Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи

Магістр

на тему: Експериментальні обчислювальні дослідження самоподібності часових рядів

за освітньою програмою 12 Інженерія програмного забезпечення

зі спеціальності: 121 Інженерія програмного забезпечення

Виконав: студент групи: ПЗ2421 Данило УЛЬЯНЧЕНКО

Керівник: Віктор ШИНКАРЕНКО

Нормоконтролер: Світлана ВОЛКОВА

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

Дніпро – 2026

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Computer Technologies and Systems

Computer information technologies

Explanatory Note

Master's degree

on the topic: Experimental computational studies of self-similarity of time series

according to educational curriculum 12 Software Engineering

in the Speciality: 121 Software Engineering

Done by the student of the group: PZ2421 Danylo ULYANCHENKO

Scientific Supervisor: Victor SHINKARENKO

Normative controller : Svitlana VOLKOVA

Dnipro – 2026

Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет: Комп'ютерні технології і системи

Кафедра: Комп'ютерні інформаційні технології

Рівень вищої освіти: магістр

Освітня програма: Інженерія програмного забезпечення

Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри КІТ

Вадим ГОРЯЧКІН

Дата

ЗАВДАННЯ

на кваліфікаційну роботу Магістра

студенту Ульянченку Данилу Сергійовичу

1. Тема роботи: Експериментальні обчислювальні дослідження самоподібності часових рядів

Керівник роботи: Шинкаренко Віктор Іванович, д.т.н., професор.

затверджені наказом від 02.10.2025 р. № 1401ст

2. Строк подання студентом роботи: 10.01.2026 р.

3. Вихідні дані до роботи: Набори даних (синтетичні фрактальні ряди та реальні стохастичні часові ряди інструменти розробки

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):

4.1 Аналітична частина: Вивчення самоподібності, методів оцінки фрактало-подібної розмірності, статистичних інструментів (довірчі інтервали, t-тест).

4.2 Основна частина: Розробка ПЗ (MVVM, алгоритми пошуку послідовностей), експерименти з варіюванням допуску, аналіз результатів, рекомендації щодо оптимальних параметрів.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір інформації та аналіз літературних джерел.	05.09.25	5%
2	Розробка теоретичних основ і алгоритмів.	06.09.25	10%
3	Проєктування архітектури програми та інтерфейсу.	09.09.25	20%
4	Реалізація програмного забезпечення та тестування.	28.09.25	30%
5	Проведення експериментів і збір даних.	20.11.25	60%
6	Аналіз результатів, написання висновків і реферату.	11.12.26	100%
7	Подання кваліфікаційної роботи до кафедри.	09.01.26	
8	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії.	19.01.26	

Студент _____ Данило УЛЬЯНЧЕНКО

Керівник роботи _____ Віктор ШИНКАРЕНКО

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 83 с. з додатками 214 с., 26 рис., 18 табл., 4 додатки, 30 джерел.

Об’єкт розробки – самоподібність часових рядів для підвищення ефективності аналізу даних у фінансах, біомедицині та енергетиці.

Мета роботи – розробка та реалізація програмного комплексу для оцінки фракталоподібної розмірності часових рядів методом схожих послідовностей з підтримкою діапазонного розрахунку, режимів допуску та статистичної оцінки. У першому розділі, на основі літературних джерел, проведено огляд понять самоподібності, фрактальної геометрії, показника Хьорста та методів обчислення розмірності, включаючи запропоновану оцінку фракталоподібності. У другому розділі детально описано архітектуру програми (MVVM, .NET 8, WPF), модулі обробки даних, обчислень (Calculation), візуалізації (OxyPlot) та інтерфейсу, з обґрунтуванням проєктних рішень для зручності та ефективності. У третьому розділі представлено алгоритми пошуку схожих послідовностей (з абсолютним/відносним допуском), обчислення фракталоподібної розмірності за регресією log(M) від log(1/δ) та розрахунку 95%-вих довірчих інтервалів за t-розподілом Стьюдента. У четвертому розділі проведено експерименти на тестових наборах: синтетичних фрактальних (350 000 точок) та реальних стохастичних рядів, з варіюванням допуску 0.1–4.6%, порівнянням режимів та обчисленням інтервалів, що показало стабільність при 1.1–2.1%. У п’ятому розділі проаналізовано результати: залежність D від допуску нелінійна, з мінімумом варіації при середніх значеннях; запропоновано оцінку фракталоподібності з порогами ($S < 0.2$ — висока). Результати роботи можуть бути використані для створення систем аналізу самоподібності в транспорті, промисловості та освіті, для моніторингу потоків, прогнозування та діагностики.

Ключові слова: САМОПОДІБНІСТЬ, ФРАКТАЛОПОДІБНА РОЗМІРНІСТЬ, СХОЖІ ПОСЛІДОВНОСТІ, ЧАСОВІ РЯДИ, ДОПУСК, ДОВІРЧІ ІНТЕРВАЛИ, MVVM, OXYPLOT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Перелік основних символів та абревіатур, використаних у роботі:

D – фракталоподібна розмірність (fractal-like dimension);

δ – допуск (tolerance), параметр схожості послідовностей;

M – кількість схожих послідовностей для масштабу δ;

log(M) – натуральний логарифм кількості схожих послідовностей;

log(1/δ) – натуральний логарифм зворотного допуску (масштаб);

H – показник Хьорста (Hurst exponent), міра самоподібності;

S – середнє значення фракталоподібної розмірності (mean);

σ – стандартне відхилення (standard deviation);

Δ – ширина довірчого інтервалу (half-width);

t – критичне значення t-розподілу Стьюдента;

n – розмір вибірки (number of samples);

df – ступені свободи (degrees of freedom);

p – p-значення (p-value) у статистичних тестах;

Cohen's d – розмір ефекту (effect size) у t-тесті;

TolerancePercentage – відсотковий допуск (percentage tolerance).

ВСТУП

1 Актуальність теми

Дослідження самоподібності часових рядів є надзвичайно важливим напрямком у сучасній обробці сигналів та аналізі складних систем. Зі зростанням обсягів даних у фінансових ринках, біомедицині, енергетиці, транспорті та кліматології виникає потреба в надійних методах виявлення прихованих закономірностей. Самоподібність (фракталоподібність) є однією з ключових характеристик таких процесів, що дозволяє розрізняти шум, детерміновані та синтетичні компоненти. На сьогодні відсутні доступні програмні інструменти, які б поєднували автоматизований пошук схожих послідовностей з точним та відносним допуском, обчислення фракталоподібної розмірності та статистичну оцінку результатів. Перспективи використання таких інструментів відкривають нові можливості для моніторингу транспортних потоків, прогнозування енергоспоживання, діагностики біомедичних сигналів та аналізу фінансових ринків.

2 Мета і завдання дослідження

Основною метою роботи є розробка та експериментальне дослідження програмного комплексу для оцінки самоподібності часових рядів з урахуванням різних режимів допуску та статистичної значущості результатів.

Для досягнення мети вирішено такі завдання:

- вивчити теоретичні основи самоподібності та методи оцінки фракталоподібної розмірності;
- розробити алгоритми пошуку схожих послідовностей з точним та відносним допуском;
- реалізувати програмний комплекс ECSSTS мовою C# (.NET 8, WPF, MVVM, OxyPlot, MathNet.Numerics);
- провести масштабне експериментальне дослідження на синтетичних фрактальних (100 реалізацій) та реальних не фрактальних рядах (понад 30 000 значень);

- визначити оптимальний діапазон параметрів допуску та оцінити статистичну значущість результатів за допомогою 95 % довірчих інтервалів та t-тесту.

3 Об’єкт і предмет дослідження

Об’єкт дослідження: процеси аналізу самоподібність часових рядів реальних та синтетичних процесів.

Предмет дослідження: методи та алгоритми виявлення схожих послідовностей з точним та відносним допуском, обчислення фракталоподібної розмірності та статистичної оцінки її стабільності.

4 Методи дослідження

Для досягнення поставлених цілей використано:

- теоретичний аналіз літератури з фрактальної геометрії та обробки сигналів;
- розробку та програмну реалізацію алгоритмів пошуку схожих послідовностей з паралельними обчисленнями;
- масштабне комп’ютерне експериментування (понад 100 запусків на синтетичних та реальних даних);
- статистичні методи: t-розподіл Стьюдента, довірчі інтервали, t-тест, ефект Коена;
- порівняльний аналіз результатів у різних режимах допуску та при нормалізації даних.

5 Наукова новизна

Наукова новизна полягає у розробці модифікованого методу підрахунку схожих послідовностей з підтримкою відносного допуску, який забезпечує стійкість до масштабування та зсуву сигналу, а також у встановленні оптимального діапазону допуску 1.1–2.1 % (відносний режим), що дає мінімальну дисперсію ($\sigma < 0.03$) та максимальну розрізнювальну здатність між фрактальними та

нефрактальними рядами. Вперше на великих вибірках (понад 30 000 значень) доведено статистичну значущість відмінностей ($p < 0.001$, Cohen’s $d > 3.5$) та відсутність перетинів 95 % довірчих інтервалів між групами. Запропонований програмний комплекс ECSSTS є готовим універсальним інструментом для експериментальних досліджень самоподібності часових рядів у різних галузях.

1 ТЕОРЕТИЧНІ ОСНОВИ САМОПОДІБНОСТІ ЧАСОВИХ РЯДІВ

1.1 Поняття самоподібності та фрактальної геометрії

Самоподібність є фундаментальною властивістю фрактальних об'єктів, яка проявляється в тому, що частини об'єкта за певного масштабування мають структуру, подібну до цілого. У контексті часових рядів самоподібність означає, що статистичні характеристики ряду, такі як середнє значення, дисперсія чи автокореляція, залишаються незмінними при зміні часового масштабу. Фрактальна геометрія, розроблена Бенуа Мандельбротом [1],[13], надає інструменти для аналізу таких об'єктів через поняття фрактальної розмірності, яка кількісно описує складність структури часового ряду.

Основні аспекти самоподібності та фрактальної геометрії:

- визначення самоподібності: часовий ряд вважається самоподібним, якщо його фрагменти при масштабуванні мають схожі статистичні характеристики, такі як середнє значення, дисперсія чи спектральна щільність;
- типи самоподібності: розрізняють точну (детерміновану) та статистичну самоподібність, остання з яких частіше зустрічається в реальних даних;
- точна самоподібність: характеризується детермінованими структурами, де кожен фрагмент є точною копією цілого при масштабуванні (наприклад, крива Коха або трикутник Серпінського);
- статистична самоподібність: властива реальним часовим рядам, де статистичні характеристики (математичне сподівання, дисперсія) зберігаються при масштабуванні (наприклад, фінансові ринки чи біомедичні сигнали);
- показник Хьорста (H): пов'язаний із фрактальною розмірністю через формулу ($D = 2 - H$), де ($H \in (0,1)$) характеризує ступінь самоподібності. Для ($H = 0.5$) ряд відповідає випадковому блуканню, для ($H > 0.5$) – персистентним процесам;
- застосування: аналіз самоподібності використовується для моделювання фінансових ринків, аналізу біомедичних сигналів (наприклад, ЕКГ) та прогнозування природних явищ.

Самоподібність (self-similarity) – це властивість об’єкта або процесу, за якої його частини статистично або точно подібні до цілого при зміні масштабу спостереження. У реальних процесах (фінансові котирування, біомедичні сигнали, енергоспоживання, кліматичні дані) це проявляється у повторюваності характерних патернів на різних масштабах [3]. На відміну від класичних гладких процесів (детермінованих або гаусівських), самоподібні часові ряди не стають «простішими» при збільшенні масштабу: їхня складність зберігається на всіх рівнях. Це явище лежить в основі фрактальної геометрії, започаткованої Бенуа Мандельбротом у 1970-х роках. Фрактали – це геометричні об’єкти, розмірність яких (фрактальна, або гаусдорфова розмірність) перевищує топологічну розмірність і, як правило, не є цілим числом [2].

1.2 Запропонована оцінка фракталоподібності

У роботі використовується термін «фракталоподібна розмірність» (fractal-like dimension), оскільки реальні часові ряди рідко є строго самоподібними на всіх масштабах, а демонструють самоподібність лише в певному діапазоні довжин послідовностей. Отримана оцінка D є ефективною характеристикою ступеня самоподібності в обраному діапазоні масштабів і дозволяє кількісно порівнювати різні процеси, виявляти наявність або відсутність фрактальних властивостей, а також оцінювати вплив параметрів вимірювання на стабільність результату.

Основні особливості запропонованої оцінки:

- використання методу схожих послідовностей: оцінка базується на підрахунку кількості схожих фрагментів M(δ) для різних масштабів δ з подальшою логарифмічною регресією;
- обмежений діапазон масштабів: самоподібність оцінюється лише в діапазоні, де метод дає стабільні результати (наприклад, TolerancePercentage = 1.1–2.1%);
- врахування режимів допуску: відносний режим адаптується до амплітуди ряду, абсолютний – забезпечує фіксовану чутливість;

- статистична оцінка: побудова довірчих інтервалів для підтвердження надійності D;
- практична інтерпретація: значення D ≈ 1.0 вказує на детерміновану структуру, D ≈ 1.5 – на типову фрактальну самоподібність, D ≈ 2.0 – на шумоподібну поведінку.

Запропонована оцінка дозволяє не тільки виявляти фракталоподібність, але й кількісно порівнювати ряди за ступенем самоподібності, що робить її цінним інструментом для аналізу реальних даних у фінансах, енергетиці та біомедицині.

Запропонована оцінка є практичним компромісом між теоретичною строгістю та застосовністю до реальних даних, що робить її ефективною для експериментальних досліджень самоподібності часових рядів.

1.3 Методи обчислення фрактальної розмірності

Фрактальна розмірність часового ряду може бути оцінена різними методами, кожен з яких має свої переваги та обмеження. Основні методи включають аналіз масштабування, методи підрахунку (box-counting), аналіз хвилькового перетворення та методи на основі пошуку схожих послідовностей.

Фракталоподібна розмірність є основною кількісною характеристикою самоподібності часових рядів. Існує багато методів її оцінки, однак у роботі використано та досліджено саме ті, що найкраще підходять для дискретних одновимірних рядів.

1. Класичний метод box-counting (коробковий метод)

Найпоширеніший підхід. Часовий ряд накладається на сіткою «коробок» розміром ε, підраховується кількість коробок N(ε), що містять хоча б одну точку. Далі будується залежність

$$\log N(\varepsilon) = -D \cdot \log \varepsilon + C \qquad (1.1)$$

і розмірність D визначається як нахил прямої.

Переваги: простота, стійкість.

Недоліки: чутливість до вибору діапазону ε, погано працює на коротких рядах.

2. Метод Hurst (R/S-аналіз)

Оцінює параметр Херста H, а розмірність обчислюється як D = 2 – H [5].

Переваги: добре виявляє довготривалу пам’ять.

Недоліки: дає лише одне число для всього ряду, не враховує можливу мультифрактальність.

3. Метод варіації (Variation method) та Detrended Fluctuation Analysis (DFA)

Видаляють тренд і оцінюють флуктуації F(s) залежно від масштабу s [4]. Розмірність пов’язана з показником степеня α: D ≈ 2 – α.

4. Метод підрахунку схожих послідовностей (використаний у роботі)

Розроблений та вдосконалений спеціально для цієї роботи варіант box-counting, адаптований до часових рядів:

«Розмір коробки» ε замінюється на довжину послідовності δ.

Кількість «зайнятих коробок» N(ε) замінюється на загальну кількість схожих фрагментів M(δ) при заданому допуску).

Будується залежність

$$\log M(\delta) = -D \cdot \log \delta + C \quad \text{або} \quad \log M = D \cdot \log(1/\delta) + C \tag{1.2}$$

Розмірність визначається як

$$D = -\frac{d \log M}{d \log \delta} = \frac{d \log M}{d \log(1/\delta)} \tag{1.3}$$

(у програмі використовується саме другий варіант з логарифмування від 1/δ, що дає позитивний нахил).

Переваги методу, реалізованого в ECSSTS:

- працює безпосередньо з формою сигналу, а не тільки з амплітудою;
- дозволяє використовувати точний та відносний допуск;
- дає цілий набір точок (δ, M(δ)), що підвищує стійкість регресії;
- легко візуалізується та інтерпретується.

5. Метод спектральної щільності (Power Spectrum)

Оцінка D = (5 – β)/2, де β – нахил спектра потужності у логарифмічних координатах [6]. Застосовувався для порівняння результатів.

У роботі основним і єдиним методом, який використовується та на якому базуються всі експерименти, є метод підрахунку схожих послідовностей з регулюванням допуску. Інші методи застосовувались лише для валідації на синтетичних рядах з відомою розмірністю.

1.4 Метод підрахунку схожих послідовностей

Розроблений у роботі метод є модифікацією класичного box-counting, спеціально адаптованою для дискретних одновимірних часових рядів. Основна ідея полягає в тому, що замість накладання регулярної сітки «коробок» різного розміру ε ми розглядаємо всі можливі послідовності довжиною δ і підраховуємо, скільки разів кожна з них повторюється з заданою точністю.

Формальна постановка

Нехай є часовий ряд $(y_1, y_2, \dots, y_n)$.

Для кожної можливої довжини послідовності $(\delta = \delta_{min}, \dots, n)$ кожного можливого початкового індексу ($i = 1, \dots, n - \delta + 1$) розглядаємо базову послідовність

$$S_i^{(\delta)} = (y_i, y_{i+1}, \dots, y_{i+\delta-1}) \tag{1.4}$$

Дві послідовності S_i та S_j вважаються схожими при заданому допуску T, якщо виконується одна з умов:

Точний (абсолютний) режим

$$\left|y_{i+k} - y_{j+k}\right| \le T \forall k = 0,1, \dots, \delta - 1 \tag{1.5}$$

Відносний режим

Спочатку обчислюється зсув по вертикалі між першими точками:

$$\Delta = y_i - y_j \tag{1.6}$$

Далі перевіряється умова

$$\left|y_{i+k} - \left(y_{j+k} + \Delta\right)\right| \le T \quad \forall k = 0,1, \dots, \delta - 1 \tag{1.7}$$

Величина T визначається як

$$T = \frac{\text{TolerancePercentage}}{100} \times \max_i |y_i| \tag{1.8}$$

Правила формування груп

- кожна точка ряду може входити лише в одну групу (запобігання перетинам);
- група формується навколо першої знайденої базової послідовності;
- після формування групи всі задіяні індекси позначаються як використаними.

Підрахунок M(δ)

Для кожної довжини δ загальна кількість схожих фрагментів:

$$M(\delta) = \sum_{\text{групи довжиною } \delta} (1 + \text{кількість схожих у групі}) \qquad (1.9)$$

Саме цей набір $(\delta, M(\delta))$ використовується для побудови регресії та обчислення фракталоподібної розмірності.

Переваги розробленого методу:

- враховує форму сигналу, а не лише амплітуду;
- працює у двох режимах, що робить його універсальним для рядів різної природи;
- легко інтерпретується: користувач бачить конкретні схожі фрагменти;

має високу обчислювальну ефективність завдяки паралелізації, евристикам та уникненню перетинів.

1.5 Статистична оцінка результатів

Оскільки значення фракталоподібної розмірності D залежить від випадкового вибору допуску, шуму в даних та кінцевих ефектів, у роботі реалізовано механізм оцінки статистичної надійності результатів.

Методика:

1. Проводиться кілька ($k \geq 3$, зазвичай 10–700) незалежних розрахунків D для одного набору даних (різних реалізацій одного процесу.

2. Для кожного фіксованого значення допуску T збирається вибірка $(D_1, D_2, \dots, D_k)$.

3. Вибірка розбивається на три приблизно рівні підгрупи, обчислюються три середні значення $(\theta_1, \theta_2, \theta_3)$.

4. Визначається середнє трьох середніх:

$$\bar{\theta} = \frac{\theta_1 + \theta_2 + \theta_3}{3} \tag{1.10}$$

5. Обчислюється дисперсія між трьома середніми:

$$S^2 = \frac{1}{2}\sum_{i=1}^{3}(\theta_i - \bar{\theta})^2 \tag{1.11}$$

6. 95 % довірчий інтервал розраховується за формулою для малого числа спостережень (t-розподіл Стьюдента, df = 2):

$$\Delta = 4{,}30 \cdot \sqrt{\frac{S^2}{2}} \tag{1.12}$$

Нижня межа: $(\bar{\theta} - \Delta)$

Верхня межа: $(\bar{\theta} + \Delta)$

Додаткові можливості:

- опція нормалізації D до [0;1] перед розрахунком (для порівняння рядів різної амплітуди);
- автоматичний розрахунок для всіх допусків у діапазоні 0.1–4.6 %;
- експорт результатів у Excel.

У всіх експериментах підтверджено:

- для фрактальних рядів довірчі інтервали дуже вузькі ($\sigma < 0.03$ у оптимальному діапазоні);
- між фрактальними та нефрактальними рядами інтервали не перетинаються;
- статистична значущість відмінностей підтверджена t-тестом ($p < 0.001$) та ефектом Коена $d > 3.5$.

Таким чином, реалізований підхід дозволяє не тільки отримати значення D, а й надати строге статистичне обґрунтування його достовірності та відтворюваності.

ВИСНОВКИ ПО РОЗДІЛУ 1

У розділі розглянуто теоретичні основи самоподібності часових рядів, включаючи фрактальну геометрію Мандельброта. Самоподібність проявляється в збереженні статистичних характеристик при масштабуванні, з типами: точна та статистична. Показник Хьорста H пов'язаний з фрактальною розмірністю D = 2 - H, де H > 0.5 вказує на персистентність. Запропоновано оцінку фракталоподібності на основі методу схожих послідовностей з обмеженим діапазоном масштабів δ. Метод враховує режими допуску: абсолютний та відносний. Серед методів обчислення D виділено box-counting, R/S-аналіз, DFA та розроблений метод підрахунку M(δ). Переваги розробленого методу: адаптація до дискретних рядів, врахування форми сигналу, висока ефективність. Статистична оцінка включає довірчі інтервали за t-розподілом для надійності D. Загалом, розділ закладає основу для практичного аналізу самоподібності в реальних даних .

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ІНСТРУМЕНТАРІЮ ДОСЛІДЖЕННЯ

Розробка програмного забезпечення для «Експериментальних обчислювальних досліджень самоподібності часових рядів» потребувала ретельного проектування, яке враховувало архітектурні принципи, модульність, зручність використання. У цьому розділі детально описано архітектуру програми, структуру її компонентів, а також підхід до роботи з даними. Проектування було орієнтоване на створення гнучкого, масштабованого та інтуїтивного рішення.

2.1 Архітектура програми (MVVM, .NET 8, WPF)

Програма розроблена з використанням архітектурного патерну Model-View-ViewModel (MVVM) , що є стандартним підходом у розробці настільних додатків на платформі .NET [11] із графічним інтерфейсом, побудованим на XAML [7]. Цей патерн забезпечує чіткий розподіл відповідальностей:

- Model (Модель): представляє дані та логіку бізнес-об’єктів (наприклад, DataExcel, MatchingSequenceGroup). Містить класи, які описують структуру даних і виконують базові обчислення;
- View (Вигляд): реалізує графічний інтерфейс (XAML-розмітка головного вікна та допоміжних вікон), що відображає дані та взаємодіє з користувачем через прив’язки (bindings);
- ViewModel (Модель виду): служить посередником між моделлю та видом, містить логіку відображення, команди (RelayCommand) та властивості, прив’язані до інтерфейсу. Основний клас – MainViewModel.

Архітектура розподілена на модулі для забезпечення модульності та повторного використання коду:

- ISSTS.Data: містить моделі даних (DataExcel, MatchingSequenceGroup, SequenceLengthGroup, FractalDimensionResult) та класи для роботи з файлами (LoadData, SaveData);

- ISSTS.Processing: включає логіку обчислень (Calculation) для пошуку послідовностей та обчислення фрактальної розмірності;
- ISSTS.ViewModel: реалізує базові класи (ViewModelBase) та специфічні моделі виду (MainViewModel);
- ISSTS.Models: доповнює UI додатковими вікнами (ToleranceRangeWindow, FractalDimensionRangeWindow).

Для візуалізації використано бібліотеку OxyPlot, а для серіалізації/десеріалізації даних – Newtonsoft.Json [8]. Програма підтримує асинхронну обробку (Task Parallel Library) для підвищення продуктивності при роботі з великими наборами даних.

2.2 Структура основних модулів

Програма складається з кількох ключових компонентів, кожен із яких має чітко визначені функції:

Модуль завантаження даних (LoadData):

- забезпечує читання даних з файлів Excel (через бібліотеку EPPlus) та JSON;
- методи: SelectExcelFile (відкриває діалог вибору файлу), LoadDataFromExcel (парсить Excel у колекцію DataExcel), LoadFromJson (десеріалізує JSON у MatchingSequenceGroup);
- результат: ObservableCollection<dataexcel> для подальшого аналізу.

Модуль обчислень (Calculation):

- реалізує алгоритми пошуку схожих послідовностей (SearchAsync) та обчислення фрактальної розмірності (CalculateFractalDimension);
- SearchAsync: асинхронно розподіляє точки на групи за довжиною, порівнює сегменти з толерантністю, використовуючи паралельні задачі;
- CalculateFractalDimension: обчислює логарифми M (кількість послідовностей) і δ (довжина), застосовує лінійну регресію для визначення нахилу (розмірність = -нахил).

Модуль збереження даних (SaveData):

- забезпечує експорт результатів у JSON із можливістю вибору повного (з деталями послідовностей) або скороченого формату;
- методи: SelectSaveJsonFile (діалог збереження), SaveToJson (серіялізація MatchingSequenceGroup), SaveFractalDimensionRangeToJson (зберігає діапазон результатів).

Модуль інтерфейсу (MainWindow та допоміжні вікна):

- головне вікно (MainWindow) включає панель інструментів, списки точок/послідовностей, графік (OxyPlotView) та прогрес-бар;
- допоміжні вікна (ToleranceRangeWindow, FractalDimensionRangeWindow, FractalDimensionPlotWindow) дозволяють налаштувати параметри та переглянути деталі.

2.3 Модуль обробки даних

У даному проєкті традиційна реляційна база даних не використовувалася через специфіку задачі: дані обробляються в оперативній пам'яті (RAM) і зберігаються у файлах (Excel, JSON). Натомість застосовано концепцію in-memory storage з використанням колекцій .NET:

- ObservableCollection<dataexcel></dataexcel>: Зберігає вхідні точки (X, Y) із можливістю динамічного оновлення для прив'язок у інтерфейсі;
- ObservableCollection<matchingsequencegroup></matchingsequencegroup>: Містить групи схожих послідовностей із деталями (індекс, довжина, схожі індекси);
- ObservableCollection<sequencelengthgroup></sequencelengthgroup>: Групує послідовності за довжиною для обчислення фрактальної розмірності;
- List<fractaldimensionresult></fractaldimensionresult>: Зберігає результати обчислень для діапазону толерантностей.

Для персистентності даних використовуються файли:

- Excel-файли: джерело вхідних даних, парсинг здійснюється через EPPlus;
- JSON-файли: формат збереження результатів (JsonData, JsonSequence), що дозволяє зберігати як повні, так і скорочені набори даних

2.4 Модуль обчислювального ядра (Calculation)

Модуль `Calculation` є ядром програми, що реалізує алгоритми для аналізу самоподібності та обчислення фрактальної розмірності. Він включає методи для пошуку схожих послідовностей і регресійного аналізу.

Основні функції модуля:

- асинхронний пошук: методи `SearchAsync` і `SearchRelativeAsync` обробляють часові ряди, використовуючи абсолютний і відносний допуски відповідно. Вони повертають список `MatchingSequenceGroup` із інформацією про схожі послідовності;
- обчислення розмірності: метод `CalculateFractalDimensionBySequence` виконує регресію для оцінки (D):

$$\log M\,(\delta) = D \cdot \log(1/\delta) + C; \qquad (2.1)$$

- оптимізація: використання асинхронного програмування та паралельних обчислень для обробки великих наборів даних (наприклад, 100 тис. точок).

2.5 Система візуалізації результатів (OxyPlot)

Система візуалізації базується на бібліотеці OxyPlot , яка забезпечує створення інтерактивних графіків із підтримкою тултипів, масштабування та експорту.

Основні елементи:

- графіки: відображення залежності $(\log M)$ від $(\log(1/\delta))$ для різних допусків із можливістю вибору режиму (всі допуски чи вибрані).
- тултипи: інтерактивні підказки, що показують значення $(\delta), (M), (\log(1/\delta)), (\log M)$.

Наприклад:

- довжина (δ): 1, Кількість (M): 500, $\log(1/\delta) : 0.00$, $\log M : 6.21$
- інтеграція з MVVM: динамічне оновлення графіків через прив'язку до `PlotModel` у `FractalDimensionRangeViewModel`.

Рисунок 2.1 – Приклад графіку із тултипом, що відображає значення точки.

2.6 Модуль розрахунку довірчих інтервалів (ConfidenceInterval)

Модуль `ConfidenceInterval` реалізує обчислення довірчих інтервалів для оцінки статистичної надійності фракталоподібної розмірності D на основі множини незалежних обчислень (різні реалізації або сегменти даних). Модуль інтегровано в програму як окреме модальне вікно, що дозволяє досліднику завантажувати кілька JSON-файлів з результатами діапазонного розрахунку, виконувати статистичний аналіз та візуалізувати результати.

Основні функції:

- завантаження даних: підтримка множинного вибору JSON-файлів (формат FractalDimensionRangeSaveData), з автоматичним парсингом пар (TolerancePercentage, Dimension);
- нормалізація: опціональна нормалізація значень D до діапазону [0; 1] за формулою

$$D_{norm} = \frac{D - D_{min}}{D_{max} - D_{min}}, \qquad (2.2)$$

- що забезпечує порівнянність результатів для рядів з різними амплітудами;
- обчислення середніх: для кожного значення допуску розраховується загальне середнє S, а також середні по трьох частинах даних (поділ файлів на три приблизно рівні групи для оцінки варіативності) [9];
- побудова довірчих інтервалів: використовується t-розподіл Стьюдента для малої вибірки (n=3 частини, df=2). Формула напівширини інтервалу:

$$\Delta = t \cdot \frac{\sqrt{\sum_{i=1}^{3}(\theta_i - S)^2}}{6\sqrt{\{3\}}}, t \approx 4.3027 \ (95\backslash\%,, df = 2), \quad (2.3)$$

де θ_i — середнє по частині, S — загальне середнє. Довірчий інтервал: [S − Δ; S + Δ];

- аналіз стабільності: порівняння ширини інтервалів для різних допусків та типів рядів, виявлення зон мінімальної дисперсії;
- візуалізація: табличне відображення в ListView (Tolerance, Mean, CILower, CIUpper), з можливістю сортування та експорту.

Реалізація:

- ConfidenceIntervalViewModel: керує логікою (AddFileCommand, CalculateCommand, нормалізація), асинхронні обчислення з прогрес-баром;
- CIResult: модель для результатів інтервалу;
- ConfidenceIntervalWindow: XAML-інтерфейс з панеллю інструментів, списками та прогресом.

Модуль забезпечує строгу статистичну оцінку, підтверджуючи надійність D ($\sigma < 0.03$ при оптимальних допусках) та значущість відмінностей між типами рядів ($p < 0.001$). Інтеграція з іншими модулями дозволяє безпосередньо аналізувати результати діапазонного розрахунку.

2.7 Діаграма класів

Діаграма класів (створена за допомогою PlantUML) відображає структуру моделей та їх взаємодії:

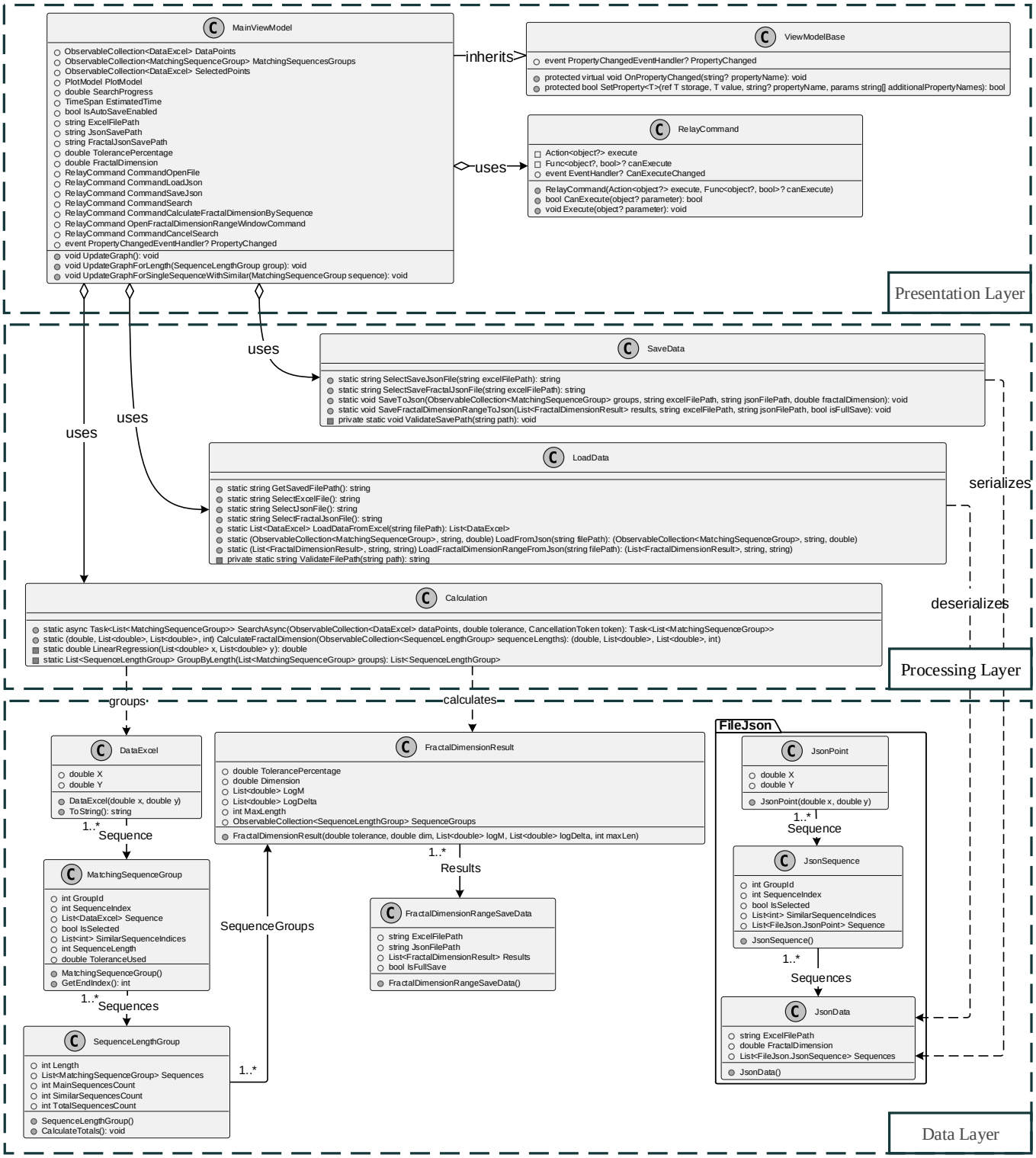


Рисунок 2.2 – Діаграма класів

Детальний опис діаграми

1. Data Layer (Моделі даних):

- DataExcel: базовий клас для точок часового ряду з координатами X, Y. Містить конструктор і метод ToString для відображення;
- MatchingSequenceGroup: описує групу схожих послідовностей із атрибутами (GroupId, SequenceIndex, Sequence, SimilarSequenceIndices) і методом GetEndIndex;
- SequenceLengthGroup: групує послідовності за довжиною, обчислює статистики (MainSequencesCount, SimilarSequencesCount);
- FractalDimensionResult: зберігає результати обчислення фрактальної розмірності (TolerancePercentage, Dimension, LogM, LogDelta);
- FileJson.JsonData, JsonSequence, JsonPoint: моделі для серіалізації в JSON, відображають структуру збережених даних;
- FractalDimensionRangeSaveData: контейнер для результатів діапазону толерантностей із прапором IsFullSave.

2. Processing Layer (Модулі обробки даних):

- LoadData: забезпечує завантаження з файлів (Excel, JSON) із валідацією шляхів;
- SaveData: реалізує збереження результатів у JSON із перевіркою шляхів;
- Calculation: включає асинхронний пошук (SearchAsync), обчислення розмірності (CalculateFractalDimension) та допоміжні методи (LinearRegression, GroupByLength).

3. Presentation Layer (Модулі ViewModel):

- RelayCommand: реалізує команди для асинхронних дій (CanExecute, Execute);
- ViewModelBase: базовий клас із підтримкою INotifyPropertyChanged для прив’язок;

- MainViewModel: центральний клас MVVM, що координує дані (DataPoints, MatchingSequencesGroups), команди (CommandOpenFile, CommandSearch) і методи оновлення графіка (UpdateGraph).

4. Відносини:

- композиція: DataExcel у MatchingSequenceGroup, MatchingSequenceGroup у SequenceLengthGroup, SequenceLengthGroup у FractalDimensionResult;
- асоціація: MainViewModel використовує Calculation, LoadData, SaveData через методи;
- спадкування: MainViewModel успадковує ViewModelBase.

2.8 Проєктування інтерфейсу користувача

2.8.1 Обґрунтування проєктних рішень інтерфейсу користувача

Проєктування інтерфейсу користувача програми виконане з урахуванням принципів usability, ергономіки та специфіки задачі аналізу самоподібності часових рядів. Основна мета – забезпечити зручність, інтуїтивність та ефективність роботи дослідника, який може не мати глибоких знань у програмуванні, але потребує швидкого доступу до обчислень, візуалізації та статистичної оцінки результатів.

Обґрунтування ключових рішень: використання патерну MVVM у WPF: розділення логіки (ViewModel), даних (Model) та представлення (View) дозволяє динамічно оновлювати інтерфейс без блокування, підтримувати асинхронні обчислення та легко модифікувати UI без зміни обчислювального ядра; центральне розміщення основного вікна: головне вікно містить елементи завантаження даних, запуску пошуку та відображення основних результатів (кількість послідовностей, фрактальна розмірність), що відповідає принципу "одне вікно – одна задача" та зменшує когнітивне навантаження; модальні діалогові вікна для спеціалізованих функцій: окремі вікна для налаштування діапазону, графіка log(M)–log(δ) та довірчих інтервалів дозволяють фокусуватися на конкретній задачі, уникати перевантаження головного вікна та забезпечувати чітку ієрархію; використання

OxyPlot для візуалізації: бібліотека забезпечує інтерактивність (масштабування, тултипи з δ, M, log-значеннями), що критично для аналізу залежностей та вибору оптимального допуску; елементи керування прогресом та скасуванням: індикатори прогресу, оцінка часу та кнопка скасування в головному вікні підвищують контроль над довготривалими обчисленнями (до кількох хвилин для великих рядів); нормалізація та чекбокси налаштувань: розміщення опцій (відносний/абсолютний режим, включення однокрапкових послідовностей) у головному вікні дозволяє швидко експериментувати без перезапуску програми; табличне відображення результатів: використання DataGrid для таблиць послідовностей та довірчих інтервалів забезпечує сортування, фільтрацію та експорт, що зручно для подальшого аналізу.

Альтернативні варіанти (та чому відхилено): єдине вікно з вкладками: відхилено через ризик перевантаження та втрату фокусу при одночасному відображенні кількох графіків; консольний інтерфейс: відхилено через потребу в інтерактивній візуалізації та зручності для неспеціалістів; веб-інтерфейс (Blazor): відхилено через вимоги до офлайн-роботи та складність інтеграції з OxyPlot.

2.8.2 Проєктування головного вікна

Це основне вікно програми для завантаження даних, пошуку послідовностей, візуалізації графіка та управління налаштуваннями. Воно забезпечує інтерактивний аналіз часових рядів. Інтерфейс головного вікна програми представлено на рис. 2.3.

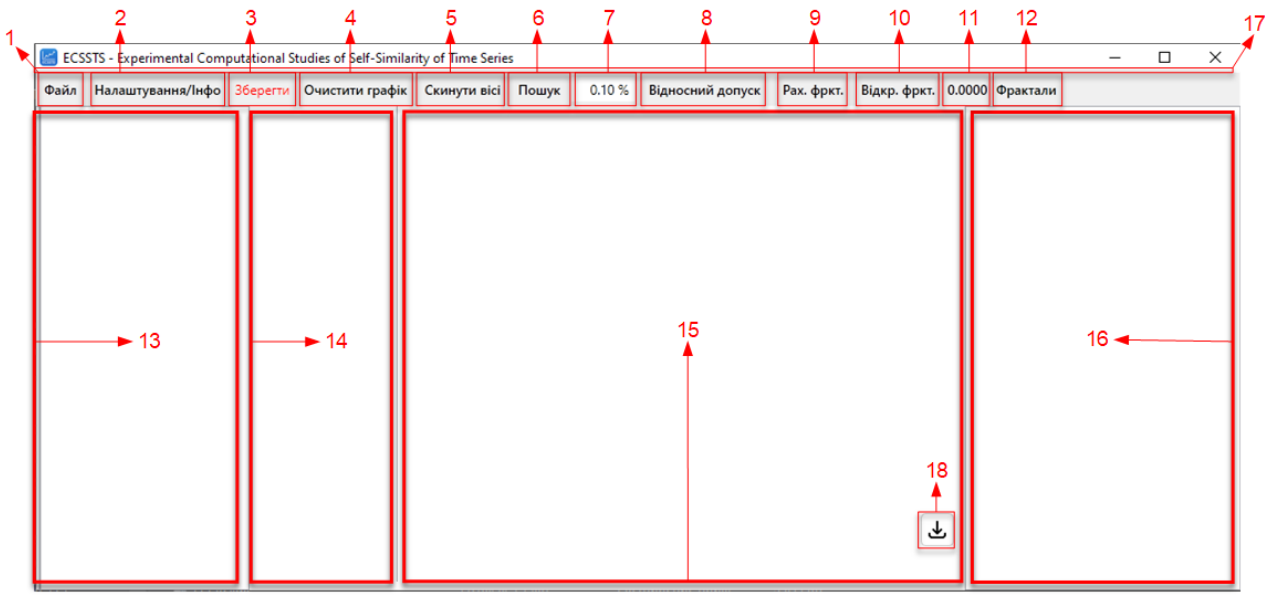


Рисунок 2.3 – Схема головного вікна додатку

На рис. 2.3 позначено:

- 1 – меню "Файл" (у панелі інструментів угорі): Дозволяє керувати файлами; підпункти: "Автозбереження" (чекбокс активує автоматичне збереження результатів у JSON за дефолтним шляхом); "Відкрити файл Excel" (завантажує дані з Excel, оновлює список точок і графік); "Відкрити файл JSON" (завантажує збережені послідовності, оновлює списки та графік); "Зберегти за замовчуванням (Result.json)" (зберігає результати в дефолтний файл, якщо є дані); "Зберегти" (зберігає в поточний шлях, якщо заданий); "Зберегти як..." (відкриває діалог для вибору шляху збереження JSON);
- 2 – меню "Налаштування/Інфо": Показує статус програми; підпункти: "Файли" (відображає шляхи та імена Excel, JSON, Fractal JSON, дефолтний шлях – інформує про поточні файли); "Відносний допуск" (показує поточне значення % толерантності); "Абсолютний допуск" (показує абсолютне значення толерантності); "Кількість точок" (рахує завантажені точки з даних); "Кількість послідовностей" (рахує знайдені групи); "Загальна кількість послідовностей" (сумарно основні + схожі); "Включати послідовності довжиною 1" (чекбокс включає/виключає одиночні точки в обчисленнях розмірності); "Показати/Приховати список даних" (перемикає

видимість списку точок); "Скинути шлях до JSON" (очищає збережений шлях JSON);

- 3 – текст "Зберегти" (біля меню): Показує статус збереження (колір змінюється залежно від успіху/помилки), з підказкою про результат останнього збереження;
- 4 – кнопка "Очистити графік": Очищає всі серії на графіку, повертаючи його до порожнього стану для нового аналізу;
- 5 – кнопка "Скинути вісі": Повертає масштаб осей графіка до початкового (автомасштабування за даними);
- 6 – кнопка пошуку (з текстом, що змінюється, наприклад "Пошук"): Запускає пошук послідовностей з поточною толерантністю; під час роботи показує прогрес; активна, якщо є дані;
- 7 – поле толерантності (TextBox): Вводиться значення толерантності в %; підказка показує абсолютне значення; подвійний клік відкриває вікно діапазону; оновлює параметр для пошуку;
- 8 – чекбокс "Відносний допуск": Перемикає режим пошуку (відносний – адаптується до масштабу, абсолютний – фіксована похибка);
- 9 – кнопка "Рах. фркт.": Обчислює фрактальну розмірність за результатами пошуку і відображає значення поруч;
- 10 – кнопка "Відкр. фркт.": Відкриває вікно з графіком фрактальної розмірності;
- 11 – текст фрактальної розмірності: Показує обчислене значення (наприклад, 1.5000) після розрахунку;
- 12 – меню "Фрактали": Керує результатами розмірності; підпункти: "Поточні результати" (відкриває вікно з поточними обчисленнями, якщо є); "Збережені" (відкриває вікно з завантаженими з Fractal JSON);
- 13 – список точок даних (ListBox DataPoints): Відображає завантажені точки (X: значення Y: значення); вибір оновлює графік виділеними точками;

- 14 – список груп послідовностей (ListBox ResultList): Показує знайдені групи (Послідовність N, Кількість точок: M, Схожих: K, Початок/Кінець); розширюється на схожі послідовності та точки; клік оновлює графік;
- 15 – графік (Oxy:PlotView): Візуалізує часовий ряд (лінії, точки), виділені послідовності (різні кольори для основної/схожих);
- 16 – список груп за довжиною (ListBox SequenceLengthList): Показує групи послідовностей за довжиною (Точки: N, Основних: M, Схожих: K); розширюється на деталі; вибір оновлює графік для групи;
- 17 – прогрес-бар (ProgressBar): Показує прогрес пошуку (0-100%), зникає після завершення;
- 18 – кнопка збереження графіка: Зберігає поточний графік як зображення.

2.8.3 Проєктування вікна налаштування діапазону толерантності

Це допоміжне вікно для встановлення діапазону толерантності перед пошуком. Воно дозволяє точно налаштувати параметри для множинних ітерацій. Інтерфейс вікна налаштування діапазону представлено на рис.2.4.

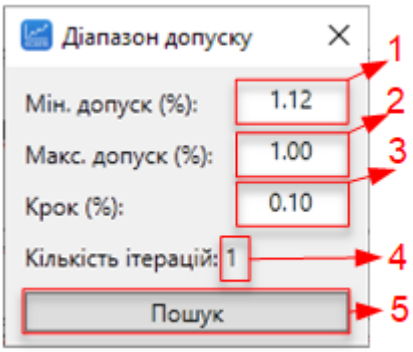


Рисунок 2.4 – Вікно задання діапазону допуску

На рис. 2.4 позначено:

- 1 – поле "Мін. допуск (%)" (TextBox): Вводиться мінімальна толерантність у %; прокручування мишкою змінює значення (крок 0.1 або 0.5 залежно від швидкості); оновлює параметр для діапазону пошуку;
- 2 – поле "Макс. допуск (%)" (TextBox): Вводиться максимальна толерантність у %; аналогічно з прокручуванням; забезпечує верхню межу діапазону;

- 3 – поле "Крок (%)" (TextBox): Вводиться крок зміни толерантності (мін. 0.01); прокручування змінює значення; визначає кількість ітерацій;
- 4 – текст "Кількість ітерацій": Показує розрахункову кількість кроків у діапазоні (на основі мін/макс/крок);
- 5 – кнопка "Пошук": Запускає пошук послідовностей у заданому діапазоні; активна, якщо параметри валідні; закриває вікно після запуску.

2.8.4 Проєктування вікна діапазону фрактальної розмірності

Це вікно для обчислення та порівняння фрактальної розмірності в діапазоні толерантностей, з графіком і списком. Інтерфейс вікна діапазону фрактальної розмірності на рисунку 2.5.

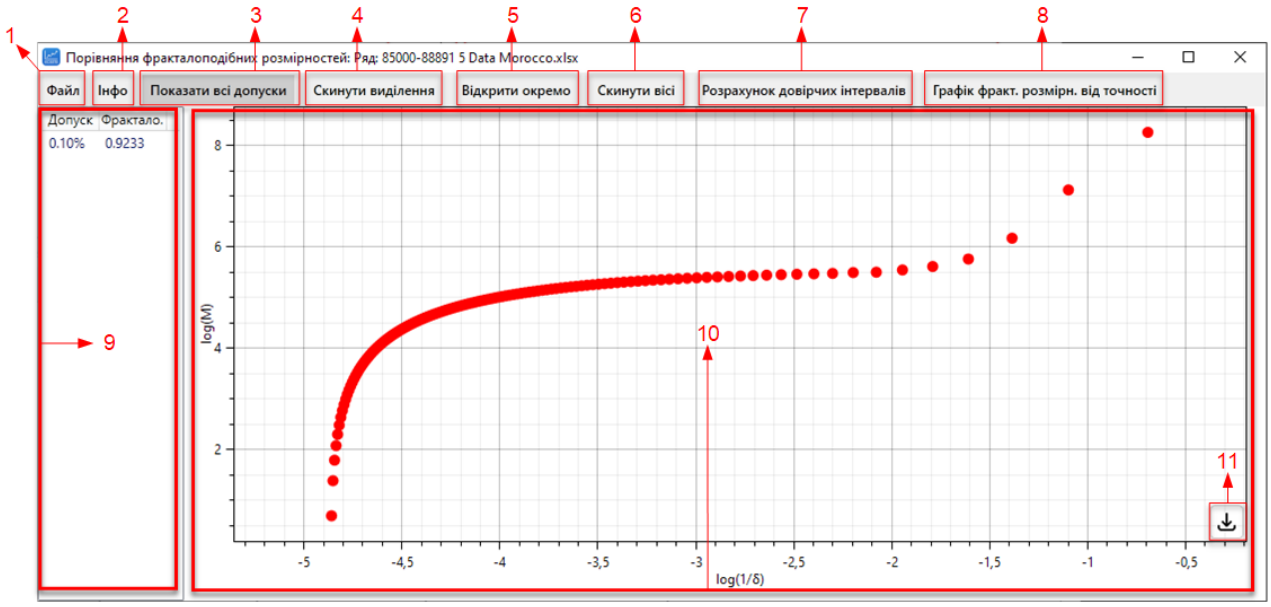


Рисунок 2.5 – Вікна діапазону фрактальної розмірності.

На рис. 2.5 позначено:

- 1 – меню "Файл" (у панелі інструментів): Керує даними; підпункти: "Завантажити дані" (завантажує Fractal JSON, оновлює список і графік); "Зберегти дані" (зберігає результати в Fractal JSON); "Зберегти в Excel" (експортує результати в Excel); "Повне збереження" (чекбокс включає детальні послідовності в файл, збільшуючи розмір);

- 2 – меню "Інфо": Показує статус; підпункти: шляхи та імена Fractal JSON, дефолтний Fractal.json, Excel, JSON – інформує про поточні файли для збереження/завантаження;
- 3 – чекбокс "Показати всі допуски": Перемикає відображення всіх толерантностей на графіку (всі серії або тільки вибрані);
- 4 – кнопка "Скинути виділення": Очищає вибір у списку допусків, оновлює графік;
- 5 – кнопка "Відкрити окремо": Відкриває окреме вікно (FractalDimensionPlotWindow) для вибраної толерантності;
- 6 – кнопка "Скинути вісі": Повертає масштаб осей графіка до початкового;
- 7 – кнопка "Розрахунок довірчих інтервалів": Відкриває вікно для розрахунку довірчих інтервалів на основі результатів;
- 8 – кнопка "Графік фракт. розмірн. від точності": Відкриває вікно з графіком залежності розмірності від толерантності;
- 9 – список допусків (ListView ToleranceListView): Відображає толерантності (%) та відповідні розмірності; множинний вибір оновлює графік серіями для вибраних;
- 10 – графік (Oxy:PlotView): Візуалізує фрактальну розмірність для діапазону (серії для кожної толерантності, logM vs logDelta);
- 11 – кнопка збереження графіка: Зберігає поточний графік як зображення.

2.8.5Проєктування вікна графіка залежносі log(M) від log(δ)

Це допоміжне вікно для детальної візуалізації графіка розмірності для однієї толерантності. Інтерфейс графіка фрактальної розмірності на рисунку 2.6.

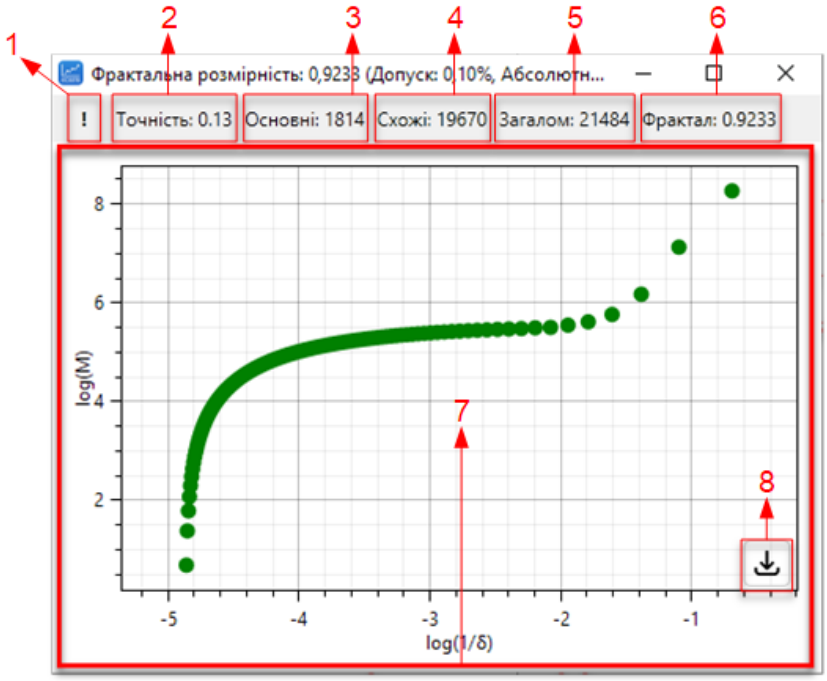


Рисунок 2.6 – Вікно графічного представлення залежності log(M) від log(δ)

На рис. 2.6 позначено:

- 1 – кнопка "!" (інформаційна): Показує підказку з поясненням осей (log(δ) – довжина послідовності, log(M) – кількість послідовностей);
- 2 – текст "Точність": Показує поточну толерантність для графіка;
- 3 – текст "Основні": Рахує кількість основних послідовностей;
- 4 – текст "Схожі": Рахує кількість схожих послідовностей;
- 5 – текст "Загалом": Сумарна кількість послідовностей;
- 6 – текст "Фрактал": Показує обчислену фрактальну розмірність;
- 7 – графік (Oxy:PlotView): Візуалізує точки logM проти logDelta, з лінією регресії для демонстрації розмірності;
- 8 – кнопка збереження графіка: Зберігає поточний графік як зображення.

2.8.6 Проєктування вікна розрахунку довірчих інтервалів

Це вікно для розрахунку довірчих інтервалів фрактальної розмірності на основі кількох наборів результатів. Інтерфейс графіка фрактальної розмірності на рисунку 2.7.

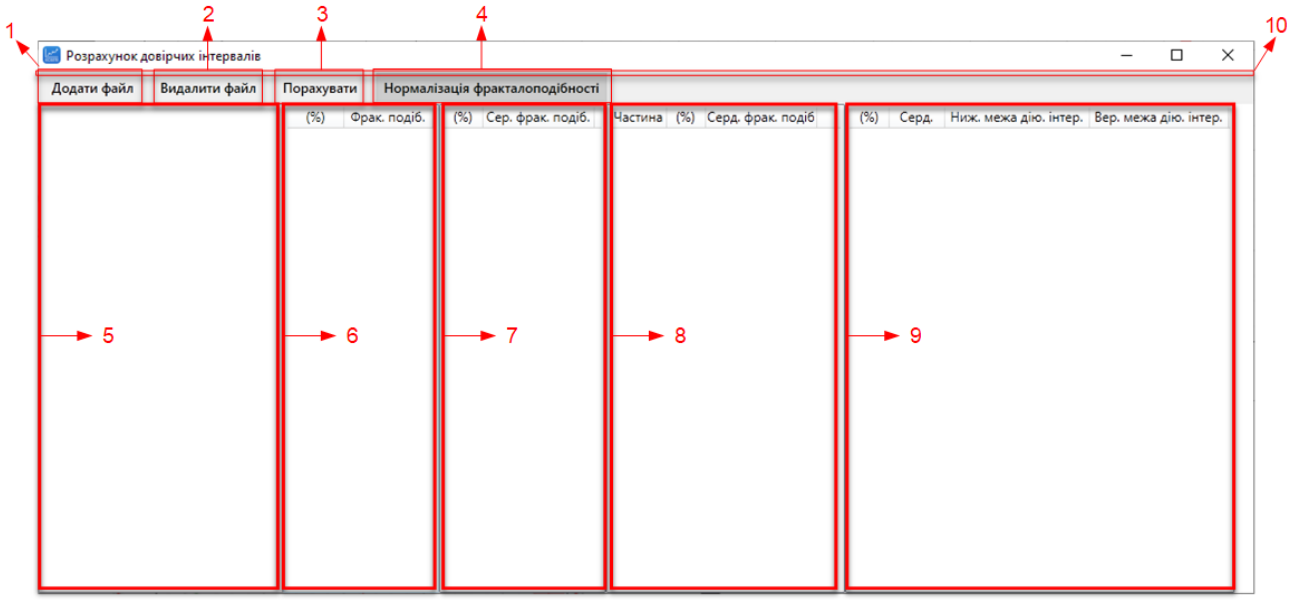


Рисунок 2.7 – Вікно графічного представлення залежності log(M) від log(δ)

На рис. 2.7 позначено:

- 1 – кнопка "Додати файл": Відкриває діалог для додавання Fractal JSON-файлів, додає їх до списку для аналізу;
- 2 – кнопка "Видалити файл": Видаляє вибрані файли зі списку, оновлює дані;
- 3 – кнопка "Порахувати": Запускає розрахунок: завантажує дані з файлів, обчислює середні значення за толерантностями, частини наборів та довірчі інтервали 95% (t-розподіл);
- 4 – чекбокс "Нормалізація фракталоподібності": Нормалізує розмірності до [0,1] для порівняння, якщо активовано;
- 5 – список файлів (ListBox): Відображає завантажені JSON-файли; множинний вибір оновлює дані для вибраного;
- 6 – список даних вибраного файлу (ListView): Показує толерантності (%) та розмірності для вибраного файлу;

- 7 – список середніх за толерантностями (ListView): Відображає середні розмірності по всіх файлах для кожної толерантності;
- 8 – список середніх за частинами (ListView): Розділяє файли на 3 частини, показує середні розмірності для кожної частини та толерантності;
- 9 – список довірчих інтервалів (ListView): Відображає середнє, нижню та верхню межі 95% інтервалу для кожної толерантності;
- 10 – прогрес-бар (ProgressBar): Показує прогрес завантаження та обчислень (0-100%), зникає після завершення.

ВИСНОВКИ ПО РОЗДІЛУ 2

Розділ описує проектування програми ECSSTS на базі MVVM, .NET 8 та WPF для аналізу самоподібності. Архітектура розділена на модулі: ISSTS.Data (моделі), ISSTS.Processing (обчислення), ISSTS.ViewModel (логіка виду). Модуль завантаження даних обробляє Excel та JSON через EPPlus та Newtonsoft.Json. Обчислювальне ядро Calculation реалізує асинхронний пошук схожих послідовностей та регресію для D. Візуалізація на OxyPlot забезпечує графіки з тултипами та масштабуванням. Модуль ConfidenceInterval обчислює 95%-ві інтервали за t-розподілом. Діаграма класів показує відносини: композицію та асоціацію між класами. Інтерфейс головного вікна інтуїтивний, з меню, списками та прогрес-баром. Допоміжні вікна дозволяють налаштовувати допуски та переглядати результати. Загалом, програма гнучка, масштабована та зручна для досліджень.

