерфейсу головного вікна системи, увага була спрямована на створення зручного та інтуїтивно зрозумілого середовища для користувача. На рисунку 2.7 показано схематичне зображення головного вікна системи з пронумерованими окремими секціями, і описано, за що кожна секція відповідає.

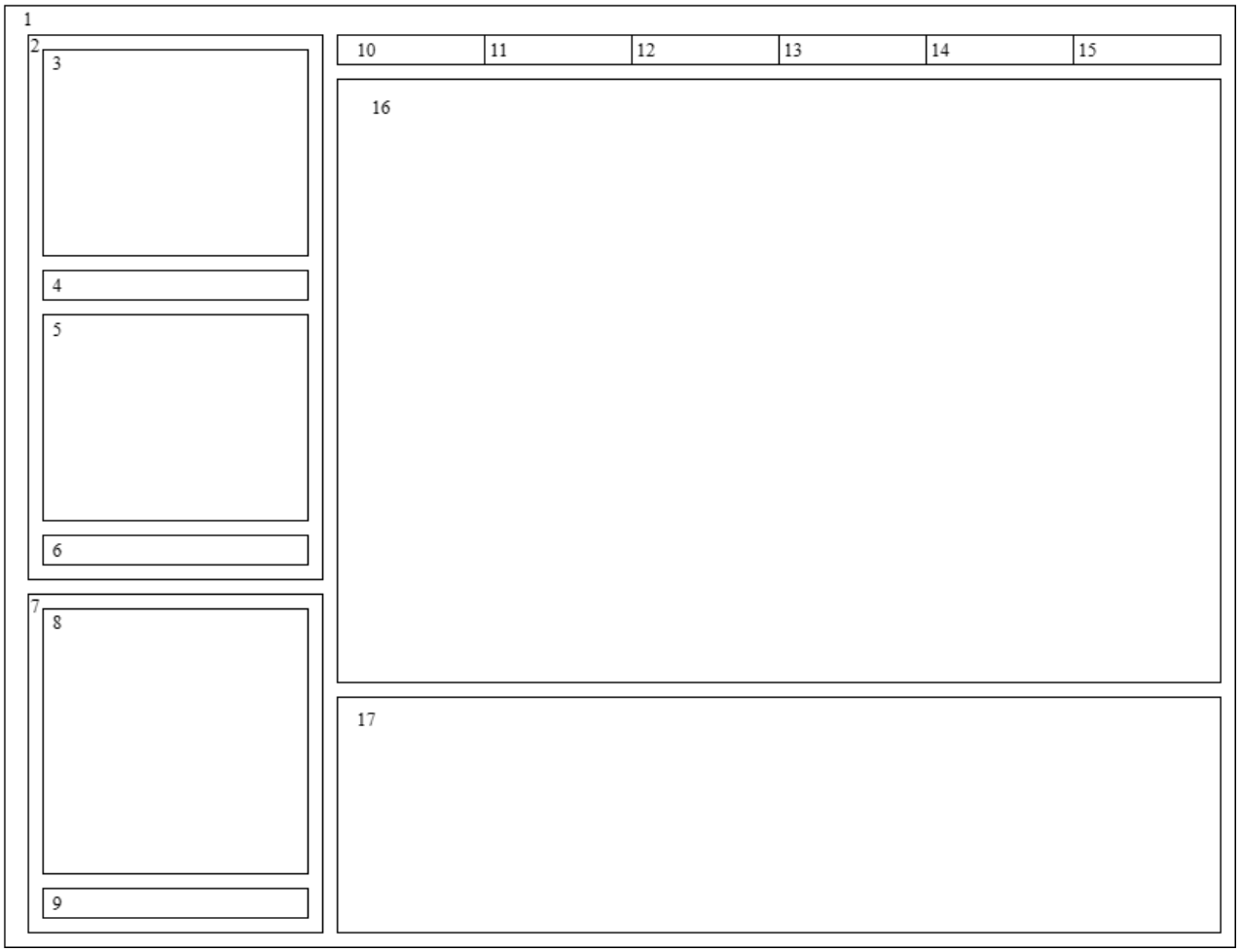


Рисунок 2.7 – Графічна схема інтерфейсу головного вікна системи

Основними елементами головного вікна є:

1. Головне вікно системи.
2. Частина вікна, де знаходиться управління імпортом або генерацією синтетичних даних.
3. Параметри налаштування генерації синтетичних даних
4. Кнопка, що при натисненні дозволяє імпортувати CSV-файл.
5. Область вікна, де після імпорту будуть відображатися колонки, отримані з CSV-файлу.
6. Прапорець, що дозволяє обрати, чи є в файлі названі колонки, чи він лише містить числові дані, що дозволяє скоротити час роботи.
7. Частина вікна, де знаходиться управління параметрами роботи з EM-алгоритмом.
8. Параметри налаштування EM-алгоритму.

9. Прапорець, що дозволяє обрати, чи буде порівняння роботи ЕМ-алгоритму з алгоритмом К-середніх.

10. Кнопка для переходу на вкладку зі згенерованим/відкритим набором даних.

11. Кнопка для переходу на вкладку з результатами роботи алгоритму.

12. Кнопка для переходу на вкладку з результатами порівняння алгоритмів.

13. Кнопка для переходу на вкладку з інформацією про роботу алгоритму.

14. Кнопка для переходу на вкладку з історією ітерацій та значень.

15. Кнопка для переходу на вкладку для експортування результатів роботи алгоритму.

Розроблена схема графічного інтерфейсу забезпечує взаємодію з програмним забезпеченням для кластеризації, інтегруючи управління даними та аналізом. Головне вікно об’єднує елементи для імпорту CSV-файлів, генерації синтетичних даних і налаштування ЕМ-алгоритму, спрощуючи підготовку та аналіз. Навігаційні вкладки надають доступ до даних, результатів, порівняння алгоритмів і експорту, підтримуючи структурований робочий процес. Ця організація відображає модульну структуру системи, підвищуючи зручність аналізу.

Інтерфейс підтримує гнучке налаштування параметрів і порівняння ЕМ-алгоритму з алгоритмом К-середніх, розширюючи аналітичні можливості системи. Область відображення даних і історія ітерацій полегшують перевірку результатів, а експорт забезпечує збереження підсумків. Отже, схема інтерфейсу спрощує взаємодію з системою, поєднуючи зручність, функціональність і підтримку складних аналітичних задач.

2.6 Висновки

В другому розділі було виконано структурний аналіз ЕМ-алгоритму, що розкрив його модульну організацію для реалізації кластеризації. Реалізовано ітеративні Е- та М-кроки, підкріплені математичними формулами

відповідальностей і оновлення параметрів. Було розроблено підсистему генерації синтетичних даних, яка використовує функції бібліотеки scikit-learn для створення різноманітних наборів даних. Система забезпечує гнучке тестування алгоритмів завдяки підтримці ізотропних і нелінійних структур даних.

Було створено підсистему інтерактивної візуалізації, що відображає результати EM-алгоритму через діаграми розсіювання та гаусівські еліпси. Реалізовано навігацію ітераціями, яка полегшує аналіз проміжних результатів кластеризації. Графічний інтерфейс об’єднує управління даними, налаштування алгоритмів і доступ до результатів через зручні навігаційні вкладки. Платформа дозволяє порівнювати EM-алгоритм із K-середніми, розширюючи аналітичні можливості.

Було сформовано цілісну систему для аналізу даних, що поєднує обчислювальну точність із зручністю використання. Модульна структура сприяє легкій інтеграції нових функцій, таких як додаткові методи аналізу. Зокрема, реалізовано експорт результатів у зручному форматі для подальшого використання. Система створює міцну основу для наукових і прикладних досліджень у галузі кластеризації.

3 . РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ СИСТЕМИ

3.1 Аналіз і обґрунтування вибору засобів для реалізації системи

Розробка системи для аналізу даних і кластеризації вимагає вибору засобів, що забезпечують належне виконання завдань, модульність і зручність використання. Метою системи було створення платформи для реалізації EM-алгоритму, генерації синтетичних даних, відображення результатів через інтерактивну візуалізацію та надання зрозумілого графічного інтерфейсу. Для цього було використано мову програмування Python, бібліотеки scikit-learn, NumPy, Matplotlib і фреймворк PyQt5 [16]. Вибір кожного інструменту ґрунтувався на його продуктивності, здатності виконувати потрібні функції, сумісності з іншими компонентами та наявності підтримки спільноти.

Для програмування системи було обрано мову Python через її універсальність, зрозумілий синтаксис і широку бібліотечну базу для обробки даних. Вона сприяє швидкому створенню прототипів і полегшує підтримку коду, що важливо для модульної системи. Python має розвинену спільноту, велику кількість документації та сумісність із бібліотеками для машинного навчання і візуалізації, такими як scikit-learn і Matplotlib. Python об’єднує всі компоненти, від обробки даних до графічного інтерфейсу, забезпечуючи коректну роботу системи. Альтернативи, такі як R або Java, менш гнучкі або складніші для подібних завдань.

Для створення графічного інтерфейсу було обрано PyQT5 завдяки широкому набору віджетів і підтримці складних структур інтерфейсу. Фреймворк забезпечує кросплатформність, дозволяючи системі працювати на різних операційних системах без значних змін. Інтеграція з Matplotlib спрощує вбудовування інтерактивних графіків, таких як діаграми розсіювання та еліпси. PyQt5 відповідає потребам системи з вкладками для даних, результатів і експорту, підтримуючи модульну організацію. У порівнянні з Tkinter, який є простішим, але обмеженим у функціональності, або wxPython, що має застарілий вигляд, PyQt5 пропонує кращий баланс можливостей і зручності.

Для програмної реалізації ЕМ-алгоритму найкращою бібліотекою буде scikit-learn. Бібліотека містить готові інструменти для кластеризації та функції для створення даних, що скорочує час розробки. Вона працює з наборами даних малого та середнього розміру, які відповідають потребам системи, і має чітку документацію. scikit-learn гарно поєднується з NumPy, що забезпечує швидку обробку числових масивів. Альтернативи, такі як TensorFlow, орієнтовані на глибоке навчання, а MLlib на великі дані, що менш підходить для системи.

Matplotlib обрано для створення інтерактивних візуалізацій через його гнучкість і здатність створювати різні графіки, такі як діаграми розсіювання та гаусівські еліпси. Бібліотека інтегрується з PyQt5, що дозволяє вбудовувати графіки в інтерфейс. Вона дає змогу точно налаштовувати візуалізації, що необхідно для відображення ітерацій ЕМ-алгоритму. Matplotlib має активну спільноту і численні приклади, що полегшують вирішення технічних питань. У порівнянні з Seaborn, який менш гнучкий, або Plotly, орієнтованим на веб-додатки, Matplotlib краще відповідає настільним системам.

Через високу швидкість операцій з масивами та матрицями було обрано бібліотеку NumPy. Вона є основою для роботи scikit-learn і Matplotlib, підтримуючи обчислення для ЕМ-алгоритму та підготовки даних. Її простота і сумісність із іншими бібліотеками роблять NumPy важливим компонентом системи. Альтернативи, такі як Pandas, більше підходять для табличних даних, але мають більший обсяг пам’яті.

Вибір мови, фреймворку та бібліотек зумовлений їхньою відповідністю потребам системи, включаючи модульність, продуктивність і зручність. Python створює універсальну платформу для поєднання всіх компонентів. PyQt5 і Matplotlib забезпечують зручний інтерфейс із інтерактивними графіками. scikit-learn і NumPy підтримують швидку обробку даних і реалізацію алгоритмів кластеризації. Ці інструменти формують цілісну систему, що відповідає задачам аналізу даних.

3.2 Розробка модуля оцінки якості кластеризації

Метод оцінки якості кластеризації забезпечує кількісний аналіз результатів EM-алгоритму та, за потреби, алгоритму K-середніх у програмному забезпеченні для аналізу даних. Модуль використовує три метрики: коефіцієнт силуету, індекс Девіса-Болдіна та скоригований індекс Ренда, що дозволяють оцінити якість кластеризації з різних аспектів. Метод інтегрується з системою, обробляючи дані та мітки кластерів для обчислень і збереження результатів для відображення чи експорту.

UML-діаграма класу [17], яку зображено на рисунку 3.1, ілюструє структуру класу ClusteringEvaluationModule. Клас містить такі атрибути, як посилання на MainWindow і evaluation_results, що відповідає за словник для зберігання результатів метрик, таких як коефіцієнт силуету чи індекс Девіса-Болдіна. Методи включають evaluate_clustering для обчислення метрик і get_evaluation_results для доступу до результатів. Модуль є частиною головного класу інтерфейсу, отримуючи дані після кластеризації.

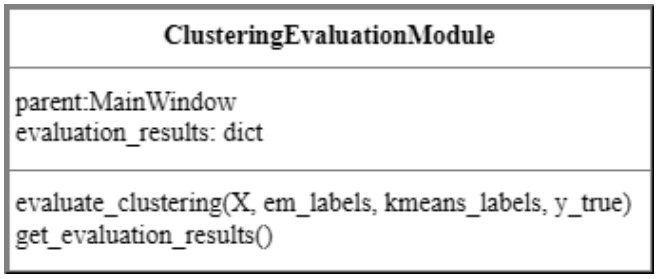


Рисунок 3.1 – Діаграма класу ClusteringEvaluationModule з відображенням його атрибутів та методів

Блок-схема, яку зображено на рисунку 3.2, охоплює весь процес роботи модуля оцінки. Починається з ініціалізації словника evaluation_results як порожнього. Далі перевіряється, чи є набір даних X і чи містить він принаймні дві точки. Якщо ні, записується повідомлення про помилку, і процес завершується. Якщо дані валідні, перевіряється наявність міток em_labels і принаймні двох кластерів. При виконанні умови обчислюються коефіцієнт силуету, індекс Девіса-Болдіна і скоригований індекс Ренда.

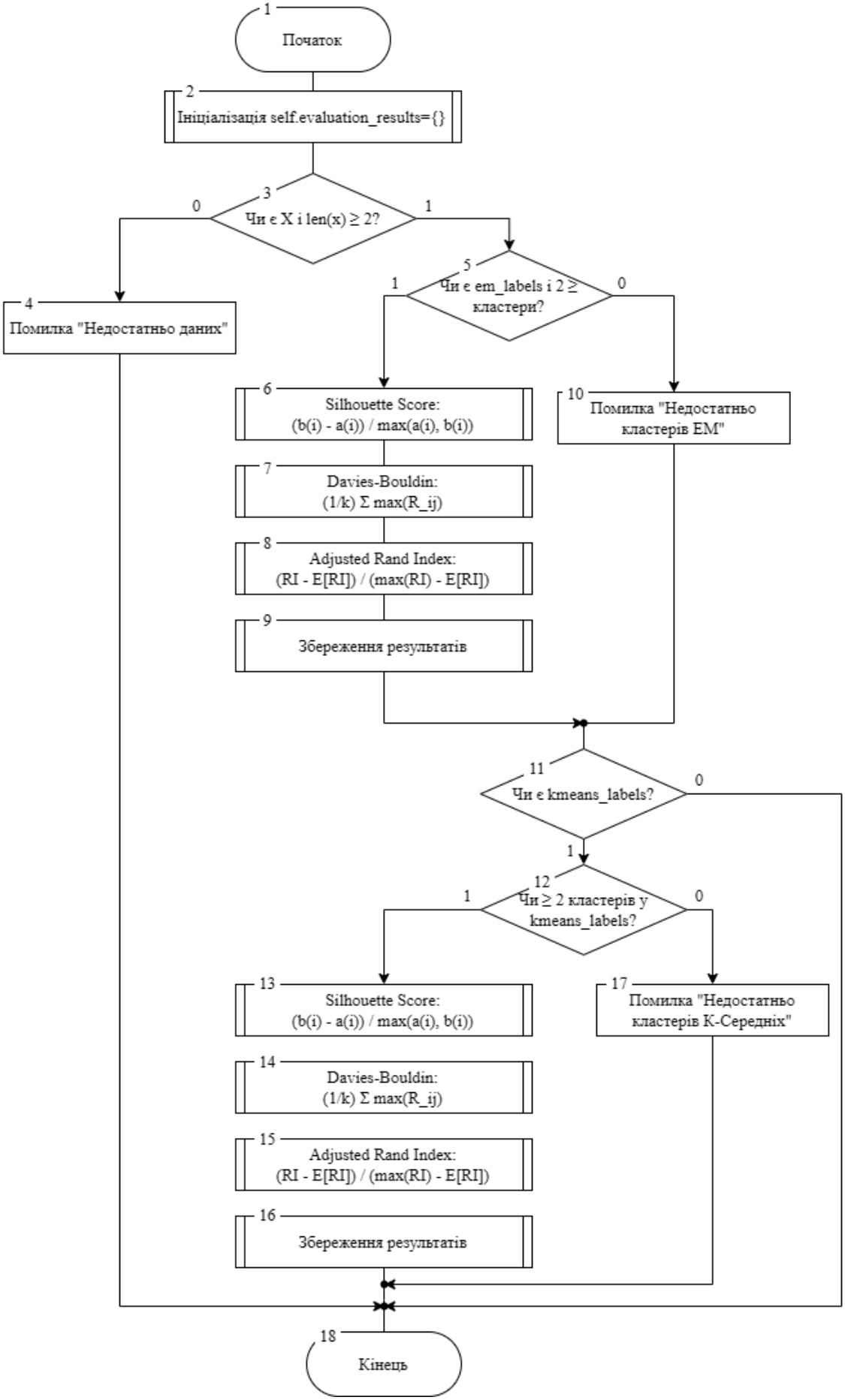


Рисунок 3.2 – Блок-схема роботи модуля оцінки якості кластеризації

Результати зберігаються в словнику. Якщо кластерів недостатньо, записується помилка для EM. Потім перевіряється наявність kmeans_labels. Якщо вони є, перевіряється кількість кластерів, і за наявності двох або більше обчислюються аналогічні метрики для K-середніх, з подальшим збереженням результатів. У разі помилки записується відповідне повідомлення. Процес завершується після обробки всіх умов.

Метод оцінки якості кластеризації базується на трьох метриках, кожна з яких аналізує різні характеристики кластерів. Коефіцієнт силуету оцінює, наскільки точки в одному кластері подібні між собою порівняно з точками в інших кластерах. Він обчислюється для кожної точки шляхом порівняння середньої відстані до точок її кластера з мінімальною середньою відстанню до іншого кластера, надаючи значення від -1 до 1, де вищі значення вказують на кращу кластеризацію. Індекс Девіса-Болдіна вимірює компактність кластерів і відстань між ними, обчислюючи середню дисперсію всередині кластерів і відстані між їхніми центроїдами, де нижчі значення свідчать про чіткіше розділення. Скоригований індекс Ренда порівнює отримані мітки кластерів із відомими істинними мітками, якщо вони доступні, оцінюючи відповідність через кількість пар точок, що збігаються в кластерах, із значеннями від -1 до 1, де 1 означає ідеальну відповідність. Ці метрики забезпечують всебічний аналіз якості кластеризації.

Коефіцієнт силуету є метрикою, що оцінює якість кластеризації, аналізуючи, наскільки точки в одному кластері подібні між собою порівняно з точками інших кластерів. Він вимірює внутрішню компактність кластера і відстань між кластерами, надаючи значення від -1 до 1. Вищі значення вказують на чітко визначені кластери, де точки ближчі до свого кластера, ніж до інших. Обчислення аналізує кожну точку даних, порівнює середні відстані до точок свого кластера з мінімальною середньою відстанню до іншого кластера, і усереднює результати. Метрика корисна для оцінки результатів алгоритмів, коли істинні мітки відсутні, допомагаючи визначити оптимальну кількість кластерів. Блок схему методу зображено на рисунку 3.2.

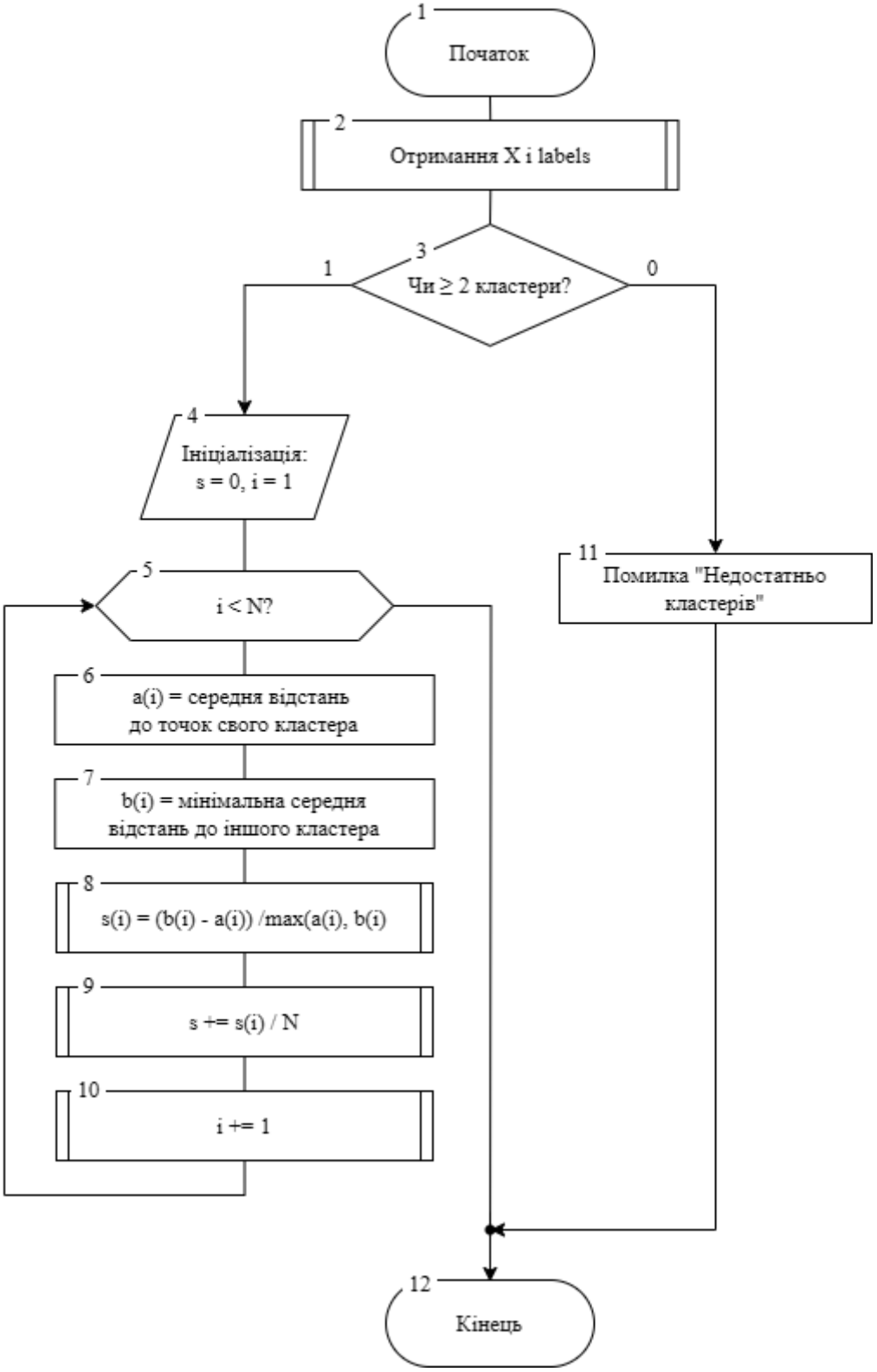


Рисунок 3.2 – Блок-схема методу обрахування коефіцієнту силуету

Процес починається з отримання набору даних X і міток кластерів, які визначають, до якого кластера належить кожна точка. Першим кроком є перевірка, чи є принаймні два кластери, оскільки метрика втрачає сенс для

одного кластера. Якщо кластерів недостатньо, процес завершується з повідомленням про помилку, що економить обчислювальні ресурси. За наявності достатньої кількості кластерів ініціалізується змінна для накопичення результату. Далі запускається цикл, який обробляє кожну точку даних послідовно. Для кожної точки обчислюється середня відстань до всіх інших точок її кластера, що відображає внутрішню когезію. Потім визначається мінімальна середня відстань до точок іншого кластера, що вказує на сепарацію. Ці значення порівнюються, щоб отримати коефіцієнт для точки, який додається до загальної суми з урахуванням кількості точок. Цикл повторюється, поки всі точки не будуть оброблені, забезпечуючи повне охоплення даних. Після завершення циклу повертається середнє значення метрики, яке відображає загальну якість кластеризації.

