МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кваліфікаційна наукова
праця на правах рукопису

КІНДЗЕРСЬКИЙ ОЛЕКСАНДР ВІТАЛІЙОВИЧ

УДК 519.876.5 : 004.891.2

ІДЕНТИФІКАЦІЯ ІНТЕРВАЛЬНИХ МОДЕЛЕЙ СИСТЕМ ПРОГРАМНИМИ АГЕНТАМИ БДЖОЛИНОЇ КОЛОНІЇ У СЕРЕДОВИЩІ NVIDIA CUDA

Спеціальність 121 – Інженерія програмного забезпечення
Галузь 12 – Інформаційні технології

Подається на здобуття ступеня доктора філософії.
Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ О.В.Кіндзерський

Науковий керівник: Дивак Микола Петрович, доктор технічних наук, професор

Тернопіль – 2026

АНОТАЦІЯ

Кіндзерський О.В. Ідентифікація інтервальних моделей систем програмними агентами бджолиної колонії у середовищі NVIDIA CUDA. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття ступеня доктора філософії за спеціальністю 121 «Інженерія програмного забезпечення» – Західноукраїнський національний університет, Тернопіль, 2026.

Підготовка здійснювалась на кафедрі комп'ютерних наук Західноукраїнського національного університету Міністерства освіти і науки України.

У дисертаційній роботі розглянуто науково-технічне розроблення методів структурної та параметричної ідентифікації інтервальних моделей систем програмними агентами метаевристичного алгоритму бджолиної колонії у середовищі NVIDIA CUDA. Метою дисертаційної роботи є зниження часової складності методів структурної та параметричної ідентифікації інтервальних моделей систем програмними агентами метаевристичного алгоритму бджолиної колонії у середовищі NVIDIA CUDA.

У вступі, наведено актуальність теми дисертаційного дослідження, мета та основні завдання, а також наукова новизна та практична значущість.

У першому розділі проведено детальний огляд проблематики ідентифікацією інтервальних моделей систем. В першому підрозділі розглянуто особливості побудови інтервальних моделей систем. Досліджено, що інтервальні моделі систем забезпечують явне й математично коректне представлення невизначеності у параметрах і вхідних впливах, надають можливість оцінювати стійкість і працездатність системи щодо збурень і похибок. В наступній частині розділу розглянуто методи параметричної та структурної ідентифікації інтервальних моделей систем. Виявлено обмеження існуючих методів ідентифікації у випадку дискретних моделей. Показано, що такого типу задачі мають NP-складність. Обґрунтовано доцільність застосування метаевристичних методів колективного інтелекту як інструменту глобальної оптимізації для

ідентифікації інтервальних моделей систем. Зокрема, обґрунтовано вибір алгоритму бджолиної колонії, як базового для подальшої реалізації завдяки його здатності підтримувати множину перспективних рішень, балансувати між пошуком нових областей і локальним уточненням та інтерпретуватися як система агентів, придатних до подальшої програмної реалізації. У заключній частині розділу описано прикладні задачі, що будуть використовуватися для апробації розроблених методів ідентифікації інтервальних моделей систем, та сформульовано науково-технічне завдання, мету та завдання дисертаційного дослідження і наведено висновки на підставі проведеного огляду літературних джерел. Зважаючи на проведений аналіз методів структурної та параметричної ідентифікації моделей систем і виявлені обмеження класичних методів ідентифікації було обґрунтовано доцільність застосування метаевристичних методів колективного інтелекту як інструменту глобальної оптимізації для ідентифікації інтервальних моделей.

У другому розділі проведено аналіз формального представлення задач параметричної і структурної ідентифікації інтервальних моделей систем. Обґрунтовано вибір технології NVIDIA CUDA для організації мультиагентної системи ідентифікації за рахунок масово паралельних обчислень на графічних процесорах. Описано метод паралельних обчислень ідентифікації з застосуванням алгоритму бджолиної колонії і сформовано гібридну схему CPU–GPU обчислень, де усі керувально-логічні фази АБК виконуються на CPU, а GPU відповідає за масово-паралельне обчислення функції мети для наборів кандидатів. Представлено універсальний шаблон ядра оцінки функції мети з динамічною спеціалізацією та компіляцією під конкретну прикладну модель. У цьому ж розділі, спираючись на розроблений метод паралельних обчислень ідентифікації з застосуванням алгоритму бджолиної колонії і гібридної схеми CPU–GPU обчислень представлено алгоритмічне забезпечення параметричної та структурної ідентифікації інтервальних моделей систем програмними агентами бджолиної колонії у середовищі NVIDIA CUDA.

У третьому розділі обґрунтовано архітектурні рішення програмної системи ідентифікації інтервальних моделей на основі алгоритму бджолиної колонії. Описано трирівнева організацію програмної системи – інтерфейс користувача, алгоритмічна бібліотека та обчислювальний CUDA бекенд. В продовжені розділу описано UML-діаграми, що формалізують як статичну структуру, так і динаміку системи. Компонентні та класові моделі відображають межі відповідальності модулів і ключові колаборації; діаграми станів описують життєві цикли контролера АБК і CUDA-виконавця з явними точками відновлення після збоїв; діаграма послідовностей показує конвеєр пакетного оцінювання та місця накладання операцій. Комп'ютерне середовище реалізоване з графічним інтерфейсом, що підтримує керування проєктами, імпорт/експорт CSV/JSON, валідацію параметрів, візуалізацію перебігу пошуку та результатів у реальному часі, профілювання. Відокремлення графічного інтерфейсу від обчислювальної бібліотеки спростило супровід і забезпечило можливість подальшого переходу до кросплатформних інтерфейсів без змін алгоритмічної частини. В завершальній частині розділу описано результати дослідження часової складності розроблених методів і програмних засобів. Показано, що в залежності від складності моделі і розмірності експериментальних даних, GPU-реалізація забезпечує скорочення часу ітерації алгоритму бджолиної колонії порівняно з CPU-реалізацією. Запропоновано та обґрунтовано вибір оптимального параметра чисельності бджолиної колонії залежно від розмірності задачі. На прикладах побудови інтервальних моделей для екологічного моніторингу встановлено закономірність, що для моделей малої розмірності (2-3 коефіцієнти) застосування запропонованої технології та обчислювальних методів є нераціональним. При зростанні розмірності інтервальної моделі до 10 коефіцієнтів та при виборі оптимального параметра чисельності бджолиної колонії – 4096 – спостерігається зменшення часової складності застосування запропонованої технології та обчислювальних методів в понад 45 разів. Встановлено, що чим вища розмірність задачі, тим вища ефективність застосування запропонованої технології та обчислювальних методів.

В четвертому розділі наведено результати апробації розробленої компютерного середовища ідентифікації інтервальних моделей систем для прикладних задач екологічного моніторингу, оцінювання достовірності контенту та реабілітації пацієнтів з порушенням рухливості верхніх кінцівок. Спершу розглянуто прикладні задачі екологічного моделювання шкідливих викидів автотранспорту, а саме побудова одновимірної моделі поширення концентрації окису вуглецю перпендикулярно до дорожнього полотна і побудова двовимірної моделі розподілу концентрації діоксиду азоту, без урахування його поширення на квадратній ділянці на висоті 1,5 над дорожнім полотном. Далі розглянуто моделювання портрету користувача соціальної новинної мережі. Резонансні теми можуть викликати вибухові сплески активності. У такому випадку зростає ризик поширення фейків чи маніпулятивного контенту через підвищену емоційність. У підсумку, особливістю таких спільнот є те, що їхній «портрет» сильно змінюється залежно від характеру новин і контексту, в якому вони з’являються. На основі даної особливості побудовано інтервальну модель, яка надає можливість оцінювання достовірності контенту, який розміщується в зазначеній мережі. Також розглянуто процес реабілітації пацієнтів з порушенням рухливості верхніх кінцівок. Пацієнт, в якого є проблеми з рухами верхніх кінцівок, проходить реабілітацію, під час якої фізіотерапевти вимірюють ключові кути згину суглобів верхніх кінцівок. На основі цих даних побудована моделі динаміки відновлення кутів рухливості суглобів верхніх кінцівок. Дана математична модель дає змогу терапевтам відстежувати відхилення від прогнозу та корегувати сеанси для персоналізації лікування, що допомагає планувати курс реабілітації та зменшувати загальну тривалість на 15-20%.

Практичне значення отриманих результатів полягає у розробці комп’ютерного середовища для математичного моделювання систем на основі аналізу інтервальних даних, яке на відміну від існуючих, імплементує інтерпретатор базисних функцій для побудови моделей та об’єднує паралельне функціонування програмних агентів і їх динамічну компіляцію, що у сукупності

спрощує доступ користувача до модулів інтервального моделювання та знижує часову складність побудови моделей.

Ключові слова: інтервальні моделі систем, математична модель, параметрична ідентифікація, структурна ідентифікація, поведінкова модель бджолиної колонії, інтервальний аналіз даних, GPU, паралельні обчислення, NVIDIA CUDA, обчислювальне моделювання, методи оптимізації, індуктивне моделювання, алгоритми ройового інтелекту.

ПЕРЕЛІК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковано основні наукові результати дисертації

1. M. Dyvak, P. Tyande, O. Kindzerskyi “Mathematical Model of a Social Network User Profile Based on Interval Data Analysis,” International Journal of Computing, vol. 24, no. 3, pp. 452-459, Oct. 2025. ISSN: 2312-538, (0,6 д.а. / 0,2 д.а.; особистий внесок: побудова математичної моделі профіля користувача соціальної мережі за допомогою алгоритму бджолиної колонії). doi:10.47839/ijc.24.3.4182 Url: https://computingonline.net/computing/article/view/4182

2. М. Дивак, О. Кіндзерський “Архітектура програмного забезпечення структурної та параметричної ідентифікації на основі алгоритму штучної бджолиної колонії з використанням технології NVIDIA CUDA,” НаукПраці ВНТУ, вип. 2, Чер 2025, ISSN: 2307-5376, (0,7 д.а. / 0,4 д.а.; особистий внесок: спроєктовано мультиагентну архітектуру програмного забезпечення ідентифікації інтервальних моделей систем на основі агентів алгоритму бджолиної колонії). doi:10.31649/2307-5376-2025-2-41-50 Url: https://praci.vntu.edu.ua/index.php/praci/article/view/837

3. М. Дивак, О. Кіндзерський “Дослідження ефективності паралельної обчислювальної схеми ідентифікації інтервальних дискретних моделей на основі ройового інтелекту” Том 331 № 1 (2024): Вісник Хмельницького національного університету. Серія: Технічні науки, с.29-37 (0,8 д.а. / 0,5 д.а.; особистий внесок: побудова математичних моделей шкідливих викидів автотранспорту, дослідження часової складності послідовної та паралельно реалізації методів параметричної ідентифікації), doi:10.31891/2307-5732-2024-331-3, ISSN: 2307-5732 Url:

https://heraldts.khmnu.edu.ua/index.php/heraldts/issue/view/2

Наукові праці, які засвідчують апробацію матеріалів дисертації

4. M. Dyvak, O. Kindzerskyi “Implementation of the structural identification for interval models based on the behavioral model of an artificial bee colony,” 2025 15th International Conference on Advanced Computer Information Technologies (ACIT), Sibenik, Croatia, 2025, pp.98-101, (0,45 д.а. / 0,3 д.а.; особистий внесок: реалізовано метод структурної ідентифікації з використанням розробленого раніше CUDA рішення параметричної ідентифікації), doi:10.1109/ACIT65614.2025.11185828, ISSN: 2770-5218

5. M. Dyvak, N. Petryshyn, O. Kindzerskyi, O. Papa, Y. Franko and O. Opalko, "Modeling of the Efficiency of Electricity Generation Processes by a Solar Power Plant Research Using the Example of a 570 W Model," 2025 15th International Conference on Advanced Computer Information Technologies (ACIT), Sibenik, Croatia, 2025, pp. 92-97, (0,5 д.а. / 0,1 д.а.; особистий внесок: побудова математичної моделі процесів генерації електроенергії), doi: 10.1109/ACIT65614.2025.11185608, ISSN: 2770-5218

6. M. Dyvak, O. Kindzerskyi “Implementation of Parallel Computation for Identification of Interval Models based on Multi-core Parallelism and CUDA Technology,” 2024 14th International Conference on Advanced Computer Information Technologies (ACIT), 2024, pp.72-76, (0,5 д.а. / 0,3 д.а.; особистий внесок: реалізовано паралельно обчислювальну схему ідентифікації для паралельних CPU потоків та для CUDA технології), doi: 10.1109/ACIT62333.2024.10712545. ISNN: 2770-5218

7. M. Dyvak, I. Spivak, T. Dyvak, O. Kindzerskyi “Modeling the Interaction of Unmanned Aerial Vehicles in a Swarm as an Object with Distributed Parameters,” 2024 14th International Conference on Advanced Computer Information Technologies (ACIT), 2024, pp.60-66., (0,6 д.а. / 0,1 д.а.; особистий внесок: побудовано математичну модель розміщення рою БПЛА), doi:10.1109/ACIT62333.2024.10712502. ISSN: 2770-5218

8. M. Dyvak, O. Kindzerskyi, L. Dostalek, M. Stetsko and J. Nowak “Parallel Computations in the Problem of Identification of Interval Discrete Models based on

Swarm Intelligence of a Bee Colony,” 2023 13th International Conference on Advanced Computer Information Technologies (ACIT), 2023, pp. 23-28, (0,6 д.а. / 0,3 д.а.; особистий внесок: запропоновано паралельно обчислювальну схему ідентифікації на основі алгоритму бджолиної колонії), doi: 10.1109/ACIT58437.2023.10275695, ISSN: 2770-5218

ANOTATION

Kindzerskyi O.V. Identification of Interval System Models by Bee Colony Software Agents in the NVIDIA CUDA Environment. – . Scientific work on the rights of the manuscript.

Thesis for the degree of Doctor of Philosophy in the specialty 121 – Software Engineering – West Ukrainian National University, Ternopil, 2026.

The research was carried out at the Department of Computer Sciences of the West Ukrainian National University, Ministry of Education and Science of Ukraine.

The dissertation addresses the scientific and technical problem of development of methods for structural and parametric identification of interval system models using software agents of the metaheuristic Artificial Bee Colony (ABC) algorithm in the NVIDIA CUDA environment. The aim of the dissertation is to reduce the time complexity of structural and parametric identification methods for interval system models by employing software agents of the metaheuristic ABC algorithm in the NVIDIA CUDA environment.

In the introduction, the relevance of the dissertation topic is substantiated, the purpose and main objectives are stated, and the scientific novelty and practical significance are outlined.

The first chapter provides a detailed review of the issues related to identifying interval system models. The first subsection discusses the specifics of constructing interval system models. It is shown that interval models ensure an explicit and mathematically correct representation of uncertainty in parameters and input influences and enable assessment of a system’s stability and operability under disturbances and errors. In the next part of the chapter, methods for parametric and structural identification of interval system models are considered. The limitations of existing identification methods for discrete models are identified. It is shown that problems of this type are NP-complex. The expediency of applying swarm-intelligence-based metaheuristic methods as instruments of global optimization for identifying interval system models is substantiated. In particular, the Artificial Bee Colony algorithm is justified as the baseline for further implementation due to its ability to maintain a set of promising solutions,

balance exploration of new regions with local refinement, and be interpreted as an agent-based system suitable for subsequent software implementation. In the concluding part of the chapter, application problems to be used for validating the developed methods for identifying interval system models are described, and the scientific and technical task, the purpose, and the objectives of the dissertation research are formulated, together with conclusions based on the literature review. Given the analysis of structural and parametric identification methods and the identified limitations of classical identification techniques, the expediency of applying swarm-intelligence metaheuristic methods as tools for global optimization in interval model identification is justified.

In the second chapter, an analysis of the formal representation of the problems of parametric and structural identification of interval system models is carried out. The choice of NVIDIA CUDA technology for organizing a multi-agent identification system through massively parallel computations on graphics processors is substantiated. A method for parallel identification computations using the Artificial Bee Colony algorithm is described and a hybrid CPU–GPU computation scheme is developed, in which all control-and-logic phases of the ABC algorithm are executed on the CPU, while the GPU is responsible for massively parallel evaluation of the objective function for sets (batches) of candidates. A universal objective-function evaluation kernel template is presented, with dynamic specialization and compilation for a specific applied model. Based on the proposed parallel computation method and the hybrid CPU–GPU scheme, the algorithmic support for parametric and structural identification of interval system models by bee-colony software agents in the NVIDIA CUDA environment is provided.

The third chapter substantiates the architectural decisions of the software system for identifying interval system models based on the Artificial Bee Colony algorithm. A three-tier organization of the software system is described, consisting of the user interface, the algorithmic library, and the CUDA computational backend. The chapter further describes UML diagrams that formalize both the static structure and the dynamic behavior of the system. Component and class models reflect module responsibilities and key collaborations; state diagrams describe the life cycles of the ABC controller and the CUDA executor with explicit recovery points after failures; the sequence diagram

illustrates the batched evaluation pipeline and the places where operations overlap. The computer environment is implemented with a graphical user interface that supports project management, CSV/JSON import/export, parameter validation, real-time visualization of the search process and results, and profiling. Separating the graphical interface from the computational library simplified maintenance and enabled a future transition to cross-platform interfaces without changes to the algorithmic part. The final part of the chapter presents the results of studying the time complexity of the developed methods and software tools. It is shown that, depending on model complexity and the dimensionality of experimental data, the GPU implementation reduces the iteration time of the Artificial Bee Colony algorithm compared to the CPU implementation. The choice of the optimal bee-colony population size parameter is proposed and justified depending on the problem dimension. Using examples of constructing interval models for environmental monitoring, it is established that for low-dimensional models (2–3 coefficients) the application of the proposed technology and computational methods is irrational. When the dimensionality of the interval model increases to 10 coefficients and the optimal bee-colony population size of 4096 is selected, the time complexity of applying the proposed technology and computational methods decreases by more than 45 times. It is found that the higher the problem dimension, the higher the efficiency of the proposed technology and computational methods.

The fourth chapter presents the results of validating the developed computer environment for identifying interval system models on applied problems of environmental monitoring, content credibility assessment, and rehabilitation of patients with impaired upper-limb mobility. First, applied problems of environmental modelling of harmful vehicle emissions are considered, namely, constructing a one-dimensional model of carbon monoxide concentration propagation perpendicular to the roadway and constructing a two-dimensional model of nitrogen dioxide concentration distribution without accounting for its dispersion over a square area at a height of 1.5 m above the roadway. Next, modelling of a social news network user profile is discussed. Resonant topics can cause explosive spikes in activity; in such cases, the risk of spreading fakes or manipulative content increases due to heightened emotionality. As a result, a feature of

such communities is that their “profile” changes substantially depending on the nature of news and the context in which it appears. Based on this feature, an interval model is constructed that enables assessment of the credibility of content posted in the network. Finally, the rehabilitation process of patients with impaired upper-limb mobility is considered. During rehabilitation, physiotherapists measure key joint flexion angles of the upper limbs. Based on these data, a dynamic model of recovery of upper-limb joint mobility angles is built. This mathematical model allows therapists to track deviations from the forecast and adjust sessions to personalize treatment, helping plan the rehabilitation course and reduce the overall duration by 15–20%.

The practical significance of the obtained results lies in the development of a computer environment for mathematical modeling of systems based on interval data analysis which, unlike existing solutions, implements an interpreter of basis functions for model construction and combines parallel operation of software agents with their dynamic compilation. Taken together, this simplifies user access to interval modeling modules and reduces the time complexity of model development.

Keywords: interval system models, mathematical model, parametric identification, structural identification, behavioural model of artificial bee colony, interval data analysis, GPU, parallel computing, NVIDIA CUDA, computational modelling, optimization methods, inductive modelling, swarm intelligence algorithms.

LIST OF PUBLISHED PAPERS BY THE TOPIC OF THESIS

Scientific papers in which the main scientific results of the dissertation were published:

1. M. Dyvak, P. Tyande, O. Kindzerskyi, “Mathematical Model of a Social Network User Profile Based on Interval Data Analysis,” International Journal of Computing, vol. 24, no. 3, pp. 452–459, Oct. 2025. ISSN: 2312-538, (0.6 p.s. / 0.2 p.s.; personal contribution: development of a mathematical model of a social network user profile using the artificial bee colony algorithm). doi:10.47839/ijc.24.3.4182 URL: https://computingonline.net/computing/article/view/4182

2. M. Dyvak, O. Kindzerskyi, “Software Architecture for Structural and Parametric Identification Based on the Artificial Bee Colony Algorithm Using NVIDIA CUDA Technology,” Naukovi Pratsi of VNTU, issue 2, Jun. 2025. ISSN: 2307-5376, (0.7 p.s. /

0.4 p.s.; personal contribution: design of a multi-agent software architecture for identifying interval system models based on bee colony algorithm agents). doi:10.31649/2307-5376-2025-2-41-50

URL: https://praci.vntu.edu.ua/index.php/praci/article/view/837

3. M. Dyvak, O. Kindzerskyi, “Study of the Efficiency of a Parallel Computational Scheme for Identification of Interval Discrete Models Based on Swarm Intelligence,” Vol. 331, No. 1 (2024): Herald of Khmelnytskyi National University. Series: Technical Sciences, pp. 29–37, (0.8 p.s. / 0.5 p.s.; personal contribution: development of mathematical models of harmful motor-vehicle emissions; study of the time complexity of sequential and parallel implementations of parametric identification methods). doi:10.31891/2307-5732-2024-331-3. ISSN: 2307-5732

URL: https://heraldts.khmnu.edu.ua/index.php/heraldts/issue/view/2

Scientific works certifying the approval of the dissertation materials:

4. M. Dyvak, O. Kindzerskyi, “Implementation of the Structural Identification for Interval Models Based on the Behavioral Model of an Artificial Bee Colony,” in 2025 15th International Conference on Advanced Computer Information Technologies (ACIT), Šibenik, Croatia, 2025, pp. 98–101, (0.45 p.s. / 0.3 p.s.; personal contribution: implementation of the structural identification method using the previously developed CUDA-based solution for parametric identification). doi:10.1109/ACIT65614.2025.11185828. ISSN: 2770-5218

5. M. Dyvak, N. Petryshyn, O. Kindzerskyi, O. Papa, Y. Franko, and O. Opalko, “Modeling of the Efficiency of Electricity Generation Processes by a Solar Power Plant Research Using the Example of a 570 W Model,” in 2025 15th International Conference on Advanced Computer Information Technologies (ACIT), Šibenik, Croatia, 2025, pp. 92–97, (0.5 p.s. / 0.1 p.s.; personal contribution: development of a mathematical model of electricity generation processes). doi:10.1109/ACIT65614.2025.11185608. ISSN: 2770-5218

6. M. Dyvak, O. Kindzerskyi, “Implementation of Parallel Computation for Identification of Interval Models Based on Multi-core Parallelism and CUDA Technology,” in 2024 14th International Conference on Advanced Computer Information

Technologies (ACIT), 2024, pp. 72–76, (0.5 p.s. / 0.3 p.s.; personal contribution: implementation of a parallel computational identification scheme for parallel CPU threads and for CUDA technology). doi:10.1109/ACIT62333.2024.10712545. ISSN: 2770-5218

7. M. Dyvak, I. Spivak, T. Dyvak, O. Kindzerskyi, “Modeling the Interaction of Unmanned Aerial Vehicles in a Swarm as an Object with Distributed Parameters,” in 2024 14th International Conference on Advanced Computer Information Technologies (ACIT), 2024, pp. 60–66, (0.6 p.s. / 0.1 p.s.; personal contribution: development of a mathematical model of UAV swarm deployment). doi:10.1109/ACIT62333.2024.10712502. ISSN: 2770-5218

8. M. Dyvak, O. Kindzerskyi, L. Dostalek, M. Stetsko, and J. Nowak, “Parallel Computations in the Problem of Identification of Interval Discrete Models Based on Swarm Intelligence of a Bee Colony,” in 2023 13th International Conference on Advanced Computer Information Technologies (ACIT), 2023, pp. 23–28, (0.6 p.s. / 0.3 p.s.; personal contribution: proposal of a parallel computational identification scheme based on the bee colony algorithm). doi:10.1109/ACIT58437.2023.10275695. ISSN: 2770-5218

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

АБК	Алгоритм бджолиної колонії
GPU	Graphics processing unit, графічний процесор
CPU	Central processing unit, центральний процесор
МНК	Метод найменших квадратів
ГА	Генетичні алгоритми
PSO	Particle Swarm Optimization
PTX	Parallel Thread Execution file format
H2D	Host to device – передача даних в пам’ять графічного процесора
D2H	Device to host – отримання даних з пам’яті графічного процесора

ВСТУП

Актуальність теми дослідження. Сучасні інформаційні технології та системи підтримки прийняття рішень, які ґрунтуються на них, відзначаються широким використанням математичних моделей реалізованих за допомогою програмних засобів [24]. Універсальність розроблювальних технологій дає можливість їх застосування для широкого кола різнорідних задач прийняття рівень, зокрема задач екологічного моніторингу та прогнозування, медицини та задач контент аналізу. В теорії математичного моделювання зазвичай застосовують один із двох підходів: дедуктивний чи індуктивний [61]. Дедуктивний підхід вимагає висунення гіпотез щодо структури та властивостей майбутніх моделей. На противагу індуктивний підхід ґрунтується безпосередньо на даних спостереження за об'єктом моделювання. Для прикладних задач прийняття рішень більш придатним є індуктивний підхід. Проте його застосування вимагає розв'язування задачі структурної [77, 93] та параметричної ідентифікації моделі [29, 43, 73] і відповідно створення програмних середовищ для побудови таких моделей. Ще однією проблемою при розв'язуванні задач ідентифікації є неточність даних. В межах стохастичного підходу розроблено ряд методів та програмних середовищ, які враховують ці похибки. Зокрема варто відзначити такі середовища як Matlab [115], R [102] та Python (NumPy/SciPy) [101, 120]. Недоліком стохастичного підходу є необхідність значної вибірки даних. Тому в доповнення до стохастичного підходу розвиваються методи аналізу інтервальних даних [63], які дають можливість в межах індуктивного підходу будувати моделі, що ґрунтуються на коротких вибірках даних. В цьому випадку отриманні дані з врахуванням похибок подають у вигляді часових інтервалів [18], а аналіз цих даних проводять процедур інтервального аналізу. Методи структурної та параметричної математичних моделей, що ґрунтуються на аналізі інтервальних даних розглянуті у роботах Вощініна А.П., Бакана Г.М., Куссуль Н. Н., Дивака М.П. [1 – 7, 12]. Разом з тим програмна реалізація цих методів є ускладненою через необхідність множинних розв'язків інтервальних систем нелінійних алгебраїчних рівняння. Задачі структурної та параметричної ідентифікації на основі інтервальних даних

відзначаються NP-обчислювальною складністю. Тому для їх розв’язування переважно використовують метаевристичні алгоритми оптимізації [112]. Останнім часом серед них широкого застосування набули алгоритми мультиагентного пошуку – алгоритм рою частинок [80, 114], мурашиний алгоритм [34, 38], алгоритм вовчої зграї [22] та алгоритм бджолиної колонії [42, 43, 67, 68, 76, 78, 106, 122, 123]. Такі підходи призводять до того, що зі збільшенням кількості агентів і розмірності оптимізаційної задачі, яка є задачею ідентифікації, різко зростає обчислювальна складність, що обмежує застосування таких методів в існуючих програмних середовищах. Одним із підходів, який забезпечує зниження часової складності є забезпечення функціонування агентів мультиагентних системи паралельно [17, 52, 103], зокрема із використання сучасних високопродуктивних обчислень на графічних прискорювачах таких як NVIDIA CUDA [32, 33]. Серед наведених вище метаеврестичних алгоритмів для паралельної роботи агентів найбільш прийнятним є алгоритми, що ґрунтуються на поведінковій моделі бджолиної колонії. Таким чином актуальним є науково-технічне завдання розроблення методів структурної та параметричної ідентифікації інтервальних моделей систем програмними агентами метаеврестичного алгоритму бджолиної колонії у середовищі NVIDIA CUDA. Розроблення відповідних методів та програмних засобів забезпечить зниження часової складності розв’язування задач структурної та параметричної ідентифікації інтервальних моделей систем, а створення програмного середовища надасть можливості користувачам розв’язати широкий спектр прикладних задач.

Мета і завдання дослідження. Метою дисертаційної роботи є зниження часової складності методів структурної та параметричної ідентифікації інтервальних моделей систем програмними агентами метаеврестичного алгоритму бджолиної колонії у середовищі NVIDIA CUDA.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

1. Проаналізувати існуючі методи та програмні засоби ідентифікації інтервальних моделей систем.

2. Розробити обчислювальні методи структурної та параметричної ідентифікації інтервальних моделей систем із застосуванням програмних агентів, що виконують функції бджолиної колонії паралельно.

3. Розробити технологію застосування середовища NVIDIA CUDA для реалізації програмних агентів, що виконують функції бджолиної колонії для реалізації методів структурної та параметричної ідентифікації інтервальних моделей систем.

4. Розробити агентно-орієнтовану архітектуру програмної системи для ідентифікації інтервальних моделей.

5. Розробити програмне середовище для ідентифікації інтервальних моделей систем.

6. Провести апробацію розроблених методів та програмних засобів для задач побудови інтервальних моделей для екологічного моніторингу шкідливих викидів автотранспорту, побудови профілю користувача соціальних мереж, відновлення рухливості суглобів верхніх кінцівок під час арт-терапії.

Об’єкт дослідження. Процеси ідентифікації математичних моделей динамічних та статичних систем.

Предмет дослідження. Обчислювальні методи ідентифікації інтервальних систем програмними агентами алгоритму бджолиної колонії у середовищі NVIDIA CUDA.

Методи дослідження. У дисертаційній роботі використано комплекс методів, що поєднує аналітичні, алгоритмічні та програмно-інженерні підходи. Для формалізації задачі ідентифікації інтервальних моделей систем застосовано методи системного аналізу, теорії оптимізації та теорії інтервальних обчислень. Розроблення алгоритмічної частини ґрунтується на принципах метаевристичних методів колективного інтелекту, зокрема алгоритму бджолиної колонії, який забезпечує пошук глобального екстремуму у багатовимірних просторах параметрів.

Для реалізації паралельних обчислень використано технологію NVIDIA CUDA з урахуванням архітектурних особливостей графічних процесорів. При

проєктуванні програмних засобів застосовано методи об’єктно-орієнтованого та компонентного програмування, а також принципи модульної архітектури, що забезпечує гнучкість та розширюваність системи. Ефективність розроблених методів і програмних реалізацій оцінювалася за допомогою експериментального моделювання на тестових наборах даних та реальних прикладах технічних систем, з використанням критеріїв точності ідентифікації та швидкодії.

Наукова новизна отриманих результатів. В межах дисертаційної роботи отримано такі наукові результати:

***уперше**:*

розроблено обчислювальні методи структурної та параметричної ідентифікації інтервальних моделей систем, які на відміну від існуючих ґрунтуються на застосуванні програмних агентів, що виконують функції поведінкової моделі бджолиної колонії паралельно, що забезпечило зниження часової складності ідентифікації інтервальних моделей систем;

запропоновано та обґрунтовано для реалізації обчислювальних методів структурної та параметричної ідентифікації інтервальних моделей систем технологію динамічної компіляції CUDA-ядра, яка забезпечує функціонування програмних агентів поведінкової моделі бджолиної колонії в середовищі NVIDIA CUDA, що у сукупності забезпечило зниження часової складності ідентифікації інтервальних моделей систем;

***набули подальшого розвитку**:*

агентно-орієнтована архітектура програмної системи для ідентифікації інтервальних моделей систем, що поєднує об’єктно-орієнтовану структуру компонентів із модульною організацією обчислювальних ядер CUDA та забезпечує гнучкість, масштабованість, використання GPU-ресурсів, що у сукупності при реалізації обчислювальних процедур ідентифікації інтервальних моделей систем знижує їх часову складність;

комп’ютерне середовище для математичного моделювання систем на основі аналізу інтервальних даних, яке на відміну від існуючих, імплементує інтерпретатор базисних функцій для побудови моделей та об’єднує паралельне

функціонування програмних агентів і їх динамічну компіляцію, що у сукупності спрощує доступ користувача до модулів інтервального моделювання та знижує часову складність побудови моделей.

Особистий внесок здобувача. Результати, викладені в цій дисертаційній роботі, отримані автором самостійно. У друкованих працях, які опубліковані у співавторстві, автору належать такі результати:

- побудовано математичну модель профіля користувача соціальної мережі за допомогою алгоритму бджолиної колонії;
- спроєктовано мультиагентну архітектуру програмного забезпечення ідентифікації інтервальних моделей систем на основі агентів алгоритму бджолиної колонії;
- побудовано математичні моделі шкідливих викидів автотранспорту, досліджено часову складність послідовної та паралельно реалізації методів параметричної ідентифікації;
- реалізовано метод структурної ідентифікації з використанням розробленого раніше CUDA рішення параметричної ідентифікації;
- побудовано математичну модель процесів генерації електроенергії;
- реалізовано паралельно обчислювальну схему ідентифікації для паралельних CPU потоків та для CUDA технології;
- побудовано математичну модель розміщення рою БПЛА;
- запропоновано паралельно обчислювальну схему ідентифікації на основі алгоритму бджолиної колонії.

Основні положення та результати дисертаційної роботи в наведених працях викладені в повному обсязі.

Апробація результатів дисертації. Основні положення і результати дисертаційної роботи презентовано на 3 конференціях, зокрема:

- 13th International Conference on Advanced Computer Information Technologies, ACIT-2023, Wroclaw, Poland, 21-23 Sept., 2023;
- 14th International Conference on Advanced Computer Information Technologies, ACIT-2024, Ceske Budejovice, Czech Republic, 19-21 Sept., 2024;

- 15th International Conference on Advanced Computer Information Technologies, ACIT-2025, Šibenik, Croatia, 17-19 Sept., 2025.

Публікації. За результатами дисертаційного дослідження опубліковано 8 наукових праць (ДОДАТОК Б) загальним обсягом сторінки, зокрема 3 статті у періодичних фахових виданнях [1, 2, 3], 2 з яких статті у періодичних наукових виданнях України категорії “Б” [2, 3] та 1 у виданнях, що індексуються наукометричною базою Scopus [1], а також 5 публікацій у матеріалах конференцій [4-8], 5 з яких індексується міжнародною наукометричною базою Scopus [4-8].

Структура та обсяг роботи. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел із 125 найменувань та 3 додатків. Загальний обсяг роботи складає 218 сторінки друкарського тексту, з них 123 сторінок основного тексту. Робота містить 28 рисунків і 15 таблиць.

Зв’язок роботи з науковими програмами, планами, темами.

Дисертаційна робота виконувалася в межах пріоритетного напряму розвитку науки і техніки «Інформаційні та комунікаційні технології», визначеного Законом України «Про пріоритетні напрями розвитку науки і техніки» (№2623-III від 11.07.2001, редакція від 13.01.2024), а також згідно з планом науково-дослідних робіт кафедри комп’ютерних наук Західноукраїнського національного університету протягом 2022 – 2025 років. Основні результати дисертаційного дослідження отримано в межах виконання таких тем:

- держбюджетного прикладного дослідження на тему «Математичне та програмне забезпечення прототипу біогазової установки з підвищеною ефективністю функціонування»(державний реєстраційний номер 0124U000076);
- держбюджетного прикладного дослідження на тему «Методи та програмні засоби для ідентифікації інтервальних моделей складних систем» (державний реєстраційний номер 0122U000627).

Усі вищезгадані роботи виконувалися за безпосередньої участі автора.

Практичне значення отриманих результатів полягає у створенні програмних засобів ідентифікації інтервальних моделей систем програмними агентами бджолиної колонії у середовищі NVIDIA CUDA. Розроблені методи та

програмні засоби забезпечують зменшення часової складності обчислень при збереженні високої точності ідентифікації параметрів моделей, що дає змогу ефективно застосовувати його у задачах аналізу, моделювання та оптимізації складних технічних, технологічних та екологічних процесів.

Запропоновані методи мультиагентної реалізації алгоритму бджолиної колонії забезпечують масштабованість і продуктивність обчислень, що особливо важливо для систем із великою кількістю параметрів або складними нелінійними залежностями. Використання розробленої архітектури програмних модулів CUDA дозволяє використовувати алгоритм до прикладних задач різної розмірності і складності, що розширює можливості практичного застосування результатів у промислових, наукових і освітніх середовищах.

Теоретичні та прикладні результати дисертаційної роботи використано:

- держбюджетного прикладного дослідження на тему «Математичне та програмне забезпечення прототипу біогазової установки з підвищеною ефективністю функціонування»(державний реєстраційний номер 0124U000076);
- держбюджетного прикладного дослідження на тему «Методи та програмні засоби для ідентифікації інтервальних моделей складних систем» (державний реєстраційний номер 0122U000627);
- в навчальному процесі Західноукраїнського національного університету на кафедрі комп’ютерних наук при викладанні дисциплін «Об’єктно-орієнтоване програмування», «Архітектура та проектування ПЗ», «Паралельні та розподілені обчислення».

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ІДЕНТИФІКАЦІЇ ІНТЕРВАЛЬНИХ МОДЕЛЕЙ СИСТЕМ

У цьому розділі проведено огляд літературних джерел, пов’язаних із проблематикою ідентифікацією інтервальних моделей систем.

