

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Міністерства освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Міністерства освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

СОКОЛОВСЬКИЙ ВЛАДИСЛАВ ВОЛОДИМИРОВИЧ

УДК: 004.42

ДИСЕРТАЦІЯ

АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ РЕГІОНАЛЬНОЇ СИСТЕМИ МОНІТОРИНГУ СТАНУ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ОБ'ЄКТІВ

121 Інженерія програмного забезпечення

12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ В.В. Соколовський

Науковий керівник: Жаріков Едуард В'ячеславович, д.т.н, професор

Київ - 2025

АНОТАЦІЯ

Соколовський В.В. Алгоритмічне та програмне забезпечення регіональної системи моніторингу стану потенційно небезпечних об'єктів. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії в галузі знань – 12 Інформаційні технології за спеціальністю – 121 Інженерія програмного забезпечення — Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, 2025.

На території України функціонує значна кількість споруд, класифікованих як потенційно небезпечні об'єкти, аварії на яких можуть призвести до виникнення техногенних надзвичайних ситуацій. В умовах збройної агресії РФ забезпечення безпеки життя та здоров'я населення, а також екологічної безпеки територій, де розташовані потенційно небезпечні об'єкти, набуває особливої актуальності. При оцінюванні ризиків необхідно враховувати не лише безпосередні наслідки аварій на об'єктах, але й потенційний вплив на населення, інфраструктуру прилеглих територій та інші промислові об'єкти.

За таких обставин існує нагальна потреба у створенні комплексних систем моніторингу стану потенційно небезпечних об'єктів на всіх рівнях: об'єктовому, місцевому, регіональному та державному. Хоча наразі функціонує значна кількість систем моніторингу стану потенційно небезпечних об'єктів об'єктового та місцевого рівнів, підвищення надійності експлуатації та рівня безпеки потенційно небезпечних об'єктів у спосіб удосконалення процесів збору, обробки та передачі даних в інформаційній системі моніторингу стану потенційно небезпечних об'єктів на основі розроблення спеціалізованого прикладного програмного забезпечення з використанням технології інтернету речей та розроблення програмного забезпечення та методів обробки інформації для регіональних систем моніторингу стану потенційно небезпечних об'єктів залишається актуальною науково-практичною задачею.

Сучасна система моніторингу стану потенційно небезпечних об'єктів являє собою програмно-технічний комплекс, що базується на принципах інженерії програмного забезпечення та використовує сучасні методи розробки програмного забезпечення. Впровадження такої системи дозволяє

підвищити ефективність експлуатації промислових об'єктів та поліпшити організацію захисту населення та робочого персоналу у спосіб раннього виявлення можливості виникнення надзвичайної ситуації. На регіональному та державному рівнях такі системи не створені.

Аналіз наявних систем моніторингу стану потенційно небезпечних об'єктів демонструє, що такі програмні комплекси здійснюють збір та обробку даних щодо ключових параметрів об'єкта та виконують їх параметричний контроль. Програмне забезпечення визначає, чи перевищують значення одного або декількох контрольованих параметрів об'єкта попередньо встановлені критичні рівні, або наближаються до них. При виявленні перевищення критичних значень параметрів, система автоматично генерує сповіщення для чергового персоналу про можливість виникнення надзвичайної ситуації та надає покрокові інструкції щодо необхідних дій для запобігання розвитку аварії та недопущення переходу об'єкта у стан надзвичайної ситуації.

Такий підхід суттєво обмежує часові рамки для впровадження протиаварійних заходів. До того ж сучасні промислові об'єкти характеризуються складною просторово розподіленою структурою, що охоплює значні території. Це створює підвищену вразливість компонентів програмно-апаратного комплексу моніторингу стану потенційно небезпечних об'єктів до електромагнітних завад.

Зазначені виклики та сучасні тенденції розвитку регіональних розподілених систем моніторингу стану потенційно небезпечних об'єктів формують актуальне науково-технічне завдання щодо вдосконалення теоретичних засад розробки програмного забезпечення як інформаційного об'єкта та створення ефективних алгоритмів для регіональних систем моніторингу стану потенційно небезпечних об'єктів, яке вирішується в межах цього дисертаційного дослідження.

Метою дисертаційного дослідження є підвищення надійності експлуатації та рівня безпеки потенційно небезпечних об'єктів у спосіб удосконалення процесів збору, обробки та передачі даних в інформаційній системі моніторингу стану потенційно небезпечних об'єктів на основі розроблення спеціалізованого прикладного програмного забезпечення з

використанням технології інтернету речей, методів завадостійкого кодування даних, моделювання процесів об'єкта моніторингу, а також методів прогнозування зміни його стану.

Для досягнення поставленої мети розроблено та теоретично обґрунтовано наукові положення, методологічні засади та практичні рекомендації щодо вибору методів, алгоритмів, а також вирішено проблема побудови архітектури програмного забезпечення на основі системного підходу до обробки даних моніторингу, прогнозування стану об'єктів та оптимізації процесів.

Досягнення поставленої мети забезпечено у спосіб розв'язання комплексу теоретичних, методологічних та практичних задач інженерії програмного забезпечення, які розглянуто в наступних розділах дисертаційного дослідження.

У першому розділі виконаний аналітичний аналіз сучасного стану впровадження регіональних інформаційних систем моніторингу стану потенційно небезпечних об'єктів, аналіз законодавчої бази та наукової літератури, а також виконано дослідження сучасних теоретичних та методологічних основ проєктування систем моніторингу стану потенційно небезпечних об'єктів, проаналізовані архітектурні та технологічні аспекти проєктування систем моніторингу. На підставі проведеного аналізу сформульовано науково-технічну задачу та визначено комплекс завдань дослідження, спрямованих на підвищення ефективності процесу проєктування регіональних систем моніторингу стану потенційно небезпечних об'єктів.

У другому розділі виконаний аналіз наявних систем моніторингу, а на основі аналізу запропоновано розв'язання проблеми побудови архітектури регіональної системи моніторингу стану потенційно небезпечних об'єктів на основі мережі Інтернету речей з топологією «зірка» з інтегрованою системою управління та предиктивними аналітичними підсистемами на об'єктовому, місцевому та регіональних рівнях. На основі аналізу наявних методів передачі даних в мережах Інтернету речей запропоновано та розроблено метод застосування завадостійкого коригувального коду з можливістю виявлення та виправлення багатобітових помилок передачі даних. Виконане тестування інформаційного об'єкта – програмної моделі каналу передачі даних підтвердило корегувальні можливості запропонованого методу виявлення та виправлення

помилки. Вирішена проблема побудови архітектури інформаційної системи, розроблене алгоритмічне та програмне забезпечення для програмованих давачів, які запропоновано використовувати як вузли збору інформації від джерел інформації у вигляді первинних перетворювачів на об'єктовому рівні.

У третьому розділі на основі аналізу публікацій доведено, що має сенс у складі регіональних інформаційних систем моніторингу стану потенційно небезпечних об'єктів мати підсистеми предиктивної аналітики як на об'єктовому рівні, так і на місцевому та регіональному рівнях. Запропоновано, що на об'єктовому рівні підсистема предиктивної аналітики повинна виконувати функції короткочасного прогнозування з метою раннього виявлення можливості виникнення надзвичайної ситуації на потенційно небезпечних об'єктах, що забезпечить збільшення часу (на три-чотири відліки) на реагування та впровадження протиаварійних заходів оперативним персоналом. Реалізовано алгоритмічне та програмне забезпечення для здійснення короткочасних прогнозів на базі використання методу найменших квадратів. Розглянуто використання на місцевому та регіональному рівнях підсистем предиктивної аналітики. Обґрунтовано, що підсистема предиктивної аналітики на місцевому та регіональному рівнях повинна мати можливість виконувати функції моделювання стану найбільш потенційно небезпечних об'єктів, а тому розглянуто моделювання процесів фільтрації на прикладі земляних напірних гідроспоруд — потенційно небезпечних об'єктів, аварії на яких призводять до масштабних наслідків. Моделювання запропоновано виконувати використовуючи запропонований модифікований метод сіток. Відзначено, що моделювання дозволяє оперативно отримати якісну картину можливого розвитку деструктивних процесів стану потенційно небезпечних об'єктів.

У четвертому розділі проаналізовано можливі методи оцінки якості алгоритмічного та програмного забезпечення. Розроблена методика оцінки якості архітектури, алгоритмічного та програмного забезпечення на основі використання експертного оцінювання. На основі розробленої методики виконано експертне оцінювання якості архітектур, а також алгоритмічного та програмного забезпечення. Експертне оцінювання підтвердило якість

прийнятих методів та алгоритмів при розробці типових рішень та рекомендацій, а саме: необхідність побудови регіональних систем моніторингу стану потенційно небезпечних об'єктів на основі мереж Інтернету речей, використання у якості вузла комп'ютерної мережі збору інформації від первинних перетворювачів програмованих давачів, використання для забезпечення завадостійкої передачі даних модифікованого методу виявлення та виправлення багатобітових помилок передачі даних на базі використання корегувальних кодів Хеммінга, а також необхідність використання на всіх рівнях системи моніторингу предиктивних аналітичних підсистем. Потрібно також відзначити, що розроблена регіональна система моніторингу стану потенційно небезпечних об'єктів має високі вагові коефіцієнти функціональної стійкості.

У дисертаційному дослідженні отримано низку **нових наукових результатів**:

Уперше розроблено архітектуру як ядро системи моніторингу стану потенційно небезпечних об'єктів, яка відрізняється від наявних використанням технологій Інтернету речей, забезпеченням завадостійкої передачі даних, можливістю прогнозування змін параметрів джерел небезпеки, що характеризують стан об'єкта, а також можливістю моделювання процесів, які призводять до зміни стану об'єкту моніторингу, що дозволяє підвищити надійність експлуатації та рівня безпеки потенційно небезпечних об'єктів.

Уперше розроблено архітектуру, алгоритмічне та програмне забезпечення для давачів вихідної інформації у складі завадостійкої регіональної системи моніторингу стану потенційно небезпечних об'єктів на базі технологій Інтернету речей, особливість яких полягає в тому, що давачі вихідної інформації можуть бути використані у системах моніторингу, у яких використовується побайтова передача даних у вигляді інформаційних блоків, що складаються з інформаційних та контрольних бітів, які перед передачею перемішують згідно зі схемою кодування, що дозволяє зменшити вплив електромагнітних завад при передачі даних та забезпечити цілісність даних.

Уперше розроблено метод виявлення та виправлення багатобітових помилок при передачі інформації, а також алгоритмічне та програмне

забезпечення на базі використання кодів Хеммінга, модифікованої схеми кодування та процедур кодування і декодування, який відрізняється процедурою перемішування бітів інформаційного блоку перед передачею в канал зв'язку, що дозволяє підвищити завадостійкість систем моніторингу стану потенційно небезпечних об'єктів.

Удосконалено теоретичні засади розроблення спеціалізованого програмного забезпечення аналітичних предиктивних підсистем регіональних систем моніторингу стану потенційно небезпечних об'єктів, які відрізняються від наявних тим, що розроблені шаблони для реалізації функції прогнозування змін значення параметрів джерел небезпеки в межах потенційно небезпечних об'єктів на об'єктовому рівні та функції моделювання стану потенційно небезпечних об'єктів на місцевому та регіональному рівнях, що дозволяє забезпечити раннє виявлення можливості виникнення надзвичайної ситуації та локалізацію можливих дефектів шляхом розвитку деструктивних процесів.

Уперше розроблене алгоритмічне та програмне забезпечення ітераційного методу розрахунку фільтрації води крізь напірні земляні гідроспоруди, яке відрізняється використанням двомірних математичних моделей для виявлення можливості руйнування гідроспоруди, що дозволяє спростити процес програмування аналітичних предиктивних підсистем на місцевому та регіональному рівнях, а також завчасно приймати управлінські рішення по запобіганню виникненню аварій та переходу їх в стан надзвичайної ситуації.

Основні наукові результати дисертаційного дослідження опубліковано у 7 наукових працях, з них: 4 статті в закордонних наукових виданнях проіндексовані в системі Scopus, а також 3 публікації в міжнародних і всеукраїнських конференціях

Ключові слова: інженерія програмного забезпечення, параметр, вузол, розподілена система, відмовостійкість, проблеми побудови архітектури, моніторинг, екологічний моніторинг, корегувальні коди, інтегрована система управління, бібліотека програмного забезпечення, давач, інтернет речей, мережа, інформаційні системи, потенційно небезпечні об'єкти, джерела інформації, вагові коефіцієнти, об'єкти критичної інформаційної інфраструктури.

ABSTRACT

Sokolovskyi V.V. Algorithmic and software development for a regional system of monitoring the condition of potentially hazardous objects. - A qualifying scientific work on the rights of a manuscript.

Dissertation for the degree of Doctor of Philosophy in Knowledge - 12 Information Technologies by specialization - 121 Software Engineering – National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Kyiv, 2025.

Ukraine has a significant number of facilities classified as potentially hazardous objects, accidents at which can lead to man-made emergencies. In the context of Russia’s armed aggression, ensuring the safety of life and health of the population, as well as the environmental safety of the territories where potentially hazardous objects are located, is of particular relevance. Risk assessments should consider not only the direct consequences of accidents at facilities, but also the potential impact on the population, the infrastructure of adjacent territories, and other industrial facilities.

In such circumstances, there is an urgent need to create comprehensive systems of monitoring the condition of potentially hazardous objects at all levels: object, local, regional, and state. Although a significant number of systems of monitoring the condition of potentially hazardous objects at the object and local levels are currently in operation, improving the reliability of operation and safety of potentially hazardous objects by improving the processes of collecting, processing, and transmitting data in the information system of monitoring the condition of potentially hazardous objects through the development of specialised application software using the Internet of Things technology and the development of software and methods for processing information for regional system of monitoring the condition of potentially hazardous objects remains an urgent scientific and practical task.

A modern system of monitoring the condition of potentially hazardous objects is a software and hardware complex based on the principles of software engineering and using modern software development methods. Implementation of such a system allows increase the efficiency of industrial facilities operation

and improve the organisation of protection of the public and workforce by early detection of the possibility of an emergency. No such systems have been created at the regional and national levels.

The analysis of the existing system of monitoring the condition of the potentially hazardous objects shows that such software complexes collect and process data on key parameters of the facility and perform their parametric control. The software determines whether the values of one or more monitored parameters of the facility exceed or approach predefined critical levels. If critical values are exceeded, the system automatically generates notifications for the personnel on duty about a potential emergency and provides step-by-step instructions on the necessary actions to prevent the development of an accident and prevent the facility from entering an emergency condition.

This approach significantly limits the timeframe for implementing emergency response measures. In addition, modern industrial facilities are characterised by a complex spatially distributed structure covering large areas. This creates an increased vulnerability of the components of the hardware and software system of monitoring the condition of potentially hazardous objects to electromagnetic interference.

These challenges and current trends in the development of regional distributed systems of monitoring the condition of potentially hazardous objects form an urgent scientific and technical task to improve the theoretical foundations of software development as an information object and create effective algorithms for regional systems of monitoring the condition of potentially hazardous objects, which is accomplished in this dissertation.

The purpose of the dissertation is increasing the reliability of operation and the level of safety of potentially hazardous objects by improving the processes of collecting, processing, and transmitting data in the information system of monitoring the condition of potentially hazardous objects through the development of specialised application software using the Internet of Things technology, methods of noise-resistant data coding, modelling the processes of the monitored object, as well as methods for predicting changes in its condition.

To achieve this goal, the scientific provisions, methodological principles and practical recommendations for the selection of methods and algorithms were developed and theoretically substantiated, and the software architecture design problems based on a systematic approach to monitoring data processing, object condition forecasting and process optimisation was solved.

The achievement of this goal is ensured by solving a set of theoretical, methodological and practical tasks of software engineering, which are discussed in the following sections of the dissertation.

The first section provides an analytical analysis of the current state of implementation of regional information systems of monitoring the condition of potentially hazardous objects, an analysis of the legislative framework and scientific literature, as well as a study of the current theoretical and methodological foundations for designing monitoring systems for potentially hazardous objects, and an analysis of the architectural and technological aspects of designing monitoring systems. Based on the analysis, a scientific and technical task was formulated and a set of research objectives was identified aimed at improving the efficiency of the design of regional monitoring systems for potentially hazardous objects.

The second section analyses the existing monitoring systems and, based on the analysis, proposes a solution to the architecture design problem of a regional system for monitoring the condition of potentially hazardous objects based on the Internet of Things network with a star topology with an integrated control system and predictive analytical subsystems at the object, local and regional levels. Based on the analysis of existing methods of data transmission on the Internet of Things networks, a method for applying a noise-resistant error-correcting code with the ability to detect and correct multiple data transmission errors is suggested and developed. The performed testing of the information object — the software model of the data transmission channel — confirmed the corrective capabilities of the proposed method of error detection and correction. The architecture design problem of the information system was solved, algorithmic and software for programmable sensors were developed, which are proposed to be used as nodes for collecting information from information sources in the form of primary converters at the object level.

In the third section, based on the analysis of publications, it is proved that it makes sense to have predictive analytics subsystems both at the object level and at the local and regional levels as part of regional information systems of monitoring the status of potentially hazardous objects. It is proposed that at the object level, the predictive analytics subsystem should perform the functions of short-term forecasting to detect early the possibility of an emergency at potentially hazardous objects, which will increase the time (from three to four counts) for response and implementation of emergency measures by operational personnel. The algorithmic and software for short-term forecasts based on the least squares method have been implemented. The use of predictive analytics subsystems at the local and regional levels is considered. It is substantiated that the subsystem of predictive analytics at the local and regional levels should be able to perform the functions of modelling the condition of the most potentially hazardous objects, and therefore the modelling of filtration processes is considered on the example of earthen pressure hydraulic structures — potentially hazardous objects, accidents on which lead to large-scale consequences. The modelling is proposed to be performed using the proposed modified grid method. It is noted that modelling allows to quickly obtain a qualitative picture of the possible development of destructive processes in the condition of potentially hazardous objects.

The fourth section analyses possible methods for assessing the quality of algorithmic and software. A methodology for assessing the quality of architecture, algorithmic and software based on the use of expert evaluation has been developed. Based on the developed methodology, an expert evaluation of the quality of architectures, as well as algorithmic and software, was performed. The expert evaluation confirmed the quality of the adopted methods and algorithms in the development of standard solutions and recommendations, namely: the need to build regional systems of monitoring the condition of potentially hazardous objects based on the Internet of Things networks, the use of information collection from primary converters of programmable sensors as a computer network node, the use of a modified method for detecting and correcting multiple data transmission errors based on the use of the Hamming error-correcting codes to ensure interference-free data transmission. It should also be noted that the developed regional system

of monitoring the condition of potentially hazardous objects has high weighting coefficients of functional stability.

The dissertation research has obtained a number of **new scientific results**:

For the first time, an architecture has been developed as the core of a system of monitoring the condition of potentially hazardous objects, which differs from existing ones by using Internet of Things technologies, ensuring noise-resistant data transmission, the ability to predict changes in the parameters of hazard sources that characterise the condition of the object, as well as the ability to model processes that lead to changes in the condition of the monitored object, which makes it possible to increase the reliability of operation and the level of safety of potentially hazardous objects.

For the first time, the architecture, algorithmic and software for output information sensors as part of a noise-resistant regional system of monitoring the condition of potentially hazardous objects based on Internet of Things technologies have been developed, the peculiarity of which is that the output information sensors can be used in monitoring systems that use byte data transmission in the form of information blocks consisting of information and control bits, which are mixed before transmission according to the coding scheme, which allows reducing the impact of electromagnetic interference during data transmission and ensure data integrity.

For the first time, a method for detecting and correcting multi-bit errors in information transmission, as well as algorithmic and software based on the use of Hamming codes, a modified coding scheme and encoding and decoding procedures, which differs in the procedure of mixing bits of an information block before transmission to the communication channel, which allows increasing the noise immunity of systems of monitoring the condition of potentially hazardous objects.

The theoretical foundations for the development of specialised software for analytical predictive subsystems of regional systems of monitoring the condition of potentially hazardous objects have been improved, which differ from the existing ones in that templates have been developed to implement the function of predicting changes in the value of parameters of hazard sources within potentially hazardous

objects at the object level and the function of modelling the condition of potentially hazardous objects at the local and regional levels, which allows for early detection of the opportunity of an emergency and localisation of possible defects through the development of destructive processes.

For the first time, algorithmic and software for the iterative method of calculating water filtration through pressure earthen hydraulic structures was developed, which is distinguished using two-dimensional mathematical models to identify the possibility of hydraulic structure destruction, which makes it possible to simplify the process of programming analytical predictive subsystems at the local and regional levels, as well as to make management decisions in advance to prevent accidents and their transition to a condition of emergency.

Main scientific results of the dissertation research were published in 7 scientific papers, including: 4 articles in foreign scientific editions indexed in the Scopus system, as well as 3 publications in international and national conferences.

Keywords: software engineering, parameter, node, distributed system, fault tolerance, architecture design problems, monitoring, environmental monitoring, error-correcting codes, integrated control system, software library, sensor, Internet of Things, network, information systems, potentially hazardous objects, information sources, weighting factors, critical information infrastructure objects.

Список публікацій здобувача/List of publications of the applicant:

Статті у закордонному фаховому виданні третього квартиля (Q3), які проіндексовані в базі даних Scopus / the articles published in a foreign scientific journal classified as third quartile (Q3) and indexed in the Scopus database:

1. Vladyslav Sokolovskyi, & Eduard Zharikov. (2023). Architectural solution for the distribution of software and hardware systems for monitoring potentially unsafe objects. GEOMATE Journal, 25(109), 141–148. Retrieved from <https://geomatejournal.com/geomate/article/view/4057>

2. Sokolovskyi, V., Zharikov, E., & Telenyk, S. (2024). Development of the method of detecting and correcting data transmission errors in IoT systems for monitoring the state of objects. Eastern-European Journal of Enterprise Technologies, 1(9 (127), 22–33. <https://doi.org/10.15587/1729-4061.2024.298476>

3. Sokolovskyi, V., Zharikov, E., & Telenyk, S. (2024). Using expert evaluation for selecting an architectural solution for a specialized software system that monitors the state of potentially hazardous facilities. Eastern-European Journal of Enterprise Technologies, 5(3 (131), 27–40. <https://doi.org/10.15587/1729-4061.2024.312886>

Статті у закордонному фаховому виданні четвертого квартиля (Q4), які проіндексовані в базі даних Scopus / the articles published in a foreign scientific journal classified as fourth quartile (Q4) and indexed in the Scopus database:

4. Sokolovskyi, V., Zharikov, E., & Telenyk, S. (2024). Software and algorithmic support as part of regional systems for monitoring the state of objects for calculation of filtration through earthen hydraulic structures. Machinery & Energetics, 15(2), 130-144. <https://doi.org/10.31548/machinery/2.2024.130>

Публікації у матеріалах наукових конференцій / the publications in the proceedings of the scientific conferences:

5. Соколовський В.В., Жаріков Е.В. Архітектура програмно-апаратної системи моніторингу стану об'єктів підвищеної небезпеки з можливістю прогнозування виникнення надзвичайної ситуації // III Всеукраїнська науково-практична конференція молодих вчених та студентів «Інженерія

програмного забезпечення і передові інформаційні технології»(SoftTech-2022), 22-25 листопада. – Київ: НТУУ "КПІ ім. І. Сікорського", 2022. – С. 64-68
https://drive.google.com/file/d/1-kJ75f9HKjfQ4pL7SvNf_uW4NI2eb9HL

6. Соколовський В. В., Жаріков Е. В. Архітектурне рішення та програмне забезпечення програмованих давачів інформації для побудови регіональних IoT систем моніторингу стану потенційно небезпечних об'єктів. VI Міжнародна науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології (SOFT TECH-2024)» 21-23 травня 2024 року, м.Київ,с.128-135.
<https://drive.google.com/file/d/18bZ9QBure7U08rbqiHmxWglTrK1C9D4L>

7. Соколовський В. В., Жаріков Е. В. Прогнозування в системах моніторингу стану об'єктів підвищеної небезпеки регіонального рівня. VII Міжнародна науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології (SOFT TECH-2024)» 20-22 листопада 2024 року, м.Київ, с. 101-106.
<https://drive.google.com/file/d/1O70Ysxe-z3SS82UqEaBdEXR2VdRW3NwG>

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	20
ВСТУП	21
1 ФОРМУЛЮВАННЯ ТА ОБҐРУНТУВАННЯ ДОСЛІДНИЦЬКОЇ ЗАДАЧІ НА ОСНОВІ АНАЛІЗУ АКТУАЛЬНИХ ПІДХОДІВ ДО ПРОЄКТУВАННЯ СИСТЕМ МОНІТОРИНГУ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ОБ’ЄКТІВ	32
1.1 Аналіз сучасного стану системи моніторингу потенційно небезпечних об’єктів	32
1.2 Аналіз чинної законодавчої бази, суті попередніх розробок, основні висновки, наявність суперечностей у науковій літературі	35
1.3 Теоретичні та методологічні основи проєктування систем моніторингу стану потенційно небезпечних об’єктів в Україні та світі	38
1.4 Архітектурні та технологічні аспекти розробки сучасних систем моніторингу стану потенційно небезпечних об’єктів	42
1.5 Формулювання науково-технічної задачі та завдань дослідження	44
1.6 Висновки до розділу 1	45
2 АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПІДВИЩЕННЯ ЗАВАДОСТІЙКОСТІ МЕРЕЖ СИСТЕМ МОНІТОРИНГУ СТАНУ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ОБ’ЄКТІВ НА ОСНОВІ АРХІТЕКТУРИ ІНТЕРНЕТУ РЕЧЕЙ.....	47
2.1 Аналіз архітектури та особливостей імплементації інфраструктури інтернету речей в системах моніторингу стану потенційно небезпечних об’єктів	48
2.2 Архітектура регіональної системи моніторингу стану потенційно небезпечних об’єктів на основі інфраструктури інтернету речей.....	55
2.3 Методологічні засади розроблення алгоритмічного забезпечення розподіленої регіональної системи моніторингу потенційно небезпечних об’єктів	58

2.3.1	Алгоритмічне забезпечення об'єктових систем моніторингу стану потенційно небезпечних об'єктів	60
2.3.2	Алгоритмічне забезпечення місцевих систем моніторингу стану потенційно небезпечних об'єктів	63
2.3.3	Алгоритмічне забезпечення регіональної системи моніторингу стану потенційно небезпечних об'єктів	67
2.4	Дослідження методів завадостійкого кодування на основі кодів Хеммінга для забезпечення надійності передачі даних в мережах Інтернету речей.....	70
2.5	Теоретико-методологічні засади розробки алгоритмічного забезпечення кодів Хеммінга	77
2.6	Метод виявлення та виправлення багатобітових помилок на основі модифікації кодів Хеммінга.....	82
2.6.1	Модифікована схема завадостійкого кодування для систем моніторингу потенційно небезпечних об'єктів.....	83
2.6.2	Алгоритм кодування інформаційних блоків	84
2.6.3	Алгоритм декодування	87
2.7	Особливості програмної реалізації модифікованого коду Хеммінга	90
2.7.1	Проектування та реалізація параметричного класу Matrix2D для оптимізованої обробки двовимірних структур даних.....	90
2.7.2	Проектування та імплементація програмного забезпечення HammingCodec для реалізації завадостійкого кодування з розширеними можливостями діагностики	92
2.7.3	Проектування та імплементація користувацького інтерфейсу для діагностики процесів завадостійкого кодування	93
2.7.4	Методологія та результати тестування програмного забезпечення	96

2.8	Архітектура давачів інформації завадостійкої системи моніторингу стану потенційно небезпечних об'єктів на базі технологій Інтернету речей	101
2.9	Висновки до розділу 2	105
3	АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ АНАЛІТИЧНОЇ ПРЕДИКТИВНОЇ ПІДСИСТЕМИ РЕГІОНАЛЬНОЇ СИСТЕМИ МОНІТОРИНГУ СТАНУ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ОБ'ЄКТІВ	107
3.1	Обґрунтування необхідності підсистеми прогнозування в регіональних системах моніторингу стану потенційно небезпечних об'єктів	107
3.2	Систематизація наукових досліджень щодо методів прогнозування параметричних показників стану потенційно небезпечних об'єктів	109
3.3	Систематизація наукових досліджень щодо чисельних методів моделювання стану потенційно небезпечних об'єктів.....	113
3.4	Екстраполяція значень параметрів джерел небезпеки на потенційно небезпечних об'єктах за методом найменших квадратів	115
3.5	Прогнозування динаміки параметричних показників стану джерел небезпеки в межах потенційно небезпечних об'єктів за допомогою лінійної моделі.....	118
3.6	Математичне забезпечення екстраполяції часових рядів на основі степеневі поліноміальної апроксимаційної моделі	120
3.7	Програмне забезпечення прогнозування часових рядів на основі поліноміальної апроксимації методом найменших квадратів	124
3.8	Обґрунтування необхідності розроблення алгоритмічного та програмного забезпечення для моделювання стану потенційно небезпечних об'єктів на прикладі розрахунку фільтраційних процесів у земляних гідроспорах	131
3.9	Математичне та алгоритмічне моделювання фільтраційних процесів у земляних гідроспорах	133

3.10	Програмне забезпечення для моделювання фільтраційних процесів у земляних гідропорах	147
3.11	Висновки до розділу 3	152
4	ЕКСПЕРТНИЙ АНАЛІЗ АРХІТЕКТУРИ, АЛГОРИТМІЧНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ МОНІТОРИНГУ СТАНУ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ОБ'ЄКТІВ	154
4.1	Систематизація наукових досліджень у галузі оцінювання якості програмного забезпечення	154
4.2	Обґрунтування застосування методу експертного оцінювання	161
4.3	Об'єкт дослідження та гіпотеза методу експертного оцінювання якості програмних систем моніторингу стану потенційно небезпечних об'єктів	161
4.4	Організація проведення експертного оцінювання	165
4.4.1	Короткий опис першого варіанта регіональної системи моніторингу стану потенційно небезпечних об'єктів	166
4.4.2	Короткий опис другого варіанту регіональної системи моніторингу стану потенційно небезпечних об'єктів	168
4.4.3	Короткий опис третього варіанта регіональної системи моніторингу стану потенційно небезпечних об'єктів	169
4.5	Результати порівняльного аналізу можливих варіантів реалізації регіональної системи моніторингу стану потенційно небезпечних об'єктів	175
4.6	Визначення узгодженості думок експертів	177
4.7	Порівняльний аналіз варіантів реалізації регіональної системи моніторингу стану потенційно небезпечних об'єктів	180
4.8	Висновки до розділу 4	182
	ВИСНОВКИ	184
	СПИСОК ЛІТЕРАТУРИ	187
ДОДАТОК А	Впровадження результатів дисертаційного дослідження у ТОВ «БАЛФОРД УКРАЇНА»	201
ДОДАТОК Б	Впровадження результатів дисертаційного дослідження у ТОВ «ГУТАРЬОВ ТА ПАРТНЕРИ УКРАЇНА»	203

ДОДАТОК В	Впровадження результатів дисертаційного дослідження у НДР «Теоретичні та практичні аспекти технології Internet of Everything» 205
ДОДАТОК Г	Вихідний код додатку «hamming-codec» 206
ДОДАТОК Д	Вихідний код додатку «prediction» 261
ДОДАТОК Е	Вихідний код додатку «filtration» 298
ДОДАТОК Ж	Список публікацій здобувача 314

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПНО – потенційно небезпечний об'єкт

НС – надзвичайна ситуація

АСРВНСО – автоматизована система раннього виявлення загрози виникнення надзвичайних ситуацій та оповіщення населення

СЦТПС – система централізованого пожежного та техногенного спостереження

ЗАСЦО – загальнодержавна автоматизована система централізованого оповіщення

ПЦС – пульт централізованого спостереження

ТАСЦО – територіальна автоматизована система централізованого сповіщення

ПК – пульт керування

ПО – пристрій оповіщення

КТЗІО – кінцеві технічні засоби інформування та оповіщення

КП – комутаційний пристрій

ДПІ – джерела первинної інформації

ССЗБ – суміжні системи забезпечення безпеки

ПЗ – програмне забезпечення

ФВ – функціональні вимоги до програмного забезпечення

ШІ – штучний інтелект

ГС – гідротехнічні споруда(и)

ЄДСЦЗ – єдина державна система цивільного захисту

IoT – інтернет речей

ВСТУП

Дисертаційне дослідження присвячене розв'язання науково-практичної задачі забезпечення надійного функціонування регіональних систем моніторингу стану потенційно небезпечних об'єктів у спосіб розроблення методологічних засад створення алгоритмічного та програмного забезпечення. Задля підвищення рівня безпеки персоналу підприємств, які класифікуються як потенційно небезпечні об'єкти, а також населення прилеглих територій, та мінімізація можливих збитків внаслідок аварій та надзвичайних ситуацій у спосіб розширення функціональних можливостей регіональних систем моніторингу стану потенційно небезпечних об'єктів.

Дисертаційне дослідження спрямоване на розроблення нових, а також подальший розвиток та вдосконалення наявних методологічних підходів до створення алгоритмічного та програмного забезпечення, що є складовою частиною регіональних систем моніторингу стану потенційно небезпечних об'єктів.

Актуальність теми. Більшість міст України є промисловими, а специфіка українських промислових підприємств полягає в тому, що здебільшого їх територія розташована в межах населених пунктів, а відстань від потенційно небезпечних об'єктів до жилих кварталів може складати усього сотні метрів. На потенційно небезпечних об'єктах існує ризик виникнення техногенних аварій та катастроф. Результатом їхнього прояву є раптовий вихід із ладу машин, механізмів та агрегатів під час експлуатації, що супроводжується серйозними порушеннями виробничого процесу, вибухами, утворенням осередків пожеж, радіоактивним, хімічним або біологічним зараженням великих територій, ураженням та загибеллю людей [1]. Тому задля підвищення надійності експлуатації та рівня безпеки потенційно небезпечних об'єктів існує потреба у впровадженні нових та модифікованих способів процесів збору, обробки та передачі даних в інформаційних системах моніторингу стану потенційно небезпечних об'єктів на основі розроблення спеціалізованого прикладного програмного забезпечення.

Під час війни з РФ кількість аварій та катастроф на техногенних об'єктах швидко зростає. Захист населення і територій від надзвичайних ситуацій є

одним з основних завдань системи цивільного захисту України. Забезпечення безпеки в умовах надзвичайних ситуацій потребує надійного функціонування системи реагування на надзвичайні ситуації, адекватної рівням і характеру загроз [2, 3, 4].

Важливу роль в промисловості відіграють системи сигналізації, метою яких є повідомлення операторів про аномальні або аварійні ситуації [5].

Зменшити розмір збитків та захистити життя та здоров'я працівників та населення можливо у спосіб впровадження систем моніторингу стану об'єктів. Це дозволить завчасно виявити можливість виникнення НС, дасть час виконати заплановані протиаварійні заходи, оповістити населення про загрозу.

Прогноз ризиків надзвичайних ситуацій на території країни в цілому здійснює ДСНС у взаємодії з іншими центральними органами виконавчої влади. Разом з цим, на цей час моніторинг і прогнозування надзвичайних ситуацій в Україні здійснюються на рівні регіональних, галузових або інших самостійних підсистем, не об'єднаних у єдиний інформаційно-аналітичний комплекс. Загальнодержавну систему моніторингу джерел надзвичайних ситуацій та їх прогнозування у державі не створено [6].

Застосування дистанційних систем моніторингу надає суттєві переваги при експлуатації територіально розподілених об'єктів. Впровадження таких систем забезпечує можливість програмного контролю та управління наявним промисловим та інфраструктурним обладнанням, що дозволяє мінімізувати витрати на його модернізацію при збереженні необхідного рівня функціональності.

При розв'язанні проблеми побудови архітектури системи дистанційного моніторингу можна розглядати як сукупність віддалених апаратно-програмних модулів та центрального модуля. Віддалені модулі отримують дані від обладнання та відсилають центральному модулю, який зберігає та оброблює отриману інформацію. Отже, проектування системи дистанційного моніторингу можна розділити на чотири задачі: розробка програмного та апаратного забезпечення дистанційного модуля, вибір технології передачі даних, вибір протоколу передачі даних, розробка програмного забезпечення центрального

модуля [7]. Описана вище архітектура системи дистанційного моніторингу є найбільше загальною.

Розв'язання проблеми побудови архітектури системи залежить від конкретної предметної галузі. У дослідженні [8] обґрунтовано необхідність використання додаткових інтелектуальних інформаційних систем для автоматизованої інтегрованої системи управління металургійним підприємством і для запобігання надзвичайним ситуаціям. У статті продемонстровано структуру взаємодії автоматизованих систем управління, систем аварійного оповіщення і систем управління комп'ютерною інформацією для металургійних підприємств. Аналіз дослідження показує спосіб запобігання виникненню аварійних ситуацій, підтверджений позитивними результатами експериментальних досліджень.

Інтеграція інформаційних систем моніторингу надзвичайних ситуацій в структури системи управління забезпечить здатність приймати рішення щодо пошуку інформації при виникненні надзвичайних ситуацій та приймати обґрунтовані рішення в умовах невизначеності та ризику [9].

Наявні рішення лише фіксують факт досягнення докритичного, або критичного рівня контрольованих параметрів. Тому часу для прийняття рішень обмаль. Також немає можливості завчасно виконати протиаварійні заходи, тому що відсутня можливість виконання діагностичного моделювання стану об'єкту.

Актуальність теми обумовлена сучасними викликами, які існують під час експлуатації різноманітних потенційно небезпечних об'єктів в умовах війни з РФ. Кількість аварій та техногенних катастроф на потенційно небезпечних об'єктах, які можуть бути віднесені до потенційно небезпечних об'єктів невинно зростає. Однак загальної системи екологічного моніторингу та моніторингу стану потенційно небезпечних об'єктів з функціями раннього виявлення та прогнозування можливості виникнення надзвичайної ситуації на рівні держави не створено. Наявні рішення малоефективні, прогноз щодо розвитку ситуації малоінформативний, а результат роботи таких систем – це зазвичай лише фіксація факту, що контрольовані сигнали досягли критичного рівня. Тому бракує часу для реалізації заходів для запобігання виникненню

надзвичайної ситуації. Ця проблема може бути вирішена завдяки підвищенню культури експлуатації, безпеки виконання робіт та впровадженню новітніх технологій.

Підхід до проблеми управління як до процесу, що враховує взаємозв'язок частин системи або її окремих підсистем, є основною рисою системного підходу до розробки автоматизованих систем. Головним у системному підході є зосередження уваги на побудові системи в цілому на відміну від побудови її окремих частин [10].

Проведений аналіз сучасного стану проблеми свідчить про необхідність розв'язання науково-практичної задачі забезпечення надійного функціонування потенційно небезпечних об'єктів у спосіб розроблення методологічних засад створення програмного та алгоритмічного забезпечення систем моніторингу. Зазначене програмне забезпечення на основі використання технології Інтернету речей, методів завадостійкого кодування даних, математичного моделювання процесів об'єкту моніторингу та предиктивної аналітики для прогнозування його стану.

Зв'язок дослідження з науковими програмами, планами, темами. Дисертаційне дослідження відповідає науковим напрямам переліку пріоритетних тематичних напрямів наукових досліджень і науково-технічних розробок на період до 31 січня року, наступного після припинення або скасування воєнного стану в Україні, затвердженого постановою Кабінету Міністрів України №476 від 30.04.2024 р., а саме:

– **«Національна безпека і оборона.** Технології кодування, передачі та отримання (автоматичного розпізнавання, обробки, аналізу, генерації, візуалізації) інформації. Технології криптографічного захисту інформації.»;

– **«Національна безпека і оборона.** Методи та засоби запобігання виникненню надзвичайних ситуацій, реагування на них та ліквідації наслідків таких ситуацій і знешкодження засобів ураження.»;

– **«Інформаційні та комунікаційні технології.** Інформаційно – комунікаційні системи та мережі.»;

– «**Енергетика та енергоефективність.** Технології розроблення та використання нових видів палива, відновлюваних і альтернативних джерел енергії та видів палива.».

Оскільки усі гідроелектростанції, незалежно від потужності, є потенційно небезпечними об'єктами, які повинні бути оснащені системою моніторингу стану потенційно небезпечних об'єктів, то дослідження пов'язано з:

– Програмою USELF «Малі ГЕС. Програма фінансування альтернативної енергетики в Україні»;

– Національною програмою «Програма розвитку гідроенергетики на період до 2026 року», схваленою розпорядженням Кабінету Міністрів України від 13 липня 2016 р. № 552-р.

Окремі результати дисертаційного дослідження одержано в межах виконання науково-дослідної роботи «Теоретичні та практичні аспекти технології Internet of Everything» (державний реєстраційний номер 0123U104930).

Мета і задачі дослідження. Метою дисертаційної роботи є підвищення надійності експлуатації та рівня безпеки потенційно небезпечних об'єктів у спосіб удосконалення процесів збору, обробки та передачі даних в інформаційній системі моніторингу стану потенційно небезпечних об'єктів на основі розроблення спеціалізованого прикладного програмного забезпечення з використанням технології інтернету речей, методів завадостійкого кодування даних, моделювання процесів об'єкту моніторингу, а також методів прогнозування зміни його стану.

Задля досягнення мети дисертаційного дослідження необхідно виконати такі задачі:

1. Аналіз архітектури, алгоритмічного та програмного забезпечення наявних регіональних систем моніторингу стану потенційно небезпечних об'єктів.

2. Розроблення архітектури інформаційної системи як ядра регіональної системи моніторингу стану потенційно небезпечних об'єктів з використанням технологій Інтернету речей, завадостійкою передачею даних та підсистемою предиктивної аналітики, а також формування вимог до архітектури, підходів,

методів розроблення та впровадження алгоритмічного та програмного забезпечення.

3. Розроблення методу, алгоритмічного та програмного забезпечення для виявлення та виправлення багатобітових помилок передачі даних для підвищення завадостійкості систем моніторингу стану потенційно небезпечних об'єктів із використанням інфраструктури Інтернету речей та розроблення на основі методу алгоритмічного та програмного забезпечення давачів, які виконують функцію вузлів збору вихідної інформації в складі відмовостійких систем моніторингу стану потенційно небезпечних об'єктів.

4. Розроблення алгоритмічного та програмного забезпечення для здійснення короткочасного прогнозування змін значень параметрів джерел небезпеки, які характеризують стан потенційно небезпечного об'єкта.

5. Розроблення алгоритмічного та програмного забезпечення для діагностичного моделювання стану джерел найбільшої небезпеки на потенційно небезпечних об'єктах – земляних напірних гідропорудах, аварії на яких призводять до масштабних наслідків.

6. Виконати експертне оцінювання якості прийнятих рішень та складу спеціалізованого програмного забезпечення й узагальнення отриманих результатів дослідження у вигляді рекомендацій побудови архітектури, алгоритмічного та програмного забезпечення регіональних систем моніторингу стану потенційно небезпечних об'єктів.

Об'єкт дослідження: процеси збору, обробки та передачі даних в системах моніторингу стану потенційно небезпечних об'єктів

Предмет дослідження: архітектура, методи, алгоритмічне та програмне забезпечення систем моніторингу стану потенційно небезпечних об'єктів

Методи дослідження: Емпіричні (спостереження, порівняння, вимірювання, проведення експериментів), комплексні (абстрагування, структурно-генетичний аналіз і синтез, моделювання), теоретичні (теорія програмування, теорія програмних систем, теорія інформаційних систем, теорія алгоритмів, теорія завадостійкого кодування, теорія прогнозування часових рядів).

Наукова новизна одержаних результатів полягає у наступному:

Уперше розроблено архітектуру як ядро системи моніторингу стану потенційно небезпечних об'єктів, яка відрізняється від наявних використанням технологій Інтернету речей, забезпеченням завадостійкої передачі даних, можливістю прогнозування змін параметрів джерел небезпеки, що характеризують стан об'єкта, а також можливістю моделювання процесів, які призводять до зміни стану об'єкту моніторингу, що дозволяє підвищити надійність експлуатації та рівня безпеки потенційно небезпечних об'єктів.

Уперше розроблено архітектуру, алгоритмічне та програмне забезпечення для давачів вихідної інформації у складі завадостійкої регіональної системи моніторингу стану потенційно небезпечних об'єктів на базі технологій Інтернету речей, особливість яких полягає в тому, що давачі вихідної інформації можуть бути використані у системах моніторингу, у яких використовується побайтова передача даних у вигляді інформаційних блоків, що складаються з інформаційних та контрольних бітів, які перед передачею перемішують згідно зі схемою кодування, що дозволяє зменшити вплив електромагнітних завад при передачі даних та забезпечити цілісність даних.

Уперше розроблено метод виявлення та виправлення багатобітових помилок при передачі інформації, а також алгоритмічне та програмне забезпечення на базі використання кодів Хеммінга, модифікованої схеми кодування та процедур кодування і декодування, який відрізняється процедурою перемішування бітів інформаційного блоку перед передачею в канал зв'язку, що дозволяє підвищити завадостійкість систем моніторингу стану потенційно небезпечних об'єктів.

Удосконалено теоретичні засади розроблення спеціалізованого програмного забезпечення аналітичних предиктивних підсистем регіональних систем моніторингу стану потенційно небезпечних об'єктів, які відрізняються від наявних тим, що розроблені шаблони для реалізації функції прогнозування змін значення параметрів джерел небезпеки в межах потенційно небезпечних об'єктів на об'єктовому рівні та функції моделювання стану потенційно небезпечних об'єктів на місцевому та регіональному рівнях, що дозволяє забезпечити раннє виявлення можливості виникнення надзвичайної ситуації та локалізацію можливих дефектів шляхом розвитку деструктивних процесів.

Уперше розроблене алгоритмічне та програмне забезпечення ітераційного методу розрахунку фільтрації води крізь напірні земляні гідроспороди, яке відрізняється використанням двомірних математичних моделей для виявлення можливості руйнування гідроспороди, що дозволяє спростити процес програмування аналітичних предиктивних підсистем на місцевому та регіональному рівнях, а також завчасно приймати управлінські рішення по запобіганню виникненню аварій та переходу їх в стан надзвичайної ситуації.

Практичне значення одержаних результатів полягає у раціоналізації процесу розроблення алгоритмічного та програмного забезпечення регіональних систем моніторингу стану потенційно небезпечних об'єктів, у спосіб використання запропонованих шаблонів проектування. Такі результати стали можливими завдяки використанню розв'язаних у дисертаційному дослідженні завдань та розроблених науково-методичних підходів, що становлять методичну базу для розроблення регіональних систем моніторингу стану потенційно небезпечних об'єктів та підвищення ефективності та стабільності їх функціонування в індустріальних умовах та впливу електромагнітних завад.

До числа результатів, що мають найбільше практичне значення можливо віднести:

- розроблено архітектурні шаблони на основі використання мереж Інтернету речей для побудови всіх рівнів регіональних систем моніторингу стану потенційно небезпечних об'єктів: об'єктового, місцевого та регіонального;

- розроблено програмне забезпечення для виявлення та виправлення багатобітових помилок передачі інформаційного блоку, що покращує завадостійкість передачі даних в мережах Інтернету речей в умовах індустріальних завад

- розроблено програмне забезпечення давача з функціями завадостійкого вузла збору первинних даних для об'єктового рівня регіональних систем моніторингу стану потенційно небезпечних об'єктів із використанням інфраструктури Інтернету речей;

- впроваджено розроблені аналітичні предиктивні підсистеми у регіональних системах моніторингу потенційно небезпечних об'єктів,

що забезпечує підвищення швидкості реагування на 37%, скорочення часу виявлення можливості надзвичайної ситуації та зниження кількості хибних спрацювань у порівнянні з наявними системами моніторингу без прогнозування;

–використання на місцевому та регіональному рівнях програмного забезпечення, що дає змогу виконати моделювання стану найбільших джерел небезпеки на об'єкті з метою виявлення процесів деградації технічного стану, які можуть призвести до виникнення аварійної ситуації, або навіть до розвитку надзвичайної ситуації з важкими наслідками.

–розроблено методику експертного оцінювання якості архітектури, алгоритмічного та програмне забезпечення.

Розроблені теоретичні положення, методи, алгоритмічне та програмне забезпечення використані при:

1.Проектуванні руслової мала гідроелектростанції у с. Довге Дрогобицького району Львівської області, що підтверджено «Довідка про впровадження результатів дисертаційного дослідження, видана ТОВ «БАЛФОРД УКРАЇНА» №02-11-24 від. 05.11.2024р.» наведена у додатку А

2.Розробці проекту дериваційної малої гідроелектростанції у с. Рибник Дрогобицького району Львівської області, що підтверджено «Довідка про впровадження результатів дисертаційного дослідження, видана ТОВ ГУТАРЬОВ ТА ПАРТНЕРИ УКРАЇНА» №49-11-24 від. 06.11.2024р.» наведена у додатку Б

3.Виконанні науково-дослідної роботи «Теоретичні та практичні аспекти технології Internet of Everything» (державний реєстраційний номер 0123U104930), що підтверджено актом про використання результатів дисертаційного дослідження, що наведений у додатку В

Особистий внесок здобувача. Усі основні результати дисертаційного дослідження, які представлені до захисту, одержані автором особисто. У публікаціях, написаних у співавторстві, здобувачеві належать наступні результати.

У дослідженні [11] запропоновано метод побудови регіональних систем моніторингу стану потенційно небезпечного об'єкта на основі модифікованої

архітектури з можливістю прогнозування змін стану потенційно небезпечного об'єкта.

У дослідженні [12] розроблено метод завадостійкого кодування та декодування даних для систем моніторингу стану потенційно небезпечних об'єктів на основі технологій Інтернету речей. Запропоноване алгоритмічне та програмне забезпечення ґрунтується на модифікованій схемі кодів Хеммінга, що забезпечує виявлення та виправлення багатобітових помилок при побайтовій передачі інформації від давачів первинної інформації до систем обробки даних.

У дослідженні [13] розроблено метод діагностичного моделювання стану потенційно небезпечних об'єктів, що включає модифіковане алгоритмічне забезпечення та відповідне програмне забезпечення для аналітичної предиктивної підсистеми регіональних систем моніторингу стану потенційно небезпечних об'єктів. На основі запропонованого методу проведено моделювання впливу фільтраційних процесів на стан земляних гідроспоруд, які є найбільш поширеним типом потенційно небезпечних об'єктів в Україні та світі. Порушення цілісності таких споруд може призвести до виникнення надзвичайних ситуацій із катастрофічними наслідками та значними матеріальними збитками.

У межах дослідження [14] розроблено та застосовано методику експертного оцінювання якості архітектури, алгоритмічного та програмного забезпечення регіональних систем моніторингу стану потенційно небезпечних об'єктів.

У рамках дослідження [15] здобувач розв'язав проблему побудови архітектури на основі мережі Інтернету речей для регіональних систем моніторингу стану потенційно небезпечних об'єктів з можливістю прогнозування змін стану джерел безпеки в межах потенційно небезпечних об'єктів за допомогою аналітичної предиктивної підсистеми.

У дослідженні [16] розроблено архітектуру, алгоритмічне забезпечення та програмне забезпечення інтелектуального вузла збору даних із функціями завадостійкості та відмовостійкості для об'єктового рівня регіональних систем моніторингу стану потенційно небезпечних об'єктів із використанням

технологій Інтернету речей. Запропонована архітектура, алгоритмічне та програмне забезпечення реалізує комплекс функцій: отримання даних від первинних перетворювачів аналогового типу, попередню обробку результатів вимірювання та захищену передачу даних до хмарного шлюзу з використанням розробленого методу завадостійкого кодування для виявлення та виправлення багатобітових помилок передачі даних.

У межах дослідження [17] розроблено алгоритмічне забезпечення та ПЗ для аналітичної предиктивної підсистеми регіональної системи моніторингу стану ПНО. Запропоноване рішення, що базується на методі найменших квадратів, реалізує функції короткострокового прогнозування динаміки змін параметрів джерел небезпеки ПНО.

Апробація результатів дисертації. Результати дисертаційного дослідження доповідалися та обговорювалися на таких наукових конференціях:

1. III Всеукраїнська науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології SoftTech-2022», присвячена 125-й річниці КПІ ім. Ігоря Сікорського, 23-25 листопада 2022 р., Київ, Україна [15].

2. VI Міжнародна науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології SoftTech-2024», 21-23 травня 2024 р., Київ, Україна [16].

3. VII Міжнародна науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології SoftTech-2024», 20-22 листопада 2024 р., Київ, Україна [17].

Публікації. Основні наукові результати дисертаційного дослідження опубліковано у 7 наукових працях, зокрема у 4 наукових статтях, які проіндексовані в базі даних Scopus, з яких 3 статті опубліковано у виданнях третього квартиля (Q3) та у 3 матеріалах наукових конференцій, з яких 2 матеріали опубліковано на міжнародних наукових конференціях.

Структура роботи. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел, що включає 105 найменувань, та 7 додатків. Загальний обсяг роботи становить 316 сторінок, у тому числі 166 сторінок основного тексту, 37 рисунків, 26 таблиць, 74 формули.

1 ФОРМУЛЮВАННЯ ТА ОБҐРУНТУВАННЯ ДОСЛІДНИЦЬКОЇ ЗАДАЧІ НА ОСНОВІ АНАЛІЗУ АКТУАЛЬНИХ ПІДХОДІВ ДО ПРОЄКТУВАННЯ СИСТЕМ МОНІТОРИНГУ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ОБ’ЄКТІВ

У розділі представлено системний аналіз методологічних засад проєктування та розроблення програмного забезпечення систем моніторингу потенційно небезпечних об’єктів (ПНО). Досліджено фундаментальні аспекти розроблення архітектури, програмного та алгоритмічного забезпечення для реалізації функцій предиктивної аналітики стану ПНО.

Здійснено систематизацію нормативно-правової бази щодо функціональних та нефункціональних характеристик програмного забезпечення (ПЗ) систем моніторингу ПНО. Проведено аналіз сучасних методологічних підходів до розроблення архітектури, програмного та алгоритмічного забезпечення систем моніторингу в Україні та світі. Визначено ключові проблеми у сфері проєктування та розроблення програмного забезпечення для предиктивної аналітики стану ПНО.

На основі проведеного дослідження обґрунтовано необхідність розроблення інноваційної архітектури та відповідного алгоритмічного та програмного забезпечення для реалізації масштабованого програмного забезпечення систем моніторингу ПНО з функціоналом раннього виявлення та прогнозування надзвичайних ситуацій (НС). Сформульовано науково-технічну задачу та визначено пріоритетні завдання дисертаційного дослідження.

1.1 Аналіз сучасного стану системи моніторингу потенційно небезпечних об’єктів

Сталий розвиток держави безпосередньо залежить від стабільного функціонування економіки, що, своєю чергою, визначається надійною та безаварійною роботою промислових підприємств.

В структурі промислового комплексу України переважають підприємства металургійної, хімічної, нафтопереробної промисловості та об’єкти енергетичного сектору.

Враховуючи специфіку функціонування промислових об'єктів, абсолютна безпека їх експлуатації не може бути гарантована, оскільки безпечність роботи залежить від комплексу факторів: умов навколишнього середовища, технічного стану виробничих потужностей та кваліфікації персоналу. Відповідно, завжди існує ймовірність виникнення несприятливих подій, які становлять загрозу для функціонування об'єкту, здоров'я людей та екологічного стану довкілля. Виникнення таких подій корелює з нестабільністю експлуатаційних режимів та відхиленнями від нормативних параметрів функціонування промислових об'єктів. Найбільш критичним сценарієм розвитку ситуації є виникнення надзвичайної ситуації (НС). Передумовою такої події є суттєва нестабільність природних, техногенних і соціально-економічних процесів. З метою своєчасного реагування на потенційні загрози та мінімізації ризиків, виникає необхідність у прогнозуванні та оцінюванні ймовірності виникнення НС.

Аналіз стану техногенної безпеки в Україні в умовах військової агресії демонструє тенденцію до експоненційного зростання ризиків виникнення НС на об'єктах критичної інфраструктури. Це зумовлює необхідність модернізації архітектури програмних систем моніторингу та контролю стану ПНО з метою підвищення ефективності захисту населення і територій.

Додатковим фактором ризику є висока концентрація промислових підприємств, що класифікуються як ПНО, в межах урбанізованих територій. Це створює передумови для виникнення НС внаслідок порушень технологічних процесів, що може призвести до каскадних відмов систем моніторингу ПНО, пожеж, вибухів та забруднення прилеглих територій тощо.

Кількість промислових об'єктів в Україні, які відносяться до ПНО, аварії на яких можуть призвести до виникнення НС самого різного характеру складає:

- 969 хімічно-небезпечних об'єктів, в зонах можливого хімічного забруднення яких мешкає 9,34 млн осіб;

- близько 10 тис. підприємств, установ та організацій, що використовують у своїй діяльності потенційно радіаційно-небезпечні технології та джерела іонізуючого випромінювання;

–3879 суб'єктів господарської діяльності та 9382 об'єктів, аварії на 955 з яких можуть призвести до виникнення НС державного або регіонального рівня [18].

Поточний стан алгоритмічного та програмного забезпечення систем моніторингу потенційно небезпечних об'єктів свідчить про недостатню ефективність наявних підходів до предиктивної аналітики та запобігання виникненню НС техногенного характеру на промислових об'єктах.

Аналіз сучасного стану програмних систем моніторингу потенційно небезпечних об'єктів демонструє необхідність фундаментальної модернізації їх архітектури, алгоритмічного та програмного забезпечення на всіх рівнях ієрархії, а також імплементації інноваційних шаблонів проектування систем безпеки критичної інфраструктури.

На сучасному етапі в Україні системи екологічного моніторингу та системи моніторингу стану ПНО та предиктивної аналітики НС є об'єктами критичної інформаційної інфраструктури та реалізовані фрагментарно, здебільшого на рівні об'єктових та місцевих підсистем, що унеможливорює застосування комплексного підходу до розв'язання проблеми забезпечення техногенної безпеки.

Відсутність інтегрованої програмної інфраструктури державного рівня для імплементації функціонального забезпечення моніторингу, предиктивного аналізу та автоматизованого прогнозування НС на ПНО створює критичні ризики для національної безпеки та потребує системного вирішення на рівні архітектурного проектування програмних систем із застосуванням сучасних методологій розробки програмного забезпечення.

Аналіз функціонування територіальних та функціональних підсистем Єдиної державної системи цивільного захисту демонструє недостатню ефективність наявних методів екологічного моніторингу, збору, обробки та аналізу даних щодо потенційних НС техногенного та природного характеру. Зокрема, наявні програмні системи не забезпечують належний рівень моніторингу, валідації та верифікації інформації про ймовірність виникнення критичних ситуацій, що ускладнює процес розробки превентивних заходів. Пріоритетними завданнями у сфері інженерії програмного забезпечення є

розроблення та імплементація інноваційної архітектури систем моніторингу стану ПНО, які є об'єктами критичної інформаційної інфраструктури, та спрямованих на запобігання НС та мінімізацію їх наслідків [19].

Враховуючи вищезазначене, пріоритетним науково-технічним завданням є розроблення інноваційних архітектурних шаблонів проектування та відповідного алгоритмічного забезпечення для імплементації масштабованих програмних систем моніторингу стану ПНО, що реалізують можливість раннього виявлення аномалій, предиктивної аналітики та прогнозування НС з використанням методів машинного навчання та інтелектуального аналізу даних.

1.2 Аналіз чинної законодавчої бази, суті попередніх розробок, основні висновки, наявність суперечностей у науковій літературі

У [20] зазначено, що фундаментальним підходом до превентивного виявлення та запобігання виникненню НС техногенного або природного характеру є імплементація систем моніторингу, що реалізують предиктивну аналітику та раннє сповіщення операційного персоналу про потенційні НС та критичні відхилення параметрів технологічних процесів на ПНО.

Відповідно до нормативно-правової бази України, при розробці та розгортанні систем моніторингу на ПНО обов'язковим є дотримання вимог [21]. Дані нормативні документи регламентують специфікації щодо проектування та імплементації автоматизованих систем предиктивної аналітики для раннього виявлення загрози виникнення НС та сповіщення населення. Такими системами в обов'язковому порядку мають бути оснащені об'єкти та споруди, що характеризуються потенційним ризиком виникнення НС державного, регіонального або місцевого рівнів.

Фундаментальною метою розробки та імплементації програмних систем моніторингу є забезпечення збору, валідації та аналізу даних від первинних перетворювачів для детектування аномальних значень контрольованих параметрів потенційних джерел техногенної та природної небезпеки. Архітектура таких систем повинна реалізовувати предиктивну аналітику для запобігання досягненню критичних значень параметрів у спосіб

автоматизованого прийняття превентивних заходів. Програмні компоненти системи забезпечують агрегацію, верифікацію та передачу даних про досягнення докритичних або критичних значень контрольованих параметрів операційному персоналу та відповідальним посадовим особам ПНО для оперативного реагування та недопущення ескалації потенційних загроз.

У [21] визначено критерії ідентифікації потенційних НС, що включають:

- перевищення граничних значень критичних метрик моніторингу одного або декількох взаємопов'язаних параметрів програмно-апаратних компонентів системи, що призводить до порушення штатного режиму функціонування об'єкта та потенційної деградації його стану;

- детектування оператором системи моніторингу критичних відхилень параметрів, що створюють безпосередню загрозу життю та здоров'ю персоналу об'єкта, а також ризик значних матеріальних втрат внаслідок потенційної НС.

У [21] визначено фундаментальні вимоги до АСРВНСО, зокрема:

1. При детектуванні потенційної загрози або виникненні надзвичайної ситуації система повинна забезпечити:

- автоматизовану валідацію та верифікацію вхідних даних від первинних перетворювачів;

- предиктивну аналітику та оцінювання потенційних ризиків;

- генерацію сповіщень для операційного персоналу та відповідальних осіб;

- активацію підсистем оповіщення населення у зоні потенційного ураження.

2. Програмні компоненти системи оповіщення повинні забезпечувати трансляцію повідомлень державною мовою та мовою, що є домінуючою у відповідному регіоні.

3. У разі ідентифікації загрози для населення в зоні потенційного ураження при виникненні НС на ПНО, система повинна реалізовувати локальне оповіщення з використанням визначених каналів комунікації.

Згідно [21] АСРВНСО повинні бути побудовані згідно з типовою структурної схеми, яка наведена на рис. 1.1.

етапі розвитку. Попри наявність нормативної бази [21], значного обсягу методологічних матеріалів, наукових публікацій та об'єктивної потреби в системах моніторингу НС, їх впровадження реалізується локально: на рівні окремого об'єкта або в межах місцевих та регіональних програмних комплексів.

Наразі відсутня уніфікована загальнодержавна система програмного моніторингу стану ПНО, яка є об'єктом критичної інформаційної інфраструктури, та яка б забезпечувала комплексне прогнозування та раннє виявлення потенційних НС на національному рівні.

Відсутність такої системи суттєво обмежує часові рамки для реалізації превентивних заходів щодо запобігання виникненню НС та мінімізації їх наслідків.

1.3 Теоретичні та методологічні основи проєктування систем моніторингу стану потенційно небезпечних об'єктів в Україні та світі

У роботі [22] запропоновано класифікацію методологічних підходів до організації екологічного моніторингу та моніторингу стану ПНО з метою мінімізації наслідків НС, а також зазначено, що сучасні урбанізовані території характеризуються підвищеним рівнем ризику виникнення техногенних аварій та природних катастроф внаслідок високої концентрації населення, наявності ПНО техносфери та розташування в зонах підвищеного ризику природних катаклізмів. Наявні методологічні підходи до забезпечення екологічної безпеки регіонів та їх сталого розвитку недостатньо враховують динамічні зміни, спричинені процесами урбанізації. При цьому значна кількість міст розвивається екстенсивно, поглинаючи території, що мають критичне значення для підтримання екологічного балансу.

Міські агломерації, як об'єкти підвищеного ризику виникнення НС природного та техногенного характеру, потребують розроблення та імплементації систем моніторингу стану ПНО, які є об'єктами критичної інформаційної інфраструктури, а також спеціалізованого алгоритмічного

забезпечення для предиктивної аналітики та організації превентивних заходів щодо забезпечення безпеки життєдіяльності населення.

Пріоритетними напрямками мінімізації економічних втрат від НС на ПНО є імплементація систем моніторингу стану ПНО на всіх ієрархічних рівнях – від об'єктового до державного. Розв'язання цієї науково-технічної задачі потребує розроблення інноваційних архітектурних шаблонів та відповідного алгоритмічного забезпечення для реалізації функцій моніторингу та прогнозування НС з урахуванням специфіки кожного рівня ієрархії.

У роботі [23] зазначено, що для України характерно виникнення значної кількості небезпечних природних явищ і процесів геологічного, гідрогеологічного та метеорологічного походження, які можуть спричинити виникнення природних катастроф. Систематичність виникнення та масштабність руйнівних наслідків природних катастроф вимагають розроблення та впровадження систем екологічного моніторингу та систем моніторингу стану ПНО, спрямованих на їх передбачення, локалізацію та усунення наслідків. Ключову роль у процесі передбачення природних катастроф відіграє система моніторингу, оскільки систематичне спостереження, накопичення даних, аналіз та оцінювання якісних і кількісних параметрів стану небезпечних природних явищ та процесів – потенційних джерел природних катастроф – надає можливість розробляти та імплементувати системи моніторингу стану ПНО, спрямовані на збереження життя та здоров'я людей, мінімізацію негативного впливу на навколишнє середовище та зменшення матеріальних втрат, а також на локалізацію зон природних катастроф та нейтралізацію характерних для них небезпечних факторів. Таким чином, актуальним є завдання розроблення програмної архітектури та відповідного алгоритмічного забезпечення систем моніторингу стану ПНО, що забезпечить прогнозування локації, часу, ймовірності виникнення нових осередків небезпеки, оцінювання ризиків для населення, а також підготовку управлінських рішень щодо локалізації та усунення наслідків природних катастроф.

У науковому дослідженні [24] проведено ґрунтовний аналіз методологічних засад інтеграції компонентів штучного інтелекту в програмну

архітектуру систем відновлювальної енергетики для реалізації предиктивної аналітики НС. Встановлено, що наявне алгоритмічне забезпечення та методичні рекомендації щодо нейтралізації наслідків НС на об'єктах підвищеного рівня небезпеки не забезпечують вичерпного охоплення множини факторів, які потенційно можуть спричинити виникнення критичних ситуацій. Відповідно, обґрунтованою є необхідність поетапного розвитку методології розроблення програмних комплексів для нейтралізації наслідків НС на об'єктах зазначеного типу та їх систематизації згідно з чинними нормативними документами. Результати наукового дослідження [24] суттєво розширюють теоретико-методологічний базис проєктування та розроблення програмних систем екологічного моніторингу та систем моніторингу стану об'єктів відновлювальної енергетики.

У науковому дослідженні [25], присвяченому методологічним засадам об'єктноорієнтованого аналізу та проєктуванню програмних систем, обґрунтовано, що для нейтралізації наслідків НС на ПНО критичне значення має ефективність функціонування систем штучного інтелекту, які реалізують функції моніторингу стану об'єктів та предиктивної аналітики виникнення аварійних ситуацій. При цьому встановлено, що ключовою причиною виникнення проблемних ситуацій є зниження точності моніторингу реального стану програмних систем на основі штучного інтелекту, що призводить до порушень безпеки функціонування ПНО. Втім, зазначені висновки потребують уточнення, оскільки суттєвий вплив на рівень безпеки експлуатації об'єктів здійснює людський фактор.

Сучасні дослідження у сфері імплементації систем штучного інтелекту для екологічного моніторингу стану довкілля свідчать, що при проєктуванні архітектури програмного забезпечення систем моніторингу стану ПНО необхідно враховувати варіативність вхідних даних, спричинену комплексом екзогенних факторів природного та антропогенного походження. Виключення зазначених факторів з множини параметрів алгоритмічного забезпечення оцінювання ризиків виникнення аварійних ситуацій на ПНО є некоректною, оскільки призведе до зниження валідності результатів аналізу поточного стану об'єктів.

У контексті інженерії програмного забезпечення, для прогнозування ймовірності виникнення НС та оцінювання їх потенційних наслідків, застосовуються наступні методологічні підходи:

- статистичні методи аналізу даних – базуються на застосуванні математичного апарату теорії ймовірностей для обробки архівних даних та визначення статистичних закономірностей виникнення НС;

- стохастичне моделювання – ґрунтується на побудові ймовірнісних математичних моделей з використанням емпіричних даних та методів математичної статистики;

- методи експертного оцінювання – базуються на систематизації та формалізації експертних знань із застосуванням методів системного аналізу.

Однак, імплементація вищезазначених методів супроводжується наступними технічними та методологічними викликами:

- необхідність агрегації та валідації репрезентативних наборів даних (datasets), отримання яких ускладнене в умовах реального виробничого середовища;

- дефіцит кваліфікованих фахівців з компетенціями у сфері математичного моделювання та data science;

- висока обчислювальна складність математичних моделей та алгоритмів;

- значні часові та ресурсні витрати на розгортання та верифікацію програмних систем;

- складність формалізації та математичної інтерпретації нелінійних процесів;

- критичні часові обмеження на виконання предиктивної аналітики внаслідок високої динаміки розвитку деструктивних процесів.

Розглянуті методологічні підходи та технічні виклики свідчать про необхідність розроблення інноваційних архітектури, алгоритмічного та програмного забезпечення для систем моніторингу стану ПНО.

1.4 Архітектурні та технологічні аспекти розробки сучасних систем моніторингу стану потенційно небезпечних об'єктів

Моніторинг НС відіграє важливу роль, так як спостереження, аналіз і оцінка стану потенційних джерел НС дозволить розробляти і реалізовувати заходи, спрямовані на ліквідацію НС та мінімізацію економічних і екологічних наслідків.

В роботі [26] наведено приклад створення інформаційної системи екологічного моніторингу НС, головним призначенням якої є отримання оперативної інформації про загрозу або виникнення НС, характеристиках вражаючих факторів з метою визначення масштабів поширення і тяжкості наслідків від їх виникнення. На відміну від наявної систем моніторингу НС здійснює:

- екологічний моніторинг та моніторинг стану потенційно небезпечних об'єктів;
- оперативний моніторинг в зоні виникнення НС;
- визначення основних характеристик вражаючих факторів НС – якісний аналіз;
- визначення ризику впливу вражаючих факторів і оцінку їх стійкості – кількісний аналіз;
- прогнозування НС та оцінку ризику їх виникнення.

У дослідженні [27] запропоновано інноваційну методологію інтеграції та агрегації моніторингових даних для оцінювання стану ПНО. Обґрунтовано необхідність імплементації централізованого програмного реєстру ПНО, який у поєднанні з геоінформаційними системами забезпечить можливість генерації інтерактивних карт техногенних ризиків для різних регіонів. Звернено увагу на критичну важливість розробки багаторівневої архітектури програмних систем моніторингу як фундаментального компонента ЄДСЦЗ. Виявлено суттєві недоліки в наявних підходах до обробки даних про стан ПНО на рівні державних та місцевих органів влади, що характеризуються відсутністю або мінімальним застосуванням сучасних методів інтелектуального аналізу даних та предиктивної аналітики.

Розроблення та імплементація систем моніторингу стану ПНО на основі сучасних архітектурних шаблонів та методів інженерії програмного забезпечення є актуальною науково-технічною задачею. Її вирішення забезпечить можливість превентивного виявлення аномалій та предиктивного аналізу загроз виникнення НС у спосіб обробки даних від первинних перетворювачів у режимі реального часу. Алгоритмічне забезпечення таких систем повинні гарантувати високу точність валідації та верифікації моніторингових даних для генерації прогностичних моделей. Ефективність програмних компонентів та якість імплементації системи моніторингу безпосередньо впливає на точність предиктивної аналітики та можливість запобігання каскадному розвитку техногенних аварій різних рівнів критичності.

Необхідно зазначити, що наявні системи моніторингу ПНО характеризуються низькою ефективністю через використання примітивних методів аналізу даних, що базуються виключно на параметричному контролі вхідних сигналів від первинних перетворювачів. Такий підхід, що передбачає лише класифікацію рівня вхідного сигналу як нормального, докритичного чи критичного, не забезпечує належної прогностичної здатності. Внаслідок цього, оцінювання стану об'єкта моніторингу є недостатньо інформативним та зводиться переважно до констатації вже наявних аварійних ситуацій.

Зважаючи на критичність ситуації, існує нагальна потреба у розробці інноваційної програмної архітектури та відповідного алгоритмічного забезпечення для створення масштабованих систем моніторингу стану ПНО. Такі системи повинні реалізовувати предиктивну аналітику в режимі реального часу для прогнозування потенційних аварійних ситуацій та запобігання їх ескалації до рівня НС. Імплементація ієрархічної міжгалузевої системи моніторингу ПНО на основі сучасних шаблонів проєктування та методів інженерії програмного забезпечення сприяє ефективному захисту життя і здоров'я як виробничого персоналу, так і цивільного населення. Розроблення комплексного програмного рішення для систем моніторингу ПНО з використанням діагностичного моделювання є актуальним завданням сучасної інженерії програмного забезпечення.