Індекс Девіса-Болдіна оцінює якість кластеризації, аналізуючи компактність кластерів і їхнє розділення. Компактність вимірюється як середня дисперсія точок усередині кластера, що показує, наскільки тісно згруповані точки. Розділення оцінюється через відстань між центроїдами кластерів, що вказує на віддаленість кластерів один від одного. Метрика обчислює відношення суми дисперсій двох кластерів до відстані між їхніми центроїдами, вибираючи максимальне значення для кожного кластера, і усереднює результати. Нижчі значення індексу свідчать про кращу кластеризацію, де кластери компактні й добре розділені. Ця метрика є результативною для порівняння результатів алгоритмів коли потрібно оцінити структурну якість кластерів.

Процес розпочинається з отримання даних X і міток кластерів. Спочатку перевіряється, чи є принаймні два кластери, що є важливою перевіркою, оскільки метрика залежить від порівняння між кластерами. Якщо кластерів недостатньо, процес завершується з помилкою, що запобігає непотрібним обчисленням. За наявності достатньої кількості кластерів ініціалізується змінна для індексу та визначається кількість кластерів.

Блок-схему роботи методу обрахування індексу Девіса-Болдіна зображено на рисунку 3.3.

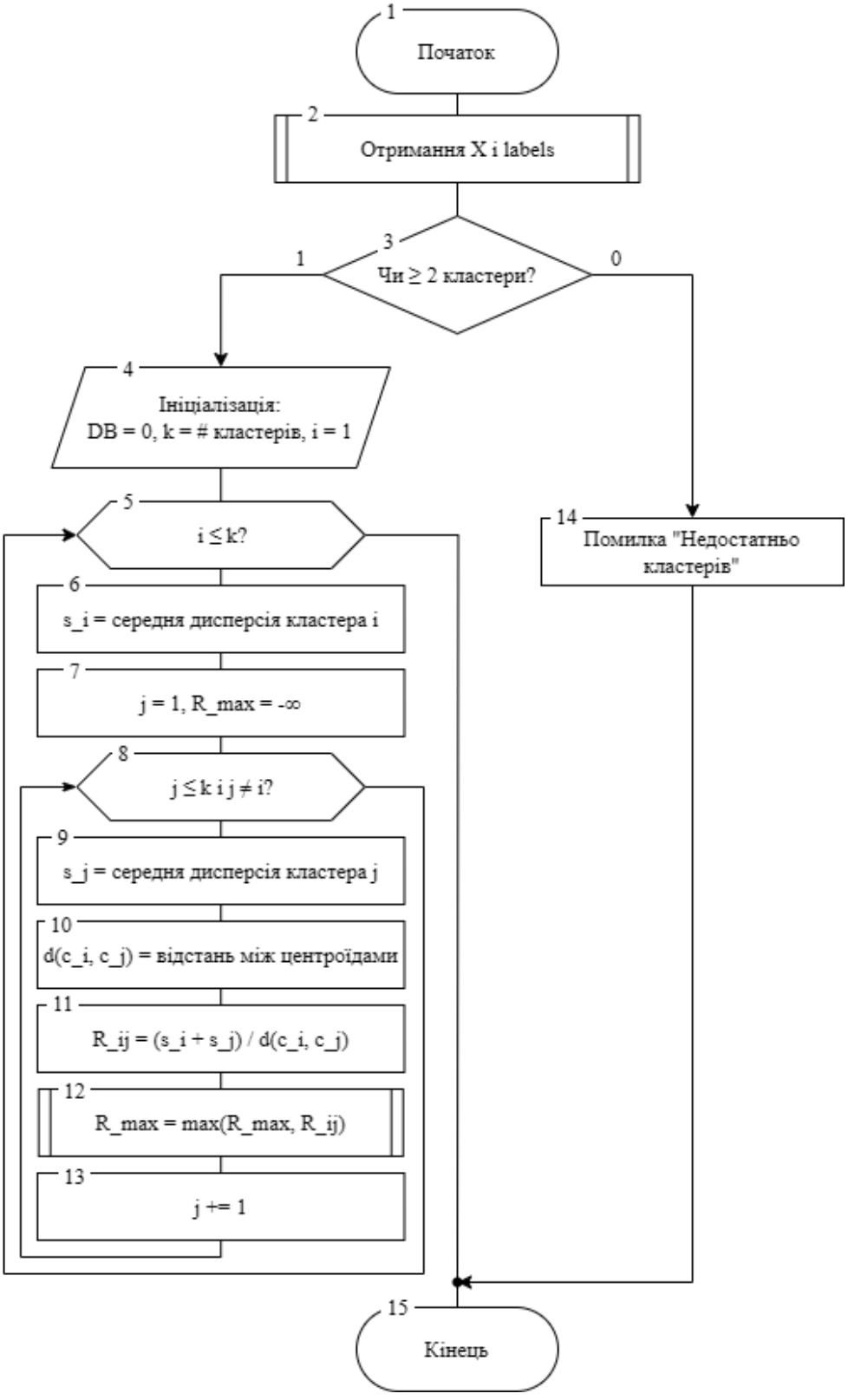


Рисунок 3.3 – Блок-схема методу обрахування індексу Девіса-Болдіна

Далі починається основний цикл по кожному кластеру, який є центральним у процесі. Для поточного кластера обчислюється середня дисперсія його точок, що відображає компактність. Далі запускається цикл по всіх інших кластерах. У

цьому циклі обчислюється дисперсія іншого кластера і відстань між центроїдами двох кластерів. На основі цих значень розраховується відношення суми дисперсій до відстані, яке порівнюється для визначення максимального значення для поточного кластера. Після завершення циклу результат додається до загального індексу з урахуванням кількості кластерів. Процес завершується поверненням середнього значення індексу, що узагальнює якість кластеризації.

Скоригований індекс Ренда оцінює відповідність між отриманими мітками кластерів і відомими істинними мітками, якщо вони доступні. Метрика аналізує пари точок, визначаючи, чи точки, що належать до одного кластера в передбачених мітках, також належать до одного кластера в істинних мітках. Обчислення базується на таблиці сполучень, яка показує кількість точок для кожної комбінації кластерів. Метрика враховує кількість збігів пар і коригує результат, щоб урахувати випадкові збіги, надаючи значення від -1 до 1, де 1 означає ідеальну відповідність. Ця метрика особливо цінна для оцінки алгоритмів кластеризації, коли є еталонні дані, дозволяючи порівнювати результати EM або K-середніх із реальними групами.

Процес розпочинається з отримання передбачених міток кластерів і істинних міток. Першим кроком є перевірка наявності y_true, оскільки метрика неможлива без еталонних даних. Якщо істинні мітки відсутні, процес завершується, повертаючи значення np.nan, що економить ресурси. Якщо y_true є, ініціалізуються змінні для побудови таблиці сполучень, яка є основою обчислень. Таблиця створюється, відображаючи кількість точок для кожної пари кластерів між передбаченими й істинними мітками. Далі обчислюються суми по рядках і стовпцях, які вказують на розміри кластерів у кожному наборі міток. На основі таблиці розраховується кількість пар точок, що збігаються в кластерах, а також очікувана кількість пар за випадкового розподілу. Ці значення використовуються для визначення скоригованого індексу, що враховує випадкові збіги. Процес завершується поверненням результату, який кількісно оцінює відповідність між мітками. Блок-схему роботи методу обрахування скоригованого індексу Ренда зображено на рисунку 3.4.

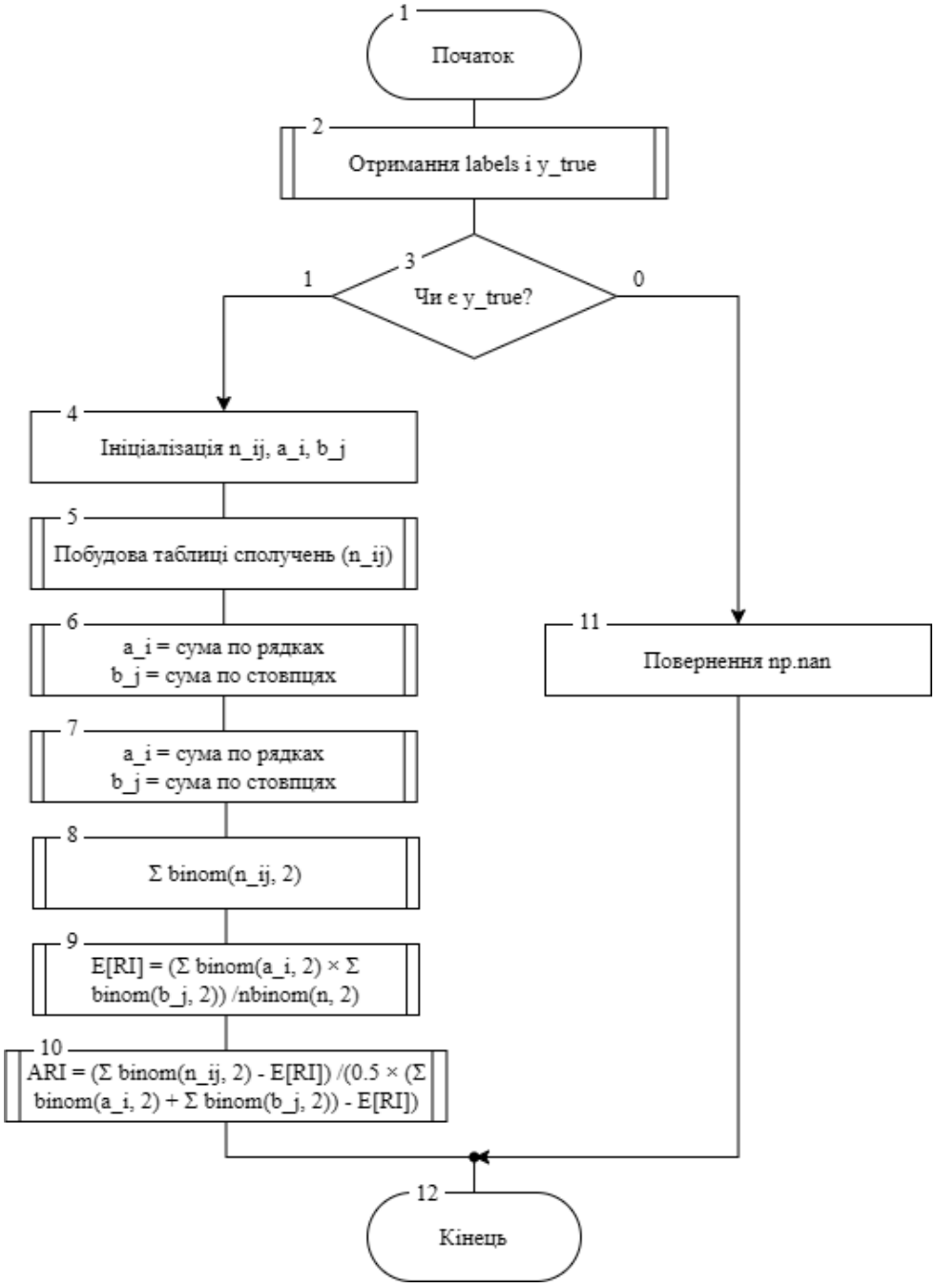


Рисунок 3.4 – Блок-схема методу обрахування скоригованого індексу Ренда

Метод оцінки якості кластеризації забезпечує кількісний аналіз результатів EM-алгоритму та K-середніх, використовуючи коефіцієнт силуету, індекс Девіса-Болдіна і скоригований індекс Ренда. Модульна реалізація сприяє інтеграції з системою, дозволяючи відображати або експортувати результати. Метод підтримує науковий аналіз даних, надаючи користувачам інструменти для оцінки якості кластеризації.

3.3 Розробка модуля експорту результатів кластеризації в PDF

Модуль експорту даних у PDF забезпечує збереження результатів ітерацій EM-алгоритму та підсумкової інформації у форматі PDF. Модуль створює візуалізації кластеризації для двовимірних і тривимірних даних, таблиці метрик для кожної ітерації та підсумкову сторінку з характеристиками алгоритму й оцінками якості. Він інтегрується з графічним інтерфейсом системи, дозволяючи зберігати результати для аналізу чи звітності.

UML-діаграма класу, яка зображена на рисунку 3.5, ілюструє структуру PDFExportModule і його зв’язок із системою. Клас містить атрибути, такі як parent, iteration_history, X, n_features, n_clusters, covar_type, is_pca_applied, dataset_name, em_model, em_runtime, evaluator. Основний метод export_to_pdf виконує експорт, а допоміжні методи забезпечують модульність. Модуль є частиною головного класу інтерфейсу, отримуючи дані після кластеризації.

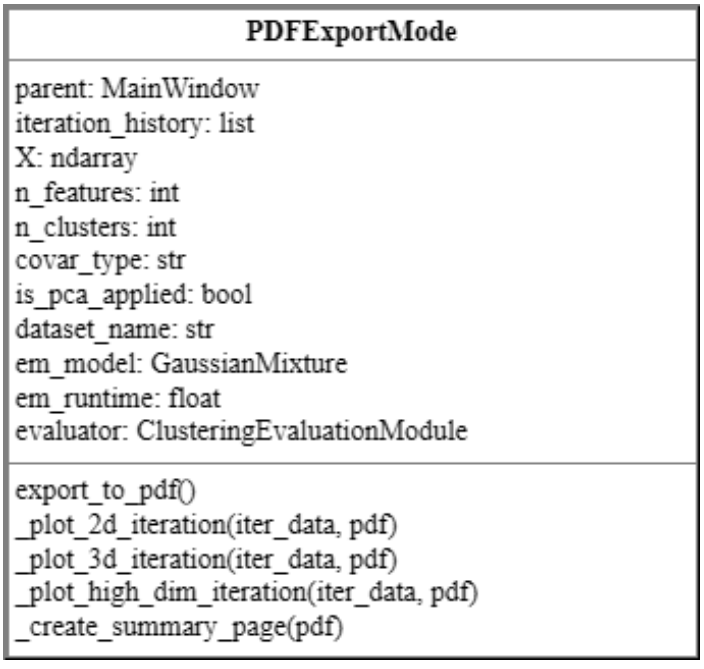


Рисунок 3.5 – Діаграма класу PDFExportModule з відображенням його атрибутів та методів

Блок схема, яка зображена на рисунку 3.6, описує загальний процес експорту. Процес розпочинається з перевірки наявності історії ітерацій. Далі відкривається діалогове вікно для вибору шляху збереження файлу. Якщо шлях

не обрано, процес припиняється. При успішному виборі створюється PDF-файл. Починається цикл по всіх ітераціях, де для кожної ітерації генерується сторінка залежно від кількості ознак: двовимірна або тривимірна візуалізація чи текст для високовимірних даних. Кожна сторінка включає таблицю метрик. Після обробки ітерацій створюється підсумкова сторінка з узагальненими результатами. PDF-файл зберігається, а графічні об’єкти очищаються.

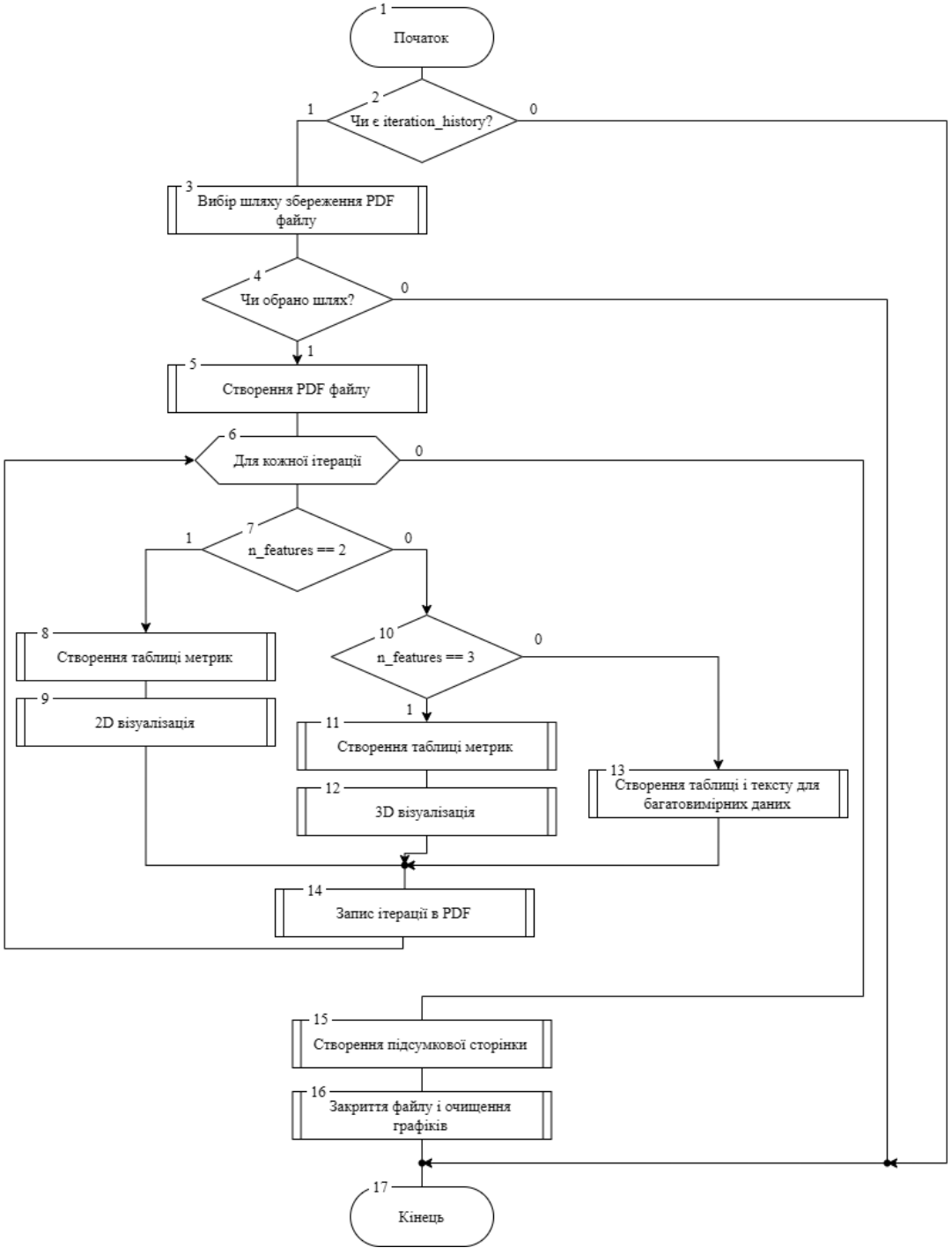


Рисунок 3.6 – Блок-схема роботи модуля PDFExportModule

Двовимірна візуалізація є важливим інструментом для аналізу результатів кластеризації, оскільки дозволяє інтуїтивно оцінити розподіл точок і кластерів у просторі. Вона використовує точкові графіки, де кожна точка позначається кольором відповідно до мітки кластера. Для відображення форми кластерів використовуються еліпси, які базуються на коваріаційних матрицях, що показують дисперсію даних у кластері. Еліпси масштабуються для різних рівнів, забезпечуючи детальне уявлення про структуру кластера. На рисунку 3.7 зображено блок-схему роботи алгоритму експорту даних двовимірної візуалізації.

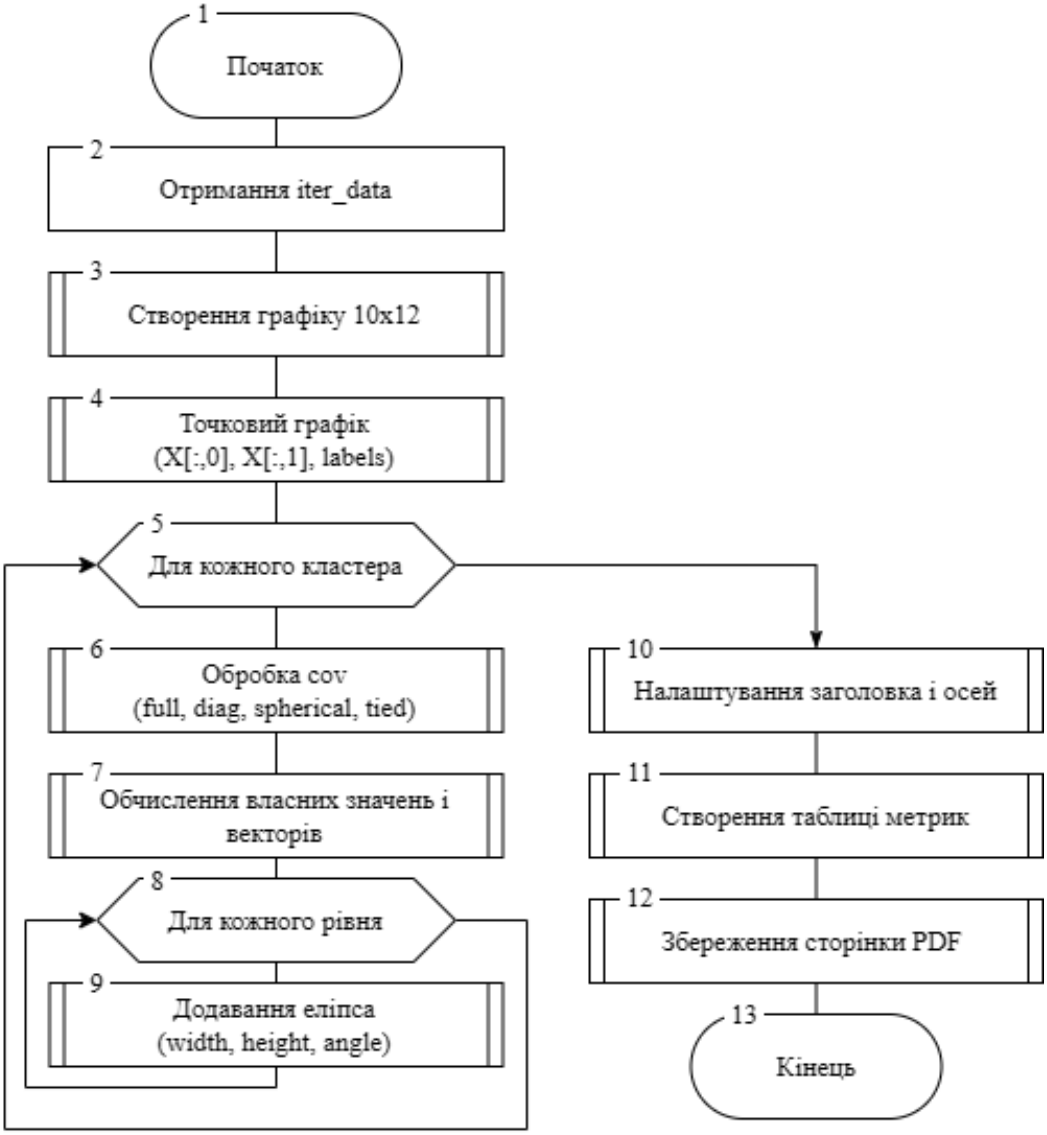


Рисунок 3.7 – Блок-схема роботи алгоритму експорту даних двовимірної візуалізації.

Процес починається з отримання даних ітерації, включаючи мітки кластерів, центри та коваріації. Потім створюється графік розміром 10×12 дюймів із двома підграфіками, верхній для візуалізації, нижній для таблиці метрик. На верхньому графіку точки даних відображаються як точковий графік із кольорами, що відповідають кластерам, використовуючи палітру viridis [18]. Центри кластерів позначаються червоними маркерами «X». Для кожного кластера обробляється коваріаційна матриця залежно від типів – повна, діагональна, сферична чи зв’язана.

Коваріація аналізується для визначення власних значень і векторів, які задають розміри й орієнтацію еліпсів. Еліпси додаються на графік із різними рівнями прозорості, що відображають різні масштаби дисперсії. Заголовок графіка вказує номер ітерації, а осі позначаються залежно від застосування PCA. Таблиця таких метрик, як логарифмічна правдоподібність, коефіцієнт силуету, скоригований індекс Ренда та конвергенція додається на нижній підграфік, форматується для читабельності, після чого сторінка зберігається у PDF. Цикл обробки еліпсів є важливим, оскільки він адаптується до типу коваріації, забезпечуючи точне відображення.