В першому підрозділі розглянуто особливості побудови інтервальних моделей систем. Досліджено, що інтервальні моделі систем забезпечують явне й математично коректне представлення невизначеності у параметрах і вхідних впливах, надають можливість оцінювати стійкість і працездатність системи щодо збурень і похибок.

В другому та третьому підрозділах описані методи параметричної та структурної ідентифікації інтервальних моделей систем. В четвертому підрозділі описані методи колективного інтелекту в задачах ідентифікації моделей систем. Показано, що для дискретних моделей, в яких кожен параметр описується інтервалом допустимих значень виявлено обмеження існуючих методів ідентифікації. Показано, що такого типу задачі мають NP-складність. Досліджено, що метаевристичні підходи на основі колективного інтелекту здатні здійснювати глобальний пошук у великорозмірних просторах параметрів, дотримуючись інтервальних обмежень і формуючи множину прийнятних рішень. Обґрунтовано вибір алгоритму бджолиної колонії для ідентифікації інтервальних моделей.

У заключній частині розділу описано прикладні задачі, що будуть використовуватися для тестування розроблених методів ідентифікації інтервальних моделей систем, та сформульовано науково-технічне завдання, мету та завдання дисертаційного дослідження і наведено висновки на підставі проведеного огляду літературних джерел.

Результати цього розділу опубліковано автором у працях [46, 47].

1.1. Особливості побудови інтервальних моделей систем

Ідентифікація систем розглядається як процес побудови математичної моделі об’єкта на основі результатів експериментальних спостережень за його вхідними та вихідними параметрами [98]. Задача ідентифікації полягає у знаходженні такої моделі, яка адекватно відтворює поведінку реальної системи за критеріями точності, стійкості та узагальнювальної здатності [83].

Під математичною моделлю у цьому контексті розуміють формалізований опис зв’язку між вхідними параметрами та вихідною реакцією. У цьому випадку формою представлення математичної моделі слугують дискретні моделі, тобто різницеві оператори, а їх налаштування проводять на основі експериментальних даних [23, 36, 71, 94]:

$$\upsilon_k = F(\upsilon_{k-d,\dots,}\upsilon_{k-1)} * \bar{g}, k = d, \dots, K \qquad (1.1)$$

де *F(*)* – деяке нелінійне перетворення, υ_k – модельована характеристика об’єкта в момент часу *k*, *d* – порядок різницевого рівняння, $\bar{g}$ – невідомий вектор параметрів різницевого рівняння.

У широкому розумінні ідентифікація охоплює як побудову структури моделі – вибір типу рівнянь, порядку динаміки, переліку змінних, так і оцінювання її числових параметрів [64, 65]. У сучасній інженерії програмного забезпечення задача ідентифікації є ключовою для аналізу, керування та прогнозування поведінки технічних, медичних, екологічних та енергетичних систем, де неможливо покладатися лише на аналітично отримані фізичні моделі [39 - 41, 53, 92, 100].

Стандартний цикл ідентифікації включає кілька основних етапів. По-перше, формується мета моделювання – діагностика стану системи, прогноз її поведінки і вибираються змінні, які будуть розглядатися як вхідні та вихідні. По-друге, проводиться експеримент або збір даних, що забезпечують збудження динаміки системи в репрезентативних режимах роботи [121]. По-третє, визначається

загальний вигляд моделі, тобто її структура. По-четверте, виконується оцінювання параметрів за вибраним критерієм оптимальності [79], найчастіше мінімізацією функції невідповідності між моделлю та експериментальними даними[99, 111]. Нарешті, проводиться валідація моделі: перевіряється її здатність відтворювати поведінку системи на даних, що не використовувалися під час оцінювання параметрів.

Зазначені етапи не є одноразовою лінійною послідовністю: у практиці вони виконуються ітераційно – невдала валідація моделі вимагає або зміни структури моделі, або повторної оцінки параметрів, або навіть корекції експерименту.

У найзагальнішому вигляді розрізняють два взаємопов'язані різновиди ідентифікації: структурну та параметричну [42, 43]. Структурна ідентифікація спрямована на вибір виду моделі, тобто визначення форми функції *F(*)*, порядку системи, релевантних вхідних змінних, наявності або відсутності зворотних зв'язків, типу нелінійностей тощо. Фактично вона відповідає на запитання «яка модель є доречною для опису цього об'єкта?». Параметрична ідентифікація [60], у свою чергу, передбачає знаходження числових значень вектора параметрів $\bar{g}$ для вже обраної структури моделі, тобто відповідає на запитання «які саме значення параметрів роблять модель максимально узгодженою з експериментальними даними?».

Таке розділення є важливим, оскільки невдала структурна ідентифікація призводить до систематичної моделювальної похибки незалежно від якості оцінювання параметрів, тоді як неточність параметричної ідентифікації, за умови коректно обраної структури, зазвичай може бути зменшена поліпшенням методу оцінювання чи якості експериментальних даних.

Критичним поняттям у контексті ідентифікації є адекватність моделі. Адекватність моделі означає ступінь відповідності побудованої математичної моделі реальній системі в умовах, для яких ця модель призначена. Модель вважається адекватною, якщо вона здатна відтворювати спостережувану динамічну поведінку об'єкта в репрезентативних робочих режимах, відображає суттєві причинно-наслідкові зв'язки між вхідними та вихідними змінними, а також

зберігає прогнозну здатність для нових вхідних впливів, що не були використані під час налаштування параметрів. Важливо, що адекватність не зводиться лише до малої середньоквадратичної похибки на навчальних даних: модель повинна бути стійкою з погляду фізичного змісту параметрів, не містити штучних залежностей, зумовлених шумом вимірювань, і не демонструвати деградації якості при використанні в інших режимах роботи системи. Таким чином, критерій адекватності поєднує узгодженість із даними, інтерпретованість та узагальнювальну здатність, і є обов'язковим етапом валідації будь-якої ідентифікованої моделі.

На цьому тлі важливим є питання невизначеності [88]. Реальні системи працюють в умовах шуму сенсорів, неточності калібрування, змін зовнішнього середовища, суб'єктивних або неповних даних (наприклад, у медичних або екологічних вимірюваннях). Тому опис об'єкта у вигляді моделі з фіксованими «точковими» параметрами часто є занадто оптимістичним. Одним зі способів більш реалістичного опису є використання інтервальних параметрів, коли кожен компонент задається не одним числом, а інтервалом припустимих значень, що відображає наявну невизначеність [63, 89].

Це дозволяє будувати не єдину модель, а сімейство моделей, узгоджених з експериментальними даними, і далі аналізувати стійкість рішень керування або прогнозів в межах цього сімейства [53, 92]. Такий підхід особливо корисний у випадках, коли критично важлива надійність, наприклад екологічні моделі оцінювання шкідливих викидів автотранспорту, медичні дані з природною варіабельністю пацієнтів.

Таким чином, ідентифікація складних систем у загальному випадку є ітеративним процесом побудови та уточнення моделі об'єкта за даними спостережень, що включає вибір структури моделі, оцінювання її параметрів та перевірку адекватності, причому окрему роль відіграє явне врахування невизначеностей у даних і параметрах.

У реальних технічних, медичних, екологічних та енерготехнологічних системах вимірювані параметри майже ніколи не відомі точно [20]: вони

зашумлені, підлягають впливу зовнішніх факторів, змінюються в часі або фіксуються з обмеженою роздільною здатністю сенсорів. Традиційний підхід до моделювання передбачає використання точкових значень параметрів, що у подальшому вводяться в модель як сталі коефіцієнти. Такий опис є зручним для математичної обробки, проте він не відображає реальну невизначеність і, відповідно, створює ризик хибних висновків щодо поведінки системи, її стійкості або допустимих режимів роботи. Інтервальний підхід усуває це обмеження [42, 43, 63, 89, 92], оскільки кожний параметр розглядається не як число, а як інтервал припустимих значень. Формально інтервал визначається як z = $[z^-, z^+]$, де z^- та z^+ є нижньою та верхньою границями невизначеності. Якщо параметр моделі υ_i неможливо оцінити однозначно через похибку вимірювання, замість υ_i використовуємо інтервальне значення $[\upsilon_i^-, \upsilon_i^+]$. Такий запис дозволяє явно зберегти інформацію про похибку і робить модель не точковою, а гарантовано-обмеженою множиною можливих поведінок.

Інтервальна арифметика визначає правила виконання алгебраїчних операцій над інтервалами таким чином, щоб результат включав усі можливі значення, які могли б виникнути при використанні будь-яких допустимих значень усередині інтервалів [18]. Наприклад, для двох інтервалів $x = [x^-, x^+]$ і $y = [y^-, y^+]$ сума задається як $x + y = [x^- + y^-, x^+ + y^+]$, а добуток, через залежність знаків, визначається як $x * y = [min(x^- * y^-, x^- * y^+, x^+ * y^+, x^+ * y^-), max(x^- * y^-, x^- * y^+, x^+ * y^+, x^+ * y^-)]$. Таке розширення звичайної арифметики дає змогу проводити обчислювальні процедури не з однією «точкою», а з цілою множиною можливих значень параметрів і вхідних сигналів, гарантуючи, що всі можливі результати лежать у знайденому інтервалі. Це є фундаментально важливим для формування гарантованих оцінок стану системи та її параметрів у присутності похибок вимірювання, шуму або неповноти даних.

На основі такого підходу будується інтервальна модель системи. Однією з ключових властивостей інтервальної моделі є здатність оцінювати стійкість системи до збурень. Якщо вхідні впливи системи, наприклад температура навколишнього середовища, концентрація реагентів, навантаження на механічний

вузол або інтенсивність трафіку, можуть змінюватися в певних діапазонах, ці діапазони можуть бути безпосередньо внесені в модель як інтервали. Далі, за допомогою інтервальної арифметики [18, 89], можна обчислити, як ці варіації впливають на можливий діапазон вихідних станів системи. У результаті отримується не просто оцінка система стабільна чи нестабільна, а межі, в яких система залишається працездатною. Це важливо для задач безпеки, прогнозування деградації або планування керуючих дій у режимах невизначеності. Іншими словами, інтервальна модель природно дає засіб оцінки чутливості системи до невизначеностей без необхідності перебору великої кількості сценаріїв.

Застосування інтервального представлення параметрів є критично важливим у низці прикладних сфер, які будуть використані надалі в роботі для експериментальної перевірки:

- Екологічне моделювання шкідливих викидів автотранспорту. Вимірювані параметри (склад вихлопних газів, навантаження двигуна, режим руху) мають суттєві похибки та змінюються залежно від умов експлуатації. Інтервальна модель дозволяє оцінювати не тільки номінальні значення викидів, а й верхні межі, що важливо для контролю ризиків перевищення екологічних нормативів [39, 43, 51].
- Біомедичні моделі в арт-терапії верхніх кінцівок. Пацієнти відрізняються за фізіологічними характеристиками, рівнем ураження, больовою реакцією, швидкістю відновлення рухливості. Дані про амплітуди рухів, м’язову активність, кутові швидкості мають індивідуальні варіації й часто неповні. Інтервальна модель дозволяє будувати не єдину середню траєкторію реабілітації, а діапазон очікуваного відновлення, що важливо для персоналізованих рекомендацій [59, 74, 97, 110].
- Виявлення фейкових новин у соціальних мережах. У задачах класифікації достовірності контенту окремі ознаки (тональність, інтенсивність поширення, часовий профіль появи повідомлень) оцінюються зі значною невизначеністю, зокрема через неповні або маніпульовані дані. Інтервальні ознаки

у моделі дозволяють врахувати сумнівність інформаційних джерел і працювати з неповною достовірністю фактів [10, 11, 16, 109, 117].

У всіх наведених випадках моделювання за фіксованими параметрами може ввести в оману, тоді як інтервальна модель явним чином фіксує поле невизначеності [18]. Для задач ідентифікації це означає, що ми шукаємо не одну найкращу точку υ, а множину допустимих параметрів υ, які узгоджуються з емпіричними даними в межах їх похибок. Завдання ідентифікації в інтервальному формулюванні можна подати як пошук такої множини параметрів υ, що для всіх експериментальних точок виконується умова узгодженості $\upsilon_i \in [z^-, z^+]$.

Це надає сильну гарантійну інтерпретацію результатів: якщо параметр потрапляє в знайдений інтервал, то він не суперечить наявним спостереженням. Такий формалізм лежить в основі гарантованої оцінки параметрів і є одним із ключових об’єктів дослідження у задачах ідентифікації інтервальних моделей.

Підсумовуючи, інтервальні моделі систем забезпечують: явне й математично коректне представлення невизначеності у параметрах і вхідних впливах; можливість оцінювати стійкість і працездатність системи щодо збурень і похибок; кращу інтерпретованість результатів у критично важливих застосуваннях, де важливі не тільки середні, але й граничні сценарії.

1.2. Методи параметричної ідентифікації інтервальних моделей

Традиційна постановка параметричної ідентифікації полягає у відшукання такого набору параметрів Θ, який мінімізує різницю між виходом моделі y(t,θ) та експериментальними спостереженнями yexp(t). Найпоширенішим критерієм є мінімізація середньоквадратичної похибки:

$$J(\Theta) = \sum_{t=1}^{N} \| y^{exp}(t) - y(t, \Theta) \|^2 \qquad (1.2)$$

що безпосередньо приводить до методів найменших квадратів (МНК) та їх варіацій [83].

У випадку лінійних за параметрами моделей ці методи мають закриту або квазізакриту форму й забезпечують швидку збіжність. Для слабо нелінійних систем застосовують градієнтні та квазідругі методи (метод найшвидшого спуску, метод Ньютона, метод Левенберга–Марквардта), що ґрунтуються на локальній апроксимації цільової функції, її градієнтів та матриць чутливості моделі до параметрів [19, 91, 111]. Такі методи ефективні для помірної кількості параметрів і за умови відносно гладкої поведінки функції $J(\Theta)$.

Однак для інтервальних дискретних моделей постають дві фундаментальні проблеми. Перша – висока розмірність простору параметрів**.** Кількість шуканих параметрів прямо залежить від кількості просторових вузлів, сегментів дискретизації. Це означає, що замість десятків параметрів ми можемо мати сотні або тисячі. За таких умов стандартні градієнтні методи часто потрапляють у локальні мінімуми або вимагають значних обчислювальних витрат для обчислення похідних і матриць чутливості. До того ж збільшується ризик надмірної підгонки параметрів під шум у даних. Друга проблема – інтервальна невизначеність даних і параметрів [48]. Класичний МНК припускає, що похибка є випадковою і добре описується статистично (наприклад, гаусовим шумом з нульовим середнім). В інтервальних моделях ми маємо не стохастичну, а гарантовано обмежену невизначеність: замість приблизно 1.2 ми маємо у межах [1.1; 1.4]. Це означає, що критерій ідентифікації має перевіряти узгодженість моделі не з однією точкою, а з цілим інтервалом допустимих спостережень. Методи інтервального аналізу похибок розглядають задачу оцінювання параметрів як пошук множини Θ, для якої модель у принципі може відтворити всі експериментальні дані в межах допустимих похибок вимірювання. Така постановка є гарантованою і більш консервативною, але вона призводить до складних, часто негладких областей допустимих рішень у просторі параметрів.

Через ці причини точні або квазіточні методи на основі градієнтів стають менш придатними: вони потребують гарної ініціалізації, не забезпечують глобальної оптимальності і погано працюють у задачах, де функція має велику

кількість локальних екстремумів, а допустимі значення параметрів описані інтервалами, а не точками.

Альтернативою класичним методам є евристичні та метаевристичні алгоритми глобальної оптимізації. Вони розглядають задачу ідентифікації як пошук мінімуму (або множини квазіоптимальних мінімумів) цільової функції без вимоги до її гладкості, опуклості чи диференційованості. Ключовими представниками цього класу є генетичні алгоритми, рій частинок та алгоритм бджолиної колонії.

Генетичні алгоритми моделюють еволюційний процес добору, схрещування і мутації популяції кандидатних рішень [21, 30, 31, 57]. Кожен можливий вектор параметрів Θ інтерпретується як «особина», якій призначається пристосованість (fitness), що відображає якість збігу моделі з даними. Далі відбираються кращі особини, комбінуються між собою та випадково змінюються з метою дослідження простору параметрів. Перевага генетичних алгоритмів полягає у здатності досліджувати великі, сильно нелінійні, неопуклі простори без потреби в похідних. Вони добре працюють у задачах високої розмірності [25], характерних для дискретних моделей із розподіленими параметрами, і можуть природно враховувати обмеження параметрів у вигляді інтервалів.

Алгоритм рою частинок базується на колективній поведінці частинок, які «рухаються» простором параметрів, орієнтуючись як на власний досвід пошуку кращих значень, так і на досвід сусідів або всієї популяції [37, 69, 107, 114]. У контексті ідентифікації це означає, що багато кандидатних векторів параметрів паралельно перевіряються щодо якості відтворення експериментальних даних, і кожен кандидат адаптивно зміщує свої значення в напрямку більш успішних сусідів. Алгоритм рою частинок не потребує диференційовності цільової функції і добре масштабується у просторах великої розмірності, оскільки оновлення положення частинок є відносно простим обчислювально.

Алгоритм бджолиної колонії моделює колективний пошук джерел «нектару», де кожна бджола відповідає за дослідження певної області простору параметрів та обмінюється інформацією з іншими бджолами про знайдені

«перспективні» області [40 – 43, 45, 67, 68]. Розподіл ролей (розвідниці, спостерігачі, робочі бджоли) забезпечує баланс між локальним уточненням рішень у вже перспективних ділянках і глобальним дослідженням нових областей простору параметрів. Це особливо важливо при ідентифікації дискретних моделей із розподіленими параметрами, де глобальний ландшафт цільової функції має безліч локальних мінімумів. Алгоритм бджолиної колонії добре адаптується до обмежень що кожен параметр має бути в інтервалі [73, 76, 78, 106], тому природно підходить до інтервальних моделей. Крім того, кожен кандидат-рішення може інтерпретуватися як агент, що робить цей алгоритм зручним для подальшої агентно-орієнтованої програмної реалізації.

Усі перелічені метаевристики мають спільну властивість: вони здійснюють глобальний пошук без припущення про лінійність або гладкість моделі. На відміну від класичних методів (МНК, градієнтних підходів), які в основному є локальними оптимізаторами і сильно залежать від початкового наближення, ройові та еволюційні алгоритми можуть віднайти множину прийнятних рішень, що відповідає природі інтервальної ідентифікації. Це критично важливо для дискретних моделей з розподіленими параметрами, оскільки метою часто є не стільки одна найкраща векторна оцінка, скільки множина узгоджених з даними параметрів, яка характеризує допустимі фізично правдоподібні стани системи.

Для дискретних моделей із розподіленими параметрами, в яких кожен параметр описується інтервалом допустимих значень і кількість цих параметрів є великою, класичні детерміновані методи (МНК, градієнтні методи) стикаються з двома обмеженнями: вони погано працюють у велико розмірних просторах рішень; вони не дають природного механізму для побудови множини допустимих параметрів при інтервальній невизначеності. На відміну від цього, метаевристичні підходи на основі колективного інтелекту – зокрема алгоритм бджолиної колонії, який є базовим у даній дисертації – здатні здійснювати глобальний пошук у великорозмірних просторах параметрів, дотримуючись інтервальних обмежень і формуючи множину прийнятних рішень. Саме тому ці алгоритми обрано як основу

для подальшої реалізації ідентифікації інтервальних моделей систем у програмному комплексі дисертаційної роботи.

1.3. Методи структурної ідентифікації інтервальних моделей

Структурна ідентифікація розглядається як вибір класу та форми моделі системи, тобто визначення того, які змінні мають бути включені до моделі, яким є характер залежностей між ними, який порядок та тип динаміки слід враховувати, а також у якій формі ці залежності описуються (лінійна, нелінійна, стохастична, нечітка тощо). Для дискретних моделей із розподіленими параметрами структурна ідентифікація має особливу складність: ми повинні не лише вирішити, які змінні та зв’язки є релевантними, але й визначити просторову структуру моделі – наприклад, які взаємодії між локальними сегментами системи є істотними, і чи потрібно включати просторові зв’язки сусідніх вузлів [42, 83, 111].

Найпростіший і найстаріший підхід до структурної ідентифікації – повний або частковий перебір можливих структур із подальшим порівнянням їхньої якості. Типовий приклад: для динамічних дискретних моделей типу ARX/ARMAX послідовно перебирають можливі порядки затримок, ступені авторегресійної частини та кількість регресорів, після чого обирають ту структуру, що мінімізує певний критерій, наприклад інформаційний критерій Акаіке або байєсівський інформаційний критерій [24]. Ці критерії балансують точність апроксимації та складність моделі, караючи перенасичені структури. У системах з малою кількістю потенційних регресорів та не дуже високим порядком динаміки такий підхід може бути прийнятним. Проте для просторово-дискретизованих систем кількість потенційних зв’язків між компонентами стану росте комбінаційно: навіть локальні взаємодії між сусідніми вузлами призводять до великої кількості кандидатних структур. Це робить повний перебір практично непридатним.

Другий класичний напрям – покроковий відбір регресорів (stepwise selection). Його суть полягає у поступовому формуванні структури моделі: спочатку модель будується з мінімального набору зв’язків, далі послідовно додаються ті змінні чи

зв’язки, які найбільше покращують критерій якості (forward selection), або навпаки – починають із повної моделі та видаляють найменш значущі зв’язки (backward elimination). Цей процес зупиняється, коли подальше ускладнення структури не дає статистично значущого покращення. Такі підходи застосовуються у лінійній і квазілінійній регресії, у субпросторових методах ідентифікації стан-простір моделей (N4SID) [119], а також при побудові спрощених динамічних рівнянь для керованих технічних процесів. Обмеження полягає в тому, що покрокові методи часто зупиняються на локально раціональній структурі, яка не є глобально оптимальною, і вони мають тенденцію виділяти структури з хорошою апроксимацією середніх режимів, але не обов’язково стійкі в межових (критичних) режимах.

Унаслідок комбінаційного вибуху кількості можливих структур для розподілених дискретних моделей класичні методи відбору стають обчислювально непридатними. Тому активно застосовують евристичні та метаевристичні підходи, які розглядають структуру як об’єкт оптимізації.

Генетичні алгоритми можуть кодувати саму структуру моделі у вигляді хромосоми [58]. Наприклад, бінарний вектор може позначати, які регресори (зв’язки) дозволені, а які відкинуті; або ж цілочисловий вектор може кодувати порядок затримок, кількість сусідів, топологію зв’язків тощо. Оператор схрещування поєднує структурні фрагменти різних батьківських моделей, тоді як мутації додають або видаляють зв’язки. Якість кожної структури оцінюється за критеріями узгодженості з даними, стабільності, інтерпретованості й складності (наприклад, штраф за занадто велику кількість зв’язків). Таким чином ГА виконують глобальний пошук по простору структур, не вимагаючи повного перебору.

Алгоритм рою частинок також може бути застосований до структурної ідентифікації, якщо визначити спосіб відображення структури моделі у вигляді вектора оптимізації. Наприклад, кожне можливе ребро просторового графа взаємодій може мати елемент «вага/значущість», і рій частинок поступово підлаштовує ці ваги, підтягаючи їх до структур, які краще відтворюють

експериментальні дані й лишаються стійкими. У випадку інтервальних структурних моделей рій може шукати не тільки «наявність зв’язку», а й діапазон його допустимого впливу. Це особливо важливо для задач, де структуру неможливо виписати аналітично (наприклад, у складних біомедичних або інформаційних мережах).

Алгоритм бджолиної колонії може використовуватися не лише для підбору числових параметрів моделі, а й для відбору структури. Тут кожна «бджола» відповідає за дослідження певної кандидатної структури моделі , а «якість нектару» відображає якість опису даних цією структурою з урахуванням інтервальних обмежень. Ролі розвідниць і спостерігачів у цьому контексті дають природний механізм балансування між:

- локальною експлуатацією вже перспективних структур (уточнення/очищення моделі),
- глобальним дослідженням нових структур.

Цей підхід добре масштабується у випадку дуже великої розмірності простору структур (наприклад, коли кожен просторовий вузол потенційно може залежати від кількох сусідів у різні моменти часу), і дозволяє формувати не одну «найкращу» структуру, а множину прийнятних структур разом із відповідними інтервалами впливів.

Структурна ідентифікація в інтервальному підході має не лише теоретичну, а й прикладну вагу. У задачі оцінки шкідливих викидів автотранспорту ми повинні вирішити, які фактори (швидкість руху, навантаження двигуна, тип палива, локальні метеоумови, топологія вуличного каньйону) реально впливають на концентрації забруднювачів у певній просторовій точці. Структура моделі визначає, які взаємодії між цими факторами вважати істотними, і які з них зберігаються у вигляді інтервальних коефіцієнтів.

Таким чином, структурна ідентифікація для інтервальних дискретних моделей із розподіленими параметрами – це процес встановлення мінімально необхідної, але інтерпретованої та фізично обґрунтованої топології залежностей, разом із діапазонами їхньої сили впливу. Класичні методи (перебір структур,

покроковий відбір, інформаційні критерії) залишаються корисними як базова оцінка, але вони стають обмеженими у високій розмірності та при інтервальній невизначеності. Евристичні та метаевристичні підходи, зокрема алгоритм бджолиної колонії, дозволяють проводити глобальний пошук структури, відбирати лише суттєві зв'язки й одночасно зберігати інтервальні характеристики взаємодій між змінними, що є критично важливим для подальшої ідентифікації й моделювання складних реальних систем.

1.4. Методи колективного інтелекту в задачах ідентифікації

Колективний інтелект розглядається як клас підходів штучного інтелекту, у яких глобально узгоджена поведінка системи виникає в результаті взаємодії великої кількості відносно простих агентів. На відміну від централізованих оптимізаційних схем, у яких існує єдиний «керуючий блок», swarm-підходи не потребують жорсткої координації: агенти діють локально, обмінюються інформацією про якість знайдених рішень, реагують на середовище та один на одного, і завдяки цьому вся популяція поступово наближається до кращих областей простору рішень. Ключові властивості таких систем – децентралізація, самоорганізація, стійкість до втрати окремих агентів і природний баланс між дослідженням нових областей простору параметрів та уточненням уже знайдених перспективних областей. Ці принципи були сформульовані для різних біоінспірованих алгоритмів, включаючи алгоритм рою частинок, мурашині алгоритми та алгоритм бджолиної колонії, і показали ефективність у задачах глобальної оптимізації високої розмірності.

Важливою рисою підходів колективного інтелекту є те, що вони не вимагають гладкості чи опуклості цільової функції, не потребують обчислення градієнтів, легко працюють із обмеженнями на параметри у вигляді інтервалів і природно допускають паралельну перевірку великої кількості кандидатних рішень. Це робить їх особливо привабливими для ідентифікації дискретних моделей із розподіленими параметрами під інтервальною невизначеністю. Там простір

параметрів є дуже великорозмірним, сильно нелінійним і має багато локальних мінімумів, а задача не полягає у знаходженні єдиного ідеального вектора параметрів, а у визначенні множини припустимих параметрів, які не суперечать експериментальним даним, з урахуванням інтервалів похибок.

Генетичні алгоритми еволюційного типу моделюють механізм природного відбору: множина кандидатних рішень (популяція) підлягає відбору, схрещуванню (кросоверу) й мутації, і таким чином формується нове покоління рішень, зазвичай зі зростаючою якістю. Генетичні алгоритми добре працюють з дискретними й комбінаторними структурами, а також з обмеженнями типу «наявний/відсутній зв’язок», тому активно застосовуються для структурної ідентифікації (відбір релевантних регресорів, топологій зв’язків, лагів тощо). У контексті параметричної ідентифікації вони дозволяють вільно досліджувати великий простір параметрів без вимоги до гладкості функції якості. Проте ГА часто вимагають ретельного налаштування операторів схрещування/мутації й можуть бути відносно «важкими» з точки зору збереження різноманіття популяції при великій кількості параметрів.

Алгоритм рою частинок моделює рух «частинок» у просторі параметрів. Кожна частинка має власну найкращу знайдену позицію та «знає» найкращу позицію, яку знайшли інші (або найближчі сусіди, або вся популяція). Оновлення положення частинки виконується як поєднання інерції, руху до власного найкращого рішення і руху до кращих рішень інших. Завдяки цьому алгоритм рою частинок дуже швидко концентрується у перспективних областях параметрів. Переваги Алгоритм рою частинок: простота реалізації, невисокі вимоги до кількості гіперпараметрів, добра поведінка у помірно великій розмірності, відсутність необхідності в градієнті. Недолік – схильність до передчасної конвергенції в локальний мінімум, якщо не вжити механізмів підтримання різноманіття. У задачах ідентифікації дискретних розподілених моделей із інтервальними обмеженнями алгоритм рою частинок застосовується як глобальний пошук початкових оцінок параметрів, які потім можуть бути локально доуточнені класичними методами (градієнтними чи Левенберга–Марквардта).

Мурашині алгоритми базуються на спостереженні за тим, як колонії мурах знаходять шляхи до джерел їжі за допомогою феромонних слідів [38]. У моделі оптимізації кожен маршрут/структура/комбінаторне рішення кодується як послідовність виборів, які мураха робить на графі; після побудови рішень мурахи підсилюють феромон на кращих шляхах, і наступні покоління з більшою ймовірністю відтворюють ці комбінації. Мурашині алгоритми особливо добре працюють для дискретних комбінаційних задач (шляхи, топології, відбір структурних зв'язків тощо) і масштабно використовується у задачах структурної ідентифікації – зокрема для вибору топології моделі, лагів або кандидатних взаємодій між змінними. Однак класичний мурашиний алгоритми менш природно пристосований до безперервних параметрів, і його типова сильна сторона – саме комбінаторна частина, а не безпосередньо точна числова параметризація моделей.

У алгоритмі бджолиної колонії популяція агентів розділяється на ролі (наприклад, робочі бджоли, розвідниці, спостерігачі), які відповідають за різні аспекти пошуку. Робочі бджоли вдосконалюють уже знайдені перспективні розв'язки, розвідниці досліджують нові області простору параметрів, а спостерігачі приймають рішення про те, які області варто посилено досліджувати далі, на основі «якісності нектара», тобто значення функції мети. Цей поділ обов'язків забезпечує природний механізм балансу між уточненням і різноманіттям без явної потреби в складній параметризації динаміки руху. Також АБК дуже зручно працює в умовах інтервальних обмежень: кожна кандидатна точка (набір параметрів моделі) може бути одразу обмежена інтервальними рамками, і «бджоли» просто не досліджують заборонені області простору параметрів.

Крім того, в АБК кожна бджола може бути інтерпретована як окремий програмний агент, який незалежно оцінює якість певної конфігурації параметрів або навіть певної структури моделі. Ця властивість є важливою для нашої подальшої програмної реалізації у вигляді набору агентів, які можуть діяти паралельно. На відміну від деяких інших методів, де взаємодія між кандидатами є глобальною й вимагає централізованої координації (наприклад, обмін «глобально найкращим» рішенням у PSO), в АБК обмін інформацією більше нагадує вибір

напрямів дослідження колонією на основі якості знайденого «нектару», що дозволяє організувати обчислювальний процес у вигляді колективу напівнезалежних агентів. Це добре узгоджується з підходом, де такі агенти надалі відображаються у вигляді потоків/керованих обчислювальних елементів.

Є кілька причин, чому саме алгоритм бджолиної колонії обрано як основний обчислювальний механізм ідентифікації інтервальних моделей у цій роботі:

1. Природна підтримка множини рішень замість одного точкового оцінювання. У задачі інтервальної ідентифікації ми не шукаємо єдину оцінку параметрів, а будуємо множину узгоджених параметрів. АБК за своєю суттю підтримує множину перспективних рішень (множину «джерел нектара»), кожне з яких може зберігатися й вдосконалюватися паралельно, без негайного витіснення альтернативних варіантів. Це добре узгоджується з set-membership підходом до інтервальної ідентифікації.

2. Гнучкість роботи з обмеженнями та інтервалами. Для дискретних моделей із розподіленими параметрами кожен компонент параметра має інтервальні межі , які випливають із вимірювань або фізичних міркувань (наприклад, коефіцієнти дифузії, швидкості реакції, межі енергетичних виходів тощо). В АБК ці обмеження враховуються безпосередньо на етапі генерації й оновлення кандидатів: не потрібно складних проєкцій або штрафних функцій великої ваги, як це іноді буває у PSO чи класичних градієнтних підходах.

3. Здатність працювати в дуже високій розмірності. У розподілених дискретних моделях (екологічні поля забруднювачів, поля фізіологічних показників) число параметрів може сягати сотень або тисяч. PSO і ГА при такій розмірності або швидко втрачають різноманіття, або «розмазуються» по простору й сходяться дуже повільно. АБК завдяки відокремленим стратегіям пошуку різних груп агентів дозволяє концентрувати ресурси на локально перспективних підпросторах, не втрачаючи здатності до глобального дослідження. Це критично важливо для зниження часу ідентифікації.

4. Модульність та агентно-орієнтована інтерпретація. Кожна бджола – це окремий агент зі своїм станом (поточний варіант параметрів, його якість, історія

локальних змін). Така агентна інтерпретація спрощує побудову програмної системи у вигляді набору взаємодіючих модулів. Вона ж полегшує наступний етап – відображення цієї множини агентів на паралельне обчислювальне середовище. У перспективі (розділи далі) це безпосередньо відповідає реалізації на графічних процесорах із використанням NVIDIA CUDA, де кожен агент може оброблятися окремим потоком.

5. Застосовність як до параметричної, так і до структурної ідентифікації. АБК можна використовувати не лише для знаходження числових значень параметрів моделі, а й для вибору самої структури (набору зв’язків, просторових взаємодій, лагів), якщо кодувати структуру як об’єкт, яким оперує бджола. Це дозволяє застосувати один і той самий оптимізаційний механізм до обох ключових підзадач дисертації – параметричної і структурної ідентифікації інтервальних моделей систем.

З огляду на ці властивості, алгоритм бджолиної колонії розглядається у цій дисертації як базовий механізм глобальної оптимізації для ідентифікації інтервальних моделей систем. Подальші розділи будуть присвячені формальній постановці задачі пошуку, проектуванню агентної взаємодії та програмній реалізації цього підходу у вигляді високопродуктивних обчислювальних засобів.

1.5. Аналіз прикладних задач для побудови інтервальних моделей

У межах дисертаційної роботи для експериментальної перевірки розроблених методів і програмних засобів були обрані наступні типи прикладних моделей: моделі шкідливих викидів автотранспорту, моделі арт-терапії верхніх кінцівок, моделі виявлення фейкових новин у соціальних мережах. Ці напрями не є випадковими або довільними: кожен із них є прикладом складної системи з високим рівнем невизначеності у вимірах та параметрах, а також є джерелом багатовимірних даних, що робить їх репрезентативними для задачі ідентифікації інтервальних моделей дискретних систем із розподіленими параметрами. Крім того, ці прикладні домени відрізняються фізичною природою (екологія, медицина,

інформаційна безпека, енергетика), що дозволяє продемонструвати універсальність методу, а не його пристосованість до однієї вузької задачі.

Моделі шкідливих викидів автотранспорту (екологічне прогнозування). Система «транспорт–довкілля» є суттєво невизначеною: рівень викидів залежить від типу двигуна, стану двигуна, палива, стилю водіння, швидкості потоку транспорту, метеорологічних умов, топографії вулиць тощо. Навіть у межах одного маршруту середні концентрації оксидів азоту, твердих частинок чи CO можуть змінюватися в рази залежно від локального режиму руху [35, 39, 43, 51, 93]. Дані для такої моделі є шумними (польові датчики, мобільні вимірювання, короткі усереднення, пропуски), а просторовий розподіл забруднення істотно неоднорідний. Це приводить до моделі з розподіленими параметрами, де кожна дискретна ділянка (відрізок дороги, зона вимірювання) має власні коефіцієнти емісії та розсіювання. Ці коефіцієнти не можуть бути оцінені як точні сталості – вони природно описуються інтервалами допустимих значень. Відповідно, задача ідентифікації полягає не лише в підборі параметрів, але й у визначенні просторової структури впливу факторів (транспортний потік, погодні умови) на локальну концентрацію домішок. Такий тип моделі дозволяє оцінювати не тільки номінальну якість повітря, а й гірший сценарій, що важливо для екологічних нормативів і ризик-орієнтованого планування.