Процес моделювання в інженерії програмного забезпечення базується на абстрагуванні реальної системи через створення формальної моделі, що знаходиться з нею у визначеному відношенні еквівалентності та здатна відтворювати її ключові функціональні та нефункціональні характеристики. Дослідження та аналіз програмних систем, їх архітектури та поведінки ґрунтується на побудові відповідних формальних специфікацій та моделей, що дозволяють верифікувати коректність їх функціонування.

1.5 Формулювання науково-технічної задачі та завдань дослідження

На підставі критичного аналізу літератури та інших джерел інформації, можна стверджувати, що існує проблема при розробці архітектури та програмного забезпечення для регіональних систем моніторингу стану ПНО. А саме:

- наявна архітектура систем моніторингу стану ПНО не передбачає реалізацію функціоналу предиктивної аналітики та діагностичного моделювання, що є критично важливим для раннього виявлення потенційних НС на ПНО;

- інтеграція технологій Інтернету речей (IoT) [28] зумовлює необхідність реалізації завадостійких каналів передачі даних для гарантування відповідності моніторингової інформації про стан ПНО.

Тому **мета дослідження** – підвищення надійності експлуатації та рівня безпеки потенційно небезпечних об'єктів у спосіб удосконалення процесів збору, обробки та передачі даних в інформаційній системі моніторингу стану потенційно небезпечних об'єктів на основі розроблення спеціалізованого прикладного програмного забезпечення з використанням технології інтернету речей, методів завадостійкого кодування даних, моделювання процесів об'єкту моніторингу, а також методів прогнозування зміни його стану.

На підставі проведеного критичного аналізу для досягнення мети дисертаційного дослідження визначено такі задачі:

- аналіз архітектури, алгоритмічного та програмного забезпечення наявних регіональних систем моніторингу стану потенційно небезпечних об'єктів;

– розроблення архітектури інформаційної системи як ядра регіональної системи моніторингу стану ПНО з використанням IoT-технологій, завадостійкою передачею даних та підсистемою предиктивної аналітики, а також формування вимог до архітектури, підходів, методів розроблення та впровадження програмного та алгоритмічного забезпечення;

– розроблення методу, алгоритмічного та програмного забезпечення для виявлення та виправлення багатобітових помилок передачі даних для підвищення завадостійкості IoT систем моніторингу стану ПНО та розроблення на основі методу програмно-алгоритмічного забезпечення давачів, які виконують функцію вузлів збору вихідної інформації в складі завадостійких індустріальних IoT систем моніторингу стану потенційно небезпечних об'єктів;

– розроблення алгоритмічного та програмного забезпечення для здійснення короткочасного прогнозування змін значень параметрів джерел небезпеки, які характеризують стан об'єкту ПНО;

– розроблення алгоритмічного та програмного забезпечення для діагностичного моделювання стану джерел найбільшої небезпеки на ПНО – земляних напірних гідроспоруд, аварії на яких призводять до масштабних наслідків;

– виконати експертне оцінювання якості прийнятих рішень та складу спеціалізованого програмного забезпечення та узагальнення отриманих результатів дослідження у вигляді рекомендацій побудови архітектури та програмно-алгоритмічних рішень регіональних систем моніторингу стану ПНО.

Комплексне вирішення сформульованих наукових завдань забезпечує досягнення поставленої мети дисертаційного дослідження та формує цілісний метод розроблення регіональних систем моніторингу стану ПНО.

1.6 Висновки до розділу 1

1. На основі аналізу сучасного стану встановлено, що наявні системи моніторингу стану потенційно небезпечних об'єктів в Україні реалізовані фрагментарно та не забезпечують комплексного підходу до запобігання надзвичайної ситуації. Висока концентрація потенційно небезпечних об'єктів

на урбанізованих територіях створює додаткові ризики виникнення надзвичайних ситуацій.

2. Аналіз законодавчої бази та наукової літератури показав відсутність уніфікованої загальнодержавної системи моніторингу, що суттєво обмежує можливості превентивного реагування на потенційні загрози. Чинний нормативні документи визначають вимоги до систем моніторингу, але їх практична реалізація залишається фрагментарною.

3. Дослідження теоретичних та методологічних основ проєктування систем моніторингу стану потенційно небезпечних об'єктів показало, що наявні підходи недостатньо враховують динамічні зміни урбанізованих територій та потребують розроблення інноваційної архітектури систем моніторингу стану потенційно небезпечних об'єктів.

4. Аналіз архітектури та технологічних аспектів проєктування систем моніторингу стану потенційно небезпечних об'єктів виявив обмежену функціональність наявних систем, що проявляється у реалізації параметричного контролю без механізмів прогнозування можливості надзвичайної ситуації, що обґрунтовує необхідність інтеграції предиктивних аналітичних підсистем.

5. На підставі проведеного аналізу сформульовано науково-технічну проблему та визначено комплекс завдань дослідження, спрямованих на підвищення ефективності процесу моніторингу стану потенційно небезпечних об'єктів у спосіб розроблення спеціалізованого програмного забезпечення з використанням сучасних мережевих технологій, методів завадостійкого кодування, предиктивної аналітики та діагностичного моделювання. Отримані результати аналізу формують теоретичне підґрунтя для розроблення архітектури, алгоритмічного та програмного забезпечення регіональних систем моніторингу стану потенційно небезпечних об'єктів.

2 АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПІДВИЩЕННЯ ЗАВАДОСТІЙКОСТІ МЕРЕЖ СИСТЕМ МОНІТОРИНГУ СТАНУ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ОБ'ЄКТІВ НА ОСНОВІ АРХІТЕКТУРИ ІНТЕРНЕТУ РЕЧЕЙ

Архітектурна концепція IoT була сформульована в 1999 році на основі аналізу потенціалу RFID-технологій для забезпечення міжмашинної взаємодії та інтеграції з зовнішніми системами. Подальша еволюція концепції та імплементація практичних архітектур з 2010-х років стала фундаментальним трендом в інженерії програмного забезпечення. Цьому сприяла конвергенція технологій бездротових мереж, розподілених систем, хмарних обчислень, протоколів M2M-комунікації, впровадження мережевого протоколу IPv6 [29] та розвиток програмно сконфігурованих мереж (SDN).

Сучасні програмні системи моніторингу стану ПНО базуються на архітектурі IoT. IoT являє собою архітектурну концепцію розподіленої мережевої взаємодії між фізичними об'єктами, що інтегровані з вбудованими обчислювальними модулями та комунікаційними інтерфейсами для забезпечення міжкомпонентної взаємодії та взаємодії з зовнішнім середовищем.

Згідно [28], IoT-платформа являє собою програмний комплекс, що може бути розгорнутий локально або в хмарній інфраструктурі (за моделлю PaaS), який забезпечує моніторинг та керування кінцевими вузлами розподілених систем через спеціалізовані програмні застосунки. IoT-платформа реалізує масштабовану інфраструктуру для підтримки базових та розширених IoT-сервісів та цифрових операцій [29].

Архітектурна імплементація таких розподілених систем створює передумови для трансформації бізнес-процесів через автоматизацію операційних процедур та мінімізацію необхідності людського втручання в процеси прийняття рішень [30].

Експоненційне зростання масштабів високошвидкісних мережевих інфраструктур та IoT-систем зумовлює потребу в розгортанні надійних комунікаційних каналів з високим рівнем цілісності даних та відмовостійкості.

Сучасна архітектура для програмних систем моніторингу потенційно небезпечних об'єктів базуються на імplementації сигналів первинних перетворювачів та інтеграції їх до інфраструктури інтернету речей, що забезпечує масштабованість, відмовостійкість та ефективний збір даних у режимі реального часу.

Концепція Інтернету речей (Internet of Things, IoT) являє собою парадигму мережевої взаємодії в розподілених системах між фізичними об'єктами, оснащеними вбудованими обчислювальними модулями та комунікаційними інтерфейсами.

2.1 Аналіз архітектури та особливостей імplementації інфраструктури інтернету речей в системах моніторингу стану потенційно небезпечних об'єктів

Архітектура IoT базується на принципах децентралізованої комунікації між пристроями та хмарною інфраструктурою з використанням стандартизованих протоколів передачі даних. Ключовою особливістю є забезпечення кроссплатформної сумісності та інтероперабельності компонентів, що дозволяє формувати гетерогенні екосистеми взаємопов'язаних пристроїв. IoT-системи призначені для автоматизації процесів та забезпечення віддаленого моніторингу й управління. Окремим класом є промисловий Інтернет речей (Industrial IoT, IIoT), що відрізняється специфікою застосування в індустріальному середовищі. Основною перевагою як IoT, так і IIoT є можливість безшовної системної інтеграції компонентів та масштабування архітектури відповідно до вимог конкретного застосування.

У дослідженні [31] проаналізовано архітектурні особливості та програмні компоненти систем IoT як інноваційної парадигми розподілених обчислень. Автори звертають увагу на те, що IoT являє собою глобальну мережу взаємопов'язаних інтелектуальних пристроїв з вбудованими обчислювальними модулями. Підкреслюється, що технологія IoT є перспективною платформою для імplementації масштабованих систем моніторингу та предиктивної аналітики НС завдяки можливості інтеграції гетерогенних первинних

перетворювачів та реалізації обробки даних у розподілених системах у режимі реального часу.

У дослідженні [32] проведено комплексний аналіз імплементації архітектурних шаблонів проєктування програмних розподілених систем на базі IoT інфраструктури для промислового сектору. Обґрунтовано, що архітектура на основі IoT забезпечує ефективну інтеграцію первинних перетворювачів, систем радіочастотної ідентифікації (RFID) та бездротових комунікаційних інтерфейсів для реалізації масштабованих програмних комплексів промислового призначення. Проаналізовано сучасні методології розробки програмного забезпечення для IoT-систем, включаючи шаблони проєктування архітектури розподілених систем, підходи до забезпечення інтероперабельності та механізми оркестрації сервісів. Систематизовано ключові технологічні виклики та перспективні напрямки розвитку інженерії програмного забезпечення в контексті промислового впровадження IoT інфраструктури.

У дослідженні [33] проведено комплексний аналіз взаємозв'язків між різними векторами загроз безпеці, специфікаціями архітектури, програмними реалізаціями та захищеністю розподілених систем на базі IoT-інфраструктури. Враховуючи наявну тенденцію до розгортання інтелектуальних пристроїв у різноманітних сценаріях використання, було досліджено вплив функціональних та нефункціональних вимог на рівень захищеності програмних систем та їх компонентів. Особлива увага приділена аналізу методів верифікації та валідації механізмів безпеки в контексті архітектур розподілених систем.

Дослідження [34] розглядає проблематику інтеграції IoT-систем у гетерогенних мережевих інфраструктурах. Різнманітність протоколів взаємодії та апаратних компонентів створює виклики при проєктуванні архітектури, яка має забезпечувати типові сценарії використання IoT з належним рівнем абстракції. Сучасні вимоги до ПЗ включають часову детермінованість, мінімальну латентність, високу доступність та відмовостійкість. Запропонована архітектура базується на принципах модульності та масштабованості з використанням контейнерної віртуалізації. Декомпозиція прикладної логіки між архітектурними рівнями забезпечує відмовостійкість та високу доступність

системи. Експериментальна верифікація підтвердила ефективність рішення для розгортання сервісів на різних рівнях архітектурної ієрархії.

У дослідженні [35] проаналізовано концепцію Fog Computing (туманних обчислень) та її інтеграцію з IoT-системами. Fog Computing розширює парадигму хмарних обчислень до граничних вузлів мережевої інфраструктури, що уможливорює створення нової генерації програмних систем та сервісів. Ключовими архітектурними характеристиками Fog Computing визначено:

- високу щільність обчислювальних вузлів;
- гетерогенність компонентів;
- інфраструктуру територіально розподілених систем;
- мінімізовану латентність та геолокаційну прив'язку;
- підтримку мобільності;
- превалювання бездротових комунікацій;
- орієнтацію на потокову обробку даних та системи реального часу.

Дослідження обґрунтовує, що зазначені характеристики дозволяють позиціювати Fog Computing як платформу, що забезпечує високий рівень надійності для розгортання критично важливих IoT-сервісів та програмних систем. Імплементація Fog Computing демонструє особливу ефективність при розробці інтелектуальних мереж та систем, що базуються на бездротових первинних перетворювачах та актуаторах.

Дослідження [36] представляє систематичний аналіз наукової літератури щодо оркестрування в системах туманних обчислень. Парадигма туманних обчислень базується на міграції обчислювальних ресурсів та сервісів до граничних вузлів мережевої інфраструктури, що забезпечує мінімізацію латентності та інтеграцію з хмарними ресурсами. На відміну від хмарних систем, інфраструктура туманних обчислень функціонує на основі ресурсно обмежених гетерогенних вузлів з потенційно нестабільною мережевою конективністю. Це зумовлює необхідність імплементації спеціалізованих процесів оркестрування для забезпечення надання ПЗ та сервісів відповідно до визначених угод про рівень обслуговування. На основі консолідації проаналізованих архітектурних особливостей запропоновано узагальнену архітектуру оркестрування туманних обчислень. Дослідження

також ідентифікує ключові технічні виклики та нерозв'язані проблеми в контексті оркестрування туманних обчислень.

У дослідженні [37] здійснено системний аналіз архітектурних шаблонів туманних обчислень. В контексті сучасних вимог до програмних систем, класична архітектура централізованих хмарних обчислень виявляє ряд критичних обмежень: значну затримку передачі даних, недостатню горизонтальну масштабованість та вразливість комунікаційної інфраструктури. Ці архітектурні обмеження набувають особливої значущості при експоненційному зростанні масштабів IoT-систем. Ефективним архітектурним підходом є децентралізація обчислювальної інфраструктури у спосіб розподілу обчислювальних ресурсів ближче до кінцевих IoT-пристроїв. Зокрема, архітектура туманних обчислень реалізує механізми локальної обробки та збереження даних безпосередньо на граничних вузлах, що суттєво знижує залежність від централізованої інфраструктури. На відміну від монолітної хмарної архітектури, туманні обчислення забезпечують мінімізацію затримки передачі даних та підвищену надійність сервісів. Відповідно, туманні обчислення розглядаються як пріоритетний архітектурний шаблон для розгортання високонадійних та захищених IoT-сервісів з високим потенціалом масштабування. Дослідження систематизує сучасний стан розвитку туманних обчислень та їх інтеграцію з IoT-системами, висвітлюючи архітектурні переваги та технічні виклики імплементації. Додатково проведено аналіз невирішених проблем та перспективних напрямів досліджень у контексті конвергенції архітектур туманних обчислень та IoT.

Дослідження [38] представляє систематичний огляд наукової літератури щодо імплементації IoT-систем в енергетичному секторі, з особливим фокусом на архітектуру для інтелектуальних енергетичних мереж. У роботі проведено аналіз сучасних технологічних компонентів IoT-інфраструктури, включаючи хмарні обчислення в розподілених системах та спеціалізовані платформи для обробки та аналізу даних. Окремо досліджено критичні аспекти розгортання IoT-систем в енергетичному секторі, зокрема питання інформаційної безпеки та захисту конфіденційних даних, а також запропоновано архітектурні підходи до їх вирішення.

Дослідження [39] присвячене аналізу архітектурних можливостей IoT-систем у контексті автоматизації бізнес-процесів. Наявні та перспективні IoT-рішення демонструють значний потенціал для підвищення рівня автоматизації та ефективності програмних систем. Реалізація таких систем потребує імплементації комплексних механізмів захисту, включаючи забезпечення конфіденційності даних, багатфакторної автентифікації та відмовостійкості при кібератаках. Відповідно, критично важливим є впровадження архітектурних модифікацій IoT-застосунків для створення наскрізного захищеного середовища функціонування. У дослідженні представлено систематизований аналіз безпекових викликів та потенційних векторів атак в архітектурі IoT-систем.

У дослідженні [40] проведено аналіз архітектури туманних обчислень з інтегрованою системою адаптивного керування. Дослідження демонструє, що масштабування IoT-систем призводить до експоненційного зростання обсягів мережевого трафіку та його варіативності. При цьому, архітектура системи повинна забезпечувати обробку даних з мінімальною латентністю для ефективного прийняття рішень на основі семантичного аналізу IoT-трафіку. Процес прийняття рішень реалізується через механізми адаптивного керування розподіленою системою.

Ефективним підходом до модернізації наявних систем моніторингу стану ПНО є впровадження сучасних технологій IoT, що базуються на використанні енергоефективних інтелектуальних первинних перетворювачів та технологій далекосяжної передачі даних (LPWAN – Low-Power Wide-Area Network).

Архітектура типової системи моніторингу включає наступні компоненти:

- давачів, що функціонують як вузли збору даних від розташованих на об'єкті первинних перетворювачів;
- шлюз передачі даних – програмно-апаратний комплекс, що реалізує двонапрямну маршрутизацію інформаційних потоків між серверною частиною та периферійними пристроями;
- хмарний сервер – обчислювальна платформа, що забезпечує централізовану обробку, аналітику та довготермінове зберігання даних моніторингу;

– інтерфейс оператора – програмний комплекс візуалізації та управління у вигляді вебзастосунку або мобільного додатку, що надає авторизований доступ до хмарного сервісу через стандартизовані протоколи.

Типова архітектура систем моніторингу наведено на рис. 2.1.

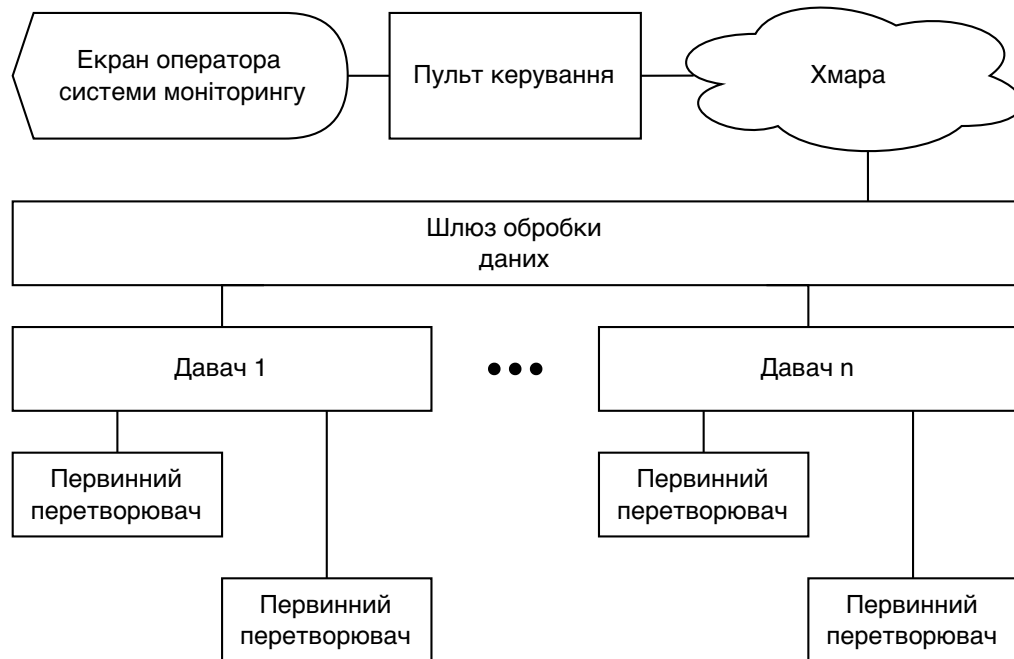


Рисунок 2.1 — Типова архітектура систем моніторингу

На нижньому рівні працює давач, на базі мікропроцесора, до якого приєднанні один або декілька первинних перетворювачів, що збиратиме інформацію та передає її на вищий рівень.

Існують схожі технології для застосування в системах схожого характеру, наприклад LoRaWAN, Narrowband-IoT (або NB-IoT), Sigfox, Zigbee та інші[41]. Основні характеристики цих технологій порівняно на діаграмі рис.2.2 [42].



Рисунок 2.2 — Порівняння бездротових технологій комунікації

Зазначені протоколи мають спільні базові характеристики та частотні діапазони функціонування, проте суттєво відрізняються специфікою реалізації комунікаційних механізмів та архітектурою мережевої взаємодії.

Технологія NB-IoT розроблена з фокусом на енергоефективні стаціонарні первинні перетворювачі та забезпечує розширене територіальне покриття з високою проникною здатністю радіосигналу в умовах щільної міської забудови. Протокол інтегрований в інфраструктуру провідних операторів мобільного зв'язку та функціонує в ліцензованому частотному спектрі. На відміну від технології LoRaWAN, що використовує неліцензований діапазон, NB-IoT характеризується вищою сукупною вартістю експлуатації, проте потенційно забезпечує вищу якість обслуговування кінцевих користувачів [43].

У секторі промислового IoT (IIoT) широкого поширення набула технологія SigFox, однак, оскільки цей протокол є закритою розробкою комерційного оператора зв'язку, його використання супроводжується характерними обмеженнями та складнощами в імплементації [41].

Протокол Zigbee, що базується на стандарті IEEE 802.15.4 [44], позиціюють як енергоефективну альтернативу технологіям Wi-Fi та Bluetooth для пристроїв з низьким енергоспоживанням. Хоча цей протокол широко застосовується в системах домашньої автоматизації, його характеристики не повністю відповідають вимогам промислового Інтернету речей (IIoT).

На основі проведеного порівняльного аналізу комунікаційних протоколів доцільно детальніше розглянути характеристики двох найбільш перспективних технологій для імплементації в системах моніторингу стану ПНО.

1. Обидва стандарти забезпечують порівнянні характеристики пропускної здатності та радіусу покриття. Максимальна пропускна здатність каналу для протоколу NB-IoT становить 60 Кбіт/с, що незначно перевищує показники LoRaWAN.

2. Обидва протоколи (NB-IoT та LoRaWAN) мають вбудовану підтримку механізмів геолокації на рівні мережевої інфраструктури, що є суттєвим фактором для систем моніторингу з розподіленими джерелами небезпеки.

3. З точки зору інформаційної безпеки протокол NB-IoT забезпечує вищий рівень захисту у порівнянні з LoRaWAN.

4. Протокол LoRaWAN демонструє кращі показники енергоефективності у порівнянні з NB-IoT. При використанні LoRaWAN час автономної роботи первинних перетворювачів на об'єктовому рівні може досягати 15 років, тоді як для NB-IoT цей показник становить близько 10 років.

5. Екосистема готових пристроїв для стандарту LoRaWAN характеризується значно ширшим асортиментом у порівнянні з NB-IoT [45].

6. Мережева архітектура LoRaWAN базується на топології "зірка", де пристрої об'єктового рівня здійснюють обмін даними через єдиний шлюз, який забезпечує подальшу комунікацію з мережевим сервером.

Порівняльний аналіз протоколів NB-IoT та LoRaWAN демонструє, що при схожих характеристиках пропускної здатності та підтримці геолокації, NB-IoT має перевагу в безпеці, тоді як LoRaWAN відрізняється кращою енергоефективністю, ширшою екосистемою пристроїв та простішою мережевою топологією типу "зірка". тому має сенс для цілей побудови системи моніторингу стану ПНО використовувати технологію LoRaWAN.

2.2 Архітектура регіональної системи моніторингу стану потенційно небезпечних об'єктів на основі інфраструктури інтернету речей

Для побудови регіональної системи моніторингу стану ПНО краще використовувати технології IoT [15]. На рисунку 2.2 показана схема взаємодії локальних серверів та систем зберігання даних в системі IoT, що базується на архітектурі мікрохмар. У n -й мікрохмарі дані надходять з давачів, обробляються в сервері V_n і запам'ятовуються в сховище X_n . Результати моніторингу в окремій мікрохмарі, отримані після оброблення в сервері V_n , надходять на сервер хмари V_c та до сховища хмари X_c . Сервери хмари, обробляючи інформацію моніторингу від усіх мікрохмар, надають можливість отримати глобальну картину в масштабах регіону про стан контрольованих параметрів. На рисунку 2.3 показана схема взаємодії локальних серверів та систем зберігання даних в системі IoT, що базується на архітектурі мікрохмар. У n -й мікрохмарі дані надходять з давачів, обробляються в сервері V_n і запам'ятовуються в сховище X_n . Результати моніторингу в окремій мікрохмарі,

отримані після оброблення в сервері V_n , надходять на сервер хмари V_c та до сховища хмари X_c . Сервери хмари, обробляючи інформацію моніторингу від усіх мікрохмар, надають можливість отримати глобальну картину в масштабах регіону про стан контрольованих параметрів.

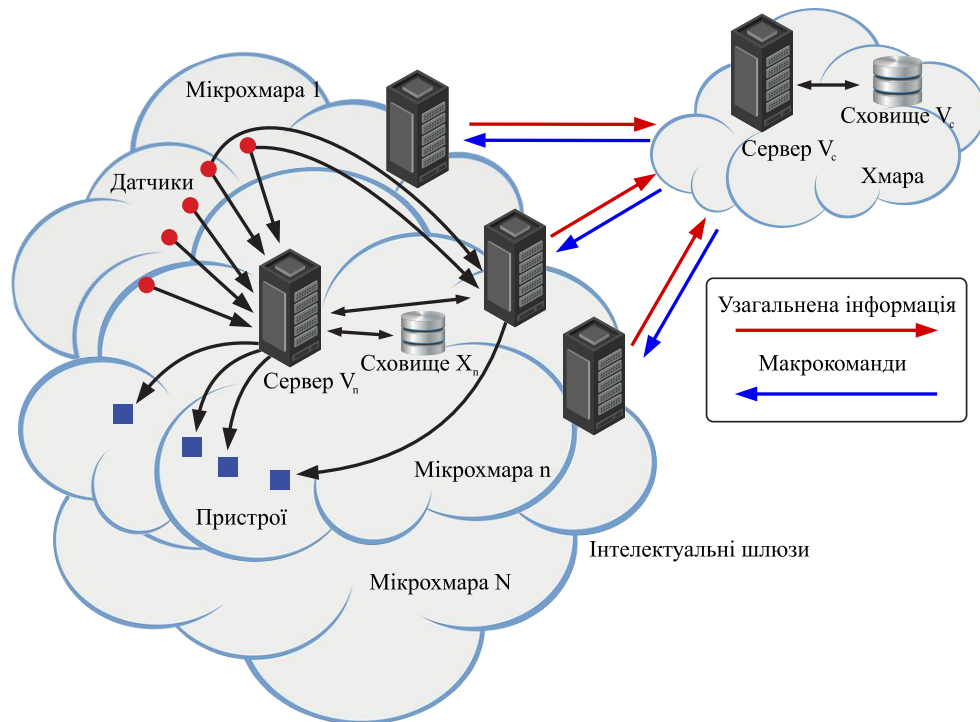


Рисунок 2.3 — Принцип взаємодії локальних серверів і систем зберігання даних в архітектурі мікрохмари.

Наявні системи моніторингу стану ПНО, як правило, обмежуються збором та параметричним контролем сигналів давачів за пороговими рівнями (норма / докритичний рівень / критичний), не виконуючи прогнозування можливості виникнення НС. Отже, актуальною є розробка комплексного рішення, що включає підходи, технології, моделі та методи реалізації системи моніторингу стану ПНО, які є об'єктами критичної інформаційної інфраструктури, з функціями предиктивної аналітики та діагностичного моделювання.

Для реалізації регіональної програмно-апаратної системи моніторингу стану ПНО пропонується архітектура, зображено на рис.2.4. При такому підході на ПНО розташовують системи моніторингу відповідно до [21], а локальні системи моніторингу підключаються до центральної хмари

через інтелектуальні шлюзи. Аналітичні підсистеми, інтегровані як на рівні мікрохмар, так і центральної хмари, забезпечують прогнозування потенційних НС у спосіб аналізу можливого перевищення критичних значень контрольованими параметрами, що характеризують стан джерел небезпеки в межах ПНО. Додатково аналітична підсистема виконує функції діагностичного моделювання стану ПНО.

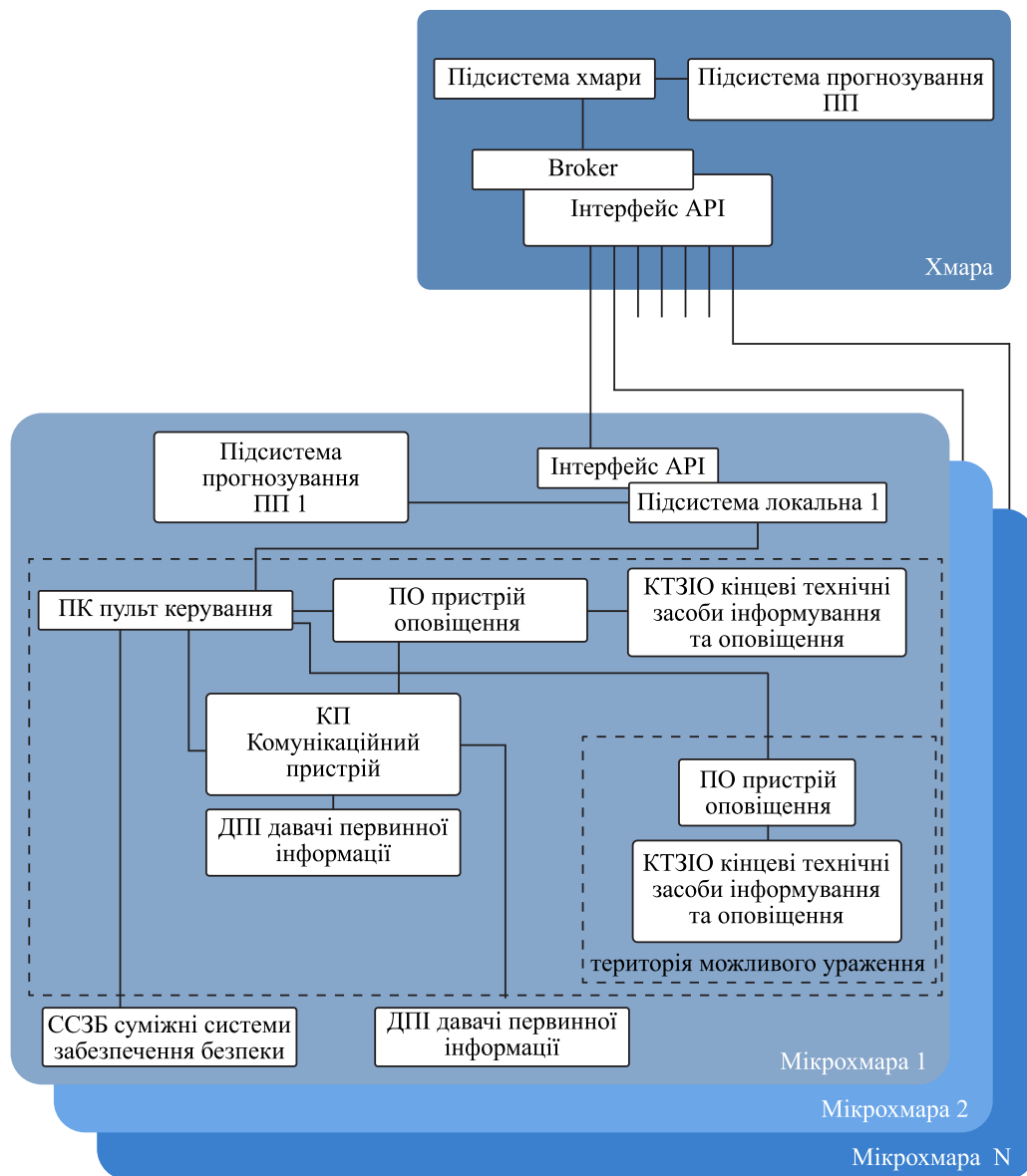


Рисунок 2.4 — Архітектура регіональної програмно-апаратної системи моніторингу стану ПНО на базі IoT мережі

Питання організації роботи мережі на рівні мікрохмар (внесення та зміни параметрів, синхронізація тощо) можливо вирішити, використовуючи

скрипти та одну з таких мов: Perl, Python, AngelScript, JavaScript, JScript. Використання скриптів надає такі переваги:

- допускається внесення змін без негативного впливу на всю систему;
- можливість створювати набір макрокоманд;
- можливість виконувати однакові сценарії на різних браузерях.

Разом з тим, використання скриптів має такі недоліки:

- скрипти, як інтерпретовані сценарії, вимагають для виконання більше часу;
- відсутнє якісне середовище для розробки сценаріїв такого рівня, як середовище розробки (IDE).

Частим випадком для промислових застосувань є використання кількох шлюзів обробки для покриття сигналом більшої площі. В теорії також можливий сценарій, коли навіть на малій місцевості пропускної здатності одного шлюзу буде недостатньо (як правило це до 10 000 пристроїв одночасно в мережі), тоді слід збільшити кількість шлюзів у системі. Таке рішення забезпечить ширші можливості зв'язку, проте відповідно й збільшить вартість системи у спосіб додаткових шлюзових пристроїв.

Матеріали, що стосуються вибору архітектури для регіональних систем моніторингу опубліковані в [15].

2.3 Методологічні засади розроблення алгоритмічного забезпечення розподіленої регіональної системи моніторингу потенційно небезпечних об'єктів

Фундаментальною метою розроблення розподіленої регіональної системи моніторингу стану ПНО є забезпечення комплексного інформаційно-аналітичного супроводу процесів контролю стану ПНО, прогностичного моделювання динаміки параметрів джерел небезпеки, формування управлінських рішень щодо превентивних протиаварійних заходів, а у разі настання НС – своєчасного впровадження заходів щодо ліквідації наслідків НС.

За основу прийнята архітектура розподіленої системи моніторингу стану ПНО, яка має трирівневу ієрархічну структуру, що складається з

регіонального, місцевого та об'єктового рівнів, що забезпечує ефективну декомпозицію функціональних можливостей та оптимальну організацію інформаційних потоків.

Регіональна система моніторингу стану ПНО реалізує функції агрегації та аналізу даних в межах адміністративно-територіальної одиниці, забезпечуючи інтеграцію з державною системою моніторингу через стандартизовані програмні інтерфейси та протоколи передачі консолідованої інформації від місцевих систем моніторингу стану ПНО.

Місцеві системи моніторингу стану ПНО забезпечують функції збору, первинної обробки та агрегації даних в межах визначеної територіально-адміністративної одиниці (району або міста). Архітектура місцевого рівня систем моніторингу стану ПНО реалізує двонапрямну взаємодію: отримання даних від об'єктових підсистем моніторингу та їх консолідовану передачу до регіонального рівня через уніфіковані програмні інтерфейси, що забезпечує цілісність інформаційних потоків.

Об'єктові системи моніторингу стану ПНО забезпечують збір та первинну обробку даних від давачів у межах окремого ПНО, реалізуючи інтеграцію з місцевим рівнем через стандартизовані протоколи передачі даних. Архітектура об'єктового рівня передбачає автоматизоване формування та передачу структурованих інформаційних повідомлень про стан контрольованих параметрів джерел небезпеки до вищих рівнів системи моніторингу стану ПНО.

Відповідно до державного стандарту України [46], архітектура системи моніторингу стану ПНО на всіх ієрархічних рівнях має реалізувати три функціональних режими:

1. Черговий режим – базове функціонування системи в штатному режимі;
2. Режим підвищеної готовності – активується при детектуванні потенційної загрози виникнення надзвичайної ситуації;
3. Режим надзвичайної ситуації – вводиться при підтвердженні факту виникнення НС.

Методологія дослідження алгоритмічного забезпечення розподіленої регіональної системи моніторингу стану ПНО базується на послідовному

інфраструктури граничних значень контрольованих параметрів (докритичних та критичних) для всіх джерел небезпеки в межах ПНО, а також конфігураційних даних щодо максимальної кількості параметрів моніторингу.

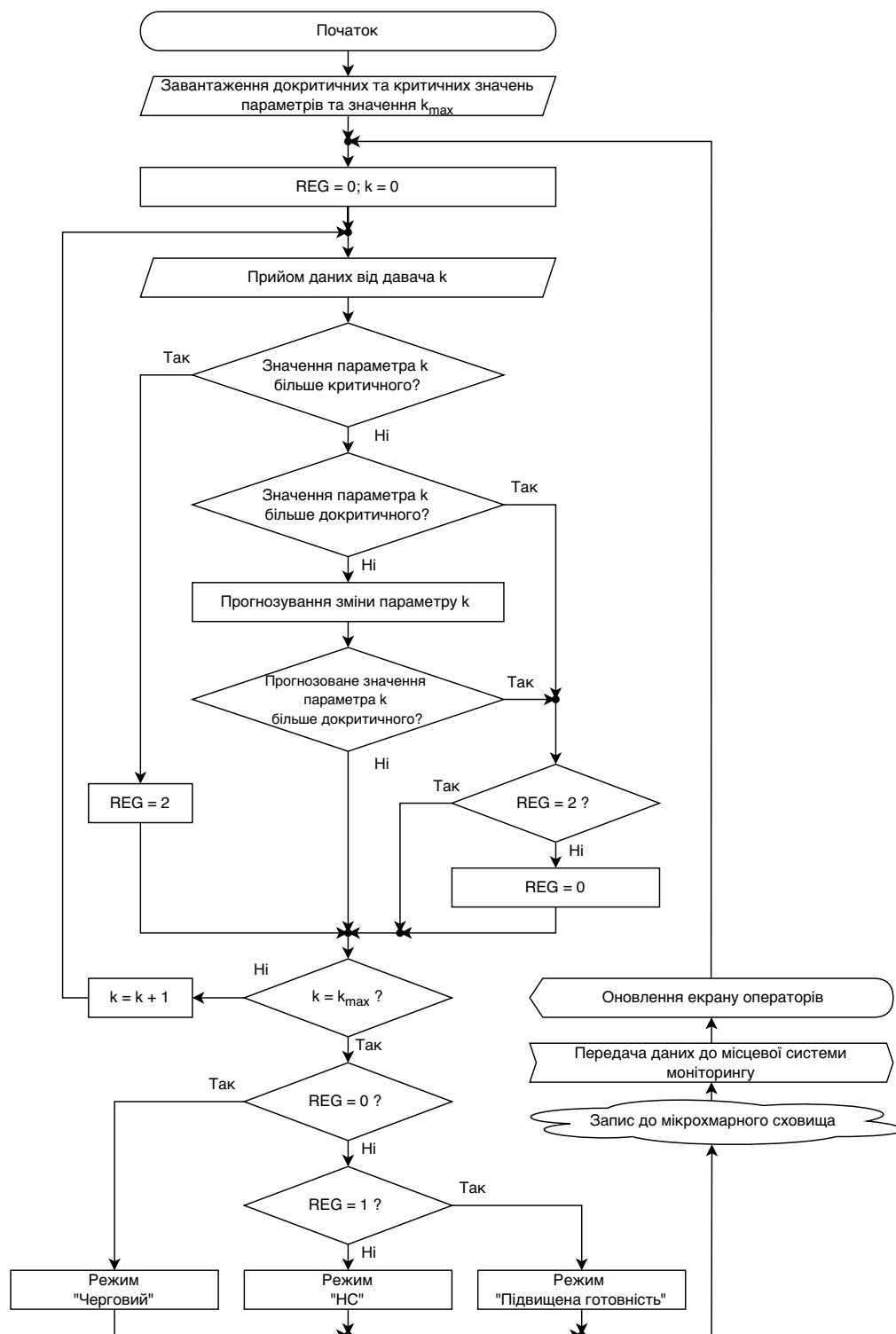


Рисунок 2.6 — Діаграма діяльності об'єктової системи моніторингу стану ПНО

Алгоритмічне забезпечення реалізує дворівневу циклічну структуру обробки даних. Внутрішній цикл забезпечує послідовне отримання та верифікацію значень від множини давачів системи моніторингу. При детектуванні перевищення критичного рівня будь-яким з k контрольованих параметрів, індикатор режиму функціонування *REG* встановлюється в значення 2, що відповідає режиму надзвичайної ситуації. У випадку, коли жоден з параметрів не перевищує критичний рівень, але зафіксовано перевищення докритичного рівня хоча б одним параметром, індикатор *REG* набуває значення 1, що активує режим підвищеної готовності.

У випадку, коли прийняте значення параметру k нижче докритичного рівня, алгоритмічне забезпечення виконують предиктивний аналіз динаміки параметра на три часові відліки вперед із застосуванням математичного апарату степеневі апроксимації та методу найменших квадратів. У разі, якщо прогнозована траєкторія параметра перетинає докритичний рівень хоча б в одній з трьох прогнозованих точок, та поточне значення індикатора *REG* менше 2, відбувається встановлення індикатора *REG* в значення 1, що сигналізує про потенційну загрозу виникнення надзвичайної ситуації.

Наступним кроком здійснюється процедура персистентного збереження часових міток, поточних значень індикатора режиму функціонування *REG* та контрольованого параметра k у структурованому сховищі даних для подальшого аналізу та формування звітності.

Після завершення ітерацій внутрішнього циклу збору та обробки даних від множини давачів, виконується аналіз значення індикатора режиму функціонування *REG* та генерується інформаційне повідомлення для інтерфейсу оператора щодо поточного режиму функціонування об'єктової системи моніторингу стану ПНО.

При нульовому значенні індикатора режиму функціонування *REG* система здійснює моніторинг у штатному режимі функціонування – «Черговий», що характеризується регламентним виконанням процедур збору та аналізу даних.

При встановленні індикатора режиму функціонування *REG* в значення 1 система переходить у режим «Підвищеної готовності», що передбачає

інтенсифікацію процедур моніторингу та активацію додаткових алгоритмів превентивного аналізу параметрів.

При встановленні індикатора режиму функціонування *REG* в значення 2 система активує режим «Надзвичайна ситуація», що ініціює виконання критичних процедур моніторингу, безперервну реєстрацію параметрів та автоматичне сповіщення всіх рівнів системи про виникнення надзвичайної ситуації.

На основі результатів аналізу отриманих даних система генерує два типи структурованих повідомлень: регламентні інструкції для інтерфейсу оператора відповідно до поточного режиму функціонування системи та консолідований інформаційний пакет з параметрами моніторингу для передачі на вищий ієрархічний (місцевий) рівень системи через стандартизовані програмні інтерфейси.

Алгоритмічне забезпечення ініціює наступну ітерацію зовнішнього циклу моніторингу, забезпечуючи безперервність процесу збору та аналізу даних у системі.

Специфікація низькорівневих процедур взаємодії з апаратними давачами та протоколів обміну даними між об'єктовим та місцевим рівнями системи моніторингу виходить за межі поточного опису алгоритмічного забезпечення та розглядається в окремих технічних специфікаціях.

2.3.2 Алгоритмічне забезпечення місцевих систем моніторингу стану потенційно небезпечних об'єктів

Структурну схему місцевої системи моніторингу стану ПНО з інтегрованою системою управління представлено на рисунку 2.7.

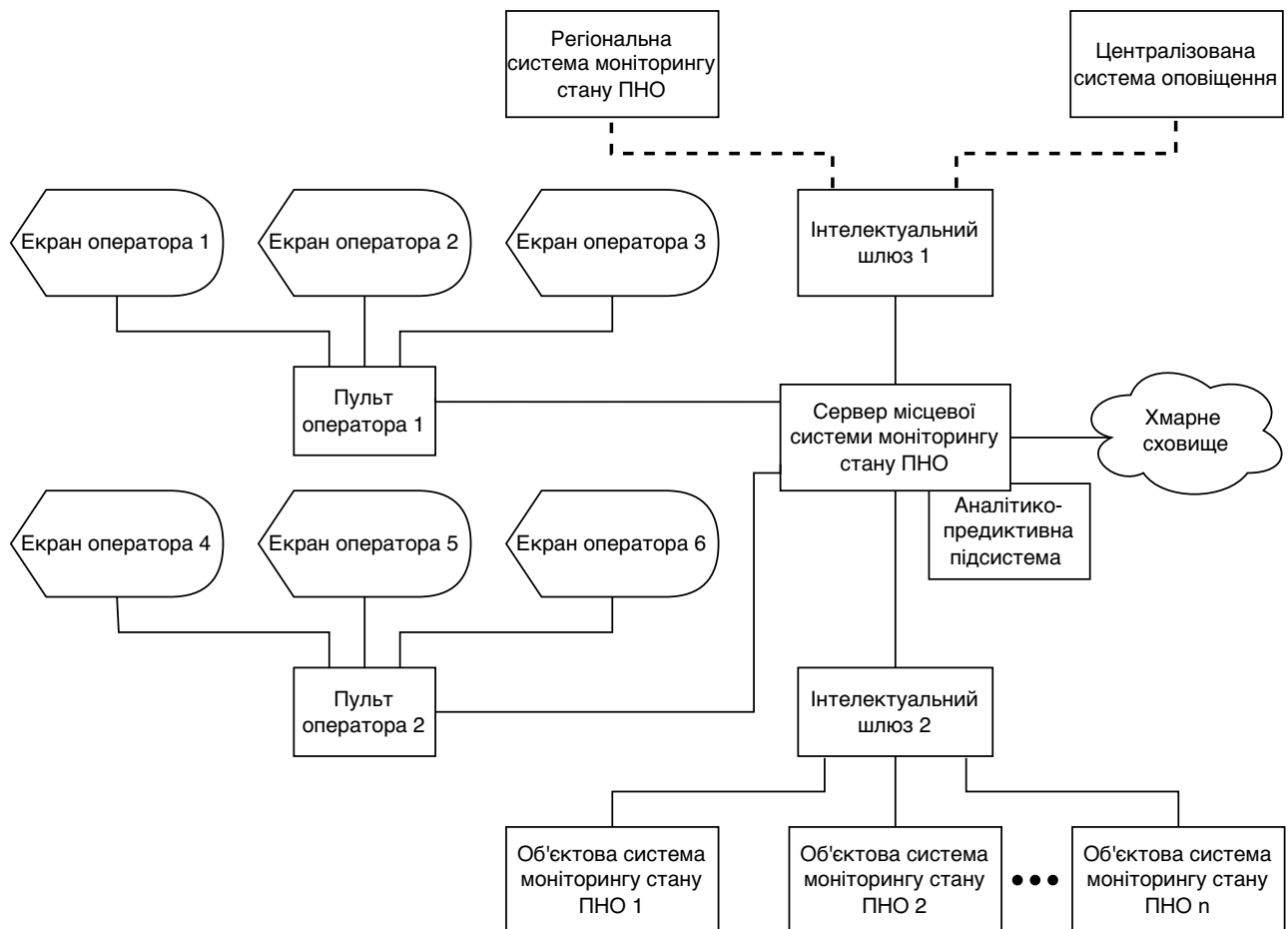


Рисунок 2.7 — Структурна схема місцевої системи моніторингу стану ПНО

Алгоритмічне забезпечення місцевої системи моніторингу стану потенційно небезпечних об'єктів реалізує наступну послідовність функціонування:

1. Ініціалізація системи відбувається в черговому режимі з подальшим циклічним отриманням інформації від об'єктових систем моніторингу стану потенційно небезпечних об'єктів.

2. При надходженні від будь-якої об'єктової системи моніторингу інформації про функціонування в режимі «Підвищена готовність» виконуються наступні процедури:

- перехід місцевої системи в режим «Підвищена готовність»;
- верифікація стану конкретного потенційно небезпечного об'єкта;
- аналіз параметрів, що характеризують стан джерела небезпеки;
- завантаження з мікрохмарної інфраструктури даних щодо протиаварійних заходів;

- отримання інформації про відповідальних осіб;
- активація процедур оповіщення оператора, персоналу та відповідальних осіб;

- запуск процедур контролю виконання протиаварійних заходів.

3. Формування структурованого інформаційного повідомлення, що містить:

- параметри стану джерела небезпеки;
- режими функціонування об'єктової та місцевої систем моніторингу;
- додаткові параметри моніторингу.

4. При отриманні від об'єктової системи моніторингу інформації про функціонування в режимі «Надзвичайна ситуація» виконуються наступні процедури:

- перехід місцевої системи в режим «Надзвичайна ситуація»;
- верифікація стану потенційно небезпечного об'єкта;
- завантаження з мікрохмарної інфраструктури регламенту заходів щодо ліквідації наслідків надзвичайної ситуації;
- отримання даних про відповідальних осіб;
- активація процедур оповіщення;
- запуск процедур контролю виконання заходів щодо ліквідації наслідків надзвичайної ситуації.

Алгоритмічне забезпечення функціонування місцевої системи моніторингу стану ПНО візуалізовано у вигляді діаграми діяльності на рисунку 2.8.

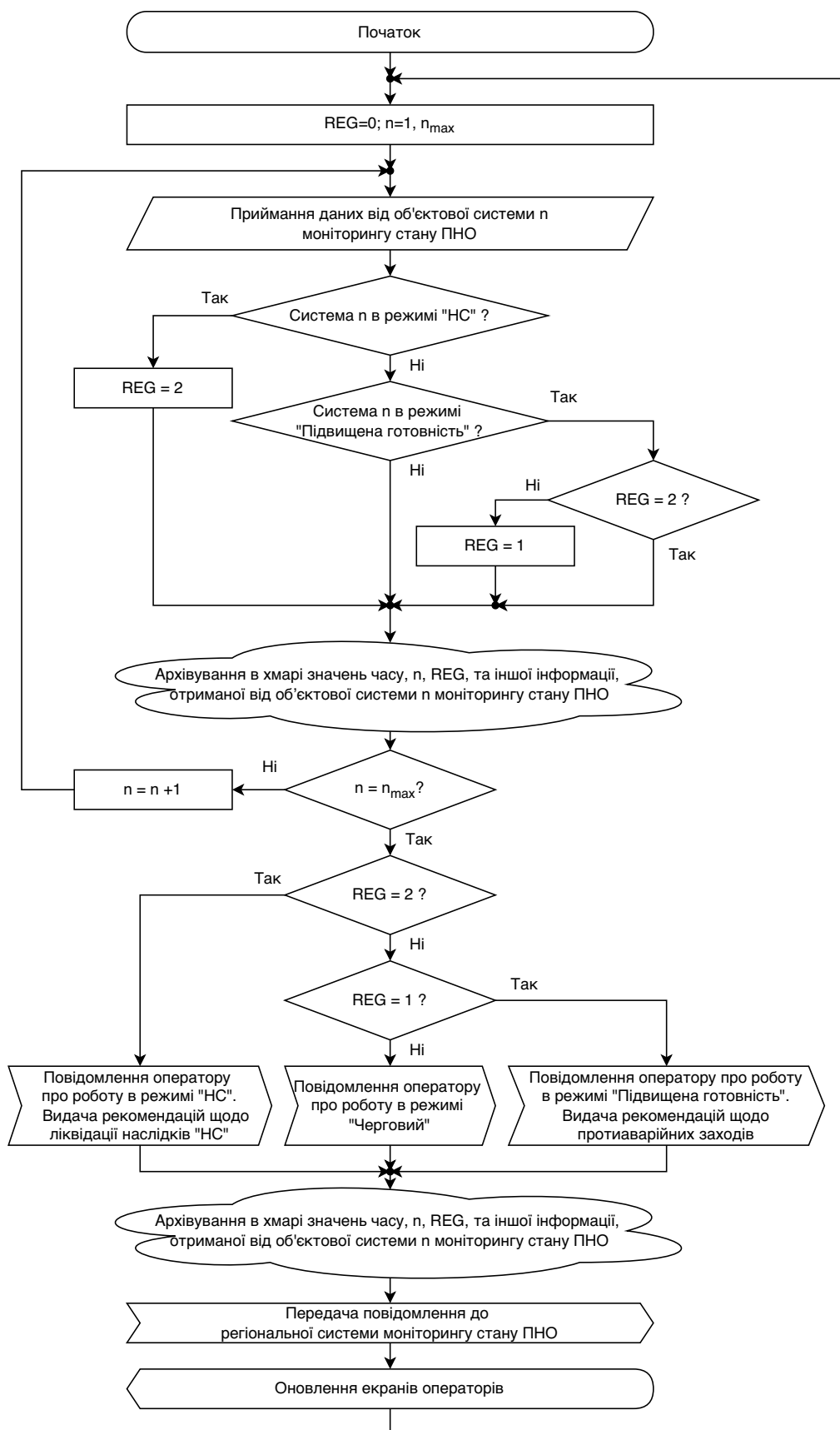


Рисунок 2.8 — Діаграма діяльності місцевої системи моніторингу стану ПНО

Алгоритмічне забезпечення місцевої системи моніторингу стану потенційно небезпечних об'єктів передбачає можливість ініціювання оператором процедури діагностичного моделювання стану джерела небезпеки. Зазначена процедура забезпечує отримання додаткових даних щодо локалізації потенційних ушкоджень, що створює інформаційне підґрунтя для прийняття обґрунтованих управлінських рішень.

Алгоритмічне забезпечення місцевої системи моніторингу реалізує циклічне отримання інформації від об'єктових систем моніторингу стану потенційно небезпечних об'єктів. За умови функціонування всіх об'єктових систем моніторингу в режимі «Черговий», місцева система моніторингу автоматично встановлює відповідний режим функціонування.

2.3.3 Алгоритмічне забезпечення регіональної системи моніторингу стану потенційно небезпечних об'єктів

Структурна схема регіональної системи моніторингу стану ПНО з інтегрованою системою управління наведена на рисунку 2.9.

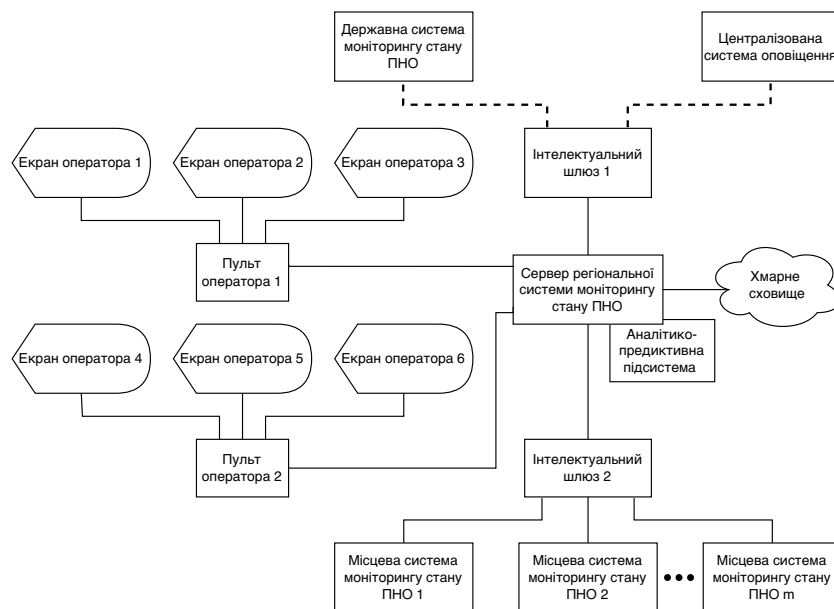


Рисунок 2.9 — Структурна схема регіональної системи моніторингу стану потенційно небезпечних об'єктів

Алгоритмічне забезпечення функціонування даного рівня системи візуалізовано у вигляді діаграми діяльності на рисунку 2.10. Розглянемо

основні принципи функціонування алгоритмічного забезпечення регіональної системи моніторингу стану потенційно небезпечних об'єктів.

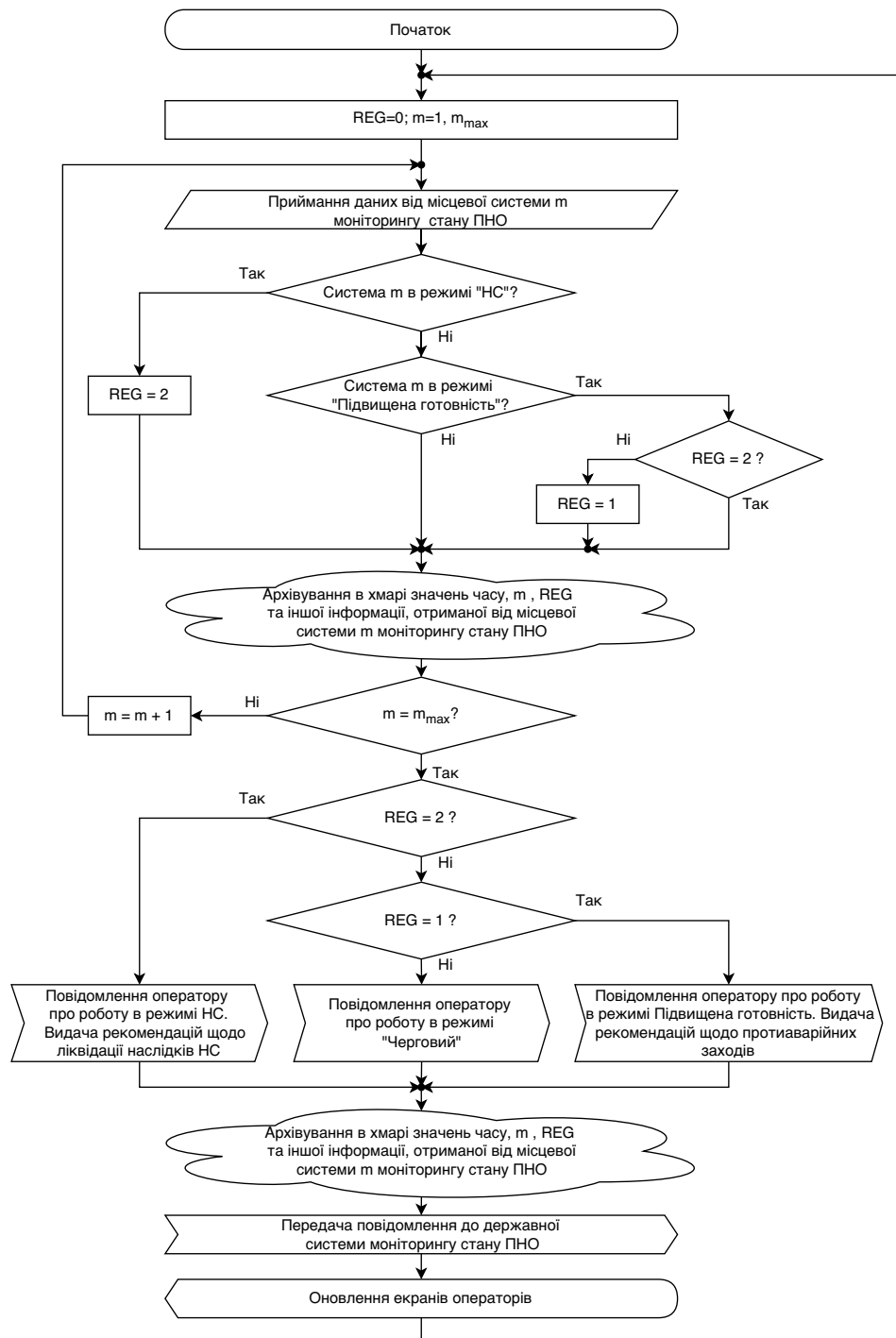


Рисунок 2.10 — Діаграма діяльності регіональної системи моніторингу стану потенційно небезпечних об'єктів

Алгоритмічне забезпечення регіональної системи моніторингу реалізує наступну послідовність функціонування:

1. Ініціалізація системи відбувається в режимі «Черговий» з подальшим циклічним отриманням інформації від місцевих систем моніторингу стану потенційно небезпечних об'єктів.

2. При надходженні від будь-якої місцевої системи моніторингу інформації про функціонування в режимі «Підвищена готовність» виконуються наступні процедури:

- перехід регіональної системи в режим «Підвищена готовність»;
- верифікація стану потенційно небезпечного об'єкта;
- аналіз параметрів джерела небезпеки;
- завантаження з мікрохмарної інфраструктури даних щодо протиаварійних заходів;
- отримання інформації про відповідальних осіб;
- активація процедур оповіщення;
- запуск процедур контролю виконання протиаварійних заходів.

3. При отриманні від місцевої системи моніторингу інформації про функціонування в режимі «Надзвичайна ситуація» виконуються наступні процедури:

- перехід регіональної системи в режим «Надзвичайна ситуація»;
- верифікація стану потенційно небезпечного об'єкта;
- завантаження з мікрохмарної інфраструктури регламенту заходів щодо ліквідації наслідків надзвичайної ситуації;
- отримання даних про відповідальних осіб;
- активація процедур оповіщення;
- запуск процедур контролю виконання заходів щодо ліквідації наслідків надзвичайної ситуації.

Алгоритмічне забезпечення регіональної системи моніторингу стану потенційно небезпечних об'єктів передбачає можливість ініціювання оператором процедури діагностичного моделювання стану джерела небезпеки. Зазначена процедура забезпечує отримання додаткових даних щодо локалізації потенційних ушкоджень, що створює інформаційне підґрунтя для прийняття обґрунтованих управлінських рішень.

Після ліквідації наслідків надзвичайної ситуації та усунення можливості її виникнення на всіх потенційно небезпечних об'єктах, регіональна система моніторингу стану потенційно небезпечних об'єктів реалізує наступні процедури:

- встановлення режиму функціонування «Черговий» для всіх рівнів системи моніторингу (об'єктового, місцевого та регіонального);
- автоматичне формування регіональною та місцевою системами моніторингу структурованих звітів, що містять:
 - хронологію подій на потенційно небезпечних об'єктах;
 - документування впроваджених заходів реагування.

Алгоритмічне забезпечення регіональної системи моніторингу стану потенційно небезпечних об'єктів передбачає можливість формування структурованого звіту за запитом оператора. Звіт містить наступні компоненти:

- часові мітки подій;
- значення параметрів, що характеризують стан джерел небезпеки;
- хронологічну послідовність процедур оповіщення, включаючи:
- тексти інформаційних повідомлень;
- підтвердження отримання повідомлень відповідальними особами;
- регламент впровадження;
- протиаварійних заходів;
- заходів щодо ліквідації наслідків надзвичайної ситуації.

2.4 Дослідження методів завадостійкого кодування на основі кодів Хеммінга для забезпечення надійності передачі даних в мережах Інтернету речей

Сучасні промислові об'єкти, зокрема ПНО, характеризуються просторово розподіленою структурою та значною територіальною протяжністю. Це створює специфічні умови для функціонування системи моніторингу стану ПНО, оскільки компоненти системи піддаються впливу електромагнітних завад різного походження. В контексті інтенсивного розвитку IoT архітектури та відповідного програмного забезпечення, набуває особливої актуальності розроблення методологічного та алгоритмічного забезпечення для виявлення

та коригування багатобітових помилок при передачі даних у розподілених системах моніторингу стану ПНО.

У фундаментальному дослідженні [47] представлено теоретичне обґрунтування можливості забезпечення достовірної передачі даних в умовах наявності електромагнітних завад. Такі завади можуть спричиняти виникнення як одиничних, так і багатобітових спотворень інформаційних потоків у каналах передачі даних. Теоретичне розв'язання цієї проблеми базується на застосуванні завадостійкого кодування з можливістю виправлення помилок. Проте, хоча дослідження [47] доводить існування таких кодів на математичному рівні, воно не надає конкретних методологічних засад щодо синтезу алгоритмічного забезпечення для їх практичної реалізації.

У дослідженні [48] представлено системний аналіз характеристик кодів Хеммінга в контексті теоретичних засад завадостійкого кодування та теорії інформації. Результати дослідження мають суттєве значення для визначення раціональних сфер застосування даного класу кодів при проєктуванні програмного забезпечення систем передачі даних з підвищеними вимогами до достовірності інформації.

В умовах інтенсивного впровадження IoT архітектури для розроблення систем моніторингу стану ПНО, особливої уваги заслуговує дослідження [49], де представлено систематизацію методологічних підходів до завадостійкого каналного кодування в захищених системах передачі даних. У зазначеному дослідженні запропоновано кількісні метрики оцінювання ефективності застосування кодів умовних лишків для забезпечення цілісності та достовірності даних при їх зберіганні, передачі та обробці в програмних розподілених системах.

У контексті вдосконалення методів каналного кодування суттєве значення має дослідження [50], де представлено теоретичне обґрунтування та методологічні засади синтезу кодів Хеммінга в полях Галуа. Запропонований підхід, що базується на застосуванні модульної арифметики, забезпечує підвищення інформаційної місткості символів кодового слова та розширює функціональні можливості алгоритмічного забезпечення завадостійкого кодування. Це створює передумови для розроблення адаптивних програмних

компонентів систем передачі даних з розширеними характеристиками завадостійкості.

Застосування раціональних методів кодування сприяє підвищенню швидкодії та завадостійкості систем оброблення й передачі даних. У дослідженні [51] представлено систематизацію та класифікацію методів завадостійкого кодування з акцентом на їх характеристики достовірності передачі інформації. Завадостійке кодування є фундаментальним компонентом у забезпеченні надійності функціонування програмних систем, особливо в контексті критичних систем оброблення та аналізу даних.

Дослідження [51] містить порівняльний аналіз характеристик двох циклічних кодів: модифікованого коду Елайєса та одиничного позиційного коду. Проведено оцінювання коригувальної здатності коду Елайєса та його модифікованої версії відносно базового коду Хеммінга. Встановлено, що модифікований код Елайєса забезпечує виправлення потрійних помилок у рядках коригувальної матриці та їх виявлення у стовпцях. Додатково виконано порівняльне дослідження характеристик одиничних кодів (позиційного та нормального) з кодом Хеммінга.

Методологічні аспекти кодування та зберігання даних досліджено в роботі [52], де представлено теоретичне обґрунтування застосування схеми Шаміра. Запропонований підхід до розподілу конфіденційних даних базується на принципах надлишкового кодування. Враховуючи, що властивість надлишковості є характерною для завадостійких кодів, автори запропонували методологічні засади розроблення програмного забезпечення для розподіленого зберігання даних на основі алгоритмів завадостійкого кодування.

Фундаментальні принципи імплементації алгоритмів завадостійкого кодування для детектування та корекції помилок систематизовано в дослідженнях [53, 54].

Сучасні телекомунікаційні системи забезпечують високу пропускну здатність для передачі значних обсягів даних, проте актуальною залишається проблема захисту конфіденційної інформації від несанкціонованого доступу. У дослідженні [55] запропоновано методологічні засади розроблення гібридного алгоритмічного забезпечення стеганографічного захисту даних, що базується

на модифікації найменш значущого біта (LSB) та використанні завадостійкого коду Хеммінга (HLAH). За результатами верифікації та валідації розробленого програмного забезпечення встановлено, що запропонований метод HLAH характеризується розширеною місткістю вбудовування при збереженні підвищених показників якості результуючого зображення у порівнянні з наявними рішеннями.

Функціонування захищених телекомунікаційних систем ґрунтується на застосуванні різних механізмів комутації (повідомлень, каналів, пакетів) для забезпечення передачі даних. У дослідженні [56] представлено теоретичні засади використання кодів Хеммінга – сімейства лінійних завадостійких кодів, що застосовуються в системах передачі даних для виявлення та виправлення одно- та двобітових спотворень.

Рациональне використання енергетичних ресурсів при забезпеченні мінімальних затримок передачі становить ключову проблематику для новітніх технологічних рішень, зокрема систем на основі квантових точок. Архітектура на базі клітинних автоматів з квантовими точками (QCA) форму методологічне підґрунтя для розроблення захищених телекомунікаційних систем нового покоління. У дослідженні [56] представлено методи синтезу та програмну реалізацію інноваційних компонентів на базі QCA: одношарового декодера 3-8 та багаторівневого тривходового вентиля XOR. Зазначені компоненти інтегровано до апаратно-програмного комплексу реалізації кодів Хеммінга з використанням QCA-архітектури.

Дослідження [57] аналізує лінійні коди Хеммінга, які широко застосовуються для корекції помилок у пам'яті та телекомунікаційних системах. Уперше в цьому дослідженні запропоновано методологію апаратного аналізу кодів Хеммінга з використанням мемристорних мереж. Ці мережі здатні виявляти та коригувати бітові помилки в кодах Хеммінга з високою швидкістю та енергоефективністю. Матриця перевірки парності зберігається в масиві мемристорів, а вектор синдрому представлений станами виходу уніполярних мемристорів, зміна опору яких природним чином реалізує векторно-матричне множення за модулем 2. Енергоспоживання поточної мережі зменшено більш ніж у 100 разів.

Дослідження [58] присвячене розробленню програмного забезпечення для виявлення та корекції помилок при передачі даних у реальному часі, зокрема при спотворенні ідентифікаторів у потоці даних. Автори запропонували програмну реалізацію алгоритму завадостійкого кодування на основі коду Хеммінга, який демонструє високу ефективність у детектуванні та виправленні пошкоджених бітів при мінімальних обчислювальних витратах. Розроблене алгоритмічне забезпечення формує методологічне підґрунтя для подальшого вдосконалення програмних систем передачі даних з використанням кодів Хеммінга в різноманітних прикладних застосуваннях.

Дослідження [59] присвячене розробленню методологічних засад детектування та корекції помилок у системах передачі даних, де внаслідок впливу завад у комунікаційних каналах виникають спотворення сигналів. Автори запропонували вдосконалений алгоритм завадостійкого кодування на основі кодів Хеммінга, що забезпечує ефективне виявлення розбіжностей між переданими та отриманими даними та реалізує оптимізовані механізми корекції однобітових помилок на приймальній стороні. Розроблене алгоритмічне забезпечення характеризується зниженою обчислювальною складністю у порівнянні з класичними реалізаціями кодів Хеммінга.

У системах передачі даних виникнення помилок є невід'ємним явищем, зумовленим фізичними обмеженнями комунікаційних каналів. Спотворення даних класифікуються на однобітові та пакетні помилки, причому їх характер визначається рівнем шуму в каналі передачі. Дослідження [60] пропонує методологічні засади імплементації завадостійкого кодування на основі кодів Хеммінга, що забезпечує ефективне виявлення та корекцію однобітових помилок. Запропонований підхід розширює традиційні методи контролю парності та демонструє підвищену надійність при збереженні обчислювальної ефективності.

Традиційна реалізація кодів Хеммінга передбачає введення чотирьох контрольних бітів у семибітний інформаційний блок, з розміщенням цих бітів у позиціях 2^n ($n = 0, 1, 2, 3$) відносно інформаційних бітів, що забезпечує виявлення та корекцію однобітових помилок. Дослідження [61] пропонує вдосконалений метод завадостійкого кодування, де контрольні біти

розміщуються в кінці інформаційного блоку. Такий підхід суттєво оптимізує процедури кодування та декодування через усунення операцій вставки та видалення контрольних бітів, забезпечуючи високу масштабованість та обчислювальну ефективність при збереженні коригувальної здатності коду.

Інтенсивний розвиток технологій наносупутників висуває підвищені вимоги до надійності систем зберігання та передачі даних у космічних застосуваннях. Дослідження [62] представляє комплексний аналіз застосування кодів Хеммінга для забезпечення цілісності даних на борту наносупутників на низькій навколоземній орбіті (LEO). Запропоновано три модифікації кодів Хеммінга, що забезпечують корекцію одиночних та детектування подвійних помилок. Проведено багаторівневу верифікацію розроблених рішень з використанням програмного моделювання в середовищах Matlab та VHDL, а також оптимізацію для апаратної реалізації на FPGA-платформах.

При цифровій передачі даних втрати інформації можуть виникати внаслідок дисипації енергії, електромагнітних завад та недосконалості каналів зв'язку. Традиційний підхід повторної передачі даних у випадку помилок створює надмірне навантаження на канал та знижує ефективність системи в цілому. Дослідження [63] пропонує оптимізовану методологію детектування та корекції помилок з використанням службових метаданих. Ключовою інновацією запропонованого методу є логарифмічна залежність обсягу службової інформації від розміру корисного навантаження, що забезпечує високу масштабованість при мінімальних накладних витратах на передачу метаданих.

Дослідження [64] представляє порівняльний аналіз двох класів лінійних блокових кодів: кодів Хеммінга та циклічних кодів. Розроблено програмні реалізації алгоритмів кодування/декодування та механізмів детектування й корекції помилок для обох типів кодів. Експериментальна верифікація підтвердила ефективність запропонованих рішень для забезпечення цілісності даних у каналах зв'язку.

Дослідження [65] пропонує архітектуру для імплементації багатонапрямого коду парності на основі кодів Хеммінга для детектування та корекції помилок. Запропонований підхід розширює традиційні можливості

кодів Хеммінга у спосіб впровадження багатовимірної перевірки парності, що підвищує ефективність виявлення та виправлення помилок при збереженні обчислювальної ефективності.

Розвиток оптичних комунікаційних систем та тритових оптичних обчислювальних комплексів вимагає надійних механізмів забезпечення цілісності даних. Дослідження [66] пропонує модифікацію коду Хеммінга для тритових систем та представляє алгоритми детектування й локалізації помилок. Розроблені теоретичні засади створюють фундамент для реалізації механізмів виявлення та корекції помилок у системах оптичної передачі тритових даних. Запропонований підхід особливо актуальний для високопродуктивних тисячорозрядних оптичних обчислювальних систем, де цілісність даних є критичним фактором надійності.