Тривимірна візуалізація розширює можливості аналізу кластеризації, дозволяючи оцінити розподіл даних у тривимірному просторі. Вона використовує тривимірні точкові графіки, де точки позначаються кольорами за мітками кластерів, а центри кластерів відображаються окремими маркерами. Для відображення форми кластерів додаються еліпсоїди, які базуються на коваріаційних матрицях, що описують дисперсію даних у трьох вимірах. Еліпсоїди масштабуються для різних рівнів, що дозволяє візуалізувати щільність і форму кластерів. Такий підхід є цінним для тривимірних наборів даних, де двовимірні проекції можуть втрачати інформацію, і допомагає оцінити якість кластеризації EM-алгоритму. Блок-схему роботи алгоритму візуалізації та експорту даних тривимірної візуалізації зображено на рисунку 3.8.

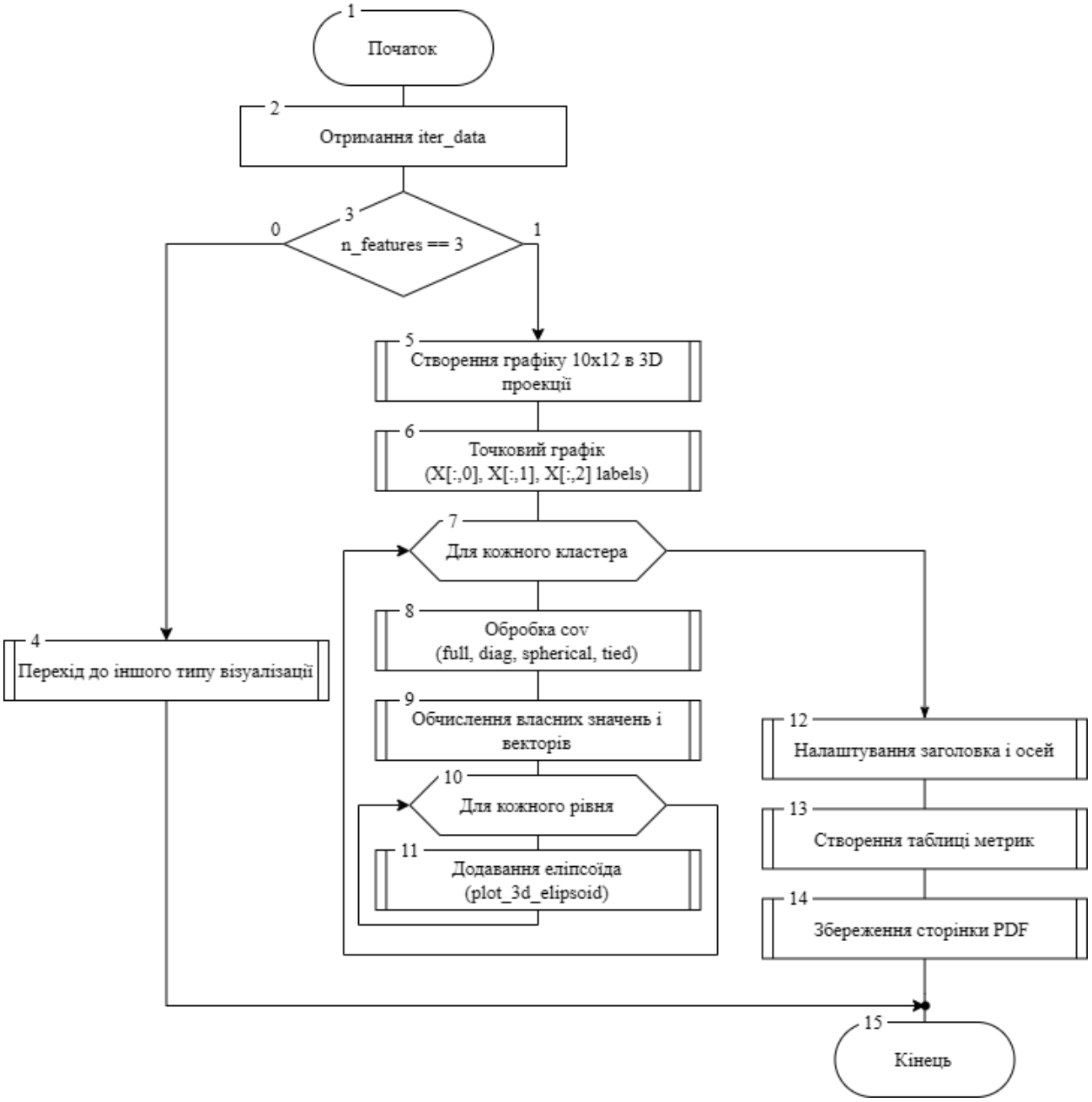


Рисунок 3.8 – Блок-схема роботи алгоритму експорту даних тривимірної візуалізації.

Процес розпочинається з отримання даних ітерації, включаючи мітки, центри й коваріації. За виконання умови створюється графік із двома підграфіками: верхній для тривимірного графіка, нижній для таблиці. Тривимірний графік ініціалізується з проекцією 3D, на якому точки даних відображаються як точковий графік із кольорами за кластерами. Центри кластерів позначаються червоними маркерами «X». Для кожного кластера

обробляється коваріаційна матриця залежно від типу коваріації. Коваріація використовується для генерації еліпсоїдів, які додаються на графік із різними рівнями прозорості, що відображають масштаби дисперсії. Цикл обробки еліпсоїдів є найважливішим, оскільки він вимагає спеціалізованої функції для тривимірного рендерингу, що підвищує обчислювальну складність. Заголовок графіка вказує номер ітерації, а осі позначаються як 3 ознаки. Таблиця метрик додається на нижній підграфік, включаючи логарифмічну правдоподібність, коефіцієнт силуету, скоригований індекс Ренда та статус конвергенції. Сторінка зберігається у PDF, а графік очищається.

Для високовимірних даних пряма візуалізація у вигляді графіків є неможливою через обмеження графічного представлення. Замість цього використовується текстовий підхід, де інформація про ітерацію відображається у структурованому вигляді. Таблиці метрик стають основним інструментом, надаючи кількісні показники, такі як логарифмічна правдоподібність, коефіцієнт силуету та скоригований індекс Ренда. Цей підхід дозволяє аналізувати результати кластеризації, коли візуальні методи непридатні.

Процес починається з отримання даних ітерації. Перевіряється, чи кількість ознак перевищує три. За виконання умови створюється графік із двома підграфіками. Верхній підграфік містить текстовий напис, що вказує номер ітерації та зазначає, що дані є багатовимірними, забезпечуючи чітке повідомлення для користувача. Цей блок є важливим, оскільки замінює графічну візуалізацію текстовою інформацією, зберігаючи інформативність. Нижній підграфік містить таблицю метрик, яка включає логарифмічну правдоподібність, коефіцієнт силуету, скоригований індекс Ренда і статус конвергенції. Після створення таблиці сторінка зберігається у PDF-файл, а графік очищається. Відсутність циклів у процесі відображає його простоту порівняно з 2D/3D-візуалізаціями, але перевірка розмірності залишається важливою для правильного вибору обробки. Блок-схему роботи алгоритму експорту результатів кластеризації багатовимірних даних зображено на рисунку 3.9.

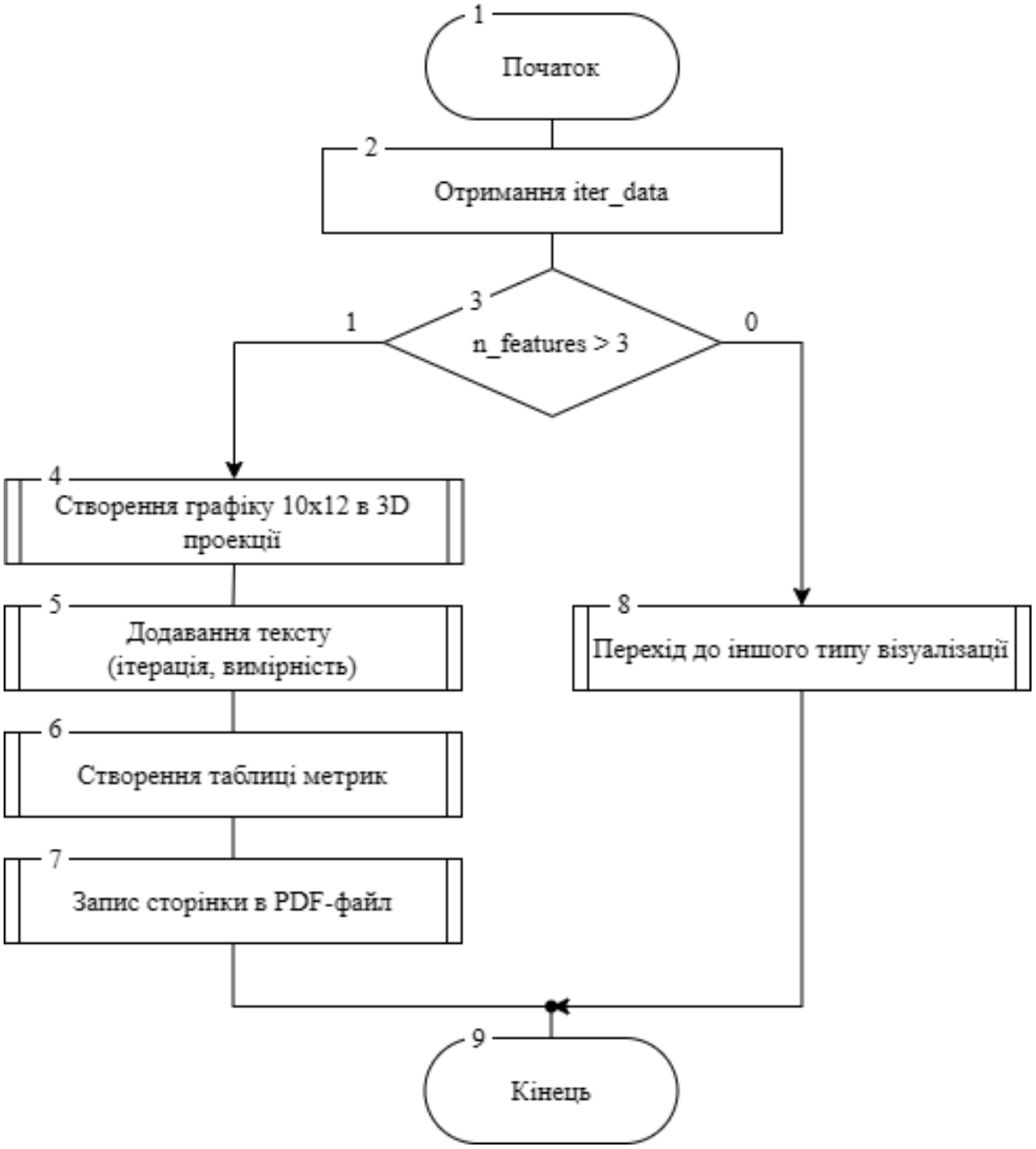


Рисунок 3.9 – Блок-схема роботи алгоритму експорту багатовимірних даних

Модуль експорту даних у PDF забезпечує структуроване збереження результатів ітерацій EM-алгоритму, включаючи візуалізації, таблиці метрик і підсумкову інформацію. Модульна реалізація сприяє інтеграції з системою, дозволяючи користувачам створювати звіти для аналізу чи презентацій. Метод підтримує науковий аналіз даних, надаючи зручний інструмент для документування результатів кластеризації.

3.4 Висновки

У третьому розділі було проведено аналіз і обґрунтовано вибір інструментів для створення системи аналізу даних із підтримкою кластеризації.

Було обрано Python, scikit-learn для алгоритмів кластеризації, matplotlib для візуалізації та PyQt5 для інтерфейсу користувача. Ці інструменти забезпечують продуктивність обробки даних і підтримують аналіз у наукових дослідженнях. Розроблено специфікацію системи для виконання кластеризації та відображення результатів.

Модуль «ClusteringEvaluationModule» реалізовано для аналізу якості кластеризації EM і K-середніх. Він обчислює коефіцієнт силуету, індекс Девіса-Болдіна та скоригований індекс Ренда, забезпечуючи оцінку результатів. Модуль інтегрується з PyQt5 для відображення даних, сприяючи перевірці надійності алгоритмів.

Модуль «PDFExportModule» створено для збереження ітерацій EM-алгоритму в PDF, включаючи візуалізації для двовимірних і тривимірних даних, таблиці метрик і підсумкову інформацію. Він адаптується до розмірності даних і працює через PyQt5. Модуль забезпечує документування результатів, що є важливим для подальшого аналізу результатів кластеризації.

4 . ТЕСТУВАННЯ СИСТЕМИ

4.1 Функціональне тестування системи

Тестування програмного забезпечення є важливим етапом розробки, спрямованим на перевірку відповідності системи заданим вимогам і забезпечення її надійності. Для системи аналізу даних із підтримкою кластеризації застосовуються принципи функціонального тестування. Функціональне тестування перевіряє окремі компоненти, такі як введення даних, виконання алгоритмів кластеризації та обчислення метрик якості, на коректність виконання їхніх завдань. Тестування охоплює як позитивні сценарії, що підтверджують правильну обробку даних, так і негативні, що виявляють поведінку системи в умовах помилок. Це дозволяє гарантувати надійність і точність роботи системи.

Теоретична основа тестування системи кластеризації базується на принципах валідації алгоритмів машинного навчання. Перевірка включає коректність роботи алгоритмів EM і K-середніх та точність обчислень метрик. Тестування також охоплює візуалізацію результатів, наприклад, точкові графіки для двовимірних і тривимірних даних та експорт у PDF, що вимагає перевірки форматування звітів. Негативні сценарії тестування допомагають виявити межі системи, забезпечуючи чіткі повідомлення про помилки через графічний інтерфейс.

Першим етапом тестування було моделювання ситуації, коли користувач намагається запустити кластеризацію даних з набором, що містить менше двох зразків. У системі змінна «X» не ініціалізована або має недостатньо даних. Алгоритм не може виконати кластеризацію, тому модуль аналізу надсилає повідомлення «No dataset loaded. Please configure a dataset». Після цього на екрані з’являється вікно з повідомленням про помилку «Для кластеризації необхідно обрати принаймні дві числові ознаки, щоб забезпечити коректне групування даних». Вікно з помилкою зображено на рисунку 4.1.

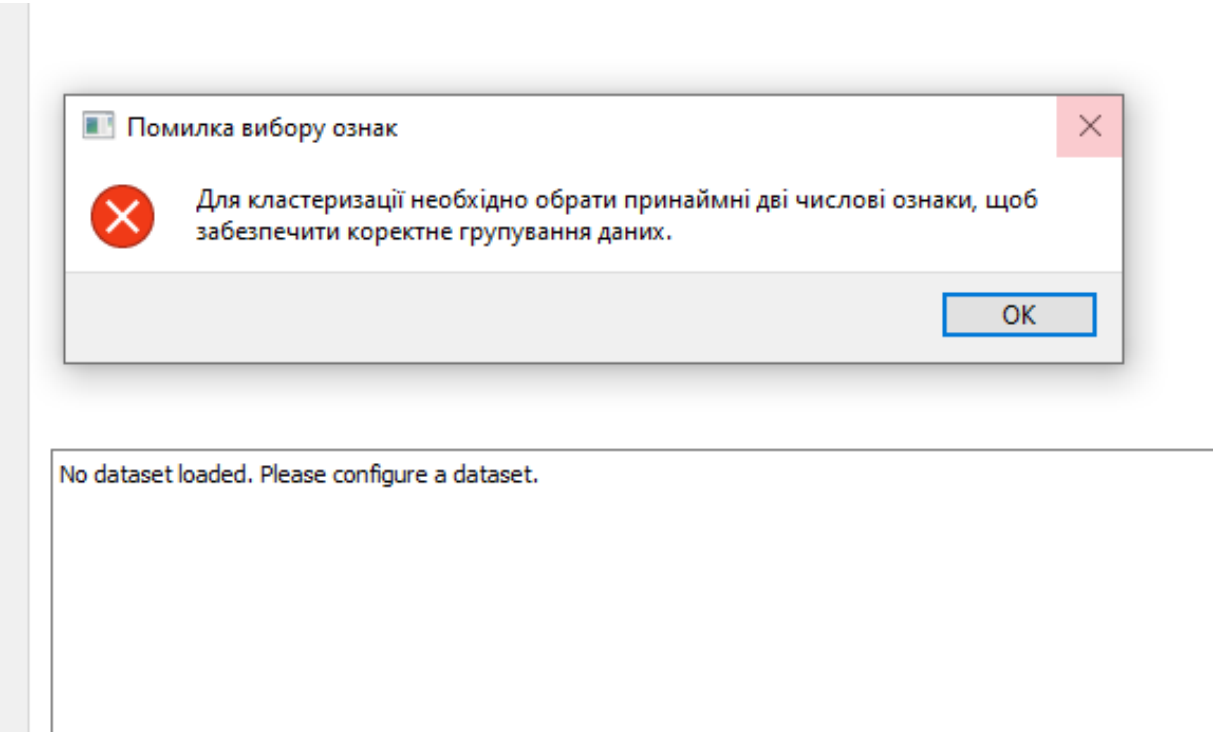


Рисунок 4.1 – Зображення вікна з повідомленням про помилку імпортування даних

Наступним етапом було тестування випадку, коли алгоритм EM повертає лише один кластер через низьку варіативність даних. Модуль ClusteringEvaluationModule виявляє, що мітки містять лише одну унікальну мітку, що унеможливлює оцінку метрик. Система відображає спливаюче вікно з повідомленням: «Для кластеризації необхідно обрати принаймні два кластери, щоб забезпечити змістовне групування та оцінку якості». Зображення вікна з помилкою зображено на рисунку 4.2.

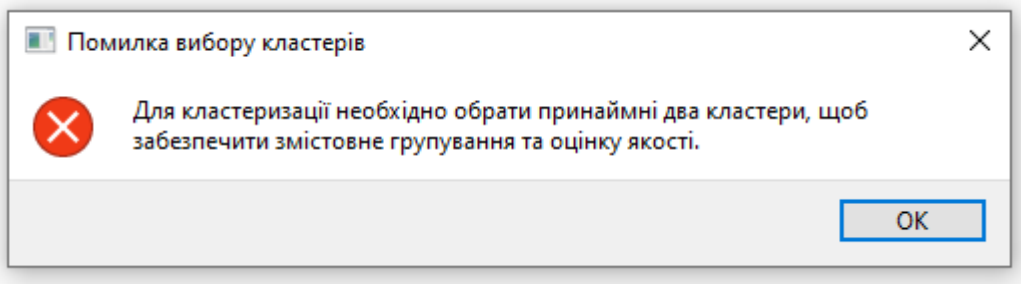


Рисунок 4.2 – Зображення вікна з повідомленням про помилку генерації кластерів та оцінювання якості кластеризації

Далі було проведено тестування якості оцінювання кластеризації та порівняння результатів роботи Ем-алгоритму і алгоритму К-середніх. Для цього було згенеровано дані за типом «Плям» та обрано 3 кластери. Після закінчення роботи алгоритмів було згенеровано порівняльну характеристику результатів, яку зображено на рисунку 4.3. За результатами оцінювання та порівняння робота ЕМ-алгоритму за загальним коефіцієнтом якості оцінилася в 82%, в той час як результат роботи алгоритму К-середніх оцінився в 68%

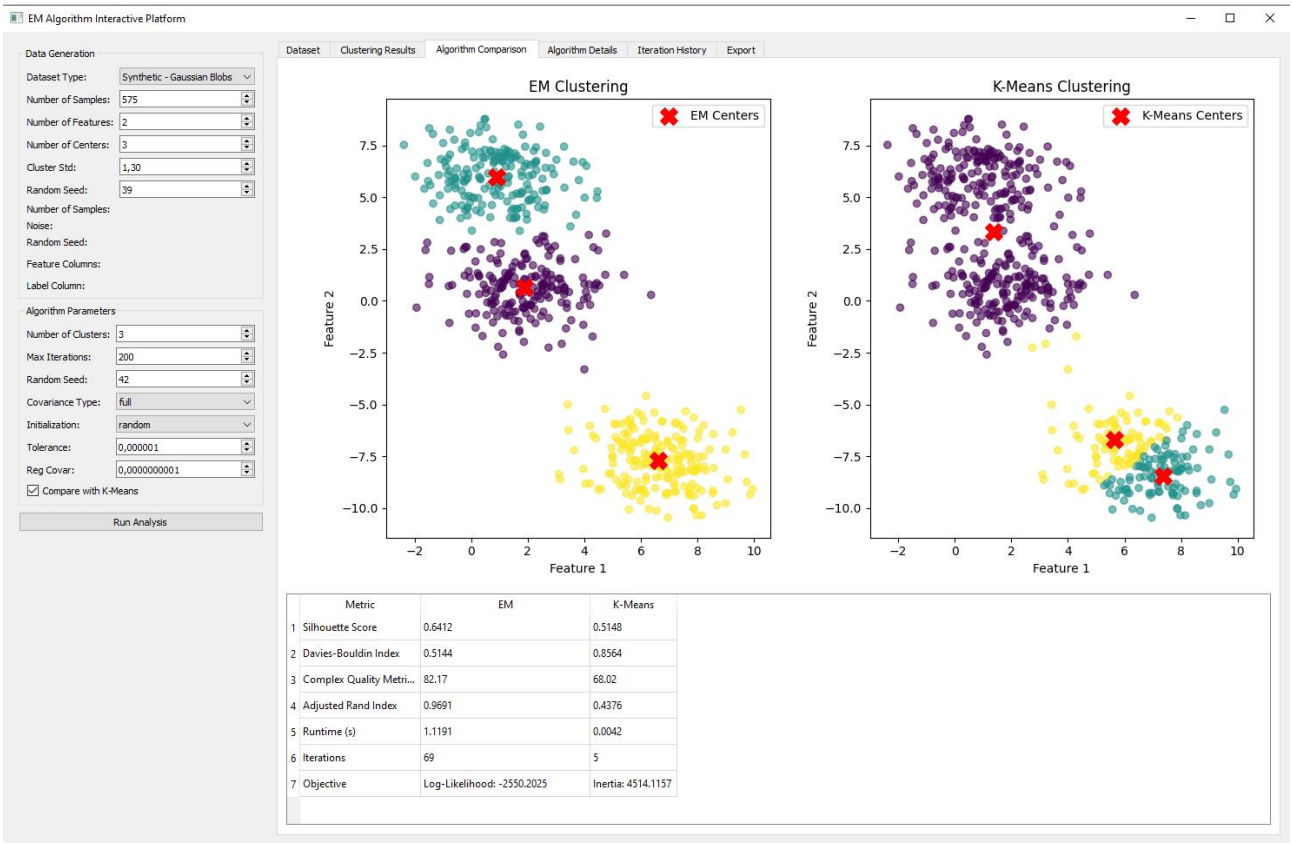


Рисунок 4.3 – Результат порівняння роботи алгоритмів та оцінювання якості кластеризації

Четвертим етапом функціонального тестування була перевірка процесу роботи ЕМ-алгоритму. Для цього після завершення роботи алгоритму було обрано випадкову ітерацію для перегляду. На рисунку 4.4 зображено 36 ітерацію кластеризації даних з проміжними параметрами роботи алгоритму та статусом. Також відповідними еліпсами відображено обрахування коваріацій відповідних даних в процесі роботи алгоритму.