Моделі арт-терапії верхніх кінцівок (медична реабілітація). У відновній медицині, зокрема при реабілітації рухових функцій руки після травм чи неврологічних уражень, об’єктом моделювання є індивідуальна динаміка пацієнта [26, 55, 75, 81]. Ця динаміка багатофакторна: амплітуда руху в суглобах, м’язова активність, рівень болю, втомлюваність, темп виконання вправ, психологічна залученість у терапію – усі ці ознаки є суб’єктивними, змінними у часі й вимірюються з високим рівнем шуму. Окрім того, пацієнти сильно відрізняються між собою, тобто в моделі не можна зафіксувати «усереднений параметр відновлення» і вважати його універсальним. Це якраз відповідає постановці інтервальної моделі: для кожного пацієнта (або навіть для кожної сесії терапії) замість одного «коефіцієнта прогресу» маємо інтервал можливих темпів

відновлення. Також така система зазвичай описується дискретно в часі (сесії, вправи, тижні терапії) і потенційно дискретно в просторі (різні суглоби, сегменти кінцівки, групи м’язів), тобто параметри є розподіленими. Ідентифікація тут потрібна не лише для прогнозу «як буде далі», але й для персоналізації терапії – вибору адекватного навантаження, діапазону рухів, інтенсивності повторень. Модель мусить бути стійкою щодо шумових і неповних медичних даних, тому інтервальний підхід і колективні евристики є природним вибором

Моделі виявлення фейкових новин у соціальних мережах (обробка інформаційних потоків). У задачі виявлення дезінформації, пропаганди або фейкових повідомлень маємо справу з інформаційною системою, яка еволюціонує в часі та просторі розповсюдження [62]. Тут параметри пов’язані не лише з текстовим вмістом (лексичні та стилістичні ознаки), але й із динамікою поширення (швидкість репостів, щільність ретрансляторів, часові патерни активності бот-мереж) [13 - 15, 118]. Ці параметри важко спостерігати безпосередньо, частина джерел є анонімною або навмисно маскує свою поведінку, а дані часто фрагментарні чи недостовірні. Тобто навіть факт «впливу одного акаунта на інший» не завжди можна вважати точно встановленим – він часто є лише ймовірно присутнім у певних межах. Таким чином, структура моделі (кого вважати джерелом, кого ретранслятором, які ознаки вважати релевантними для класифікації як фейк) сама по собі містить невизначеність. Це безпосередньо відповідає концепції інтервальної структурної ідентифікації: окремі зв’язки між змінними (джерело → аудиторія, початковий пост → хвиля поширення) існують не як «так/ні», а як «є в певному діапазоні впливу». Такий клас моделей є цінним для тестування, оскільки дозволяє перевіряти здатність метаевристичних алгоритмів не тільки підбирати параметри, а й формувати правдоподібну структуру інформаційної взаємодії.

Хоча наведені приклади належать до різних галузей (екологія довкілля, медична реабілітація, інформаційна безпека), з погляду ідентифікації інтервальних моделей вони мають однакові критично важливі властивості:

1. Невизначеність вимірювань і параметрів. У всіх чотирьох випадках дані спостережень є зашумленими, частково неповними або суб’єктивними. Це змушує працювати не з точковими параметрами, а з інтервалами припустимих значень.

2. Просторова або структурна розподіленість параметрів. У викидах автотранспорту – просторові ділянки вуличної мережі; у терапії кінцівок – сегменти кінцівки, суглоби, групи м’язів; у розповсюдженні фейкових новин – мережа акаунтів і каналів поширення. Тобто параметри моделі не є «глобально єдиними», вони залежать від локального контексту.

3. Багатовимірність і взаємозалежність факторів. Кожна система не описується одним фактором – навпаки, вплив формується комбінацією багатьох показників одночасно (екологічних, фізіологічних, поведінкових, технологічних). Це породжує ризик сильно корельованих змінних та складних нелінійних залежностей.

4. Потреба в глобальному пошуку рішень. Через велику кількість параметрів й наявність багатьох локальних мінімумів класичні локальні оптимізаційні методи стають ненадійними або просто не сходяться до прийнятного розв’язку. Потрібні глобальні стратегії пошуку – саме ті, що реалізуються в алгоритмах колективного інтелекту.

5. Критична роль “гіршого сценарію”, а не лише середнього. В екології важливий пік забруднення, а не середнє; у реабілітації важлива безпечна межа навантаження, а не тільки середня швидкість прогресу; у виявленні фейків важливі агресивні хвилі поширення, а не типова активність.

Таким чином, зазначені сфери застосування не лише служать для ілюстрації працездатності методу, а й задають вимоги до нього. Вони вимагають: інтервального представлення невизначеностей; можливості працювати з дискретними моделями з розподіленими параметрами, динамічними та статичними моделями; здатності виконувати глобальний пошук у високій розмірності. Ці вимоги безпосередньо ведуть до вибору підходів на основі алгоритмів колективного інтелекту, зокрема алгоритму бджолиної колонії. Як відзначалося

задачі структурної та параметричної ідентифікації на основі інтервальних даних відзначаються NP-обчислювальною складністю. Такі підходи призводять до того, що зі збільшенням кількості агентів і розмірності задачі ідентифікації різко зростає обчислювальна складність, що обмежує застосування таких методів в існуючих програмних середовищах. Одним із підходів, який забезпечує зниження часової складності є забезпечення функціонування агентів мультиагентних системи паралельно, зокрема із використання сучасних високопродуктивних обчислень на графічних прискорювачах таких як NVIDIA CUDA.

Виходячи із проведеного аналізу літературних джерел, сформульовано науково-технічне завдання, яке полягає у розробці методів структурної та параметричної ідентифікації інтервальних моделей систем програмними агентами метаевристичного алгоритму бджолиної колонії у середовищі NVIDIA CUDA.

Таким чином, метою дисертаційної роботи є зниження часової складності методів структурної та параметричної ідентифікації інтервальних моделей систем програмними агентами метаевристичного алгоритму бджолиної колонії у середовищі NVIDIA CUDA.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

1. Проаналізувати існуючі методи та програмні засоби ідентифікації інтервальних моделей систем.
2. Розробити методи структурної та параметричної ідентифікації інтервальних моделей систем із застосуванням програмних агентів, що виконують функції бджолиної колонії паралельно.
3. Розробити технологію застосування середовища NVIDIA CUDA для реалізації програмних агентів, що виконують функції бджолиної колонії для реалізації методів структурної та параметричної ідентифікації інтервальних моделей систем.
4. Розробити агентно-орієнтовану архітектуру програмної системи для ідентифікації інтервальних моделей.
5. Розробити програмне середовище для ідентифікації інтервальних моделей систем.

6. Провести апробацію розроблених методів та програмних засобів для задач побудови інтервальних моделей для екологічного моніторингу шкідливих викидів автотранспорту, побудови профілю користувача соціальних мереж, відновлення рухливості суглобів верхніх кінцівок під час арт-терапії.

Висовки до розділу 1

1. Проведено аналіз методів та програмних засобів параметричної й структурної ідентифікації інтервальних моделей систем. Розглянуто інтервальний підхід до моделювання й обґрунтовано доцільність представлення кожного параметра моделі не як фіксованого значення, а як інтервалу допустимих значень. Показано, що в такій постановці задача ідентифікації полягає у знаходженні не одного оптимального вектора параметрів, а множини узгоджених параметрів, які не суперечать експериментальним даним з урахуванням похибок вимірювання. Це забезпечує гарантовану оцінку поведінки системи в умовах шумів, неповних вимірів та варіабельності процесів.

2. Виявлено обмеження існуючих методів ідентифікації у випадку дискретних моделей. Показано, що такого типу задачі мають NP-складність. Обґрунтовано доцільність застосування метаевристичних методів колективного інтелекту як інструменту глобальної оптимізації для ідентифікації інтервальних моделей систем. Зокрема, обґрунтовано вибір алгоритму бджолиної колонії, як базового для подальшої реалізації завдяки його здатності підтримувати множину перспективних рішень, балансувати між пошуком нових областей і локальним уточненням та інтерпретуватися як система агентів, придатних до подальшої паралельної програмної реалізації.

3. Виходячи із проведеного аналізу літературних джерел, сформульовано науково-технічне завдання, яке полягає в розробленні методів структурної та параметричної ідентифікації інтервальних моделей систем програмними агентами метаеврестичного алгоритму бджолиної колонії у середовищі NVIDIA CUDA. Також сформульовано завдання дисертаційного дослідження.

РОЗДІЛ 2. МЕТОД МУЛЬТИАГЕНТНОЇ ІДЕНТИФІКАЦІЇ ІНТЕРВАЛЬНИХ МОДЕЛЕЙ НА ОСНОВІ АБК З ВИКОРИСТАННЯМ NVIDIA CUDA

У попередньому підрозділі проведено аналіз підходів до структурної та параметричної ідентифікації моделей систем. Виявлено обмеження класичних методів ідентифікації і було обґрунтовано доцільність застосування метаевристичних методів колективного інтелекту як інструменту глобальної оптимізації для ідентифікації інтервальних моделей.

У даному розділі описано формальну постановку задач параметричної і структурної ідентифікації інтервальних моделей систем. Обґрунтовано вибір технології NVIDIA CUDA для організації мультиагентної системи ідентифікації за рахунок масово паралельних обчислень на графічних процесорах. Описано метод паралельних обчислень ідентифікації з застосуванням алгоритму бджолиної колонії і сформовано гібридну схему CPU–GPU обчислень, де усі керувально-логічні фази АБК виконуються на CPU, а GPU відповідає за масово-паралельне обчислення функції мети для наборів кандидатів. Представлено універсальний шаблон ядра оцінки функції мети з динамічною спеціалізацією та компіляцією під конкретну практичну модель.

Формалізовано алгоритмічне забезпечення параметричної та структурної ідентифікації інтервальних моделей систем з використанням алгоритму бджолиної колонії.

Основні результати цього розділу опубліковано автором у працях [44, 46, 47].

2.1. Постановка задач ідентифікації інтервальних моделей систем

Моделювання об’єктів на основі інтервальних даних має низку переваг перед стохастичним підходом. Серед них – відсутність вимоги дослідження статистичних характеристик об’єкта моделювання. Як відомо, це зменшує кількість

експериментів. Тому інтервальний підхід більш придатний для дослідження властивостей об’єкта в умовах обмеженої вибірки даних.

Як вже зазначалось, задача параметричної ідентифікації інтервальних дискретних моделей складних об’єктів ґрунтується на інтервальних даних у формі:

$$[z^{-}_{i,j,h,k}; z^{+}_{i,j,h,k}], i = 0, \dots, I, j = 0, \dots, J, h = 0, \dots, H, k = 0, \dots, K, \quad (2.1)$$

де $[z^{-}_{i,j,h,k}; z^{+}_{i,j,h,k}]$, – числовий інтервал отриманої характеристики об’єкта експериментально в точці з дискретними координатами $i = 0, \dots, I, j = 0, \dots, J, h = 0, \dots, H$ та з часовою дискретою, $k = 0, \dots, K$ [5, 6, 42, 43].

Враховуючи множинний спосіб представлення результатів експерименту, тобто у вигляді інтервалів (2.1), задача параметричної ідентифікації математичної моделі характеристики об’єкта переважно має множину рівнозначних розв’язків. Математичну модель об’єкта розглядаємо, як дискретне рівняння у такому вигляді [84]:

$$v_{i,j,h,k}(\vec{V}) = f_1(\vec{V}) * g_1 + f_2(\vec{V}) * g_2 + \cdots + f_m(\vec{V}) * g_m$$
$$i = d, \dots, I, j = d, \dots, J, h = d, \dots, H, k = d, \dots, K, \quad (2.2)$$

де $v_{i,j,h,k}$ - означає модельоване значення характеристики об’єкта поза точками вимірювань та у дискретні моменти часу $k = d, \dots, K$; d – порядок різницевої схеми; $\vec{g}$– вектор параметрів моделі, значення яких необхідно оцінити на основі інтервальних даних; $f_1(\vec{V}), f_2(\vec{V}), f_m(\vec{V})$ – набір базисних функцій, Тобто для кожної базисної функції матимемо у виразі (2.2), такий вектор [5]:

$$\vec{V} = (v_{i-d,j-d,h-d,k-d}, v_{i-d+1,j-d,h-d,k-d}, \dots, v_{i-d+1,d,d,d}, \dots, v_{i,j,h-1,k})^T \quad (2.3)$$

Якщо у якійсь спосіб отримано оцінки вектора параметрів $\vec{g}$, то математична модель характеристики об'єкта матиме вигляд інтервального різницевого рівняння [43]:

$$[v_{i,j,h,k}\left(\left[\vec{\hat{V}}\right]\right)] = [f_1\left(\left[\vec{\hat{V}}\right]\right)] * \hat{g}_1 + [f_2\left(\left[\vec{\hat{V}}\right]\right)] * \hat{g}_2 + \cdots + f_m\left(\left[\vec{\hat{V}}\right]\right) * \hat{g}_m$$
$$i = d, \dots, I, j = d, \dots, J, h = d, \dots, H, k = d, \dots, K, \quad (2.4)$$

де $\left[\vec{\hat{V}}\right] = \left[\hat{v}_{i-d,j-d,h-d,k-d}\right], \dots, \left[\hat{v}_{i,j-d,h,k}\right], \dots, \left[\hat{v}_{i,j,h-1,k}\right]$ – інтервальний вектор з компонентами, які означають обчислені інтервальні оцінки $[v^-{}_{i,j,h,k}; v^+{}_{i,j,h,k}]$ характеристики.

Спираючись на гіпотезу, що обчислені інтервальні оцінки $[v^-{}_{i,j,h,k}; v^+{}_{i,j,h,k}]$ модельованої характеристики мають належати отриманим експериментально числовим інтервалам $[z^-_{i,j,h,k}; z^+_{i,j,h,k}]$, цієї ж характеристики об'єкта в точці з дискретними координатами $i = 0, \dots, I, j = 0, \dots, J, h = 0, \dots, H$ та у дискретні моменти часу $k = 0, \dots, K$, отримують математичну задачу для обчислення оцінки $\vec{g}$ вектора параметрів моделі-претендента [5, 6]:

$$\begin{cases} \left[\hat{v}^-_{0,0,0,0}; \hat{v}^+_{0,0,0,0}\right] \subseteq [z^-_{0,0,0,0}; z^+_{0,0,0,0}], \dots, \\ \left[\hat{v}^-_{d-1,d-1,d-1,d-1}; \hat{v}^+_{d-1,d-1,d-1,d-1}\right] \subseteq \\ [z^-_{d-1,d-1,d-1,d-1}; z^+_{d-1,d-1,d-1,d-1}] \\ z^-_{i,j,h,j} \le [f_1\left(\left[\vec{\hat{V}}\right]\right)] * \hat{g}_1 + [f_2\left(\left[\vec{\hat{V}}\right]\right)] * \hat{g}_2 + \cdots \\ + f_m\left(\left[\vec{\hat{V}}\right]\right) * \hat{g}_m \le z^+_{i,j,h,k}; \\ i = d, \dots, I, j = d, \dots, J, h = d, \dots, H, k = d, \dots, K \end{cases} \quad (2.5)$$

Отримана система є інтервальною системою нелінійних алгебричних рівнянь (ІСНАР) відносно невідомих компонент вектора $\vec{g}$ оцінок параметрів моделі. Причому, у цій ІСНАР вирази $\left[\hat{v}^-_{0,0,0,0}; \hat{v}^+_{0,0,0,0}\right] \subseteq [z^-_{0,0,0,0}; z^+_{0,0,0,0}], \dots, \left[\hat{v}^-_{d-1,d-1,d-1,d-1}; \hat{v}^+_{d-1,d-1,d-1,d-1}\right] \subseteq [z^-_{d-1,d-1,d-1,d-1}; z^+_{d-1,d-1,d-1,d-1}]$ - це початкові умови для різницевої схеми (2.4).

Математичний вираз у вигляді рівняння (2.4) називаємо інтервальною дискретною моделлю об’єкта, яку, як бачимо, налаштовуємо на основі експериментальних даних спостережень.

Множина розв’язків ІСНАР (2.5) визначає компоненти векторів $\vec{g}$ оцінок параметрів множини моделей-претендентів. Враховуючи високу обчислювальну складність (комбінаторну) розв’язування цієї ІСНАР, на практиці, обчислюють тільки точкові оцінки параметрів $\vec{g}$. При цьому процес оцінювання трансформують до розв’язування такої оптимізаційної задачі [7]:

$$\delta\left(\hat{\vec{g}}_l\right) \overset{\hat{\vec{g}}_l}{\rightarrow} min, \hat{\vec{g}}_l \subset \left[g_{jl}^{low}; g_{jl}^{up}\right], j = 1, \dots, m, l = 1, \dots, S \qquad (6)$$

де $g_{jl}^{low}, g_{jl}^{up}$ - нижнє та верхнє можливе значення кожного параметра моделі.

Сумісність ІСНАР (2.5) означатиме належність інтервалів значень прогнозованої характеристики $[\hat{v}_k]$ динамічного об’єкта на усіх часових дискретах $k = 0, \dots, K$ до експериментально отриманих інтервалів $[z_k^-; z_k^+]$.

Ітераційна процедура для методів оцінювання розв’язків ІСНАР (2.5) ґрунтується на кожній ітерації якості оцінки параметрів математичної моделі. Якість оцінки параметрів, задаємо величиною $\delta\left(\hat{\vec{g}}_l\right)$ у вигляді різниці центрів найбільш віддалених між собою модельованого та експериментального інтервалів для кожної часової дискрети, якщо ці інтервали не перетинаються. У випадку перетину цих інтервалів, функцію $\delta\left(\hat{\vec{g}}_l\right)$ визначатимемо найменшою шириною їх перетину. Вираз для функції $\delta\left(\hat{\vec{g}}_l\right)$ представимо у такому вигляді [6]:

$$\delta\left(\hat{\vec{g}}_l\right) = \max_{i=1,\dots,N}\{|mid([\hat{v}_k]) - mid([z_k^-; z_k^+])|\},$$
$$\text{якщо } [\hat{v}_k] \cap [z_k^-; z_k^+] = \emptyset, \exists k = 0, \dots, K \qquad (2.7)$$

$$\delta\left(\hat{\vec{g}}_l\right) = \max_{i=1,\dots,N}\{wid([\hat{v}_k]) - wid([\hat{v}_k] \cap [z_k^-; z_k^+])\},$$
$$\text{якщо } [\hat{v}_k] \cap [z_k^-; z_k^+] = \emptyset, \forall k = 0, \dots, K \qquad (2.8)$$

Очевидно, що для забезпечення збіжності ітераційної процедури, необхідно забезпечити таку послідовність оцінювання значень функції $\delta(\hat{\vec{g}}_1), \dots, \delta(\hat{\vec{g}}_l)$, обчислених за виразом (2.7) або (2.8), яка призводить до виконання таких умов:

$$\delta(\hat{\vec{g}}_1) > \delta(\hat{\vec{g}}_2) >, \dots, \delta(\hat{\vec{g}}_{l=L} \subset \Omega), \qquad (2.9)$$

де Ω позначає область розв'язків ІСНАР.

Розглянемо задачу структурної ідентифікації інтервальних моделей динамічних об'єктів. Математична модель об'єкта представлена у вигляді різницевого рівняння [6, 7]:

$$v_k = \vec{f}^T(v_{k-d}, \dots, v_{k-1}) * \vec{g}, k = d, \dots, K \qquad (2.10)$$

де v_k – модельована характеристика динамічного об'єкта на часовій дискреті $k = d, \dots, K$, d – порядок дискретної моделі, $\vec{g}$ – вектор невідомих параметрів моделі, $\vec{f}^T$ – вектор базисних функцій.

На відміну від задачі параметричної ідентифікації, вектор базисних функцій в задачі структурної ідентифікації є невідомим.

Для представлення математичної моделі характеристики динамічного об'єкта уведемо поняття множини структурних елементів [42]:

$$\lambda_s = \{f_1^s(v_{k-d}, \dots, v_{k-1}) * g_1^s, \dots, f_m^s(v_{k-d}, \dots, v_{k-1}) * g_m^s\} \qquad (2.11)$$

λ_s використовуємо для позначення s-тої структури, що відповідає певному набору структурних елементів, на основі якого будуємо s-ту модель у вигляді (2.10). Тоді математичні моделі, які розглядатимемо у процесі структурної ідентифікації будуть мати наступний вигляд [47]:

$$v_k(\lambda_s) = f_1^s(v_{k-d}, \dots, v_{k-1}) * g_1^s + \cdots + f_m^s(v_{k-d} \dots, v_{k-1}) * g_m^s \quad (2.12)$$

Зважаючи на отримані експериментальні інтервальні дані задаємо умову узгодженості математичної моделі:

$$v_k(\lambda_s, V_k) \in [z_k^-, z_k^+], \forall k = 0, \dots, d-1, d, \dots, K \quad (2.13)$$

де z_k^-, z_k^+ – нижня і верхня межа експериментально отриманих значень характеристики, $v_k(\lambda_s, V_k)$ – означає істинне значення вихідної характеристики для фіксованого набору структурних елементів λ_s і для фіксованих значень вектора $\vec{V} = (v_{k-d}, \dots, v_{k-1})^T$, у часових дискретах $k = 0, \dots, K$. Приймаючи до уваги умову (2.13) отримаємо наступну систему:

$$\begin{cases} z_0^- \le f_1^s(\vec{V}_0) * g_1^s + \cdots + f_m^s(\vec{V}_0) * g_m^s \le z_0^+ \\ \vdots \\ z_{d-1}^- \le f_1^s(\vec{V}_{d-1}) * g_1^s + \cdots + f_m^s(\vec{V}_{d-1}) * g_m^s \le z_{d-1}^+ \\ z_d^- \le f_1^s(\vec{V}_d) * g_1^s + \cdots + f_m^s(\vec{V}_d) * g_m^s \le z_d^+ \\ \vdots \\ z_K^- \le f_1^s(\vec{V}_K) * g_1^s + \cdots + f_m^s(\vec{V}_K) * g_m^s \le z_K^+ \end{cases} \quad (2.14)$$

Спираючись на той факт що в отриманій ІСНАР перші d інтервальних рівнянь це початкові умови, то перепишемо її у наступному вигляді:

$$\begin{cases} [v_0^-; v_0^+] \subseteq [z_0^-, z_0^+], \dots, [v_{d-1}^-; v_{d-1}^+] \subseteq [z_{d-1}^-, z_{d-1}^+] \\ \vdots \\ z_k^- \le f_1^s(\vec{V}_k) * g_1^s + \cdots + f_m^s(\vec{V}_k) * g_m^s \le z_k^+, k = d, \dots, K \end{cases} \quad (2.15)$$

Після проведення даних перетворень ми отримали загальну форму задачі параметричної ідентифікації інтервальних моделей динамічних об’єктів у вигляді ІНСАР для окремої s-тої моделі претендента. Розв’язок даної системо отримуємо з реалізації ітераційної процедури, на кожній ітерації якої, обчислюємо функцію

якості оцінки параметрів математичної моделі. У результаті перетворень, детально описаних в [7], задачу структурної ідентифікації інтервальних моделей динамічних об'єктів сформуємо у вигляді оптимізаційної задачі:

$$\begin{gathered}\delta(\lambda_s) \xrightarrow{\lambda_s=\left\{f_1^s(\vec{V})*g_{l1}^s,\ldots,f_{m_s}^s(\vec{V})*g_{lm_s}^s\right\}} min, \\ (m_s \in [I_{min}, I_{max}], f_1^s(\vec{V}), f_{m_s}^s(\vec{V}) \in F, \\ \hat{g}_{jl}^S \in \left[g_{jl}^-; g_{jl}^+\right], j = 1, \ldots, m, l = 1, \ldots, S\end{gathered} \tag{2.16}$$

де m_s – кількість структурних елементів s-ої інтервальної моделі, $[I_{min}, I_{max}]$ – мінімальне і максимальне значення кількості структурних елементів в моделі, S – кількість моделей-претендентів. F – множина потенційних структурних елементів моделі. $\hat{g}_{jl}^S$ – вектор оцінок параметрів моделі-претендента з структурою λ_s.

Варто зауважити, що в оптимізаційній задачі (2.16) цільова функція $\delta(\lambda_s)$ представлена алгоритмічно, відповідно саме цим обґрунтовано використання метаевристичного алгоритму оптимізації у вигляді алгоритму функціонування штучної колонії медоносних бджіл.

2.2. Обґрунтування вибору технології NVIDIA CUDA

CUDA – це модель програмування й виконання для графічних процесорів NVIDIA, побудована на парадигмі SIMT (Single Instruction, Multiple Threads) [28, 32, 33, 54, 90, 105]. На відміну від класичної SIMD-векторизації (де одна інструкція застосовується до фіксованого векторного регістра), у SIMT програміст мислить потоками: ми описуємо скалярний код «для одного логічного екземпляра обчислення», а обладнання одночасно запускає тисячі таких екземплярів як легкі потоки, які виконуються синхронізованими групами. Цей підхід поєднує простоту послідовного мислення з масштабом масово-паралельного виконання.

У CUDA програма-користувач запускає ядро (kernel) – в функцію, яку виконують безліч потоків. Потоки організуються у блоки потоків (thread blocks), а

блоки – у сітку (grid) [33]. Сітка – це весь фронт паралельної роботи ядра; блок – «міні-колектив» потоків, які можуть взаємодіяти через швидку спільну пам’ять і бар’єри синхронізації; потік – найдрібніша одиниця виконання, що оперує власними регістрами та індексом (для доступу до відповідного фрагмента даних).

Фізично GPU складається з багатьох мультипроцесорів (Streaming Multiprocessors, SM). Кожен SM апаратно планує виконання потоків варпами (warp) – фіксованими групами потоків (звично 32), що йдуть однією інструкційною доріжкою. Якщо всередині варпа різні потоки проходять різні гілки коду (умовні оператори), SM послідовно виконує ці гілки (явище дивергенції), тимчасово маскуючи неактивні потоки. Тому добре спроєктований CUDA-код мінімізує гілкування усередині варпа або реалізує його безгілково.

Щоб приховувати латентність доступів до пам’яті, SM утримує багато варпів у резидентному стані (концепція occupancy): поки один варп чекає на пам’ять, інший виконує обчислення. Кількість резидентних варпів обмежується ресурсами – регістрами та спільною пам’яттю (shared memory), які ядро декларує на блок. Тому баланс між розміром блока, використанням регістрів і обсягом shared-пам’яті визначає заповнення і, зрештою, продуктивність.

Ієрархія пам’яті в CUDA критична для швидкодії. Регістри – найшвидші, приватні для потоку. Спільна пам’ять (shared) – дуже швидка пам’ять усередині SM, спільна для потоків одного блока; її використовують для повторно вживаних даних і блокових редукцій. Глобальна пам’ять – велике, але відносно повільніше сховище, спільне для всієї сітки. Також існують L1/L2 кеші, константна та текстурна пам’яті, які пришвидшують певні патерни доступу. Для глобальної пам’яті визначальним є коалесцентний доступ: коли потоки одного варпа читають/пишуть послідовні адреси, апарат об’єднує їхні запити в невелику кількість великих транзакцій. Звідси випливає практичне правило зберігати дані у форматі «масивів полів» (SoA), щоб індекс потоку відповідав лінійному сегменту пам’яті.

Модель синхронізації в CUDA двошарова. Всередині блока потоки можуть узгоджуватися бар’єром, обмінюватися даними через shared-пам’ять і

використовувати атомарні операції. Між блоками пряма синхронізація в межах одного запуску ядра відсутня; загальна узгодженість досягається на межах ядер (kernel-by-kernel) або через більш складні засоби (кооперативні групи, розширені механізми запуску). Така організація заохочує алгоритми з слабо зв'язаними підзадачами, які можна розкласти на незалежні блоки обчислень.

Запуск ядер відбувається з хоста – CPU, який ініціює конфігурацію сітки та блоків (розмір grid/block) і передає керування пристрою – GPU. Для конвеєризації та перекриття копіювань із обчисленнями використовують потоки: незалежні черги команд, де копіювання даних і виконання ядер можуть накладатися за часом. Це дає змогу підтримувати високий ступінь завантаження пристрою навіть у сценаріях із частими запуском невеликих ядер.

З погляду алгоритмічного дизайну, SIMT-модель найкраще працює тоді, коли завдання можна сформулювати як велику кількість однакових скалярних обчислень з мінімальною взаємодією між ними та регулярними шаблонами доступу до пам'яті. Саме так виглядає кількісне ядро в нашій постановці: для кожного програмного агента багаторазово й незалежно обчислюється значення функції мети, а далі виконується коротка фаза відбору, що потребує лише локальних редукцій. У такій конфігурації «один агент = один потік» дає природну відповідність між логікою алгоритму та апаратною організацією GPU.

Нарешті, важливо розрізняти логічний паралелізм програми і фактичне розкладання на ресурси. CUDA-планувальник може виконувати блоки в довільному порядку і на різних SM – розробник не керує прив'язкою блоків до конкретних мультипроцесорів. Це спрощує масштабування: збільшення обсягу даних чи кількості кандидатів зазвичай зводиться до збільшення сітки, після чого планувальник самостійно розподіляє роботу по наявних SM, підтримуючи високу пропускну здатність за рахунок великої кількості резидентних варпів і приховування латентностей.

Вибір технології NVIDIA CUDA в цій дисертації не є суто інженерним рішенням. Він випливає безпосередньо з математичної постановки задачі ідентифікації інтервальних моделей систем [34, 52, 80, 104, 124], зокрема у

формулюванні на основі алгоритму бджолиної колонії, і з характеру обчислювального навантаження, яке виникає під час реалізації цього алгоритму.

Ідентифікація на основі АБК вимагає масово-паралельної оцінки великої множини кандидатів. Алгоритм бджолиної колонії оперує популяцією кандидатних рішень – програмних агентів. На кожній ітерації для кожного агента обчислюється функція мети – тобто наскільки добре відповідний набір параметрів (або структура) відтворює спостережувану поведінку системи з урахуванням інтервальної невизначеності. Ці оцінки не залежать одна від одної: агент A може бути оцінений повністю незалежно від агента B. У термінах паралельних обчислень це називається слабо зв’язаним навантаженням – коли одна й та сама операція повторюється для великої кількості незалежних екземплярів. Саме до такого навантаження оптимізовані сучасні графічні процесори: вони призначені для того, щоб одночасно виконувати велику кількість однакових обчислювальних потоків, кожен із яких обробляє свій екземпляр даних.

У контексті АБК це означає: чим більша популяція агентів S, тим краще завантажується GPU і тим продуктивнішим стає пошук. Для CPU ситуація протилежна: після того як задіяно всі апаратні ядра, подальше збільшення S не прискорює обчислення – кандидати починають чекати своєї черги для обчислення. На GPU ж кілька тисяч агентів можуть бути оброблені практично одночасно, оскільки модель виконання CUDA допускає запуск тисяч потоків, розподілених між багатьма мультипроцесорами.

Одна із ключових причин, чому АБК добре масштабується на GPU, полягає в тому, що взаємодія між агентами зводиться до дуже простих операцій відбору – порівняння які рішення кращі і статистичного вибору перспективних джерел для подальшого вивчення. Ця взаємодія не вимагає повного обміну станами між усіма агентами на кожному кроці. Практично це означає, що більшість часу потоки працюють із локальними даними, не блокуючи один одного.

Це різко відрізняється від паралельних методів оптимізації, які потребують частих глобальних редукцій або синхронізації градієнтів (як у класичних градієнтних методах у великих просторах параметрів) – такі методи гірше

піддаються розгортанню на багатьох потоках без втрат ефективності. У нашому випадку міжагентна координація може бути винесена або в окремі короткі редукційні ядра GPU (які збирають показники якості і визначають найкращі рішення), або навіть частково виконана на CPU між ітераціями. Тобто алгоритм АБК за своєю природою створений так, що легко розділяється на незалежні фрагменти обчислень і добре лягає на масово-паралельну архітектуру.

У задачах ідентифікації інтервальних моделей систем особливо важливими є випадки, коли модель не є простою алгебраїчною залежністю, а описана різницевими рівняннями. У такому випадку обчислення для одного агента перестає бути звичайною алгебраїчною формулою і перетворюється на велику кількість рекурентних обчислень. Класично це робиться послідовно для кожного кандидата, що при великих популяціях стає вузьким місцем. CUDA якраз дозволяє віддати кожного такого кандидата на цілий блок потоків і проганяти його динаміку паралельно по часово-просторовій сітці. Це означає, що навіть у випадку складнішої моделі обчислення лишається масштабованим. Для CPU збільшення складності окремого кандидата означає, що час на одного кандидата росте, а можливість паралелізуватися не покращується пропорційно – адже кількість ядер не збільшується.

У даній роботі ідентифікація розглядається не як «знайти один оптимальний набір параметрів», а як «побудувати множину допустимих параметрів, узгоджених з даними». Це означає, що система має дослідити не одну-дві точки простору параметрів, а широку область цього простору. Такий режим роботи вимагає дуже великої кількості перевірок – обчислень функції мети. Вартість цих перевірок визначає реалістичність методу: чи можемо ми дозволити собі досліджувати десятки тисяч кандидатів за розумний час.

Саме тут GPU дає не просто прискорення у відсотках, а якісну зміну режиму: він робить можливим дослідження великих популяцій, систематичне підтримання різноманіття рішень, архівування множини прийнятних параметрів, а не тільки одного локального мінімуму. CPU в цьому сценарії обмежує розмір популяції та

глибину пошуку. Тобто вибір CUDA напряму підтримує наукову мету дисертації – знаходити не просто найкраще, а множину допустимих інтервальних рішень.

CUDA є природною платформою для поставленої задачі, оскільки:

- модель АВС з інтервальними обмеженнями є масово-паралельною за своєю суттю;
- обчислення для різних кандидатів незалежні й можуть виконуватись одночасно тисячами потоків;
- обміни між кандидатами зводяться до простих редукцій та вибору найкращих;
- масштабування популяції і складності моделі на GPU лінійно підсилює пошук, тоді як на CPU дуже швидко впирається у межу апаратного паралелізму.

Інакше кажучи використання технології CUDA не просто пришвидшує вже існуючу процедуру ідентифікації, а робить можливим масово-паралельну побудову множини припустимих інтервальних моделей, яка на звичайних паралельних обчисленнях на CPU була би або надто повільною, або вимагала би радикального спрощення моделі.

2.3. Метод організації паралельних обчислень ідентифікації з застосуванням алгоритму бджолиної колонії

У запропонованій реалізації алгоритму бджолиної колонії використано гібридну CPU–GPU схему: усі керувально-логічні фази алгоритму виконуються на CPU, тоді як GPU прискорює масово-паралельне обчислення функції мети для всієї популяції кандидатів. Такий поділ праці дозволяє, з одного боку, не ускладнювати код гілкуваннями на пристрої, а з іншого – повністю завантажувати GPU однорідними обчисленнями, що ідеально відповідають SIMT-парадигмі.

В основі методу оргазнізації паралельних обчислень лежить розроблений шаблон ядра для GPU обчислень, динамічна спеціалізація під прикладну модель та його JIT компіляція.

Базовим будівельним блоком є шаблон ядра CUDA:

```
__global__ void EvaluateNectar(const KernelFoodSource* fss,
                               const float* dz,
                               const float* du,
                               bool isInterval,
                               float* r,
                               int N)
{
    int t = blockDim.x * blockIdx.x + threadIdx.x;
    if (t < N)
    {
        XXXDimData::Evaluate(fss[t], dz, du, isInterval, r[t]);
    }
}
```

Цей каркас залишає незмінною схему розпаралелювання за кандидатами («один потік = один кандидат»), але динамічно підміняє «тілесність» обчислення у статичному методі. Відповідно реалізовано три варіанти ядра:

- OneDimData::Evaluate (часовий ряд/1D сітка);
- TwoDimData::Evaluate (2D сітка);
- ThreeDimData::Evaluate (3D сітка).

Спеціалізація «тілесності» виконується динамічно: під конкретну прикладну задачу збирається PTX файл з підстановкою формул для обчислення, початкових даних та розмірів сітки, після чого PTX файл завантажується в контекст пристрою. Усі розміри сіток та допуски підставляються на етапі генерації PTX файлу, що усуває гілкування та дозволяє компілятору ефективно розкладати масиви у регістри та локальну пам’ять. На стороні .NET це забезпечено зв’язкою C# [113] та програмного пакета ManagedCUDA: хост-код формує батч кандидатів, завантажує PTX, конфігурує сітку/блоки, викликає ядро, а потім забирає масив значень функції мети.

Оцінювання інтервальної узгодженості реалізовано безпосередньо на пристрої з елементарними інтервальними операціями, а також з функціями перетину та ширини інтервалу. Для кожної точки часово-просторової сітки (k), (i,j) або (k,i,j) обчислюється інтервальний прогноз V моделі і порівнюється з

інтервальним спостереженням Z. Міра локальної невідповідності формулюється безгілково.

Значення функції мети кандидата r визначається через редукцію цієї міри на сітці, у нашому випадку – максимальне значеннння відхилення оцінки від експериментальних даних. Така постановка придатна до SIMT: локальні операції виконуються незалежно, редукція завершується у спільній пам'яті блока, а в глобальну пам'ять записується тільки один скаляр $r[t]$ на кандидата.

Дані кандидатів подаються у вирівняному вигляді із щільним зберіганням параметрів у масиві. Експериментальні дані Z та керуючі впливи U розкладаються лінійно в порядку, який відповідає порядку обходу потоками, забезпечуючи коалесцентні читання/записи. Для часто використовуваних коефіцієнтів моделі застосовується shared-пам'ять на рівні блока.

Типовий цикл має вигляд: CPU: GenerateCandidates() → GPU: EvaluateNectar[…]() → CPU: SelectAndUpdate():

- створюється контекст CUDA, завантажується і заповнюється шаблон PTX ядра;
- фіксуються дані експерименту Z та U на пристрої;
- на кожній ітерації АБК CPU формує батч кандидатів, передає буфер на графічний процесор та отримує результат оцінки;
- виконується відбір/оновлення популяції на CPU.