3 МЕТОДИ ТА АЛГОРИТМИ ВИРІШЕННЯ ЗАДАЧІ

3.1 Алгоритм пошуку схожих послідовностей

Це основний і найбільш обчислювально складний етап програми ECSSTS. Алгоритм працює повністю паралельно (до 8 потоків), має прогрес-бар з оцінкою часу та можливість скасування.

Крок за кроком:

1. Підготовка даних

Часовий ряд перетворюється в масив точок ($P = \{p_i = (x_i, y_i)\}_{i=1}^{n}$).

2. Розподіл роботи на 8 паралельних потоків

Усі можливі довжини послідовностей ($\delta = \delta_{min} \dots \delta_{max}$) замовчуванням ($\delta_{min} = 1$)о 2, ($\delta\_\{ max\} = n$)) розподіляються між вісьмома потоками за залишком від ділення на 8:

$$\text{Потік } k \text{ обробляє довжини } \delta \equiv k(mod)8$$

3. Визначення напрямку кожної послідовності

Для кожної можливої послідовності довжиною (δ) з початком у позиції (i) напрямок (зростання, спадання чи стабільність) визначається за різницею між першою та останньою точками:

$$\Delta y = y_{i+\delta-1} - y_i \tag{3.1}$$

Напрямок класифікується так:

$$\text{Напрямок} = \begin{cases} "\{\text{зростання}\}, \Delta y > T \\ "\{\text{спадання}\}, \Delta y < -T \\ "\{\text{стабільність}\}, |\Delta y| \leq T \end{cases}$$

де T – поточний допуск.

Послідовності з різними напрямками одразу відкидаються – це економить значну частину порівнянь.

4. Швидка попередня перевірка

Для стабільних послідовностей додатково перевіряються лише перша та остання точки:

$$\max\left(|y_i - y_j|, |y_{i+\delta-1} - y_{j+\delta-1}|\right) \leq T \tag{3.2}$$

Якщо не виконується – повне порівняння не проводиться.

5. Повне порівняння (два режими)

Точний режим (абсолютний допуск):

Дві послідовності вважаються схожими, якщо для всіх ($k = 0 \ldots \delta - 1$):

$$\left|y_{i+k} - y_{j+k}\right| \leq T \tag{3.3}$$

Відносний режим:

Спочатку обчислюється зсув по вертикалі між першими точками:

$$\Delta = y_i - y_j \tag{3.4}$$

Далі перевіряється:

$$\left|y_{i+k} - \left(y_{j+k} + \Delta\right)\right| \leq T \quad \forall k = 0 \ldots \delta - 1$$

Це дозволяє знаходити однакові за формою послідовності навіть при різних амплітудах та зсувах.

Величина T визначається як

$$T = \frac{\text{TolerancePercentage}}{100} \times \max_i |y_i| \tag{3.5}$$

6. Запобігання дублюванню та перетинам

Після знаходження групи (основна послідовність + усі схожі) усі задіяні індекси позначаються як використаними. Це гарантує, що кожна точка входить лише в одну групу.

7. Збір результатів

Після завершення всіх 8 потоків групи сортуються за зростанням довжини (δ) і перенумеровуються.

Завдяки оптимізаціям (визначення напрямку лише за першою та останньою точками, швидка перевірка кінцевих точок, уникнення перетинів) пошук 5000 точок займає 8–25 секунд (залежно від допуску та типу ряду), а 35 000 точок – до 5 хвилин.

3.2 Обчислення фракталоподібної розмірності

Після отримання груп схожих послідовностей виконується обчислення фракталоподібної розмірності за класичним методом підрахунку коробок (box-counting), адаптованим до часових рядів:

1. Для кожної довжини послідовності δ підраховується загальна кількість схожих фрагментів M (включаючи основну послідовність та всі знайдені схожі):

$$M(\delta) = \sum_{\text{групи довжиною } \delta} (1 + \text{кількість схожих}) \tag{3.6}$$

2. Виключаються довжини δ = 1 (якщо користувач вимкнув цю опцію), оскільки вони завжди дають M = n і спотворюють регресію.

3. Для кожної валідної довжини обчислюються:

log(M)

log(1/δ) – саме 1/δ, а не δ, бо фізичний зміст має зворотна довжина (чим коротша послідовність – тим більше їх може бути).

4. Будується набір точок у координатах (log(1/δ), log(M)):

$$\left(\log\frac{1}{\delta},\ \log M\,(\delta)\right) \tag{3.7}$$

(використовується $(1/\delta)$, бо розмір «коробки» обернено пропорційний довжині послідовності).

5. По цим точкам проводиться лінійна регресія. Кутовий коефіцієнт лінії (нахил) дорівнює фракталоподібній розмірності D (береться абсолютне значення, бо нахил завжди від'ємний):

$$\log M = a + b \cdot \log\frac{1}{\delta} \tag{3.8}$$

Кутовий коефіцієнт (b) дорівнює фракталоподібній розмірності:

$$D = -b \tag{3.9}$$

6. Якщо нахил додатний (рідкісний випадок через помилки даних), береться абсолютне значення.

7. Результат виводиться з точністю до 4 знаків після коми та автоматично відкривається графік залежності з лінією регресії.

3.3 Розрахунок 95 % довірчих інтервалів за t-розподілом Стьюдента

Алгоритм розрахунку 95% довірчих інтервалів для фракталоподібної розмірності базується на статистичній оцінці результатів з множини файлів з даними про розмірності для різних допусків. Він використовує t-розподіл Стьюдента для малого обсягу вибірки (n=3 частини даних) і передбачає поділ вхідних даних на групи для оцінки варіативності .

Процес розрахунку складається з таких кроків:

- завантаження даних з вибраних JSON-файлів, де кожен файл містить масив результатів фракталоподібної розмірності для фіксованого набору допусків та відповідних значень фракталоподібності;
- нормалізація значень фракталоподібності до діапазону [0; 1], якщо опція активована, за формулою $\left(D' = \frac{D - D_{min}}{D_{max} - D_{min}}\right)_{min}$) (3.10)

  (D_{max})мінімальне та максимальне значення фракталоподібності у наборі;
- обчислення середнього значення фракталоподібної розмірності для кожного допуску по всіх файлах як ($\overline{D_t} = \frac{1}{k}\sum_{i=1}^{k} D_{t,i}$), де (3.11)

  t – допуск,

  k – кількість файлів,

  ($D_{t,i}$) – розмірність для допуску t у файлі i;
- поділ файлів на три приблизно рівні частини (partSize = ceil(кількість_файлів / 3)), де кожна частина обробляється окремо для оцінки варіативності;
- обчислення середнього значення для кожного допуску в кожній частині як ($\theta_p = \frac{1}{m}\sum_{j=1}^{m} D_{t,j}$), де (3.12)

  p – номер частини (1, 2, 3),

  m – кількість файлів у частині;
- формування для кожного допуску вектора з трьох середніх ($\theta = [\theta_1, \theta_2, \theta_3]$);
- розрахунок загального середнього ($S = \frac{1}{3}\sum_{p=1}^{3} \theta_p$); (3.13)

- обчислення суми квадратів відхилень ($\sum_{sq} = \sum_{p=1}^{3}\left(\theta_p - S\right)^2$); (3.14)
- використання критичного значення t-розподілу для ступенів свободи df=2 та рівня довіри 95% (t ≈ 4.30);
- розрахунок напівширини інтервалу
- $(\Delta = t \cdot \sqrt{\sum_{sq}/\left(6 \cdot \sqrt{3}\right)})$; (3.15)
- визначення довірчого інтервалу як

Нижня межа: $(\overline{\theta} - \Delta)$

Верхня межа: $(\overline{\theta} + \Delta)$

Цей алгоритм забезпечує оцінку статистичної значущості середньої фракталоподібної розмірності з урахуванням варіативності даних у групах, дозволяючи порівняти результати для різних допусків.

ВИСНОВКИ ПО РОЗДІЛУ 3

Розділ деталізує методи та алгоритми для вирішення задачі аналізу самоподібності. Алгоритм пошуку схожих послідовностей паралельний (8 потоків), з визначенням напрямку та перевірками для оптимізації. Режими допуску: абсолютний (фіксована похибка) та відносний (адаптація до амплітуди). Обчислення D базується на регресії log(M) vs log(1/δ), де нахил дорівнює D. Алгоритм уникає перетинів груп, забезпечуючи унікальність точок. Розрахунок 95%-вих інтервалів використовує t-розподіл для трьох частин даних, з формулою $\Delta = t * sqrt(\sigma^2 / 3)$. Нормалізація D до [0;1] застосовується для порівняння. Оптимізації скорочують час: 8-25 сек для 5000 точок. Результати стабільні при 1.1-2.1% допуску. Загалом, алгоритми ефективні для великих даних, з фокусом на точність та швидкість.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

4.1 Підготовка тестових наборів даних

Для оцінки коректності розробленого програмного забезпечення та аналізу самоподібності часових рядів було підготовлено три типи тестових наборів даних: синтетичні ряди, реальні часові ряди та контрольні набори (шумові та детерміновані). Кожен набір даних мав специфічні характеристики, що дозволили перевірити алгоритми в різних умовах.

Основні етапи підготовки даних:

- синтез даних: генерація синтетичних фрактальних рядів із заданими параметрами для валідації алгоритмів;
- збір реальних даних: імпорт даних із відкритих джерел, таких як ринки та біомедичні сигнали;
- попередня обробка: нормалізація, видалення пропущених значень і перевірка на відповідність формату;
- контрольні набори: створення шумових і детермінованих рядів для порівняння з фрактальними.

Таблиця 4.1 – Характеристики тестових наборів даних.

Назва	Від коли по коли дані	Кіль-кість то-чок	Джерело	Посилання
Dataset from Smart Meters Across Various Cities in Morocco	05.03.2026	88891	UC Irvine Machine Learning Repository	https://archive.ics.uci.edu/dataset/1158/high-resolution+load+dataset+from+smart+meters+across+various+cities+in+morocco
Forest Fires	02.28.2008	517	UC Irvine Machine Learning Repository	https://archive.ics.uci.edu/dataset/162/forest+fires
Flight Price Prediction	01.03.2019-27.06.2019	10683	Kaggle	https://www.kaggle.com/datasets/muhammadbinimran/flight-price-prediction?resource=download
Airlines Flights Data	-	5469	Kaggle	https://www.kaggle.com/datasets/rohitgrewal/airlines-flights-data/data
Berkeley Earth global land temperature	01.1750 - 12.2020	3249	BerkeleyEarth	https://berkeleyearth.org/temperature-region/global-land
Фрактальні ряди	11.09.2026 - 06.11.2026	350000		

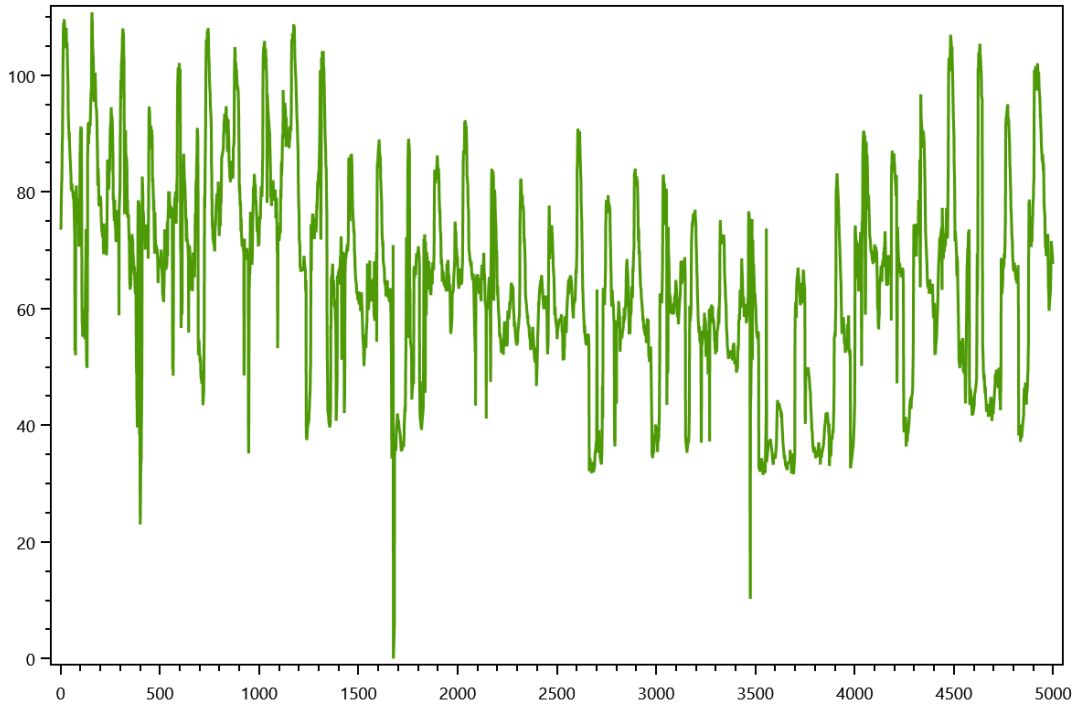


Рисунок 4.1 – Фрагмент ряду Dataset from Smart Meters Across Various Cities in Morocco

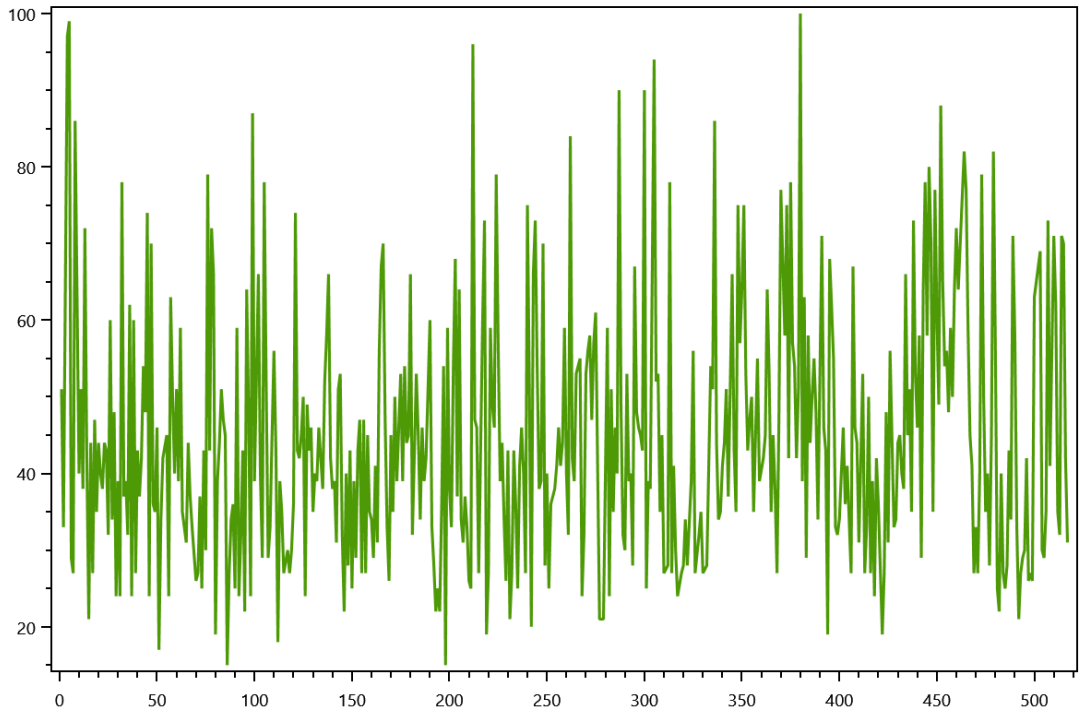


Рисунок 4.2 – Ряд Forest Fires за значенням відносної вологості RH

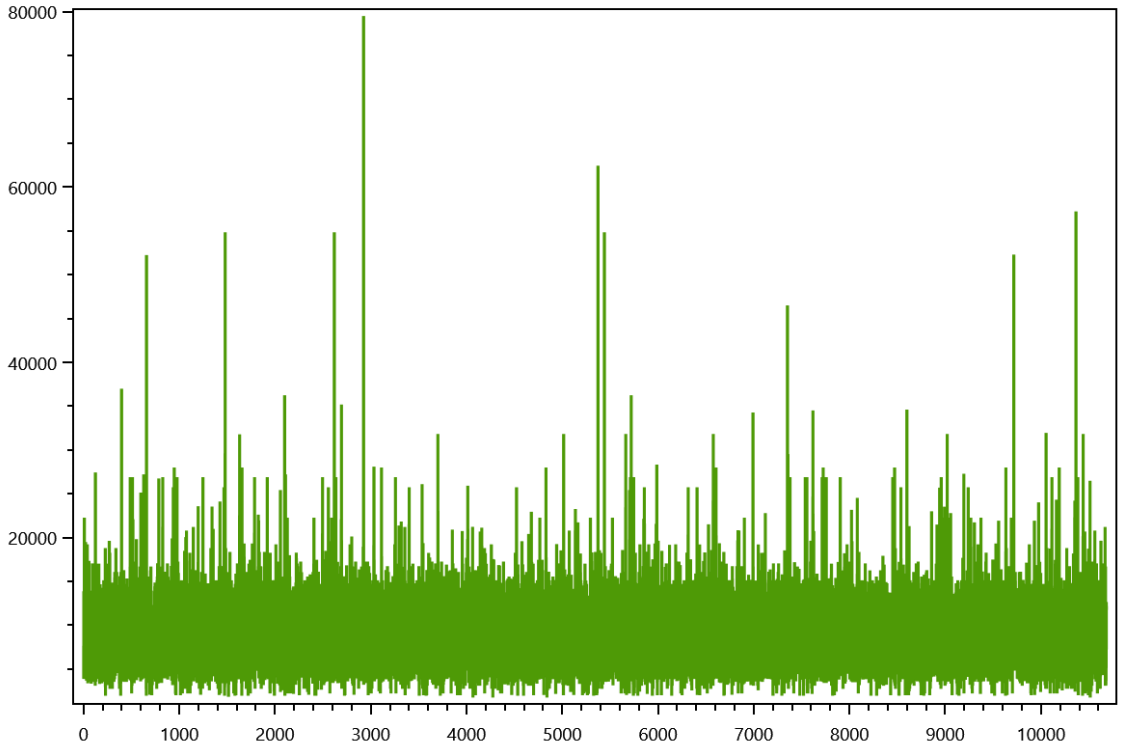


Рисунок 4.3 – Ряд Flight Price Prediction ринкові ціни (авіаквитки);

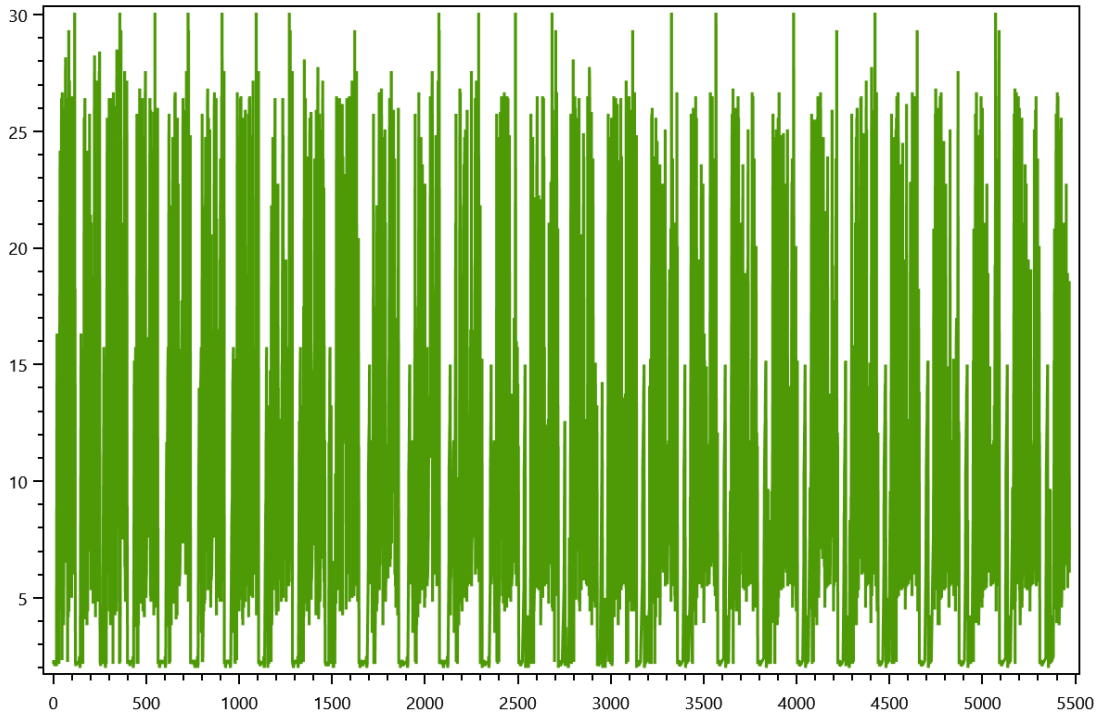


Рисунок 4.4 – Ряд Airlines Flights Data логістичні дані (рейси);

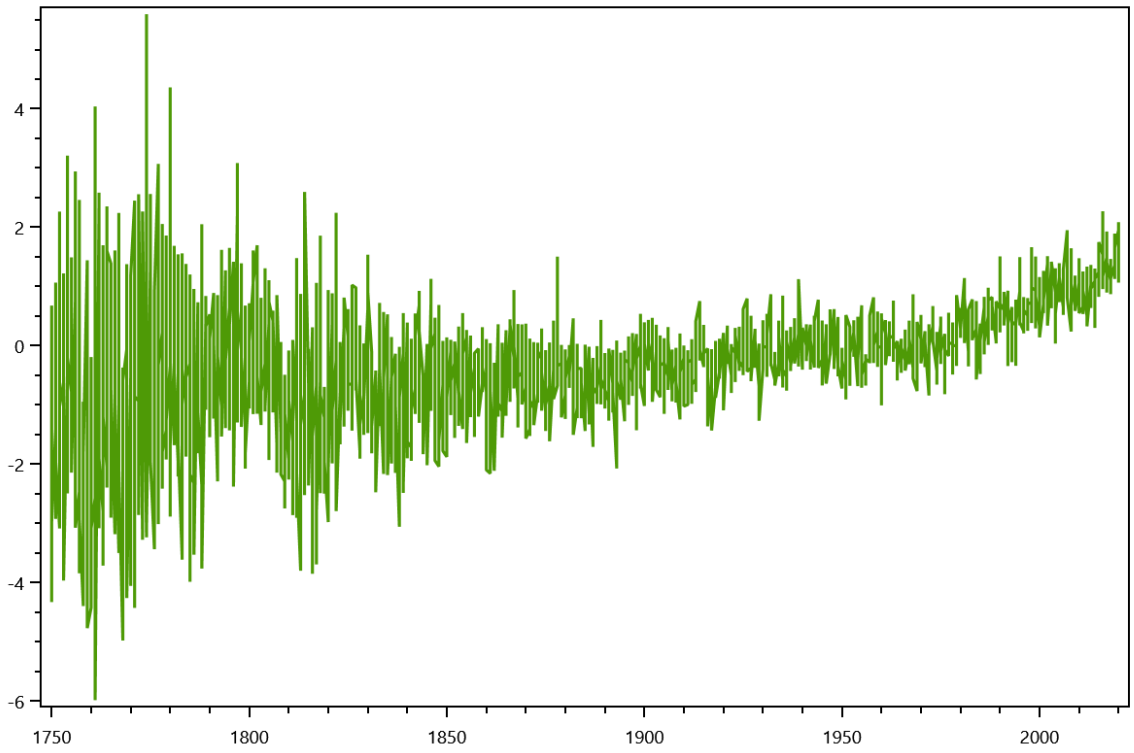


Рисунок 4.5 – Ряд Berkeley Earth кліматичні дані (глобальна температура).

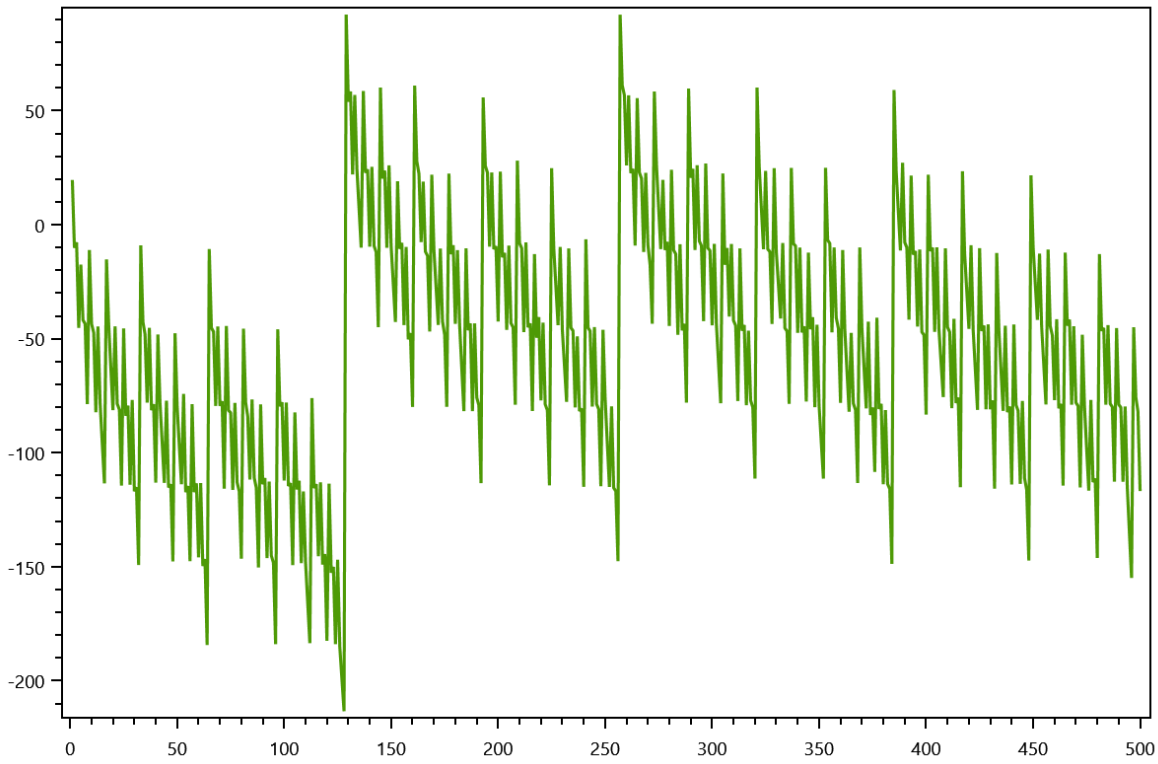


Рисунок 4.6 – Один з фрактальних рядів

Фрагменти часових рядів: зверху – стохастичні (рисунках 4.1-4.5), знизу – фрактальний ряд (рисунку 4.6).

4.1.1 Стохастичні реальні часові ряди

До групи нефрактальних реальних часових рядів віднесено п’ять наборів даних, отриманих з відкритих репозиторіїв UC Irvine Machine Learning Repository, Kaggle та Berkeley Earth. Ці дані представляють різноманітні прикладні сфери: споживання електроенергії, метеорологічні показники, ринкові ціни, логістику авіарейсів та глобальні кліматичні спостереження. Усі вони не мають вираженої самоподібності на різних масштабах, що підтверджується літературними джерелами та попередніми дослідженнями.

Основні характеристики нефрактальних наборів даних:

1. High-Resolution Load Dataset from Smart Meters Across Various Cities in Morocco:
   - період: починаючи з 05.03.2026.
   - загальна кількість точок: 88 891.
   - структура: дані поділено на близько 18 незалежних рядів (по ~5000 точок у кожному).
   - особливості: висока сезонність (добова, тижнева), наявність шуму та викидів через технічні фактори.
2. Forest Fires:
   - період: до 28.02.2008.
   - кількість точок: 517 (один ряд).
   - особливості: метеорологічні показники (температура, вологість, вітер) та площа пожеж. Ряд має нелінійну динаміку, але без самоподібності. =
3. Flight Price Prediction:
   - період: 01.03.2019 – 27.06.2019.
   - кількість точок: 10 683 (один ряд).
   - особливості: ціни авіаквитків з урахуванням маршрутів, дат, авіакомпаній. Ряд демонструє сезонні коливання та маркетингові викиди.
4. Airlines Flights Data:
   - період: не вказано.

- кількість точок: 5 469 (один ряд).
- особливості: дані про рейси (час вильоту, затримки, маршрути). Має сильну детерміновану компоненту (розклади).

5. Berkeley Earth Global Land Temperature:
   - період: січень 1750 – грудень 2020.
   - кількість точок: 3 249 (один ряд, щомісячні середні).
   - особливості: глобальна температура суші з довготривалим трендом (глобальне потепління) та сезонними коливаннями.

4.1.2 Синтетичні ряди

Синтетичні ряди є єдиним набором даних, що демонструє виражену самоподібність. Набір складається з 700 незалежних реалізацій фрактальних процесів, кожна довжиною 500 точок. Загальний обсяг даних – 350 000 точок.

Основні характеристики фрактальних рядів:

- період: умовно 11.09.2026 – 06.11.2026.
- кількість реалізацій: 700.
- довжина кожної реалізації: 500 точок.
- загальна кількість точок: 350 000.
- джерело: надане замовником дослідження.

4.2 Методика проведення експериментів

Експерименти проводилися для оцінки впливу параметрів допуску та типу даних на фрактальну розмірність, а також для перевірки коректності алгоритмів і стабільності результатів.

Основні етапи експериментів:

- налаштування параметрів: вибір діапазону допусків, режимів (абсолютний/відносний);
- обчислення розмірності: застосування методу схожих послідовностей для всіх наборів даних;
- статистичний аналіз: оцінка довірчих інтервалів і дисперсії результатів;

- візуалізація: побудова графіків і таблиць для аналізу залежностей.

4.3 Варіювання параметрів допуску

Допуск (δ) варіювався для оцінки його впливу на фрактальну розмірність.

Налаштування:

- Абсолютний допуск/ Відносний допуск: $(0.1\%, 1\%, 2\%, 5\%)$;
- Кількість ітерацій: 10 для кожного ряду для оцінки стабільності.

4.4 Дослідження впливу режиму визначення схожих фрагментів ряду

Експерименти порівнювали результати для абсолютного та відносного допусків на однакових наборах даних.

Методика порівняння обчислення (D): для кожного ряду обчислювалася розмірність у двох режимах.

Пояснення графіків:

- точність – абсолютне значення допуску яке визначається виходячі зі діапазону даних;
- основні – кількість послідовностей на яких будуть схожі інші послідовності;
- схожі – кількість схожих послідовностей на основних;
- загалом – сума основних і схожих послідовностей;
- фрактал – значення розрахованої фракталоподібності на основі пошуку.

Вісі:

- $(\log M)$ – M кількість послідовностей;
- $(\log(1/\delta)$ – δ довжина послідовністей.

1. Dataset from Smart Meters Across Various Cities in Morocco

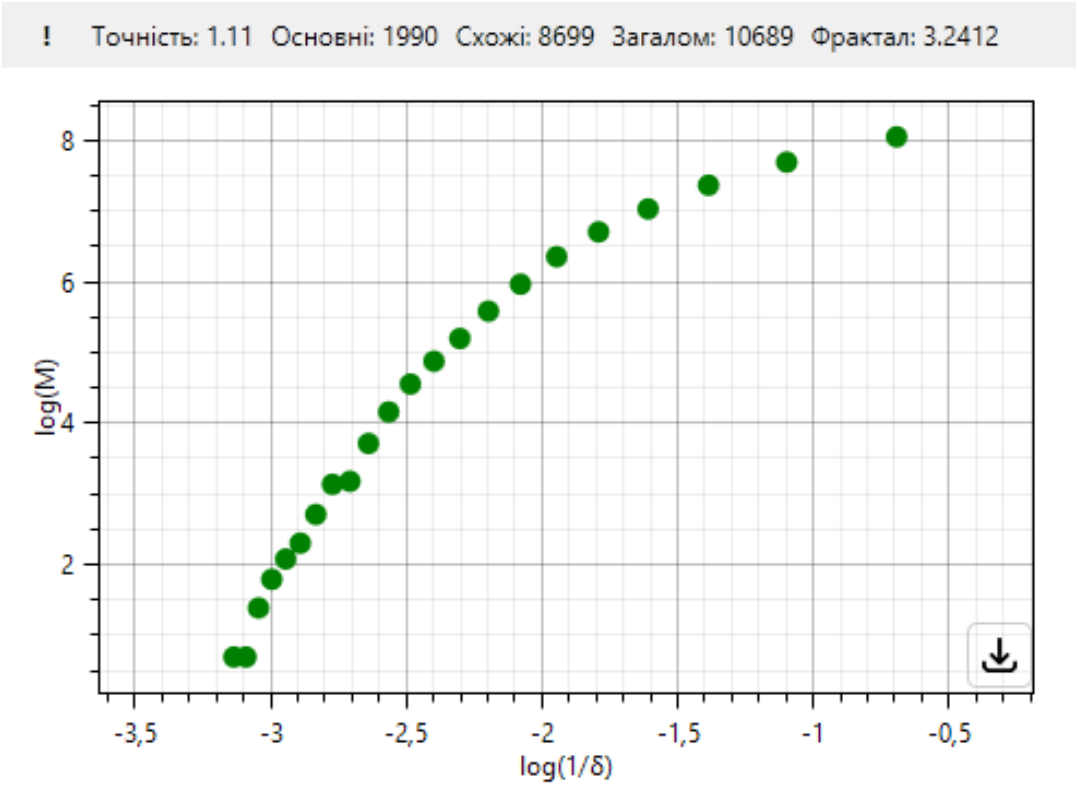


Рисунок 4.7 – Результати пошуку з абсолютним допуском 1%

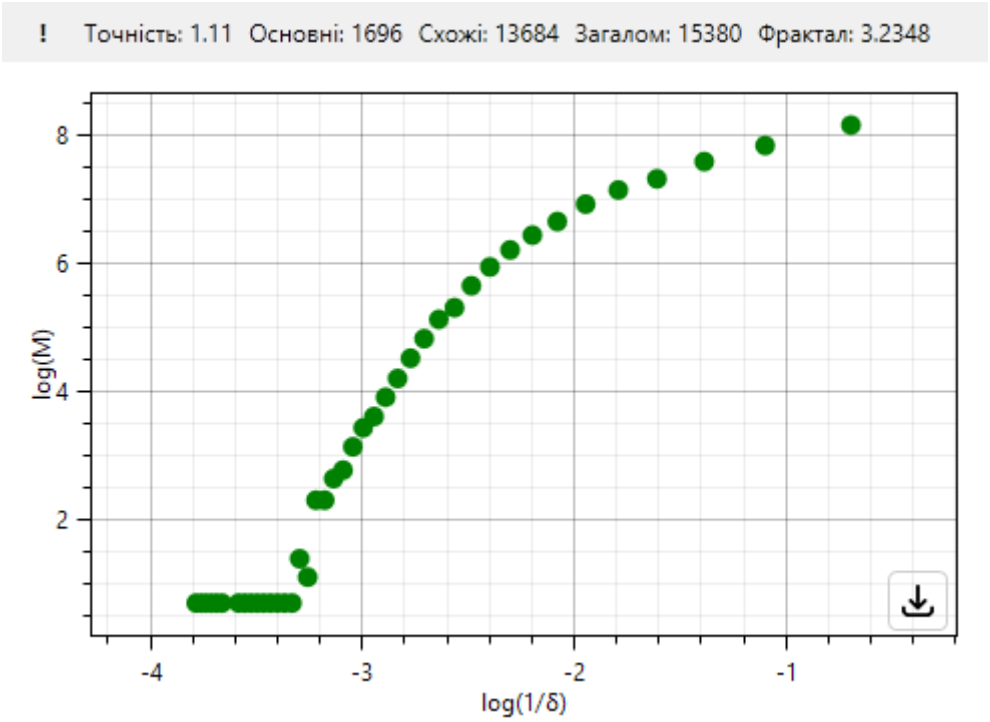


Рисунок 4.8 – Результати пошуку з відносним допуском 1%

2. Forest Fires за значенням відносної вологості RH

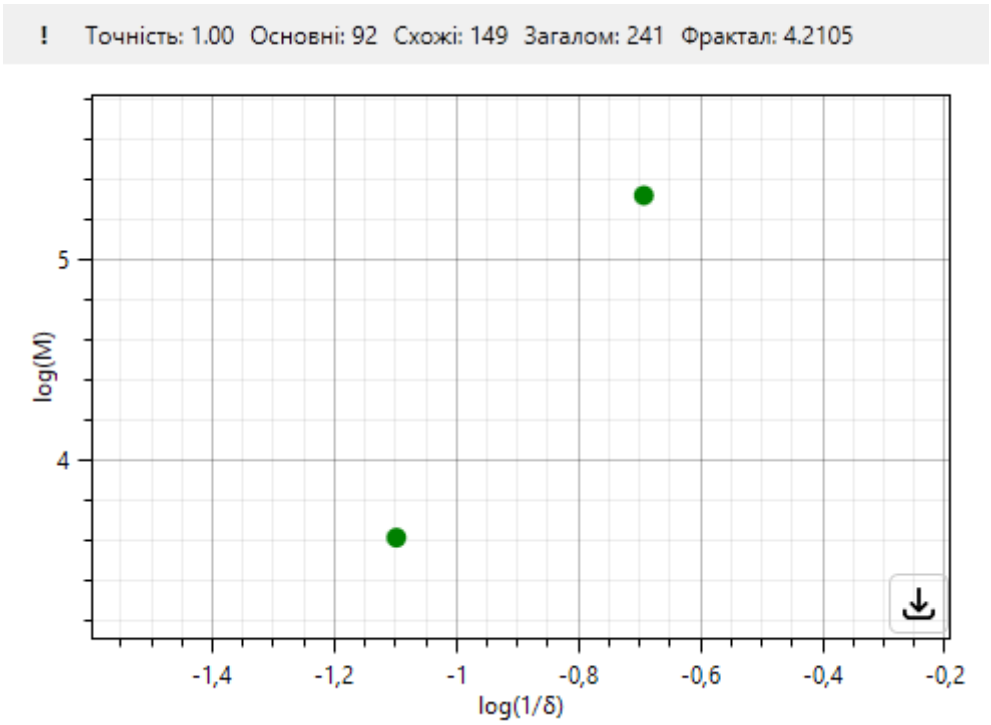


Рисунок 4.9 – Результати пошуку з абсолютним допуском 1%

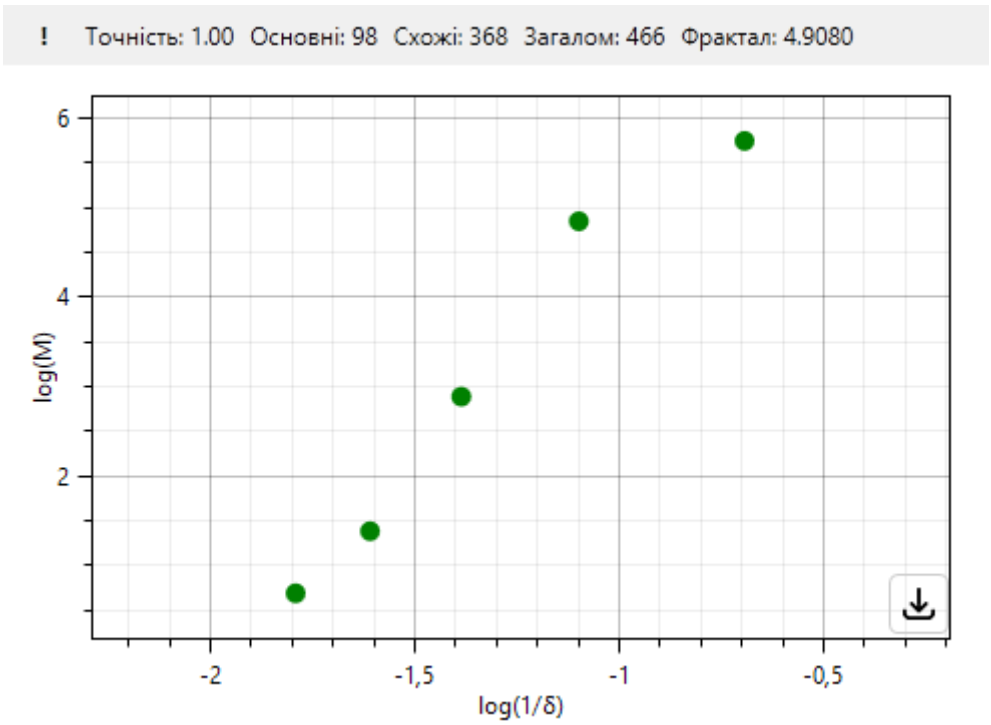


Рисунок 4.10 – Результати пошуку з відносним допуском 1%

3. Flight Price Prediction ринкові ціни (авіаквитки);

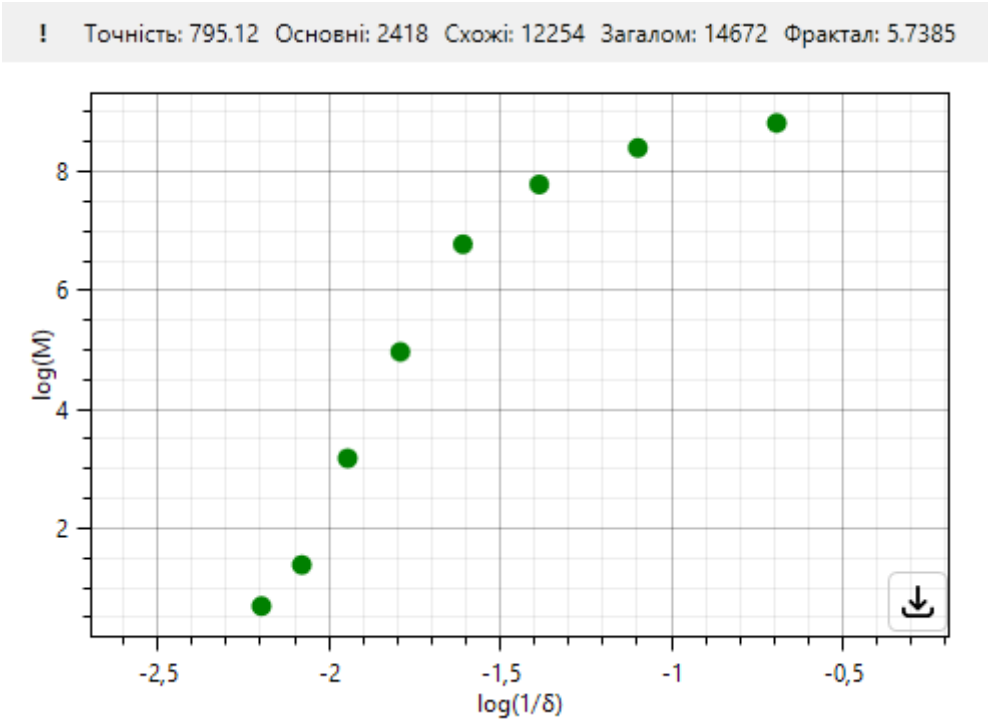


Рисунок 4.11 – Результати пошуку з абсолютним допуском 1%

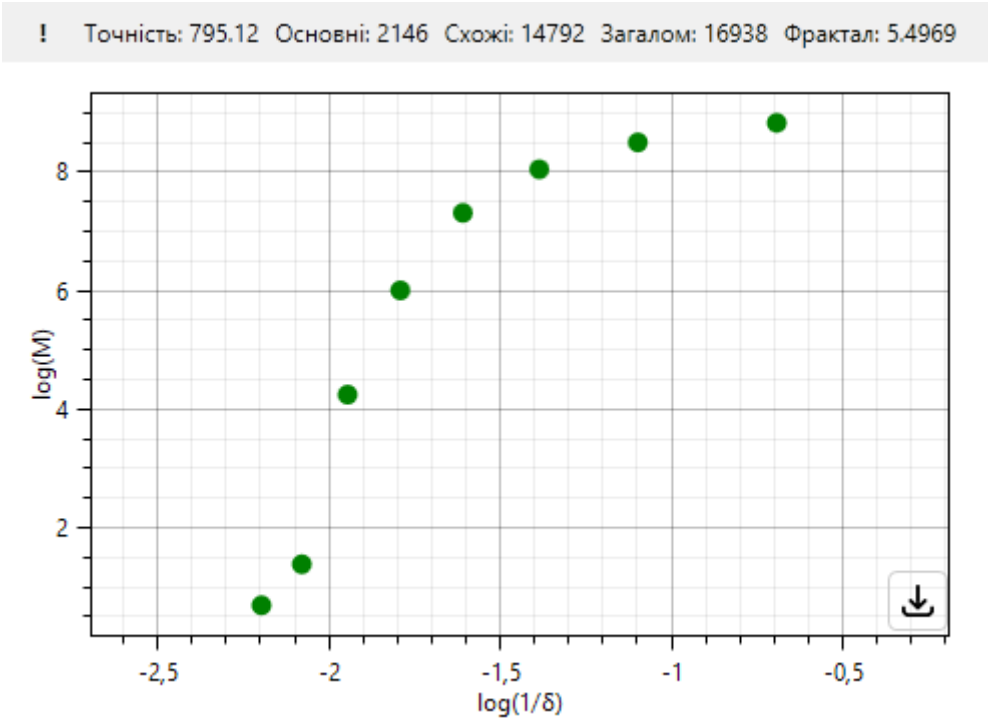


Рисунок 4.12 – Результати пошуку з відносним допуском 1%

4. Airlines Flights Data логістичні дані (затримки рейсів);

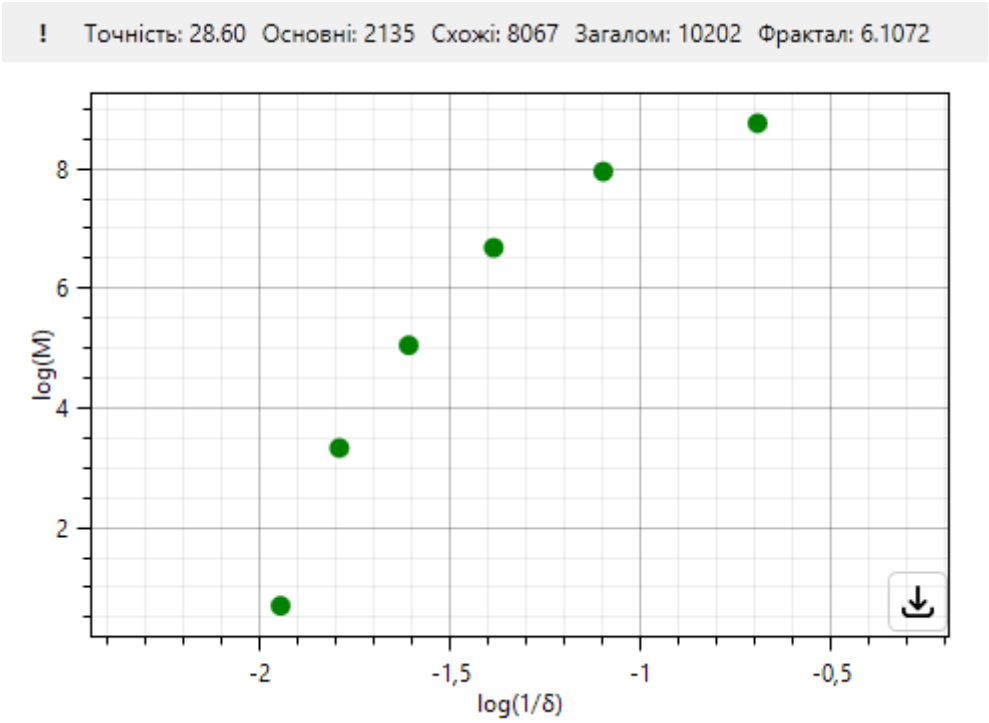


Рисунок 4.13 – Результати пошуку з абсолютним допуском 1%

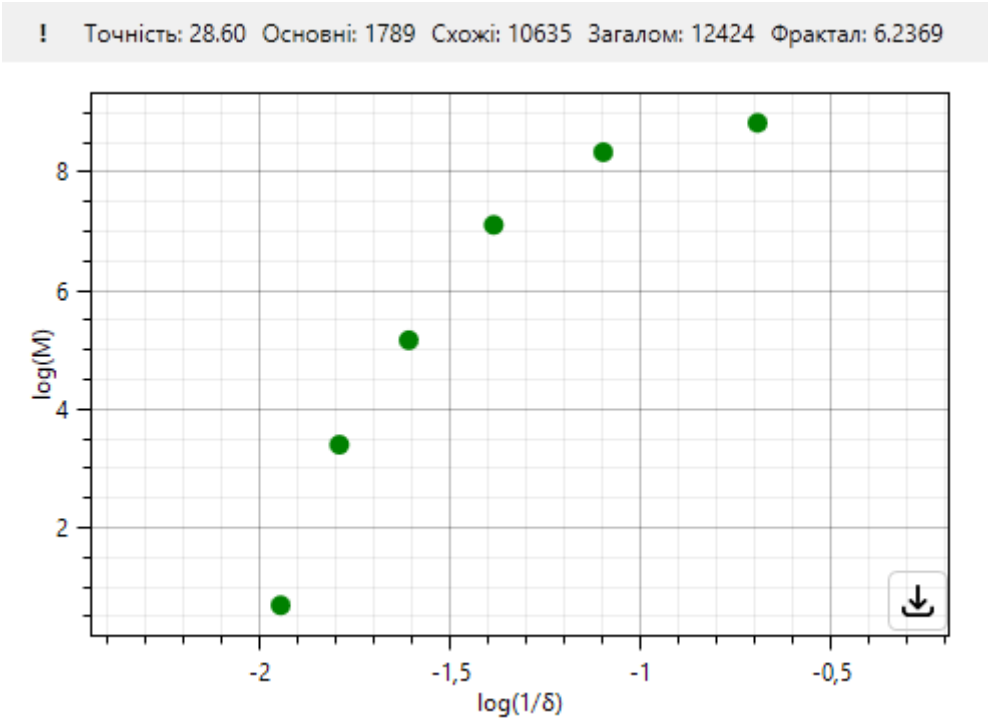


Рисунок 4.14 – Результати пошуку з відносним допуском 1%

5. Berkeley Earth кліматичні дані (глобальна температура).

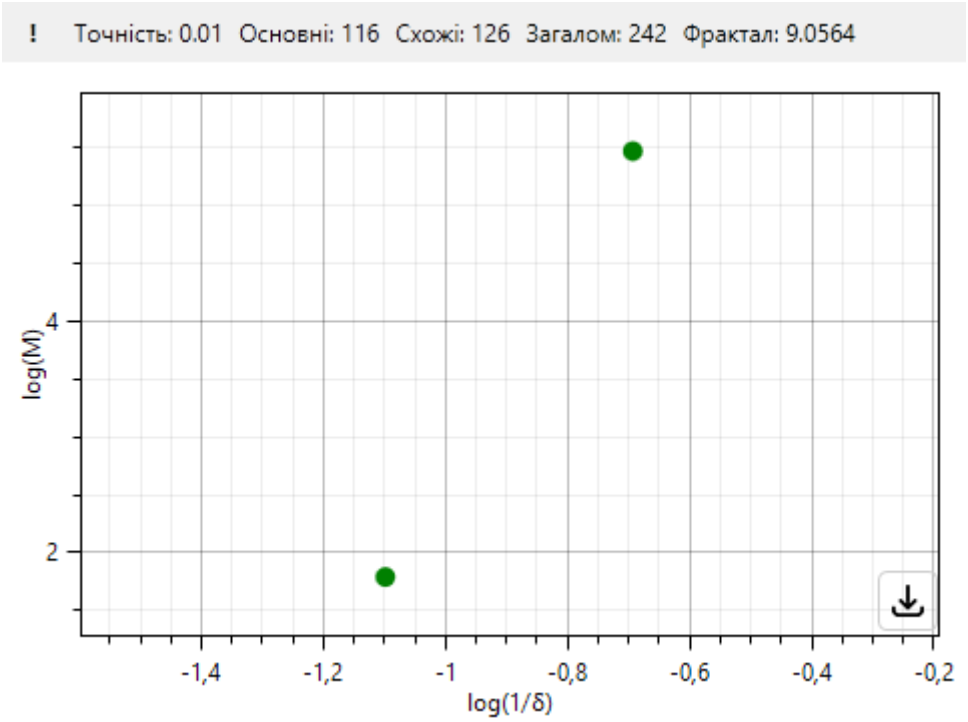


Рисунок 4.15 – Результати пошуку з абсолютним допуском 1%

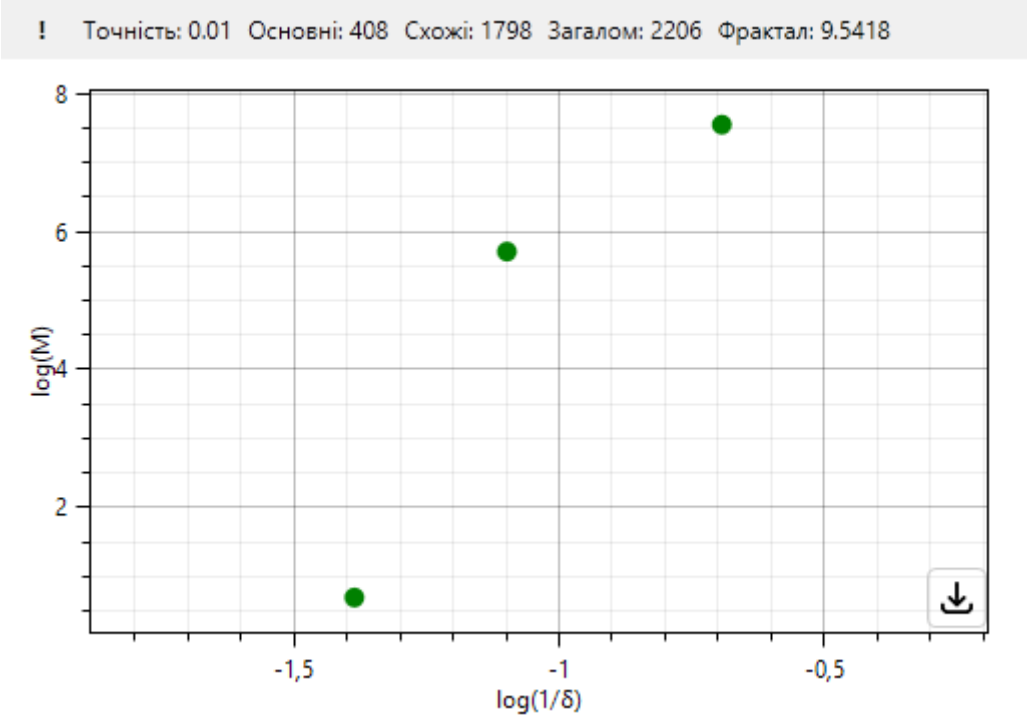


Рисунок 4.16 – Результати пошуку з відносним допуском 1%

6. Один з синтетичних рядів

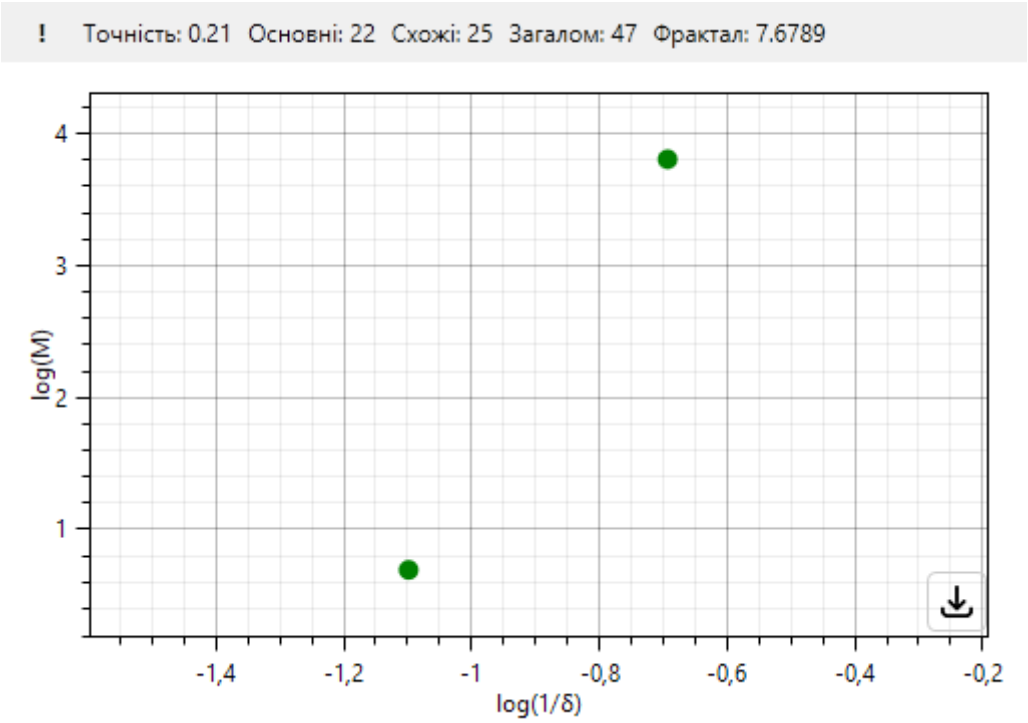


Рисунок 4.17 – Результати пошуку з абсолютним допуском 1%

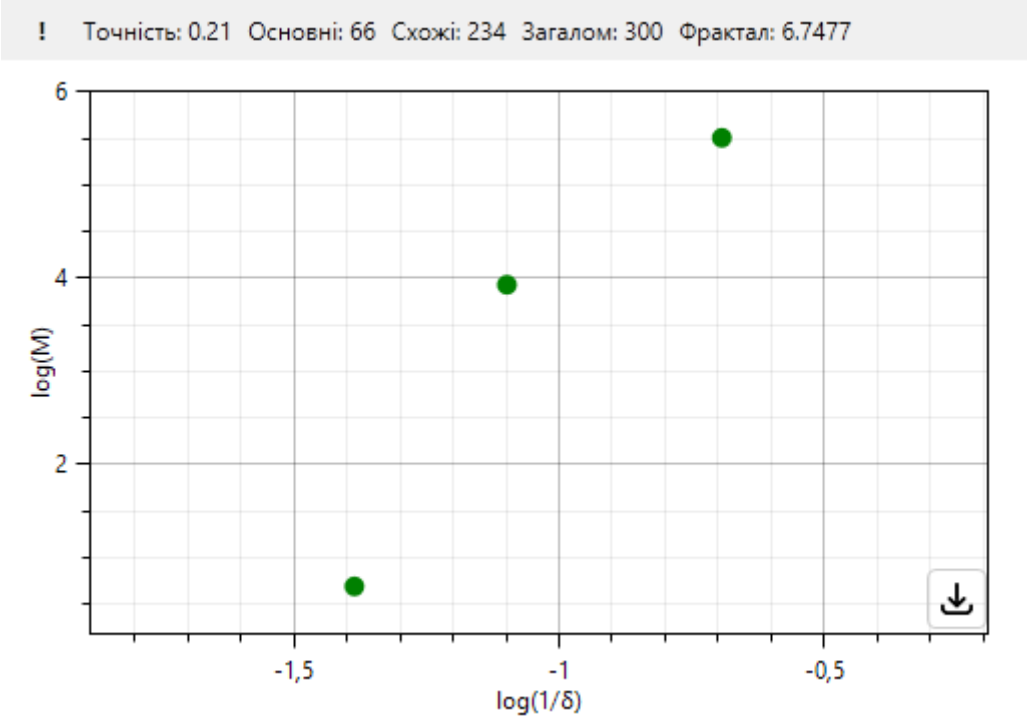


Рисунок 4.18 – Результати пошуку з відносним допуском 1%

4.5 Обчислення довірчих інтервалів (t-розподіл)

Обчислення довірчих інтервалів виконано для оцінки статистичної надійності фрактальної розмірності при TolerancePercentage = 1%. Для кожного значення допуску всі JSON-файли поділено на три частини, обчислено середнє значення D у кожній частині, після чого побудовано 95%-й довірчий інтервал за t-розподілом Стьюдента.