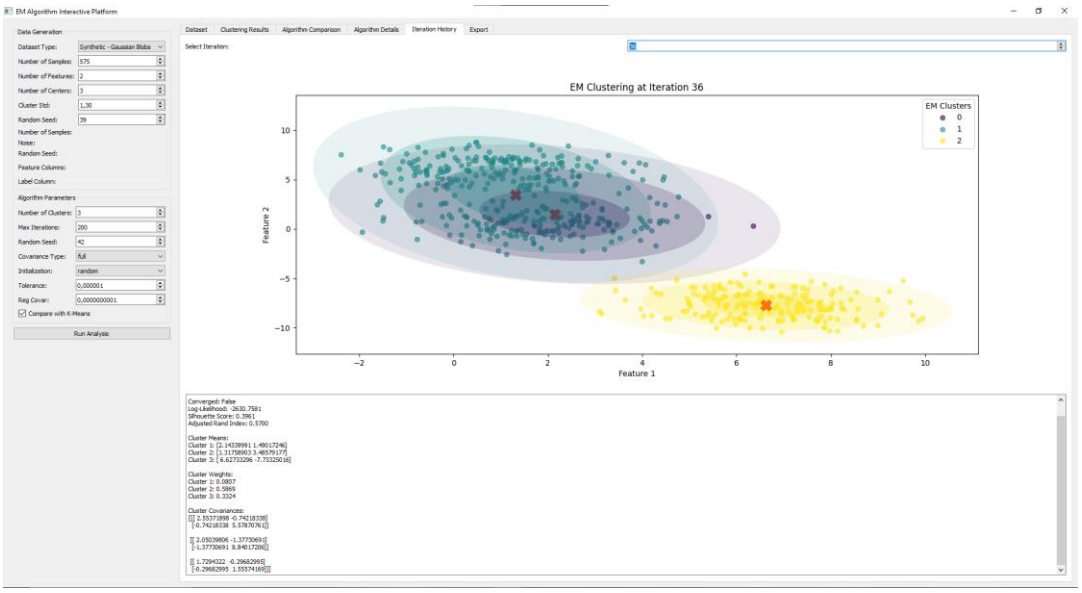
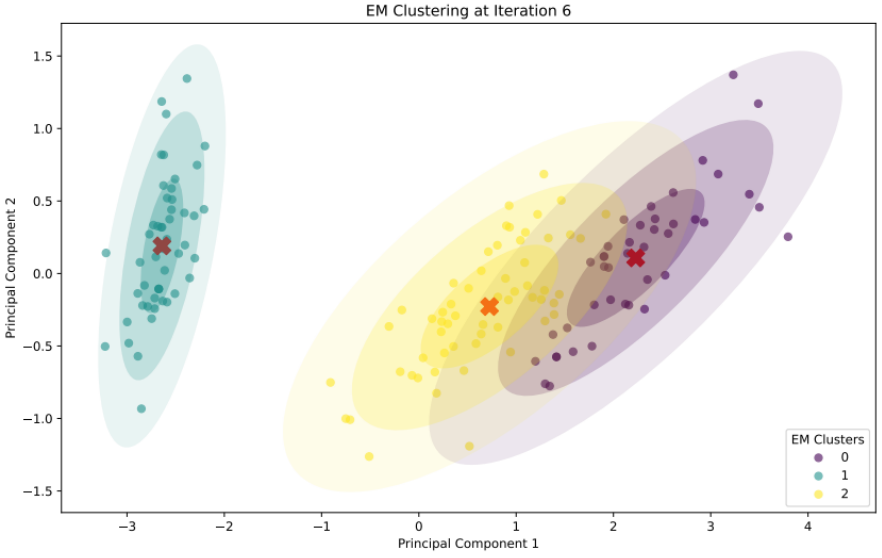


Рисунок 4.4 – Вікно перегляду проміжного результату роботи EM-алгоритму з відповідними проміжними даними

Останнім етапом тестування системи буде перевірка модуля експорту результатів кластеризації в PDF-файл. На вкладці експорту було натиснуто на кнопку «Export iteration PDF». Після цього було відкрито згенерований файл. На рисунку 4.5 зображено контент 6 сторінки PDA-файлу.



Metric	Value
Log-Likelihood	-284.8037
Silhouette Score	0.5439
Adjusted Rand Index	N/A
Converged	False

Рисунок 4.5 – Відображення 6 ітерації в експортованому PDF-файлі

Тестування показало, що система коректно обробляє дані, обчислюючи метрики та створюючи візуалізації для наборів даних. Помилки відображаються через спливаючі вікна з повідомленнями про їх причини. Експорт у PDF створює структуровані звіти без збоїв. Тестування підтвердило функціональність системи у виконанні кластеризації, оцінці якості та експорті результатів. Сценарії з помилками показали належну обробку помилкових даних, а позитивні – точність метрик, візуалізацій і PDF-звітів. Система є надійним інструментом, що забезпечує аналіз і документування результатів кластеризації.

4.2 Розробка інструкції користувача

Програма дозволяє користувачам генерувати синтетичні набори даних, завантажувати CSV-файли, налаштовувати параметри алгоритму, порівнювати результати з K-Means, а також візуалізувати та експортувати результати. Платформа створена з акцентом на наукову точність, інтуїтивно зрозумілий інтерфейс та підтримку української мови, що робить її доступною для дослідників, студентів та фахівців у галузі аналізу даних.

Програма підтримує кілька типів даних: синтетичні набори та користувацькі CSV-файли з числовими ознаками. Користувачі можуть застосовувати PCA для зниження розмірності даних, оцінювати якість кластеризації за допомогою метрик, а також аналізувати ітераційний процес EM-алгоритму. Візуалізації включають графіки кластерів, коваріаційні еліпси, ймовірності належності та логарифмічну правдоподібність. Експорт результатів доступний у форматах CSV та PDF.

Робота з програмою починається з вибору типу набору даних у відповідному розділі інтерфейсу. Для синтетичних даних користувач налаштовує параметри, такі як кількість зразків, ознак, центрів кластерів або рівень шуму, залежно від обраного типу даних. При завантаженні файлу CSV необхідно обрати числові ознаки та стовпець із мітками класів. Після підготовки даних користувач переходить до налаштування параметрів алгоритму, де визначає кількість кластерів, максимальну кількість ітерацій, тип коваріаційної

матриці, метод ініціалізації та інші характеристики EM-алгоритму. Додатково можна активувати порівняння з алгоритмом K-Means для оцінки відмінностей у результатах. Після завершення налаштувань запуск аналізу ініціює процес кластеризації, а результати відображаються у кількох вкладках інтерфейсу. У першій вкладці представлено візуалізацію вихідного набору даних у двовимірному або тривимірному форматі залежно від кількості ознак. Друга вкладка містить графіки кластерів, еліпси або еліпсоїди впевненості, ймовірності приналежності, щільності гаусівських розподілів, а також значення логарифмічної правдоподібності та параметри моделі. Третя вкладка дозволяє порівняти результати EM-алгоритму та K-Means за метриками якості й часом виконання. Четверта вкладка забезпечує аналіз кожної ітерації EM-алгоритму з відповідними візуалізаціями та статистикою. Остання вкладка призначена для експорту результатів. В таблиці 4.1 наведено мінімальні та рекомендовані системні вимоги.

Таблиця 4.1 – Системні вимоги

Конфігурація	Мінімальні системні вимоги	Рекомендовані системні вимоги
Операційна система	Windows 10/11, macOS 10.15, Linux	Windows 10/11, macOS 10.15, Linux
Процесор	2-ядерний, 2.0 ГГц	4-ядерний, 2.4 ГГц
Оперативна пам’ять	4 ГБ	8 ГБ
Простір на диску	100 МБ	200 МБ
Відеокарта	NVIDIA GeForce GTX 1050	NVIDIA GeForce GTX 3060
Монітор	Роздільна здатність 1366x768	Роздільна здатність 1920x1080

Інтерактивна платформа для дослідження властивостей EM-алгоритму є потужним інструментом для аналізу даних, який поєднує гнучкість налаштувань,

детальну візуалізацію та комплексну оцінку якості кластеризації. Завдяки підтримці синтетичних і користувацьких даних, а також інтеграції комплексної метрики якості, програма підходить як для навчальних, так і для дослідницьких цілей. Інтуїтивний інтерфейс та експорт результатів у різних форматах забезпечують зручність використання. Для оптимальної роботи рекомендується забезпечити відповідність системним вимогам. Платформа може бути розширена додатковими функціями, такими як підтримка інших алгоритмів кластеризації або розширена обробка даних.

4.3 Висновки

Було виконано функціональне тестування інтерактивної платформи EM-алгоритму, що підтвердило її відповідність технічним вимогам. Здійснено перевірку всіх ключових модулів, включаючи генерацію синтетичних даних, обробку CSV-файлів, кластеризацію та оцінку якості. Створено надійний інструмент, придатний для навчальних і дослідницьких цілей у галузі аналізу даних. Було розроблено детальну інструкцію користувача, яка забезпечує чітке розуміння роботи з платформою для користувачів із різним рівнем підготовки. Складено таблицю з системними вимогами.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було розроблено методи та програмні засоби для інтерактивної візуалізації та експериментального дослідження EM-алгоритму.

Проведено аналіз існуючих методів і засобів візуалізації та експериментального дослідження EM-алгоритму. Обґрунтовано необхідність розробки методів підвищення інтерактивності та наочності візуалізації EM-алгоритму з метою полегшення його вивчення та дослідження властивостей.

Подальшого розвитку отримав метод інтерактивної візуалізації EM-алгоритму, особливість якого полягає в забезпеченні динамічного відображення проміжних результатів кожної ітерації з можливістю зміни параметрів у реальному часі, що дозволяє розглядати більше можливих варіантів без потреби повторного запуску алгоритму.

Подальшого розвитку отримав метод оцінки якості кластеризації, особливість якого полягає у використанні комбінації зовнішніх та внутрішніх метрик, що дозволяє різнобічно аналізувати якість кластеризації.

Подальшого розвитку отримав метод генерації синтетичних даних для тестування EM-алгоритму, особливість якого полягає у використанні параметричних моделей з можливістю інтерактивного налаштування, що дозволяє розширити спектр експериментальних сценаріїв.

Практичне значення отриманих результатів полягає в тому, що на основі теоретичних досліджень розроблено алгоритми та програмну платформу для інтерактивного дослідження властивостей EM-алгоритму з використанням бібліотеки scikit-learn.

Розроблено методи підвищення інтерактивності візуалізації процесу роботи EM-алгоритму та методи підвищення наочності й інформативності представлення результатів його роботи, що дозволяє з максимальною наочністю та інтерактивністю аналізувати та вивчати особливості алгоритму.

На основі проведених у БКР досліджень розроблено програмні компоненти

та інтерактивну експериментальну платформу. Проведене тестування підтвердило достовірність теоретичних досліджень методів візуалізації та засобів їх реалізації.

Розроблені засоби візуалізації та аналізу EM-алгоритму можна використати в освітній галузі для підвищення інтерактивності та наочності вивчення алгоритмів машинного навчання, а також у дослідницьких цілях для глибокого аналізу властивостей EM-алгоритму.

Ключові слова: машинне навчання, EM-алгоритм, кластеризація, інтерактивна візуалізація, scikit-learn, експериментальна платформа.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Liu S., Zhou Z.-H. Machine Learning. Singapore : Springer, 2021 – 578 p.

2. Madala H. R. Inductive Learning Algorithms for Complex Systems Modeling. Taylor & Francis Group, 2019. 380 p.

3. Krishnan T., McLachlan G. J. EM Algorithm and Extensions. Wiley & Sons, Incorporated, John, 2007. 384 p.

4. Фоменко. Д. С. ЕМ-алгоритм як метод оптимізації у задачах кластеризації. Матеріали LIV Всеукраїнської науково-технічної конференції професорсько-викладацького складу, науковців, аспірантів та студентів підрозділів університету з участю працівників підприємств (НТКП ВНТУ-2025). Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/ paper/view/24503/ (дата звернення: 15.04.2025)

5. Aggarwal C. C., Reddy C. K. Data Clustering: Algorithms and Applications. Taylor & Francis Group, 2018. 652 p.

6. Charles L. Byrne. The EM Algorithm: Theory, Applications and Related Methods. Massachusetts: University of Massachusetts Lowell, 2017. – 110 p.

7. ELKI Data Mining Framework. URL: https://elki-project.github.io/ (date of access: 18.04.2025).

8. Home. WEKA. URL: https://www.weka.io/ (date of access: 22.04.2025).

9. MATLAB. MathWorks - Maker of MATLAB and Simulink - MATLAB & Simulink. URL: https://www.mathworks.com/products/matlab.html (date of access: 26.04.2025).

10. Open for Innovation | KNIME. KNIME. URL: https://www. knime.com/ (date of access: 30.04.2025).

11. CSV File Reading and Writing. Python documentation. URL: https://docs.python.org/uk/3.13/library/csv.html (date of access: 02.05.2025).

12. Microsoft. Visual Studio Code - Code Editing. Redefined. URL: https://code.visualstudio.com/ (date of access: 04.05.2025).

13. Python Clustering. URL: https://scikit-learn.org/stable/modules/clustering.html (date of access: 06.05.2025).

14. NumPy. URL: https://numpy.org/ (date of access: 10.05.2025).

15. Pandas - Python Data Analysis Library. URL: https://pandas.pydata.org/ (date of access: 17.00.2025).

16. PyQt5. URL: https://pypi.org/project/PyQt5/ (date of access: 24.05.2025).

17. What is Unified Modeling Language (UML)?. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ (date of access: 26.05.2025).

18. Introduction to the viridis color maps. The Comprehensive R Archive Network. URL: https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html (date of access: 27.05.2025).

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
"21" березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка інтерактивної експериментальної платформи для досліджень властивостей EM-алгоритму з використанням бібліотеки scikit-learn» за спеціальністю 121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ к.т.н., доцент Д.І. Катєльніков
"21" 03 2025 р.
Виконав:
_____ студент гр. 4ПІ-21б Д. С. Фоменко
"21" 03 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка інтерактивної експериментальної платформи для досліджень властивостей EM-алгоритму з використанням бібліотеки scikit-learn».

Галузь застосування – освітні та наукові дослідження в галузі машинного навчання.

2. Підстава для розробки.

Підставою для розробки бакалаврської кваліфікаційної роботи є індивідуальне завдання на БКР та наказ від 20 березня 2025 р. №97 ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є підвищення продуктивності засобів формування тривимірних зображень за рахунок зменшення обчислювальної складності моделей формування світлових ефектів, а також адаптивного вибору методів зафарбовування.

Призначення роботи – розробка методів і засобів для вивчення EM-алгоритму і дослідження його властивостей.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Liu S., Zhou Z.-H. Machine Learning. Singapore: Springer, 2021 – 578 p.

2. Madala H. R. Inductive Learning Algorithms for Complex Systems Modeling. Taylor & Francis Group, 2019. 380 p.

3. Krishnan T., McLachlan G. J. EM Algorithm and Extensions. Wiley & Sons, Incorporated, John, 2007. 384 p.

5. Технічні вимоги

Методи кластеризації – EM-алгоритм, K-Means; набори даних – Gaussian Blobs, Moons, Circles, CSV-файли; параметри алгоритму – комплексна метрика якості; розмірність даних – від 2 до 12 ознак; методи візуалізації – Matplotlib; програмне середовище – Python, PyQt5; середовище розробки – Visual Studio Code; вихідні результати – кластеризовані дані з візуалізацією еліпсів коваріації, значення метрики якості; вхідні дані: вибір типу набору даних, налаштування параметрів кластеризації, вибір метрик якості.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

- пояснювальна записка до БКР;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану систем автоматизації та кластеризації даних	25.03.25 – 07.04.25
2	Розробка підсистеми генерації синтетичних даних	08.04.25 – 20.04.25
3	Розробка підсистеми інтерактивної візуалізації ЕМ-алгоритму	21.04.25 – 03.05.25
4	Розробка модуля оцінки якості кластеризації	04.04.25 – 15.05.025
5	Тестування роботи застосунку	16.05.25 – 24.05.25
6	Оформлення матеріалів до захисту БКР	25.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка інтерактивної експериментальної платформи для досліджень властивостей ЕМ-алгоритму з використанням бібліотеки scikit-learn

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, 4ПІ-21Б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 7,6 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) Катєльніков Д.І., ктн., доц. каф. ПЗ (прізвище, ініціали, посада)

Здобувач (підпис) Фоменко Д. С. (прізвище, ініціали)