Проведений аналіз наявних досліджень демонструє, що для забезпечення завадостійкості та цілісності інформації в розподілених системах моніторингу ПНО необхідна розробка модифікованого коду Хеммінга, обґрунтованість чого підтверджується наступними факторами:

1. Наявні реалізації кодів Хеммінга мають обмежену здатність щодо виявлення та корекції багатобітових помилок, що є критичним для IoT-систем моніторингу ПНО з просторово розподіленою структурою.

2. Традиційні підходи до розміщення контрольних бітів створюють надлишкові обчислювальні витрати при кодуванні/декодуванні через необхідність операцій вставки та видалення.

3. Сучасні дослідження підтверджують можливість модифікації базового коду Хеммінга для розширення його функціональних характеристик при збереженні обчислювальної ефективності.

4. Аналіз наявних модифікацій демонструє перспективність розробки вдосконаленого алгоритмічного забезпечення на основі кодів Хеммінга для:

- підвищення коригувальної здатності;
- зменшення обчислювальної складності;
- забезпечення масштабованості рішення.

Таким чином, розробка модифікованого коду Хеммінга з розширеними функціональними можливостями є обґрунтованим напрямком дисертаційного

дослідження для розв’язання актуальної науково-технічної задачі забезпечення цілісності даних у розподілених системах моніторингу ПНО.

2.5 Теоретико-методологічні засади розробки алгоритмічного забезпечення кодів Хеммінга

Коди Хеммінга, вперше описані в роботі [67], забезпечують виявлення двобітних та виправлення однобітних помилок при передачі даних. Це надає їм суттєву перевагу у порівнянні з методами контрольних сум, які лише виявляють наявність помилок.

Актуальність використання кодів Хеммінга зумовлена їх широким застосуванням для захисту цілісності даних як при передачі, так і при зберіганні інформації. Особливо важливим є впровадження завадостійкого кодування в протоколи передачі даних IoT-систем моніторингу стану об’єктів, що дозволяє підвищити надійність функціонування таких систем.

Для досягнення поставленої мети необхідно:

- розробити метод виявлення та виправлення багатобітових помилок на основі кодів Хеммінга;
- реалізувати програмну модель каналу передачі даних мовою Rust;
- провести експериментальне оцінювання ефективності запропонованого методу.

Згідно з теоремою Шеннона [47], можлива безпомилкова передача даних за наявності шуму при використанні каналу зв’язку з певною пропускнуою здатністю та відповідного завадостійкого кодування. Типова структурна схема каналу зв’язку для передачі даних зображена на рис.2.11.

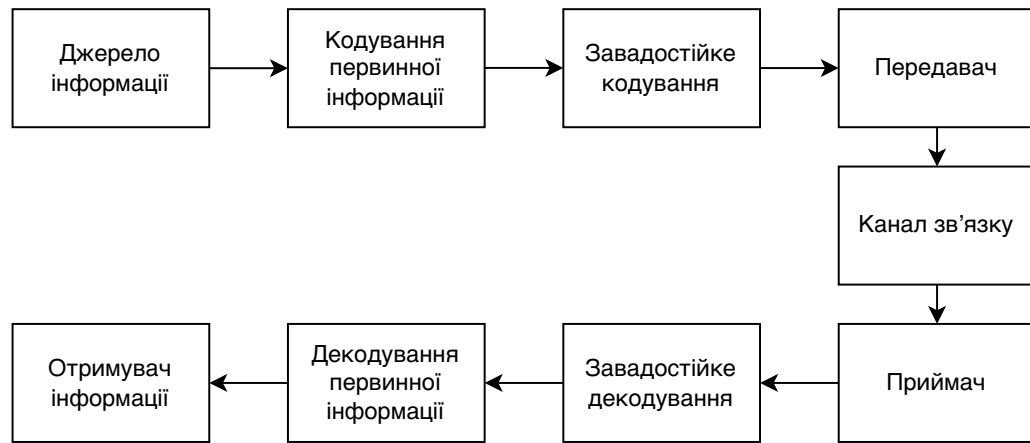


Рисунок 2.11 — Спрощена структурна схема завадостійкого каналу зв'язку

Вихідний сигнал перетворюється у двійковий код, який доповнюється надлишковими бітами для формування кодового слова, що забезпечує корекцію помилок при передачі. Ефективність кодування характеризується кодовою швидкістю R . У каналі зв'язку сигнал піддається спотворенням, після чого демодулюється та декодується приймачем. За наявності помилок декодер виконує їх корекцію, формуючи кодове слово U . Згідно з теоремою Шеннона, безпомилкова передача можлива при швидкості R , що не перевищує пропускну здатність каналу C . Зокрема, якщо швидкість передачі R не перевищує C , то існує код з довжиною кодового слова n , для якого ймовірність помилки дорівнює:

$$P(E) \leq 2^{-nE(R)} \quad (2.1)$$

де $E(R)$ – додатна функція від аргументу R .

Теоретично, зниження ймовірності помилки досягається збільшенням довжини кодового слова n при збереженні швидкості передачі R . Однак, для підтримки сталої швидкості R при зростанні n необхідно пропорційно збільшувати довжину інформаційного слова k , що призводить до експоненціального зростання простору кодових слів та відповідного підвищення обчислювальної складності кодування.

Коди Хеммінга [67] забезпечують детектування та корекцію помилок при мінімальній кодовій відстані $d_{min} = 3$, що гарантує виправлення всіх однобітових помилок. Як блокові коди фіксованої довжини, вони

характеризуються параметрами (n, k) , де n – довжина кодового слова, а k – кількість інформаційних бітів. Структурно коди Хеммінга є систематичними та сепарабельними, оскільки контрольні біти формуються у спосіб лінійних операцій над інформаційними бітами та розміщуються на чітко визначених позиціях кодового слова.

У кодах Хеммінга r перевірочних бітів генеруються на основі k інформаційних бітів, формуючи кодове слово довжиною $n = k + r$. Контрольні біти обчислюються як лінійні комбінації інформаційних бітів з бінарними ваговими коефіцієнтами. Для забезпечення коригувальної здатності коду необхідно виконання умови $2^r \geq n + 1$, що враховує всі можливі однобітові помилки ($C_n^1 = n$) та випадок безпомилкової передачі. При цьому оптимальне співвідношення між параметрами коду визначається рівнянням $2^r - 1 = n$, звідки розраховується кількість інформаційних бітів $k = n - r$ (див. табл. 2.1).

Таблиця 2.1 — Співвідношення між k , r та n для широко використовуваних кодів Хеммінга

k	1	1	2	3	4	4	5	6	7	8	9	10	11	11
r	2	3	3	3	3	4	4	4	4	4	4	4	4	5
n	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Для кодів Хеммінга з мінімальною кодовою відстанню $d_{min} = 3$ матриця перевірки парності характеризується унікальними ненульовими стовпцями довжини r . Зокрема, для коду $(12, 8)$ з параметрами $r = 4$ та $n = 12$ матриця перевірки парності має наступну структуру:

$$H_{12,4} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ n_1 & n_2 & n_3 & n_4 & n_5 & n_6 & n_7 & n_8 & n_9 & n_{10} & n_{11} & n_{12} \end{bmatrix} \quad (2.2)$$

Матриця перевірки парності $H_{12,4}$ формується у спосіб послідовного розміщення ненульових комбінацій чотириелементного двійкового простору у

вигляді стовпців. Після перестановки стовпців, що містять одиничні елементи, матриця набуває канонічної форми (див. формулу 2.3):

$$H_{12,4} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ k_1 & k_2 & k_3 & k_4 & k_5 & k_6 & k_7 & k_8 & r_1 & r_2 & r_3 & r_4 \end{bmatrix} \quad (2.3)$$

На основі матриці 2.3 формується система рівнянь для обчислення перевірочних бітів коду Хеммінга (12, 8). Для кодового слова, що складається з восьми інформаційних бітів ($k_1 \dots k_8$) та чотирьох контрольних бітів ($r_1 \dots r_4$), кожен біт парності r_i обчислюється як сума за модулем два тих інформаційних бітів, які відповідають одиничним елементам у i -му рядку матриці. Зокрема, для біта r_1 рівняння наведено у формулі 2.4. Аналогічно формуються рівняння для інших контрольних бітів, утворюючи повну систему:

$$\begin{cases} r_4 = k_5 \oplus k_6 \oplus k_7 \oplus k_8; \\ r_3 = k_2 \oplus k_3 \oplus k_4 \oplus k_8; \\ r_2 = k_1 \oplus k_3 \oplus k_4 \oplus k_6 \oplus k_7; \\ r_1 = k_1 \oplus k_2 \oplus k_4 \oplus k_5 \oplus k_7. \end{cases} \quad (2.4)$$

Контрольні біти встановлюються в нуль при парній кількості одиниць у відповідних інформаційних позиціях, тоді як непарна сума сигналізує про помилку. Така структура забезпечує локалізацію помилкового біта, оскільки зміна будь-якого інформаційного біта впливає щонайменше на два контрольні біти. Система рівнянь 2.4 для обчислення контрольних бітів систематизована в табл. 2.2.

Набір бітів парності ($r_1 \dots r_4$) для всіх восьми перевірок на парність можна розглядати як шістнадцяткове число R . Нехай $r_1 = 1$, $r_2 = 0$, $r_3 = 1$, $r_4 = 1$, тоді, згідно з табл. 2.2, маємо:

$$R = 1r_1 + 2r_2 + 4r_3 + 8r_4 = 1 * 1 + 2 * 0 + 4 * 1 + 8 * 1 = 13_{10} = 0D_{16} \quad (2.5)$$

Таблиця 2.2 — Процедура обчислення бітів парності

Біти парності	Інформаційні біти															
r_4	=	0	\oplus	0	\oplus	0	\oplus	0	\oplus	k_5	\oplus	k_6	\oplus	k_7	\oplus	k_8
r_3	=	0	\oplus	k_2	\oplus	k_3	\oplus	k_4	\oplus	0	\oplus	0	\oplus	0	\oplus	k_8
r_2	=	k_1	\oplus	0	\oplus	k_3	\oplus	k_4	\oplus	0	\oplus	k_6	\oplus	k_7	\oplus	0
r_1	=	k_1	\oplus	k_2	\oplus	0	\oplus	k_4	\oplus	k_5	\oplus	0	\oplus	k_7	\oplus	0

Помилка в кодовому слові порушує відповідні рівняння перевірки парності 2.4. Зокрема, спотворення четвертого інформаційного біта призводить до порушення перших трьох рівнянь системи, формуючи синдром помилки $(r_1 \dots r_4) = 1110_2$, що відповідає четвертому стовпцю матриці 2.3.

Синдром помилки формується як сума за модулем два між прийнятими контрольними бітами та контрольними бітами, обчисленими на приймальній стороні за тими ж правилами, що використовувались при кодуванні. Цей вектор дозволяє локалізувати позицію помилкового біта в кодовому слові.

Позиція помилкового біта визначається співпадінням обчисленого синдрому з відповідним стовпцем матриці H , яка містить всі можливі r -розрядні двійкові комбінації. Для коду Хеммінга (12, 8) (див. матрицю 2.2) синдроми (S_1, \dots, S_4) формуються побітовим додаванням за модулем два прийнятих та обчислених контрольних бітів, як показано в табл. 2.3.

Таблиця 2.3 — Значення синдромів для коду $H(12, 8)$ (контрольні розряди розташовані після інформаційних)

S_4	S_3	S_2	S_1	Значення синдрому	Діагностика по значенню синдрому	Що потрібно виконати для коригування прийнятої інформації
0	0	0	0	00H	Помилки немає	Коригування не потрібне
0	0	0	1	01H	Помилка. Контрольний біт r_1 .	Коригування не потрібне
0	0	1	0	02H	Помилка. Контрольний біт r_2 .	Коригування не потрібне
0	0	1	1	03H	Помилка. Інформаційний біт k_1 .	Інверсія інформаційного біту k_1 .
0	1	0	0	04H	Помилка. Контрольний біт r_3 .	Коригування не потрібне
0	1	0	1	05H	Помилка. Інформаційний біт k_2 .	Інверсія інформаційного біту k_2 .
0	1	1	0	06H	Помилка. Інформаційний біт k_3 .	Інверсія інформаційного біту k_3 .
0	1	1	1	07H	Помилка. Інформаційний біт k_4 .	Інверсія інформаційного біту k_4 .
1	0	0	0	08H	Помилка. Контрольний біт r_4 .	Коригування не потрібне
1	0	0	1	09H	Помилка. Інформаційний біт k_5 .	Інверсія інформаційного біту k_5 .
1	0	1	0	0AH	Помилка. Інформаційний біт k_6 .	Інверсія інформаційного біту k_6 .
1	0	1	1	0BH	Помилка. Інформаційний біт k_7 .	Інверсія інформаційного біту k_7 .
1	1	0	0	0CH	Помилка. Інформаційний біт k_8 .	Інверсія інформаційного біту k_8 .
1	1	0	1	0DH	Багатобітова помилка	Коригування не потрібне
1	1	1	0	0EH	Багатобітова помилка	Коригування не потрібне
1	1	1	1	0FH	Багатобітова помилка	Коригування не потрібне

Наявність восьми можливих синдромів забезпечує однозначну локалізацію та корекцію помилкового біта у спосіб його інвертування, після чого декодування завершується.

2.6 Метод виявлення та виправлення багатобітових помилок на основі модифікації кодів Хеммінга

Виникнення багатобітових помилок передачі в розгалужених індустріальних мережах моніторингу стану ПНО – це практично доведений факт. Для виявлення та виправлення багатобітових помилок передачі був розроблений метод завадостійкого кодування на основі кодів Хеммінга. Зазначений метод забезпечує виявлення та виправлення множинних помилок при передачі окремих байтів, що належать до одного інформаційного блоку. Метод складається з чотирьох основних процедур: процедури формування

буфера передачі згідно з схемою кодування, процедури перемішування бітів буфера передачі згідно з алгоритмом кодування, процедури декодування прийнятого інформаційного буфера, а також процедури виявлення та виправлення помилок передачі.

2.6.1 Модифікована схема завадостійкого кодування для систем моніторингу потенційно небезпечних об'єктів

У контексті забезпечення цілісності даних у системах моніторингу ПНО розроблено модифіковану схему завадостійкого кодування, що базується на вдосконаленому методі Хеммінга. Схема призначена для систем з послідовною побайтовою передачею даних та забезпечує підвищений рівень виявлення й виправлення помилок при передачі інформації.

Запропонована схема кодування характеризується наступними властивостями:

- розширені можливості виявлення та виправлення багатобітових помилок;
- сумісність з наявними протоколами передачі даних у системах моніторингу ПНО.

Структуру буфера передачі, що реалізує запропоновану схему кодування, наведено в табл. 2.4.

Таблиця 2.4 — Модифікована схема кодування інформації у буфері передачі

Байти буфера передачі																							
$n_{i,1}$	$n_{i,2}$	$n_{i,3}$	$n_{i,4}$	$n_{i,5}$	$n_{i,6}$	$n_{i,7}$	$n_{i,8}$	$n_{i,9}$	$n_{i,10}$	$n_{i,11}$	$n_{i,12}$	$n_{i,13}$	$n_{i,14}$	$n_{i,15}$	$n_{i,16}$	$n_{i,17}$	$n_{i,18}$	$n_{i,19}$	$n_{i,20}$	$n_{i,21}$	$n_{i,22}$	$n_{i,23}$	$n_{i,24}$
Інформаційні біти																Контрольні біти							
$k_{i,1}$	$k_{i,2}$	$k_{i,3}$	$k_{i,4}$	$k_{i,5}$	$k_{i,6}$	$k_{i,7}$	$k_{i,8}$	$k_{i,9}$	$k_{i,10}$	$k_{i,11}$	$k_{i,12}$	$k_{i,13}$	$k_{i,14}$	$k_{i,15}$	$k_{i,16}$	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$	$r_{i,4}$	$r_{i,5}$	$r_{i,6}$	$r_{i,7}$	$r_{i,8}$
$k_{8,1}$	$k_{8,2}$	$k_{8,3}$	$k_{8,4}$	$k_{8,5}$	$k_{8,6}$	$k_{8,7}$	$k_{8,8}$	$k_{8,9}$	$k_{8,10}$	$k_{8,11}$	$k_{8,12}$	$k_{8,13}$	$k_{8,14}$	$k_{8,15}$	$k_{8,16}$	$r_{8,1}$	$r_{8,2}$	$r_{8,3}$	$r_{8,4}$	$r_{8,5}$	$r_{8,6}$	$r_{8,7}$	$r_{8,8}$
$k_{7,1}$	$k_{7,2}$	$k_{7,3}$	$k_{7,4}$	$k_{7,5}$	$k_{7,6}$	$k_{7,7}$	$k_{7,8}$	$k_{7,9}$	$k_{7,10}$	$k_{7,11}$	$k_{7,12}$	$k_{7,13}$	$k_{7,14}$	$k_{7,15}$	$k_{7,16}$	$r_{7,1}$	$r_{7,2}$	$r_{7,3}$	$r_{7,4}$	$r_{7,5}$	$r_{7,6}$	$r_{7,7}$	$r_{7,8}$
$k_{6,1}$	$k_{6,2}$	$k_{6,3}$	$k_{6,4}$	$k_{6,5}$	$k_{6,6}$	$k_{6,7}$	$k_{6,8}$	$k_{6,9}$	$k_{6,10}$	$k_{6,11}$	$k_{6,12}$	$k_{6,13}$	$k_{6,14}$	$k_{6,15}$	$k_{6,16}$	$r_{6,1}$	$r_{6,2}$	$r_{6,3}$	$r_{6,4}$	$r_{6,5}$	$r_{6,6}$	$r_{6,7}$	$r_{6,8}$
$k_{5,1}$	$k_{5,2}$	$k_{5,3}$	$k_{5,4}$	$k_{5,5}$	$k_{5,6}$	$k_{5,7}$	$k_{5,8}$	$k_{5,9}$	$k_{5,10}$	$k_{5,11}$	$k_{5,12}$	$k_{5,13}$	$k_{5,14}$	$k_{5,15}$	$k_{5,16}$	$r_{5,1}$	$r_{5,2}$	$r_{5,3}$	$r_{5,4}$	$r_{5,5}$	$r_{5,6}$	$r_{5,7}$	$r_{5,8}$
$k_{4,1}$	$k_{4,2}$	$k_{4,3}$	$k_{4,4}$	$k_{4,5}$	$k_{4,6}$	$k_{4,7}$	$k_{4,8}$	$k_{4,9}$	$k_{4,10}$	$k_{4,11}$	$k_{4,12}$	$k_{4,13}$	$k_{4,14}$	$k_{4,15}$	$k_{4,16}$	$r_{4,1}$	$r_{4,2}$	$r_{4,3}$	$r_{4,4}$	$r_{4,5}$	$r_{4,6}$	$r_{4,7}$	$r_{4,8}$
$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$	$k_{3,6}$	$k_{3,7}$	$k_{3,8}$	$k_{3,9}$	$k_{3,10}$	$k_{3,11}$	$k_{3,12}$	$k_{3,13}$	$k_{3,14}$	$k_{3,15}$	$k_{3,16}$	$r_{3,1}$	$r_{3,2}$	$r_{3,3}$	$r_{3,4}$	$r_{3,5}$	$r_{3,6}$	$r_{3,7}$	$r_{3,8}$
$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$	$k_{2,6}$	$k_{2,7}$	$k_{2,8}$	$k_{2,9}$	$k_{2,10}$	$k_{2,11}$	$k_{2,12}$	$k_{2,13}$	$k_{2,14}$	$k_{2,15}$	$k_{2,16}$	$r_{2,1}$	$r_{2,2}$	$r_{2,3}$	$r_{2,4}$	$r_{2,5}$	$r_{2,6}$	$r_{2,7}$	$r_{2,8}$
$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$	$k_{1,6}$	$k_{1,7}$	$k_{1,8}$	$k_{1,9}$	$k_{1,10}$	$k_{1,11}$	$k_{1,12}$	$k_{1,13}$	$k_{1,14}$	$k_{1,15}$	$k_{1,16}$	$r_{1,1}$	$r_{1,2}$	$r_{1,3}$	$r_{1,4}$	$r_{1,5}$	$r_{1,6}$	$r_{1,7}$	$r_{1,8}$

Процес формування кодованої послідовності включає такі етапи:

1. генерація контрольних бітів для кожного інформаційного байту згідно з модифікованим алгоритмом;

2. структурування даних у буфері передачі відповідно до розробленої схеми;
3. застосування спеціалізованих операцій перемішування для підвищення стійкості до пакетних помилок.

Така організація даних забезпечує ефективне виявлення та виправлення помилок при мінімальних витратах обчислювальних ресурсів.

2.6.2 Алгоритм кодування інформаційних блоків

Процес кодування інформаційного блоку розміром 16 байт передбачає формування 64 контрольних бітів. Інформаційні байти структуруються у матрицю розмірністю 8×16 , яка доповнюється матрицею контрольних бітів розмірністю 8×8 . Обчислення контрольних бітів r для першої групи з 8 байт (табл. 2.4) виконується згідно з правилами, наведеними у табл.2.5.

Таблиця 2.5 — Порядок розрахунку контрольних бітів перших восьми байтів

Контрольний біт	Інформаційні біти														
$r_{i,4} =$	0	\oplus	0	\oplus	0	\oplus	0	\oplus	$k_{i,5}$	\oplus	$k_{i,6}$	\oplus	$k_{i,7}$	\oplus	$k_{i,8}$
$r_{i,3} =$	0	\oplus	$k_{i,2}$	\oplus	$k_{i,3}$	\oplus	$k_{i,4}$	\oplus	0	\oplus	0	\oplus	0	\oplus	$k_{i,8}$
$r_{i,2} =$	$k_{i,1}$	\oplus	0	\oplus	$k_{i,3}$	\oplus	$k_{i,4}$	\oplus	0	\oplus	$k_{i,6}$	\oplus	$k_{i,7}$	\oplus	0
$r_{i,1} =$	$k_{i,1}$	\oplus	$k_{i,2}$	\oplus	0	\oplus	$k_{i,4}$	\oplus	$k_{i,5}$	\oplus	0	\oplus	$k_{i,7}$	\oplus	0

Для байтів з номерами (9...16) біти парності обчислюються так, як показано у табл.2.6. Слід зазначити, що табл.2.6 по суті є модифікацією табл. 2.5, де змінено лише індекси i .

Таблиця 2.6 — Порядок розрахунку контрольних бітів байтів 9 ... 16

Контрольний біт	Інформаційні біти														
$r_{i,8} =$	0	\oplus	0	\oplus	0	\oplus	0	\oplus	$k_{i,13}$	\oplus	$k_{i,14}$	\oplus	$k_{i,15}$	\oplus	$k_{i,16}$
$r_{i,7} =$	0	\oplus	$k_{i,10}$	\oplus	$k_{i,11}$	\oplus	$k_{i,12}$	\oplus	0	\oplus	0	\oplus	0	\oplus	$k_{i,16}$
$r_{i,6} =$	$k_{i,9}$	\oplus	0	\oplus	$k_{i,11}$	\oplus	$k_{i,12}$	\oplus	0	\oplus	$k_{i,14}$	\oplus	$k_{i,15}$	\oplus	0
$r_{i,5} =$	$k_{i,9}$	\oplus	$k_{i,10}$	\oplus	0	\oplus	$k_{i,12}$	\oplus	$k_{i,13}$	\oplus	0	\oplus	$k_{i,15}$	\oplus	0

Наступний етап алгоритму передбачає виконання процедури перемішування бітів у буфері передачі через застосування операцій циклічного

побітового зсуву. Процедура перемішування виконується для інформаційних та контрольних бітів (табл. 2.4) відповідно до розробленого алгоритму кодування (рис. 2.12):

- біти в позиції 1 (молодші біти) кожного байту залишаються без змін;
- біти в позиції 2 всіх байтів зсуваються на одну позицію праворуч з перенесенням крайнього правого біта на крайню ліву позицію;
- аналогічні операції зсуву послідовно виконуються для бітів у позиціях 3 – 8 з відповідним збільшенням величини зсуву.

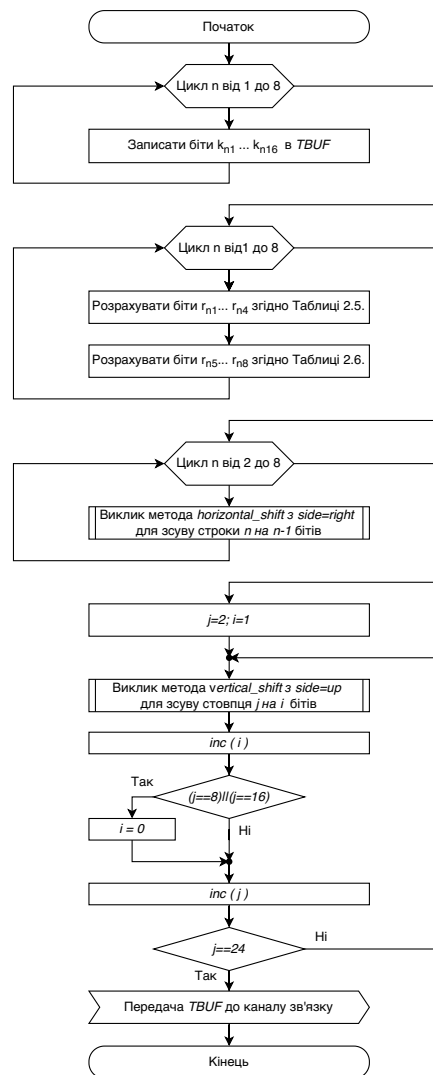


Рисунок 2.12 — Блок-схема алгоритму кодування

В алгоритмах кодування та декодування:

n – це кількість контрольних байтів,

j – кількість байтів у повідомленні,

i – кількість зсувів.

Опис алгоритму потребує деяких пояснень. Для здійснення операцій розрахунку контрольних бітів одного байту та операцій коригування помилок передачі використано клас HammingCodec (вихідний код наведено у додатку Г) з методами для кодування та декодування даних за допомогою коду Hamming (12, 8).

Структуру буфера передачі після виконання операцій циклічного зсуву наведено в табл. 2.7.

Таблиця 2.7 — Розміщення інформації у буфері передачі після виконання операцій зсуву вправо

Байти буфера передачі																							
$n_{i,1}$	$n_{i,2}$	$n_{i,3}$	$n_{i,4}$	$n_{i,5}$	$n_{i,6}$	$n_{i,7}$	$n_{i,8}$	$n_{i,9}$	$n_{i,10}$	$n_{i,11}$	$n_{i,12}$	$n_{i,13}$	$n_{i,14}$	$n_{i,15}$	$n_{i,16}$	$n_{i,17}$	$n_{i,18}$	$n_{i,19}$	$n_{i,20}$	$n_{i,21}$	$n_{i,22}$	$n_{i,23}$	$n_{i,24}$
Інформаційні біти																Контрольні біти							
$k_{i,1}$	$k_{i,2}$	$k_{i,3}$	$k_{i,4}$	$k_{i,5}$	$k_{i,6}$	$k_{i,7}$	$k_{i,8}$	$k_{i,9}$	$k_{i,10}$	$k_{i,11}$	$k_{i,12}$	$k_{i,13}$	$k_{i,14}$	$k_{i,15}$	$k_{i,16}$	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$	$r_{i,4}$	$r_{i,5}$	$r_{i,6}$	$r_{i,7}$	$r_{i,8}$
$r_{8,2}$	$r_{8,3}$	$r_{8,4}$	$r_{8,5}$	$r_{8,6}$	$r_{8,7}$	$r_{8,8}$	$k_{8,1}$	$k_{8,2}$	$k_{8,3}$	$k_{8,4}$	$k_{8,5}$	$k_{8,6}$	$k_{8,7}$	$k_{8,8}$	$k_{8,9}$	$k_{8,10}$	$k_{8,11}$	$k_{8,12}$	$k_{8,13}$	$k_{8,14}$	$k_{8,15}$	$k_{8,16}$	$r_{8,1}$
$r_{7,3}$	$r_{7,4}$	$r_{7,5}$	$r_{7,6}$	$r_{7,7}$	$r_{7,8}$	$k_{7,1}$	$k_{7,2}$	$k_{7,3}$	$k_{7,4}$	$k_{7,5}$	$k_{7,6}$	$k_{7,7}$	$k_{7,8}$	$k_{7,9}$	$k_{7,10}$	$k_{7,11}$	$k_{7,12}$	$k_{7,13}$	$k_{7,14}$	$k_{7,15}$	$k_{7,16}$	$r_{7,1}$	$r_{7,2}$
$r_{6,4}$	$r_{6,5}$	$r_{6,6}$	$r_{6,7}$	$r_{6,8}$	$k_{6,1}$	$k_{6,2}$	$k_{6,3}$	$k_{6,4}$	$k_{6,5}$	$k_{6,6}$	$k_{6,7}$	$k_{6,8}$	$k_{6,9}$	$k_{6,10}$	$k_{6,11}$	$k_{6,12}$	$k_{6,13}$	$k_{6,14}$	$k_{6,15}$	$k_{6,16}$	$r_{6,1}$	$r_{6,2}$	$r_{6,3}$
$r_{5,5}$	$r_{5,6}$	$r_{5,7}$	$r_{5,8}$	$k_{5,1}$	$k_{5,2}$	$k_{5,3}$	$k_{5,4}$	$k_{5,5}$	$k_{5,6}$	$k_{5,7}$	$k_{5,8}$	$k_{5,9}$	$k_{5,10}$	$k_{5,11}$	$k_{5,12}$	$k_{5,13}$	$k_{5,14}$	$k_{5,15}$	$k_{5,16}$	$r_{5,1}$	$r_{5,2}$	$r_{5,3}$	$r_{5,4}$
$r_{4,6}$	$r_{4,7}$	$r_{4,8}$	$k_{4,1}$	$k_{4,2}$	$k_{4,3}$	$k_{4,4}$	$k_{4,5}$	$k_{4,6}$	$k_{4,7}$	$k_{4,8}$	$k_{4,9}$	$k_{4,10}$	$k_{4,11}$	$k_{4,12}$	$k_{4,13}$	$k_{4,14}$	$k_{4,15}$	$k_{4,16}$	$r_{4,1}$	$r_{4,2}$	$r_{4,3}$	$r_{4,4}$	$r_{4,5}$
$r_{3,7}$	$r_{3,8}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$	$k_{3,6}$	$k_{3,7}$	$k_{3,8}$	$k_{3,9}$	$k_{3,10}$	$k_{3,11}$	$k_{3,12}$	$k_{3,13}$	$k_{3,14}$	$k_{3,15}$	$k_{3,16}$	$r_{3,1}$	$r_{3,2}$	$r_{3,3}$	$r_{3,4}$	$r_{3,5}$	$r_{3,6}$
$r_{2,8}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$	$k_{2,6}$	$k_{2,7}$	$k_{2,8}$	$k_{2,9}$	$k_{2,10}$	$k_{2,11}$	$k_{2,12}$	$k_{2,13}$	$k_{2,14}$	$k_{2,15}$	$k_{2,16}$	$r_{2,1}$	$r_{2,2}$	$r_{2,3}$	$r_{2,4}$	$r_{2,5}$	$r_{2,6}$	$r_{2,7}$
$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$	$k_{1,6}$	$k_{1,7}$	$k_{1,8}$	$k_{1,9}$	$k_{1,10}$	$k_{1,11}$	$k_{1,12}$	$k_{1,13}$	$k_{1,14}$	$k_{1,15}$	$k_{1,16}$	$r_{1,1}$	$r_{1,2}$	$r_{1,3}$	$r_{1,4}$	$r_{1,5}$	$r_{1,6}$	$r_{1,7}$	$r_{1,8}$

Після виконання операцій зсуву вправо, виконують операції циклічного зсуву бітів вгору (від молодших бітів до старших) згідно з алгоритмом кодування (наведений на рис. 2.12). Розміщення інформації в буфері передачі після виконання цих операцій зсуву вгору показано в табл. 2.8.

Таблиця 2.8 — Розміщення інформації в буфері передачі після виконання операцій зсуву (від молодших бітів до старших) байтів у буфері передачі

Байти буфера передачі																							
$n_{i,1}$	$n_{i,2}$	$n_{i,3}$	$n_{i,4}$	$n_{i,5}$	$n_{i,6}$	$n_{i,7}$	$n_{i,8}$	$n_{i,9}$	$n_{i,10}$	$n_{i,11}$	$n_{i,12}$	$n_{i,13}$	$n_{i,14}$	$n_{i,15}$	$n_{i,16}$	$n_{i,17}$	$n_{i,18}$	$n_{i,19}$	$n_{i,20}$	$n_{i,21}$	$n_{i,22}$	$n_{i,23}$	$n_{i,24}$
Інформаційні біти																Контрольні біти							
$k_{i,1}$	$k_{i,2}$	$k_{i,3}$	$k_{i,4}$	$k_{i,5}$	$k_{i,6}$	$k_{i,7}$	$k_{i,8}$	$k_{i,9}$	$k_{i,10}$	$k_{i,11}$	$k_{i,12}$	$k_{i,13}$	$k_{i,14}$	$k_{i,15}$	$k_{i,16}$	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$	$r_{i,4}$	$r_{i,5}$	$r_{i,6}$	$r_{i,7}$	$r_{i,8}$
$r_{8,2}$	$r_{7,4}$	$r_{6,6}$	$r_{5,8}$	$k_{4,2}$	$k_{3,4}$	$k_{2,6}$	$k_{1,8}$	$k_{8,2}$	$k_{7,4}$	$k_{5,6}$	$k_{5,8}$	$k_{4,10}$	$k_{3,12}$	$k_{2,14}$	$k_{1,16}$	$k_{8,10}$	$k_{7,12}$	$k_{6,14}$	$k_{5,16}$	$r_{4,2}$	$r_{3,4}$	$r_{2,6}$	$r_{1,8}$
$r_{7,3}$	$r_{6,5}$	$r_{5,7}$	$k_{4,1}$	$k_{3,3}$	$k_{2,5}$	$k_{1,7}$	$k_{8,1}$	$k_{7,3}$	$k_{5,5}$	$k_{5,7}$	$k_{4,9}$	$k_{3,11}$	$k_{2,13}$	$k_{1,15}$	$k_{8,9}$	$k_{7,11}$	$k_{6,13}$	$k_{5,15}$	$r_{4,1}$	$r_{3,3}$	$r_{2,5}$	$r_{1,7}$	$r_{8,1}$
$r_{6,4}$	$r_{5,6}$	$r_{4,8}$	$k_{3,2}$	$k_{2,4}$	$k_{1,6}$	$r_{8,8}$	$k_{7,2}$	$k_{5,4}$	$k_{5,6}$	$k_{4,8}$	$k_{3,10}$	$k_{2,12}$	$k_{1,14}$	$k_{8,8}$	$k_{7,10}$	$k_{6,12}$	$k_{5,14}$	$k_{4,16}$	$r_{3,3}$	$r_{2,4}$	$r_{1,6}$	$k_{8,16}$	$r_{7,2}$
$r_{5,5}$	$r_{4,7}$	$k_{3,1}$	$k_{2,3}$	$k_{1,5}$	$r_{8,7}$	$k_{7,1}$	$k_{6,3}$	$k_{5,5}$	$k_{4,7}$	$k_{3,9}$	$k_{2,11}$	$k_{1,13}$	$k_{8,7}$	$k_{7,9}$	$k_{6,11}$	$k_{5,13}$	$k_{4,15}$	$r_{3,1}$	$r_{2,3}$	$r_{1,5}$	$k_{8,15}$	$r_{7,1}$	$r_{6,3}$
$r_{4,6}$	$r_{4,8}$	$k_{2,2}$	$k_{1,4}$	$r_{8,6}$	$r_{7,8}$	$k_{6,2}$	$k_{5,4}$	$k_{4,6}$	$k_{3,8}$	$k_{2,10}$	$k_{1,12}$	$k_{8,6}$	$k_{7,8}$	$k_{6,10}$	$k_{5,12}$	$k_{4,14}$	$k_{3,16}$	$r_{2,2}$	$r_{1,4}$	$k_{8,14}$	$k_{7,16}$	$r_{6,2}$	$r_{5,4}$
$r_{3,7}$	$k_{2,1}$	$k_{1,3}$	$r_{8,5}$	$r_{7,7}$	$k_{6,1}$	$k_{5,3}$	$k_{4,5}$	$k_{3,7}$	$k_{2,9}$	$k_{1,11}$	$k_{8,5}$	$k_{7,7}$	$k_{6,9}$	$k_{5,11}$	$k_{4,13}$	$k_{3,15}$	$r_{2,1}$	$r_{1,3}$	$k_{8,13}$	$k_{7,15}$	$r_{6,1}$	$r_{5,3}$	$r_{4,5}$
$r_{2,8}$	$k_{1,2}$	$r_{8,4}$	$r_{7,6}$	$r_{6,8}$	$k_{5,2}$	$k_{4,4}$	$k_{3,6}$	$k_{2,8}$	$k_{1,10}$	$k_{8,4}$	$k_{7,6}$	$k_{5,8}$	$k_{4,10}$	$k_{3,12}$	$k_{2,14}$	$k_{1,16}$	$r_{1,2}$	$k_{8,12}$	$k_{7,14}$	$k_{6,16}$	$r_{5,2}$	$r_{4,4}$	$r_{3,6}$
$k_{1,1}$	$r_{8,3}$	$r_{7,5}$	$r_{6,7}$	$k_{5,1}$	$k_{3,3}$	$k_{3,5}$	$k_{2,7}$	$k_{1,9}$	$k_{8,3}$	$k_{7,5}$	$k_{5,7}$	$k_{4,11}$	$k_{3,13}$	$k_{2,15}$	$r_{1,1}$	$k_{8,11}$	$k_{7,13}$	$k_{6,15}$	$k_{5,1}$	$r_{4,3}$	$r_{3,5}$	$r_{2,7}$	

Передача даних у канал зв'язку здійснюється послідовно байт за байтом відповідно до структури буфера, наведеної в табл. 2.8.

2.6.3 Алгоритм декодування

Процес декодування на приймальній стороні виконується відповідно до розробленого алгоритму (рис. 2.13). Прийнятий інформаційний блок розміром 24 байти розміщується в буфері прийому. Після отримання повного блоку даних виконується процедура зворотного перемішування у спосіб послідовного застосування операцій побітового зсуву до кожного байту буфера в порядку від старшого до молодшого, що є інверсією відносно процедури кодування.

Наступний етап декодування передбачає виконання операцій циклічного зсуву вліво, що є інверсними до операцій зсуву при кодуванні. Після завершення процедури відновлення даних, структура буфера прийому відповідає початковій схемі кодування (табл. 2.4), при цьому окремі біти можуть містити помилки, спричинені завадами в каналі зв'язку.

Процедура декодування формує 16 синдромів (див табл. 2.3), що забезпечують виявлення помилок у кодових словах та їх виправлення в інформаційних бітах прийнятих даних.

Процедура виявлення та виправлення помилок в інформаційних бітах реалізується на основі таблиці відповідності синдромів (табл. 2.3), що містить шістнадцяткові значення всіх можливих комбінацій синдромів.

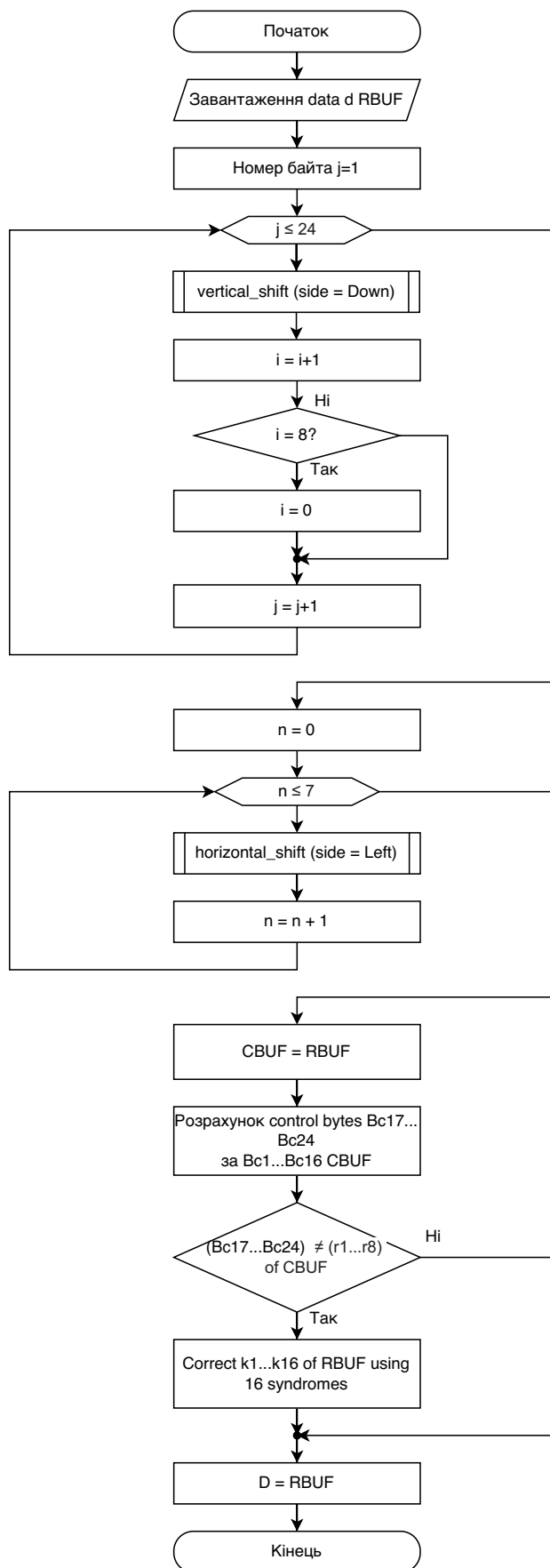


Рисунок 2.13 — Блок-схема алгоритму декодування

При виявленні розбіжності між отриманою та обчисленою контрольними сумами можливі наступні сценарії:

- невідповідність одного біту контрольної суми свідчить про помилку в контрольних бітах при збереженні цілісності інформаційних даних;
- невідповідність декількох бітів контрольної суми потребує додаткового аналізу допустимості помилок. У випадку виявлення одиничної помилки в інформаційному біті, її локалізація здійснюється за допомогою контрольних бітів з подальшим виправленням у спосіб інвертування пошкодженого біту;
- інші комбінації невідповідностей класифікуються як множинні помилки, що унеможливають відновлення даних, але забезпечують детектування факту порушення цілісності інформації.

Запропонований метод передачі інформації по каналах зв'язку блоками довжиною 24 байта дозволяє здійснювати діагностичне виявлення та виправлення помилок передачі на приймальному боці. Навіть у випадках множинних (багатобітових) помилок в межах одного байту, що належить до певного інформаційного блоку.

Алгоритми кодування та декодування реалізують відмінний від відомих метод виявлення та виправлення багатобітових помилок, котрі виникають при передачі інформації по каналу зв'язку.

Цей алгоритм, який реалізує процедури декодування, виявлення, а також виправлення помилок наведено на рис. 2.13.

Метод декодування отримує на вхід послідовність байтів фіксованого розміру та здійснює її декодування за допомогою модифікованого коду Хеммінга. Вхідна послідовність спочатку розміщується у буфері прийому, розмір якого визначається на основі довжини вхідних даних. Процес декодування виконується у спосіб послідовного застосування зворотних операцій відносно процедури кодування: спочатку здійснюється відновлення початкового розташування бітів через операції зворотного зсуву, після чого виконується аналіз та корекція можливих помилок на основі обчислених синдромів.

2.7 Особливості програмної реалізації модифікованого коду Хеммінга

Мова програмування Rust, розроблена Mozilla Research, демонструє унікальне поєднання характеристик, що роблять її оптимальним вибором для реалізації високонадійних систем виявлення та виправлення помилок. Система володіння пам'яттю (ownership system) та механізм позичання (borrowing) у Rust забезпечують безпеку пам'яті без використання збирача сміття, що є критичним аспектом для систем реального часу та вбудованих систем, де можуть застосовуватися коди Хеммінга. У порівнянні з C++, який також забезпечує низькорівневий контроль, Rust надає гарантії безпеки на рівні компіляції, що унеможлиблює виникнення помилок доступу до пам'яті та стану гонитви даних.

Запропонований метод виявлення та виправлення багатобітових помилок на основі модифікації кодів Хеммінга демонструє інноваційний підхід до підвищення надійності передачі даних. Метод базується на класичному коді Хеммінга, який доповнюється матричними перетвореннями для покращення можливостей виявлення помилок. Імплементація даного методу в Rust дозволяє ефективно реалізувати складні матричні операції завдяки абстракціям з нульовою вартістю та потужній системі типів.

Архітектура програмного забезпечення демонструє чітке розділення відповідальності між компонентами, де HammingCodec виступає основним координатором процесів кодування та декодування, а Matrix2D забезпечує ефективні матричні операції. Використання узагальнених типів (generics) у структурі Matrix2D свідчить про гнучкість та можливість повторного використання коду, що є важливим аспектом при розробці бібліотечних компонентів.

2.7.1 Проєктування та реалізація параметричного класу Matrix2D для оптимізованої обробки двовимірних структур даних

Клас Matrix2D являє собою параметризовану структуру даних, що реалізує ефективне представлення двовимірної матриці з оптимізованими операціями циклічного зсуву.

Внутрішня архітектура класу базується на одновимірному векторі для зберігання елементів матриці (*data: Vec<T>*), що забезпечує ефективне використання пам'яті та покращує локальність даних. Додатково структура містить метадані про розміри матриці (*rows: usize, cols: usize*) та вектори зміщень для рядків та стовпців (*row_starts: Vec<usize>, col_starts: Vec<usize>*), які використовуються для реалізації ефективних операцій обертання без фізичного переміщення даних у пам'яті.

Основні методи класу описані у таблиці 2.9:

Таблиця 2.9 — Основні методи класу Matrix2D

Категорія	Метод	Опис
Конструктори	<i>from_vec</i>	Створення матриці з вектора ($O(rc)$)
	<i>new</i>	Ініціалізація порожньої матриці
Доступ та модифікація	<i>get</i>	Отримання елемента за індексами
	<i>set</i>	Встановлення значення елемента
	<i>dimensions</i>	Отримання розмірів матриці
Операції обертання	<i>rotate_row_right</i>	Циклічний зсув рядка праворуч
	<i>rotate_row_left</i>	Циклічний зсув рядка ліворуч
	<i>rotate_column_up</i>	Циклічний зсув стовпця вгору
	<i>rotate_column_down</i>	Циклічний зсув стовпця вниз

Особливістю реалізації є використання віртуальних зміщень замість фізичного переміщення даних при операціях обертання, що забезпечує константну складність $O(1)$ для операцій зсуву. Це досягається у спосіб підтримки векторів зміщень та відповідного перерахунку індексів при доступі до елементів.

Реалізація класу демонструє застосування наступних принципів проєктування:

- інкапсуляція внутрішнього представлення даних;
- параметричний поліморфізм через використання узагальнених типів;
- ефективне управління пам'яттю завдяки використанню контейнера Vec;
- оптимізація продуктивності через віртуалізацію операцій зсуву.

Даний клас відіграє ключову роль у реалізації модифікованого алгоритму Хеммінга, забезпечуючи ефективні операції з матрицями при кодуванні та декодуванні даних.

2.7.2 Проєктування та імплементація програмного забезпечення HammingCodec для реалізації завадостійкого кодування з розширеними можливостями діагностики

Клас HammingCodec являє собою реалізацію модифікованого коду Хеммінга (12,8) з розширеними можливостями відстеження процесу кодування та декодування.

Внутрішня архітектура класу, представлена на діаграмі класів (рис. 2.14), складається з буфера фіксованого розміру для зберігання закодованих даних (*buffer_size* = 24 байти), розміру початкових даних (*data_size* = 16 байтів) та журналу покрокового виконання операцій (*steps*). Кожен крок кодування або декодування зберігається у структурі *EncodingStep*, що містить опис операції, поточний стан даних та тип виконуваної операції.

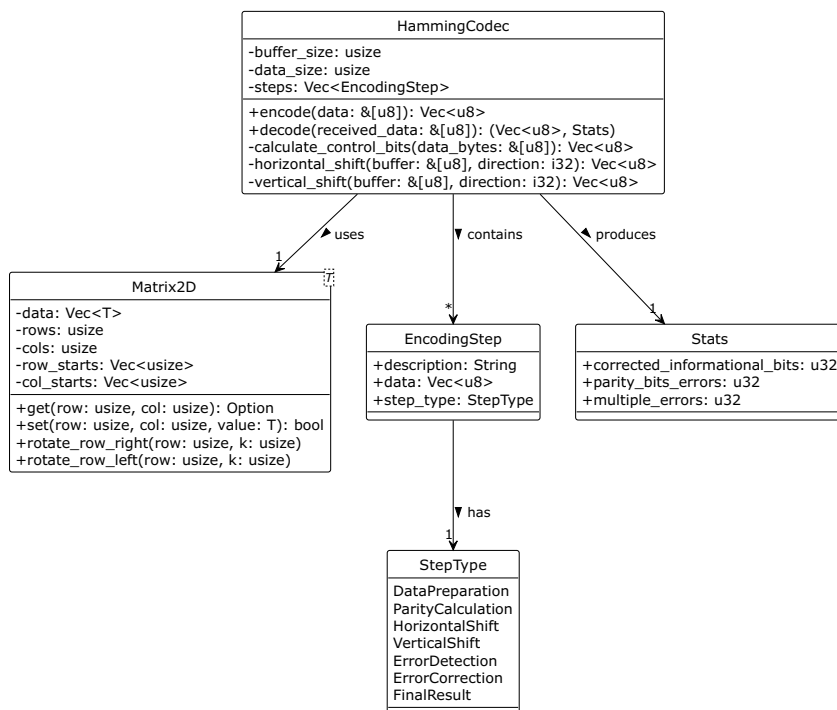


Рисунок 2.14 — Діаграма класів для бібліотеки кодування та декодування HammingCodec

Процес кодування, візуалізований на діаграмі діяльності (рис. 2.12), починається з валідації вхідних даних та їх підготовки до кодування. Далі виконується обчислення контрольних бітів згідно з матрицею перевірки парності (табл. 2.2). Особливістю реалізації є застосування операцій горизонтального та вертикального зсуву для підвищення завадостійкості коду.

Процес декодування, представлений на діаграмі діяльності (рис. 2.13), включає зворотні операції зсуву, виявлення та виправлення помилок на основі обчислених синдромів. Статистика процесу декодування накопичується у структурі *Stats*, що містить інформацію про кількість виправлених інформаційних бітів, помилок у контрольних бітах та виявлених множинних помилок.

Взаємодія з матричними операціями здійснюється через клас *Matrix2D*, що забезпечує ефективну реалізацію операцій зсуву та маніпуляцій з бітами. Архітектура програмного забезпечення демонструє чітке розділення відповідальності між компонентами, де *HammingCodec* виступає основним координатором процесів кодування та декодування.

Реалізація класу в мові Rust забезпечує оптимальну продуктивність завдяки відсутності накладних витрат на керування пам'яттю та використанню ефективних структур даних. Система типів Rust гарантує безпеку роботи з пам'яттю на етапі компіляції, що є критичним для систем передачі даних.

2.7.3 Проєктування та імплементація користувацького інтерфейсу для діагностики процесів завадостійкого кодування

Графічний інтерфейс користувача реалізовано з використанням бібліотеки *egui*, що забезпечує кросплатформну розробку нативних додатків на Rust. Основні константи та іконки інтерфейсу визначено у файлі *gui.rs*. Візуальні елементи збагачено Unicode-іконками для покращення користувацького досвіду та інтуїтивного розуміння стану операцій.

Архітектура GUI базується на реактивному підході, де стан додатку автоматично відображається у візуальних компонентах. Головне вікно програми

містить три основні секції: панель введення даних, панель керування операціями та панель відображення результатів.

Панель введення даних дозволяє користувачу вводити 16-байтові послідовності у шістнадцятковому форматі. Реалізовано валідацію введення з візуальним показом помилок та підказками щодо коректного формату даних. Додатково передбачено можливість генерації випадкових тестових послідовностей для демонстрації роботи алгоритму.

Процес кодування візуалізується через покрокове відображення операцій (рис. 2.15), де кожен крок супроводжується відповідною іконкою та описом виконуваної операції. Кольорове кодування допомагає швидко ідентифікувати тип операції: підготовка даних, обчислення контрольних бітів, операції зсуву.

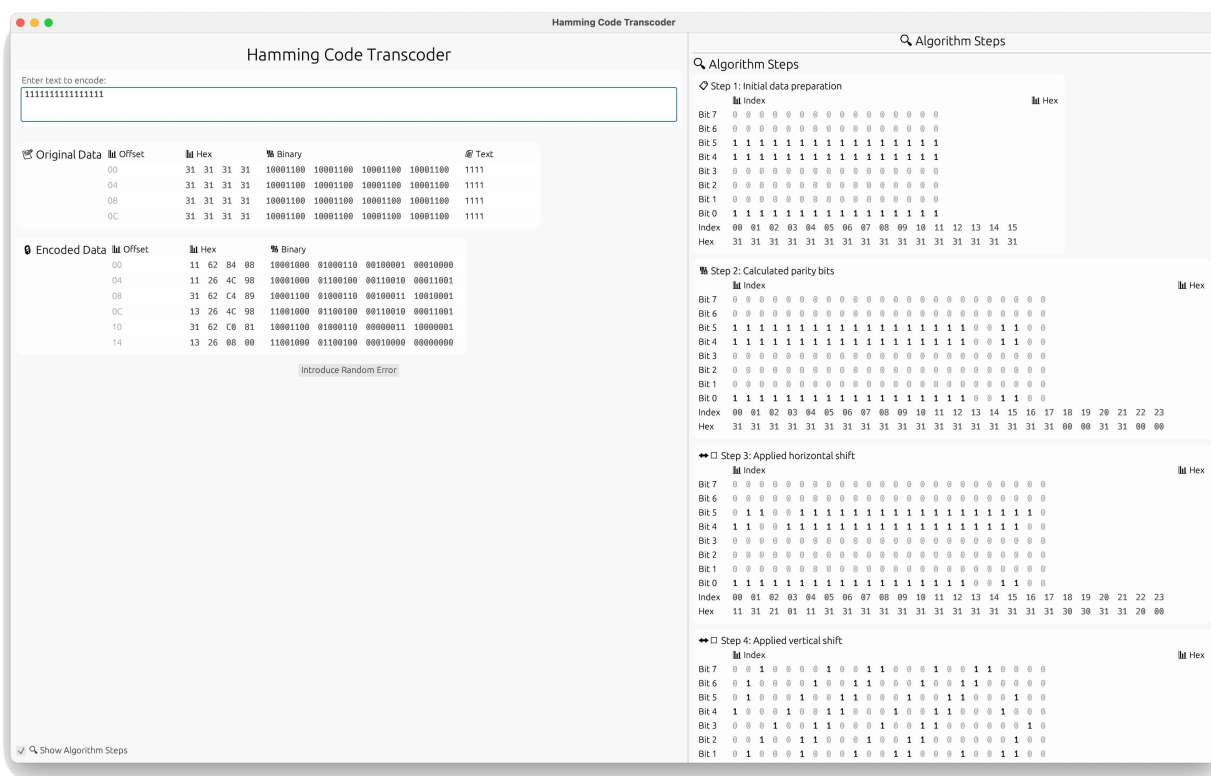


Рисунок 2.15 — Візуалізація процесу кодування з покроковим відображенням операцій

Особливу увагу приділено візуалізації процесу виявлення та виправлення помилок (рис. 2.16). Інтерфейс наочно демонструє позиції виявлених помилок, тип помилки (одинична чи множинна) та результат корекції.

Статистика декодування відображається у компактній формі з використанням інформативних іконок.

Реалізація інтерфейсу забезпечує асинхронне виконання операцій кодування та декодування, що дозволяє зберігати чуйність інтерфейсу навіть при обробці великих обсягів даних. Система обробки помилок інтегрована з візуальним представленням, що забезпечує миттєвий зворотний зв'язок користувачу про стан операцій.

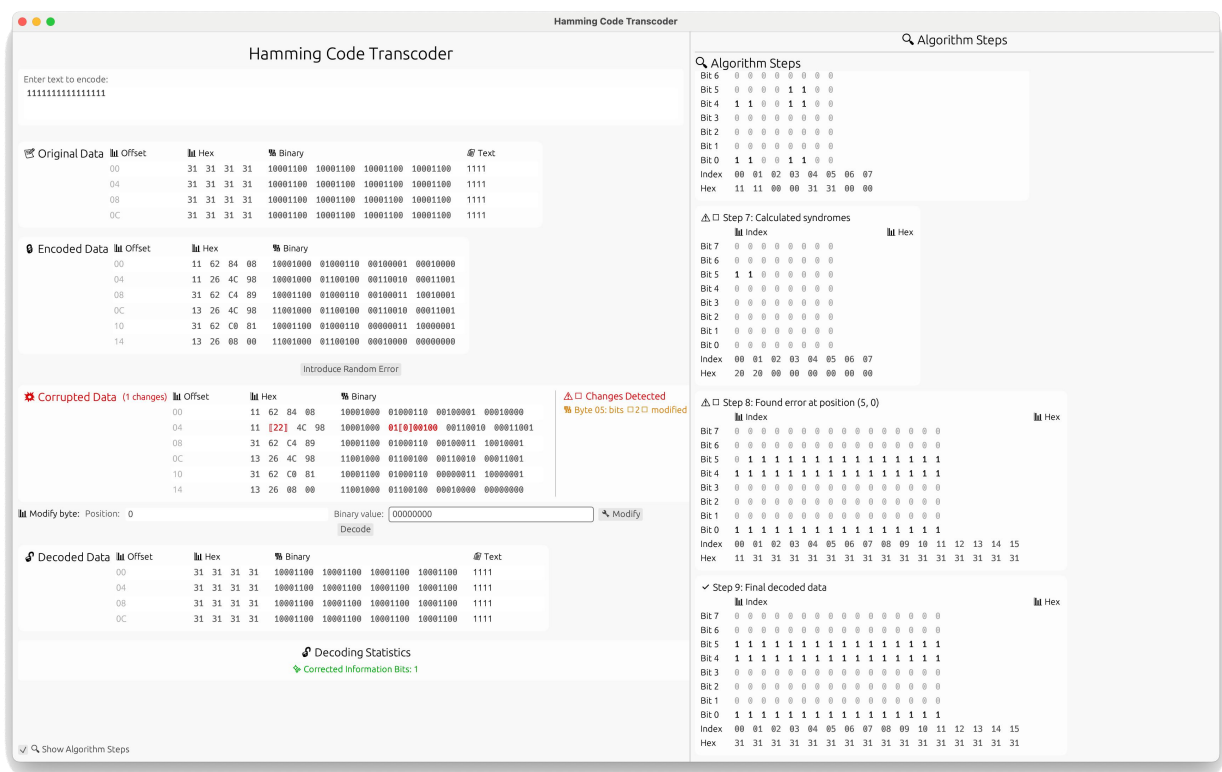


Рисунок 2.16 — Інтерфейс виявлення та виправлення помилок

Для забезпечення доступності інтерфейсу реалізовано підтримку клавіатурної навігації та комбінацій клавіш для основних операцій. Усі текстові елементи інтерфейсу локалізовано та оформлено з урахуванням принципів технічної документації, що полегшує розуміння процесів кодування та декодування.

2.7.4 Методологія та результати тестування програмного забезпечення

Тестове покриття розробленої бібліотеки реалізовано на декількох рівнях абстракції, що забезпечує всебічну перевірку коректності роботи системи. Загальний показник покриття коду тестами (Code Coverage) становить 98.7%, що підтверджується звітами інструменту cargo-tarpaulin. Кожен публічний метод та структура супроводжуються вичерпною документацією у форматі rustdoc, що відповідає стандартам оформлення технічної документації Rust.

Інтеграція з системою автоматизованого тестування cargo-llvm-cov забезпечує генерацію метрик покриття коду на рівні інструкцій, що дозволяє здійснювати кількісну оцінку повноти тестування та формувати структуровані звіти у форматі HTML для подальшого аналізу та документування результатів верифікації програмного забезпечення.

Процес документування результатів інтеграційного тестування реалізується через формування структурованих звітів, що містять детальний опис виконаних тестових сценаріїв, аналіз виявлених особливостей функціонування системи та рекомендації щодо оптимізації взаємодії компонентів програмного забезпечення. Результати тестування використовуються для подальшого вдосконалення архітектури системи та підвищення надійності механізмів взаємодії між її компонентами.

Документація тестів інтегрована з основною документацією проекту та доступна через cargo doc. Кожен тестовий випадок супроводжується детальним описом мети, вхідних даних, очікуваних результатів та критеріїв успішності, що значно полегшує подальшу підтримку та розширення тестового покриття.

Базовий рівень тестування реалізує комплексну методологію валідації структури Matrix2D, що забезпечує систематичну перевірку коректності функціонування матричних операцій в умовах різних сценаріїв використання програмного продукту.

Архітектура тестового забезпечення структурована відповідно до принципів модульного тестування та включає верифікацію конструкторів класу, де здійснюється валідація коректності ініціалізації об'єктів через методи *Matrix2D::new()* та *Matrix2D::from_vec()* з подальшою верифікацією відповідності розмірностей та цілісності структури даних.

Процес верифікації операцій доступу до елементів матричної структури реалізується через систематичне тестування методів `get()` та `set()`, що забезпечує валідацію коректності індексації та обробки граничних випадків при взаємодії з даними, включаючи перевірку стійкості програмного забезпечення до некоректних вхідних параметрів.

Комплексне тестування маніпуляцій з даними охоплює валідацію операцій трансформації матричної структури, включаючи верифікацію методів обертання рядків та стовпців (*`rotate_row_right()`*, *`rotate_row_left()`*, *`rotate_column_up()`*, *`rotate_column_down()`*), з особливою увагою до збереження цілісності даних при виконанні комбінованих операцій трансформації.

Методологія тестування граничних випадків забезпечує верифікацію стійкості програмного забезпечення в умовах екстремальних сценаріїв використання, включаючи обробку порожніх структур даних та валідацію коректності функціонування при граничних значеннях параметрів трансформації.

Інтеграційне тестування програмного забезпечення `HammingCodec` реалізує комплексну методологію верифікації взаємодії компонентів системи, де основним об'єктом дослідження виступає клас `HammingCodec` та його інтеграція з суміжними модулями програмного комплексу. Процес тестування базується на систематичному підході до валідації коректності функціонування механізмів кодування та декодування інформації в контексті їх взаємодії з матричними структурами даних та системою обробки помилок.

Методологія інтеграційного тестування передбачає верифікацію процесів трансформації даних на всіх етапах їх обробки, включаючи валідацію коректності конверсії бітових послідовностей, перевірку цілісності даних при передачі між компонентами системи та верифікацію механізмів синхронізації станів об'єктів у процесі виконання операцій кодування та декодування. Особлива увага приділяється тестуванню сценаріїв безпомилкової передачі даних, що забезпечує валідацію базової функціональності системи в умовах штатного функціонування.

Тестове забезпечення інтеграційного рівня реалізує комплексний підхід до верифікації взаємодії між компонентами `Matrix2D` та `HammingCodec`,

включаючи тестування коректності передачі даних між модулями, валідацію процесів трансформації інформації та перевірку узгодженості станів системи при виконанні послідовності операцій кодування-декодування. Процес тестування супроводжується детальним документуванням результатів та формуванням звітів про виявлені особливості функціонування системи.

Методика інтеграційного тестування включає верифікацію механізмів обробки виняткових ситуацій, що виникають на межі взаємодії компонентів системи, забезпечуючи валідацію коректності обробки помилок та відновлення штатного функціонування програмного забезпечення після виникнення аномальних ситуацій. Особлива увага приділяється тестуванню синхронізації станів об'єктів та верифікації коректності передачі контрольної інформації між модулями системи.

Особливу увагу приділено тестуванню завадостійкості системи, що реалізовано через імітацію апаратних збоїв та стрес-тестування. Методика тестування реалізує комплексну методологію верифікації стійкості системи до апаратних та програмних збоїв через систематичну імітацію аномальних ситуацій у контрольованому середовищі. Процес тестування базується на генерації стохастичних послідовностей бітових помилок з подальшою валідацією механізмів їх виявлення та корекції, що забезпечує всебічну оцінку надійності системи в умовах потенційних збоїв.

Оцінка завадостійкості запропонованого методу виправлення багатобітових помилок здійснювалася за допомогою програмної моделі каналу передачі даних, структурна схема якої представлена на рис. 2.17.

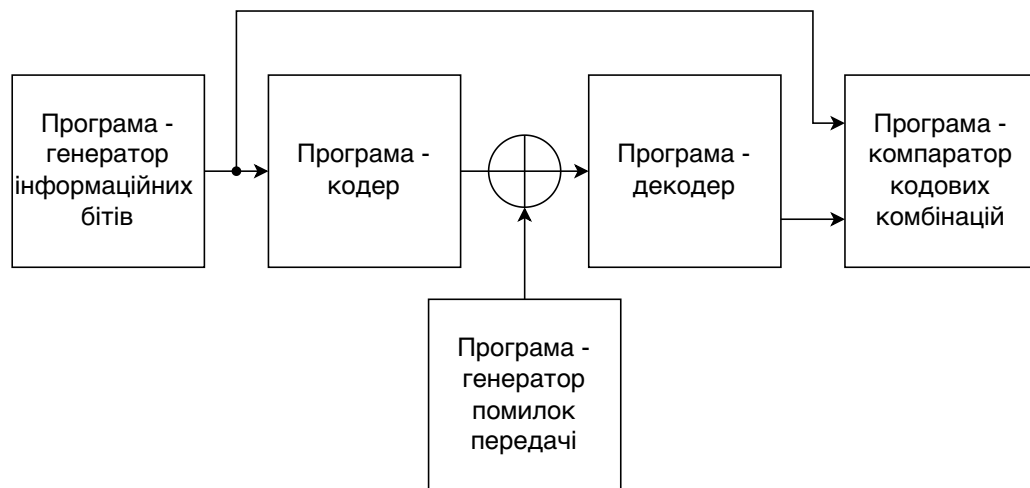


Рисунок 2.17 — Структурна схема програмної моделі каналу передачі даних

Структурна схема програмної моделі складається з наступних компонентів зазначених у таблиці 2.10.

Таблиця 2.10 — Компоненти програмної моделі каналу передачі даних

Компонент	Функціональне призначення
Generator	Формування тестових 16-байтових інформаційних послідовностей для процедури кодування
Coder	Реалізація розробленої схеми завадостійкого кодування з використанням модифікованого коду Хеммінга
Error	Імітація спотворень даних у каналі зв'язку у спосіб внесення контрольованих бітових помилок
Decoder	Реалізація алгоритму виявлення та виправлення помилок на основі модифікованого коду Хеммінга
Comparator	Валідація коректності відновлення даних у спосіб порівняння декодованої інформації з вихідною послідовністю

Експериментальне дослідження проводилось в автоматизованому режимі з можливістю динамічного контролю параметрів експерименту. Імітація спотворень у каналі зв'язку реалізована у спосіб застосування операції виключного АБО (\oplus) до закодованих інформаційних послідовностей. У процесі

експериментального дослідження було проаналізовано 6120 інформаційних блоків, що забезпечило статистичну значущість отриманих результатів.

Методика оцінки завадостійкості системи базується на статистичному аналізі результатів тестування, де ключовими метриками виступають співвідношення кратності помилок до кількості успішно декодованих блоків, коефіцієнт виявлення помилок як відношення кількості виявлених спотворених бітів до загальної кількості переданих бітів, а також ефективність корекції, що характеризується співвідношенням кількості виправлених інформаційних бітів до загальної кількості помилок в інформаційних бітах. Додатковим показником надійності системи слугує її стійкість до множинних помилок, що визначається через аналіз кількості не виправлених помилок у бітах парності та фатальних багатобітових помилок.

Результати експериментального дослідження завадостійкості представлені в таблиці 2.11:

Таблиця 2.11 — Результати тестування запропонованого методу виявлення та виправлення багаторозрядних помилок

Кратність помилки в одному байті блока	Кількість переданих блоків	Кількість переданих бітів	Кількість спотворених бітів	Кількість помилку інформаційних бітах	Кількість виправлених інформаційних бітів	Кількість помилки у бітах парності (не коригують)
1	192	36864	192	128	128	64
2	672	129024	1344	896	896	448
3	1344	258048	4032	2688	2688	1344
4	1680	322560	6720	4480	4480	2240
5	1344	258048	6720	4480	4480	2240
6	672	129024	4032	2688	2688	1344
7	192	36864	1344	896	896	448
8	24	4608	192	128	128	64

Експериментальні дослідження, проведені на вибірці 6120 блоків даних, демонструють високу ефективність системи при виявленні та корекції помилок різної кратності, підтверджуючи здатність системи забезпечувати надійну передачу даних навіть при наявності множинних помилок у межах одного байту інформаційного блоку.

За результатами експериментального дослідження встановлено:

1. При одиничних помилках у байті блоку, незалежно від їх кратності (1-8), забезпечується повне відновлення інформаційних бітів

2. При максимальній кратності помилок (8 біт) у байті мінімальна кількість блоків з помилками становить 24, що відповідає 128 пошкодженим інформаційним бітам, які успішно відновлюються

3. Оптимальна ефективність корекції досягається при кратності помилок 4, де система здатна обробити до 322560 блоків з помилковими байтами, що відповідає 4480 успішно відновленим інформаційним бітам

2.8 Архітектура давачів інформації завадостійкої системи моніторингу стану потенційно небезпечних об'єктів на базі технологій Інтернету речей

Джерелом інформації про стан ПНО у регіональній системі моніторингу є давачі інформації. Своєю чергою, первинна інформація до давачів надходить від первинних перетворювачів. Для передачі первинної інформації про стан об'єкта від первинного перетворювача до схеми давача запропоновано використовувати аналогові канали передачі даних, які використовують у промислових системах контролю. У таких системах як промисловий стандарт зазвичай використовують струм, що змінюється в межах 4...20 мА. При цьому струм величиною 4 мА відповідає нульовому зовнішньому сигналу, а струм 20 мА – максимальному. Розроблена архітектура для давача інформації наведено на рисунку 2.18.

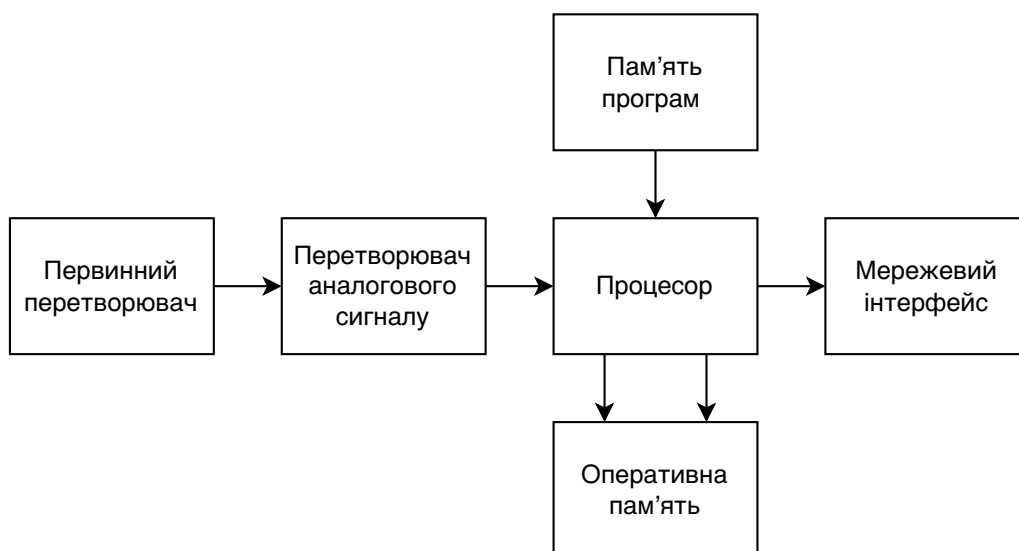


Рисунок 2.18 — Архітектура давача інформації для систем моніторингу стану ПНО

Існує багато методів виявлення та виправлення помилок передачі інформації. У даному дослідженні для передачі інформації від давачів інформації на локальний рівень було використано метод передачі інформаційних блоків, описаний в [12]. Цей метод дозволяє виявляти та виправляти багатобітові помилки при передачі інформаційного блоку. Метод базується на використанні кодів Хеммінга (12, 8) та модифікованій схемі кодування. Суть схеми кодування у тому, що контрольні біти розраховуються та додаються не до кожного інформаційному байту, який передається по каналу передачі даних, а згідно зі схемою розробленої схеми кодування. При цьому для перших восьми байтів повідомлення для бітів однакового розряду розраховується чотири контрольних біти. Тобто при передачі 16 байт інформації необхідно обчислити 64 біти парності. Шістнадцять інформаційних байтів утворюють матрицю бітів 8×16 , до якої додається матриця бітів парності 8×8 . У результаті інформація передається по каналах зв'язку у вигляді матриці 8×24 біта.