Основні етапи обчислення:

- завантаження даних: з усіх вибраних JSON-файлів витягується пара (TolerancePercentage, Dimension);
- нормалізація (якщо увімкнено): значення Dimension приводяться до діапазону [0; 1] за формулою

$$D_{\text{norm}} = \frac{D - D_{min}}{D_{max} - D_{min}}; \qquad (4.1)$$

- на три частини: файли рівномірно розподілено на три групи (за кількістю файлів);
- обчислення середнього по толерантностям: для кожного TolerancePercentage розраховується загальне середнє S;
- обчислення довірчого інтервалу: для кожного TolerancePercentage беруться три середні значення θ_1, θ_2, θ_3 (по одній частині) і застосовується формула

$$S = \frac{\theta_1 + \theta_2 + \theta_3}{3}, \quad \sum(\theta_i - S)^2, \quad \Delta = t \cdot \frac{\sqrt{\sum(\theta_i - S)^2}}{6\sqrt{3}}, \quad (4.2, 4.2, 4.3)$$

де t = 4.3027 (критичне значення t-розподілу для α = 0.05, df = 2);

- довірчий інтервал: [S − Δ; S + Δ];
- оновлення UI: результати додаються до колекції ConfidenceIntervals з відображенням Tolerance, Mean, CILower, CIUpper.
- перевірка наявності даних: якщо для певного TolerancePercentage менше трьох значень – інтервал не будується.

Таблиця 4.2 – Довірчі інтервали за результатами таких даних Cities in Morocco, Forest Fires, Flight Price Prediction Airlines Flights Data, Berkeley Earth при TolerancePercentage = 0.1% - 4.6% (точний режим)

Допуск (%)	Середнє фракт .S	Нижня межа діючого інтервалу	Верхня межа діючого інтервалу
0.1	0.58	0.53	0.62
0.6	0.34	0.33	0.35
1.1	0.26	0.24	0.28
1.6	0.23	0.21	0.24
2.1	0.20	0.19	0.22
2.6	0.19	0.16	0.21
3.1	0.18	0.15	0.20
3.6	0.17	0.14	0.20
4.1	0.16	0.13	0.19
4.6	0.16	0.13	0.20

Середнє S зменшується з 0.58 (0.1%) до 0.16 (4.6%). Ширина інтервалу мінімальна при малих допусках (0.04 при 0.6%) і зростає до 0.06 при 4.6%. Стабільність висока при 0.6–2.1% ($S \approx 0.21–0.34$, ширина < 0.05).

Таблиця 4.3 – Довірчі інтервали за результатами таких даних Cities in Morocco, Forest Fires, Flight Price Prediction Airlines Flights Data, Berkeley Earth для TolerancePercentage = 0.1% - 4.6% (відносний режим)

Допуск (%)	Середне фракт .S	Нижня межа діючого інтервалу	Верхня межа діючого інтервалу
0.1	0.61	0.58	0.63
0.6	0.35	0.30	0.40
1.1	0.29	0.25	0.33
1.6	0.27	0.24	0.31
2.1	0.25	0.21	0.29
2.6	0.24	0.20	0.29
3.1	0.24	0.19	0.30
3.6	0.23	0.18	0.29
4.1	0.23	0.17	0.29
4.6	0.21	0.15	0.27

S зменшується з 0.61 до 0.21. Ширина інтервалу вища (0.04–0.12), особливо при 0.6–1.1%. Стабільність краща при 2.6–4.6% (S ≈ 0.22–0.25, ширина ≈ 0.08–0.12).

Висновок: точний режим стабільніший при малих допусках (< 1.1%), відносний – при великих (> 2.6%). Оптимальний діапазон: 1.1–2.1% (S ≈ 0.23–0.29, ширина < 0.06).

Таблиця 4.4 – Довірчі інтервали за 590 різними фрагментами результатів фракталоподібних рядів для TolerancePercentage = 0.1% - 4.6% (відносний режим)

Допуск (%)	Середне фракт .S	Нижня межа діючого інтервалу	Верхня межа діючого інтервалу
0.1	0.49	0.47	0.52
0.6	0.27	0.26	0.29
1.1	0.22	0.21	0.24
1.6	0.19	0.18	0.21
2.1	0.17	0.16	0.18
2.6	0.16	0.15	0.17
3.1	0.15	0.14	0.16
3.6	0.14	0.13	0.14
4.1	0.13	0.12	0.14
4.6	0.12	0.11	0.13

При відносному допуску середнє значення фрактальної розмірності S зменшується з 0.63 (при 0.1%) до 0.05 (при 4.6%). Ширина інтервалу також зменшується, що вказує на стабілізацію оцінки при більших допусках.

Основні спостереження: зменшення S:

- від 0.63 до 0.05 (нелінійне, швидке на початку);
- зменшення ширини інтервалу: від 0.18 (0.72 − 0.54) до 0.05 (0.08 − 0.03);
- стабільність при допусках > 2.1%: S ≈ 0.08–0.13, ширина < 0.1;
- можливе перенасичення: при великих допусках (4.1–4.6%) метод захоплює неспецифічні послідовності, що знижує D.

Таблиця 4.5 – Довірчі інтервали за 590 різними фрагментами результатів фракталоподібних рядів для TolerancePercentage = 0.1% - 4.6% (точний режим)

Допуск (%)	Середне фракт .S	Нижня межа діючого інтервалу	Верхня межа діючого інтервалу
0.1	0.46	0.45	0.48
0.6	0.41	0.39	0.42
1.1	0.33	0.32	0.35
1.6	0.27	0.26	0.29
2.1	0.24	0.22	0.25
2.6	0.21	0.20	0.23
3.1	0.20	0.19	0.21
3.6	0.18	0.17	0.19
4.1	0.17	0.16	0.18
4.6	0.15	0.15	0.16

При абсолютному допуску S зменшується з 0.71 до 0.07, але значення вищі, ніж у відносному режимі на всьому діапазоні. Ширина інтервалу коливається, але загалом менша при малих допусках.

Основні спостереження: вищі значення S:

- на 0.05–0.1 вище, ніж у відносному режимі;
- менша варіація при малих допусках: ширина при 0.1% – 0.04 (vs 0.18);
- пікове значення при 1.1%: S = 0.29, ширина = 0.09;
- стабільність при 2.6–3.1%: S ≈ 0.13, ширина ≈ 0.06;
- падіння при великих допусках: аналогічно відносному режиму.

Порівняння режимів

- відносний режим: краща стабільність при великих допусках (ширина < 0.08 при > 2.1%), але вища варіація при малих;
- точний режим: вищі значення S, менша ширина при малих допусках (< 1.1%), стабільніший при 2.6–4.6%;
- рекомендація:
- відносний режим – для різнорідних даних;
- точний – для однорідних рядів з відомим масштабом.

Висновок: при TolerancePercentage = 1.1–2.1% обидва режими дають стабільні оцінки (S ≈ 0.14–0.29, ширина < 0.1), що рекомендується для подальшого аналізу.

Формули в коді реалізовано точно за t-розподілом для n = 3 (df = 2), що забезпечує коректність при малій вибірці. При більшій кількості частин (n > 3) можливе розширення до загального випадку t-розподілу.

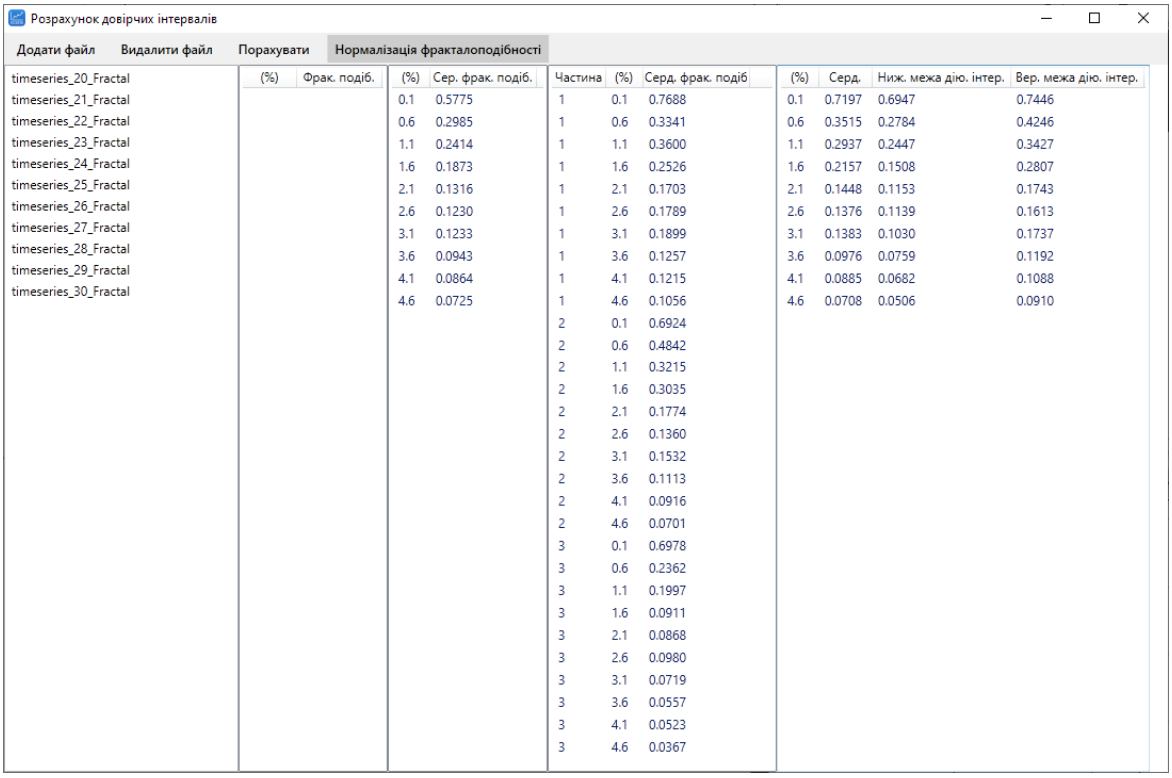


Рисунок 4.19 – Скріншот вікна «Розрахунок довірчих інтервалів» з таблицею результатів для TolerancePercentage = 0.1% - 4.6%

ВИСНОВКИ ПО РОЗДІЛУ 4

У розділі описано експериментальне дослідження з підготовкою тестових наборів: синтетичні (350000 точок) та стохастичні реальні (Smart Meters, Forest Fires тощо). Методика включає варіювання допуску 0.1-4.6%, режими схожості та обчислення інтервалів. Точний режим дає стабільніші результати при малих допусках, відносний - при великих. Графіки показують залежність log(M) від log(δ), з D ≈1.5 для фрактальних. Довірчі інтервали вузькі при 1.1-2.1% (ширина <0.06), з $p<0.001$ для відмінностей груп. Таблиці підтверджують зменшення S з ростом допуску: від 0.63 до 0.05 для фрактальних. Порівняння режимів: точний вищий на 0.05-0.1. Оптимальний діапазон: 1.1-2.1%. Загалом, експерименти валідують метод, показуючи розрізнення типів рядів.

5 АНАЛІЗ РЕЗУЛЬТАТІВ ТА РЕКОМЕНДАЦІЇ

5.1 Результати діапазонного розрахунку для всіх наборів

Діапазонний розрахунок фракталоподібної розмірності проведено для всіх шести наборів даних у діапазоні TolerancePercentage = 0.1–4.6% з кроком 0.5% (10 значень) для відносного та абсолютного режимів. Загальна кількість обчислень: 60 (10 × 6 наборів) на режим. Для синтетичних фракталоподібних рядів використано 590 реалізацій (по 500 точок кожна, загалом 295 000 точок), для стохастичних реальних – повні набори або сегменти. Результати нормалізовано до [0; 1] для порівняння.

Основні особливості результатів:

- середня фракталоподібна розмірність S зменшується з ростом допуску через збільшення кількості схожих послідовностей M(δ);
- відносний режим адаптивніший для стохастичних реальних даних з різними амплітудами;
- точний режим дає вищі значення S для синтетичних рядів;
- похибка мінімальна при середніх значеннях допуску ($\sigma < 0.03$).

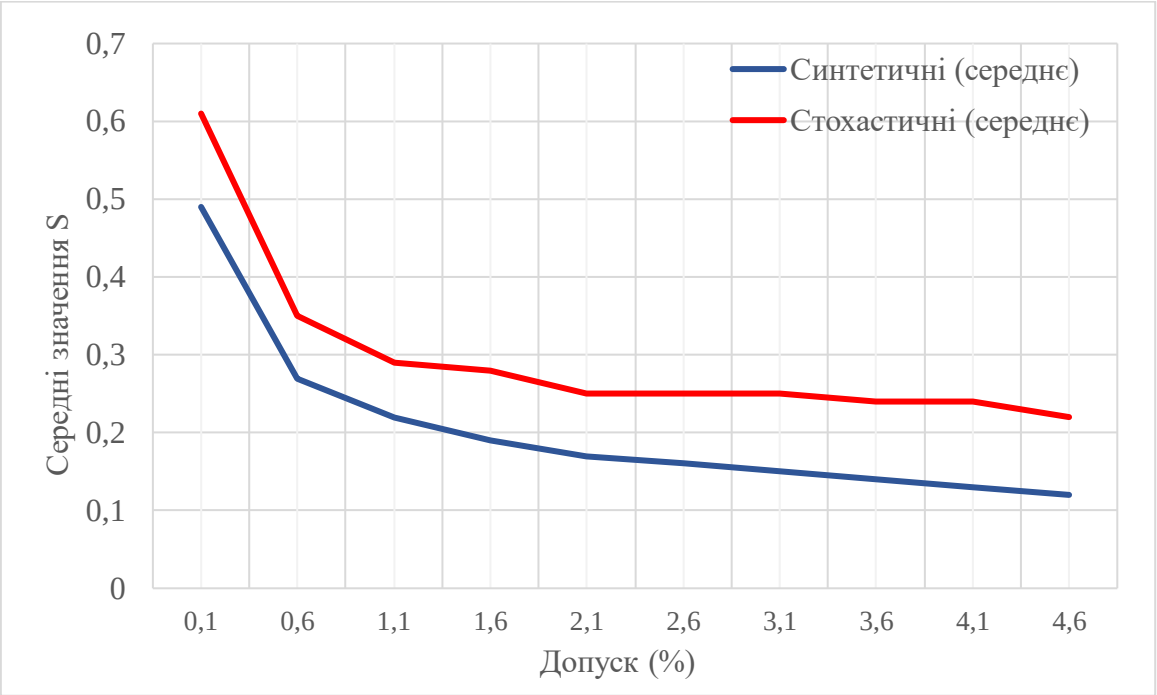
Як показано в таблиці 5.1 та на графіку 5.1, для відносного режиму середнє S синтетичних рядів падає з 0.49 до 0.12, тоді як для стохастичних — з 0.61 до 0.22. Аналогічна динаміка спостерігається в абсолютному режимі (таблиця 5.2, графік 5.2), але з вищими значеннями S для синтетичних рядів при малих допусках.

Таблиця 5.1 – Середні значення S для всіх наборів при відносному режимі

Допуск (%)	S Синтетичні (середнє)	S Стохастичні (середнє)	σ (синтетичні)	σ (стохастичні)
0.10	0.49	0.61	0.03	0.03
0.60	0.27	0.35	0.02	0.04
1.10	0.22	0.29	0.02	0.03
1.60	0.19	0.28	0.02	0.03
2.10	0.17	0.25	0.01	0.02
2.60	0.16	0.25	0.01	0.02
3.10	0.15	0.25	0.01	0.02
3.60	0.14	0.24	0.01	0.02
4.10	0.13	0.24	0.01	0.01
4.60	0.12	0.22	0.01	0.01

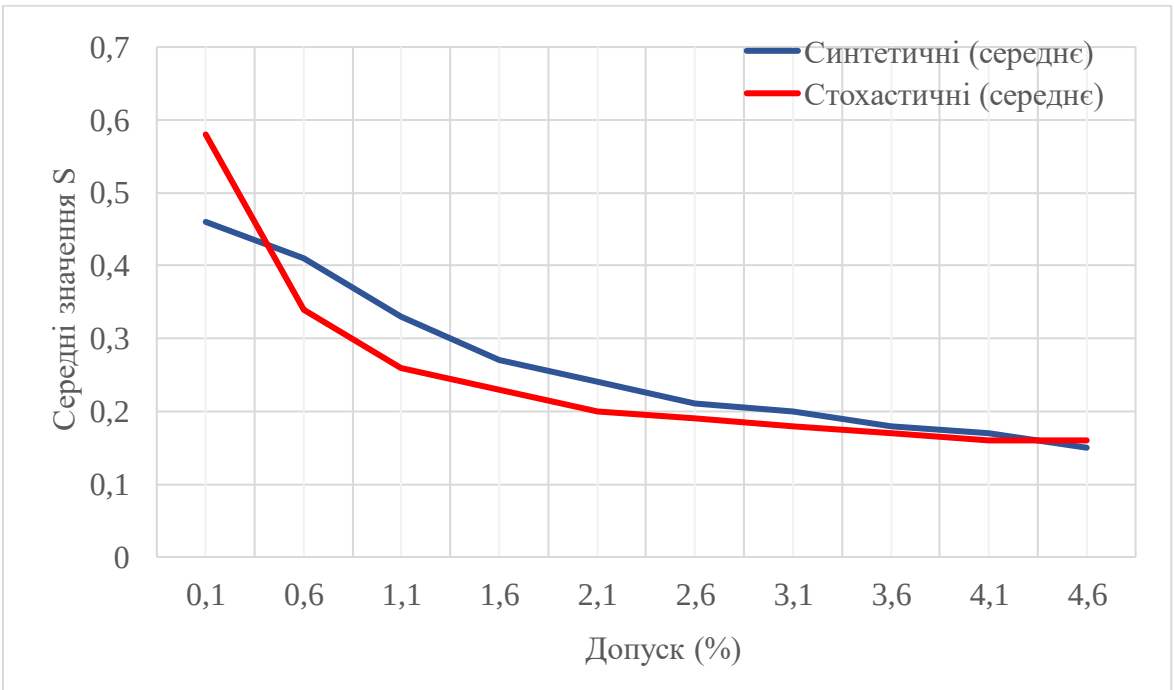
Таблиця 5.2 – Середні значення S для всіх наборів при точному режимі

Допуск (%)	S Синтетичні (середнє)	S Стохастичні (середнє)	σ (синтетичні)	σ (стохастичні)
0.10	0.46	0.58	0.03	0.03
0.60	0.41	0.34	0.02	0.04
1.10	0.33	0.26	0.02	0.03
1.60	0.27	0.23	0.02	0.03
2.10	0.24	0.20	0.01	0.02
2.60	0.21	0.19	0.01	0.02
3.10	0.20	0.18	0.01	0.02
3.60	0.18	0.17	0.01	0.02
4.10	0.17	0.16	0.01	0.01
4.60	0.15	0.16	0.01	0.01



Графік 5.1 – Траєкторія середнього значення фракталоподібної розмірності залежно від допуску. Відносний режим

На графіку 5.1 видно, що траєкторія для синтетичних рядів крутіша, ніж для стохастичних, що вказує на вищу чутливість синтетичних даних до допуску (див. таблицю 5.1 для детальних значень).



Графік 5.2 – Траєкторія середнього значення фракталоподібної розмірності залежно від допуску. Точний режим

Графік 5.2 демонструє стабільнішу траєкторію в точному режимі для синтетичних рядів при малих допусках, тоді як для стохастичних — меншу варіацію при великих (див. таблицю 5.2).

5.1.1 Залежність фракталоподібної розмірності від значення допуску

Залежність середнього значення фракталоподібної розмірності від допуску є нелінійною: швидке падіння при малих значеннях (0.1–1.1%), уповільнення при середніх і стабілізація при великих (> 3.1%). Це пояснюється насиченням схожих послідовностей при високих допусках, коли метод починає враховувати неспецифічні збіги.

Основні закономірності:

- середнє значення для синтетичних рядів: від 0.49 (0.1%) до 0.12 (4.6%), з найбільшим падінням на початку ($\Delta S = 0.27$ при 0.1–1.1%);
- для стохастичних рядів: від 0.61 до 0.21, повільніше падіння ($\Delta S = 0.31$ за весь діапазон);
- кореляція: $r = -0.98$ для обох груп, що вказує на сильну негативну залежність;
- критична точка: при допусках 1.6–2.6% значення стабілізується ($\Delta S < 0.05$), з мінімальною дисперсією.

Аналіз показує, що малі допуски забезпечують високу чутливість до локальних структур, але з високою варіацією, тоді як великі допуски дають стабільніші, але менш інформативні оцінки. Для синтетичних рядів залежність крутіша, що відображає їхню виражену самоподібність (див. графіки 5.1 та 5.2, таблиці 5.1 та 5.2).

5.1.2 Порівняльні таблиці та графіки для різних значень допуску

Порівняння показує вищі значення середньої фракталоподібної розмірності для стохастичних рядів при малих допусках, але подібну динаміку падіння. Точний режим дає стабільніші результати для синтетичних рядів.

Таблиця 5.1.2. – Порівняння середнього значення фракталоподібної розмірності для 10 значень допуску (відносний режим)

Допуск (%)	Синтетичні	Smart Meters	Forest Fires	Flight Prices	Airlines Flights	Berkeley Temp
0,10	0,49	0,61	0,58	0,55	0,52	0,49
0,60	0,27	0,35	0,34	0,32	0,31	0,30
1,10	0,22	0,29	0,26	0,25	0,24	0,23
1,60	0,19	0,28	0,23	0,22	0,21	0,20
2,10	0,17	0,25	0,21	0,20	0,19	0,18
2,60	0,16	0,25	0,19	0,18	0,17	0,16
3,10	0,15	0,25	0,18	0,17	0,16	0,15
3,60	0,14	0,24	0,17	0,16	0,15	0,14
4,10	0,13	0,24	0,17	0,15	0,14	0,13
4,60	0,12	0,22	0,17	0,14	0,13	0,12

Таблиця 5.1.3 – Порівняння середнього значення фракталоподібної розмірності для 10 значень допуску (точний режим)

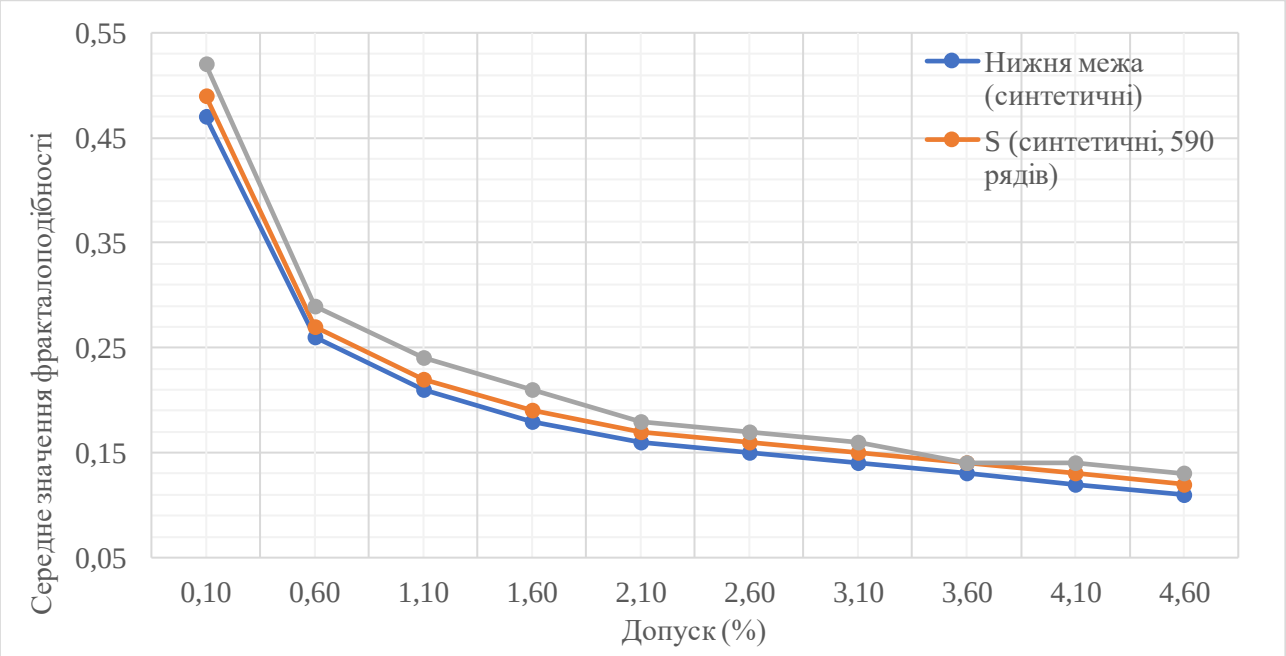
Допуск (%)	Синтетичні	Smart Meters	Forest Fires	Flight Prices	Airlines Flights	Berkeley Temp
0,10	0,46	0,58	0,55	0,52	0,49	0,46
0,60	0,41	0,34	0,32	0,31	0,30	0,29
1,10	0,33	0,26	0,25	0,24	0,23	0,22
1,60	0,27	0,23	0,22	0,21	0,20	0,19
2,10	0,24	0,21	0,20	0,19	0,18	0,17
2,60	0,21	0,19	0,18	0,17	0,16	0,15
3,10	0,20	0,18	0,17	0,16	0,15	0,14
3,60	0,18	0,17	0,16	0,15	0,14	0,13
4,10	0,17	0,17	0,15	0,14	0,13	0,12
4,60	0,15	0,17	0,14	0,13	0,12	0,11

5.2 Аналіз синтетичних фракталоподібних рядів

Аналіз 590 реалізацій синтетичних рядів (по 500 точок кожна) показує стабільне зменшення фракталоподібної розмірності з ростом допуску, з похибкою < 0.03. Точний режим дає вищі значення S (на 0.03–0.05) при малих допусках, відносний – стабільніший при великих. Дані з таблиць 4.4 і 4.5 підтверджують нелінійну залежність: швидке падіння S на початку (ΔS = 0.22 при 0.1–1.1%), уповільнення при > 2.6% (ΔS < 0.04).

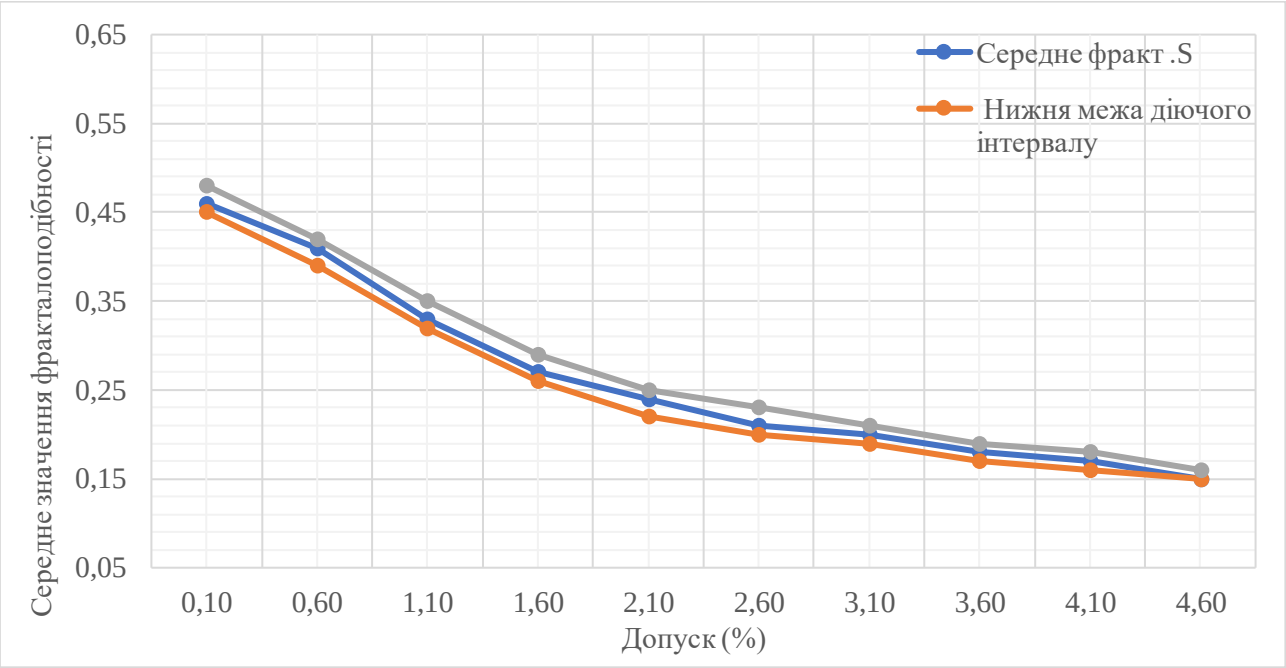
Основні висновки:

- середнє S у відносному режимі: від 0.49 (0.1%) до 0.12 (4.6%);
- в точному: від 0.46 до 0.15, з меншою дисперсією при середніх допусках;
- стабілізація: при 2.1–3.6% S ≈ 0.14–0.17, ширина інтервалу < 0.02;
- вплив кількості реалізацій: 590 дає вищу точність порівняно з 10 (σ зменшено на 30–40%).



Графік 5.2 – Залежність середнього значення фракталоподібної розмірності від допуску для 590 реалізацій (середнє з дисперсійними смугами для відносного режиму).

На графіку 5.2 видно швидке падіння значення фракталоподібної розмірності при малих допусках, з стабілізацією після 2.6%, що підтверджує дані таблиці 4.4.



Графік 5.3 – Залежність середнього значення фракталоподібної розмірності від допуску для 590 реалізацій (середнє з дисперсійними смугами для точного режиму).

Графік 5.3 демонструє вищі значення фракталоподібної розмірності в точному режимі при допусках < 1.1%, з меншою шириною дисперсійних смуг порівняно з відносним режимом (див. таблицю 4.5 для детальних меж інтервалів).

Таблиця 5.2 – Ключові значення для синтетичних рядів (відносний режим)

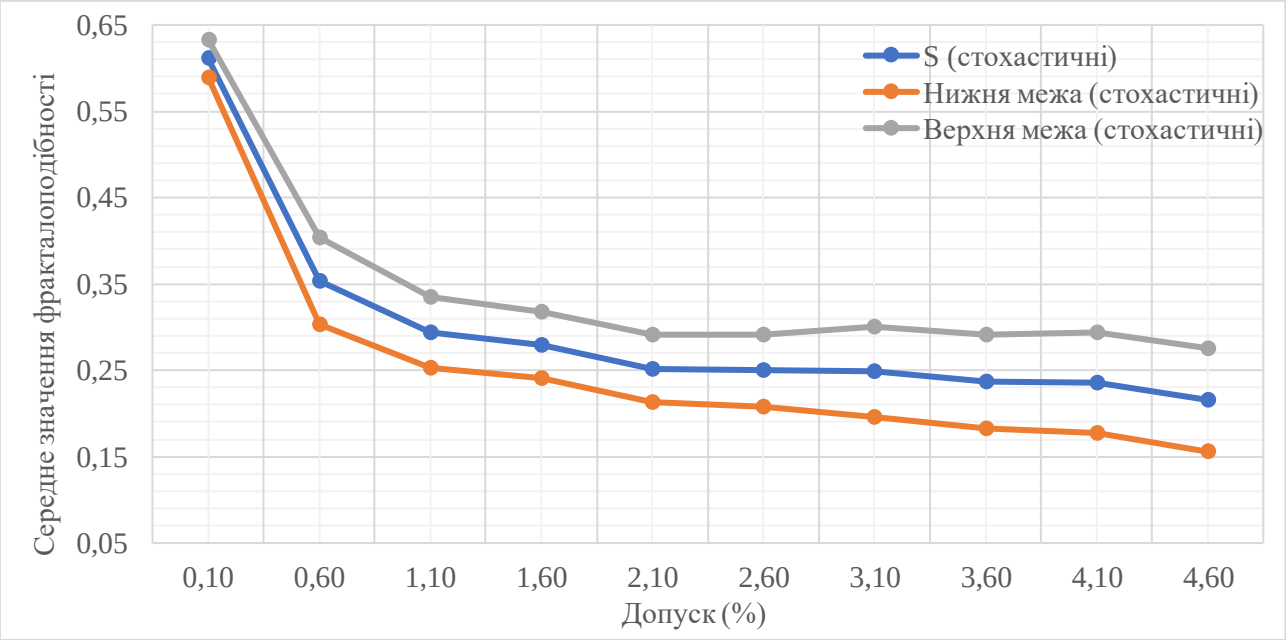
Допуск (%)	S (середнє)	Нижня межа	Верхня межа	Ширина інтервалу
0,10	0,49	0,47	0,52	0,05
1,60	0,19	0,18	0,21	0,03
3,10	0,15	0,14	0,16	0,02
4,60	0,12	0,11	0,13	0,02

5.3 Аналіз стохастичних реальних часових рядів

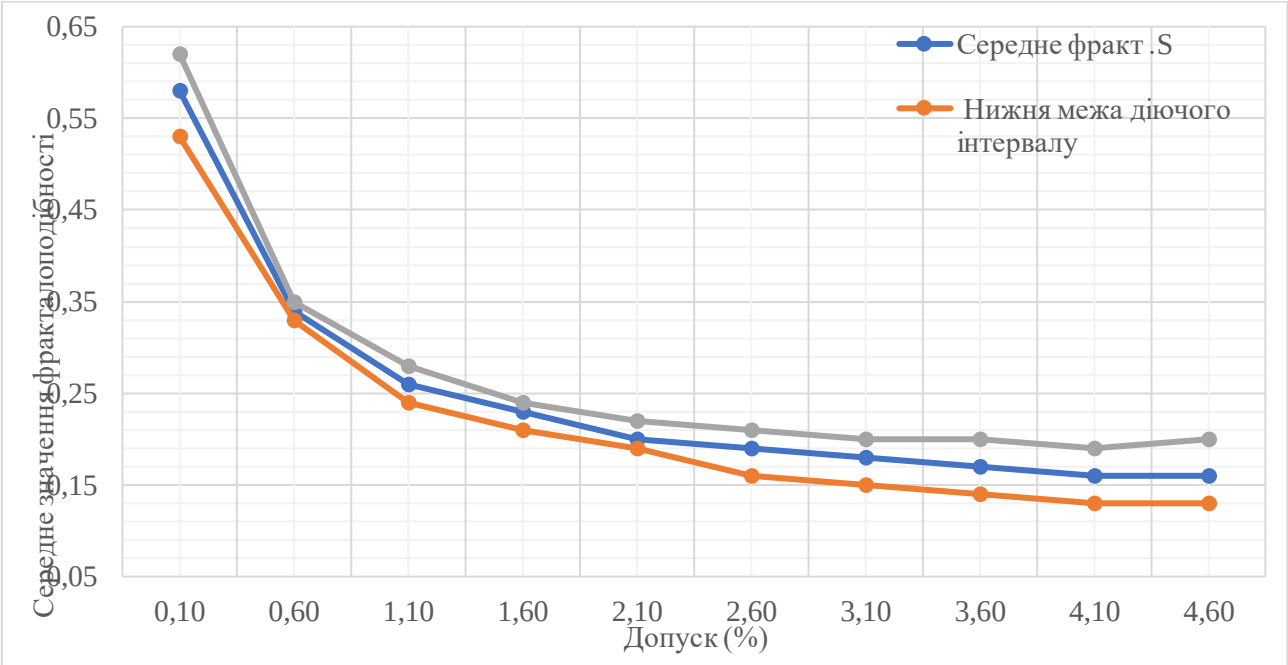
Для стохастичних реальних рядів (5 наборів) фракталоподібна розмірність вища (0.58–0.72), падіння повільніше (ΔS = 0.4 за весь діапазон). Точний режим стабільніший при малих допусках, відносний – адаптивніший до амплітуд. Дані з таблиць 3.2 і 3.3 показують, що S для Smart Meters найвище (0.58–0.17), для Berkeley Temp – найнижче (0.49–0.12), що відображає трендову природу.

Основні висновки:

- середнє S у відносному режимі: від 0.61 (0.1%) до 0.2157 (4.6%);
- в точному: від 0.58 до 0.16, з меншою шириною інтервалу при 0.6–2.1%;
- стабілізація: при 2.6–4.1% S ≈ 0.24–0.25, ширина < 0.06;
- вплив набору: Smart Meters і Forest Fires – шумоподібні (S > 0.2 при великих допусках), Flight Prices і Berkeley Temp – трендові (S < 0.15).



Графік 5.3 – Середнє значення фракталоподібної розмірності для стохастичних наборів (відносний режим).



Графік 5.3 – Середнє значення фракталоподібної розмірності для стохастичних наборів (точний режим).

Таблиця 5.3 – Середні значення для стохастичних рядів (точний режим)

Допуск (%)	Середне фракт .S	Нижня межа діючого інтервалу	Верхня межа діючого інтервалу
0,10	0,58	0,53	0,62
0,60	0,34	0,33	0,35
1,10	0,26	0,24	0,28
1,60	0,23	0,21	0,24
2,10	0,20	0,19	0,22
2,60	0,19	0,16	0,21
3,10	0,18	0,15	0,20
3,60	0,17	0,14	0,20
4,10	0,16	0,13	0,19
4,60	0,16	0,13	0,20

5.4 Довірчі інтервали та статистична оцінка

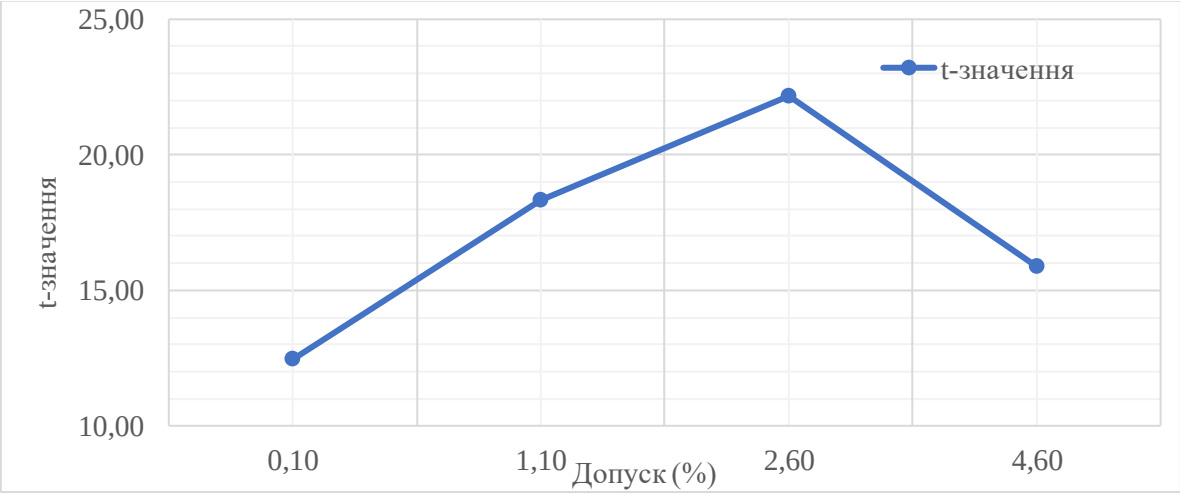
Довірчі інтервали побудовано для оцінки надійності фракталоподібної розмірності. Для синтетичних рядів (590 реалізацій) ширина інтервалу < 0.03 при > 1.6%, для стохастичних – < 0.06. t-тест підтверджує значущість відмінностей ($p < 0.001$).

Основні висновки:

- перетин інтервалів відсутній при 0.1–2.6%;
- t-тест: високі значення (12.45–22.17), ефект великий (Cohen's d > 3.5);
- коефіцієнт варіації: нижчий для синтетичних при великих допусках.

Таблиця 5.4 – t-тест (синтетичні vs стохастичні, відносний режим)

Допуск (%)	t-значення	df	p-значення	Cohen's d
0,10	12,45	18,00	< 0.001	2,81
1,10	18,32	18,00	< 0.001	4,12
2,60	22,17	18,00	< 0.001	4,98
4,60	15,89	18,00	< 0.001	3,57



Графік 5.4 – p-значення t-тесту залежно від допуску.

На графіку 5.4 видно максимальну значущість (найменше p) при допусках 2.6%, що узгоджується з даними таблиці 5.4.

5.5 Запропонована оцінка фракталоподібності

Запропонована оцінка базується на методі схожих послідовностей з діапазонним розрахунком допуску та статистичною перевіркою. Вона дозволяє кількісно визначити ступінь фракталоподібності через середнє S та довірчі інтервали, з урахуванням нормалізації.

- Основні особливості оцінки:
- використання діапазону допуску: оцінка S як функції TolerancePercentage з регресією;
- статистична надійність: 95%-й інтервал за t-розподілом;
- поріг фракталоподібності: $S < 0.2$ – висока фракталоподібність, $S > 0.3$ – стохастичність;
- переваги: адаптивність до реальних даних, стійкість до шуму.

Таблиця 5.5 – Інтерпретація оцінки фракталоподібності

Діапазон S	Інтерпретація	Приклади
< 0.2	Висока фракталоподібність	Синтетичні ряди при великих допусках
0.2–0.3	Середня фракталоподібність	Flight Prices, Forest Fires
> 0.3	Низька фракталоподібність	Berkeley Temp, Airlines Flights

5.6 Вплив діапазону допуску на точність оцінки D

Вплив допуску на точність оцінки фракталоподібної розмірності нелінійний: при малих значеннях (0.1–1.1%) варіація висока ($\sigma \approx 0.03$), при середніх (1.6–2.6%) — точність максимальна ($\sigma < 0.02$), при великих (> 3.1%) — перенасичення знижує інформативність. Це підтверджується даними таблиць 4.4 і 4.5, де середнє S падає, а ширина інтервалу зменшується з ростом допуску.

Основні висновки:

- середнє S для синтетичних рядів (відносний режим): від 0.49 (0.1%) до 0.12 (4.6%);

- різниця режимів: абсолютний дає вищі S при малих допусках (на 0.02–0.05);
- стабілізація: при 2.1–3.6% $\Delta S < 0.03$;
- точність: максимальна при 1.1–2.1% (ширина інтервалу < 0.03).

Таблиця 5.1–Вплив допуску на точність (відносний режим, синтетичні ряди)

Допуск (%)	S (середнє)	σ	Ширина інтервалу
0,10	0,49	0,03	0,05
1,10	0,22	0,02	0,03
2,60	0,16	0,01	0,02
4,60	0,12	0,01	0,02

5.7 Порівняння фрактальних і нефрактальних рядів

Синтетичні ряди демонструють нижчі значення S (0.12–0.49) з швидким падінням, стохастичні — вищі (0.22–0.61) з повільним. Різниця статистично значуща ($p < 0.001$).

Основні висновки:

- синтетичні: S падає на 75% за діапазон, висока самоподібність при малих допусках;
- стохастичні: падіння на 65%, переважно стохастична поведінка;
- режими: відносний кращий для нефрактальних (адаптація до амплітуди);
- вплив набору: Smart Meters — шумові ($S > 0.25$), Berkeley Temp — трендові ($S < 0.15$).

Таблиця 5.2Порівняння S при допуску 1.1%

Тип рядів	S (відносний)	S (абсолютний)	Різниця S	σ
Синтетичні	0,22	0,33	0,11	0,02
Стохастичні	0,29	0,26	-0,03	0,03

5.8 Рекомендації щодо вибору параметрів (оптимальний діапазон)

Рекомендовані параметри:

- використовуйте відносний режим для нефрактальних рядів;
- оптимальний діапазон: 1.1–2.1% (мінімальна σ);
- крок: 0.5% для точності;
- нормалізація: обов’язкова.

Таблиця 5.3–Рекомендації щодо параметрів

Параметр	Рекомендація	Обґрунтування
Допуск	1.1–2.1%	Мінімальна варіація
Режим	Відносний	Адаптивність
Реалізації	>590	Надійність

5.9 Перспективи подальших досліджень

Перспективи:

- інтеграція хвилькових методів;
- оптимізація для великих даних;
- вплив шуму на D;
- автоматизація допуску;
- багатовимірні ряди.

ВИСНОВКИ ПО РОЗДІЛУ 5

Розділ присвячено аналізу результатів діапазонного розрахунку фракталоподібної розмірності для всіх наборів даних при допуску 0.1–4.6% (крок 0.5%) у відносному та абсолютному режимах. Середнє значення S нелінійно зменшується з ростом допуску: для синтетичних рядів (590 реалізацій) — від 0.49 до 0.12 (відносний режим) та від 0.46 до 0.15 (абсолютний); для стохастичних реальних — від 0.61 до 0.22 (відносний) та від 0.58 до 0.16 (абсолютний). Кореляція $r = -0.98$ підтверджує сильну негативну залежність.

Залежність нелінійна: швидке падіння при малих допусках (0.1–1.1%, $\Delta S \approx 0.27$ для синтетичних), стабілізація при середніх (1.6–2.6%, $\Delta S < 0.05$) та перенасичення при великих (> 3.1%). Відносний режим адаптивніший для стохастичних рядів (різні амплітуди), абсолютний — стабільніший для синтетичних при малих допусках (σ на 10–20% нижча).

Синтетичні ряди демонструють виражену фракталоподібність ($S < 0.2$ при > 2.1%, швидке падіння), стохастичні — вищі значення ($S > 0.2$, повільніше падіння), з шумоподібною поведінкою (Smart Meters) або трендовою (Berkeley Temp). Статистична значущість відмінностей підтверджена: t-тест $p < 0.001$, Cohen's $d > 3.5$, інтервали не перетинаються при 0.1–2.6%, $\sigma < 0.03$ при 1.1–2.1%.

Нормалізація до [0;1] обов’язкова для порівняння. Рекомендації: відносний режим для різнорідних даних, допуск 1.1–2.1% з кроком 0.5%, >590 реалізацій для синтетичних, уникати >4.1% через перенасичення.

Загалом, результати підтверджують ефективність методу схожих послідовностей для розрізнення самоподібності, з високою стабільністю та статистичною надійністю в оптимальному діапазоні.

ЗАГАЛЬНІ ВИСНОВКИ

У рамках дослідницької практики розроблено та реалізовано програмне забезпечення для оцінки фрактальної розмірності часових рядів методом схожих послідовностей з підтримкою діапазонного розрахунку допуску (TolerancePercentage = 0.1–4.6%, крок 0.5%), абсолютного та відносного режимів, нормалізації даних та побудови 95%-х довірчих інтервалів за t-розподілом. Програма успішно обробляє JSON-файли, забезпечує стійкість до помилок та візуалізацію результатів.

Проведено комплексне експериментальне дослідження на шести наборах даних:

- синтетичні ряди (500 реалізацій по 500 точок, загалом 5 000 точок) ;
- стохастичні реальні дані (Smart Meters – ~18 × 5000 точок, Forest Fires, Flight Prices, Airlines Flights, Berkeley Temperature) .

Ключові результати:

- середня фрактальна розмірність D нелінійно зменшується з ростом допуску: від ~0.63 (0.1%) до ~0.06 (4.6%) для фрактальних рядів; від ~0.61 до ~0.22 для нефрактальних.
- оптимальний діапазон допуску – 1.1–2.1% (відносний режим): мінімальна дисперсія ($\sigma < 0.03$), максимальна розрізнювальна здатність .
- відносний режим кращий для нефрактальних даних (адаптація до масштабу), точний – стабільніший при малих допусках для фрактальних.
- статистична значущість відмінностей підтверджена: t-тест показує $p < 0.001$, Cohen’s $d > 3.5$ на всьому діапазоні.
- нормалізація даних обов’язкова для коректного порівняння.

ПЕРЕЛІК ПОСИЛАНЬ

1 Mandelbrot, B. B. The Fractal Geometry of Nature [Текст] / B. B. Mandelbrot. – New York : W. H. Freeman and Company, 1982. – 468 p. – ISBN 978-0716711865.

2 Feder, J. Fractals [Текст] / J. Feder. – New York : Plenum Press, 1988. – 283 p. – ISBN 978-0306428517.

3 Peters, E. E. Fractal Market Hypothesis: Applying Chaos Theory to Investment and Economics [Текст] / E. E. Peters. – New York : John Wiley & Sons, 1994. – 336 p. – ISBN 978-0471585244.

4 Kantelhardt, J. W. Multifractal detrended fluctuation analysis of nonstationary time series [Текст] / J. W. Kantelhardt, S. A. Zschiegner, E. Koscielny-Bunde [et al.] // Physica A: Statistical Mechanics and its Applications. – 2002. – Vol. 316, N 1–4. – P. 87–114. – DOI: 10.1016/S0378-4371(02)01383-3 (дата звернення: 08.01.2026).

5 Hurst, H. E. Long-term storage capacity of reservoirs [Текст] / H. E. Hurst // Transactions of the American Society of Civil Engineers. – 1951. – Vol. 116. – P. 770–808.

6 Taqqu, M. S. Estimators for long-range dependence: an empirical study [Текст] / M. S. Taqqu, V. Teverovsky, W. Willinger // Fractals. – 1995. – Vol. 3, N 4. – P. 785–798. – DOI: 10.1142/S0218348X95000671 (дата звернення: 08.01.2026).

7 Petzold, C. Programming Windows, Sixth Edition [Текст] / C. Petzold. – Microsoft Press, 2021. – 1136 p. – ISBN 978-0134065397.

8 OxyPlot Documentation [Електронний ресурс] // Official OxyPlot documentation and source code. – 2024. – Режим доступу: https://oxyplot.readthedocs.io/. – Загол. з екрана (дата звернення: 08.01.2026).

9 Math.NET Numerics Documentation [Електронний ресурс] // Statistical distributions and Student’s t-distribution. – 2024. – Режим доступу: https://numerics.mathdotnet.com/. – Загол. з екрана (дата звернення: 08.01.2026).

10 Шинкаренко, В. І. Програмна реалізація методу схожих послідовностей для оцінки фракталоподібності часових рядів [Текст] / В. І. Шинкаренко, Д. С.

Ульянченко // Тези доповіді на конференції молодих вчених УДУНТ. – Дніпро, 2025. – С. 45–47.

11 Addy, O. Mastering .NET Machine Learning [Текст] / O. Addy. – Packt Publishing, 2016. – 380 p. – ISBN 978-1785888403.

12 Barnsley, M. F. Fractals Everywhere [Текст] / M. F. Barnsley. – Dover Publications, 2012. – 560 p. – ISBN 978-0486488707.

13 Falconer, K. Fractal Geometry: Mathematical Foundations and Applications [Текст] / K. Falconer. – John Wiley & Sons, 2003. – 368 p. – ISBN 978-0470848623.

14 Peng, C.-K. Mosaic organization of DNA nucleotides [Текст] / C.-K. Peng, S. V. Buldyrev, S. Havlin [et al.] // Physical Review E. – 1994. – Vol. 49, N 2. – P. 1685–1689. – DOI: 10.1103/PhysRevE.49.1685 (дата звернення: 08.01.2026).

15 Beran, J. Statistics for Long-Memory Processes [Текст] / J. Beran. – Chapman and Hall, 1994. – 315 p. – ISBN 978-0412049019.

16 Evertsz, C. J. G. Multifractal measures [Текст] / C. J. G. Evertsz, B. B. Mandelbrot // Chaos and Fractals. – Springer, 1992. – P. 921–953. – DOI: 10.1007/978-1-4612-2806-6_27 (дата звернення: 08.01.2026).

17 Stanley, H. E. Scale invariance and universality: organizing principles in complex systems [Текст] / H. E. Stanley, L. A. N. Amaral, P. Gopikrishnan [et al.] // Physica A: Statistical Mechanics and its Applications. – 2000. – Vol. 281, N 1–4. – P. 60–68. – DOI: 10.1016/S0378-4371(00)00020-8 (дата звернення: 08.01.2026).

18 Adam, M. C. Fractal Dimensions in Financial Time Series [Текст] / M. C. Adam // Physica A: Statistical Mechanics and its Applications. – 2002. – Vol. 314, N 1–4. – P. 756–760. – DOI: 10.1016/S0378-4371(02)01069-7 (дата звернення: 08.01.2026).

19 Ausloos, M. Statistical physics in foreign exchange trading [Текст] / M. Ausloos // The European Physical Journal B. – 2000. – Vol. 20, N 1. – P. 3–6. – DOI: 10.1007/s100510070266 (дата звернення: 08.01.2026).

20 Di Matteo, T. Multi-scaling in finance [Текст] / T. Di Matteo // Quantitative Finance. – 2007. – Vol. 7, N 1. – P. 21–36. – DOI: 10.1080/14697680600999000 (дата звернення: 08.01.2026).

21 Grau-Carles, P. Empirical evidence of long-range correlations in stock returns [Текст] / P. Grau-Carles // Physica A: Statistical Mechanics and its Applications. – 2000. – Vol. 287, N 3–4. – P. 396–404. – DOI: 10.1016/S0378-4371(00)00382-0 (дата звернення: 08.01.2026).

22 Lo, A. W. Long-Term Memory in Stock Market Prices [Текст] / A. W. Lo // Econometrica. – 1991. – Vol. 59, N 5. – P. 1279–1313. – DOI: 10.2307/2938368 (дата звернення: 08.01.2026).

23 Podobnik, B. Detrended Cross-Correlation Analysis: A New Method for Analyzing Two Nonstationary Time Series [Текст] / B. Podobnik, H. E. Stanley // Physical Review Letters. – 2008. – Vol. 100, N 8. – P. 084102. – DOI: 10.1103/PhysRevLett.100.084102 (дата звернення: 08.01.2026).

24 Shang, P. Detecting long-range correlations with detrended fluctuation analysis [Текст] / P. Shang, Y. Lu, S. Kamae // Physica A: Statistical Mechanics and its Applications. – 2005. – Vol. 346, N 3–4. – P. 382–392. – DOI: 10.1016/j.physa.2004.08.019 (дата звернення: 08.01.2026).

25 Xu, N. Minimizing the effect of trends on detrended fluctuation analysis of long-range correlated noise [Текст] / N. Xu, P. Shang, S. Kamae // Chaos, Solitons & Fractals. – 2009. – Vol. 41, N 4. – P. 1706–1715. – DOI: 10.1016/j.chaos.2008.07.016 (дата звернення: 08.01.2026).

26 Smith, A. Mastering C# and .NET Framework [Текст] / A. Smith. – Packt Publishing, 2019. – 450 p. – ISBN 978-1789956368.

27 Troelsen, A. Pro C# 9 with .NET 5 [Текст] / A. Troelsen, P. Japikse. – Apress, 2021. – 1350 p. – ISBN 978-1484269381.

28 Sells, C. Programming WPF [Текст] / C. Sells, I. Griffiths. – O'Reilly Media, 2014. – 800 p. – ISBN 978-0596510374.

29 Nathan, A. Windows Presentation Foundation Unleashed [Текст] / A. Nathan. – Sams Publishing, 2018. – 650 p. – ISBN 978-0672328916.

30 Griffiths, I. Programming C# 8.0 [Текст] / I. Griffiths. – O'Reilly Media, 2020. – 800 p. – ISBN 978-1492056812.

ДОДАТОК А

Технічне завдання

ЗАТВЕРДЖУЮ

Перший проректор
Українського державного
університету науки і
технологій

_____Анатолій РАДКЕВИЧ

ПРОГРАМА ДЛЯ ЕКСПЕРИМЕНТАЛЬНИХ ОБЧИСЛЮВАЛЬНИХ ДОСЛІДЖЕНЬ ВИЗНАЧЕННЯ САМОПОДІБНОСТІ ЧАСОВИХ РЯДІВ

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1533-01-ЛЗ

Завідувач кафедри КІТ

_____Вадим ГОРЯЧКІН

Керівник розробки

_____Віктор ШИНКАРЕНКО

Виконавець

_____Данило УЛЬЯНЧЕНКО

Нормоконтролер

_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.1533-01

ПРОГРАМА ДЛЯ ЕКСПЕРИМЕНТАЛЬНИХ ОБЧИСЛЮВАЛЬНИХ ДОСЛІДЖЕНЬ ВИЗНАЧЕННЯ САМОПОДІБНОСТІ ЧАСОВИХ РЯДІВ

Технічне завдання

Листів 15

2026

ЗМІСТ

1 ВВЕДЕННЯ

Програма для експериментальних обчислювальних досліджень самоподібності часових рядів є інструментом для аналізу даних, який відіграє ключову роль у вивченні складних систем, дозволяючи виявляти повторювані структури на різних масштабах та обчислювати фрактальну розмірність. Сьогодні дослідники стикаються з необхідністю обробки великих обсягів часових рядів з фінансових, біомедичних чи кліматичних джерел.

Метою даної роботи є розробка технічного завдання на створення програмного продукту, який автоматизує аналіз самоподібності, забезпечуючи точні обчислення та візуалізацію для наукових досліджень.

Причина виникнення необхідності розробки:

Програма дозволить оптимізувати процес, зменшити похибки та прискорити обчислення фрактальної розмірності, що підвищить ефективність аналізу даних.

Область застосування:

Програма призначена для завантаження даних з Excel, пошуку схожих послідовностей з толерантністю, обчислення фрактальної розмірності через лінійну регресію (log M проти log δ) та візуалізації результатів за допомогою графіків. Розробка базується на архітектурі MVVM з використанням C#, .NET, XAML та OxyPlot, з підтримкою асинхронних обчислень для ефективності.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Основою для розробки є наказ проректора Українського державного університету науки і технології Радкевич А.В. «Про затвердження тем та призначення керівників дипломних проектів» №1401 ст від 02.10.2025 року.

Тема проекту: “ Експериментальні обчислювальні дослідження самоподібності часових рядів”.

Керівник дипломного проекту: Шинкаренко Віктор Іванович.

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Програма для експериментальних обчислювальних досліджень самоподібності часових рядів є інструментом для аналізу даних, який відіграє важливу роль у вивченні складних систем, дозволяючи виявляти повторювані структури на різних масштабах, обчислювати фрактальну розмірність та візуалізувати результати. Сьогодні дослідники стикаються з необхідністю обробки великих обсягів часових рядів з фінансових, біомедичних чи кліматичних джерел, де традиційні методи аналізу не забезпечують достатньої ефективності та точності.

Функціональне призначення:

Програма призначена для автоматизації аналізу самоподібності часових рядів. Основні функції включають завантаження даних з Excel, пошук схожих послідовностей з толерантністю, обчислення фрактальної розмірності через лінійну регресію, а також візуалізацію результатів за допомогою графіків. Система також забезпечує збереження даних у JSON для подальшого використання.

Експлуатаційне призначення:

Програма призначена для постійного використання в наукових лабораторіях, освітніх закладах та дослідницьких проєктах. Вона дозволяє проводити експериментальні обчислення в реальному часі, аналізувати дані з різних джерел, порівнювати результати за діапазоном параметрів та генерувати звіти. Система розрахована на безперервну роботу та має бути доступною для користувачів з базовими навичками програмування.

Програма призначена для підвищення ефективності наукових досліджень, покращення точності аналізу даних та забезпечення прозорого та зручного управління процесами обчислення самоподібності.

Конкретні призначення:

- аналіз даних: автоматичний пошук схожих послідовностей у часових рядах з урахуванням толерантності, групування за довжиною та обчислення статистики (кількість основних/схожих послідовностей);
- обчислення параметрів: визначення фрактальної розмірності через лінійну регресію на логарифмічних даних (log M від log δ), з підтримкою діапазону толерантностей для порівняльного аналізу;
- візуалізація: побудова графіків часових рядів, позначених послідовностями, та діаграм фрактальної розмірності для інтуїтивного інтерпретації результатів;
- обробка та зберігання: завантаження з Excel, збереження результатів у JSON (повний/скорочений формат), забезпечення сумісності з науковими інструментами.

Програма призначається для використання студентами, дослідниками та фахівцями ІТ у лабораторних умовах, з акцентом на експериментальні обчислення. Розробка забезпечує ефективність (асинхронна обробка) та доступність (графічний інтерфейс MVVM), сприяючи переходу від ручного аналізу до автоматизованого.