Додаток В – Лістинг програми

```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_pdf import PdfPages
from matplotlib.figure import Figure
import seaborn as sns
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, adjusted_rand_score, davies_bouldin_score
from sklearn.datasets import make_blobs, make_moons, make_circles
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
    QTabWidget, QGroupBox, QFormLayout, QComboBox,
    QPushButton, QFileDialog, QCheckBox, QLabel, QSpinBox,
    QDoubleSpinBox, QTableWidget, QTableWidgetItem, QTextEdit,
    QListWidget, QListWidgetItem, QMessageBox
)
from PyQt5.QtCore import Qt
import io
import os
from matplotlib.patches import Ellipse
import time
from scipy.stats import norm
from mpl_toolkits.mplot3d import Axes3D
from scipy import linalg


class DataImportModule:
    def __init__(self, parent):
        self.parent = parent
        self.df = None
        self.X = None
        self.X_original = None
        self.y_true = None
        self.n_features = 0
        self.pca = None
        self.is_pca_applied = False

    def upload_csv(self):
        file_path, _ = QFileDialog.getOpenFileName(self.parent, "Open CSV File", "", "CSV Files (*.csv)")
        if file_path:
            try:
                self.df = pd.read_csv(file_path)
                self.parent.feature_cols.clear()
                numeric_cols = self.df.select_dtypes(include=[np.number]).columns.tolist()
                for col in numeric_cols:
                    item = QListWidgetItem(col)
                    item.setFlags(item.flags() | Qt.ItemIsSelectable)
                    self.parent.feature_cols.addItem(item)
```

```
            self.parent.feature_cols.setEnabled(True)
            self.parent.label_col.clear()
            self.parent.label_col.addItems(["None"] + list(self.df.columns))
            self.parent.update_ui()
        except Exception as e:
            self.parent.data_stats.setText(f"Error loading file: {str(e)}")

    def generate_dataset(self):
        dataset_type = self.parent.dataset_combo.currentText()
        self.X = None
        self.X_original = None
        self.y_true = None
        self.n_features = 0
        self.pca = None
        self.is_pca_applied = False

        if dataset_type == "Synthetic - Gaussian Blobs":
            self.X, self.y_true = make_blobs(
                n_samples=self.parent.blobs_samples.value(),
                n_features=self.parent.blobs_features.value(),
                centers=self.parent.blobs_centers.value(),
                cluster_std=self.parent.blobs_std.value(),
                random_state=self.parent.blobs_seed.value()
            )
            self.n_features = self.parent.blobs_features.value()

        elif dataset_type == "Synthetic - Moons":
            self.X, self.y_true = make_moons(
                n_samples=self.parent.moons_samples.value(),
                noise=self.parent.moons_noise.value(),
                random_state=self.parent.moons_seed.value()
            )
            self.n_features = 2

        elif dataset_type == "Synthetic - Circles":
            self.X, self.y_true = make_circles(
                n_samples=self.parent.moons_samples.value(),
                noise=self.parent.moons_noise.value(),
                random_state=self.parent.moons_seed.value()
            )
            self.n_features = 2

        elif dataset_type == "Upload Custom CSV" and self.df is not None:
            selected_features = [self.parent.feature_cols.item(i).text() for i in
range(self.parent.feature_cols.count())
                                 if self.parent.feature_cols.item(i).isSelected()]
            if not selected_features:
                self.parent.data_stats.setText("Please select at least one numeric feature column.")
                return False
            if len(selected_features) == 1:
                QMessageBox.critical(
                    self.parent,
                    "Помилка вибору ознак",
                    "Для кластеризації необхідно обрати принаймні дві числові ознаки, щоб забезпечити
коректне групування даних."
                )
                return False
```

```
            try:
                self.X_original = self.df[selected_features].values
                if not np.issubdtype(self.X_original.dtype, np.number):
                    self.parent.data_stats.setText("Selected feature columns contain non-numeric data.")
                    return False

                if np.any(np.isnan(self.X_original)):
                    self.parent.data_stats.setText("Dataset contains missing values. Applying mean imputation.")
                    imputer = SimpleImputer(strategy='mean')
                    self.X_original = imputer.fit_transform(self.X_original)
                    if np.any(np.isnan(self.X_original)):
                        self.parent.data_stats.setText("Imputation failed: Unable to handle all missing values.")
                        return False

                self.n_features = len(selected_features)
                if self.parent.has_labels.isChecked() and self.parent.label_col.currentText() != "None":
                    self.y_true = self.df[self.parent.label_col.currentText()].values
                    if np.any(np.isnan(self.y_true)):
                        self.parent.data_stats.setText("Label column contains missing values, which are not
supported.")
                        return False
                self.pca = PCA(n_components=2, random_state=self.parent.algo_seed.value())
                self.X = self.pca.fit_transform(self.X_original)
                self.n_features = 2
                self.is_pca_applied = True
            except Exception as e:
                self.parent.data_stats.setText(f"Error processing CSV data: {str(e)}")
                return False

        self.parent.X = self.X
        self.parent.X_original = self.X_original
        self.parent.y_true = self.y_true
        self.parent.n_features = self.n_features
        self.parent.pca = self.pca
        self.parent.is_pca_applied = self.is_pca_applied
        return self.X is not None and self.n_features > 0


class ClusteringEvaluationModule:
    def __init__(self, parent):
        self.parent = parent
        self.evaluation_results = {}

    def _compute_complex_metric(self, silhouette, davies_bouldin, ari, y_true=None):
        """Обчислює комплексну метрику якості кластеризації у відсотках."""
        # Нормалізація silhouette score: з [-1, 1] до [0, 1]
        norm_silhouette = (silhouette + 1) / 2 if not np.isnan(silhouette) else 0
        # Нормалізація Davies-Bouldin: нижче краще, використовуємо 1 / (1 + DB)
        norm_db = 1 / (1 + davies_bouldin) if not np.isnan(davies_bouldin) else 0
        # Нормалізація ARI: з [-1, 1] до [0, 1], якщо y_true доступний
        norm_ari = (ari + 1) / 2 if y_true is not None and not np.isnan(ari) else None

        # Визначення ваг
        if norm_ari is not None:
            weights = {'silhouette': 0.4, 'davies_bouldin': 0.3, 'ari': 0.3}
            score = (norm_silhouette * weights['silhouette'] +
                     norm_db * weights['davies_bouldin'] +
```

```
                norm_ari * weights['ari'])
        else:
            # Якщо ARI недоступний, перерозподіляємо вагу
            weights = {'silhouette': 0.55, 'davies_bouldin': 0.45}
            score = (norm_silhouette * weights['silhouette'] +
                norm_db * weights['davies_bouldin'])

        # Перетворення у відсотки
        return score * 100

    def evaluate_clustering(self, X, em_labels, kmeans_labels=None, y_true=None):
        self.evaluation_results = {}

        if X is None or len(X) < 2:
            self.evaluation_results['error'] = "Недостатньо даних для оцінки"
            return

        if em_labels is not None and len(np.unique(em_labels)) >= 2:
            silhouette_em = silhouette_score(X, em_labels)
            db_index_em = davies_bouldin_score(X, em_labels)
            ari_em = adjusted_rand_score(y_true, em_labels) if y_true is not None else np.nan
            complex_em = self._compute_complex_metric(silhouette_em, db_index_em, ari_em, y_true)
            self.evaluation_results['EM'] = {
                'silhouette': silhouette_em,
                'davies_bouldin': db_index_em,
                'ari': ari_em,
                'complex_metric': complex_em
            }
        else:
            self.evaluation_results['EM'] = {'error': "Недостатньо кластерів для оцінки EM"}

        if kmeans_labels is not None and len(np.unique(kmeans_labels)) >= 2:
            silhouette_km = silhouette_score(X, kmeans_labels)
            db_index_km = davies_bouldin_score(X, kmeans_labels)
            ari_km = adjusted_rand_score(y_true, kmeans_labels) if y_true is not None else np.nan
            complex_km = self._compute_complex_metric(silhouette_km, db_index_km, ari_km, y_true)
            self.evaluation_results['KMeans'] = {
                'silhouette': silhouette_km,
                'davies_bouldin': db_index_km,
                'ari': ari_km,
                'complex_metric': complex_km
            }
        else:
            if kmeans_labels is not None:
                self.evaluation_results['KMeans'] = {'error': "Недостатньо кластерів для оцінки K-Means"}

    def get_evaluation_results(self):
        return self.evaluation_results


class EMApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("EM Algorithm Interactive Platform")
        self.setGeometry(100, 100, 1200, 800)

        self.X = None
```

```
        self.X_original = None
        self.y_true = None
        self.n_features = 0
        self.em_labels = None
        self.em_probs = None
        self.kmeans_labels = None
        self.em_model = None
        self.kmeans_model = None
        self.em_runtime = None
        self.kmeans_runtime = None
        self.log_likelihoods = None
        self.pca = None
        self.is_pca_applied = False
        self.iteration_history = []

        self.data_importer = DataImportModule(self)
        self.evaluator = ClusteringEvaluationModule(self)

        self.init_ui()
        self.update_ui()

    def init_ui(self):
        main_widget = QWidget()
        self.setCentralWidget(main_widget)
        main_layout = QHBoxLayout(main_widget)

        config_widget = QWidget()
        config_widget.setFixedWidth(300)
        config_layout = QVBoxLayout(config_widget)

        dataset_group = QGroupBox("Data Generation")
        dataset_layout = QFormLayout()

        self.dataset_combo = QComboBox()
        self.dataset_combo.addItems(["Synthetic - Gaussian Blobs", "Synthetic - Moons",
                                     "Synthetic - Circles", "Upload Custom CSV"])
        self.dataset_combo.currentIndexChanged.connect(self.update_dataset_ui)
        dataset_layout.addRow("Dataset Type:", self.dataset_combo)

        self.blobs_samples = QSpinBox()
        self.blobs_samples.setRange(100, 2000)
        self.blobs_samples.setValue(500)
        dataset_layout.addRow("Number of Samples:", self.blobs_samples)

        self.blobs_features = QSpinBox()
        self.blobs_features.setRange(2, 10)
        self.blobs_features.setValue(2)
        dataset_layout.addRow("Number of Features:", self.blobs_features)

        self.blobs_centers = QSpinBox()
        self.blobs_centers.setRange(2, 10)
        self.blobs_centers.setValue(3)
        dataset_layout.addRow("Number of Centers:", self.blobs_centers)

        self.blobs_std = QDoubleSpinBox()
        self.blobs_std.setRange(0.1, 2.0)
        self.blobs_std.setValue(1.0)
```

```
self.blobs_std.setSingleStep(0.1)
dataset_layout.addRow("Cluster Std:", self.blobs_std)

self.blobs_seed = QSpinBox()
self.blobs_seed.setRange(0, 100)
self.blobs_seed.setValue(42)
dataset_layout.addRow("Random Seed:", self.blobs_seed)

self.moons_samples = QSpinBox()
self.moons_samples.setRange(100, 2000)
self.moons_samples.setValue(500)
dataset_layout.addRow("Number of Samples:", self.moons_samples)

self.moons_noise = QDoubleSpinBox()
self.moons_noise.setRange(0.01, 0.5)
self.moons_noise.setValue(0.1)
self.moons_noise.setSingleStep(0.01)
dataset_layout.addRow("Noise:", self.moons_noise)

self.moons_seed = QSpinBox()
self.moons_seed.setRange(0, 100)
self.moons_seed.setValue(42)
dataset_layout.addRow("Random Seed:", self.moons_seed)

self.upload_button = QPushButton("Upload CSV")
self.upload_button.clicked.connect(self.data_importer.upload_csv)
dataset_layout.addRow(self.upload_button)

self.feature_cols = QListWidget()
self.feature_cols.setSelectionMode(QListWidget.MultiSelection)
self.feature_cols.setEnabled(False)
dataset_layout.addRow("Feature Columns:", self.feature_cols)

self.has_labels = QCheckBox("Has Labels")
self.has_labels.stateChanged.connect(self.update_label_ui)
dataset_layout.addRow(self.has_labels)

self.label_col = QComboBox()
self.label_col.setEnabled(False)
dataset_layout.addRow("Label Column:", self.label_col)

dataset_group.setLayout(dataset_layout)
config_layout.addWidget(dataset_group)

algo_group = QGroupBox("Algorithm Parameters")
algo_layout = QFormLayout()

self.n_clusters = QSpinBox()
self.n_clusters.setRange(1, 10)
self.n_clusters.setValue(3)
algo_layout.addRow("Number of Clusters:", self.n_clusters)

self.max_iter = QSpinBox()
self.max_iter.setRange(10, 1000)
self.max_iter.setValue(100)
algo_layout.addRow("Max Iterations:", self.max_iter)
```

```
self.algo_seed = QSpinBox()
self.algo_seed.setRange(0, 100)
self.algo_seed.setValue(42)
algo_layout.addRow("Random Seed:", self.algo_seed)

self.covar_type = QComboBox()
self.covar_type.addItems(["full", "tied", "diag", "spherical"])
algo_layout.addRow("Covariance Type:", self.covar_type)

self.init_method = QComboBox()
self.init_method.addItems(["kmeans", "random"])
algo_layout.addRow("Initialization:", self.init_method)

self.tol = QDoubleSpinBox()
self.tol.setRange(1e-6, 1e-2)
self.tol.setValue(1e-3)
self.tol.setDecimals(6)
algo_layout.addRow("Tolerance:", self.tol)

self.reg_covar = QDoubleSpinBox()
self.reg_covar.setRange(1e-10, 1e-2)
self.reg_covar.setValue(1e-6)
self.reg_covar.setDecimals(10)
algo_layout.addRow("Reg Covar:", self.reg_covar)

self.compare_kmeans = QCheckBox("Compare with K-Means")
self.compare_kmeans.setChecked(True)
algo_layout.addRow(self.compare_kmeans)

algo_group.setLayout(algo_layout)
config_layout.addWidget(algo_group)

self.run_button = QPushButton("Run Analysis")
self.run_button.clicked.connect(self.run_analysis)
config_layout.addWidget(self.run_button)

config_layout.addStretch()
main_layout.addWidget(config_widget)

self.tabs = QTabWidget()
main_layout.addWidget(self.tabs)

self.data_tab = QWidget()
self.data_layout = QVBoxLayout(self.data_tab)
self.data_canvas = FigureCanvas(Figure(figsize=(10, 6)))
self.data_layout.addWidget(self.data_canvas)
self.data_stats = QTextEdit()
self.data_stats.setReadOnly(True)
self.data_layout.addWidget(self.data_stats)
self.tabs.addTab(self.data_tab, "Dataset")

self.results_tab = QTabWidget()
self.tabs.addTab(self.results_tab, "Clustering Results")

self.comparison_tab = QWidget()
self.comparison_layout = QVBoxLayout(self.comparison_tab)
self.comparison_canvas = FigureCanvas(Figure(figsize=(10, 6)))
```

```
        self.comparison_layout.addWidget(self.comparison_canvas)
        self.comparison_table = QTableWidget()
        self.comparison_layout.addWidget(self.comparison_table)
        self.tabs.addTab(self.comparison_tab, "Algorithm Comparison")

        self.details_tab = QWidget()
        self.details_layout = QVBoxLayout(self.details_tab)
        self.details_text = QTextEdit()
        self.details_text.setReadOnly(True)
        self.details_layout.addWidget(self.details_text)
        self.tabs.addTab(self.details_tab, "Algorithm Details")

        self.iteration_tab = QWidget()
        self.iteration_layout = QVBoxLayout(self.iteration_tab)
        self.iteration_selector_layout = QHBoxLayout()
        self.iteration_label = QLabel("Select Iteration:")
        self.iteration_selector = QSpinBox()
        self.iteration_selector.setRange(1, 1)
        self.iteration_selector.valueChanged.connect(self.update_iteration_ui)
        self.iteration_selector_layout.addWidget(self.iteration_label)
        self.iteration_selector_layout.addWidget(self.iteration_selector)
        self.iteration_layout.addLayout(self.iteration_selector_layout)
        self.iteration_canvas = FigureCanvas(Figure(figsize=(10, 6)))
        self.iteration_layout.addWidget(self.iteration_canvas)
        self.iteration_stats = QTextEdit()
        self.iteration_stats.setReadOnly(True)
        self.iteration_layout.addWidget(self.iteration_stats)
        self.tabs.addTab(self.iteration_tab, "Iteration History")

        self.export_tab = QWidget()
        self.export_layout = QVBoxLayout(self.export_tab)
        self.export_table = QTableWidget()
        self.export_layout.addWidget(self.export_table)
        self.export_csv_button = QPushButton("Export CSV")
        self.export_csv_button.clicked.connect(self.export_csv)
        self.export_layout.addWidget(self.export_csv_button)
        self.export_report_button = QPushButton("Generate Report")
        self.export_report_button.clicked.connect(self.generate_report)
        self.export_layout.addWidget(self.export_report_button)
        self.export_pdf_button = QPushButton("Export Iteration PDF")
        self.export_pdf_button.clicked.connect(self.export_iteration_pdf)
        self.export_layout.addWidget(self.export_pdf_button)
        self.tabs.addTab(self.export_tab, "Export")

    def update_dataset_ui(self):
        dataset_type = self.dataset_combo.currentText()
        self.blobs_samples.setVisible(dataset_type == "Synthetic - Gaussian Blobs")
        self.blobs_features.setVisible(dataset_type == "Synthetic - Gaussian Blobs")
        self.blobs_centers.setVisible(dataset_type == "Synthetic - Gaussian Blobs")
        self.blobs_std.setVisible(dataset_type == "Synthetic - Gaussian Blobs")
        self.blobs_seed.setVisible(dataset_type == "Synthetic - Gaussian Blobs")
        self.moons_samples.setVisible(dataset_type in ["Synthetic - Moons", "Synthetic - Circles"])
        self.moons_noise.setVisible(dataset_type in ["Synthetic - Moons", "Synthetic - Circles"])
        self.moons_seed.setVisible(dataset_type in ["Synthetic - Moons", "Synthetic - Circles"])
        self.upload_button.setVisible(dataset_type == "Upload Custom CSV")
        self.feature_cols.setVisible(dataset_type == "Upload Custom CSV")
        self.has_labels.setVisible(dataset_type == "Upload Custom CSV")
```

```
        self.label_col.setVisible(dataset_type == "Upload Custom CSV" and self.has_labels.isChecked())

    def update_label_ui(self):
        self.label_col.setEnabled(self.has_labels.isChecked())

    def plot_3d_ellipsoid(self, ax, mean, cov, color, alpha, level):
        eigvals, eigvecs = linalg.eigh(cov)
        order = eigvals.argsort()[::-1]
        eigvals, eigvecs = eigvals[order], eigvecs[:, order]
        radii = level * np.sqrt(eigvals)
        u = np.linspace(0, 2 * np.pi, 20)
        v = np.linspace(0, np.pi, 20)
        x = radii[0] * np.outer(np.cos(u), np.sin(v))
        y = radii[1] * np.outer(np.sin(u), np.sin(v))
        z = radii[2] * np.outer(np.ones_like(u), np.cos(v))
        for i in range(len(x)):
            for j in range(len(x)):
                [x[i,j], y[i,j], z[i,j]] = np.dot(eigvecs, [x[i,j], y[i,j], z[i,j]]) + mean
        ax.plot_wireframe(x, y, z, color=color, alpha=alpha, rstride=2, cstride=2)

    def run_analysis(self):
        if self.n_clusters.value() == 1:
            QMessageBox.critical(
                self,
                "Помилка вибору кластерів",
                "Для кластеризації необхідно обрати принаймні два кластери, щоб забезпечити змістовне групування та оцінку якості."
            )
            return

        if not self.data_importer.generate_dataset():
            return

        self.iteration_history = []
        start_time = time.time()

        for i in range(1, self.max_iter.value() + 1):
            temp_model = GaussianMixture(
                n_components=self.n_clusters.value(),
                covariance_type=self.covar_type.currentText(),
                tol=self.tol.value(),
                reg_covar=self.reg_covar.value(),
                max_iter=i,
                n_init=1,
                init_params=self.init_method.currentText(),
                random_state=self.algo_seed.value()
            )
            temp_model.fit(self.X)
            labels = temp_model.predict(self.X)
            probs = temp_model.predict_proba(self.X)
            log_likelihood = temp_model.score(self.X) * self.X.shape[0]
            silhouette = silhouette_score(self.X, labels) if self.n_clusters.value() > 1 and len(np.unique(labels)) > 1 else np.nan
            ari = adjusted_rand_score(self.y_true, labels) if self.y_true is not None else np.nan
            self.iteration_history.append({
                'iteration': i,
                'labels': labels.copy(),
```

```
                'probs': probs.copy(),
                'means': temp_model.means_.copy(),
                'covariances': temp_model.covariances_.copy(),
                'weights': temp_model.weights_.copy(),
                'log_likelihood': log_likelihood,
                'silhouette': silhouette,
                'ari': ari,
                'converged': temp_model.converged_
            })
            if temp_model.converged_:
                break

        self.em_model = GaussianMixture(
            n_components=self.n_clusters.value(),
            covariance_type=self.covar_type.currentText(),
            tol=self.tol.value(),
            reg_covar=self.reg_covar.value(),
            max_iter=self.max_iter.value(),
            n_init=1,
            init_params=self.init_method.currentText(),
            random_state=self.algo_seed.value()
        )
        self.em_model.fit(self.X)
        self.em_labels = self.em_model.predict(self.X)
        self.em_probs = self.em_model.predict_proba(self.X)
        self.log_likelihoods = [iter_data['log_likelihood'] for iter_data in self.iteration_history]
        self.em_runtime = time.time() - start_time

        if self.compare_kmeans.isChecked():
            start_time = time.time()
            self.kmeans_model = KMeans(
                n_clusters=self.n_clusters.value(),
                max_iter=self.max_iter.value(),
                random_state=self.algo_seed.value()
            )
            self.kmeans_model.fit(self.X)
            self.kmeans_labels = self.kmeans_model.labels_
            self.kmeans_runtime = time.time() - start_time

        self.evaluator.evaluate_clustering(self.X, self.em_labels, self.kmeans_labels, self.y_true)
        self.iteration_selector.setRange(1, len(self.iteration_history))
        self.update_ui()

    def update_ui(self):
        if self.X is None or self.n_features == 0:
            self.data_stats.setText("No dataset loaded. Please configure a dataset.")
            return

        self.data_canvas.figure.clear()
        if self.n_features == 2:
            ax = self.data_canvas.figure.add_subplot(111)
            if self.y_true is not None:
                scatter = ax.scatter(self.X[:, 0], self.X[:, 1], c=self.y_true, alpha=0.6, cmap='viridis')
                ax.legend(*scatter.legend_elements(), title="True Clusters")
            else:
                scatter = ax.scatter(self.X[:, 0], self.X[:, 1], c='blue', alpha=0.6)
            ax.set_title('Original Dataset' + (' (PCA Reduced to 2D)' if self.is_pca_applied else ''))
```

```
            ax.set_xlabel('Principal Component 1' if self.is_pca_applied else 'Feature 1')
            ax.set_ylabel('Principal Component 2' if self.is_pca_applied else 'Feature 2')
        elif self.n_features == 3:
            ax = self.data_canvas.figure.add_subplot(111, projection='3d')
            if self.y_true is not None:
                scatter = ax.scatter(self.X[:, 0], self.X[:, 1], self.X[:, 2], c=self.y_true, alpha=0.6, cmap='viridis')
                ax.legend(*scatter.legend_elements(), title="True Clusters")
            else:
                scatter = ax.scatter(self.X[:, 0], self.X[:, 1], self.X[:, 2], c='blue', alpha=0.6)
            ax.set_title('Original Dataset (3D)')
            ax.set_xlabel('Feature 1')
            ax.set_ylabel('Feature 2')
            ax.set_zlabel('Feature 3')
        else:
            ax = self.data_canvas.figure.add_subplot(111)
            ax.text(0.5, 0.5, 'High-dimensional data.\nSee statistics below.',
                    horizontalalignment='center', verticalalignment='center')
        self.data_canvas.draw()

        stats_text = f"Dataset Shape: {self.X.shape}\n"
        if self.is_pca_applied:
            stats_text += f"Original Shape (before PCA): {self.X_original.shape}\n"
            stats_text += f"Explained Variance Ratio: {self.pca.explained_variance_ratio_.round(4)}\n"
        if self.y_true is not None:
            stats_text += f"Number of unique classes: {len(np.unique(self.y_true))}\n"
        if self.n_features <= 10:
            stats_df = pd.DataFrame({
                'Min': np.min(self.X, axis=0),
                'Max': np.max(self.X, axis=0),
                'Mean': np.mean(self.X, axis=0),
                'Std': np.std(self.X, axis=0)
            }, index=[f"Principal Component {i+1}" if self.is_pca_applied else f"Feature {i+1}"
                      for i in range(self.n_features)])
            stats_text += "\nFeature Statistics:\n" + stats_df.to_string()
        self.data_stats.setText(stats_text)

        self.results_tab.clear()
        if self.em_labels is not None:
            if self.n_features == 2:
                cluster_widget = QWidget()
                cluster_layout = QVBoxLayout(cluster_widget)
                cluster_canvas = FigureCanvas(Figure(figsize=(10, 6)))
                cluster_layout.addWidget(cluster_canvas)
                ax = cluster_canvas.figure.add_subplot(111)
                colors = plt.cm.viridis(np.linspace(0, 1, self.n_clusters.value()))
                scatter = ax.scatter(self.X[:, 0], self.X[:, 1], c=self.em_labels, alpha=0.6, cmap='viridis')
                ax.scatter(self.em_model.means_[:, 0], self.em_model.means_[:, 1], marker='X', c='red', s=200,
label='EM Centers')
                for i in range(self.n_clusters.value()):
                    mean = self.em_model.means_[i]
                    cov = self.em_model.covariances_[i] if self.covar_type.currentText() in ['full', 'diag', 'spherical']
else self.em_model.covariances_
                    if self.covar_type.currentText() == 'spherical':
                        cov = np.eye(2) * cov[i]
                    elif self.covar_type.currentText() == 'diag':
                        cov = np.diag(cov[i])
                    elif self.covar_type.currentText() == 'tied':
```

```
                cov = self.em_model.covariances_
            eigvals, eigvecs = np.linalg.eigh(cov)
            order = eigvals.argsort()[::-1]
            eigvals, eigvecs = eigvals[order], eigvecs[:, order]
            angle = np.degrees(np.arctan2(*eigvecs[:, 0][::-1]))
            for level, alpha in zip([1, 2, 3], [0.3, 0.2, 0.1]):
                width, height = 2 * level * np.sqrt(eigvals)
                ellipse = Ellipse(xy=mean, width=width, height=height, angle=angle,
                          edgecolor='none', fc=colors[i], alpha=alpha)
                ax.add_patch(ellipse)
        ax.set_title('EM Clustering Results with Confidence Ellipses')
        ax.set_xlabel('Principal Component 1' if self.is_pca_applied else 'Feature 1')
        ax.set_ylabel('Principal Component 2' if self.is_pca_applied else 'Feature 2')
        ax.legend(*scatter.legend_elements(), title="EM Clusters")
        cluster_canvas.draw()
        self.results_tab.addTab(cluster_widget, "Cluster Assignments")

        heat_probs_widget = QWidget()
        heat_probs_layout = QVBoxLayout(heat_probs_widget)
        heat_probs_canvas = FigureCanvas(Figure(figsize=(4 * self.n_clusters.value(), 4)))
        heat_probs_layout.addWidget(heat_probs_canvas)
        fig = heat_probs_canvas.figure
        for i in range(self.n_clusters.value()):
            ax = fig.add_subplot(1, self.n_clusters.value(), i + 1)
            points = ax.scatter(self.X[:, 0], self.X[:, 1], c=self.em_probs[:, i], cmap='viridis',
                                vmin=0, vmax=1, alpha=0.8)
            ax.scatter(self.em_model.means_[i, 0], self.em_model.means_[i, 1], marker='X', c='red', s=200)
            ax.set_title(f'Cluster {i+1} Probability')
            fig.colorbar(points, ax=ax)
            ax.set_xlabel('Principal Component 1' if self.is_pca_applied else 'Feature 1')
            ax.set_ylabel('Principal Component 2' if self.is_pca_applied else 'Feature 2')
        fig.tight_layout()
        heat_probs_canvas.draw()
        self.results_tab.addTab(heat_probs_widget, "Membership Probabilities (Heat)")

    elif self.n_features == 3:
        cluster_widget = QWidget()
        cluster_layout = QVBoxLayout(cluster_widget)
        cluster_canvas = FigureCanvas(Figure(figsize=(10, 6)))
        cluster_layout.addWidget(cluster_canvas)
        ax = cluster_canvas.figure.add_subplot(111, projection='3d')
        colors = plt.cm.viridis(np.linspace(0, 1, self.n_clusters.value()))
        scatter = ax.scatter(self.X[:, 0], self.X[:, 1], self.X[:, 2], c=self.em_labels, alpha=0.6, cmap='viridis')
        ax.scatter(self.em_model.means_[:, 0], self.em_model.means_[:, 1], self.em_model.means_[:, 2],
                   marker='X', c='red', s=200, label='EM Centers')
        for i in range(self.n_clusters.value()):
            mean = self.em_model.means_[i]
            cov = self.em_model.covariances_[i] if self.covar_type.currentText() in ['full', 'diag', 'spherical']
else self.em_model.covariances_
            if self.covar_type.currentText() == 'spherical':
                cov = np.eye(3) * cov[i]
            elif self.covar_type.currentText() == 'diag':
                cov = np.diag(cov[i])
            elif self.covar_type.currentText() == 'tied':
                cov = self.em_model.covariances_
            for level, alpha in zip([1, 2, 3], [0.3, 0.2, 0.1]):
                self.plot_3d_ellipsoid(ax, mean, cov, colors[i], alpha, level)
```

```
            ax.set_title('EM Clustering Results with Confidence Ellipsoids')
            ax.set_xlabel('Feature 1')
            ax.set_ylabel('Feature 2')
            ax.set_zlabel('Feature 3')
            ax.legend(*scatter.legend_elements(), title="EM Clusters")
            cluster_canvas.draw()
            self.results_tab.addTab(cluster_widget, "Cluster Assignments")

            heat_probs_widget = QWidget()
            heat_probs_layout = QVBoxLayout(heat_probs_widget)
            heat_probs_canvas = FigureCanvas(Figure(figsize=(4 * self.n_clusters.value(), 4)))
            heat_probs_layout.addWidget(heat_probs_canvas)
            fig = heat_probs_canvas.figure
            for i in range(self.n_clusters.value()):
                ax = fig.add_subplot(1, self.n_clusters.value(), i + 1, projection='3d')
                points = ax.scatter(self.X[:, 0], self.X[:, 1], self.X[:, 2], c=self.em_probs[:, i],
                                    cmap='viridis', vmin=0, vmax=1, alpha=0.8)
                ax.scatter(self.em_model.means_[i, 0], self.em_model.means_[i, 1], self.em_model.means_[i, 2],
                           marker='X', c='red', s=200)
                ax.set_title(f'Cluster {i+1} Probability')
                fig.colorbar(points, ax=ax)
                ax.set_xlabel('Feature 1')
                ax.set_ylabel('Feature 2')
                ax.set_zlabel('Feature 3')
            fig.tight_layout()
            heat_probs_canvas.draw()
            self.results_tab.addTab(heat_probs_widget, "Membership Probabilities (Heat)")

        gaussian_widget = QWidget()
        gaussian_layout = QVBoxLayout(gaussian_widget)
        gaussian_canvas = FigureCanvas(Figure(figsize=(10, 8)))
        gaussian_layout.addWidget(gaussian_canvas)
        fig = gaussian_canvas.figure
        colors = plt.cm.viridis(np.linspace(0, 1, self.n_clusters.value()))

        ax_x = fig.add_subplot(311 if self.n_features == 3 else 211)
        x_range = np.linspace(self.X[:, 0].min() - 1, self.X[:, 0].max() + 1, 100)
        for i in range(self.n_clusters.value()):
            mean = self.em_model.means_[i]
            cov = self.em_model.covariances_[i] if self.covar_type.currentText() in ['full', 'diag', 'spherical'] else
self.em_model.covariances_
            if self.covar_type.currentText() == 'spherical':
                cov = np.eye(self.n_features) * cov[i]
            elif self.covar_type.currentText() == 'diag':
                cov = np.diag(cov[i])
            elif self.covar_type.currentText() == 'tied':
                cov = self.em_model.covariances_
            std_x = np.sqrt(cov[0, 0]) if self.covar_type.currentText() in ['full', 'tied'] else np.sqrt(cov[0])
            x_density = norm.pdf(x_range, mean[0], std_x)
            ax_x.plot(x_range, x_density, color=colors[i], label=f'Cluster {i+1}')
            ax_x.fill_between(x_range, x_density, alpha=0.3, color=colors[i])
        ax_x.set_title(f"Gaussian Density - {'Principal Component 1' if self.is_pca_applied else 'Feature 1'}")
        ax_x.set_xlabel('Principal Component 1' if self.is_pca_applied else 'Feature 1')
        ax_x.set_ylabel('Density')
        ax_x.legend()
        ax_x.grid(True)
```

```
        ax_y = fig.add_subplot(312 if self.n_features == 3 else 212)
        y_range = np.linspace(self.X[:, 1].min() - 1, self.X[:, 1].max() + 1, 100)
        for i in range(self.n_clusters.value()):
            mean = self.em_model.means_[i]
            cov = self.em_model.covariances_[i] if self.covar_type.currentText() in ['full', 'diag', 'spherical'] else
self.em_model.covariances_
            if self.covar_type.currentText() == 'spherical':
                cov = np.eye(self.n_features) * cov[i]
            elif self.covar_type.currentText() == 'diag':
                cov = np.diag(cov[i])
            elif self.covar_type.currentText() == 'tied':
                cov = self.em_model.covariances_
            std_y = np.sqrt(cov[1, 1]) if self.covar_type.currentText() in ['full', 'tied'] else np.sqrt(cov[1])
            y_density = norm.pdf(y_range, mean[1], std_y)
            ax_y.plot(y_range, y_density, color=colors[i], label=f'Cluster {i+1}')
            ax_y.fill_between(y_range, y_density, alpha=0.3, color=colors[i])
        ax_y.set_title(f"Gaussian Density - {'Principal Component 2' if self.is_pca_applied else 'Feature 2'}")
        ax_y.set_xlabel('Principal Component 2' if self.is_pca_applied else 'Feature 2')
        ax_y.set_ylabel('Density')
        ax_y.legend()
        ax_y.grid(True)

        if self.n_features == 3:
            ax_z = fig.add_subplot(313)