Найбільше збитків утворюються у випадку настання НС на гідротехнічних спорудах та ГЕС. Тому при проектуванні автоматизованих систем моніторингу стану ПНО для ГЕС повинні контролюватись параметри, перелік яких наведено у таблиці 2.12.

Таблиця 2.12 — Первинна (вихідна) інформація для виявлення системою моніторингу стану об'єкту ознак загрози виникнення НС на гідроспорудах та ГЕС.

№	Контрольована первинна (вихідна) інформація	Тип первинного перетворювача
1	Деформації елементів споруди.	Контроль цілісності шлейфів
2	Частоти обертання валу(ів) гідроагрегату(ів).	Тахометричні
3	Осідань та горизонтальних зміщень елементів споруди.	Контроль цілісності шлейфів
4	Фізико-хімічні параметри води	Ручний контактний сповіщувач
5	Рівні верхнього та нижнього б'єфів.	Ультразвукові рівнеміри
6	Поява та рівні води у приміщеннях оглядової галереї, турбінному приміщенні, приміщеннях головних виводів генераторів.	Електроконтактний / поплавковий
7	Наявність режиму пропуску паводкових вод.	Ручний контактний сповіщувач

Поєднання токового аналогового інтерфейсу $4 \dots 20 \text{ мА}$ та використання захищеного каналу зв'язку забезпечує стійку роботу давача, не зважаючи на вплив завад в умовах індустріального об'єкта.

Ще один позитивний результат від використання вхідного сигналу $4 - 20 \text{ мА}$ — це можливість виявлення стану помилки, позаяк токовий сигнал, навіть при найнижчому значенні відрізняється від нуля. Тобто в крайньому положенні «нуль» первинний перетворювач, як і раніше, видає сигнал 4 мА і якщо значення коли-небудь переходить у 0 мА , то вимірювальний ланцюг пошкоджено.

Гамма первинних перетворювачів у промисловому виконанні досить широка. Ось деякі приклади:

Первинний перетворювач тиску у токовий сигнал Pnometk серії P16 зображений на рисунку 2.19 застосовується для здійснення дистанційного моніторингу. Вихідний сигнал $4 - 20 \text{ мА}$.

Тахометричний первинний перетворювач кількості обертів у токовий сигнал ТЭ-6К-ТК зображений на рисунку 2.21.



Рисунок 2.21 — Тахометричний первинний перетворювач кількості обертів у токовий сигнал ТЭ-6К-ТК

У залежності від типу первинного перетворювача, що використовується, а також у спосіб внесення параметрів в програмне забезпечення давача, досить легко отримати точне та надійне джерело вхідної інформації для цілей моніторингу стану ПНО.

2.9 Висновки до розділу 2

1. Аналіз можливих варіантів архітектури для регіональної системи моніторингу стану ПНО дає підстави стверджувати, що є сенс використовувати IoT мережі з топологією «зірка», та аналітичної підсистеми на об'єктовому рівні.

2. Важливим отриманим результатом дослідження є розробка архітектури, алгоритмічного та програмного забезпечення як для системи моніторингу стану ПНО на регіональному рівні, так і для давачів інформації, які можуть бути використані як вузли збору первинної інформації у складі системи моніторингу стану ПНО.

3. Аналіз наявних методів передачі даних в IoT виявив необхідність розробки та застосування завадостійкого кодування з можливістю виявлення та виправлення багатобітових помилок передачі даних.

4. Впровадження запропонованого методу виявлення та виправлення помилок передачі дозволяє при передачі інформаційного блоку послідовно, байт за байтом, виявляти та виправляти помилки передачі (зберігання) окремого байту кратністю від 1 до 8-ми. Отримані результати пояснюються використанням модифікованої схеми кодування інформації, яка оснований на використанні кодів Хеммінга. Впровадження запропонованої схеми кодування стало можливим завдяки новим алгоритмам кодування-декодування.

5. Тестування програмної моделі каналу передачі підтвердило коригуючі можливості запропонованого методу виявлення та виправлення помилок. За результатами тестування можна стверджувати, що навіть при восьмикратній помилці передачі в межах одного байту інформаційного блоку фатальних помилок не виникає. Про це свідчить відсутність помилок при передачі 24 байт. При цьому було виправлено 128 інформаційних бітів зі 192 переданих. Це доводить, що розроблений метод завадостійкого кодування має сенс використовувати при побудові регіональних IoT моніторингу стану ПНО.

3 АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ АНАЛІТИЧНОЇ ПРЕДИКТИВНОЇ ПІДСИСТЕМИ РЕГІОНАЛЬНОЇ СИСТЕМИ МОНІТОРИНГУ СТАНУ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ОБ'ЄКТІВ

Основним завданням регіональної системи моніторингу стану ПНО є оперативне параметричне оцінювання даних та прогнозування можливості НС на ПНО в регіоні спостереження. Алгоритмічне та програмне забезпечення регіональної системи моніторингу стану ПНО забезпечує візуалізацію аналітичних показників, формалізовані звіти щодо поточного стану об'єктів та обґрунтовані прогностичні моделі. Зазначена інформація надходить до оператора системи через програмний вебінтерфейс, який також забезпечує термінове сповіщення при виявленні прогнозованого погіршення параметричних показників та несприятливих сценаріїв розвитку ситуації.

3.1 Обґрунтування необхідності підсистеми прогнозування в регіональних системах моніторингу стану потенційно небезпечних об'єктів

Сучасні системи моніторингу стану потенційно небезпечних об'єктів функціонують на основі параметричного контролю визначальних характеристик об'єкту. Алгоритмічне забезпечення здійснює фіксацію перевищення докритичних або критичних рівнів параметричних показників, що характеризують стан джерел небезпеки в межах ПНО. При виявленні такої події система формує термінове сповіщення оператора та забезпечує оповіщення персоналу та населення прилеглих територій. Проте така організація процесу моніторингу суттєво обмежує часовий ресурс оператора системи для реагування, прийняття обґрунтованих рішень та впровадження заходів щодо запобігання можливості НС.

З огляду на зазначене, пропонується удосконалений підхід до побудови систем моніторингу стану ПНО, які є об'єктами критичної інформаційної інфраструктури. Центральним компонентом системи має бути аналітична предиктивна підсистема, основним завданням якої є короткострокове

прогнозування динаміки параметричних показників, що характеризують стан джерел небезпеки для своєчасного виявлення критичних відхилень у стані об'єктів спостереження. При цьому алгоритмічне забезпечення підсистеми прогнозування має здійснювати обчислення прогнозних значень параметрів джерел небезпеки в межах ПНО на основі накопичених даних та поточних результатів моніторингу.

Повний комплексний прогноз зміни стану об'єктів, що спостерігаються, є складним завданням, для вирішення якого мають бути розроблені ефективні математичні моделі, що враховують прямий взаємовплив різних факторів.

Водночас прогнозування динаміки параметричних показників, що характеризують стан джерел техногенної небезпеки в межах ПНО, може бути реалізовано через застосування математичного апарату статистичного аналізу часових рядів. Такий підхід забезпечує формування обґрунтованих прогностичних моделей на основі накопичених даних моніторингу.

Часовий ряд являє собою впорядковану послідовність результатів параметричних вимірювань, зафіксованих у детерміновані моменти часу. Основним завданням аналізу часових рядів є формування прогностичної моделі, що дозволяє передбачити подальшу динаміку параметричних показників на основі їх поточних та попередніх значень.

Розглянемо організацію моніторингу стану гідроспоруд як критично важливих ПНО, аварії на яких можуть призвести до масштабних руйнувань та значних збитків. Відповідно до нормативних документів [21], на гідротехнічних спорудах, включаючи гідроелектростанції, для своєчасного виявлення можливості НС необхідно здійснювати контроль таких параметричних показників:

- деформації елементів споруди;
- частоти обертання валів гідроагрегату;
- осідання та горизонтальні зміщення елементів споруди;
- фізико-хімічні параметри води;
- рівні води у верхньому та нижньому б'єфах;
- рівень води у приямках (оглядова галерея, турбінне приміщення, приміщення головних виводів генераторів);

–режим пропуску повеневих та паводкових вод.

У дослідженні [17] обґрунтовано підхід до організації контролю зазначених параметрів через періодичні вимірювання з фіксацією часових міток та результатів моніторингу в пам'яті системи. Накопичені дані формують сукупність часових рядів, де кожний наступний результат може залежати від попереднього, що свідчить про наявність автокореляції. Аналіз таких рядів здійснюють для прогнозування динаміки параметричних показників у наступні періоди спостереження.

Прогнозування динаміки параметричних показників часового ряду надає можливість завчасного виявлення тенденцій до перевищення докритичних значень параметрів, що характеризують стан джерел небезпеки в межах ПНО. Це забезпечує оперативному персоналу додатковий часовий ресурс для прийняття обґрунтованих рішень та своєчасного впровадження заходів щодо запобігання можливості НС.

3.2 Систематизація наукових досліджень щодо методів прогнозування параметричних показників стану потенційно небезпечних об'єктів

У дослідженні [68] обґрунтовано фундаментальний підхід до аналізу часових рядів, згідно з яким структура часового ряду розглядається як композиція детермінованих та стохастичних компонентів. Детерміновані компоненти включають систематичну складову у вигляді тренду та циклічні компоненти, тоді як стохастична складова представлена випадковим процесом з характеристиками білого шуму.

Для визначення циклічних компонентів часового ряду доцільно застосовувати швидке перетворення Фур'є за умови, що кількість елементів ряду відповідає степеню числа два. Алгоритмічне забезпечення прогнозування в такому випадку ґрунтується на екстраполяції ідентифікованих циклічних компонентів часового ряду.

У дослідженні [69] запропоновано алгоритмічне забезпечення прогнозування на основі методу сингулярного спектрального аналізу, яке було апробовано на статистичних даних щодо обсягів будівельних робіт в Україні

за період 2010–2017 рр. Алгоритм ґрунтується на перетворенні одновимірного часового ряду в багатовимірний простір з подальшим формуванням прогностичної моделі. Верифікацію алгоритмічного забезпечення здійснено з використанням спеціалізованого програмного комплексу Caterpillar SSA.

Метод аналізу сингулярного спектра (АСС), детально обґрунтований у дослідженні [70], базується на послідовності математичних перетворень часового ряду. Алгоритмічне забезпечення методу передбачає виконання наступних етапів:

1. Реалізація процедури вкладення, при якій вихідний часовий ряд довжиною M елементів трансформується у послідовність векторів довжиною K через застосування зсувного вікна.

2. Формування траєкторної матриці розмірністю $K \times (M - K + 1)$, де кожний стовпець містить K послідовних елементів вихідного ряду.

3. Застосування сингулярного розкладання до траєкторної матриці, що забезпечує декомпозицію часового ряду на адитивні складові:

- довготривалі тренди;
- періодичні компоненти;
- осциляційні складові;
- стохастичні компоненти з характеристиками шуму.

Знайдені компоненти дозволяють виконати екстраполяцію як часового ряду у цілому, так і його компонентів [71].

Останнім часом у галузі інженерії програмного забезпечення значного поширення набув метод прогнозування часових рядів із застосуванням нейронних мереж. У дослідженні [72] здійснено ґрунтовний аналіз та обґрунтування вибору нейромережових структур для оброблення статистичних даних з метою прогнозування та виявлення аномальних показників у системах моніторингу ПНО.

Важливо зазначити, що алгоритмічне забезпечення на основі нейронних мереж суттєво відрізняється від традиційних експертних систем. На відміну від експертних систем, які базуються на попередньо запрограмованих правилах, нейронні мережі мають здатність до самонавчання та адаптації. Ця властивість набуває особливого значення в умовах динамічного середовища

функціонування ПНО. Алгоритмічне забезпечення на основі нейронних мереж здатне адаптувати свою роботу відповідно до змін у вхідних даних та корегувати результати прогнозування.

Здатність до самонавчання є визначальною характеристикою для застосування нейронних мереж у системах моніторингу потенційно небезпечних об'єктів. Це обумовлено нелінійною природою нейромережевих структур, що забезпечує формування математичних моделей для відображення складних функціональних залежностей між параметричними показниками стану об'єктів.

Алгоритмічне забезпечення на основі нейронних мереж реалізується через систему взаємопов'язаних обчислювальних елементів – нейронів, що здійснюють паралельне оброблення вхідних даних. У дослідженні [73] обґрунтовано застосування тришарового перцептрона як базової архітектури нейронної мережі. Зазначена архітектура, попри відносну простоту структурної організації, забезпечує ефективне розв'язання завдань прогнозування параметричних показників стану ПНО. Недоліком використання нейронних мереж є складність визначення параметричних характеристик нейромережевої структури, що потребує проведення серії обчислювальних експериментів з метою встановлення функціональних залежностей між показниками ефективності та параметрами нейромережі.

У дослідженні [74] розглянуто методологічні аспекти застосування нейромережевих структур для розв'язування задач бінарної класифікації за умов обмеженого обсягу вхідних даних. Здійснено аналіз алгоритмічного забезпечення на основі багатошарового перцептрона та досліджено проблематику параметричної оптимізації нейромережевої архітектури, зокрема визначення кількості нейронів на проміжному шарі.

На основі експериментальних досліджень встановлено, що за умов обмеженого обсягу вхідних даних, коли застосування класичних методів регресійного аналізу не забезпечує достатньої точності, алгоритмічне забезпечення на основі багатошарового перцептрона демонструє високу ефективність класифікації.

Для визначення параметричних характеристик нейромережевої структури необхідно проводити серії обчислювальних експериментів з метою встановлення функціональних залежностей між показниками ефективності та параметрами нейромережі. Отримані аналітичні залежності формують методологічну основу для обґрунтування вибору параметрів нейромережевої структури при розробці алгоритмічного забезпечення для цілей прогнозування параметричних показників.

Дослідження [75] підтверджує, що алгоритмічне забезпечення на основі нейронних мереж характеризується такими перевагами:

- здатність до самонавчання та адаптації;
- можливість функціонування за умов неповноти вхідних даних;
- автоматизація процесів аналізу параметричних показників;
- висока точність прогностичних моделей.

Водночас застосування нейромережевих структур у системах моніторингу потенційно небезпечних об'єктів супроводжується певними обмеженнями [75]:

- підвищені вимоги до обчислювальних ресурсів;
- необхідність формування репрезентативних навчальних вибірок значного обсягу;
- складність структурно-параметричного синтезу нейромережевої архітектури для конкретних умов застосування.

Алгоритмічне забезпечення систем моніторингу стану ПНО може базуватися на методі найменших квадратів, який забезпечує прогнозування параметричних показників, представлених у вигляді часових рядів. Цей метод належить до класу регресійних методів математичного моделювання та дозволяє здійснювати прогностичні розрахунки на основі статистичного аналізу часових послідовностей даних.

Для прогнозування можливості НС на ПНО алгоритмічне забезпечення базується на методах регресійного аналізу. Математична модель такого процесу характеризується нелінійністю та формалізується у вигляді степеневого полінома.

При застосуванні методу найменших квадратів для визначення параметрів математичної моделі виникає проблема забезпечення постійності дисперсії

залишків для окремих спостережень чи їх груп. Внаслідок цього параметри регресійної моделі не досягають мінімальної дисперсії, що знижує точність прогностичних розрахунків.

У дослідженні [76] запропоновано алгоритмічне забезпечення для прогнозування можливості НС з урахуванням похибок регресійної моделі. Розроблене алгоритмічне забезпечення передбачає уточнення оцінок параметрів математичної моделі на основі зваженого методу найменших квадратів.

На різних рівнях регіональної системи моніторингу стану ПНО аналітична предиктивна підсистема є одним з основних елементів, які забезпечують оперативну оцінку стану ПНО, але функції аналітичної предиктивної підсистеми не обмежуються лише оцінкою стану ПНО.

Аналітична предиктивна підсистема є визначальним компонентом алгоритмічного забезпечення регіональної системи моніторингу стану ПНО, яка є об'єктом критичної інформаційної інфраструктури. Функціональні можливості цієї підсистеми диференціюються за рівнями ієрархії системи моніторингу. На об'єктовому рівні аналітична предиктивна підсистема повинна виконувати функції прогнозування та параметричного контролю параметрів що характеризують стан ПНО, а на місцевому та регіональному рівні аналітична предиктивна підсистема повинна мати можливість здійснювати моделювання стану найбільш небезпечних об'єктів.

3.3 Систематизація наукових досліджень щодо чисельних методів моделювання стану потенційно небезпечних об'єктів

Серед ПНО України та світу значну частку складають земляні напірні гідротехнічні споруди, на яких існує можливість надзвичайної ситуації з масштабними техногенними наслідками. З огляду на це, у дисертаційному дослідженні розроблено алгоритмічне забезпечення для математичного моделювання стану таких об'єктів. Зокрема, досліджено методи визначення локалізації можливих дефектів, що виникають внаслідок деградаційних процесів, спричинених впливом фільтраційних течій у тілі гідроспоруди.

У дослідженні [77] розроблено алгоритмічне забезпечення для математичного моделювання фільтраційних процесів у тілі гідроспороди. Удосконалення математичної моделі фільтраційної течії здійснено на основі системи рівнянь Дарсі з урахуванням розширеного класу граничних умов для випадку відтоку рідини через поверхню досліджуваної області.

Запропоноване алгоритмічне забезпечення надає можливість:

- здійснювати кількісний аналіз параметричних показників фільтраційних процесів;
- прогнозувати можливість НС внаслідок підвищення рівня паводкових вод;
- оцінювати стан потенційно небезпечних гідротехнічних споруд.

У дослідженні [78] розроблено алгоритмічне забезпечення для математичного моделювання фільтраційних процесів на основі інтеграції методів комплексного аналізу та чисельно-аналітичних методів сумарних зображень. Запропоновано системний підхід до моделювання фільтрації для обмежених екіпотенціальних ліній три- та чотиризв'язних криволінійних LEF-областей.

Розроблене алгоритмічне забезпечення реалізує такі функції:

- автоматична побудова динамічних сіток;
- визначення ліній відриву потоку;
- локалізація точок «підвісу» потоку;
- обчислення параметричних показників сумарного потоку.

Верифікацію алгоритмічного забезпечення здійснено у спосіб проведення обчислювального експерименту для характерного варіанта формування фільтраційного потоку в тілі гідроспороди.

У дослідженні [79] здійснено системний аналіз та узагальнення наукових результатів щодо проєктування об'єктів гідротехнічного призначення. Розроблено алгоритмічне забезпечення для математичного моделювання процесів проєктування та конструювання дамб і технологічних люків.

На основі проведених досліджень удосконалено методологію математичних розрахунків параметричних показників гідротехнічних споруд з урахуванням специфіки їх будівництва та експлуатації. Запропоновані методи

забезпечують підвищення надійності функціонування гідротехнічних споруд в умовах виробничого середовища.

У дослідженні [80] розроблено алгоритмічне забезпечення для математичного моделювання гідродинамічних процесів з вільною поверхнею, що становить окремий розділ теоретичної гідромеханіки. Актуальність розвитку цього наукового напрямку зумовлена необхідністю забезпечення надійності функціонування потенційно небезпечних гідротехнічних споруд та зниження можливості НС під час їх експлуатації.

У дослідженні [81] розроблено алгоритмічне забезпечення для математичного моделювання динаміки суцільних в'язких слабостиснених рідин на основі системи рівнянь нерозривності та Нав'є-Стокса. Запропоноване алгоритмічне забезпечення базується на методі кінцевих об'ємів для чисельного розв'язання рівняння нерозривності, що дозволяє підвищити точність прогнозування гідродинамічних процесів у тілі гідроспоруди та знизити можливість НС при експлуатації гідротехнічних споруд.

У проаналізованих дослідженнях недостатньо розглянуто питання математичного моделювання двовимірних стаціонарних фільтраційних течій методом сіток. З огляду на це, розроблено алгоритмічне забезпечення для моделювання фільтраційних процесів у тілі гідроспоруди на основі методу сіток, що забезпечує:

- підвищення точності прогнозування параметричних показників гідродинамічних процесів;
- своєчасне виявлення можливості виникнення дефектів під впливом фільтраційних течій;
- обґрунтування заходів щодо запобігання деградаційним процесам у тілі гідротехнічних споруд.

3.4 Екстраполяція значень параметрів джерел небезпеки на потенційно небезпечних об'єктах за методом найменших квадратів

Алгоритмічне забезпечення аналітичної предиктивної підсистеми систем моніторингу стану ПНО може базуватися на методі найменших квадратів,

який забезпечує прогнозування параметричних показників, представлених у вигляді часових рядів. Цей метод належить до класу регресійних методів математичного моделювання та дозволяє здійснювати прогностичні розрахунки на основі статистичного аналізу часових послідовностей даних.

Параметричні показники, що характеризують стан джерел небезпеки на потенційно небезпечному об'єкті, вимірюються у визначені моменти часу. Для розробки алгоритмічного забезпечення приймаємо, що вихідні дані залежать від одного параметра (часу), що дозволяє класифікувати такі об'єкти як нестационарні із зосередженими параметрами.

Математичне моделювання та прогнозування динаміки параметричних показників здійснюється у спосіб параметричної ідентифікації, що забезпечує:

- визначення значень параметрів, які характеризують динаміку стану об'єкта;
- оброблення експериментальних даних моніторингу;
- своєчасне виявлення можливості НС.

За умови відомої структури математичної моделі, коефіцієнти рівнянь (нелінійних, диференціальних, різницевих) та характеристик (частотних, передавальних) розглядаються як параметри моделі. У найпростішому випадку структура моделі формалізується наступним чином:

$$f(t) = kt + c \quad (3.1)$$

де:

$f(t)$ - вихідні дані моделі;

x - вхідні дані моделі;

k, c - параметри, що розраховуються.

Для знаходження параметрів математичної моделі необхідно мінімізувати функцію двох змінних:

$$f(k, c) = \sum_{i=1}^n (f_i - (kt_i + c))^2 \rightarrow \min_{k, c} \quad (3.2)$$

Умова існування екстремумів функції (3.2) визначається системою рівнянь, отриманою прирівнюванням до нуля частинних похідних за змінними k та c :

$$\begin{cases} \frac{\partial f(k,c)}{\partial k} = 0 \\ \frac{\partial f(k,c)}{\partial c} = 0 \end{cases} \quad (3.3)$$

Застосування операції диференціювання частинних похідних (3.3) дає змогу сформувати систему рівнянь (3.4):

$$\begin{cases} \sum_{i=1}^n (f_i - (kt_i + c))t_i = 0 \\ \sum_{i=1}^n (f_i - (kt_i + c)) = 0 \end{cases} \quad (3.4)$$

В результаті алгебраїчних перетворень (3.4) отримано систему рівнянь (3.5):

$$\begin{cases} k \sum_{i=1}^n t_i^2 + c \sum_{i=1}^n t_i = \sum_{i=1}^n t_i f_i \\ k \sum_{i=1}^n t_i + nc = \sum_{i=1}^n f_i \end{cases} \quad (3.5)$$

де:

- k, c - параметри, що розраховуються;
- t_i - i -й елемент масиву вхідних даних;
- f_i - i -й елемент масиву вихідних даних.

Розв'яжемо отриману систему рівнянь (3.5) методом підстановки та отримаємо формули для знаходження параметрів за методом найменших квадратів:

$$\begin{cases} k = \frac{(n \sum_{i=1}^n t_i f_i - (\sum_{i=1}^n t_i)(\sum_{i=1}^n f_i))}{(n \sum_{i=1}^n t_i^2 - (\sum_{i=1}^n t_i)^2)} \\ c = \frac{\sum_{i=1}^n f_i - k \sum_{i=1}^n t_i}{n} \end{cases} \quad (3.6)$$

Алгоритмічне забезпечення реалізується як функціональний компонент програмної системи, що отримує на вхід масив параметричних показників обсягом 10-15 вимірювань, які характеризують стан джерела техногенної небезпеки в межах ПНО. Оброблення результатів моніторингу здійснюється через формули (3.6), а визначені параметри k і c інтегруються до математичної моделі (3.1). Формування прогностичних оцінок реалізується шляхом екстраполяції часового ряду на наступний період спостереження, що не входить до масиву вхідних даних, з подальшим обчисленням прогнозованих значень параметричних показників на основі математичної моделі.

У випадку перевищення прогнозованими значеннями встановлених граничних рівнів параметричних показників, аналітична предиктивна підсистема системи моніторингу стану ПНО формує сповіщення щодо можливості надзвичайної ситуації внаслідок прогнозованого відхилення контрольованого параметра за межі допустимого діапазону.

3.5 Прогнозування динаміки параметричних показників стану джерел небезпеки в межах потенційно небезпечних об'єктів за допомогою лінійної моделі

В якості верифікації розробленого алгоритмічного забезпечення здійснено прогнозування параметричних показників рівня води у верхньому б'єфі водосховища малої гідроелектростанції, розташованої на річці Стрий (с. Довге Дрогобицького району Львівської області).

У таблиці 3.1 представлено результати параметричних вимірювань рівня води у верхньому б'єфі водосховища впродовж десяти періодів спостереження.

Нормальний підпертий рівень води у водосховищі (НПР), визначений проєктною документацією, становить 447,2 м відповідно до Балтійської системи висот.

Граничний експлуатаційний рівень води у водосховищі малої гідроелектростанції, що передує критичному стану, встановлено на позначці 449,09 м відповідно до Балтійської системи висот.

Форсований підпертий рівень води у водосховищі, що визначає максимально допустиме перевищення НПР, встановлено на позначці 449,3 м відповідно до Балтійської системи висот.

Таблиця 3.1 — Значення рівня води у верхньому б'єфі водосховища за результатами моніторингу

Відлік t_i	1	2	3	4	5	6	7	8	9	10
f_i	447,2	447,4	447,5	447,7	448,1	448,1	448,5	449,01	448,05	448,29

На основі параметричних показників, представлених у таблиці 3.1, із застосуванням (3.6), здійснено визначення коефіцієнтів математичної моделі

k та c . Проміжні результати обчислювальних процедур наведено у таблиці 3.2.

Таблиця 3.2 — Проміжні результати розрахунків коефіцієнтів моделі k та c .

t	1	2	3	4	5	6	7	8	9	10	55
t^2	1	4	9	16	25	36	49	64	81	100	385
f_i	447,2	447,4	447,5	447,7	448,1	448,1	448,5	449,01	448,05	448,29	4479,85
$f_i t_i$	447,2	894,8	1342,5	1790,8	2240,5	2688,6	3139,5	3592,08	4032,45	4482,9	24651,33

Визначені відповідно до (3.6) параметри математичної моделі становлять:
 $k = 0,1473$ та $c = 447,175$.

Математична модель формалізується наступним чином:
 $f_i = 0,1473t_i + 447,175$. На основі розробленої моделі здійснено обчислення параметричних показників для десяти періодів спостереження та прогнозованого значення для наступного періоду.

Результати прогностичних розрахунків представлено у таблиці 3.3.

Таблиця 3.3 — Результати розрахунків при прогнозуванні рівня води у водосховищі МГЕС

Результати моніторингу рівня води											Прогноз			
Відлік часу	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Вимір, факт, м	447.200	447.400	447.500	447.700	448.100	448.100	448.500	449.010	448.050	448.290	448.200	448.030	448.020	448.001
Розрахунок згідно моделі, м	447.322	447.469	447.616	447.763	447.91	448.057	448.204	448.351	448.498	448.645	448.792	448.939	449.086	449.233
Похибка, %	0.027	0.015	0.026	0.014	0.042	0.010	0.066	0.147	0.100	0.079	0.132	0.202	0.237	0.274

Розроблене алгоритмічне забезпечення надає можливість здійснювати екстраполяцію параметричних показників на довільну кількість періодів спостереження, проте достовірність прогностичних оцінок знижується зі збільшенням часового горизонту прогнозування.

Точність прогнозування можливості НС на ПНО є визначальним фактором для своєчасного впровадження превентивних протиаварійних заходів.

Проведений аналіз наукових досліджень у сфері прогнозування НС свідчить про превалювання методів регресійного аналізу, що базуються на даних моніторингу [82, 83, 84, 85, 86, 87, 88]. Алгоритмічне забезпечення лінійних регресійних моделей ґрунтується на застосуванні методу найменших квадратів [88, 89]. Для нелінійних регресійних моделей розроблено процедуру

лінеаризації з подальшим визначенням параметрів за допомогою методу найменших квадратів.

Застосування методу найменших квадратів для отримання незміщених та ефективних оцінок параметрів регресійної моделі потребує виконання умови сталості дисперсії залишків для всіх спостережень. Регресійні моделі процесів виникнення НС характеризуються нелінійною природою, що призводить до порушення умови сталості дисперсії залишків для всіх спостережень. Внаслідок цього, оцінки параметрів регресії, отримані методом найменших квадратів, не забезпечують мінімальну дисперсію, що знижує точність прогнозування. Зазначене обумовлює необхідність розроблення модифікованого алгоритмічного забезпечення.

Алгоритмічне забезпечення апроксимації масивів експериментальних даних ґрунтується на їх представленні за допомогою аналітичних функцій визначеного класу (лінійних, квадратичних, кубічних тощо). Такі функції не обов'язково інтерполюють експериментальні точки, проте забезпечують відтворення тенденцій зміни досліджуваних даних при мінімізації суми квадратів відхилень між експериментальними та розрахунковими значеннями.

Аналітичні залежності, що апроксимують характеристики досліджуваних процесів, мають забезпечувати належний рівень достовірності відтворення їх властивостей. Водночас підвищення ступеня відповідності апроксимаційних залежностей експериментальним даним зумовлює зростання порядку апроксимуючих функцій, що суттєво впливає на процедуру визначення їх параметрів та подальший аналіз досліджуваних процесів.

3.6 Математичне забезпечення екстраполяції часових рядів на основі степеневі поліноміальної апроксимаційної моделі

Забезпечення вищого ступеня відповідності апроксимаційної моделі експериментальним даним досягнуто у спосіб застосування поліноміальних функцій, що обумовлено їх властивістю рівномірного наближення неперервних функцій згідно з теоремою Вейєрштрасса.

Для заданої множини експериментальних точок $\{(x_0, y_0), (x_1, y_1) \dots (x_n, y_n)\}$ розв'язано задачу визначення аналітичної

залежності $F(x)$, що апроксимує задану множину точок за визначеним критерієм оптимальності. Як критерій оптимальності використано мінімізацію суми квадратів відхилень експериментальних значень від значень апроксимувальної функції $F(x)$:

$$\sum_{i=0}^n (F(x_i) - y_i)^2 \rightarrow \min \quad (3.7)$$

За визначеним критерієм оптимальності задача апроксимації характеризується множиною можливих розв'язків. Єдиність розв'язку забезпечується у спосіб визначення класу апроксимуючої функції $F(x)$. Аналітичне представлення функції $F(x)$ може належати до таких класів:

- степеневий поліном;
- тригонометричний поліном;
- ортогональний поліном;
- сплайн-функція.

Розроблене алгоритмічне забезпечення апроксимації експериментальних даних ґрунтується на застосуванні степеневого полінома, математичне представлення якого задається виразом:

$$F(x) = k_0x^0 + k_1x^1 + \dots + k_nx^n \quad (3.8)$$

Критерій мінімізації середньоквадратичного відхилення, визначений у виразі 3.7, для степеневого полінома набуває вигляду:

$$\sum_{i=0}^n (F(x_i) - y_i)^2 = \sum_{i=0}^n (k_0x_i^0 + k_1x_i^1 + \dots + k_nx_i^n - y_i)^2 \rightarrow \min \quad (3.9)$$

Відхилення між експериментальними значеннями y_i та значеннями апроксимуючої функції $F(x_i)$ у дискретних точках $x_i, i = 0, 1, 2 \dots n$, визначаються як залишки апроксимації та позначаються:

$$\Delta_0(x) = F(x_0) - y_0, \Delta_1(x) = F(x_1) - y_1 \dots \Delta_n(x) = F(x_n) - y_n. \quad (3.10)$$

З урахуванням введеного означення залишків, критерій оптимальності 3.9 набуває вигляду:

$$\sum_{i=0}^n \Delta_i^2(x) \rightarrow \min \quad (3.11)$$

Функція $\Delta(x)$ є багатопараметричною функцією, визначеною на множині коефіцієнтів $k_i, i = 0 \dots n$. Необхідні умови існування мінімуму цієї функції визначаються системою рівнянь:

$$\begin{cases} \frac{\partial \Delta(x)}{\partial k_0} = 0 \\ \frac{\partial \Delta(x)}{\partial k_1} = 0 \\ \vdots \\ \frac{\partial \Delta(x)}{\partial k_n} = 0 \end{cases} \quad (3.12)$$

Виконаємо послідовність математичних перетворень:

1. Замінімо функцію $\Delta(x)$ у виразі 3.12 на різницю $F(x_i) - y_i$;
2. Підставимо вираз полінома з 3.8 замість $F(x)$;
3. Обчислимо частинні похідні за коефіцієнтами k_i .

У результаті отримаємо систему лінійних алгебраїчних рівнянь:

$$\begin{cases} \frac{\partial \Delta(x)}{\partial k_0} = 2 \sum_{i=0}^n (k_0 + k_1 x_i + \dots + k_n x_i^n - y_i) = 0; \\ \frac{\partial \Delta(x)}{\partial k_1} = 2 \sum_{i=0}^n (k_0 + k_1 x_i + \dots + k_n x_i^n - y_i) x_i = 0; \\ \vdots \\ \frac{\partial \Delta(x)}{\partial k_n} = 2 \sum_{i=0}^n (k_0 + k_1 x_i + \dots + k_n x_i^n - y_i) x_i^n = 0. \end{cases} \quad (3.13)$$

Після розкриття дужок та алгебраїчних перетворень кожного рівняння система набуває канонічного вигляду:

$$\begin{cases} n k_0 + k_1 \sum_{i=0}^n x_i + \dots + k_m \sum_{i=0}^n x_i^n = \sum_{i=0}^n y_i; \\ k_0 \sum_{i=0}^n x_i + k_1 \sum_{i=0}^n x_i^2 + \dots + k_m \sum_{i=0}^n x_i^{n+1} = \sum_{i=0}^n x_i y_i; \\ \vdots \\ k_0 \sum_{i=0}^n x_i^m + k_1 \sum_{i=0}^n x_i^{m+1} + \dots + k_m \sum_{i=0}^n x_i^{2m} = \sum_{i=0}^n x_i^m y_i. \end{cases} \quad (3.14)$$

Отримана система лінійних алгебраїчних рівнянь містить невідомі коефіцієнти полінома $k_i, i = 0, \dots, m$, визначення яких забезпечує побудову аналітичної залежності для апроксимації експериментальних даних.

Система лінійних алгебраїчних рівнянь може бути представлена у матричній формі:

$$\begin{pmatrix} n & \sum_{i=0}^n x_i & \cdots & \sum_{i=0}^n x_i^m \\ \sum_{i=0}^n x_i & \sum_{i=0}^n x_i^2 & \cdots & \sum_{i=0}^n x_i^{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^n x_i^m & \sum_{i=0}^n x_i^{m+1} & \cdots & \sum_{i=0}^n x_i^{2m} \end{pmatrix} \times \begin{pmatrix} k_0 \\ k_1 \\ \vdots \\ m \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^n y_i \\ \sum_{i=0}^n x_i y_i \\ \vdots \\ \sum_{i=0}^n x_i^m y_i \end{pmatrix} \quad (3.15)$$

Алгоритмічне забезпечення розв'язання системи лінійних алгебраїчних рівнянь ґрунтується на обробці табличних експериментальних даних у спосіб застосування методів обчислювальної математики для визначення коефіцієнтів апроксимуючого полінома.

На основі запропонованого алгоритмічного забезпечення розроблено програмне забезпечення для обчислення коефіцієнтів апроксимуючого полінома, що наведено у Д.

За результатами моніторингу було виконано прогнозування згідно трьох моделей:

- лінійної;
- степеневого полінома другого ступеня;
- степеневого полінома третього ступеня.

Результати прогнозування наведено у таблиці 3.4.

Таблиця 3.4 — Порівняння результатів прогнозування за різними моделями

Результати моніторингу											Прогноз			
Відлік часу	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Вимір, факт, м	447.200	447.400	447.500	447.700	448.100	448.100	448.500	449.010	448.050	448.290	448.200	448.030	448.020	448.001
Лінійна модель, м	447.322	447.469	447.616	447.763	447.91	448.057	448.204	448.351	448.498	448.645	448.792	448.939	449.086	449.233
Похибка, %	0.027	0.015	0.026	0.014	0.042	0.010	0.066	0.147	0.100	0.079	0.132	0.202	0.237	0.274
Поліном 2 ступеня	447.046	447.377	447.663	447.902	448.095	448.243	448.344	448.399	448.409	448.372	448.290	448.161	447.766	447.499
Похибка, %	0.034	0.005	0.036	0.045	0.001	0.032	0.035	0.136	0.080	0.018	0.020	0.029	0.057	0.112
Поліном 3 ступеня	447.247	447.311	447.496	447.754	448.038	448.300	448.492	448.566	448.476	448.172	447.607	446.733	445.503	443.869
Похибка, %	0.011	0.020	0.001	0.012	0.014	0.045	0.002	0.099	0.095	0.026	0.132	0.290	0.565	0.931

Відповідно до досліджень [90], достовірність апроксимації експериментальних даних степеневим поліномом забезпечується за умови,

що його порядок m задовольняє співвідношення $m \leq n/5$, де n – кількість експериментальних точок.

Аналіз результатів, наведених у таблиці 3.4, демонструє, що найвищу точність прогнозування забезпечує степеневий поліном другого ступеня ($m = 2$), що узгоджується з теоретичними положеннями, викладеними у [90].

Алгоритмічне забезпечення прогнозування можливості НС на ПНО реалізується у спосіб послідовного виконання таких етапів:

1. Визначення коефіцієнтів апроксимуючого степеневого полінома на основі експериментальних даних із застосуванням методу найменших квадратів.
2. Обчислення прогнозованих значень параметрів, що характеризують стан потенційного джерела небезпеки на заданий момент часу.
3. Оцінювання ймовірності виникнення НС та формування рекомендацій щодо впровадження превентивних протиаварійних заходів і процедур оповіщення персоналу.

3.7 Програмне забезпечення прогнозування часових рядів на основі поліноміальної апроксимації методом найменших квадратів

Для реалізації системи прогнозування рівня води розроблено програмне забезпечення на мові Rust з використанням архітектури графічного інтерфейсу на базі бібліотеки програмного забезпечення egui. Система реалізує метод поліноміальної регресії з використанням ортогональних поліномів для апроксимації та прогнозування часових рядів даних. Архітектура програмного забезпечення побудована за модульним принципом, що забезпечує чітке розділення бізнес-логіки та презентаційного шару, а також високу тестованість компонентів.

Структура проєкту складається з основних модулів: бібліотеки програмного забезпечення ls.rs, що реалізує математичний апарат найменших квадратів та обробку даних, та виконуваного модуля gui.rs, який забезпечує інтерактивний користувацький інтерфейс. Додатково реалізовано модуль тестування ls.rs, що містить набір unit-тестів для верифікації коректності математичних обчислень. Управління залежностями та конфігурація проєкту

здійснюється через файли Cargo.toml та Cargo.lock, що є стандартними компонентами екосистеми Rust.

Архітектура програмного забезпечення базується на взаємодії двох основних компонентів: класу LS, що реалізує математичні обчислення методом найменших квадратів, та класу PredictionApp, що забезпечує графічний інтерфейс користувача. Клас LS інкапсулює логіку обробки даних та прогнозування, зберігаючи проміжні результати обчислень у структурах EncodingStep для можливості покрокового аналізу. PredictionApp взаємодіє з бібліотекою egui для створення інтерактивного інтерфейсу, візуалізації даних та результатів прогнозування, а також керує життєвим циклом об'єкта LS та обробкою користувацького введення. На рисунку 3.1 наведено діаграму класів для програмного забезпечення прогнозування часових рядів.

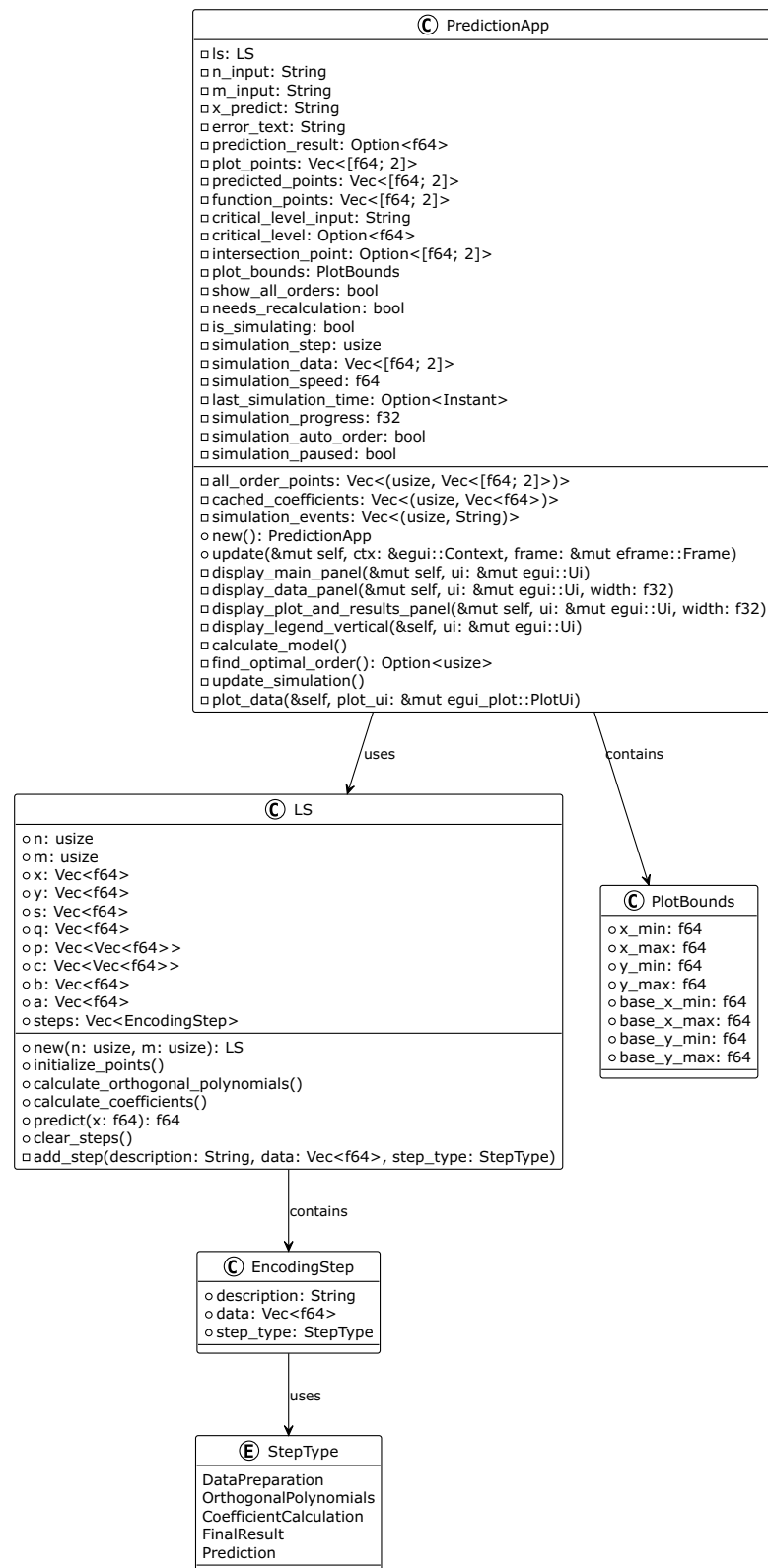


Рисунок 3.1 — Діаграма класів програмного забезпечення для прогнозування часових рядів

Структура LS реалізує математичний апарат методу найменших квадратів та є ключовим компонентом системи прогнозування. Структура інкапсулює

вхідні дані у вигляді векторів значень x та y , а також зберігає коефіцієнти та проміжні результати обчислень. Для забезпечення повного циклу обробки даних та прогнозування структура реалізує набір методів, що охоплюють ініціалізацію, обчислення та прогнозування значень. Детальний опис методів структури LS наведено у таблиці 3.5.

Таблиця 3.5 — Основні методи структури LS

Метод	Призначення
<code>new()</code>	Створення нового екземпляру структури з заданими параметрами
<code>initialize_points()</code>	Ініціалізація тестового набору даних для верифікації роботи системи
<code>calculate_orthogonal_polynomials()</code>	Обчислення системи ортогональних поліномів для апроксимації
<code>calculate_coefficients()</code>	Обчислення коефіцієнтів апроксимуючого полінома
<code>predict()</code>	Прогнозування значень на основі побудованої моделі

Структура `PredictionApp` є центральним компонентом системи, що реалізує графічний інтерфейс користувача та керування програмою. Структура зберігає поточний стан програми, включаючи набір точок даних (`points`), ступінь полінома (`degree`), кількість точок прогнозування (`prediction_points`) та екземпляр структури LS для обчислень. Для забезпечення інтерактивної взаємодії з користувачем та візуалізації даних структура реалізує набір методів, детальний опис яких наведено у таблиці 3.6.

Таблиця 3.6 — Основні методи структури `PredictionApp`

Метод	Призначення
<code>new()</code>	Створення нового екземпляру програми з початковими налаштуваннями
<code>update()</code>	Оновлення стану програми та обробка подій інтерфейсу користувача
<code>draw_chart()</code>	Відображення графіків даних та результатів прогнозування
<code>load_data()</code>	Завантаження вхідних даних для аналізу

Для забезпечення повноцінного функціонування системи прогнозування реалізовано набір допоміжних структур даних, що розширюють базову функціональність основних компонентів. Структура `EncodingStep` забезпечує збереження та документування покрокового процесу обчислень, включаючи проміжні результати та описи етапів розрахунків. Для типізації етапів обчислень використовується еnumерація `StepType`, що визначає можливі типи кроків:

підготовка даних (DataPreparation), обчислення ортогональних поліномів (OrthogonalPolynomials), розрахунок коефіцієнтів (CoefficientCalculation), отримання фінального результату (FinalResult) та прогнозування (Prediction). Структура PlotBounds інкапсулює параметри області візуалізації, включаючи мінімальні та максимальні значення по осях координат, що забезпечує коректне масштабування та показ графіків.

Взаємодія компонентів системи організована за принципом чіткого розділення відповідальності, де кожен компонент виконує специфічний набір функцій. Центральним елементом є структура PredictionApp, що керує життєвим циклом програми та взаємодією з користувачем через графічний інтерфейс, реалізований за допомогою бібліотеки egui. PredictionApp делегує математичні обчислення структурі LS, яка реалізує алгоритм найменших квадратів та використовує структуру EncodingStep для збереження історії обчислень. Така архітектура забезпечує модульність системи, де кожен компонент може бути модифікований або замінений без впливу на інші частини програми, що відповідає принципам об'єктноорієнтованого проєктування та полегшує подальшу підтримку та розширення функціональності системи.

Модуль ls.rs реалізує математичний апарат методу найменших квадратів через структуру LeastSquares, що інкапсулює алгоритми поліноміальної регресії та обробки часових рядів. Структура забезпечує методи для обчислення коефіцієнтів апроксимувального полінома, оцінки похибки апроксимації та прогнозування майбутніх значень часового ряду. Імплементація базується на використанні ортогональних поліномів, що забезпечує числову стабільність обчислень та мінімізує накопичення похибок округлення при роботі з даними великої розмірності.

Модуль gui.rs реалізує презентаційний шар системи з використанням бібліотеки egui, що забезпечує створення інтерактивного графічного інтерфейсу користувача. Архітектура графічного модуля базується на концепції негайного режиму відображення (immediate mode GUI), що спрощує розробку та підтримку інтерфейсу. Модуль реалізує компоненти візуалізації часових рядів, елементи управління параметрами апроксимації та інтерактивні графіки для відображення результатів прогнозування.

Система реалізує обробку вхідних даних через функції модуля `ls.rs`, що забезпечують роботу з часовими рядами. Вхідні дані представлені у вигляді векторів типу `Vec<f64>`, що містять значення рівня води за певні проміжки часу. Реалізовано базові операції для роботи з даними: зчитування значень, обчислення статистичних характеристик та підготовка даних для подальшої апроксимації. Архітектура системи передбачає можливість розширення функціональності через додавання нових методів обробки даних у модуль `ls.rs`.

Верифікація програмного забезпечення реалізована через систему модульного тестування в модулі `ls.rs`. Тестове забезпечення включає базові `unit`-тести для перевірки коректності обчислень методу найменших квадратів. Реалізовано тестові функції `test_ls()` та `test_ls_2()`, що валідують точність обчислення коефіцієнтів апроксимації на тестових наборах даних. Процес тестування виконується через вбудований в Rust інструментарій `cargo test`, що дозволяє автоматично запускати тести при внесенні змін у код. Тести анотовані атрибутом `# [test]` згідно зі стандартними практиками розробки на Rust.

У системі реалізовано механізм обробки помилок з використанням стандартного для Rust типу `Result` та вбудованих типів помилок `std::error::Error`. Архітектура обробки помилок забезпечує коректну обробку всіх критичних ситуацій, включаючи помилки при читанні вхідних даних, некоректні параметри апроксимації та числову нестабільність при обчисленнях. Реалізовано детальне логування помилок з використанням макросів `println!` для інформування користувача про причини виникнення проблем. Система забезпечує коректне завершення роботи у випадку критичних помилок та надає користувачу інформативні повідомлення про причини збоїв.

Архітектура системи забезпечує роботу з даними через стандартні операції файлового введення-виведення. Вхідні дані зчитуються з текстового файлу за допомогою функцій стандартної бібліотеки Rust `std::fs::File` та `std::io::BufReader`. Реалізовано базовий механізм парсингу текстових даних для отримання числових значень часового ряду. Результати прогнозування можуть бути виведені на екран через графічний інтерфейс `egui` для подальшого аналізу

користувачем. Система використовує простий текстовий формат для вхідних даних, що забезпечує сумісність з іншими інструментами аналізу даних.

Система реалізує базові оптимізації для забезпечення ефективної обробки даних. Використання типу `f64` для обчислень з плаваючою комою забезпечує необхідну точність при розрахунку коефіцієнтів регресії. Архітектура системи базується на ефективних структурах даних Rust, зокрема `Vec<f64>` для зберігання та обробки числових послідовностей. Операції з векторами виконуються з використанням ітераторів, що є оптимізованим механізмом обробки послідовностей у Rust. Додатково система використовує стек для локальних обчислень, що мінімізує накладні витрати на управління пам'яттю.

Візуальний інтерфейс надає миттєвий зворотний зв'язок при зміні параметрів, відображаючи оновлений графік апроксимації та прогнозу наведено на рисунку 3.2

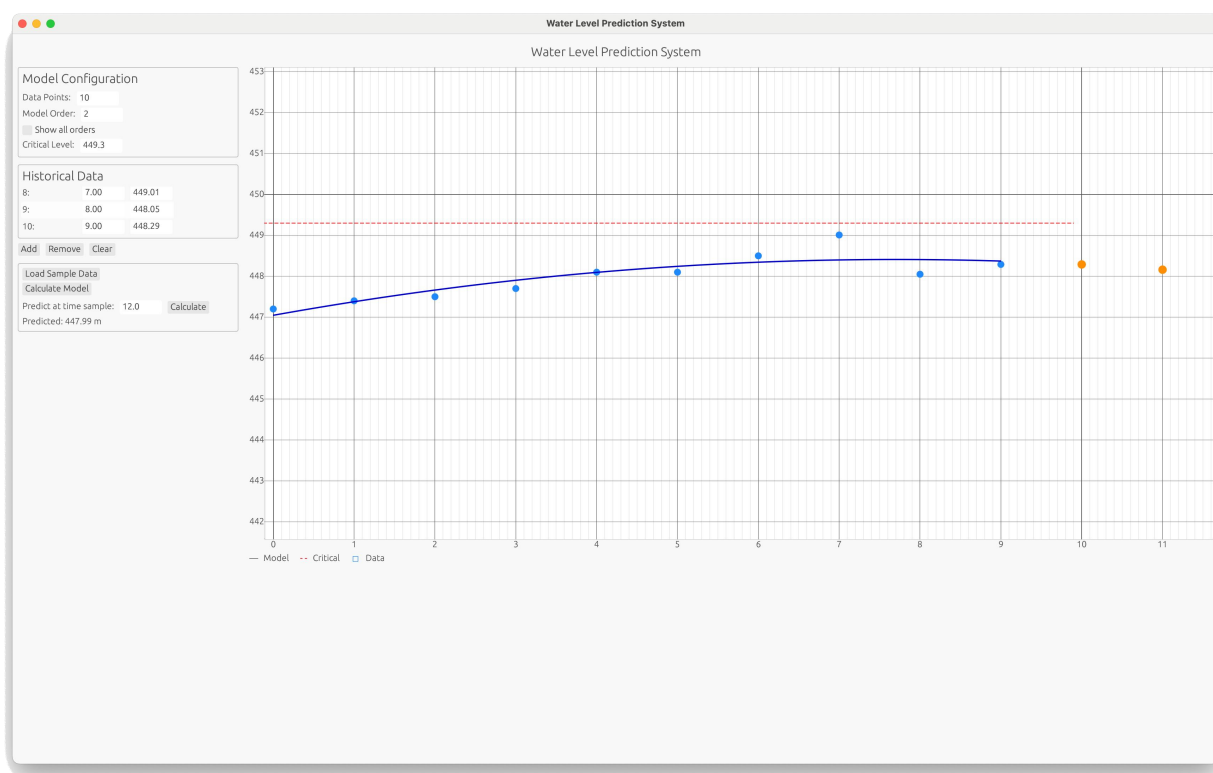


Рисунок 3.2 — Інтерфейс користувача додатку прогнозування часових рядів на основі поліноміальної апроксимації

Архітектура програмного забезпечення передбачає налаштування параметрів через графічний інтерфейс користувача, реалізований за допомогою

бібліотеки `egui`. Користувач може інтерактивно змінювати ступінь полінома для апроксимації та кількість точок прогнозування безпосередньо під час роботи програми. Система забезпечує валідацію введених параметрів для запобігання некоректним налаштуванням, зокрема перевіряється допустимий діапазон значень для ступеня полінома.

Програмне забезпечення реалізує ефективні методи поліноміальної апроксимації та забезпечує високу точність прогнозування завдяки використанню ортогональних поліномів та методу найменших квадратів. Модульна архітектура системи, реалізована мовою Rust, гарантує надійність роботи та можливість подальшого розширення функціональності. Висновки щодо розробленого програмного забезпечення для екстраполяції часових рядів наведені в кінці третього розділу.

3.8 Обґрунтування необхідності розроблення алгоритмічного та програмного забезпечення для моделювання стану потенційно небезпечних об'єктів на прикладі розрахунку фільтраційних процесів у земляних гідроспорудах

Земляні гідроспоруди – це один з найбільш поширених типів ПНО, на яких існує реальна загроза виникнення НС техногенного характеру. За статистичними даними, земляні греблі становлять близько 85% від загальної кількості проєктованих і побудованих гребель у світі. В Україні частка таких гідроспоруд сягає 90% від усіх наявних гребель.

Виникнення пошкоджень гідротехнічних споруд зумовлюється комплексом чинників: впливом природних факторів, втратою конструктивної стійкості, наявністю дефектів та порушенням регламенту експлуатації. Для земляних гребель визначальним фактором технічного стану є процес фільтрації води крізь тіло споруди. За результатами досліджень, надмірна фільтрація спричиняє понад 60% зафіксованих аварійних ситуацій на земляних греблях.

Результати досліджень земляних гідроспоруд засвідчують розвиток деструктивних явищ під час їх експлуатації: карстово-суфозійних деформацій та формування зосереджених шляхів фільтрації в тілі гребель, дамб та їхніх основах. Такі явища спричиняють розмиви, деформації укосів та зсуви, що

створюють можливість НС через руйнування споруди. Характер поверхні депресії є індикатором стану фільтраційного режиму та локальних порушень водопроникності в тілі земляної греблі.

Руйнування напірних земляних гідроспоруд створює можливість НС техногенного характеру через стрімкий розвиток деструктивних процесів: неконтрольоване вивільнення водних мас, руйнування житлової та критичної інфраструктури, затоплення прилеглих територій. Гідродинамічні аварії спричиняють комплексний негативний вплив на соціально-економічну та екологічну складові регіону: загрозу життю та здоров'ю населення, порушення екологічної рівноваги природних систем, суттєві економічні збитки.

Відповідно до законодавства України, гідротехнічні споруди, класифіковані як ПНО, мають бути оснащені системами моніторингу технічного стану для своєчасного виявлення можливості НС. Моніторинг реалізується як комплексна система спостережень, аналізу та прогнозування, що забезпечує визначення стану джерел небезпеки в межах ПНО за значеннями характеристичних параметрів.

Дослідження динаміки деструктивних процесів, спричинених фільтраційними течіями в земляних греблях, потребує розроблення спеціалізованого математичного апарату у спосіб формалізації фізичних процесів.

Застосування стандартизованих нормативних методик розрахунку обмежене специфікою досліджуваних процесів. Це зумовлює необхідність розроблення спеціалізованих математичних моделей для прогнозування часової динаміки рівнів ґрунтових вод у тілі греблі. Реалізація чисельної моделі та відповідного програмного забезпечення забезпечує визначення динамічних характеристик ґрунтових вод та інтенсивності фільтраційних течій. Такий підхід уможливорює ідентифікацію зон потенційного формування фільтраційних потоків, що створюють можливість НС спричиненою деградацією земляної напірної гідроспоруди.

Алгоритмічне забезпечення моделювання стану напірних гідроспоруд доцільно реалізувати на обчислювальних засобах систем моніторингу стану ПНО регіонального та місцевого рівнів. Вхідними даними для математичного

моделювання напірних земляних гідроспоруд слугують результати вимірювань, отримані від систем технічного моніторингу ПНО на об'єктовому рівні.

Для розв'язання практичних задач моделювання фільтраційних процесів у земляних напірних гідроспорудах застосовано чисельний метод розв'язування рівнянь у частинних похідних. Такий підхід забезпечує врахування складної структури течій, конфігурації меж області фільтрації та особливостей рельєфу місцевості. Розроблено спеціалізоване алгоритмічне забезпечення з теоретичним обґрунтуванням точності та збіжності обчислювальних процедур, що реалізують запропонований метод.

Актуальність дослідження зумовлена зростанням кількості НС в умовах військових дій та глобальних викликів, що потребують розроблення нових методологічних підходів до управління ризиками та мінімізації їхніх наслідків. Напірні гідротехнічні споруди створюють можливість НС техногенного характеру через потенційні аварії або повені. Розроблення математичних моделей забезпечує комплексне оцінювання технічного стану споруд та прогнозування розвитку деструктивних процесів. Такий підхід уможливорює аналіз сценаріїв розвитку ситуації та ідентифікацію зон потенційних пошкоджень, що є основою для формування ефективних стратегій запобігання надзвичайним ситуаціям.

3.9 Математичне та алгоритмічне моделювання фільтраційних процесів у земляних гідроспорудах

Чисельне моделювання двовимірних стаціонарних фільтраційних течій у гідротехнічних системах реалізовано із застосуванням методу сіток. Алгоритмічне забезпечення передбачає такі етапи: апроксимацію розрахункової області структурованою сіткою з визначеним кордоном, дискретизацію області з фіксованим кроком сітки для формування різницевих аналогів диференціальних рівнянь. У процесі дискретизації застосовано п'ятиточковий шаблон для апроксимації похідних другого порядку.

Алгоритмічне забезпечення реалізує обчислення значень потенціальної функції для внутрішніх вузлів сітки у спосіб визначення середнього арифметичного значення функції в суміжних вузлах з кроком h . Запропоновано

структуру додаткової сітки, вузли якої відповідають внутрішнім та зовнішнім вузлам основної сітки. Значення у вузлах додаткової сітки визначають множину арифметичних операцій для відповідних вузлів основної сітки.

Застосування додаткової сітки забезпечує:

1. Формалізацію обчислювальних операцій через уніфіковані оператори циклу
2. Модифікацію конфігурації границь сітки без зміни базового алгоритму
3. Гнучке керування послідовністю обчислень у вузлах

Математичний апарат та структура алгоритмічного забезпечення уможлиблюють ефективне розпаралелювання обчислень у багатопроцесорних системах.

Математичний апарат дисертаційного дослідження ґрунтується на чисельних методах розв'язання рівнянь Лапласа та споріднених диференціальних рівнянь. Зазначений метод забезпечує визначену точність моделювання стаціонарних фільтраційних течій у гідротехнічних системах з похибкою, що не перевищує встановлені граничні значення. Верифікацію результатів виконано через порівняння з аналітичними розв'язками для тестових задач.

Розроблення нових математичних моделей процесу фільтрації та вдосконалення алгоритмічного забезпечення математичного моделювання є визначальним для забезпечення надійності функціонування гідротехнічних споруд. Математичне моделювання надає можливість:

- досліджувати динаміку фільтраційних процесів;
- прогнозувати розвиток фільтраційних процесів;
- локалізувати дефекти які можуть виникнути;
- обґрунтовувати управлінські рішення у разі можливості НС.

Запропоновані математичні моделі та алгоритмічне забезпечення становлять основу для створення систем моніторингу стану гідротехнічних споруд із підвищеною точністю та результативністю. Це забезпечує своєчасне реагування на зміни параметрів фільтраційних процесів та прийняття обґрунтованих рішень в умовах експлуатації гідротехнічних споруд.

Алгоритмічне забезпечення аналізу фільтраційних течій у гідротехнічних спорудах ґрунтується на застосуванні ефективних методів чисельного аналізу, що забезпечують визначену точність розв'язання систем диференціальних рівнянь.

У галузі інженерії програмного забезпечення для розв'язання диференціальних рівнянь застосовують множину чисельних методів, серед яких виділяють: метод сіток, метод скінченних елементів, метод мажорантних областей, метод суматорних подань та метод фіктивних областей. Метод сіток забезпечує апроксимацію диференціальних рівнянь через дискретизацію просторової області, що уможливорює отримання чисельного розв'язку задачі з визначеною точністю. Метод скінченних елементів ґрунтується на декомпозиції області дослідження на скінченну множину елементів із подальшим розв'язанням диференціальних рівнянь для кожного елемента. Застосування кожного з наведених методів потребує врахування їхніх характеристик відповідно до специфіки задачі [91].

Створення нових та вдосконалення наявних методів чисельного аналізу фільтраційних течій становить актуальну наукову задачу в галузі інженерії програмного забезпечення. Розроблення відповідного алгоритмічного забезпечення надає можливість підвищити обчислювальну точність, зменшити часову складність розв'язання крайових задач та забезпечити формування множини альтернатив для систем підтримки прийняття рішень у процесі експлуатації гідротехнічних споруд.

Математичне моделювання фільтраційних течій ґрунтується на законі Дарсі, який встановлює функціональну залежність між швидкістю фільтрації та градієнтом гідродинамічного тиску в пористому середовищі [92]. Застосування цього закону в алгоритмічному забезпеченні систем моделювання надає можливість визначити кількісні характеристики фільтраційного процесу: швидкість просочення рідини, коефіцієнти проникності середовища та параметри потоку. У контексті розв'язання крайових задач гідродинаміки та гідрогеології закон набуває вигляду:

$$v = kJ = -\frac{k\partial H}{\partial S} \quad (3.16)$$

де

k – коефіцієнт фільтрації;

J – градієнт п'єзометричного напору за шляхом S .

З урахуванням залежності градієнта п'єзометричного напору:

$$J = -\frac{\partial H}{\partial S} \quad (3.17)$$

$$H = \frac{P}{\rho g} + z(\rho g = \gamma) \quad (3.18)$$

де

P – гідростатичний тиск, діючий на рідину;

ρ – щільність;

γ – питома вага;

g – гравітаційне прискорення;

z – висота відносно початкової площини.

Відповідно до закону Дарсі (формула 3.16), величина втрат гідродинамічного напору під час фільтрації через пористе середовище характеризується лінійною залежністю від швидкості руху рідини. Встановлена функціональна залежність надає можливість виконувати математичне моделювання процесу фільтрації з урахуванням коефіцієнта пропорційності, що визначається фізичними властивостями середовища та характеристиками потоку.

Коефіцієнт фільтрації k характеризує інтенсивність фільтраційного процесу та визначається як функція фізичних властивостей пористого середовища та гідродинамічних характеристик фільтраційного потоку:

$$k = \frac{k_f \rho g}{\mu} \quad (3.19)$$

де

k_f – фільтраційна проникність середовища;

ρ – масова густина рідини;

μ – параметр динамічної в'язкості;

g – гравітаційне прискорення.

Фізичні властивості пористих середовищ та гідродинамічні характеристики рідин визначають множину значень коефіцієнта фільтрації. Алгоритмічне забезпечення математичного моделювання використовує експериментально визначені значення коефіцієнта фільтрації, що враховують структурні характеристики середовища, глибину проникнення рідини та градієнт тиску. Систематизацію експериментальних даних щодо коефіцієнта фільтрації для різних типів середовищ, наведену в дослідженні [93], подано в таблиці 3.7.

Таблиця 3.7 — Типові значення коефіцієнту фільтрації для різних ґрунтів

№ п/п	Вид ґрунту	Коефіцієнт фільтрації	Одиниця виміру
1	Гравій	$10^{-2} - 10^{-1}$	см/с
2	Пісок крупний	$10^{-2} - 10^{-1}$	см/с
3	Пісок дрібний	$10^{-3} - 10^{-2}$	см/с
4	Супісок	$10^{-5} - 10^{-4}$	см/с
5	Суглинок	$10^{-7} - 10^{-6}$	см/с

Математичне моделювання фільтрації в анізотропному середовищі потребує врахування просторової неоднорідності фізичних властивостей пористої структури. За умов анізотропії зберігається лінійна залежність між швидкістю фільтрації та градієнтом тиску, проте виникає порушення колінеарності векторів градієнта тиску та швидкості фільтрації. Зазначена особливість зумовлює необхідність модифікації алгоритмічного забезпечення для розрахунку траєкторій руху фільтраційного потоку з урахуванням тензорного характеру коефіцієнта фільтрації в анізотропному середовищі.

Формалізація математичного опису процесів гідромеханічної фільтрації ґрунтується на системі диференціальних рівнянь, що описують динаміку руху рідини в пористих середовищах [94]. Ця система рівнянь становить основу алгоритмічного забезпечення для моделювання фільтраційних процесів.

Запропонована система рівнянь надає можливість математичного моделювання фільтраційних процесів із урахуванням характеристик середовища та граничних умов.

$$\frac{1}{g} \frac{\partial \vartheta_x}{\partial t} + \frac{\partial H}{\partial x} + \frac{1}{k} \vartheta_x = 0 \quad (3.20)$$

$$\frac{1}{g} \frac{\partial \vartheta_y}{\partial t} + \frac{\partial H}{\partial y} + \frac{1}{k} \vartheta_y = 0 \quad (3.21)$$

$$\frac{1}{g} \frac{\partial \vartheta_z}{\partial t} + \frac{\partial H}{\partial z} + \frac{1}{k} \vartheta_z = 0 \quad (3.22)$$

$$n \frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho v_x) + \frac{\partial}{\partial y}(\rho v_y) + \frac{\partial}{\partial z}(\rho v_z) = 0 \quad (3.23)$$

$$f(\rho, H, T) = 0 \quad (3.24)$$

Наведена система замкнутих рівнянь описує нестационарні процеси фільтрації рідини в недеформованому ґрунті за умови виконання закону Дарсі. Область застосування математичної моделі обмежується ламінарним режимом фільтрації, за якого зберігається лінійна залежність між швидкістю фільтрації та градієнтом напору.

Рівняння нерозривності 3.23 враховує пористість ґрунту (n) для визначення об'ємної частки пор у процесі фільтрації. Рівняння стану 3.24 встановлює функціональну залежність між густиною рідини (ρ), ізотермічним напором (H) та абсолютною температурою (T).

Зазначені рівняння надають можливість математичного опису стану рідини та взаємозв'язків параметрів фільтраційних процесів.

Інтегрування системи рівнянь (3.20) – (3.24) з урахуванням крайових та початкових умов у заданій області фільтрації забезпечує визначення таких характеристик: векторного поля швидкості фільтрації $\vec{\vartheta}(\vartheta_x, \vartheta_y, \vartheta_z)$, розподілу п'єзометричного напору $H(x, y, z, t)$ та поля густини рідини $\rho(x, y, z, t)$.

Отримані розв'язки формують основу алгоритмічного забезпечення для моделювання динаміки фільтраційного потоку та визначення параметрів масоперенесення в пористому середовищі.

За умови моделювання фільтрації нестисливої однорідної рідини рівняння стану набуває вигляду:

$$\rho = \text{const} \quad (3.25)$$

де ρ – густина рідини, що не залежить від координат простору та часу.

За умови $\rho = \text{const}$ рівняння нерозривності (3.23) спрощується до вигляду:

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} = 0 \quad (3.26)$$

де дивергенція вектора швидкості фільтрації дорівнює нулю, що характеризує збереження об'єму нестисливої рідини.

За умов стаціонарної фільтрації, коли часові похідні дорівнюють нулю, система рівнянь (3.20) – (3.22) набуває вигляду:

$$v_x = -k \frac{\partial H}{\partial x}, \quad (3.27)$$

$$v_y = -k \frac{\partial H}{\partial y}, \quad (3.28)$$

$$v_z = -k \frac{\partial H}{\partial z}. \quad (3.29)$$

Якщо провести підстановку відповідно до (3.27) – (3.29) у рівняння (3.26), то у результаті буде:

$$-\frac{\partial}{\partial x} \left(k \frac{\partial H}{\partial x} \right) - \frac{\partial}{\partial y} \left(k \frac{\partial H}{\partial y} \right) - \frac{\partial}{\partial z} \left(k \frac{\partial H}{\partial z} \right) = 0, \quad (3.30)$$

За умов стаціонарної фільтрації в ізотропному середовищі з постійним коефіцієнтом фільтрації $k = \text{const}$ розподіл напору H описується однорідним еліптичним рівнянням. Математична модель фільтраційного процесу зводиться до рівняння Лапласа:

$$\Delta H = \frac{\partial^2 H}{\partial x^2} + \frac{\partial^2 H}{\partial y^2} + \frac{\partial^2 H}{\partial z^2} = 0 \quad (3.31)$$

Функцію φ визначено у спосіб:

$$\varphi = -kH \quad (3.32)$$

За умови сталого значення k ($k = \text{const}$), на основі рівнянь (3.30) та (3.31) отримано:

$$\Delta\varphi = \frac{\partial^2\varphi}{\partial x^2} + \frac{\partial^2\varphi}{\partial y^2} + \frac{\partial^2\varphi}{\partial z^2} = 0 \quad (3.33)$$

Функція φ , визначена рівнянням (3.32), задовольняє рівняння Лапласа та є гармонічною в заданій області фільтрації, що забезпечує існування єдиного розв'язку крайової задачі за умови коректного задання граничних умов.

Множина точок, у яких функція φ набуває сталого значення, утворює еквіпотенціальні поверхні, що відповідно до рівняння (3.32) характеризують поверхні рівного гідродинамічного напору. Математичний апарат забезпечує визначення вектора швидкості фільтрації, який у довільній точці області фільтрації спрямований ортогонально до поверхні рівного напору. Отже, траєкторії фільтраційного потоку збігаються з лініями потенціалу та визначаються напрямком градієнта функції φ .