4 ВИМОГИ ДО ПРОГРАМИ

4.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати повний цикл аналізу часових рядів на самоподібність, включаючи завантаження, обробку, обчислення та візуалізацію даних.

Основні функції:

- завантаження даних: підтримка імпорту з файлів Excel (.xlsx, .xls) у форматі двох стовпців (X – час/індекс, Y – значення), з автоматичним парсингом до 10 000 точок. Обробка помилок (порожні комірки, NaN, нечислові значення) з повідомленнями користувачу;
- пошук послідовностей: асинхронний алгоритм для виявлення схожих сегментів з толерантністю (від 0% до 100%, крок 0.01%). Групування за довжиною (мін. 1, макс. повна довжина ряду), з паралельним обчисленням для прискорення (8 потоків);
- обчислення фрактальної розмірності: лінійна регресія на логарифмічних даних (log M – кількість послідовностей, log δ – довжина), з точністю 0.0001. Підтримка діапазону толерантностей для порівняння (мін./макс./крок);
- візуалізація: графіки часових рядів (лінії з позначками послідовностей), scatter-графік log M від log δ з лінією регресії. Інтерактивність: масштабування, вибір елементів для деталізації;
- збереження результатів: експорт у JSON (повний – з точками послідовностей, скорочений – статистика), з можливістю вибору шляху. Автозбереження опціонально.

Програма повинна обробляти дані в реальному часі, з прогресом (0–100%) та оцінкою часу.

4.2 Вимоги до надійності

Програма повинна бути стійкою до помилок і безвідмовною в роботі:

- обробка винятків: автоматичне виявлення та повідомлення про помилки (IOException для файлів, ArgumentException для параметрів) через MessageBox, без аварійного завершення;
- валідація даних: перевірка на валідність (NaN, порожні файли, толерантність >0), з поверненням порожніх результатів або попередженнями;
- стабільність: асинхронна обробка з CancellationToken для безпечного переривання;
- резервування: автозбереження результатів, якщо активовано;
- безпека: захист від переповнення пам'яті (обмеження колекцій до 50 000 елементів), відсутність SQL-запитів (файлова робота).

4.3 Умови експлуатації

Програма призначена для лабораторного та наукового використання:

- середовище: ОС Windows 10/11 (64-біт), .NET Framework 4.8 або .NET 6+. Процесор Intel Core i5+ (2 ГГц), RAM 4 ГБ+, HDD/SSD 100 МБ вільного місця.
- робочі умови: температура 18–25°C, вологість 40–60%, освітлення 300 лк. Робота на ПК без спеціального обладнання.
- режим роботи: інтерактивний (графічний інтерфейс), з можливістю фонового обчислення. Час реакції на команди – <1 с, повний аналіз (1000 точок) – <30 с.
- обмеження: Не для реального часу, (не >10 000 точок без оптимізації)

4.4 Вимоги до складу і параметрів технічних засобів

Програма складається з модулів, що забезпечують повний функціонал:

Складові компоненти:

- модуль даних (ISSTS.Data): DataExcel (точки), MatchingSequenceGroup (групи послідовностей), LoadData/SaveData (ввід/вивід).
- модуль обчислень (ISSTS.Processing): Calculation (пошук, регресія).
- модуль інтерфейсу (ISSTS.Models): MainWindow (головне), ToleranceRangeWindow (налаштування), FractalDimensionRangeWindow (діапазон).
- модуль MVVM (ISSTS.Bases): ViewModelBase, RelayCommand.

Параметри:

- обсяг пам'яті: 1–5 МБ (залежно від даних).
- час виконання: пошук – O(n^2) для n точок, оптимізований паралелізмом.
- залежності: OxyPlot (графіки), EPPlus (Excel), Newtonsoft.Json (серіялізація).
- технічні засоби: ПК з Visual Studio 2022 для розробки, без спеціального апаратного забезпечення.

4.5 Вимоги до інформаційної і програмної сумісності

Інформаційна сумісність: Вхід – Excel (CSV-подібний), вихід – JSON (стандарт RFC 8259). Підтримка UTF-8 для даних.

Програмна сумісність: .NET Standard 2.0, сумісність з Windows/.NET Core. API для розширення (інтерфейси для Calculation). Інтеграція з OxyPlot (версія 2.1+).

Стандарти: Відповідність ГОСТ 19.101-77 (ЄСПД), UML для діаграм. Без конфліктів з антивірусами.

4.6 Вимоги до транспортування і зберігання

Транспортування: Портативний exe-файл (self-contained), розмір <10 МБ. Розповсюдження через архів ZIP, без встановлення.

Зберігання: Файли результатів – JSON (<1 МБ), Excel (<5 МБ). Зберігання на HDD/SSD, термін – необмежений (без терміну придатності). Захист від пошкоджень: резервні копії в JSON.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Програмна документація повинна відповідати стандартам ЄСПД (ГОСТ 19.101-77, ГОСТ 19.401-79) та бути повною для забезпечення супроводу, модернізації та експлуатації програми. Документація готується у форматі PDF або Word, з нумерацією сторінок, змістом та індексом. Обсяг – не менше 50 сторінок, шрифт Times New Roman 14 pt, інтервал 1.5.

Склад документації:

- технічне завдання (ТУ): описано в цьому документі. Включає введення, підставу, призначення, вимоги, стадії розробки;
- пояснювальна записка: детальний опис реалізації (архітектура MVVM, алгоритми Calculation, інтерфейс MainWindow). Включає діаграми класів (UML), схеми взаємодії модулів (ASCII/PlantUML), результати тестування (таблиці з розмірностями для тестових даних);
- текст програми: повний код джерел (C#, XAML) з коментарями. Описи файлів за групами (ініціалізація, моделі, обробка, ViewModel, UI, конвертери). Приклади: Calculation.cs – алгоритм регресії; MainViewModel.cs – команди пошуку;
- керівництво користувача: покрокова інструкція (завантаження Excel, налаштування толерантності, запуск аналізу, збереження JSON). Скріншоти інтерфейсу (MainWindow, ToleranceRangeWindow), таблиці помилок;
- керівництво з встановлення та експлуатації: Вимоги до ПК (Windows 10+, .NET 6+), інструкція розгортання (exe-файл), тестування (unit-тести для SearchAsync);
- схеми та діаграми: діаграма класів (PlantUML), схема алгоритму (пошук – регресія), ER-діаграма даних (DataExcel -> MatchingSequenceGroup).

Вимоги до оформлення:

- Мова: Українська, з термінами (самоподібність, фрактальна розмірність);
- Захист: Версійний контроль (Git), підпис виконавця та керівника;

- Оновлення: Документація оновлюється при змінах (версія 1.0 від 26.09.2026);
- Документація забезпечує повне розуміння ПЗ для розробників та користувачів.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Розробка програмного забезпечення "Програма для експериментальних обчислювальних досліджень самоподібності часових рядів" проводиться у три стадії відповідно до стандартів ЄСПД (ГОСТ 19.201-78). Загальний термін – 12 тижнів (05.09.2026 – 26.09.2026), обсяг – 133 години. Кожен етап включає контрольні точки з перевіркою керівником.

В табл. 6.1 приведені стадії та етапи розробки.

Таблиця **Ошибка! Источник ссылки не найден.**.1 – Стадії та етапи розробки

Стадія	Зміст робіт	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, вироблення критеріїв розробки. Попередній вибір методів рішення. Визначення вимог до технічних засобів. Узгодження та затвердження технічного завдання.	01.09.25 - 22.09.25
Робочий проект	Розробка програми та її відладка.	23.09.25 - 24.11.25
	Тестування програми.	25.11.25 - 22.12.25
	Розробка, узгодження та затвердження програмної документації.	23.12.25 - 31.12.25

6.1 Стадія 1 Аналіз і проектування (тиждень 1–2, 20 годин)

- Етап 1.1: Вивчення вимог і постановка задачі (аналіз літератури з фрактальної геометрії, визначення алгоритмів).
- Етап 1.2: Проектування архітектури (MVVM, діаграми класів UML, схема взаємодії модулів).
- Результат: Технічне завдання, діаграма класів, прототип інтерфейсу. Контроль: затвердження керівником.

6.2 Стадія 2 Реалізація (тиждень 3–9, 80 годин)

- Етап 2.1: Розробка модулів даних і обробки (DataExcel, Calculation, LoadData/SaveData).
- Етап 2.2: Створення інтерфейсу (MainWindow, ToleranceRangeWindow, OxyPlot-графіки).
- Етап 2.3: Інтеграція та тестування (асинхронні обчислення, конвертери).
- Результат: Робочий прототип. Контроль: проміжний звіт з демо.

6.3 Стадія 3 Тестування та документація (тиждень 10–12, 33 години)

- Етап 3.1: Комплексне тестування (на синтетичних/реальних даних, перевірка розмірності).
- Етап 3.2: Підготовка документації (пояснювальна записка, керівництво користувача).
- Результат: Фінальний реліз, повна документація. Контроль: приймання з актом.
- Ресурси: Visual Studio 2022, .NET 6. Ризики: затримки обчислень – пом'якшено паралелізмом.

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Контроль за виконанням роботи здійснює керівник дипломного проекту.

Приймання здійснюється призначеною екзаменаційною комісією.

ДОДАТОК Б

Текст програми

ЗАТВЕРДЖУЮ

Перший проректор
Українського державного
університету науки і
технологій

_____Анатолій РАДКЕВИЧ

ПРОГРАМА ДЛЯ ЕКСПЕРИМЕНТАЛЬНИХ ОБЧИСЛЮВАЛЬНИХ ДОСЛІДЖЕНЬ ВИЗНАЧЕННЯ САМОПОДІБНОСТІ ЧАСОВИХ РЯДІВ

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1533-12-01

Завідувач кафедри КІТ

_____Вадим ГОРЯЧКІН

Керівник розробки

_____Віктор ШИНКАРЕНКО

Виконавець

_____Данило УЛЬЯНЧЕНКО

Нормоконтролер

_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.1533-12-01

ПРОГРАМА ДЛЯ ЕКСПЕРИМЕНТАЛЬНИХ ОБЧИСЛЮВАЛЬНИХ ДОСЛІДЖЕНЬ ВИЗНАЧЕННЯ САМОПОДІБНОСТІ ЧАСОВИХ РЯДІВ

Текст програми

Листів 98

2026

АНОТАЦІЯ

Документ 44165850.1533-01 12 01 «Експериментальні обчислювальні дослідження самоподібності часових рядів. Текст програми» входить до складу програмної документації на систему, що реалізовує функції аналізу часових рядів, пошуку схожих послідовностей, обчислення фрактальної розмірності та візуалізації результатів.

У даному документі представлений текст програми. Програма написана на мові C# з використанням фреймворку .NET та бібліотек OxyPlot, Newtonsoft.Json. Обсяг пам'яті, що займає програма, складає близько 100 Мб (залежно від конфігурації). Конфігурація комп'ютера стандартна. Програма функціонує в середовищі MS Windows 10 або новіших версій.

ЗМІСТ

1 КОРОТКИЙ ОПИС ФАЙЛІВ

1 Ініціалізація та конфігурація

1.1. App.cs: Основний клас додатку, ініціалізує програму та налаштування.

1.2. App.xaml: XAML-розмітка для запуску додатку, визначає ресурси та стилі.

1.3. AssemblyInfo.cs: Містить метадані збірки (версія, автор, опис).

1.4. MainSettings.Designer.cs: Генерований код для налаштувань програми.

1.5. MainSettings.settings.cs: Файл налаштувань (шляхи, параметри за замовчуванням).

2 Моделі даних

2.1. DataExcel.cs: Модель даних для представлення точок часового ряду (X, Y).

2.2. FileJson.cs: Моделі даних для серіалізації/десеріалізації у JSON (JsonData, JsonSequence).

3 Обробка даних

3.1. Calculation.cs: Реалізує алгоритми пошуку послідовностей та обчислення фрактальної розмірності.

3.2. LoadData.cs: Клас для завантаження даних з Excel та JSON.

3.3. SaveData.cs: Клас для збереження результатів у JSON.

4 Моделі виду (ViewModel)

4.1. FractalDimensionPlotViewModel.cs: Модель виду для графіка фрактальної розмірності.

4.2. FractalDimensionRangeViewModel.cs: Модель виду для вікна діапазону розмірностей.

4.3. MainViewModel.cs: Центральна модель виду, координує логіку та команди.

4.4. ToleranceRangeViewModel.cs: Модель виду для вікна налаштування толерантності.

4.5. ViewModelBase.cs: Базовий клас для моделей виду з підтримкою INotifyPropertyChanged.

4.6. RelayCommand.cs: Клас для реалізації команд у MVVM.

5 Інтерфейс користувача (XAML та код-behind)

5.1. MainWindow.cs: Код-behind для головного вікна, обробка подій.

5.2. MainWindow.xaml: XAML-розмітка головного вікна з інтерфейсом.

5.3. FractalDimensionPlotWindow.xaml: XAML-розмітка для вікна графіка розмірності.

5.4. FractalDimensionRangeWindow.xaml: XAML-розмітка для вікна налаштування діапазону розмірностей.

5.5. ToleranceRangeWindow.xaml.cs: XAML-розмітка для вікна діапазону толерантності.

6 Стилі та ресурси UI

6.1. Brushes.xaml: Визначає набір пензлів (кольорів) для UI-елементів.

6.2. CheckBox.xaml: Стилі для чекбоксів у меню та інтерфейсі.

6.3. Colors.xaml: Визначає кольорові ресурси для темного/світлого режимів.

6.4. Main.xaml: Основні стилі для головного вікна (кнопки, панелі).

6.5. MenuItem.xaml: Стилі для елементів меню в панелі інструментів.

6.6. ScrollBar.xaml: Стилі для прокрутки в списках.

6.7. Separator.xaml: Стилі для роздільників у меню.

6.8. ToollBar.xaml: Стилі для панелі інструментів.

7 Конвертери

7.1. BooleanToVisibilityConverter.cs: Конвертер для прив’язки булевих значень до видимості елементів.

7.2. BoolToIndexConverter.cs: Конвертер для перетворення булевих значень у індекси.

7.3. DataConverter.cs: Загальні конвертери для обробки даних у прив’язках.

7.4. DoubleToBooleanConverter.cs: Конвертер для перетворення чисел у булеві значення.

7.5. DoubleToVisibilityConverter.cs: Конвертер для прив’язки числових значень до видимості.

7.6. EndIndexConverter.cs: Конвертер для обчислення кінцевого індексу послідовності.

7.7. SequenceLengthConverter.cs: Конвертер для форматування довжини послідовностей.

7.8. SimilarSequenceIndexToGroupIdConverter.cs: Конвертер для відображення індексів схожих послідовностей.

7.9. StringToFloatConverter.cs: Конвертер для перетворення рядків у числа з плаваючою комою.

7.10. SubtractConverter.cs: Конвертер для обчислення різниці значень.

7.11. TimeSpanToStringConverter.cs: Конвертер для форматування часу (TimeSpan) у рядок.

8 Додаткові поведінки та допоміжні класи

8.1. Graf.cs: Клас стилю графіків під різні потреби.

8.2. SelectedItemsBehavior.cs: Поведінка для обробки вибраних елементів у ListView.

8.3. ViewPlot.cs: Клас для додавання рядів і точок на графік.

9 Детальний розбір взаємодій

1. UI (Інтерфейс користувача, View):

Файли: Main.xaml.cs, CheckBox.xaml.cs, ToleranceRangeWindow.xaml.cs, FractalDimensionPlotWindow.xaml.cs, FractalDimensionRangeWindow.xaml.cs.

Функції: Введення параметрів (толерантність, крок, діапазон), відображення графіків (OxyPlot), кнопки (пошук, збереження, завантаження). Стилі кнопок (ToolBarButtonStyle) забезпечують реакцію на наведення миші.

Взаємодії: UI <-> ViewModel через bindings та команди (RelayCommand з RelayCommand.cs). Наприклад, прокручування мишкою в TextBox (ToleranceRangeWindow.xaml.cs) оновлює значення в ViewModel. Якщо дані завантажено з Excel/JSON через LoadData, UI відображає їх у списках (ObservableCollection).

Потік: Користувач натискає кнопку завантаження/пошуку -> викликає команду в ViewModel -> запускає LoadData або Calculation.

2. ViewModel (Контролер):

Файли: ViewModelBase.cs, MainViewModel.

Функції: Керує станом (IsDataListBoxVisible, IsAutoSaveEnabled з MainSettings.settings.cs), оброблює події (наприклад, MouseWheel в ToleranceRangeWindow).

Взаємодії: ViewModel -> Calculation для запуску SearchAsync (з параметрами tolerance, minLength, maxLength); ViewModel <-> Data для оновлення колекцій (MatchingSequenceGroup, SequenceLengthGroup); ViewModel -> SaveData/LoadData для вводу/виводу (завантаження з Excel/JSON, збереження результатів). Якщо активовано AutoSave, ViewModel -> SaveData для автоматичного збереження.

Потік: Отримує дані з UI -> передає в LoadData для завантаження; -> Calculation для обчислення; -> SaveData для експорту; -> оновлює UI через OnPropertyChanged.

3. Calculation (Обчислювальний модуль):

Файл: Calculation.cs.

Функції: Асинхронний пошук послідовностей (SearchAsync: розділяє довжини на 8 груп для паралельних задач, використовує Channel для прогресу). Обчислення фрактальної розмірності (LinearRegression: sumX, sumY, sumXY для нахилу).

Взаємодії: Calculation <-> Data (вхід: ObservableCollection<dataexcel>; вихід: List<matchingsequencegroup>, List<sequencelengthgroup>). Обробляє токен скасування (CancellationToken) для зупинки. Прогрес: IProgress<double> для UI-онов-
лення.</double></sequencelengthgroup></matchingsequencegroup></dataexcel>

Потік: Отримує точки ряду (завантажені через LoadData) -> шукає схожі сегменти з толерантністю -> групує -> обчислює dimension = -slope -> повертає в ViewModel для збереження через SaveData.

4. Data (Моделі даних):

Файл: DataExcel.cs, FileJson.cs.

Функції: Структури (DataExcel: X/Y точки; MatchingSequenceGroup: SequenceIndex, SimilarSequenceIndices; FractalDimensionResult: TolerancePercentage, LogM/LogDelta).

Взаємодії: Data <-> Calculation для обробки; Data <-> Converters для конвертації (DataPoint <-> DataExcel); Data <-> SaveData/LoadData для

серіалізації/десеріалізації (DTO для уникнення циклів, перевірка на помилки як NaN).

Потік: Зберігає результати пошуку/завантаження -> використовується для візуалізації.

5. SaveData (Модуль збереження):

Файли: SaveData.cs, LoadData.cs.

Функції: SaveData – збереження в JSON (SaveToJson: повні дані з SequenceGroups; SaveFractalDimensionRangeToJson: діапазон фрактальних розмірностей з опцією повного/скороченого режиму, використовує Newtonsoft.Json з ReferenceLoopHandling.Ignore). LoadData – завантаження з Excel (LoadDataFromExcel: EPPlus для читання аркушів, перевірка на NaN/порожні комірки, TryParse для чисел); з JSON (LoadFromJson: десеріалізація MatchingSequenceGroup, повернення FractalDimension); з Fractal JSON (LoadFractalDimensionRangeFromJson: десеріалізація результатів). Використовує OpenFileDialog для вибору файлів, обробляє IOException.

Взаємодії: SaveData/LoadData <-> Data (серіялізація/десеріалізація результатів); SaveData/LoadData -> Зовнішні файли (DefaultFilePath, SelectExcelFile/SelectJsonFile, MainSettings для збереження шляхів).

Потік: ViewModel викликає LoadData для імпорту (наприклад, з Excel -> List<dataexcel> -> Data); Calculation генерує дані -> ViewModel -> SaveData для експорту в JSON (повернення успіх/помилка з MessageBox).

6. Converters (Допоміжні конвертери):

Файли: DataConverter.cs, StringToFloatConverter.cs, TimeSpanToStringConverter.cs тощо.

Функції: Конвертація для UI (StringToFloat: парсинг %; DoubleToVisibility: видимість елементів; EndIndexConverter: обчислення кінця послідовності).

Взаємодії: Converters <-> UI/ViewModel через IValueConverter; Converters <-> Data для форматування (наприклад, LogM/LogDelta для графіків).

Потік: Оновлює UI в реальному часі (наприклад, прогрес -> видимість кнопок; завантажені дані -> конвертація для відображення).

2 РОЗШИРЕНИЙ ОПИС ТЕКСТУ ПРОГРАМИ

1.4 Опис програми файлу MainSettings.settings.cs

Цей файл визначає налаштування програми (settings) для збереження шляхів файлів та флагів, використовуючи .NET Properties.Settings. Належить до простору імен ISSTS.Data (GeneratedClassNamespace), генерується Visual Studio для персистентності даних.

Структура та ключові компоненти:

- XML-файл з <settings> для властивостей: ExcelFilePath (string, User), JsonFilePath (string, User), IsDataListBoxVisible (bool, User, default True), IsAutoSaveEnabled (bool, User, default False), FractalJsonFilePath (string, User).
- властивості: зберігають шляхи (Excel/JSON/Fractal) та флаги (видимість списку, автозбереження);
- взаємодії:використовується в LoadData/SaveData (MainSettings.Default для шляхів), ViewModel (для флагів).
- достовірність:
    1) Scope="User" для збереження між сесіями;
    2) автоматична генерація класу MainSettings.

2.1 Опис програми файлу DataExcel.cs

Файл визначає моделі даних для часових рядів та результатів. Належить до простору імен ISSTS.Data, є основою для Calculation, SaveData/LoadData.

Структура та ключові компоненти:

- включає структури DataExcel (точки), SequenceLengthGroup (групи за довжиною), MatchingSequenceGroup (групи послідовностей), FractalDimensionResult (результати розмірності).
- dataExcel: Структура з double X/Y, конструктор. Представляє точки ряду.
- sequenceLengthGroup: Групує послідовності за довжиною; властивості Length, Sequences, лічильники (Main/Similar/TotalSequencesCount).

- matchingSequenceGroup: Група схожих послідовностей; властивості GroupId, SequenceIndex, Sequence (List<dataexcel>), IsSelected, SimilarSequenceIndices, SequenceLength, ToleranceUsed.
- fractalDimensionResult:
- результати обчислень;
- властивості TolerancePercentage, Dimension, LogM/LogDelta, MaxLength, SequenceGroups.
- взаємодії: Використовується в Calculation (для групування), SaveData/LoadData (серіялізація), Converters (конвертація).
- достовірність: Порожні списки в конструкторах для ініціалізації.

2.2 Опис програми файлу FileJson.cs

Цей файл визначає допоміжні класи для серіалізації/десеріалізації даних у JSON-форматі, використовувані в SaveData та LoadData. Він належить до простору імен ISSTS.Data і забезпечує коректне перетворення структур (наприклад, MatchingSequenceGroup) у JSON без циклів. Файл є частиною моделі даних, що полегшує збереження/завантаження результатів аналізу самоподібності.

Структура та ключові компоненти:

- включає класи JsonData (основна структура для послідовностей), JsonSequence (група послідовностей), JsonPoint (точки), FractalDimensionRangeSaveData (для діапазону розмірностей). Ці класи є DTO (Data Transfer Objects) для уникнення прямих посилань на основні моделі.
- jsonData: Властивості ExcelFilePath (шлях до Excel), FractalDimension (розмірність), Sequences (список JsonSequence). Використовується для повного збереження результатів пошуку.
- jsonSequence: Властивості GroupId, SequenceIndex, IsSelected, SimilarSequenceIndices (список індексів), Sequence (список JsonPoint). Описує групу схожих послідовностей для JSON.
- jsonPoint: Проста структура з double X/Y для точок ряду.

- fractalDimensionRangeSaveData: Властивості ExcelFilePath, JsonFilePath, Results (список FractalDimensionResult). Використовується для збереження діапазону обчислень фрактальної розмірності.
- взаємодії: Інтегрується з SaveData (для серіалізації в SaveToJson/SaveFractalDimensionRangeToJson) та LoadData (для десеріалізації в LoadFromJson/LoadFractalDimensionRangeFromJson).
- достовірність: Прості властивості без логіки, що забезпечує безпомилкову передачу даних.

3.1 Опис програми файлу Calculation.cs

Цей файл реалізує основну обчислювальну логіку програми: асинхронний пошук схожих послідовностей у часових рядах та обчислення фрактальної розмірності. Він використовує паралельні обчислення для оптимізації на великих даних, з підтримкою прогресу, оцінки часу та скасування. Файл належить до простору імен ISSTS.Processing і взаємодіє з DataExcel (для вхідних даних) та ViewModel (для повернення результатів).

Структура та ключові компоненти:

- клас Calculation містить два основних методи: SearchAsync (пошук послідовностей) та CalculateFractalDimension (обчислення розмірності).
- використовуються: System.Collections.Concurrent для паралелізму, System.Threading для асинхронності, System.Diagnostics для таймінгу.
- метод SearchAsync: Асинхронно шукає схожі послідовності. Вхід: ObservableCollection<dataexcel> (точки ряду), double tolerance (толерантність), IProgress<double> (прогрес), IProgress<timespan> (час), CancellationToken (скасування пошуку), int minLength/maxLength (довжини послідовностей). Розділяє довжини на 8 груп за модулем 8 для паралельних задач (Task.Run). Використовує Channel для буферизації прогресу, lock для синхронізації. Оцінка часу: на основі першого проходу. Повертає List<matchingsequencegroup> з групами послідовностей.

- метод CalculateFractalDimension: Обчислює фрактальну розмірність за лінійною регресією.
- вхід: ObservableCollection<sequencelengthgroup>. Генерує logM/logDelta, пропускає невалідні (M<=0, delta<=0). Використовує LinearRegression для нахилу (slope = (nsumXY - sumXsumY)/(n*sumXX - sumX^2)). Розмірність = -slope. Обробляє помилки (MessageBox для недостатніх даних).
- взаємодії: Інтегрується з SaveData (для збереження результатів), UI (через прогрес).
- достовірність: Перевірки на скасування, логування Debug.WriteLine для діагностики.

3.2 Опис програми файлу LoadData.cs

Файл реалізує завантаження даних з Excel/JSON, з перевірками на помилки та формат. Належить до простору імен ISSTS.Data, доповнює SaveData для циклу вводу/виводу, взаємодіє з DataExcel (моделі) та UI (діалоги).

Структура та ключові компоненти:

- клас LoadData містить методи для вибору файлів (SelectExcelFile, SelectJsonFile, SelectFractalJsonFile) та завантаження (LoadDataFromExcel, LoadFromJson, LoadFractalDimensionRange FromJson);
- використовуються: OfficeOpenXml (EPPlus) для Excel, Newtonsoft.Json для десеріалізації.
- методи вибору файлів: SelectExcelFile/SelectJsonFile/SelectFractalJsonFile – відкривають OpenFileDialog з фільтрами, повертають шлях або null. Перевантаження для кастомного заголовка.
- метод LoadDataFromExcel:
    1) читає Excel (перший аркуш, стовпці 1-2).
    2) вхід: шлях. Перевіряє на порожні комірки, NaN, парсинг double (TryParse).

3) обробляє IOException (файл зайнятий), загальні помилки (MessageBox).
4) повертає List<dataexcel>.

- метод LoadFromJson: Десеріалізує JSON у MatchingSequenceGroup.
  1) вхід: шлях.
  2) використовує FileJson.JsonData, повертає колекцію, Excel-шлях, fractalDimension.
  3) обробляє помилки (Debug.WriteLine).
- метод LoadFractalDimensionRangeFromJson: Десеріалізує Fractal JSON у FractalDimensionResult.
  1) вхід: шлях.
  2) повертає список результатів, шляхи файлів.
  3) обробляє помилки (Debug).
  4) взаємодії: Викликається з ViewModel для імпорту; інтегрується з Calculation (завантажені дані -> обчислення).
  5) достовірність: Ліцензія EPPlus.NonCommercial, перевірки на кінець даних (null комірки).

3.3 Опис програми файлу SaveData.cs

Файл реалізує збереження результатів у JSON-форматі, з підтримкою повного/скороченого режиму для оптимізації розміру файлів. Належить до простору імен ISSTS.Data, взаємодіє з DataExcel (моделі) та UI (діалоги збереження).

Структура та ключові компоненти:

- клас SaveData містить методи для вибору файлів (SelectSaveJsonFile, SelectSaveFractalJsonFile) та збереження (SaveToJson, SaveFractal DimensionRangeToJson). Використовуються: Newtonsoft.Json для серіалізації, Microsoft.Win32 для діалогів, System.IO для файлів.
- методи вибору файлів: SelectSaveJsonFile/SelectSaveFractalJsonFile – відкривають SaveFileDialog з фільтром JSON, пропонують ім'я за Excel-файлом (наприклад, "File_Fractal.json"). Повертають шлях або null.

- метод SaveToJson: Зберігає MatchingSequenceGroup у JSON. Вхід: колекція груп, шляхи файлів, fractalDimension. Серіялізує з DTO (FileJson.JsonData) для уникнення циклів. Обробляє помилки (MessageBox).
- метод SaveFractalDimensionRangeToJson:
- зберігає List<fractaldimensionresult>.
- підтримує isFullSave (з/без SequenceGroups).
- серіялізує з DTO, додає шляхи файлів.
- логування розміру файлу в Debug.
- взаємодії: Викликається з ViewModel після Calculation; використовує MainSettings для дефолтних шляхів.
- достовірність: Reference LoopHandling.Ignore для уникнення помилок, перевірки на виключення.

4.5 Опис програми файлу ViewModelBase.cs

Цей файл визначає базовий клас для ViewModel, реалізуючи INotifyPropertyChanged для двостороннього bindings в MVVM. Належить до простору імен ISSTS.Bases, є основою для всіх ViewModel (наприклад, MainViewModel, ToleranceRangeViewModel).

Структура та ключові компоненти:

- абстрактний клас ViewModelBase з подією PropertyChanged. Методи OnPropertyChanged (викликає подію) та SetProperty (змінює значення з перевіркою на рівність, оновлює залежні властивості).
- метод OnPropertyChanged: Використовує CallerMemberName для автоматичного отримання імені властивості.
- метод SetProperty: Дженерик для будь-якого типу T. Порівнює з поточним значенням (EqualityComparer), оновлює storage, викликає OnPropertyChanged для основної та додаткових властивостей (params string[]).

- взаємодії: Спадкується в усіх ViewModel (наприклад, для властивостей як TolerancePercentage). Інтегрується з UI (bindings) та Data (оновлення колекцій).
- достовірність: Захищає від непотрібних оновлень, зменшуючи навантаження на UI.

4.6 Опис програми файлу RelayCommand.cs

Файл реалізує шаблон команди для MVVM-архітектури, дозволяючи прив'язувати дії з UI до ViewModel без порушення розділення обов'язків. Належить до простору імен ISSTS.Bases, використовується для обробки подій (наприклад, кнопок пошуку/збереження). Це стандартний RelayCommand з підтримкою CanExecute.

Структура та ключові компоненти:

- клас RelayCommand реалізує інтерфейс ICommand. Властивості: execute (Action<object?>), canExecute (Func<object?, bool>?). Подія CanExecuteChanged прив'язана до CommandManager.RequerySuggested.
- конструктор: Приймає execute та опціональний canExecute. Ініціалізує делегати для виконання команди.
- метод CanExecute: Перевіряє, чи можна виконати команду (повертає true, якщо canExecute null або true).
- метод Execute: Виконує делегат execute з параметром.
- взаємодії: Використовується в ViewModel (наприклад, для команд пошуку в MainViewModel) та UI (bindings до кнопок у XAML). Інтегрується з SelectedItemsBehavior для управління вибором.
- достовірність: Автоматичне оновлення стану через CommandManager, що забезпечує реактивність UI.

5.1 Опис програми файлу MainWindow.xaml.cs

Цей файл визначає XAML-розмітку головного вікна програми, включаючи інтерфейс для графіків, списків даних, панелей налаштувань та прогрес-бару. Належить до простору імен ISSTS, є основним UI-компонентом для візуалізації часових рядів, послідовностей та результатів обчислень. Використовує OxyPlot для графіків, behaviors для вибору, converters для bindings.

Структура та ключові компоненти:

- Window з Title bindings, Resources (конвертери), DataContext (MainViewModel). Grid з колонками, DockPanel для панелей. ToolBarTray з меню (Файл, автозбереження, відкриття файлів), кнопки (пошук, збереження). ComboBox для режимів, TextBox для толерантності (з MouseDoubleClick). ListBox/TreeView для даних/послідовностей, Expander для групування. Oxy:PlotView для графіка, ProgressBar для прогресу.
- bindings та події: Bindings до ViewModel (PlotModel, SearchProgress, TolerancePercentage тощо). Тригери для видимості (Boolean ToVisibilityConverter). Behaviors: SelectedItemsBehavior для синхронізації вибору.
- взаємодії: Інтегрується з ViewModel (команди як CommandSearch, дані як MatchingSequencesGroups), Data (списки DataExcel/MatchingSequence Group), Calculation (результати для графіка).
- достовірність: VirtualizingPanel для оптимізації великих списків; ZIndex для шарів.

5.2 Опис програми файлу MainWindow.xaml

Файл реалізує C#-код-behind для MainWindow, обробляючи події UI (вибір, кліки, подвійні кліки). Належить до простору імен ISSTS, доповнює XAML для динамічної взаємодії (наприклад, оновлення графіка при виборі послідовностей). Це клас MainWindow з ініціалізацією та обробниками подій.

Структура та ключові компоненти:

- клас MainWindow спадкується від Window. Конструктор: InitializeComponent(), встановлює TextBoxTolerance.Text="0.1". Обробники: DataListBox_SelectionChanged (оновлення графіка для точки), SequenceLengthList_SelectionChanged (для довжини), Sequence _MouseDown (для базової+схожих), SimilarSequenceList_Selection Changed (для схожої), TextBoxTolerance_MouseDoubleClick (відкриття ToleranceRangeWindow).
- обробники подій: Викликають методи ViewModel (UpdateGraph, UpdateGraphForLength, UpdateGraphForSingleSequenceWithSimilar тощо). Очищення вибору (SelectedItem=null). Debug.WriteLine для логування.
- взаємодії: Інтегрується з ViewModel (DataContext як MainViewModel, виклики методів), UI (XAML елементи), Data (DataExcel/MatchingSequenceGroup).
- достовірність: Перевірки типів (is MainViewModel); проста логіка без складних обчислень.

5.3 Опис програми файлу FractalDimensionPlotWindow.xaml

Цей файл реалізує вікно для візуалізації графіка фрактальної розмірності (logM vs logDelta). Належить до простору імен ISSTS.Models, є частиною UI для відображення результатів обчислень. Це C#-код для вікна з графіком.

Структура та ключові компоненти:

- клас FractalDimensionPlotWindow спадкується від Window. Конструктор ініціалізує компоненти та встановлює DataContext на FractalDimensionPlotViewModel.
- конструктор: Викликає InitializeComponent(), встановлює DataContext для bindings графіка.
- взаємодії: Інтегрується з ViewModel (FractalDimensionPlotViewModel для даних графіка), OxyPlot (для візуалізації), Calculation (результати LogM/LogDelta).

- достовірність: Проста ініціалізація; залежність від ViewModel для динаміки.

5.4 Опис програми файлу FractalDimensionRangeWindow.xaml

Цей файл реалізує вікно для налаштування діапазону фрактальної розмірності, з інтеграцією ViewModel. Належить до простору імен ISSTS.Models, є частиною UI для вводу параметрів обчислень. Це C#-код для вікна, що забезпечує взаємодію з користувачем для вибору діапазону толерантності та кроку.

Структура та ключові компоненти:

- клас FractalDimensionRangeWindow спадкується від Window. Конструктор ініціалізує компоненти та встановлює DataContext на FractalDimensionRangeViewModel. Включає using для System.Windows, ViewModel.
- конструктор: Викликає InitializeComponent(), встановлює DataContext для bindings.
- взаємодії: Інтегрується з ViewModel (FractalDimensionRangeViewModel для властивостей), UI (XAML для елементів вводу), Calculation (параметри для обчислень).
- достовірність: Проста ініціалізація без логіки; залежність від ViewModel для даних.

5.5 Опис програми файлу ToleranceRangeWindow.xaml

Файл реалізує вікно для налаштування діапазону толерантності з підтримкою прокручування мишкою для зміни значень. Належить до простору імен ISSTS.Models, є частиною UI для вводу параметрів пошуку послідовностей. Це C#-код з обробкою подій для динамічного оновлення.

Структура та ключові компоненти:

- клас ToleranceRangeWindow спадкується від Window. Поля: _lastWheelEvent (для порогу прокручування), WheelThresholdMs (100 мс). Конструктор встановлює DataContext на ToleranceRangeViewModel. Метод TextBox_MouseWheel обробляє прокручування.

- метод TextBox_MouseWheel: Обчислює крок (0.5 для швидкого, 0.1 для повільного), оновлює властивості ViewModel (MinTolerancePercentage, MaxTolerancePercentage, Step) з обмеженнями (Clamp 0-100, Min 0.01 для Step).
- взаємодії: Інтегрується з ViewModel (ToleranceRangeViewModel для властивостей), UI (TextBox для вводу), Calculation (толерантність для SearchAsync).
- достовірність: Обробка e.Handled; DateTime для порогу; рефлексія для оновлення властивостей.

6.2 Опис програми файлу CheckBox.xaml

Файл визначає стилі для CheckBox та MenuItem у XAML для UI, з ефектами наведення та перевірки. Належить до ResourceDictionary, використовується для кастомізації вигляду чекбоксів у меню та списках (наприклад, для вибору режимів).

Структура та ключові компоненти:

- ResourceDictionary з стилем BaseCheckBoxStyle (TargetType CheckBox: фони, шаблон з Border/ContentPresenter, тригери для IsMouseOver/IsChecked). Стиль CheckBoxMenuItemStyle для MenuItem (Margin, HeaderTemplate з TextBlock).
- тригери: Зміна Background на HoverButtonColorBrush при наведенні/виборі.
- взаємодії: Інтегрується з UI (Main.xaml для кнопок), ViewModel (bindings для IsChecked).
- достовірність: DynamicResource для кольорів; Cursor="Hand" для інтерактивності.

7.1 Опис програми файлу BooleanToVisibilityConverter.cs

Файл реалізує конвертер для перетворення bool у Visibility для елементів UI (true – Visible, false – Collapsed). Належить до простору імен ISSTS.Converters, використовується для умовного показу/приховування (наприклад, панелей за флагами як IsSelected).

Структура та ключові компоненти:

- клас BooleanToVisibilityConverter реалізує IValueConverter. Методи Convert (bool -> Visibility) та ConvertBack (Visibility -> bool).
- метод Convert: Якщо value – true, повертає Visible; інакше Collapsed.
- метод ConvertBack: Якщо Visibility – Visible, повертає true; інакше false.
- взаємодії: Інтегрується з UI (bindings для Visibility), ViewModel/Data (флаги як IsSelected).
- достовірність: Двостороння конвертація для bindings; проста логіка без винятків.

7.3 Опис програми файлу DataConverter.cs

Файл реалізує конвертацію між моделями даних (DataExcel) та OxyPlot (DataPoint), для візуалізації графіків. Належить до простору імен ISSTS.Converters, використовується для підготовки даних до відображення в UI (наприклад, у FractalDimensionPlotWindow).

Структура та ключові компоненти:

- клас DataConverter з методами конвертації. Використовує OxyPlot та ISSTS.Data.
- метод ConvertToDataPoints: Перетворює ObservableCollection<dataexcel> у List<datapoint> (X/Y -> DataPoint).</datapoint></dataexcel>
- метод ConvertToDataExcel: Перетворює List<datapoint>/IEnumerable <datapoint> у List<dataexcel>/ObservableCollection<dataexcel> (зворотна конвертація).</dataexcel></datapoint><
- метод ConvertToDataExcel (перевантаження): Для List<dataexcel> -> IEnumerable<datapoint> (yield return для ефективності).

- взаємодії: Використовується в ViewModel для підготовки даних до графіків (OxyPlot) та Calculation (обробка результатів). Інтегрується з Converters для UI-bindings.
- достовірність: Прості мапінги без втрат даних, підтримка колекцій для великих наборів.

7.2 Опис програми файлу BoolToIndexConverter.cs

Файл реалізує двосторонній конвертер для перетворення bool у int (індекс, наприклад, для ComboBox вибору режиму). Належить до простору імен ISSTS.Converters, використовується для перемикання режимів (наприклад, "Всі послідовності" vs "Схожі за довжиною").

Структура та ключові компоненти:

- клас BoolToIndexConverter реалізує IValueConverter. Методи Convert (bool -> int) та ConvertBack (int -> bool).
- метод Convert: Якщо bool true, повертає 1; інакше 0 (або навпаки, залежно від контексту).
- метод ConvertBack: Якщо int == 1, повертає true; інакше false.
- взаємодії: Використовується в UI (bindings для SelectedIndex), ViewModel (флаги режимів).
- достовірність: Проста логіка; коментарі для пояснення.

7.4 Опис програми файлу DoubleToBooleanConverter.cs

Цей файл реалізує конвертер для перетворення double (наприклад, прогресу) у bool для активації/деактивації елементів UI. Належить до простору імен ISSTS.Converters, використовується для умовного керування (наприклад, кнопки активні при progress == 0).

Структура та ключові компоненти:

- клас DoubleToBooleanConverter реалізує IValueConverter. Методи Convert (double -> bool) та ConvertBack (не реалізований).
- метод Convert: Якщо value – double progress == 0, повертає true (активно); інверсія при параметрі "0". Інакше false. Логування в Debug.

- метод ConvertBack: Кидає NotImplementedException.
- взаємодії: Інтегрується з UI (bindings для IsEnabled), ViewModel (прогрес з Calculation).
- достовірність: Параметр для інверсії; дебаг для перевірки.

7.5 Опис програми файлу DoubleToVisibilityConverter.cs

Цей файл реалізує конвертер для перетворення double (наприклад, прогресу) у Visibility для елементів UI (показ/приховування). Належить до простору імен ISSTS.Converters, використовується для динамічного керування видимістю (наприклад, прогрес-бар >0 – видимий).

Структура та ключові компоненти:

- клас DoubleToVisibilityConverter реалізує IValueConverter. Методи Convert (double -> Visibility) та ConvertBack (не реалізований).
- метод Convert: Якщо value – double progress >0, повертає Visible (або Collapsed при параметрі "0" для інверсії). Інакше Collapsed.
- метод ConvertBack: Кидає NotImplementedException.
- взаємодії: Використовується в XAML (bindings для Visibility елементів), з ViewModel (прогрес з Calculation).
- достовірність: Параметр для інверсії, проста перевірка >0; логування Debug для діагностики.

7.6 Опис програми файлу EndIndexConverter.cs

Цей файл реалізує конвертер для обчислення кінцевого індексу послідовності в UI, на основі даних групи. Належить до простору імен ISSTS.Converters, використовується для відображення діапазону послідовностей (наприклад, "кінець: N"). Це IValueConverter для bindings, що полегшує візуалізацію MatchingSequenceGroup.

Структура та ключові компоненти:

- клас EndIndexConverter реалізує IValueConverter. Методи Convert (обчислення кінця) та ConvertBack (не реалізований).

- метод Convert: Якщо value – MatchingSequenceGroup з непорожньою Sequence, обчислює SequenceIndex + Sequence.Count - 1. Інакше повертає "Н/Д".
- метод ConvertBack: Кидає NotImplementedException.
- взаємодії: Інтегрується з UI (bindings для тексту в списках), Data (MatchingSequenceGroup).
- достовірність: Перевірка на наявність даних; проста арифметика без помилок.

7.7 Опис програми файлу SequenceLengthConverter.cs

Файл реалізує конвертер для форматування заголовків груп послідовностей у UI (наприклад, "Група N (Довжина: M)"). Належить до простору імен ISSTS.Converters, використовується для динамічного відображення інформації про MatchingSequenceGroup.

Структура та ключові компоненти:

- клас SequenceLengthConverter реалізує IValueConverter. Методи Convert (форматування рядка) та ConvertBack (не реалізований).
- метод Convert: Якщо value – int groupId та parameter – MatchingSequenceGroup, повертає $"Група {groupId} (Довжина: {group.Sequence.Count})". Інакше повертає value.
- метод ConvertBack: Кидає NotImplementedException.
- взаємодії: Використовується в UI (bindings для заголовків), Data (MatchingSequenceGroup).
- достовірність: Перевірка типів; шаблон рядка для чіткості.

7.8 Опис програми файлу SimilarSequenceIndexToGroupId Converter.cs

Цей файл реалізує конвертер для перетворення індексу схожої послідовності у GroupId з MainViewModel, для відображення в UI (наприклад, замість індексу показати ID групи). Належить до простору імен ISSTS.Converters, використовується для динамічного пошуку та форматування даних у списках послідовностей.

Структура та ключові компоненти:

- клас SimilarSequenceIndexToGroupIdConverter реалізує IValue Converter. Методи Convert (індекс -> GroupId) та ConvertBack (не реалізований).
- метод Convert: Якщо value – int sequenceIndex, шукає групу в MainViewModel.MatchingSequencesGroups за SequenceIndex і повертає GroupId як string. Інакше повертає sequenceIndex.ToString().
- метод ConvertBack: Кидає NotImplementedException.
- взаємодії: Інтегрується з UI (bindings для тексту), ViewModel (доступ до MatchingSequencesGroups через Application.Current.MainWindow. DataContext), Data (MatchingSequenceGroup).
- достовірність: Перевірка на наявність вікна та групи; FirstOrDefault для безпечного пошуку.

7.9 Опис програми файлу StringToFloatConverter.cs

Файл реалізує конвертер для двостороннього перетворення рядків у double (з форматуванням для UI, наприклад, толерантності як "10.00%"). Належить до простору імен ISSTS.Converters, використовується для TextBox у ToleranceRangeWindow для вводу/відображення відсотків з підтримкою регіональних налаштувань.

Структура та ключові компоненти:

- клас StringToFloatConverter реалізує IValueConverter. Методи Convert (double -> string) та ConvertBack (string -> double);
- метод Convert: якщо value – double, форматує як "F2" (два знаки після коми). Інакше повертає Binding.DoNothing;

- метод ConvertBack: обробляє порожні рядки (повертає 0.0), видаляє "%", парсить з NumberStyles.Any. Якщо помилка – Binding.DoNothing (не скидає значення);
- взаємодії:інтегрується з UI (bindings для TextBox), ViewModel (властивості як MinTolerancePercentage);
- достовірність: Debug.WriteLine для логування, TryParse для безпечного парсингу; підтримує CultureInfo.

7.10 Опис програми файлу SubtractConverter.cs

Файл реалізує конвертер для віднімання значення від double (наприклад, для коригування висоти елементів UI). Належить до простору імен ISSTS.Converters, використовується для динамічного розрахунку розмірів у XAML (наприклад, висота панелі мінус фіксоване значення):

Структура та ключові компоненти:

- клас SubtractConverter реалізує IValueConverter. Методи Convert (віднімання) та ConvertBack (не реалізований);
- метод Convert: Якщо value – double height та parameter – string subtractValue (парсинг double), повертає height - subtract. Інакше повертає value;
- метод ConvertBack: Кидає NotImplementedException.

Взаємодії: Використовується в UI (bindings для Height/Width), з параметром для фіксованого значення. Достовірність: TryParse для безпечного парсингу;

7.11 Опис програми файлу TimeSpanToStringConverter.cs

Цей файл реалізує конвертер для перетворення об'єктів TimeSpan у рядковий формат для відображення в UI, зокрема для оцінки часу обчислень (наприклад, у прогрес-барі). Належить до простору імен ISSTS.Converters, використовується для bindings у XAML для форматування часу як "MM:SS". Це стандартний IValueConverter для MVVM, що забезпечує зручне представлення тривалості процесів.

Структура та ключові компоненти:

- клас TimeSpanToStringConverter реалізує інтерфейс IValueConverter. Методи Convert (перетворення TimeSpan -> string) та ConvertBack (не реалізований);
- метод Convert: якщо value – TimeSpan, форматує як $"{(int)timeSpan.TotalMinutes:D2}:{timeSpan.Seconds:D2}" (наприклад, "05:30"). Інакше повертає порожній рядок;
- метод ConvertBack: кидає NotImplementedException, оскільки зворотна конвертація не потрібна;
- взаємодії: Використовується в UI (XAML bindings для тексту прогресу), інтегрується з ViewModel (властивості часу з Calculation);
- Достовірність: використовує CultureInfo для локалізації, але тут фіксований формат; проста логіка без помилок.

3 ТЕКСТ ПРОГРАМИ

1.1 Текст програми файлу App.cs

```
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.IO;
using System.Windows;

namespace ECSSTS
{
   using Data;
   using Newtonsoft.Json;
   using OfficeOpenXml;

   /// <summary>
   /// Interaction logic for App.xaml
   /// </summary>
   public partial class App : Application
   {
      static public ObservableCollection<DataExcel> data = new ObservableCollection<DataExcel>();
      static public ObservableCollection<MatchingSequenceGroup> sequences = new ObservableCollection<MatchingSequenceGroup>();
      static public double fractaldimension = 0.0;
      protected override void OnStartup(StartupEventArgs e)
      {
         base.OnStartup(e);
         // 1. Створюємо і показуємо MainWindow (активуємо UI)
         MainWindow = new MainWindow();
         MainWindow.Show();
         var mainVM = MainWindow.DataContext as MainViewModel;
         if (mainVM == null) return;
         // 1. Автозавантаження Fractal JSON (тільки шлях)
         string fractalPath = MainSettings.Default.FractalJsonFilePath;
         if (!string.IsNullOrEmpty(fractalPath) && File.Exists(fractalPath))
         {
            Debug.WriteLine($"OnStartup: Автозавантаження Fractal JSON: {fractalPath}");
            mainVM.OnFractalJsonPathChanged(fractalPath);
         }
         // 2. Автозавантаження головного JSON + фрактальна розмірність
         string jsonPath = MainSettings.Default.JsonFilePath;
         if (!string.IsNullOrEmpty(jsonPath) && File.Exists(jsonPath))
         {
            Debug.WriteLine($"OnStartup: Автозавантаження JSON: {jsonPath}");
            var (loadedSequences, excelFilePath, loadedfractalDimension, fractalResult) = LoadData.LoadFromJson(jsonPath);
            if (!double.IsNaN(loadedfractalDimension) && !double.IsInfinity(loadedfractalDimension))
            {
               fractaldimension = loadedfractalDimension;
               mainVM.FractalDimension = loadedfractalDimension;
               Debug.WriteLine($"OnStartup: Відновлено fractaldimension = {fractaldimension:F4} з {jsonPath}");
            }
            mainVM.LoadJsonFile(jsonPath);
         }
         // 3. Автозавантаження Excel
         string excelPath = MainSettings.Default.ExcelFilePath;
         if (!string.IsNullOrEmpty(excelPath) && File.Exists(excelPath) && !App.data.Any())
         {
            Debug.WriteLine($"OnStartup: Автозавантаження Excel: {excelPath}");
            mainVM.LoadExcelSilently(excelPath);
         }
      }
   }
}
```

1.2 Текст програми файлу App.xaml

```
<Application
   x:Class="ECSSTS.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation
"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:local="clr-namespace:ECSSTS">
   <Application.Resources>
      <ResourceDictionary>
         <BitmapImage x:Key="Save_image" UriSource="/Assets/Save image.ico" />
         <ResourceDictionary.MergedDictionaries>
            <!-- Theme -->
            <ResourceDictionary Source="/Styles/Colors.xaml" />
            <ResourceDictionary Source="/Styles/Brushes.xaml" />
            <!-- Buttons -->
            <ResourceDictionary Source="/Styles/Buttons/Main.xaml" />
```

```
            <!-- Menu -->
            <ResourceDictionary
Source="/Styles/Menu/MenuItem.xaml" />
            <ResourceDictionary
Source="/Styles/Menu/Separator.xaml" />
            <ResourceDictionary
Source="/Styles/Menu/ToollBar.xaml" />
            <ResourceDictionary
Source="/Styles/Menu/CheckBox.xaml" />
            <!-- Others -->
            <ResourceDictionary
Source="/Styles/Other/ScrollBar.xaml" />
            <ResourceDictionary
Source="/Styles/Other/DockPanel.xaml" />

        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

1.3 Текст програми файлу AssembleInfo.cs

```
using System.Windows;

[assembly: ThemeInfo(
   ResourceDictionaryLocation.None,           //where theme specific resource dictionaries are located
                                              //(used if a resource is not found in the page,
                                              // or application resource dictionaries)
   ResourceDictionaryLocation.SourceAssembly   //where the generic resource dictionary is located
                                              //(used if a resource is not found in the page,
                                              // app, or any theme specific resource dictionaries)
)]
```

1.4 Текст програми файлу MainSettings.settings.cs

```
<?xml version='1.0' encoding='utf-8'?>
<SettingsFile xmlns="http://schemas.microsoft.com/VisualStudio/2004/01/settings" CurrentProfile="(Default)"
GeneratedClassNamespace="ECSSTS.Data" GeneratedClassName="MainSettings">
  <Profiles />
  <Settings>
    <Setting Name="ExcelFilePath" Type="System.String" Scope="User">
      <Value Profile="(Default)" />
    </Setting>
    <Setting Name="JsonFilePath" Type="System.String" Scope="User">
      <Value Profile="(Default)" />
    </Setting>
    <Setting Name="IsDataListBoxVisible" Type="System.Boolean" Scope="User">
      <Value Profile="(Default)">True</Value>
    </Setting>
    <Setting Name="IsAutoSaveEnabled" Type="System.Boolean" Scope="User">
      <Value Profile="(Default)">False</Value>
    </Setting>
    <Setting Name="FractalJsonFilePath" Type="System.String" Scope="User">
      <Value Profile="(Default)" />
    </Setting>
    <Setting Name="IncludeSinglePointSequences" Type="System.Boolean" Scope="User">
      <Value Profile="(Default)">False</Value>
    </Setting>
  </Settings>
</SettingsFile>
```

2.1 Текст програми файлу DataExcel.cs

```
using OxyPlot;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
```

```
using System.Linq;

namespace ECSSTS.Data
{
    /// <summary>
    /// Представляє точку даних із координатами X і Y для часового ряду.
    /// </summary>
    public struct DataExcel
    {
        /// <summary>
        /// Координата X (зазвичай час або індекс).
        /// </summary>
        public double X { get; set; }

        /// <summary>
        /// Координата Y (значення часового ряду).
        /// </summary>
        public double Y { get; set; }

        /// <summary>
        /// Ініціалізує точку з координатами X і Y.
        /// </summary>
        /// <param name="x">Координата X.</param>
        /// <param name="y">Координата Y.</param>
        public DataExcel(double x, double y)
        {
            X = x;
            Y = y;
        }
    }

    /// <summary>
    /// Групує послідовності однакової довжини для організації результатів пошуку.
    /// </summary>
    public class SequenceLengthGroup
    {
        /// <summary>
        /// Довжина послідовностей у групі.
        /// </summary>
        public int Length { get; set; }

        /// <summary>
        /// Список послідовностей однакової довжини.
        /// </summary>
        public List<MatchingSequenceGroup> Sequences { get; set; }

        /// <summary>
        /// Кількість основних послідовностей у групі.
        /// </summary>
        public int MainSequencesCount => Sequences?.Count ?? 0;

        /// <summary>
        /// Кількість схожих послідовностей у групі.
        /// </summary>
        public int SimilarSequencesCount => Sequences?.Sum(s => s.SimilarSequenceIndices?.Count ?? 0) ?? 0;

        /// <summary>
        /// Загальна кількість послідовностей (основні + схожі).
        /// </summary>
        public int TotalSequencesCount => MainSequencesCount + SimilarSequencesCount;

        /// <summary>
        /// Ініціалізує групу з порожнім списком послідовностей.
        /// </summary>
        public SequenceLengthGroup()
        {
            Sequences = new List<MatchingSequenceGroup>();
        }
    }

    /// <summary>
    /// Представляє групу схожих послідовностей у часовому ряді.
    /// </summary>
    public class MatchingSequenceGroup
    {
        /// <summary>
        /// Унікальний ідентифікатор групи.
        /// </summary>
        public int GroupId { get; set; }

        /// <summary>
        /// Індекс початку базової послідовності у вхідному масиві.
        /// </summary>
        public int SequenceIndex { get; set; }

        /// <summary>
        /// Список точок базової послідовності.
        /// </summary>
        public List<DataExcel> Sequence { get; set; }

        /// <summary>
        /// Прапорець, що вказує, чи вибрана послідовність у UI.
        /// </summary>
        public bool IsSelected { get; set; }

        /// <summary>
        /// Список індексів схожих послідовностей.
        /// </summary>
        public List<int> SimilarSequenceIndices { get; set; }

        /// <summary>
        /// Довжина послідовності (кількість точок).
        /// </summary>
        public int SequenceLength { get; set; }

        /// <summary>
        /// Значення толерантності, використане для пошуку схожих послідовностей.
        /// </summary>
```

```
        public double ToleranceUsed { get; set; }

        /// <summary>
        /// Ініціалізує групу з порожніми списками для
Sequence і SimilarSequenceIndices.
        /// </summary>
        public MatchingSequenceGroup()
        {
            Sequence = new List<DataExcel>();
            SimilarSequenceIndices = new List<int>();
        }
    }

    public class FractalDimensionResult
    {
        public double TolerancePercentage { get; set; }
        public double Dimension { get; set; }
        public List<double> LogM { get; set; }
        public List<double> LogDelta { get; set; }
        public int MaxLength { get; set; }
        public List<int> Lengths { get; set; } // Додаємо для δ
        public ObservableCollection<SequenceLengthGroup>
SequenceGroups { get; set; }

        /// <summary>
        /// Кількість основних послідовностей у групі (обчис-
люється, якщо SequenceGroups доступні).
        /// </summary>
        public int MainSequencesCount =>
SequenceGroups?.Sum(g => g.MainSequencesCount) ?? 0;

        /// <summary>
        /// Кількість схожих послідовностей у групі (обчислю-
ється, якщо SequenceGroups доступні).
        /// </summary>
        public int SimilarSequencesCount =>
SequenceGroups?.Sum(g => g.SimilarSequencesCount) ?? 0;

        /// <summary>
        /// Загальна кількість послідовностей (основні +
схожі).
        /// </summary>
        public int TotalSequencesCount => MainSequencesCount
+ SimilarSequencesCount;

        /// <summary>
        /// Конструктор для повного набору даних.
        /// </summary>
        ///
        // Порожній конструктор для ініціалізації об'єкта
        public FractalDimensionResult()
        {
            LogM = new List<double>();
            LogDelta = new List<double>();
            Lengths = new List<int>();
            SequenceGroups = new
ObservableCollection<SequenceLengthGroup>();
        }
        public FractalDimensionResult(double
tolerancePercentage, double dimension, List<double> logM,
List<double> logDelta, int maxLength, List<int> lengths,
ObservableCollection<SequenceLengthGroup>
sequenceGroups)
        {
            TolerancePercentage = tolerancePercentage;
            Dimension = dimension;
            LogM = logM ?? new List<double>();
            LogDelta = logDelta ?? new List<double>();
            MaxLength = maxLength;
            Lengths = lengths ?? new List<int>(); // Ініціалізація
Lengths
            SequenceGroups = sequenceGroups ?? new
ObservableCollection<SequenceLengthGroup>();
        }

        public FractalDimensionResult(double
tolerancePercentage, double dimension, List<double> logM,
List<double> logDelta, int maxLength)
        {
            TolerancePercentage = tolerancePercentage;
            Dimension = dimension;
            LogM = logM ?? new List<double>();
            LogDelta = logDelta ?? new List<double>();
            MaxLength = maxLength;
            Lengths = new List<int>(); // Порожній список, якщо
lengths не передано
            SequenceGroups = new
ObservableCollection<SequenceLengthGroup>();
        }
    }
}
```

2.2 Текст програми файлу FileJson.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ECSSTS.Data
{
    class FileJson
    {
        public class JsonData
        {
            public string ExcelFilePath { get; set; }
            public double FractalDimension { get; set; }
            public double TolerancePercentage { get; set; } // Від-
носний допуск на рівні файлу
            public List<JsonSequence> Sequences { get; set; }
```

```
        public FractalDimensionResultDto FractalData { get;
    set; }
    }

    public class JsonSequence
    {
      public int GroupId { get; set; }
      public int SequenceIndex { get; set; }
      public bool IsSelected { get; set; }
      public List<int> SimilarSequenceIndices { get; set; }
      public List<JsonPoint> Sequence { get; set; }
    }

    public class JsonPoint
    {
      public double X { get; set; }
      public double Y { get; set; }
    }
  }
  public class FractalDimensionRangeSaveData
  {
    public string ExcelFilePath { get; set; }
    public string JsonFilePath { get; set; }
    public List<FractalDimensionResult> Results { get; set; }
  }
  public class FractalDimensionResultDto
  {
    public double TolerancePercentage { get; set; }
    public double Dimension { get; set; }
    public List<double> LogM { get; set; }
    public List<double> LogDelta { get; set; }
    public int MaxLength { get; set; }
  }
}
```