            z_range = np.linspace(self.X[:, 2].min() - 1, self.X[:, 2].max() + 1, 100)
            for i in range(self.n_clusters.value()):
                mean = self.em_model.means_[i]
                cov = self.em_model.covariances_[i] if self.covar_type.currentText() in ['full', 'diag', 'spherical']
else self.em_model.covariances_
                if self.covar_type.currentText() == 'spherical':
                    cov = np.eye(3) * cov[i]
                elif self.covar_type.currentText() == 'diag':
                    cov = np.diag(cov[i])
                elif self.covar_type.currentText() == 'tied':
                    cov = self.em_model.covariances_
                std_z = np.sqrt(cov[2, 2]) if self.covar_type.currentText() in ['full', 'tied'] else np.sqrt(cov[2])
                z_density = norm.pdf(z_range, mean[2], std_z)
                ax_z.plot(z_range, z_density, color=colors[i], label=f'Cluster {i+1}')
                ax_z.fill_between(z_range, z_density, alpha=0.3, color=colors[i])
            ax_z.set_title('Gaussian Density - Feature 3')
            ax_z.set_xlabel('Feature 3')
            ax_z.set_ylabel('Density')
            ax_z.legend()
            ax_z.grid(True)

        fig.tight_layout()
        gaussian_canvas.draw()
        self.results_tab.addTab(gaussian_widget, "Gaussian Density")

        if self.log_likelihoods and self.n_features == 2:
            ll_cov_widget = QWidget()
            ll_cov_layout = QVBoxLayout(ll_cov_widget)
            ll_cov_canvas = FigureCanvas(Figure(figsize=(10, 6)))
            ll_cov_layout.addWidget(ll_cov_canvas)
            ax = ll_cov_canvas.figure.add_subplot(111)
            x_min, x_max = self.X[:, 0].min() - 1, self.X[:, 0].max() + 1
            y_min, y_max = self.X[:, 1].min() - 1, self.X[:, 1].max() + 1
```

```
            xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
            X_grid = np.c_[xx.ravel(), yy.ravel()]
            Z = self.em_model.score_samples(X_grid)
            Z = Z.reshape(xx.shape)
            contour = ax.contour(xx, yy, Z, levels=10, cmap='viridis_r')
            ax.scatter(self.X[:, 0], self.X[:, 1], c=self.em_labels, alpha=0.6, cmap='viridis')
            ax.scatter(self.em_model.means_[:, 0], self.em_model.means_[:, 1], marker='X', c='red', s=200,
label='EM Centers')
            ax.set_title('Log-Likelihood Covariance')
            ax.set_xlabel('Principal Component 1' if self.is_pca_applied else 'Feature 1')
            ax.set_ylabel('Principal Component 2' if self.is_pca_applied else 'Feature 2')
            ax.legend()
            ll_cov_canvas.figure.colorbar(contour, ax=ax, label='Log-Likelihood')
            ll_cov_canvas.draw()
            self.results_tab.addTab(ll_cov_widget, "Log-Likelihood Covariance")

        if self.log_likelihoods:
            ll_value_widget = QWidget()
            ll_value_layout = QVBoxLayout(ll_value_widget)
            ll_value_canvas = FigureCanvas(Figure(figsize=(10, 6)))
            ll_value_layout.addWidget(ll_value_canvas)
            ax = ll_value_canvas.figure.add_subplot(111)
            ax.plot(range(1, len(self.log_likelihoods) + 1), self.log_likelihoods, marker='o')
            ax.set_title('Log-Likelihood vs. Iteration')
            ax.set_xlabel('Iteration')
            ax.set_ylabel('Log-Likelihood')
            ax.grid(True)
            ll_value_canvas.draw()
            self.results_tab.addTab(ll_value_widget, "Log-Likelihood Values")

        params_widget = QWidget()
        params_layout = QVBoxLayout(params_widget)
        if self.n_features == 2:
            params_canvas = FigureCanvas(Figure(figsize=(10, 6)))
            params_layout.addWidget(params_canvas)
            ax = params_canvas.figure.add_subplot(111)
            ax.scatter(self.X[:, 0], self.X[:, 1], c=self.em_labels, alpha=0.6, cmap='viridis')
            ax.scatter(self.em_model.means_[:, 0], self.em_model.means_[:, 1], marker='X', c='red', s=200,
label='EM Centers')
            for i in range(self.n_clusters.value()):
                mean = self.em_model.means_[i]
                cov = self.em_model.covariances_[i] if self.covar_type.currentText() in ['full', 'diag', 'spherical']
else self.em_model.covariances_
                if self.covar_type.currentText() == 'spherical':
                    cov = np.eye(2) * cov[i]
                elif self.covar_type.currentText() == 'diag':
                    cov = np.diag(cov[i])
                elif self.covar_type.currentText() == 'tied':
                    cov = self.em_model.covariances_
                eigvals, eigvecs = np.linalg.eigh(cov)
                order = eigvals.argsort()[::-1]
                eigvals, eigvecs = eigvals[order], eigvecs[:, order]
                angle = np.degrees(np.arctan2(*eigvecs[:, 0][::-1]))
                width, height = 2 * np.sqrt(5.991 * eigvals)
                ellipse = Ellipse(xy=mean, width=width, height=height, angle=angle,
                                  edgecolor='black', fc='None', lw=2)
                ax.add_patch(ellipse)
```

```
            ax.set_title('EM Cluster Means and Covariances')
            ax.set_xlabel('Principal Component 1' if self.is_pca_applied else 'Feature 1')
            ax.set_ylabel('Principal Component 2' if self.is_pca_applied else 'Feature 2')
            ax.legend()
            params_canvas.draw()
        elif self.n_features == 3:
            params_canvas = FigureCanvas(Figure(figsize=(10, 6)))
            params_layout.addWidget(params_canvas)
            ax = params_canvas.figure.add_subplot(111, projection='3d')
            ax.scatter(self.X[:, 0], self.X[:, 1], self.X[:, 2], c=self.em_labels, alpha=0.6, cmap='viridis')
            ax.scatter(self.em_model.means_[:, 0], self.em_model.means_[:, 1], self.em_model.means_[:, 2],
                       marker='X', c='red', s=200, label='EM Centers')
            colors = plt.cm.viridis(np.linspace(0, 1, self.n_clusters.value()))
            for i in range(self.n_clusters.value()):
                mean = self.em_model.means_[i]
                cov = self.em_model.covariances_[i] if self.covar_type.currentText() in ['full', 'diag', 'spherical']
else self.em_model.covariances_
                if self.covar_type.currentText() == 'spherical':
                    cov = np.eye(3) * cov[i]
                elif self.covar_type.currentText() == 'diag':
                    cov = np.diag(cov[i])
                elif self.covar_type.currentText() == 'tied':
                    cov = self.em_model.covariances_
                self.plot_3d_ellipsoid(ax, mean, cov, 'black', 0.5, level=1.96)
            ax.set_title('EM Cluster Means and Covariances')
            ax.set_xlabel('Feature 1')
            ax.set_ylabel('Feature 2')
            ax.set_zlabel('Feature 3')
            ax.legend()
            params_canvas.draw()
        params_text = QTextEdit()
        params_text.setReadOnly(True)
        params_text.setText(
            f"### Model Parameters\n\n"
            f"**Means:**\n{self.em_model.means_}\n\n"
            f"**Weights:**\n{self.em_model.weights_}\n\n"
            f"**Covariances:**\n{self.em_model.covariances_}\n"
        )
        params_layout.addWidget(params_text)
        self.results_tab.addTab(params_widget, "Model Parameters")

        metrics_widget = QWidget()
        metrics_layout = QHBoxLayout(metrics_widget)
        em_table = QTableWidget()
        em_table.setColumnCount(2)
        em_table.setHorizontalHeaderLabels(["Metric", "Value"])
        metrics = [
            ("Log Likelihood", f"{self.em_model.score(self.X) * self.X.shape[0]:.4f}"),
            ("BIC", f"{self.em_model.bic(self.X):.4f}"),
            ("AIC", f"{self.em_model.aic(self.X):.4f}"),
            ("Iterations", str(self.em_model.n_iter_)),
            ("Runtime (s)", f"{self.em_runtime:.4f}")
        ]
        eval_results = self.evaluator.get_evaluation_results()
        if 'EM' in eval_results and 'error' not in eval_results['EM']:
            metrics.extend([
                ("Silhouette Score", f"{eval_results['EM']['silhouette']:.4f}"),
```

```
                ("Davies-Bouldin Index", f"{eval_results['EM']['davies_bouldin']:.4f}"),
                ("Complex Quality Metric (%)", f"{eval_results['EM']['complex_metric']:.2f}")
            ])
            if not np.isnan(eval_results['EM']['ari']):
                metrics.append(("Adjusted Rand Index", f"{eval_results['EM']['ari']:.4f}"))
        em_table.setRowCount(len(metrics))
        for i, (metric, value) in enumerate(metrics):
            em_table.setItem(i, 0, QTableWidgetItem(metric))
            em_table.setItem(i, 1, QTableWidgetItem(value))
        metrics_layout.addWidget(em_table)

        if self.compare_kmeans.isChecked() and self.kmeans_labels is not None:
            kmeans_table = QTableWidget()
            kmeans_table.setColumnCount(2)
            kmeans_table.setHorizontalHeaderLabels(["Metric", "Value"])
            kmeans_metrics = [
                ("Inertia", f"{self.kmeans_model.inertia_:.4f}"),
                ("Iterations", str(self.kmeans_model.n_iter_)),
                ("Runtime (s)", f"{self.kmeans_runtime:.4f}")
            ]
            if 'KMeans' in eval_results and 'error' not in eval_results['KMeans']:
                kmeans_metrics.extend([
                    ("Silhouette Score", f"{eval_results['KMeans']['silhouette']:.4f}"),
                    ("Davies-Bouldin Index", f"{eval_results['KMeans']['davies_bouldin']:.4f}"),
                    ("Complex Quality Metric (%)", f"{eval_results['KMeans']['complex_metric']:.2f}")
                ])
                if not np.isnan(eval_results['KMeans']['ari']):
                    kmeans_metrics.append(("Adjusted Rand Index", f"{eval_results['KMeans']['ari']:.4f}"))
            kmeans_table.setRowCount(len(kmeans_metrics))
            for i, (metric, value) in enumerate(kmeans_metrics):
                kmeans_table.setItem(i, 0, QTableWidgetItem(metric))
                kmeans_table.setItem(i, 1, QTableWidgetItem(value))
            metrics_layout.addWidget(kmeans_table)
        self.results_tab.addTab(metrics_widget, "Metrics")

    self.comparison_canvas.figure.clear()
    self.comparison_table.clear()
    if self.em_labels is not None and self.compare_kmeans.isChecked() and self.kmeans_labels is not None
and self.n_features == 2:
        fig = self.comparison_canvas.figure
        ax1 = fig.add_subplot(121)
        ax1.scatter(self.X[:, 0], self.X[:, 1], c=self.em_labels, alpha=0.6, cmap='viridis')
        ax1.scatter(self.em_model.means_[:, 0], self.em_model.means_[:, 1], marker='X', c='red', s=200,
label='EM Centers')
        ax1.set_title('EM Clustering')
        ax1.set_xlabel('Principal Component 1' if self.is_pca_applied else 'Feature 1')
        ax1.set_ylabel('Principal Component 2' if self.is_pca_applied else 'Feature 2')
        ax1.legend()

        ax2 = fig.add_subplot(122)
        ax2.scatter(self.X[:, 0], self.X[:, 1], c=self.kmeans_labels, alpha=0.6, cmap='viridis')
        ax2.scatter(self.kmeans_model.cluster_centers_[:, 0], self.kmeans_model.cluster_centers_[:, 1],
                    marker='X', c='red', s=200, label='K-Means Centers')
        ax2.set_title('K-Means Clustering')
        ax2.set_xlabel('Principal Component 1' if self.is_pca_applied else 'Feature 1')
        ax2.set_ylabel('Principal Component 2' if self.is_pca_applied else 'Feature 2')
        ax2.legend()
```

```
        fig.tight_layout()
        self.comparison_canvas.draw()

        eval_results = self.evaluator.get_evaluation_results()
        comparison_metrics = [
            ("Silhouette Score",
             f"{eval_results['EM']['silhouette']:.4f}" if 'EM' in eval_results and 'error' not in eval_results['EM']
else "N/A",
             f"{eval_results['KMeans']['silhouette']:.4f}" if 'KMeans' in eval_results and 'error' not in
eval_results['KMeans'] else "N/A"),
            ("Davies-Bouldin Index",
             f"{eval_results['EM']['davies_bouldin']:.4f}" if 'EM' in eval_results and 'error' not in
eval_results['EM'] else "N/A",
             f"{eval_results['KMeans']['davies_bouldin']:.4f}" if 'KMeans' in eval_results and 'error' not in
eval_results['KMeans'] else "N/A"),
            ("Complex Quality Metric (%)",
             f"{eval_results['EM']['complex_metric']:.2f}" if 'EM' in eval_results and 'error' not in
eval_results['EM'] else "N/A",
             f"{eval_results['KMeans']['complex_metric']:.2f}" if 'KMeans' in eval_results and 'error' not in
eval_results['KMeans'] else "N/A"),
            ("Adjusted Rand Index",
             f"{eval_results['EM']['ari']:.4f}" if 'EM' in eval_results and not np.isnan(eval_results['EM']['ari'])
else "N/A",
             f"{eval_results['KMeans']['ari']:.4f}" if 'KMeans' in eval_results and not
np.isnan(eval_results['KMeans']['ari']) else "N/A"),
            ("Runtime (s)", f"{self.em_runtime:.4f}", f"{self.kmeans_runtime:.4f}"),
            ("Iterations", str(self.em_model.n_iter_), str(self.kmeans_model.n_iter_)),
            ("Objective", f"Log-Likelihood: {self.em_model.score(self.X) * self.X.shape[0]:.4f}",
             f"Inertia: {self.kmeans_model.inertia_:.4f}")
        ]
        self.comparison_table.setRowCount(len(comparison_metrics))
        self.comparison_table.setColumnCount(3)
        self.comparison_table.setHorizontalHeaderLabels(["Metric", "EM", "K-Means"])
        for i, (metric, em_val, kmeans_val) in enumerate(comparison_metrics):
            self.comparison_table.setItem(i, 0, QTableWidgetItem(metric))
            self.comparison_table.setItem(i, 1, QTableWidgetItem(em_val))
            self.comparison_table.setItem(i, 2, QTableWidgetItem(kmeans_val))

    details_text = """
    ### Expectation-Maximization (EM) Algorithm Steps

    The EM algorithm is an iterative method used for finding maximum likelihood estimates of parameters
in statistical models with unobserved latent variables. In the context of Gaussian Mixture Models, it works as
follows:

    #### 1. Initialization
    - Initialize the means (μ), covariances (Σ), and mixing coefficients (π) of the Gaussian components.
    - This can be done randomly or using K-means clustering.

    #### 2. Expectation Step (E-step)
    - Calculate the "responsibility" each Gaussian component takes for explaining each data point.
    - For each data point x_i and component k, compute:
      γ(z_ik) = π_k * N(x_i|μ_k,Σ_k) / Σ_j [π_j * N(x_i|μ_j,Σ_j)]
    - This is essentially the posterior probability that data point x_i belongs to cluster k.

    #### 3. Maximization Step (M-step)
    - Update the parameters (μ, Σ, π) based on the computed responsibilities:
```

```
        - μ_k = Σ_i [γ(z_ik) * x_i] / N_k
        - Σ_k = Σ_i [γ(z_ik) * (x_i - μ_k) * (x_i - μ_k)^T] / N_k
        - π_k = N_k / N
    - Where N_k = Σ_i [γ(z_ik)] is the effective number of points assigned to cluster k.

    #### 4. Convergence Check
    - Calculate the log-likelihood of the data given the updated parameters.
    - If the change in log-likelihood is below a threshold or maximum iterations are reached, stop.
    - Otherwise, go back to step 2 (E-step).

    The EM algorithm guarantees that the log-likelihood increases with each iteration, eventually converging to a local maximum.
    """
    if self.em_model is not None and hasattr(self.em_model, 'lower_bound_'):
        details_text += f"\n### Final Log-Likelihood\n\n- Final Lower Bound: {self.em_model.lower_bound_:.4f}\n"
    self.details_text.setText(details_text)

    if self.em_labels is not None:
        if self.is_pca_applied:
            export_df = pd.DataFrame(self.X, columns=['Principal Component 1', 'Principal Component 2'])
        else:
            export_df = pd.DataFrame(self.X, columns=[f"Feature_{i+1}" for i in range(self.n_features)])
        export_df['EM_Cluster'] = self.em_labels
        if self.n_clusters.value() <= 10:
            for i in range(self.n_clusters.value()):
                export_df[f'Prob_Cluster_{i+1}'] = self.em_probs[:, i]
        if self.compare_kmeans.isChecked() and self.kmeans_labels is not None:
            export_df['KMeans_Cluster'] = self.kmeans_labels
        if self.y_true is not None:
            export_df['True_Label'] = self.y_true
        if self.is_pca_applied and self.X_original.shape[1] <= 10:
            for i in range(self.X_original.shape[1]):
                export_df[f'Original_Feature_{i+1}'] = self.X_original[:, i]

        self.export_table.setRowCount(min(export_df.shape[0], 10))
        self.export_table.setColumnCount(export_df.shape[1])
        self.export_table.setHorizontalHeaderLabels(export_df.columns)
        for i in range(min(export_df.shape[0], 10)):
            for j in range(export_df.shape[1]):
                self.export_table.setItem(i, j, QTableWidgetItem(str(export_df.iloc[i, j])))
        self.export_df = export_df

    self.update_iteration_ui()

def update_iteration_ui(self):
    if not self.iteration_history:
        self.iteration_stats.setText("No iteration data available.")
        self.iteration_canvas.figure.clear()
        self.iteration_canvas.draw()
        return

    iter_idx = self.iteration_selector.value() - 1
    iter_data = self.iteration_history[iter_idx]

    self.iteration_canvas.figure.clear()
    if self.n_features == 2:
```

```
            ax = self.iteration_canvas.figure.add_subplot(111)
            colors = plt.cm.viridis(np.linspace(0, 1, self.n_clusters.value()))
            scatter = ax.scatter(self.X[:, 0], self.X[:, 1], c=iter_data['labels'], alpha=0.6, cmap='viridis')
            ax.scatter(iter_data['means'][:, 0], iter_data['means'][:, 1], marker='X', c='red', s=200, label='EM Centers')
            for i in range(self.n_clusters.value()):
                mean = iter_data['means'][i]
                cov = iter_data['covariances'][i] if self.covar_type.currentText() in ['full', 'diag', 'spherical'] else iter_data['covariances']
                if self.covar_type.currentText() == 'spherical':
                    cov = np.eye(2) * cov[i]
                elif self.covar_type.currentText() == 'diag':
                    cov = np.diag(cov[i])
                elif self.covar_type.currentText() == 'tied':
                    cov = iter_data['covariances']
                eigvals, eigvecs = np.linalg.eigh(cov)
                order = eigvals.argsort()[::-1]
                eigvals, eigvecs = eigvals[order], eigvecs[:, order]
                angle = np.degrees(np.arctan2(*eigvecs[:, 0][::-1]))
                for level, alpha in zip([1, 2, 3], [0.3, 0.2, 0.1]):
                    width, height = 2 * level * np.sqrt(eigvals)
                    ellipse = Ellipse(xy=mean, width=width, height=height, angle=angle,
                                      edgecolor='none', fc=colors[i], alpha=alpha)
                    ax.add_patch(ellipse)
            ax.set_title(f'EM Clustering at Iteration {iter_data["iteration"]}')
            ax.set_xlabel('Principal Component 1' if self.is_pca_applied else 'Feature 1')
            ax.set_ylabel('Principal Component 2' if self.is_pca_applied else 'Feature 2')
            ax.legend(*scatter.legend_elements(), title="EM Clusters")
        elif self.n_features == 3:
            ax = self.iteration_canvas.figure.add_subplot(111, projection='3d')
            colors = plt.cm.viridis(np.linspace(0, 1, self.n_clusters.value()))
            scatter = ax.scatter(self.X[:, 0], self.X[:, 1], self.X[:, 2], c=iter_data['labels'], alpha=0.6, cmap='viridis')
            ax.scatter(iter_data['means'][:, 0], iter_data['means'][:, 1], iter_data['means'][:, 2],
                       marker='X', c='red', s=200, label='EM Centers')
            for i in range(self.n_clusters.value()):
                mean = iter_data['means'][i]
                cov = iter_data['covariances'][i] if self.covar_type.currentText() in ['full', 'diag', 'spherical'] else iter_data['covariances']
                if self.covar_type.currentText() == 'spherical':
                    cov = np.eye(3) * cov[i]
                elif self.covar_type.currentText() == 'diag':
                    cov = np.diag(cov[i])
                elif self.covar_type.currentText() == 'tied':
                    cov = iter_data['covariances']
                for level, alpha in zip([1, 2, 3], [0.3, 0.2, 0.1]):
                    self.plot_3d_ellipsoid(ax, mean, cov, colors[i], alpha, level)
            ax.set_title(f'EM Clustering at Iteration {iter_data["iteration"]}')
            ax.set_xlabel('Feature 1')
            ax.set_ylabel('Feature 2')
            ax.set_zlabel('Feature 3')
            ax.legend(*scatter.legend_elements(), title="EM Clusters")
        else:
            ax = self.iteration_canvas.figure.add_subplot(111)
            ax.text(0.5, 0.5, f'Iteration {iter_data["iteration"]}:\nHigh-dimensional data.',
                    horizontalalignment='center', verticalalignment='center')
        self.iteration_canvas.draw()
```

```
        stats_text = f"Iteration {iter_data['iteration']}\n\n"
        stats_text += f"Converged: {iter_data['converged']}\n"
        stats_text += f"Log-Likelihood: {iter_data['log_likelihood']:.4f}\n"
        if not np.isnan(iter_data['silhouette']):
            stats_text += f"Silhouette Score: {iter_data['silhouette']:.4f}\n"
        if not np.isnan(iter_data['ari']):
            stats_text += f"Adjusted Rand Index: {iter_data['ari']:.4f}\n"
        stats_text += "\nCluster Means:\n" + '\n'.join([f"Cluster {i+1}: {mean}" for i, mean in
enumerate(iter_data['means'])])
        stats_text += "\n\nCluster Weights:\n" + '\n'.join([f"Cluster {i+1}: {weight:.4f}" for i, weight in
enumerate(iter_data['weights'])])
        stats_text += "\n\nCluster Covariances:\n" + str(iter_data['covariances'])
        self.iteration_stats.setText(stats_text)

    def export_iteration_pdf(self):
        if not self.iteration_history:
            return

        file_path, _ = QFileDialog.getSaveFileName(self, "Save Iteration PDF", "", "PDF Files (*.pdf)")
        if not file_path:
            return

        with PdfPages(file_path) as pdf:
            for iter_data in self.iteration_history:
                fig = plt.figure(figsize=(10, 12))

                if self.n_features == 2:
                    ax = fig.add_subplot(211)
                    colors = plt.cm.viridis(np.linspace(0, 1, self.n_clusters.value()))
                    scatter = ax.scatter(self.X[:, 0], self.X[:, 1], c=iter_data['labels'], alpha=0.6, cmap='viridis')
                    ax.scatter(iter_data['means'][:, 0], iter_data['means'][:, 1], marker='X', c='red', s=200, label='EM
Centers')
                    for i in range(self.n_clusters.value()):
                        mean = iter_data['means'][i]
                        cov = iter_data['covariances'][i] if self.covar_type.currentText() in ['full', 'diag', 'spherical'] else
iter_data['covariances']
                        if self.covar_type.currentText() == 'spherical':
                            cov = np.eye(2) * cov[i]
                        elif self.covar_type.currentText() == 'diag':
                            cov = np.diag(cov[i])
                        elif self.covar_type.currentText() == 'tied':
                            cov = iter_data['covariances']
                        eigvals, eigvecs = np.linalg.eigh(cov)
                        order = eigvals.argsort()[::-1]
                        eigvals, eigvecs = eigvals[order], eigvecs[:, order]
                        angle = np.degrees(np.arctan2(*eigvecs[:, 0][::-1]))
                        for level, alpha in zip([1, 2, 3], [0.3, 0.2, 0.1]):
                            width, height = 2 * level * np.sqrt(eigvals)
                            ellipse = Ellipse(xy=mean, width=width, height=height, angle=angle,
                                              edgecolor='none', fc=colors[i], alpha=alpha)
                            ax.add_patch(ellipse)
                    ax.set_title(f'EM Clustering at Iteration {iter_data["iteration"]}')
                    ax.set_xlabel('Principal Component 1' if self.is_pca_applied else 'Feature 1')
                    ax.set_ylabel('Principal Component 2' if self.is_pca_applied else 'Feature 2')
                    ax.legend(*scatter.legend_elements(), title="EM Clusters")
                elif self.n_features == 3:
                    ax = fig.add_subplot(211, projection='3d')
```

```
                colors = plt.cm.viridis(np.linspace(0, 1, self.n_clusters.value()))
                scatter = ax.scatter(self.X[:, 0], self.X[:, 1], self.X[:, 2], c=iter_data['labels'], alpha=0.6,
cmap='viridis')
                ax.scatter(iter_data['means'][:, 0], self.iteration_history[0]['means'][:, 1], iter_data['means'][:, 2],
                        marker='X', c='red', s=200, label='EM Centers')
                for i in range(self.n_clusters.value()):
                    mean = iter_data['means'][i]
                    cov = iter_data['covariances'][i] if self.covar_type.currentText() in ['full', 'diag', 'spherical'] else
iter_data['covariances']
                    if self.covar_type.currentText() == 'spherical':
                        cov = np.eye(3) * cov[i]
                    elif self.covar_type.currentText() == 'diag':
                        cov = np.diag(cov[i])
                    elif self.covar_type.currentText() == 'tied':
                        cov = iter_data['covariances']
                    for level, alpha in zip([1, 2, 3], [0.3, 0.2, 0.1]):
                        self.plot_3d_ellipsoid(ax, mean, cov, colors[i], alpha, level)
                ax.set_title(f'EM Clustering at Iteration {iter_data["iteration"]}')
                ax.set_xlabel('Feature 1')
                ax.set_ylabel('Feature 2')
                ax.set_zlabel('Feature 3')
                ax.legend(*scatter.legend_elements(), title="EM Clusters")
            else:
                ax = fig.add_subplot(211)
                ax.text(0.5, 0.5, f'Iteration {iter_data["iteration"]}:\nHigh-dimensional data.',
                        horizontalalignment='center', verticalalignment='center')

            ax_table = fig.add_subplot(212)
            ax_table.axis('off')
            metrics = [
                ['Metric', 'Value'],
                ['Log-Likelihood', f"{iter_data['log_likelihood']:.4f}"],
                ['Silhouette Score', f"{iter_data['silhouette']:.4f}" if not np.isnan(iter_data['silhouette']) else
'N/A'],
                ['Adjusted Rand Index', f"{iter_data['ari']:.4f}" if not np.isnan(iter_data['ari']) else 'N/A'],
                ['Converged', str(iter_data['converged'])]
            ]
            table = ax_table.table(cellText=metrics, loc='center', cellLoc='center')
            table.auto_set_font_size(False)
            table.set_fontsize(8)
            table.scale(1, 1.5)

            fig.tight_layout()
            pdf.savefig(fig)
            plt.close(fig)

        fig = plt.figure(figsize=(10, 12))
        ax = fig.add_subplot(111)
        ax.axis('off')
        eval_results = self.evaluator.get_evaluation_results()
        summary_text = (
            f"EM Algorithm Summary\n\n"
            f"Dataset: {self.dataset_combo.currentText()}\n"
            f"Number of Samples: {self.X.shape[0]}\n"
            f"Number of Features: {self.n_features}{' (PCA reduced)' if self.is_pca_applied else ''}\n"
            f"Number of Clusters: {self.n_clusters.value()}\n"
            f"Total Iterations: {self.em_model.n_iter_}\n"
```

```
            f"Final Log-Likelihood: {self.em_model.score(self.X) * self.X.shape[0]:.4f}\n"
            f"Converged: {self.em_model.converged_}\n"
            f"Runtime (s): {self.em_runtime:.4f}\n"
        )
        if 'EM' in eval_results and 'error' not in eval_results['EM']:
            summary_text += (
                f"\nEvaluation Metrics (EM):\n"
                f"- Silhouette Score: {eval_results['EM']['silhouette']:.4f}\n"
                f"- Davies-Bouldin Index: {eval_results['EM']['davies_bouldin']:.4f}\n"
            )
            if not np.isnan(eval_results['EM']['ari']):
                summary_text += f"- Adjusted Rand Index: {eval_results['EM']['ari']:.4f}\n"
        if 'KMeans' in eval_results and 'error' not in eval_results['KMeans']:
            summary_text += (
                f"\nEvaluation Metrics (K-Means):\n"
                f"- Silhouette Score: {eval_results['KMeans']['silhouette']:.4f}\n"
                f"- Davies-Bouldin Index: {eval_results['KMeans']['davies_bouldin']:.4f}\n"
            )
            if not np.isnan(eval_results['KMeans']['ari']):
                summary_text += f"- Adjusted Rand Index: {eval_results['KMeans']['ari']:.4f}\n"
        ax.text(0.5, 0.5, summary_text, horizontalalignment='center', verticalalignment='center', fontsize=12)
        pdf.savefig(fig)
        plt.close(fig)

def export_csv(self):
    if hasattr(self, 'export_df'):
        file_path, _ = QFileDialog.getSaveFileName(self, "Save CSV", "", "CSV Files (*.csv)")
        if file_path:
            self.export_df.to_csv(file_path, index=False)

def generate_report(self):
    if self.em_model is None:
        return

    report = io.StringIO()
    report.write("# EM Clustering Analysis Report\n\n")
    report.write(f"Generated on: {pd.Timestamp.now().strftime('%Y-%m-%d %H:%M:%S')}\n\n")
    report.write("## Dataset Information\n\n")
    report.write(f"- Dataset Type: {self.dataset_combo.currentText()}\n")
    report.write(f"- Number of Samples: {self.X.shape[0]}\n")
    report.write(f"- Number of Features: {self.n_features}")
    if self.is_pca_applied:
        report.write(f" (PCA reduced from {self.X_original.shape[1]} features)\n")
        report.write(f"- Explained Variance Ratio: {self.pca.explained_variance_ratio_.round(4)}\n")
    else:
        report.write("\n")
    if self.y_true is not None:
        report.write(f"- Number of True Classes: {len(np.unique(self.y_true))}\n")
    report.write("\n")
    report.write("## Algorithm Parameters\n\n")
    report.write(f"- Number of Clusters: {self.n_clusters.value()}\n")
    report.write(f"- Covariance Type: {self.covar_type.currentText()}\n")
    report.write(f"- Initialization Method: {self.init_method.currentText()}\n")
    report.write(f"- Maximum Iterations: {self.max_iter.value()}\n")
    report.write(f"- Convergence Tolerance: {self.tol.value()}\n")
    report.write(f"- Covariance Regularization: {self.reg_covar.value()}\n")
    report.write("\n")
```

```
report.write("## Results\n\n")
report.write(f"- Number of Iterations: {self.em_model.n_iter_}\n")
report.write(f"- Final Log-Likelihood: {self.em_model.score(self.X) * self.X.shape[0]:.4f}\n")
report.write(f"- BIC: {self.em_model.bic(self.X):.4f}\n")
report.write(f"- AIC: {self.em_model.aic(self.X):.4f}\n")
eval_results = self.evaluator.get_evaluation_results()
if 'EM' in eval_results and 'error' not in eval_results['EM']:
    report.write(f"- Silhouette Score: {eval_results['EM']['silhouette']:.4f}\n")
    report.write(f"- Davies-Bouldin Index: {eval_results['EM']['davies_bouldin']:.4f}\n")
    if not np.isnan(eval_results['EM']['ari']):
        report.write(f"- Adjusted Rand Index: {eval_results['EM']['ari']:.4f}\n")
report.write(f"- Runtime (s): {self.em_runtime:.4f}\n")
report.write("\n")
report.write("## Iteration History\n\n")
for iter_data in self.iteration_history:
    report.write(f"### Iteration {iter_data['iteration']}\n")
    report.write(f"- Log-Likelihood: {iter_data['log_likelihood']:.4f}\n")
    if not np.isnan(iter_data['silhouette']):
        report.write(f"- Silhouette Score: {iter_data['silhouette']:.4f}\n")
    if not np.isnan(iter_data['ari']):
        report.write(f"- Adjusted Rand Index: {iter_data['ari']:.4f}\n")
    report.write(f"- Converged: {iter_data['converged']}\n")
    report.write("- Cluster Means:\n")
    for i, mean in enumerate(iter_data['means']):
                    report.write(f"  - Cluster {i+1}: {mean}\n")
    report.write("- Cluster Weights:\n")
    for i, weight in enumerate(iter_data['weights']):
        report.write(f"  - Cluster {i+1}: {weight:.4f}\n")
    report.write("\n")
report.write("## Cluster Information\n\n")
cluster_sizes = np.bincount(self.em_labels, minlength=self.n_clusters.value())
report.write("### Cluster Sizes\n\n")
for i, size in enumerate(cluster_sizes):
    report.write(f"- Cluster {i+1}: {size} samples ({size/len(self.em_labels):.2%})\n")
report.write("\n")
report.write(f"### Cluster Means ({'PCA Space' if self.is_pca_applied else 'Feature Space'})\n\n")
for i in range(self.n_clusters.value()):
    report.write(f"- Cluster {i+1}: {', '.join([f'{x:.4f}' for x in self.em_model.means_[i]])}\n")
report.write("\n")
report.write("### Cluster Weights\n\n")
for i, weight in enumerate(self.em_model.weights_):
    report.write(f"- Cluster {i+1}: {weight:.4f}\n")
report.write("\n")
if self.compare_kmeans.isChecked() and self.kmeans_labels is not None:
    report.write("## Comparison with K-Means\n\n")
    report.write(f"- K-Means Iterations: {self.kmeans_model.n_iter_}\n")
    report.write(f"- K-Means Inertia: {self.kmeans_model.inertia_:.4f}\n")
    if 'KMeans' in eval_results and 'error' not in eval_results['KMeans']:
        report.write(f"- K-Means Silhouette Score: {eval_results['KMeans']['silhouette']:.4f}\n")
        report.write(f"- K-Means Davies-Bouldin Index: {eval_results['KMeans']['davies_bouldin']:.4f}\n")
        if not np.isnan(eval_results['KMeans']['ari']):
            report.write(f"- K-Means Adjusted Rand Index: {eval_results['KMeans']['ari']:.4f}\n")
    report.write(f"- K-Means Runtime (s): {self.kmeans_runtime:.4f}\n")
    report.write("\n")
    confusion = pd.crosstab(pd.Series(self.em_labels, name='EM'),
                            pd.Series(self.kmeans_labels, name='KMeans'))
    agreement = np.sum(np.max(confusion.values, axis=0)) / np.sum(confusion.values)
```

```
            report.write(f"- Agreement between EM and K-Means: {agreement:.2%}\n")
        report.write("\n")
        report.write("## Conclusion\n\n")
        report.write("The EM algorithm identified clusters with the following characteristics:\n\n")
        for i in range(self.n_clusters.value()):
            report.write(f"### Cluster {i+1}\n\n")
            report.write(f"- Size: {cluster_sizes[i]} samples ({cluster_sizes[i]/len(self.em_labels):.2%})\n")
            report.write(f"- Weight: {self.em_model.weights_[i]:.4f}\n")
            report.write(f"- Center ({'PCA' if self.is_pca_applied else 'Features'}): {', '.join([f'{x:.4f}' for x in
self.em_model.means_[i]])}\n")
            report.write("\n")

        file_path, _ = QFileDialog.getSaveFileName(self, "Save Report", "", "Markdown Files(*.md)")
        if file_path:
            with open(file_path, 'w') as f:
                f.write(report.getvalue())

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = EMApp()
    window.show()
    sys.exit(app.exec_())
```