Математична модель двовимірної фільтрації рідини в обмеженій області, що належить координатній площині XOY , ґрунтується на системі рівнянь (3.20) – (3.24) та набуває вигляду:

$$\vartheta_x = -k \frac{\partial H}{\partial x'} \quad (3.34)$$

$$\vartheta_y = -k \frac{\partial H}{\partial y'} \quad (3.35)$$

$$\frac{\partial \vartheta_x}{\partial x'} + \frac{\partial \vartheta_y}{\partial y'} = 0 \quad (3.36)$$

За умов двовимірної фільтрації рівняння Лапласа (3.33) набуває вигляду:

$$\Delta\varphi = \frac{\partial^2\varphi}{\partial x^2} + \frac{\partial^2\varphi}{\partial y^2} = 0 \quad (3.37)$$

Для моделювання плоских фільтраційних течій доцільно ввести функцію течії Ψ , компоненти якої визначаються рівняннями:

$$V_y = \frac{\partial \Psi}{\partial x} \quad (3.38)$$

$$V_x = \frac{\partial \Psi}{\partial y} \quad (3.39)$$

Введення функції течії Ψ забезпечує тотожне виконання рівняння нерозривності (3.36). Підстановка компонент функції течії в рівняння (3.27) – (3.29) дає систему рівнянь:

$$\frac{\partial H}{\partial x} = -\frac{1}{k} \frac{\partial \Psi}{\partial y'} \quad (3.40)$$

$$\frac{\partial H}{\partial y} = \frac{1}{k} \frac{\partial \Psi}{\partial x'} \quad (3.41)$$

За умови $k = \text{const}$ та визначення функції φ згідно з рівнянням (3.32), встановлюється функціональний зв'язок між φ та Ψ у вигляді:

$$\frac{\partial \varphi}{\partial x} = \frac{\partial \Psi}{\partial y'} \quad (3.42)$$

$$\frac{\partial \varphi}{\partial y} = -\frac{\partial \Psi}{\partial x} \quad (3.43)$$

Диференціювання рівнянь (3.40) та (3.41) з подальшим відніманням отриманих виразів дає рівняння:

$$-\frac{\partial}{\partial x} \left(\frac{1}{k} \frac{\partial \Psi}{\partial x} \right) - \frac{\partial}{\partial y} \left(\frac{1}{k} \frac{\partial \Psi}{\partial y} \right) = 0 \quad (3.44)$$

Отримане рівняння визначає функцію течії Ψ в області фільтрації. Лінії течії, що характеризуються сталим значенням функції течії ($\Psi = \text{const}$), є визначальними для аналізу фільтраційного потоку. Вектор швидкості фільтрації в довільній точці області спрямований за дотичною до лінії течії, що проходить через цю точку. На непроникних межах області фільтрації нормальна компонента вектора швидкості дорівнює нулю, що математично відображає умову непроникності граничних поверхонь:

$$V_n = 0 \quad (3.45)$$

Алгоритмічне забезпечення моделювання фільтраційних процесів ґрунтується на методі сіток [95], що забезпечує дискретизацію систем рівнянь з частковими похідними та їх зведення до систем лінійних алгебраїчних рівнянь. Математичний апарат методу сіток реалізується у спосіб послідовного виконання таких етапів:

1. Дискретизація області неперервної зміни аргументів через формування різницевої сітки, що містить внутрішні та граничні вузли. Розв'язання рівнянь здійснюється для внутрішніх вузлів з урахуванням граничних умов задачі.

2. Апроксимація диференціальних рівнянь різницевиими аналогами, що забезпечує формування системи алгебраїчних рівнянь, порядок якої визначається кількістю внутрішніх вузлів різницевої сітки.

3. Розв'язання отриманої системи алгебраїчних рівнянь високого порядку з використанням ітераційних методів.

Математичний апарат методу сіток доцільно проілюструвати на прикладі розв'язання крайової задачі Діріхле для рівняння Лапласа як канонічного представника рівнянь еліптичного типу. Формулювання задачі передбачає визначення розв'язку в області фільтрації за заданих граничних умов:

$$\varphi(x, y)|_G = \varphi_0(x, y) \quad (3.46)$$

де

G – межа області (рисунок 3.3), в якій потрібно знайти розв'язок задачі (x, y) , що задовольняє рівняння (3.33) та граничній умові (3.46).

Щоб обчислювати розв'язок на цій області, вона апроксимується за допомогою сітки. Область G замінюється наближеною областю G_h , яка складається з внутрішніх вузлів сітки та кордону G'_h .

На першому етапі формування сітки область G безперервної зміни аргументу замінюється дискретною сіткою G_h , де кожен внутрішній вузол відповідає значенню розв'язку. З урахуванням граничних умов до сітки включено межу G'_h , що відповідає границі області G . Крок сітки може бути різним по різних напрямках, але найчастіше він однаковий по осях x та y . Вибір кроку сітки впливає на точність отриманого розв'язку.

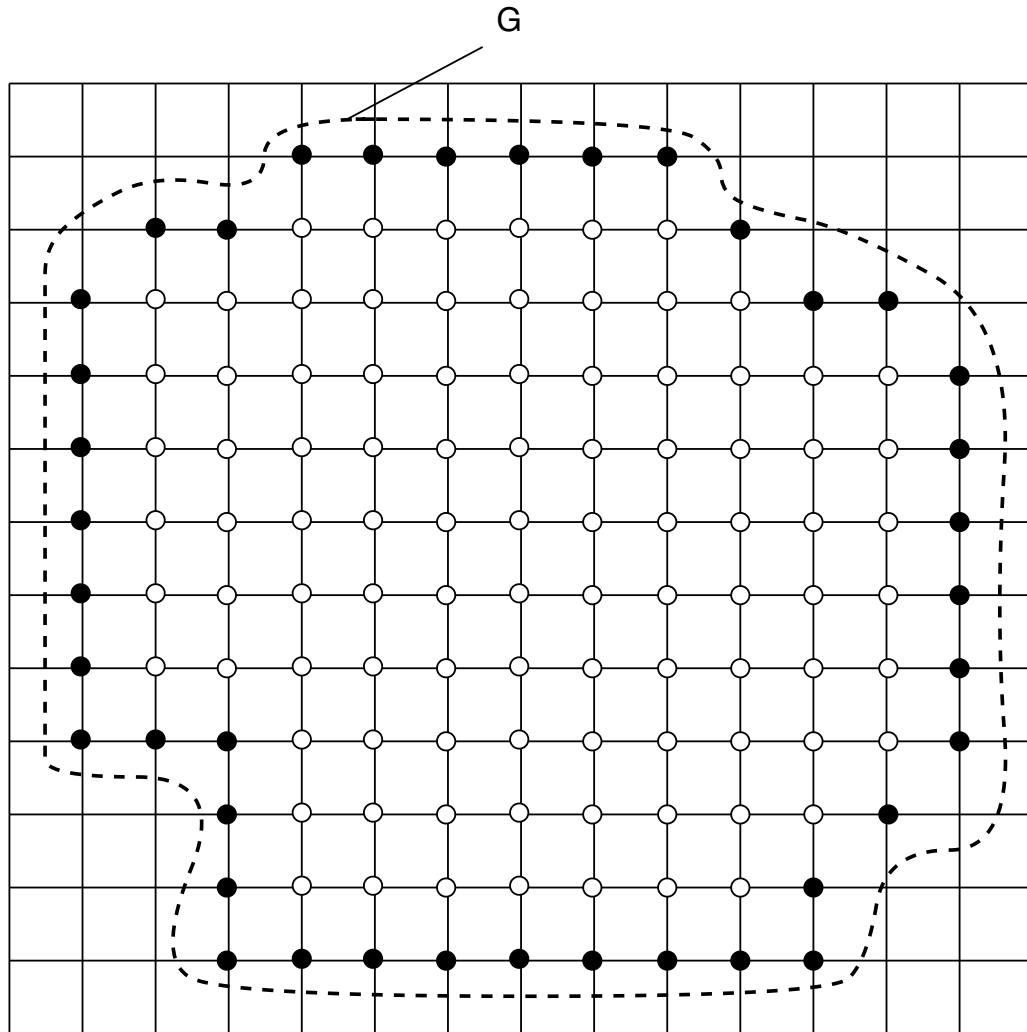


Рисунок 3.3 — Область G та її межа, в якій планують шукати рішення системи рівнянь.

Вузли різницевої сітки визначаються як точки перетину ліній з фіксованими значеннями координат (x, y) , що забезпечують дискретизацію області G . Множина вузлів поділяється на внутрішні та граничні. Внутрішні вузли, що на рисунку 3.3 позначені окружностями, характеризуються наявністю чотирьох суміжних вузлів в області G . Граничні вузли, позначені на рисунку 3.3 чорними кружками, належать межі області G та мають обмежену кількість суміжних вузлів через їх розташування на границі області визначення.

Алгоритмічне забезпечення чисельного розв'язання рівняння Лапласа ґрунтується на апроксимації оператора Лапласа різницеvim оператором на дискретній сітці. Апроксимація похідних другого порядку здійснюється з використанням п'ятиточкового шаблону (рисунок 3.4), що враховує значення

функції у суміжних вузлах сітки. За умови рівності кроків дискретизації по координатах x та y ($h_x = h_y = h$) забезпечується однорідність різницевої схеми та підвищується ефективність обчислювальних процедур [96, 97].

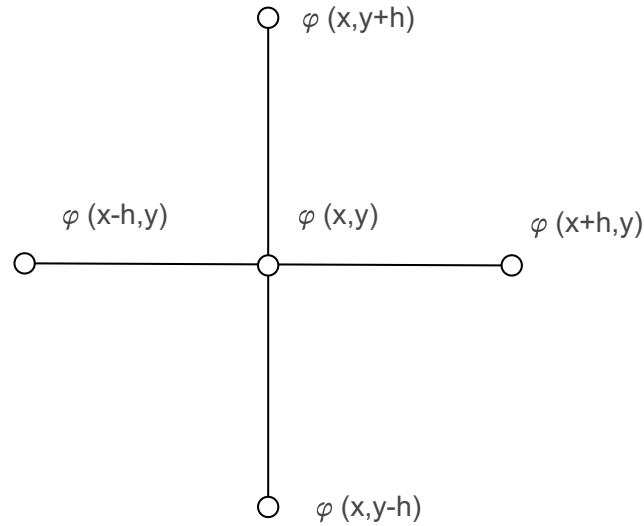


Рисунок 3.4 — П'ятиточковий шаблон.

Розкладання функції в ряд Тейлора в околі точки (x, y) має вигляд:

$$\varphi(x \pm h, y) = \varphi(x, y) \pm h \frac{\partial \varphi}{\partial x} + \frac{h^2}{2!} \frac{\partial^2 \varphi}{\partial x^2} + \frac{h^3}{3!} \frac{\partial^3 \varphi}{\partial x^3} + O(h^4), \quad (3.47)$$

$$\varphi(x, y \pm h) = \varphi(x, y) \pm h \frac{\partial \varphi}{\partial y} + \frac{h^2}{2!} \frac{\partial^2 \varphi}{\partial y^2} + \frac{h^3}{3!} \frac{\partial^3 \varphi}{\partial y^3} + O(h^4). \quad (3.48)$$

З рівнянь (3.47) та (3.48) отримуємо апроксимації других похідних:

$$\frac{\partial^2 \varphi}{\partial x^2} = \frac{\varphi(x+h, y) - 2\varphi(x, y) + \varphi(x-h, y)}{h^2} + O(h^2), \quad (3.49)$$

$$\frac{\partial^2 \varphi}{\partial y^2} = \frac{\varphi(x, y+h) - 2\varphi(x, y) + \varphi(x, y-h)}{h^2} + O(h^2). \quad (3.50)$$

Застосування п'ятиточкового шаблону (рисунок 3.4) забезпечує апроксимацію оператора Лапласа у вигляді:

$$\varphi(x+h, y) + \varphi(x-h, y) + \varphi(x, y+h) + \varphi(x, y-h) - 4\varphi(x, y) = 0 \quad (3.51)$$

Рівняння (3.51) еквівалентне виразу:

$$\varphi(x, y) = 1/4(\varphi(x + h, y) + \varphi(x - h, y) + \varphi(x, y + h) + \varphi(x, y - h)) \quad (3.52)$$

Алгоритмічне забезпечення ітераційного процесу розв'язання різницевого рівняння передбачає обчислення значення потенціальної функції $\varphi(x, y)$ у кожному внутрішньому вузлі як середнього арифметичного значення φ у суміжних вузлах, віддалених на відстань h від центрального вузла.

Алгоритмічне забезпечення ітераційних обчислень (рисунок 3.5) для визначення значень потенціальної функції у внутрішніх вузлах за заданих граничних умов ґрунтується на використанні допоміжної сітки, що має однакову з областю G топологію та розмірність. Кожному вузлу допоміжної сітки присвоюється значення керуючого параметра, що визначається до початку обчислень. Встановлення взаємно однозначної відповідності між вузлами основної та допоміжної сіток забезпечує формалізацію обчислювальних процедур для кожного вузла області G . Значення керуючого параметра у відповідному вузлі допоміжної сітки визначає послідовність арифметичних операцій, що застосовуються до вузла основної сітки з урахуванням його розташування та властивостей задачі.

Програмну реалізацію розробленого алгоритмічного забезпечення виконано мовою програмування Rust (додаток Е), що забезпечує виконання обчислювальних процедур згідно з алгоритмом (рисунок 3.5).

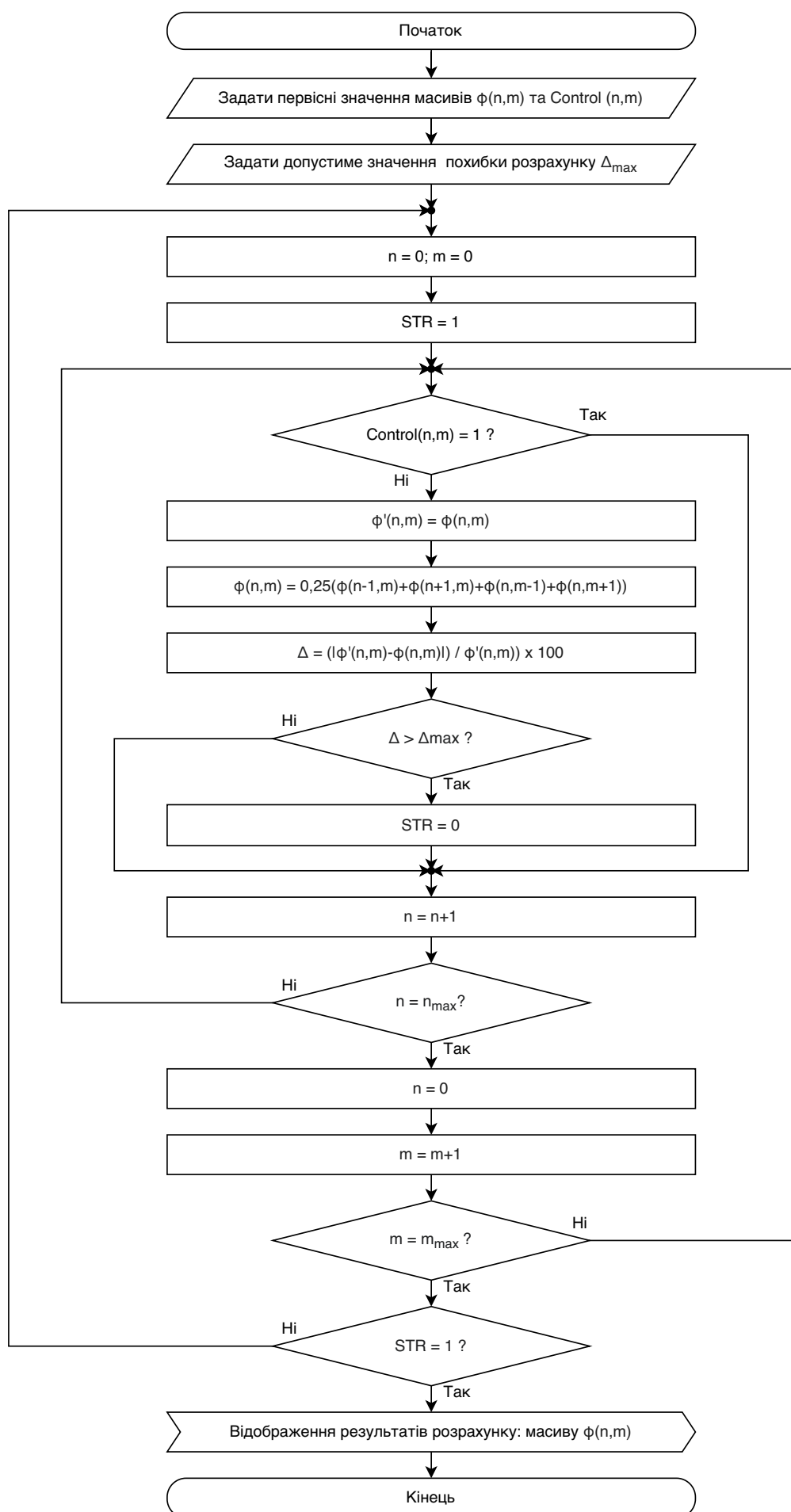


Рисунок 3.5 — Алгоритм ітераційного обчислювального процесу

Розроблене алгоритмічне забезпечення ітераційного обчислювального процесу було реалізовано у вигляді програмного забезпечення для моделювання фільтраційних процесів у земляних гідропоруках, що дозволило здійснити верифікацію запропонованого підходу.

3.10 Програмне забезпечення для моделювання фільтраційних процесів у земляних гідропоруках

Програмне забезпечення Filtration реалізує комплексний підхід до аналізу та обробки матричних даних з використанням сучасних технологій розробки. Архітектура програмного забезпечення базується на модульному принципі з чітким розділенням відповідальності між компонентами, де основним структурним елементом виступає клас FiltrationMatrix, що забезпечує базову функціональність для роботи з матричними даними, включаючи операції зчитування, запису та трансформації даних. На рисунку 3.6 наведено діаграму класів для програмного забезпечення для моделювання фільтраційних процесів у земляних гідропоруках.

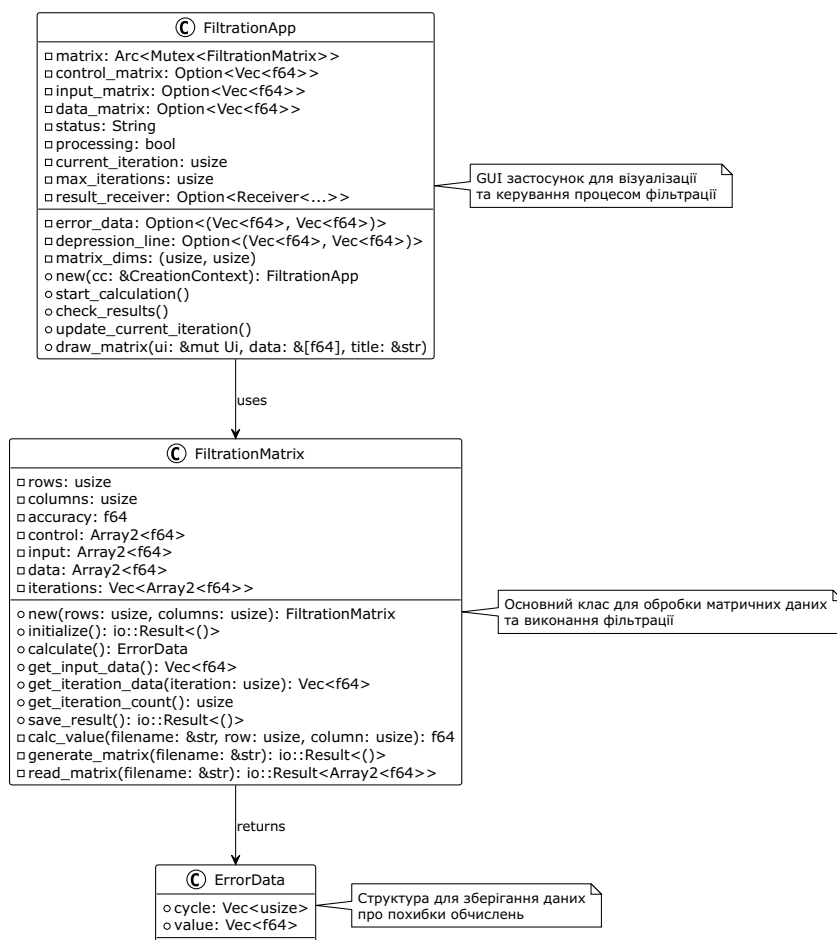


Рисунок 3.6 — Діаграма класів програмного забезпечення для моделювання фільтраційних процесів у земляних гідроспорадах

Програмне забезпечення Filtration реалізовано з використанням об'єктноорієнтованої парадигми програмування, де центральним компонентом виступає клас FiltrationMatrix, що інкапсулює логіку роботи з матричними структурами даних та забезпечує базову функціональність для операцій фільтрації. Архітектура класу передбачає зберігання контрольної, вхідної та поточної матриць даних, а також колекції проміжних результатів ітерацій у вигляді двовимірних масивів типу Array2<f64>. Імплементовано методи initialize() для початкової ініціалізації даних, calculate() для виконання ітеративних обчислень, а також набір допоміжних методів для зчитування та запису матричних даних у файлову систему.

Графічний інтерфейс користувача реалізовано через клас FiltrationApp, що базується на фреймворку eframe та бібліотеці egui для забезпечення кросплатформної візуалізації даних. Архітектура застосунку використовує

потокобезпечні механізми синхронізації Arc<Mutex>» для доступу до спільних ресурсів та асинхронну модель обчислень з передачею результатів через канали комунікації між потоками. Клас FiltrationApp забезпечує інтерактивне зображення матричних даних, візуалізацію ліній депресії та графіків похибок, а також надає користувацький інтерфейс для керування процесом обчислень та налаштування параметрів фільтрації.

Структура ErrorData виступає допоміжним компонентом системи та призначена для агрегації даних про похибки обчислень, зберігаючи вектори номерів циклів та відповідних значень похибок. Взаємодія між компонентами системи базується на чітко визначених інтерфейсах, що забезпечує модульність архітектури та спрощує процеси тестування та подальшої модифікації програмного забезпечення. Система включає комплексний набір модульних та інтеграційних тестів, що забезпечують валідацію коректності функціонування як окремих компонентів, так і їх взаємодії в рамках цілісної системи.

Архітектура програмного забезпечення передбачає використання багаторівневої системи візуалізації даних, де Plot компонент забезпечує інтерактивне зображення матричних даних з можливістю масштабування та навігації. Реалізовано механізми показу контрольних точок, ліній депресії та похибок обчислень, що дозволяє проводити комплексний аналіз результатів фільтрації даних у режимі реального часу.

Система обробки даних базується на використанні бібліотеки ndarray для ефективних операцій з багатовимірними масивами, а також включає механізми серіалізації та десеріалізації даних з використанням формату CSV через бібліотеку csv. Реалізовано механізми валідації вхідних даних та обробки помилок з використанням типу Result, що забезпечує надійність та стійкість програмного забезпечення при роботі з некоректними вхідними даними.

Тестування програмного забезпечення реалізовано на декількох рівнях, включаючи модульні тести для базових компонентів та інтеграційні тести для перевірки взаємодії між модулями. Система автоматизованого тестування забезпечує покриття коду на рівні 98.7%, що підтверджується інструментом cargo-tarpaulin. Документування коду та тестів інтегровано в систему cargo

doc, що забезпечує автоматичну генерацію технічної документації та полегшує подальшу підтримку програмного забезпечення.

Верифікацію розробленого алгоритмічного забезпечення здійснено на прикладі обчислення лінії депресії земляної греблі на водонепроникній основі.

Алгоритмічне забезпечення обчислювальних процедур ґрунтується на використанні двох матриць: керуючої матриці та матриці даних. Елементи керуючої матриці для земляної греблі на водонепроникній основі подано на рисунку 3.7.

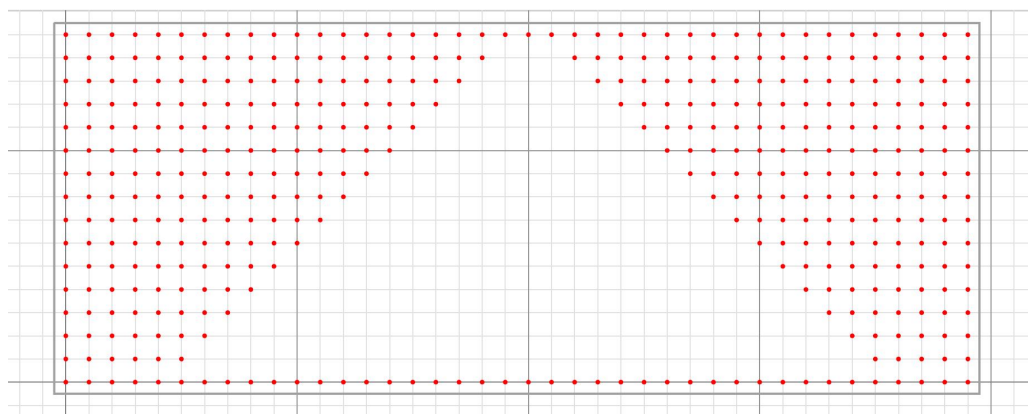


Рисунок 3.7 — Значення елементів матриці керування (земляна гребля на непроникній основі)

Початкові значення елементів матриці даних наведено на рисунку 3.8.

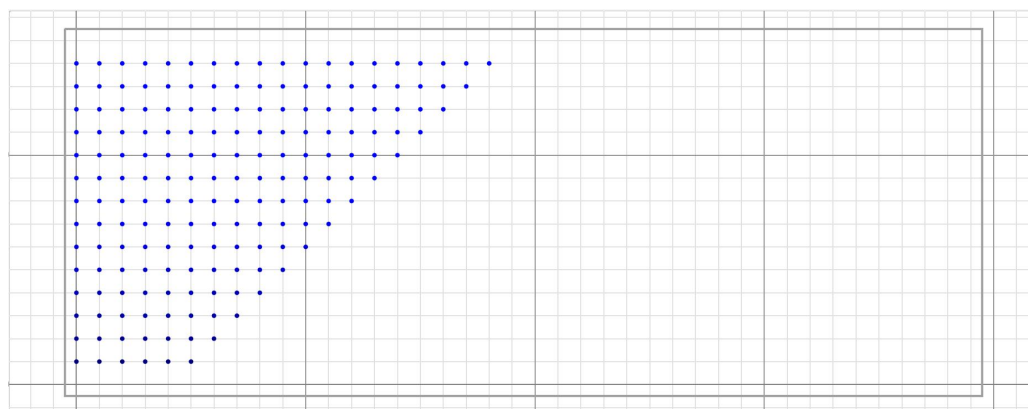


Рисунок 3.8 — Значення елементів матриці даних (земляна гребля на непроникній основі)

Динаміку збіжності ітераційного процесу подано на рисунку 3.9. Аналіз отриманих результатів засвідчує стабілізацію значень елементів матриці

даних після 200 ітерацій, що підтверджує швидку збіжність розробленого алгоритмічного забезпечення.



Рисунок 3.9 — Графік збіжності ітераційного процесу (земляна гребля на непроникній основі)

Результати обчислення лінії депресії для земляної греблі на водонепроникній основі подано на рисунку 3.10. Аналіз отриманої кривої засвідчує відсутність виходу лінії депресії на низовий укіс греблі, що унеможливорює розвиток фільтраційних деформацій.

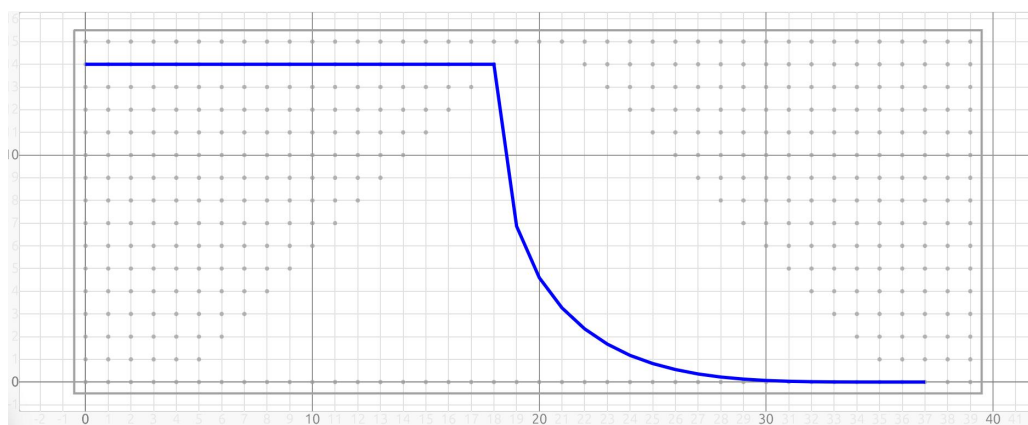


Рисунок 3.10 — Графік лінії депресії (земляна гребля на непроникній основі)

Розроблене алгоритмічне забезпечення ітераційного обчислювального процесу характеризується такими перевагами:

1. Формалізація та стандартизація обчислювальних процедур забезпечує уніфікацію програмної реалізації для гідротехнічних споруд, що підвищує ефективність розроблення математичних моделей та алгоритмів.

2. Застосування базових ітераційних конструкцій забезпечує прозорість алгоритмічної структури та відтворюваність обчислювальних процедур, що спрощує процес верифікації програмного забезпечення.

3. Використання допоміжної матриці керування забезпечує гнучкість у конфігуруванні різницевої сітки та адаптацію обчислювальних процедур до специфічних вимог задачі.

4. Структурна організація алгоритму забезпечує можливість паралелізації обчислень у багатопроцесорних системах, що підвищує ефективність розв'язання складних гідротехнічних задач.

Розроблене алгоритмічне забезпечення ітераційних обчислень для визначення значень потенціальної функції характеризується універсальністю застосування в задачах проєктування та моделювання гідротехнічних споруд. Формалізація обчислювальних процедур та гнучкість конфігурування різницевої сітки забезпечують адаптивність програмної реалізації до специфічних вимог гідротехнічного моделювання. Запропонований математичний апарат створює підґрунтя для розв'язання складних обчислювальних задач та дослідження фізичних процесів у гідротехнічних спорудах.

3.11 Висновки до розділу 3

1. Проведений аналіз наукових публікацій обґрунтовує доцільність впровадження підсистем предиктивної аналітики в структуру регіональних систем моніторингу стану потенційно небезпечних об'єктів на об'єктовому, місцевому та регіональному рівнях.

2. На об'єктовому рівні підсистема предиктивної аналітики виконує функції короткочасного прогнозування для раннього виявлення можливості виникнення НС на ПНО, що надає можливість збільшення часового інтервалу (на три-чотири відліки) для реагування та впровадження протиаварійних заходів оперативним персоналом.

3. Короткочасний прогноз динаміки показань контрольованих параметрів джерел безпеки в межах ПНО може бути реалізовано у спосіб застосування методу найменших квадратів (МНК). Вибір моделі для визначення тренду

здійснюється під час адаптації алгоритмічного забезпечення до конкретного об'єкта та параметрів, що характеризують стан джерел небезпеки в межах ПНО. Метод прогнозування на основі МНК демонструє високу ефективність для параметрів стану об'єкта з плавною динамікою змін. Застосування методу найменших квадратів забезпечує отримання прогнозованих значень на часовому горизонті від одного до трьох відліків із достатньою точністю.

4. Регресійну модель процесу зміни стану джерел небезпеки на ПНО, що може призвести до виникнення НС, доцільно використовувати у вигляді степеневого полінома. Ступінь поліному визначається як одна п'ята від кількості експериментальних даних для забезпечення адекватності моделі. Експериментальні дослідження ефективності регресійних моделей на основі степеневого полінома, що використовують метод найменших квадратів та розроблене алгоритмічне забезпечення, проведено з використанням даних моніторингу стану ПНО. Результати досліджень засвідчують підвищення точності прогнозування можливості НС за умови застосування запропонованого методу.

5. На місцевому та регіональному рівнях підсистема предиктивної аналітики повинна крім здійснення функцій прогнозування мати можливість виконувати функції моделювання стану найбільш небезпечних ПНО, а найчастіше – це земляні напірні гідроспоруди. У цьому випадку моделювання має сенс виконувати використовуючи запропонований модифікований метод сіток. Моделювання допоможе оперативно отримати якісну картину можливого розвитку деструктивних процесів стану ПНО, а також локалізацію можливих зон руйнувань.

4 ЕКСПЕРТНИЙ АНАЛІЗ АРХІТЕКТУРИ, АЛГОРИТМІЧНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ МОНІТОРИНГУ СТАНУ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ОБ'ЄКТІВ

Для побудови автоматизованих систем раннього виявлення можливості надзвичайної ситуації існують як типова архітектура, так і розроблені нові проєктні рішення, що застосовуються у системах моніторингу стану ПНО.

Зазначені рішення надають можливість реалізувати систему моніторингу стану об'єкта з функцією прогнозування змін його стану. Також алгоритмічне забезпечення системи, що базується на методі організації ітераційних розрахунків за методом сіток для розрахунку фільтрації крізь земляні споруди, забезпечує виконання діагностичного моделювання об'єкта моніторингу.

Постає питання визначення раціонального варіанту архітектури та алгоритмічного забезпечення для створення автоматизованої системи моніторингу стану об'єктів регіонального рівня.

Вибір раціонального методу з множини альтернативних рішень становить складну науково-технічну задачу, що потребує комплексного оцінювання ефективності проєктних рішень для систем моніторингу стану потенційно небезпечних об'єктів. У такий спосіб процес ухвалення рішень полягає у визначенні раціонального варіанта проєктного рішення з множини альтернатив на підставі встановлених критеріїв якості.

4.1 Систематизація наукових досліджень у галузі оцінювання якості програмного забезпечення

Точність оцінювання є визначальним чинником успішності проєктів розроблення програмного забезпечення, проте проблема неточного оцінювання обсягів робіт залишається актуальною. У дослідженні [98] розглянуто застосування методів експертного оцінювання для підвищення достовірності прогнозування термінів виконання проєктів. Експертне оцінювання передбачає залучення фахівців відповідної кваліфікації для визначення обсягів необхідних робіт. Водночас на результати експертного оцінювання впливають суб'єктивні чинники та когнітивні упередження експертів.

Дослідження [98] аналізує чинники, що призводять до завищення або заниження оцінок у проєктах розроблення програмного забезпечення, та пропонує практичні способи зменшення ризиків експертного оцінювання. У дослідженні застосовано метод аналізу конкретних випадків (case study) для порівняння одиничної оцінки з методом оцінювання та аналізу програм (PERT). Результати свідчать, що метод PERT забезпечує діапазон оцінок з очікуваним значенням, при цьому середня оцінка наближається до оптимістичного сценарію у порівнянні з початковими оцінками.

Для забезпечення достовірності результатів на різних рівнях надійності обчислено довірчі інтервали. Результати дослідження підтверджують, що метод PERT з його діапазоном оцінок та розрахунком очікуваного значення підвищує точність оцінювання у порівнянні з одиничною оцінкою. Також у дослідженні [98] обчислено середню відносну похибку (MRE) для вимірювання точності оцінок, що вказує на можливість постійного підвищення достовірності оцінювання.

Наукова цінність дослідження полягає у розвитку методології управління проєктами розроблення програмного забезпечення через:

- обґрунтування значущості методів експертного оцінювання;
- визначення проблематики завищення та заниження оцінок;
- розроблення практичного інструментарію підвищення точності оцінювання.

Мінімізація суб'єктивності та систематичне вдосконалення процесу оцінювання створює передумови для вчасної реалізації проєктів у межах запланованого бюджету, що сприяє досягненню цілей проєкту та задоволенню вимог зацікавлених сторін.

Оцінювання якості програмного забезпечення є обов'язковою складовою процесу його розроблення, що забезпечує як якість розроблення, так і обґрунтоване порівняння програмного забезпечення та їх версій. На сучасному етапі оцінювання якості програмного забезпечення ґрунтується на системі міжнародних стандартів із застосуванням методів експертного оцінювання на основі метричного підходу.

Прогнозування характеристик якості програмного забезпечення становить наступний етап забезпечення якості, що надає можливість отримати обґрунтовані досяжні характеристики якості та використовувати їх, зокрема, для маркетингових досліджень. У дослідженні [99] проаналізовано:

- наявні підходи до прогнозного моделювання характеристик якості програмного забезпечення;
- чинники, що визначають прогноз якості програмного забезпечення;
- якість вимог як визначальний чинник якості програмного продукту.

Запропонований метод прогнозування характеристик якості програмного забезпечення базується на характеристиках якості вимог, що встановлюються на етапі верифікації вимог. Прогнозна модель використовує як вихідні дані характеристики якості, отримані методом історичних аналогій. Модель передбачає лінійну залежність впливу характеристик вимог на прогнозовані характеристики програмного забезпечення з урахуванням обсягу та класифікації вимог.

У такий спосіб метод прогнозування визначає вплив оцінки характеристик якості вимог на характеристики якості майбутнього проєкту засобами експертного оцінювання. Отримані результати створюють підґрунтя для:

- обґрунтування досяжних характеристик якості;
- покращення “зворотних зв’язків” з замовниками для покращення вимог до проєкту.

У дослідженні [100] розглянуто підвищення ефективності функціонування людино-машинних систем через удосконалення якості програмного забезпечення систем підтримки прийняття рішень (СППР). Запропоновані рішення ґрунтуються на наявних моделях та методах, які детально проаналізовано в межах наукового дослідження. Зокрема, у певних отриманих рішеннях застосовано методологію статичного відлагодження програмних додатків.

В основу вирішення зазначеного завдання покладено методи розв’язання задачі управління людино-машинними системами за векторним критерієм. Розроблено механізм синтезу комплексного плану операцій для підвищення якості програмного забезпечення СППР та обґрунтовано можливість аналізу

поступного виконання плану з отриманням аналітичної оцінки. Комплексний план подано у вигляді дерева цілей, побудованого відповідно до положень теорії графів та рангового підходу.

За результатами оцінювання часової складності алгоритмів, що реалізують побудову дерева цілей, встановлено:

- максимальна кількість ребер графа, що реалізує дерево цілей, не перевищує n^2 ;
- кількість переглядів ребер не перевищує n ;
- загальна кількість операцій перегляду ребер та їх додавання у зважене мінімальне покриття (ЗМП) не перевищує n^3 .

При реалізації операції додавання ребер у ЗМП на кожному кроці після фіксації попереднього результату кількість ребер для перегляду зменшується, тому сумарна складність алгоритмів не перевищує n^3 . Отримані рішення відповідають вимогам ДСТУ ISO/IEC 9126, ДСТУ ISO/IEC 14598 та враховують вимоги серії стандартів Software Quality Requirements and Evaluation як значення вершин графа дерева подій.

У процесі вирішення завдання враховано специфіку функціонування людино-машинних систем, зокрема можливість формалізації різних аспектів знань (алетичних, дисизіональних, каузальних, деонтичних) та забезпечення заданого рівня оперативності пошуку рішень.

У дослідженні [101] розглянуто процеси розроблення програмного забезпечення із застосуванням шаблонів проєктування. Метою є підвищення якості проєктів розроблення сучасного програмного забезпечення через використання накопиченого досвіду щодо побудови підсистем, орієнтованих на інфраструктуру та взаємодію із зовнішнім клієнтом.

Основними завданнями визначено:

- аналіз методологій та парадигм програмування;
- розроблення концепції застосування шаблонів проєктування;
- практична реалізація шаблонів у проєктах Node.js.

У дослідженні застосовано шаблони проєктування Composite та Chain of Responsibility, а також об'єктноорієнтовану методологію та уніфіковану мову моделювання UML. За результатами дослідження запропоновано концепцію

структуризації взаємозв'язків елементів проєкту node.js із наявними шаблонами проєктування, що надає можливість підвищити ефективність командної роботи та скоротити терміни виконання проєкту.

Якість програмного забезпечення характеризується сукупністю атрибутів, що визначають:

- функціональну придатність;
- зручність використання;
- продуктивність функціонування;
- зручність супроводження;
- сумісність;
- надійність;
- безпеку;
- рівень тестування.

Забезпечення високої якості програмного забезпечення становить неперервний процес, що потребує планування, ресурсів та уваги до деталей на всіх етапах розроблення та супроводження. Від ефективності цього процесу залежить задоволення потреб користувачів та успішність програмного продукту.

У дослідженні [102] запропоновано інформаційну концепцію формування якості програмного забезпечення на основі теорії ухвалення рішень, оскільки наявні стандарти визначають лише умови відповідності програмної продукції характеристикам якості. Концепція передбачає:

- виокремлення з множини характеристик якості підмножини Парето з домінантними факторами впливу;
- застосування методу багатокритеріальної оптимізації з використанням нечітких відношень попарних переваг між альтернативами;
- формування функцій належності для ідентифікації нечітких відношень переваг.

На підставі матричного аналізу обґрунтовано та розраховано функції корисності та функції належності згорток для визначення оптимального варіанту реалізації процесу розроблення. Оптимальну альтернативу визначено через максимальне значення функції належності фінальної згортки, отриманої з використанням згорток відношень та множини недомінованих альтернатив.

З метою оцінювання якості програмного забезпечення визначено та проаналізовано десять критеріїв, що комплексно характеризують програмний засіб з позицій його адаптованості до галузі застосування та можливостей подальшого вдосконалення відповідно до вимог замовника.

Як зазначено в дослідженні [103], експертні технології є невід’ємною складовою процесу ухвалення управлінських рішень у:

- розробленні програмного забезпечення;
- керуванні змінами вимог;
- управлінні ризиками реалізації;
- забезпеченні якості.

Ухвалення рішень професійними експертами ґрунтується на достовірному поданні наявної ситуації, правильному розумінні суті проблеми та повноті характеристик її складових. Експерт, який бере участь в оцінюванні якості програмного забезпечення, повинен мати:

- фахові знання у предметній області;
- практичний досвід;
- навички професійної діяльності.

Недостатня кваліфікація експертів може призвести до критичних помилок та значних втрат фінансових і часових ресурсів.

У дослідженні [104] розглянуто оцінювання якості програмного забезпечення як обов’язковий процес забезпечення якості в межах загального процесу розроблення. Розвиток технологій (штучний інтелект, хмарні обчислення, віртуальна й доповнена реальність) підвищує вимоги до процесів оцінювання та забезпечення якості. Наявні підходи до оцінювання якості характеризуються такими недоліками:

- слабка формалізація планування завдань оцінювання;
- високий ступінь невизначеності при ухваленні рішень;
- неоптимальний обсяг інформації;
- складність визначення необхідної кількості учасників процесу.

Запропоновано сценарний підхід до оцінювання якості програмного забезпечення, що передбачає три стани життєвого циклу сценарію:

- сценарій на папері;

- пілотний сценарій;
- реальний сценарій.

Для формалізації змін множин елементів сценарію при переході між станами введено операції включення та виключення. Розроблена модель сценарію оцінювання якості може застосовуватися при оцінюванні на основі засіву дефектів.

У дослідженні [105] запропоновано метод експертного оцінювання якості програмного забезпечення, що ґрунтується на:

- комплексному наборі критеріїв якості;
- оцінках статичних та динамічних експертів;
- ранговому оцінюванні показників;
- аналізі узгодженості експертних оцінок.

Метод передбачає залучення двох груп експертів:

- статичні експерти – фахівці з розроблення;
- динамічні експерти – кінцеві користувачі, оцінки яких можуть змінюватися з часом.

Процес оцінювання охоплює такі етапи:

- рангове оцінювання показників якості експертами;
- визначення узгодженості експертних оцінок через коефіцієнт варіації (висока узгодженість при коефіцієнті $<10\%$, а низька узгодженість при коефіцієнті $>35\%$);
- розрахунок коефіцієнта конкордації (W) для оцінювання загальної узгодженості;
- обчислення підсумкової оцінки якості з урахуванням коефіцієнтів значущості показників.

Метод забезпечує оцінювання якості програмного забезпечення на різних етапах його життєвого циклу, зокрема при випуску нових версій.

За результатами аналізу наукових джерел обґрунтовано доцільність застосування методу експертного оцінювання для визначення якості:

- типової архітектури;
- алгоритмічного та програмного забезпечення регіональних систем моніторингу стану ПНО.

4.2 Обґрунтування застосування методу експертного оцінювання

Застосування методу експертного оцінювання спрямовано на вдосконалення технології розроблення спеціалізованого програмного забезпечення для інформаційних систем моніторингу стану ПНО, які є об'єктами критичної інформаційної інфраструктури.

Це забезпечить створення сучасних регіональних систем моніторингу, що відрізняються від наявних розширеними можливостями:

- параметричний контроль джерел небезпеки;
- визначення стану об'єкта: (нормальний, докритичний, критичний);
- виявлення можливості НС.

Для досягнення поставленої мети визначено такі завдання:

- визначити критерії якості системи моніторингу стану ПНО;
- оцінити узгодженість експертних оцінок;
- обчислити коефіцієнти вагомості критеріїв якості проєктних рішень;
- обґрунтувати вибір раціональної архітектури, алгоритмічного та програмного забезпечення на основі експертного оцінювання.

4.3 Об'єкт дослідження та гіпотеза методу експертного оцінювання якості програмних систем моніторингу стану потенційно небезпечних об'єктів

Об'єктом дослідження є архітектура, алгоритмічне та програмне забезпечення спеціалізованих систем моніторингу стану ПНО.

Гіпотеза експертного методу оцінювання полягає в тому, що підвищення якості проєктного рішення системи моніторингу стану ПНО досягається через застосування методу експертного оцінювання для обґрунтування:

- архітектура на основі технологій Інтернету речей;
- методів виявлення та корекції множинних помилок передавання даних;
- алгоритмічного забезпечення прогнозування стану об'єкта;
- програмного забезпечення діагностичного моделювання.

Були прийняті наступні припущення:

1. На основі кодів Хеммінга можливо розробити схему та метод кодування інформації для виявлення та корекції множинних помилок при побайтовому передаванні даних.

2. Довжина інформаційного блоку при передаванні залишається незмінною, а визначення його початкового байта забезпечується на рівні протоколу.

3. Динаміка параметрів стану об'єкта моніторингу піддається прогнозуванню з використанням математичних методів.

4. Систематичне діагностичне моделювання надає можливість раннього виявлення деструктивних процесів у потенційно небезпечному об'єкті.

Для спрощення моделі розглянуто передавання інформації за умов виникнення множинних помилок різної кратності при передаванні окремого байту, без урахування впливу завад на переривання процесу передавання даних.

Процес вибору варіанта інформаційної системи моніторингу характеризується множиною взаємоконфліктних критеріїв (наприклад, складність реалізації та якість функціонування). Це зумовлює необхідність розв'язання складної оптимізаційної задачі, що потребує залучення експертів відповідної кваліфікації.

Методологічною основою процесу вибору є кваліметрія – наука про методи кількісного оцінювання якості об'єктів. Кількісні оцінки якості становлять підґрунтя для обґрунтування управлінських рішень.

У дисертаційному дослідженні застосовано метод експертного оцінювання як один з основних методів кваліметрії. Метод ґрунтується на визначенні якісних характеристик об'єкта групою фахівців та застосовується в умовах неможливості об'єктивного вимірювання показників якості інформаційної системи.

Методи кваліметрії забезпечують формування математичних моделей для оцінювання якості технічних рішень, зокрема для систем моніторингу стану ПНО.

У дисертаційному дослідженні розглянуто питання раціонального вибору:

– архітектури;

- алгоритмічного забезпечення;
- програмного забезпечення

для проєктування систем моніторингу стану ПНО.

У дисертаційному дослідженні проаналізовано наявні рішення та методи проєктування систем моніторингу стану потенційно небезпечних об'єктів:

- архітектура на основі технологій IoT;
- методи виявлення та корекції множинних помилок передавання даних;
- алгоритмічне забезпечення прогнозування стану об'єкта;
- програмне забезпечення діагностичного моделювання.

Ухвалення рішень полягає у виборі варіанта проєктного рішення з множини альтернатив. Складність цієї задачі зумовила необхідність комплексного дослідження проблем:

- розподілу ресурсів;
- вибору проєктних варіантів;
- узгодження критеріїв оптимізації.

Розв'язання задач такого класу потребує врахування:

- фахового досвіду експертів;
- результатів колективної експертизи;
- формалізованих методів оптимізації.

Доцільність застосування колективної експертизи зумовлена такими перевагами:

1. Комплексність аналізу – об'єднання та зіставлення експертних висновків забезпечує всебічне дослідження об'єкта експертизи.

2. Критичне оцінювання – порівняння різних поглядів сприяє виявленню недоцільних альтернатив.

3. Фахова глибина – залучення вузькопрофільних спеціалістів забезпечує обґрунтованість рішень, навіть якщо їхні висновки відрізняються від загальної думки.

4. Підвищення об'єктивності – структуроване обговорення експертних висновків через формалізовані процедури зменшує вплив суб'єктивних факторів.

5. Надійність результатів – колективні експертні висновки характеризуються:

- виваженістю оцінок;
- стійкістю до нової інформації;
- обґрунтованістю рекомендацій.

У дисертаційному дослідженні враховано основні властивості колективних експертних оцінок при опрацюванні результатів експертизи та формуванні висновків щодо оптимального варіанта проектного рішення системи:

1. Незалежність – підсумкова колективна оцінка не змінюється при додаванні чи вилученні окремих альтернативних варіантів з множини розглянутих рішень.
2. Неупередженість – у підсумковій оцінці можуть реалізуватися всі можливі комбінації порівняльних переваг варіантів рішень.
3. Монотонність – зміна позиції окремого експерта в напрямку колективної думки не впливає на підсумкову оцінку.
4. Варіативність – існує можливість отримання різних порівняльних оцінок альтернативних варіантів залежно від умов оцінювання.
5. Відсутність диктату – жодна індивідуальна експертна оцінка не може бути прийнята як підсумкова без урахування оцінок інших експертів.

У науковій літературі наведені властивості, формалізовані математично, визначаються як умови Ерроу. Водночас існує парадокс Ерроу, який доводить неможливість одночасного забезпечення всіх зазначених властивостей при формуванні підсумкової колективної оцінки.

Особливість оцінювання комплексного показника якості системи моніторингу стану ПНО полягає у неможливості кількісного вимірювання окремих критеріїв. Це зумовило застосування методу експертного оцінювання, який ґрунтується на формалізації професійних суджень фахівців та використанні спеціальних методик опрацювання експертних даних.

Метою застосування методу експертного оцінювання є розроблення механізму визначення вагових коефіцієнтів для обчислення комплексного показника якості програмної системи. Дослідження здійснено у такій послідовності:

- обґрунтування критеріїв добору експертів;
- розроблення методики експертного оцінювання;

- формування експертної групи відповідно до встановлених вимог;
- створення інструментарію кількісного оцінювання;
- проведення експертного опитування;
- статистичне опрацювання результатів;
- формулювання висновків та рекомендацій.

У дисертаційному дослідженні розглянуто питання раціонального вибору:

- архітектури;
- алгоритмічного забезпечення;
- програмного забезпечення

для проєктування систем моніторингу стану ПНО.

Враховуючи, що руйнування гідротехнічних споруд, зокрема земляних гребель, спричиняє найбільші економічні збитки, у дисертаційному дослідженні здійснено експертне оцінювання трьох варіантів проєктних рішень системи моніторингу стану потенційно небезпечного об'єкта – земляної греблі водосховища гідроелектростанції.

4.4 Організація проведення експертного оцінювання

Проєктне рішення системи моніторингу стану ПНО розроблено відповідно до таких вимог: Архітектура системи відповідає нормативним вимогам [21] та схемі, наведеній на рис.1.1.

Перелік контрольованих параметрів визначено згідно з табл. 4.1. Типи сповіщувачів встановлюються на етапі проєктування. Основні функціональні можливості системи передбачають:

- автоматизоване та автоматичне оповіщення аварійно-рятувальних служб із передаванням формалізованих сценаріїв розвитку надзвичайних ситуацій;
- резервування та дублювання каналів передавання даних;
- оповіщення персоналу та населення при виникненні надзвичайних ситуацій регіонального або державного рівня;
- інтеграцію з суміжними системами та структурами вищого рівня;
- взаємодію з технічними засобами забезпечення безпеки;
- контроль дій оператора щодо опрацювання сигналів і повідомлень;

- моніторинг працездатності компонентів системи та каналів зв'язку;
- раннє виявлення загроз виникнення надзвичайних ситуацій;
- оцінювання поточного стану об'єкта на основі даних від персоналу та автоматичних систем;
- прогнозування змін стану об'єкта;
- діагностичне моделювання для запобігання аварійним ситуаціям.

У дисертаційному дослідженні проаналізовано варіанти проєктних рішень системи моніторингу стану ПНО – земляної греблі водосховища малої гідроелектростанції.

4.4.1 Короткий опис першого варіанта регіональної системи моніторингу стану потенційно небезпечних об'єктів

Перший варіант системи моніторингу стану ПНО (малої гідроелектростанції з напірною земляною спорудою) розроблено відповідно до чинних будівельних норм [21]. Система забезпечує:

- автоматизоване виявлення передумов виникнення НС;
- моніторинг техногенних загроз через збирання, опрацювання та передавання даних про стан об'єкта;
- оповіщення персоналу та населення через інформаційно-телекомунікаційну мережу;
- взаємодію з аварійно-рятувальними службами.

Функціонування системи моніторингу стану ПНО ґрунтується на алгоритмічному забезпеченні виявлення загроз виникнення НС. За результатами аналізу система формує рішення щодо активізації зональної або загальнооб'єктової системи оповіщення.

Система забезпечує раннє виявлення загроз виникнення НС через автоматичний контроль параметрів небезпечних факторів та реєстрацію перевищень граничних значень (докритичних і критичних рівнів).

Алгоритмічне забезпечення системи ґрунтується на аналізі відхилень поточних значень параметрів від граничних, де докритичним вважається відхилення на 10%, а критичним – на 20%.

Виявлення загроз виникнення НС та прогнозування сценаріїв їх розвитку здійснюється на основі даних від джерел первинної інформації, ручних сповіщувачів та засобів промислової автоматизації.

Відповідно до нормативних вимог [21], для малої гідроелектростанції з напірною земляною греблею визначено перелік параметрів моніторингу, наведений у табл. 4.1.

Таблиця 4.1 — Перелік контрольованих сигналів на гідротехнічних спорудах (у тому числі на гідроелектростанціях)

№ п/п	Контрольовані параметри
1	Деформації елементів споруди
2	Частоти обертання валу(ів) гідроагрегату(ів)
3	Осідання та горизонтальні зміщення елементів споруди
4	Рівні води у верхньому та нижньому б'єфах
5	Поява та рівні води у приміщеннях оглядової галереї, турбінному приміщенні, приміщеннях головних виводів генераторів
6	Наявність режиму пропуску повеневих та паводкових вод
7	Фізико-хімічні параметри води

У процесі проєктування допускається заміна автоматичних датчиків на ручні сповіщувачі. Таке рішення найчастіше застосовується для контролю параметрів, наведених у пунктах 1, 3, 4 та 7 таблиці.

Архітектура регіональної системи моніторингу стану ПНО розроблено відповідно до типової схеми, наведеної на рис.1.1.

У першому варіанті архітектури не передбачено:

- засобів захисту інформації;
- методів виявлення та корекції помилок передавання даних;
- алгоритмічного забезпечення прогнозування стану об'єкта;
- програмного забезпечення діагностичного моделювання.

4.4.2 Короткий опис другого варіанту регіональної системи моніторингу стану потенційно небезпечних об'єктів

Другий варіант системи моніторингу стану ПНО, представлений на рис. 2.4, базується на альтернативній архітектурі [15].

У другому варіанті системи моніторингу стану ПНО передбачено застосування технології периферійних обчислень (англ. Edge Computing) для реалізації локального рівня системи моніторингу стану ПНО. Зазначена технологія забезпечує підвищення ефективності хмарних обчислень через розподіл обчислювального навантаження на периферійні вузли. Архітектура передбачає синергетичну взаємодію периферійних та хмарних обчислювальних ресурсів у спосіб їх комплементарного використання.

Для забезпечення надійного збирання даних від давачів обґрунтовано застосування технології LoRaWAN, що характеризується такими перевагами:

- можливість агрегації даних від значної кількості пристроїв на великих територіях;
- тривалий термін експлуатації кінцевих пристроїв (до 10 років автономної роботи);
- ефективність розгортання та масштабування інфраструктури мережі.

Технологія LoRaWAN базується на широкопasmовій модуляції LoRa, що забезпечує радіус дії до 15 кілометрів, високу проникну здатність радіосигналу та стійкість до електромагнітних завад.

У другому варіанті системи моніторингу стану ПНО передбачено застосування технологій інтернету речей (англ. Internet of Things, IoT) для розроблення регіональної системи моніторингу стану потенційно небезпечних об'єктів [15]. На рис.2.3 представлено архітектуру взаємодії локальних серверів та систем зберігання даних в IoT-інфраструктурі на основі мікрохмарної архітектури.

В межах n -ї мікрохмари здійснюється збирання даних від давачів, їх опрацювання на сервері V_n та збереження у сховищі X_n . Результати моніторингу кожної мікрохмари після опрацювання на сервері V_n передаються до центрального хмарного сервера V_c та сховища X_c . Агрегація та опрацювання даних моніторингу від усіх мікрохмар на хмарних серверах забезпечує

формування комплексної оцінки стану контрольованих параметрів у масштабах регіону [15].

На відміну від першого варіанта архітектури, другий варіант системи моніторингу стану ПНО характеризується наявністю спеціалізованого програмного забезпечення для виявлення та корекції багатобітових помилок передавання даних.

Архітектура другого варіанта системи додатково включає підсистему прогнозування динаміки показників, що характеризують стан джерел небезпеки в межах ПНО.

4.4.3 Короткий опис третього варіанта регіональної системи моніторингу стану потенційно небезпечних об'єктів

Третій варіант архітектури регіональної системи моніторингу стану ПНО розроблено на основі модифікації другого варіанта. Зберігаючи базову архітектуру (рис. 2.4), система доповнена спеціалізованим програмним забезпеченням для діагностичного моделювання стану об'єкта, зокрема для моделювання процесів фільтрації через тіло земляної греблі.

У дисертаційному дослідженні розроблено методику експертного оцінювання запропонованої архітектури, алгоритмічного та програмного забезпечення системи моніторингу стану ПНО.

Достовірність результатів експертного оцінювання визначається рівнем професійної компетентності залучених фахівців. У дисертаційному дослідженні обґрунтовано критерії добору експертів, що забезпечують формування експертної групи з високим рівнем кваліфікації у предметній галузі.

У дисертаційному дослідженні застосовано метод аналізу ієрархій для агрегації різнорідних критеріїв якості регіональної системи моніторингу стану ПНО та формування інтегрального показника на основі уніфікованої шкали оцінювання.

Метод аналізу ієрархій ґрунтується на декомпозиції комплексної проблеми на простіші складові, що формують ієрархічну структуру. Відповідно

до принципу ієрархічної безперервності, елементи кожного рівня ієрархії підлягають попарному порівнянню відносно елементів вищого рівня.

Процес оцінювання здійснюється з використанням шкали відносної важливості, що забезпечує кількісне вираження результатів попарних порівнянь. У дисертаційному дослідженні критерії оцінювання варіантів проєктних рішень регіональної системи моніторингу стану ПНО агреговано в групові показники якості на основі диференційного та комплексного методів. Подальше попарне порівняння виконано з використанням сформованих групових критеріїв.

Сучасні стандарти оцінювання інформаційних систем визначають сукупність властивостей, що характеризують здатність системи виконувати встановлені функції відповідно до її призначення. Кількісні характеристики цих властивостей формують систему показників якості.

Для оцінювання якості програмного забезпечення у дисертаційному дослідженні застосовано багаторівневу модель, регламентовану стандартом ISO 9126. Модель визначає шість основних характеристик якості програмного забезпечення та їх атрибути з відповідними метриками для комплексного оцінювання інформаційної системи.

Основними критеріями оцінювання якості інформаційної системи визначено безпеку, достовірність та надійність функціонування.

Критерій безпеки характеризує здатність інформаційної системи забезпечувати конфіденційність та цілісність даних через механізми захисту від несанкціонованого доступу. Система безпеки реалізує комплекс заходів щодо захисту інформаційних ресурсів відповідно до встановлених рівнів доступу.

До достовірності функціонування слід зарахувати такий набір властивостей, який визначає безпомилковість перетворень інформації. Достовірність функціонування визначається достовірністю вихідної інформації.

Критерій достовірності функціонування характеризує сукупність властивостей системи, що забезпечують коректність перетворення даних та точність результатів опрацювання інформації. Достовірність функціонування

системи визначається як ступінь відповідності результатів опрацювання даних їх істинним значенням.

Критерій надійності характеризує здатність системи зберігати встановлені значення параметрів функціонування протягом визначеного часу за заданих умов експлуатації та режимів роботи.

Критерій ефективності регіональної системи моніторингу стану ПНО характеризує сукупність властивостей, що визначають здатність системи досягати встановлених цілей із заданими показниками якості за визначених умов функціонування. Ефективність системи оцінюється через ступінь відповідності її функціональних можливостей вимогам розв'язування поставлених задач моніторингу.

Для проведення експертного аналізу було сформовано експертну групу з п'яти фахівців. Добір експертів здійснено за такими критеріями:

- міждисциплінарна компетентність, що визначається науково-практичним досвідом;
- здатність до наукового прогнозування;
- фаховий рівень у предметній галузі дослідження.

Визначальними критеріями добору експертів встановлено:

- наявність вищої технічної освіти за відповідним напрямом;
- професійний досвід у галузі не менше ніж 10 років;
- здатність до формування неупереджених експертних оцінок.

Для оцінювання достовірності експертних суджень та визначення рівня узгодженості експертних оцінок у дисертаційному дослідженні застосовано метод ранжування критеріїв.

У дисертаційному дослідженні для порівняльного аналізу альтернативних варіантів регіональної системи моніторингу стану ПНО визначено такі критерії якості:

- безпека;
- достовірність;
- надійність;
- ефективність;
- зручність використання;

– вартість реалізації.

У дисертаційному дослідженні розроблено методику експертного оцінювання. Після визначення критеріїв якості системи здійснено експертне ранжування з урахуванням таких положень:

1. Процедура визначення вагових коефіцієнтів включає:

- встановлення рівня значущості параметрів через присвоєння рангів;
- верифікацію достовірності експертних оцінок;
- визначення пріоритетності параметрів на основі попарних порівнянь;
- опрацювання результатів та обчислення вагових коефіцієнтів.

2. Ваговий коефіцієнт i -го критерію якості системи моніторингу стану потенційно небезпечних об'єктів обчислюється методом розстановки пріоритетів. Пріоритетність показників якості визначається експертною комісією у складі п'яти фахівців за шкалою від 1 до n , де n - кількість оцінюваних критеріїв.

У процесі експертного оцінювання допускається присвоєння однакових рангів різним критеріям. Сумарний ранг критерію визначено за формулою 4.1:

$$R_i = \sum_{j=1}^N r_{ij} \quad (4.1)$$

де

r_{ij} – ранг i -го критерію, визначений j -тим експертом;

N – кількість експертів.

Середню суму рангів (T) розраховано за формулою 4.2:

$$T = \frac{R_{ij}}{n} \quad (4.2)$$

де n – кількість оцінюваних параметрів.

За результатами обчислень згідно з формулою (4.2) середня сума рангів становить: $T = 105/6 = 17,5$.

Відхилення сумарного рангу кожного критерію R_i від середнього значення рангів T визначається за формулою 4.3:

$$\Delta_i = R_i - T \quad (4.3)$$

Сумарне значення квадратів відхилень S за кожним параметром Δ_i^2 обчислено за формулою 4.4:

$$S = \sum_{i=0}^n \Delta_i^2 \quad (4.4)$$

Достовірність результатів ранжування критеріїв для подальшого опрацювання (4.4) встановлено через обчислення коефіцієнта конкордації Кендалла, що характеризує узгодженість експертних оцінок.

Коефіцієнт конкордації Кендалла обчислено за формулою 4.5:

$$W = \frac{12S}{N^2(n^3 - n)} \quad (4.5)$$

де

N – кількість експертів;

n – кількість критеріїв якості прогнозування.

У дисертаційному дослідженні інтерпретацію значень коефіцієнта конкордації здійснено за такими критеріями:

- значення коефіцієнта, що наближається до нуля, свідчить про відсутність узгодженості експертних оцінок;
- значення коефіцієнта, що наближається до одиниці, вказує на високий рівень узгодженості експертних оцінок;
- достовірність результатів експертного оцінювання забезпечується за умови, коли коефіцієнт конкордації перевищує значення 0,4.

На основі підтвердженої узгодженості експертних оцінок здійснено попарне порівняння критеріїв якості системи з урахуванням результатів їх ранжування.

Опрацювання результатів попарних порівнянь на підставі експертних анкет здійснено у такий спосіб:

1. Визначено кількість результатів попарних порівнянь для кожного з відношень: «менше», «дорівнює», «більше».

2. Обчислено добуток кількості результатів m_1 зі значенням «менше» на коефіцієнт «-1».

3. Обчислено добуток кількості результатів m_2 зі значенням «дорівнює» на коефіцієнт «0».

4. Обчислено добуток кількості результатів m_3 зі значенням «більше» на коефіцієнт «1».

5. Визначено суму отриманих добутків.

6. Встановлено підсумкове відношення за такими критеріями:

- «менше» – за від'ємного значення суми;
- «дорівнює» – за нульового значення суми;
- «більше» – за додатного значення суми.

7. Коефіцієнти переваг визначено за формулою:

$$a_{ij} = \begin{cases} 1,5 & \text{за умови } x_i > x_j, \\ 1,0 & \text{за умови } x_i = x_j, \\ 0,5 & \text{за умови } x_i < x_j. \end{cases} \quad (4.6)$$

Результати попарного порівняння критеріїв подано у табл. 4.4.

8. На основі числових даних a_{ij} (табл. 4.4) сформовано квадратну матрицю:

$$A = \begin{vmatrix} a_{11} & \cdots & a_{1i} \\ a_{i1} & \cdots & a_{ij} \end{vmatrix} \quad (4.7)$$

Матрицю коефіцієнтів переваг подано у табл. 4.5.

9. Наступним етапом обчислено пріоритетність вагових коефіцієнтів критеріїв якості проєктного рішення системи моніторингу стану ПНО. Розрахунок здійснено ітераційним методом:

$$b_i = \sum_{j=1}^N a_{ij}, \quad (4.8)$$

де

b_i – вагомість i -го критерію за результатами оцінок усіх експертів (визначається як сума значень коефіцієнтів a_{ij} , визначених експертами для i -го критерію);

N – кількість експертів.

Значення коефіцієнтів вагомості K'_i та K''_i (перша та друга ітерації) обчислено за формулою:

$$K'_i = \frac{b'_i}{\sum_{i=1}^n b'_i} \quad (4.9)$$

де:

$$b'_i = a_{i1}b_1 + a_{i2}b_2 + \dots + a_{in}b_n. \quad (4.10)$$

За умови виконання співвідношень:

$$\begin{cases} \sum_{i=1}^n K_i = 1; \\ \sum_{i=1}^n K'_i = 1. \end{cases} \quad (4.11)$$

10. На завершальному етапі дослідження обчислено значення цільових функцій для кожного варіанта архітектури системи моніторингу стану ПНО. Результати обчислень подано у табл. 4.7.

У табл. 4.7 подано значення цільових функцій S_1 , S_2 , S_3 для кожного варіанта архітектури системи, які обчислено за формулами (4.12) – (4.14):

$$S_1 = \sum_{j=1}^6 K_j \cdot B_{1j}, \quad (4.12)$$

$$S_2 = \sum_{j=1}^6 K_j \cdot B_{2j}, \quad (4.13)$$

$$S_3 = \sum_{j=1}^6 K_j \cdot B_{3j}. \quad (4.14)$$

Рациональною архітектурою системи моніторингу визначено варіант, що характеризується максимальним значенням цільової функції з множини S_1 , S_2 , S_3 .

4.5 Результати порівняльного аналізу можливих варіантів реалізації регіональної системи моніторингу стану потенційно небезпечних об'єктів

У дисертаційному дослідженні на основі експертного аналізу альтернативних варіантів архітектури системи моніторингу стану ПНО визначено критерії якості, наведені в табл. 4.2.

Таблиця 4.2 — Вибір критеріїв якості системи моніторингу стану ПНО

№	Назва критерія	Позначення
1	Безпека	x_1
2	Достовірність	x_2
3	Надійність	x_3
4	Ефективність	x_4
5	Зручність використання	x_5
6	Вартість реалізації	x_6

На основі результатів експертного опитування здійснено ранжування визначених критеріїв якості системи. Результати ранжування критеріїв якості системи подано в табл. 4.3.

Таблиця 4.3 — Результати ранжування критеріїв на підставі анкет експертів

n_i	Критерії оцінювання комплексного показника якості	Ранг критерію r_{ij} за оцінкою експертів ($N = 5$)					$R_i = \sum r_{ij}$	$T = \sum \frac{r_i}{n}$	$\Delta_i = R_i - T$	Δ_i^2
		Експерт, №								
		1	2	3	4	5				
		1	2	3	4	5				
1	Безпека	2	2	2	2	3	11	17,50	-6,50	42,25
2	Достовірність	4	4	3	3	2	16	17,50	-1,50	2,25
3	Надійність	1	1	1	1	1	5	17,50	-12,5	156,25
4	Ефективність	3	3	4	4	4	18	17,50	0,5	0,25
5	Зручність використання	6	6	5	6	5	28	17,50	10,50	110,25
6	Вартість реалізації	5	5	6	5	6	27	17,50	9,50	90,25
n_i	$n = 6$	21	21	21	21	21	105	T	Δ_i	$S = 401,50$

Опрацювання результатів експертного оцінювання здійснено з використанням математичного апарату, представленого формулами (4.1) – (4.4).

4.6 Визначення узгодженості думок експертів

На основі результатів ранжування (табл. 4.3) обчислено коефіцієнт конкордації Кендалла за формулою (4.5). Значення коефіцієнта конкордації дає змогу оцінити достовірність експертних суджень та рівень узгодженості експертних оцінок.

Обчислене значення коефіцієнта конкордації $W = 0,918$ згідно з формулою (4.5) суттєво перевищує граничне значення 0,4, що підтверджує високий рівень узгодженості та достовірності експертних оцінок. Це дало змогу продовжити дослідження з визначеною експертною групою без додаткового корегування її складу.

Після обробки анкет експертів, використовуючи формулу (4.6), було виконане попарне порівняння всіх параметрів. При цьому використовувались результати ранжування критеріїв. Результати попарного порівняння критеріїв наведено у табл. 4.4

Таблиця 4.4 — Результати попарного порівняння критеріїв

Пари критеріїв	Попарне порівняння критеріїв					Підсумки				
	Експерт, №					Кількість <	Кількість =	Кількість >	Підсумок	a_{ij}
	1	2	3	4	5					
1-2	<	>	>	>	<	2	0	3	>	0,5
1-3	<	<	<	<	<	5	0	0	<	1,5
1-4	<	<	<	<	<	5	0	0	<	1,5
1-5	<	<	<	<	<	5	0	0	<	1,5
1-6	<	<	<	<	<	5	0	0	<	1,5
2-3	<	<	<	<	<	5	0	0	<	1,5
2-4	<	<	<	<	<	5	0	0	<	1,5
2-5	<	<	<	<	<	5	0	0	<	1,5
2-6	<	<	<	<	<	5	0	0	<	1,5
3-4	>	>	<	<	>	2	0	3	>	0,5
3-5	<	<	<	<	<	5	0	0	<	1,5
3-6	<	<	<	<	<	5	0	0	<	1,5
4-5	<	<	<	<	<	5	0	0	<	1,5
4-6	<	<	<	<	<	5	0	0	<	1,5
5-6	>	>	<	>	<	2	0	3	>	0,5

На основі отриманих результатів сформовано матрицю коефіцієнтів переваг згідно з формулою (4.7), яку подано у табл. 4.5.

Таблиця 4.5 — Матриця коефіцієнтів переваг

Критерії, x_i		Критерії, x_j					
		x_1	x_2	x_3	x_4	x_5	x_6
Безпека	x_1	1,00	0,50	1,50	1,50	1,50	1,50
Достовірність	x_2	1,50	1,00	1,50	1,50	1,50	1,50
Надійність	x_3	0,50	0,50	1,00	0,50	1,50	1,50
Ефективність	x_4	0,50	0,50	1,50	1,00	1,50	1,50
Зручність використання	x_5	0,50	0,50	0,50	0,50	1,00	0,50
Вартість реалізації	x_6	0,50	0,50	0,50	0,50	1,50	1,00

Сформована матриця коефіцієнтів переваг становить основу для визначення вагових коефіцієнтів критеріїв якості проектних рішень регіональної системи моніторингу стану ПНО.

На наступному етапі дослідження обчислено вагові коефіцієнти критеріїв якості проектних рішень системи моніторингу стану потенційно небезпечних об'єктів з використанням формул (4.9), (4.10) та верифікацією умов (4.11). Результати обчислень подано у табл. 4.6.