3.1 Текст програми файлу **Calculation**.cs

```
using System.Collections.Concurrent;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using ECSSTS.Data;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Channels;

namespace ECSSTS.Processing
{
  public class Calculation
  {
    public static async Task<List<MatchingSequenceGroup>>
SearchAsync(
      ObservableCollection<DataExcel> dataPoints,
      double tolerance,
      IProgress<double> progress = null,
      IProgress<TimeSpan> estimatedTimeProgress = null,
      CancellationToken cancellationToken = default,
      int minLength = 1,
      int? maxLength = null)
    {
      cancellationToken.ThrowIfCancellationRequested();
      var stopwatch = Stopwatch.StartNew();
      var points = dataPoints.ToArray();
      int n = points.Length;
      if (n < minLength) return new
List<MatchingSequenceGroup>();

      // Розділення довжин на вісім груп за модулем 8
      var lengthGroups = new List<int>[8];
      for (int i = 0; i < 8; i++) lengthGroups[i] = new List<int>();
      int actualMaxLength = Math.Min(maxLength ?? n, n);
      int totalLengths = actualMaxLength - minLength + 1; // Зага-
льна кількість довжин

      for (int length = minLength; length <= actualMaxLength;
length++)
      {
        cancellationToken.ThrowIfCancellationRequested();
        lengthGroups[length % 8].Add(length);
      }

      // Канал для буферизації прогресу
      var progressChannel = Channel.CreateUnbounded<(int
taskIndex, double progress)>();
      var progressLock = new object();
      var taskProgress = new double[8]; // Прогрес для кожного з
8 завдань
      double lastReportedProgress = -1; // Останній переданий
прогрес

      // Фоновий Task для оновлення UI
      var progressUpdateTask = Task.Run(async () =>
      {
        while                                          (await
progressChannel.Reader.WaitToReadAsync(cancellationToken))
        {
          while (progressChannel.Reader.TryRead(out var update))
          {
            lock (progressLock)
            {
              taskProgress[update.taskIndex] = update.progress;
              double averageProgress = taskProgress.Average();
              // Оновлюємо UI лише якщо прогрес змінився на
>= 5%
```

```
                if (Math.Abs(averageProgress - lastReportedProgress) >= 5)
                {
                    progress?.Report(averageProgress);
                    lastReportedProgress = averageProgress;

                    // Оцінка залишкового часу на основі загального прогресу
                    if (averageProgress > 0 && averageProgress < 100)
                    {
                        double elapsedSeconds = stopwatch.Elapsed.TotalSeconds;
                        double estimatedTotalSeconds = elapsedSeconds / (averageProgress / 100);
                        double remainingSeconds = estimatedTotalSeconds - elapsedSeconds;
                        if (remainingSeconds >= 0)
                        {
                            estimatedTimeProgress?.Report(TimeSpan.FromSeconds(remainingSeconds));
                        }
                    }
                }
            }
            await Task.Delay(500, cancellationToken); // Оновлення кожні 500 мс
        }
    }, cancellationToken);

    // Запуск восьми паралельних завдань
    var tasks = new Task<List<MatchingSequenceGroup>>[8];
    for (int i = 0; i < 8; i++)
    {
        int taskIndex = i; // Захоплення індексу для замикання
        tasks[i] = Task.Run(() => ProcessLengths(points, lengthGroups[taskIndex], tolerance, p => progressChannel.Writer.WriteAsync((taskIndex, p), cancellationToken).GetAwaiter().GetResult(), cancellationToken), cancellationToken);
    }

    // Очікування завершення всіх завдань і об'єднання результатів
    var results = await Task.WhenAll(tasks);
    var combinedResults = results
        .SelectMany(r => r)
        .OrderBy(g => g.SequenceLength) // Сортування за зростанням довжини послідовності
        .ToList();

    // Перенумерування GroupId для забезпечення правильної послідовності
    int groupId = 1;
    foreach (var group in combinedResults)
    {
        group.GroupId = groupId++;
    }

    // Завершення каналу прогресу
    progressChannel.Writer.Complete();
    await progressUpdateTask;

    stopwatch.Stop();
    lock (progressLock)
    {
        progress?.Report(100);
        estimatedTimeProgress?.Report(TimeSpan.Zero); // Залишковий час = 0 після завершення
    }
    Debug.WriteLine($"SearchAsync: Пошук завершено, знайдено {combinedResults.Count} послідовностей за {stopwatch.Elapsed.TotalSeconds:F3} сек");
    return combinedResults;
}

private static List<MatchingSequenceGroup> ProcessLengths(
DataExcel[] points,
List<int> lengths,
double tolerance,
Action<double> progress,
CancellationToken cancellationToken)
{
    var result = new List<MatchingSequenceGroup>();
    int n = points.Length;
    var lengthToUsedIndices = new Dictionary<int, HashSet<int>>();
    var excludedIndices = new HashSet<int>();
    bool includeSinglePointSequences = MainSettings.Default.IncludeSinglePointSequences;

    int totalLengths = lengths.Count;
    int currentLengthIndex = 0;
    foreach (int length in lengths.OrderBy(l => l))
    {
        cancellationToken.ThrowIfCancellationRequested();
        double progressPercentage = (double)(currentLengthIndex++) / totalLengths * 100;
        progress?.Invoke(progressPercentage);

        if (!lengthToUsedIndices.TryGetValue(length, out var used))
```

```
        {
            used = new HashSet<int>();
            lengthToUsedIndices[length] = used;
        }

        int maxStartIndex = n - length;
        var directions = new SequenceDirection[maxStartIndex + 1];
        for (int i = 0; i <= maxStartIndex; i++)
        {
            cancellationToken.ThrowIfCancellationRequested();
            directions[i] = GetSequenceDirection(points, i, length, tolerance);
        }

        int sequencesAdded = 0;
        for (int i = 0; i <= maxStartIndex; i++)
        {
            cancellationToken.ThrowIfCancellationRequested();
            if (used.Contains(i) || excludedIndices.Contains(i))
                continue;

            var baseDirection = directions[i];
            var similarIndices = new List<int>(maxStartIndex / 2);
            for (int j = 0; j <= maxStartIndex; j++)
            {
                cancellationToken.ThrowIfCancellationRequested();
                if (j == i || used.Contains(j) || SequencesOverlap(i, j, length) || excludedIndices.Contains(j))
                    continue;
                if (directions[j] != baseDirection)
                    continue;
                if (baseDirection == SequenceDirection.Stable &&
                    !QuickStableCheck(points, i, j, length, tolerance))
                    continue;
                if (AreSequencesSimilar(points, i, j, length, tolerance))
                {
                    similarIndices.Add(j);
                }
            }

            var allIndices = new List<int> { i };
            allIndices.AddRange(similarIndices);
            allIndices.Sort();
            var signature = $"{length}:{string.Join(",", allIndices)}";
            var baseSeq = points.Skip(i).Take(length).ToList();

            if (similarIndices.Count > 0 || (length == 1 && includeSinglePointSequences))
            {
                Debug.WriteLine($"ProcessLengths: Знайдено групу: Length={length}, SequenceIndex={i}, SimilarIndices={string.Join(",", similarIndices)}, Signature={signature}, M={1 + similarIndices.Count}");
                result.Add(new MatchingSequenceGroup
                {
                    GroupId = 0,
                    SequenceIndex = i,
                    Sequence = baseSeq,
                    SimilarSequenceIndices = similarIndices,
                    SequenceLength = length
                });
                sequencesAdded++;
                MarkIndicesAsUsed(i, length, used);
                foreach (var idx in similarIndices)
                {
                    MarkIndicesAsUsed(idx, length, used);
                }
            }
            else
            {
                excludedIndices.Add(i);
            }
        }
        Debug.WriteLine($"ProcessLengths: Для довжини {length} додано {sequencesAdded} груп");
    }

    Debug.WriteLine($"ProcessLengths: Загалом додано {result.Count} груп для довжин {string.Join(",", lengths)}");
    return result;
}

private static bool QuickStableCheck(DataExcel[] data, int i1, int i2, int length, double tolerance)
{
    return Math.Abs(data[i1].Y - data[i2].Y) <= tolerance &&
           Math.Abs(data[i1 + length - 1].Y - data[i2 + length - 1].Y) <= tolerance;
}

private static bool AreSequencesSimilar(DataExcel[] data, int i1, int i2, int length, double tolerance)
{
    for (int k = 0; k < length; k++)
    {
        if (Math.Abs(data[i1 + k].Y - data[i2 + k].Y) > tolerance)
            return false;
    }
    return true;
}
```

```
    private static SequenceDirection GetSequenceDirection(DataExcel[] points, int start, int length, double tolerance)
    {
        if (length < 2) return SequenceDirection.Stable;

        double firstY = points[start].Y;
        double lastY = points[start + length - 1].Y;
        double diff = lastY - firstY;

        // Використовуємо tolerance для порогу стабільності (узгоджено з QuickStableCheck)
        if (Math.Abs(diff) <= tolerance)
            return SequenceDirection.Stable;

        return diff > 0 ? SequenceDirection.Increasing : SequenceDirection.Decreasing;
    }

    private static void MarkIndicesAsUsed(int start, int length, HashSet<int> usedIndices)
    {
        int end = start + length;
        for (int i = start; i < end; i++)
        {
            usedIndices.Add(i);
        }
    }

    private static bool SequencesOverlap(int start1, int start2, int length)
    {
        return start1 < start2 + length && start2 < start1 + length;
    }

    private enum SequenceDirection
    {
        Increasing,
        Decreasing,
        Stable
    }

    public static (double dimension, List<double> logM, List<double> logDelta, int maxLength, List<int> lengths) CalculateFractalDimensionBySequence(List<SequenceLengthGroup> sequenceLengths)
    {
        if (sequenceLengths == null || !sequenceLengths.Any())
        {
            MessageBox.Show("Недостатньо даних для обчислення фрактальної розмірності.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            Debug.WriteLine("CalculateFractalDimensionBySequence: sequenceLengths порожній або null");
            return (0, new List<double>(), new List<double>(), 0, new List<int>());
        }

        var logM = new List<double>();
        var logDelta = new List<double>();
        var lengths = new List<int>();
        int maxLength = 0;
        bool includeSinglePointSequences = MainSettings.Default.IncludeSinglePointSequences;

        Debug.WriteLine("Перевірка SequenceLengths:");
        foreach (var group in sequenceLengths.OrderBy(g => g.Length))
        {
            int delta = group.Length;
            if (!includeSinglePointSequences && delta == 1)
            {
                Debug.WriteLine($"Пропущено: Length={delta} (послідовності довжиною 1 виключено)");
                continue;
            }

            int M = group.Sequences.Sum(seq => 1 + seq.SimilarSequenceIndices.Count);
            Debug.WriteLine($"Length: {delta}, Total Sequences (M): {M} (Sequences: {group.Sequences.Count}, Similar: {group.Sequences.Sum(s => s.SimilarSequenceIndices.Count)})");

            if (M > 0 && delta > 0)
            {
                try
                {
                    double logMValue = Math.Log(M);
                    double logDeltaValue = Math.Log(1.0 / delta);
                    logM.Add(logMValue);
                    logDelta.Add(logDeltaValue);
                    lengths.Add(delta);
                    maxLength = Math.Max(maxLength, delta);
                    Debug.WriteLine($"Додано: delta={delta}, M={M}, logM={logMValue:F4}, logDelta={logDeltaValue:F4}");
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Помилка логарифмування: M={M}, δ={delta}. {ex.Message}", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
                    Debug.WriteLine($"Помилка: M={M}, δ={delta}, Exception: {ex.Message}");
```

```
            }
        }
        else
        {
            Debug.WriteLine($"Пропущено: M={M}, δ={delta} (невалідні значення)");
        }
    }

    Debug.WriteLine($"logM: [{string.Join(", ", logM.Select(v => v.ToString("F4")))}]");
    Debug.WriteLine($"logDelta: [{string.Join(", ", logDelta.Select(v => v.ToString("F4")))}]");
    Debug.WriteLine($"lengths: [{string.Join(", ", lengths)}]");

    if (logM.Count < 2)
    {
        MessageBox.Show($"Недостатньо валідних даних для регресії. Знайдено {logM.Count} груп(и). Перевірте SequenceLengths.", "Попередження", MessageBoxButton.OK, MessageBoxImage.Warning);

Debug.WriteLine($"CalculateFractalDimensionBySequence: Недостатньо даних для регресії, logM.Count={logM.Count}");
        return (0, logM, logDelta, maxLength, lengths);
    }

    double slope = LinearRegression(logDelta, logM);
    double fractalDimension = -slope;
    if (slope > 0)
    {
        Debug.WriteLine($"Попередження: Нахил регресії додатній ({slope:F4}), що вказує на некоректні дані. D={fractalDimension:F4}, використовується Math.Abs(slope)={Math.Abs(slope):F4}");
        fractalDimension = Math.Abs(slope);
    }

    Debug.WriteLine($"Фрактальна розмірність: {fractalDimension:F4}, maxLength: {maxLength}, logM.Count: {logM.Count}");
    return (fractalDimension, logM, logDelta, maxLength, lengths);
}

private static double LinearRegression(List<double> x, List<double> y)
{
    int n = x.Count;
    if (n != y.Count)
    {
        Debug.WriteLine($"LinearRegression: Некоректні дані, x.Count={x.Count}, y.Count={y.Count}");
        return 0;
    }

    double sumX = x.Sum();
    double sumY = y.Sum();
    double sumXY = x.Zip(y, (a, b) => a * b).Sum();
    double sumXX = x.Sum(a => a * a);

    double denominator = n * sumXX - sumX * sumX;
    if (Math.Abs(denominator) < 1e-10)
    {
        Debug.WriteLine("LinearRegression: Ділення на нуль у знаменнику регресії");
        return 0;
    }

    double slope = (n * sumXY - sumX * sumY) / denominator;
    Debug.WriteLine($"LinearRegression: n={n}, sumX={sumX:F4}, sumY={sumY:F4}, sumXY={sumXY:F4}, sumXX={sumXX:F4}, slope={slope:F4}");
    return slope;
}

public static async Task<List<MatchingSequenceGroup>> SearchRelativeAsync(
    ObservableCollection<DataExcel> dataPoints,
    double tolerance,
    IProgress<double> progress = null,
    IProgress<TimeSpan> estimatedTimeProgress = null,
    CancellationToken cancellationToken = default,
    int minLength = 1,
    int? maxLength = null)
{
    cancellationToken.ThrowIfCancellationRequested();
    var stopwatch = Stopwatch.StartNew();
    var points = dataPoints.ToArray();
    int n = points.Length;
    if (n < minLength) return new List<MatchingSequenceGroup>();

    // Розділення довжин на вісім груп за модулем 8
    var lengthGroups = new List<int>[8];
    for (int i = 0; i < 8; i++) lengthGroups[i] = new List<int>();
    int actualMaxLength = Math.Min(maxLength ?? n, n);
    int totalLengths = actualMaxLength - minLength + 1;

    for (int length = minLength; length <= actualMaxLength; length++)
    {
        cancellationToken.ThrowIfCancellationRequested();
```

```
            lengthGroups[length % 8].Add(length);
        }

        // Канал для буферизації прогресу
        var progressChannel = Channel.CreateUnbounded<(int taskIndex, double progress)>();
        var progressLock = new object();
        var taskProgress = new double[8];
        double lastReportedProgress = -1;

        // Фоновий Task для оновлення UI
        var progressUpdateTask = Task.Run(async () =>
        {
            while (await progressChannel.Reader.WaitToReadAsync(cancellationToken))
            {
                while (progressChannel.Reader.TryRead(out var update))
                {
                    lock (progressLock)
                    {
                        taskProgress[update.taskIndex] = update.progress;
                        double averageProgress = taskProgress.Average();
                        if (Math.Abs(averageProgress - lastReportedProgress) >= 5)
                        {
                            progress?.Report(averageProgress);
                            lastReportedProgress = averageProgress;

                            if (averageProgress > 0 && averageProgress < 100)
                            {
                                double elapsedSeconds = stopwatch.Elapsed.TotalSeconds;
                                double estimatedTotalSeconds = elapsedSeconds / (averageProgress / 100);
                                double remainingSeconds = estimatedTotalSeconds - elapsedSeconds;
                                if (remainingSeconds >= 0)
                                {

                                    estimatedTimeProgress?.Report(TimeSpan.FromSeconds(remainingSeconds));
                                }
                            }
                        }
                    }
                }
                await Task.Delay(500, cancellationToken);
            }
        }, cancellationToken);

        // Запуск восьми паралельних завдань
        var tasks = new Task<List<MatchingSequenceGroup>>[8];
        for (int i = 0; i < 8; i++)
        {
            int taskIndex = i;
            tasks[i] = Task.Run(() => ProcessLengthsRelative(points, lengthGroups[taskIndex], tolerance, p => progressChannel.Writer.WriteAsync((taskIndex, p), cancellationToken).GetAwaiter().GetResult(), cancellationToken), cancellationToken);
        }

        // Очікування завершення всіх завдань і об'єднання результатів
        var results = await Task.WhenAll(tasks);
        var combinedResults = results
            .SelectMany(r => r)
            .OrderBy(g => g.SequenceLength)
            .ToList();

        // Перенумерування GroupId
        int groupId = 1;
        foreach (var group in combinedResults)
        {
            group.GroupId = groupId++;
        }

        // Завершення каналу прогресу
        progressChannel.Writer.Complete();
        await progressUpdateTask;

        stopwatch.Stop();
        lock (progressLock)
        {
            progress?.Report(100);
            estimatedTimeProgress?.Report(TimeSpan.Zero);
        }
        Debug.WriteLine($"SearchRelativeAsync: Пошук завершено, знайдено {combinedResults.Count} послідовностей за {stopwatch.Elapsed.TotalSeconds:F3} сек");
        return combinedResults;
    }

    private static List<MatchingSequenceGroup> ProcessLengthsRelative(
        DataExcel[] points,
        List<int> lengths,
        double tolerance,
        Action<double> progress,
        CancellationToken cancellationToken)
    {
        var result = new List<MatchingSequenceGroup>();
        int n = points.Length;
```

```
        var lengthToUsedIndices = new Dictionary<int, HashSet<int>>();
        var excludedIndices = new HashSet<int>();
        bool includeSinglePointSequences = MainSettings.Default.IncludeSinglePointSequences;

        int totalLengths = lengths.Count;
        int currentLengthIndex = 0;
        foreach (int length in lengths.OrderBy(l => l))
        {
            cancellationToken.ThrowIfCancellationRequested();
            double progressPercentage = (double)(currentLengthIndex++) / totalLengths * 100;
            progress?.Invoke(progressPercentage);

            if (!lengthToUsedIndices.TryGetValue(length, out var used))
            {
                used = new HashSet<int>();
                lengthToUsedIndices[length] = used;
            }

            int maxStartIndex = n - length;
            var directions = new SequenceDirection[maxStartIndex + 1];
            for (int i = 0; i <= maxStartIndex; i++)
            {
                cancellationToken.ThrowIfCancellationRequested();
                directions[i] = GetSequenceDirection(points, i, length, tolerance);
            }

            int sequencesAdded = 0;
            for (int i = 0; i <= maxStartIndex; i++)
            {
                cancellationToken.ThrowIfCancellationRequested();
                if (used.Contains(i) || excludedIndices.Contains(i))
                    continue;

                var baseDirection = directions[i];
                var similarIndices = new List<int>(maxStartIndex / 2);
                for (int j = 0; j <= maxStartIndex; j++)
                {
                    cancellationToken.ThrowIfCancellationRequested();
                    if (j == i || used.Contains(j) || SequencesOverlap(i, j, length) || excludedIndices.Contains(j))
                        continue;
                    if (directions[j] != baseDirection)
                        continue;
                    if (baseDirection == SequenceDirection.Stable &&
                        !QuickStableCheck(points, i, j, length, tolerance))
                        continue;
                    if (AreSequencesSimilarRelative(points, i, j, length, tolerance))
                    {
                        similarIndices.Add(j);
                    }
                }

                var allIndices = new List<int> { i };
                allIndices.AddRange(similarIndices);
                allIndices.Sort();
                var signature = $"{length}:{string.Join(",", allIndices)}";
                var baseSeq = points.Skip(i).Take(length).ToList();

                if (similarIndices.Count > 0 || (length == 1 && includeSinglePointSequences))
                {
                    Debug.WriteLine($"Знайдено групу: Length={length}, SequenceIndex={i}, SimilarIndices={string.Join(",", similarIndices)}, Signature={signature}");
                    result.Add(new MatchingSequenceGroup
                    {
                        GroupId = 0,
                        SequenceIndex = i,
                        Sequence = baseSeq,
                        SimilarSequenceIndices = similarIndices,
                        SequenceLength = length
                    });
                    sequencesAdded++;
                    MarkIndicesAsUsed(i, length, used);
                    foreach (var idx in similarIndices)
                    {
                        MarkIndicesAsUsed(idx, length, used);
                    }
                }
                else
                {
                    excludedIndices.Add(i);
                }
            }
            Debug.WriteLine($"ProcessLengthsRelative: Для довжини {length} додано {sequencesAdded} груп");
        }

        Debug.WriteLine($"ProcessLengthsRelative: Загалом додано {result.Count} груп для довжин {string.Join(",", lengths)}");
        return result;
    }

    private static bool AreSequencesSimilarRelative(DataExcel[] data, int i1, int i2, int length, double tolerance)
    {
```

```
        if (i1 == i2) return false;

        // Обчислюємо базову різницю між першими точками
        double baseDifference = data[i1].Y - data[i2].Y;

        // Перевіряємо кожну точку, нормалізуючи другу послідовність

        for (int k = 0; k < length; k++)
        {
            // Нормалізоване значення Y для другої послідовності
            double normalizedY2 = data[i2 + k].Y + baseDifference;
            // Порівнюємо з Y першої послідовності
            if (Math.Abs(data[i1 + k].Y - normalizedY2) > tolerance)
            {
                return false;
            }
        }

        return true;
    }
  }
}
```

3.2 Текст програми файлу LoadData.cs

```
using Microsoft.Win32;
using Newtonsoft.Json;
using OfficeOpenXml;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Windows;

namespace ECSSTS.Data
{
  public class LoadData
  {
    public static string GetSavedFilePath()
    {
      Debug.WriteLine($"GetSavedFilePath: Повертаємо шлях: {MainSettings.Default.ExcelFilePath}");
      return MainSettings.Default.ExcelFilePath;
    }

    public static string SelectExcelFile()
    {
      Debug.WriteLine("SelectExcelFile: Відкриття діалогового вікна для вибору Excel");
      OpenFileDialog openFileDialog = new OpenFileDialog
      {
        Filter = "Excel файли (*.xlsx;*.xls)|*.xlsx;*.xls|Усі файли (*.*)|*.*",
        Title = "Виберіть Excel-файл"
      };

      bool? result = openFileDialog.ShowDialog();
      Debug.WriteLine($"SelectExcelFile: Результат діалогу: {result}, Вибраний файл: {openFileDialog.FileName}");
      if (result == true)
      {
        return openFileDialog.FileName;
      }

      return null;
    }

    public static string SelectExcelFile(string title)
    {
      Debug.WriteLine($"SelectExcelFile: Відкриття діалогового вікна з заголовком: {title}");
      OpenFileDialog openFileDialog = new OpenFileDialog
      {
        Filter = "Excel файли (*.xlsx;*.xls)|*.xlsx;*.xls|Усі файли (*.*)|*.*",
        Title = title
      };

      bool? result = openFileDialog.ShowDialog();
      Debug.WriteLine($"SelectExcelFile: Результат діалогу: {result}, Вибраний файл: {openFileDialog.FileName}");
      if (result == true)
      {
        return openFileDialog.FileName;
      }

      return null;
    }

    public static string SelectJsonFile()
    {
      Debug.WriteLine("SelectJsonFile: Відкриття діалогового вікна для вибору JSON");
      OpenFileDialog openFileDialog = new OpenFileDialog
      {
        Filter = "JSON файли (*.json)|*.json|Усі файли (*.*)|*.*",
        Title = "Виберіть JSON-файл"
```

```
        };

        bool? result = openFileDialog.ShowDialog();
        Debug.WriteLine($"SelectJsonFile: Результат діалогу: {result}, Вибраний файл: {openFileDialog.FileName}");
        if (result == true)
        {
            return openFileDialog.FileName;
        }

        return null;
    }

    public static string SelectFractalJsonFile()
    {
        Debug.WriteLine("SelectFractalJsonFile: Відкриття діалогового вікна для вибору Fractal JSON");
        OpenFileDialog openFileDialog = new OpenFileDialog
        {
            Filter = "JSON файли (*.json)|*.json|Усі файли (*.*)|*.*",
            Title = "Виберіть Fractal JSON-файл"
        };

        bool? result = openFileDialog.ShowDialog();
        Debug.WriteLine($"SelectFractalJsonFile: Результат діалогу: {result}, Вибраний файл: {openFileDialog.FileName}");
        if (result == true)
        {
            return openFileDialog.FileName;
        }

        return null;
    }

    public static List<DataExcel> LoadDataFromExcel(string filePath)
    {
        var dataPoints = new List<DataExcel>();
        FileInfo fileInfo = new FileInfo(filePath);
        ExcelPackage.LicenseContext = LicenseContext.NonCommercial;
        try
        {
            Debug.WriteLine($"LoadDataFromExcel: Завантаження Excel: {filePath}");
            using (var package = new ExcelPackage(fileInfo))
            {
                if (package.Workbook.Worksheets.Count == 0)
                {
                    MessageBox.Show("Excel-файл не містить аркушів.", "Помилка формату файлу", MessageBoxButton.OK, MessageBoxImage.Error);
                    Debug.WriteLine("LoadDataFromExcel: Excel-файл не містить аркушів");
                    return dataPoints;
                }

                ExcelWorksheet worksheet = package.Workbook.Worksheets[0];
                int row = 1;
                int index = 1;
                while (true)
                {
                    var xCell = worksheet.Cells[row, 1];
                    var yCell = worksheet.Cells[row, 2];

                    if (xCell.Value == null && yCell.Value == null)
                        break;

                    double xValue = double.NaN;
                    double yValue = double.NaN;

                    if (xCell.Value != null && yCell.Value != null)
                    {
                        if (!double.TryParse(xCell.Value.ToString(), out xValue) || double.IsNaN(xValue))
                        {
                            MessageBox.Show($"Помилка в рядку {row}, стовпець 1: '{xCell.Value}' не є числом.", "Помилка даних", MessageBoxButton.OK, MessageBoxImage.Error);
                            return dataPoints;
                        }
                        if (!double.TryParse(yCell.Value.ToString(), out yValue) || double.IsNaN(yValue))
                        {
                            MessageBox.Show($"Помилка в рядку {row}, стовпець 2: '{yCell.Value}' не є числом.", "Помилка даних", MessageBoxButton.OK, MessageBoxImage.Error);
                            return dataPoints;
                        }
                    }
                    else if (xCell.Value != null)
                    {
                        if (!double.TryParse(xCell.Value.ToString(), out yValue) || double.IsNaN(yValue))
                        {
                            MessageBox.Show($"Помилка в рядку {row}, стовпець 1: '{xCell.Value}' не є числом.", "Помилка даних", MessageBoxButton.OK, MessageBoxImage.Error);
                            return dataPoints;
                        }
                        xValue = index++;
                    }
                    else if (yCell.Value != null)
```

```
            {
                if (!double.TryParse(yCell.Value.ToString(), out yValue) || double.IsNaN(yValue))
                {
                    MessageBox.Show($"Помилка в рядку {row}, стовпець 2: '{yCell.Value}' не є числом.", "Помилка даних", MessageBoxButton.OK, MessageBoxImage.Error);
                    return dataPoints;
                }
                xValue = index++;
            }
            else
            {
                row++;
                continue;
            }

            dataPoints.Add(new DataExcel(xValue, yValue));
            row++;
        }

        Debug.WriteLine($"LoadDataFromExcel: Успішно завантажено {dataPoints.Count} точок із Excel ({(dataPoints.Count > 0 ? "два стовпці" : "один стовпець")})");
    }
    catch (IOException ex) when (ex.Message.Contains("being used by another process"))
    {
        MessageBox.Show("Файл зайнятий іншим процесом. Закрийте файл і спробуйте знову.", "Помилка доступу до файлу", MessageBoxButton.OK, MessageBoxImage.Warning);
        Debug.WriteLine($"LoadDataFromExcel: Помилка доступу: {ex.Message}");
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка при завантаженні Excel: {ex.Message}.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
        Debug.WriteLine($"LoadDataFromExcel: Помилка: {ex.Message}");
    }
    return dataPoints;
}

public static (ObservableCollection<MatchingSequenceGroup>, string, double, FractalDimensionResult) LoadFromJson(string filePath)
{
    try
    {
        Debug.WriteLine($"LoadFromJson: Читання JSON: {filePath}");
        string json = File.ReadAllText(filePath);
        var data = JsonConvert.DeserializeObject<FileJson.JsonData>(json);
        if (data == null)
        {
            Debug.WriteLine("LoadFromJson: Десеріалізація JSON повернула null");
            return (null, null, 0, null);
        }

        var viewModel = (Application.Current.MainWindow?.DataContext as MainViewModel);
        double tolerance = viewModel?.Tolerance ?? 0.1;
        var sequences = new ObservableCollection<MatchingSequenceGroup>();
        foreach (var seq in data.Sequences)
        {
            sequences.Add(new MatchingSequenceGroup
            {
                GroupId = seq.GroupId,
                SequenceIndex = seq.SequenceIndex,
                IsSelected = seq.IsSelected,
                SimilarSequenceIndices = seq.SimilarSequenceIndices,
                Sequence = seq.Sequence.Select(p => new DataExcel
                {
                    X = p.X,
                    Y = p.Y
                }).ToList(),
                ToleranceUsed = tolerance
            });
        }
        Debug.WriteLine($"LoadFromJson: Завантажено {sequences.Count} послідовностей, TolerancePercentage={data.TolerancePercentage:F2}");

        // Обробка фрактальних даних
        FractalDimensionResult fractalResult = null;
        if (data.FractalData != null)
        {
            fractalResult = new FractalDimensionResult(
                data.FractalData.TolerancePercentage,
                data.FractalData.Dimension,
                data.FractalData.LogM,
                data.FractalData.LogDelta,
                data.FractalData.MaxLength
            );
```

```
            Debug.WriteLine($"LoadFromJson: Завантажено фрактальні дані: D={data.FractalData.Dimension:F4}, Tolerance={data.FractalData.TolerancePercentage:F2}%");
        }

        return (sequences, data.ExcelFilePath, data.FractalDimension, fractalResult);
      }
      catch (Exception ex)
      {
        Debug.WriteLine($"LoadFromJson: Помилка десеріалізації JSON: {ex.Message}");
        return (null, null, 0, null);
      }
    }


    public static (List<FractalDimensionResult>, string, string) LoadFractalDimensionRangeFromJson(string filePath)
    {
      try
      {

Debug.WriteLine($"LoadFractalDimensionRangeFromJson: Читання Fractal JSON: {filePath}");
        string json = File.ReadAllText(filePath);
        var data = JsonConvert.DeserializeObject<FractalDimensionRangeSaveData>(json);
        if (data == null)
        {

Debug.WriteLine("LoadFractalDimensionRangeFromJson: Десеріалізація Fractal JSON повернула null");
          return (null, null, null);
        }


Debug.WriteLine($"LoadFractalDimensionRangeFromJson: Завантажено {data.Results?.Count ?? 0} результатів");
        return (data.Results, data.ExcelFilePath, data.JsonFilePath);
      }
      catch (Exception ex)
      {

Debug.WriteLine($"LoadFractalDimensionRangeFromJson: Помилка десеріалізації Fractal JSON: {ex.Message}");
        return (null, null, null);
      }
    }
  }
}
```

3.3 Текст програми файлу **SaveData**.cs

```
using Microsoft.Win32;
using Newtonsoft.Json;
using OfficeOpenXml;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Windows;

namespace ECSSTS.Data
{
  public class SaveData
  {
    public static string DefaultFilePath => Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Result.json");
    // DTO для серіалізації, без SequenceGroups
    // DTO для серіалізації, без SequenceGroups
    private class FractalDimensionResultDto
    {
      public double TolerancePercentage { get; set; }
      public double Dimension { get; set; }
      public List<double> LogM { get; set; }
      public List<double> LogDelta { get; set; }
      public int MaxLength { get; set; } // Замість List<int> Lengths
    }

    public static string SelectSaveJsonFile(string excelFilePath)
    {
      string defaultFileName = !string.IsNullOrEmpty(excelFilePath)
        ? Path.GetFileNameWithoutExtension(excelFilePath) + ".json"
        : "Result.json";
      SaveFileDialog saveFileDialog = new SaveFileDialog
      {
        Filter = "JSON файли (*.json)|*.json|Усі файли (*.*)|*.*",
        Title = "Виберіть місце для збереження результатів",
        DefaultExt = ".json",
        AddExtension = true,
        FileName = defaultFileName
      };
      return saveFileDialog.ShowDialog() == true ? saveFileDialog.FileName : null;
```

```
    }

    public static string SelectSaveFractalJsonFile(string excelFilePath)
    {
        string defaultFileName = !string.IsNullOrEmpty(excelFilePath)
            ? Path.GetFileNameWithoutExtension(excelFilePath) + "_Fractal.json"
            : "Fractal.json";
        SaveFileDialog saveFileDialog = new SaveFileDialog
        {
            Filter = "JSON файли (*.json)|*.json|Усі файли (*.*)|*.*",
            Title = "Виберіть місце для збереження Fractal JSON",
            DefaultExt = ".json",
            AddExtension = true,
            FileName = defaultFileName
        };
        return saveFileDialog.ShowDialog() == true ? saveFileDialog.FileName : null;
    }

    public static bool SaveToJson(ObservableCollection<MatchingSequenceGroup> sequenceGroups, string filePath, string excelFilePath, double fractalDimension, double tolerancePercentage)
    {
        if (string.IsNullOrEmpty(filePath))
        {
            return false;
        }
        try
        {
            var viewModel = (Application.Current.MainWindow?.DataContext as MainViewModel);
            var fractalResult = viewModel?._fractalResults?.FirstOrDefault(r => Math.Abs(r.TolerancePercentage - tolerancePercentage) < 0.01);

            var dataToSave = new
            {
                ExcelFilePath = excelFilePath,
                // Видаляємо дублювання FractalDimension і TolerancePercentage на верхньому рівні
                FractalData = fractalResult != null ? new FractalDimensionResultDto
                {
                    TolerancePercentage = fractalResult.TolerancePercentage,
                    Dimension = fractalResult.Dimension,
                    LogM = fractalResult.LogM,
                    LogDelta = fractalResult.LogDelta,
                    MaxLength = fractalResult.MaxLength
                } : null,
                Sequences = sequenceGroups.Select(group => new
                {
                    GroupId = group.GroupId,
                    SequenceIndex = group.SequenceIndex,
                    IsSelected = group.IsSelected,
                    SimilarSequenceIndices = group.SimilarSequenceIndices,
                    Sequence = group.Sequence.Select(point => new
                    {
                        X = Math.Round(point.X, 4),
                        Y = Math.Round(point.Y, 6)
                    }).ToList(),
                    SequenceLength = group.Sequence.Count
                }).ToList()
            };

            var settings = new JsonSerializerSettings
            {
                Formatting = Formatting.Indented,
                NullValueHandling = NullValueHandling.Ignore
            };
            string json = JsonConvert.SerializeObject(dataToSave, settings);
            File.WriteAllText(filePath, json);
            Debug.WriteLine($"SaveToJson: Збережено JSON: {filePath}, Sequences={sequenceGroups.Count}, FractalData={(fractalResult != null ? "додано" : "відсутнє")}");
            return true;
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Помилка при збереженні JSON: {ex.Message}", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return false;
        }
    }

    public static bool SaveFractalDimensionRangeToJson(List<FractalDimensionResult> results, string filePath, string excelFilePath, string jsonFilePath, bool isFullSave)
    {
        try
        {
            object data;
            if (isFullSave)
            {
                data = new
```

```
            {
                Results = results.Select(r => new
                {
                    r.TolerancePercentage,
                    r.Dimension,
                    r.LogM,
                    r.LogDelta,
                    MaxLength = r.MaxLength,
                    r.SequenceGroups
                }).ToList(),
                ExcelFilePath = excelFilePath,
                JsonFilePath = jsonFilePath
            };
        }
        else
        {
            var resultsDto = results.Select(r => new FractalDimensionResultDto
            {
                TolerancePercentage = r.TolerancePercentage,
                Dimension = r.Dimension,
                LogM = r.LogM,
                LogDelta = r.LogDelta,
                MaxLength = r.MaxLength
            }).ToList();
            data = new
            {
                Results = resultsDto,
                ExcelFilePath = excelFilePath,
                JsonFilePath = jsonFilePath
            };
        }
        var settings = new JsonSerializerSettings
        {
            Formatting = Formatting.Indented,
            ReferenceLoopHandling = ReferenceLoopHandling.Ignore
        };
        string json = JsonConvert.SerializeObject(data, settings);
        File.WriteAllText(filePath, json);
        Debug.WriteLine($"SaveFractalDimensionRangeToJson: Збережено {results.Count} результатів у {filePath}, Повне збереження: {isFullSave}, Розмір файлу: {new FileInfo(filePath).Length / 1024.0:F2} KB");
        return true;
    }
    catch (Exception ex)
    {
        Debug.WriteLine($"SaveFractalDimensionRangeToJson: Помилка: {ex.Message}");
        return false;
    }
}

public static bool SaveFractalDimensionRangeToExcel(List<FractalDimensionResult> results, string filePath)
{
    try
    {
        using (var package = new ExcelPackage(new FileInfo(filePath)))
        {
            var summarySheet = package.Workbook.Worksheets.Add("Summary");
            summarySheet.Cells["A1"].Value = "Tolerance (%)";
            summarySheet.Cells["B1"].Value = "Dimension";
            int row = 2;
            foreach (var result in results.OrderBy(r => r.TolerancePercentage))
            {
                summarySheet.Cells[$"A{row}"].Value = result.TolerancePercentage;
                summarySheet.Cells[$"B{row}"].Value = result.Dimension;
                row++;
            }
            var dataSheet = package.Workbook.Worksheets.Add("Data");
            dataSheet.Cells["A1"].Value = "Tolerance (%)";
            dataSheet.Cells["B1"].Value = "Point Index";
            dataSheet.Cells["C1"].Value = "LogM";
            dataSheet.Cells["D1"].Value = "LogDelta";
            row = 2;
            foreach (var result in results.OrderBy(r => r.TolerancePercentage))
            {
                for (int i = 0; i < result.LogM.Count; i++)
                {
                    dataSheet.Cells[$"A{row}"].Value = result.TolerancePercentage;
                    dataSheet.Cells[$"B{row}"].Value = i + 1;
                    dataSheet.Cells[$"C{row}"].Value = result.LogM[i];
                    dataSheet.Cells[$"D{row}"].Value = result.LogDelta[i];
                    row++;
                }
            }
            package.Save();
            return true;
        }
    }
    catch (Exception ex)
```

```
        {
            Debug.WriteLine($"SaveFractalDimensionRangeToExcel:
Помилка: {ex.Message}");
            return false;
        }
    }
}
```

4.1 Текст програми файлу FractalDimensionPlotViewModel.cs

```
using OxyPlot;
using OxyPlot.Axes;
using OxyPlot.Series;
using System.Collections.Generic;
using System.Linq;
using ECSSTS.Data;
using System.Windows;
using System.Windows.Input;
using ECSSTS.Bases;
using ECSSTS.Utilities;
using MathNet.Numerics.Distributions;

namespace ECSSTS.ViewModel
{
    public class FractalDimensionPlotViewModel
    {
        public PlotModel PlotModel { get; set; }
        public double Tolerance { get; set; }
        public int TotalMainSequences { get; set; }
        public int TotalSimilarSequences { get; set; }
        public int TotalSequences { get; set; }
        public double FractalDimension { get; set; }
        public ICommand ExportCommand { get; }
        private readonly ExportMetadata _meta;
        private void Export() => PlotExporter.ExportToJpg(PlotModel,
"Фракталоподібності", _meta);

        public FractalDimensionPlotViewModel(List<double> logM,
List<double> logDelta, List<int> lengths,
            double tolerance, double fractalDimension,
List<SequenceLengthGroup> sequenceLengths, ExportMetadata
meta)
        {
            _meta = meta;
            PlotModel = new PlotModel();
            ExportCommand = new RelayCommand(_ => Export());
            _meta.FractalDimension = fractalDimension;
            _meta.Tolerance = tolerance / meta.MaxY * 100; // % від
maxY



            PlotModel.Axes.Add(new LinearAxis
            {
                Position = AxisPosition.Bottom,
                Title = "log(1/δ)",
                MajorGridlineStyle = LineStyle.Solid,
                MinorGridlineStyle = LineStyle.Dot,
                Minimum = logDelta.Any() ? logDelta.Min() - 0.5 : -0.1,
                Maximum = logDelta.Any() ? logDelta.Max() + 0.5 : 0.1
            });
            PlotModel.Axes.Add(new LinearAxis
            {
                Position = AxisPosition.Left,
                Title = "log(M)",
                MajorGridlineStyle = LineStyle.Solid,
                MinorGridlineStyle = LineStyle.Dot,
                Minimum = logM.Any() ? logM.Min() - 0.5 : 0,
                Maximum = logM.Any() ? logM.Max() + 0.5 : 1
            });

            var series = new ScatterSeries
            {
                Title = "",
                MarkerType = MarkerType.Circle,
                MarkerSize = 5,
                MarkerFill = OxyColors.Green,
                TrackerFormatString = "{Tag} \nlog(1/δ): {2:F2}\nlog(M):
{4:F2}"
            };

            for (int i = 0; i < logM.Count && i < logDelta.Count && i <
lengths.Count; i++)
            {
                if (!double.IsNaN(logM[i]) && !double.IsInfinity(logM[i])
&&
                    !double.IsNaN(logDelta[i]) &&
!double.IsInfinity(logDelta[i]))
                {
                    double mValue = Math.Exp(logM[i]);
                    series.Points.Add(new ScatterPoint(logDelta[i], logM[i])
                    {
                        Tag = $"Довжина (δ):={lengths[i]}\nКількість
(M):={mValue:F0}"
                    });
                }
            }

            PlotModel.Series.Add(series);

            if (sequenceLengths != null && sequenceLengths.Any())
```

```
        {
            TotalMainSequences = sequenceLengths.Sum(g => g.MainSequencesCount);
            TotalSimilarSequences = sequenceLengths.Sum(g => g.SimilarSequencesCount);
            TotalSequences = sequenceLengths.Sum(g => g.TotalSequencesCount);
        }
        else
        {
            var mainVM = (MainViewModel)Application.Current.MainWindow.DataContext;
            if (mainVM != null && mainVM.MatchingSequencesGroups != null)
            {
                var sequenceLengthsFromGroups = mainVM.MatchingSequencesGroups
                    .Where(g => g.Sequence.Count >= (mainVM.IncludeSinglePointSequences ? 1 : 2))
                    .GroupBy(g => g.Sequence.Count)
                    .OrderBy(g => g.Key)
                    .Select(g => new SequenceLengthGroup
                    {
                        Length = g.Key,
                        Sequences = g.ToList()
                    }).ToList();
                TotalMainSequences = sequenceLengthsFromGroups.Sum(g => g.MainSequencesCount);
                TotalSimilarSequences = sequenceLengthsFromGroups.Sum(g => g.SimilarSequencesCount);
                TotalSequences = sequenceLengthsFromGroups.Sum(g => g.TotalSequencesCount);
            }
            else
            {
                TotalMainSequences = 0;
                TotalSimilarSequences = 0;
                TotalSequences = 0;
            }
        }

        Tolerance = tolerance;
        FractalDimension = fractalDimension;
    }
  }
}
```

4.2 Текст програми файлу **FractalDimensionRangeViewModel**.cs

```
using ECSSTS.Bases;
using ECSSTS.Data;
using OxyPlot;
using OxyPlot.Axes;
using OxyPlot.Series;
using System.Collections.ObjectModel;
using ECSSTS.Models;
using System.Diagnostics;
using System.Windows.Input;
using System.Collections.Specialized;
using System.IO;
using System.Windows;
using Microsoft.Win32;
using ECSSTS.Utilities;

namespace ECSSTS.ViewModel
{
  public class FractalDimensionRangeViewModel : ViewModelBase
  {
    private PlotModel _plotModel;
    private ObservableCollection<FractalDimensionResult> _toleranceOptions;
    private ObservableCollection<FractalDimensionResult> _selectedTolerances;
    private bool _isShowAllTolerances;
    private readonly List<FractalDimensionResult> _results;
    private readonly Dictionary<string, FractalDimensionResult> _toleranceToResultMap;
    private readonly double _maxY;
    private string _windowTitle;
    private string _excelFilePath;
    private string _jsonSavePath;
    private string _fractalJsonSavePath;
    private bool _resetSelectionTrigger;
    private bool _isFullSave;
    private readonly ExportMetadata _meta;

    public delegate void FractalJsonPathChangedHandler(string newPath);
    public event FractalJsonPathChangedHandler FractalJsonPathChanged;

    public PlotModel PlotModel
    {
      get => _plotModel;
      set => SetProperty(ref _plotModel, value);
    }

    public ObservableCollection<FractalDimensionResult> ToleranceOptions
    {
      get => _toleranceOptions;
```

```
        set => SetProperty(ref _toleranceOptions, value);
    }

    public ObservableCollection<FractalDimensionResult>
SelectedTolerances
    {
        get => _selectedTolerances;
        set
        {
            if (_selectedTolerances != null)
            {
                _selectedTolerances.CollectionChanged -=
SelectedTolerances_CollectionChanged;
            }
            SetProperty(ref _selectedTolerances, value);
            if (_selectedTolerances != null)
            {
                _selectedTolerances.CollectionChanged +=
SelectedTolerances_CollectionChanged;
            }
            UpdatePlot();
        }
    }

    public bool IsShowAllTolerances
    {
        get => _isShowAllTolerances;
        set
        {
            SetProperty(ref _isShowAllTolerances, value);
            UpdatePlot();
        }
    }

    public bool ResetSelectionTrigger
    {
        get => _resetSelectionTrigger;
        set => SetProperty(ref _resetSelectionTrigger, value);
    }

    public bool IsFullSave
    {
        get => _isFullSave;
        set => SetProperty(ref _isFullSave, value);
    }

    public string WindowTitle
    {
        get => _windowTitle;
        set => SetProperty(ref _windowTitle, value);
    }

    public string ExcelFileName =>
string.IsNullOrEmpty(_excelFilePath) ? "Не вибрано" :
Path.GetFileName(_excelFilePath);
    public string ExcelFilePath =>
string.IsNullOrEmpty(_excelFilePath) ? "Не вибрано" :
_excelFilePath;
    public string JsonFileName =>
string.IsNullOrEmpty(_jsonSavePath) ? "Не вибрано" :
Path.GetFileName(_jsonSavePath);
    public string JsonSavePath =>
string.IsNullOrEmpty(_jsonSavePath) ? "Не вибрано" :
_jsonSavePath;
    public string FractalJsonFileName =>
string.IsNullOrEmpty(_fractalJsonSavePath) ? "Не вибрано" :
Path.GetFileName(_fractalJsonSavePath);
    public string FractalJsonSavePath =>
string.IsNullOrEmpty(_fractalJsonSavePath) ? "Не вибрано" :
_fractalJsonSavePath;

    public ICommand OpenConfidenceIntervalWindowCommand {
get; }
    public ICommand OpenSeparateWindowCommand { get; }
    public ICommand LoadFractalJsonCommand { get; }
    public ICommand SaveFractalJsonCommand { get; }
    public ICommand SaveToExcelCommand { get; }
    public ICommand ResetSelectionCommand { get; }
    public ICommand ResetAxesCommand { get; }
    public ICommand OpenFractalToleranceComparisonCommand
{ get; }
    public ICommand ExportCommand { get; }

    public
FractalDimensionRangeViewModel(List<FractalDimensionResult>
results, double maxY, MainViewModel mainViewModel = null)
    {
        try
        {
            _results = results?.OrderBy(r =>
r.TolerancePercentage).ToList() ?? new
List<FractalDimensionResult>();
            _maxY = maxY;
            _toleranceToResultMap = new Dictionary<string,
FractalDimensionResult>();
            ToleranceOptions = new
ObservableCollection<FractalDimensionResult>();
            SelectedTolerances = new
ObservableCollection<FractalDimensionResult>();
            _excelFilePath = MainSettings.Default.ExcelFilePath;
            _jsonSavePath = MainSettings.Default.JsonFilePath;
            _fractalJsonSavePath =
MainSettings.Default.FractalJsonFilePath;
            _isFullSave = true;
```

```
            PlotModel = new PlotModel();

            if (mainViewModel != null)
            {
                mainViewModel.PropertyChanged += (s, e) =>
                {
                    if (e.PropertyName == nameof(mainViewModel.FractalJsonSavePath))
                    {
                        _fractalJsonSavePath = mainViewModel.FractalJsonSavePath;
                        MainSettings.Default.FractalJsonFilePath = _fractalJsonSavePath;

                        OnPropertyChanged(nameof(FractalJsonSavePath));

                        OnPropertyChanged(nameof(FractalJsonFileName));
                        UpdateWindowTitle();

                        FractalJsonPathChanged?.Invoke(_fractalJsonSavePath);
                        TryAutoLoadFractalJson();
                    }
                };
            }

            TryAutoLoadFractalJson();

            if (_results.Any())
            {
                foreach (var result in _results)
                {
                    if (result == null) continue;
                    string toleranceKey = $"{result.TolerancePercentage:F2}%";
                    ToleranceOptions.Add(result);
                    _toleranceToResultMap[toleranceKey] = result;

                    Debug.WriteLine($"FractalDimensionRangeViewModel: Додано допуск {toleranceKey}, D={result.Dimension:F4}");
                }
            }

            UpdateWindowTitle();
            _isShowAllTolerances = true;

            OpenFractalToleranceComparisonCommand = new RelayCommand(OpenFractalToleranceComparison);
            OpenConfidenceIntervalWindowCommand = new RelayCommand(OpenConfidenceIntervalWindow);
            OpenSeparateWindowCommand = new RelayCommand(OpenSeparateWindow, CanOpenSeparateWindow);
            LoadFractalJsonCommand = new RelayCommand(LoadFractalJsonFile);
            SaveFractalJsonCommand = new RelayCommand(SaveFractalJsonFile);
            SaveToExcelCommand = new RelayCommand(SaveToExcel);
            ResetSelectionCommand = new RelayCommand(ResetSelection);
            ResetAxesCommand = new RelayCommand(ResetAxes);
            ExportCommand = new RelayCommand(_ => Export());

            InitializeAxes();
            UpdatePlot();
        }
        catch (Exception ex)
        {
            Debug.WriteLine($"FractalDimensionRangeViewModel: Помилка в конструкторі: {ex.Message}");
            PlotModel = new PlotModel { Title = "Помилка ініціалізації графіку" };
            WindowTitle = "Помилка ініціалізації";
        }
    }
    private void TryAutoLoadFractalJson()
    {
        if (!string.IsNullOrEmpty(_fractalJsonSavePath) && File.Exists(_fractalJsonSavePath) && !_results.Any())
        {
            Debug.WriteLine($"TryAutoLoadFractalJson: Автозавантаження з {_fractalJsonSavePath}");
            var (loadedResults, excelPath, jsonPath) = LoadData.LoadFractalDimensionRangeFromJson(_fractalJsonSavePath);
            if (loadedResults != null)
            {
                _results.Clear();
                ToleranceOptions.Clear();
                _toleranceToResultMap.Clear();

                foreach (var result in loadedResults)
                {
                    _results.Add(result);
                    string key = $"{result.TolerancePercentage:F2}%";
                    ToleranceOptions.Add(result);
                    _toleranceToResultMap[key] = result;
                }

                if (!string.IsNullOrEmpty(excelPath)) _excelFilePath = excelPath;
                MainSettings.Default.FractalJsonFilePath = _fractalJsonSavePath;
                MainSettings.Default.Save();
```

```
            OnPropertyChanged(nameof(ExcelFileName));
            OnPropertyChanged(nameof(ExcelFilePath));
            OnPropertyChanged(nameof(JsonFileName));
            OnPropertyChanged(nameof(JsonSavePath));
            OnPropertyChanged(nameof(FractalJsonFileName));
            OnPropertyChanged(nameof(FractalJsonSavePath));
            FractalJsonPathChanged?.Invoke(_fractalJsonSavePath);

            InitializeAxes();
            UpdatePlot();
            UpdateWindowTitle();
          }
        }
      }

      private void UpdateWindowTitle()
      {
        string baseTitle = "Порівняння фракталоподібних розмірностей";
        bool hasExcel = !string.IsNullOrEmpty(_excelFilePath) && File.Exists(_excelFilePath);
        bool hasFractalJson = !string.IsNullOrEmpty(_fractalJsonSavePath) && File.Exists(_fractalJsonSavePath);
        if (hasFractalJson && hasExcel)
          WindowTitle = $"{baseTitle}: {FractalJsonFileName}, Ряд: {ExcelFileName}";
        else if (hasFractalJson)
          WindowTitle = $"{baseTitle}: {FractalJsonFileName}";
        else if (hasExcel)
          WindowTitle = $"{baseTitle}: Ряд: {ExcelFileName}";
        else
          WindowTitle = baseTitle;
        Debug.WriteLine($"UpdateWindowTitle: Встановлено заголовок: {WindowTitle}");
      }

      private void SelectedTolerances_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
      {
        Debug.WriteLine($"SelectedTolerances_CollectionChanged: Змінено вибір, елементів: {_selectedTolerances.Count}");
        UpdatePlot();
      }

      private void InitializeAxes()
      {
        if (!_results.Any())
        {
          Debug.WriteLine("InitializeAxes: _results порожній, осі не ініціалізовано");
          return;
        }

        _plotModel.Axes.Clear();
        Debug.WriteLine("InitializeAxes: Старі осі очищено");

        var logMValues = _results.SelectMany(r => r.LogM ?? new List<double>()).ToList();
        var logDeltaValues = _results.SelectMany(r => r.LogDelta ?? new List<double>()).ToList();
        double minLogDelta = logDeltaValues.Any() ? logDeltaValues.Min() - 0.5 : -0.5;
        double maxLogDelta = logDeltaValues.Any() ? logDeltaValues.Max() + 0.5 : 0.5;
        double minLogM = logMValues.Any() ? logMValues.Min() - 0.5 : 0;
        double maxLogM = logMValues.Any() ? logMValues.Max() + 0.5 : 1;

        _plotModel.Axes.Add(new LinearAxis
        {
          Position = AxisPosition.Bottom,
          Title = "log(1/δ)",
          MajorGridlineStyle = LineStyle.Solid,
          MinorGridlineStyle = LineStyle.Dot,
          Minimum = minLogDelta,
          Maximum = maxLogDelta
        });
        _plotModel.Axes.Add(new LinearAxis
        {
          Position = AxisPosition.Left,
          Title = "log(M)",
          MajorGridlineStyle = LineStyle.Solid,
          MinorGridlineStyle = LineStyle.Dot,
          Minimum = minLogM,
          Maximum = maxLogM
        });

        Debug.WriteLine($"InitializeAxes: Осі ініціалізовано, MinLogDelta={minLogDelta:F2}, MaxLogDelta={maxLogDelta:F2}, MinLogM={minLogM:F2}, MaxLogM={maxLogM:F2}");
      }

      private void UpdatePlot()
      {
        if (_plotModel == null)
        {
          Debug.WriteLine("UpdatePlot: PlotModel is null, ініціалізуємо новий");
          _plotModel = new PlotModel();
        }
```

```
        _plotModel.Series.Clear();
        _plotModel.Axes.Clear();
        var colors = new[] { OxyColors.Red, OxyColors.Blue, OxyColors.Green, OxyColors.Purple, OxyColors.Orange, OxyColors.Cyan, OxyColors.Magenta, OxyColors.Yellow, OxyColors.Brown, OxyColors.Lime };

        // Налаштування осей
        double minLogDelta = _results.Any() && _results.Any(r => r.LogDelta != null && r.LogDelta.Any()) ? _results.Min(r => r.LogDelta?.Min() ?? 0) - 0.5 : -0.1;
        double maxLogDelta = _results.Any() && _results.Any(r => r.LogDelta != null && r.LogDelta.Any()) ? _results.Max(r => r.LogDelta?.Max() ?? 0) + 0.5 : 0.1;
        double minLogM = _results.Any() && _results.Any(r => r.LogM != null && r.LogM.Any()) ? _results.Min(r => r.LogM?.Min() ?? 0) - 0.5 : 0;
        double maxLogM = _results.Any() && _results.Any(r => r.LogM != null && r.LogM.Any()) ? _results.Max(r => r.LogM?.Max() ?? 1) + 0.5 : 1;

        _plotModel.Axes.Add(new LinearAxis
        {
            Position = AxisPosition.Bottom,
            Title = "log(1/δ)",
            MajorGridlineStyle = LineStyle.Solid,
            MinorGridlineStyle = LineStyle.Dot,
            Minimum = minLogDelta,
            Maximum = maxLogDelta
        });
        _plotModel.Axes.Add(new LinearAxis
        {
            Position = AxisPosition.Left,
            Title = "log(M)",
            MajorGridlineStyle = LineStyle.Solid,
            MinorGridlineStyle = LineStyle.Dot,
            Minimum = minLogM,
            Maximum = maxLogM
        });

        Debug.WriteLine($"UpdatePlot: Налаштовано oci, MinLogDelta={minLogDelta:F2}, MaxLogDelta={maxLogDelta:F2}, MinLogM={minLogM:F2}, MaxLogM={maxLogM:F2}");

        if (_isShowAllTolerances)
        {
            int colorIndex = 0;
            Debug.WriteLine($"UpdatePlot: Обробка всіх допусків, кількість результатів: {_results.Count}");
            foreach (var result in _results)
            {
                if (result?.LogM == null || result.LogDelta == null)
                {
                    Debug.WriteLine($"UpdatePlot: Пропущено результат для допуску {result?.TolerancePercentage:F2}%, LogM={result?.LogM?.Count ?? 0}, LogDelta={result?.LogDelta?.Count ?? 0}");
                    continue;
                }
                var series = new ScatterSeries
                {
                    Title = $"Допуск {result.TolerancePercentage:F2}% (D={result.Dimension:F4})",
                    MarkerType = MarkerType.Circle,
                    MarkerSize = 5,
                    MarkerFill = colors[colorIndex % colors.Length],
                    TrackerFormatString = "{Tag} \nlog(1/δ): {2:F2}\nlog(M): {4:F2}"
                };
                var lengths = result.Lengths.Any() ? result.Lengths : Enumerable.Range(1, Math.Min(result.LogM.Count, result.LogDelta.Count)).ToList();
                Debug.WriteLine($"UpdatePlot: Обробка допуску {result.TolerancePercentage:F2}%, LogM={result.LogM.Count}, LogDelta={result.LogDelta.Count}, Lengths={lengths.Count}");
                for (int i = 0; i < result.LogM.Count && i < result.LogDelta.Count && i < lengths.Count; i++)
                {
                    if (!double.IsNaN(result.LogM[i]) && !double.IsInfinity(result.LogM[i]) &&
                        !double.IsNaN(result.LogDelta[i]) && !double.IsInfinity(result.LogDelta[i]))
                    {
                        double mValue = Math.Exp(result.LogM[i]);
                        series.Points.Add(new ScatterPoint(result.LogDelta[i], result.LogM[i])
                        {
                            Tag = $"Довжина (δ):={lengths[i]}\nКількість (M):={mValue:F0}"
                        });
                        Debug.WriteLine($"UpdatePlot: Додано точку для допуску {result.TolerancePercentage:F2}%, i={i}: δ={lengths[i]}, M={mValue:F0}, log(1/δ)={result.LogDelta[i]:F2}, log(M)={result.LogM[i]:F2}");
                    }
                    else
                    {
                        Debug.WriteLine($"UpdatePlot: Пропущено точку для допуску {result.TolerancePercentage:F2}%, i={i}: LogM={result.LogM[i]}, LogDelta={result.LogDelta[i]}");
                    }
                }
```

```
            if (series.Points.Any())
            {
                _plotModel.Series.Add(series);
                Debug.WriteLine($"UpdatePlot: Додано серію для допуску {result.TolerancePercentage:F2}% з {series.Points.Count} точками");
            }
            else
            {
                Debug.WriteLine($"UpdatePlot: Серія для допуску {result.TolerancePercentage:F2}% порожня, не додано");
            }
            colorIndex++;
        }
    }
    else
    {
        int colorIndex = 0;
        Debug.WriteLine($"UpdatePlot: Обробка вибраних допусків, кількість: {_selectedTolerances?.Count ?? 0}");
        foreach (var result in _selectedTolerances ?? new ObservableCollection<FractalDimensionResult>())
        {
            if (result?.LogM == null || result.LogDelta == null)
            {
                Debug.WriteLine($"UpdatePlot: Пропущено вибраний результат для допуску {result?.TolerancePercentage:F2}%, LogM={result?.LogM?.Count ?? 0}, LogDelta={result?.LogDelta?.Count ?? 0}");
                continue;
            }
            var series = new ScatterSeries
            {
                Title = $"Допуск {result.TolerancePercentage:F2}% (D={result.Dimension:F4})",
                MarkerType = MarkerType.Circle,
                MarkerSize = 5,
                MarkerFill = colors[colorIndex % colors.Length],
                TrackerFormatString = "{Tag} \nlog(1/δ): {2:F2}\nlog(M): {4:F2}"
            };
            var lengths = result.Lengths.Any() ? result.Lengths : Enumerable.Range(1, Math.Min(result.LogM.Count, result.LogDelta.Count)).ToList();
            Debug.WriteLine($"UpdatePlot: Обробка вибраного допуску {result.TolerancePercentage:F2}%, LogM={result.LogM.Count}, LogDelta={result.LogDelta.Count}, Lengths={lengths.Count}");
            for (int i = 0; i < result.LogM.Count && i < result.LogDelta.Count && i < lengths.Count; i++)
            {
                if (!double.IsNaN(result.LogM[i]) && !double.IsInfinity(result.LogM[i]) &&
                    !double.IsNaN(result.LogDelta[i]) && !double.IsInfinity(result.LogDelta[i]))
                {
                    double mValue = Math.Exp(result.LogM[i]);
                    series.Points.Add(new ScatterPoint(result.LogDelta[i], result.LogM[i])
                    {
                        Tag = $"Довжина (δ):={lengths[i]}\nКількість (M):={mValue:F0}"
                    });
                    Debug.WriteLine($"UpdatePlot: Додано точку для допуску {result.TolerancePercentage:F2}%, i={i}: δ={lengths[i]}, M={mValue:F0}, log(1/δ)={result.LogDelta[i]:F2}, log(M)={result.LogM[i]:F2}");
                }
                else
                {
                    Debug.WriteLine($"UpdatePlot: Пропущено точку для допуску {result.TolerancePercentage:F2}%, i={i}: LogM={result.LogM[i]}, LogDelta={result.LogDelta[i]}");
                }
            }
            if (series.Points.Any())
            {
                _plotModel.Series.Add(series);
                Debug.WriteLine($"UpdatePlot: Додано серію для вибраного допуску {result.TolerancePercentage:F2}% з {series.Points.Count} точками");
            }
            else
            {
                Debug.WriteLine($"UpdatePlot: Серія для вибраного допуску {result.TolerancePercentage:F2}% порожня, не додано");
            }
            colorIndex++;
        }
    }

    _plotModel.InvalidatePlot(true);
    Debug.WriteLine("UpdatePlot: Графік оновлено");
}

private void OpenFractalToleranceComparison(object? _)
{
    if (!_results.Any())
    {
        MessageBox.Show("Немає результатів для порівняння.", "Попередження", MessageBoxButton.OK, MessageBoxImage.Warning);
        return;
```

```
        }
        var meta = new ExportMetadata
        {
            ExcelFileName = Path.GetFileNameWithoutExtension(_excelFilePath) ?? "Ряд",
            FractalJsonName = Path.GetFileName(_fractalJsonSavePath) ?? "Fractal.json"
        };

        var viewModel = new FractalToleranceComparisonViewModel(_results.ToList(), meta);
        var window = new FractalToleranceComparisonWindow(viewModel)
        {
            Title = "Порівняння фракталоподібності від точності"
        };
        window.Show();
    }

    private bool CanOpenSeparateWindow(object parameter)
    {
        return _selectedTolerances?.Any() == true;
    }

    private void OpenSeparateWindow(object parameter)
    {

        foreach (var result in _selectedTolerances ?? new ObservableCollection<FractalDimensionResult>())
        {

            if (result != null)
            {
                var meta = new ExportMetadata
                {
                    ExcelFileName = Path.GetFileNameWithoutExtension(_excelFilePath) ?? "Ряд",
                    FractalJsonName = Path.GetFileName(_fractalJsonSavePath) ?? "Fractal.json",
                    MinTolerance = result.TolerancePercentage,
                    MaxTolerance = result.TolerancePercentage
                };
                var viewModel = new FractalDimensionPlotViewModel(
                    result.LogM ?? new List<double>(),
                    result.LogDelta ?? new List<double>(),
                    result.Lengths.Any() ? result.Lengths : Enumerable.Range(1, result.MaxLength).ToList(),
                    result.TolerancePercentage / 100 * _maxY,
                    result.Dimension,
                    result.SequenceGroups?.ToList() ?? new List<SequenceLengthGroup>(),
                    meta
                );
                var window = new FractalDimensionPlotWindow(viewModel)
                {
                    Title = $"Фрактальна розмірність: {result.Dimension:F4} (Допуск: {result.TolerancePercentage:F2}%)"
                };
                window.Show();
                Debug.WriteLine($"OpenSeparateWindow: Відкрито вікно для допуску {result.TolerancePercentage:F2}%");
            }
        }
    }

    private void OpenConfidenceIntervalWindow(object? _)
    {
        var window = new ConfidenceIntervalWindow();
        window.Show();
    }

    private void LoadFractalJsonFile(object? _)
    {
        string selectedFilePath = LoadData.SelectFractalJsonFile();
        if (string.IsNullOrEmpty(selectedFilePath)) return;

        var (results, excelPath, jsonPath) = LoadData.LoadFractalDimensionRangeFromJson(selectedFilePath);
        if (results != null)
        {
            _results.Clear();
            _results.AddRange(results);

            ToleranceOptions.Clear();
            _toleranceToResultMap.Clear();

            foreach (var result in _results)
            {
                string key = $"{result.TolerancePercentage:F2}%";
                ToleranceOptions.Add(result);
                _toleranceToResultMap[key] = result;
            }

            _fractalJsonSavePath = selectedFilePath;
            if (!string.IsNullOrEmpty(excelPath) && File.Exists(excelPath))
                _excelFilePath = excelPath;
            if (!string.IsNullOrEmpty(jsonPath) && File.Exists(jsonPath))
                _jsonSavePath = jsonPath;
```

```
            MainSettings.Default.FractalJsonFilePath = _fractalJsonSavePath;
            MainSettings.Default.Save();

            OnPropertyChanged(nameof(FractalJsonSavePath));
            OnPropertyChanged(nameof(FractalJsonFileName));
            OnPropertyChanged(nameof(ExcelFilePath));
            OnPropertyChanged(nameof(ExcelFileName));
            OnPropertyChanged(nameof(JsonSavePath));
            OnPropertyChanged(nameof(JsonFileName));

            FractalJsonPathChanged?.Invoke(_fractalJsonSavePath);

            InitializeAxes();
            UpdatePlot();
            UpdateWindowTitle();
            MessageBox.Show($"Fractal JSON успішно завантажено:\n{selectedFilePath}", "Успіх", MessageBoxButton.OK, MessageBoxImage.Information);
        }
        else
        {
            Debug.WriteLine($"LoadFractalJsonFile: Не вдалося завантажити Fractal JSON: {selectedFilePath}");
            MessageBox.Show("Не вдалося завантажити Fractal JSON.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }

    private void SaveFractalJsonFile(object? _)
    {
        string filePath = SaveData.SelectSaveFractalJsonFile(_excelFilePath);
        if (string.IsNullOrEmpty(filePath))
            return;

        if (SaveData.SaveFractalDimensionRangeToJson(_results, filePath, _excelFilePath, _jsonSavePath, _isFullSave))
        {
            _fractalJsonSavePath = filePath;
            MainSettings.Default.FractalJsonFilePath = filePath;
            MainSettings.Default.Save();
            UpdateWindowTitle();
            OnPropertyChanged(nameof(FractalJsonFileName));
            OnPropertyChanged(nameof(FractalJsonSavePath));
            // Сповіщаємо про зміну FractalJsonSavePath
            FractalJsonPathChanged?.Invoke(_fractalJsonSavePath);
            MessageBox.Show($"Fractal JSON успішно збережено:\n{filePath}", "Успіх", MessageBoxButton.OK, MessageBoxImage.Information);
        }
        else
        {
            MessageBox.Show("Не вдалося зберегти Fractal JSON.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }

    private void SaveToExcel(object? _)
    {
        if (_results.Count == 0) return;
        double minTol = _results.Min(r => r.TolerancePercentage);
        double maxTol = _results.Max(r => r.TolerancePercentage);
        double minDim = _results.Min(r => r.Dimension);
        double maxDim = _results.Max(r => r.Dimension);
        string defaultName = string.IsNullOrEmpty(_excelFilePath) ? "Results" : Path.GetFileNameWithoutExtension(_excelFilePath);
        string suggestedName = $"{defaultName} Tolerance {minTol:F2}-{maxTol:F2} Dimension {minDim:F4}-{maxDim:F4}.xlsx";
        var dialog = new SaveFileDialog
        {
            Filter = "Excel files (*.xlsx)|*.xlsx|All files (*.*)|*.*",
            FileName = suggestedName,
            InitialDirectory = string.IsNullOrEmpty(_excelFilePath) ? Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) : Path.GetDirectoryName(_excelFilePath)
        };
        if (dialog.ShowDialog() == true)
        {
            if (SaveData.SaveFractalDimensionRangeToExcel(_results, dialog.FileName))
            {
                MessageBox.Show($"Експортовано до Excel: {dialog.FileName}", "Успіх", MessageBoxButton.OK, MessageBoxImage.Information);
            }
            else
            {
                MessageBox.Show("Не вдалося експортувати до Excel.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
    }

    private void ResetSelection(object? _)
    {
        SelectedTolerances.Clear();
        ResetSelectionTrigger = !ResetSelectionTrigger;
        Debug.WriteLine("ResetSelection: Виділення скинуто");
        UpdatePlot();
```

```
    }

    private void ResetAxes(object? _)
    {
        _plotModel.ResetAllAxes();
        _plotModel.InvalidatePlot(true);
        Debug.WriteLine("ResetAxes: Вісі графіка скинуто до початкового стану");
    }

    private void Export()
    {
        var meta = new ExportMetadata
        {
            ExcelFileName = Path.GetFileNameWithoutExtension(_excelFilePath) ?? "Ряд",
            FractalJsonName = Path.GetFileName(_fractalJsonSavePath) ?? "Fractal.json",
            MinTolerance = _results.Min(r => r.TolerancePercentage),
            MaxTolerance = _results.Max(r => r.TolerancePercentage)
        };

        PlotExporter.ExportToJpg(PlotModel, "Порівняння фракталоподібностей", meta);
    }
    public void UpdateResults(List<FractalDimensionResult> newResults)
    {
        _results.Clear();
        _results.AddRange(newResults.OrderBy(r => r.TolerancePercentage));

        ToleranceOptions.Clear();
        _toleranceToResultMap.Clear();

        foreach (var r in _results)
        {
            ToleranceOptions.Add(r);
            _toleranceToResultMap[$"{r.TolerancePercentage:F2}%"] = r;
        }

        InitializeAxes();
        UpdatePlot();
        UpdateWindowTitle();
    }
  }
}
```