У межах однієї ітерації АБК оцінка різних кандидатів не взаємозалежна, тому паралельний розклад коректний. Обчислювальна вартість однієї ітерації наближено:

$$T_{iter} \approx T_{CPUgen} + T_{H2D} + T_{GPU\ eval} + T_{D2G} + T_{CPUselect} \qquad (2.17)$$

де домінує $T_{GPU\ eval}$ за великих розмірів популяції. Основні обмеження – чутливість до дивергенції (використання шаблонів і динамічної компіляції виключає формування великих гілкових умов), пропускна здатність PCIe (передається тільки

динамічні змінні), обсяг регістрів при великій кількості претендентів (тому доцільно тримати «1 кандидат = 1 блок»

Запропонований метод організації паралельних обчислень поєднує універсальний шаблон ядра з динамічною спеціалізацією «тілесності» під конкретну модель та розмірність сітки. Гібридний цикл АБК (на CPU логіка алгоритму, на GPU батч-оцінка функції мети) забезпечує масштабованість, відтворюваність і високу продуктивність задач інтервальної ідентифікації, не вимагаючи складної синхронізації між агентами і повністю використовуючи сильні сторони SIMT-архітектури CUDA.

2.4. Алгоритмічне забезпечення задачі параметричної ідентифікації на основі алгоритму бджолиної колонії

Як вже зазначалось, найефективнішим методом розв’язування задачі параметричної ідентифікації інтервальної дискретної моделі, як оптимізаційної задачі (2.6) є метаевристичний алгоритм штучної бджолиної колонії, який ґрунтується на принципах ройового інтелекту. Основна ідея АБК полягає у реалізації поведінки колонії медоносних бджіл у процесі пошуку нектару.

У контексті поведінки бджолиної колонії, спочатку із вулика вилітають бджоли-розвідники, які шукають джерела нектару у випадковому напрямі. Вони визначають цінність обраного джерела – кількісна характеристика, яка залежить від кількості нектару та відстані до вулика. Дану фазу будемо називати ініціалізацією. Після повернення до вулика, наступає фаза робочих бджіл. На основі отриманої інформації бджолами-розвідниками, робочі бджоли обирають джерело нектару, до якого полетять. Робочі бджоли мають дослідників джерел нектару. Це фаза бджіл дослідників. Чим більша цінність джерела нектару, тим більше бджіл до нього полетить. Якщо джерело вичерпалося, наступає фаза бджіл-розвідників, які як і у випадку ініціалізації, випадковим чином обирають нове джерело нектару.

Розглянемо дану поведінкову модель в рамках розв'язування задачі параметричної ідентифікації (2.6). Представимо даний алгоритм у вигляді схеми, наведеної на рис. 2.1.

На початку роботи алгоритму визначаємо вхідні дані та налаштовуємо параметри алгоритму, такі як *S* – чисельність усієї популяції бджіл, *MCN* – загальна кількість ітерацій, *mcn=1* – номер поточної ітерації, *LIMIT* – число, яке визначає вичерпність джерела [46].

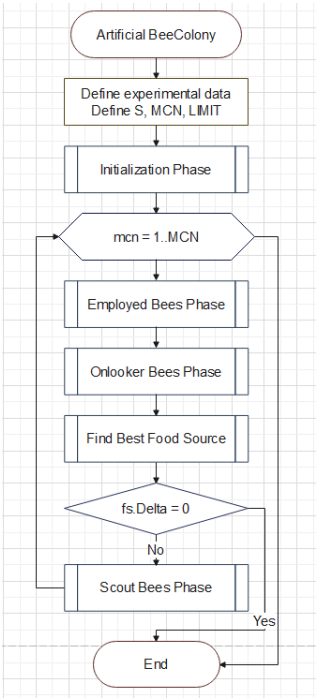


Рис. 2.1 Схема алгоритму параметричної ідентифікації

Тепер розглянемо її паралельну реалізацію на кожній фазі.

Фаза ініціалізації. Вектори, які визначають можливі точки мінімуму функції мети із (2.6) – вектори оцінок параметрів позначаємо за $\hat{\vec{g}}_l$. У контексті поведінкової моделі бджолиної колонії це означає, що кожен вектор координат джерела нектару відповідає одній l-тій бджолі. Ініціалізуємо вектори $\hat{\vec{g}}_l$ випакдовим чином використовуючи наступну формулу:

$$\left[\hat{g}_{jl}^{-}; \hat{g}_{jl}^{+}\right] = [g_{jl}^{low}] + rand(0,1) * \left(g_{jl}^{up} - g_{jl}^{low}\right), j = 1, \dots, m, l = 1, \dots, S \quad (2.18)$$

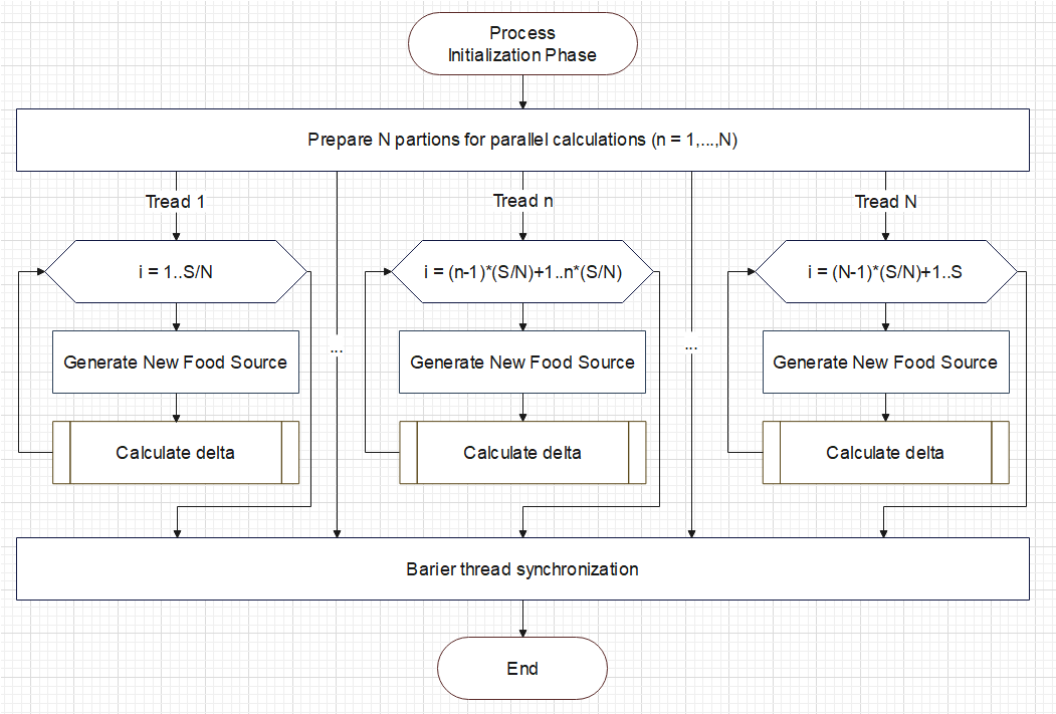


Рис. 2.2 Схема процедури Initialization Phase з розпаралеленням обчислювального процесу

Після цього проводимо обчислення якості оцінки параметрів математичної моделі за формулами (2.7) і (2.8). Оскільки вище описані обчислення незалежні між собою для кожного вектора $\hat{\vec{g}}_l$, їх можна провести паралельно [52, 66, 70]. Для проведення паралельних обчислень розділимо усю множину векторів $\hat{\vec{g}}_l$ на N частин, де N – кількість обчислювальних вузлів системи, і остача ділення S на N рівна 0 [82, 86, 87, 95]. Фаза ініціалізація представлена на схемі на рис.2.2.

Фаза робочих бджіл. В контексті оптимізаційної задачі дана фаза означає пошук нових оцінок розв'язків з меншим значення функції мети (2.7) і (2.8). Для обчислення координат можливих точок локального мінімуму функції мети використовуємо такі формули [46]:

$$\hat{g}_{jl}^{mcn} = \hat{g}_{jl} + \Phi_{jl} * \left(\hat{g}_{jl} - \hat{g}_{jp}\right), j = 1, \dots, m, p \neq l = 1, \dots, S \quad (2.19)$$

$$\hat{g}_{jl}^{mcn} = \hat{g}_{jl} - \Phi_{jl} * \left(\hat{g}_{jl} - \hat{g}_{jp}\right), j = 1, \dots, m, p \neq l = 1, \dots, S \quad (2.2)$$

де Φ_{jl} – випадкове число з діапазону [-1;1], , $j = 1, \dots, m$ – випадково обраний індекс параметра, $\hat{g}_p$ – випадково обраний вектор координат нектару із $p \neq l = 1, \dots, S$.

Формулу (2.20) використовуємо в тому випадку якщо отриманий параметр $\hat{g}_{jl}^{mcn}$ виходить за визначені межі $[g_{jl}^{low}; g_{jl}^{up}]$.

Після обчислення координат можливих точок мінімуму $\hat{g}_{jl}^{mcn}$ проводиться попарне порівняння існуючих і поточних значень оцінок параметрів із застосуванням функції мети (2.7) і (2.8):

$$\hat{\bar{g}}_l = \{\hat{\bar{g}}_l, \text{якщо } \delta(\hat{\bar{g}}_l) \leq \delta(\hat{\bar{g}}_l^{mcn})\}$$
$$\text{або} \qquad (2.21)$$
$$\hat{\bar{g}}_l = \{\hat{\bar{g}}_l^{mcn}, \text{якщо } \delta(\hat{\bar{g}}_l) > \delta(\hat{\bar{g}}_l^{mcn})\}$$

Процес порівняння (2.21) відображений на схемі на рис. 2.3. В контексті алгоритму інкрементуємо лічильник спроб змінити дане джерело нектару, щоб мати можливість уникнути локального мінімуму в майбутньому.

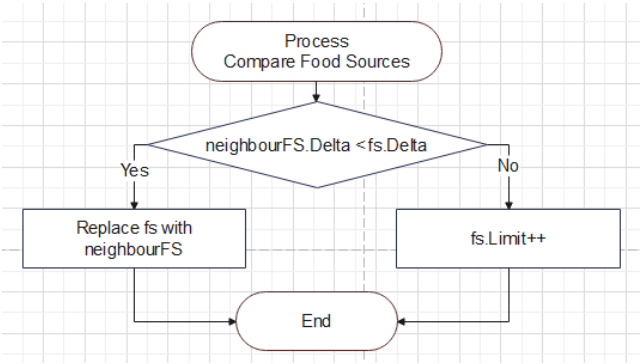


Рис. 2.3 Схема процедури Compare Food Sources

Оскільки вибір координат можливих точок локального мінімуму функції мети за формулами (2.19) або (2.20) залежать від випадково обраного вектора координат сусіднього джерела нектару, то спершу проведемо генерацію нових координат для дослідження для усіх поточних координат. Обчислення функції мети і порівняння значень оцінок параметрів незалежне для кожного вектора $\hat{\bar{g}}_l^{mcn}$. Тому розділимо усю множину векторів $\hat{\bar{g}}_l^{mcn}$ на N частин і проведемо обчислення паралельно [46]. Блок схема паралельного алгоритму фази робочих бджіл зображено на рис.2.4.

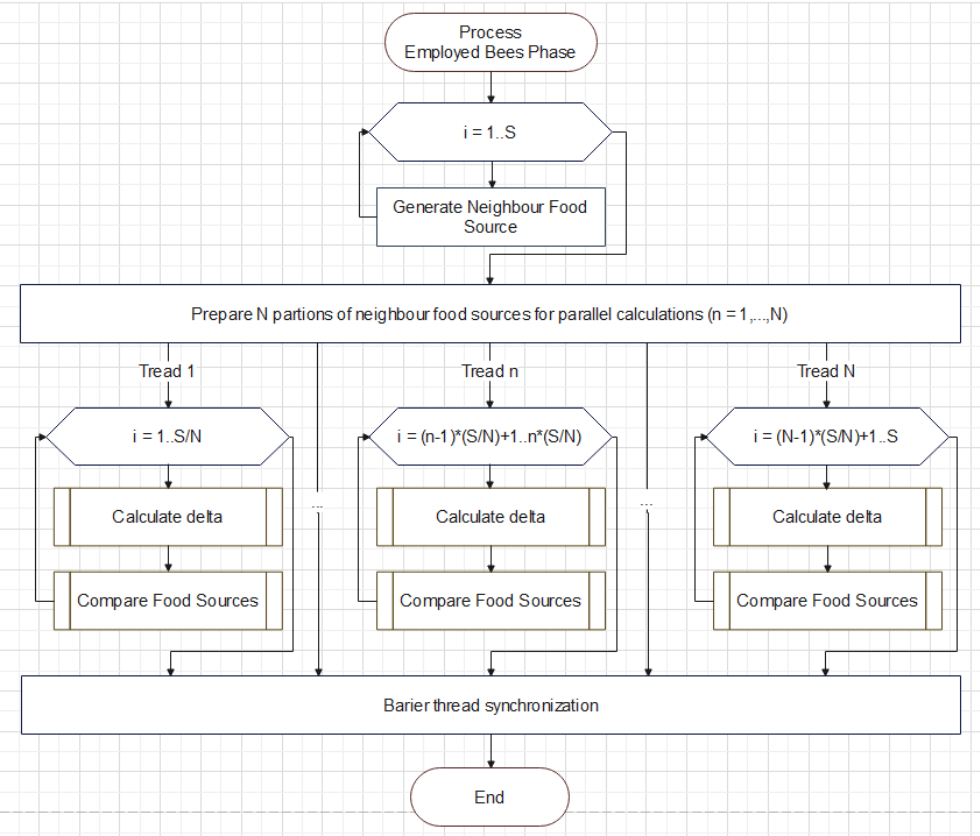


Рис. 2.4 Схема процедури Employed Bees Phase з розпаралеленням обчислювального процесу

Фаза бджіл дослідників. В контексті оптимізаційної задачі, на цій стадії визначаємо найбільш ймовірні точки (вектори значень параметрів), в околі яких необхідно проводити детальне дослідження функції мети. Для цього для кожної точки, встановленої на попередній фазі, розраховуємо ймовірність P_l за наступною формулою:

$$P_l = \frac{1-\delta([\hat{\bar{g}}_l])}{\sum_{l=1}^{S}(1-\delta([\hat{\bar{g}}_l]))} \tag{2.22}$$

Для проведення паралельних обчислень для кожної поточної точки підготуємо окіл точок для дослідження. На підставі розрахованих ймовірностей за формлуою (2.22) визначається кількість точок для дослідження околу можливих локальних мінімумів функції мети – $m_l = int(P_l * S)$ точок. Відповідно у випадку $m_l > 0$, генеруємо m_l сусідніх точок. Після обчислення координат кожної точки околу, запускаємо N потоків для обчислення, як і на попередніх фазах. В кожному

потоці для точок у яких $m_l > 0$, для кожної точки з околу обчислюємо функцію мети за формулою (2.7) чи (2.8) і проводимо порівняння за формулою (2.21). Дана фаза відображена в схемі на рис.2.5.

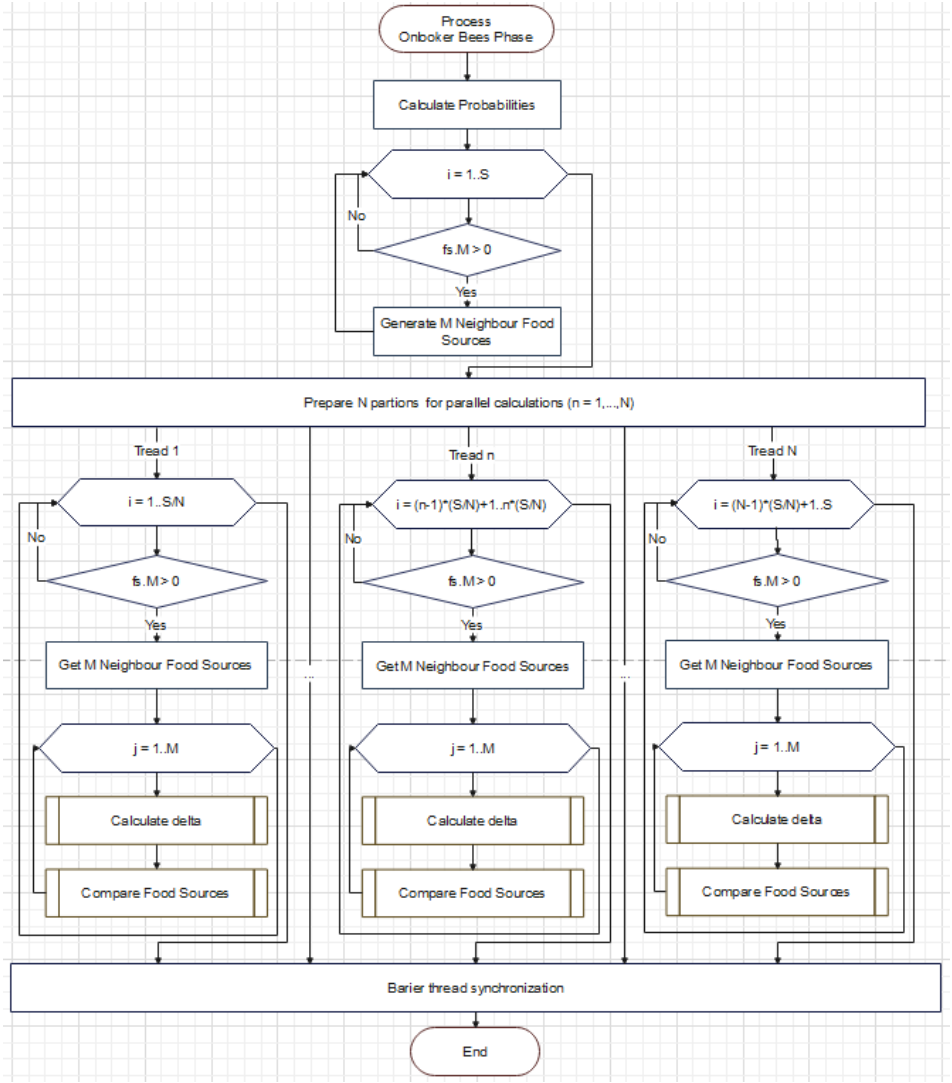


Рис. 2.5 Схема процедури Onlooker Bees Phase з розпаралеленням обчислювального процесу

Наступний кроком є пошук точки з найменшим значенням функції мети на даній ітерації алгоритму. Для того щоб дане значення не враховувалося в процесі виходу з локальних мінімумів встановимо лічильник спроб *limit = 0*, рис.2.6. Якщо функція мети $\delta(\hat{\vec{g}}_l) = 0$, це свідчить що розв'язок оптимізаційної задачі знайдено, відповідно завершуємо виконання алгоритму.

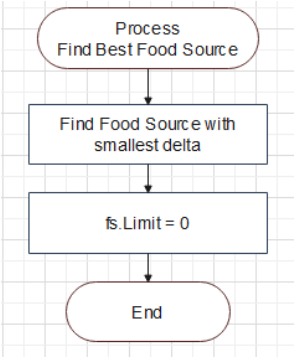


Рис. 2.6 Схема процедури Find Best Food Source

Фаза бджіл розвідників. Для уникнення зациклення на локальних мінімумах функції мети, використовується фаза бджіл розвідників. В контексті поведінкової моделі бджолиної колонії це свідчить про вичерпання джерела нектару. Тому для кожної точки перевіряємо чи не перевищує її лічильник спроб *limit* граничне значення *LIMIT*, визначене на етапі налаштування алгоритму [46]. Точки, що перевищили *LIMIT* ми замінюємо новими, які генеруємо за допомогою формули (2.10). Дана фаза відображена на рис. 2.7.

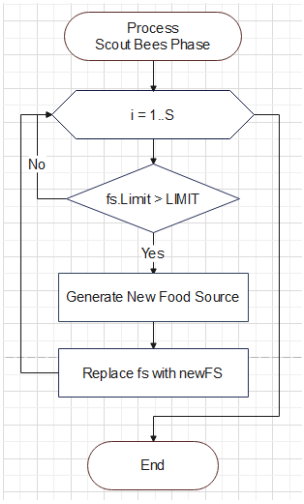


Рис. 2.7 Схема процедури Scout Bees Phase

2.5. Алгоритмічне забезпечення задачі структурної ідентифікації на основі алгоритму бджолиної колонії

Розглянемо основні фази методу структурної ідентифікації моделей динамічних об’єктів на основі поведінкових моделей бджолиної колонії. На рис. 2.8 у вигляді схеми наведено узагальнену обчислювальну послідовність перетворення структур.

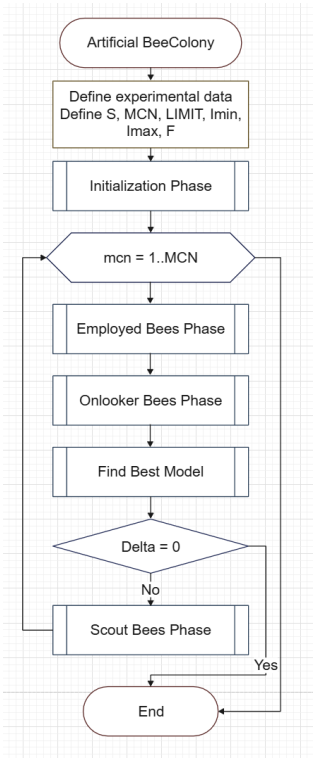


Рис. 2.8 Схема алгоритму структурної ідентифікації

Першим кроком алгоритму є введення початкових змінних: $[z_k^-, z_k^+]$ – інтервальні дані отримані в ході проведення експерименту, S – чисельність усієї популяції бджіл, що відповідає кількості моделей-претендентів, MCN – загальна кількість ітерацій, $LIMIT$ – число, яке визначає вичерпність джерела, $[I_{min}, I_{max}]$ – мінімальне і максимальне значення кількості структурних елементів в моделі та множину структурних елементів F [47].

На фазі ініціалізації формуємо початкову множину моделей-претендентів Λ_0 потужністю S випадковим чином із набору структурних елементів F. (Рис.2.9), метод Generate New Model. За допомогою алгоритму параметричної ідентифікації [46] для кожної моделі обчислюємо значення функції мети.

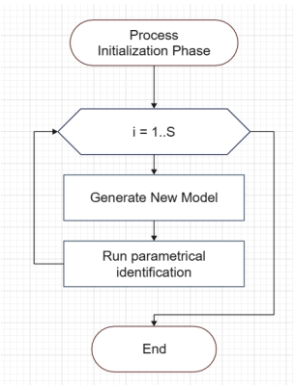


Рис. 2.9 Схема процедури Initialization Phase

Важливо підмітити, що для алгоритму параметричної ідентифікації було доопрацьовано умови для завершення алгоритму. Так як при структурні ідентифікації більшість моделей-претендентів не описують математичну модель адекватно умови $\delta(\lambda_s) = 0$ не достатньо, оскільки алгоритм параметричної оптимізації ніколи не зможе її досягти. Окрім обмеження по кількості ітерації *MCN* у алгоритмі параметричної ідентифікації введено змінну, що характеризує покращення моделі після кожної ітерації. Якщо значення функції мети на поточні ітерації покращилося менше очікуваного значення завершуємо алгоритм параметричної ідентифікації: $\delta_{mcn} - \delta_{mcn-1} < \Delta_\delta$

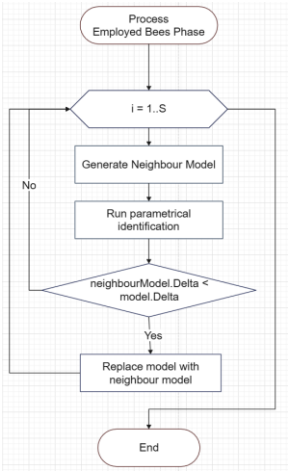


Рис. 2.10 Схема процедури Employed Bees Phase

Схема на рис. 2.10 відображає реалізацію фази робочих бджіл, що відповідає за синтез множини поточних моделей Λ'_{mcn}. Метод Generate Neighbour Model на основі моделі λ_s формує нову модель λ'_s, яка є “околом”. При цьому нову модель λ'_s формує у спосіб випадкового вибору і заміни частини структурних елементів поточної моделі λ_s. При обчисленні елементів n_s, які потрібно замінити, враховуємо якість $\delta(\lambda_s)$ поточної моделі і кількість її елементів $m_s \in [I_{min}, I_{max}]$ [7]:

$$n_s = \begin{cases} int\left(\left(1 - \frac{min\{\delta(\lambda_s)|s=1..S\}}{\delta(\lambda_s)}\right) * m_s\right), \\ if\ \delta(\lambda_s) \neq min\{\delta(\lambda_s)|s = 1..S\}\ and\ n_s \neq 0 \\ \\ 1, \\ if\ \delta(\lambda_s) = min\{\delta(\lambda_s)|s = 1..S\}\ or\ n_s = 0 \end{cases} \quad (2.23)$$

На цій же фазі проводимо селекцію, для вибору кращої моделі з поточної і генерованої:

$$\lambda_s^1 = \begin{cases} \lambda_s, \ if \ \delta(\lambda_s) < \delta(\lambda_s') \\ \lambda_s', \ if \ \delta(\lambda_s') < \delta(\lambda_s) \end{cases} \quad (2.24)$$

Схема на рис.2.11 Відображає реалізацію фази бджіл дослідників. В контексті задачі структурної ідентифікації це означає визначення околу дослідження поточної моделі λ_s. Для генерації околу використовуємо описаний вище метод Generate Neighbour Model. Для визначення кількості моделей в околі будемо використовувати ймовірнісний підхід, який описаний наступною формулою і реалізований в методі Calculate Probabilities:

$$P_s(\lambda_s^1) = \frac{1-\delta(\lambda_s^1)}{\sum_{s=1}^{S}(1-\delta(\lambda_s^1))}, s = 1..S \quad (2.25)$$

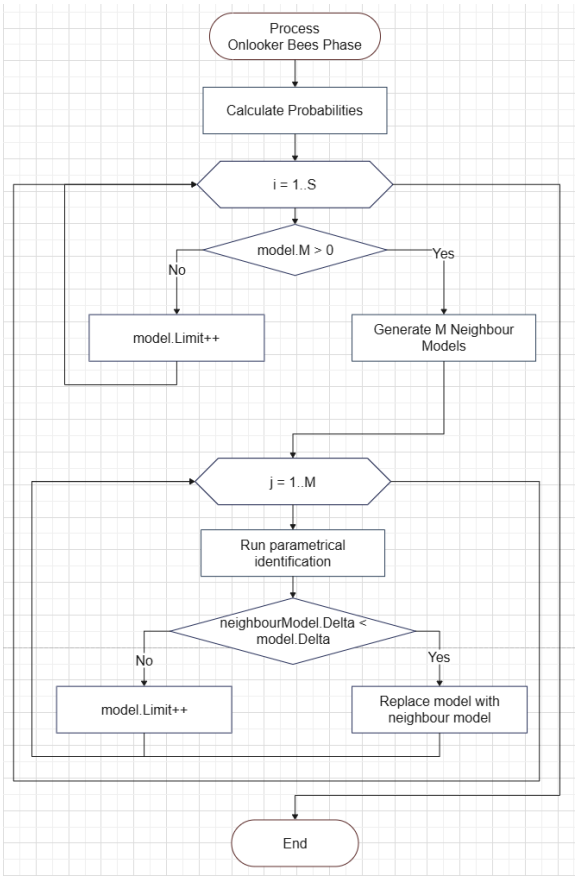


Рис. 2.11 Схема процедури Onlooker Bees Phase

Тепер, отримуємо точні значення кількості новостворених моделей в околі $m_s = int(P_s * S)$. Для кожної моделі введемо показник limit, що відповідає за вичерпність джерела нектару в контексті поведінкової моделі бджолиної колоній, і слугує умовою виходу із локальних мінімумів в контексті розв'язання оптимізаційної задачі структурної ідентифікації. Далі на цій фазі проводимо групову селекцію для кожного сформованого околу поточної моделі λ_s. Значення лічильника limit збільшуємо на 1 кожного у якщо при груповій селекції поточна модель не оновилася.

За допомогою метода Find Best Model обираємо модель з найменшим значенням функції мети $\delta(\lambda_s)$. Якщо $\delta(\lambda_s) = 0$, завершуємо процедуру структурної ідентифікації, в протилежному випадку переходимо до наступної фази.

Схема на рис. 2.12 Відображає реалізацію фази бджіл розвідників. Дана фаза реалізовує механізм виходу з локальних мінімумів. Якщо значення лічильника limit поточної моделі λ_s перевищило значення змінної *LIMIT*, вказаної на етапі ініціалізації, тоді ця модель вважається локальним мінімум і її потрібно замінити. Нову модель генеруємо випадковим чином, так само як на фазі ініціалізації, за допомогою методу Generate New Model.

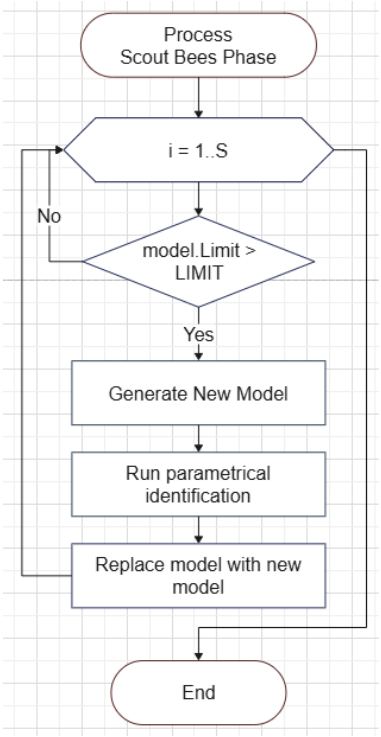


Рис. 2.12 Схема процедури Scout Bees Phase

Висовки до розділу 2

1. Проведено аналіз формального представлення задач параметричної і структурної ідентифікації інтервальних моделей систем. Показано, що обидві задачі є оптимізаційними задачами у випадку, коли замість пошуку інтервальних оцінок параметрів моделі достатньо знайти їх точкові значення.

2. Вперше розроблено методи структурної та параметричної ідентифікації інтервальних моделей систем, які на відміну від існуючих ґрунтуються на застосуванні програмних агентів, що виконують функції поведінкової моделі бджолиної колонії паралельно, що забезпечило зниження часової складності ідентифікації інтервальних моделей систем.

3. Вперше запропоновано та обґрунтовано для реалізації методів структурної та параметричної ідентифікації інтервальних моделей систем технологію динамічної компіляції CUDA-ядра, яка забезпечує функціонування програмних агентів поведінкової моделі бджолиної колонії в середовищі NVIDIA CUDA, що у сукупності забезпечило зниження часової складності ідентифікації інтервальних моделей систем. Сформовано гібридну схему CPU–GPU, в якій усі керувально-логічні фази АБК виконуються на CPU, а GPU використовується для масово-паралельного обчислення функції мети для наборів кандидатів. Запропонований універсальний шаблон ядра оцінки функції мети з динамічною спеціалізацією та компіляцією під конкретну практичну модель забезпечує повторне використання каркаса ядра та швидку адаптацію до постановок задач без зміни інтерфейсу виклику.

РОЗДІЛ 3. АГЕНТНО ОРІЄНТОВАНА АРХІТЕКТУРА ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ІДЕНТИФІКАЦІЇ ІНТЕРВАЛЬНИХ МОДЕЛЕЙ

У попередньому розділі розглянуто алгоритмічне забезпечення структурної та параметричної ідентифікації інтервальних моделей систем та методи організації мультиагентної паралельної обчислювальної схеми алгоритму бджолиної колонії.

У даному розділі обґрунтовано архітектурні рішення програмної системи ідентифікації інтервальних моделей на основі алгоритму бджолиної колонії. Описано трирівнева організацію програмної системи – інтерфейс користувача, алгоритмічна бібліотека та обчислювальний CUDA бекенд. Інверсія залежностей через інтерфейс обчислювача дозволила ізолювати логіку АБК від конкретного середовища виконання та підтримати підстановність обчислювачів (послідовний, паралельний CPU, CUDA).

Описано UML-діаграми, що формалізують як статичну структуру, так і динаміку системи. Компонентні та класові моделі відображають межі відповідальності модулів і ключові колаборації; діаграми станів описують життєві цикли контролера АБК і CUDA-виконавця з явними точками відновлення після збоїв; діаграма послідовностей показує конвеєр пакетного оцінювання та місця накладання операцій.

Описано графічний інтерфейс користувача, що реалізує сценарії керування експериментами: конфігурування параметрів, імпорт/експорт CSV/JSON, збереження/відновлення проєктів, візуалізацію перебігу та результатів у реальному часі, інформування про прогрес довготривалих запусків.

Досліджено ефективність та часову складність розроблених алгоритмів ідентифікації інтервальних моделей систем із застосуванням технології NVIDIA CUDA.

Основні результати цього розділу опубліковано автором у працях [8, 9, 45, 49].

3.1. Архітектура програмного забезпечення

Програмна система побудована за трирівневою гібридною архітектурою з чітким розподілом відповідальностей [27, 56]: UI-рівень (WinForms) для конфігурації задачі та візуалізації; Core-рівень (бізнес-логіка) для керування алгоритмом АБК і вибору стратегії виконання; CUDA-бекенд для високопродуктивного обчислення функції мети на GPU. Такий поділ забезпечує підтримуваність, розширюваність і ефективне використання ресурсів.

Керувально-логічні фази АБК (генерація кандидатів, відбір, *scout*, критерії зупинки) реалізовано у Core, який оркеструє виконання та обирає один із трьох режимів: послідовний, паралельний на CPU, або CUDA-GPU для великомасштабних задач. Це рішення фіксує єдиний алгоритмічний контур і дозволяє масштабуватися із зростанням розмірності без зміни зовнішніх контрактів [116].

CUDA-рівень відповідає за масово-паралельні обчислення функції мети (fitness), що є найбільш часозатратною частиною процесу ідентифікації. Ключова ідея – динамічна генерація ядра: шаблон кернела наповнюється під конкретну модель, компілюється NVRTC у PTX та виконується на GPU [96]. Так ми уникаємо надлишкового гілкування в кернелі, зберігаємо SIMT-дружній код і швидко адаптуємося до нових задач.

Архітектура підпорядкована принципу мінімізації обмінів між хостом і пристроєм [85]: між ітераціями АБК передаються лише батчі кандидатів, експериментальні дані і повертаються скалярні значення функції мети. Пам’ять і передавання керуються спеціалізованим менеджером (виділення память, H2D/D2H, очистка) з профілюванням, щоб уникати витоків і досягати стабільної пропускної здатності.

Принципи відтворюваності й спостережуваності закладені в основу: UI/дані/результати ізольовані, підтримуються імпорт/експорт (CSV/JSON), логування часу, збір метрик продуктивності та механізми коректного відновлення після помилок, що критично для GPU-виконання.

Користувачі можуть імпортувати дані у форматах CSV або JSON, після чого вони проходять валідацію та перетворюються у внутрішні подання, придатні для оптимізації; також можливе ручне заповнення цих даних через інтерфейс користувача. Система підтримує збереження та завантаження повних станів проєкту – включно з параметрами, даними та результатами – у форматі JSON. Для цього використовується уніфікована JSON схема наведена нижче.

```
{
 "$schema": "http://json-schema.org/draft-07/schema#",
 "title": "ABC Identification Config",
 "type": "object",
 "required": ["S", "Limit", "LowerBounds", "UpperBounds", "GLength", "DataGridModel"],
 "properties": {
  "S": { "type": "integer", "minimum": 1, "description": "Population size"},
  "Mcn": { "type": ["integer", "number"], "description": "Max cycle number (-1 = till solution)." },
  "Limit": { "type": "integer", "minimum": 0, "description": "Trials limit before scout reinitialization." },
  "DeltaCap": { "type": "number", "minimum": 0 },
  "RelativeDeltaCap": { "type": "number", "minimum": 0 },
  "ParallelOption": { "type": "integer", "minimum": 0, "description": "0=sequential, 1=CPU-parallel, 2=GPU" },
  "ThreadsNumber": { "type": "integer", "minimum": 1 },
  "ResetLimit": { "type": "boolean" },
  "LowerBounds": {
   "type": "array",
   "items": { "type": "number" },
   "minItems": 1,
   "description": "Length should equal GLength."
  },
  "UpperBounds": {
   "type": "array",
   "items": { "type": "number" },
   "minItems": 1,
   "description": "Length should equal GLength."
  },
  "NumberOfRuns": { "type": "integer", "minimum": 1 },
  "GLength": { "type": "integer", "minimum": 1 },
  "DeltaZ": { "type": "number", "minimum": 0 },
  "DeltaV": { "type": "number", "minimum": 0 },
  "InitK": { "type": "integer", "minimum": 0 },
  "InitI": { "type": "integer", "minimum": 0 },
  "InitJ": { "type": "integer", "minimum": 0 },
  "Func": { "type": "string"},
  "EvaluationCount": { "type": "boolean" },
  "MeasureTime": { "type": "boolean" },
  "DataGridModel": {
```

```
  "type": "object",
  "required": ["Dim", "IsInterval"],
  "properties": {
    "Dim": { "type": "integer", "enum": [1, 2, 3] },
    "IsInterval": { "type": "boolean" },
    "OneDimData": {
      "type": ["array", "null"],
      "description": "For Dim=1: array of numbers or [[L,R], ...] when IsInterval=true."
    },
    "OneDimDataU": { "type": ["array", "null"] },
    "TwoDimData": {
      "type": ["array", "null"],
      "items": { "type": "array", "items": { "type": ["number", "array"] } },
      "description": "For Dim=2: array of rows; cells are numbers or [L,R] if IsInterval=true."
    },
    "TwoDimDataU": { "type": ["array", "null"] },
    "ThreeDimData": {
      "type": ["array", "null"],
      "description": "For Dim=3: 3D array [[[...]]]; numbers or [L,R] if IsInterval=true."
    },
    "ThreeDimDataU": { "type": ["array", "null"] }
  }
},
"TestDataGridModel": {
  "type": ["object", "null"],
  "properties": { "$ref": "#/properties/DataGridModel" }
},
"ControlDataGridModel": { "type": ["object", "null"] },
"TestControlDataGridModel": { "type": ["object", "null"] },
"Result": { "type": ["object", "null"] },
"StructuralData": {
  "type": "object",
  "required": ["IsEnabled"],
  "properties": {
    "IsEnabled": { "type": "boolean" },
    "S": { "type": "integer", "minimum": 1 },
    "Mcn": { "type": ["integer", "number"] },
    "Limit": { "type": "integer", "minimum": 0 },
    "DeltaCap": { "type": "number", "minimum": 0 },
    "RelativeDeltaCap": { "type": "number", "minimum": 0 },
    "OrderK": { "type": "integer", "minimum": 0 },
    "OrderI": { "type": "integer", "minimum": 0 },
    "OrderJ": { "type": "integer", "minimum": 0 },
    "Power": { "type": "integer", "minimum": 0 },
    "MinI": { "type": "integer", "minimum": 0 },
    "MaxI": { "type": "integer", "minimum": 0 },
    "Multiplication": { "type": "boolean" },
```

```
      "Division": { "type": "boolean" },
      "Elements": {
        "type": "array",
        "items": { "type": "string" },
        "description": "Structural basis terms (e.g., 'V[i-0,j-1]', '1/(V[i-0,j-2])', ...)."
      }
    }
  }
 }
}
```

Результати оптимізації візуалізуються в режимі реального часу за допомогою графіків і таблиць, що дає змогу користувачам ефективно інтерпретувати та аналізувати отримані результати; див. діаграму потоку даних на рис. 3.1.