Додаток Г – Графічна частина

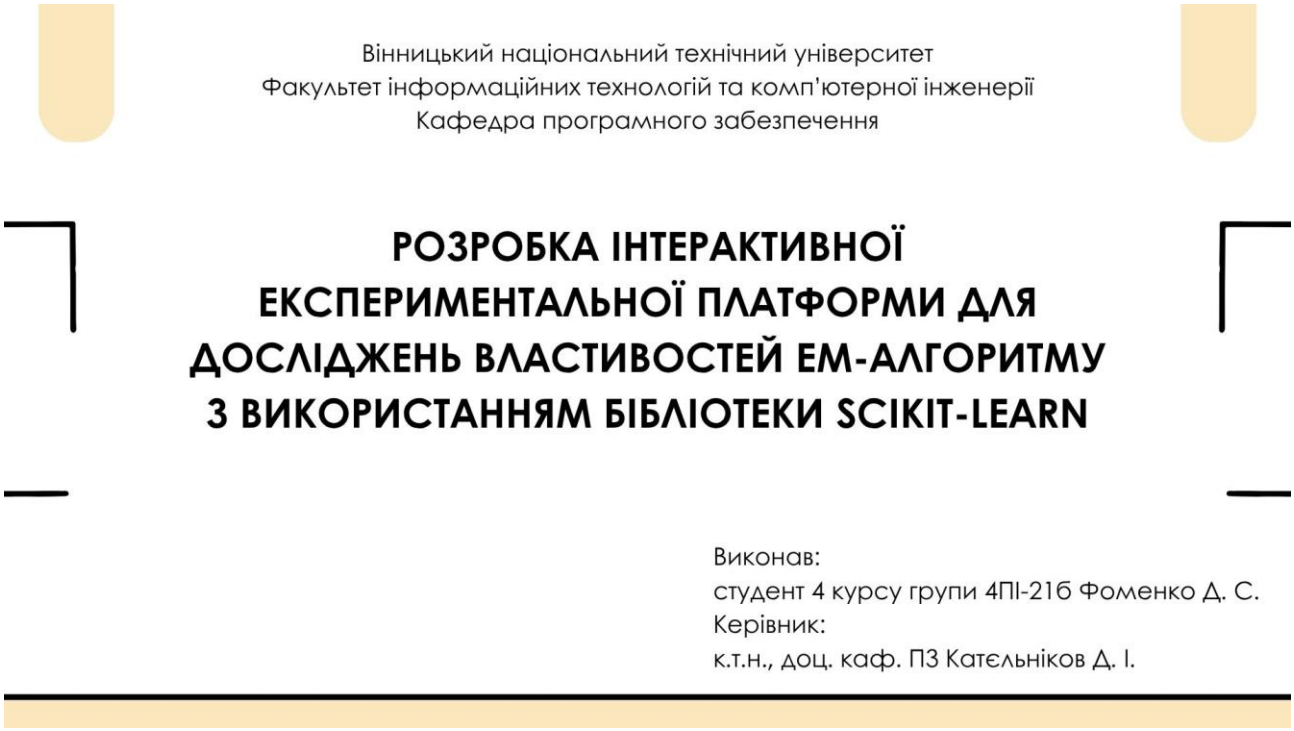


Рисунок Г.1 – Титульний слайд

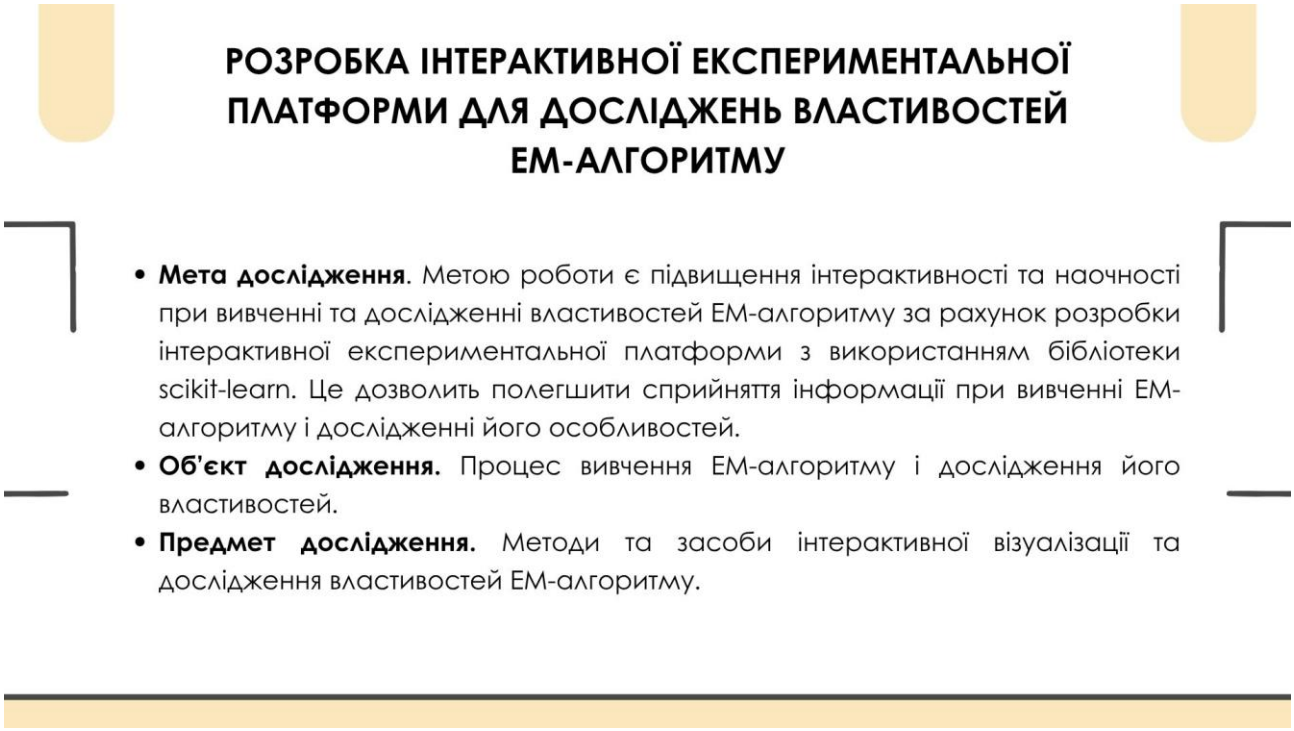


Рисунок Г.2 – Мета, об’єкт і предмет дослідження

РОЗРОБКА ІНТЕРАКТИВНОЇ ЕКСПЕРИМЕНТАЛЬНОЇ ПЛАТФОРМИ ДЛЯ ДОСЛІДЖЕНЬ ВЛАСТИВОСТЕЙ ЕМ-АЛГОРИТМУ

- **Новизна отриманих результатів.**

1. Подальшого розвитку отримав метод інтерактивної візуалізації ЕМ-алгоритму, який, на відміну від існуючих рішень, забезпечує динамічне відображення проміжних результатів кожної ітерації з можливістю зміни параметрів у реальному часі. Це дозволяє розглядати більше можливих варіантів без потреби повторного запуску алгоритму.
2. Подальшого розвитку отримав метод оцінки якості кластеризації, який, на відміну від існуючих рішень, передбачає використання комбінації зовнішніх та внутрішніх метрик з їх динамічною візуалізацією, що дозволяє різнобічно аналізувати якість кластеризації.
3. Подальшого розвитку отримав метод генерації синтетичних даних для тестування ЕМ-алгоритму, у якому, на відміну від існуючих, використано параметричні моделі з можливістю інтерактивного налаштування, що дозволило розширити спектр експериментальних сценаріїв.

- **Практична цінність отриманих результатів.** Полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмну платформу для інтерактивного дослідження властивостей ЕМ-алгоритму.

Рисунок Г.3 – Наукова новизна та практична цінність

ЗАДАЧІ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

- провести аналіз існуючих методів і засобів візуалізації та експериментального дослідження ЕМ-алгоритму для визначення напрямків підвищення їх інтерактивності та наочності;
- запропонувати нові:

1. методи підвищення інтерактивності візуалізації процесу роботи ЕМ-алгоритму;
2. методи підвищення наочності та інформативності представлення результатів роботи ЕМ-алгоритму;

- розробити програмні компоненти та інтерактивну експериментальну платформу на основі запропонованих методів;
- провести експериментальні дослідження розроблених засобів візуалізації та аналізу ЕМ-алгоритму.

Рисунок Г.4 – Задачі бакалаврської кваліфікаційної роботи

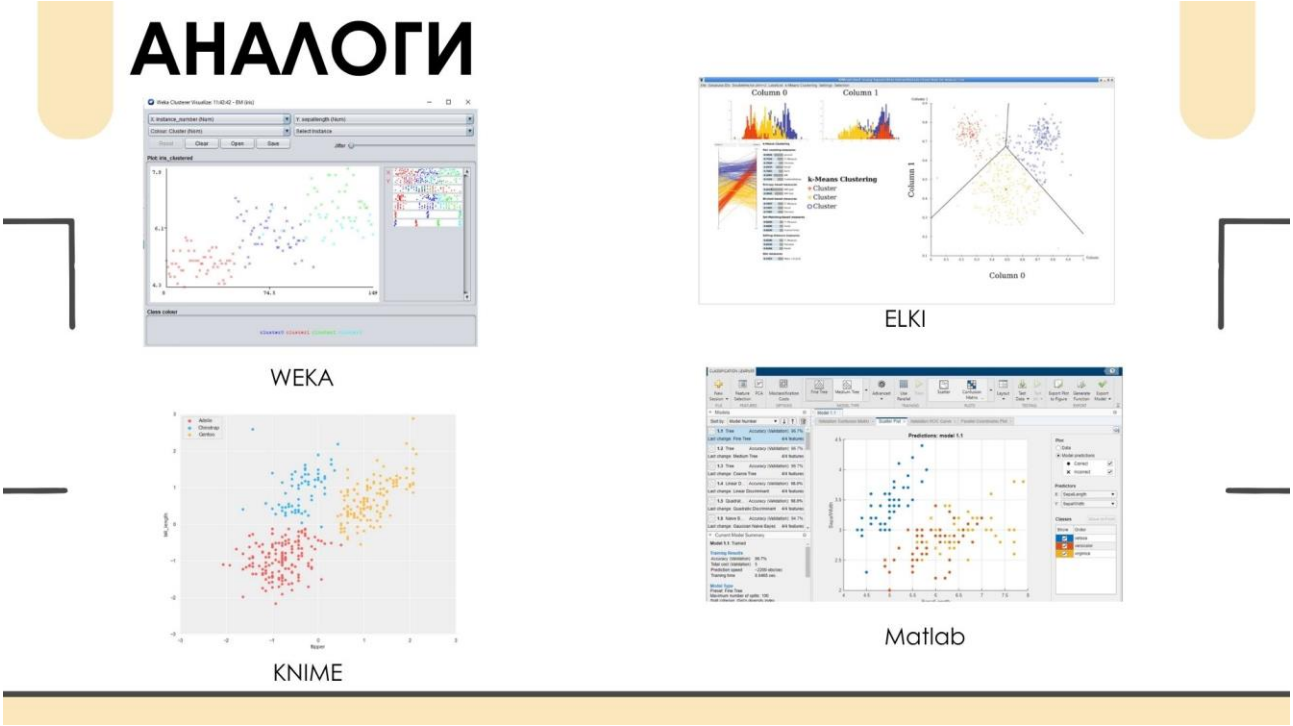


Рисунок Г.5 – Аналоги

ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ

	ELKI	WEKA	MATLAB	KNIME	Власна розробка
Інтерактивне налаштування синтетичних даних	1	1	1	1	1
Візуалізація проміжних ітерацій EM-алгоритму	0	1	1	0	1
Автоматична обробка пропущених значень у даних	0	1	0	1	1
Обчислення внутрішніх та зовнішніх метрик якості	1	1	1	1	1
Порівняння з альтернативним алгоритмом кластеризації	1	1	1	1	1
Експорт результатів у PDF з ітераційними даними	0	0	1	1	1
Підтримка 3D-візуалізації кластерів	0	0	1	0	1
Сумарний коефіцієнт	43%	71%	86%	71%	100%

Рисунок Г.6 – Порівняльний аналіз аналогів

ОБРАНІ ЗАСОБИ ДЛЯ РЕАЛІЗАЦІЇ

•**Python** - універсальність, зрозумілий синтаксис, широка бібліотечна база для обробки даних та машинного навчання;
•**PyQt5** - створення графічного інтерфейсу з широким набором віджетів, кросплатформність, інтеграція з Matplotlib;
•**scikit-learn** - готові інструменти для реалізації EM-алгоритму та кластеризації, чітка документація, сумісність з NumPy;
•**Matplotlib** - створення інтерактивних візуалізацій (діаграми розсіювання, гаусівські еліпси), інтеграція з PyQt5, гнучкість налаштувань;
•**NumPy** - високошвидкісні операції з масивами та матрицями, основа для scikit-learn і Matplotlib, простота використання;
Причини вибору: забезпечення модульності системи, продуктивності обчислень, зручності розробки та інтеграції всіх компонентів у єдину платформу для аналізу даних і кластеризації.