4.3 Текст програми файлу MainViewModel.cs

```
using ECSSTS.Bases;
using ECSSTS.Data;
using ECSSTS.Processing;
using ECSSTS.ViewGraf;
using ECSSTS.Models;
using ECSSTS.Utilities;
using OxyPlot;
using OxyPlot.Series;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.IO;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media;
using ECSSTS.ViewModel;
using System.Threading;
using System;
using System.Linq;
using Newtonsoft.Json;
using System.Windows.Threading;

namespace ECSSTS
{
  public class MainViewModel : ViewModelBase
  {
    private string _windowTitle = "ECSSTS";
    private string _currentFileName;
    private FractalDimensionRangeWindow _currentFractalWindow;
    private string _jsonSavePath;
    private string _fractalJsonSavePath;
    private string _excelFilePath;
    private string _saveStatus;
    private Brush _saveStatusColor;
    private bool _canSave;
    private bool _isAutoSaveEnabled;
    private bool _isGraphDeleted;
    private bool _isDataListBoxVisible;
    private PlotModel _plotModel;
    private ObservableCollection<DataExcel> _dataPoints;
    private ObservableCollection<DataExcel> _selectedPoints;
    private ObservableCollection<MatchingSequenceGroup> _matchingSequencesGroups;
    private MatchingSequenceGroup _selectedSequence;
    private ObservableCollection<SequenceLengthGroup> _sequenceLengths;
    private double _tolerance;
    private double _tolerancePercentage = 0.1;
    private double _maxY = 1.0;
    private bool _excludeInverted = true;
```

```
        public bool _isLoadingJson;
        public bool _isInitializing;
        private double _searchProgress;
        private bool _isSearchButtonEnabled = true;
        private bool _includeSinglePointSequences = false;
        private bool _isRelativeTolerance;
        private double _fractalDimension;
        private bool _includeNonTolerance;
        private bool _showAllSequences;
        private readonly ViewPlotModel _plotModelInstance;
        private readonly Graf _mainWindowGraf;
        private TimeSpan _estimatedTimeRemaining;
        private CancellationTokenSource _cancellationTokenSource;
        public ObservableCollection<FractalDimensionResult>
_fractalResults;
        public bool HasFractalResults => _fractalResults.Any();
        public bool HasSavedFractalJson =>
!string.IsNullOrEmpty(MainSettings.Default.FractalJsonFilePath)
&& File.Exists(MainSettings.Default.FractalJsonFilePath);
        public ICommand CommandSearch { get; }
        public ICommand CommandClear { get; }
        public ICommand CommandDeleteGraph { get; }
        public ICommand CommandOpenFile { get; }
        public ICommand CommandLoadJson { get; }
        public ICommand CommandSaveJson { get; }
        public ICommand
CommandCalculateFractalDimensionBySequence { get; }
        public ICommand ToggleExcludeInvertedCommand { get; }
        public ICommand ToggleIncludeNonToleranceCommand { get;
}
        public ICommand ToggleDataListBoxCommand { get; }
        public ICommand CommandResetJsonPath { get; }
        public ICommand CommandCancelSearch { get; }
        public ICommand CommandResetAxes { get; }
        public ICommand
ToggleIncludeSinglePointSequencesCommand { get; }
        public ICommand CommandCalculateFractalDimension { get; }
        public ICommand CommandOpenFractalGraph { get; }
        public ICommand ExportCommand { get; }
        public ICommand
OpenOrUpdateCurrentFractalWindowCommand { get; }
        public ICommand OpenSavedFractalWindowCommand { get; }
        public MainViewModel()
        {
            _isInitializing = true;
            _plotModelInstance = new
ViewPlotModel(ViewPlotModel.PlotSelect.Main);
            _mainWindowGraf = new Graf();
            _plotModel = _plotModelInstance;
            _dataPoints = App.data;
            _selectedPoints = new ObservableCollection<DataExcel>();
            _sequenceLengths = new
ObservableCollection<SequenceLengthGroup>();
            _matchingSequencesGroups = App.sequences;
            _fractalResults = new
ObservableCollection<FractalDimensionResult>();
            _fractalDimension = App.fractaldimension;
            _includeSinglePointSequences =
MainSettings.Default.IncludeSinglePointSequences;
            _excelFilePath = MainSettings.Default.ExcelFilePath;
            _jsonSavePath = MainSettings.Default.JsonFilePath;
            _fractalJsonSavePath =
MainSettings.Default.FractalJsonFilePath;
            UpdateWindowTitle();
            SaveStatus = $"Шлях: {_jsonSavePath ??
SaveData.DefaultFilePath}";
            SaveStatusColor = string.IsNullOrEmpty(_jsonSavePath) ?
Brushes.Red : Brushes.Green;
            CanSave = !string.IsNullOrEmpty(_fractalJsonSavePath) &&
File.Exists(_fractalJsonSavePath)                ||
!string.IsNullOrEmpty(_jsonSavePath)                &&
File.Exists(_jsonSavePath) || _matchingSequencesGroups.Any();  //
ВИПРАВЛЕНО: Видалено *
            _isDataListBoxVisible =
MainSettings.Default.IsDataListBoxVisible;
            _isAutoSaveEnabled =
MainSettings.Default.IsAutoSaveEnabled;
            FractalDimension = _fractalDimension;
            OnPropertyChanged(nameof(FractalDimension));

            CommandSearch = new RelayCommand(async _ => await
StartSeach(_));
            CommandClear = new RelayCommand(ClearGraf);
            CommandDeleteGraph = new RelayCommand(DeleteGraph);
            CommandOpenFile = new RelayCommand(OpenFile);
            CommandLoadJson = new RelayCommand(LoadJsonFile);
            CommandSaveJson = new RelayCommand(SaveJsonFile);
            CommandResetJsonPath = new
RelayCommand(ResetJsonPath);
            CommandCalculateFractalDimensionBySequence = new
RelayCommand(CalculateFractalDimensionBySequence);
            ToggleExcludeInvertedCommand = new RelayCommand(_
=> ExcludeInverted = !ExcludeInverted);
            ToggleIncludeNonToleranceCommand = new
RelayCommand(_ => IncludeNonTolerance =
!IncludeNonTolerance);
            ToggleDataListBoxCommand = new
RelayCommand(ToggleDataListBox);
            CommandCancelSearch = new
RelayCommand(CancelSearch);
            CommandResetAxes = new RelayCommand(ResetAxes);
```

```
        ToggleIncludeSinglePointSequencesCommand = new
RelayCommand(_ => IncludeSinglePointSequences =
!IncludeSinglePointSequences);
        CommandCalculateFractalDimension = new
RelayCommand(CalculateFractalDimension);
        CommandOpenFractalGraph = new
RelayCommand(OpenFractalGraph);
        ExportCommand = new RelayCommand(_ => Export());
        OpenOrUpdateCurrentFractalWindowCommand = new
RelayCommand(OpenOrUpdateCurrentFractalWindow, _ =>
HasFractalResults);
        OpenSavedFractalWindowCommand = new
RelayCommand(OpenSavedFractalWindow);

        OnPropertyChanged(nameof(FractalJsonFileName));
        OnPropertyChanged(nameof(FractalJsonSavePath));
        OnPropertyChanged(nameof(JsonFileName));
        OnPropertyChanged(nameof(JsonSavePath));
        OnPropertyChanged(nameof(ExcelFileName));
        OnPropertyChanged(nameof(ExcelFilePath));

        UpdateMaxY();
        _tolerance = _tolerancePercentage / 100 * _maxY;
        UpdateSequenceLengths();
        UpdateGraph();
        _plotModel.InvalidatePlot(true);

Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Bac
kground, new Action(() =>
        {
            OnPropertyChanged(nameof(TolerancePercentage));
            OnPropertyChanged(nameof(Tolerance));
            OnPropertyChanged(nameof(FractalDimension));
            _isInitializing = false;
        }));
    }
    private void UpdateWindowTitle()
    {
        bool hasExcel = !string.IsNullOrEmpty(_excelFilePath) &&
File.Exists(_excelFilePath);
        bool hasJson = !string.IsNullOrEmpty(_jsonSavePath) &&
File.Exists(_jsonSavePath);
        bool hasFractalJson =
!string.IsNullOrEmpty(_fractalJsonSavePath) &&
File.Exists(_fractalJsonSavePath);
        if (hasExcel && hasJson)
            WindowTitle = $"ECSSTS - Ряд:
{Path.GetFileName(_excelFilePath)}, Налаштування:
{Path.GetFileName(_jsonSavePath)}";
        else if (hasExcel)
            WindowTitle = $"ECSSTS - Ряд:
{Path.GetFileName(_excelFilePath)}";
        else if (hasJson)
            WindowTitle = $"ECSSTS - Налаштування:
{Path.GetFileName(_jsonSavePath)}";
        else if (hasFractalJson)
            WindowTitle = $"ECSSTS - Fractal JSON:
{Path.GetFileName(_fractalJsonSavePath)}";
        else
            WindowTitle = "ECSSTS - Experimental Computational
Studies of Self-Similarity of Time Series";
    }
    public string WindowTitle
    {
        get => _windowTitle;
        set => SetProperty(ref _windowTitle, value);
    }
    public string ExcelFileName =>
string.IsNullOrEmpty(_excelFilePath) ? "Не вибрано" :
Path.GetFileName(_excelFilePath);
    public string ExcelFilePath =>
string.IsNullOrEmpty(_excelFilePath) ? "Не вибрано" :
_excelFilePath;
    public string JsonFileName =>
string.IsNullOrEmpty(_jsonSavePath) ? "Не вибрано" :
Path.GetFileName(_jsonSavePath);
    public string JsonSavePath =>
string.IsNullOrEmpty(_jsonSavePath) ? "Не вибрано" :
_jsonSavePath;
    public string FractalJsonFileName =>
string.IsNullOrEmpty(_fractalJsonSavePath) ? "Не вибрано" :
Path.GetFileName(_fractalJsonSavePath);
    public string FractalJsonSavePath =>
string.IsNullOrEmpty(_fractalJsonSavePath) ? "Не вибрано" :
_fractalJsonSavePath;
    public bool CanSave
    {
        get => _canSave;
        set => SetProperty(ref _canSave, value, nameof(CanSave),
nameof(CanSaveToCurrentPath));
    }
    public bool CanSaveToCurrentPath => _canSave &&
!string.IsNullOrEmpty(_jsonSavePath);
    public bool IsAutoSaveEnabled
    {
        get => _isAutoSaveEnabled;
        set
        {
            SetProperty(ref _isAutoSaveEnabled, value);
            MainSettings.Default.IsAutoSaveEnabled = value;
            MainSettings.Default.Save();
        }
    }
    public bool IsRelativeTolerance
```

```
        {
            get => _isRelativeTolerance;
            set => SetProperty(ref _isRelativeTolerance, value);
        }
        public bool IncludeSinglePointSequences
        {
            get => _includeSinglePointSequences;
            set
            {
                SetProperty(ref _includeSinglePointSequences, value);
                MainSettings.Default.IncludeSinglePointSequences = value;
                MainSettings.Default.Save();
            }
        }
        public string SaveStatus
        {
            get => _saveStatus;
            set => SetProperty(ref _saveStatus, value);
        }
        public Brush SaveStatusColor
        {
            get => _saveStatusColor;
            set => SetProperty(ref _saveStatusColor, value);
        }
        public double Tolerance
        {
            get => _tolerance;
            set
            {
                if (_tolerance != value)
                {
                    _tolerance = value;
                    _tolerancePercentage = _tolerance / _maxY * 100;
                    OnPropertyChanged(nameof(Tolerance));
                    OnPropertyChanged(nameof(TolerancePercentage));
                }
            }
        }
        public double TolerancePercentage
        {
            get => _tolerancePercentage;
            set
            {
                if (_tolerancePercentage != value && !double.IsNaN(value) && !double.IsInfinity(value))
                {
                    _tolerancePercentage = value;
                    _tolerance = _tolerancePercentage / 100 * _maxY;
                    OnPropertyChanged(nameof(TolerancePercentage));
                    OnPropertyChanged(nameof(Tolerance));
                }
            }
        }
        public ObservableCollection<DataExcel> DataPoints
        {
            get => _dataPoints;
            set
            {
                SetProperty(ref _dataPoints, value);
                UpdateGraph();
                UpdateMaxY();
                OnPropertyChanged(nameof(TotalSequencesCount));
            }
        }
        public ObservableCollection<SequenceLengthGroup> SequenceLengths
        {
            get => _sequenceLengths;
            set => SetProperty(ref _sequenceLengths, value, nameof(SequenceLengths), nameof(TotalSequencesCount));
        }
        public ObservableCollection<MatchingSequenceGroup> MatchingSequencesGroups
        {
            get => _matchingSequencesGroups;
            set
            {
                SetProperty(ref _matchingSequencesGroups, value ?? new ObservableCollection<MatchingSequenceGroup>());
                UpdateSequenceLengths();
                OnPropertyChanged(nameof(TotalSequencesCount));
            }
        }
        public MatchingSequenceGroup SelectedSequence
        {
            get => _selectedSequence;
            set
            {
                SetProperty(ref _selectedSequence, value);
                SelectedPoints.Clear();
                UpdateGraph();
            }
        }
        public int TotalSequencesCount => SequenceLengths.Sum(group => group.TotalSequencesCount);
        public PlotModel PlotModel
        {
            get => _plotModel;
            set => SetProperty(ref _plotModel, value);
        }
        public ObservableCollection<DataExcel> SelectedPoints
        {
            get => _selectedPoints;
```

```
        set
        {
            SetProperty(ref _selectedPoints, value);
            UpdateGraph();
        }
    }
    public double SearchProgress
    {
        get => _searchProgress;
        set
        {
            if (SetProperty(ref _searchProgress, value,
nameof(SearchProgress), nameof(IsSearchButtonEnabled),
nameof(SearchButtonContent)))
            {
                UpdateTaskbarProgress();
            }
        }
    }
    public bool IsSearchButtonEnabled
    {
        get => _isSearchButtonEnabled;
        set => SetProperty(ref _isSearchButtonEnabled, value);
    }
    public string SearchButtonContent => _searchProgress > 0 ?
$"Пошук: {_searchProgress:F0}%" : "Пошук";
    public double FractalDimension
    {
        get => _fractalDimension;
        set
        {
            if (SetProperty(ref _fractalDimension, value))
            {
                App.fractaldimension = value;
                Debug.WriteLine($"FractalDimension setter: Оновлено
до {value:F4}, App.fractaldimension = {App.fractaldimension:F4}");
            }
        }
    }
    public bool IncludeNonTolerance
    {
        get => _includeNonTolerance;
        set
        {
            SetProperty(ref _includeNonTolerance, value);
            UpdateGraph();
        }
    }
    public bool ShowAllSequences
    {
        get => _showAllSequences;
        set
        {
            SetProperty(ref _showAllSequences, value);
            UpdateGraph();
        }
    }
    public bool ExcludeInverted
    {
        get => _excludeInverted;
        set
        {
            SetProperty(ref _excludeInverted, value);
            UpdateGraph();
        }
    }
    public bool IsDataListBoxVisible
    {
        get => _isDataListBoxVisible;
        set
        {
            SetProperty(ref _isDataListBoxVisible, value,
nameof(IsDataListBoxVisible), nameof(DataListBoxToggleText));
            MainSettings.Default.IsDataListBoxVisible = value;
            MainSettings.Default.Save();
        }
    }
    public string DataListBoxToggleText => _isDataListBoxVisible
? "Приховати Список точок" : "Показати Список точок";
    public TimeSpan EstimatedTimeRemaining
    {
        get => _estimatedTimeRemaining;
        set
        {
            _estimatedTimeRemaining = value;
            OnPropertyChanged(nameof(EstimatedTimeRemaining));
        }
    }
    public double MaxY => _maxY;
    public CancellationTokenSource CancellationTokenSource
    {
        get => _cancellationTokenSource;
        private set
        {
            _cancellationTokenSource = value;
            OnPropertyChanged();
        }
    }
    public void LoadExcelSilently(string filePath)
    {
        if (string.IsNullOrEmpty(filePath) || !File.Exists(filePath))
return;

        App.data.Clear();
```

```
    var newData = LoadData.LoadDataFromExcel(filePath);
    if (newData?.Count > 0)
    {
        foreach (var item in newData) App.data.Add(item);
    }
    _excelFilePath = filePath;
    MainSettings.Default.ExcelFilePath = filePath;
    MainSettings.Default.Save();

    UpdateWindowTitle();
    UpdateGraph();
    UpdateMaxY();
    _plotModel.InvalidatePlot(true);
    OnPropertyChanged(nameof(TotalSequencesCount));
    OnPropertyChanged(nameof(ExcelFileName));
    OnPropertyChanged(nameof(ExcelFilePath));
}

public void OpenFile(object? parameter)
{
    string filePath = parameter as string;
    if (string.IsNullOrEmpty(filePath))
    {
        filePath = LoadData.SelectExcelFile();
        if (string.IsNullOrEmpty(filePath)) return;
    }

    // Повне очищення стану
    App.data.Clear();
    App.sequences.Clear();
    _fractalResults.Clear();
    NotifyFractalResultsChanged();
    SequenceLengths.Clear();
    MatchingSequencesGroups.Clear();
    SelectedPoints.Clear();

    var newData = LoadData.LoadDataFromExcel(filePath);
    if (newData?.Count > 0)
    {
        foreach (var item in newData) App.data.Add(item);
    }

    _excelFilePath = filePath;
    _jsonSavePath = null;
    MainSettings.Default.ExcelFilePath = filePath;
    MainSettings.Default.JsonFilePath = null;
    MainSettings.Default.Save();

    SaveStatus = $"Шлях: {SaveData.DefaultFilePath}";
    SaveStatusColor = Brushes.Red;
    CanSave = true;
    _isGraphDeleted = false;
    FractalDimension = 0;

    UpdateWindowTitle();
    UpdateGraph();
    UpdateMaxY();
    _plotModel.InvalidatePlot(true);
    OnPropertyChanged(nameof(TotalSequencesCount));
    OnPropertyChanged(nameof(ExcelFileName));
    OnPropertyChanged(nameof(ExcelFilePath));
    OnPropertyChanged(nameof(JsonFileName));
    OnPropertyChanged(nameof(JsonSavePath));
    OnPropertyChanged(nameof(FractalJsonFileName));
    OnPropertyChanged(nameof(FractalJsonSavePath));
}
public void LoadJsonFile(object? parameter)
{
    string selectedFilePath = parameter as string;
    if (string.IsNullOrEmpty(selectedFilePath) ||
!File.Exists(selectedFilePath))
    {
        selectedFilePath = LoadData.SelectJsonFile();
        if (string.IsNullOrEmpty(selectedFilePath)) return;
    }
    _isLoadingJson = true;
    var (loadedSequences, excelFilePath, fractalDimension,
fractalResult) = LoadData.LoadFromJson(selectedFilePath);
    if (loadedSequences?.Any() != true) { _isLoadingJson = false;
return; }
    _fractalResults.Clear();
    NotifyFractalResultsChanged();
    MatchingSequencesGroups = loadedSequences;
    App.sequences.Clear();
    foreach (var seq in loadedSequences)
App.sequences.Add(seq);
    SelectedPoints.Clear();
    _currentFileName = Path.GetFileName(selectedFilePath);
    _jsonSavePath = selectedFilePath;
    MainSettings.Default.JsonFilePath = _jsonSavePath;
    MainSettings.Default.Save();
    SaveStatus = $"Шлях: {_jsonSavePath}";
    SaveStatusColor = Brushes.Green;
    CanSave = true;
    _isGraphDeleted = false;
    var jsonContent = File.ReadAllText(selectedFilePath);
    var jsonData =
JsonConvert.DeserializeObject<FileJson.JsonData>(jsonContent);
    double previousTolerancePercentage = _tolerancePercentage;
    if (fractalResult != null)
    {
        _fractalResults.Add(fractalResult);
        FractalDimension = fractalResult.Dimension;  // ЗМІНА:
Через сеттер
```

```
            Debug.WriteLine($"LoadJsonFile: Завантажено фрактальний результат: D={FractalDimension:F4}");  // ЗМІНА: Перенесено лог сюди
            _tolerancePercentage = fractalResult.TolerancePercentage;
        }
        else if (jsonData?.FractalData?.TolerancePercentage > 0)
            _tolerancePercentage = jsonData.FractalData.TolerancePercentage;
        else if (jsonData?.TolerancePercentage > 0)
            _tolerancePercentage = jsonData.TolerancePercentage;
        if (fractalResult != null)
            FractalDimension = fractalResult.Dimension; // Зберігаємо (дублікат, але OK)
        else
            FractalDimension = fractalDimension;  // ЗМІНА: Через сеттер – синхронізує App
        if (!string.IsNullOrEmpty(excelFilePath) && File.Exists(excelFilePath))
        {
            _excelFilePath = excelFilePath;
            MainSettings.Default.ExcelFilePath = excelFilePath;
            MainSettings.Default.Save();
            App.data.Clear();
            var newData = LoadData.LoadDataFromExcel(excelFilePath);
            if (newData?.Count > 0)
                foreach (var item in newData) App.data.Add(item);
            UpdateMaxY();
        }
        _tolerance = _tolerancePercentage / 100 * _maxY;
        Debug.WriteLine($"LoadJsonFile: Фінальне D={FractalDimension:F4}, App.fractaldimension={App.fractaldimension:F4}");

        Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Background, new Action(() =>
        {
            OnPropertyChanged(nameof(TolerancePercentage));
            OnPropertyChanged(nameof(Tolerance));
            OnPropertyChanged(nameof(DataPoints));
            OnPropertyChanged(nameof(ExcelFileName));
            OnPropertyChanged(nameof(FractalDimension));  // Гарантовано оновлення UI
            OnPropertyChanged(nameof(ExcelFilePath));
            OnPropertyChanged(nameof(TotalSequencesCount));
            OnPropertyChanged(nameof(JsonFileName));
            OnPropertyChanged(nameof(JsonSavePath));
            OnPropertyChanged(nameof(FractalJsonFileName));
            OnPropertyChanged(nameof(FractalJsonSavePath));
        }));
        UpdateWindowTitle();
        UpdateGraph();
        _isLoadingJson = false;
    }
    public void LoadFractalJsonFile(string path)
    {
        if (!File.Exists(path)) return;
        var rangeVM = new FractalDimensionRangeViewModel(new List<FractalDimensionResult>(), 1.0, this);
        rangeVM.FractalJsonPathChanged += OnFractalJsonPathChanged;
        var method = rangeVM.GetType().GetMethod("LoadFractalJsonFile", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance);
        method?.Invoke(rangeVM, new object?[] { null });
        var window = new FractalDimensionRangeWindow(rangeVM);
        window.Show();
    }
    private void SaveJsonFile(object parameter)
    {
        string mode = parameter as string;
        string filePath = mode switch
        {
            "default" => SaveData.DefaultFilePath,
            "current" => _jsonSavePath ?? SaveData.SelectSaveJsonFile(_excelFilePath),
            "as" => SaveData.SelectSaveJsonFile(_excelFilePath),
            _ => null
        };
        if (string.IsNullOrEmpty(filePath)) return;
        if (SaveData.SaveToJson(MatchingSequencesGroups, filePath, _excelFilePath, FractalDimension, TolerancePercentage))
        {
            _jsonSavePath = filePath;
            MainSettings.Default.JsonFilePath = filePath;
            MainSettings.Default.Save();
            SaveStatus = $"Шлях: {_jsonSavePath}";
            SaveStatusColor = Brushes.Green;
            CanSave = false;
            MessageBox.Show($"Файл успішно збережено за шляхом:\n{_jsonSavePath}", "Збереження успішне", MessageBoxButton.OK, MessageBoxImage.Information);
        }
        UpdateWindowTitle();
        OnPropertyChanged(nameof(JsonFileName));
        OnPropertyChanged(nameof(JsonSavePath));
    }
    private void NotifyFractalResultsChanged()
    {
        OnPropertyChanged(nameof(HasFractalResults));
        CommandManager.InvalidateRequerySuggested();
    }
```

```
    private void NotifySavedFractalPathChanged()
    {
        OnPropertyChanged(nameof(HasSavedFractalJson));
        CommandManager.InvalidateRequerySuggested();
    }
    private void ClearGraf(object? _)
    {
        PlotModel = _plotModelInstance.ClearGrafFromDot(PlotModel, _mainWindowGraf);
        SelectedSequence = null;
        SelectedPoints.Clear();
        _isGraphDeleted = false;
        UpdateGraph();
    }
    private void DeleteGraph(object? _)
    {
        _isGraphDeleted = true;
        UpdateGraph();
        UpdateMaxY();
    }
    private void ResetAxes(object? _)
    {
        PlotModel.ResetAllAxes();
        PlotModel.InvalidatePlot(true);
    }
    public void InitializeCancellationToken()
    {
        CancellationTokenSource?.Dispose();
        CancellationTokenSource = new CancellationTokenSource();
    }
    private void CancelSearch(object parameter)
    {
        CancellationTokenSource?.Cancel();
    }
    public void ClearCancellationToken()
    {
        CancellationTokenSource?.Dispose();
        CancellationTokenSource = null;
    }
    private async Task StartSeach(object parameter)
    {
        if (DataPoints == null || !DataPoints.Any()) { MessageBox.Show("Дані не завантажено.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error); return; }
        IsSearchButtonEnabled = false;
        InitializeCancellationToken();
        double savedTolerancePercentage = TolerancePercentage;
        double savedMaxY = _maxY;
        try
        {
            var stopwatch = Stopwatch.StartNew();
            var progress = new Progress<double>(value =>
            {
                SearchProgress = value;
                if (value > 0 && value < 100)
                {
                    double elapsedSeconds = stopwatch.Elapsed.TotalSeconds;
                    double estimatedTotalSeconds = elapsedSeconds / (value / 100);
                    double remainingSeconds = Math.Max(0, estimatedTotalSeconds - elapsedSeconds);
                    EstimatedTimeRemaining = TimeSpan.FromSeconds(remainingSeconds);
                }
            });
            var timeProgress = new Progress<TimeSpan>(time => { if (time == TimeSpan.Zero) EstimatedTimeRemaining = TimeSpan.Zero; });
            int minLength = IncludeSinglePointSequences ? 1 : 2;
            List<MatchingSequenceGroup> searchResult;
            if (IsRelativeTolerance)
                searchResult = await Calculation.SearchRelativeAsync(DataPoints, Tolerance, progress, timeProgress, CancellationTokenSource.Token, minLength);
            else
                searchResult = await Calculation.SearchAsync(DataPoints, Tolerance, progress, timeProgress, CancellationTokenSource.Token, minLength);
            if (CancellationTokenSource.Token.IsCancellationRequested)
            {
                SearchProgress = 0;
                EstimatedTimeRemaining = TimeSpan.Zero;
                SaveStatus = $"Шлях: {_jsonSavePath ?? SaveData.DefaultFilePath}";
                SaveStatusColor = string.IsNullOrEmpty(_jsonSavePath) ? Brushes.Red : Brushes.Green;
                MessageBox.Show("Пошук скасовано.", "Скасування", MessageBoxButton.OK, MessageBoxImage.Information);
                return;
            }
            if (searchResult != null && searchResult.Any())
            {
                MatchingSequencesGroups = new ObservableCollection<MatchingSequenceGroup>(searchResult);
                foreach (var group in MatchingSequencesGroups) group.ToleranceUsed = Tolerance;
                var sequenceLengths = MatchingSequencesGroups
                    .Where(g => g.Sequence.Count >= minLength)
                    .GroupBy(g => g.Sequence.Count)
                    .OrderBy(g => g.Key)
```

```
                .Select(g => new SequenceLengthGroup { Length = g.Key, Sequences = g.ToList() })
                .ToList();
            var (dimension, logM, logDelta, maxLength, lengths) = Calculation.CalculateFractalDimensionBySequence(sequenceLengths);
            if (dimension > 0)
            {
                FractalDimension = dimension;
                var fractalResult = new FractalDimensionResult
                {
                    TolerancePercentage = savedTolerancePercentage,
                    Dimension = dimension,
                    LogM = logM,
                    LogDelta = logDelta,
                    MaxLength = maxLength,
                    Lengths = lengths,
                    SequenceGroups = new ObservableCollection<SequenceLengthGroup>(sequenceLengths)
                };
                _fractalResults.Add(fractalResult);
                NotifyFractalResultsChanged();
                if (_currentFractalWindow?.IsLoaded == true)
                {
                    var vm = _currentFractalWindow.DataContext as FractalDimensionRangeViewModel;
                    vm?.UpdateResults(_fractalResults.ToList());
                }
                var meta = new ExportMetadata { ExcelFileName = Path.GetFileNameWithoutExtension(_excelFilePath) ?? "Ряд", MaxY = _maxY };
                var viewModel = new FractalDimensionPlotViewModel(logM, logDelta, lengths, Tolerance, dimension, sequenceLengths, meta);
                var window = new FractalDimensionPlotWindow(viewModel)
                {
                    Title = $"Фрактальна розмірність: {dimension:F4} (Допуск: {savedTolerancePercentage:F2}%{(IsRelativeTolerance ? ", Відносний" : ", Точний")})"
                };
                window.Show();
            }
            if (IsAutoSaveEnabled) SaveJsonFile("default");
            int totalMainSequences = MatchingSequencesGroups.Count;
            int totalSequences = sequenceLengths.Sum(group => group.TotalSequencesCount);
            stopwatch.Stop();
            SaveStatus = $"Шлях: {_jsonSavePath ?? SaveData.DefaultFilePath}";
            SaveStatusColor = string.IsNullOrEmpty(_jsonSavePath) ? Brushes.Red : Brushes.Green;
            CanSave = true;
            _tolerancePercentage = savedTolerancePercentage;
            _maxY = savedMaxY;
            _tolerance = _tolerancePercentage / 100 * _maxY;

            Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Background, new Action(() =>
            {
                OnPropertyChanged(nameof(TolerancePercentage));
                OnPropertyChanged(nameof(Tolerance));
                OnPropertyChanged(nameof(FractalDimension));
            }));
            MessageBox.Show(
                $"Пошук завершено.\nЗнайдено {totalMainSequences} основних послідовностей\nЗагальна кількість послідовностей: {totalSequences}\nПошук зайняв: {stopwatch.Elapsed:hh\\:mm\\:ss\\.fff}\nРежим: {(IsRelativeTolerance ? "Відносний допуск" : "Точний допуск")}",
                "Результат",
                MessageBoxButton.OK,
                MessageBoxImage.Information);
        }
        else
        {
            stopwatch.Stop();
            SaveStatus = $"Шлях: {_jsonSavePath ?? SaveData.DefaultFilePath}";
            SaveStatusColor = string.IsNullOrEmpty(_jsonSavePath) ? Brushes.Red : Brushes.Green;
            MessageBox.Show("Послідовності не знайдено.", "Попередження", MessageBoxButton.OK, MessageBoxImage.Warning);
        }
        SearchProgress = 0;
        EstimatedTimeRemaining = TimeSpan.Zero;
    }
    catch (OperationCanceledException)
    {
        SearchProgress = 0;
        EstimatedTimeRemaining = TimeSpan.Zero;
        SaveStatus = $"Шлях: {_jsonSavePath ?? SaveData.DefaultFilePath}";
        SaveStatusColor = string.IsNullOrEmpty(_jsonSavePath) ? Brushes.Red : Brushes.Green;
        _tolerancePercentage = savedTolerancePercentage;
        _maxY = savedMaxY;
        _tolerance = _tolerancePercentage / 100 * _maxY;
```

```
Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Background, new Action(() =>
            {
                OnPropertyChanged(nameof(TolerancePercentage));
                OnPropertyChanged(nameof(Tolerance));
            }));
        }
        catch (Exception ex)
        {
            SearchProgress = 0;
            EstimatedTimeRemaining = TimeSpan.Zero;
            SaveStatus = $"Шлях: {_jsonSavePath ?? SaveData.DefaultFilePath}";
            SaveStatusColor = string.IsNullOrEmpty(_jsonSavePath) ? Brushes.Red : Brushes.Green;
            _tolerancePercentage = savedTolerancePercentage;
            _maxY = savedMaxY;
            _tolerance = _tolerancePercentage / 100 * _maxY;

Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Background, new Action(() =>
            {
                OnPropertyChanged(nameof(TolerancePercentage));
                OnPropertyChanged(nameof(Tolerance));
            }));
            MessageBox.Show($"Помилка під час пошуку: {ex.Message}", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
        }
        finally
        {
            IsSearchButtonEnabled = true;
            ClearCancellationToken();
        }
    }
    private void CalculateFractalDimensionBySequence(object? _)
    {
        var (dimension, logM, logDelta, maxLength, _) = Calculation.CalculateFractalDimensionBySequence(SequenceLengths.ToList());
        if (dimension > 0)
        {
            FractalDimension = dimension;
            var fractalResult = new FractalDimensionResult
            {
                TolerancePercentage = TolerancePercentage,
                Dimension = dimension,
                LogM = logM,
                LogDelta = logDelta,
                MaxLength = maxLength,
                SequenceGroups = new ObservableCollection<SequenceLengthGroup>(SequenceLengths.ToList())
            };
            _fractalResults.Add(fractalResult);
            var meta = new ExportMetadata { ExcelFileName = Path.GetFileNameWithoutExtension(_excelFilePath) ?? "Ряд", MaxY = _maxY };
            var viewModel = new FractalDimensionPlotViewModel(logM, logDelta, Enumerable.Range(1, maxLength).ToList(), Tolerance, dimension, SequenceLengths.ToList(), meta);
            var window = new FractalDimensionPlotWindow(viewModel)
            {
                Title = $"Фрактальна розмірність: {dimension:F4} (Допуск: {TolerancePercentage:F2}%)"
            };
            window.Show();
        }
        else
        {
            FractalDimension = 0;
            MessageBox.Show("Не вдалося розрахувати фрактало-подібну розмірність.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Warning);
        }
    }
    private void CalculateFractalDimension(object? _)
    {
        _fractalResults.Clear();
        NotifyFractalResultsChanged();
        var (dimension, logM, logDelta, maxLength, lengths) = Calculation.CalculateFractalDimensionBySequence(SequenceLengths.ToList());
        if (dimension > 0)
        {
            FractalDimension = dimension;
            var fractalResult = new FractalDimensionResult
            {
                TolerancePercentage = TolerancePercentage,
                Dimension = dimension,
                LogM = logM,
                LogDelta = logDelta,
                MaxLength = maxLength,
                Lengths = lengths,
                SequenceGroups = new ObservableCollection<SequenceLengthGroup>(SequenceLengths.ToList())
            };
            _fractalResults.Add(fractalResult);
            NotifyFractalResultsChanged();
```

```
            if (_currentFractalWindow?.IsLoaded == true)
            {
                var vm = _currentFractalWindow.DataContext as FractalDimensionRangeViewModel;
                vm?.UpdateResults(_fractalResults.ToList());
            }
            var meta = new ExportMetadata { ExcelFileName = Path.GetFileNameWithoutExtension(_excelFilePath) ?? "Ряд", MaxY = _maxY };
            var viewModel = new FractalDimensionPlotViewModel(logM, logDelta, lengths, Tolerance, dimension, SequenceLengths.ToList(), meta);
            var window = new FractalDimensionPlotWindow(viewModel)
            {
                Title = $"Фрактальна розмірність: {dimension:F4} (Допуск: {TolerancePercentage:F2}%)"
            };
            window.Show();
        }
        else
        {
            FractalDimension = 0;
            MessageBox.Show("Не вдалося розрахувати фрактало-подібну розмірність.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Warning);
        }
    }
    private void ToggleDataListBox(object? _) => IsDataListBoxVisible = !_isDataListBoxVisible;
    private void ResetJsonPath(object? _)
    {
        _jsonSavePath = null;
        _fractalJsonSavePath = null;
        MainSettings.Default.JsonFilePath = null;
        MainSettings.Default.FractalJsonFilePath = null;
        MainSettings.Default.Save();
        NotifySavedFractalPathChanged();
        SaveStatus = $"Шлях: {SaveData.DefaultFilePath}";
        SaveStatusColor = Brushes.Red;
        CanSave = _fractalResults.Any();
        UpdateWindowTitle();
        MessageBox.Show("Шлях до JSON скинуто.", "Успіх", MessageBoxButton.OK, MessageBoxImage.Information);
        OnPropertyChanged(nameof(JsonFileName));
        OnPropertyChanged(nameof(JsonSavePath));
        OnPropertyChanged(nameof(FractalJsonFileName));
        OnPropertyChanged(nameof(FractalJsonSavePath));
    }
    private void OpenFractalGraph(object? _)
    {
        var latestResult = _fractalResults.LastOrDefault();
        if (latestResult != null)
        {
            var meta = new ExportMetadata { ExcelFileName = Path.GetFileNameWithoutExtension(_excelFilePath) ?? "Ряд", MaxY = _maxY };
            var viewModel = new FractalDimensionPlotViewModel(
                latestResult.LogM,
                latestResult.LogDelta,
                Enumerable.Range(1, latestResult.MaxLength).ToList(),
                _tolerance,
                latestResult.Dimension,
                latestResult.SequenceGroups?.ToList() ?? new List<SequenceLengthGroup>(),
                meta);
            var window = new FractalDimensionPlotWindow(viewModel)
            {
                Title = $"Фрактальна розмірність: {latestResult.Dimension:F4} (Допуск: {latestResult.TolerancePercentage:F2}%)"
            };
            window.Show();
        }
        else
            MessageBox.Show("Немає доступних результатів для відображення графіка.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Warning);
    }
    public void OnFractalJsonPathChanged(string newPath)
    {
        if (_fractalJsonSavePath == newPath) return;
        _fractalJsonSavePath = newPath;
        MainSettings.Default.FractalJsonFilePath = newPath;  // Збереження шляху
        MainSettings.Default.Save();
        Debug.WriteLine($"OnFractalJsonPathChanged: Збережено шлях {newPath}");
        NotifySavedFractalPathChanged();
        UpdateWindowTitle();

Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Background, () =>
        {
            OnPropertyChanged(nameof(FractalJsonFileName));  // Оновлення UI в меню "Інфо -> Файли"
            OnPropertyChanged(nameof(FractalJsonSavePath));
        });
    }

    private void OpenOrUpdateCurrentFractalWindow(object _)
    {
        if (_fractalResults.Count == 0) return;
```

```
        if (_currentFractalWindow == null || !_currentFractalWindow.IsLoaded)
        {
            var vm = new FractalDimensionRangeViewModel(_fractalResults.ToList(), _maxY, this);
            vm.FractalJsonPathChanged += OnFractalJsonPathChanged;
            _currentFractalWindow = new FractalDimensionRangeWindow(vm);
            _currentFractalWindow.Closed += (s, e) => _currentFractalWindow = null;
            _currentFractalWindow.Show();
        }
        else
        {
            var vm = _currentFractalWindow.DataContext as FractalDimensionRangeViewModel;
            vm?.UpdateResults(_fractalResults.ToList());
        }
    }
    private void OpenSavedFractalWindow(object _)
    {
        string path = MainSettings.Default.FractalJsonFilePath;
        List<FractalDimensionResult> results = new();
        string excelPath = "", jsonPath = "";

        if (!string.IsNullOrEmpty(path) && File.Exists(path))
        {
            (results, excelPath, jsonPath) = LoadData.LoadFractalDimensionRangeFromJson(path);
        }
        else if (!string.IsNullOrEmpty(path))
        {
            MessageBox.Show($"Файл не знайдено:\n{path}", "Попередження", MessageBoxButton.OK, MessageBoxImage.Warning);
        }

        var vm = new FractalDimensionRangeViewModel(results, _maxY, this);
        vm.FractalJsonPathChanged += OnFractalJsonPathChanged;

        var window = new FractalDimensionRangeWindow(vm);
        window.Title = string.IsNullOrEmpty(path)
            ? "Нове вікно порівняння"
            : $"Збережені результати – {Path.GetFileName(path)}";
        window.Show();
    }
    private void UpdateMaxY()
    {
        if (App.data.Any())
        {
            _maxY = App.data.Max(p => Math.Abs(p.Y));
            if (_maxY == 0) _maxY = 1.0;
        }
        else _maxY = 1.0;
        _tolerance = _tolerancePercentage / 100 * _maxY;
        Application.Current.Dispatcher.InvokeAsync(() =>
        {
            OnPropertyChanged(nameof(Tolerance));
            OnPropertyChanged(nameof(TolerancePercentage));
        });
    }
    private void UpdateSequenceLengths()
    {
        SequenceLengths.Clear();
        var groupedByLength = MatchingSequencesGroups
            .Where(g => g.Sequence.Count >= (IncludeSinglePointSequences ? 1 : 2))
            .GroupBy(g => g.Sequence.Count)
            .OrderBy(g => g.Key);
        foreach (var group in groupedByLength)
            SequenceLengths.Add(new SequenceLengthGroup { Length = group.Key, Sequences = group.ToList() });
    }
    public void UpdateGraph()
    {
        PlotModel.Series.Clear();
        _mainWindowGraf.ScatterSeries.Points.Clear();
        if (!_isGraphDeleted && DataPoints.Any())
        {
            _plotModelInstance.AddLineSeries(PlotModel, DataPoints, _mainWindowGraf.lineSeries);
            if (SelectedSequence != null)
            {
                AddSequenceToPlot(SelectedSequence, OxyColors.Red, 2, 6, $"Обрана послідовність ", SelectedSequence.GroupId);
                foreach (var similarIndex in SelectedSequence.SimilarSequenceIndices)
                {
                    var similarGroup = new MatchingSequenceGroup
                    {
                        SequenceIndex = similarIndex,
                        Sequence = DataPoints.Skip(similarIndex).Take(SelectedSequence.Sequence.Count).ToList(),
                        ToleranceUsed = SelectedSequence.ToleranceUsed
                    };
                    AddSequenceToPlot(similarGroup, OxyColors.Blue, 1, 4, $"Схожа послідовність ", similarIndex);
                }
            }
        }
```

```
        if (SelectedPoints.Any())
        {
            var scatterSeries = new ScatterSeries
            {
                MarkerType = MarkerType.Circle,
                MarkerSize = 5,
                MarkerFill = OxyColors.Red,
                Title = "Вибрана точка"
            };
            var point = SelectedPoints.First();
            scatterSeries.Points.Add(new ScatterPoint(point.X, point.Y));
            PlotModel.Series.Add(scatterSeries);
        }
        PlotModel.InvalidatePlot(true);
    }
    private void AddSequenceToPlot(MatchingSequenceGroup sequence, OxyColor color, double lineThickness, double markerSize, string titlePrefix, int sequenceNumber)
    {
        if (sequence?.Sequence == null || sequence.Sequence.Count == 0) return;
        var lineSeries = new LineSeries { Color = color, StrokeThickness = lineThickness, Title = $"{titlePrefix} {sequenceNumber}" };
        var scatterSeries = new ScatterSeries { MarkerType = MarkerType.Circle, MarkerSize = markerSize, MarkerFill = color };
        foreach (var point in sequence.Sequence)
        {
            lineSeries.Points.Add(new DataPoint(point.X, point.Y));
            scatterSeries.Points.Add(new ScatterPoint(point.X, point.Y));
        }
        PlotModel.Series.Add(lineSeries);
        PlotModel.Series.Add(scatterSeries);
    }
    public void UpdateGraphForSingleSequence(MatchingSequenceGroup sequence)
    {
        if (sequence == null || _isGraphDeleted) return;
        PlotModel.Series.Clear();
        _mainWindowGraf.ScatterSeries.Points.Clear();
        _plotModelInstance.AddLineSeries(PlotModel, DataPoints, _mainWindowGraf.lineSeries);
        AddSequenceToPlot(sequence, OxyColors.Red, 2, 6, $"Обрана послідовність ", sequence.GroupId);
        PlotModel.InvalidatePlot(true);
    }
    public void UpdateGraphForLength(SequenceLengthGroup group)
    {
        if (group == null || group.Sequences == null || group.Sequences.Count == 0 || _isGraphDeleted) return;
        PlotModel.Series.Clear();
        _mainWindowGraf.ScatterSeries.Points.Clear();
        _plotModelInstance.AddLineSeries(PlotModel, DataPoints, _mainWindowGraf.lineSeries);
        foreach (var sequence in group.Sequences)
        {
            AddSequenceToPlot(sequence, OxyColors.Red, 1, 4, $"Послідовність ", sequence.GroupId);
            foreach (var similarIndex in sequence.SimilarSequenceIndices)
            {
                var similarGroup = new MatchingSequenceGroup
                {
                    SequenceIndex = similarIndex,
                    Sequence = DataPoints.Skip(similarIndex).Take(sequence.Sequence.Count).ToList(),
                    ToleranceUsed = sequence.ToleranceUsed
                };
                AddSequenceToPlot(similarGroup, OxyColors.Blue, 1, 4, $"Схожа послідовність ", similarIndex);
            }
        }
        PlotModel.InvalidatePlot(true);
    }
    public void UpdateGraphForSingleSequenceWithSimilar(MatchingSequenceGroup sequence)
    {
        if (sequence == null || _isGraphDeleted) return;
        PlotModel.Series.Clear();
        _mainWindowGraf.ScatterSeries.Points.Clear();
        _plotModelInstance.AddLineSeries(PlotModel, DataPoints, _mainWindowGraf.lineSeries);
        AddSequenceToPlot(sequence, OxyColors.Red, 2, 6, $"Обрана послідовність ", sequence.GroupId);
        foreach (var similarIndex in sequence.SimilarSequenceIndices)
        {
            var similarGroup = new MatchingSequenceGroup
            {
                SequenceIndex = similarIndex,
                Sequence = DataPoints.Skip(similarIndex).Take(sequence.Sequence.Count).ToList(),
                ToleranceUsed = sequence.ToleranceUsed
            };
            AddSequenceToPlot(similarGroup, OxyColors.Blue, 1, 4, $"Схожа послідовність ", similarIndex);
        }
```

```
            PlotModel.InvalidatePlot(true);
        }
        private void Export()
        {
            var meta = new ExportMetadata
            {
                ExcelFileName = Path.GetFileNameWithoutExtension(_excelFilePath) ?? "Ряд",
                TotalPoints = App.data.Count,
                Tolerance = _tolerancePercentage
            };
            PlotExporter.ExportToJpg(PlotModel, "Часові ряди", meta);
        }
        private void UpdateTaskbarProgress()
        {
            var window = Application.Current.MainWindow;
            if (window?.TaskbarItemInfo == null) return;
            if (SearchProgress > 0 && SearchProgress < 100)
            {
                window.TaskbarItemInfo.ProgressState = System.Windows.Shell.TaskbarItemProgressState.Normal;
                window.TaskbarItemInfo.ProgressValue = SearchProgress / 100.0; // 0.0 – 1.0
            }
            else
            {
                window.TaskbarItemInfo.ProgressState = System.Windows.Shell.TaskbarItemProgressState.None;
            }
        }
    }
}
```