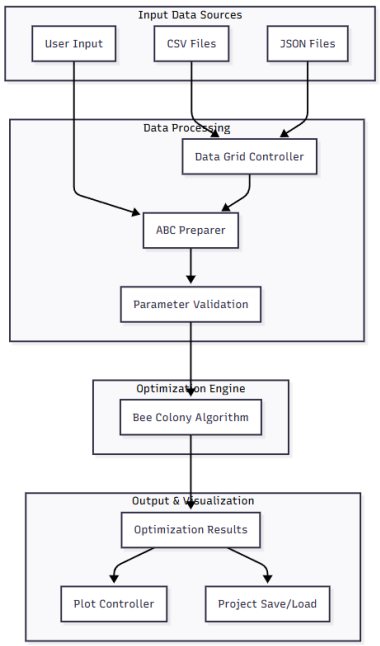


Рис. 3.1 Діаграма потоку даних.

Файли CSV використовуються для введення даних отриманих в рамках експерименту. Система підтримує одно-, дво- та тривимірні точкові та інтервальні дані.

Описані в цьому підрозділі діаграми формалізують поведінку програмної системи впродовж усього життєвого циклу оптимізації: від ініціалізації конфігурації та вибору стратегії виконання (CPU/CPU-паралельно/CUDA) до ітераційного керування алгоритмом бджолиної колонії з фазами Employed, Onlooker та Scout, а також окремо – життєвий цикл CUDA-виконавця (генерація/завантаження PTX, H2D, запуск кернела, D2H, обробка збоїв). Така

формалізація забезпечує однозначність трактування станів, подій і умов переходів, фіксує точки керованого відновлення (NVRTC/PTX-помилки, «device lost», некоректні дані) та інваріанти відтворюваності (моменти фіксації seed, межі для стохастичних рішень), а також узгоджується з компонентною та sequence-діаграмами, роблячи архітектуру системи прозорою для аналізу, тестування й подальшого розширення.

Діаграма станів ініціалізації (рис 3.2) моделює життєвий цикл контролера під час старту: побудову «тілесності» цільової функції, вибір режиму паралелізації (CPU/CPU-паралельно/CUDA) та створення обчислювального виконавця.

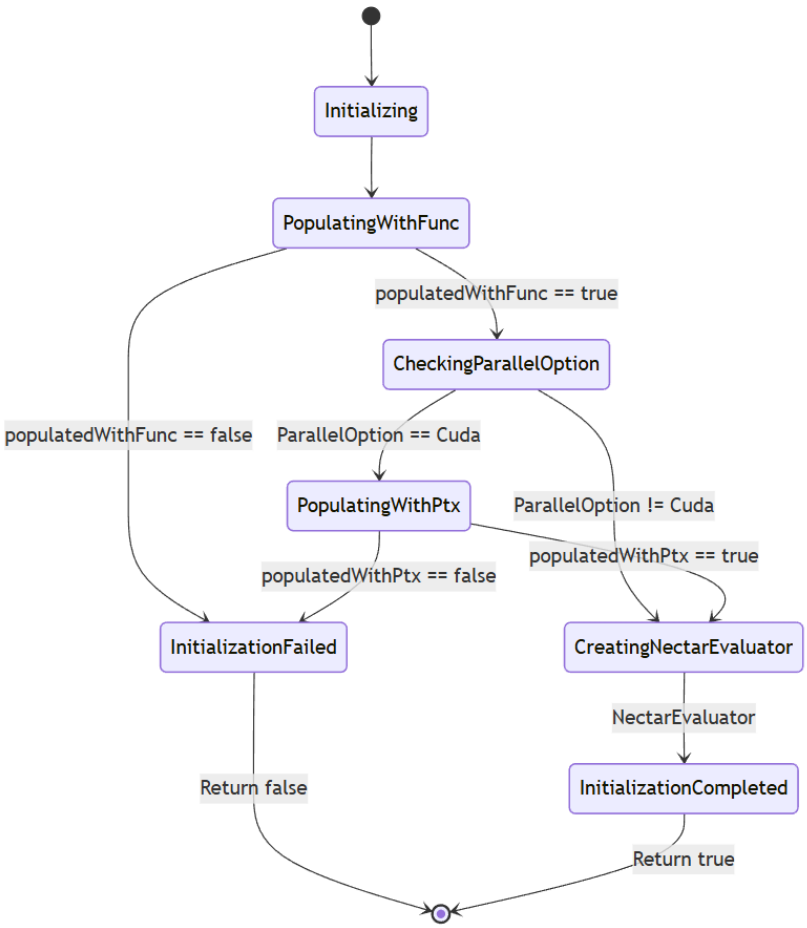


Рис. 3.2 Діаграма станів ініціалізації

Початковий стан: Initializing. Фінальні стани: InitializationFailed (помилка, повертається false) або InitializationCompleted (успіх, повертається true).

Ключова логіка:

1. PopulatingWithFunc – формування «тілесності» функції мети;

2. CheckingParallelOption – вибір стратегії виконання;

3. за потреби PopulatingWithPtx – генерація/підготовка PTX;

4. CreatingNectarEvaluator – створення оцінювача (CPU або GPU) та підготовка буферів.

Якщо будь-який крок завершується невдало, відбувається перехід у InitializationFailed; у разі успіху – у Ready. Події визначають розгалуження за ознаками populatedWithFunc, ParallelOption, populatedWithPtx і результатом Initialize ().

Табл. 3.1 – Стани фази ініціалізації

Стан	Призначення	Entry/Do/Exit (основне)
Constructed	Контролер створено, ресурси не ініціалізовані	–
Initializing	Послідовність кроків InitializeAsync()	entry: завантажити конфіг/дані; do: валідація; exit: лог стану
PopulatingWithFunc	Формування «тілесності» цільової функції під модель	do: побудова формул
CheckingParallelOption	Вибір режиму: CPU чи CUDA	do: перевірити ParallelOption
PopulatingWithPtx	Генерація PTX файлу для кернела	do: NVRTC/ завантаження PTX
CreatingNectarEvaluator	Створення оцінювача функції мети	exit: зафіксувати ресурси
Ready	Ініціалізацію успішно завершено	entry: прапорець «готово»
InitializationFailed	Ініціалізація неуспішна	entry: встановити повідомлення/код помилки

Табл. 3.2 – Переходи фази ініціалізації

Звідки → Куди	Подія	Умова (guard)	Дія
Constructed → Initializing	InitializeAsync()	–	Запуск послідовності ініціалізації
PopulatingWithFunc → InitializationFailed	–	populatedWithFunc == false	Лог причини, зупинка
PopulatingWithFunc → CheckingParallelOption	–	populatedWithFunc == true	–
CheckingParallelOption → PopulatingWithPtx	–	ParallelOption == Cuda	Підготовка до NVRTC/PTX
CheckingParallelOption → CreatingNectarEvaluator	–	ParallelOption != Cuda	Створити CPU-виконавець
PopulatingWithPtx → InitializationFailed	–	populatedWithPtx == false	Лог помилки компіляції/завантаження
PopulatingWithPtx → CreatingNectarEvaluator	–	populatedWithPtx == true	Розв’язати функцію кернела
Initializing → InitializationFailed	–	InitializeAsync() == false	Повернути false
Initializing → Ready	–	InitializeAsync() == true	Повернути true

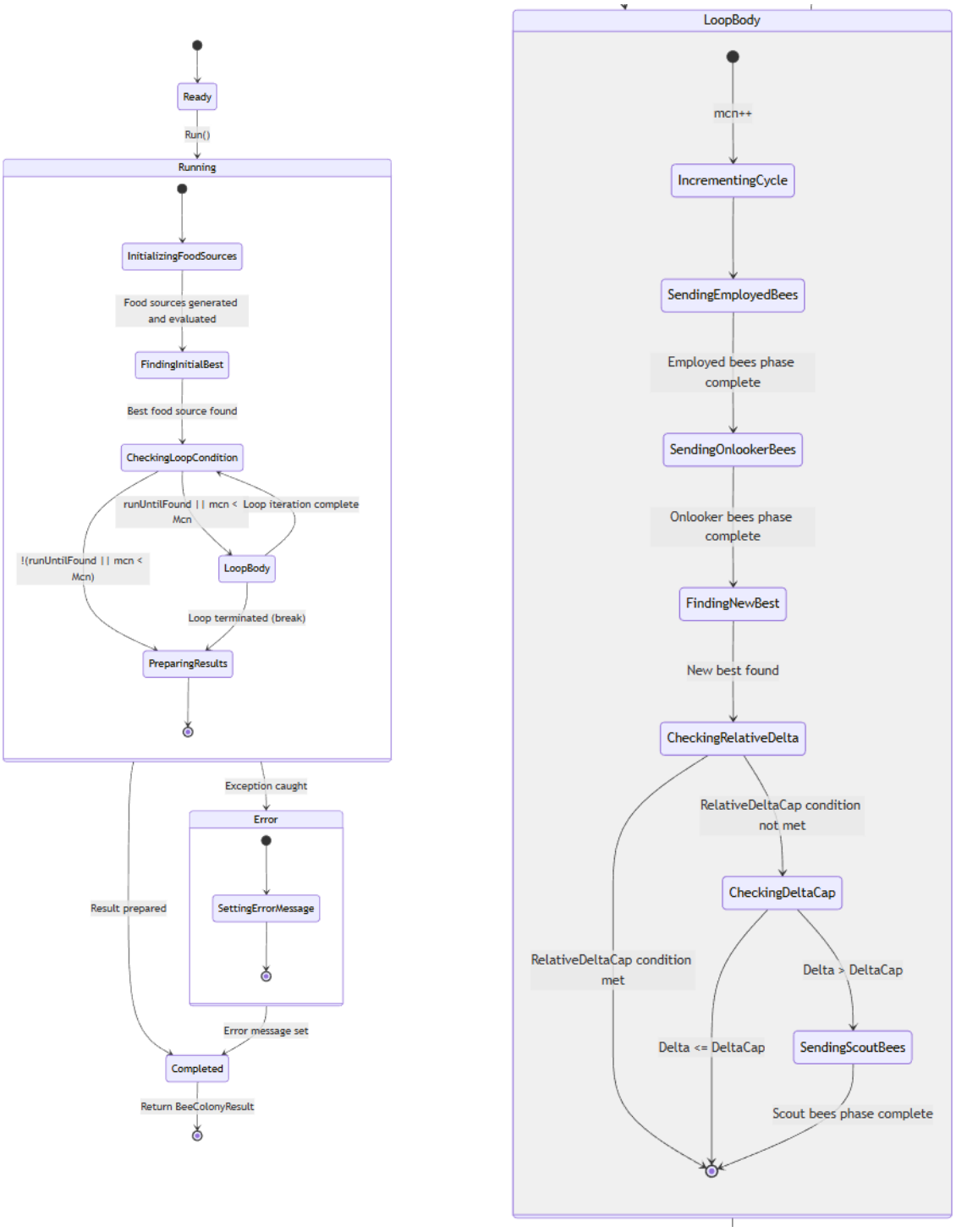


Рис. 3.3 Діаграма станів алгоритму бджолиної колонії

Діаграма на рис. 3.3 описує ітераційний цикл алгоритму бджолиної колонії: ініціалізацію популяції, фази *Employed* і *Onlooker*, механізм *Scout*, перевірку критеріїв зупинки та підготовку результату. Початковий стан: Ready. Запуск: Ready → Running (подія Run(CancellationToken)). Завершення: Completed (повернення BeeColonyResult) або гілка Error → Completed. Після генерації та первинної оцінки джерел (InitializingFoodSources) відбирається

початковий найкращий (FindingInitialBest) і перевіряється умова циклу (CheckingLoopCondition). У разі продовження виконується композитний підстан LoopBody: інкремент циклу (IncrementingCycle), фаза SendingEmployedBees, фаза SendingOnlookerBees, оновлення глобального найкращого (FindingNewBest) і перевірки CheckingRelativeDelta та CheckingDeltaCap. Якщо покращення недостатнє, запускаються SendingScoutBees. По завершенні ітерації повернення у CheckingLoopCondition або дострокове припинення (PreparingResults). Помилки обробляються у стані Error з контрольованим завершенням у Completed.

Табл. 3.3 – Стани основного циклу алгоритмк бджолиної колонії

Стан	Призначення	Entry/Do/Exit (основне)
Ready	Готовність до запуску оптимізації	–
Running	Композитний стан основного циклу	entry: підготовка метрик
InitializingFoodSources	Генерація початкових джерел і первинна оцінка	do: батч-оцінка (CPU/GPU)
FindingInitialBest	Вибір початково найкращого	–
CheckingLoopCondition	Перевірка умови `runUntilFound	
LoopBody	Одна ітерація циклу	–
IncrementingCycle	Збільшення лічильника ітерацій	do: mcn++
SendingEmployedBees	Фаза employed (сусіди, батч-оцінка)	do: H2D → kernel → D2H
SendingOnlookerBees	Фаза onlooker (рулетка/рангування, батч-оцінка)	do: H2D → kernel → D2H
FindingNewBest	Оновлення глобально найкращого	–

CheckingRelativeDelta	Перевірка відносного покращення	–
CheckingDeltaCap	Перевірка абсолютної дельти	–
SendingScoutBees	Реініціалізація вичерпаних джерел	do: random reinit, reset trials
PreparingResults	Формування результатів	entry: архів Θ*\Theta^*Θ*, метрики
Error	Обробка виключення	entry: лог помилки, контекст
Completed	Повернення BeeColonyResult	exit: звільнення ресурсів (за потреби)

Табл. 3.4 – Переходи основного циклу алгоритмк бджолиної колонії

Звідки → Куди	Подія	Умова (guard)	Дія
Ready → Running	Run	–	Запуск оптимізації
InitializingFoodSources → FindingInitialBest	–	Початкові джерела згенеровано та оцінено	–
FindingInitialBest → CheckingLoopCondition	–	Найкращий знайдений	–
CheckingLoopCondition → LoopBody	–	runUntilFound	
CheckingLoopCondition → PreparingResults	–	!runUntilFound	
LoopBody → IncrementingCycle	–	–	mcn++

IncrementingCycle → SendingEmployedBees	–	–	Згенерувати сусідів (employed)
SendingEmployedBees → SendingOnlookerBees	–	Фазу employed завершено	–
SendingOnlookerBees → FindingNewBest	–	Фазу onlooker завершено	–
FindingNewBest → CheckingRelativeDelta	–	Новий найкращий знайдено	–
CheckingRelativeDelta → [вихід з LoopBody]	–	Умова RelativeDeltaCap виконана	Перервати ітерації
CheckingRelativeDelta → CheckingDeltaCap	–	Інакше	–
CheckingDeltaCap → [вихід з LoopBody]	–	Delta <= DeltaCap	Перервати ітерації
CheckingDeltaCap → SendingScoutBees	–	Delta > DeltaCap	Реініціалізація вичерпаних
SendingScoutBees → [вихід з LoopBody]	–	Фазу scout завершено	–
LoopBody → CheckingLoopCondition	–	Ітерацію завершено	–
LoopBody → PreparingResults	–	Примусовий break	Завершити
Running → Error	Exception	–	Обробити помилку
Error → Completed	–	Повідомлення встановлено	Повернути результат з помилкою
Running → Completed	–	Результат підготовлено	Повернути BeeColonyResult

Діаграми на рис 3.4 і рис. 3.5 формалізують життєвий цикл CUDA-виконавця, який відповідає за масово-паралельне обчислення функції мети: від створення контексту та завантаження ядра до виконання батч-оцінювання та коректної утилізації ресурсів.

Початковий стан: Constructed. Ключові завершальні стани: Ready (виконавець готовий до роботи), Disposed (усі ресурси звільнено), Failed (ініціалізація неуспішна).

Логіка ініціалізації. Після виклику конструктора перехід у композитний стан Initializing: створення контексту CUDA (CreatingCudaContext), перевірка наявності PTX (CheckingPtx), завантаження ядра (LoadingKernel). Успіх веде в Ready, збої – у Failed.

Логіка виконання. Виклик Evaluate(FoodSource[]) переводить з Ready у композитний стан Evaluating. Якщо батч порожній (CheckingEmpty) – повернення у Ready. Інакше – оновлення лічильника оцінок (UpdatingCount), старт асинхронного завдання (StartingTask) і вхід у EvaluatingInternal: встановлення контексту (SettingCudaContext), підготовка хост-даних (PreparingHostData), виділення пам’яті на пристрої (AllocatingDeviceMemory), конфігурування ядра (ConfiguringKernel), запуск (RunningKernel), копіювання результатів (CopyingResults) і звільнення пристроєвої пам’яті (DisposingDeviceMemory). Після успішного завершення встановлюються обчислені значення нектару (SettingNectarValues) і повернення у Ready.

Утилізація. Виклик Dispose() переводить у композитний стан Disposing: спершу загальна утилізація базових ресурсів (DisposingBase), далі – контексту CUDA (DisposingCudaContext), після чого – Disposed.

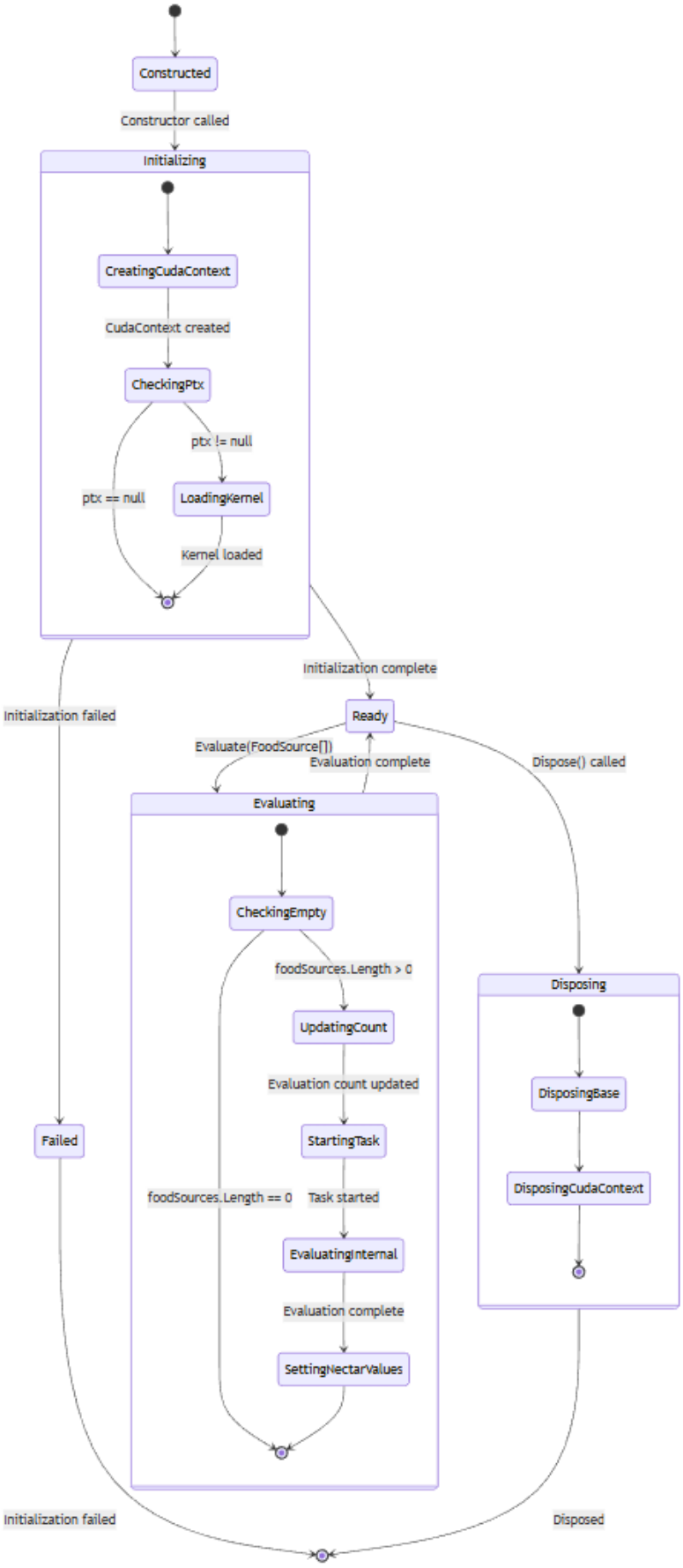


Рис. 3.4 Діаграма станів CUDA обчислювача

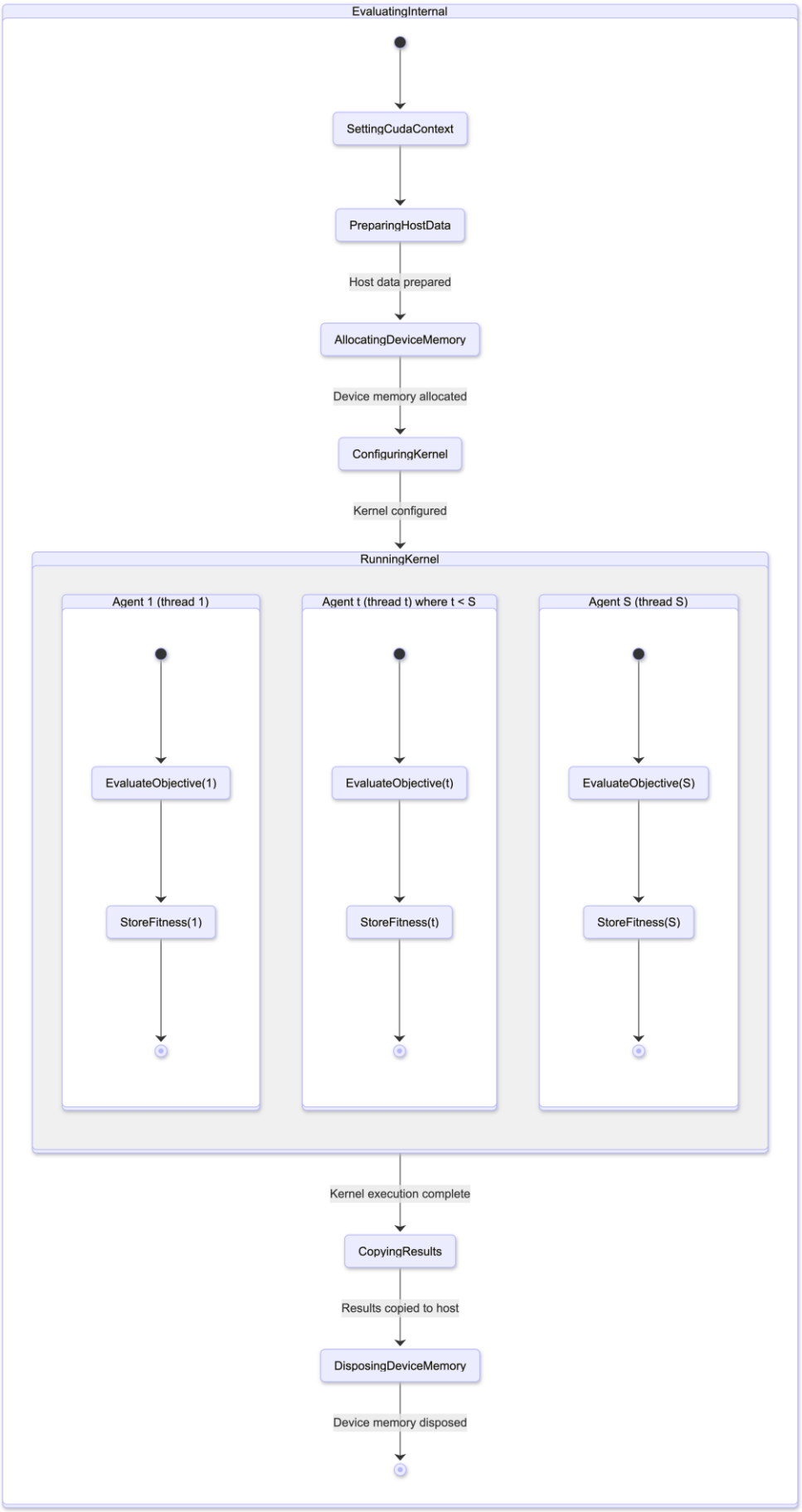


Рис. 3.5 Діаграма станів CUDA обчислювача (EvaluatingInternal)

Таблиця 3.5 Стани CUDA обчислювальника

Стан	Призначення	Entry/Do/Exit (основне)
Constructed	Об’єкт створено, ресурси не готові	–
Initializing	Композитний стан ініціалізації	entry: перевірити конфігурацію; do: підготовка
CreatingCudaContext	Створення контексту CUDA	do: cuCtxCreate / вибір пристрою
CheckingPtx	Перевірка наявності PTX	do: валідація PTX
LoadingKernel	Завантаження модуля та функції ядра	do: cuModuleLoadDataEx, cuModuleGetFunction
Ready	Виконавець готовий приймати дані для обчислення	entry: контекст активний, PTX завантажено
Evaluating	Композитний стан оцінювання батчу	–
CheckingEmpty	Перевірка порожнього батчу	–
UpdatingCount	Оновлення лічильників	do: ++evaluationCount
StartingTask	Пуск асинхронного завдання	do: створити Task
EvaluatingInternal	Композит: власне GPU-обчислення	–
SettingCudaContext	Прив’язка контексту до поточного потоку	do: cuCtxSetCurrent
PreparingHostData	Формування буферів хоста	do: маршалінг і валідація

AllocatingDeviceMemory	Виділення device-пам’яті	do: cuMemAlloc / cudaMalloc
ConfiguringKernel	Виставлення grid/block параметрів	do: <<<grid, block, stream>>>
RunningKernel	Запуск EvaluateNectar	do: запуск, перевірка статусу
CopyingResults	D2H копіювання результатів	do: cudaMemcpyAsync
DisposingDeviceMemory	Звільнення device-буферів батчу	do: cuMemFree
SettingNectarValues	Присвоєння значень нектару на хості	do: присвоїти значення функції мети
Disposing	Композит: завершальна утилізація	–
DisposingBase	Звільнення базових ресурсів	do: Dispose
DisposingCudaContext	Очищення контексту CUDA	do: cuModuleUnload, cuCtxDestroy
Failed	Ініціалізація неуспішна	entry: зберегти код/повідомлення
Disposed	Усі ресурси звільнено	–

Таблиця 3.6 Переходи CUDA обчислювальника

Звідки → Куди	Подія	Умова (guard)	Дія
[*] → Constructed	–	–	Створення об’єкта
Constructed → Initializing	Constructor called	–	Запустити послідовність init
CreatingCudaContext → CheckingPtx	–	CudaContext created	Контекст успішно створено

CheckingPtx → LoadingKernel	–	ptx != null	Готувати завантаження ядра
CheckingPtx → [*]	–	ptx == null	Завершити підстан (очікування PTX або режим без GPU)
LoadingKernel → [*]	–	Kernel loaded	Модуль і функцію ядра завантажено
Initializing → Ready	–	Initialization complete	Готовий до виконання
Initializing → Failed	–	Initialization failed	Лог помилки, зупинка init
Ready → Evaluating	Evaluate	–	Старт обчислення батчу
Evaluating → Ready	–	Evaluation complete	Повернення після оцінки
CheckingEmpty → [*]	–	foodSources.Length == 0	Ранній вихід без обчислень
CheckingEmpty → UpdatingCount	–	foodSources.Length > 0	Продовжити
UpdatingCount → StartingTask	–	Evaluation count updated	Запустити Task
StartingTask → EvaluatingInternal	–	Task started	Перейти до GPU-етапів
EvaluatingInternal → SettingNectarValues	–	Evaluation complete	Заповнити r[] у джерелах
Ready → Disposing	Dispose() called	–	Почати утилізацію
DisposingBase → DisposingCudaContext	–	–	Звільнення поетапно

DisposingCudaContext → [*]	–	–	Завершити композит
Disposing → Disposed	–	Disposed	Кінцевий стан
Failed → [*]	–	Initialization failed	Завершення з помилкою

3.2. UML діаграми програмного забезпечення

Система дотримується чіткого та логічного потоку даних, реалізуючи сучасні патерни паралельних обчислень [66, 82, 86], що відображено на діаграмі послідовності на рис. 3.6:

1. Введення користувача. Користувач конфігурує задачу оптимізації через інтерфейс (див. рис. 3.8–3.9): параметри АБК, межі інтервалів, режим виконання (CPU/CPU-паралельно/CUDA), критерії зупинки.
2. Перевірка параметрів і підготовка даних. Основна бібліотека валідує конфігурацію, імпортує дані (CSV/JSON), нормалізує їх та формує внутрішні подання.
3. Ініціація оптимізації. Запускається цикл алгоритму бджолиної колонії з вибором стратегії виконання (послідовна, паралельна на CPU або CUDA) відповідно до конфігурації та доступних ресурсів.
4. Обчислення значень цільової функції.
   - У CUDA-режимі бекенд генерує/завантажує спеціалізоване ядро, після чого виконує конвеєр H2D → Kernel → D2H: батч кандидатів передається на GPU, ядро EvaluateNectar обчислює значення функції якості, результати повертаються на хост.
   - У CPU-режимі обчислення виконуються послідовно або багатопотоково на процесорі.
5. Ітераційний перебіг і завершення. АБК виконує фази Employed, Onlooker, Scout. Цикл повторюється до досягнення критерію зупинки.

6. Обробка результатів. Після завершення система відображає найкращі параметри, формує звіт і візуалізацію у вигляду графіка, а також надає можливість експорту результатів (див. рис. 3.10).

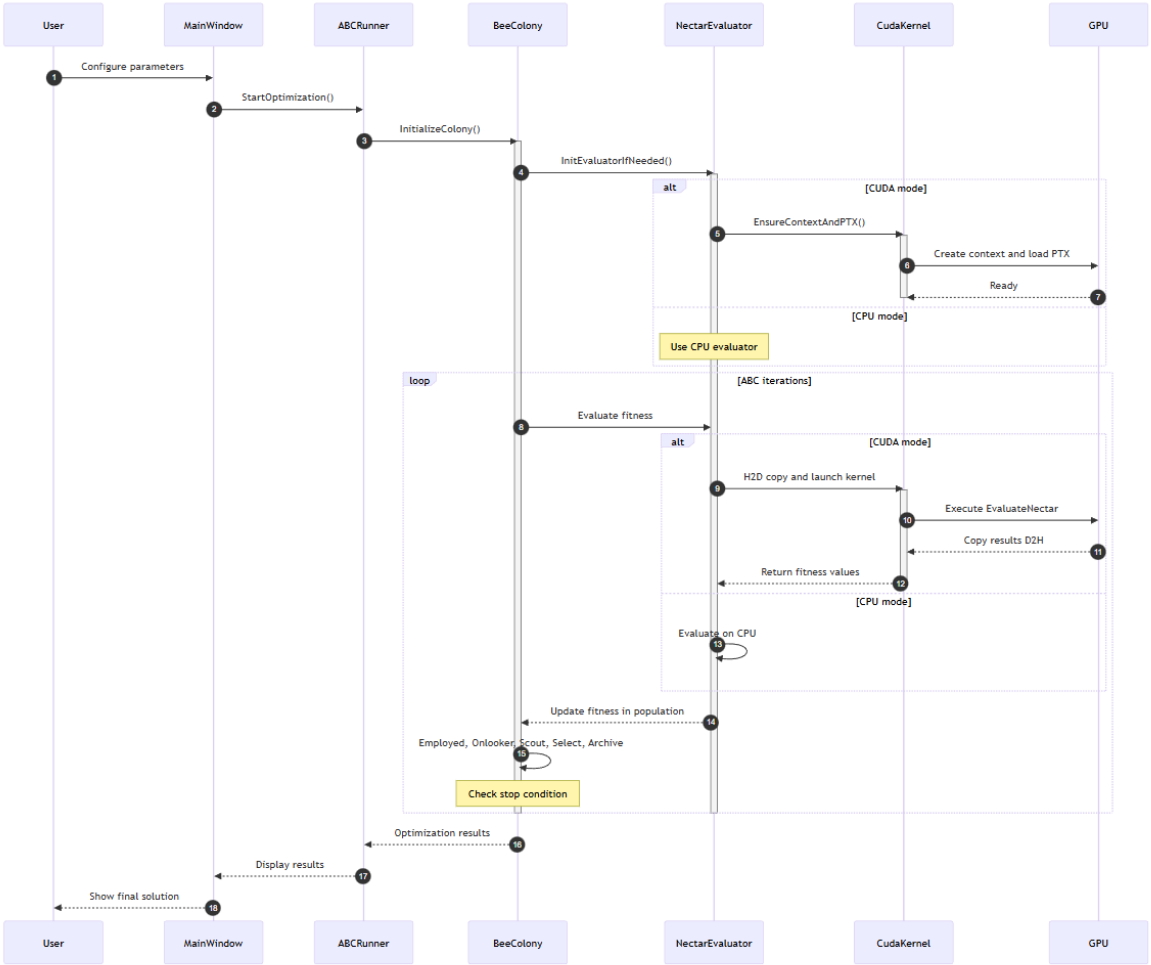


Рис. 3.6. Діаграма послідовності взаємодії компонентів

Діаграма класів на рис 3.7 фіксує внутрішню організацію програмної системи як поєднання керувальної логіки, обчислювальних виконавців і типів даних. Роль інтерфейсного шару виконує MainWindow: він приймає параметри задачі, ініціює запуск і відображає результати. Щоб ізолювати презентаційний рівень від деталей алгоритму, між ними розміщено фасад ABCRunner, який готує потрібну конфігурацію та делегує виконання відповідній колонії – параметричній (BeeColony) або структурній (StructuralBeeColony). Такий поділ спрощує керування сценаріями запуску й унеможливлює проникнення UI-залежностей у алгоритмічне ядро.