Рисунок Г.7 – Порівняльний аналіз аналогів

ФОРМУЛИ РОБОТИ EM-АЛГОРИТМУ

1. E-крок (крок очікування)

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}, \quad (2.1)$$

де π_k – вага k-ї компоненти, $\mathcal{N}(x_n|\mu_k, \Sigma_k)$ – гаусівська густина ймовірності для точки x_n з середнім μ_k і коваріацією Σ_k, μ_k – середнє k-ї компоненти, Σ_k – коваріаційна матриця k-ї компоненти, $\mathcal{N}$ – кількість точок даних, K – кількість кластерів.

2. M-крок (крок максимізації)

$$\pi_k = \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk}, \quad (2.2)$$

де π_k – нова вага k-ї компоненти, γ_{nk} – відповідальність точки x_n для k-ї компоненти, N – кількість точок даних.

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}}, \quad (2.3)$$

де μ_k – нове середнє k-ї компоненти, x_n – точка даних, γ_{nk} – відповідальність.

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma_{nk} (x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma_{nk}}, \quad (2.4)$$

де Σ_k – нова коваріаційна матриця k-ї компоненти, $(x_n - \mu_k)(x_n - \mu_k)^T$ – зовнішній добуток вектора відхилення, γ_{nk} – відповідальність.

3. Логарифмічна правдоподібність

$$\ln p(X|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right), \quad (2.5)$$

де X – набір даних, π – вектор ваг, μ – вектор середніх, Σ – набір коваріаційних матриць, $\mathcal{N}(x_n|\mu_k, \Sigma_k)$ – гаусівська густина.

Рисунок Г.8 – Формули роботи EM-алгоритму

БЛОК-СХЕМА РОБОТИ ПРОГРАМНОГО ЗАСТОСУНКУ

У процесі розробки інтерактивної експериментальної платформи для досліджень властивостей EM-алгоритму було створено блок-схему, яка відображає логіку роботи системи.

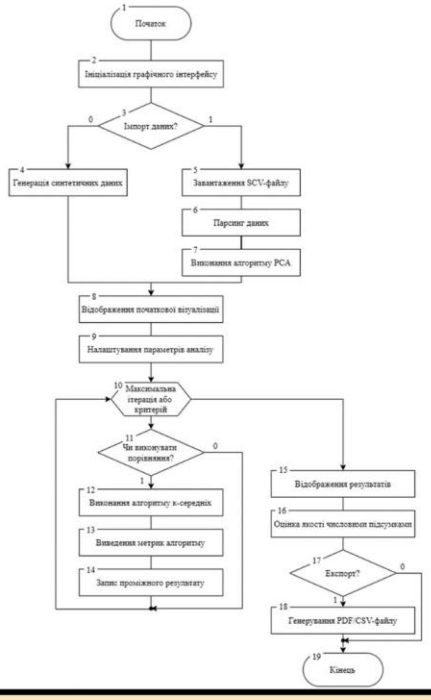


Рисунок Г.9 – Блок-схема роботи програмного застосунку

ФОРМУЛИ ГЕНЕРАЦІЇ СИНТЕТИЧНИХ ДАНИХ

Формула генерації плям

$$x_n \sim \mathcal{N}(\mu_k, \sigma_k^2 I), \tag{2.6}$$

де μ_k – центр k-ї компоненти, σ_k– стандартне відхилення, що визначається параметром шуму, I – одинична матриця, $\mathcal{N}$ – гаусівський розподіл.

Формула генерації місяців

$$x_n = \begin{bmatrix}\cos(\theta_n)\\ \sin(\theta_n)\end{bmatrix} + \epsilon_n, \quad \theta_n \in [0, \pi], \tag{2.7}$$

де θ_n – кут для півмісяця, $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ – гаусівський шум, σ – параметр шуму.

Формула генерації кілець

$$x_n = r\begin{bmatrix}\cos(\theta_n)\\ \sin(\theta_n)\end{bmatrix} + \epsilon_n, \tag{2.8}$$

де r – радіус кільця, що залежить від параметра factor, $\theta_n \in [0,2\pi]$ – кут, $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ – гаусівський шум, σ – параметр шуму.

Рисунок Г.10 – Формули для генерації синтетичних даних

БЛОК-СХЕМА РОБОТИ МОДУЛЯ ОЦІНКИ ЯКОСТІ КЛАСТЕРИЗАЦІЇ

Метод оцінки якості кластеризації забезпечує кількісний аналіз результатів EM-алгоритму та, за потреби, алгоритму K-середніх у програмному забезпеченні для аналізу даних. Модуль використовує три метрики: коефіцієнт силуету, індекс Девіса-Болдіна та скоригований індекс Ренда, що дозволяють оцінити якість кластеризації з різних аспектів. Метод інтегрується з системою, обробляючи дані та мітки кластерів для обчислень і збереження результатів для відображення чи експорту.

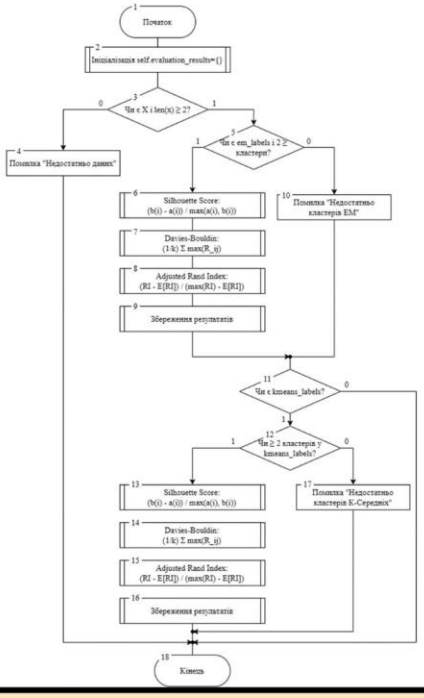


Рисунок Г.11 – Блок-схема роботи модуля оцінки якості кластеризації

Діаграма класу PDFExportModule

UML-діаграма класу ілюструє структуру PDFExportModule і його зв'язок із системою. Клас містить атрибути, такі як батьківський об'єкт, історія ітерацій, матриця даних, кількість ознак, кількість кластерів, тип коваріації, чи застосовано PCA, назва набору даних, EM-модель, час виконання EM, оцінювач якості. Основний метод export_to_pdf виконує експорт, а допоміжні методи забезпечують модульність. Модуль є частиною головного класу інтерфейсу, отримуючи дані після кластеризації.

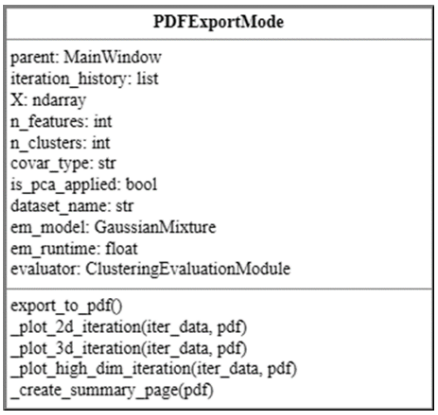


Рисунок Г.12 – Діаграма класу PDFExportModule

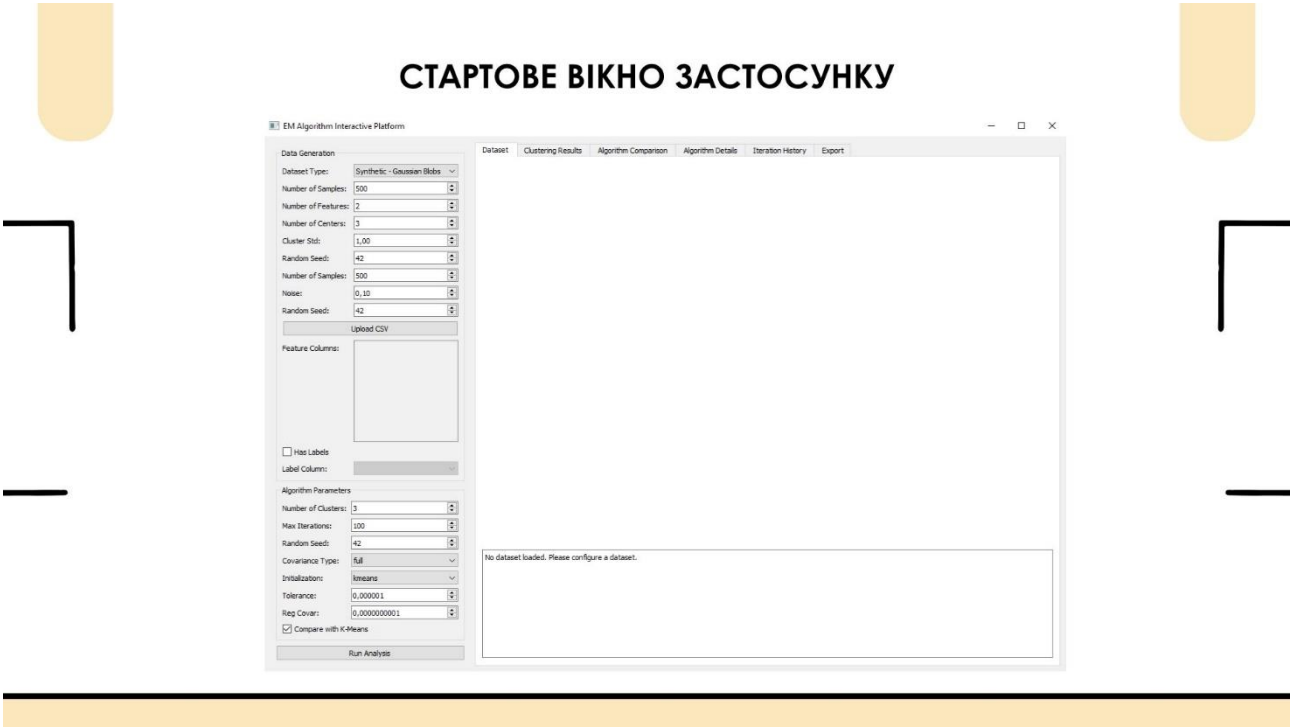


Рисунок Г.13 – Стартове вікно застосунку

Рисунок Г.14 – Приклади згенерованих синтетичних даних

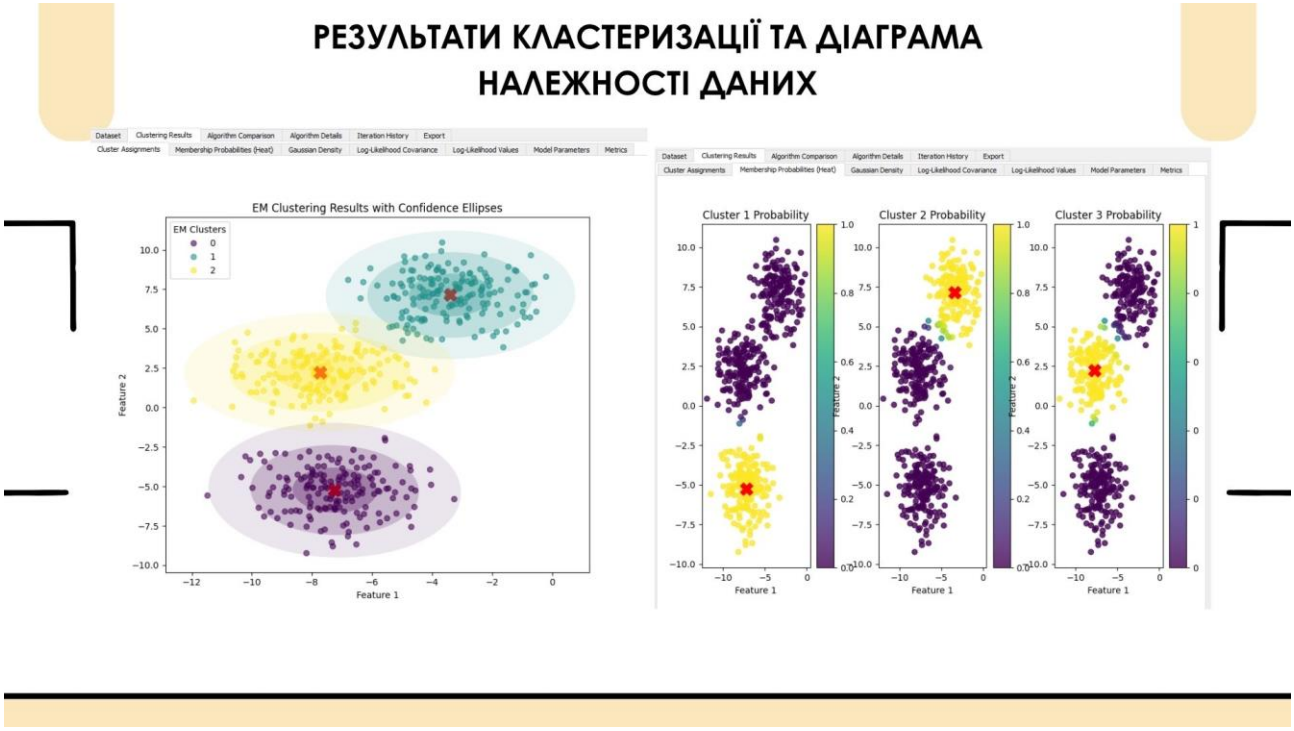
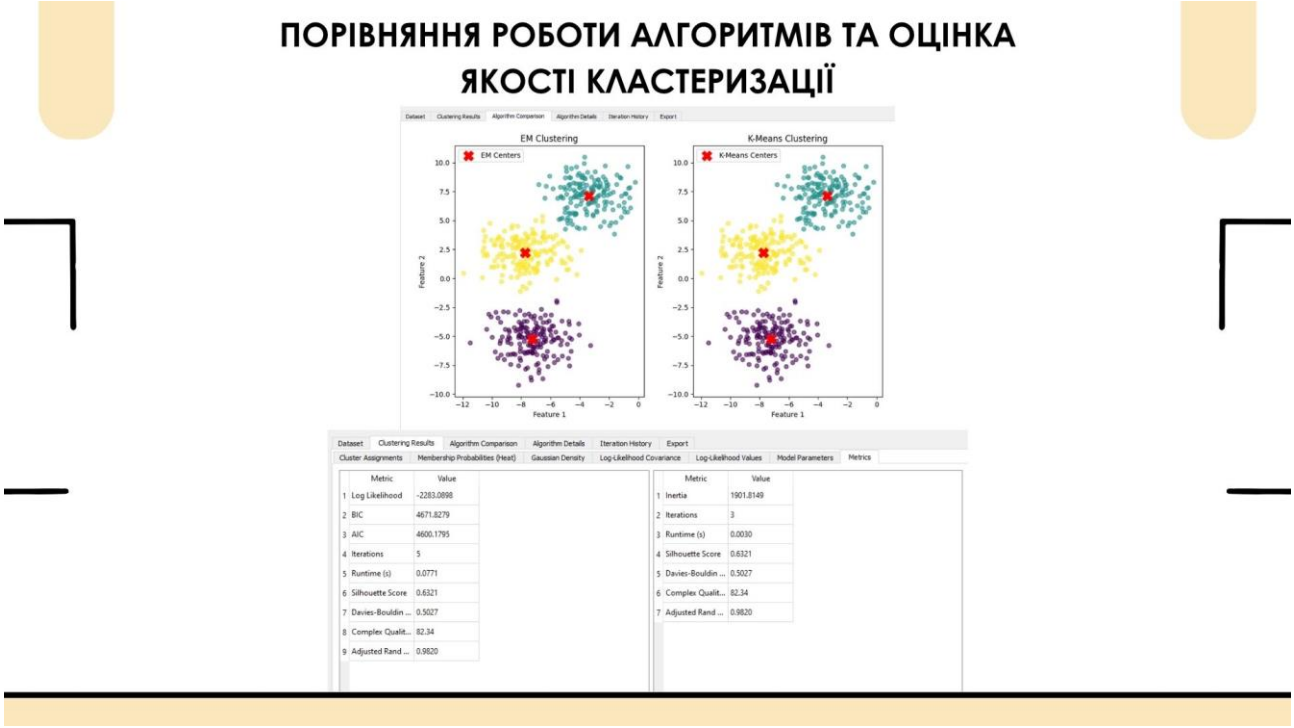


Рисунок Г.15 – Результати кластеризації та діаграма належності даних



	Metric	Value
1	Log Likelihood	-2283.0898
2	BIC	4671.8279
3	AIC	4600.1795
4	Iterations	5
5	Runtime (s)	0.0771
6	Silhouette Score	0.6321
7	Davies-Bouldin ...	0.5027
8	Complex Qualit...	82.34
9	Adjusted Rand ...	0.9820

	Metric	Value
1	Inertia	1901.8149
2	Iterations	3
3	Runtime (s)	0.0030
4	Silhouette Score	0.6321
5	Davies-Bouldin ...	0.5027
6	Complex Qualit...	82.34
7	Adjusted Rand ...	0.9820

Рисунок Г.16 – Порівняння роботи алгоритмів та метрики оцінки якості кластеризації

ДОВІДКА З ОПИСОМ РОБОТИ АЛГОРИТМУ

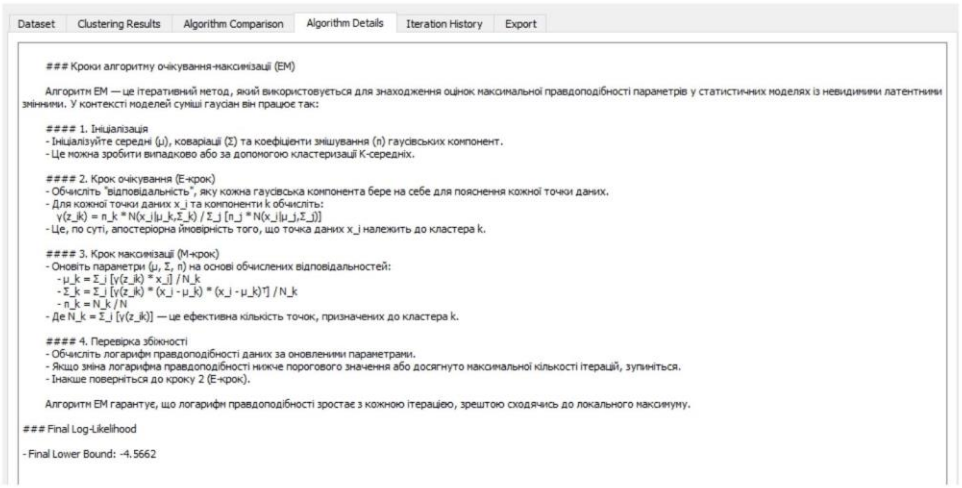


Рисунок Г.17 – Довідка з описом роботи алгоритму

ПУБЛІКАЦІЇ

- Основні результати досліджень опубліковано в матеріалах конференції «LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (2025)»

- Фоменко. Д. С. ЕМ-алгоритм як метод оптимізації у задачах кластеризації. Матеріали LIV Всеукраїнської науково-технічної конференції професорсько-викладацького складу, науковців, аспірантів та студентів підрозділів університету з участю працівників підприємств (НТКП ВНТУ-2025). Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24503/

Рисунок Г.18 – Публікації

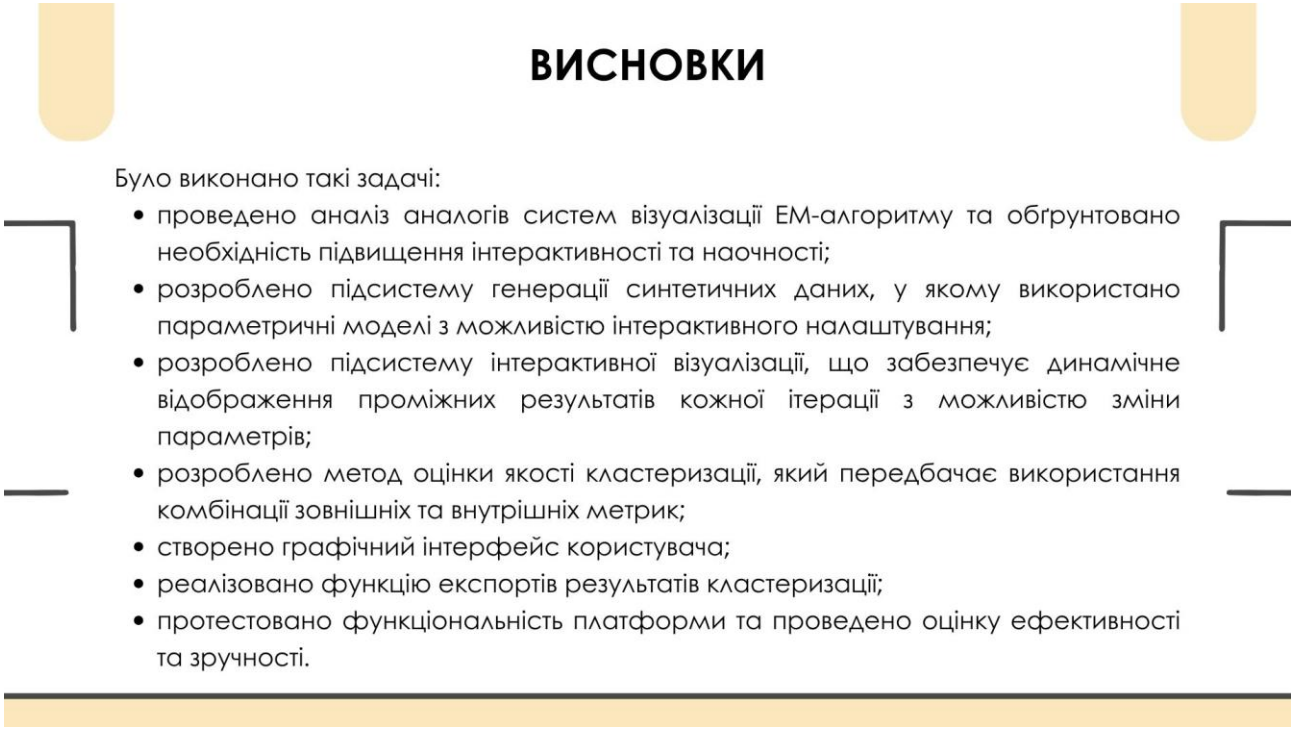


Рисунок Г.19 – Висновки

Рисунок Г.20 – Кінець презентації