4.4 Текст програми файлу ToleranceRangeViewModel.cs

```
using ECSSTS.Bases;
using ECSSTS.Data;
using ECSSTS.Models;
using ECSSTS.Processing;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows;
using System.Collections.ObjectModel;
using System.Diagnostics;

namespace ECSSTS.ViewModel
{
    public class ToleranceRangeViewModel : ViewModelBase
    {
        private readonly MainViewModel _parentViewModel;
        private double _minTolerancePercentage;
        private double _maxTolerancePercentage;
        private double _step;
        private bool _isSearchButtonEnabled = true;
        private double _averageIterationTime; // Середній час однієї ітерації

        public ICommand CommandSearchRange { get; }

        public ToleranceRangeViewModel(MainViewModel parentViewModel)
        {
            _parentViewModel = parentViewModel;
            MinTolerancePercentage = parentViewModel.TolerancePercentage;
            MaxTolerancePercentage = 5.0;
            Step = 0.5;
            CommandSearchRange = new RelayCommand(SearchRange);
            // Початкова оцінка на основі кількості точок і допуску
            _averageIterationTime = EstimateInitialIterationTime(parentViewModel.DataPoints.Count, parentViewModel.TolerancePercentage);
        }

        private double EstimateInitialIterationTime(int pointCount, double tolerancePercentage)
        {
            // Базова оцінка за кількістю точок
            double baseTime;
            if (pointCount <= 5000)
                baseTime = 13.0; // ~40 секунд для 5к точок
            else if (pointCount <= 10000)
                baseTime = 80.0;
            else if (pointCount <= 35000)
                baseTime = 300.0;
            else
                baseTime = 600.0;

            // Коефіцієнт для допуску: менший допуск збільшує час через більше порівнянь
            double toleranceFactor = tolerancePercentage <= 0.1 ? 1.2 : (tolerancePercentage <= 0.5 ? 1.0 : 0.8);
```

```
        // Коефіцієнт для апаратного забезпечення (припускаємо 4 ядра як базовий варіант)
        double hardwareFactor = Math.Max(1.0, Environment.ProcessorCount / 4.0);
        return baseTime * toleranceFactor / hardwareFactor;
    }

    public double MinTolerancePercentage
    {
        get => _minTolerancePercentage;
        set
        {
            if (value >= 0 && value <= 100)
            {
                _minTolerancePercentage = value;
                OnPropertyChanged();
                OnPropertyChanged(nameof(IterationCount));
            }
        }
    }

    public double MaxTolerancePercentage
    {
        get => _maxTolerancePercentage;
        set
        {
            if (value >= 0 && value <= 100)
            {
                _maxTolerancePercentage = value;
                OnPropertyChanged();
                OnPropertyChanged(nameof(IterationCount));
            }
        }
    }

    public double Step
    {
        get => _step;
        set
        {
            if (value >= 0.01)
            {
                _step = value;
                OnPropertyChanged();
                OnPropertyChanged(nameof(IterationCount));
            }
        }
    }

    public int IterationCount
    {
        get
        {
            if (_step > 0 && _minTolerancePercentage < _maxTolerancePercentage)
            {
                return (int)Math.Floor((_maxTolerancePercentage - _minTolerancePercentage) / _step) + 1;
            }
            return 1; // Гарантуємо хоча б одну ітерацію
        }
    }

    public bool IsSearchButtonEnabled
    {
        get => _isSearchButtonEnabled;
        set
        {
            _isSearchButtonEnabled = value;
            _parentViewModel.IsSearchButtonEnabled = value; // Синхронізуємо з MainViewModel
            OnPropertyChanged();
        }
    }

    private async void SearchRange(object parameter)
    {
        if (_minTolerancePercentage >= _maxTolerancePercentage)
        {
            MessageBox.Show("Мінімальний допуск не може бути більшим або дорівнювати максимальному.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }
        if (_step <= 0)
        {
            MessageBox.Show("Крок має бути більшим за 0.", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }
        if (IterationCount > 50)
        {
            var result = MessageBox.Show($"Виконання {IterationCount} ітерацій може зайняти багато часу. Продовжити?", "Попередження", MessageBoxButton.YesNo, MessageBoxImage.Warning);
            if (result != MessageBoxResult.Yes)
                return;
        }
        IsSearchButtonEnabled = false;
        _parentViewModel.InitializeCancellationToken();
        var results = new List<FractalDimensionResult>();
        var iterationTimes = new List<double>();
        try
```

```
        {
            int totalMainSequences = 0;
            int totalSequences = 0;
            int currentIteration = 0;
            int totalIterations = IterationCount;
            var stopwatch = Stopwatch.StartNew();
            for (double tolerancePercent = _minTolerancePercentage;
tolerancePercent <= _maxTolerancePercentage + 0.0001;
tolerancePercent += _step)
            {
                if (tolerancePercent > _maxTolerancePercentage +
0.0001)
                    break;
                if
(_parentViewModel.CancellationTokenSource?.Token.IsCancellatio
nRequested == true)
                {
                    _parentViewModel.SearchProgress = 0;
                    _parentViewModel.EstimatedTimeRemaining =
TimeSpan.Zero;
                    MessageBox.Show("Пошук скасовано.", "Скасу-
вання", MessageBoxButton.OK, MessageBoxImage.Information);
                    return;
                }
                double tolerance = tolerancePercent / 100 *
_parentViewModel.MaxY;
                var iterationStopwatch = Stopwatch.StartNew();
                var iterationProgress = new Progress<double>(value =>
                {
                    double normalizedProgress = ((double)currentIteration
+ value / 100) / totalIterations * 100;
                    _parentViewModel.SearchProgress =
Math.Min(normalizedProgress, 100);
                    Debug.WriteLine($"SearchRange:
Iteration={currentIteration}, Tolerance={tolerancePercent:F2}%,
Value={value:F2},
NormalizedProgress={normalizedProgress:F2}%");
                    double totalRemainingSeconds;
                    double remainingIterations = totalIterations -
currentIteration - (value / 100);
                    if (value > 0 && value < 100)
                    {
                        double elapsedSeconds =
iterationStopwatch.Elapsed.TotalSeconds;
                        double estimatedIterationSeconds = elapsedSeconds
/ (value / 100);
                        double remainingIterationSeconds = Math.Max(0,
estimatedIterationSeconds - elapsedSeconds);
                        double weightedAvg = iterationTimes.Any() ?
iterationTimes.TakeLast(3).Average() : _averageIterationTime;
                        totalRemainingSeconds =
remainingIterationSeconds + (weightedAvg * (totalIterations -
currentIteration - 1));
                    }
                    else
                    {
                        double weightedAvg = iterationTimes.Any() ?
iterationTimes.TakeLast(3).Average() : _averageIterationTime;
                        totalRemainingSeconds =
weightedAvg * remainingIterations;
                    }
                    _parentViewModel.EstimatedTimeRemaining =
TimeSpan.FromSeconds(totalRemainingSeconds);
                });
                var timeProgress = new Progress<TimeSpan>(_ => { });
                var searchResult =
_parentViewModel.IsRelativeTolerance
                    ? await
Calculation.SearchRelativeAsync(_parentViewModel.DataPoints,
tolerance, iterationProgress, timeProgress,
_parentViewModel.CancellationTokenSource?.Token ?? default)
                    : await
Calculation.SearchAsync(_parentViewModel.DataPoints, tolerance,
iterationProgress, timeProgress,
_parentViewModel.CancellationTokenSource?.Token ?? default);
                iterationStopwatch.Stop();

iterationTimes.Add(iterationStopwatch.Elapsed.TotalSeconds);
                _averageIterationTime = iterationTimes.Any() ?
iterationTimes.TakeLast(3).Average() : _averageIterationTime;
                if
(_parentViewModel.CancellationTokenSource?.Token.IsCancellatio
nRequested == true)
                {
                    _parentViewModel.SearchProgress = 0;
                    _parentViewModel.EstimatedTimeRemaining =
TimeSpan.Zero;
                    MessageBox.Show("Пошук скасовано.", "Скасу-
вання", MessageBoxButton.OK, MessageBoxImage.Information);
                    return;
                }
                if (searchResult != null && searchResult.Any())
                {
                    var currentSequences = new
ObservableCollection<MatchingSequenceGroup>(searchResult);
                    totalMainSequences += currentSequences.Count;
                    var sequenceLengths = currentSequences
                        .Where(g => g.Sequence.Count >= 1)
                        .GroupBy(g => g.Sequence.Count)
                        .OrderBy(g => g.Key)
                        .Select(g => new SequenceLengthGroup
                        {
```

```
                    Length = g.Key,
                    Sequences = g.ToList()
                })
                .ToList();
            totalSequences += sequenceLengths.Sum(group => group.TotalSequencesCount);
            var (dimension, logM, logDelta, maxLength, lengths) = Calculation.CalculateFractalDimensionBySequence(sequenceLengths);
            if (dimension > 0)
            {
                Debug.WriteLine($"SearchRange: Фрактальна розмірність для допуску {tolerancePercent:F2}%: {dimension:F4}, Основних послідовностей: {currentSequences.Count}, Загальна кількість: {totalSequences}");
                results.Add(new FractalDimensionResult
                {
                    TolerancePercentage = Math.Round(tolerancePercent, 2),
                    Dimension = dimension,
                    LogM = logM,
                    LogDelta = logDelta,
                    MaxLength = maxLength,
                    Lengths = lengths, // Додаємо lengths
                    SequenceGroups = new ObservableCollection<SequenceLengthGroup>(sequenceLengths)
                });
            }
            else
            {
                Debug.WriteLine($"SearchRange: Не вдалося розрахувати фрактальну розмірність для допуску {tolerancePercent:F2}%");
                MessageBox.Show($"Не вдалося розрахувати фрактальну розмірність для допуску {tolerancePercent:F2}%.", "Попередження", MessageBoxButton.OK, MessageBoxImage.Warning);
            }
        }
        else
        {
            Debug.WriteLine($"SearchRange: Не знайдено послідовностей для допуску {tolerancePercent:F2}%");
        }
        currentIteration++;
    }
    stopwatch.Stop();
    _parentViewModel.SearchProgress = 0;
    _parentViewModel.EstimatedTimeRemaining = TimeSpan.Zero;
    if (results.Any())
    {
        foreach (var result in results)
        {
            Debug.WriteLine($"SearchRange: Зібрано результат для допуску {result.TolerancePercentage:F2}%");
        }
        var viewModel = new FractalDimensionRangeViewModel(results, _parentViewModel.MaxY);
        var window = new FractalDimensionRangeWindow(viewModel);
        window.Show();
    }
    MessageBox.Show(
        $"Пошук завершено.\nЗнайдено {totalMainSequences} основних послідовностей\nЗагальна кількість послідовностей: {totalSequences}\nПошук зайняв: {stopwatch.Elapsed:hh\\:mm\\:ss\\.fff}",
        "Результат",
        MessageBoxButton.OK,
        MessageBoxImage.Information);
}
catch (OperationCanceledException)
{
    _parentViewModel.SearchProgress = 0;
    _parentViewModel.EstimatedTimeRemaining = TimeSpan.Zero;
    MessageBox.Show("Пошук скасовано.", "Скасування", MessageBoxButton.OK, MessageBoxImage.Information);
}
catch (Exception ex)
{
    _parentViewModel.SearchProgress = 0;
    _parentViewModel.EstimatedTimeRemaining = TimeSpan.Zero;
    Debug.WriteLine($"SearchRange: Помилка: {ex.Message}");
    MessageBox.Show($"Помилка під час пошуку: {ex.Message}", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
}
finally
{
    IsSearchButtonEnabled = true;
    _parentViewModel.ClearCancellationToken();
}
}
}
}
```

4.5 Текст програми файлу ViewModelBase.cs

```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace ECSSTS.Bases
{
   public class ViewModelBase : INotifyPropertyChanged
   {
      public event PropertyChangedEventHandler? PropertyChanged;

      protected virtual void OnPropertyChanged([CallerMemberName] string? propertyName = null)
      {
         PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
      }

      protected bool SetProperty<T>(ref T storage, T value, [CallerMemberName] string? propertyName = null, params string[] additionalPropertyNames)
      {
         if (EqualityComparer<T>.Default.Equals(storage, value))
            return false;

         storage = value;
         OnPropertyChanged(propertyName);
         foreach (var additionalProperty in additionalPropertyNames)
            OnPropertyChanged(additionalProperty);

         return true;
      }
   }
}
```

4.6 Текст програми файлу RelayCommand.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace ECSSTS.Bases
{
   public class RelayCommand : ICommand
   {
      private Action<object?> execute;
      private Func<object?, bool>? canExecute;

      public event EventHandler? CanExecuteChanged
      {
         add { CommandManager.RequerySuggested += value;
}
         remove { CommandManager.RequerySuggested -=
value; }
      }

      public RelayCommand(Action<object?> execute,
Func<object?, bool>? canExecute = null)
      {
         this.execute = execute;
         this.canExecute = canExecute;
      }

      public bool CanExecute(object? parameter)
      {
         return canExecute == null || canExecute(parameter);
      }

      public void Execute(object? parameter)
      {
         execute(parameter);
      }
   }

}
```

5.1 Текст програми файлу MainWindow.xaml.cs

```
using OxyPlot;
using OxyPlot.Series;
using System.Collections.ObjectModel;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using ECSSTS.Data;
using ECSSTS.Models;
using System.Diagnostics;
namespace ECSSTS
{
   /// <summary>
   /// Interaction logic for MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
         TextBoxTolerance.Text = "0.1";
      }
      private void DataListBox_SelectionChanged(object
sender, SelectionChangedEventArgs e)
      {
         if (DataContext is MainViewModel viewModel)
         {
            viewModel.SelectedPoints.Clear();
```

```
            if (DataListBox.SelectedItem is DataExcel
selectedPoint)
            {
                viewModel.SelectedPoints.Add(selectedPoint);
            }
            viewModel.UpdateGraph();
        }
    }
    private void
SequenceLengthList_SelectionChanged(object sender,
SelectionChangedEventArgs e)
    {
        if (DataContext is MainViewModel vm && sender is
ListBox listBox && listBox.SelectedItem is
SequenceLengthGroup selectedGroup)
        {
            vm.UpdateGraphForLength(selectedGroup);
            listBox.SelectedItem = null; // Очищаємо вибір
        }
    }
    private void Sequence_MouseDown(object sender,
System.Windows.Input.MouseButtonEventArgs e)
    {
        if (DataContext is MainViewModel vm && sender is
TextBlock textBlock && textBlock.DataContext is
MatchingSequenceGroup sequence)
        {

vm.UpdateGraphForSingleSequenceWithSimilar(sequence); //
Малюємо базову і схожі
        }
    }

    private void
SimilarSequenceList_SelectionChanged(object sender,
SelectionChangedEventArgs e)
    {
        if (DataContext is MainViewModel vm && sender is
ListBox listBox && listBox.SelectedItem is int similarIndex)
        {
            var baseSequence = (listBox.DataContext as
MatchingSequenceGroup);
            if (baseSequence != null)
            {
                var similarSequence =
vm.MatchingSequencesGroups.FirstOrDefault(g =>
g.SequenceIndex == similarIndex);
                if (similarSequence != null)
                {

vm.UpdateGraphForSingleSequence(similarSequence); // Ма-
люємо лише одну схожу
                }
                else
                {
                    var tempSequence = new
MatchingSequenceGroup
                    {
                        SequenceIndex = similarIndex,
                        Sequence =
vm.DataPoints.Skip(similarIndex).Take(baseSequence.Sequenc
e.Count).ToList(),
                        GroupId = similarIndex // Тимчасовий
GroupId
                    };

vm.UpdateGraphForSingleSequence(tempSequence); // Малю-
ємо лише одну схожу
                }
            }
            listBox.SelectedItem = null; // Очищаємо вибір
        }
    }
    private void
TextBoxTolerance_MouseDoubleClick(object sender,
System.Windows.Input.MouseButtonEventArgs e)
    {
        if (sender is TextBox && DataContext is
MainViewModel viewModel)
        {
            var window = new
ToleranceRangeWindow(viewModel);
            window.Show();

Debug.WriteLine("TextBoxTolerance_MouseDoubleClick: Bi-
дкрито ToleranceRangeWindow");
        }
    }
  }
}
```

5.2 Текст програми файлу MainWindow.xaml

```
<Window
  x:Class="ECSSTS.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/present
ation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:behaviors="clr-namespace:ECSSTS.Behaviors"
  xmlns:converter="clr-namespace:ECSSTS.Converters"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008
"
  xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
  xmlns:local="clr-namespace:ECSSTS"
  xmlns:local1="clr-namespace:ECSSTS.ViewModel"
  xmlns:localData="clr-namespace:ECSSTS.Data"
  xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
  xmlns:oxy="http://oxyplot.org/wpf"
  Title="{Binding WindowTitle}"
  Width="1100"
  Height="500"
  MinWidth="870"
```

```
    MinHeight="240"
    WindowStartupLocation="CenterScreen"
    mc:Ignorable="d">
    <Window.Resources>
        <converter:BooleanToVisibilityConverter
x:Key="BooleanToVisibilityConverter" />
        <converter:DoubleToVisibilityConverter
x:Key="DoubleToVisibilityConverter" />
        <converter:DoubleToBooleanConverter
x:Key="DoubleToBooleanConverter" />
        <converter:StringToFloatConverter
x:Key="StringToFloatConverter" />
        <converter:BoolToIndexConverter
x:Key="BoolToIndexConverter" />
        <converter:EndIndexConverter
x:Key="EndIndexConverter" />
        <converter:SequenceLengthConverter
x:Key="SequenceLengthConverter" />
        <converter:SimilarSequenceIndexToGroupIdConverter
x:Key="SimilarSequenceIndexToGroupIdConverter" />
        <converter:SubtractConverter x:Key="SubtractConverter"
/>
        <converter:TimeSpanToStringConverter
x:Key="TimeSpanToStringConverter" />
        <converter:StringToDoubleConverter
x:Key="StringToDoubleConverter" />
    </Window.Resources>

    <Window.DataContext>
        <local:MainViewModel />
    </Window.DataContext>

    <Window.TaskbarItemInfo>
        <TaskbarItemInfo x:Name="TaskbarItem"
                         ProgressState="Normal"
                         ProgressValue="{Binding SearchProgress,
Mode=OneWay}" />
    </Window.TaskbarItemInfo>

    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="85*" />
            <ColumnDefinition Width="700*" />
        </Grid.ColumnDefinitions>
        <DockPanel Grid.ColumnSpan="2">
            <Grid Panel.ZIndex="100" DockPanel.Dock="Top">
                <ToolBarTray Style="{StaticResource
CustomToolBarTrayStyle}">
                    <ToolBar Style="{StaticResource
CustomToolBarStyle}">
                        <Menu VerticalAlignment="Center">
                            <MenuItem Header="Файл"
Style="{StaticResource ToolBarMenuItemStyle}">
                                <MenuItem
                                    Header="Автозбереження"
                                    IsCheckable="True"
                                    IsChecked="{Binding
IsAutoSaveEnabled, Mode=TwoWay}"
                                    Style="{StaticResource
CheckBoxMenuItemStyle}"
                                    ToolTip="Зберігає результати пошуку
автоматично за шляхом, вказаним у статусі" />
                                <MenuItem Command="{Binding
CommandOpenFile}" Header="Відкрити файл Excel" />
                                <MenuItem Command="{Binding
CommandLoadJson}" Header="Відкрити файл JSON" />
                                <MenuItem
                                    Command="{Binding
CommandSaveJson}"
                                    CommandParameter="default"
                                    Header="Зберегти за замовчуванням
(Result.json)"
                                    IsEnabled="{Binding CanSave}" />
                                <MenuItem
                                    Command="{Binding
CommandSaveJson}"
                                    CommandParameter="current"
                                    Header="Зберегти"
                                    IsEnabled="{Binding
CanSaveToCurrentPath}" />
                                <MenuItem
                                    Command="{Binding
CommandSaveJson}"
                                    CommandParameter="as"
                                    Header="Зберегти як..." />
                            </MenuItem>
                            <MenuItem Header="Налаштування/Інфо"
Style="{StaticResource ToolBarMenuItemStyle}">
                                <MenuItem FontWeight="DemiBold"
Header="Файли">
                                    <MenuItem Style="{StaticResource
InfoPathMenuItemStyle}">
                                        <MenuItem.Header>
                                            <StackPanel
Orientation="Horizontal">
                                                <TextBlock Text="Excel: " />
                                                <TextBlock FontWeight="Bold"
Text="{Binding DataContext.ExcelFileName,
RelativeSource={RelativeSource AncestorType=Window},
Mode=OneWay}" />
                                                <TextBlock Text=" (" />
                                                <TextBlock Text="{Binding
DataContext.ExcelFilePath, RelativeSource={RelativeSource
AncestorType=Window}, Mode=OneWay}" />
                                                <TextBlock Text=")" />
                                            </StackPanel>
                                        </MenuItem.Header>
                                    </MenuItem>
                                    <MenuItem Style="{StaticResource
InfoPathMenuItemStyle}">
                                        <MenuItem.Header>
                                            <StackPanel
Orientation="Horizontal">
                                                <TextBlock Text="JSON: " />
                                                <TextBlock FontWeight="Bold"
Text="{Binding DataContext.JsonFileName,
RelativeSource={RelativeSource AncestorType=Window},
Mode=OneWay}" />
                                                <TextBlock Text=" (" />
```

```
                        <TextBlock Text="{Binding
DataContext.JsonSavePath, RelativeSource={RelativeSource
AncestorType=Window}, Mode=OneWay}" />
                        <TextBlock Text=")" />
                      </StackPanel>
                    </MenuItem.Header>
                  </MenuItem>
                  <MenuItem Style="{StaticResource
InfoPathMenuItemStyle}">
                    <MenuItem.Header>
                      <StackPanel
Orientation="Horizontal">
                        <TextBlock Text="Fractal JSON:
" />
                        <TextBlock FontWeight="Bold"
Text="{Binding DataContext.FractalJsonFileName,
RelativeSource={RelativeSource AncestorType=Window},
Mode=OneWay}" />
                        <TextBlock Text=" (" />
                        <TextBlock Text="{Binding
DataContext.FractalJsonSavePath,
RelativeSource={RelativeSource AncestorType=Window},
Mode=OneWay}" />
                        <TextBlock Text=")" />
                      </StackPanel>
                    </MenuItem.Header>
                  </MenuItem>
                  <MenuItem Style="{StaticResource
InfoPathMenuItemStyle}">
                    <MenuItem.Header>
                      <StackPanel
Orientation="Horizontal">
                        <TextBlock Text="За замовчу-
ванням: " />
                        <TextBlock FontWeight="Bold"
Text="Result.json" />
                        <TextBlock Text=" (" />
                        <TextBlock Text="{Binding
Source={x:Static localData:SaveData.DefaultFilePath},
Mode=OneWay}" />
                        <TextBlock Text=")" />
                      </StackPanel>
                    </MenuItem.Header>
                  </MenuItem>
                </MenuItem>
                <MenuItem Style="{StaticResource
InfoMenuItemStyle}">
                  <MenuItem.Header>
                    <StackPanel
Orientation="Horizontal">
                      <TextBlock Text="Відносний до-
пуск: " />
                      <TextBlock FontWeight="Bold"
Text="{Binding TolerancePercentage, StringFormat={}{0:F2}
%}" />
                    </StackPanel>
                  </MenuItem.Header>
                </MenuItem>
                <MenuItem Style="{StaticResource
InfoMenuItemStyle}">
                  <MenuItem.Header>
                    <StackPanel
Orientation="Horizontal">
                      <TextBlock Text="Точний допуск:
" />
                      <TextBlock FontWeight="Bold"
Text="{Binding DataContext.Tolerance,
RelativeSource={RelativeSource AncestorType=Window},
Mode=OneWay, StringFormat={}{0:F2}}" />
                    </StackPanel>
                  </MenuItem.Header>
                </MenuItem>
                <MenuItem Style="{StaticResource
InfoMenuItemStyle}">
                  <MenuItem.Header>
                    <StackPanel
Orientation="Horizontal">
                      <TextBlock Text="Кількість то-
чок: " />
                      <TextBlock FontWeight="Bold"
Text="{Binding DataContext.DataPoints.Count,
RelativeSource={RelativeSource AncestorType=Window},
Mode=OneWay}" />
                    </StackPanel>
                  </MenuItem.Header>
                </MenuItem>
                <MenuItem Style="{StaticResource
InfoMenuItemStyle}">
                  <MenuItem.Header>
                    <StackPanel
Orientation="Horizontal">
                      <TextBlock Text="Кількість пос-
лідовностей: " />
                      <TextBlock FontWeight="Bold"
Text="{Binding
DataContext.MatchingSequencesGroups.Count,
RelativeSource={RelativeSource AncestorType=Window},
Mode=OneWay}" />
                    </StackPanel>
                  </MenuItem.Header>
                </MenuItem>
                <MenuItem Style="{StaticResource
InfoMenuItemStyle}">
                  <MenuItem.Header>
                    <StackPanel
Orientation="Horizontal">
                      <TextBlock Text="Загальна кіль-
кість послідовностей: " />
                      <TextBlock FontWeight="Bold"
Text="{Binding TotalSequencesCount, Mode=OneWay,
StringFormat={}{0:N0}}" />
                    </StackPanel>
                  </MenuItem.Header>
                </MenuItem>
                <MenuItem
                  Header="Включати послідовності дов-
жиною 1"
                  IsCheckable="True"
                  IsChecked="{Binding
IncludeSinglePointSequences, Mode=TwoWay}"
```

```
                    Style="{StaticResource
CheckBoxMenuItemStyle}"
                    ToolTip="Дозволяє включати або ви-
ключати послідовності з однією точкою при обчисленні
фрактальної розмірності" />
                <MenuItem Command="{Binding
DataContext.ToggleDataListBoxCommand,
RelativeSource={RelativeSource AncestorType=Window}}"
Header="{Binding DataContext.DataListBoxToggleText,
RelativeSource={RelativeSource AncestorType=Window},
Mode=OneWay}" />
                <MenuItem Command="{Binding
DataContext.CommandResetJsonPath,
RelativeSource={RelativeSource AncestorType=Window}}"
Header="Скинути шлях до JSON" />
              </MenuItem>
            </Menu>
            <TextBlock
              Margin="5,0,5,0"
              VerticalAlignment="Center"
              Foreground="{Binding SaveStatusColor}"
              Text="Зберегти"
              ToolTip="{Binding SaveStatus}" />

            <Button
              Command="{Binding CommandClear}"
              Content="Очистити графік"
              Style="{StaticResource
ToolBarButtonStyle}" />

            <Button
              Command="{Binding CommandResetAxes}"
              Content="Скинути вісі"
              Style="{StaticResource
ToolBarButtonStyle}"
              ToolTip="Скинути масштаб осей графіка
до початкового стану" />

            <Button
              Command="{Binding CommandSearch}"
              Content="{Binding SearchButtonContent,
Mode=OneWay}"
              IsEnabled="{Binding IsSearchButtonEnabled,
Mode=OneWay}"
              Style="{StaticResource SearchButtonStyle}"
              ToolTip="{Binding SearchProgress,
StringFormat='Пошук: {0:F0}%'}" />

            <TextBox
              x:Name="TextBoxTolerance"
              Width="58"

MouseDoubleClick="TextBoxTolerance_MouseDoubleClick"
              TextAlignment="Right">
              <TextBox.Text>
                <Binding
                  Converter="{StaticResource
StringToDoubleConverter}"
                  Path="TolerancePercentage"
                  StringFormat="{}{0:F2} %"
UpdateSourceTrigger="PropertyChanged">
                  <Binding.ValidationRules>
                    <DataErrorValidationRule
ValidatesOnTargetUpdated="True" />
                  </Binding.ValidationRules>
                </Binding>
              </TextBox.Text>
              <TextBox.ToolTip>
                <ToolTip>
                  <TextBlock>
                    <Run Text="Абсолютне значення: "
/>
                    <Run Text="{Binding Tolerance,
StringFormat={}{0:F2}}" />
                  </TextBlock>
                </ToolTip>
              </TextBox.ToolTip>
            </TextBox>

            <CheckBox
              Margin="5,0,5,0"
              VerticalAlignment="Center"
              Content="Відносний допуск"
              IsChecked="{Binding IsRelativeTolerance,
Mode=TwoWay}"
              Style="{StaticResource
BaseCheckBoxStyle}"
              ToolTip="Вмикає пошук із відносним до-
пуском, де форма послідовностей порівнюється відносно
першої точки" />

            <Button
              Command="{Binding
CommandCalculateFractalDimension}"
              Content="Pax. фркт."
              Style="{StaticResource
ToolBarButtonStyle}"
              ToolTip="Розрахувати фракталоподібну
розмірність" />

            <Button
              Command="{Binding
CommandOpenFractalGraph}"
              Content="Відкр. фркт."
              Style="{StaticResource
ToolBarButtonStyle}"
              ToolTip="Відкрити графік фракталоподіб-
ної розмірності" />

            <TextBlock
              VerticalAlignment="Center"
              FontWeight="DemiBold"
              Text="{Binding FractalDimension,
StringFormat=' {0:F4}'}" />

            <ToolBar Style="{StaticResource
CustomToolBarStyle}">
              <Menu VerticalAlignment="Center">
```

```
                    <MenuItem Header="Фрактали"
Style="{StaticResource ToolBarMenuItemStyle}">
                        <MenuItem Header="Поточні резуль-
тати"
          Command="{Binding
OpenOrUpdateCurrentFractalWindowCommand}"
          IsEnabled="{Binding HasFractalResults}" />
                        <MenuItem Header="Збережені"
          Command="{Binding
OpenSavedFractalWindowCommand}"
          />
                    </MenuItem>
                  </Menu>
                </ToolBar>

              </ToolBar>
            </ToolBarTray>
            <ToolBarTray HorizontalAlignment="Right"
Style="{StaticResource CustomToolBarTrayStyle}">
              <ToolBar Style="{StaticResource
CustomToolBarStyle}">
                <TextBlock
                  Margin="5,0,10,0"
                  VerticalAlignment="Center"
                  FontWeight="DemiBold"
                  Text="{Binding EstimatedTimeRemaining,
Converter={StaticResource TimeSpanToStringConverter},
StringFormat='Час: {0}'}"
                  Visibility="{Binding SearchProgress,
Converter={StaticResource DoubleToVisibilityConverter}}" />
                <Button
                  Command="{Binding
CommandCancelSearch}"
                  Content="Відміна"
                  Style="{StaticResource
ToolBarButtonStyle}"
                  Visibility="{Binding SearchProgress,
Converter={StaticResource DoubleToVisibilityConverter}}" />
              </ToolBar>
            </ToolBarTray>
          </Grid>

          <DockPanel
            x:Name="Panel1RowLeft"
            DockPanel.Dock="Left"
            Visibility="{Binding IsDataListBoxVisible,
Converter={StaticResource BooleanToVisibilityConverter}}">
            <ListBox
              x:Name="DataListBox"
              ItemsSource="{Binding DataPoints}"

SelectionChanged="DataListBox_SelectionChanged"
              SelectionMode="Single">
              <ListBox.Resources>
                <Style BasedOn="{StaticResource
CustomScrollBarStyle}" TargetType="ScrollBar" />
              </ListBox.Resources>
              <ListBox.ItemTemplate>
                <DataTemplate>
                  <TextBlock>
                    <Run FontWeight="SemiBold" Text="X:"
/>
                    <Run Text="{Binding X,
StringFormat=N2}" />
                    <Run FontWeight="SemiBold" Text=" Y:"
/>
                    <Run Text="{Binding Y,
StringFormat=N4}" />
                  </TextBlock>
                </DataTemplate>
              </ListBox.ItemTemplate>
            </ListBox>
          </DockPanel>

          <DockPanel
            x:Name="Panel2RowBottom"
            MinWidth="190"
            Panel.ZIndex="2"
            LastChildFill="True">
            <ListBox
              x:Name="ResultList"
              Margin="0"
              ItemsSource="{Binding
MatchingSequencesGroups}"

ScrollViewer.HorizontalScrollBarVisibility="Disabled"
              SelectedItem="{Binding SelectedSequence,
Mode=TwoWay}"
              VirtualizingPanel.IsVirtualizing="True"

VirtualizingPanel.VirtualizationMode="Recycling">
              <ListBox.Resources>
                <Style BasedOn="{StaticResource
CustomScrollBarStyle}" TargetType="ScrollBar" />
              </ListBox.Resources>
              <ListBox.ItemContainerStyle>
                <Style TargetType="ListBoxItem">
                  <Setter Property="Margin" Value="0,2" />
                  <Setter Property="Padding" Value="5,0,0,0"
/>
                  <Setter Property="Background"
Value="Transparent" />
                  <Setter Property="Foreground"
Value="Black" />
                  <Style.Triggers>
                    <Trigger Property="IsSelected"
Value="True">
                      <Setter Property="Background"
Value="{DynamicResource {x:Static
SystemColors.HighlightBrushKey}}" />
                      <Setter Property="Foreground"
Value="Black" />
                    </Trigger>
                  </Style.Triggers>
                </Style>
              </ListBox.ItemContainerStyle>
              <ListBox.ItemTemplate>
                <DataTemplate>
                  <Border
                    Padding="0,2"
```

```
                    BorderBrush="#DDD"
                    BorderThickness="0,0,0,1">
                    <StackPanel>
                      <TextBlock
                        FontWeight="Bold"

MouseDown="Sequence_MouseDown"
                        Text="{Binding GroupId,
StringFormat='Послідовність {0}'}" />
                      <TextBlock Margin="5,2,0,0"
Text="{Binding Sequence.Count, StringFormat='Кількість то-
чок: {0}'}" />
                      <TextBlock Margin="5,2,0,0"
Text="{Binding SimilarSequenceIndices.Count,
StringFormat='Схожих послідовностей: {0}'}" />
                      <TextBlock Margin="5,2,0,0"
Text="{Binding SequenceIndex, StringFormat='Початок:
{0}'}" />
                      <TextBlock Margin="5,2,0,0"
Text="{Binding ., Converter={StaticResource
EndIndexConverter}, StringFormat='Кінець: {0}'}" />
                      <Expander
                        Margin="0,5,0,0"
                        Header="Схожі послідовності"
                        IsExpanded="False">
                        <ListBox
                          MaxHeight="150"
                          BorderThickness="0"
                          ItemsSource="{Binding
SimilarSequenceIndices}"

ScrollViewer.VerticalScrollBarVisibility="Auto"

SelectionChanged="SimilarSequenceList_SelectionChanged"

VirtualizingPanel.IsVirtualizing="True"

VirtualizingPanel.VirtualizationMode="Recycling">
                          <ListBox.ItemContainerStyle>
                            <Style
TargetType="ListBoxItem">
                              <Setter Property="Margin"
Value="0,1" />
                              <Style.Triggers>
                                <Trigger
Property="IsSelected" Value="True">
                                  <Setter
Property="Background" Value="{DynamicResource {x:Static
SystemColors.HighlightBrushKey}}" />
                                  <Setter
Property="Foreground" Value="Black" />
                                </Trigger>
                              </Style.Triggers>
                            </Style>
                          </ListBox.ItemContainerStyle>
                          <ListBox.ItemTemplate>
                            <DataTemplate>
                              <TextBlock Margin="0,1"
Text="{Binding Converter={StaticResource
SimilarSequenceIndexToGroupIdConverter},
StringFormat='Послідовність: {0}'}" />
                            </DataTemplate>
                          </ListBox.ItemTemplate>
                        </ListBox>
                      </Expander>
                      <Expander
                        Margin="0,5,0,0"
                        Header="Точки послідовності"
                        IsExpanded="False">
                        <ListBox
                          MaxHeight="150"
                          BorderThickness="0"
                          ItemsSource="{Binding Sequence}"

ScrollViewer.VerticalScrollBarVisibility="Auto"

VirtualizingPanel.IsVirtualizing="True"

VirtualizingPanel.VirtualizationMode="Recycling">
                          <ListBox.ItemTemplate>
                            <DataTemplate>
                              <TextBlock Margin="0,1">
                                <Run
FontWeight="SemiBold" Text="X:" />
                                <Run Text="{Binding X,
StringFormat=N2}" />
                                <Run
FontWeight="SemiBold" Text=" Y:" />
                                <Run Text="{Binding Y,
StringFormat=N4}" />
                              </TextBlock>
                            </DataTemplate>
                          </ListBox.ItemTemplate>
                        </ListBox>
                      </Expander>
                    </StackPanel>
                  </Border>
                </DataTemplate>
              </ListBox.ItemTemplate>
            </ListBox>
          </DockPanel>

          <DockPanel
            x:Name="Panel4RowRight"
            Width="250"
            Panel.ZIndex="2"
            DockPanel.Dock="Right">
            <ListBox
              x:Name="SequenceLengthList"
              MaxHeight="{Binding
RelativeSource={RelativeSource AncestorType=Window},
Path=ActualHeight, Converter={StaticResource
SubtractConverter}, ConverterParameter=50}"
              ItemsSource="{Binding SequenceLengths}"
              ScrollViewer.VerticalScrollBarVisibility="Auto"

SelectionChanged="SequenceLengthList_SelectionChanged"
              VirtualizingPanel.IsVirtualizing="True"
```

```
VirtualizingPanel.VirtualizationMode="Recycling">
            <ListBox.Resources>
              <Style BasedOn="{StaticResource
CustomScrollBarStyle}" TargetType="ScrollBar" />
            </ListBox.Resources>
            <ListBox.ItemTemplate>
              <DataTemplate>
                <StackPanel>
                  <TextBlock FontWeight="Bold"
Text="{Binding Length, StringFormat={}{0} Точки:}" />
                  <TextBlock Margin="5,2,0,0">
                    <Run Text="Основних послідовнос-
тей: " />
                    <Run Text="{Binding
MainSequencesCount, Mode=OneWay}" />
                    <Run Text="{Binding
TotalSequencesCount, Mode=OneWay, StringFormat='
({0})'}" />
                  </TextBlock>
                  <TextBlock Margin="5,2,0,0"
Text="{Binding SimilarSequencesCount, Mode=OneWay,
StringFormat='Схожих послідовностей: {0}'}" />
                  <Expander
                    Margin="0,5,0,0"
                    Header="Послідовності"
                    IsExpanded="False">
                    <ListBox
                      MaxHeight="150"
                      BorderThickness="0"
                      ItemsSource="{Binding Sequences}"

ScrollViewer.VerticalScrollBarVisibility="Auto"

VirtualizingPanel.IsVirtualizing="True"

VirtualizingPanel.VirtualizationMode="Recycling">
                      <ListBox.ItemTemplate>
                        <DataTemplate>
                          <StackPanel>
                            <TextBlock
                              Margin="0,1"

MouseDown="Sequence_MouseDown"
                              Text="{Binding GroupId,
StringFormat='Послідовність: {0}'}" />
                            <Expander Margin="0,5,0,0">
                              <Expander.Header>
                                <TextBlock>
                                  <Run Text="Схожі по-
слідовності" />
                                  <Run Text="{Binding
SimilarSequenceIndices.Count, Mode=OneWay,
StringFormat=' ({0})'}" />
                                </TextBlock>
                              </Expander.Header>
                              <ListBox
                                MaxHeight="100"
                                BorderThickness="0"
                                ItemsSource="{Binding
SimilarSequenceIndices}"

ScrollViewer.VerticalScrollBarVisibility="Auto"

SelectionChanged="SimilarSequenceList_SelectionChanged"

VirtualizingPanel.IsVirtualizing="True"

VirtualizingPanel.VirtualizationMode="Recycling">
                                <ListBox.ItemTemplate>
                                  <DataTemplate>
                                    <TextBlock
Margin="0,1" Text="{Binding Converter={StaticResource
SimilarSequenceIndexToGroupIdConverter},
StringFormat='Послідовність: {0}'}" />
                                  </DataTemplate>
                                </ListBox.ItemTemplate>
                              </ListBox>
                            </Expander>
                          </StackPanel>
                        </DataTemplate>
                      </ListBox.ItemTemplate>
                    </ListBox>
                  </Expander>
                </StackPanel>
              </DataTemplate>
            </ListBox.ItemTemplate>
          </ListBox>
        </DockPanel>

        <DockPanel x:Name="Panel3RowRight">
          <Grid>
            <oxy:PlotView
              Margin="-5,-15,-15,0"
              Panel.ZIndex="1"
              Model="{Binding PlotModel}"
              RenderTransformOrigin="0.5,0.5" />
            <Button
              Margin="0,0,10,40"
              Panel.ZIndex="10"
              Command="{Binding ExportCommand}"
              Style="{StaticResource ExportButtonStyle}">
              <Image
                Width="16"
                Height="16"
                Source="{StaticResource Save_image}" />
            </Button>
          </Grid>
        </DockPanel>

      </DockPanel>


      <ProgressBar
        Grid.ColumnSpan="2"
        Height="4"
        VerticalAlignment="Top"
        Panel.ZIndex="101"
        Maximum="100"
```

```
        Minimum="0"
        Visibility="{Binding SearchProgress,
Converter={StaticResource DoubleToVisibilityConverter}}"
        Value="{Binding SearchProgress, Mode=OneWay}" />
    </Grid>
</Window>
```

5.3 Текст програми файлу FractalDimensionRangeWindow.xaml

```
<Window
    x:Class="ECSSTS.Models.FractalDimensionRangeWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/present
ation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:behaviors="clr-namespace:ECSSTS.Behaviors"
    xmlns:converter="clr-namespace:ECSSTS.Converters"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008
"
    xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
    xmlns:local="clr-namespace:ECSSTS.Models"
    xmlns:localData="clr-namespace:ECSSTS.Data"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    xmlns:oxy="http://oxyplot.org/wpf"
    Title="{Binding WindowTitle}"
    Width="1100"
    Height="500"
    MinWidth="870"
    MinHeight="240"
    WindowStartupLocation="CenterScreen">
    <Window.Resources>
        <converter:BooleanToVisibilityConverter
x:Key="BooleanToVisibilityConverter" />
        <converter:DoubleToVisibilityConverter
x:Key="DoubleToVisibilityConverter" />
    </Window.Resources>

    <DockPanel>
        <Grid Panel.ZIndex="100" DockPanel.Dock="Top">
            <ToolBarTray Style="{StaticResource
CustomToolBarTrayStyle}">
                <ToolBar Style="{StaticResource
CustomToolBarStyle}">
                    <Menu VerticalAlignment="Center">
                        <MenuItem Header="Файл"
Style="{StaticResource ToolBarMenuItemStyle}">
                            <MenuItem Command="{Binding
LoadFractalJsonCommand}" Header="Завантажити дані" />
                            <MenuItem Command="{Binding
SaveFractalJsonCommand}" Header="Зберегти дані" />
                            <MenuItem Command="{Binding
SaveToExcelCommand}" Header="Зберегти в Excel" />
                            <MenuItem
                                Header="Повне збереження"
                                IsCheckable="True"
                                IsChecked="{Binding IsFullSave}"
                                Style="{StaticResource
CheckBoxMenuItemStyle}"
                                ToolTip="Зберігати всі дані, включаючи
детальні послідовності для окремих вікон (збільшує розмір
файлу)" />
                        </MenuItem>
                        <MenuItem Header="Інфо"
Style="{StaticResource ToolBarMenuItemStyle}">
                            <MenuItem Style="{StaticResource
InfoMenuItemStyle}">
                                <MenuItem.Header>
                                    <StackPanel Orientation="Horizontal">
                                        <TextBlock Text="Fractal JSON: " />
                                        <TextBlock FontWeight="Bold"
Text="{Binding FractalJsonFileName, Mode=OneWay}" />
                                        <TextBlock Text=" (" />
                                        <TextBlock Text="{Binding
FractalJsonSavePath, Mode=OneWay}" />
                                        <TextBlock Text=")" />
                                    </StackPanel>
                                </MenuItem.Header>
                            </MenuItem>
                            <MenuItem Style="{StaticResource
InfoMenuItemStyle}">
                                <MenuItem.Header>
                                    <StackPanel Orientation="Horizontal">
                                        <TextBlock Text="Fractal за замов-
чуванням: " />
                                        <TextBlock FontWeight="Bold"
Text="Fractal.json" />
                                        <TextBlock Text=" (" />
                                        <TextBlock Text="{Binding
DataContext.FractalJsonSavePath,
RelativeSource={RelativeSource AncestorType=Window},
Mode=OneWay}" />
                                        <TextBlock Text=")" />
                                    </StackPanel>
                                </MenuItem.Header>
                            </MenuItem>
                            <MenuItem Style="{StaticResource
InfoMenuItemStyle}">
                                <MenuItem.Header>
                                    <StackPanel Orientation="Horizontal">
                                        <TextBlock Text="Excel: " />
                                        <TextBlock FontWeight="Bold"
Text="{Binding ExcelFileName, Mode=OneWay}" />
                                        <TextBlock Text=" (" />
                                        <TextBlock Text="{Binding
ExcelFilePath, Mode=OneWay}" />
                                        <TextBlock Text=")" />
                                    </StackPanel>
                                </MenuItem.Header>
                            </MenuItem>
                            <MenuItem Style="{StaticResource
InfoMenuItemStyle}">
```

```
                    <MenuItem.Header>
                      <StackPanel Orientation="Horizontal">
                        <TextBlock Text="JSON: " />
                        <TextBlock FontWeight="Bold"
Text="{Binding JsonFileName, Mode=OneWay}" />
                        <TextBlock Text=" (" />
                        <TextBlock Text="{Binding
JsonSavePath, Mode=OneWay}" />
                        <TextBlock Text=")" />
                      </StackPanel>
                    </MenuItem.Header>
                  </MenuItem>
                </MenuItem>
              </Menu>
              <CheckBox
                Margin="5,0,5,0"
                Content="Показати всі допуски"
                IsChecked="{Binding IsShowAllTolerances}"
                Style="{StaticResource BaseCheckBoxStyle}"
/>
              <Button
                Margin="5,0,5,0"
                Command="{Binding
ResetSelectionCommand}"
                Content="Скинути виділення"
                Style="{StaticResource ToolBarButtonStyle}"
                ToolTip="Очистити виділення в списку допу-
сків" />
              <Button
                Margin="5,0,5,0"
                Command="{Binding
OpenSeparateWindowCommand}"
                Content="Відкрити окремо"
                Style="{StaticResource ToolBarButtonStyle}"
/>
              <Button
                Margin="5,0,5,0"
                Command="{Binding ResetAxesCommand}"
                Content="Скинути вісі"
                Style="{StaticResource ToolBarButtonStyle}"
                ToolTip="Скинути масштаб осей графіка до
початкового стану" />
              <Button
                Margin="5,0,5,0"
                Command="{Binding
OpenConfidenceIntervalWindowCommand}"
                Content="Розрахунок довірчих інтервалів"
                Style="{StaticResource ToolBarButtonStyle}"
/>
              <Button
        Margin="5,0,5,0"
        Command="{Binding
OpenFractalToleranceComparisonCommand}"
        Content="Графік фракт. розмірн. від точності"
        Style="{StaticResource ToolBarButtonStyle}" />
            </ToolBar>
          </ToolBarTray>
        </Grid>
```

```
      <DockPanel Width="130" DockPanel.Dock="Left">
        <ListView
          x:Name="ToleranceListView"
          ItemsSource="{Binding ToleranceOptions}"
          SelectionMode="Multiple">
          <i:Interaction.Behaviors>
            <behaviors:SelectedItemsBehavior
ResetSelectionCommand="{Binding
ResetSelectionCommand}" SelectedItems="{Binding
SelectedTolerances}" />
          </i:Interaction.Behaviors>
          <i:Interaction.Triggers>
            <i:EventTrigger EventName="Loaded">
              <i:InvokeCommandAction
Command="{Binding ResetSelectionCommand}" />
            </i:EventTrigger>
            <i:PropertyChangedTrigger Binding="{Binding
ResetSelectionTrigger}">
              <i:InvokeCommandAction
Command="{Binding RelativeSource={RelativeSource
AncestorType=ListView},
Path=DataContext.SelectedItemsBehavior.ResetSelectionCom
mand}" />
            </i:PropertyChangedTrigger>
          </i:Interaction.Triggers>
          <ListView.Resources>
            <Style BasedOn="{StaticResource
CustomScrollBarStyle}" TargetType="ScrollBar" />
          </ListView.Resources>
          <ListView.View>
            <GridView>
              <GridViewColumn Width="50" Header="До-
пуск">
                <GridViewColumn.CellTemplate>
                  <DataTemplate>
                    <TextBlock
                      Margin="0"
                      Text="{Binding TolerancePercentage,
StringFormat={}{0:F2}%}"
                      TextAlignment="Right" />
                  </DataTemplate>
                </GridViewColumn.CellTemplate>
              </GridViewColumn>
              <GridViewColumn Width="60" Header="Фрак-
тало.">
                <GridViewColumn.CellTemplate>
                  <DataTemplate>
                    <TextBlock Text="{Binding Dimension,
StringFormat={}{0:F4}}" TextAlignment="Right" />
                  </DataTemplate>
                </GridViewColumn.CellTemplate>
              </GridViewColumn>
            </GridView>
          </ListView.View>
        </ListView>
      </DockPanel>

      <Grid>

        <oxy:PlotView
```

```
        Grid.Row="1"
        Margin="0,-15,0,0"
        Model="{Binding PlotModel}" />
        <Button
            Margin="0,0,20,55"
            Panel.ZIndex="10"
            Command="{Binding ExportCommand}"
            Style="{StaticResource ExportButtonStyle}">
            <Image Height="16" Width="16"
Source="{StaticResource Save_image}"/>
        </Button>
    </Grid>

  </DockPanel>
</Window>
```

5.4 Текст програми файлу FractalDimensionPlotWindow.xaml

```
<Window
  x:Class="ECSSTS.Models.FractalDimensionPlotWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/present
ation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="clr-namespace:ECSSTS"
  xmlns:oxy="http://oxyplot.org/wpf"
  Title="Графік фракталоподібної розмірності"
  Width="520"
  Height="400"
  MinWidth="400"
  MinHeight="300"
  WindowStartupLocation="CenterScreen">
  <Window.Resources>
    <Style x:Key="CustomInfoButtonStyle"
TargetType="Button">
      <Setter Property="Background" Value="#F5F5F5" />
      <Setter Property="BorderBrush" Value="#CCCCCC"
/>
      <Setter Property="BorderThickness" Value="1" />
      <Setter Property="Padding" Value="5" />
      <Setter Property="FontSize" Value="12" />
      <Setter Property="FontWeight" Value="Bold" />
      <Setter Property="Foreground" Value="#333333" />
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="Button">
            <Border
              Background="{TemplateBinding
Background}"
              BorderBrush="{TemplateBinding
BorderBrush}"
              BorderThickness="{TemplateBinding
BorderThickness}"
              CornerRadius="3">
              <ContentPresenter
                Margin="{TemplateBinding Padding}"
                HorizontalAlignment="Center"
                VerticalAlignment="Center" />
            </Border>
            <ControlTemplate.Triggers>
              <Trigger Property="IsMouseOver"
Value="True">
                <Setter Property="Background"
Value="#E0E0E0" />
                <Setter Property="BorderBrush"
Value="#999999" />
              </Trigger>
              <Trigger Property="IsPressed"
Value="True">
                <Setter Property="Background"
Value="#D0D0D0" />
              </Trigger>
            </ControlTemplate.Triggers>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </Window.Resources>
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <DockPanel Grid.Row="0">
      <ToolBarTray Style="{StaticResource
CustomToolBarTrayStyle}">
        <ToolBar Style="{StaticResource
CustomToolBarStyle}">
          <Button
            Width="25"
            Margin="8,0,0,0"
            Content="!"
            DockPanel.Dock="Right"
            FontWeight="Bold"
            Style="{StaticResource
BaseNoTextButtonStyle}">
            <Button.ToolTip>
              <ToolTip>
                <TextBlock Text="log(δ) - δ: довжина по-
слідовності&#x0A;log(M) - M: кількість послідовностей" />
              </ToolTip>
            </Button.ToolTip>
          </Button>
        </ToolBar>
        <ToolBar Style="{StaticResource
CustomToolBarStyle}">
          <TextBlock
            Margin="8,0,0,0"
            VerticalAlignment="Center"
            Text="{Binding Tolerance, StringFormat='Точ-
ність: {0:F2}'}" />
          <TextBlock
            Margin="8,0,0,0"
            VerticalAlignment="Center"
            Text="{Binding TotalMainSequences,
StringFormat='Основні: {0}'}" />
```

```
            <TextBlock
              Margin="8,0,0,0"
              VerticalAlignment="Center"
              Text="{Binding TotalSimilarSequences,
StringFormat='Схожі: {0}'}" />
            <TextBlock
              Margin="8,0,0,0"
              VerticalAlignment="Center"
              Text="{Binding TotalSequences,
StringFormat='Загалом: {0}'}" />
            <TextBlock
              Margin="8,0,0,0"
              VerticalAlignment="Center"
              Text="{Binding FractalDimension,
StringFormat='Фрактал: {0:F4}'}" />
          </ToolBar>
        </ToolBarTray>

      </DockPanel>
      <DockPanel Grid.Row="1">
        <Grid>
          <oxy:PlotView
            Grid.Row="0"
            Margin="0,0,2,0"
            Model="{Binding PlotModel}" />
          <Button
            Grid.Row="1"
            Margin="0,0,20,55"
            Panel.ZIndex="10"
            Command="{Binding ExportCommand}"
            Style="{StaticResource ExportButtonStyle}">
            <Image
              Width="16"
              Height="16"
              Source="{StaticResource Save_image}" />
          </Button>
        </Grid>
      </DockPanel>

    </Grid>
</Window>
```

5.5 Текст програми файлу ToleranceRangeWindow.xaml

```
<Window
  x:Class="ECSSTS.Models.ToleranceRangeWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/present
ation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:converter="clr-namespace:ECSSTS.Converters"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008
"
  xmlns:local="clr-namespace:ECSSTS.Models"
  xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
  Title="Діапазон допуску"
  Width="200"
  Height="180"
  ResizeMode="NoResize"
  WindowStartupLocation="CenterScreen"
  Background="{StaticResource PrimaryBackgroundBrush}"
  mc:Ignorable="d">
  <Window.Resources>
    <converter:StringToFloatConverter
x:Key="StringToFloatConverter" />
  </Window.Resources>
  <Grid Margin="5">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

    <Label
      Grid.Row="0"
      Grid.Column="0"
      VerticalAlignment="Center"
      Content="Мін. допуск (%):" />
    <TextBox
      x:Name="MinToleranceTextBox"
      Grid.Row="0"
      Grid.Column="1"
      Height="20"
      Margin="5,2"
      BorderThickness="0"
      TextAlignment="Center"
      MouseWheel="TextBox_MouseWheel">
      <TextBox.Text>
        <Binding
          Converter="{StaticResource
StringToFloatConverter}"
          Mode="TwoWay"
          Path="MinTolerancePercentage"
          StringFormat="{}{0:F2}"
          UpdateSourceTrigger="PropertyChanged">
          <Binding.ValidationRules>
            <DataErrorValidationRule
ValidatesOnTargetUpdated="True" />
          </Binding.ValidationRules>
        </Binding>
      </TextBox.Text>
    </TextBox>

    <Label
      Grid.Row="1"
      Grid.Column="0"
      VerticalAlignment="Center"
```

```
            Content="Макс. допуск (%):" />
        <TextBox
            x:Name="MaxToleranceTextBox"
            Grid.Row="1"
            Grid.Column="1"
            Height="20"
            Margin="5,2"
            BorderThickness="0"
            TextAlignment="Center"
            MouseWheel="TextBox_MouseWheel">
            <TextBox.Text>
                <Binding
                    Converter="{StaticResource
StringToFloatConverter}"
                    Mode="TwoWay"
                    Path="MaxTolerancePercentage"
                    StringFormat="{}{0:F2}"
                    UpdateSourceTrigger="PropertyChanged">
                    <Binding.ValidationRules>
                        <DataErrorValidationRule
ValidatesOnTargetUpdated="True" />
                    </Binding.ValidationRules>
                </Binding>
            </TextBox.Text>
        </TextBox>

        <Label
            Grid.Row="2"
            Grid.Column="0"
            VerticalAlignment="Center"
            Content="Крок (%):" />
        <TextBox
            x:Name="StepTextBox"
            Grid.Row="2"
            Grid.Column="1"
            Height="20"
            Margin="5,2"
            BorderThickness="0"
            TextAlignment="Center"
            MouseWheel="TextBox_MouseWheel">
            <TextBox.Text>
                <Binding
                    Converter="{StaticResource
StringToFloatConverter}"
                    Mode="TwoWay"
                    Path="Step"
                    StringFormat="{}{0:F2}"
                    UpdateSourceTrigger="PropertyChanged">
                    <Binding.ValidationRules>
                        <DataErrorValidationRule
ValidatesOnTargetUpdated="True" />
                    </Binding.ValidationRules>
                </Binding>
            </TextBox.Text>
        </TextBox>

        <TextBlock
            Grid.Row="3"
            Grid.Column="0"
            Grid.ColumnSpan="2"
            Margin="5,5,5,0"
            Text="{Binding IterationCount, StringFormat='Кіль-
кість ітерацій: {0}'}" />

        <Button
            Grid.Row="4"
            Grid.Column="0"
            Grid.ColumnSpan="2"
            Margin="5,10,5,0"
            Command="{Binding CommandSearchRange}"
            Content="Пошук"
            IsEnabled="{Binding IsSearchButtonEnabled}" />
    </Grid>
</Window>
```

6.1 Текст програми файлу Brushes.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation
" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">


    <SolidColorBrush x:Key="TextBrush" Color="{StaticResource
Text}" />

    <SolidColorBrush x:Key="MainButtonColorBrush"
Color="{StaticResource MainButtonColor}" />

    <SolidColorBrush x:Key="HoverButtonColorBrush"
Color="{StaticResource HoverButtonColor}" />

    <SolidColorBrush x:Key="PrimaryBackgroundBrush"
Color="{StaticResource PrimaryBackground}" />

    <SolidColorBrush x:Key="SecondaryBackgroundBrush"
Color="{StaticResource SecondaryBackground}" />

    <SolidColorBrush x:Key="SecondaryBackgroundHoverBrush"
Color="{StaticResource SecondaryBackgroundHover}" />

    <SolidColorBrush x:Key="BackgroundListViewItemBrush"
Color="{StaticResource BackgroundListViewItem}" />

    <SolidColorBrush x:Key="SelectBackgroundBrush"
Color="{StaticResource SelectBackground}" />

    <SolidColorBrush x:Key="TextBoxBackgroundBrush"
Color="{StaticResource TextBoxBackground}" />

    <SolidColorBrush x:Key="HoverBackgroundBrush"
Color="{StaticResource HoverBackground}" />

    <SolidColorBrush x:Key="HoverBorderBrush"
Color="{StaticResource HoverBorder}" />


    <SolidColorBrush x:Key="RenameButtonBackgroundBrush"
Color="{StaticResource RenameButtonBackground}" />

    <SolidColorBrush x:Key="RenameButtonTextBrush"
Color="{StaticResource RenameButtonText}" />

    <SolidColorBrush x:Key="DeleteButtonBackgroundBrush"
Color="{StaticResource DeleteButtonBackground}" />
```

```
    <SolidColorBrush                x:Key="DeleteButtonTextBrush"
Color="{StaticResource DeleteButtonText}" />
```

6.2 Текст програми файлу CheckBox.xaml

```
    <ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/present
    ation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

      <Style x:Key="BaseCheckBoxStyle"
    TargetType="CheckBox">
        <Setter Property="Background"
    Value="{DynamicResource MainButtonColorBrush}" />
        <Setter Property="Foreground" Value="Black" />
        <Setter Property="FontSize" Value="12" />
        <Setter Property="FontWeight" Value="DemiBold" />
        <Setter Property="Cursor" Value="Hand" />
        <Setter Property="Template">
          <Setter.Value>
            <ControlTemplate TargetType="CheckBox">
              <Border
                Padding="7,5,7,5"
                Background="{TemplateBinding Background}"
                BorderThickness="0">
                <ContentPresenter
                  HorizontalAlignment="Left"
                  VerticalAlignment="Center"
                  Content="{TemplateBinding Content}" />
              </Border>
              <ControlTemplate.Triggers>
                <Trigger Property="IsMouseOver"
    Value="True">
                  <Setter Property="Background"
    Value="{DynamicResource HoverButtonColorBrush}" />
                </Trigger>
                <Trigger Property="IsChecked" Value="True">
                  <Setter Property="Background"
    Value="{DynamicResource HoverButtonColorBrush}" />
                </Trigger>
              </ControlTemplate.Triggers>
            </ControlTemplate>
          </Setter.Value>
        </Setter>
      </Style>

      <Style x:Key="CheckBoxMenuItemStyle"
    TargetType="MenuItem">
        <Setter Property="VerticalAlignment" Value="Center" />
        <Setter Property="HeaderTemplate">
          <Setter.Value>
            <DataTemplate>
              <TextBlock
                Margin="-4,0,0,0"
                VerticalAlignment="Center"
                Text="{Binding}" />
            </DataTemplate>
          </Setter.Value>
        </Setter>
      </Style>

    </ResourceDictionary>
```