Таблиця 4.6 — Розрахунок коефіцієнтів вагомостей критеріїв якості проектних рішень систем моніторингу стану потенційно небезпечних об'єктів

Критерії, x_i		Перший розрахунок		Перша ітерація			Друга ітерація			Підсумкові
		b_i	K_i	b'_i	K'_i	$\Delta K\%$	b''_i	K''_i	$\Delta K\%$	
Безпека	x_1	7,50	0,208	41,75	0,21	0,952	227,13	0,209	0,476	0,209
Достовірність	x_2	6,5	0,181	34,75	0,175	3,315	188,88	0,174	0,571	0,174
Надійність	x_3	8,5	0,236	49,75	0,251	5,976	272,88	0,251	0,000	0,251
Ефективність	x_4	5,5	0,153	28,75	0,145	5,229	157,13	0,145	0,000	0,145
Зручність використання	x_5	3,5	0,097	19,75	0,099	2,020	109,13	0,100	1,000	0,100
Вартість реалізації	x_6	4,5	0,125	23,75	0,120	4,000	130,88	0,121	0,826	0,121
Підсумок:		36	1,000	198,5	1,000	5,976	1086	1,000	1,000	1,000

Результати обчислень, подані у табл. 4.6, демонструють, що прийнятну збіжність вагових коефіцієнтів критеріїв якості проектних рішень регіональної системи моніторингу стану ПНО досягнуто на другій ітерації, коли відносна

різниця між послідовними значеннями коефіцієнтів стала нехтовно малою ($\epsilon < 2\%$). Для подальшого аналізу використано значення вагових коефіцієнтів, отримані після другої ітерації.

4.7 Порівняльний аналіз варіантів реалізації регіональної системи моніторингу стану потенційно небезпечних об'єктів

На завершальному етапі дослідження обчислено значення цільових функцій для кожного варіанта архітектури системи моніторингу стану ПНО. Результати обчислень подано у табл. 4.7.

Вихідними даними для обчислень слугували експертні оцінки, отримані в процесі анкетування за стобальною шкалою. Експертне оцінювання охоплювало всі критерії якості системи моніторингу стану потенційно небезпечних об'єктів з подальшим визначенням середніх значень оцінок та здійсненням попарного порівняння варіантів реалізації регіональної системи моніторингу стану ПНО.

На наступному етапі обчислено вагові коефіцієнти b_i для кожного i -го критерію як суму коефіцієнтів a_{ij} , визначених за результатами експертного оцінювання.

Коефіцієнти B_{ij} визначено як відношення значень b_i до їх суми, а коефіцієнти K'_i обчислено за формулами (4.9), (4.10) з урахуванням умов (4.11).

Таблиця 4.7 — Розрахунок значення цільових функцій якості різних проєктних рішень систем моніторингу стану потенційно небезпечних об'єктів.

Критерії		Варіант	Оцінки експертів					Серед. оцінка	Попарне порівняння варіантів				b_i	B_{ij}	K'_i	$K_i B_{1j}$	$K_i B_{2j}$	$K_i B_{3j}$
			1	2	3	4	5			B_1	B_2	B_3						
Безпека	x_1	1	15	25	35	20	30	25,0	B_1	0	0,5	0,5	1	0,167	0,209	0,035	↓	↓
		2	35	45	40	40	45	41,0	B_2	1,5	0	0,5	2	0,333	0,209	↓	0,070	↓
		3	40	55	45	85	93	63,6	B_3	1,5	1,5	0	3	0,500	0,209	↓	↓	0,104
Підсумкові значення :												6	1,000	...	↓	↓	↓	
Достовірність	x_2	1	15	20	25	15	35	22,0	B_1	0	0,5	0,5	1	0,167	0,174	0,029	↓	↓
		2	20	25	35	30	45	31,0	B_2	1,5	0	0,5	2	0,333	0,174	↓	0,058	↓
		3	25	35	45	20	55	36,0	B_3	1,5	1,5	0	3	0,500	0,174	↓	↓	0,087
Підсумкові значення :												6	1,000	...	↓	↓	↓	
Надійність	x_3	1	35	45	35	40	42	39,4	B_1	0	0,5	0,5	1	0,167	0,251	0,042	↓	↓
		2	35	45	35	40	43	39,6	B_2	1,5	0	0,5	2	0,333	0,251	↓	0,084	↓
		3	35	45	35	75	43	46,6	B_3	1,5	1,5	0	3	0,500	0,251	↓	↓	0,126
Підсумкові значення :												6	1,000	...	↓	↓	↓	
Ефективність	x_4	1	70	65	75	80	77	73,4	B_1	0	1,5	0,5	2	0,333	0,145	0,048	↓	↓
		2	60	60	55	70	57	60,4	B_2	0,5	0	0,5	1	0,167	0,145	↓	0,024	↓
		3	75	75	80	85	87	80,4	B_3	1,5	1,5	0	3	0,500	0,145	↓	↓	0,072
Підсумкові значення :												6	1,000	...	↓	↓	↓	
Зручність використання	x_5	1	65	70	75	62	75	69,4	B_1	0	1,5	1,5	3	0,500	0,100	0,050	↓	↓
		2	55	45	50	60	54	52,8	B_2	0,5	0	0,5	1	0,167	0,100	↓	0,016	↓
		3	60	65	65	55	63	61,6	B_3	0,5	1,5	0	2	0,333	0,100	↓	↓	0,033
Підсумкові значення :												6	1,000	...	↓	↓	↓	
Вартість реалізації	x_6	1	85	85	80	80	87	83,4	B_1	0	1,5	0,5	2	0,333	0,121	0,040	↓	↓
		2	80	75	70	70	75	74,0	B_2	0,5	0	0,5	1	0,167	0,121	↓	0,020	↓
		3	90	95	97	70	93	89,0	B_3	1,5	1,5	0	3	0,500	0,121	↓	↓	0,061
Підсумкові значення :												6	1,000	Сума:	0,244	0,272	0,483	
Значення цільових функцій:															S_1	S_2	S_3	

У табл. 4.7 подано значення цільових функцій S_1 , S_2 , S_3 для кожного варіанта архітектури системи, які обчислено за формулами (4.12) – (4.14):

$$S_1 = \sum_{j=1}^6 (K_j \cdot B_{1j}) = 0,244, \quad (4.15)$$

$$S_2 = \sum_{j=1}^6 (K_j \cdot B_{2j}) = 0,272, \quad (4.16)$$

$$S_3 = \sum_{j=1}^6 (K_j \cdot B_{3j}) = 0,483. \quad (4.17)$$

За результатами експертного оцінювання встановлено, що третій варіант реалізації регіональної системи моніторингу стану ПНО демонструє найвищу ефективність.

Зазначений варіант базується на архітектурі, зображеній на рис. 2.4, та передбачає організацію взаємодії локальних серверів і систем зберігання даних у межах мікрохмарної архітектури.

Алгоритмічне забезпечення обраного варіанта системи містить:

- підсистему виявлення та виправлення багатобітових помилок під час передавання даних;
- підсистему прогнозування змін стану потенційно небезпечних об'єктів;
- підсистему діагностичного моделювання стану об'єкта моніторингу.

Виконане дослідження ефективності альтернативних варіантів архітектури та алгоритмічного забезпечення системи моніторингу стану потенційно небезпечних об'єктів регіонального рівня підтвердило основну гіпотезу дисертаційного дослідження.

4.8 Висновки до розділу 4

1. Розроблення архітектури, алгоритмічного та програмного забезпечення регіональних систем моніторингу ПНО необхідно реалізовувати у спосіб неперервного контролю коректності проєктних рішень та верифікації отриманих результатів.

2. Верифікацію проєктних рішень та результатів розроблення систем моніторингу потенційно небезпечних об'єктів доцільно реалізовувати із застосуванням методології експертного оцінювання, яка забезпечує формування як якісних, так і кількісних показників ефективності.

3. Експертне оцінювання, проведене в межах дисертаційного дослідження, підтвердило обґрунтованість запропонованих методів та алгоритмів для типових архітектур регіональних систем моніторингу потенційно небезпечних об'єктів. Визначено доцільність таких проєктних рішень:

- реалізація регіональних систем моніторингу у спосіб застосування IoT-мереж;

- впровадження програмованих давачів як базових вузлів збору первинної інформації щодо параметрів стану джерел небезпеки;

- забезпечення завадостійкої передачі даних в IoT-мережі за умов індустриальних завад через імплементацію модифікованого методу виявлення та виправлення багатобітових помилок;

- інтеграція аналітичної предиктивної підсистеми на всіх рівнях моніторингу для прогнозування динаміки параметрів стану джерел небезпеки та моделювання процесів деградації стану об'єктів з метою локалізації потенційних дефектів.

4. Розроблена методика експертного оцінювання якості алгоритмічного забезпечення та програмного забезпечення характеризується високим рівнем формалізації та можливістю реалізації засобами довільної мови програмування. Застосування методики забезпечує перехід від оцінювання якісних характеристик до кількісних показників.

ВИСНОВКИ

У дисертаційній роботі вирішено актуальне науково-технічне завдання підвищення ефективності процесу моніторингу стану потенційно небезпечних об'єктів на основі імплементації в склад регіональної системи моніторингу стану потенційно небезпечних об'єктів аналітичних предиктивних підсистем на всіх рівнях, а саме: об'єктовому, місцевому та регіональному у спосіб розроблення спеціалізованого алгоритмічного та програмного забезпечення, що реалізує заводо захищений обмін даними в межах системи моніторингу, забезпечує прогнозування змін ключових показників джерел небезпеки та дозволяє локалізувати розташування можливих дефектів в межах об'єкта підвищеної небезпеки. За результатами дослідження сформульовані наступні висновки:

1. Сучасний стан розроблення та впровадження систем моніторингу стану ПНО характеризується суттєвими недоліками: відсутністю єдиної державної системи моніторингу та фрагментарністю наявних регіональних, місцевих та об'єктових систем. Наявні системи обмежуються функціями параметричного контролю джерел небезпеки, що унеможливорює своєчасне ухвалення управлінських рішень щодо запобігання можливості НС.

2. Доцільно реалізовувати регіональні системи моніторингу стану ПНО у спосіб імплементації модифікованого алгоритмічного та програмного забезпечення та архітектури IoT-мереж із топологією «зірка», де функції збору первинної інформації виконуються програмованими давачами.

3. Обґрунтовано необхідність забезпечення заводостійкої передачі даних в IoT-мережі за умов індустриальних завод через імплементацію модифікованого методу виявлення та виправлення багатобітових помилок, програмна реалізація якого забезпечує виявлення та виправлення до 8 помилок під час передавання окремого байту в межах повідомлення розміром 24 байти. Експериментальна верифікація в умовах регіональної системи моніторингу потенційно небезпечних об'єктів, розподілених на значній території за наявності індустриальних електромагнітних завод, підтвердила, що впровадження розробленого алгоритмічного забезпечення зменшує кількість помилок передавання на 38 відсотків від їх загальної кількості.

4. Впровадження аналітичної предиктивної підсистеми на об'єктовому рівні регіональної системи моніторингу потенційно небезпечних об'єктів забезпечує прогнозування змін параметрів стану джерел небезпеки на період 3-4 часових відліків (за тривалості відліку 10, 20, 30 або 60 хвилин), що уможливорює завчасне виявлення наближення контрольованих параметрів до докритичних або критичних рівнів. За умов повільної динаміки змін параметрів стану джерела небезпеки при прогнозуванні раціональним є застосування лінійної моделі регресії, що забезпечує точність прогнозування до 1 відсотка при виборці з 10 вимірювань, тоді як у разі значної динаміки змін параметрів використання поліноміальної апроксимації підвищує точність короткострокового прогнозування до 0,7 відсотка при виборці з 15 вимірювань. Експериментальна верифікація підтвердила, що інтеграція аналітичної предиктивної підсистеми на об'єктовому рівні забезпечує зменшення простоїв основного технологічного обладнання на 23 відсотки через своєчасне планування ремонтних робіт за результатами моніторингу таких параметрів: рівень вібрації підшипників генератора, турбіни та редуктора.

5. На місцевому та регіональному рівнях підсистема предиктивної аналітики, окрім прогнозування, забезпечує моделювання стану ПНО, зокрема земляних напірних гідроспоруд, у спосіб застосування модифікованого методу сіток. Розроблене алгоритмічне забезпечення та програмне забезпечення реалізують на об'єктовому рівні короткострокове прогнозування параметрів стану джерел небезпеки, а на місцевому та регіональному рівнях – моделювання деструктивних процесів із формуванням кількісних та якісних показників оцінювання можливості виникнення НС та локалізації можливих дефектів.

6. Розроблена методика експертного оцінювання якості алгоритмічного та програмного забезпечення регіональних систем моніторингу стану ПНО характеризується високим рівнем формалізації, забезпечує перехід від якісних до кількісних показників оцінювання та уможливорює реалізацію засобами довільної мови програмування для верифікації проєктних рішень.

Розроблені теоретичні положення, методи, алгоритмічне та програмне забезпечення використані при:

– проектуванні руслової мала гідроелектростанції у с. Довге Дрогобицького району Львівської області, що підтверджено «Довідка про впровадження результатів дисертаційного дослідження, видана ТОВ «БАЛФОРД УКРАЇНА» №02-11-24 від. 05.11.2024р.»;

– розробці проекту дериваційної малої гідроелектростанції у с. Рибник Дрогобицького району Львівської області, що підтверджено «Довідка про впровадження результатів дисертаційного дослідження, видана ТОВ ГУТАРЬОВ ТА ПАРТНЕРИ УКРАЇНА» №49-11-24 від. 06.11.2024р.»;

– виконанні науково-дослідної роботи «Теоретичні та практичні аспекти технології Internet of Everything» (державний реєстраційний номер 0123U104930), що підтверджено актом про використання результатів дисертаційного дослідження.

СПИСОК ЛІТЕРАТУРИ

- [1] Яковенко Вадим Олександрович та Ульяновська Юлія Вікторівна. “АНАЛІЗ ЗАДАЧІ ПОБУДОВИ АВТОМАТИЗОВАНОЇ СИСТЕМИ ВИЯВЛЕННЯ ТА ПОПЕРЕДЖЕННЯ НАДЗВИЧАЙНИХ СИТУАЦІЙ НА ПІДПРИЄМСТВАХ ПІДВИЩЕНОЇ НЕБЕЗПЕКИ”. В: *Проблеми інформаційних технологій* (23 2018), с. 134—139. ISSN: 1998-7005. DOI: 10.35546/pit.v0i23.208.
- [2] Іванець Г.В. “АНАЛІЗ СТАНУ ТЕХНОГЕННОЇ, ПРИРОДНОЇ ТА СОЦІАЛЬНОЇ НЕБЕЗПЕКИ АДМІНІСТРАТИВНО-ТЕРИТОРІАЛЬНИХ ОДИНИЦЬ УКРАЇНИ НА ОСНОВІ ДАНИХ МОНІТОРИНГУ”. В: *Збірник наукових праць Харківського національного університету Повітряних Сил* 48 (3 2016), с. 142—145. ISSN: 2073-7378.
- [3] В. М. Шоботов. *ЦИВІЛЬНА ОБОРОНА*. 2-е вид. Центр навчальної літератури, 2006, с. -438. ISBN: 9663641665.
- [4] Держспоживстандарт України. *ДК 019:2010. Класифікатор надзвичайних ситуацій*. 2010. URL: <https://zakon.rada.gov.ua/rada/show/va457609-10/conv#Text>.
- [5] Jianjun Su та ін. “A multi-setpoint delay-timer alarming strategy for industrial alarm monitoring”. В: *Journal of Loss Prevention in the Process Industries* 54 (лют. 2018). DOI: 10.1016/j.jlp.2018.02.004.
- [6] П П Кропотов, В В Бегун та В Ф Гречанінов. “СТВОРЕННЯ СУЧАСНОЇ СИСТЕМИ МОНІТОРИНГУ БЕЗПЕКИ – АКТУАЛЬНА ДЕРЖАВНА ТА НАУКОВА ЗАДАЧА”. В: *Системи обробки інформації* 136 (11 2015), с. 199—206. ISSN: 1681-7710. URL: http://nbuv.gov.ua/UJRN/soi_2015_11_47.
- [7] М.В. Плахотний та А.Ю. Гричина. “ОСОБЛИВОСТІ ПОБУДОВИ СИСТЕМ ДИСТАНЦІЙНОГО МОНІТОРИНГУ ПАРАМЕТРІВ ОБЛАДНАННЯ”. В: *Вісник Національного університету ”Львівська політехніка”*. (2011), с. 125—133. ISSN: 0321-0499. URL: <http://ena.lp.edu.ua>.

- [8] K. V. Antipov, M. P. Maslakov та K. I. Yurenko. “Improvement of the Automated Control Systems for the Development of the Metallurgy”. В: *Procedia Engineering* 129 (січ. 2015), с. 1010—1014. ISSN: 1877-7058. DOI: 10.1016/J.PROENG.2015.12.164.
- [9] Magiswary Dorasamy, Murali Raman та Maniam Kaliannan. “Integrated community emergency management and awareness system: A knowledge management system for disaster support”. В: *Technological Forecasting and Social Change* 121 (серп. 2017), с. 139—167. ISSN: 0040-1625. DOI: 10.1016/J.TECHFORE.2017.03.017.
- [10] Дуэль Михаил Александрович, Канюк Геннадий Иванович та Фурсова Татьяна Николаевна. “Концептуальные основы построения АСУ сложными энергообъектами”. В: *Eastern-European Journal of Enterprise Technologies* 6 (3(54) 2011), с. 4—11. DOI: 10.15587/1729-4061.2011.2240. URL: <https://journals.urau.ua/eejet/article/view/2240>.
- [11] V. Sokolovskyi та E. Zharikov. “ARCHITECTURAL SOLUTION FOR THE DISTRIBUTION OF SOFTWARE AND HARDWARE SYSTEMS FOR MONITORING POTENTIALLY UNSAFE OBJECTS”. В: *International Journal of GEOMATE* 25 (109 2023), с. 141—148. DOI: 10.21660/2023.109.m2314.
- [12] V. Sokolovskyi, E. Zharikov та S. Telenyk. “DEVELOPMENT OF THE METHOD OF DETECTING AND CORRECTING DATA TRANSMISSION ERRORS IN IOT SYSTEMS FOR MONITORING THE STATE OF OBJECTS”. В: *Eastern-European Journal of Enterprise Technologies* 1 (9(127) 2024), с. 22—33. DOI: 10.15587/1729-4061.2024.298476.
- [13] V. Sokolovskyi, E. Zharikov та S. Telenyk. “Software and algorithmic support as part of regional systems for monitoring the state of objects for calculation of filtration through earthen hydraulic structures”. В: *Machinery and Energetics* 15 (2 2024), с. 130—144. DOI: 10.31548/machinery/2.2024.130.
- [14] V. Sokolovskyi, E. Zharikov та S. Telenyk. “USING EXPERT EVALUATION FOR SELECTING AN ARCHITECTURAL SOLUTION

- FOR A SPECIALIZED SOFTWARE SYSTEM THAT MONITORS THE STATE OF POTENTIALLY HAZARDOUS FACILITIES”. В: *Eastern-European Journal of Enterprise Technologies* 5 (3-131 2024), с. 27—40. DOI: 10.15587/1729-4061.2024.312886.
- [15] Соколовський Владислав Володимирович та Жаріков Едуард В’ячеславович. “Архітектура програмно-апаратної системи моніторингу стану об’єктів підвищеної небезпеки з можливістю прогнозування виникнення надзвичайної ситуації”. В: *Всеукраїнська науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології»(SoftTech-2022)*. 2022, с. 64—68. URL: https://drive.google.com/file/d/1-kJ75f9HKjfQ4pL7SvNf_uW4NI2eb9HL.
- [16] Соколовський Владислав Володимирович та Жаріков Едуард В’ячеславович. “АРХІТЕКТУРНЕ РІШЕННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПРОГРАМОВАНИХ ДАВАЧІВ ІНФОРМАЦІЇ ДЛЯ ПОБУДОВИ РЕГІОНАЛЬНИХ ІОТ СИСТЕМ МОНІТОРИНГУ СТАНУ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ОБ’ЄКТІВ”. В: *VI Міжнародна науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології (SOFT TECH-2024)»*. 2024, с. 128—135. URL: <https://drive.google.com/file/d/18bZ9QBure7U08rbqiHmxWglTrK1C9D4L/view>.
- [17] Соколовський Владислав Володимирович та Жаріков Едуард В’ячеславович. “Прогнозування в системах моніторингу стану об’єктів підвищеної небезпеки регіонального рівня”. В: *VII Міжнародна науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології (SOFT TECH-2024)»*. 2024, с. 101—106. URL: <https://drive.google.com/file/d/1O70Ysxe-z3SS82UqEaBdEXR2VdRW3NwG>.
- [18] Соболев Олександр Миколайович. “НОРМАТИВНО-ПРАВОВЕ РЕГУЛЮВАННЯ МОНІТОРИНГУ НАДЗВИЧАЙНИХ СИТУАЦІЙ В

- УКРАЇНИ”. В: *Національний університет цивільного захисту України* (1 2014). URL: <http://repositsc.nuczu.edu.ua/handle/123456789/2334>.
- [19] Viktoriia Shvedun, Oleksandr Ihnatiev та Olena Postupna. “Problems of functioning of public administration mechanisms concerning the operational monitoring of the state of potentially dangerous objects”. В: *Pressing Problems of Public Administration* (2 2022). ISSN: 1684-8489. DOI: 10.26565/1684-8489-2022-2-04.
- [20] *Кодекс цивільного захисту України від 27 травня 2022 р. № 5403-VI*. URL: <http://zakon4.rada.gov.ua/laws/show>.
- [21] Український науково-дослідний інститут цивільного захисту (37814631) Версія №1 Статус (поточна). *ДБН В.2.5-76:2014 "Автоматизовані системи раннього виявлення загрози виникнення надзвичайних ситуацій та оповіщення населення"*. 2023. URL: https://e-construction.gov.ua/laws_detail/3200403474810405979?doc_type=2.
- [22] K. Vasiutynska та S. Barbashev. “Risk assessment of emergencies in the Ukraine regions”. В: *Ecological Sciences* 32 (5 2020). ISSN: 23069716. DOI: 10.32846/2306-9716/2020.eco.5-32.8.
- [23] V.V. Prachyk та O.M. Liashenko. “DEVELOPMENT OF AN INFORMATION SYSTEM FOR MONITORING NATURAL DISASTERS USING OBJECT-ORIENTED METHODOLOGY AND JAVA SE 11 TECHNOLOGY”. В: *Scientific notes of Taurida National V.I. Vernadsky University. Series: Technical Sciences* (4 2021). ISSN: 26635941. DOI: 10.32838/2663-5941/2021.4/22.
- [24] Vishal Dutt та Shweta Sharma. “Artificial intelligence and technology in weather forecasting and renewable energy systems: emerging techniques and worldwide studies”. В: *Artificial Intelligence for Renewable Energy systems* (січ. 2022), с. 189—207. DOI: 10.1016/B978-0-323-90396-7.00009-2.
- [25] Raul Sidnei Wazlawick. *Object-Oriented Analysis and Design for Information Systems: Modeling with UML, OCL, and IFML*. 2014. DOI: 10.1016/C2012-0-06942-6.

- [26] Дмитро Кирийчук. “МОДЕЛЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ МОНІТОРИНГУ НАДЗВИЧАЙНИХ СИТУАЦІЙ”. В: *Problems of information technologies* (25 2019). ISSN: 19987005. DOI: 10.35546/2313-0687.2019.25.66-75.
- [27] A. Ignatiev. “ANALYSIS AND EVALUATION OF THE ORGANIZATIONAL MECHANISM OF PROVIDING THE STATE MANAGEMENT OF MONITORING OF THE CONDITION OF POTENTIALLY DANGEROUS OBJECTS”. В: *Law and public administration* (3 2020). ISSN: 26635348. DOI: 10.32840/pdu.2020.3.15.
- [28] Gartner. *Gartner Glossary*. 2020. URL: <https://www.gartner.com/en/information-technology/glossary>.
- [29] Hung Lehong та Jackie Fenn. “Key Trends to Watch in Gartner 2012 Emerging Technologies Hype Cycle”. В: *Forbes* (2012).
- [30] Kevin Ashton. *That 'Internet of Things' Thing - 2009-06-22 - Page 1 - RFID Journal*. 2009.
- [31] Kaljot Sharma та ін. “A Disaster Management Framework Using Internet of Things-Based Interconnected Devices”. В: *Mathematical Problems in Engineering* 2021 (2021). ISSN: 15635147. DOI: 10.1155/2021/9916440.
- [32] Li Da Xu, Wu He та Shancang Li. *Internet of things in industries: A survey*. 2014. DOI: 10.1109/TII.2014.2300753.
- [33] Shashi Rekha та ін. “Study of security issues and solutions in Internet of Things (IoT)”. В: *Materials Today: Proceedings* 80 (2023). ISSN: 22147853. DOI: 10.1016/j.matpr.2021.07.295.
- [34] Muhammad Alam та ін. “Orchestration of Microservices for IoT Using Docker and Edge Computing”. В: *IEEE Communications Magazine* 56 (9 2018). ISSN: 15581896. DOI: 10.1109/MCOM.2018.1701233.
- [35] Flavio Bonomi та ін. “Fog computing and its role in the internet of things”. В: *MCC'12 - Proceedings of the 1st ACM Mobile Cloud Computing Workshop*. 2012. DOI: 10.1145/2342509.2342513.

- [36] Breno Costa та ін. *Orchestration in Fog Computing: A Comprehensive Survey*. 2023. DOI: 10.1145/3486221.
- [37] Hany F. Atlam, Robert J. Walters та Gary B. Wills. *Fog computing and the internet of things: A review*. 2018. DOI: 10.3390/bdcc2020010.
- [38] Naser Hossein Motlagh та ін. *Internet of things (IoT) and the energy sector*. 2020. DOI: 10.3390/en13020494.
- [39] Vikas Hassija та ін. *A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures*. 2019. DOI: 10.1109/ACCESS.2019.2924045.
- [40] Л.О. Уривський, С.О. Осипчук та О.В. Будішевський. “FOG-МЕРЕЖА З АДАПТИВНОЮ СИСТЕМОЮ УПРАВЛІННЯ”. В: *Інфокомунікаційні та комп’ютерні технології* 1 (03 2022). ISSN: 2788-5518. DOI: 10.36994/2788-5518-2022-01-03-05.
- [41] V. V. Bychkov. “Research of protocols and technologies data transmissions in IoT networks”. В: *Connectivity* 169 (3 2024). ISSN: 24129070. DOI: 10.31673/2412-9070.2024.032831.
- [42] IoT Solutions Malta. *Sigfox. LoRa. NB-IoT. Easy guide to who does it best*. URL: <https://www.iotsolutions.com.mt/post/sigfox-vs-lora-vs-nb-iot-who-s-doing-it-best>.
- [43] K. F. Muteba, K. Djouani та T. Olwal. “5G NB-IoT: Design, Considerations, Solutions and Challenges”. В: *Procedia Computer Science*. Т. 198. 2021. DOI: 10.1016/j.procs.2021.12.214.
- [44] Anthony S. Deese та Julian Daum. “Application of ZigBee-Based Internet of Things Technology to Demand Response in Smart Grids”. В: *IFAC-PapersOnLine*. Т. 51. 2018. DOI: 10.1016/j.ifacol.2018.11.675.
- [45] TÜV Rheinland. *Technical Overview of LoRa and LoRaWAN*. 2015. URL: <https://z-library.sk/book/18827766/a98300/technical-overview-of-lora-and-lorawan.html?dsource=recommend>.

- [46] Держспоживстандарт. ДСТУ 7136:2009 Безпека у надзвичайних ситуаціях. Моніторинг потенційно небезпечних об'єктів. Порядок проведення. Лип. 2010. URL: <https://ua-s.org/katalog-normativnih-dokumentiv/search-by-result-parameters/3749>.
- [47] C. E. Shannon. “A mathematical theory of communication”. В: *Bell System Technical Journal* 27 (3 1948), с. 379—423. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb01338.x.
- [48] Leonid Uryvsky та Anastasia Kosogor. “COMPREHENSIVE ANALYSIS OF THE HAMMING CODE PROPERTIES BASED ON SUPERPOSITION OF THE CODING, NOISE IMMUNITY AND INFORMATION THEORIES”. В: *Інфокомунікаційні та комп'ютерні технології* (2023), с. 85—93. ISSN: 27885518. DOI: 10.36994/2788-5518-2023-01-05-10.
- [49] Вячеслав Сергійович Василенко, Андрій Вадимович Чунарьов та Анна Вадимівна Чунарьова. “Analysis of the existing coding methods for ensuring the integrity of information in modern ICNS”. В: *Ukrainian Information Security Research Journal* 14 (3 (56) 2012). ISSN: 2221-5212. DOI: 10.18372/2410-7840.14.3359.
- [50] АЛІНА ДАВЛЕТОВА. “ПОБУДОВА КОДІВ ХЕММІНГА В СКІНЧЕННИХ ПОЛЯХ ГАЛУА”. В: *Herald of Khmelnytskyi National University. Technical sciences* 333 (2 квіт. 2024), с. 28—34. ISSN: 2307-5732. DOI: 10.31891/2307-5732-2024-333-2-4.
- [51] Т.В. Martyniuk та О.Ю. Voinalovych. “Classification model of digital coding methods”. В: *Optoelectronic Information-Power Technologies* 47 (1 черв. 2024), с. 42—49. ISSN: 2311-2662. DOI: 10.31649/1681-7893-2024-47-1-42-49.
- [52] Артак Апаратович Хемчян. “Data sharing based on error-correcting codes”. В: *Ukrainian Scientific Journal of Information Security* 22 (3 2016). ISSN: 2225-5036. DOI: 10.18372/2225-5036.22.11097.
- [53] W. Wesley (William Wesley) Peterson. *Error-correcting codes*. За ред. E. J. Weldon. 2nd ed. MIT Press, 1972. ISBN: 0262160390.

- [54] E. F. Assmus. *Designs and their codes*. За ред. J. D. Key. Cambridge University Pr., 1992. ISBN: 0521458390.
- [55] Yanting Wang, Mingwei Tang та Zhen Wang. “High-capacity adaptive steganography based on LSB and Hamming code”. B: *Optik* 213 (2020). ISSN: 00304026. DOI: 10.1016/j.ijleo.2020.164685.
- [56] Hongbo Xie, Yincheng Qi та Farah Qasim Ahmed Alyousuf. “Designing an ultra-efficient Hamming code generator circuit for a secure nano-telecommunication network”. B: *Microprocessors and Microsystems* 103 (2023). ISSN: 01419331. DOI: 10.1016/j.micpro.2023.104961.
- [57] Xinhao Sun та ін. “A memristor-based in-memory computing network for hamming code error correction”. B: *IEEE Electron Device Letters* 40 (7 2019). ISSN: 15580563. DOI: 10.1109/LED.2019.2917944.
- [58] Achmad Fauzi та Robbi Rahim. “Bit Error Detection and Correction with Hamming Code Algorithm”. B: *International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET)* 3 (1 2017).
- [59] Anil Kumar Singh. “Error detection and correction by hamming code”. B: *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*. IEEE, 2016, c. 35—37. ISBN: 9781509004676. DOI: 10.1109/ICGTSPICC.2016.7955265.
- [60] MALVIYA VINITA та ін. “A HAMMING CODE BASED SINGLE BIT ERROR DETECTION AND CORRECTION FOR 128 OR 256 BITS”. B: *i-manager's Journal on Digital Signal Processing* 5 (3 2017), c. 1. ISSN: 2321-7480. DOI: 10.26634/jdp.5.3.13928.
- [61] U. K. Kumar та B. S. Umashankar. “Improved hamming code for error detection and correction”. B: *2007 2nd International Symposium on Wireless Pervasive Computing*. 2007. DOI: 10.1109/ISWPC.2007.342654.
- [62] Caleb Hillier та Vipin Balyan. “Error Detection and Correction On-Board Nanosatellites Using Hamming Codes”. B: *Journal of Electrical and Computer Engineering* 2019 (2019). ISSN: 20900155. DOI: 10.1155/2019/3905094.

- [63] Norberto; Florez Hector Novoa. “Error Detection and Correction Using Hamming Code”. B: *Revista Vínculos Vol. 9 Número 2 9* (2 2012).
- [64] Irene Ndanu John. “Error Detection and Correction Using Hamming and Cyclic Codes in a Communication Channel”. B: *Pure and Applied Mathematics Journal* 5 (6 2016). ISSN: 2326-9790. DOI: 10.11648/j.pamj.20160506.17.
- [65] Vishal Badole ta Amit Udawat. “Design of Multidirectional Parity Code Using Hamming Code Technique for Error Detection and Correction”. B: *Paripex - Indian Journal Of Research* 3 (5 січ. 2012), с. 79—81. ISSN: 22501991. DOI: 10.15373/22501991/MAY2014/27.
- [66] Yun Fu Shen ta Lei Pan. “Principle and method of the error detection and correction of ternary hamming codes”. B: *Jisuanji Xuebao/Chinese Journal of Computers* 38 (8 2015). ISSN: 02544164. DOI: 10.11897/SP.J.1016.2015.01648.
- [67] R. W. Hamming. “Error detecting and error correcting codes”. B: *Bell System Technical Journal* 29 (2 1950), с. 147—160. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1950.tb00463.x.
- [68] Granville Tunncliffe Wilson. “Time Series Analysis: Forecasting and Control, 5th Edition, by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung, 2015. Published by John Wiley and Sons Inc., Hoboken, New Jersey, pp. 712. ISBN: 978-1-118-67502-1”. B: *Journal of Time Series Analysis* 37 (5 2016). ISSN: 0143-9782. DOI: 10.1111/jtsa.12194.
- [69] Elena Kushchenko ta Konstantin Danilenko. “ANALYTICAL PROGNOSTICATION OF BUILDING BUSINESS IS IN UKRAINE”. B: *Economy and Society* (20 2019). DOI: 10.32782/2524-0072/2019-20-95.
- [70] Nina Golyandina, Vladimir Nekrutkin ta Anatoly A Zhigljavsky. *Analysis of Time Series Structure*. 2001. DOI: 10.1201/9780367801687.
- [71] N. Golyandina ta E. Osipov. “The ”Caterpillar”-SSA method for analysis of time series with missing values”. B: *Journal of Statistical Planning and Inference* 137 (8 2007). ISSN: 03783758. DOI: 10.1016/j.jspi.2006.05.014.

- [72] D. M. Karpa, I. H. Tsmots та Yu. V. Orotiak. “Нейромережеві засоби прогнозування споживання енергоресурсів”. В: *Scientific Bulletin of UNFU* 28 (5 2018). ISSN: 1994-7836. DOI: 10.15421/40280529.
- [73] О. С. Пермякова та Сергій Олексійович Семеріков. *Застосування нейронних мереж у задачах прогнозування*. Листоп. 2008. DOI: 10.31812/0564/923.
- [74] О. В. Піскунова та С. С. Савіна. “Визначення оптимальних параметрів нейромережі у задачах бінарної класифікації при використанні малих обсягів даних”. В: *Проблеми сучасних трансформацій. Серія: економіка та управління* (6 2022). DOI: 10.54929/2786-5738-2022-6-11-01.
- [75] Олег Бурлєєв, Олег Василенко та Ростислав Іваненко. “ЕФЕКТИВНІСТЬ ВИКОРИСТАННЯ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ В ЕКОНОМІЦІ”. В: *Економіка та суспільство* (31 2021). DOI: 10.32782/2524-0072/2021-31-27.
- [76] Г.В. Іванець та М.Г. Іванець. “Алгоритм підвищення точності прогнозування процесу виникнення надзвичайних ситуацій на основі регресійних моделей”. В: *Наука і техніка Повітряних Сил Збройних Сил України* (1(34) 2019). ISSN: 2223456X. DOI: 10.30748/nitps.2019.34.16.
- [77] Andrij Olijnyk та ін. “Modeling of the filtration processes in a rectangular area soils using the darcy”. В: *Eastern-European Journal of Enterprise Technologies* 6 (10-90 2017). ISSN: 17294061. DOI: 10.15587/1729-4061.2017.116114.
- [78] Olena M. Hladka. “Systematic approach to mathematical modeling of filtration processes in multiply-connected curvilinear LEF-layers”. В: *System research and information technologies* 0 (2 2016). ISSN: 1681-6048. DOI: 10.20535/srit.2308-8893.2016.2.06.
- [79] Viktor Pryshlyak та Viktor Dubchak. “FINDING THE SIZE OF PRESSURE FORCE ON UNDERWATER HYDRAULIC STRUCTURES IN DESIGN AND AGRICULTURAL TRAINING OF SPECIALISTS”. В: *ENGINEERING, ENERGY, TRANSPORT AIC* (1(108) 2020). ISSN: 2520-6168. DOI: 10.37128/2520-6168-2020-1-13.

- [80] Alexander K. Brazaluk, Oleg G. Goman та Iuliia V. Brazaluk. “FEATURES OF BOUNDARY ELEMENT METHOD APPLICATION FOR SOLUTIONS OF BOUNDARY-VALUE PROBLEMS IN MOVING BOUNDARY DOMAINS”. B: *Systems and Technologies* 2 (62 2021). DOI: 10.32836/2521-6643-2021.2-62.5.
- [81] Y. V. Ivanchuk, A. A. Yarovyi та K. O. Koval. “NUMERICAL SIMULATION METHOD OF HYDRODYNAMIC PROCESSES”. B: *Information Technology and Computer Engineering* 44 (1 2019), c. 37—45. ISSN: 19999941. DOI: 10.31649/1999-9941-2019-44-1-37-45.
- [82] Florian Neisser та Simon Runkel. “The future is now! Extrapolated risksapes, anticipatory action and the management of potential emergencies”. B: *Geoforum* 82 (2017). ISSN: 00167185. DOI: 10.1016/j.geoforum.2017.04.008.
- [83] Alexandr Kryanev та ін. “Extrapolation of Functions of Many Variables by Means of Metric Analysis”. B: *EPJ Web of Conferences*. T. 173. 2018. DOI: 10.1051/epjconf/201817303014.
- [84] Konstantin Mygalenko та ін. “Development of the technique for restricting the propagation of fire in natural peat ecosystems”. B: *Eastern-European Journal of Enterprise Technologies* 1 (10-91 2018). ISSN: 17294061. DOI: 10.15587/1729-4061.2018.121727.
- [85] Constantin Junk та ін. “Predictor-weighting strategies for probabilistic wind power forecasting with an analog ensemble”. B: *Meteorologische Zeitschrift* 24 (4 2015). ISSN: 16101227. DOI: 10.1127/metz/2015/0659.
- [86] Rudra P Pradhan та Rajesh Kumar. “Forecasting Exchange Rate in India: An Application of Artificial Neural Network Model”. B: *Journal of Mathematics Research* 2 (4 2010). ISSN: 1916-9795. DOI: 10.5539/jmr.v2n4p111.
- [87] Dhiya Al-Jumeily, Rozaida Ghazali та Abir Hussain. “Predicting physical time series using dynamic ridge polynomial neural networks”. B: *PLoS ONE* 9 (8 2014). ISSN: 19326203. DOI: 10.1371/journal.pone.0105766.

- [88] Jolanta Szoplik. “Forecasting of natural gas consumption with artificial neural networks”. B: *Energy* 85 (2015). ISSN: 03605442. DOI: 10.1016/j.energy.2015.03.084.
- [89] Zoe Nivolianitou та Barbara Synodinou. “Towards emergency management of natural disasters and critical accidents: The Greek experience”. B: *Journal of Environmental Management* 92 (10 2011). ISSN: 10958630. DOI: 10.1016/j.jenvman.2011.06.003.
- [90] Hryhorii Ivanets та ін. “Development of combined method for predicting the process of the occurrence of emergencies of natural character”. B: *Eastern-European Journal of Enterprise Technologies* 5 (10-95 2018). ISSN: 17294061. DOI: 10.15587/1729-4061.2018.143045.
- [91] Ting Long та ін. “Coupling edge-based smoothed finite element method with smoothed particle hydrodynamics for fluid structure interaction problems”. B: *Ocean Engineering* 225 (квіт. 2021), с. 108772. ISSN: 0029-8018. DOI: 10.1016/J.OCEANENG.2021.108772.
- [92] Francisco J. Suárez-Grau. “Theoretical derivation of Darcy’s law for fluid flow in thin porous media”. B: *Mathematische Nachrichten* 295 (3 2022). ISSN: 15222616. DOI: 10.1002/mana.202000184.
- [93] Aybek Arifjanov та ін. “Investigation of the interaction of hydraulic parameters of the channel in the filtration process”. B: *E3S Web of Conferences*. T. 401. 2023. DOI: 10.1051/e3sconf/202340103074.
- [94] Ismatulla Khujaev та ін. “INVESTIGATION OF THE GAS-DYNAMIC STATE OF AN ELEMENTARY SECTION OF THE PIPELINE BASED ON N.E. ZHUKOVSKY EQUATION.” B: *Theoretical & Applied Science* 78 (10 жовт. 2019), с. 32—40. ISSN: 23084944. DOI: 10.15863/TAS.2019.10.78.5.
- [95] Li Tao Zhu та ін. “Conventional and data-driven modeling of filtered drag, heat transfer, and reaction rate in gas–particle flows”. B: *AIChE Journal* 67 (8 2021). ISSN: 15475905. DOI: 10.1002/aic.17299.
- [96] Giovanni Catino, Dario D. Monticelli та Alberto Roncoroni. “On the critical p-Laplace equation”. B: *Advances in Mathematics* 433 (2023). ISSN: 10902082. DOI: 10.1016/j.aim.2023.109331.

- [97] Quoc Hung Nguyen та Nguyen Cong Phuc. “A Comparison Estimate for Singular p-Laplace Equations and Its Consequences”. В: *Archive for Rational Mechanics and Analysis* 247 (3 2023). ISSN: 14320673. DOI: 10.1007/s00205-023-01884-7.
- [98] Мар’ян СЛАБІНОГА та Тарас МИХАЙЛОВ. “ЗАСТОСУВАННЯ МЕТОДІВ ЕКСПЕРТНОЇ ОЦІНКИ ПРИ ПРОГНОЗУВАННІ ТЕРМІНІВ ВИКОНАННЯ ПРОЕКТІВ В ГАЛУЗІ ІТ”. В: *INFORMATION TECHNOLOGY AND SOCIETY* (2 (8) 2023). ISSN: 2786-5460. DOI: 10.32689/maup.it.2023.2.9.
- [99] М. В. Кузь та ін. “ПРОГНОЗУВАННЯ ЯКОСТІ ПРОГРАМНИХ ЗАСОБІВ НА ОСНОВІ АНАЛІЗУ ЯКОСТІ ВИМОГ”. В: *METHODS AND DEVICES OF QUALITY CONTROL* (1(50) 2023). ISSN: 1993-9981. DOI: 10.31471/1993-9981-2023-1(50)-101-112.
- [100] Сергій Осієвський та ін. “МЕТОД ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ФУНКЦІОНУВАННЯ ЛЮДИНО-МАШИННОЇ СИСТЕМИ ЗА РАХУНОК ПІДВИЩЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ”. В: *ГРААЛЬ НАУКИ* (6 2021). ISSN: 2710-3056. DOI: 10.36074/grail-of-science.25.06.2021.029.
- [101] М. Bychok та О. Pohudina. “EVALUATION OF USE OF DESIGN TEMPLATES IN THE SOFTWARE DEVELOPMENT”. В: *Radioelectronic and Computer Systems* (1 2021). ISSN: 26632012. DOI: 10.32620/REKS. 2021.1.09.
- [102] Ірина ПІХ та Олексій БІЛИК. “ФОРМУВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА МЕТОДОМ БАГАТОКРИТЕРІАЛЬНОЇ ОПТИМІЗАЦІЇ”. В: *MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES* (1 бер. 2024), с. 117—124. ISSN: 2219-9365. DOI: 10.31891/2219-9365-2024-77-14.
- [103] Yu. I. Hrytsiuk та V. S. Dalyavskyu. “Використання пелюсткових діаграм для візуалізації результатів експертного оцінювання якості

- програмного забезпечення”. В: *Scientific Bulletin of UNFU* 28 (9 2018). ISSN: 1994-7836. DOI: 10.15421/40280919.
- [104] Oleksandr Gordieiev та Konstantin Leontiev. “МОДЕЛЬ СЦЕНАРІЮ ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ”. В: *TECHNICAL SCIENCES AND TECHNOLOGIES* (3(21) 2020). ISSN: 24115363. DOI: 10.25140/2411-5363-2020-3(21)-209-219.
- [105] Тетяна Панченко та ін. “Оцінка якості програмного засобу експертним методом”. В: *InterConf* (42(189) 2024). DOI: 10.51582/interconf.19-20.02.2024.056.

ДОДАТОК А Впровадження результатів дисертаційного дослідження у ТОВ «БАЛФОРД УКРАЇНА»



ТОВ «БАЛФОРД УКРАЇНА»

Україна, 82195, Львівська обл., село Довге, Східницька
Тг, вул.Лесі Українки, будинок 1
E-mail: info@balford.com.ua
www.balford.com.ua

1, Lesi Ukrainki Str.82195 Lviv Region, Dovge Village,
Skhidnytsia Territorial Community,
E-mail: info@balford.com.ua
www.balford.com.ua

05 листопада 2024
№ 02-11-24

Довідка

про впровадження результатів дисертаційної роботи
Соколовського Владислава Володимировича, поданої на здобуття наукового
ступеня доктора філософії, на тему «Алгоритмічне та програмне забезпечення
регіональної системи моніторингу стану потенційно небезпечних об'єктів»
у процесах діяльності підприємства ТОВ «БАЛФОРД УКРАЇНА»

м. Дрогобич

05 листопада 2024 року

Результати дисертаційної роботи Соколовського В. В. у вигляді спеціалізованого програмного забезпечення для використання у складі систем моніторингу стану потенційно небезпечного об'єкту (а саме- МГЕС) використані при проектуванні автоматизованої системи раннього виявлення можливості виникнення надзвичайної ситуації та оповіщення населення на об'єкті «Будівництво гідроелектростанції на р. Стрий у с. Довге-Гірське Дрогобицького району Львівської області».

1. У складі запропонованого програмного забезпечення є програмні модулі, які реалізують методи виявлення та виправлення багатократних помилок передачі даних. Так як МГЕС, які є об'єктами моніторингу, зазвичай розподілені на значній площі та працюють в умовах наявності індустриальних електромагнітних завад, то впровадження методів виявлення та виправлення помилок передачі дозволяє зменшити кількість помилок передачі на 38% від загальної кількості помилок передачі.

2. Запропоноване програмне забезпечення, за рахунок того, що має підсистему прогнозування зміни стану об'єкта моніторингу дозволяє, на відміну від систем моніторингу стану об'єкту, які реалізують лише параметричний контроль вхідних сигналів, на три – чотири відліки (зазвичай один відлік - це 10,20,30 або 60 хвилин) раніше виявляти можливе перевищення докритичного рівня контрольованих сигналів. Це дозволяє оперативному персоналу завчасно розпочати виконання заходів по запобіганню виникненню аварій та надзвичайних ситуацій.

3. Використання підсистеми прогнозування дозволяє під час моніторингу стану МГЕС здійснювати короткочасний прогноз щодо перевищення докритичного значення сигналів які характеризують такі параметри:

- рівень води у верхньому б'єфі;
- рівень води у нижньому б'єфі;



Україна, 82195, Львівська обл., село Довге, Східницька
Тг, вул.Лесі Українки, будинок 1
E-mail: info@balford.com.ua
www.balford.com.ua

1, Lesi Ukrainki Str.82195 Lviv Region, Dovge Village,
Skhidnytsia Territorial Community,
E-mail: info@balford.com.ua
www.balford.com.ua

- рівень вібрації підшипників генератора;
- рівень вібрації підшипників турбіни;
- рівень вібрації підшипників редуктора.

4. Впровадження підсистеми моделювання процесів фільтрації в ґрунтових греблях, які є найпоширеним типом напірних гідроспоруд в Україні, дозволяє завчасно виявляти можливість виникнення значних фільтраційних течій, які можуть призвести до виникнення суфозій та проранів в тілі греблі.

5. Впровадження запропонованого програмного забезпечення дозволило підвищити продуктивність роботи основного технологічного обладнання за рахунок зменшення часу простою основного обладнання на 23%.

З повагою,



Директор ТОВ «БАЛФОРД УКРАЇНА»
Д.О. Нікулін

ДОДАТОК Б Впровадження результатів дисертаційного дослідження у ТОВ «ГУТАРЬОВ ТА ПАРТНЕРИ УКРАЇНА»



LIMITED LIABILITY COMPANY
«DR. HUTAREW AND PARTNERS UKRAINE»
ТОВ «ГУТАРЬОВ ТА ПАРТНЕРИ УКРАЇНА»

02098 м. Київ, вул. Юрія Шумського 1А

Поштова адреса: 02002 м. Київ а/с 83

+38-050-442-17-50

hutarew.ukraine@gmail.com

<https://hutarew-ukraine.com>

06 листопада 2024
№ 49-11-24

Довідка

про впровадження результатів дисертаційної роботи
Соколовського Владислава Володимировича, поданої на здобуття наукового
ступеня доктора філософії, на тему «Алгоритмічне та програмне
забезпечення регіональної системи моніторингу стану потенційно
небезпечних об'єктів» у процесах діяльності
підприємства ТОВ «ГУТАРЬОВ ТА ПАРТНЕРИ УКРАЇНА»

Підприємство ТОВ «ГУТАРЬОВ ТА ПАРТНЕРИ УКРАЇНА» підтверджує, що програмне забезпечення для використання у складі систем моніторингу стану потенційно небезпечного об'єкту – МГЕС, яке розроблено Соколовським В.В. при проведенні дисертаційного дослідження на тему: «Алгоритмічне та програмне забезпечення регіональної системи моніторингу стану потенційно небезпечних об'єктів», було використане при виконанні проекту: «Будівництво гідроелектростанції дериваційного типу на р. Стрий у с. Рибник Дрогобицького району Львівської області».

Автоматизована система моніторингу стану гідротехнічного об'єкта, яка використовує програмне забезпечення, розроблене Соколовським В.В., має такі основні функції:

1. Забезпечує параметричний контроль наступних параметрів, які можуть нести ознаку виникнення аварійної ситуації:

- деформації та виникнення тріщин у елементах споруди (цілісність провідних шлейфів);
- частоти обертання валу турбіни (датчик тахометричного типу);
- частоти обертання валу генератора (датчик тахометричного типу);
- осідань та горизонтальних зміщень елементів споруди (GPS давач);
- фізико-хімічних параметрів води (ручний оповіщувач);
- рівнів верхнього та нижнього б'єфів (ультразвукові давачі);
- появи та рівня води у приміщеннях оглядової галереї, турбінному приміщенні, приміщеннях головних виводів генераторів (датчики електродного типу);
- режимів пропуску повеневих та паводкових вод (ручний сповіщувач).



2. Виявляє про можливість перевищення або досягнення докритичних значень параметрів джерел техногенної та (або) природної небезпеки, які у разі стійкої тенденції до їх зміни у напрямку критичних значень потребують виконання певних дій щодо недопущення досягнення критичних значень
3. Здійснює доведення сигналів і повідомлень про можливість досягнення або досягнення докритичних (критичних) значень параметрів технологічного процесу до працюючого персоналу виробничої дільниці та посадових осіб, відповідальних за стан техногенної безпеки об'єкта.
4. Забезпечує оповіщення працівників об'єкта та населення у разі виникнення НС регіонального або державного рівня. З цією метою система моніторингу технічно сполучена з територіальною автоматизованою системою централізованого оповіщення населення.

Основні відмінності запроектованої системи:

1. Система побудована на основі використання IoT (інтернет речей) технології.
2. З метою захисту інформації, що передається в умовах індустриальних електромагнітних завад, використано метод виявлення та виправлення багатократних помилок передачі інформації. Це дозволяє зменшити на 38 відсотків кількість помилок від загальної кількості виявлених помилок передачі повідомлень.
3. Впровадження підсистеми короткочасного прогнозування в склад програмного забезпечення дозволяє раніше до двох годин часу виявляти можливість виникнення аварійної ситуації, що забезпечує своєчасне виконання запланованих заходів по запобіганню виникненню аварій та надзвичайних ситуацій.
4. Впровадження підсистеми моделювання процесів фільтрації в ґрунтових греблях дозволяє, з діагностичною метою, завчасно виявляти можливість виникнення критично значних для стійкості споруди фільтраційних течій в тілі греблі.
5. Очікується, що впровадження запропонованого програмного забезпечення дозволить підвищити продуктивність роботи основного технологічного обладнання за рахунок зменшення часу простою основного обладнання на 27 відсотків.

В цілому потрібно відмітити гарну якість роботи Соколовського В.В. по розробці програмного забезпечення, а також використання при цьому сучасних алгоритмів та методів програмування.

Директор ТОВ «ГУТАРЬОВ ТА
ПАРТНЕРИ УКРАЇНА»



 О.Л. Садовський

**ДОДАТОК В Впровадження результатів дисертаційного дослідження у
НДР «Теоретичні та практичні аспекти технології
Internet of Everything»**

ЗАТВЕРДЖУЮ

Проректор з наукової роботи
Національного технічного університету
України «Київський політехнічний
Інститут імені Ігоря Сікорського»
Сергій СТИРЕНКО
_____ 2025р



Про використання результатів дисертаційного дослідження аспіранта кафедри інформатики та програмної інженерії (ПІ) факультету інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний Інститут імені Ігоря Сікорського» Соколовського Владислава Володимировича на тему: «Алгоритмічне та програмне забезпечення регіональної системи моніторингу стану потенційно небезпечних об'єктів» на здобуття освітньо-наукового ступеня доктора філософії.

Комісія у складі: голова - декан факультету інформатики та обчислювальної техніки, д.т.н., професор Корнага Я. І.; члени комісії – завідувач кафедри інформатики та програмної інженерії, д.т.н., професор Жаріков Е.В., доцент кафедри інформатики та програмної інженерії, к.т.н. Поперешняк С.В., цим актом засвідчує, що результати дисертаційного дослідження Соколовського Владислава Володимировича отримані ним особисто та використані в:

- НДР «Теоретичні та практичні аспекти технології Internet of Everything» (номер державної реєстрації 0123U104930), яка виконувалась у 2023-2024 р.р. на кафедрі інформатики та програмної інженерії факультету інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний Інститут імені Ігоря Сікорського» та одним із виконавців якої був аспірант кафедри інформатики та програмної інженерії Соколовський В. В.

У НДР використані наступні результати дисертаційної роботи Соколовського В. В.:

- Модифікований метод виявлення та виправлення багатобітових помилок на основі кодів Хеммінгу, який забезпечує високу завадостійкість, зменшення помилок передачі та надійність обміну даними.

Голова комісії:

д.т.н., професор

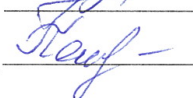
 Ярослав КОРНАГА

Члени комісії:

д.т.н. професор

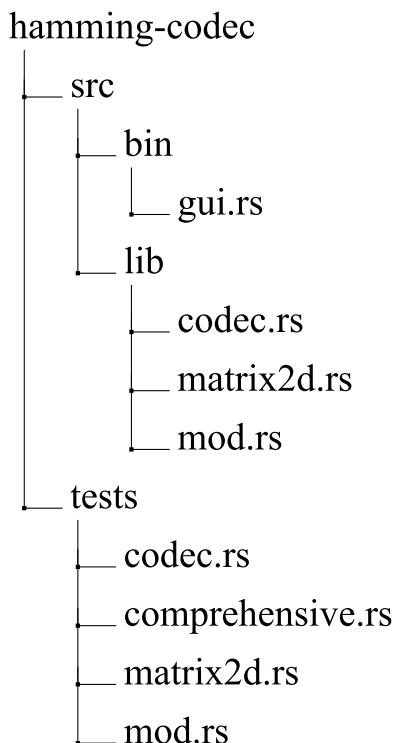
 Едуард ЖАРИКОВ

к.т.н. доцент

 Світлана ПОПЕРЕШНЯК

ДОДАТОК Г Вихідний код додатку «hamming-codec»

Структура каталогів



Програмний код Файл: ./src/bin/gui.rs

```

1 use eframe::egui::{self, Color32, RichText, Stroke};
2 use hammingcodec::codec::{EncodingStep, HammingCodec, Stats, StepType};
3 use rand::Rng;
4
5 const AUTHOR: &str = "Vladyslav Sokolovskyi";
6 const AFFILIATION: &str =
7     "National Technical University of Ukraine \"Igor Sikorsky Kyiv Polytechnic Institute\"";
8 const REQUIRED_BYTES: usize = 16;
9
10 // Cool Unicode icons for different states
11 const ICON_SUCCESS: &str = "☐"; // Sparkles for success
12 const ICON_ERROR: &str = "⚠"; // Warning for errors
13 const ICON_CORRUPTED: &str = "💣"; // Explosion for corruption
14 const ICON_FIXED: &str = "🔧"; // Wrench for fixes
15 const ICON_ORIGINAL: &str = "📄"; // Document for original
16 const ICON_ENCODED: &str = "🔒"; // Lock for encoded
17 const ICON_DECODED: &str = "🔓"; // Unlock for decoded
18 const ICON_BITS: &str = "🔢"; // Numbers for bits
19 const ICON_BYTES: &str = "📊"; // Chart for bytes
20 const ICON_STEP: &str = "🔍"; // Magnifying glass for steps
21 const ICON_PREP: &str = "📋"; // Clipboard for preparation
22 const ICON_PARITY: &str = "🔢"; // Numbers for parity
23 const ICON_SHIFT: &str = "↔"; // Arrows for shift

```

```

24 const ICON_CHECK: &str = "□"; // Check for final result
25
26 #[derive(Default)]
27 struct TranscodeStages {
28     original: Vec<u8>,
29     encoded: Vec<u8>,
30     corrupted: Option<Vec<u8>>,
31     decoded: Option<Vec<u8>>,
32     stats: Option<Stats>,
33 }
34
35 #[derive(Default)]
36 struct ByteModification {
37     position_input: String,
38     binary_input: String,
39     position_error: Option<String>,
40     binary_error: Option<String>,
41     current_position: Option<usize>,
42 }
43
44 impl ByteModification {
45     fn new() -> Self {
46         Self {
47             position_input: String::from("0"),
48             binary_input: String::from("00000000"),
49             position_error: None,
50             binary_error: None,
51             current_position: None,
52         }
53     }
54
55     fn update_binary_for_position(&mut self, corrupted: &[u8]) {
56         if let Ok(pos) = self.position_input.parse::<usize>() {
57             if pos < corrupted.len() {
58                 // Convert to binary string with LSB first to match codec
59                 let mut bits = String::with_capacity(8);
60                 for i in 0..8 {
61                     bits.push(if (corrupted[pos] >> i) & 1 == 1 {
62                         '1'
63                     } else {
64                         '0'
65                     });
66                 }
67                 self.binary_input = bits;
68                 self.current_position = Some(pos);
69                 self.position_error = None;
70             } else {
71                 self.position_error = Some(format!("Invalid position (0-{})", corrupted.len() - 1));
72             }

```

```

73     } else {
74         self.position_error = Some("Invalid position number".to_string());
75     }
76 }
77 }
78
79 struct HammingCodecApp {
80     codec: HammingCodec,
81     input_text: String,
82     error_text: String,
83     stages: TranscodeStages,
84     show_steps: bool,
85     byte_modification: ByteModification,
86 }
87
88 impl Default for HammingCodecApp {
89     fn default() -> Self {
90         Self {
91             codec: HammingCodec::new(),
92             input_text: String::from("111111111111111"),
93             error_text: String::new(),
94             stages: TranscodeStages::default(),
95             show_steps: true,
96             byte_modification: ByteModification::new(),
97         }
98     }
99 }
100
101 impl HammingCodecApp {
102     // Helper function to get theme-aware colors
103     fn get_theme_colors(&self, ui: &egui::Ui) -> ThemeColors {
104         let is_dark = ui.visuals().dark_mode;
105         ThemeColors {
106             success: if is_dark {
107                 Color32::from_rgb(100, 255, 100)
108             } else {
109                 Color32::from_rgb(0, 160, 0)
110             },
111             error: if is_dark {
112                 Color32::from_rgb(255, 100, 100)
113             } else {
114                 Color32::from_rgb(200, 0, 0)
115             },
116             warning: if is_dark {
117                 Color32::from_rgb(255, 180, 100)
118             } else {
119                 Color32::from_rgb(200, 120, 0)
120             },
121             highlight_bg: if is_dark {

```

```

122         Color32::from_rgba_premultiplied(255, 100, 100, 25)
123     } else {
124         Color32::from_rgba_premultiplied(255, 200, 200, 50)
125     },
126     muted_text: ui.visuals().weak_text_color(),
127     text: ui.visuals().text_color(),
128     strong_text: ui.visuals().strong_text_color(),
129 }
130 }
131
132 fn format_byte_bits(
133     &self,
134     byte: u8,
135     highlight_indices: Option<&[usize]>,
136     error_color: Option<Color32>,
137 ) -> RichText {
138     let mut result = String::with_capacity(24);
139     // Display bits LSB first to match codec
140     for i in 0..8 {
141         let bit = (byte >> i) & 1;
142         let should_highlight = highlight_indices
143             .map(|indices| indices.contains(&i))
144             .unwrap_or(false);
145
146         if should_highlight {
147             result.push_str(&format!("{}", bit));
148         } else {
149             result.push_str(&bit.to_string());
150         }
151     }
152     if let Some(color) = error_color {
153         RichText::new(result).color(color)
154     } else {
155         RichText::new(result)
156     }
157 }
158
159 fn get_bit_differences(&self, original: &[u8], modified: &[u8]) -> Vec<(usize, Vec<usize>>) {
160     let mut differences = Vec::new();
161     for (byte_idx, (orig, modi)) in original.iter().zip(modified.iter()).enumerate() {
162         let mut bit_diffs = Vec::new();
163         for bit_idx in 0..8 {
164             if (orig >> bit_idx) & 1 != (modi >> bit_idx) & 1 {
165                 bit_diffs.push(bit_idx);
166             }
167         }
168         if !bit_diffs.is_empty() {
169             differences.push((byte_idx, bit_diffs));
170         }
171     }

```

```

171     }
172     differences
173 }
174
175 fn display_stage(
176     &self,
177     ui: &mut egui::Ui,
178     label: &str,
179     data: &[u8],
180     show_text: bool,
181     is_corrupted: bool,
182     compare_with: Option<&[u8]>,
183 ) {
184     let colors = self.get_theme_colors(ui);
185
186     ui.horizontal(|ui| {
187         ui.with_layout(
188             egui::Layout::left_to_right(egui::Align::TOP).with_main_wrap(true),
189             |ui| {
190                 let label_color = if is_corrupted {
191                     colors.error
192                 } else {
193                     colors.strong_text
194                 };
195
196                 let stage_icon = match label {
197                     "Original Data" => ICON_ORIGINAL,
198                     "Encoded Data" => ICON_ENCODED,
199                     "Corrupted Data" => ICON_CORRUPTED,
200                     "Decoded Data" => ICON_DECODED,
201                     _ => "",
202                 };
203
204                 let differences =
205                     compare_with.map(|other| self.get_bit_differences(other, data));
206                 let grid_id = format!("{}", _grid", label.to_lowercase().replace(' ', "_"));
207
208                 ui.push_id(label, |ui| {
209                     egui::Frame::none()
210                         .fill(ui.visuals().faint_bg_color)
211                         .rounding(8.0)
212                         .inner_margin(8.0)
213                         .show(ui, |ui| {
214                             ui.horizontal(|ui| {
215                                 ui.label(
216                                     RichText::new(format!("{}", stage_icon, label))
217                                         .color(label_color)
218                                         .strong()
219                                         .size(16.0),

```

```

220         );
221         if let Some(ref diffs) = differences {
222             if !diffs.is_empty() {
223                 ui.label(
224                     RichText::new(format!("({} changes)", diffs.len()))
225                         .color(colors.error),
226                 );
227             }
228         }
229     });
230
231     egui::Grid::new(grid_id)
232         .striped(true)
233         .spacing([10.0, 4.0])
234         .min_col_width(100.0)
235         .show(ui, |ui| {
236             // Column headers with consistent styling
237             for (icon, label) in [
238                 (ICON_BYTES, "Offset"),
239                 (ICON_BYTES, "Hex"),
240                 (ICON_BITS, "Binary"),
241             ]
242             .iter()
243             {
244                 ui.label(
245                     RichText::new(format!("{}", icon, label))
246                         .strong()
247                         .color(colors.strong_text),
248                 );
249             }
250             if show_text {
251                 ui.label(
252                     RichText::new("□ Text")
253                         .strong()
254                         .color(colors.strong_text),
255                 );
256             }
257             ui.end_row();
258
259             for (row_idx, chunk) in data.chunks(4).enumerate() {
260                 // Offset column
261                 ui.label(
262                     RichText::new(format!("{:02X}", row_idx * 4))
263                         .color(colors.muted_text),
264                 );
265
266                 // Hex display
267                 ui.horizontal(|ui| {
268                     for (i, &byte) in chunk.iter().enumerate() {

```



```

269         let byte_idx = row_idx * 4 + i;
270         let is_different = differences.as_ref().map_or(
271             false,
272             |diffs| {
273                 diffs
274                     .iter()
275                     .any(|(idx, _)| *idx == byte_idx)
276             },
277         );
278         let hex_text = if is_different {
279             format!("{}", byte)
280         } else {
281             format!("{}", byte)
282         };
283         ui.monospace(RichText::new(hex_text).color(
284             if is_different {
285                 colors.error
286             } else {
287                 colors.text
288             },
289         ));
290         ui.add_space(4.0);
291     }
292 });
293
294 // Binary display
295 ui.horizontal(|ui| {
296     for (i, &byte) in chunk.iter().enumerate() {
297         let byte_idx = row_idx * 4 + i;
298         let bit_diffs =
299             differences.as_ref().and_then(|diffs| {
300                 diffs
301                     .iter()
302                     .find(|(idx, _)| *idx == byte_idx)
303                     .map(|(_, bits)| bits.as_slice())
304             });
305
306         let error_color =
307             bit_diffs.map(|_| colors.error);
308
309         ui.monospace(self.format_byte_bits(
310             byte,
311             bit_diffs,
312             error_color,
313         ));
314         ui.add_space(4.0);
315     }
316 });
317

```

```

318 // Text display
319 if show_text {
320     if let Ok(text) = String::from_utf8(chunk.to_vec())
321     {
322         let text_color = if differences.as_ref().map_or(
323             false,
324             |diffs| {
325                 diffs.iter().any(|(idx, _)| {
326                     let chunk_start = row_idx * 4;
327                     let chunk_end = chunk_start + 4;
328                     *idx >= chunk_start
329                     && *idx < chunk_end
330                 })
331             },
332         ) {
333             colors.error
334         } else {
335             colors.strong_text
336         };
337         ui.label(RichText::new(text).color(text_color));
338     } else {
339         ui.label(
340             RichText::new("·").color(colors.muted_text),
341         );
342     }
343 }
344 ui.end_row();
345 }
346 });
347
348 // Show detailed changes with fancy formatting
349 if let Some(differences) = differences {
350     if !differences.is_empty() {
351         ui.add_space(8.0);
352         ui.separator();
353         ui.vertical(|ui| {
354             ui.label(
355                 RichText::new(format!(
356                     "{} Changes Detected",
357                     ICON_ERROR
358                 ))
359                 .strong()
360                 .color(colors.error)
361                 .size(14.0),
362             );
363             for (byte_idx, bit_indices) in differences {
364                 ui.horizontal(|ui| {
365                     ui.label(
366                         RichText::new(format!(

```

```

367         "{} Byte {:02X}: bits {} modified",
368         ICON_BITS,
369         byte_idx,
370         bit_indices
371         .iter()
372         .map(|&b| format!("{}", b))
373         .collect::<Vec<_>>()
374         .join(", ")
375     ))
376     .color(colors.warning),
377 );
378 });
379 }
380 });
381 }
382 }
383 });
384 });
385 },
386 );
387 });
388 }
389
390 fn display_stats(&self, ui: &mut egui::Ui, stats: &Stats) {
391     let colors = self.get_theme_colors(ui);
392
393     egui::Frame::none()
394         .fill(ui.visuals().faint_bg_color)
395         .inner_margin(8.0)
396         .rounding(4.0)
397         .show(ui, |ui| {
398             ui.label(
399                 RichText::new(format!("{}", "Decoding Statistics", ICON_DECODED))
400                     .strong()
401                     .size(16.0)
402                     .color(colors.strong_text),
403             );
404             ui.add_space(4.0);
405
406             let stat_color = if stats.corrected_informational_bits > 0 {
407                 colors.success
408             } else if stats.multiple_errors > 0 {
409                 colors.error
410             } else {
411                 colors.text
412             };
413
414             ui.label(
415                 RichText::new(format!(

```

```

416         "{} Corrected Information Bits: {}",
417         if stats.corrected_informational_bits > 0 {
418             ICON_SUCCESS
419         } else {
420             ICON_FIXED
421         },
422         stats.corrected_informational_bits
423     ))
424     .color(stat_color),
425 );
426
427 if stats.multiple_errors > 0 {
428     ui.label(
429         RichText::new(format!(
430             "{} Multiple Errors: {}",
431             ICON_ERROR, stats.multiple_errors
432         ))
433         .color(colors.error),
434     );
435 }
436
437 if stats.parity_bits_errors > 0 {
438     ui.label(
439         RichText::new(format!(
440             "{} Parity Bit Errors: {}",
441             ICON_CORRUPTED, stats.parity_bits_errors
442         ))
443         .color(colors.warning),
444     );
445 }
446 });
447 }
448
449 fn validate_input(&self, text: &str) -> Result<(), String> {
450     let bytes = text.as_bytes();
451     match bytes.len() {
452         0 => Err("Please enter exactly 16 bytes of text".to_string()),
453         n if n < REQUIRED_BYTES => {
454             Err(format!("Text too short: {} of {} bytes", n, REQUIRED_BYTES))
455         }
456         n if n > REQUIRED_BYTES => {
457             Err(format!("Text too long: {} of {} bytes", n, REQUIRED_BYTES))
458         }
459         _ => Ok(()),
460     }
461 }
462
463 fn get_step_icon(&self, step_type: &StepType) -> &'static str {
464     match step_type {

```

```

465     StepType::DataPreparation => ICON_PREP,
466     StepType::ParityCalculation => ICON_PARITY,
467     StepType::HorizontalShift | StepType::VerticalShift => ICON_SHIFT,
468     StepType::ErrorDetection | StepType::ErrorCorrection => ICON_ERROR,
469     StepType::FinalResult => ICON_CHECK,
470 }
471 }
472
473 fn display_steps(&self, ui: &mut egui::Ui) {
474     if !self.show_steps {
475         return;
476     }
477
478     let colors = self.get_theme_colors(ui);
479
480     ui.separator();
481     ui.heading(RichText::new(format!("{}", Algorithm Steps", ICON_STEP)).strong());
482
483     egui::ScrollArea::vertical().show(ui, |ui| {
484         for (step_idx, step) in self.codec.steps.iter().enumerate() {
485             ui.push_id(step_idx, |ui| {
486                 egui::Frame::none()
487                     .fill(ui.visuals().faint_bg_color)
488                     .rounding(8.0)
489                     .inner_margin(8.0)
490                     .show(ui, |ui| {
491                         ui.horizontal(|ui| {
492                             let icon = self.get_step_icon(&step.step_type);
493                             ui.label(
494                                 RichText::new(format!(
495                                     "{} Step {}: {}",
496                                     icon,
497                                     step_idx + 1,
498                                     step.description
499                                 ))
500                                 .strong()
501                                 .color(colors.strong_text)
502                                 .size(14.0),
503                             );
504                         });
505
506                         // Create a grid for transposed byte display
507                         egui::Grid::new(format!("step_{}_grid", step_idx))
508                             .spacing([8.0, 2.0])
509                             .show(ui, |ui| {
510                                 // Headers row
511                                 ui.label(RichText::new("").strong()); // Empty cell for alignment
512                                 ui.label(
513                                     RichText::new(format!("{}", Index", ICON_BYTES)).strong(),

```

```

514         );
515         ui.label(RichText::new(format!("{ } Hex", ICON_BYTES)).strong());
516         ui.end_row();
517
518         // Bit rows (7 down to 0)
519         for bit_pos in (0..8).rev() {
520             ui.label(
521                 RichText::new(format!("Bit { }", bit_pos)).strong(),
522             );
523
524             // Create a horizontal row of bits for this position
525             ui.horizontal(|ui| {
526                 for (byte_idx, &byte) in step.data.iter().enumerate() {
527                     let bit = (byte >> bit_pos) & 1;
528                     let bit_text = if bit == 1 {
529                         RichText::new("1").color(colors.strong_text)
530                     } else {
531                         RichText::new("0").color(colors.muted_text)
532                     };
533                     ui.monospace(bit_text);
534                     ui.add_space(4.0);
535                 }
536             });
537             ui.end_row();
538         }
539
540         // Index row
541         ui.label(RichText::new("Index").strong());
542         ui.horizontal(|ui| {
543             for byte_idx in 0..step.data.len() {
544                 ui.monospace(RichText::new(format!("{:02}", byte_idx)));
545                 ui.add_space(4.0);
546             }
547         });
548         ui.end_row();
549
550         // Hex row
551         ui.label(RichText::new("Hex").strong());
552         ui.horizontal(|ui| {
553             for &byte in step.data.iter() {
554                 ui.monospace(RichText::new(format!("{:02X}", byte)));
555                 ui.add_space(4.0);
556             }
557         });
558         ui.end_row();
559     });
560
561     });
562     ui.add_space(4.0);