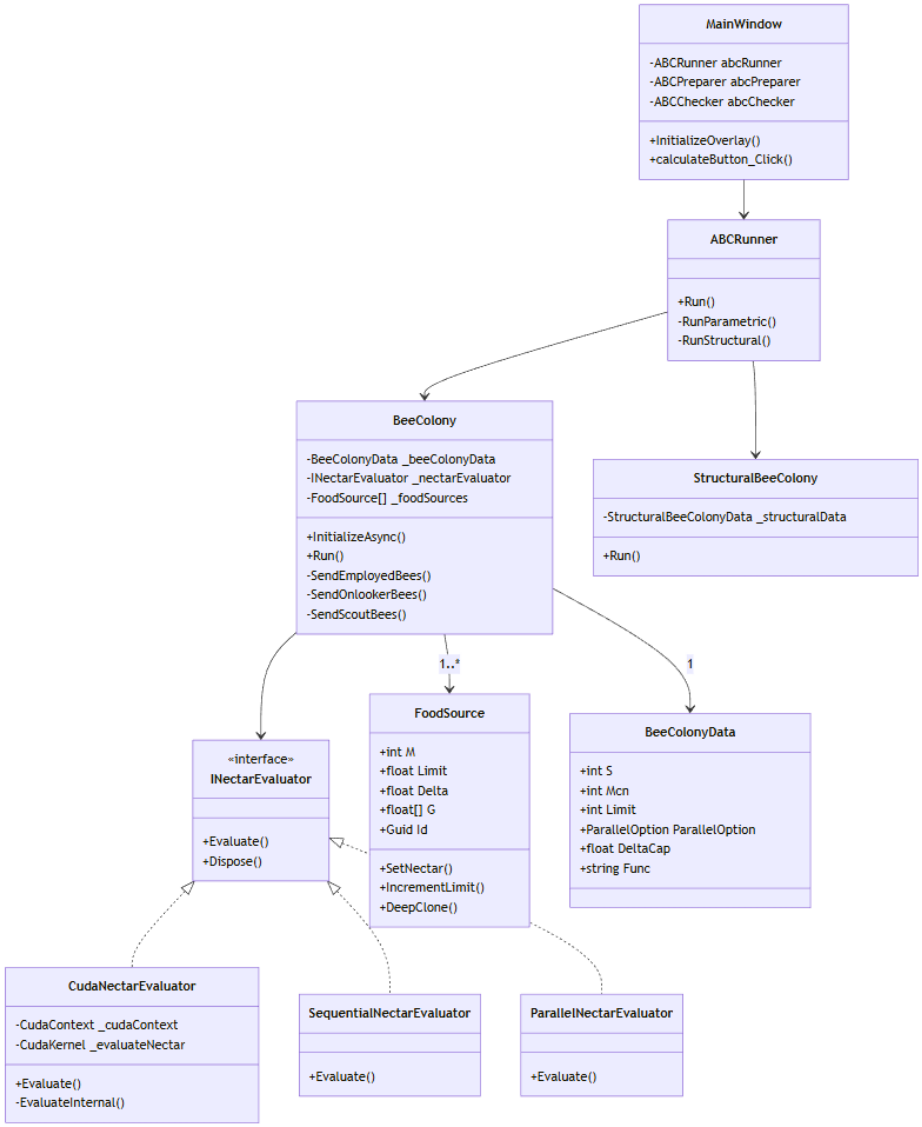


Рис. 3.7. Діаграма класів програмного забезпечення

Алгоритмічний центр системи – BeeColony. Саме тут реалізовано життєвий цикл алгоритму бджолиної колонії: ініціалізацію популяції, фази employed/onlooker/scout, відбір і перевірку критеріїв зупинки. Колонія працює з конфігурацією BeeColonyData та колекцією кандидатів FoodSource, підтримуючи інваріанти коректності (узгоджені межі параметрів, розмір популяції, ліміти стагнації). Визначальним проектним рішенням є інверсія залежностей: для обчислення значень цільової функції колонія звертається не до конкретної реалізації, а до абстракції INectarEvaluator. Це дозволяє підмінювати спосіб обчислень без змін у логіці ABC та легко порівнювати CPU і GPU-режими.

Обчислювальний шар представлено трьома реалізаціями інтерфейсу INectarEvaluator. CudaNectarEvaluator інкапсулює взаємодію з GPU (контекст, завантаження/виклик ядра EvaluateNectar, керування пам’яттю) і забезпечує масово-паралельну оцінку батчів кандидатів. SequentialNectarEvaluator і ParallelNectarEvaluator виконують еквівалентні розрахунки на процесорі – послідовно або багатопотоково – і слугують як базові та резервні стратегії. Така симетрія інтерфейсів спрощує профілювання, відтворюваність експериментів.

Модель кандидата параметрів зосереджено в FoodSource: тут зберігається вектор G, службові лічильники та допоміжні операції оновлення «нектару». Винос цих даних у окремий тип забезпечує чистоту API колонії та полегшує глибоке копіювання й паралельну обробку. Конфігураційні параметри алгоритму згруповано в BeeColonyData, що уможливлює валідацію налаштувань і їх повторне використання в різних запусках без зміни коду ядра.

Узагальнюючи, архітектура спрямована на модульність і розширюваність: UI взаємодіє лише з фасадом, алгоритм – лише з абстракцією обчислювача, а конкретні бекенди приховані за чітким контрактом. Це рішення одночасно підсилює тестованість, продуктивність (GPU-прискорення без втручання в керувальну логіку) та відтворюваність (сталі інтерфейси даних і прозоре керування конфігураціями). Така організація класів узгоджується з компонентною моделю і демонструє чисту межу відповідальностей між презентацією, оркестрацією і виконанням обчислень.

3.3. Особливості інтерфейсу користувача

Інтерфейс користувача реалізовано засобами Windows Forms (4), що забезпечує знайоме та чуйне середовище взаємодії для кінцевого користувача. Інтерфейс проєктувався як окремий шар, розв’язаний від алгоритмічної логіки, завдяки чому еволюція методів ідентифікації не потребує змін у UI; такий поділ підвищує супровідність і відкриває можливості для майбутніх інтерфейсів (веб або кросплатформні рішення).

У системі підтримано керування проєктами: користувач може зберігати та завантажувати повні стани оптимізації (параметри, дані, результати) для відтворюваності експериментів і спільної роботи. Налаштування параметрів відбувається через валідувані форми вводу; введення перевіряється на повноту, діапазони та сумісність, що запобігає запуску некоректних задач рис.3.7 і рис.3.8.

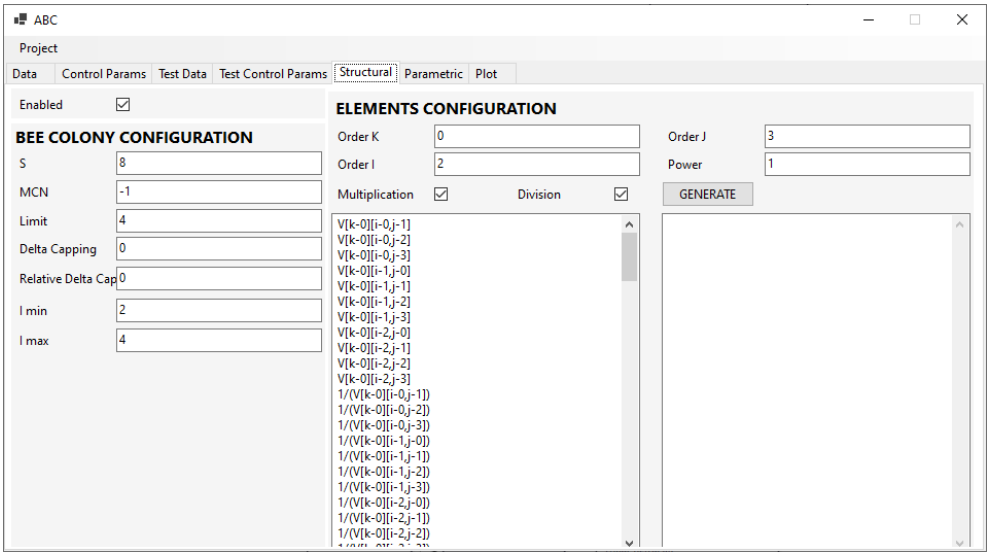


Рис. 3.8 Ввід параметрів для структурної ідентифікації.

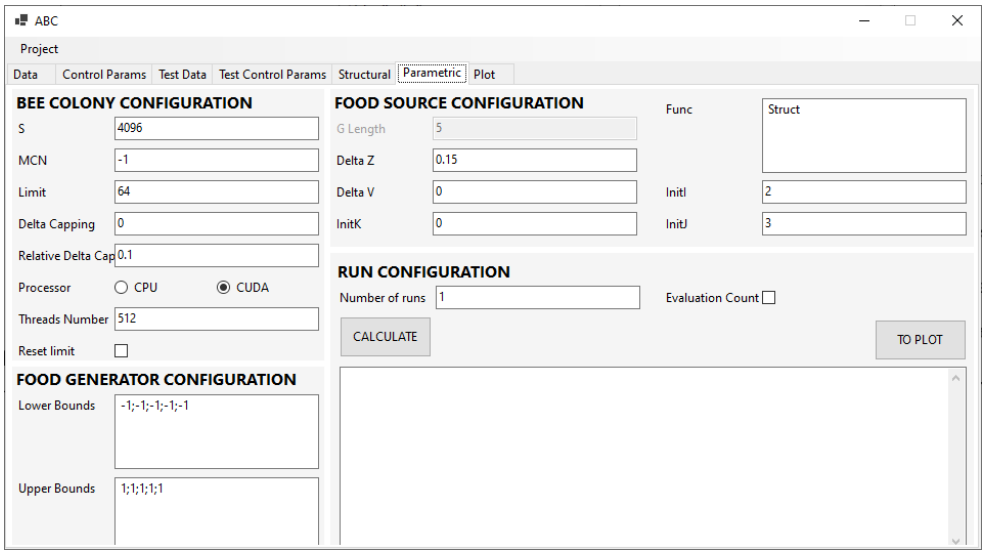
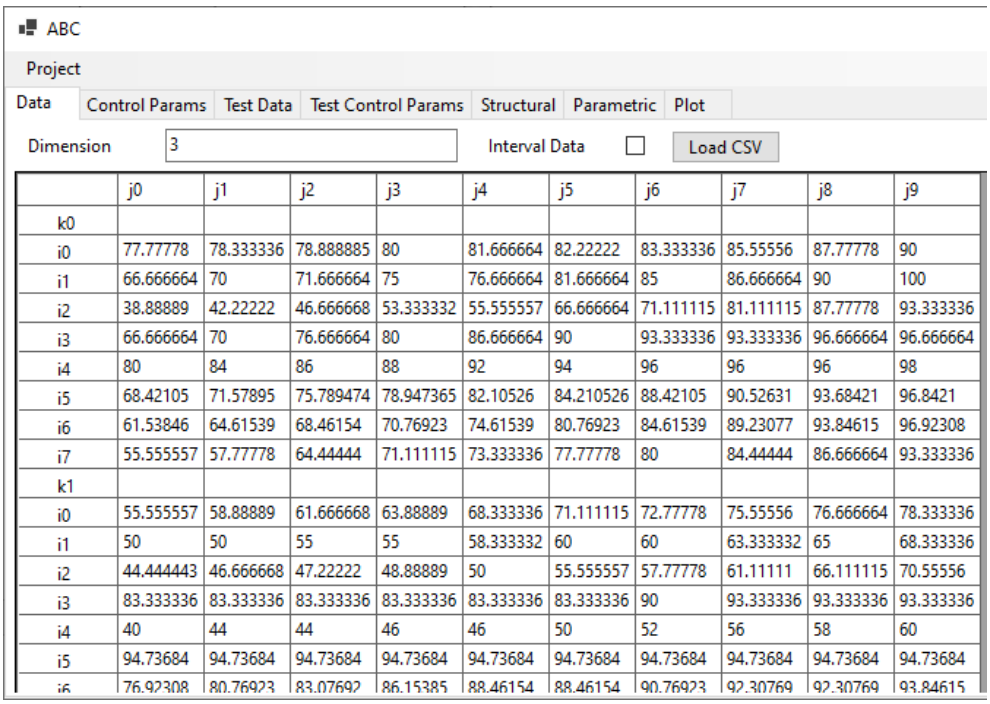


Рис. 3.9 Ввід параметрів для параметричної ідентифікації.

Імпорт/експорт даних підтримує формати CSV і JSON, що полегшує інтеграцію з зовнішніми інструментами та наборами даних; приклад відображення таблиці з імпортованими даними наведено на рис. 3.9.



	j0	j1	j2	j3	j4	j5	j6	j7	j8	j9
k0										
i0	77.77778	78.333336	78.888885	80	81.666664	82.22222	83.333336	85.55556	87.77778	90
i1	66.666664	70	71.666664	75	76.666664	81.666664	85	86.666664	90	100
i2	38.88889	42.22222	46.666668	53.333332	55.555557	66.666664	71.111115	81.111115	87.77778	93.333336
i3	66.666664	70	76.666664	80	86.666664	90	93.333336	93.333336	96.666664	96.666664
i4	80	84	86	88	92	94	96	96	96	98
i5	68.42105	71.57895	75.789474	78.947365	82.10526	84.210526	88.42105	90.52631	93.68421	96.8421
i6	61.53846	64.61539	68.46154	70.76923	74.61539	80.76923	84.61539	89.23077	93.84615	96.92308
i7	55.555557	57.77778	64.44444	71.111115	73.333336	77.77778	80	84.44444	86.666664	93.333336
k1										
i0	55.555557	58.88889	61.666668	63.88889	68.333336	71.111115	72.77778	75.55556	76.666664	78.333336
i1	50	50	55	55	58.333332	60	60	63.333332	65	68.333336
i2	44.444443	46.666668	47.22222	48.88889	50	55.555557	57.77778	61.11111	66.111115	70.55556
i3	83.333336	83.333336	83.333336	83.333336	83.333336	83.333336	90	93.333336	93.333336	93.333336
i4	40	44	44	46	46	50	52	56	58	60
i5	94.73684	94.73684	94.73684	94.73684	94.73684	94.73684	94.73684	94.73684	94.73684	94.73684
i6	76.92308	80.76923	83.07692	86.15385	88.46154	88.46154	90.76923	92.30769	92.30769	93.84615

Рис. 3.10 Табличне подання даних, імпортованих із CSV.

Для візуалізації в реальному часі передбачено графіки та табличні подання, які дозволяють стежити за перебігом ідентифікації та аналізувати проміжні/підсумкові результати (рис. 3.11). Під час тривалих обчислень користувач отримує зворотний зв’язок щодо прогресу – накладні панелі (overlay), індикатори стану та повідомлення про етапи виконання (рис. 3.12).

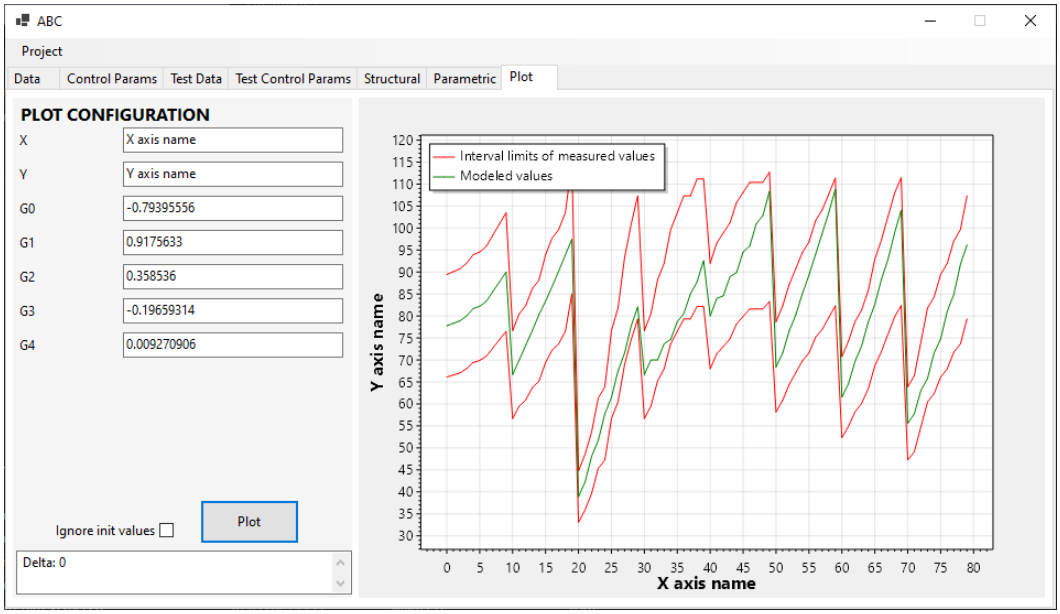


Рис. 3.11. Графік із результатами ідентифікації (візуалізація прогресу та підсумків).

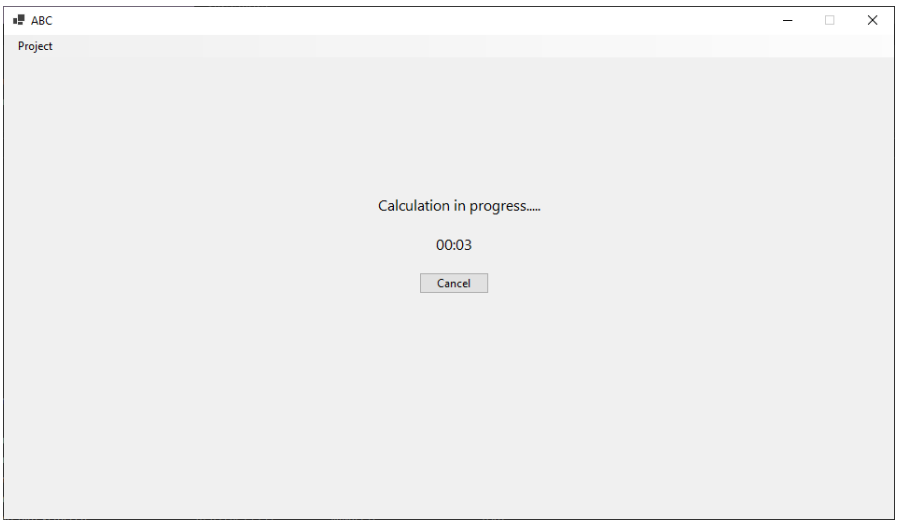


Рис. 3.12. Накладка (overlay) процесу виконання ідентифікації.

З огляду на вимоги надійності й зручності, у підсистемі UI додатково реалізовано такі аспекти. По-перше, асинхронність і відмова від блокування головного потоку: довгі обчислення виконуються в окремих задачах з можливістю коректного скасування, а елементи керування залишаються інтерактивними. По-друге, збереження стану (останній відкритий проєкт, вибрані параметри, макет вікон/панелей) для швидкого повернення до роботи. По-третє, локалізація та узгодженість з українськими налаштуваннями (десятковий роздільник, формати дат/чисел, шрифти), що мінімізує помилки введення та підвищує читабельність. Окрему увагу приділено дружності до помилок: валідаційні повідомлення пояснюють причину відхилення даних і пропонують виправлення, а журнали подій спрощують аналіз збоїв та профілювання. Нарешті, інтерфейс підтримує експорт результатів і звітів (графіки, таблиці, конфігураційні JSON-файли) для документування експериментів і порівняльного аналізу.

Архітектурно інтерфейс наслідує підхід розділення відповідальностей: MainWindow взаємодіє лише з фасадом запуску (ABCRunner) і допоміжними сервісами підготовки/перевірки параметрів; логіка алгоритму, обчислювачі (CPU/паралельні/CUDA) та моделі даних інкапсульовані у внутрішніх шарах. Зв’язки між компонентами UI та іншими рівнями показано на діаграмі класів (рис. 3.5), що ілюструє слабке зчеплення та чіткі контракти взаємодії.

3.4. Дослідження ефективності алгоритмів бджолиної колонії із застосуванням технології NVIDIA CUDA

Порівняння ефективності технологій розпаралелення обчислювальних процесів проведемо на прикладах.

Приклад 1.

Розглянемо параметричну ідентифікацію математичної моделі процесу поширення концентрації окису вуглецю на прямолінійній ділянці вулиці внаслідок рівномірного руху транспортного потоку з постійною потужністю викидів. Вказаний процес описано різницевим рівнянням у такому загальному вигляді:

$$v_k = g_1 + g_2 * v_{k-1} + g_3 * v_{k-2}, k = 2, \dots, K$$

Для ідентифікації для ідентифікації параметрів цієї моделі отримано концентрації окису вуглецю з відносною похибкою 5%. Вимірювання проводилися перпендикулярно до дороги на відстані від 0 до 100 метрів кожні10 метрів [39, 43].

За початкові умови покладемо інтервальні оцінки з відхиленнями у межах +-2.2% виміряної концентрації окису вуглецю. В результаті застосування вищеописаного методу параметричної ідентифікації моделі, отримали:

$$\hat{v}_k = 0{,}15330651 + 0{,}7754277 * \hat{v}_{k-1} + 0{,}09512973 * \hat{v}_{k-2}$$

Вимірювання часу виконання програми алгоритму проводилися на персональному комп’ютері з такими характеристиками: 4 ядерний CPU: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz; RAM: 16Gb Kingston DDR3 SDRAM, GPU: NVIDIA GeForce GTX 960 4096 MB GDDR5

Загалом, проведено по 100 експериментів для таких початкових параметрів алгоритму: S = 64, 1024, 4096, 16384, MCN – до знаходження розв’язку, LIMIT = S/2.

Результати обчислення часової складності виконання алгоритму наведено в таблиці 3.7.

Таблиця 3.7 Порівняльна характеристика реалізації алгоритму бджолиної колонії на CPU і GPU

S, кількість бджіл	Час паралельного алгоритму на CPU , мс	Середня кількість поколінь для знаходження розв'язку паралельного алгоритму на CPU, MCN	Час паралельного алгоритму на GPU, мс	Середня кількість поколінь для знаходження розв'язку паралельного алгоритму на GPU, MCN	Коефіцієнт	Коефіцієнт з урахуванням кількості ітерацій
64	4,2	48,08	156,2	48,34	0,03	0,03
1024	44,5	8,5	72,9	8	0,61	0,57
4096	42,72	5,07	84,9	5,7	0,50	0,57
16384	149,7	2,4	115,5	2,5	1,3	1,35

Приклад 2.

Розглянемо приклад застосування запропонованого методу до моделювання розподілу концентрації діоксиду азоту, який спричинено шкідливими викидами у вихлопних газах автотранспорту, без урахування його поширення. Для отримання експериментальних даних вимірювання проводили на висоті h=1,5 метра від дорожнього полотна на ділянці розміром 32.4328 на 32.4328 метра із дискретизацією заданої ділянки по 20 точок для кожної координати, враховуючи відносну похибку вимірювання 20%.

Вказаний процес описано різницевим рівнянням у такому загальному вигляді [35, 51]:

$$v_{i,j} = g_1 + g_2 * v_{i,j-1} + g_3 * v_{i-1,j} + g_4 * v_{i-1,j-1} + g_5 * v_{i,j-2} + g_6 * v_{i,j-3}, i = 1, \dots, I, j = 3, \dots, J$$

За початкові умови покладемо інтервальні оцінки з відхиленнями у межах +-5% виміряної концентрації діоксиду азоту, перший стовпець і 3 перші рядки. В результаті методу параметричної ідентифікації ІДМ, отримали таку модель:

$$\hat{v}_{i,j} = 0{,}94892436 + 0{,}61438215 * \hat{v}_{i,j-1} - 0{,}34142834 * \hat{v}_{i-1,j} + 0{,}31097192 * \hat{v}_{i-1,j-1} - 0{,}7126774 * \hat{v}_{i,j-2} - 0{,}31465775 * \hat{v}_{i,j-3}$$

Для цього випадку проведено 10 експериментів для заданих параметрів алгоритму: S = 32, MCN – до знаходження розв’язку, LIMIT = S/2.

Результати обчислення часової складності виконання алгоритму для цієї задачі з розпаралеленням обчислювальних потоків та без розпаралелення, наведено в таблиці 3.8.

Таблиця 3.8 Порівняльна характеристика послідовної і паралельної реалізації алгоритму бджолиної колонії

S, кількість бджіл	Час паралельного алгоритму на CPU, мс	Середня кількість поколінь для знаходження розв’язку паралельного алгоритм	Час паралельного алгоритму на GPU, мс	Середня кількість поколінь для знаходження розв’язку паралельного алгоритм	Коефіцієнт	Коефіцієнт з урахуванням кількості ітерацій

		у на CPU, MCN		у на GPU, MCN		
32	16849,8	5351,6	6270,2	2633,3	2,687282702	1,322300161
256	15624,2	632,2	2794,1	975,9	5,591854264	8,631905373
1024	46323	467	1668	482	27,77158273	28,66360359
4096	16150	43	508	61	31,79133858	45,09934078

Як бачимо з обох наведених прикладів, застосуваня технології CUDA для розпаралелення обчислювальної задачі ідентифікації параметрів інтервальних дискретних моделей у вигляді різницевих рівнянь дає вищу ефективність у випадку збільшення кількості обчислювальних агентів (бджіл у рої) в обчислювальному алгоритмі. Також в процесі досліджень для цих прикладів було визначено оптимальну кількість агентів, як це проілюстровано на рис. 3.13.

В таблиці 3.9 показано залежність часу обчислення від обраної кількості бджіл. Початкові умови збережені з попереднього експерименту, LIMIT = S/8

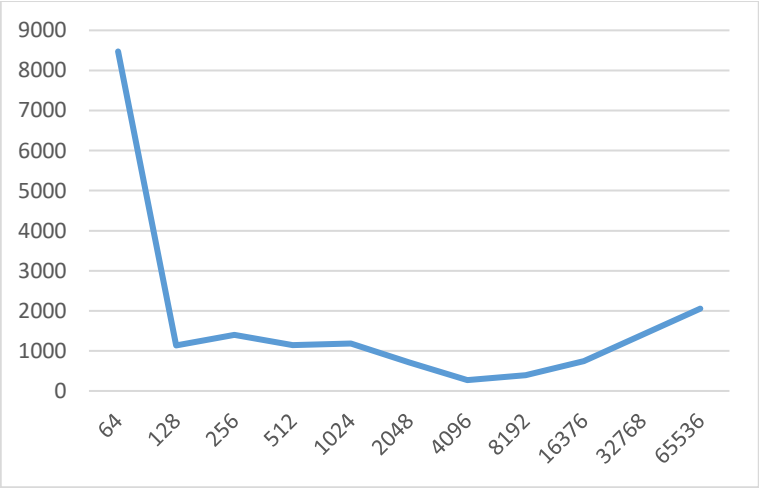


Рис.3.13. Залежність часової складності від кількості обчислювальних агентів

Таблиця 3.9 Залежність часу обчислення від розміру популяції

S, кількість бджіл	Limit = S / 8	Середній час паралельного алгоритму на GPU, мс	Середня кількість поколінь для знаходження розв'язку паралельного алгоритму на GPU, MCN	Середня кількість обчислення функції мети паралельного алгоритму на GPU, MCN
32	4	35009,5	13724,8	881 314
64	8	8473,4	3173,6	407 750
128	16	1132,6	389,4	100 065
256	32	1397,5	447,4	229 665
512	64	1144,6	356,1	365 209
1024	128	1185,8	330,6	677 775
2048	256	713,3	62,3	256 858
4096	512	270,6	28,6	238 343
8192	1024	392,4	27,6	460 336
16376	2048	746,6	21,6	723 787
32768	4096	1398,8	21,1	1 415 548
65536	8192	2057,8	13,6	1 848 094

Як бачимо, в даних випадках оптимальна кількість обчислювальних агентів 4096, оскільки при цьому досягається мінімальна часова складність. Також бачимо, що оптимальна кількість обчислювальних агентів із застосуванням технології CUDA становить від 128 до 16376. Подальше збільшення кількості обчислювальних агентів призводить до втрати досягнутої ефективності. також в процесі експериментальних досліджень для цих прикладів було визначено оптимальну кількість обчислювальних агентів.

Висновки до розділу 3

1. Подальшого розвитку набула агентно-орієнтована архітектура програмної системи для ідентифікації інтервальних моделей систем, що поєднує об’єктно-орієнтовану структуру компонентів із модульною організацією обчислювальних ядер CUDA та забезпечує гнучкість, масштабованість, використання GPU-ресурсів, що у сукупності при реалізації обчислювальних процедур ідентифікації інтервальних моделей систем знижує їх часову складність.

2. Розроблено UML-діаграми, що формалізують статичну структуру і динаміку системи, компонентні та класові моделі фіксують межі відповідальності модулів і ключові колаборації, діаграми станів описуютбь життєві цикли контролера АБК і CUDA-виконавця з явними точками відновлення після збоїв, діаграма послідовностей відображає конвеєр пакетного оцінювання та місця накладання операцій.

3. Розроблено комп’ютерне середовище для математичного моделювання систем на основі аналізу інтервальних даних, яке на відміну від існуючих, імплементує інтерпретатор базисних функцій для побудови моделей та об’єднує паралельне функціонування програмних агентів і їх динамічну компіляцію. Комп’ютерне середовище реалізоване з графічним інтерфейсом, в якій підтримано керування проєктами, імпорт/експорт CSV/JSON, валідацію параметрів, візуалізацію перебігу пошуку та результатів у реальному часі, профілювання. Відокремлення графічного інтерфейсу від обчислювальної бібліотеки спростило супровід і забезпечило можливість подальшого переходу до кросплатформних інтерфейсів без змін алгоритмічної частини.

4. Проведено дослідження часової складності розроблених методів і програмних засобів. Показано, що в залежності від складності моделі і розмірності експериментальних даних, GPU-реалізація забезпечує скорочення часу ітерації алгоритму бджолиної колонії порівняно з CPU-реалізацією. На прикладах побудови інтервальних моделей для екологічного моніторингу встановлено закономірність, що для моделей малої розмірності (2-3 коефіцієнти) застосування запропонованої технології та обчислювальних методів є нераціональним. При зростанні розмірності інтервальної моделі до 10 коефіцієнтів спостерігається

зменшення часової складності застосування запропонованої технології та обчислювальних методів в понад 45 разів. Встановлено, що чим вища розмірність задачі, тим вища ефективність застосування запропонованої технології та обчислювальних методів.

РОЗДІЛ 4. ПРИКЛАДНІ АСПЕКТИ ПРОГРАМНОЇ СИСТЕМИ

В даному розділі наведено застосування розробленої програмної системи ідентифікації інтервальних моделей систем для прикладних задач екологічного моніторингу, оцінювання достовірності контенту та реабілітації пацієнтів з порушенням рухливості верхніх кінцівок.

В першому підрозділі розглянуто прикладні задачі екологічного моделювання шкідливих викидів автотранспорту, а саме побудова одновимірної моделі поширення концентрації окису вуглецю перпендикулярно до дорожнього полотна і побудова двовимірної моделі розподілу концентрації діоксиду азоту, без урахування його поширення на квадратній ділянці на висоті 1,5 над дорожнім полотном.

В другому підрозділі розглянуто моделювання портрету користувача соціальної новинної мережі. Резонансні теми можуть викликати вибухові сплески активності. У такому випадку зростає ризик поширення фейків чи маніпулятивного контенту через підвищену емоційність. У підсумку, особливістю таких спільнот є те, що їхній «портрет» сильно змінюється залежно від характеру новин і контексту, в якому вони з’являються. На основі даної особливості розглянуто побудову статичної інтервальної моделі оцінювання достовірності контенту, який розміщується в зазначеній мережі.

В третьому підрозділі розглянуто процес реабілітації пацієнтів з порушенням рухливості верхніх кінцівок. Пацієнт, в якого є проблеми з рухами верхніх кінцівок, проходить реабілітацію, під час якої фізіотерапевти вимірюють ключові кути згину суглобів верхніх кінцівок. На основі цих даних розглянуто побудова моделі динаміки відновлення кутів рухливості суглобів верхніх кінцівок.

Результати досліджень цього розділу опубліковано автором у працях [9, 50].

4.1. Прикладні задачі екологічного моделювання шкідливих викидів транспортного потоку

Розглянемо застосування розробленого методу для екологічного моделювання шкідливих викидів автотранспорту. Система «транспорт–довкілля» є суттєво невизначеною: рівень викидів залежить від типу двигуна, стану двигуна, палива, стилю водіння, швидкості потоку транспорту, метеорологічних умов, топографії вулиць тощо. Навіть у межах одного маршруту середні концентрації оксидів азоту, твердих частинок чи CO можуть змінюватися в рази залежно від локального режиму руху.

Першою прикладною задачею розглянемо модель поширення концентрації окису вуглецю CO в однорідному середовищі внаслідок рівномірного руху транспорту з постійною потужністю викидів.

Для ідентифікації проведено вимірювання концентрації окису вуглицю з відносною похибкою 5% перпендикулярно до дороги на відстані від 0 до 100 метрів із кроком в 10 метрів [39, 43]. Дані експерименту наведено в таблиці 4.1.

Таблиця 4.1 Інтервальні дані вимірювання концентрації окису вуглецю

Дискрета, k	Виміряна концентрація CO, z_k	Нижня межа виміряної концентрації CO, z_k^-	Верхня межа виміряної концентрації CO, z_k^+
k0	55	52,25	57,75
k1	47	44,65	49,35
k2	43	40,85	45,15
k3	37	35,15	38,85
k4	32	30,4	33,6
k5	30	28,5	31,5
k6	26	24,7	27,3
k7	23	21,85	24,15
k8	20	19	21
k9	18	17,1	18,9
k10	16	15,2	16,8

Для структурної ідентифікації було обрану наступну конфігурацію:

- Множина структурних елементів F: V[k-1], V[k-2], V[k-3], V[k-1]*V[k-1], V[k-1]*V[k-2], V[k-1]*V[k-3], V[k-2]*V[k-2], V[k-2]*V[k-3], V[k-3]*V[k-3];
- Кількість структурних елементів моделі: $[I_{min}, I_{max}]$=[2,3];
- Популяція: S = 8;
- Вичерпність: LIMIT = 4

Для параметричної ідентифікації було обрану наступну конфігурацію:

- Популяція: S = 8;
- Вичерпність: LIMIT = 4;
- Δ_δ = 0.1;
- Очислювальна схема: CPU, 4 потоки
- Δ_z = 0.05 – відносна похибка вимірювання 5%, яка використовується для побудови інтервальних значень z, отриманих експериментально;
- Δ_v = 0.03 – значення, яке використовується для побудови інтервальних значень розраховуваних даних.

В результаті ідентифікації отримали модель, зображену на рис.4.1, що описує поширення концентрації окису вуглецю CO: $v_k = g_0 + g_1 * v_{k-1} + g_2 * v_{k-3}$, g_0 = 0.6396168, g_1 = 0.26399562, g_2 = 0.46799546

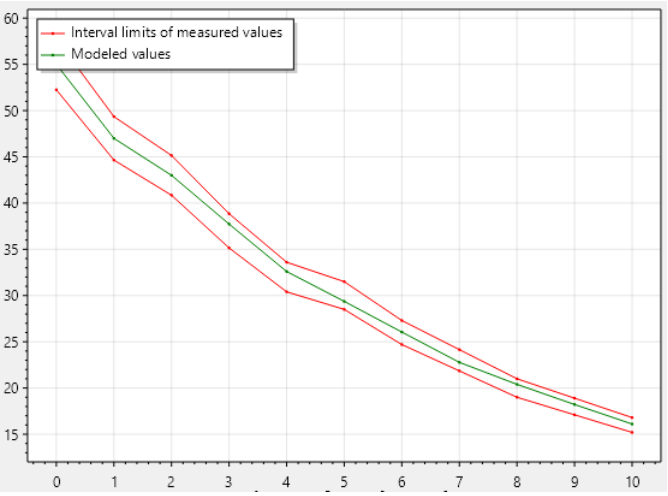


Рис.4.1. Візуалізація моделі поширення концентрації CO

Наступним розглянемо приклад застосування запропонованого методу до моделювання розподілу концентрації діоксиду азоту, який спричинено шкідливими викидами у вихлопних газах автотранспорту, без урахування його поширення [35, 51]. Для отримання експериментальних даних вимірювання проводили на висоті h=1,5 метра від дорожнього полотна на ділянці розміром 32.4328 на 32.4328 метра

із дискретизацією заданої ділянки по 20 точок для кожної координати, враховуючи відносну похибку вимірювання 20%.