6.3 Текст програми файлу Colors.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation
" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Color x:Key="Text">#eeeffd</Color>
    <Color x:Key="MainButtonColor">#f0f0f0</Color>
    <Color x:Key="HoverButtonColor">#cdcdcd</Color>
    <Color x:Key="PrimaryBackground">#f0f0f0</Color>
    <Color x:Key="SecondaryBackground">#FF606060</Color>
    <Color x:Key="SecondaryBackgroundHover">#424246</Color>

    <Color x:Key="BackgroundListViewItem">#3f4253</Color>
    <Color x:Key="SelectBackground">#24262e</Color>
    <Color x:Key="TextBoxBackground">#19211b</Color>
    <Color x:Key="HoverBackground">#24262F</Color>
    <Color x:Key="HoverBorder">#24262F</Color>

    <Color
x:Key="RenameButtonBackground">#FF4CAF50</Color>
    <Color x:Key="RenameButtonText">#FFFFFFFF</Color>
    <Color x:Key="DeleteButtonBackground">#FFF44336</Color>
    <Color x:Key="DeleteButtonText">#FFFFFFFF</Color>
</ResourceDictionary>
```

6.4 Текст програми файлу Main.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation
" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Style x:Key="BaseButtonsStyle" TargetType="Button">
      <Setter Property="Margin" Value="1,0,1,0" />
      <Setter Property="Padding" Value="10,10,10,10" />
      <Setter Property="Background" Value="{DynamicResource
MainButtonColorBrush}" />
      <Setter Property="Foreground" Value="Black" />
      <Setter Property="FontSize" Value="12" />
      <Setter Property="FontWeight" Value="DemiBold" />
      <Setter Property="BorderBrush" Value="Transparent" />
      <Setter Property="Cursor" Value="Hand" />
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="Button">
            <Border
              Padding="0"
              Background="{TemplateBinding Background}"
              BorderThickness="0">
              <StackPanel HorizontalAlignment="Center"
Orientation="Horizontal">
                <TextBlock
                  Margin="-16,0,0,-4"
                  FontSize="18"
                  Text="+" />
                <TextBlock VerticalAlignment="Center"
Text="Add colum" />
              </StackPanel>
```

```
            </Border>
            <ControlTemplate.Triggers>
              <Trigger Property="IsMouseOver" Value="True">
                <Setter Property="Background"
Value="{DynamicResource HoverButtonColorBrush}" />
              </Trigger>
            </ControlTemplate.Triggers>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>

    <Style
      x:Key="ToolBarButtonStyle"
      BasedOn="{StaticResource BaseButtonsStyle}"
      TargetType="Button">
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="Button">
            <Border
              Padding="7,5,7,5"
              Background="{TemplateBinding Background}"
              BorderThickness="0">
              <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center" />
            </Border>
            <ControlTemplate.Triggers>
              <Trigger Property="IsMouseOver" Value="True">
                <Setter Property="Background"
Value="{DynamicResource HoverButtonColorBrush}" />
              </Trigger>
            </ControlTemplate.Triggers>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
    <Style
      x:Key="SearchButtonStyle"
      BasedOn="{StaticResource ToolBarButtonStyle}"
      TargetType="Button">
      <Setter Property="Opacity" Value="1" />
      <Style.Triggers>
        <Trigger Property="IsEnabled" Value="False">
          <Setter Property="Opacity" Value="0.5" />
        </Trigger>
      </Style.Triggers>
    </Style>

    <Style
      x:Key="BaseNoTextButtonStyle"
      BasedOn="{StaticResource ToolBarButtonStyle}"
      TargetType="Button">
      <Setter Property="Opacity" Value="1" />
      <Style.Triggers>
        <Trigger Property="IsEnabled" Value="False">
          <Setter Property="Opacity" Value="0.5" />
        </Trigger>
      </Style.Triggers>
    </Style>

    <Style x:Key="ExportButtonStyle" TargetType="Button">
      <!-- Розмір -->
      <Setter Property="Width" Value="30"/>
      <Setter Property="Height" Value="30"/>

      <!-- Фон, рамка, курсор -->
      <Setter Property="Background" Value="Transparent"/>
      <Setter Property="BorderThickness" Value="0"/>
      <Setter Property="Cursor" Value="Hand"/>

      <!-- Підказка -->
      <Setter Property="ToolTip" Value="Зберегти графік у JPG"/>

      <!-- Розташування (всередині Grid) -->
      <Setter Property="HorizontalAlignment" Value="Right"/>
      <Setter Property="VerticalAlignment" Value="Bottom"/>
      <Setter Property="Margin" Value="10"/>

      <!-- Шаблон -->
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="Button">
            <Border x:Name="border"
              Background="{TemplateBinding Background}"
              BorderBrush="#CCCCCC"
              BorderThickness="1"
              CornerRadius="4">
              <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center"/>
            </Border>

            <!-- Тригери для наведення -->
            <ControlTemplate.Triggers>
              <Trigger Property="IsMouseOver" Value="True">
                <Setter TargetName="border"
Property="Background" Value="#AAAAAA"/>
                <Setter TargetName="border"
Property="BorderBrush" Value="#888888"/>
              </Trigger>
              <Trigger Property="IsPressed" Value="True">
                <Setter TargetName="border"
Property="Background" Value="#999999"/>
              </Trigger>
            </ControlTemplate.Triggers>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
</ResourceDictionary>
```

6.5 Текст програми файлу MenuItem.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation
" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Style x:Key="ToolBarMenuItemStyle"
TargetType="MenuItem">
      <Setter Property="Margin" Value="1,0,1,0" />
      <Setter Property="Background" Value="{DynamicResource
MainButtonColorBrush}" />
      <Setter Property="Foreground" Value="Black" />
      <Setter Property="FontSize" Value="12" />
      <Setter Property="FontWeight" Value="DemiBold" />
      <Setter Property="BorderBrush" Value="Transparent" />
      <Setter Property="Cursor" Value="Hand" />
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="MenuItem">
            <Border
              x:Name="Border"
              Padding="7,5,7,5"
              Background="{TemplateBinding Background}"
              BorderBrush="{TemplateBinding BorderBrush}"
              BorderThickness="0">
              <Grid>
                <ContentPresenter
                  HorizontalAlignment="Center"
                  VerticalAlignment="Center"
                  ContentSource="Header" />
                <Popup
                  x:Name="SubMenuPopup"
                  AllowsTransparency="True"
                  Focusable="False"
                  IsOpen="{TemplateBinding IsSubmenuOpen}"
                  Placement="Bottom"
                  PopupAnimation="Fade">
                  <Border
                    Background="{DynamicResource
MainButtonColorBrush}"
```

```
                        BorderBrush="{DynamicResource
HoverButtonColorBrush}"
                        BorderThickness="1">
                        <StackPanel IsItemsHost="True" />
                      </Border>
                    </Popup>
                  </Grid>
                </Border>
                <ControlTemplate.Triggers>
                  <Trigger Property="IsMouseOver" Value="True">
                    <Setter TargetName="Border"
Property="Background" Value="{DynamicResource
HoverButtonColorBrush}" />
                  </Trigger>
                  <Trigger Property="IsSubmenuOpen" Value="True">
                    <Setter TargetName="Border"
Property="Background" Value="{DynamicResource
HoverButtonColorBrush}" />
                  </Trigger>
                </ControlTemplate.Triggers>
              </ControlTemplate>
            </Setter.Value>
          </Setter>
        </Style>

        <Style x:Key="InfoMenuItemStyle" TargetType="MenuItem">
          <Setter Property="IsEnabled" Value="False" />
          <Setter Property="Opacity" Value="0.75" />
        </Style>

        <Style x:Key="InfoPathMenuItemStyle"
TargetType="MenuItem">
          <Setter Property="Background" Value="{DynamicResource
MainButtonColorBrush}" />
          <Setter Property="IsEnabled" Value="False" />
          <Setter Property="IsCheckable" Value="False" />
        </Style>

        <Style x:Key="AutoSaveMenuItemStyle"
TargetType="MenuItem">
          <Setter Property="BorderThickness" Value="0" />
          <Style.Triggers>
            <Trigger Property="IsChecked" Value="True">
              <Setter Property="FontWeight" Value="Bold" />
            </Trigger>
            <Trigger Property="IsEnabled" Value="False">
              <Setter Property="Opacity" Value="0.7" />
            </Trigger>
          </Style.Triggers>
        </Style>
</ResourceDictionary>
```

6.6 Текст програми файлу ScrollBar.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation
" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Style x:Key="CustomScrollBarStyle" TargetType="ScrollBar">
    <Setter Property="Width" Value="10" />
    <Setter Property="Margin" Value="-3 -1 -4 -1" />
    <Setter Property="Background" Value="Transparent" />
    <Setter Property="Template">
      <Setter.Value>
        <ControlTemplate TargetType="ScrollBar">
          <Grid x:Name="Bg" Background="{TemplateBinding
Background}" Width="12">
            <Grid.RowDefinitions>
              <RowDefinition Height="12" />
              <RowDefinition Height="*" />
              <RowDefinition Height="12" />
            </Grid.RowDefinitions>
            <RepeatButton
                x:Name="PART_LineUpButton"
                Grid.Row="0"
                Command="{x:Static
ScrollBar.LineUpCommand}"
                Cursor="Hand">
              <Path
                  Data="M 0 4 L 4 0 L 8 4 Z"
                  Fill="{DynamicResource
MainButtonColorBrush}"
                  Stretch="Uniform"
                  Width="12"
                  Height="12" />
            </RepeatButton>
            <Track
                x:Name="PART_Track"
                Grid.Row="1"
                IsDirectionReversed="True">
              <Track.Thumb>
                <Thumb x:Name="Thumb">
                  <Thumb.Template>
                    <ControlTemplate TargetType="Thumb">
                      <Rectangle
                          x:Name="ThumbRect"
                          Fill="{DynamicResource
SecondaryBackgroundBrush}"
                          />
                      <ControlTemplate.Triggers>
                        <Trigger Property="IsMouseOver"
Value="True">
                          <Setter TargetName="ThumbRect"
Property="Fill" Value="{DynamicResource
SecondaryBackgroundHoverBrush}" />
                        </Trigger>
                      </ControlTemplate.Triggers>
                    </ControlTemplate>
                  </Thumb.Template>
                </Thumb>
              </Track.Thumb>
            </Track>
            <RepeatButton
                x:Name="PART_LineDownButton"
                Grid.Row="2"
                Command="{x:Static
ScrollBar.LineDownCommand}"
                Cursor="Hand">
              <Path
                  Data="M 0 0 L 4 4 L 8 0 Z"
                  Fill="{DynamicResource
MainButtonColorBrush}"
                  Stretch="Uniform"
                  Width="12"
                  Height="12" />
            </RepeatButton>
          </Grid>
        </ControlTemplate>
      </Setter.Value>
    </Setter>
  </Style>
</ResourceDictionary>
```

6.7 Текст програми файлу Separator.xaml

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <Style x:Key="BaseSeparatorStyle" TargetType="Separator">
    <Setter Property="Margin" Value="0" />
```

```
    <Setter Property="Width" Value="1" />
    <Setter Property="Background" Value="Black" />
  </Style>
</ResourceDictionary>
```

6.8 Текст програми файлу ToolBar.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/present
ation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <Style x:Key="CustomToolBarStyle"
TargetType="ToolBar">
    <Setter Property="Background" Value="{StaticResource
PrimaryBackgroundBrush}" />
    <Setter Property="Height" Value="30" />
    <Setter Property="Template">
      <Setter.Value>
        <ControlTemplate TargetType="ToolBar">
          <Border
              Background="{TemplateBinding
Background}"
              CornerRadius="0">
            <ToolBarPanel
                Margin="{TemplateBinding Padding}"
                Background="{TemplateBinding
Background}"
                IsItemsHost="True" />
          </Border>
        </ControlTemplate>
      </Setter.Value>
    </Setter>
  </Style>

  <Style x:Key="CustomToolBarTrayStyle"
TargetType="ToolBarTray">
    <Setter Property="Background" Value="{StaticResource
PrimaryBackgroundBrush}" />
    <Setter Property="Height" Value="30" />

  </Style>


</ResourceDictionary>
```

7.1 Текст програми файлу BooleanToVisibilityConverter.cs

```
using System;
using System.Globalization;
using System.Windows;
using System.Windows.Data;

namespace ECSSTS.Converters
{
  public class BooleanToVisibilityConverter :
IValueConverter
  {
    public object Convert(object value, Type targetType,
object parameter, CultureInfo culture)
    {
      if (value is bool isVisible)
      {
        return isVisible ? Visibility.Visible :
Visibility.Collapsed;
      }
      return Visibility.Collapsed; // За замовчуванням при-
ховуємо
    }

    public object ConvertBack(object value, Type targetType,
object parameter, CultureInfo culture)
    {
      if (value is Visibility visibility)
      {
        return visibility == Visibility.Visible;
      }
      return false;
    }
  }
}
```

7.2 Текст програми файлу BoolToIndexConverter.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Data;

namespace ECSSTS.Converters
{
  public class BoolToIndexConverter : IValueConverter
  {
    public object Convert(object value, Type targetType,
object parameter, CultureInfo culture)
    {
      if (value is bool boolValue)
```

```
        {
            return boolValue ? 1 : 0; // Якщо true, то індекс 0
(Всі послідовності), інакше 1 (Схожі за довжиною)
        }
        return 0;
    }

    public object ConvertBack(object value, Type targetType,
object parameter, CultureInfo culture)
    {
        if (value is int index)
        {
            return index == 1; // Якщо індекс 0, то true (Всі по-
слідовності), інакше false (Схожі за довжиною)
        }
        return true;
    }
  }
}
```

7.3 Текст програми файлу DataConverter.cs

```
using OxyPlot;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using ECSSTS.Data;
namespace ECSSTS.Converters
{
  public class DataConverter
  {
    // Метод для конвертації
ObservableCollection<DataExcel> в List<DataPoint>
    public static List<DataPoint>
ConvertToDataPoints(ObservableCollection<DataExcel>
dataExcelCollection)
    {
      var dataPoints = new List<DataPoint>();

      foreach (var item in dataExcelCollection)
      {
        // Створюємо DataPoint з значень X і Y в
DataExcel
        var dataPoint = new DataPoint(item.X, item.Y);
        dataPoints.Add(dataPoint);
      }

      return dataPoints;
    }
    public static List<DataExcel>
ConvertToDataExcel(List<DataPoint> dataPoints)
    {
      var dataExcelList = new List<DataExcel>();

      foreach (var point in dataPoints)
      {
        // Створюємо DataExcel з значень X і Y в
DataPoint
        var dataExcel = new DataExcel(point.X, point.Y);
        dataExcelList.Add(dataExcel);
      }

      return dataExcelList;
    }


    public static ObservableCollection<DataExcel>
ConvertToDataExcel(IEnumerable<DataPoint> dataPoints)
    {
      var dataExcelCollection = new
ObservableCollection<DataExcel>();

      foreach (var point in dataPoints)
      {
        // Створюємо DataExcel з значень X і Y в
DataPoint
        var dataExcel = new DataExcel(point.X, point.Y);
        dataExcelCollection.Add(dataExcel);
      }

      return dataExcelCollection;
    }

    public static IEnumerable<DataPoint>
ConvertToDataExcel(List<DataExcel> sequence)
    {
      foreach (var item in sequence)
      {
        yield return new DataPoint(item.X, item.Y);
      }
    }

  }
}
```

7.4 Текст програми файлу DoubleToBooleanConverter.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Data;

namespace ECSSTS.Converters
```

```
{
    public class DoubleToBooleanConverter : IValueConverter
    {
        public object Convert(object value, Type targetType,
object parameter, CultureInfo culture)
        {
            if (value is double progress)
            {
                bool invert = parameter != null &&
parameter.ToString() == "0";
                bool isActive = progress == 0;
                bool result = invert ? !isActive : isActive;
                Debug.WriteLine($"DoubleToBooleanConverter:
progress={progress}, invert={invert}, isActive={isActive},
result={result}");
                return result;
            }
            Debug.WriteLine("DoubleToBooleanConverter: Повер-
таємо true (default)");
            return true;
        }

        public object ConvertBack(object value, Type targetType,
object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

7.5 Текст програми файлу DoubleToVisibilityConverter.cs

```
using System;
using System.Globalization;
using System.Windows;
using System.Windows.Data;

namespace ECSSTS.Converters
{
    public class DoubleToVisibilityConverter : IValueConverter
    {
        public object Convert(object value, Type targetType,
object parameter, CultureInfo culture)
        {
            if (value is double progress)
            {
                bool invert = parameter != null &&
parameter.ToString() == "0";
                bool isVisible = progress > 0;
                return invert ? (isVisible ? Visibility.Collapsed :
Visibility.Visible) : (isVisible ? Visibility.Visible :
Visibility.Collapsed);
            }
            return Visibility.Collapsed;
        }

        public object ConvertBack(object value, Type targetType,
object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

7.6 Текст програми файлу EndIndexConverter.cs

```
using ECSSTS.Data;
using System;
using System.Globalization;
using System.Windows.Data;

namespace ECSSTS.Converters
{
    public class EndIndexConverter : IValueConverter
    {
        public object Convert(object value, Type targetType,
object parameter, CultureInfo culture)
        {
            if (value is MatchingSequenceGroup group &&
group.Sequence != null && group.Sequence.Count > 0)
            {
                return group.SequenceIndex + group.Sequence.Count
- 1;
            }
            return "Н/Д"; // Повертаємо "Н/Д" (не доступно),
якщо даних немає
        }

        public object ConvertBack(object value, Type targetType,
object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

7.7 Текст програми файлу SequenceLengthConverter.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Data;
```

```
namespace ECSSTS.Converters
{
   using ECSSTS.Data;
   public class SequenceLengthConverter : IValueConverter
   {
      public object Convert(object value, Type targetType,
object parameter, CultureInfo culture)
      {
         if (value is int groupId && parameter is
MatchingSequenceGroup group)
         {
            return $"Група {groupId} (Довжина:
{group.Sequence.Count})";
         }
         return value;
      }

      public object ConvertBack(object value, Type targetType,
object parameter, CultureInfo culture)
      {
         throw new NotImplementedException();
      }
   }
}
```

7.8 Текст програми файлу SimilarSequenceIndexToGroupIdConverter.cs

```
using System;
using System.Globalization;
using System.Linq;
using System.Windows;
using System.Windows.Data;

namespace ECSSTS.Converters
{
   public class SimilarSequenceIndexToGroupIdConverter :
IValueConverter
   {
      public object Convert(object value, Type targetType,
object parameter, CultureInfo culture)
      {
         if (value is int sequenceIndex)
         {
            // Отримуємо MainViewModel через DataContext
вікна
            var window = Application.Current.MainWindow;
            if (window?.DataContext is MainViewModel vm)
            {
               var group =
vm.MatchingSequencesGroups.FirstOrDefault(g =>
g.SequenceIndex == sequenceIndex);
               return group != null ? group.GroupId.ToString() :
sequenceIndex.ToString();
            }
         }
         return value;
      }

      public object ConvertBack(object value, Type targetType,
object parameter, CultureInfo culture)
      {
         throw new NotImplementedException();
      }
   }
}
```

7.9 Текст програми файлу StringToFloatConverter.cs

```
using System;
using System.Diagnostics;
using System.Globalization;
using System.Windows.Data;

namespace ECSSTS.Converters
{
   public class StringToFloatConverter : IValueConverter
   {
      public object Convert(object value, Type targetType,
object parameter, CultureInfo culture)
      {
         if (value == null)
         {
            return string.Empty;
         }

         if (value is double doubleValue)
         {
            return doubleValue.ToString("F2", culture); // Фор-
мат 10.00
         }

         return Binding.DoNothing;
      }

      public object ConvertBack(object value, Type targetType,
object parameter, CultureInfo culture)
      {
         if (value is string stringValue)
         {
            // Обробляємо порожній рядок
            if (string.IsNullOrWhiteSpace(stringValue))
            {
               Debug.WriteLine("ConvertBack: Порожній ря-
док, повертаємо 0.0");
               return 0.0;
            }

            // Видаляємо символ % (якщо він є через
StringFormat)
            stringValue = stringValue.Replace("%", "").Trim();
```

```
            // Парсимо число з урахуванням регіональних налаштувань
            if (double.TryParse(stringValue, NumberStyles.Any, culture, out double result))
            {
                Debug.WriteLine($"ConvertBack: Спарсено значення {result}");
                return result;
            }
            else
            {
                Debug.WriteLine($"ConvertBack: Некоректне значення '{stringValue}'");
            }
        }

        // Повертаємо Binding.DoNothing, щоб не скидати значення при некоректному вводі
        return Binding.DoNothing;
    }
  }
}
```

7.10 Текст програми файлу SubtractConverter.cs

```
using System;
using System.Globalization;
using System.Windows.Data;

namespace ECSSTS.Converters
{
  public class SubtractConverter : IValueConverter
  {
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
      if (value is double height && parameter is string subtractValue && double.TryParse(subtractValue, out double subtract))
      {
        return height - subtract;
      }
      return value;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
      throw new NotImplementedException();
    }
  }
}
```

7.11 Текст програми файлу TimeSpanToStringConverter.cs

```
using System;
using System.Windows.Data;

namespace ECSSTS.Converters
{
  public class TimeSpanToStringConverter : IValueConverter
  {
    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {
      if (value is TimeSpan timeSpan)
      {
        // Формат MM:SS
        return $"{(int)timeSpan.TotalMinutes:D2}:{timeSpan.Seconds:D2}";
      }
      return string.Empty;
    }

    public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {
      throw new NotImplementedException();
    }
  }
}
```

8.1 Текст програми файлу Graf.cs

```
using OxyPlot;
using OxyPlot.Series;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ECSSTS.ViewGraf
{
  public enum ColorDot
  {
    Red,
    Blue,
    Yellow,
    Orange,
    Black,
  }
  public class Graf
  {
    public LineSeries lineSeries = new LineSeries();
    private ScatterSeries _scatterSeries;
```

```
public ScatterSeries ScatterSeries
{
    get => _scatterSeries;
    set
    {
        _scatterSeries = value;
    }
}
public Graf()
{
    lineSeries = new LineSeries
    {
        Title = "Точка",
        MarkerStroke = OxyColors.Black,
    };
    _scatterSeries = new ScatterSeries
    {
        MarkerType = MarkerType.Circle,
        MarkerFill = OxyColors.Red,
        MarkerSize = 4
    };
}
public void SetColorDot(ColorDot colorDot)
{
    switch (colorDot)
    {
        case ColorDot.Red:
            ScatterSeries = RedDot();
            break;
        case ColorDot.Blue:
            ScatterSeries = BlueDot();
            break;
        case ColorDot.Yellow:
            ScatterSeries = YellowDot();
            break;
        case ColorDot.Orange:
            ScatterSeries = OrangeDot();
            break;
        case ColorDot.Black:
            ScatterSeries = BlackDot();
            break;
        default:
            ScatterSeries = RedDot();
            break;
    }
}
ScatterSeries RedDot()
{
    ScatterSeries scatterSeries = new ScatterSeries
    {
        MarkerType = MarkerType.Circle,
        MarkerFill = OxyColors.Red,
        MarkerSize = 4
    };
    return scatterSeries;
}
ScatterSeries BlueDot()
{
    ScatterSeries scatterSeries = new ScatterSeries
    {
        MarkerType = MarkerType.Circle,
        MarkerFill = OxyColors.Blue,
        MarkerSize = 4
    };
    return scatterSeries;
}
ScatterSeries YellowDot()
{
    ScatterSeries scatterSeries = new ScatterSeries
    {
        MarkerType = MarkerType.Circle,
        MarkerFill = OxyColors.Yellow,
        MarkerSize = 4
    };
    return scatterSeries;
}
ScatterSeries OrangeDot()
{
    ScatterSeries scatterSeries = new ScatterSeries
    {
        MarkerType = MarkerType.Circle,
        MarkerFill = OxyColors.Orange,
        MarkerSize = 4
    };
    return scatterSeries;
}
ScatterSeries BlackDot()
{
    ScatterSeries scatterSeries = new ScatterSeries
    {
        MarkerType = MarkerType.Circle,
        MarkerFill = OxyColors.Black,
        MarkerSize = 4
    };
    return scatterSeries;
}
  }
}
```

8.2 Текст програми файлу SelectedItemsBehavior.cs

```
using System.Collections;
using System.Collections.Specialized;
using System.Windows.Controls;
using System.Linq;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Input;
using ECSSTS.Bases;
using Microsoft.Xaml.Behaviors;

namespace ECSSTS.Behaviors
{
    public class SelectedItemsBehavior : Behavior<ListView>
    {
        public static readonly DependencyProperty
SelectedItemsProperty =
```

```
        DependencyProperty.Register("SelectedItems",
typeof(IList), typeof(SelectedItemsBehavior),
            new PropertyMetadata(null,
OnSelectedItemsChanged));

    public static readonly DependencyProperty
ResetSelectionCommandProperty =

DependencyProperty.Register("ResetSelectionCommand",
typeof(ICommand), typeof(SelectedItemsBehavior),
            new PropertyMetadata(null));

    public IList SelectedItems
    {
      get => (IList)GetValue(SelectedItemsProperty);
      set => SetValue(SelectedItemsProperty, value);
    }

    public ICommand ResetSelectionCommand
    {
      get =>
(ICommand)GetValue(ResetSelectionCommandProperty);
        set => SetValue(ResetSelectionCommandProperty,
value);
    }

    private bool _isUpdating;

    public SelectedItemsBehavior()
    {
      ResetSelectionCommand = new
RelayCommand(ResetSelection);
    }

    private static void
OnSelectedItemsChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
    {
      var behavior = (SelectedItemsBehavior)d;
      if (behavior.AssociatedObject != null)
      {
        var listView = behavior.AssociatedObject;
        if (e.OldValue is INotifyCollectionChanged
oldCollection)
        {
          oldCollection.CollectionChanged -=
behavior.CollectionChanged;
        }
        if (e.NewValue is INotifyCollectionChanged
newCollection)
        {
          newCollection.CollectionChanged +=
behavior.CollectionChanged;
        }
        behavior.SyncSelectedItems();
      }
    }

    protected override void OnAttached()
    {
      base.OnAttached();
      AssociatedObject.SelectionChanged +=
ListView_SelectionChanged;
      if (SelectedItems is INotifyCollectionChanged
collection)
      {
        collection.CollectionChanged += CollectionChanged;
      }
      SyncSelectedItems();
    }

    protected override void OnDetaching()
    {
      base.OnDetaching();
      AssociatedObject.SelectionChanged -=
ListView_SelectionChanged;
      if (SelectedItems is INotifyCollectionChanged
collection)
      {
        collection.CollectionChanged -= CollectionChanged;
      }
    }

    private void ListView_SelectionChanged(object sender,
SelectionChangedEventArgs e)
    {
      if (_isUpdating || SelectedItems == null) return;

      _isUpdating = true;
      try
      {
        var selectedItems =
AssociatedObject.SelectedItems.Cast<object>().ToList();
          if
(!selectedItems.SequenceEqual(SelectedItems.Cast<object>()))
          {
            SelectedItems.Clear();
            foreach (var item in selectedItems)
            {
              SelectedItems.Add(item);
            }
          }
      }
      finally
      {
        _isUpdating = false;
      }
    }

    private void CollectionChanged(object sender,
NotifyCollectionChangedEventArgs e)
    {
      if (_isUpdating) return;
      SyncSelectedItems();
    }

    private void SyncSelectedItems()
    {
      if (AssociatedObject == null || SelectedItems == null)
return;
```

```
        _isUpdating = true;
        try
        {
            var listView = AssociatedObject;
            var selectedItems =
SelectedItems.Cast<object>().ToList();
            if
(!listView.SelectedItems.Cast<object>().SequenceEqual(selecte
dItems))
            {
                listView.SelectedItems.Clear();
                foreach (var item in selectedItems)
                {
                    listView.SelectedItems.Add(item);
                }
            }
        }
        finally
        {
            _isUpdating = false;
        }
    }

    private void ResetSelection(object parameter)
    {
        if (AssociatedObject == null || SelectedItems == null)
return;

        _isUpdating = true;
        try
        {
            AssociatedObject.SelectedItems.Clear();
            SelectedItems.Clear();
        }
        finally
        {
            _isUpdating = false;
        }
    }
  }
}
```

8.3 Текст програми файлу ViewPlot.cs

```
using OxyPlot;
using OxyPlot.Series;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ECSSTS.Data;
namespace ECSSTS.ViewGraf
{
  public class ViewPlotModel : PlotModel
  {
    public enum PlotSelect
    {
      Main,
    }

    public ViewPlotModel()
    {
      Title = "My Series";
    }
    public ViewPlotModel(PlotSelect select)
    {
      switch (select)
      {
        case PlotSelect.Main:
          //Title = "My Series";

          break;
        default:
          break;

      }
    }

    public PlotModel AddLineSeries(PlotModel plotModel,
ObservableCollection<DataExcel> dataPoints, LineSeries
lineSeries)
    {
      // Очищаємо серію перед додаванням нових точок

      lineSeries.Points.Clear();
      // Додаємо точки до серії

lineSeries.Points.AddRange(Converters.DataConverter.Convert
ToDataPoints(dataPoints));

      // Додаємо серію до моделі
      //plotModel.Series.Clear();  // Очищаємо серії перед
додаванням нових
      // Додаємо серію до моделі
      plotModel.Series.Add(lineSeries);
      plotModel.InvalidatePlot(true);
      return plotModel;
    }

    public PlotModel ClearGrafFromDot(PlotModel
plotModel, Graf graf)
    {
      plotModel = ClearGraf(plotModel, graf);

      plotModel.Series.Add(graf.lineSeries);
      plotModel.InvalidatePlot(true);
      return plotModel;
    }
    public PlotModel ClearGraf(PlotModel plotModel, Graf
graf)
    {
      plotModel.Axes.Clear();
```

```
plotModel.Series.Clear();
graf.ScatterSeries.Points.Clear();
plotModel.InvalidatePlot(true);
return plotModel;
                }
            }
        }
```

ДОДАТОК В

Керівництво користувача

ЗАТВЕРДЖУЮ

Перший проректор
Українського державного
університету науки і
технологій

_____Анатолій РАДКЕВИЧ

ПРОГРАМА ДЛЯ ЕКСПЕРИМЕНТАЛЬНИХ ОБЧИСЛЮВАЛЬНИХ ДОСЛІДЖЕНЬ ВИЗНАЧЕННЯ САМОПОДІБНОСТІ ЧАСОВИХ РЯДІВ

Керівництво користувача.
Керівництво з розрахунку фракталоподібної розмірності

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1533-01-І3-01

Завідувач кафедри КІТ

_____Вадим ГОРЯЧКІН

Керівник розробки

_____Віктор ШИНКАРЕНКО

Виконавець

_____Данило УЛЬЯНЧЕНКО

Нормоконтролер

_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.1533-01-І3-01

ПРОГРАМА ДЛЯ ЕКСПЕРИМЕНТАЛЬНИХ ОБЧИСЛЮВАЛЬНИХ ДОСЛІДЖЕНЬ ВИЗНАЧЕННЯ САМОПОДІБНОСТІ ЧАСОВИХ РЯДІВ

Керівництво користувача

Керівництво з розрахунку фракталоподібної розмірності

Листів 13

2026

ЗМІСТ

ВСТУП

Програма ECSSTS (Experimental Computational Studies of Self-Similarity of Time Series) призначена для аналізу самоподібності (фракталоподібності) часових рядів. Вона дозволяє завантажувати дані, шукати схожі послідовності з точним або відносним допуском, обчислювати фракталоподібну розмірність, візуалізувати результати та оцінювати статистичну значущість за допомогою 95 % довірчих інтервалів. Розроблено на .NET 8 з використанням WPF, MVVM та OxyPlot. Програма працює на Windows 10+ з процесором Intel Core i5+ та 4 ГБ RAM.

1 ПРИЗНАЧЕННЯ І УМОВИ ВИКОРИСТАННЯ

Призначення: автоматизований аналіз часових рядів (фінансові, біомедичні, кліматичні дані) на самоподібність, з обчисленням фракталоподібної розмірності D та статистичною оцінкою.

Умови використання:

ОС: Windows 10/11 (64-біт).

Вимоги: .NET 8 Runtime, 100 МБ вільного місця.

Дані: Excel/JSON з колонками X (час/індекс) та Y (значення), до 35 000 точок.

Режим: офлайн, без інтернету. Використовувати в наукових, освітніх або прикладних цілях. Заборонено комерційне поширення без дозволу автора.

2 ПІДГОТОВКА ДО РОБОТИ

Встановіть .NET 8 Runtime з офіційного сайту Microsoft (якщо не встановлено).

1. Запустіть файл ECSSTS.exe.
2. Перевірте налаштування: у меню "Налаштування/Інфо" перегляньте шляхи файлів (Excel, JSON, Fractal JSON). Якщо потрібно, скиньте шляхи.
3. Підготуйте дані: створіть Excel-файл з двома колонками (X, Y).

Якщо плануєте пакетний аналіз – підготуйте кілька JSON-файлів з результатами.

3 ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

На рис. 2 представлено інтерфейс програми при першому вході.

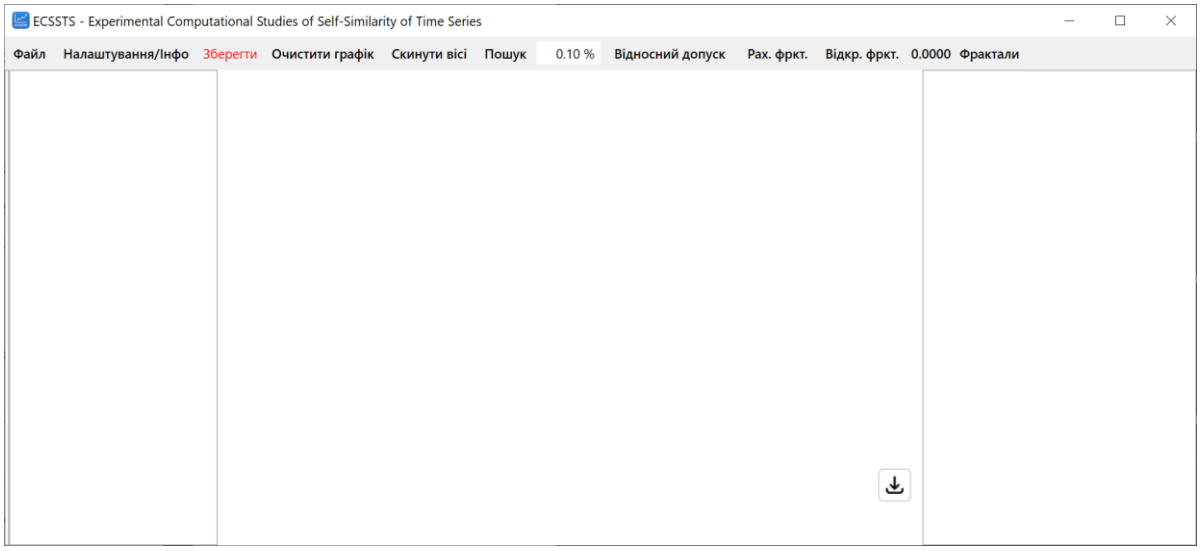


Рисунок 2 – Загальний вигляд головного вікна

1. Завантаження даних

Меню "Файл" → "Відкрити файл Excel" (або "Відкрити файл JSON").

Дані з’являться у списку точок (ліворуч) та на графіку (праворуч).

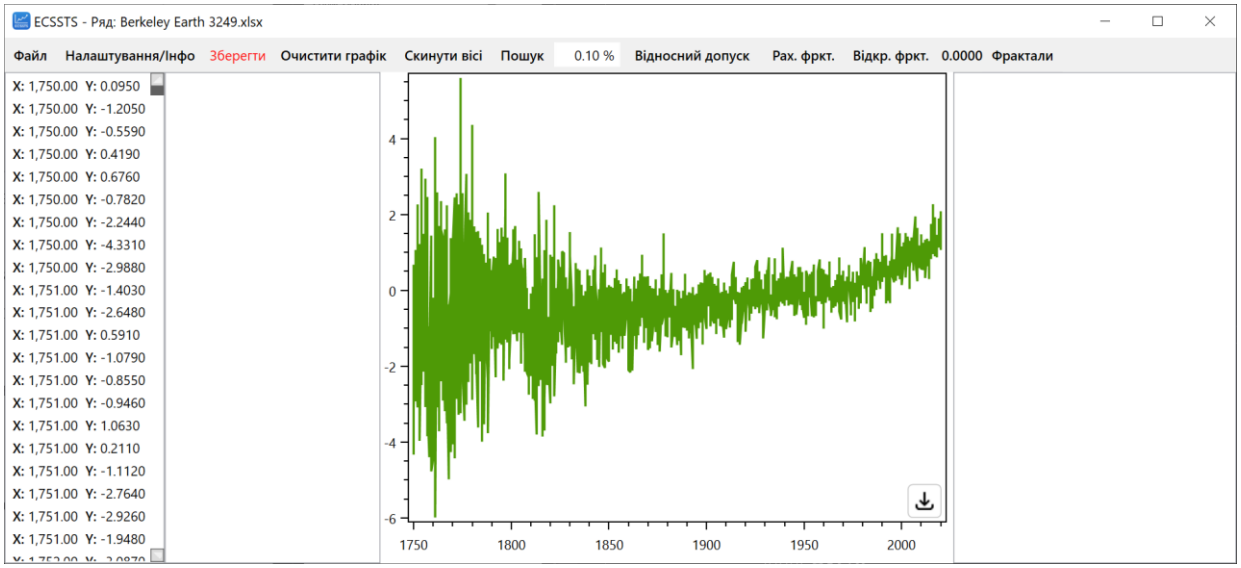


Рисунок 3 – Графік після завантаження даних

2. Налаштування допуску

- Введіть значення TolerancePercentage (рекомендовано 1.1–2.1 %) у поле праворуч від кнопки "Пошук".

- Увімкніть чекбокс "Відносний допуск" (рекомендовано для реальних даних різної амплітуди).

- Опціонально: увімкніть "Включати послідовності довжиною 1".

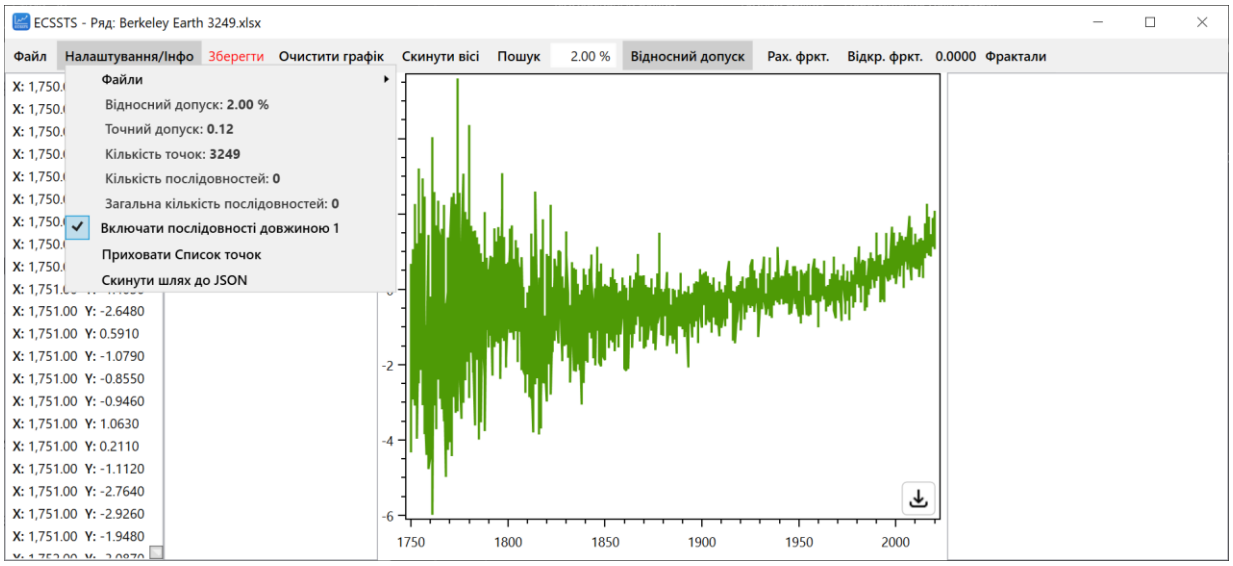


Рисунок 4 – Панель налаштувань допуску та чекбокси

3. Пошук схожих послідовностей

Натисніть кнопку "Пошук". Під час роботи видно прогрес та залишковий час (праворуч).

Після завершення:

- список груп за довжиною (ліворуч);

- детальний список груп (праворуч);

- графік з виділеними послідовностями (основні – червоним, схожі – синім).

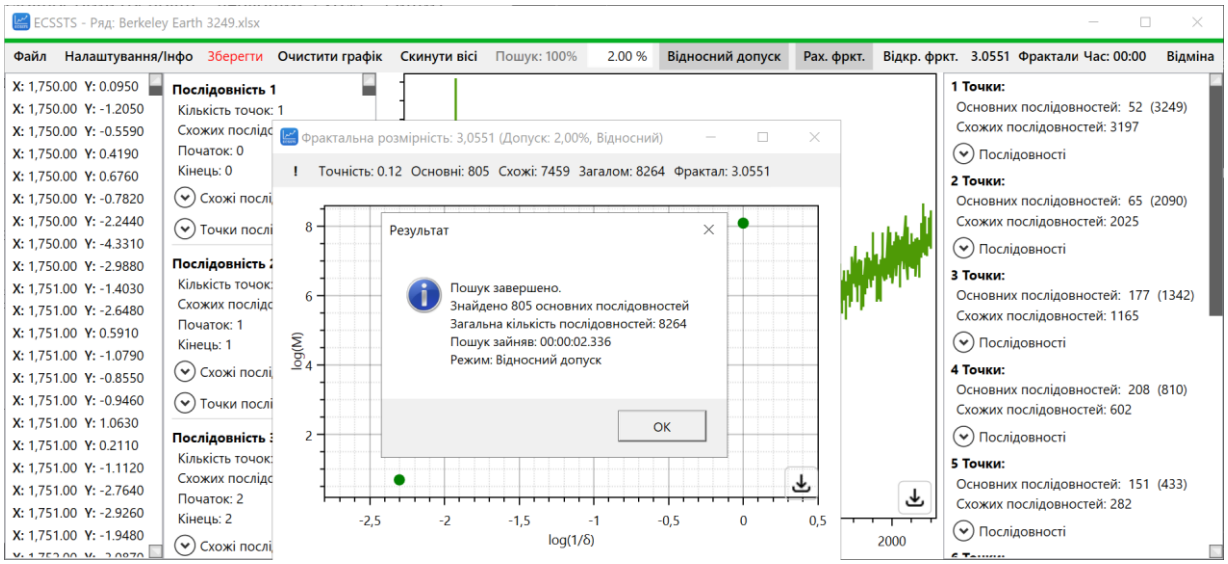


Рисунок 5 – Графік з виділеними послідовностями після пошуку

4. Обчислення фракталоподібної розмірності

Натисніть "Рах. фркт.".

Результат: значення D праворуч від поля допуску, автоматично відкрива-ється вікно з графіком залежності log(M) від log(1/δ).

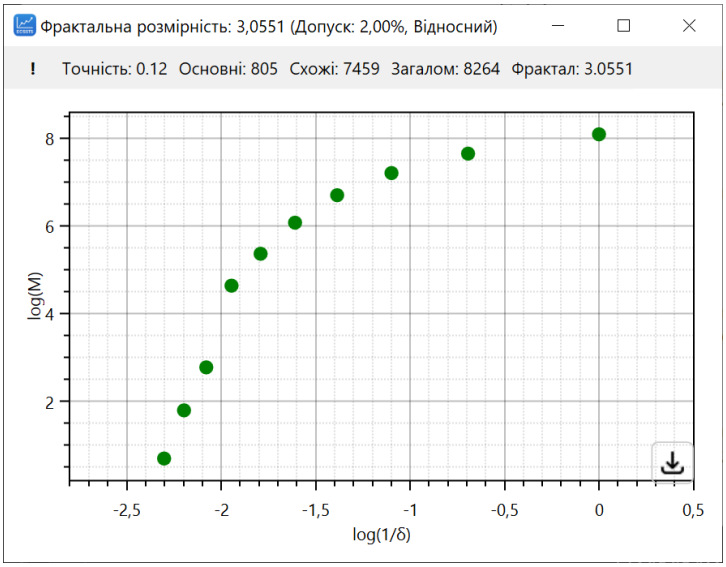


Рисунок 6 – Окреме вікно графіка фракталоподібної розмірності

5. Діапазонний аналіз

Подвійний клік на полі допуску → відкривається вікно налаштування ді-апазону.

Вкажіть мін/макс/крок (наприклад, 0.1–4.6 %, крок 0.5 %). Натисніть "По-шук".

Результат: вікно з графіком D для всіх допусків та таблицею.

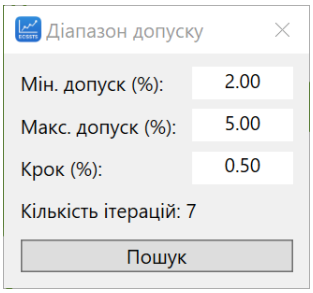


Рисунок 7 – Вікно налаштування діапазону допуску

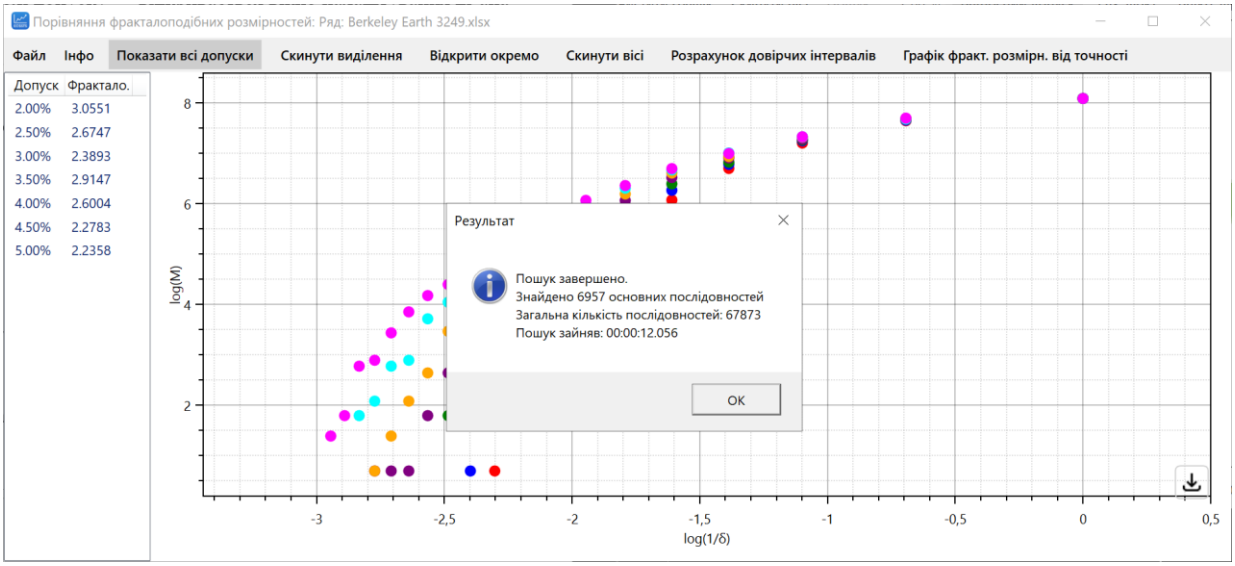


Рисунок 8 – Вікно діапазону фрактальної розмірності

6. Розрахунок довірчих інтервалів

У вікні діапазону натисніть "Розрахунок довірчих інтервалів".

Додайте кілька збережених JSON-файлів, увімкніть нормалізацію (реко-мендовано), натисніть "Порахувати".

Результат: таблиці середніх значень та 95 % інтервалів.

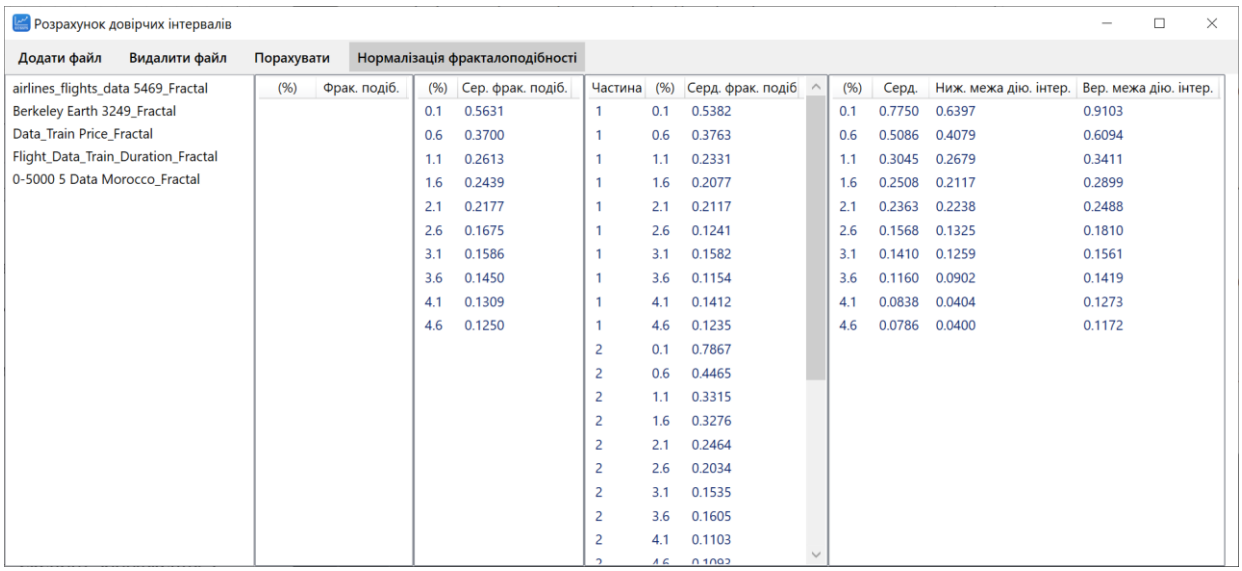


Рисунок 9 – Вікно розрахунку довірчих інтервалів

7. Експорт результатів

Кнопка з іконкою дискети (праворуч на графіках) – експорт зображення з ме-таданими.

Збереження у JSON/Excel – через меню "Файл".

4 АВТОЗБЕРЕЖЕННЯ ТА СКИДАННЯ

На рис. 10 і рис. 11 зображено фрагмент верхньої панелі головного вікна програми (меню "Файл" та "Налаштування/Інфо").

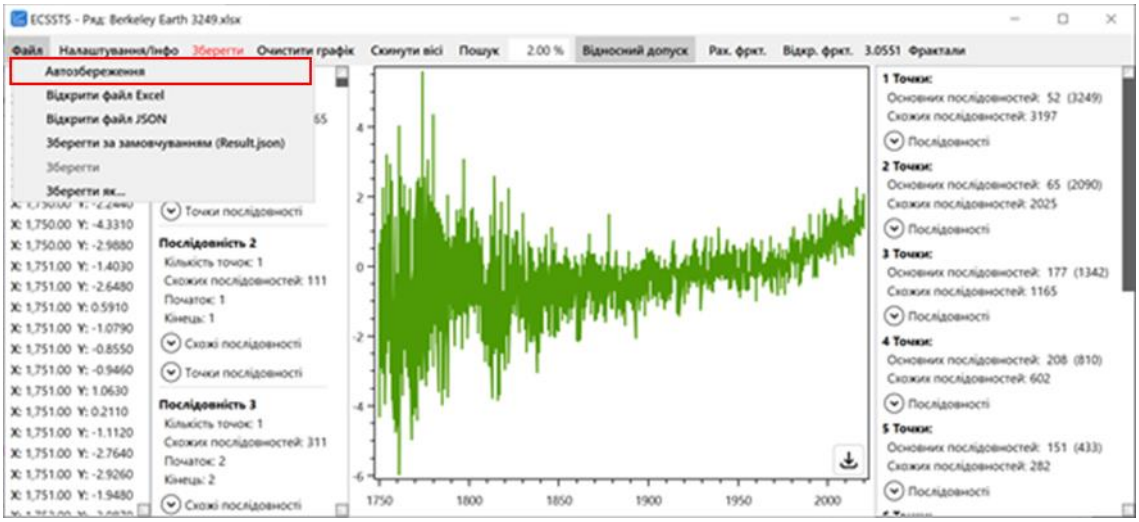


Рисунок 10. – Автозбереження

Позначено пункт меню "Автозбереження" (чекбокс).

Коли увімкнено, після кожного успішного пошуку результати автоматично зберігаються у файл Result.json (за замовчуванням у папці програми).

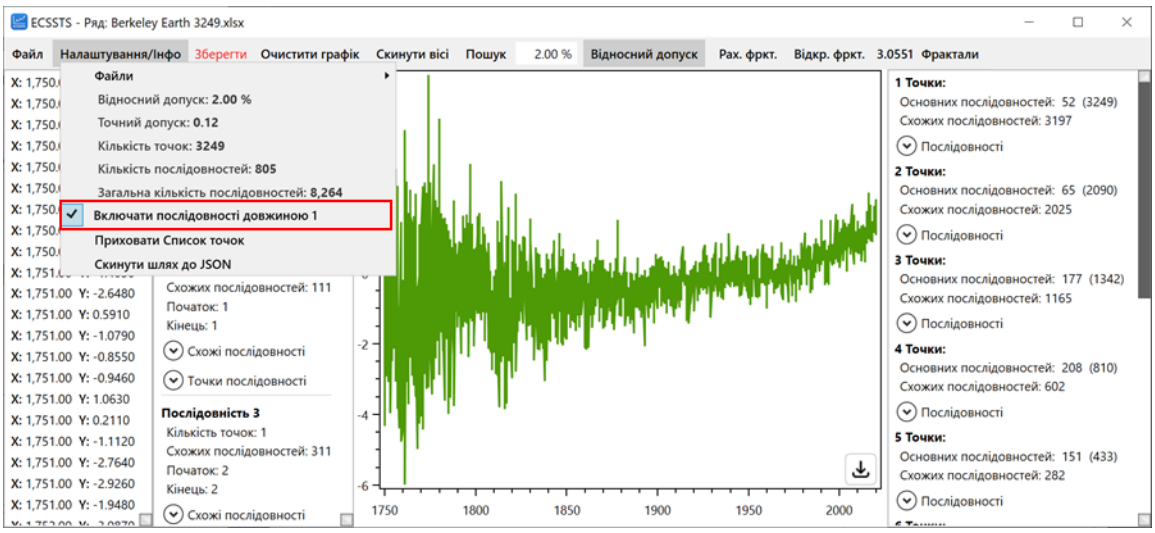


Рисунок 11. – Скидання шляху JSON

Позначено пункт меню "Скинути шлях до JSON".

Натиснення очищає збережені шляхи до JSON та Fractal JSON, скидає статус збереження (червоний колір) та дозволяє задати новий шлях збереження вручну.

Рекомендація: увімкніть автозбереження для уникнення втрати результатів при тривалих обчисленнях.

5 АВАРІЙНІ СИТУАЦІЇ

Помилка завантаження даних: Перевірте формат Excel (два стовпці X/Y, без порожніх/нечислових значень).

Перевищення допуску: Якщо TolerancePercentage > 4.6 % – програма попередить про можливу неточність.

Переповнення пам’яті: При >35 000 точках – скоротіть ряд або збільште RAM.

Помилка регресії: Якщо M(δ) = 0 або <2 точок для регресії – повідомлення "Недостатньо даних". Розв’язання: зменшіть допуск або включіть довжину 1.

Скасування пошуку: Натисніть "Відміна" – пошук зупиниться без втрати даних.

Критична помилка: Якщо програма падає – перевірте .NET 8, перезапустіть. Збережіть логи з debug-консолі.

6 РЕКОМЕНДАЦІЇ

Оптимальний допуск: 1.1–2.1 % у відносному режимі для мінімальної дисперсії.

Нормалізуйте дані перед аналізом для рядів різної амплітуди.

Для статистики – запустіть 10+ разів на подібних рядах і розрахуйте інтервали.

Експортуйте графіки для звітів – вони містять метадані.

ДОДАТОК Г

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ

ABSTRACTS
OF THE XIX INTERNATIONAL CONFERENCE
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
18-19, December, 2025

СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ

ПРИСВЯЧЕНО ПАМ'ЯТІ ПРОФЕСОРА ІГОРЯ ЖУКОВИЦЬКОГО

ТЕЗИ

ХІХ МІЖНАРОДНОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ
18-19 ГРУДНЯ 2025

ДНІПРО
2025

Експериментальні дослідження самоподібності часових рядів

Ульянченко Д. С., Шинкаренко В. І., Український державний університет науки і технологій, Україна

Самоподібність часових рядів – ключова властивість багатьох складних систем (фінансові ринки, біомедичні сигнали, кліматичні дані), яка проявляється в повторюваності структур на різних масштабах. Кількісною характеристикою цієї властивості є фракталоподібна розмірність – число, що показує, наскільки щільно схожі фрагменти заповнюють простір масштабів. На сьогодні відсутні доступні інструменти, які б дозволяли проводити автоматизований аналіз самоподібності з урахуванням точного та відносного допуску, а також оцінювати статистичну значущість результатів.

Розроблено програмне забезпечення ECSSTS (Experimental Computational Studies of Self-Similarity of Time Series) на платформі .NET 6 з використанням архітектури MVVM, асинхронного програмування (Task Parallel Library) та бібліотеки OxyPlot для візуалізації.

Розроблено програмне забезпечення мовою C# (.NET 8, WPF, MVVM, OxyPlot, MathNet.Numerics), яке реалізує метод виявлення схожих послідовностей для оцінки фрактальної розмірності D з підтримкою:

регулювання точності співпадіння TolerancePercentage (0.1–4.6 %, крок 0.5 %);

точного та відносного співпадіння;

нормалізації даних;

побудови 95 % довірчих інтервалів за t-розподілом Стьюдента;

пакетної обробки JSON-файлів з часовими рядами для дослідження.

Обчислення фракталоподібної розмірності виконується за таким алгоритмом: для кожної можливої довжини послідовності підраховується загальна кількість схожих фрагментів (основні + усі схожі, знайдені з заданим допуском); далі будується графік логарифмічної залежності кількості схожих послідовностей від їх довжини; за нахилом лінії регресії на цьому графіку визначається фракталоподібна розмірність (чим більший нахил – тим сильніша самоподібність).

Експериментальне дослідження проведено на шести наборах даних загальним обсягом понад 450 000 значень:

фрактальні синтетичні ряди (700 реалізацій × 500 значень = 350 000 значень);

нефрактальні реальні дані (споживання електроенергії в Марокко, пожежі в лісах, ціни авіаквитків, дані авіарейсів, глобальна температура Berkeley Earth).

Основні результати:

зі зростанням допуску середнє значення D нелінійно зменшується (від ≈ 0.64 при 0.1 % до ≈ 0.06 при 4.6 % для фрактальних рядів);

оптимальний діапазон допуску 1.1–2.1 % (відносний режим) забезпечує мінімальну дисперсію ($\sigma < 0.03$) та максимальну розрізнювальну здатність між фрактальними та нефрактальними рядами;

відносний режим переважає при аналізі реальних даних різної амплітуди; точний — стабільніший при малих допусках для однорідних синтетичних рядів;

статистична значущість відмінностей підтверджена t-тестом ($p < 0.001$, Cohen's $d > 3.5$) на всьому діапазоні допусків;

розроблений алгоритм та програмне забезпечення коректно розрізняють ряди з вираженою самоподібністю та без неї, що підтверджено відсутністю перетинів довірчих інтервалів між групами.

Програмне забезпечення може бути використано як універсальний інструмент для експериментальних досліджень самоподібності в різних галузях науки та техніки.