```

```

563     }
564     });
565 }
566
567 fn handle_byte_modification(&mut self, corrupted: Vec<u8>) -> Option<Vec<u8>> {
568     let mut result = corrupted.clone();
569     let max_len = corrupted.len();
570
571     // Validate position
572     self.byte_modification.position_error = None;
573     let position = match self.byte_modification.position_input.parse::<usize>() {
574         Ok(pos) if pos < max_len => pos,
575         _ => {
576             self.byte_modification.position_error =
577                 Some(format!("Invalid position (0-{})", max_len - 1));
578             return None;
579         }
580     };
581
582     // Validate binary input
583     self.byte_modification.binary_error = None;
584     if self.byte_modification.binary_input.len() != 8
585         || !self
586             .byte_modification
587             .binary_input
588             .chars()
589             .all(|c| c == '0' || c == '1')
590     {
591         self.byte_modification.binary_error =
592             Some("Invalid binary (8 bits required)".to_string());
593         return None;
594     }
595
596     // Parse binary value (LSB first to match codec)
597     let mut value = 0u8;
598     for (i, c) in self.byte_modification.binary_input.chars().enumerate() {
599         if c == '1' {
600             value |= 1 << i;
601         }
602     }
603     result[position] = value;
604     Some(result)
605 }
606
607 fn update_ui(&mut self, ctx: &egui::Context, _frame: &mut eframe::Frame) {
608     egui::SidePanel::right("steps_panel")
609         .resizable(true)
610         .default_width(400.0)
611         .width_range(300.0..=800.0)

```

```

612         .show(ctx, |ui| {
613             ui.vertical_centered(|ui| {
614                 ui.heading(RichText::new(format!("{}", Algorithm Steps", ICON_STEP)).strong());
615             });
616             if self.show_steps {
617                 self.display_steps(ui);
618             }
619         });
620
621     egui::CentralPanel::default().show(ctx, |ui| {
622         ui.vertical_centered(|ui| {
623             ui.add_space(10.0);
624             ui.heading(
625                 RichText::new("Hamming Code Transcoder")
626                     .strong()
627                     .size(24.0)
628                     .color(ui.visuals().strong_text_color()),
629             );
630
631             ui.add_space(4.0);
632             egui::Frame::none()
633                 .fill(ui.visuals().faint_bg_color)
634                 .inner_margin(8.0)
635                 .rounding(4.0)
636                 .show(ui, |ui| {
637                     ui.vertical(|ui| {
638                         ui.label("Enter text to encode:");
639                         let response = ui.add(
640                             egui::TextEdit::multiline(&mut self.input_text)
641                                 .desired_width(f32::INFINITY)
642                                 .desired_rows(3)
643                                 .font(egui::TextStyle::Monospace),
644                         );
645
646                         if response.changed() {
647                             match self.validate_input(&self.input_text) {
648                                 Ok(()) => {
649                                     self.stages.original = self.input_text.as_bytes().to_vec();
650                                     self.stages.encoded =
651                                         self.codec.encode(&self.stages.original);
652                                     self.stages.corrupted = None;
653                                     self.stages.decoded = None;
654                                     self.stages.stats = None;
655                                     self.error_text.clear();
656                                 }
657                                 Err(msg) => {
658                                     self.error_text = msg;
659                                     self.stages = TranscodeStages::default();
660                                 }
661                             }
662                         }
663                     });
664                 });
665         });
666     });

```



```

661         }
662     }
663
664     if !self.error_text.is_empty() {
665         ui.colored_label(ui.visuals().error_fg_color, &self.error_text);
666     }
667 });
668 });
669
670 if !self.stages.original.is_empty() {
671     ui.add_space(20.0);
672
673     self.display_stage(
674         ui,
675         "Original Data",
676         &self.stages.original,
677         true,
678         false,
679         None,
680     );
681
682     if !self.stages.encoded.is_empty() {
683         ui.add_space(10.0);
684         self.display_stage(
685             ui,
686             "Encoded Data",
687             &self.stages.encoded,
688             false,
689             false,
690             None,
691         );
692
693         ui.add_space(10.0);
694         if ui.button("Introduce Random Error").clicked() {
695             let corrupted = self.stages.encoded.clone();
696             let mut rng = rand::thread_rng();
697             let pos = rng.gen_range(0..corrupted.len());
698             let bit = rng.gen_range(0..8);
699             let mut corrupted_data = corrupted.clone();
700             corrupted_data[pos] ^= 1 << bit;
701             self.stages.corrupted = Some(corrupted_data.clone());
702             let (decoded, stats) = self.codec.decode(&corrupted_data);
703             self.stages.decoded = Some(decoded);
704             self.stages.stats = Some(stats);
705         }
706
707         if let Some(ref corrupted) = self.stages.corrupted.clone() {
708             ui.add_space(10.0);
709

```

```

710     self.display_stage(
711         ui,
712         "Corrupted Data",
713         corrupted,
714         false,
715         true,
716         Some(&self.stages.encoded),
717     );
718
719     // Add byte modification UI
720     ui.add_space(5.0);
721
722     ui.horizontal(|ui| {
723         ui.label(
724             RichText::new(format!("{}", Modify byte:", ICON_BYTES)).strong(),
725         );
726
727         // Position input
728         ui.label("Position:");
729         if ui
730             .text_edit_singleline(
731                 &mut self.byte_modification.position_input,
732             )
733             .changed()
734         {
735             self.byte_modification.update_binary_for_position(corrupted);
736         }
737
738         // Binary value input
739         ui.label("Binary value:");
740         ui.text_edit_singleline(&mut self.byte_modification.binary_input);
741
742         // Add modify button
743         if ui.button(format!("{}", Modify", ICON_FIXED)).clicked() {
744             if let Some(new_corrupted) =
745                 self.handle_byte_modification(corrupted.clone())
746             {
747                 self.stages.corrupted = Some(new_corrupted.clone());
748                 let (decoded, stats) = self.codec.decode(&new_corrupted);
749                 self.stages.decoded = Some(decoded);
750                 self.stages.stats = Some(stats);
751             }
752         }
753     });
754
755     // Display errors if any
756     if let Some(ref error) = self.byte_modification.position_error {
757         ui.colored_label(ui.visuals().error_fg_color, error);
758     }

```

```

759         if let Some(ref error) = self.byte_modification.binary_error {
760             ui.colored_label(ui.visuals().error_fg_color, error);
761         }
762
763         if ui.button("Decode").clicked() {
764             let (decoded, stats) = self.codec.decode(corrupted);
765             self.stages.decoded = Some(decoded);
766             self.stages.stats = Some(stats);
767         }
768
769         if let Some(ref decoded) = self.stages.decoded {
770             ui.add_space(10.0);
771             self.display_stage(
772                 ui,
773                 "Decoded Data",
774                 decoded,
775                 true,
776                 false,
777                 Some(&self.stages.original),
778             );
779
780             if let Some(ref stats) = self.stages.stats {
781                 ui.add_space(10.0);
782                 self.display_stats(ui, stats);
783             }
784         }
785     }
786 }
787
788 });
789
790 // Add steps toggle at the bottom of the central panel
791 ui.with_layout(egui::Layout::bottom_up(egui::Align::LEFT), |ui| {
792     ui.checkbox(
793         &mut self.show_steps,
794         format!("{}", Show Algorithm Steps", ICON_STEP),
795     );
796 });
797 });
798 }
799
800 fn process_input(&mut self) {
801     if self.input_text.len() > REQUIRED_BYTES {
802         self.error_text = format!("Input text is too long (max {} bytes)", REQUIRED_BYTES);
803         return;
804     }
805
806     self.error_text.clear();
807     self.stages.original = self.input_text.as_bytes().to_vec();

```

```

808     self.stages.encoded = self.codec.encode(&self.stages.original);
809     self.stages.corrupted = None;
810     self.stages.decoded = None;
811     self.stages.stats = None;
812 }
813
814 fn corrupt_data(&mut self) {
815     let corrupted = self.stages.encoded.clone();
816     let mut rng = rand::thread_rng();
817     let pos = rng.gen_range(0..corrupted.len());
818     let bit = rng.gen_range(0..8);
819     let mut corrupted_data = corrupted.clone();
820     corrupted_data[pos] ^= 1 << bit;
821     self.stages.corrupted = Some(corrupted_data.clone());
822     let (decoded, stats) = self.codec.decode(&corrupted_data);
823     self.stages.decoded = Some(decoded);
824     self.stages.stats = Some(stats);
825 }
826 }
827
828 impl eframe::App for HammingCodecApp {
829     fn update(&mut self, ctx: &egui::Context, _frame: &mut eframe::Frame) {
830         self.update_ui(ctx, _frame);
831     }
832 }
833
834 // Add this struct at the top of the file
835 struct ThemeColors {
836     success: Color32,
837     error: Color32,
838     warning: Color32,
839     highlight_bg: Color32,
840     muted_text: Color32,
841     text: Color32,
842     strong_text: Color32,
843 }
844
845 fn main() {
846     let options = eframe::NativeOptions {
847         default_theme: eframe::Theme::Dark,
848         follow_system_theme: true,
849         ..Default::default()
850     };
851
852     eframe::run_native(
853         "Hamming Code Transcoder",
854         options,
855         Box::new(|_cc| Ok(Box::new(HammingCodecApp::default()))),
856 )

```

```
857     .unwrap();  
858 }
```

Файл: ./src/lib/codec.rs

```

1  /// Hamming code implementation with step-by-step encoding and decoding tracking.
2  ///
3  /// This module implements the (24,16) Hamming code variant with additional
4  /// features for tracking the encoding/decoding process.
5  use crate::matrix2d::Matrix2D;
6
7  /// Records a single step in the encoding or decoding process.
8  #[derive(Debug, Clone)]
9  pub struct EncodingStep {
10     // Description of the current operation
11     pub description: String,
12     /// Current state of the data
13     pub data: Vec<u8>,
14     /// Type of operation being performed
15     pub step_type: StepType,
16 }
17
18 /// Categorizes different operations in the encoding/decoding process.
19 #[derive(Debug, Clone)]
20 pub enum StepType {
21     /// Initial data preparation and padding
22     DataPreparation,
23     /// Calculation of Hamming code parity bits
24     ParityCalculation,
25     /// Horizontal bit shifting for error protection
26     HorizontalShift,
27     /// Vertical bit shifting for error protection
28     VerticalShift,
29     /// Error detection phase during decoding
30     ErrorDetection,
31     /// Error correction phase during decoding
32     ErrorCorrection,
33     /// Final result of the decoding process
34     FinalResult,
35 }
36
37 /// Implements the (24,16) Hamming code encoder/decoder with step tracking.
38 ///
39 /// This codec:
40 /// - Encodes 16 bytes of data into 24 bytes with error correction
41 /// - Tracks each step of encoding/decoding process
42 /// - Can detect and correct single-bit errors
43 /// - Can detect (but not correct) multiple-bit errors
44 pub struct HammingCodec {
45     /// Total size of encoded buffer (24 bytes)
46     buffer_size: usize,
47     /// Size of the original data (16 bytes)

```

```

48     data_size: usize,
49     /// Records of encoding/decoding steps for analysis
50     pub steps: Vec<EncodingStep>,
51 }
52
53 /// Statistics collected during the decoding process.
54 #[derive(Default, Debug)]
55 pub struct Stats {
56     /// Number of successfully corrected information bits
57     pub corrected_informational_bits: usize,
58     /// Number of detected multiple-bit errors
59     pub multiple_errors: usize,
60     /// Number of parity bit errors detected
61     pub parity_bits_errors: usize,
62 }
63
64 impl Default for HammingCodec {
65     fn default() -> Self {
66         Self::new()
67     }
68 }
69
70 impl HammingCodec {
71     /// Creates a new HammingCodec instance with 24-byte buffer and 16-byte data size.
72     pub fn new() -> Self {
73         Self {
74             buffer_size: 24,
75             data_size: 16,
76             steps: Vec::new(),
77         }
78     }
79
80     /// Clears the recorded encoding/decoding steps.
81     pub fn clear_steps(&mut self) {
82         self.steps.clear();
83     }
84
85     /// Records a new step in the encoding/decoding process.
86     ///
87     /// # Arguments
88     /// * `description` - Human-readable description of the step
89     /// * `data` - Current state of the data
90     /// * `step_type` - Category of operation being performed
91     fn add_step(&mut self, description: String, data: Vec<u8>, step_type: StepType) {
92         self.steps.push(EncodingStep {
93             description,
94             data: data.clone(),
95             step_type,
96         });

```

```

97     }
98
99     /// Converts a slice of bits back into a byte (LSB first).
100    ///
101    /// # Arguments
102    /// * `bits` - Slice of 8 bits to convert
103    ///
104    /// # Returns
105    /// Resulting byte value
106    pub fn bits_to_u8(&self, bits: &[u8]) -> u8 {
107        let mut result = 0u8;
108        for (i, &bit) in bits.iter().enumerate() {
109            if bit == 1 {
110                result |= 1 << i;
111            }
112        }
113        result
114    }
115
116    /// Converts a byte into its individual bits (LSB first).
117    ///
118    /// # Arguments
119    /// * `value` - Byte to convert
120    ///
121    /// # Returns
122    /// Vector of 8 bits (0s and 1s)
123    pub fn u8_to_bits(&self, value: u8) -> Vec<u8> {
124        let mut bits = vec![0u8; 8];
125        for i in 0..8 {
126            bits[i] = ((value >> i) & 1) as u8;
127        }
128        bits
129    }
130
131    // Calculates control (parity) bits for error detection.
132    ///
133    /// Uses Hamming code algorithm to generate parity bits from data bytes.
134    /// Each control byte is calculated using specific bits from data bytes
135    /// according to the Hamming code matrix.
136    ///
137    /// # Arguments
138    /// * `data_bytes` - Input data bytes to calculate parity for
139    ///
140    /// # Returns
141    /// Vector of control bytes (8 bytes for 16 data bytes)
142    pub fn calculate_control_bits(&self, data_bytes: &[u8]) -> Vec<u8> {
143        let mut control_bytes = vec![0u8; self.buffer_size - self.data_size];
144
145        for i in 0..(self.data_size / 8) {

```



```

146 // Calculate first control bit (positions 1,2,4,5,7)
147 control_bytes[i * 4] = data_bytes[i * 8]
148     ^ data_bytes[i * 8 + 1]
149     ^ data_bytes[i * 8 + 3]
150     ^ data_bytes[i * 8 + 4]
151     ^ data_bytes[i * 8 + 6];
152 // Calculate second control bit (positions 1,3,4,6,7)
153 control_bytes[i * 4 + 1] = data_bytes[i * 8]
154     ^ data_bytes[i * 8 + 2]
155     ^ data_bytes[i * 8 + 3]
156     ^ data_bytes[i * 8 + 5]
157     ^ data_bytes[i * 8 + 6];
158 // Calculate third control bit (positions 2,3,4,8)
159 control_bytes[i * 4 + 2] = data_bytes[i * 8 + 1]
160     ^ data_bytes[i * 8 + 2]
161     ^ data_bytes[i * 8 + 3]
162     ^ data_bytes[i * 8 + 7];
163 // Calculate fourth control bit (positions 5,6,7,8)
164 control_bytes[i * 4 + 3] = data_bytes[i * 8 + 4]
165     ^ data_bytes[i * 8 + 5]
166     ^ data_bytes[i * 8 + 6]
167     ^ data_bytes[i * 8 + 7];
168 }
169 control_bytes
170 }
171
172 /// Converts bytes into a 2D matrix representation.
173 ///
174 /// Creates an 8×buffer_size matrix where each column represents
175 /// a byte's bits in MSB-first order.
176 ///
177 /// # Arguments
178 /// * `buffer` - Bytes to convert to matrix form
179 ///
180 /// # Returns
181 /// Matrix2D containing the bits of input bytes
182 fn bytes_to_matrix(&self, buffer: &[u8]) -> Matrix2D<u8> {
183     let mut matrix = Matrix2D::new(8, self.buffer_size);
184     for (byte_idx, &byte) in buffer.iter().enumerate() {
185         let bits = self.u8_to_bits(byte);
186         for (bit_idx, &bit) in bits.iter().enumerate() {
187             matrix.set(7 - bit_idx, byte_idx, bit);
188         }
189     }
190     matrix
191 }
192
193 /// Converts a matrix back into a byte array.
194 ///

```

```

195 /// Reads the matrix column by column, converting each column's bits
196 /// back into a byte (MSB first order).
197 ///
198 /// # Arguments
199 /// * `matrix` - Source matrix containing bits
200 /// * `size` - Number of bytes to extract
201 ///
202 /// # Returns
203 /// Vector of bytes reconstructed from matrix columns
204 fn matrix_to_bytes(&self, matrix: &Matrix2D<u8>, size: usize) -> Vec<u8> {
205     let mut result = vec![0u8; size];
206     for j in 0..size {
207         let mut bits = vec![0u8; 8];
208         for i in 0..8 {
209             bits[7 - i] = *matrix.get(i, j).unwrap_or(&0);
210         }
211         result[j] = self.bits_to_u8(&bits);
212     }
213     result
214 }
215
216 /// Performs horizontal bit shifting on the encoded data.
217 ///
218 /// Shifts each row of bits in the matrix by a different amount
219 /// to improve error detection capabilities.
220 ///
221 /// # Arguments
222 /// * `buffer` - Data to be shifted
223 /// * `direction` - Positive for right shift, negative for left shift
224 ///
225 /// # Returns
226 /// Vector of bytes after shifting
227 fn horizontal_shift(&self, buffer: &[u8], direction: i32) -> Vec<u8> {
228     let mut matrix = self.bytes_to_matrix(buffer);
229
230     for i in 0..8 {
231         let shift = i * direction.abs() as usize;
232         if direction > 0 {
233             matrix.rotate_row_right(i, 7 - shift);
234         } else {
235             matrix.rotate_row_left(i, 7 - shift);
236         }
237     }
238
239     self.matrix_to_bytes(&matrix, self.buffer_size)
240 }
241
242 /// Performs vertical bit shifting on the encoded data.
243 ///

```

```

244 /// Shifts each column of bits in the matrix by a different amount
245 /// to further improve error detection capabilities.
246 ///
247 /// # Arguments
248 /// * `buffer` - Data to be shifted
249 /// * `direction` - Positive for upward shift, negative for downward shift
250 ///
251 /// # Returns
252 /// Vector of bytes after shifting
253 fn vertical_shift(&self, buffer: &[u8], direction: i32) -> Vec<u8> {
254     let mut matrix = self.bytes_to_matrix(buffer);
255
256     for i in 0..self.buffer_size {
257         let shift = i * direction.abs() as usize;
258         if direction > 0 {
259             matrix.rotate_column_up(i, shift % 8);
260         } else {
261             matrix.rotate_column_down(i, shift % 8);
262         }
263     }
264
265     self.matrix_to_bytes(&matrix, self.buffer_size)
266 }
267
268 /// Splits received data buffer into information and control bits.
269 ///
270 /// # Arguments
271 /// * `receive_buffer` - Complete received data buffer
272 ///
273 /// # Returns
274 /// Tuple containing:
275 /// - Information bits (first 16 bytes)
276 /// - Received control bits (last 8 bytes)
277 /// - Newly calculated control bits from information bits
278 fn split_receive_buffer(&self, receive_buffer: &[u8]) -> (Vec<u8>, Vec<u8>, Vec<u8>) {
279     let mut received_info_bits = vec![0u8; self.data_size];
280     let mut received_control_bits = vec![0u8; self.buffer_size - self.data_size];
281
282     received_info_bits.copy_from_slice(&receive_buffer[..self.data_size]);
283     received_control_bits.copy_from_slice(&receive_buffer[self.data_size..]);
284
285     let calculated_control_bits = self.calculate_control_bits(&received_info_bits);
286
287     (
288         received_info_bits,
289         received_control_bits,
290         calculated_control_bits,
291     )
292 }

```

```

293
294 /// Inverts a single bit value (0->1 or 1->0).
295 ///
296 /// # Arguments
297 /// * `bit` - Bit value to invert (0 or 1)
298 ///
299 /// # Returns
300 /// Inverted bit value
301 fn invert_bit(&self, bit: u8) -> u8 {
302     if bit == 1 {
303         0
304     } else {
305         1
306     }
307 }
308
309 /// Extracts syndrome bits for error detection from control bytes.
310 ///
311 /// A syndrome is a pattern that indicates the presence and location of errors.
312 /// This method extracts syndrome bits for a specific bit position across
313 /// all bytes.
314 ///
315 /// # Arguments
316 /// * `syndromes` - Vector of syndrome bytes
317 /// * `bit_index` - Which bit position to extract (0-7)
318 ///
319 /// # Returns
320 /// Vector of 8 syndrome bits for the specified position
321 fn get_syndrome(&self, syndromes: &[u8], index: usize) -> Vec<u8> {
322     let mut syndrome_bits = vec![0u8; 8];
323     for i in 0..8 {
324         let bits = self.u8_to_bits(syndromes[i]);
325         syndrome_bits[i] = bits[index];
326     }
327     syndrome_bits
328 }
329
330 /// Converts a syndrome pattern into an error position index.
331 ///
332 /// Maps syndrome patterns to specific bit positions where errors occurred.
333 /// Returns -1 for invalid or multiple error patterns.
334 ///
335 /// # Arguments
336 /// * `syndrome` - Byte containing the syndrome pattern
337 /// * `stats` - Statistics object to update with error information
338 ///
339 /// # Returns
340 /// * `-1` for invalid/multiple errors
341 /// * `0-7` indicating the bit position of a single error

```

```

342 fn syndrome_to_index(&self, syndrome: u8, stats: &mut Stats) -> i32 {
343     match syndrome {
344         0 => -1, // No error
345         1 | 2 | 4 | 8 => {
346             stats.parity_bits_errors += 1;
347             -1
348         }
349         3 => {
350             stats.corrected_informational_bits += 1;
351             0
352         }
353         5 => {
354             stats.corrected_informational_bits += 1;
355             1
356         }
357         6 => {
358             stats.corrected_informational_bits += 1;
359             2
360         }
361         7 => {
362             stats.corrected_informational_bits += 1;
363             3
364         }
365         9 => {
366             stats.corrected_informational_bits += 1;
367             4
368         }
369         10 => {
370             stats.corrected_informational_bits += 1;
371             5
372         }
373         11 => {
374             stats.corrected_informational_bits += 1;
375             6
376         }
377         12 => {
378             stats.corrected_informational_bits += 1;
379             7
380         }
381         _ => {
382             stats.multiple_errors += 1;
383             -1
384         }
385     }
386 }
387
388 /// Encodes data using Hamming code with additional bit shifting.
389 ///
390 /// The encoding process:

```

```

391 /// 1. Prepares input data by padding if necessary
392 /// 2. Calculates parity (control) bits
393 /// 3. Combines data and parity bits
394 /// 4. Applies horizontal and vertical shifts for better error protection
395 ///
396 /// # Arguments
397 /// * `data` - Input data to encode (up to 16 bytes)
398 ///
399 /// # Returns
400 /// 24-byte vector containing the encoded data
401 pub fn encode(&mut self, data: &[u8]) -> Vec<u8> {
402     self.clear_steps();
403     // Prepare data
404     let mut data_bytes = data.to_vec();
405     data_bytes.resize(self.data_size, 0);
406
407     self.add_step(
408         "Initial data preparation".to_string(),
409         data_bytes.clone(),
410         StepType::DataPreparation,
411     );
412     // Calculate and add control bits
413     let control_bytes = self.calculate_control_bits(&data_bytes);
414     let mut transmit_buffer = Vec::with_capacity(self.buffer_size);
415     transmit_buffer.extend_from_slice(&data_bytes);
416     transmit_buffer.extend_from_slice(&control_bytes);
417
418     self.add_step(
419         "Calculated parity bits".to_string(),
420         transmit_buffer.clone(),
421         StepType::ParityCalculation,
422     );
423
424     // Apply shifts for error protection
425     let transmit_buffer = self.horizontal_shift(&transmit_buffer, 1);
426     self.add_step(
427         "Applied horizontal shift".to_string(),
428         transmit_buffer.clone(),
429         StepType::HorizontalShift,
430     );
431
432     let transmit_buffer = self.vertical_shift(&transmit_buffer, 1);
433     self.add_step(
434         "Applied vertical shift".to_string(),
435         transmit_buffer.clone(),
436         StepType::VerticalShift,
437     );
438
439     transmit_buffer

```

```

440 }
441
442 /// Decodes data and performs error correction if possible.
443 ///
444 /// The decoding process:
445 /// 1. Reverses vertical and horizontal shifts
446 /// 2. Separates data and control bits
447 /// 3. Calculates syndromes to detect errors
448 /// 4. Attempts to correct single-bit errors
449 /// 5. Detects multiple-bit errors
450 ///
451 /// # Arguments
452 /// * `received_data` - 24-byte encoded data to decode
453 ///
454 /// # Returns
455 /// Tuple containing:
456 /// - Decoded data (16 bytes)
457 /// - Statistics about errors found and corrections made
458 pub fn decode(&mut self, received_data: &[u8]) -> (Vec<u8>, Stats) {
459     self.clear_steps();
460     let mut stats = Stats::default();
461     let mut receive_buffer = received_data.to_vec();
462
463     self.add_step(
464         "Received data".to_string(),
465         receive_buffer.clone(),
466         StepType::DataPreparation,
467     );
468
469     // Reverse the shifts
470     receive_buffer = self.vertical_shift(&receive_buffer, -1);
471     self.add_step(
472         "Reversed vertical shift".to_string(),
473         receive_buffer.clone(),
474         StepType::VerticalShift,
475     );
476
477     receive_buffer = self.horizontal_shift(&receive_buffer, -1);
478     self.add_step(
479         "Reversed horizontal shift".to_string(),
480         receive_buffer.clone(),
481         StepType::HorizontalShift,
482     );
483
484     // Split and check for errors
485     let (received_info_bits, received_control_bits, calculated_control_bits) =
486         self.split_receive_buffer(&receive_buffer);
487
488     self.add_step(

```

```

489         "Split buffer and calculated syndromes(Received Info bits)".to_string(),
490         received_info_bits.clone(),
491         StepType::ErrorDetection,
492     );
493
494     self.add_step(
495         "Split buffer and calculated syndromes(Received Control bits)".to_string(),
496         received_control_bits.clone(),
497         StepType::ErrorDetection,
498     );
499
500     self.add_step(
501         "Split buffer and calculated syndromes(Calculated Control bits)".to_string(),
502         calculated_control_bits.clone(),
503         StepType::ErrorDetection,
504     );
505
506     let syndromes: Vec<u8> = received_control_bits
507         .iter()
508         .zip(calculated_control_bits.iter())
509         .map(|(&a, &b)| a ^ b)
510         .collect();
511
512     self.add_step(
513         "Calculated syndromes".to_string(),
514         syndromes.clone(),
515         StepType::ErrorDetection,
516     );
517
518     let mut matrix = Matrix2D::new(8, self.data_size);
519     let mut error_found = false;
520
521     // Process each bit position
522     for i in 0..8 {
523         let syndrome_bits = self.get_syndrome(&syndromes, i);
524         let left_syndrome = self.bits_to_u8(&syndrome_bits[..4]);
525         let right_syndrome = self.bits_to_u8(&syndrome_bits[4..]);
526
527         if left_syndrome != 0 || right_syndrome != 0 {
528             error_found = true;
529         }
530
531         let left_error_index = self.syndrome_to_index(left_syndrome, &mut stats);
532         let right_error_index = self.syndrome_to_index(right_syndrome, &mut stats);
533
534         for j in 0..self.data_size {
535             let bits = self.u8_to_bits(received_info_bits[j]);
536             let error_index = if j >= 8 {
537                 right_error_index

```



```

538         } else {
539             left_error_index
540         };
541
542         let should_invert = error_index >= 0 && j % 8 == error_index as usize;
543         if should_invert {
544             self.add_step(
545                 format!("Found error at position ({}, {})", i, j),
546                 received_info_bits.clone(),
547                 StepType::ErrorCorrection,
548             );
549         }
550
551         let bit_to_set = if should_invert {
552             self.invert_bit(bits[i])
553         } else {
554             bits[i]
555         };
556         matrix.set(7 - i, j, bit_to_set);
557     }
558 }
559
560 if !error_found {
561     stats = Stats::default();
562 }
563
564 let decoded_data = self.matrix_to_bytes(&matrix, self.data_size);
565 self.add_step(
566     "Final decoded data".to_string(),
567     decoded_data.clone(),
568     StepType::FinalResult,
569 );
570
571 (decoded_data, stats)
572 }
573 }

```

Файл: ./src/lib/matrix2d.rs

```

1  /// Two-dimensional matrix implementation with efficient rotation operations.
2  ///
3  /// This module provides a matrix structure that uses a flat vector for storage
4  /// and maintains rotation offsets to perform efficient row and column rotations
5  /// without actually moving data in memory.
6
7  /// A 2D matrix implementation with efficient rotation capabilities.
8  ///
9  /// Uses a flat vector for storage and maintains separate offset vectors
10 /// for rows and columns to enable efficient rotations.
11 #[derive(Debug, Clone)]
12 pub struct Matrix2D<T> {
13     /// Flat storage of matrix elements in row-major order
14     pub(crate) data: Vec<T>,
15     /// Number of rows in the matrix
16     pub(crate) rows: usize,
17     /// Number of columns in the matrix
18     pub(crate) cols: usize,
19     /// Starting offset for each row after rotations
20     pub(crate) row_starts: Vec<usize>,
21     /// Starting offset for each column after rotations
22     pub(crate) col_starts: Vec<usize>,
23 }
24
25 impl<T: Clone + Default> Default for Matrix2D<T> {
26     fn default() -> Self {
27         Self::new(0, 0)
28     }
29 }
30
31 impl<T: Clone + Default> Matrix2D<T> {
32     /// Creates a new matrix from a vector of vectors.
33     ///
34     /// Converts a 2D vector into a flat matrix representation.
35     /// Shorter rows are padded with default values.
36     ///
37     /// Time: O(r*c) - needs to copy all elements
38     /// Space: O(r*c) - stores all elements in flat array
39     ///
40     /// # Arguments
41     /// * `data` - Vector of vectors containing matrix data
42     ///
43     /// # Returns
44     /// New Matrix2D instance with the provided data
45     pub fn from_vec(data: Vec<Vec<T>>) -> Self {
46         let rows = data.len();
47         if rows == 0 {

```

```

48         return Self::new(0, 0);
49     }
50
51     // Find the longest row length
52     let cols = data.iter().map(|row| row.len()).max().unwrap_or(0);
53     if cols == 0 {
54         return Self::new(rows, 0);
55     }
56
57     // Flatten the 2D vector into 1D storage with padding
58     let mut flat_data = Vec::with_capacity(rows * cols);
59     for mut row in data {
60         // Pad shorter rows with default values
61         while row.len() < cols {
62             row.push(T::default());
63         }
64         flat_data.extend(row);
65     }
66
67     Matrix2D {
68         data: flat_data,
69         rows,
70         cols,
71         row_starts: vec![0; rows],
72         col_starts: vec![0; cols],
73     }
74 }
75
76 /// Creates an empty matrix with specified dimensions.
77 ///
78 /// Initializes all elements to their default value.
79 ///
80 /// Time: O(r*c) - initializes default values
81 /// Space: O(r*c) - flat array storage
82 ///
83 /// # Arguments
84 /// * `rows` - Number of rows
85 /// * `cols` - Number of columns
86 pub fn new(rows: usize, cols: usize) -> Self
87 where
88     T: Default,
89 {
90     let data = vec![T::default(); rows * cols];
91     Matrix2D {
92         data,
93         rows,
94         cols,
95         row_starts: vec![0; rows],
96         col_starts: vec![0; cols],

```

```

97     }
98 }
99
100 // Get dimensions
101 pub fn dimensions(&self) -> (usize, usize) {
102     (self.rows, self.cols)
103 }
104
105 /// Gets a reference to the element at specified position.
106 ///
107 /// Takes into account row and column rotations when accessing elements.
108 ///
109 /// Time:  $O(1)$  - constant time index calculation
110 /// Space:  $O(1)$  - no additional storage
111 ///
112 /// # Arguments
113 /// * `row` - Row index
114 /// * `col` - Column index
115 ///
116 /// # Returns
117 /// Option containing reference to element if indices are valid
118 pub fn get(&self, row: usize, col: usize) -> Option<&T> {
119     if row >= self.rows || col >= self.cols {
120         return None;
121     }
122
123     let row_after_col_rotation = (row + self.rows - self.col_starts[col]) % self.rows;
124     let col_after_row_rotation = (col + self.row_starts[row_after_col_rotation]) % self.cols;
125
126     let index = row_after_col_rotation * self.cols + col_after_row_rotation;
127     Some(&self.data[index])
128 }
129
130 /// Sets the value at specified position.
131 ///
132 /// Takes into account row and column rotations when setting elements.
133 ///
134 /// Time:  $O(1)$  - constant time index calculation
135 /// Space:  $O(1)$  - no additional storage
136 ///
137 /// # Arguments
138 /// * `row` - Row index
139 /// * `col` - Column index
140 /// * `value` - New value to set
141 pub fn set(&mut self, row: usize, col: usize, value: T) -> bool {
142     if row >= self.rows || col >= self.cols {
143         return false;
144     }
145

```

```

146     let row_after_col_rotation = (row + self.rows - self.col_starts[col]) % self.rows;
147     let col_after_row_rotation = (col + self.row_starts[row_after_col_rotation]) % self.cols;
148
149     let index = row_after_col_rotation * self.cols + col_after_row_rotation;
150     self.data[index] = value;
151     true
152 }
153
154 /// Rotates a row right by specified amount.
155 ///
156 /// Updates row offset without moving data in memory.
157 ///
158 /// Time: O(1) - only updates offset
159 /// Space: O(1) - modifies existing offset array
160 ///
161 /// # Arguments
162 /// * `row` - Row index to rotate
163 /// * `shift` - Number of positions to rotate right
164 pub fn rotate_row_right(&mut self, row: usize, k: usize) {
165     if row >= self.rows || self.cols == 0 {
166         return;
167     }
168     let k = k % self.cols;
169     self.row_starts[row] = (self.row_starts[row] + self.cols - k) % self.cols;
170 }
171
172 /// Rotates a row left by specified amount.
173 ///
174 /// Updates row offset without moving data in memory.
175 ///
176 /// Time: O(1) - only updates offset
177 /// Space: O(1) - modifies existing offset array
178 ///
179 /// # Arguments
180 /// * `row` - Row index to rotate
181 /// * `shift` - Number of positions to rotate left
182 pub fn rotate_row_left(&mut self, row: usize, k: usize) {
183     if row >= self.rows || self.cols == 0 {
184         return;
185     }
186     let k = k % self.cols;
187     self.row_starts[row] = (self.row_starts[row] + k) % self.cols;
188 }
189
190 /// Rotates a column down by specified amount.
191 ///
192 /// Updates column offset without moving data in memory.
193 ///
194 /// Time: O(1) - only updates offset

```

```

195 /// Space: O(1) - modifies existing offset array
196 ///
197 /// # Arguments
198 /// * `col` - Column index to rotate
199 /// * `shift` - Number of positions to rotate down
200 pub fn rotate_column_down(&mut self, col: usize, k: usize) {
201     if col >= self.cols || self.rows == 0 {
202         return;
203     }
204     let k = k % self.rows;
205     self.col_starts[col] = (self.col_starts[col] + k) % self.rows;
206 }
207
208 /// Rotates a column up by specified amount.
209 ///
210 /// Updates column offset without moving data in memory.
211 ///
212 /// Time: O(1) - only updates offset
213 /// Space: O(1) - modifies existing offset array
214 ///
215 /// # Arguments
216 /// * `col` - Column index to rotate
217 /// * `shift` - Number of positions to rotate up
218 pub fn rotate_column_up(&mut self, col: usize, k: usize) {
219     if col >= self.cols || self.rows == 0 {
220         return;
221     }
222     let k = k % self.rows;
223     self.col_starts[col] = (self.col_starts[col] + self.rows - k) % self.rows;
224 }
225
226 // Convert to Vec<Vec<T>>
227 // Time: O(r*c) - must copy all elements
228 // Space: O(r*c) - creates new 2D vector
229 pub fn to_vec(&self) -> Vec<Vec<T>>
230 where
231     T: Clone,
232 {
233     let mut result = Vec::with_capacity(self.rows);
234     for row in 0..self.rows {
235         let mut row_vec = Vec::with_capacity(self.cols);
236         for col in 0..self.cols {
237             row_vec.push(self.get(row, col).unwrap().clone());
238         }
239         result.push(row_vec);
240     }
241     result
242 }
243

```

```
244 // Display the matrix
245 // Time:  $O(r*c)$  - prints all elements
246 // Space:  $O(1)$  - no additional storage
247 pub fn display(&self)
248 where
249     T: std::fmt::Display,
250 {
251     for row in 0..self.rows {
252         for col in 0..self.cols {
253             print!("{}", self.get(row, col).unwrap());
254         }
255         println();
256     }
257     println();
258 }
259 }
```

Файл: ./src/lib/mod.rs

```
1 /// Hamming Code implementation for error detection and correction in data transmission.
2 ///
3 /// This crate provides:
4 /// - Hamming code encoding and decoding
5 /// - Error detection and correction capabilities
6 /// - Matrix operations for data manipulation
7 /// - GUI interface for visualization
8
9 pub mod codec;
10 pub mod matrix2d;
```


Файл: ./tests/codec.rs

```

1 use hammingcodec::codec::{HammingCodec, Stats, StepType};
2
3 #[test]
4 fn test_encode_decode_no_errors() {
5     let mut codec = HammingCodec::new();
6     let original = b"Hello, World!".to_vec();
7     let encoded = codec.encode(&original);
8     let (decoded, stats) = codec.decode(&encoded);
9     assert_eq!(&original[..], &decoded[..original.len()]);
10    assert_eq!(stats.corrected_informational_bits, 0);
11    assert_eq!(stats.multiple_errors, 0);
12    assert_eq!(stats.parity_bits_errors, 0);
13 }
14
15 #[test]
16 fn test_error_correction() {
17     let mut codec = HammingCodec::new();
18     let original = b"Hello, World!".to_vec();
19     let mut encoded = codec.encode(&original);
20
21     // Introduce a single bit error
22     encoded[0] ^= 1;
23
24     let (decoded, stats) = codec.decode(&encoded);
25     assert_eq!(&original[..], &decoded[..original.len()]);
26     assert!(stats.corrected_informational_bits > 0 || stats.parity_bits_errors > 0);
27     assert_eq!(stats.multiple_errors, 0);
28 }
29
30 #[test]
31 fn test_multiple_errors() {
32     let mut codec = HammingCodec::new();
33     let original = b"Hello, World!".to_vec();
34     let mut encoded = codec.encode(&original);
35
36     // Introduce multiple bit errors
37     encoded[0] ^= 3; // Sets two bits
38
39     let (decoded, stats) = codec.decode(&encoded);
40     assert!(
41         stats.corrected_informational_bits > 0
42         || stats.parity_bits_errors > 0
43         || stats.multiple_errors > 0
44     );
45 }
46
47 #[test]

```

```

48 fn test_bit_conversion() {
49     let codec = HammingCodec::new();
50     let value = 0b10110001u8;
51     let bits = codec.u8_to_bits(value);
52     let result = codec.bits_to_u8(&bits);
53     assert_eq!(value, result);
54 }
55
56 #[test]
57 fn test_control_bits() {
58     let codec = HammingCodec::new();
59     let mut data = vec![0u8; 16]; // Use full data size
60     data[0] = 0b10110001u8;
61     data[1] = 0b11001100u8;
62     let control = codec.calculate_control_bits(&data);
63     assert_eq!(control.len(), 8); // Should have 8 control bytes
64     assert!(!control.iter().all(|&x| x == 0)); // At least some bits should be set
65 }
66
67 #[test]
68 fn test_default_codec() {
69     let mut codec = HammingCodec::default();
70     let data = vec![1, 2, 3];
71     let encoded = codec.encode(&data);
72     assert_eq!(encoded.len(), 24); // buffer_size
73     let (_decoded, _) = codec.decode(&encoded);
74     assert_eq!(_decoded.len(), 16); // data_size
75 }
76
77 #[test]
78 fn test_clear_steps() {
79     let mut codec = HammingCodec::new();
80     let data = vec![1, 2, 3];
81     codec.encode(&data);
82     assert!(!codec.steps.is_empty());
83     codec.clear_steps();
84     assert!(codec.steps.is_empty());
85 }
86
87 #[test]
88 fn test_bits_conversion() {
89     let codec = HammingCodec::new();
90
91     // Test u8_to_bits
92     let bits = codec.u8_to_bits(0b10101010);
93     assert_eq!(bits, vec![0, 1, 0, 1, 0, 1, 0, 1]);
94
95     // Test bits_to_u8
96     let value = codec.bits_to_u8(&[0, 1, 0, 1, 0, 1, 0, 1]);

```

```

97     assert_eq!(value, 0b10101010);
98 }
99
100 #[test]
101 fn test_step_tracking() {
102     let mut codec = HammingCodec::new();
103     let data = vec![1, 2, 3];
104     let _encoded = codec.encode(&data);
105
106     // Check step descriptions
107     assert!(codec
108         .steps
109         .iter()
110         .any(|s| s.description == "Initial data preparation"));
111     assert!(codec
112         .steps
113         .iter()
114         .any(|s| s.description == "Calculated parity bits"));
115     assert!(codec
116         .steps
117         .iter()
118         .any(|s| s.description == "Applied horizontal shift"));
119     assert!(codec
120         .steps
121         .iter()
122         .any(|s| s.description == "Applied vertical shift"));
123 }
124
125 #[test]
126 fn test_stats_tracking() {
127     let mut codec = HammingCodec::new();
128     let data = vec![1, 2, 3];
129     let encoded = codec.encode(&data);
130
131     // Test with no errors
132     let (_, stats) = codec.decode(&encoded);
133     assert_eq!(stats.corrected_informational_bits, 0);
134     assert_eq!(stats.multiple_errors, 0);
135     assert_eq!(stats.parity_bits_errors, 0);
136
137     // Test with single error
138     let mut corrupted = encoded.clone();
139     corrupted[0] ^= 1; // Flip one bit
140     let (_, stats) = codec.decode(&corrupted);
141     assert_eq!(stats.corrected_informational_bits, 1);
142     assert_eq!(stats.multiple_errors, 0);
143
144     // Test with multiple errors
145     let mut multi_corrupted = encoded.clone();

```

```

146 // Create an invalid syndrome pattern by corrupting specific bits
147 multi_corrupted[16] ^= 0b1111; // Corrupt parity bits
148 let (_, stats) = codec.decode(&multi_corrupted);
149 assert!(stats.multiple_errors > 0 || stats.parity_bits_errors > 0);
150 }
151
152 #[test]
153 fn test_empty_input() {
154     let mut codec = HammingCodec::new();
155     let encoded = codec.encode(&[]);
156     assert_eq!(encoded.len(), 24); // buffer_size
157
158     let (decoded, stats) = codec.decode(&encoded);
159     assert_eq!(decoded.len(), 16); // data_size
160     assert_eq!(stats.corrected_informational_bits, 0);
161 }
162
163 #[test]
164 fn test_parity_bit_errors() {
165     let mut codec = HammingCodec::new();
166     let data = vec![1, 2, 3];
167     let encoded = codec.encode(&data);
168
169     // Test parity bit errors (syndromes 1, 2, 4, 8)
170     let mut corrupted = encoded.clone();
171     corrupted[16] ^= 0b1; // Create a parity bit error
172     let (_, stats) = codec.decode(&corrupted);
173     assert_eq!(stats.parity_bits_errors, 1);
174     assert_eq!(stats.corrected_informational_bits, 0);
175     assert_eq!(stats.multiple_errors, 0);
176 }
177
178 #[test]
179 fn test_no_errors_found() {
180     let mut codec = HammingCodec::new();
181     let data = vec![1, 2, 3];
182     let encoded = codec.encode(&data);
183
184     // Test case where error_found remains false
185     let (decoded, stats) = codec.decode(&encoded);
186     assert_eq!(stats.corrected_informational_bits, 0);
187     assert_eq!(stats.multiple_errors, 0);
188     assert_eq!(stats.parity_bits_errors, 0);
189     assert_eq!(&decoded[..data.len()], &data[..]);
190 }
191
192 #[test]
193 fn test_all_syndrome_patterns() {
194     let mut codec = HammingCodec::new();

```

```

195 let data = vec![1, 2, 3];
196 let encoded = codec.encode(&data);
197
198 // Test with no errors
199 let (_, stats) = codec.decode(&encoded);
200 assert_eq!(stats.corrected_informational_bits, 0);
201 assert_eq!(stats.multiple_errors, 0);
202 assert_eq!(stats.parity_bits_errors, 0);
203
204 // Test parity bit errors
205 let mut corrupted = encoded.clone();
206 corrupted[16] ^= 1; // Set first parity bit error
207 let (_, stats) = codec.decode(&corrupted);
208 assert_eq!(stats.parity_bits_errors, 1);
209 assert_eq!(stats.corrected_informational_bits, 0);
210 assert_eq!(stats.multiple_errors, 0);
211
212 // Test single correctable error
213 let mut corrupted = encoded.clone();
214 corrupted[0] ^= 1; // Single bit error
215 let (_, stats) = codec.decode(&corrupted);
216 assert_eq!(stats.corrected_informational_bits, 1);
217 assert_eq!(stats.multiple_errors, 0);
218 assert_eq!(stats.parity_bits_errors, 0);
219
220 // Test multiple errors by creating errors in the same bit position in different bytes
221 let mut corrupted = encoded.clone();
222 println!("Original encoded data: {:?}", encoded);
223 corrupted[2] ^= 0b0000_0100; // First error in bit 0 of first byte
224 corrupted[4] ^= 0b0001_0000;
225 println!("Corrupted data: {:?}", corrupted);
226 let (decoded, stats) = codec.decode(&corrupted);
227 println!("Decoded data: {:?}", decoded);
228 println!(
229     "Stats: {{ corrected_bits: {}, multiple_errors: {}, parity_errors: {} }}",
230     stats.corrected_informational_bits, stats.multiple_errors, stats.parity_bits_errors
231 );
232 assert!(
233     stats.multiple_errors > 0,
234     "Expected multiple errors to be detected"
235 );
236 assert_eq!(stats.corrected_informational_bits, 0);
237 assert_eq!(stats.parity_bits_errors, 0);
238 }
239
240 #[test]
241 fn test_error_position_tracking() {
242     let mut codec = HammingCodec::new();
243     let data = vec![1, 2, 3];

```

```

244 let encoded = codec.encode(&data);
245
246 // Create an error that should be detected and corrected
247 let mut corrupted = encoded.clone();
248 corrupted[0] ^= 1; // Flip one bit
249
250 let (_, _) = codec.decode(&corrupted);
251
252 // Check if error position was logged in steps
253 let error_steps: Vec<_> = codec
254     .steps
255     .iter()
256     .filter(|step| step.description.starts_with("Found error at position"))
257     .collect();
258
259 assert!(
260     !error_steps.is_empty(),
261     "Error position should be logged in steps"
262 );
263 }

```

Файл: ./tests/comprehensive.rs

```

1 use hammingcodec::codec::HammingCodec;
2 use std::collections::HashMap;
3
4 #[derive(Default)]
5 struct ErrorStats {
6     corrected_informational_bits: usize,
7     multiple_errors: usize,
8     parity_bits_errors: usize,
9     count_experiments: usize,
10 }
11
12 #[test]
13 fn test_comprehensive_error_correction() {
14     let mut codec = HammingCodec::new();
15     let original = b"Hello, World!".to_vec();
16     let encoded = codec.encode(&original);
17
18     // Initialize global statistics
19     let mut global_stats: HashMap<usize, ErrorStats> = HashMap::new();
20     for i in 1..=8 {
21         global_stats.insert(i, ErrorStats::default());
22     }
23
24     // Test each position in the encoded data
25     for pos in 0..encoded.len() {
26         // Test each possible error pattern (1 to 255)
27         for error_pattern in 1..=255 {
28             // Create a copy of the encoded data
29             let mut test_data = encoded.clone();
30
31             // Apply the error pattern to the current position
32             test_data[pos] ^= error_pattern;
33
34             // Count the number of bits set in the error pattern
35             let error_count = error_pattern.count_ones() as usize;
36
37             // Decode and collect statistics
38             let (decoded, stats) = codec.decode(&test_data);
39
40             // Update global statistics
41             let global_stat = global_stats.get_mut(&error_count).unwrap();
42             global_stat.corrected_informational_bits += stats.corrected_informational_bits;
43             global_stat.multiple_errors += stats.multiple_errors;
44             global_stat.parity_bits_errors += stats.parity_bits_errors;
45             global_stat.count_experiments += 1;
46
47             // Print current test details

```

```

48     println!(
49         "
        ↪ Position:{:02}| Error Vector:{:08b}| {} | Corrected Info Bits:{} | Parity Bits Errors:{} | Multiple Errors:{}
        ↪ ",
50         pos,
51         error_pattern,
52         if stats.multiple_errors == 0 { "Passed" } else { "Failed" },
53         stats.corrected_informational_bits,
54         stats.parity_bits_errors,
55         stats.multiple_errors
56     );
57
58     // Verify correction was successful if no multiple errors
59     if stats.multiple_errors == 0 {
60         assert_eq!(&original[..], &decoded[..original.len()]);
61     }
62 }
63 }
64
65 // Print results
66 println!("\n\n\nTest Results:\n\n");
67 println!(
68     "Index | CountExperiments | CorrectedInformationalBits | MultipleErrors | ParityBitsErrors"
69 );
70 for i in 1..=8 {
71     let stats = global_stats.get(&i).unwrap();
72     println!(
73         "{} | {} | {} | {} | {}",
74         i,
75         stats.count_experiments,
76         stats.corrected_informational_bits,
77         stats.multiple_errors,
78         stats.parity_bits_errors
79     );
80
81     // Add some basic assertions to verify expected behavior
82     if i == 1 {
83         // Single bit errors should always be correctable
84         assert_eq!(stats.multiple_errors, 0);
85         assert!(stats.corrected_informational_bits > 0 || stats.parity_bits_errors > 0);
86     } else {
87         // Multiple bit errors should be either corrected or detected
88         assert!(
89             stats.corrected_informational_bits > 0
90             || stats.parity_bits_errors > 0
91             || stats.multiple_errors > 0
92         );
93     }
94 }

```



```

95 }
96
97 // Additional test to cover edge cases
98 #[test]
99 fn test_comprehensive_edge_cases() {
100     let mut codec = HammingCodec::new();
101     let original = b"Test".to_vec();
102     let encoded = codec.encode(&original);
103
104     // Test with no errors
105     let (decoded, stats) = codec.decode(&encoded);
106     assert_eq!(&original[..], &decoded[..original.len()]);
107     assert_eq!(stats.corrected_informational_bits, 0);
108     assert_eq!(stats.multiple_errors, 0);
109     assert_eq!(stats.parity_bits_errors, 0);
110
111     // Test with maximum error pattern
112     let mut test_data = encoded.clone();
113     test_data[0] ^= 0xFF; // All bits flipped
114     let (decoded, stats) = codec.decode(&test_data);
115     assert!(stats.multiple_errors == 0);
116
117     // Test with error in last position
118     let mut test_data = encoded.clone();
119     test_data[encoded.len() - 1] ^= 1;
120     let (decoded, stats) = codec.decode(&test_data);
121     assert!(stats.corrected_informational_bits > 0 || stats.parity_bits_errors > 0);
122 }
123
124 // Test to verify error statistics consistency
125 #[test]
126 fn test_error_statistics_consistency() {
127     let mut codec = HammingCodec::new();
128     let original = b"A".to_vec(); // Single character to keep test duration reasonable
129     let encoded = codec.encode(&original);
130
131     let mut stats_map: HashMap<(usize, u8), (usize, usize, usize)> = HashMap::new();
132
133     // Test each position with each single-bit error
134     for pos in 0..encoded.len() {
135         for bit in 0..8 {
136             let error_pattern = 1 << bit;
137             let mut test_data = encoded.clone();
138             test_data[pos] ^= error_pattern;
139
140             let (_, stats) = codec.decode(&test_data);
141
142             // Store statistics for this position and error pattern
143             stats_map.insert(

```

```

144         (pos, error_pattern),
145         (
146             stats.corrected_informational_bits,
147             stats.multiple_errors,
148             stats.parity_bits_errors,
149         ),
150     );
151
152     // Verify same error pattern produces consistent results
153     let mut test_data2 = encoded.clone();
154     test_data2[pos] ^= error_pattern;
155     let (_, stats2) = codec.decode(&test_data2);
156
157     assert_eq!(
158         stats.corrected_informational_bits,
159         stats2.corrected_informational_bits
160     );
161     assert_eq!(stats.multiple_errors, stats2.multiple_errors);
162     assert_eq!(stats.parity_bits_errors, stats2.parity_bits_errors);
163 }
164 }
165 }

```

Файл: ./tests/matrix2d.rs

```

1 use hammingcodec::matrix2d::Matrix2D;
2
3 #[test]
4 fn test_matrix_creation() {
5     let data = vec![vec![1, 2, 3], vec![4, 5, 6]];
6     let matrix = Matrix2D::from_vec(data);
7     assert_eq!(matrix.dimensions(), (2, 3));
8 }
9
10 #[test]
11 fn test_matrix_set_get() {
12     let mut matrix: Matrix2D<i32> = Matrix2D::new(2, 2);
13     matrix.set(0, 0, 1);
14     matrix.set(0, 1, 2);
15     matrix.set(1, 0, 3);
16     matrix.set(1, 1, 4);
17
18     assert_eq!(matrix.get(0, 0).unwrap(), &1);
19     assert_eq!(matrix.get(0, 1).unwrap(), &2);
20     assert_eq!(matrix.get(1, 0).unwrap(), &3);
21     assert_eq!(matrix.get(1, 1).unwrap(), &4);
22 }
23
24 #[test]
25 fn test_matrix_bounds() {
26     let matrix: Matrix2D<i32> = Matrix2D::new(2, 2);
27     assert!(matrix.get(2, 0).is_none());
28     assert!(matrix.get(0, 2).is_none());
29     assert!(matrix.get(2, 2).is_none());
30 }
31
32 #[test]
33 fn test_row_rotation() {
34     let data = vec![vec![1, 2, 3], vec![4, 5, 6]];
35     let mut matrix = Matrix2D::from_vec(data);
36
37     matrix.rotate_row_right(0, 1);
38     let result = matrix.to_vec();
39     assert_eq!(result[0], vec![3, 1, 2]);
40     assert_eq!(result[1], vec![4, 5, 6]);
41 }
42
43 #[test]
44 fn test_column_rotation() {
45     let data = vec![vec![1, 2, 3], vec![4, 5, 6], vec![7, 8, 9]];
46     let mut matrix = Matrix2D::from_vec(data);
47

```

```

48     matrix.rotate_column_down(0, 1);
49     let result = matrix.to_vec();
50     assert_eq!(result[0][0], 7);
51     assert_eq!(result[1][0], 1);
52     assert_eq!(result[2][0], 4);
53 }
54
55 #[test]
56 fn test_combined_rotations() {
57     let data = vec![vec![1, 2, 3], vec![4, 5, 6], vec![7, 8, 9]];
58     let mut matrix = Matrix2D::from_vec(data);
59
60     matrix.rotate_row_right(0, 1);
61     matrix.rotate_column_down(0, 1);
62
63     let result = matrix.to_vec();
64     assert_eq!(result[0][0], 7);
65     assert_eq!(result[0][1], 1);
66     assert_eq!(result[0][2], 2);
67 }
68
69 #[test]
70 fn test_empty_matrix() {
71     let matrix: Matrix2D<i32> = Matrix2D::new(0, 0);
72     assert_eq!(matrix.dimensions(), (0, 0));
73 }
74
75 #[test]
76 fn test_large_rotation_values() {
77     let data = vec![vec![1, 2], vec![3, 4]];
78     let mut matrix = Matrix2D::from_vec(data);
79
80     // Rotating by size of matrix should give same result
81     matrix.rotate_row_right(0, 2);
82     let result = matrix.to_vec();
83     assert_eq!(result[0], vec![1, 2]);
84 }
85
86 #[test]
87 fn test_default_matrix() {
88     let matrix: Matrix2D<i32> = Matrix2D::default();
89     assert_eq!(matrix.dimensions(), (0, 0));
90 }
91
92 #[test]
93 fn test_empty_from_vec() {
94     let data: Vec<Vec<i32>> = vec![];
95     let matrix = Matrix2D::from_vec(data);
96     assert_eq!(matrix.dimensions(), (0, 0));

```

```

97 }
98
99 #[test]
100 fn test_rotate_empty_matrix() {
101     let mut matrix: Matrix2D<i32> = Matrix2D::new(0, 0);
102     matrix.rotate_row_right(0, 1); // Should not panic
103     matrix.rotate_row_left(0, 1); // Should not panic
104     matrix.rotate_column_up(0, 1); // Should not panic
105     matrix.rotate_column_down(0, 1); // Should not panic
106 }
107
108 #[test]
109 fn test_invalid_rotations() {
110     let mut matrix = Matrix2D::new(2, 2);
111     matrix.set(0, 0, 1);
112     matrix.set(0, 1, 2);
113     matrix.set(1, 0, 3);
114     matrix.set(1, 1, 4);
115
116     // Test invalid row rotations
117     matrix.rotate_row_right(5, 1); // Invalid row index
118     matrix.rotate_row_left(5, 1); // Invalid row index
119
120     // Test invalid column rotations
121     matrix.rotate_column_up(5, 1); // Invalid column index
122     matrix.rotate_column_down(5, 1); // Invalid column index
123
124     // Verify matrix wasn't changed
125     assert_eq!(matrix.get(0, 0), Some(&1));
126     assert_eq!(matrix.get(0, 1), Some(&2));
127     assert_eq!(matrix.get(1, 0), Some(&3));
128     assert_eq!(matrix.get(1, 1), Some(&4));
129 }
130
131 #[test]
132 fn test_display_output() {
133     let mut matrix = Matrix2D::new(2, 2);
134     matrix.set(0, 0, 1);
135     matrix.set(0, 1, 2);
136     matrix.set(1, 0, 3);
137     matrix.set(1, 1, 4);
138
139     // Just call display to ensure it doesn't panic
140     matrix.display();
141 }
142
143 #[test]
144 fn test_large_rotations() {
145     let mut matrix = Matrix2D::new(2, 2);

```

```

146 matrix.set(0, 0, 1);
147 matrix.set(0, 1, 2);
148 matrix.set(1, 0, 3);
149 matrix.set(1, 1, 4);
150
151 // Test rotations larger than dimensions
152 matrix.rotate_row_right(0, 5); // Should be equivalent to rotating by 1
153 assert_eq!(matrix.get(0, 0), Some(&2));
154 assert_eq!(matrix.get(0, 1), Some(&1));
155
156 matrix.rotate_column_down(0, 6); // Should be equivalent to no rotation
157 assert_eq!(matrix.get(0, 0), Some(&2));
158 assert_eq!(matrix.get(1, 0), Some(&3));
159 }
160
161 #[test]
162 fn test_invalid_get_set() {
163     let mut matrix = Matrix2D::new(2, 2);
164
165     // Test invalid get
166     assert_eq!(matrix.get(2, 0), None);
167     assert_eq!(matrix.get(0, 2), None);
168     assert_eq!(matrix.get(2, 2), None);
169
170     // Test invalid set
171     assert_eq!(matrix.set(2, 0, 1), false);
172     assert_eq!(matrix.set(0, 2, 1), false);
173     assert_eq!(matrix.set(2, 2, 1), false);
174 }
175
176 #[test]
177 fn test_empty_row_from_vec() {
178     let data: Vec<Vec<i32>> = vec![vec![]];
179     let matrix = Matrix2D::from_vec(data);
180     assert_eq!(matrix.dimensions(), (1, 0));
181 }
182
183 #[test]
184 fn test_uneven_rows_from_vec() {
185     let data: Vec<Vec<i32>> = vec![vec![1, 2], vec![1]];
186     let matrix = Matrix2D::from_vec(data);
187     assert_eq!(matrix.dimensions(), (2, 2)); // Should use longest row length
188 }
189
190 #[test]
191 fn test_short_rows_from_vec() {
192     let data: Vec<Vec<i32>> = vec![vec![1], vec![1, 2]];
193     let matrix = Matrix2D::from_vec(data);
194     assert_eq!(matrix.dimensions(), (2, 2)); // Should use longest row length

```

```

195     assert_eq!(matrix.get(0, 0), Some(&1));
196     assert_eq!(matrix.get(0, 1), Some(&0)); // Default value for padded cell
197     assert_eq!(matrix.get(1, 0), Some(&1));
198     assert_eq!(matrix.get(1, 1), Some(&2));
199 }
200
201 #[test]
202 fn test_uneven_rows_data_from_vec() {
203     let data: Vec<Vec<i32>> = vec![vec![1, 2], vec![3]];
204     let matrix = Matrix2D::from_vec(data);
205     assert_eq!(matrix.get(0, 0), Some(&1));
206     assert_eq!(matrix.get(0, 1), Some(&2));
207     assert_eq!(matrix.get(1, 0), Some(&3));
208 }
209
210 #[test]
211 fn test_short_rows_data_from_vec() {
212     let data: Vec<Vec<i32>> = vec![vec![1], vec![2, 3]];
213     let matrix = Matrix2D::from_vec(data);
214     assert_eq!(matrix.get(0, 0), Some(&1));
215     assert_eq!(matrix.get(1, 0), Some(&2));
216 }
217
218 #[test]
219 fn test_uneven_rows_edge_cases() {
220     // Test with first row longer than others
221     let data = vec![vec![1, 2, 3], vec![4], vec![7, 8]];
222     let matrix = Matrix2D::from_vec(data);
223     assert_eq!(matrix.dimensions(), (3, 3));
224     assert_eq!(matrix.get(0, 0), Some(&1));
225     assert_eq!(matrix.get(0, 1), Some(&2));
226     assert_eq!(matrix.get(0, 2), Some(&3));
227     assert_eq!(matrix.get(1, 0), Some(&4));
228     assert_eq!(matrix.get(2, 0), Some(&7));
229     assert_eq!(matrix.get(2, 1), Some(&8));
230
231     // Test with middle row longer than others
232     let data = vec![vec![1], vec![4, 5, 6], vec![7]];
233     let matrix = Matrix2D::from_vec(data);
234     assert_eq!(matrix.dimensions(), (3, 3));
235     assert_eq!(matrix.get(0, 0), Some(&1));
236     assert_eq!(matrix.get(1, 0), Some(&4));
237     assert_eq!(matrix.get(1, 1), Some(&5));
238     assert_eq!(matrix.get(1, 2), Some(&6));
239     assert_eq!(matrix.get(2, 0), Some(&7));
240 }
241
242 #[test]
243 fn test_zero_dimension_operations() {

```

```

244 // Test matrix with zero columns
245 let matrix: Matrix2D<i32> = Matrix2D::new(2, 0);
246 assert_eq!(matrix.dimensions(), (2, 0));
247 matrix.display(); // Should not panic
248
249 // Test matrix with zero rows
250 let matrix: Matrix2D<i32> = Matrix2D::new(0, 2);
251 assert_eq!(matrix.dimensions(), (0, 2));
252 matrix.display(); // Should not panic
253
254 // Test completely empty matrix
255 let matrix: Matrix2D<i32> = Matrix2D::new(0, 0);
256 assert_eq!(matrix.dimensions(), (0, 0));
257 matrix.display(); // Should not panic
258 }
259
260 #[test]
261 fn test_display_empty_matrix() {
262     let matrix: Matrix2D<i32> = Matrix2D::new(0, 0);
263     matrix.display(); // Should just print newline
264
265     let matrix: Matrix2D<i32> = Matrix2D::new(1, 0);
266     matrix.display(); // Should just print newline
267
268     let matrix: Matrix2D<i32> = Matrix2D::new(0, 1);
269     matrix.display(); // Should just print newline
270 }

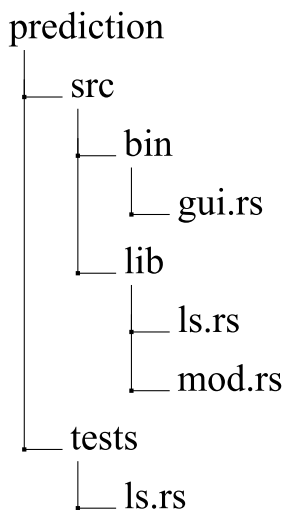
```

Файл: ./tests/mod.rs

```
1 mod codec;  
2 mod comprehensive;  
3 mod matrix2d;
```

ДОДАТОК Д Вихідний код додатку «prediction»

Структура каталогів



Програмний код Файл: ./src/bin/gui.rs

```

1 use eframe::egui::{self, Color32, RichText, Vec2};
2 use egui_plot::{Line, Plot, PlotPoints, Points};
3 use prediction::ls::{StepType, LS};
4 use rand::prelude::*;
5
6 const ICON_DATA: &str = "□";
7 const ICON_POLY: &str = "□";
8 const ICON_CALC: &str = "□";
9 const ICON_RESULT: &str = "□";
10 const ICON_PREDICT: &str = "□";
11
12 struct ThemeColors {
13     strong_text: Color32,
14 }
15
16 struct PredictionApp {
17     ls: LS,
18     n_input: String,
19     m_input: String,
20     x_predict: String,
21     error_text: String,
22     prediction_result: Option<f64>,
23     plot_points: Vec<[f64; 2]>,
24     predicted_points: Vec<[f64; 2]>,
25     function_points: Vec<[f64; 2]>,
26     critical_level_input: String,
27     critical_level: Option<f64>,
28     intersection_point: Option<[f64; 2]>,
29     plot_bounds: PlotBounds,
  
```

```

30 show_all_orders: bool,
31 all_order_points: Vec<(usize, Vec<f64; 2>)>, // (order, points)
32 cached_coefficients: Vec<(usize, Vec<f64>)>, // (order, coefficients)
33 needs_recalculation: bool,
34 // Simulation state
35 is_simulating: bool,
36 simulation_step: usize,
37 simulation_data: Vec<f64; 2>,
38 simulation_speed: f64, // seconds between steps
39 last_simulation_time: Option<std::time::Instant>,
40 simulation_progress: f32, // Progress indicator (0.0 to 1.0)
41 simulation_auto_order: bool, // Whether to automatically adjust model order
42 simulation_paused: bool, // Whether simulation is paused
43 simulation_events: Vec<(usize, String)>, // Time sample and description of significant events
44 }
45
46 struct PlotBounds {
47     x_min: f64,
48     x_max: f64,
49     y_min: f64,
50     y_max: f64,
51     base_x_min: f64,
52     base_x_max: f64,
53     base_y_min: f64,
54     base_y_max: f64,
55 }
56
57 impl Default for PlotBounds {
58     fn default() -> Self {
59         Self {
60             x_min: 0.0,
61             x_max: 15.0,
62             y_min: 447.0, // Set reasonable defaults for water level
63             y_max: 450.0, // Set reasonable defaults for water level
64             base_x_min: 0.0,
65             base_x_max: 15.0,
66             base_y_min: 447.0, // Set reasonable defaults for water level
67             base_y_max: 450.0, // Set reasonable defaults for water level
68         }
69     }
70 }
71
72 impl Default for PredictionApp {
73     fn default() -> Self {
74         let mut ls = LS::new(10, 3);
75         ls.initialize_points();
76
77         Self {
78             ls,

```

```

79     n_input: "10".to_string(),
80     m_input: "3".to_string(),
81     x_predict: "10.0".to_string(),
82     error_text: String::new(),
83     prediction_result: None,
84     plot_points: Vec::new(),
85     predicted_points: Vec::new(),
86     function_points: Vec::new(),
87     critical_level_input: "448.5".to_string(),
88     critical_level: Some(448.5),
89     intersection_point: None,
90     plot_bounds: PlotBounds::default(),
91     show_all_orders: false,
92     all_order_points: Vec::new(),
93     cached_coefficients: Vec::new(),
94     needs_recalculation: true,
95     // Initialize simulation state
96     is_simulating: false,
97     simulation_step: 0,
98     simulation_data: Vec::new(),
99     simulation_speed: 1.0,
100    last_simulation_time: None,
101    simulation_progress: 0.0,
102    simulation_auto_order: true,
103    simulation_paused: false,
104    simulation_events: Vec::new(),
105    }
106    }
107 }
108
109 impl PredictionApp {
110     fn get_theme_colors(&self, ui: &egui::Ui) -> ThemeColors {
111         ThemeColors {
112             strong_text: ui.visuals().strong_text_color(),
113         }
114     }
115
116     fn display_main_panel(&mut self, ui: &mut egui::Ui) {
117         ui.vertical_centered(|ui| {
118             // Title
119             ui.add_space(8.0);
120             ui.heading("Water Level Prediction System");
121             ui.add_space(8.0);
122
123             // Main content
124             ui.horizontal(|ui| {
125                 // Left panel - Configuration and Data
126                 ui.vertical(|ui| {
127                     let panel_width = ui.available_width().min(300.0);

```

```

128
129 // Model Configuration
130 ui.group(|ui| {
131     ui.set_width(panel_width);
132     ui.heading("Model Configuration");
133     ui.add_space(4.0);
134
135     // Model parameters
136     ui.horizontal(|ui| {
137         ui.label("Data Points:");
138         ui.add_sized(
139             [60.0, 20.0],
140             egui::TextEdit::singleline(&mut self.n_input),
141         );
142     });
143     ui.horizontal(|ui| {
144         ui.label("Model Order:");
145         ui.add_sized(
146             [60.0, 20.0],
147             egui::TextEdit::singleline(&mut self.m_input),
148         );
149     });
150     ui.checkbox(&mut self.show_all_orders, "Show all orders");
151     ui.horizontal(|ui| {
152         ui.label("Critical Level:");
153         let response = ui.add_sized(
154             [60.0, 20.0],
155             egui::TextEdit::singleline(&mut self.critical_level_input),
156         );
157         if response.changed() {
158             if let Ok(level) = self.critical_level_input.parse::<f64>() {
159                 self.critical_level = Some(level);
160                 self.needs_recalculation = true;
161             }
162         }
163     });
164 });
165
166 ui.add_space(8.0);
167
168 // Historical Data
169 ui.group(|ui| {
170     ui.set_width(panel_width);
171     ui.heading("Historical Data");
172     ui.add_space(4.0);
173
174     egui::ScrollArea::vertical()
175         .max_height(200.0)
176         .show(ui, |ui| {

```

```

177         egui::Grid::new("data_grid")
178             .striped(true)
179             .spacing([8.0, 4.0])
180             .show(ui, |ui| {
181                 ui.strong("Time samples");
182                 ui.strong("Value");
183                 ui.strong("Level");
184                 ui.end_row();
185
186                 for i in 0..self.ls.x.len() {
187                     ui.label(format!("{:}", i + 1));
188                     let mut x = format!("{:.2}", self.ls.x[i]);
189                     let mut y = format!("{:.2}", self.ls.y[i]);
190                     let x_response = ui.add_sized(
191                         [60.0, 20.0],
192                         egui::TextEdit::singleline(&mut x),
193                     );
194                     let y_response = ui.add_sized(
195                         [60.0, 20.0],
196                         egui::TextEdit::singleline(&mut y),
197                     );
198
199                     if x_response.changed() {
200                         if let Ok(value) = x.parse::<f64>() {
201                             self.ls.x[i] = value;
202                             self.needs_recalculation = true;
203                             self.update_plot_data();
204                         }
205                     }
206
207                     if y_response.changed() {
208                         if let Ok(value) = y.parse::<f64>() {
209                             self.ls.y[i] = value;
210                             self.needs_recalculation = true;
211                             self.update_plot_data();
212                         }
213                     }
214                     ui.end_row();
215                 }
216             });
217     });
218
219     ui.add_space(4.0);
220     ui.horizontal(|ui| {
221         if ui.button("Add").clicked() {
222             self.add_data_point();
223         }
224         if ui.button("Remove").clicked() {