Таблиця 4.2 Вимірювання концентрацій діоксиду азоту NO2 на заданих дискретах

i/j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	45,134	45,604	45,134	44,663	44,663	44,193	44,193	44,663	45,134	45,604	45,604	45,134	47,014	47,014	47,01	47,484	47,484	48,895	48,425	47,484
1	48,425	48,425	48,895	49,835	50,305	49,835	49,365	49,365	49,365	49,365	49,835	49,835	50,305	49,835	49,84	50,775	49,835	48,895	48,895	48,425
2	49,835	49,365	49,365	50,775	49,835	49,835	49,835	49,365	48,895	49,835	50,305	50,775	51,245	51,245	51,72	52,186	51,245	50,775	51,245	50,775
3	48,895	47,484	47,014	47,014	47,014	46,544	47,954	47,014	46,074	47,484	47,954	47,954	48,425	47,954	47,95	48,895	48,425	47,484	47,484	46,544
4	46,074	46,074	45,604	44,663	45,134	44,663	45,134	44,663	44,193	45,134	44,193	44,193	45,134	44,663	44,19	45,134	45,134	44,663	45,604	45,604
5	46,544	46,544	47,014	47,484	48,895	48,895	48,425	48,895	48,425	49,365	49,365	49,835	51,716	51,716	51,72	52,656	52,656	53,126	53,126	55,007
6	56,887	58,298	58,768	59,708	60,648	61,118	60,648	61,589	61,589	61,589	62,059	61,589	61,118	62,059	61,59	61,589	62,529	62,059	61,589	62,529
7	62,529	63,469	63,469	63,939	65,35	65,35	65,35	65,82	65,82	64,88	64,409	63,469	62,999	62,999	62,06	61,589	62,059	60,648	60,178	61,118
8	60,648	60,178	60,648	60,178	60,648	61,118	60,648	61,589	61,589	61,118	62,059	62,529	61,589	62,059	63	63,469	64,88	65,35	67,23	67,7
9	67,23	68,171	69,581	70,051	70,051	70,991	71,462	72,402	73,342	73,812	75,223	76,163	77,103	78,514	79,92	80,394	81,335	82,275	83,215	84,626
10	85,566	85,566	87,917	88,387	88,857	89,797	92,148	93,088	94,028	96,379	97,319	99,2	99,67	100,14	101,1	102,02	101,08	102,02	101,55	101,55
11	101,08	100,61	100,14	100,14	100,14	99,67	100,61	99,67	98,26	97,789	96,379	95,909	94,969	94,028	94,5	94,028	93,088	92,148	90,737	88,387
12	88,387	86,506	86,036	86,036	85,096	84,155	84,626	83,685	83,215	84,155	82,745	81,335	80,864	79,924	80,39	78,984	78,044	78,514	77,573	76,633
13	77,103	76,633	76,163	76,163	75,693	75,223	75,693	74,753	74,753	74,753	74,282	73,812	74,282	73,342	72,4	71,932	71,462	71,462	70,521	69,581
14	69,581	70,051	69,111	69,111	68,171	67,23	67,23	66,29	66,29	65,35	64,88	65,35	66,76	66,29	67,23	68,171	67,7	67,23	67,23	66,76
15	66,76	66,76	65,82	66,76	65,35	64,88	64,88	64,409	62,999	63,469	63,469	62,529	62,999	62,529	62,53	62,999	62,529	62,059	62,999	62,059
16	61,118	60,648	59,708	60,178	59,708	59,238	60,178	59,708	59,238	59,708	59,238	58,768	58,298	57,827	57,83	57,827	57,357	56,887	57,827	56,887
17	57,357	58,298	57,357	57,357	58,298	58,298	61,118	61,589	62,059	63,939	63,939	64,409	65,35	65,82	65,82	65,82	65,82	65,82	66,76	66,76
18	66,76	67,7	66,76	66,76	66,29	65,82	64,409	64,409	63,469	63,939	63,469	62,059	62,529	61,118	61,12	60,648	61,118	58,768	59,238	58,768
19	58,768	58,298	57,357	56,887	57,357	56,887	56,417	57,827	56,417	55,477	55,477	55,477	55,947	55,477	55,01	54,066	54,066	53,126	52,186	52,186

Для структурної ідентифікації було обрану наступну конфігурацію:

- Множина структурних елементів F: V[i-0,j-1], V[i-0,j-2], V[i-0,j-3], V[i-1,j-0], V[i-1,j-1], V[i-1,j-2], V[i-1,j-3], V[i-0,j-1]*V[i-0,j-1], V[i-0,j-1]*V[i-0,j-2], V[i-0,j-1]*V[i-0,j-3], V[i-0,j-1]*V[i-1,j-0], V[i-0,j-1]*V[i-1,j-1], V[i-0,j-1]*V[i-1,j-2], V[i-0,j-1]*V[i-1,j-3], V[i-0,j-2]*V[i-0,j-2], V[i-0,j-2]*V[i-0,j-3], V[i-0,j-2]*V[i-1,j-0], V[i-0,j-2]*V[i-1,j-1], V[i-0,j-2]*V[i-1,j-2], V[i-0,j-2]*V[i-1,j-3], V[i-0,j-3]*V[i-0,j-3], V[i-0,j-3]*V[i-1,j-0], V[i-0,j-3]*V[i-1,j-1], V[i-0,j-3]*V[i-1,j-2], V[i-0,j-3]*V[i-1,j-3], V[i-1,j-0]*V[i-1,j-0], V[i-1,j-0]*V[i-1,j-1], V[i-1,j-0]*V[i-1,j-2], V[i-1,j-0]*V[i-1,j-3], V[i-1,j-1]*V[i-1,j-1], V[i-1,j-1]*V[i-1,j-2], V[i-1,j-1]*V[i-1,j-3], V[i-1,j-2]*V[i-1,j-2], V[i-1,j-2]*V[i-1,j-3], V[i-1,j-3]*V[i-1,j-3];
- Кількість структурних елементів моделі: $[I_{min}, I_{max}]$=[4,6];
- Популяція: S = 8;
- Вичерпність: LIMIT = 4

Для параметричної ідентифікації було обрану наступну конфігурацію:

- Популяція: S = 4096;
- Вичерпність: LIMIT = 64
- Δ_δ = 0.1;
- Очислювальна схема: $CUDA,$ 512 thread per block
- Δ_z = 0.2 – відносна похибка вимірювання 5%, яка використовується для побудови інтервальних значень z, отриманих експериментально;

- $\Delta_v = 0.05$ – значення, яке використовується для побудови інтервальних значень розраховуваних даних.

В результаті ідентифікації тримали наступну модель, зображену на рис.4.1, що описує поширення концентрації NO2: $v_{i,j} = g_0 + g_1 * v_{i,j-1} + g_2 * v_{i-1,j} + g_3 * v_{i-1,j-1} + g_4 * v_{i,j-2} + g_5 * v_{i,j-3}$, $g_0 = 0.9594967$, $g_1 = 0.91184974$, $g_2 = 0.62336797$, $g_3 = -0.65045875$, $g_4 = 0.08794141$, $g_5 = 0.011207244$.

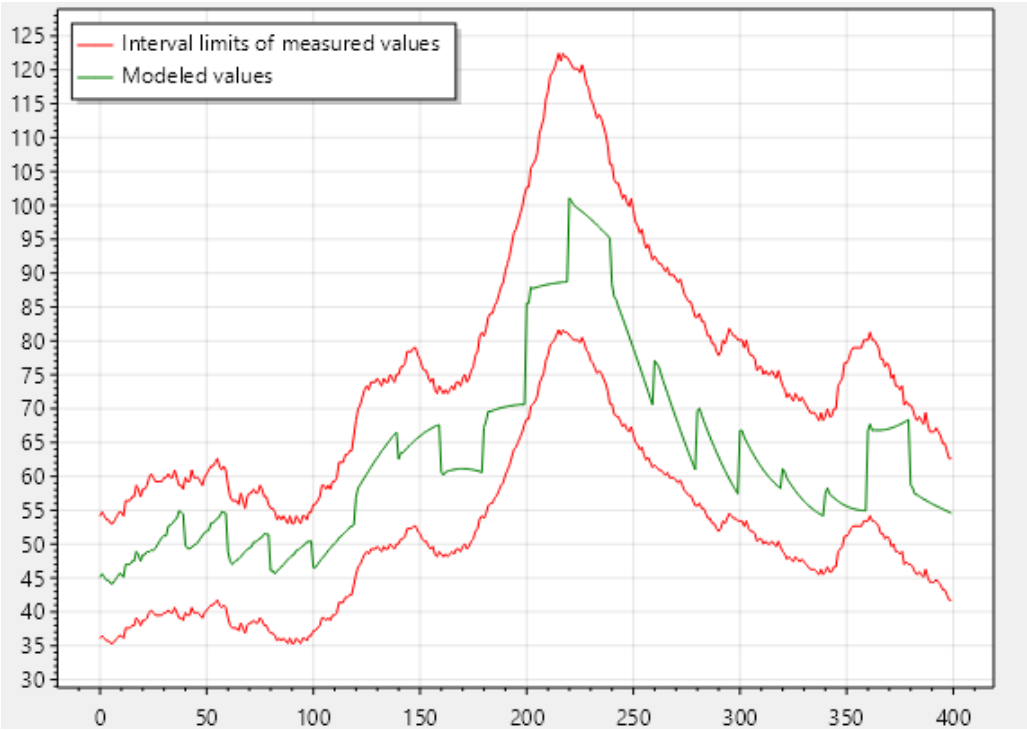


Рис.4.2. Візуалізація моделі поширення концентрації NO2

4.2. Математична модель портрету користувача у соціальній мережі

Розглянемо застосування розробленого методу для моделювання портрету соціальної мережі. Зазначена мережа відноситься до новинної мережі, яка об'єднує спільноту, що орієнтовані на новинний контент. Особливістю цієї мережі є швидка реакція на новий контент. Взаємодія членів спільноти переважно зростає протягом перших кількох годин, після чого рівень взаємодії знижується, якщо новина не викликає довготривалого резонансу. У такій мережі можуть виникати аномалії через резонансні теми, тобто викликати вибухові сплески активності. У такому випадку зростає ризик поширення фейків чи маніпулятивного контенту через підвищену емоційність [108, 109]. Як вже зазначалося, спільноти, орієнтовані на

новинний контент, характеризуються швидкою динамікою взаємодії. Також таким спільнотам притаманні поляризовані реакції та чутливістю до маніпулятивного контенту [125]. Вони мають гетерогенну аудиторію, яка прагне швидкого доступу до актуальної інформації. У підсумку, особливістю таких спільнот є те, що їхній «портрет» сильно змінюється залежно від характеру новин і контексту, в якому вони з'являються. Саме ця особливість може слугувати для розпізнавання фейкового контенту, який переважно викликає неприродній резонанс портрету користувачів.

Для дослідження портрету, зазначеної соціальної мережі, було використано N =20 випадків розміщення контенту різного характеру, та зафіксовано портрет спільноти за допомого таких чинників: x_1, - кількість постів, поширень або лайків, зроблених користувачами протягом перших 10 хвилин після появи контенту; x_2 – динаміка коментарів або реакцій через кожні 10 хвилин (вимірюємо середньою кількістю приросту коментарів); x_3 - кількість унікальних користувачів за пів доби; x_4 – коефіцієнт вірусного поширення контенту. Результати досліджень цієї мережі наведено в таблиці 4.3.

Таблиця 4.3. Результати досліджень контенту в мережі

N контенту	Кількість лайків перші 10 хв	Динаміка поширення (середнє значення)	Кількість унікальних користувачів, що характеризує час поширення новини, (тисяч)	Коефіцієнт вірусного поширення	Ступінь достовірності контенту
i	x_1	x_2	x_3	x_4	$[y_i^-; y_i^+]$
1	52	30,1	2,114	2,5	[0,9; 1]
2	35	24,8	1,878	2,2	[0,95; 1]

3	64	38	2,789	3,4	[0,78;0,91]
4	92	41	2,830	3,2	[0,54;0,71]
5	28	20,5	1,500	1,9	[0,96; 1]
6	60	36,7	2,600	3,1	[0,85; 0,93]
7	75	39,9	2,750	3,3	[0,72; 0,84]
8	100	45	3,000	3,8	[0,50; 0,65]
9	110	50,2	3,300	4	[0,35; 0,52]
10	20	15,5	1,300	1,6	[0,97; 1]
11	40	22,3	1,700	2,1	[0,94; 0,99]
12	88	42	2,900	3,5	[0,60; 0,75]
13	99	47,3	3,200	3,9	[0,42; 0,60]
14	120	52,5	3,450	4,2	[0,25; 0,45]
15	18	14,2	1,200	1,4	[0,98; 1]
16	47	27,5	2,000	2,3	[0,88; 0,94]
17	85	43,1	2,950	3,6	[0,66; 0,80]
18	105	49,5	3,350	4,1	[0,38; 0,55]
19	115	53,7	3,600	4,3	[0,21; 0,40]
20	125	55,8	3,800	4,5	[0,12; 0,28]

Для структурної ідентифікації було обрану наступну конфігурацію:

- Множина структурних елементів F: V[i-0,j-1], V[i-0,j-2], V[i-0,j-3], V[i-0,j-4], 1/(V[i-0,j-1]), 1/(V[i-0,j-2]), 1/(V[i-0,j-3]), 1/(V[i-0,j-4]), V[i-0,j-1]/(V[i-0,j-2]), V[i-0,j-1]/(V[i-0,j-3]), V[i-0,j-1]/(V[i-0,j-4]), V[i-0,j-2]/(V[i-0,j-1]), V[i-0,j-2]/(V[i-0,j-3]), V[i-0,j-2]/(V[i-0,j-4]), V[i-0,j-3]/(V[i-0,j-1]), V[i-0,j-3]/(V[i-0,j-2]), V[i-0,j-3]/(V[i-0,j-4]), V[i-0,j-4]/(V[i-0,j-1]), V[i-0,j-4]/(V[i-0,j-2]), V[i-0,j-4]/(V[i-0,j-3]);
- Кількість структурних елементів моделі: $[I_{min}, I_{max}]$=[2,4];
- Популяція: S = 8;
- Вичерпність: LIMIT = 4

Для параметричної ідентифікації було обрану наступну конфігурацію:

- Популяція: S = 4096;
- Вичерпність: LIMIT = 64
- Δ_δ = 0.1;
- Очислювальна схема: $CUDA,\ 1024$ thread per block
- Δ_z = 0.1 – відносна похибка вимірювання 10%, яка використовується для побудови інтервальних значень z, отриманих експериментально;
- Δ_v = 0 – значення, яке використовується для побудови інтервальних значень розраховуваних даних.

В результаті ідентифікації тримали наступну модель: $v_k = g_0 + g_1 * v_{i,j-1} * v_{i-4,j} + g_2 * v_{i,j-3} / v_{i,j-2}$, g_0=0,491103; g =-0,001786;g_2 =0,048772.

Зважаючи на те що x1 = V[i, j-4], x2 = V[i, j-3], x3 = V[i, j-2], x4 = V[i, j-1] сформуємо відповідну статичну математичну модель:

$$y(\vec{X}) = 0{,}491103 - 0{,}001786 \cdot x_1 \cdot x_4 + 0{,}048772 \cdot x_2/x_3 \quad .$$

Як бачимо, отримана інтервальна модель містить три коефіцієнти та відповідно три структурні елементи. Також бачимо, що математична модель нелінійна відносно чинників, від яких залежить достовірність контенту.

Тепер можемо використовувати отриману інтервальну модель для оцінювання достовірності контенту, який розміщується в зазначеній мережі.

Приклад 1. Розглянемо приклад застосування розробленої математичної моделі для прогнозування достовірності контенту.

В мережі появилася новина «1» .

Кількість лайків через появу новини у перші 10 хв становила x_1 = 28; динаміка (приріст) поширення новини по мережі, зафіксована протягом перших шести годин за кожних 10 хвилин в середньому становила x_2 = 29; кількість унікальних користувачів, що характеризує час поширення новини, за пів доби становила x_3=2 353 учасники; коефіцієнт вірусного поширення складає x_4=2,3.

Користуючись отриманою математичною моделлю (2.28), отримаємо:

$$y(\vec{X}) = 0{,}491103 - 0{,}001786 \cdot 28 \cdot 2{,}3 + 0{,}048772 \cdot 29/2{,}353 = 0{,}98$$

Отриманий результат означає, що новина є достовірною, що повністю відповідає дійсності, виходячи з контексту новини.

Приклад 2. Розглянемо наступний приклад застосування розробленої математичної моделі для прогнозування достовірності контенту.

В мережі появилася новина «2» .

Кількість лайків через появу новини у перші 10 хв становила x_1 =138; динаміка (приріст) поширення новини по мережі, зафіксована протягом перших шести годин за кожних 10 хвилин в середньому становила x_2 = 64, що означає достатньо швидке поширення контенту; кількість унікальних користувачів, за пів доби становила x_3=4 372 учасники, що говорить про короткий час поширення; коефіцієнт вірусного поширення складає x_4=4,2.

Користуючись отриманою математичною моделлю (2.28), отримаємо:

$$y(\vec{X}) = 0{,}491103 - 0{,}001786 \cdot 138 \cdot 4{,}2 + 0{,}048772 \cdot 64/4{,}372 = 0{,}17$$

Отриманий результат означає, що новина є недостовірною, що повністю відповідає дійсності, виходячи з контексту новини.

Отримані результати говорять про те, що розроблена модель може використовуватися для оцінювання достовірності контенту.

4.3. Математичні моделі динаміки відновлення кутів рухливості суглобів верхніх кінцівок

Розглянемо детально процес реабілітації пацієнтів з порушенням рухливості верхніх кінцівок. Пацієнт, в якого є проблеми з рухами верхніх кінцівок, проходить реабілітацію, під час якої важливо автоматично вимірювати кути згину суглобів для оцінки прогресу. Наприклад, якщо у пацієнта після інсульту спостерігаються проблеми з рухами кисті руки, то об’єм рухів в суглобах зап’ястя є обмеженим і людина потребує реабілітації. Фізіотерапевти вимірюють ключові кути згину суглобів верхніх кінцівок:

Для плечових суглобів (вихідне положення – плече вздовж осі тіла):

- згинання 160–180°;
- розгинання 50–60°;
- відведення (абдукція) 180°;

- приведення (аддукція) 30°;
- горизонтальне згинання вперед 130° (вихідне положення – плече відведене горизонтально);
- горизонтальне розгинання у напрямку до спини 45°.

Для розробки інтервальних математичних моделей динаміки відновлення кутів рухливості суглобів верхніх кінцівок були обрані три ключові кути плечового суглоба, які є критичними для оцінки прогресу реабілітації: згинання (160–180°), розгинання (50–60°) та відведення (абдукція, 180°). Ці кути вибрано через їхню поширеність у порушеннях моторики верхніх кінцівок, спричинених травмами чи захворюваннями, та можливість точного вимірювання за допомогою систем на зразок Vidnova. Дослідження проводилося на п'яти пацієнтах з різними діагнозами: Анастасія (25 років, жінка, субакроміальний бурсит правої верхньої кінцівки), Валентина (22 роки, жінка, епікондиліт правого ліктьового суглоба), Сергій (42 роки, чоловік, посттравматичний адгезивний капсуліт правого плечового суглоба), Антон (48 років, чоловік, остеоартроз правого плечового суглоба) та Альберт (20 років, чоловік, перелом правої ключичної та плечової кісток). Кожен пацієнт пройшов по 10 сеансів реабілітації з використанням системи Vidnova, під час яких вимірювалися зазначені кути в реальному часі. Дані експерименту відображені в таблицях 4.4, 4.5 та 4.6.

Для структурної ідентифікації було обрану наступну конфігурацію:

- Множина структурних елементів F: V[i-0,j-1], V[i-0,j-2], V[i-0,j-3], V[i-0,j-4], V[i-0,j-1]*V[i-0,j-1], V[i-0,j-1]*V[i-0,j-2], V[i-0,j-1]*V[i-0,j-3], V[i-0,j-1]*V[i-0,j-4], V[i-0,j-2]*V[i-0,j-2], V[i-0,j-2]*V[i-0,j-3], V[i-0,j-2]*V[i-0,j-4], V[i-0,j-3]*V[i-0,j-3], V[i-0,j-3]*V[i-0,j-4], V[i-0,j-4]*V[i-0,j-4], … , V[i-0,j-1]*V[i-0,j-3]/(V[i-0,j-2]*V[i-0,j-4]), V[i-0,j-1]*V[i-0,j-3]/(V[i-0,j-4]*V[i-0,j-4]), V[i-0,j-1]*V[i-0,j-4]/(V[i-0,j-2]), V[i-0,j-1]*V[i-0,j-4]/(V[i-0,j-3]), V[i-0,j-1]*V[i-0,j-4]/(V[i-0,j-2]*V[i-0,j-2]), V[i-0,j-1]*V[i-0,j-4]/(V[i-0,j-2]*V[i-0,j-3]), V[i-0,j-1]*V[i-0,j-4]/(V[i-0,j-3]*V[i-0,j-3]), V[i-0,j-2]*V[i-0,j-2]/(V[i-0,j-1]), … , V[i-0,j-4]*V[i-0,j-4]/(V[i-0,j-1]*V[i-0,j-1]), V[i-0,j-4]*V[i-0,j-4]/(V[i-0,j-1]*V[i-0,j-2]), V[i-0,j-4]*V[i-0,j-4]/(V[i-0,j-1]*V[i-0,j-3]), V[i-0,j-4]*V[i-0,j-4]/(V[i-0,j-2]*V[i-0,j-2]), V[i-0,j-4]*V[i-0,j-4]/(V[i-0,j-2]*V[i-0,j-3]), V[i-0,j-4]*V[i-0,j-4]/(V[i-0,j-3]*V[i-0,j-3]);

- Кількість структурних елементів моделі: $[I_{min}, I_{max}]$=[2,4];
- Популяція: S = 8;
- Вичерпність: LIMIT = 4

Для параметричної ідентифікації було обрану наступну конфігурацію:

- Популяція: S = 4096;
- Вичерпність: LIMIT = 64
- Δ_δ = 0.1;
- Очислювальна схема: $CUDA,$ 512 thread per block
- Δ_z = 0.1 – відносна похибка вимірювання 10%, яка використовується для побудови інтервальних значень z, отриманих експериментально;
- Δ_v = 0 – значення, яке використовується для побудови інтервальних значень розраховуваних даних.

Таблиця 4.4. Таблиця динаміки відновлення кута згинання (160–180°) верхніх кінцівок

	1	2	3	4	5	6	7	8	9	10
Анастасія	140	141	142	144	147	148	150	154	158	162
Сергій	130	134	144	146	148	150	152	156	168	178
Валентина	60	65	71	78	87	95	105	116	128	146
Антон	100	106	111	115	123	128	131	136	138	141
Альберт	90	100	111	116	124	129	132	143	144	146

В результаті ідентифікації тримали наступну модель: $v_{i,j} = g_0 + g_1 * v_{i,j-1} + g_2 * v_{i,j-3} + g_3 * v_{i,j-1} * v_{i,j-2}$, $g_0 =$ 0.5566865, $g_1 =$ 0.97192764, $g_2 =$ 0.35143995, $g_3 = = -0.0019254182$.

Оскільки модель є незалежною між пацієнтами, а тільки від днів спостереження перепишемо модель в наступному вигляді, де k – день спостереження:

$$v_k = 0.5566865 + 0.97192764 * v_{k-1} + 0.35143995 * v_{k-3} - 0.0019254182 * v_{k-1} * v_{k-2}$$

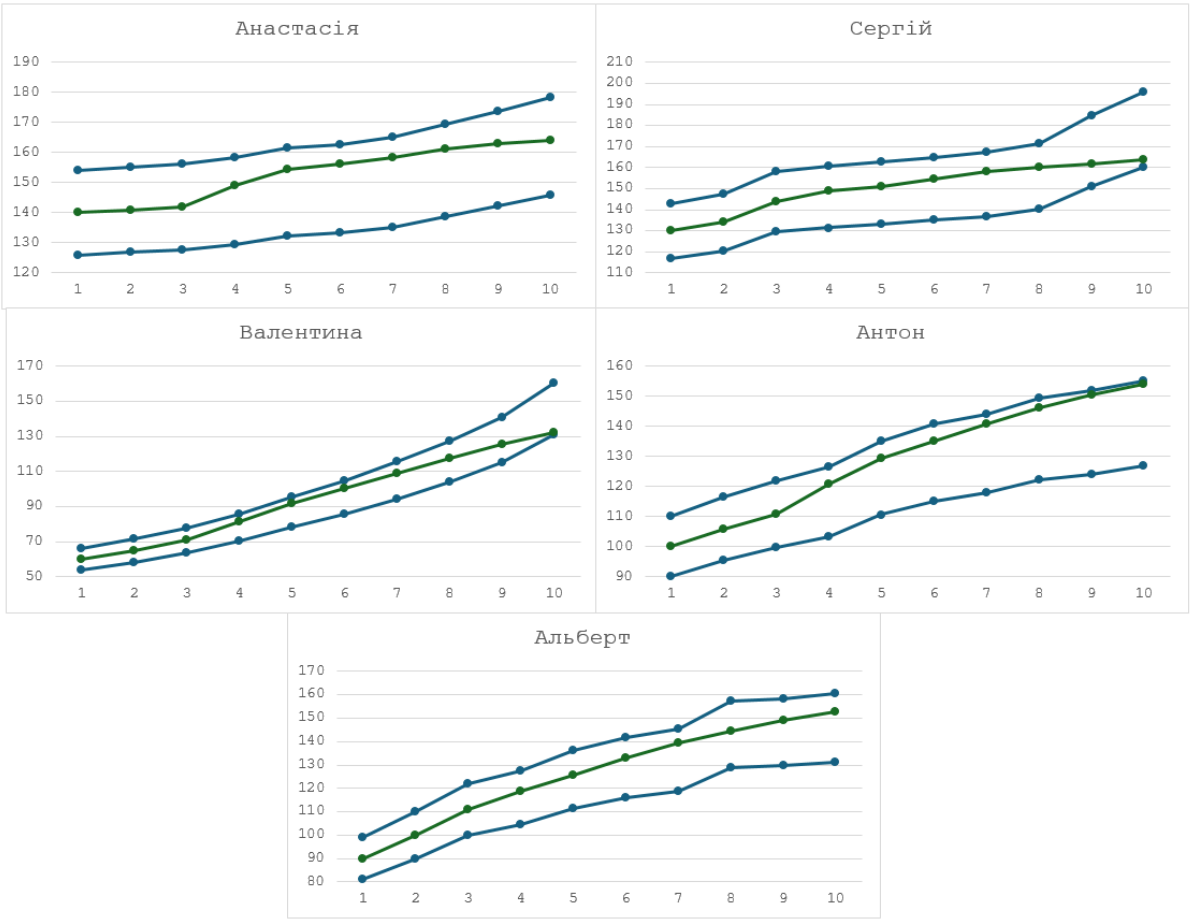


Рис.4.3. Візуалізація моделі динаміки відновлення кута згинання (160–180°) верхніх кінцівок для кожного пацієнта

Таблиця 4.5. Таблиця динаміки відновлення кута розгинання (50–60°) верхніх кінцівок

	1	2	3	4	5	6	7	8	9	10
Анастасія	40	42	43	45	46	49	51	52	54	60
Сергій	30	38	39	40	42	44	48	51	56	58
Валентина	25	28	30	33	38	41	44	46	50	54
Антон	30	30	33	33	35	36	36	38	39	41
Альберт	20	23	27	29	29	33	34	36	38	41

В результаті ідентифікації тримали наступну модель: $v_{i,j} = g_0 + g_1 * v_{i,j-1} * v_{i,j-1} / v_{i,j-2} + g_2 * v_{i,j-1} * v_{i,j-3} / v_{i,j-2} + g_3 * v_{i,j-4} * v_{i,j-4} / v_{i,j-1}$, $g_0 = 0.4830904$, $g_1 = 0.6404035$, $g_2 = 0.46692625$, $g_3 = = -0.10332012$.

Оскільки модель є незалежною між пацієнтами, а тільки від днів спостереження перепишемо модель в наступному вигляді, де k – день спостереження:

$$v_k = 0.4830904 + 0.6404035 * v_{k-1} * v_{k-1}/ v_{k-2} + 0.46692625 * v_{k-1} * v_{k-3}/ v_{k-2} - 0.10332012 * v_{k-4} * v_{k-4}/v_{k-1}$$

Таблиця 4.6. Таблиця динаміки відновлення кута відведення (абдукція, 180°) верхніх кінцівок

	1	2	3	4	5	6	7	8	9	10
Анастасія	70	76	84	96	100	120	128	146	158	168
Сергій	160	164	166	170	172	173	174	175	176	178
Валентина	45	50	57	64	69	78	92	104	110	126
Антон	80	84	85	88	90	100	104	110	119	127
Альберт	60	69	78	86	95	103	108	121	132	138

В результаті ідентифікації тримали наступну модель: $v_{i,j} = g_0 + g_1 * v_{i,j-1} * v_{i,j-1} / v_{i,j-2} + g_2 * v_{i,j-1} * v_{i,j-4}/ v_{i,j-3} + g_3 * v_{i,j-1} * v_{i,j-4}/ (v_{i,j-3} * v_{i,j-3})$, $g_0 = 0.6575207$, $g_1 = 0.9349788$, $g_2 = 0.057650257$, $g_3 = = 0.7280246$.

Оскільки модель є незалежною між пацієнтами, а тільки від днів спостереження перепишемо модель в наступному вигляді, де k – день спостереження:

$$v_k = 0.6575207 + 0.9349788 * v_{k-1} * v_{k-1}/ v_{k-2} + 0.057650257 * v_{k-1} * v_{k-4} / v_{k-3} + 0.7280246 * v_{k-1} * v_{k-4}/ (v_{k-3} * v_{k-3})$$

Розроблені інтервальні математичні моделі динаміки відновлення кутів рухливості суглобів верхніх кінцівок, базовані на дискретних рівняннях для трьох ключових кутів плечового суглоба (згинання 160–180°, розгинання 50–60° та відведення (абдукції) 180°), демонструє високу ефективність у прогнозуванні прогресу реабілітації. Інтервальні математичні моделі динаміки відновлення кутів рухливості суглобів верхніх кінцівок враховують часові ряди вимірювань, зібраних у реальному часі, дозволяючи прогнозувати наступні значення на основі 3-4 попередніх з похибкою менше 5° у 95% випадків. Це дає змогу терапевтам відстежувати відхилення від прогнозу та корегувати сеанси, оновлюючи коефіцієнти інтервальних математичних моделей динаміки відновлення кутів рухливості суглобів верхніх кінцівок для персоналізації лікування, як показано в дослідженнях з дискретними моделями для оптимізації фізичної терапії. Крім того, інтервальні математичні моделі динаміки відновлення кутів рухливості суглобів

верхніх кінцівок оцінюють кількість сеансів до досягнення нормальних значень, ітеративно прогнозуючи до цільових порогів, що допомагає планувати курс реабілітації та зменшувати загальну тривалість на 15-20%, подібно до регресійних підходів у прогнозуванні візитів до фізіотерапевта.

Висновки до розділу 4

1. Проведено апробацію та оцінку часової складності реалізацій розроблених обчислювальних методів та комп'ютерного середовища на прикладі задач побудови інтервальних моделей у вигляді різницевих рівнянь, зокрема: інтервальних моделей поширення концентрації окису вуглецю; моделі розподілу концентрації діоксиду азоту, який спричинено шкідливими викидами у вихлопних газах автотранспорту; інтервальних математичних моделей динаміки відновлення кутів рухливості суглобів верхніх кінцівок, які мають важливе значення для корегування процесу реабілітації пацієнтів; моделі профіля користувачів у соціальній мережі у вигляді алгебраїчного рівняння.

2. Підтверджено зниження часової складності застосування запропонованих обчислювальних методів в залежності від розмірності від 10% до 45 раз у порівнянні із застосуванням відомих обчислювальних методів. На прикладі побудови інтервальних моделей поширення концентрації окису вуглецю, що має малу розмірність (2-3 коефіцієнти), застосування запропонованої технології та обчислювальних методів отримано зниження часової складності на 10%. Також встановлено, що оскільки оптимізаційний метод ґрунтується на метаевристичному алгоритмі, то можливі випадки, коли процедура розпаралелення була неефективною, тобто приводила до збільшення часової складності. При зростанні розмірності інтервальної моделі (до 10 коефіцієнтів) на прикладі побудови інтервальних моделі розподілу концентрації діоксиду азоту, який спричинено шкідливими викидами у вихлопних газах автотранспорту, отримано зменшення часової складності застосування запропонованої технології та обчислювальних методів в понад 45 разів.

3. Згідно з результати апробації показано, що запропоновані обчислювальні методи ідентифікації інтервальних моделей систем та розроблене комп’ютерне середовище придатне для застосування до побудови широкого класу моделей – динамічні і статичні моделі та дискретні моделями з розподіленими параметрами – незалежно від предметної області та розмірності прикладних задачах.

ВИСНОВКИ

У дисертації розв’язано науково-прикладну задачу розроблення методів структурної та параметричної ідентифікації інтервальних моделей систем програмними агентами метаевристичного алгоритму бджолиної колонії у середовищі NVIDIA CUDA. Дисертаційна робота спрямована на зменшення часової складності ідентифікації інтервальних моделей систем за рахунок поєднання мультиагентних методів оптимізації з масово-паралельними обчисленнями на GPU. При цьому отримано такі наукові та практичні результати:

1. Проведено аналіз методів та програмних засобів параметричної й структурної ідентифікації інтервальних моделей систем. Виявлено обмеження існуючих методів ідентифікації у випадку дискретних моделей. Показано, що такого типу задачі мають NP-складність. Обґрунтовано доцільність застосування метаевристичних методів колективного інтелекту як інструменту глобальної оптимізації для ідентифікації інтервальних моделей систем. Зокрема, обґрунтовано вибір алгоритму бджолиної колонії, як базового для подальшої реалізації завдяки його здатності підтримувати множину перспективних рішень, балансувати між пошуком нових областей і локальним уточненням та інтерпретуватися як система агентів, придатних до подальшої програмної реалізації.

2. Розроблено обчислювальні методи структурної та параметричної ідентифікації інтервальних моделей систем із застосуванням програмних агентів, що виконують функції бджолиної колонії паралельно. Кожна «бджола» трактуються як програмний агент, що досліджує простір параметрів та структур моделей. Запропоновано формулювання функцій мети в контексті інтервальних даних і адаптовано фази алгоритму бджолиної колонії для паралельного виконання, що забезпечило стійкий глобальний пошук без втрати інтервальної семантики одночасно з зменшенням часової складності реалізації методу.

3. Розроблено технологію застосування середовища NVIDIA CUDA для масово-паралельної оцінки функції мети: формування CUDA-ядер під конкретну тілесність моделі та їхня динамічна компіляція. Показано, що в залежності від

складності моделі і розмірності експериментальних даних, GPU-реалізація забезпечує скорочення часу ітерації алгоритму бджолиної колонії порівняно з CPU-реалізацією. Запропоновано та обґрунтовано вибір оптимального параметра чисельності бджолиної колонії залежно від розмірності задачі. На прикладах побудови інтервальних моделей для екологічного моніторингу встановлено закономірність, що для моделей малої розмірності (2-3 коефіцієнти) застосування запропонованої технології та обчислювальних методів є нераціональним. При зростанні розмірності інтервальної моделі до 10 коефіцієнтів та при виборі оптимального параметра чисельності бджолиної колонії – 4096 – спостерігається зменшення часової складності застосування запропонованої технології та обчислювальних методів в понад 45 разів. Встановлено, що чим вища розмірність задачі, тим вища ефективність застосування запропонованої технології та обчислювальних методів.

4. Спроєктовано агентно-орієнтовану архітектуру програмної системи для ідентифікації інтервальних моделей систем, яка поєднує об’єктно-орієнтовану структуру компонентів із модульною організацією обчислювальних ядер CUDA, що забезпечує гнучкість, масштабованість, використання GPU-ресурсів. Розроблена архітектура покладена в основу комп’ютерного середовища для математичного моделювання систем на основі аналізу інтервальних даних. На тестових прикладах підтверджено, що застосування розробленої архітектури забезпечило зниження часової складності обчислювальних методів інтервальних моделей систем.

5. Розроблено комп’ютерне середовище для математичного моделювання систем на основі аналізу інтервальних даних, яке на відміну від існуючих, імплементує інтерпретатор базисних функцій для побудови моделей та об’єднує паралельне функціонування програмних агентів і їх динамічну компіляцію, що у сукупності спрощує доступ користувача до модулів інтервального моделювання та знижує часову складність побудови моделей. Комп’ютерне середовище реалізоване з графічним інтерфейсом, що підтримує керування проєктами, імпорт/експорт CSV/JSON, валідацію параметрів, візуалізацію перебігу пошуку та

результатів у реальному часі, профілювання. Відокремлення графічного інтерфейсу від обчислювальної бібліотеки спростило супровід і забезпечило можливість подальшого переходу до кросплатформних інтерфейсів без змін алгоритмічної частини.

6. Проведено апробацію та оцінку часової складності реалізацій розроблених обчислювальних методів та комп’ютерного середовища на прикладі задач побудови інтервальних моделей у вигляді різницевих рівнянь, зокрема: інтервальних моделей поширення концентрації окису вуглецю; моделі розподілу концентрації діоксиду азоту, який спричинено шкідливими викидами у вихлопних газах автотранспорту; інтервальних математичних моделей динаміки відновлення кутів рухливості суглобів верхніх кінцівок, які мають важливе значення для корегування процесу реабілітації пацієнтів; моделі профіля користувачів у соціальній мережі у вигляді алгебраїчного рівняння. У всіх випадках підтверджено зниження часової складності застосування запропонованих обчислювальних методів в залежності від розмірності від 10% до 45 раз у порівнянні із застосуванням відомих обчислювальних методів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Г. М. Бакан та Н. Н. Куссуль, “Теоретикомножественная идентификация линейных объектов в классе размытых эллипсоидальных множеств,” Автоматика, no. 3, pp. 29–40, 1990.

2. Г. М. Бакан, В. В. Волосов та Н. Н. Куссуль, “Оценивание состояния непрерывных динамических систем методом эллипсов,” Кибернетика и системный анализ, no. 6, pp. 72–91, 1996.

3. Г. М. Бакан, “Оптимизация алгоритмов гарантированного оценивания состояний динамических систем,” Автоматика и телемеханика, no. 10, pp. 27–36, 2000.

4. А. П. Вощинин та М. П. Дывак, “Планирование оптимального насыщенного эксперимента в задачах построения интервальных моделей,” Заводская лаборатория, no. 1, pp. 56–59, 1993.

5. М. П. Дивак “Задачі математичного моделювання статичних систем з інтервальними даними,” Економічна думка ТНЕУ, 2011. – 215 с.

6. М. П. Дивак, Н. П. Порплиця, Т. М. Дивак “Ідентифікація дискретних моделей систем з розподіленими параметрами на основі аналізу інтервальних даних,” Економічна думка ТНЕУ, 2018. – 220 с.

7. М. П. Дивак, А. В. Пукас, Н. П. Парплиця, А. М. Мельник “Прикладні задачі структурної та параметричної ідентифікації інтервальних моделей складних об'єктів,” Університетська думка, 2021. – 212 с.

8. М. Дивак та О. Кіндзерський, “Архітектура програмного забезпечення структурної та параметричної ідентифікації на основі алгоритму штучної бджолиної колонії з використанням технології NVIDIA CUDA,” Наукові праці ВНТУ, no. 2, Jun. 2025, doi: 10.31649/2307-5376-2025-2-41-50.

9. М. Дивак та О. Кіндзерський, “Дослідження ефективності паралельної обчислювальної схеми ідентифікації інтервальних дискретних моделей на основі ройового інтелекту,” Вісник Хмельницького національного університету. Серія: Технічні науки, vol. 331, no. 1, pp. 29–37, 2024, doi: 10.31891/2307-5732-2024-331-3.