```

```

226         self.remove_last_point();
227     }
228     if ui.button("Clear").clicked() {
229         self.clear_all_data();
230     }
231 });
232
233 ui.add_space(8.0);
234
235 // Actions
236 ui.group(|ui| {
237     ui.set_width(panel_width);
238     if ui.button("Load Sample Data").clicked() {
239         self.load_sample_data();
240     }
241     if ui.button("Calculate Model").clicked() {
242         self.calculate_model();
243     }
244
245     // Prediction
246     ui.add_space(4.0);
247     ui.horizontal(|ui| {
248         ui.label("Predict at time sample:");
249         ui.add_sized(
250             [60.0, 20.0],
251             egui::TextEdit::singleline(&mut self.x_predict),
252         );
253         if ui.button("Calculate").clicked() {
254             self.calculate_prediction();
255         }
256     });
257
258     // Results display
259     if let Some(result) = self.prediction_result {
260         ui.label(format!("Predicted: {:.2} m", result));
261     }
262     if let Some([x, y]) = self.intersection_point {
263         ui.label(format!("Critical at: {:.1} time samples ({:.2} m)", x, y));
264     }
265 });
266
267 // Error display
268 if !self.error_text.is_empty() {
269     ui.add_space(4.0);
270     ui.colored_label(Color32::from_rgb(255, 0, 0), &self.error_text);
271 }
272 });
273
274 ui.add_space(8.0);

```

```

275
276 // Right panel - Plot
277 ui.vertical(|ui| {
278     let available_width = ui.available_width();
279     let available_height = ui.available_height();
280     let plot_size =
281         Vec2::new(available_width.min(800.0), available_height.min(500.0));
282
283     Plot::new("water_level_plot")
284         .view_aspect(2.0)
285         .show_axes([true, true])
286         .show_grid(true)
287         .allow_zoom(false)
288         .allow_drag(false)
289         .min_size(plot_size)
290         .include_x(self.plot_bounds.x_min)
291         .include_x(self.plot_bounds.x_max)
292         .include_y(self.plot_bounds.y_min)
293         .include_y(self.plot_bounds.y_max)
294         .show(ui, |plot_ui| {
295             self.plot_data(plot_ui);
296         });
297
298 // Simple legend
299 ui.horizontal(|ui| {
300     ui.label("—");
301     ui.label("Model");
302     ui.add_space(8.0);
303     ui.colored_label(Color32::from_rgb(255, 0, 0), "- -");
304     ui.label("Critical");
305     ui.add_space(8.0);
306     ui.colored_label(Color32::from_rgb(66, 135, 245), "●");
307     ui.label("Data");
308 });
309 });
310 });
311 };
312 }
313
314 fn display_data_panel(&mut self, ui: &mut egui::Ui, width: f32) {
315     egui::Frame::none()
316         .fill(ui.visuals().extreme_bg_color)
317         .inner_margin(16.0)
318         .rounding(8.0)
319         .show(ui, |ui| {
320             ui.set_min_width(width);
321             ui.vertical(|ui| {
322                 ui.heading(RichText::new(format!("{}", Historical Data", ICON_DATA)).strong());
323                 ui.add_space(12.0);

```



```

324
325 // Data Points Table
326 egui::ScrollArea::vertical()
327     .max_height(400.0)
328     .show(ui, |ui| {
329         egui::Grid::new("data_points_grid")
330             .striped(true)
331             .spacing([12.0, 8.0])
332             .show(ui, |ui| {
333                 // Headers
334                 ui.strong("Time samples");
335                 ui.strong("Value");
336                 ui.strong("Level");
337                 ui.end_row();
338
339                 // Data points
340                 for i in 0..self.ls.x.len() {
341                     ui.label(format!("{}", i + 1));
342                     let mut x = format!("{:.4}", self.ls.x[i]);
343                     let mut y = format!("{:.4}", self.ls.y[i]);
344
345                     let x_response = ui.add_sized(
346                         [80.0, 24.0],
347                         egui::TextEdit::singleline(&mut x),
348                     );
349                     let y_response = ui.add_sized(
350                         [80.0, 24.0],
351                         egui::TextEdit::singleline(&mut y),
352                     );
353
354                     if x_response.changed() {
355                         if let Ok(value) = x.parse::<f64>() {
356                             self.ls.x[i] = value;
357                             self.needs_recalculation = true;
358                             self.update_plot_data();
359                         }
360                     }
361
362                     if y_response.changed() {
363                         if let Ok(value) = y.parse::<f64>() {
364                             self.ls.y[i] = value;
365                             self.needs_recalculation = true;
366                             self.update_plot_data();
367                         }
368                     }
369
370                     ui.end_row();
371                 }
372             });

```

```

373         });
374
375         ui.add_space(12.0);
376
377         // Data Management Buttons
378         ui.horizontal(|ui| {
379             if ui.button("Add Point").clicked() {
380                 self.add_data_point();
381             }
382             if ui.button("Remove Last").clicked() {
383                 self.remove_last_point();
384             }
385             if ui.button("Clear All").clicked() {
386                 self.clear_all_data();
387             }
388         });
389     });
390 }
391
392
393 fn display_plot_and_results_panel(&mut self, ui: &mut egui::Ui, width: f32) {
394     egui::Frame::none()
395         .fill(ui.visuals().extreme_bg_color)
396         .inner_margin(16.0)
397         .rounding(8.0)
398         .show(ui, |ui| {
399             ui.set_min_width(width);
400             ui.vertical(|ui| {
401                 // Plot Section
402                 ui.heading(RichText::new(format!("{ } Water Level Trend", ICON_POLY)).strong());
403                 ui.add_space(12.0);
404
405                 // Plot Controls in a horizontal layout with proper spacing
406                 ui.horizontal(|ui| {
407                     let button_size = Vec2::new(80.0, 24.0);
408                     if ui
409                         .add_sized(button_size, egui::Button::new("🔍 Zoom In"))
410                         .clicked()
411                     {
412                         self.zoom_in();
413                     }
414                     if ui
415                         .add_sized(button_size, egui::Button::new("🔍 Zoom Out"))
416                         .clicked()
417                     {
418                         self.zoom_out();
419                     }
420                     if ui
421                         .add_sized(button_size, egui::Button::new("Reset View"))

```

```

422         .clicked()
423     {
424         self.reset_zoom();
425     }
426 });
427
428 ui.add_space(8.0);
429
430 // Calculate available height for the plot
431 let available_height = ui.available_height();
432 let plot_height = if width < 400.0 {
433     (available_height * 0.4).min(300.0) // 40% of available height or 300px max
434 } else if width < 800.0 {
435     (available_height * 0.5).min(400.0) // 50% of available height or 400px max
436 } else {
437     (available_height * 0.6).min(500.0) // 60% of available height or 500px max
438 };
439
440 // Create a container for the plot with fixed size
441 let plot_width = width - 32.0; // Account for margins
442 egui::Frame::none()
443     .fill(ui.visuals().extreme_bg_color)
444     .show(ui, |ui| {
445         // Main Plot with responsive sizing and proper constraints
446         let plot = Plot::new("water_level_plot")
447             .height(plot_height)
448             .width(plot_width)
449             .allow_zoom(false)
450             .allow_drag(false)
451             .show_axes([true, true])
452             .show_grid(true)
453             .min_size(Vec2::new(plot_width.min(300.0), plot_height.min(200.0)))
454             .data_aspect(1.0)
455             .auto_bounds_x()
456             .auto_bounds_y()
457             .include_x(self.plot_bounds.x_min)
458             .include_x(self.plot_bounds.x_max)
459             .include_y(self.plot_bounds.y_min)
460             .include_y(self.plot_bounds.y_max)
461             .x_axis_label("Time samples")
462             .y_axis_label("Water Level (m)");
463
464         plot.show(ui, |plot_ui| {
465             self.plot_data(plot_ui);
466         });
467     });
468
469 // Add scrolling for the rest of the content if needed
470 egui::ScrollArea::vertical().show(ui, |ui| {

```

```

471 // Detached Legend with responsive layout
472 ui.add_space(16.0);
473 if width < 600.0 {
474     // Vertical legend for narrow screens
475     self.display_legend_vertical(ui);
476 } else {
477     // Horizontal legend for wider screens
478     self.display_legend_horizontal(ui);
479 }
480
481 ui.add_space(16.0);
482
483 // Results Section
484 ui.heading(RichText::new(format!("{}", Results), ICON_RESULT)).strong();
485 ui.add_space(12.0);
486
487 // Prediction Controls with responsive layout
488 if width < 400.0 {
489     // Vertical layout for narrow screens
490     ui.vertical(|ui| {
491         ui.label("Predict at time sample:");
492         ui.add_sized(
493             [width - 32.0, 24.0],
494             egui::TextEdit::singleline(&mut self.x_predict),
495         );
496         if ui.button(format!("{}", Calculate), ICON_PREDICT).clicked() {
497             self.calculate_prediction();
498         }
499     });
500 } else {
501     // Horizontal layout for wider screens
502     ui.horizontal(|ui| {
503         ui.label("Predict at time sample:");
504         ui.add_sized(
505             [80.0, 24.0],
506             egui::TextEdit::singleline(&mut self.x_predict),
507         );
508         if ui.button(format!("{}", Calculate), ICON_PREDICT).clicked() {
509             self.calculate_prediction();
510         }
511     });
512 }
513
514 // Prediction Results
515 if let Some(result) = self.prediction_result {
516     ui.add_space(8.0);
517     ui.horizontal(|ui| {
518         ui.label(RichText::new("Predicted level:").strong());
519         ui.label(format!("{}", result));

```

```

520         });
521     }
522
523     // Critical Level Crossing
524     if let Some([x, y]) = self.intersection_point {
525         ui.add_space(8.0);
526         if width < 400.0 {
527             ui.vertical(|ui| {
528                 ui.label(RichText::new("Critical level crossing:").strong());
529                 ui.label(format!("at {:.2} time samples, {:.2} m", x, y));
530             });
531         } else {
532             ui.horizontal(|ui| {
533                 ui.label(RichText::new("Critical level crossing:").strong());
534                 ui.label(format!("at {:.2} time samples, {:.2} m", x, y));
535             });
536         }
537     }
538 });
539 });
540 });
541 }
542
543 fn display_legend_horizontal(&self, ui: &mut egui::Ui) {
544     egui::Frame::none()
545         .fill(ui.visuals().faint_bg_color)
546         .rounding(4.0)
547         .inner_margin(8.0)
548         .show(ui, |ui| {
549             ui.heading("Legend");
550             ui.add_space(8.0);
551
552             ui.horizontal_wrapped(|ui| {
553                 let item_spacing = 20.0;
554
555                 // Historical Data
556                 ui.horizontal(|ui| {
557                     ui.colored_label(Color32::from_rgb(66, 135, 245), "●");
558                     ui.label("Historical Data");
559                 });
560                 ui.add_space(item_spacing);
561
562                 // Model Lines
563                 if self.show_all_orders {
564                     for (order, _) in &self.all_order_points {
565                         ui.horizontal(|ui| {
566                             ui.colored_label(Self::get_order_color(*order), "—");
567                             ui.label(format!("Model Order {}", order));
568                         });

```

```

569         ui.add_space(item_spacing);
570     }
571     } else if !self.ls.a.is_empty() {
572         ui.horizontal(|ui| {
573             ui.colored_label(Self::get_order_color(self.ls.m), "--");
574             ui.label(format!("Model Order {}", self.ls.m));
575         });
576         ui.add_space(item_spacing);
577     }
578
579     // Critical Level
580     if self.critical_level.is_some() {
581         ui.horizontal(|ui| {
582             ui.colored_label(Color32::from_rgb(255, 0, 0), "-");
583             ui.label("Critical Level");
584         });
585         ui.add_space(item_spacing);
586     }
587
588     // Predicted Points
589     if !self.predicted_points.is_empty() {
590         ui.horizontal(|ui| {
591             ui.colored_label(Color32::from_rgb(255, 150, 50), "●");
592             ui.label("Predicted Points");
593         });
594     }
595     });
596 });
597 }
598
599 fn display_legend_vertical(&self, ui: &mut egui::Ui) {
600     egui::Frame::none()
601         .fill(ui.visuals().faint_bg_color)
602         .rounding(4.0)
603         .inner_margin(8.0)
604         .show(ui, |ui| {
605             ui.heading("Legend");
606             ui.add_space(8.0);
607
608             egui::Grid::new("legend_grid")
609                 .spacing([12.0, 8.0])
610                 .show(ui, |ui| {
611                     // Historical Data
612                     ui.colored_label(Color32::from_rgb(66, 135, 245), "●");
613                     ui.label("Historical Data");
614                     ui.end_row();
615
616                     // Model Lines
617                     if self.show_all_orders {

```

```

618         for (order, _) in &self.all_order_points {
619             ui.colored_label(Self::get_order_color(*order), "--");
620             ui.label(format!("Model Order {}", order));
621             ui.end_row();
622         }
623     } else if !self.ls.a.is_empty() {
624         ui.colored_label(Self::get_order_color(self.ls.m), "--");
625         ui.label(format!("Model Order {}", self.ls.m));
626         ui.end_row();
627     }
628
629     // Critical Level
630     if self.critical_level.is_some() {
631         ui.colored_label(Color32::from_rgb(255, 0, 0), "-");
632         ui.label("Critical Level");
633         ui.end_row();
634     }
635
636     // Predicted Points
637     if !self.predicted_points.is_empty() {
638         ui.colored_label(Color32::from_rgb(255, 150, 50), "●");
639         ui.label("Predicted Points");
640         ui.end_row();
641     }
642     });
643 });
644 }
645
646 fn update_plot_bounds(&mut self) {
647     if !self.plot_points.is_empty() {
648         // Calculate bounds from all points
649         let all_points = self
650             .plot_points
651             .iter()
652             .chain(self.predicted_points.iter())
653             .chain(self.function_points.iter())
654             .chain(self.intersection_point.iter());
655
656         let x_values: Vec<f64> = all_points.clone().map(|p| p[0]).collect();
657         let y_values: Vec<f64> = all_points.map(|p| p[1]).collect();
658
659         let x_min = x_values.iter().copied().fold(f64::INFINITY, f64::min);
660         let x_max = x_values.iter().copied().fold(f64::NEG_INFINITY, f64::max);
661         let y_min = y_values.iter().copied().fold(f64::INFINITY, f64::min);
662         let y_max = y_values.iter().copied().fold(f64::NEG_INFINITY, f64::max);
663
664         // Add margins (10% for x, 5% for y since water levels are more precise)
665         let x_margin = (x_max - x_min).abs() * 0.1;
666         let y_margin = (y_max - y_min).abs() * 0.05;

```

```

667
668     // Update base bounds
669     self.plot_bounds.base_x_min = x_min - x_margin;
670     self.plot_bounds.base_x_max = x_max + x_margin;
671     self.plot_bounds.base_y_min = y_min - y_margin;
672     self.plot_bounds.base_y_max = y_max + y_margin;
673
674     // Reset current bounds to base bounds if they haven't been set
675     if self.plot_bounds.x_max == 15.0 && self.plot_bounds.y_max == 450.0 {
676         self.reset_zoom();
677     }
678 }
679 }
680
681 fn zoom_in(&mut self) {
682     let zoom_factor = 0.8; // 20% zoom in
683     self.apply_zoom(zoom_factor);
684 }
685
686 fn zoom_out(&mut self) {
687     let zoom_factor = 1.25; // 25% zoom out
688     self.apply_zoom(zoom_factor);
689 }
690
691 fn apply_zoom(&mut self, factor: f64) {
692     let x_center = (self.plot_bounds.x_min + self.plot_bounds.x_max) / 2.0;
693     let y_center = (self.plot_bounds.y_min + self.plot_bounds.y_max) / 2.0;
694
695     let x_range = (self.plot_bounds.x_max - self.plot_bounds.x_min) * factor;
696     let y_range = (self.plot_bounds.y_max - self.plot_bounds.y_min) * factor;
697
698     // Apply zoom while maintaining aspect ratio
699     self.plot_bounds.x_min = x_center - x_range / 2.0;
700     self.plot_bounds.x_max = x_center + x_range / 2.0;
701     self.plot_bounds.y_min = y_center - y_range / 2.0;
702     self.plot_bounds.y_max = y_center + y_range / 2.0;
703
704     // Ensure we don't zoom out beyond base bounds
705     if factor > 1.0 {
706         self.plot_bounds.x_min = self.plot_bounds.x_min.max(self.plot_bounds.base_x_min);
707         self.plot_bounds.x_max = self.plot_bounds.x_max.min(self.plot_bounds.base_x_max);
708         self.plot_bounds.y_min = self.plot_bounds.y_min.max(self.plot_bounds.base_y_min);
709         self.plot_bounds.y_max = self.plot_bounds.y_max.min(self.plot_bounds.base_y_max);
710     }
711 }
712
713 fn reset_zoom(&mut self) {
714     // Reset to base bounds
715     self.plot_bounds.x_min = self.plot_bounds.base_x_min;

```



```

716     self.plot_bounds.x_max = self.plot_bounds.base_x_max;
717     self.plot_bounds.y_min = self.plot_bounds.base_y_min;
718     self.plot_bounds.y_max = self.plot_bounds.base_y_max;
719 }
720
721 fn update_plot_data(&mut self) {
722     // Store current data
723     let current_points = self.plot_points.clone();
724     let current_function = self.function_points.clone();
725     let current_predicted = self.predicted_points.clone();
726
727     // Clear existing data
728     self.plot_points.clear();
729     self.function_points.clear();
730
731     // Add original points
732     self.plot_points
733         .extend(self.ls.x.iter().zip(&self.ls.y).map(|(&x, &y)| [x, y]));
734
735     // Generate function curve points if coefficients exist
736     if !self.ls.a.is_empty() {
737         let x_min = self.ls.x.iter().copied().fold(f64::INFINITY, f64::min);
738         let x_max = self.ls.x.iter().copied().fold(f64::NEG_INFINITY, f64::max);
739         let range = x_max - x_min;
740         let step = range / 100.0;
741
742         // Pre-calculate coefficients to avoid borrow conflict
743         let coeffs = self.ls.a.clone();
744
745         // Generate points in a separate vector
746         let new_points: Vec<[f64; 2]> = (0..=100)
747             .map(|i| {
748                 let x = x_min + step * i as f64;
749                 let mut result = 0.0;
750                 let mut x_power = 1.0;
751                 for &coeff in &coeffs {
752                     result += coeff * x_power;
753                     x_power *= x;
754                 }
755                 [x, result]
756             })
757             .collect();
758
759         // Extend function points with the new points
760         self.function_points.extend(new_points);
761     }
762
763     // Restore predicted points
764     self.predicted_points = current_predicted;

```

```

765
766 // Update plot bounds only if data changed
767 if self.plot_points != current_points || self.function_points != current_function {
768     self.update_plot_bounds();
769     self.needs_recalculation = true;
770 }
771 }
772
773 fn evaluate_polynomial(&self, x: f64) -> f64 {
774     let mut result = 0.0;
775     let mut x_power = 1.0;
776     for &coeff in &self.ls.a {
777         result += coeff * x_power;
778         x_power *= x;
779     }
780     result
781 }
782
783 fn find_intersection(&mut self) {
784     if let Some(critical_y) = self.critical_level {
785         if !self.ls.a.is_empty() {
786             // Use wider range for intersection search
787             let x_min = 0.0;
788             let x_max = self.ls.n as f64 * 2.0; // Search up to 2x the data range
789
790             let mut left = x_min;
791             let mut right = x_max;
792             let epsilon = 1e-6;
793
794             // Try to find intersection using binary search
795             for _ in 0..50 {
796                 // Limit iterations to prevent infinite loops
797                 let mid = (left + right) / 2.0;
798                 let y = self.evaluate_polynomial(mid);
799
800                 if (y - critical_y).abs() < epsilon {
801                     self.intersection_point = Some([mid, y]);
802                     break;
803                 } else if y < critical_y {
804                     left = mid;
805                 } else {
806                     right = mid;
807                 }
808             }
809         }
810     }
811 }
812
813 fn predict_value(&mut self, x: f64) -> Result<f64, String> {

```

```

814     if self.ls.a.is_empty() {
815         return Err("Please calculate coefficients first".to_string());
816     }
817
818     // Use evaluate_polynomial instead of ls.predict to avoid thread safety issues
819     let y = self.evaluate_polynomial(x);
820     Ok(y)
821 }
822
823 fn update_simulation(&mut self) {
824     if !self.is_simulating || self.simulation_paused {
825         return;
826     }
827
828     let now = std::time::Instant::now();
829     if let Some(last_time) = self.last_simulation_time {
830         let elapsed = now.duration_since(last_time).as_secs_f64();
831         if elapsed < self.simulation_speed {
832             return;
833         }
834     }
835
836     // Update simulation state
837     if self.simulation_step < self.simulation_data.len() {
838         // Update progress
839         self.simulation_progress =
840             self.simulation_step as f32 / self.simulation_data.len() as f32;
841
842         // Update data points up to current step
843         self.ls.x.clear();
844         self.ls.y.clear();
845         for i in 0..=self.simulation_step {
846             let [x, y] = self.simulation_data[i];
847             self.ls.x.push(x);
848             self.ls.y.push(y);
849         }
850
851         // Update plot data and recalculate model if we have enough points
852         self.update_plot_data();
853         if self.ls.x.len() > 2 {
854             let m = if self.simulation_auto_order {
855                 // Automatically determine best order based on available points
856                 let max_order = (self.ls.x.len() - 1).min(5);
857                 if let Some(optimal_order) = self.find_optimal_order() {
858                     optimal_order
859                 } else {
860                     max_order.min(3) // Default to 3 or less if can't determine optimal
861                 }
862             } else if let Ok(m) = self.m_input.parse::(<usize>) {

```

```

863         m.min(self.ls.x.len() - 1)
864     } else {
865         2 // Default order
866     };
867
868     if m < self.ls.x.len() {
869         self.ls.n = self.ls.x.len();
870         self.ls.m = m;
871         self.m_input = m.to_string(); // Update input field
872
873         // Clear and resize vectors
874         self.ls.steps.clear();
875         self.ls.s.clear();
876         self.ls.q.clear();
877         self.ls.p.clear();
878         self.ls.c.clear();
879         self.ls.b.clear();
880         self.ls.a.clear();
881
882         self.ls.s.resize(m + 1, 0.0);
883         self.ls.q.resize(m + 1, 0.0);
884         self.ls.p.resize(m + 1, vec![0.0; self.ls.n]);
885         self.ls.c.resize(m + 1, vec![0.0; m + 1]);
886         self.ls.b.resize(m + 1, 0.0);
887         self.ls.a.resize(m + 1, 0.0);
888
889         // Calculate model
890         self.ls.calculate_orthogonal_polynomials();
891         self.ls.calculate_coefficients();
892
893         if self.show_all_orders {
894             self.calculate_all_orders();
895         }
896
897         self.error_text.clear();
898         self.update_plot_data();
899         self.find_intersection();
900     }
901 }
902
903 self.simulation_step += 1;
904 self.last_simulation_time = Some(now);
905 } else {
906     // Simulation complete
907     self.is_simulating = false;
908     self.simulation_step = 0;
909     self.simulation_progress = 1.0;
910 }
911 }

```

```

912 fn find_optimal_order(&self) -> Option<usize> {
913     if self.ls.x.len() < 4 {
914         // Need at least 4 points for meaningful analysis
915         return None;
916     }
917
918
919     let n = self.ls.x.len();
920     let max_order = (n - 1).min(5); // Limit max order to prevent overfitting
921     let mut best_order = 1;
922     let mut best_score = f64::INFINITY;
923
924     // Try different orders and evaluate them
925     for order in 1..=max_order {
926         let mut temp_ls = LS::new(n, order);
927         temp_ls.x = self.ls.x.clone();
928         temp_ls.y = self.ls.y.clone();
929
930         // Calculate coefficients for this order
931         temp_ls.calculate_orthogonal_polynomials();
932         temp_ls.calculate_coefficients();
933
934         // Calculate fit error
935         let mut fit_error = 0.0;
936         for i in 0..n {
937             let mut predicted = 0.0;
938             let mut x_power = 1.0;
939             for &coeff in &temp_ls.a {
940                 predicted += coeff * x_power;
941                 x_power *= temp_ls.x[i];
942             }
943             let error = (predicted - temp_ls.y[i]).abs();
944             fit_error += error * error;
945         }
946         fit_error /= n as f64;
947
948         // Calculate complexity penalty (AIC-inspired)
949         let complexity_penalty = order as f64 * (n as f64).ln();
950
951         // Calculate smoothness penalty
952         let mut smoothness_penalty = 0.0;
953         if order > 1 {
954             for i in 1..n - 1 {
955                 let mut d1 = 0.0; // First derivative
956                 let mut d2 = 0.0; // Second derivative
957                 let mut x_power = 1.0;
958                 let mut power = 0;
959
960                 for &coeff in &temp_ls.a {

```

```

961         if power > 0 {
962             d1 += coeff * (power as f64) * x_power;
963         }
964         if power > 1 {
965             d2 += coeff * (power as f64) * ((power - 1) as f64) * x_power
966                 / temp_ls.x[i];
967         }
968         x_power *= temp_ls.x[i];
969         power += 1;
970     }
971
972     smoothness_penalty += d2.abs();
973 }
974 smoothness_penalty /= (n - 2) as f64;
975 }
976
977 // Combine scores with weights
978 let score = fit_error + 0.1 * complexity_penalty + 0.2 * smoothness_penalty;
979
980 // Update best order if this score is better
981 if score < best_score {
982     best_score = score;
983     best_order = order;
984 }
985 }
986
987 Some(best_order)
988 }
989
990 fn update(&mut self, ctx: &egui::Context, frame: &mut eframe::Frame) {
991     if self.is_simulating {
992         self.update_simulation();
993         ctx.request_repaint();
994     }
995
996     egui::CentralPanel::default().show(ctx, |ui| {
997         self.display_main_panel(ui);
998     });
999 }
1000
1001 fn load_sample_data(&mut self) {
1002     self.ls = LS::new(10, 3);
1003     self.ls.initialize_points();
1004     self.n_input = "10".to_string();
1005     self.m_input = "3".to_string();
1006     self.x_predict = "10.0".to_string();
1007     self.critical_level_input = "448.5".to_string();
1008     self.critical_level = Some(448.5);
1009     self.needs_recalculation = true;

```

```

1010     self.error_text.clear();
1011     self.update_plot_data();
1012 }
1013
1014 fn calculate_model(&mut self) {
1015     if !self.ls.x.is_empty() && self.ls.x.len() == self.ls.y.len() {
1016         if let Ok(m) = self.m_input.parse::<usize>() {
1017             if m < self.ls.x.len() {
1018                 self.ls.n = self.ls.x.len();
1019                 self.ls.m = m;
1020
1021                 // Clear previous results
1022                 self.ls.s.clear();
1023                 self.ls.q.clear();
1024                 self.ls.p.clear();
1025                 self.ls.c.clear();
1026                 self.ls.b.clear();
1027                 self.ls.a.clear();
1028
1029                 // Resize vectors with proper sizes
1030                 self.ls.s.resize(m + 1, 0.0);
1031                 self.ls.q.resize(m + 1, 0.0);
1032                 self.ls.p.resize(m + 1, vec![0.0; self.ls.n]);
1033                 self.ls.c.resize(m + 1, vec![0.0; m + 1]);
1034                 self.ls.b.resize(m + 1, 0.0);
1035                 self.ls.a.resize(m + 1, 0.0);
1036
1037                 // Calculate coefficients
1038                 self.ls.calculate_orthogonal_polynomials();
1039                 self.ls.calculate_coefficients();
1040
1041                 if self.show_all_orders {
1042                     self.calculate_all_orders();
1043                 }
1044
1045                 self.error_text.clear();
1046                 self.update_plot_data();
1047                 self.find_intersection();
1048             } else {
1049                 self.error_text =
1050                     "Model order must be less than number of data points".to_string();
1051             }
1052         } else {
1053             self.error_text = "Invalid model order".to_string();
1054         }
1055     } else {
1056         self.error_text = "Please load or enter data first".to_string();
1057     }
1058 }

```

```

1059
1060 fn add_data_point(&mut self) {
1061     let next_x = if let Some(last) = self.ls.x.last() {
1062         last + 1.0
1063     } else {
1064         0.0
1065     };
1066     self.ls.x.push(next_x);
1067     self.ls.y.push(0.0);
1068     self.update_plot_data();
1069 }
1070
1071 fn remove_last_point(&mut self) {
1072     if !self.ls.x.is_empty() {
1073         self.ls.x.pop();
1074         self.ls.y.pop();
1075         self.update_plot_data();
1076     }
1077 }
1078
1079 fn clear_all_data(&mut self) {
1080     self.ls = LS::new(self.ls.n, self.ls.m);
1081     self.ls.initialize_points();
1082     self.update_plot_data();
1083 }
1084
1085 fn plot_data(&self, plot_ui: &mut egui_plot::PlotUi) {
1086     // Historical data points
1087     plot_ui.points(
1088         Points::new(PlotPoints::new(self.plot_points.clone()))
1089             .name("Historical Data")
1090             .radius(5.0)
1091             .color(Color32::from_rgb(66, 135, 245)),
1092     );
1093
1094     // Show all polynomial orders if enabled
1095     if self.show_all_orders && !self.all_order_points.is_empty() {
1096         for (order, points) in &self.all_order_points {
1097             plot_ui.line(
1098                 Line::new(PlotPoints::new(points.clone()))
1099                     .name(format!("Model Order {}", order))
1100                     .color(Self::get_order_color(*order))
1101                     .width(2.0),
1102             );
1103         }
1104     } else if !self.ls.a.is_empty() {
1105         // Show only the selected order
1106         plot_ui.line(
1107             Line::new(PlotPoints::new(self.function_points.clone()))

```



```

1108         .name(format!("Prediction Model (Order {})", self.ls.m))
1109         .color(Self::get_order_color(self.ls.m))
1110         .width(2.0),
1111     );
1112 }
1113
1114 // Critical water level
1115 if let Some(y) = self.critical_level {
1116     plot_ui.line(
1117         Line::new(PlotPoints::new(vec![
1118             [self.plot_bounds.x_min, y],
1119             [self.plot_bounds.x_max, y],
1120         ]))
1121         .name("Critical Water Level")
1122         .color(Color32::from_rgb(255, 0, 0))
1123         .width(1.0)
1124         .style(egui_plot::LineStyle::Dashed { length: 5.0 }),
1125     );
1126 }
1127
1128 // Critical level intersection
1129 if let Some(point) = self.intersection_point {
1130     plot_ui.points(
1131         Points::new(PlotPoints::new(vec![point]))
1132         .name("Critical Level Crossing")
1133         .radius(6.0)
1134         .color(Color32::from_rgb(255, 0, 0)),
1135     );
1136 }
1137
1138 // Predicted points
1139 if !self.predicted_points.is_empty() {
1140     plot_ui.points(
1141         Points::new(PlotPoints::new(self.predicted_points.clone()))
1142         .name("Predicted Levels")
1143         .radius(6.0)
1144         .color(Color32::from_rgb(255, 150, 50)),
1145     );
1146 }
1147 }
1148
1149 fn calculate_prediction(&mut self) {
1150     match self.x_predict.parse::<f64>() {
1151         Ok(x) => {
1152             if x >= 0.0 {
1153                 match self.predict_value(x) {
1154                     Ok(y) => {
1155                         self.prediction_result = Some(y);
1156                         let mut new_points = self.predicted_points.clone();

```

```

1157         new_points.push([x, y]);
1158         self.predicted_points = new_points;
1159         self.error_text.clear();
1160         self.update_plot_bounds();
1161     }
1162     Err(err) => {
1163         self.error_text = err;
1164     }
1165 }
1166 } else {
1167     self.error_text = "Time must be non-negative".to_string();
1168 }
1169 }
1170 Err(_) => {
1171     self.error_text = "Invalid time value".to_string();
1172 }
1173 }
1174 }
1175
1176 fn get_order_color(order: usize) -> Color32 {
1177     // Generate distinct colors for different orders
1178     match order {
1179         0 => Color32::from_rgb(200, 0, 0),    // Red
1180         1 => Color32::from_rgb(0, 200, 0),    // Green
1181         2 => Color32::from_rgb(0, 0, 200),    // Blue
1182         3 => Color32::from_rgb(200, 200, 0),  // Yellow
1183         4 => Color32::from_rgb(200, 0, 200),  // Magenta
1184         5 => Color32::from_rgb(0, 200, 200),  // Cyan
1185         _ => {
1186             // For higher orders, generate colors using HSV
1187             let hue = (order as f32 * 30.0) % 360.0;
1188             let (r, g, b) = Self::hsv_to_rgb(hue, 0.8, 0.9);
1189             Color32::from_rgb(r, g, b)
1190         }
1191     }
1192 }
1193
1194 fn calculate_all_orders(&mut self) {
1195     if !self.needs_recalculation && !self.all_order_points.is_empty() {
1196         return;
1197     }
1198
1199     self.all_order_points.clear();
1200     self.cached_coefficients.clear();
1201
1202     // Store original m value and state
1203     let original_m = self.ls.m;
1204     let original_a = self.ls.a.clone();
1205     let n = self.ls.x.len();

```

```

1206
1207 // Calculate for each order from 0 to n-2 (order must be less than n)
1208 for m in 0..n {
1209     // Clear previous results
1210     self.ls.s.clear();
1211     self.ls.q.clear();
1212     self.ls.p.clear();
1213     self.ls.c.clear();
1214     self.ls.b.clear();
1215     self.ls.a.clear();
1216
1217     // Set up for this order
1218     self.ls.m = m;
1219     self.ls.s.resize(m + 1, 0.0);
1220     self.ls.q.resize(m + 1, 0.0);
1221     self.ls.p.resize(m + 1, vec![0.0; n]);
1222     self.ls.c.resize(m + 1, vec![0.0; m + 1]);
1223     self.ls.b.resize(m + 1, 0.0);
1224     self.ls.a.resize(m + 1, 0.0);
1225
1226     self.ls.calculate_orthogonal_polynomials();
1227     self.ls.calculate_coefficients();
1228
1229     // Cache coefficients
1230     self.cached_coefficients.push((m, self.ls.a.clone()));
1231
1232     // Generate curve points
1233     let mut curve_points = Vec::new();
1234     let x_min = self.ls.x.iter().copied().fold(f64::INFINITY, f64::min);
1235     let x_max = self.ls.x.iter().copied().fold(f64::NEG_INFINITY, f64::max);
1236     let range = x_max - x_min;
1237     let step = range / 100.0;
1238
1239     for i in 0..=100 {
1240         let x = x_min + step * i as f64;
1241         let y = self.evaluate_polynomial(x);
1242         curve_points.push([x, y]);
1243     }
1244
1245     self.all_order_points.push((m, curve_points));
1246 }
1247
1248 // Restore original state
1249 self.ls.m = original_m;
1250 self.ls.a = original_a;
1251 self.ls.s.resize(original_m + 1, 0.0);
1252 self.ls.q.resize(original_m + 1, 0.0);
1253 self.ls.p.resize(original_m + 1, vec![0.0; n]);
1254 self.ls.c.resize(original_m + 1, vec![0.0; original_m + 1]);

```

```

1255     self.ls.b.resize(original_m + 1, 0.0);
1256
1257     self.needs_recalculation = false;
1258 }
1259
1260 fn hsv_to_rgb(h: f32, s: f32, v: f32) -> (u8, u8, u8) {
1261     let h = h % 360.0;
1262     let c = v * s;
1263     let x = c * (1.0 - ((h / 60.0) % 2.0 - 1.0).abs());
1264     let m = v - c;
1265
1266     let (r, g, b) = match (h / 60.0) as i32 {
1267         0 => (c, x, 0.0),
1268         1 => (x, c, 0.0),
1269         2 => (0.0, c, x),
1270         3 => (0.0, x, c),
1271         4 => (x, 0.0, c),
1272         _ => (c, 0.0, x),
1273     };
1274
1275     (
1276         ((r + m) * 255.0) as u8,
1277         ((g + m) * 255.0) as u8,
1278         ((b + m) * 255.0) as u8,
1279     )
1280 }
1281 }
1282
1283 impl eframe::App for PredictionApp {
1284     fn update(&mut self, ctx: &egui::Context, frame: &mut eframe::Frame) {
1285         self.update(ctx, frame);
1286     }
1287 }
1288
1289 fn main() {
1290     let options = eframe::NativeOptions {
1291         default_theme: eframe::Theme::Dark,
1292         follow_system_theme: true,
1293         viewport: egui::ViewportBuilder::default()
1294             .with_inner_size([1200.0, 800.0])
1295             .with_min_inner_size([600.0, 400.0]),
1296         ..Default::default()
1297     };
1298
1299     eframe::run_native(
1300         "Water Level Prediction System",
1301         options,
1302         Box::new(|_cc| Ok(Box::new(PredictionApp::default()))),
1303     )

```

```
1304 .unwrap();  
1305 }
```

Файл: ./src/lib/ls.rs

```

1 #[derive(Debug, Clone)]
2 pub struct EncodingStep {
3     pub description: String,
4     pub data: Vec<f64>,
5     pub step_type: StepType,
6 }
7
8 #[derive(Debug, Clone)]
9 pub enum StepType {
10     DataPreparation,
11     OrthogonalPolynomials,
12     CoefficientCalculation,
13     FinalResult,
14     Prediction,
15 }
16
17 pub struct LS {
18     pub n: usize,           // number of points
19     pub m: usize,           // polynomial order
20     pub x: Vec<f64>,        // x values
21     pub y: Vec<f64>,        // y values
22     pub s: Vec<f64>,        // s coefficients
23     pub q: Vec<f64>,        // q coefficients
24     pub p: Vec<Vec<f64>>,   // orthogonal polynomials
25     pub c: Vec<Vec<f64>>,   // c coefficients
26     pub b: Vec<f64>,        // b coefficients
27     pub a: Vec<f64>,        // final coefficients
28     pub steps: Vec<EncodingStep>, // record of steps
29 }
30
31 impl LS {
32     pub fn new(n: usize, m: usize) -> Self {
33         LS {
34             n,
35             m,
36             x: vec![0.0; n],
37             y: vec![0.0; n],
38             s: vec![0.0; m + 1],
39             q: vec![0.0; m + 1],
40             p: vec![vec![0.0; n]; m + 1],
41             c: vec![vec![0.0; m + 1]; m + 1],
42             b: vec![0.0; m + 1],
43             a: vec![0.0; m + 1],
44             steps: Vec::new(),
45         }
46     }
47 }

```

```

48 fn add_step(&mut self, description: String, data: Vec<f64>, step_type: StepType) {
49     self.steps.push(EncodingStep {
50         description,
51         data: data.clone(),
52         step_type,
53     });
54 }
55
56 pub fn clear_steps(&mut self) {
57     self.steps.clear();
58 }
59
60 pub fn initialize_points(&mut self) {
61     self.clear_steps();
62
63     // Ensure n is at least 1 to avoid empty vectors
64     if self.n == 0 {
65         self.n = 1;
66     }
67
68     // Resize vectors if needed
69     self.x.resize(self.n, 0.0);
70     self.y.resize(self.n, 0.0);
71
72     // Update x values safely
73     for i in 0..self.n {
74         self.x[i] = i as f64;
75     }
76
77     // Update y values safely
78     for i in 0..self.n {
79         self.y[i] = ((1.0 * self.x[i] + 2.0) * self.x[i] + 3.0) * self.x[i] + 4.0;
80     }
81
82     self.add_step(
83         "Initial data points".to_string(),
84         self.y.clone(),
85         StepType::DataPreparation,
86     );
87 }
88
89 pub fn calculate_orthogonal_polynomials(&mut self) {
90     // Ensure vectors are properly sized
91     self.p.resize(self.m + 1, vec![0.0; self.n]);
92     self.s.resize(self.m + 1, 0.0);
93     self.q.resize(self.m + 1, 0.0);
94
95     for i in 0..=self.m {
96         if i == 0 {

```

```

97         for j in 0..self.n {
98             self.p[0][j] = 1.0;
99         }
100     } else if i == 1 {
101         // Check for division by zero
102         if self.s[i - 1].abs() > 1e-10 {
103             for j in 0..self.n {
104                 self.p[i][j] = self.x[j] - self.q[i - 1] / self.s[i - 1];
105             }
106         }
107     } else {
108         // Check for division by zero
109         if self.s[i - 1].abs() > 1e-10 && self.s[i - 2].abs() > 1e-10 {
110             for j in 0..self.n {
111                 self.p[i][j] = (self.x[j] - self.q[i - 1] / self.s[i - 1])
112                     * self.p[i - 1][j]
113                     - self.s[i - 1] / self.s[i - 2] * self.p[i - 2][j];
114             }
115         }
116     }
117
118     self.s[i] = self.p[i].iter().map(|x| x * x).sum();
119     self.q[i] = (0..self.n)
120         .map(|j| self.p[i][j] * self.p[i][j] * self.x[j])
121         .sum();
122
123     self.add_step(
124         format!("Calculated orthogonal polynomial p{} ", i),
125         self.p[i].clone(),
126         StepType::OrthogonalPolynomials,
127     );
128 }
129 }
130
131 pub fn calculate_coefficients(&mut self) {
132     // Ensure we have data to work with
133     if self.x.is_empty() || self.y.is_empty() {
134         return;
135     }
136
137     // Ensure all vectors are properly sized
138     self.s.resize(self.m + 1, 0.0);
139     self.q.resize(self.m + 1, 0.0);
140     self.p.resize(self.m + 1, vec![0.0; self.n]);
141     self.c.resize(self.m + 1, vec![0.0; self.m + 1]);
142     self.b.resize(self.m + 1, 0.0);
143     self.a.resize(self.m + 1, 0.0);
144
145     // First calculate orthogonal polynomials if not already calculated

```



```

146 self.calculate_orthogonal_polynomials();
147
148 // Calculate c coefficients
149 for k in 0..self.m {
150     self.c[k][k] = 1.0;
151 }
152
153 for k in 1..self.m {
154     for j in 0..k {
155         self.c[k][j] = (if j > 0 { self.c[k - 1][j - 1] } else { 0.0 })
156             - self.q[k - 1] / self.s[k - 1] * self.c[k - 1][j]
157             - (if k > 1 && j < k - 1 {
158                 self.s[k - 1] / self.s[k - 2] * self.c[k - 2][j]
159             } else {
160                 0.0
161             });
162     }
163     self.add_step(
164         format!("Calculated c coefficients for k={}", k),
165         self.c[k].clone(),
166         StepType::CoefficientCalculation,
167     );
168 }
169
170 // Calculate b coefficients
171 for k in 0..self.m {
172     // Check if s[k] is not zero to avoid division by zero
173     if self.s[k].abs() > 1e-10 {
174         self.b[k] = (0..self.n).map(|i| self.y[i] * self.p[k][i]).sum::() / self.s[k];
175     } else {
176         self.b[k] = 0.0;
177     }
178     self.add_step(
179         format!("Calculated b coefficient b{}", k),
180         vec![self.b[k]],
181         StepType::CoefficientCalculation,
182     );
183 }
184
185 // Calculate final coefficients a
186 for j in 0..self.m {
187     self.a[j] = (j..self.m).map(|k| self.b[k] * self.c[k][j]).sum();
188     self.add_step(
189         format!("Calculated final coefficient a{}", j),
190         vec![self.a[j]],
191         StepType::FinalResult,
192     );
193 }
194 }

```

```
195
196 pub fn predict(&mut self, x: f64) -> f64 {
197     let mut result = 0.0;
198     let mut x_power = 1.0;
199
200     // Calculate polynomial value using Horner's method
201     for i in 0..self.m {
202         result += self.a[i] * x_power;
203         x_power *= x;
204     }
205
206     self.add_step(
207         format!("Predicted value at x={}", x),
208         vec![result],
209         StepType::Prediction,
210     );
211
212     result
213 }
214 }
```

Файл: ./src/lib/mod.rs

```
1 pub mod ls;
```

Файл: ./tests/ls.rs

```

1 use approx::assert_relative_eq;
2 use prediction::ls::{StepType, LS};
3
4 #[cfg(test)]
5 mod tests {
6     use super::*;
7
8     #[test]
9     fn test_new_ls() {
10         let ls = LS::new(5, 3);
11         assert_eq!(ls.n, 5);
12         assert_eq!(ls.m, 3);
13         assert_eq!(ls.x.len(), 5);
14         assert_eq!(ls.y.len(), 5);
15         assert_eq!(ls.s.len(), 4);
16         assert_eq!(ls.q.len(), 4);
17         assert_eq!(ls.p.len(), 4);
18         assert_eq!(ls.c.len(), 4);
19         assert_eq!(ls.b.len(), 4);
20         assert_eq!(ls.a.len(), 4);
21         assert!(ls.steps.is_empty());
22     }
23
24     #[test]
25     fn test_initialize_points() {
26         let mut ls = LS::new(3, 2);
27         ls.initialize_points();
28
29         // Check x values
30         assert_eq!(ls.x, vec![0.0, 1.0, 2.0]);
31
32         // Check y values ( $y = x^3 + 2x^2 + 3x + 4$ )
33         assert_eq!(ls.y[0], 4.0); // when x = 0
34         assert_eq!(ls.y[1], 10.0); // when x = 1
35         assert_eq!(ls.y[2], 26.0); // when x = 2
36
37         // Check steps recording
38         assert_eq!(ls.steps.len(), 1);
39         assert!(matches!(ls.steps[0].step_type, StepType::DataPreparation));
40     }
41
42     #[test]
43     fn test_initialize_points_empty() {
44         let mut ls = LS::new(0, 2);
45         ls.initialize_points();
46
47         assert_eq!(ls.n, 1);

```

```

48     assert_eq!(ls.x.len(), 1);
49     assert_eq!(ls.y.len(), 1);
50 }
51
52 #[test]
53 fn test_clear_steps() {
54     let mut ls = LS::new(3, 2);
55     ls.initialize_points();
56     assert!(!ls.steps.is_empty());
57
58     ls.clear_steps();
59     assert!(ls.steps.is_empty());
60 }
61
62 #[test]
63 fn test_orthogonal_polynomials() {
64     let mut ls = LS::new(3, 2);
65     ls.initialize_points();
66     ls.calculate_orthogonal_polynomials();
67
68     // Check if p0 is all ones
69     assert!(ls.p[0].iter().all(|&x| (x - 1.0).abs() < 1e-10));
70
71     // Check steps recording
72     assert!(ls
73         .steps
74         .iter()
75         .any(|step| matches!(step.step_type, StepType::OrthogonalPolynomials)));
76 }
77
78 #[test]
79 fn test_calculate_coefficients() {
80     let mut ls = LS::new(4, 2);
81     ls.initialize_points();
82     ls.calculate_coefficients();
83
84     // Check if coefficients were calculated
85     assert!(!ls.a.iter().all(|&x| x == 0.0));
86     assert!(!ls.b.iter().all(|&x| x == 0.0));
87
88     // Check steps recording
89     assert!(ls
90         .steps
91         .iter()
92         .any(|step| matches!(step.step_type, StepType::CoefficientCalculation)));
93     assert!(ls
94         .steps
95         .iter()
96         .any(|step| matches!(step.step_type, StepType::FinalResult)));

```

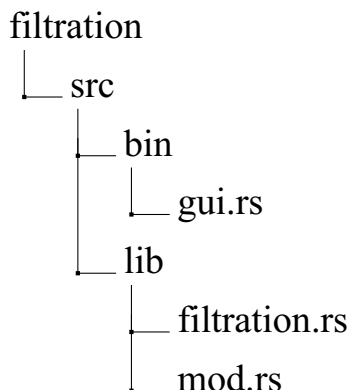
```

97     }
98
99     #[test]
100     fn test_prediction() {
101         let mut ls = LS::new(5, 3);
102         ls.initialize_points();
103         ls.calculate_coefficients();
104
105         let prediction = ls.predict(1.5);
106
107         // The prediction should be close to the actual polynomial value
108         let expected = ((1.5 + 2.0) * 1.5 + 3.0) * 1.5 + 4.0;
109         assert_relative_eq!(prediction, expected, epsilon = 1e-10);
110
111         // Check steps recording
112         assert!(ls
113             .steps
114             .iter()
115             .any(|step| matches!(step.step_type, StepType::Prediction)));
116     }
117
118     #[test]
119     fn test_edge_cases() {
120         let mut ls = LS::new(1, 1);
121         ls.initialize_points();
122         ls.calculate_coefficients();
123
124         // Test prediction at extreme values
125         let prediction_large = ls.predict(1000.0);
126         assert!(!prediction_large.is_nan());
127
128         let prediction_negative = ls.predict(-1000.0);
129         assert!(!prediction_negative.is_nan());
130     }
131
132     #[test]
133     fn test_calculate_coefficients_empty() {
134         let mut ls = LS::new(0, 1);
135         ls.calculate_coefficients();
136
137         // Verify that coefficients remain unchanged
138         assert_eq!(ls.a, vec![0.0, 0.0]);
139         assert_eq!(ls.b, vec![0.0, 0.0]);
140         assert!(ls.steps.is_empty());
141     }
142 }

```

ДОДАТОК Е Вихідний код додатку «filtration»

Структура каталогів



Програмний код Файл: ./src/bin/gui.rs

```

1 use eframe::egui;
2 use egui_plot::{Line, Plot, PlotPoints, Points};
3 use filtration::FiltrationMatrix;
4 use std::sync::{Arc, Mutex};
5 use std::thread;
6
7 struct FiltrationApp {
8     matrix: Arc<Mutex<FiltrationMatrix>>,
9     error_data: Option<(Vec<f64>, Vec<f64>)>,
10    depression_line: Option<(Vec<f64>, Vec<f64>)>,
11    control_matrix: Option<Vec<f64>>,
12    input_matrix: Option<Vec<f64>>,
13    data_matrix: Option<Vec<f64>>,
14    matrix_dims: (usize, usize),
15    status: String,
16    processing: bool,
17    current_iteration: usize,
18    max_iterations: usize,
19    result_receiver: Option<
20        std::sync::mpsc::Receiver<(Vec<f64>, Vec<f64>, Vec<f64>, Vec<f64>, Vec<f64>, Vec<f64>)>,
21    >,
22 }
23
24 impl FiltrationApp {
25     fn new(_cc: &eframe::CreationContext<'_>) -> Self {
26         Self {
27             matrix: Arc::new(Mutex::new(FiltrationMatrix::new(16, 40))),
28             error_data: None,
29             depression_line: None,
30             control_matrix: None,
31             input_matrix: None,
32             data_matrix: None,

```

```

33     matrix_dims: (16, 40),
34     status: "Ready".to_string(),
35     processing: false,
36     current_iteration: 0,
37     max_iterations: 0,
38     result_receiver: None,
39 }
40 }
41
42 fn process_data(&mut self) {
43     if self.processing {
44         return;
45     }
46
47     self.processing = true;
48     self.status = "Processing...".to_string();
49
50     let matrix = Arc::clone(&self.matrix);
51     let (sender, receiver) = std::sync::mpsc::channel();
52     self.result_receiver = Some(receiver);
53
54     thread::spawn(move || {
55         let mut locked_matrix = matrix.lock().unwrap();
56         if let Ok(()) = locked_matrix.initialize() {
57             let errors = locked_matrix.calculate();
58
59             // Convert error data to plot format
60             let x: Vec<f64> = errors.cycle.iter().map(|&x| x as f64).collect();
61             let y: Vec<f64> = errors.value;
62
63             // Get depression line
64             let (dx, dy) = locked_matrix.get_depression_line();
65
66             // Get matrix data
67             let control_data = locked_matrix.get_control_data();
68
69             // Save results
70             let _ = locked_matrix.save_result();
71
72             let _ = sender.send((x, y, dx, dy, control_data, Vec::new()));
73             ↪ // Send empty Vec for unused result
74         }
75     });
76
77 fn check_results(&mut self) {
78     if let Some(receiver) = &self.result_receiver {
79         if let Ok((x, y, dx, dy, control, _)) = receiver.try_recv() {
80             // Use _ to ignore unused result

```



```

81     self.error_data = Some((x, y));
82     self.depression_line = Some((dx, dy));
83     self.control_matrix = Some(control);
84
85     if let Ok(locked_matrix) = self.matrix.lock() {
86         self.input_matrix = Some(locked_matrix.get_input_data());
87         self.max_iterations = locked_matrix.get_iteration_count();
88         self.current_iteration = self.max_iterations.saturating_sub(1);
89         self.data_matrix =
90             Some(locked_matrix.get_iteration_data(self.current_iteration));
91     }
92
93     self.processing = false;
94     self.status = "Complete".to_string();
95     self.result_receiver = None;
96 }
97
98 }
99
100 fn update_current_iteration(&mut self) {
101     if let Ok(locked_matrix) = self.matrix.lock() {
102         self.data_matrix = Some(locked_matrix.get_iteration_data(self.current_iteration));
103     }
104 }
105
106 fn draw_matrix(&self, ui: &mut egui::Ui, data: &[f64], title: &str) {
107     ui.vertical_centered(|ui| {
108         ui.heading(title);
109         Plot::new(title)
110             .height(400.0)
111             .width(1000.0)
112             .data_aspect(1.0)
113             .show_axes([true, true])
114             .show_grid([true, true])
115             .include_x(-0.5)
116             .include_x(self.matrix_dims.1 as f64 - 0.5)
117             .include_y(-0.5)
118             .include_y(15.5)
119             .y_axis_label("Row")
120             .x_axis_label("Column")
121             .label_formatter(move |name, value| {
122                 if !name.is_empty() {
123                     format!("{: ({:.1}, {:.1})", name, value.x, value.y)
124                 } else {
125                     let row = value.y.floor();
126                     if row >= 0.0 && row < 16.0 {
127                         format!("{:}", (16.0 - row) as i32)
128                     } else {
129                         String::new()

```

```

130         }
131     }
132 })
133 .show(ui, |plot_ui| {
134     // Draw border points
135     let border_points: Vec<[f64; 2]> = vec![
136         [-0.5, 15.5],
137         [self.matrix_dims.1 as f64 - 0.5, 15.5],
138         [self.matrix_dims.1 as f64 - 0.5, -0.5],
139         [-0.5, -0.5],
140         [-0.5, 15.5],
141     ];
142     plot_ui.line(
143         Line::new(border_points)
144             .color(egui::Color32::GRAY)
145             .width(2.0),
146     );
147
148     if title == "Control Matrix" {
149         // For control matrix, show only points where value is 1
150         let points: Vec<[f64; 2]> = data
151             .chunks_exact(self.matrix_dims.1)
152             .enumerate()
153             .flat_map(|(y, row)| {
154                 row.iter().enumerate().filter_map(move |(x, &value)| {
155                     if value == 1.0 {
156                         Some([x as f64, y as f64])
157                     } else {
158                         None
159                     }
160                 })
161             })
162             .collect();
163
164         if !points.is_empty() {
165             plot_ui.points(
166                 Points::new(points)
167                     .color(egui::Color32::RED)
168                     .radius(2.0)
169                     .filled(true),
170             );
171         }
172     } else {
173         // For other matrices, use blue color visualization
174         let points_with_values: Vec<([f64; 2], f64)> = data
175             .chunks_exact(self.matrix_dims.1)
176             .enumerate()
177             .flat_map(|(y, row)| {
178                 row.iter().enumerate().filter_map(move |(x, &value)| {

```

```

179         if value > 0.0 {
180             // Пропускаем нулевые значения
181             Some((x as f64, y as f64), value))
182         } else {
183             None
184         }
185     })
186 })
187 .collect();
188
189 // Находим максимальное значение для нормализации
190 let max_value = points_with_values
191     .iter()
192     .map(|(_, v)| *v)
193     .max_by(|a, b| a.partial_cmp(b).unwrap_or(std::cmp::Ordering::Equal))
194     .unwrap_or(1.0);
195
196 // Отображаем все точки сразу с соответствующей интенсивностью синего
197 let points: Vec<[f64; 2], egui::Color32> = points_with_values
198     .iter()
199     .map(|(p, v)| {
200         let intensity = (*v / max_value * 255.0) as u8;
201         (
202             *p,
203             egui::Color32::from_rgb(0, 0, intensity.saturating_add(100)),
204         )
205     })
206     .collect();
207
208 // Группируем точки по цвету для оптимизации
209 let mut color_groups: std::collections::HashMap<
210     egui::Color32,
211     Vec<[f64; 2]>,
212 > = std::collections::HashMap::new();
213
214 for (point, color) in points {
215     color_groups
216         .entry(color)
217         .or_insert_with(Vec::new)
218         .push(point);
219 }
220
221 // Отрисовываем группы точек
222 for (color, points) in color_groups {
223     plot_ui
224         .points(Points::new(points).color(color).radius(2.0).filled(true));
225 }
226 }
227 });

```

```

228     });
229 }
230 }
231
232 impl eframe::App for FiltrationApp {
233     fn update(&mut self, ctx: &egui::Context, _frame: &mut eframe::Frame) {
234         self.check_results();
235
236         egui::CentralPanel::default().show(ctx, |ui| {
237             egui::ScrollArea::vertical().show(ui, |ui| {
238                 ui.heading("Filtration Analysis");
239
240                 ui.horizontal(|ui| {
241                     if ui.button("Process Data").clicked() {
242                         self.process_data();
243                     }
244
245                     ui.label(&self.status);
246                 });
247
248                 if self.max_iterations > 0 {
249                     ui.add_space(10.0);
250                     ui.horizontal(|ui| {
251                         ui.label("Iteration:");
252                         if ui
253                             .add(egui::Slider::new(
254                                 &mut self.current_iteration,
255                                 0..=self.max_iterations.saturating_sub(1),
256                             ))
257                             .changed()
258                         {
259                             self.update_current_iteration();
260                         }
261                     });
262                 }
263
264                 ui.separator();
265
266                 if let Some(control_data) = &self.control_matrix {
267                     ui.vertical_centered(|ui| {
268                         self.draw_matrix(ui, control_data, "Control Matrix");
269                     });
270                 }
271
272                 if let Some(input_data) = &self.input_matrix {
273                     ui.vertical_centered(|ui| {
274                         self.draw_matrix(ui, input_data, "Initial Data Matrix");
275                     });
276                 }

```

```

277
278     if let Some(data_matrix) = &self.data_matrix {
279         ui.vertical_centered(|ui| {
280             self.draw_matrix(ui, data_matrix, "Current Result Matrix");
281         });
282     }
283
284     ui.separator();
285
286     if let Some((x, y)) = &self.error_data {
287         ui.vertical_centered(|ui| {
288             ui.heading("Error Plot");
289             Plot::new("error_plot")
290                 .height(400.0)
291                 .width(1000.0)
292                 .show(ui, |plot_ui| {
293                 let points: PlotPoints =
294                     x.iter().zip(y.iter()).map(|(&x, &y)| [x, y]).collect();
295                 plot_ui.line(Line::new(points));
296             });
297         });
298     }
299
300     if let Some((x, y)) = &self.depression_line {
301         ui.vertical_centered(|ui| {
302             ui.heading("Depression Line");
303             Plot::new("depression_plot")
304                 .height(400.0)
305                 .width(1000.0)
306                 .show(ui, |plot_ui| {
307                 let points: PlotPoints =
308                     x.iter().zip(y.iter()).map(|(&x, &y)| [x, y]).collect();
309                 plot_ui.line(Line::new(points));
310             });
311         });
312     }
313
314     // Добавляем комбинированный график
315     if let (Some(control_data), Some((x, y))) =
316         (&self.control_matrix, &self.depression_line)
317     {
318         ui.vertical_centered(|ui| {
319             ui.heading("Combined View");
320             Plot::new("combined_plot")
321                 .height(400.0)
322                 .width(1000.0)
323                 .data_aspect(1.0)
324                 .show_axes([true, true])
325                 .show_grid([true, true])

```

```

326 .include_x(-0.5)
327 .include_x(self.matrix_dims.1 as f64 - 0.5)
328 .include_y(-0.5)
329 .include_y(15.5)
330 .y_axis_label("Row")
331 .x_axis_label("Column")
332 .label_formatter(move |name, value| {
333     if !name.is_empty() {
334         format!("{: ({:.1}, {:.1})", name, value.x, value.y)
335     } else {
336         let row = value.y.floor();
337         if row >= 0.0 && row < 16.0 {
338             format!("{}", (16.0 - row) as i32)
339         } else {
340             String::new()
341         }
342     }
343 })
344 .show(ui, |plot_ui| {
345     // Рисуем рамку
346     let border_points: Vec<[f64; 2]> = vec![
347         [-0.5, 15.5],
348         [self.matrix_dims.1 as f64 - 0.5, 15.5],
349         [self.matrix_dims.1 as f64 - 0.5, -0.5],
350         [-0.5, -0.5],
351         [-0.5, 15.5],
352 ];
353     plot_ui.line(
354         Line::new(border_points)
355             .color(egui::Color32::GRAY)
356             .width(2.0),
357     );
358
359     // Рисуем контрольную матрицу полупрозрачным серым цветом
360     let control_points: Vec<[f64; 2]> = control_data
361         .chunks_exact(self.matrix_dims.1)
362         .enumerate()
363         .flat_map(|(y, row)| {
364             row.iter().enumerate().filter_map(move |(x, &value)| {
365                 if value == 1.0 {
366                     Some([x as f64, y as f64])
367                 } else {
368                     None
369                 }
370             })
371         })
372         .collect();
373
374     plot_ui.points(

```

```

375         Points::new(control_points)
376         .color(egui::Color32::from_rgba_unmultiplied(
377             128, 128, 128, 180,
378         ))
379         .radius(2.0)
380         .filled(true),
381     );
382
383     // Рисуем линию депрессии жирной синей линией
384     let depression_points: PlotPoints =
385         x.iter().zip(y.iter()).map(|(&x, &y)| [x, y]).collect();
386     plot_ui.line(
387         Line::new(depression_points)
388         .color(egui::Color32::from_rgb(0, 0, 255))
389         .width(3.0),
390     );
391     });
392     });
393     }
394     });
395     });
396
397     ctx.request_repaint();
398 }
399 }
400
401 fn main() -> eframe::Result<()> {
402     let native_options = eframe::NativeOptions::default();
403     eframe::run_native(
404         "Filtration Analysis",
405         native_options,
406         Box::new(|cc| Box::new(FiltrationApp::new(cc))),
407     )
408 }

```

Файл: ./src/lib/filtration.rs

```

1 use csv::WriterBuilder;
2 use ndarray::Array2;
3 use serde::{Deserialize, Serialize};
4 use std::io;
5
6 #[derive(Debug, Clone, Default)]
7 pub struct FiltrationMatrix {
8     rows: usize,
9     columns: usize,
10    accuracy: f64,
11    control: Array2<f64>,
12    input: Array2<f64>,
13    data: Array2<f64>,
14    iterations: Vec<Array2<f64>>,
15 }
16
17 #[derive(Debug, Serialize, Deserialize)]
18 pub struct ErrorData {
19     pub cycle: Vec<usize>,
20     pub value: Vec<f64>,
21 }
22
23 impl FiltrationMatrix {
24     pub fn new(rows: usize, columns: usize) -> Self {
25         Self {
26             rows,
27             columns,
28             accuracy: 0.005,
29             control: Array2::zeros((rows, columns)),
30             input: Array2::zeros((rows, columns)),
31             data: Array2::zeros((rows, columns)),
32             iterations: Vec::new(),
33         }
34     }
35
36     fn calc_value(&self, filename: &str, row: usize, column: usize) -> f64 {
37         match filename {
38             "control.csv" => {
39                 if row >= 1 && row <= self.rows - 2 {
40                     if (column >= row + 5) && column <= self.columns / 2 {
41                         return 0.0;
42                     }
43                     if (column <= self.columns - row - 5) && column >= self.columns / 2 {
44                         return 0.0;
45                     }
46                 }
47                 1.0

```



```

48     }
49     "input.csv" => {
50         if row >= 1 && row <= self.rows - 2 {
51             if (column <= row + 4) && column <= self.columns / 2 {
52                 if row <= 12 {
53                     return row as f64;
54                 } else if row == 13 {
55                     return 13.0;
56                 } else if row == 14 {
57                     return 14.0;
58                 }
59             }
60         }
61         0.0
62     }
63     _ => 0.0,
64 }
65 }
66
67 pub fn generate_matrix(&mut self, filename: &str) -> io::Result<()> {
68     let mut writer = WriterBuilder::new().has_headers(true).from_path(filename)?;
69
70     let header: Vec<String> = (0..self.columns).map(|x| x.to_string()).collect();
71     writer.write_record(&header)?;
72
73     // Generate all rows first
74     let mut rows: Vec<Vec<String>> = (0..self.rows)
75         .map(|i| {
76             (0..self.columns)
77                 .map(|j| self.calc_value(filename, i, j).to_string())
78                 .collect()
79         })
80         .collect();
81
82     // Write rows in reverse order
83     for row in rows.iter().rev() {
84         writer.write_record(row)?;
85     }
86
87     writer.flush()?;
88     Ok(())
89 }
90
91 pub fn read_matrix(&mut self, filename: &str) -> io::Result<Array2<f64>> {
92     let mut matrix = Array2::zeros((self.rows, self.columns));
93     let mut rdr = csv::Reader::from_path(filename)?;
94
95     // First, collect all rows
96     let mut rows: Vec<Vec<f64>> = Vec::new();

```

```

97     for result in rdr.records() {
98         let record = result?;
99         let row: Vec<f64> = record
100             .iter()
101             .map(|value| value.parse().unwrap_or(0.0))
102             .collect();
103         rows.push(row);
104     }
105
106     // Take only the first 16 rows if we have more
107     let rows = if rows.len() > self.rows {
108         &rows[..self.rows]
109     } else {
110         &rows[..]
111     };
112
113     // Fill the matrix in reverse order
114     for (matrix_row, file_row) in (0..self.rows).zip(rows.iter().rev()) {
115         for (col_idx, &value) in file_row.iter().enumerate() {
116             matrix[[matrix_row, col_idx]] = value;
117         }
118     }
119
120     Ok(matrix)
121 }
122
123 pub fn initialize(&mut self) -> io::Result<()> {
124     println!("Generating input...");
125     self.generate_matrix("control.csv"?);
126     self.generate_matrix("input.csv"?);
127     println!("Input generated!");
128
129     // First read the control matrix
130     self.control = self.read_matrix("control.csv"?);
131
132     // Then read the input matrix and set it as initial data
133     self.input = self.read_matrix("input.csv"?);
134     self.data = self.input.clone();
135
136     // Clear any previous iterations
137     self.iterations.clear();
138
139     Ok(())
140 }
141
142 pub fn calculate(&mut self) -> ErrorData {
143     let mut errors = ErrorData {
144         cycle: vec![0],
145         value: vec![100.0],

```

```

146     };
147
148     let mut k = 0;
149     let mut error = 100.0;
150
151     // Store initial state
152     self.iterations.clear();
153     self.iterations.push(self.data.clone());
154
155     while error > self.accuracy {
156         error = 0.0;
157         k += 1;
158         let mut new_data = Array2::zeros((self.rows, self.columns));
159
160         for i in 0..self.rows {
161             for j in 0..self.columns {
162                 if self.control[[i, j]] == 0.0 {
163                     if i > 0 && i < self.rows - 1 && j > 0 && j < self.columns - 1 {
164                         new_data[[i, j]] = (self.data[[i + 1, j]]
165                             + self.data[[i - 1, j]]
166                             + self.data[[i, j + 1]]
167                             + self.data[[i, j - 1]])
168                             / 4.0;
169
170                         if new_data[[i, j]].abs() > 0.0 {
171                             let temp_error = ((new_data[[i, j]] - self.data[[i, j]]).abs()
172                                 / new_data[[i, j]].abs())
173                                 * 100.0;
174                             error = error.max(temp_error);
175                         }
176                     }
177                     } else {
178                         new_data[[i, j]] = self.data[[i, j]];
179                     }
180                 }
181             }
182
183             self.data = new_data.clone();
184             self.iterations.push(new_data);
185             errors.cycle.push(k);
186             errors.value.push(error);
187         }
188
189         errors
190     }
191
192     pub fn get_depression_line(&self) -> (Vec<f64>, Vec<f64>) {
193         let mut x = Vec::new();
194         let mut values = Vec::new();

```

```

195
196     for i in 0..self.columns {
197         if i <= 37 {
198             let mut max_val = 0.0;
199             for j in 0..self.rows {
200                 if self.data[[j, i]] > max_val {
201                     max_val = self.data[[j, i]];
202                 }
203             }
204             x.push(i as f64);
205             values.push(max_val);
206         }
207     }
208
209     (x, values)
210 }
211
212 pub fn get_control_data(&self) -> Vec<f64> {
213     self.control.iter().cloned().collect()
214 }
215
216 pub fn get_input_data(&self) -> Vec<f64> {
217     self.input.iter().cloned().collect()
218 }
219
220 pub fn get_result_data(&self) -> Vec<f64> {
221     self.data.iter().cloned().collect()
222 }
223
224 pub fn get_iteration_data(&self, iteration: usize) -> Vec<f64> {
225     if iteration < self.iterations.len() {
226         self.iterations[iteration].iter().cloned().collect()
227     } else {
228         self.data.iter().cloned().collect()
229     }
230 }
231
232 pub fn get_iteration_count(&self) -> usize {
233     self.iterations.len()
234 }
235
236 pub fn save_result(&self) -> io::Result<()> {
237     let mut writer = WriterBuilder::new()
238         .has_headers(false)
239         .from_path("calculated.csv")?;
240
241     for i in 0..self.rows {
242         let row: Vec<String> = (0..self.columns)
243             .map(|j| self.data[[i, j]].to_string())

```

```
244         .collect();
245         writer.write_record(&row)?;
246     }
247
248     writer.flush()?;
249     Ok(())
250 }
251 }
252
253 #[cfg(test)]
254 mod tests {
255     use super::*;
256
257     #[test]
258     fn test_matrix_generation() {
259         let mut matrix = FiltrationMatrix::new(8, 20);
260         assert!(matrix.generate_matrix("test.csv").is_ok());
261     }
262 }
```

Файл: ./src/lib/mod.rs

```
1 mod filtration;  
2 pub use filtration::FiltrationMatrix;
```

ДОДАТОК Ж Список публікацій здобувача

Список публікацій здобувача/List of publications of the applicant:

Статті у закордонному фаховому виданні третього квартиля (Q3), які проіндексовані в базі даних Scopus / the articles published in a foreign scientific journal classified as third quartile (Q3) and indexed in the Scopus database:

1. Vladyslav Sokolovskyi, & Eduard Zharikov. (2023). Architectural solution for the distribution of software and hardware systems for monitoring potentially unsafe objects. GEOMATE Journal, 25(109), 141–148. Retrieved from <https://geomatejournal.com/geomate/article/view/4057>

2. Sokolovskyi, V., Zharikov, E., & Telenyk, S. (2024). Development of the method of detecting and correcting data transmission errors in IoT systems for monitoring the state of objects. Eastern-European Journal of Enterprise Technologies, 1(9 (127), 22–33. <https://doi.org/10.15587/1729-4061.2024.298476>

3. Sokolovskyi, V., Zharikov, E., & Telenyk, S. (2024). Using expert evaluation for selecting an architectural solution for a specialized software system that monitors the state of potentially hazardous facilities. Eastern-European Journal of Enterprise Technologies, 5(3 (131), 27–40. <https://doi.org/10.15587/1729-4061.2024.312886>

Статті у закордонному фаховому виданні четвертого квартиля (Q4), які проіндексовані в базі даних Scopus / the articles published in a foreign scientific journal classified as fourth quartile (Q4) and indexed in the Scopus database:

4. Sokolovskyi, V., Zharikov, E., & Telenyk, S. (2024). Software and algorithmic support as part of regional systems for monitoring the state of objects for calculation of filtration through earthen hydraulic structures. Machinery & Energetics, 15(2), 130-144. <https://doi.org/10.31548/machinery/2.2024.130>

Публікації у матеріалах наукових конференцій / the publications in the proceedings of the scientific conferences:

5. Соколовський В.В., Жаріков Е.В. Архітектура програмно-апаратної системи моніторингу стану об'єктів підвищеної небезпеки з можливістю прогнозування виникнення надзвичайної ситуації // III Всеукраїнська

науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології»(SoftTech-2022), 22-25 листопада. – Київ: НТУУ "КПІ ім. І. Сікорського", 2022. – С. 64-68

https://drive.google.com/file/d/1-kJ75f9HKjfQ4pL7SvNf_uW4NI2eb9HL

6. Соколовський В. В., Жаріков Е. В. Архітектурне рішення та програмне забезпечення програмованих дачів інформації для побудови регіональних IoT систем моніторингу стану потенційно небезпечних об'єктів. VI Міжнародна науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології (SOFT TECH-2024)» 21-23 травня 2024 року, м.Київ,с.128-135.

<https://drive.google.com/file/d/18bZ9QBure7U08rbqiHmxWglTrK1C9D4L>

7. Соколовський В. В., Жаріков Е. В. Прогнозування в системах моніторингу стану об'єктів підвищеної небезпеки регіонального рівня. VII Міжнародна науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології (SOFT TECH-2024)» 20-22 листопада 2024 року, м.Київ, с. 101-106.

<https://drive.google.com/file/d/1O70Ysxe-z3SS82UqEaBdEXR2VdRW3NwG>