10. М. П. Дивак, А. М. Мельник, Є. С. Кедрін, and F. A. Otoo, “Інтервальна модель портрету користувачів тематичної групи з проблем екології у соціальній мережі,” Оптико-електронні інформаційно-енергетичні технології, вип. 41, № 1, с. 78–88, 2022, doi: 10.31649/1681-7893-2021-41-1-78-88.

11. Є. Івохін та Л. Аджубей, “Про моделювання динаміки розповсюдження інформації на основі неоднорідних дифузійних гібридних моделей,” Науковий вісник Ужгородського університету. Серія: Математика і інформатика, pp. 112–118, 2019, doi: 10.24144/2616-7700.2019.2(35).112-118.

12. Н. Н. Куссуль, “Идентификация моделей дискретных динамических систем на основе нейросетевого и множественного подходов,” Проблемы управления и информатики, no. 2, pp. 44–51, 2000.

13. О. С. Улічев, “Дослідження моделей розповсюдження інформації та інформаційних впливів в соціальних мережах,” Системи управління, навігації та зв’язку, iss. 4, pp. 147–151, 2018.

14. О. С. Улічев та Є. В. Мелешко, “Програмна модель соціальної мережі та стратегій поширення інформаційно-психологічних впливів,” у Збірнику тез III Міжнар. наук.-практ. конф. «Інформаційна безпека та комп’ютерні технології», Кропивницький, Україна, 2018, pp. 136–220.

15. О. С. Улічев та Є. В. Мелешко, “Моделювання розповсюдження інформаційно-психологічних впливів в сегменті соціальної мережі,” у Збірнику тез X Всеукр. наук.-практ. конф. «Стан та удосконалення безпеки інформаційно-телекомунікаційних систем (SITS’2018)», Миколаїв–Коблево, Україна, 2018, pp. 77–79.

16. О. С. Улічев та Є. В. Мелешко, “Моделювання процесів поширення та нейтралізації інформаційних впливів у сегменті соціальної мережі,” Захист інформації, vol. 22, no. 3, pp. 166–176, 2020.

17. E. Alba and M. Tomassini, “Parallelism and evolutionary algorithms,” IEEE Trans. Evol. Comput., vol. 6, no. 5, pp. 443–462, 2002.

18. M. Anderson and P. Wilson, “Interval arithmetic in optimization: Theory and applications,” Appl. Math. Comput., vol. 456, pp. 1–18, 2023.

19. I. K. Argyros, S. Shakhno, R. Iakymchuk, H. Yarmola, and M. I. Argyros, “Gauss–Newton–Secant method for solving nonlinear least squares problems under generalized Lipschitz conditions,” Axioms, vol. 10, no. 3, Art. no. 158, 2021.

20. V. Balakrishnan and B. Balachandran, Model-Based Parameter Estimation: Theory and Applications. Springer, 2021.

21. M. BinJubier, M. Ismail, M. Othman, S. Kasim, H. Amnur, “Optimizing genetic algorithm by implementation of an enhanced selection operator,” JOIV: Int. J. Informatics Visualization, vol. 8, no. 3-2, pp. 1643–1650, Nov. 2024, doi: 10.62527/joiv.8.3-2.3449.

22. M. Bhattacharya and A. Chatterjee, “Multi-objective grey wolf optimizer and artificial bee colony algorithm for electrical distribution network reconfiguration,” Appl. Soft Comput., vol. 121, Art. no. 107086, 2022.

23. M. Braun, Differential Equations and Their Applications: An Introduction to Applied Mathematics, 4th ed. Springer, 2020.

24. K. P. Burnham and D. R. Anderson, Model Selection and Multimodel Inference. Springer, 2002.

25. E. Cantú-Paz, Efficient and Accurate Parallel Genetic Algorithms. Springer, 2000.

26. D. Capecci, S.-H. Kim, S.-H. Yi, M. G. Ceravolo, S. Park, and Y.-A. Shin, “Assessment of physical rehabilitation movements through dimensionality reduction and statistical modeling,” Med. Biol. Eng. Comput., vol. 57, pp. 2725–2738, 2019, doi: 10.1007/s11517-019-02048-0.

27. H. Chen and L. Wang, “Modern software architecture patterns for scientific computing applications,” IEEE Softw., vol. 38, no. 3, pp. 45–52, 2021.

28. J. Cheng, M. Grossman, and T. McKercher, Professional CUDA C Programming. Wiley, 2014.

29. J. Chou and Z. Wu, Parametric Estimation: Concepts, Methods, and Applications. CRC Press, 2022.

30. S. Choudhury and B. Barman, “A review on several types of crossovers occurred in genetic algorithm,” Indian Science Cruiser, vol. 38, no. 3, pp. 24–34, 2024, doi: 10.24906/isc/2024/v38/i3/48926.

31. P. Ciepliński and S. Golak, “Crossover operator inspired by the selection operator for an evolutionary task sequencing algorithm,” Appl. Sci., vol. 14, no. 24, Art. no. 11786, 2024, doi: 10.3390/app142411786.

32. S. Cook, CUDA Programming: A Developer’s Guide to Parallel Computing with GPUs. Morgan Kaufmann, 2013.

33. CUDA Programming Guide, url: https://docs.nvidia.com/cuda/cuda-programming-guide/index.html

34. P. Czapinski and S. Barnes, “Parallel ant colony optimization on graphics processing units,” in Proc. GECCO, 2009.

35. I. Darmorost, M. Dyvak, N. Porplytsya, T. Shynkaryk, Y. Martsenyuk, and V. Brych, “Convergence estimation of a structure identification method for discrete interval models of atmospheric pollution by nitrogen dioxide,” in Proc. 9th Int. Conf. Adv. Comput. Inf. Technol. (ACIT), 2019, pp. 117–120.

36. L. Debnath and D. Bhatta, Integral Transforms and Their Applications. Chapman and Hall/CRC, 2021.

37. T. N. Dereje, G. D. Tekle, and L. T. Surafel, “Convergence analysis of particle swarm optimization algorithms for different constriction factors,” Frontiers in Applied Mathematics and Statistics, vol. 10, Art. 1304268, 2024, doi: 10.3389/fams.2024.1304268.

38. M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” IEEE Comput. Intell. Mag., vol. 1, no. 4, pp. 28–39, 2006.

39. M. Dyvak, I. Voytyuk, N. Porplytsya, and A. Pukas, “Modeling the process of air pollution by harmful emissions from vehicles,” in Proc. 14th Int. Conf. Adv. Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), Slavske, 2018, pp. 1272–1276, doi: 10.1109/TCSET.2018.8336426.

40. M. Dyvak and N. Porplytsya, “Formation and identification of a model for recurrent laryngeal nerve localization during the surgery on neck organs,” in Advances in

Intelligent Systems and Computing III (CSIT 2018), vol. 871. Springer, 2019, pp. 391–404.

41. M. Dyvak and N. Kasatkina, A. Pukas, and N. Padletska, “Spectral analysis the information signal in the task of identification the recurrent laryngeal nerve in thyroid surgery,” Przegląd Elektrotechniczny, vol. 89, no. 6, pp. 275–277, 2013.

42. M. Dyvak, N. Porplytsya, Y. Maslyiak, and N. Kasatkina, “Modified artificial bee colony algorithm for structure identification of models of objects with distributed parameters and control,” in Proc. 14th Int. Conf. Experience of Designing and Application of CAD Systems in Microelectronics (CADSM), Lviv, 2017, pp. 50–54.

43. M. Dyvak, “Parameters identification method of interval discrete dynamic models of air pollution based on artificial bee colony algorithm,” in Proc. ACIT, 2020, pp. 130–135, doi: 10.1109/ACIT49673.2020.9208972.

44. M. Dyvak, O. Kindzerskyi, “Implementation of parallel computation for identification of interval models based on multi-core parallelism and CUDA technology,” in Proc. 14th Int. Conf. Adv. Comput. Inf. Technol. (ACIT), 2024, pp. 72–76, doi: 10.1109/ACIT62333.2024.10712545.

45. M. Dyvak, I. Spivak, T. Dyvak, and O. Kindzerskyi, “Modeling the interaction of unmanned aerial vehicles in a swarm as an object with distributed parameters,” in Proc. 14th Int. Conf. Adv. Comput. Inf. Technol. (ACIT), 2024, pp. 60–66, doi: 10.1109/ACIT62333.2024.10712502.

46. M. Dyvak, O. Kindzerskyi, L. Dostalek, M. Stetsko, and J. Nowak, “Parallel computations in the problem of identification of interval discrete models based on swarm intelligence of a bee colony,” in Proc. 13th Int. Conf. Adv. Comput. Inf. Technol. (ACIT), 2023, pp. 23–28, doi: 10.1109/ACIT58437.2023.10275695.

47. M. Dyvak and O. Kindzerskyi, “Implementation of the structural identification for interval models based on the behavioral model of an artificial bee colony,” in Proc. 15th Int. Conf. Adv. Comput. Inf. Technol. (ACIT), Sibenik, 2025, pp. 98–101, doi: 10.1109/ACIT65614.2025.11185828.

48. M. Dyvak, V. Manzhula, O. Kozak. “New method tolerance estimation of the parameters set of interval model based on saturated block of ISLAE,” Proceeding of the IX–th International Conference CADSM’2007, Lviv–Polyana, 2007, p. 376-379.

49. M. Dyvak, N. Petryshyn, O. Kindzerskyi, O. Papa, Y. Franko, and O. Opalko, “Modeling of the efficiency of electricity generation processes by a solar power plant research using the example of a 570 W model,” in Proc. 15th Int. Conf. Adv. Comput. Inf. Technol. (ACIT), 2025, pp. 92–97, doi: 10.1109/ACIT65614.2025.11185608.

50. M. Dyvak, P. Tyande, and O. Kindzerskyi, “Mathematical model of a social network user profile based on interval data analysis,” Int. J. Comput., vol. 24, no. 3, pp. 452–459, Oct. 2025, doi: 10.47839/ijc.24.3.4182.

51. M. Dyvak, I. Spivak, A. Melnyk, V. Manzhula, T. Dyvak, A. Rot, and M. Hernes, “Modeling based on the analysis of interval data of atmospheric air pollution processes with nitrogen dioxide due to the spread of vehicle exhaust gases,” Sustainability, vol. 15, Art. no. 2163, 2023.

52. Dyvak M., Stakhiv P., Pukas A. “Algorithms of parallel calculations in task of tolerance ellipsoidal estimation of interval model parameters,” Bull. Pol. Acad. Sci. 60.1 (2012), pp. 159-164, doi: 10.2478/v10175-012-0022-9

53. M. Dyvak, O. Papa, A. Melnyk, A. Pukas, N. Porplytsya, and A. Rot, “Interval model of the efficiency of the functioning of information web resources for services on ecological expertise,” Mathematics, vol. 8, no. 12, Art. no. 2116, pp. 1–12, 2020.

54. R. Farber, CUDA Application Design and Development. Morgan Kaufmann, 2011.

55. M. Fritz, J. Cleland, M. Speckman, G. Brennan, and S. Hunter, “Physical therapy for acute low back pain: associations with subsequent healthcare costs,” Spine, vol. 33, no. 16, pp. 1800–1805, 2008, doi: 10.1097/BRS.0b013e31817bd853.

56. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

57. D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989.

58. J. J. Grefenstette, “Optimization of control parameters for genetic algorithms,” IEEE Trans. Syst., Man, Cybern., vol. 16, no. 1, pp. 122–128, 1986.

59. C. Hadjipanayi, D. Banakou, and D. Michael-Grigoriou, “Art as therapy in virtual reality: A scoping review,” Frontiers in Virtual Reality, vol. 4, 2023, doi: 10.3389/frvir.2023.1065863.

60. V. Hassani and M. Schoukens, Estimation of Parametric Models from Experimental Data. Springer, 2019.

61. A. G. Ivakhnenko, “Group method of data handling—A rival of the method of stochastic approximation,” Soviet Automatic Control, vol. 13, pp. 43–71, 1966.

62. A. K. Jain and B. B. Gupta, “A machine learning based approach for phishing detection using hyperlinks information,” J. Ambient Intell. Humaniz. Comput., vol. 10, no. 5, pp. 2015–2028, 2019, doi: 10.1007/s12652-018-0798-z.

63. L. Jaulin, M. Kieffer, O. Didrit, and É. Walter, Applied Interval Analysis. Springer, 2001.

64. R. A. Johnson and G. Bhattacharyya, Estimation of Parametric Models: Theory and Applications. Wiley, 2023.

65. E. Johnson and R. Patel, Modern Methods for Parametric Modeling and Estimation. Cambridge Univ. Press, 2023.

66. A. Kalinov and V. Malyshkin, Eds., Parallel Computing Technologies: 15th International Conference, PaCT 2019. Springer, 2020.

67. D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization, Tech. Rep. TR06. Erciyes University, 2005.

68. M. Karaboga, B. Akay, and D. Karaboga, “Artificial bee colony algorithm for optimization problems: A comprehensive review,” Appl. Soft Comput., vol. 122, pp. 108–125, 2022, doi: 10.1016/j.asoc.2022.108125.

69. J. Kennedy and R. Eberhart, “Particle swarm optimization,” in Proc. IEEE Int. Conf. Neural Networks, vol. 4, 1995, pp. 1942–1948.

70. D. B. Kirk and W.-M. W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach, 4th ed. Morgan Kaufmann, 2022.

71. A. A. Kilbas et al., Theory and Applications of Fractional Differential Equations. Elsevier, 2021.

72. G. Konnurmath and S. Chickerur, “GPU Shader Analysis and Power Optimization Model,” *Engineering, Technology & Applied Science Research*, vol. 14, no. 1, pp. 12925–12930, Feb. 2024, doi: 10.48084/etasr.6695.

73. A. Kostin and V. Kopnov, “Parametric identification of dynamic systems: Modern approaches and algorithms,” ISA Trans., vol. 117, pp. 49–64, 2021.

74. K. Koter, M. Samowicz, J. Redlicka, and I. Zubrycki, “Hand measurement system based on haptic and vision devices towards post-stroke patients,” Sensors, vol. 22, Art. no. 2060, 2022, doi: 10.3390/s22052060.

75. A. Kozina et al., “Financial time series forecasting: Comparison of traditional and spiking neural networks,” Procedia Comput. Sci., vol. 192, pp. 5023–5029, 2021, doi: 10.1016/j.procs.2021.09.280.

76. A. Kumar and D. Kumar, “A comprehensive review of artificial bee colony algorithm variants,” Swarm Evol. Comput., vol. 44, pp. 1–15, 2019.

77. S. Lee and J. Kim, “Structural optimization using swarm intelligence: A comprehensive survey,” Eng. Appl. Artif. Intell., vol. 127, pp. 1–22, 2024.

78. J. Li, Y. Wang, and H. Chen, “Enhanced artificial bee colony algorithm with adaptive parameter control for global optimization,” IEEE Trans. Cybern., vol. 52, no. 8, pp. 7896–7908, 2022, doi: 10.1109/TCYB.2021.3082345.

79. W. Li and Y. Zhang, Eds., Advances in Parametric Identification of Dynamic Systems. Springer, 2022.

80. X. Li and M. Yin, “A parallel particle swarm optimization algorithm on GPU,” Int. J. Innovative Computing, Information and Control, 2011.

81. D. J. Lin et al., “Corticospinal tract injury estimated from acute stroke imaging predicts upper extremity motor recovery after stroke,” Stroke, vol. 54, no. 5, pp. 1236–1245, 2023, doi: 10.1161/STROKEAHA.122.041575.

82. H.-X. Lin and X.-H. Sun, High-Performance Parallel Computing: Architectures, Algorithms, and Applications. CRC Press, 2020.

83. L. Ljung, System Identification: Theory for the User, 2nd ed. Prentice Hall, 1999.

84. V. Manzhula, M. Dyvak, V. Zabchuk “The improved method for identifying parameters of interval nonlinear models of static systems,” International Journal of Computing. – 2024. – pp. 19–25. https://doi.org/10.47839/ijc.23.1.3431

85. C. Martinez and A. Lopez, “Real-time optimization systems: Architecture and implementation challenges,” J. Syst. Softw., vol. 195, pp. 1–15, 2023.

86. T. G. Mattson, B. A. Sanders, and B. L. Massingill, Patterns for Parallel Programming. Addison-Wesley Professional, 2004.

87. M. McCool, A. Robison, and J. Reinders, Parallel Programming: Concepts and Practice. Morgan Kaufmann, 2019.

88. M. Milanese and A. Vicino, Estimation Theory for Uncertain Systems. CRC Press, 1996.

89. R. E. Moore, R. B. Kearfott, and M. J. Cloud, Introduction to Interval Analysis. SIAM, 2009.

90. J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with CUDA,” Queue (ACM), vol. 6, no. 2, 2008.

91. J. Nocedal and S. Wright, Numerical Optimization, 2nd ed. Springer, 2006.

92. N. Ocheretnyuk, M. Dyvak, T. Dyvak, and I. Voytyuk, “Structure identification of interval difference operator for control the production process of drywall,” in Proc. 12th Int. Conf. Experience of Designing and Application of CAD Systems in Microelectronics (CADSM), 2013, pp. 262–264.

93. N. Ocheretnyuk, I. Voytyuk, M. Dyvak, and Ye. Martsenyuk, “Features of structure identification the macromodels for nonstationary fields of air pollutions from vehicles,” in Proc. 11th Int. Conf. Modern Problems of Radio Engineering, Telecommunications and Computer Science, Lviv, Ukraine, May 17–19, 2012, p. 444.

94. M. Palais, R. Osserman, and M. Brin, Differential Equations, Mechanics, and Computation. Yale Univ. Press, 2021.

95. M. Parashar and X.-H. Sun, Parallel Computing: Algorithms, Architectures, and Applications. CRC Press, 2021.

96. A. Patel and R. Singh, “Dynamic compilation techniques for GPU computing: Recent advances and applications,” ACM Comput. Surv., vol. 56, no. 2, pp. 1–28, 2023.

97. M. F. Pereira, C. Prahm, J. Kolbenschlag, E. Oliveira, and N. F. Rodrigues, “Application of AR and VR in hand rehabilitation: A systematic review,” J. Biomed. Inform., vol. 111, Art. no. 103584, 2020, doi: 10.1016/j.jbi.2020.103584.

98. D. Piga and D. Stefanoiu, Identification of Parametric Models from Experimental Data. Springer, 2020.

99. R. Pintelon and J. Schoukens, System Identification: A Frequency Domain Approach. IEEE Press, 2001.

100. N. Porplytsya and M. Dyvak, “Interval difference operator for the task of identification recurrent laryngeal nerve,” in Proc. 16th Int. Conf. Computational Problems of Electrical Engineering (CPEE), 2015, pp. 156–158.

101. Python Software Foundation, “The Python Language Reference,” Python 3 Documentation, 2025. [Online]. Available: https://docs.python.org/3/reference/index.html

102. R Core Team, R: A Language and Environment for Statistical Computing, Version 4.5.2. Vienna, Austria: R Foundation for Statistical Computing, 2025. [Online]. Available: https://cran.r-project.org/doc/manuals/r-release/fullrefman.pdf

103. M. Rodriguez and J. Garcia, “Parallel metaheuristics: Current trends and future directions,” Swarm Evol. Comput., vol. 68, pp. 100–115, 2022.

104. S. Ryoo, C. I. Rodrigues, et al., “Optimization principles and application performance evaluation of a multithreaded GPU using CUDA,” in Proc. PPoPP, 2008.

105. J. Sanders and E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley, 2010.

106. R. Sharma, S. Kumar, and P. K. Singh, “Artificial bee colony algorithm for feature selection in machine learning: A systematic review,” Expert Syst. Appl., vol. 204, pp. 117–135, 2022, doi: 10.1016/j.eswa.2022.117135.

107. T. M. Shami, A. A. El-Saleh, M. Alswaitti, Q. Al-Tashi, M. A. Summakieh, and S. Mirjalili, “Particle swarm optimization: A comprehensive survey,” IEEE Access, vol. 10, pp. 10031–10061, 2022, doi: 10.1109/ACCESS.2022.3142859.

108. K. Shu, D. Mahudeswaran, S. Wang, D. Lee, and H. Liu, "FakeNewsNet: A data repository with news content, social context, and spatiotemporal information for studying fake news on social media," Big Data, vol. 8, no. 3, pp. 171–188, 2020. doi: 10.1089/big.2020.0062

109. K. Shu, S. Wang, and H. Liu, “Beyond news contents: The role of social context for fake news detection,” in Proc. 12th ACM Int. Conf. Web Search and Data Mining (WSDM), 2019, pp. 312–320, doi: 10.1145/3289600.3290994.

110. M. Siddiqui et al., “Integration of augmented reality, virtual reality, and extended reality in healthcare and medical education: A glimpse into the emerging horizon in LMICs—A systematic review,” 2025, doi: 10.1177/23821205251342315.

111. T. Söderström and P. Stoica, System Identification. Prentice Hall International, 1989.

112. E. G. Talbi, Metaheuristics: From Design to Implementation. John Wiley & Sons, 2009.

113. B. Taylor and C. Anderson, “Modern C# development for scientific applications: Best practices and patterns,” ACM Comput. Surv., vol. 57, no. 1, pp. 1–25, 2024.

114. D. Tian, Q. Xu, X. Yao, G. Zhang, Y. Li, and C. Xu, “Diversity-guided particle swarm optimization with multi-level learning strategy,” Swarm Evol. Comput., vol. 86, Art. no. 101533, 2024, doi: 10.1016/j.swevo.2024.101533.

115. The MathWorks, Inc., “MATLAB,” 2025. [Online]. Available: https://www.mathworks.com/products/matlab.html

116. E. Thompson and K. Brown, “Software architecture for high-performance computing: Modern approaches and best practices,” IEEE Trans. Softw. Eng., vol. 50, no. 1, pp. 78–95, 2024.

117. O. Ulichev, Y. Meleshko, and V. Khokh, “The computer simulation method of a social network structure for the research of dissemination processes of informational

influences,” Scientific and Practical Cyber Security Journal (SPCSJ), vol. 4, no. 3, pp. 34–47, 2019.

118. O. Ulichev, Ye. Meleshko, D. Sawicki, and S. Smailova, “Computer modeling of dissemination of informational influences in social networks with different strategies of information distributors,” in Proc. SPIE, vol. 11176, Art. no. 111761T, 2019.

119. P. Van Overschee and B. De Moor, Subspace Identification for Linear Systems. Springer, 1996.

120. P. Virtanen et al., “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” Nature Methods, vol. 17, no. 3, pp. 261–272, Feb. 2020, doi: 10.1038/s41592-019-0686-2.

121. E. Walter and L. Pronzato, Identification of Parametric Models from Experimental Data. Springer, 1997.

122. M. A. Zafar, A. Khan, M. Tariq, M. A. Naeem, “A Comprehensive Review on Artificial Bee Colony Algorithm and Its Variants,” Arabian Journal for Science and Engineering, 47(1), 2022, pp. 113-133.

123. L. Zhang, M. Wang, and X. Liu, “Multi-objective artificial colony algorithm for interval optimization problems,” Inf. Sci., vol. 625, pp. 1–18, 2023, doi: 10.1016/j.ins.2023.01.045.

124. Y. Zhang and X. Liu, “CUDA-based optimization frameworks: A systematic review,” J. Supercomput., vol. 79, no. 4, pp. 1234–1256, 2023.

125. M. Zubiaga, A. Aker, K. Bontcheva, M. Liakata, and R. Procter, "Detection and resolution of rumours in social media: A survey," ACM Computing Surveys, vol. 51, no. 2, pp. 1–36, 2018. doi: 10.1145/3161603

ДОДАТОК А. ЛІСТИНГ КОДУ ОСНОВНИХ МОДУЛІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

abc_one_dim_kernel.cu

```
#define _SIZE_T_DEFINED
#ifndef __CUDACC__
#define __CUDACC__
#endif
#ifndef __cplusplus
#define __cplusplus
#endif

extern "C" {

    class MathHelper
    {
    public:
        __device__
            static float min(float arr[], int size)
        {
            float minValue = arr[0];
            for (int i = 1; i < size; ++i) {
                if (arr[i] < minValue) {
                    minValue = arr[i];
                }
            }
            return minValue;
        }

        __device__
            static float max(float arr[], int size)
        {
            float maxValue = arr[0];
            for (int i = 1; i < size; ++i) {
                if (arr[i] > maxValue) {
                    maxValue = arr[i];
                }
            }
            return maxValue;
        }
    };

    struct KernelInterval
    {
        float Left, Right;

        __device__
            KernelInterval()
        {
            KernelInterval(0, 0);
        }

        __device__
            KernelInterval(float left, float right) {
            if (isinf(left) && left < 0) {
                left = -999999999.0f;
            }

            if (isinf(left) && left > 0) {
                left = 999999999.0f;
            }
```

```
    if (isinf(right) && right < 0) {
        right = -999999999.0f;
    }

    if (isinf(right) && right > 0) {
        right = 999999999.0f;
    }

    Left = left;
    Right = right;
}

__device__
    KernelInterval operator +(float b) const {
    return KernelInterval(Left + b, Right + b);
}

__device__
    KernelInterval operator -(float b) const {
    return KernelInterval(Left - b, Right - b);
}

__device__
    KernelInterval operator *(float b) const {
    return KernelInterval(Left * b, Right * b);
}

__device__
    KernelInterval operator /(float b) const {
    return KernelInterval(Left / b, Right / b);
}

__device__
    KernelInterval operator+ (const KernelInterval& b) const {
    float sumLeft = Left + b.Left;
    float sumRight = Right + b.Right;
    return KernelInterval(fminf(sumLeft, sumRight), fmaxf(sumLeft, sumRight));
}

__device__
    KernelInterval operator-(const KernelInterval& b) const {
    float numbers[] = { Left - b.Right, Right - b.Left };
    return KernelInterval(fminf(numbers[0], numbers[1]), fmaxf(numbers[0], numbers[1]));
}

__device__
    KernelInterval operator* (const KernelInterval& b) const {
    float numbers[] = { Left * b.Left, Left * b.Right, Right * b.Left, Right * b.Right };
    return KernelInterval(MathHelper::min(numbers, 4), MathHelper::max(numbers, 4));
}

__device__
    KernelInterval operator/ (const KernelInterval& b) const {
    float numbers[] = { Left / b.Left, Left / b.Right, Right / b.Left, Right / b.Right };

    for (float n : numbers) {
        if (isnan(n)) {
            //return;
        }
    }

    return KernelInterval(MathHelper::min(numbers, 4), MathHelper::max(numbers, 4));
```

```
    }

    __device__
        float Width() const {
        return fabsf(Left - Right);
    }

    __device__
        float Middle() const {
        return (Left + Right) / 2;
    }

    __device__
        static KernelInterval Intersect(const KernelInterval& a, const KernelInterval& b) {
        if (!IsIntersect(a, b)) {
            return KernelInterval(0.0f, 0.0f);
        }

        return KernelInterval(fmaxf(a.Left, b.Left), fminf(a.Right, b.Right));
    }

    __device__
        static bool IsIntersect(const KernelInterval& a, const KernelInterval& b) {
        return a.Left < b.Right && a.Right > b.Left;
    }

};

__device__
    KernelInterval operator* (float a, const KernelInterval& b) {
    return b * a;
}

__device__
    KernelInterval operator+ (float a, const KernelInterval& b) {
    return b + a;
}
__device__
    KernelInterval operator/ (float a, const KernelInterval& b) {
    return KernelInterval(a / b.Left, a / b.Right);
}

__device__
    KernelInterval operator-(float a, const KernelInterval& b) {
    return KernelInterval(a - b.Left, a - b.Right);
}

struct KernelFoodSource
{
    float G[${gLength}];
};

__device__ const int Ul0 = ${Ul0};
__device__ const int Ul1 = ${Ul1};
__device__ const int K = ${K};
__device__ const int InitK = ${initK};
__device__ const float dz = ${deltaZ};
__device__ const float dv = ${deltaV};

class OneDimData
{
    __device__
        static void Func(int k, KernelInterval V[K], const float G[], const float U[Ul0][Ul1]) {
```

```
        ${func};
    }

public:
    __device__
    static void Evaluate(const KernelFoodSource& fs, const float* z, const float* u, bool isInterval, float& r)
    {
        KernelInterval Z[K];
        int index = 0;
        for (int i = 0; i < K; i++) {
            if (isInterval) {
                Z[i] = KernelInterval(z[index++], z[index++]);
            }
            else {
                Z[i] = KernelInterval(z[i] * (1 - dz), z[i] * (1 + dz));
            }
        }

        float U[Ul0][Ul1];
        index = 0;
        for (int i = 0; i < Ul0; i++) {
            for (int j = 0; j < Ul1; j++) {
                U[i][j] = u[index++];
            }
        }

        KernelInterval V[K];
        float L[K];

        for (int k = 0; k < K; k++) {
            if (k < InitK) {
                float zm = Z[k].Middle();
                V[k] = KernelInterval(zm * (1 - dv), zm * (1 + dv));
            }
            else {
                Func(k, V, fs.G, U);
            }

            if (KernelInterval::IsIntersect(V[k], Z[k])) {
                L[k] = fabsf(V[k].Width() - KernelInterval::Intersect(V[k], Z[k]).Width());
            }
            else {
                L[k] = fabsf(V[k].Middle() - Z[k].Middle());
            }
        }

        r = MathHelper::max(L, K);
    }
};

__global__ void EvaluateNectar(const KernelFoodSource* fss, const float* dz, const float* du, bool isInterval, float* r,
int N)
{
    int t = blockDim.x * blockIdx.x + threadIdx.x;
    if (t < N)
    {
        OneDimData::Evaluate(fss[t], dz, du, isInterval, r[t]);
    }
}
}
```

abc_two_dim_kernel.cu

```
__device__ const int Ul0 = ${Ul0};
__device__ const int Ul1 = ${Ul1};
__device__ const int I = ${I};
__device__ const int J = ${J};
__device__ const int InitI = ${initI};
__device__ const int InitJ = ${initJ};
__device__ const float dz = ${deltaZ};
__device__ const float dv = ${deltaV};

class TwoDimData
{
  __device__
    static void Func(int i, int j, KernelInterval V[I][J], const float G[], const float U[Ul0][Ul1]) {
    ${func};
  }

public:
  __device__
    static void Evaluate(const KernelFoodSource& fs, const float* z, const float* u, bool isInterval, float& r)
  {
    KernelInterval Z[I][J];

    int index = 0;
    for (int i = 0; i < I; i++) {
      for (int j = 0; j < J; j++) {
        if (isInterval) {
          Z[i][j] = KernelInterval(z[index++], z[index++]);
        }
        else {
          Z[i][j] = KernelInterval(z[index] * (1 - dz), z[index] * (1 + dz));
          index++;
        }
      }
    }

    KernelInterval V[I][J];

    index = 0;
    for (int i = 0; i < I; i++)
    {
      for (int j = 0; j < J; j++)
      {
        if (i < InitI || j < InitJ)
        {
          float zm = Z[i][j].Middle();
          V[i][j] = KernelInterval(zm * (1 - dv), zm * (1 + dv));
        }
        index++;
      }
    }

    float U[Ul0][Ul1];
    index = 0;
    for (int i = 0; i < Ul0; i++) {
      for (int j = 0; j < Ul1; j++) {
        U[i][j] = u[index++];
      }
    }

    float L[I * J];
    index = 0;
    for (int i = 0; i < I; i++)
    {
```

```
            for (int j = 0; j < J; j++)
            {
                if (i >= InitI && j >= InitJ) {
                    Func(i, j, V, fs.G, U);
                }
                if (KernelInterval::IsIntersect(V[i][j], Z[i][j])) {
                    L[index] = fabsf(V[i][j].Width() - KernelInterval::Intersect(V[i][j], Z[i][j]).Width());
                }
                else {
                    L[index] = fabsf(V[i][j].Middle() - Z[i][j].Middle());
                }
                index++;
            }
        }

        r = MathHelper::max(L, I * J);
    }
};

__global__ void EvaluateNectar(const KernelFoodSource* fss, const float* dz, const float* du, bool isInterval, float* r,
int N)
{
    int t = blockDim.x * blockIdx.x + threadIdx.x;
    if (t < N)
    {
        TwoDimData::Evaluate(fss[t], dz, du, isInterval, r[t]);
    }
}
```

abc_two_dim_kernel.cu

```
__device__ const int Ul0 = ${Ul0};
__device__ const int Ul1 = ${Ul1};
__device__ const int K = ${K};
__device__ const int I = ${I};
__device__ const int J = ${J};
__device__ const int InitK = ${initK};
__device__ const int InitI = ${initI};
__device__ const int InitJ = ${initJ};
__device__ const float dz = ${deltaZ};
__device__ const float dv = ${deltaV};

class ThreeDimData
{
    __device__
        static void Func(int k, int i, int j, KernelInterval V[K][I][J], const float G[], const float U[Ul0][Ul1]) {
        ${func};
    }

public:
    __device__
        static void Evaluate(const KernelFoodSource& fs, const float* z, const float* u, bool isInterval, float& r)
    {
        KernelInterval Z[K][I][J];

        int index = 0;
        for (int k = 0; k < K; k++) {
            for (int i = 0; i < I; i++) {
                for (int j = 0; j < J; j++) {
                    if (isInterval) {
                        Z[k][i][j] = KernelInterval(z[index++], z[index++]);
                    }
                    else {
```

```
                    Z[k][i][j] = KernelInterval(z[index] * (1 - dz), z[index] * (1 + dz));
                    index++;
                }
            }
        }
    }

    KernelInterval V[K][I][J];

    index = 0;
    for (int k = 0; k < K; k++) {
        for (int i = 0; i < I; i++) {
            for (int j = 0; j < J; j++) {
                if (k < InitK || i < InitI || j < InitJ) {
                    float zm = Z[k][i][j].Middle();
                    V[k][i][j] = KernelInterval(zm * (1 - dv), zm * (1 + dv));
                }
                index++;
            }
        }
    }

    float U[Ul0][Ul1];
    index = 0;
    for (int i = 0; i < Ul0; i++) {
        for (int j = 0; j < Ul1; j++) {
            U[i][j] = u[index++];
        }
    }

    float L[K * I * J];
    index = 0;
    for (int k = 0; k < K; k++) {
        for (int i = 0; i < I; i++) {
            for (int j = 0; j < J; j++) {
                if (k >= InitK && i >= InitI && j >= InitJ) {
                    Func(k, i, j, V, fs.G, U);
                }
                if (KernelInterval::IsIntersect(V[k][i][j], Z[k][i][j])) {
                    L[index] = fabsf(V[k][i][j].Width() - KernelInterval::Intersect(V[k][i][j], Z[k][i][j]).Width());
                }
                else {
                    L[index] = fabsf(V[k][i][j].Middle() - Z[k][i][j].Middle());
                }
                index++;
            }
        }
    }

    r = MathHelper::max(L, K * I * J);
  }
};

__global__ void EvaluateNectar(const KernelFoodSource* fss, const float* dz, const float* du, bool isInterval, float* r,
int N)
{
  int t = blockDim.x * blockIdx.x + threadIdx.x;
  if (t < N)
  {
    ThreeDimData::Evaluate(fss[t], dz, du, isInterval, r[t]);
  }
}
```

ArtificialBeeColonyLibrary.dll

```
using ArtificialBeeColonyLibrary.Data.FoodSourceData;

namespace ArtificialBeeColonyLibrary.Algorithm.FoodSources;

public static class FoodSourceEvaluatorFactory
{
    public static IFoodSourceEvaluator Create(FoodSourceDataBase foodSourceDataBase)
    {
        switch (foodSourceDataBase)
        {
            case OneDimFoodSourceData data:
                return new OneDimFoodSourceNectarEvaluator(data);
            case OneDimFoodSourceIntervalData data:
                return new OneDimFoodSourceNectarEvaluator(data);
            case TwoDimFoodSourceData data:
                return new TwoDimFoodSourceNectarEvaluator(data);
            case TwoDimFoodSourceIntervalData data:
                return new TwoDimFoodSourceNectarEvaluator(data);
            case ThreeDimFoodSourceData data:
                return new ThreeDimFoodSourceNectarEvaluator(data);
            case ThreeDimFoodSourceIntervalData data:
                return new ThreeDimFoodSourceNectarEvaluator(data);
            default:
                throw new Exception("FoodSourceData is not supported.");
        }
    }
}
using ArtificialBeeColonyLibrary.Data;

namespace ArtificialBeeColonyLibrary.Algorithm.FoodSources;

public abstract class FoodSourceNectarEvaluatorBase : IFoodSourceEvaluator
{
    protected int _gLength;
    protected float _deltaZ;
    protected float _deltaV;

    protected int _evaluationCount;
    protected bool _evaluateCountEnabled;

    protected float[,] _u;

    public int EvaluationCount => _evaluationCount;

    protected FoodSourceNectarEvaluatorBase(int gLength, float deltaZ, float deltaV, bool evaluateCountEnabled, float[,] u)
    {
        _gLength = gLength;
        _deltaZ = deltaZ;
        _deltaV = deltaV;
        _evaluateCountEnabled = evaluateCountEnabled;
        _u = u;
    }

    public void ResetEvaluationCount()
    {
        _evaluationCount = 0;
    }

    public void UpdateEvaluationCount(int value)
    {
```

```
        _evaluationCount += value;
    }

    protected bool CheckG(int length)
    {
        return _gLength == length;
    }

    protected float CalculateLambda(Interval v, Interval z)
    {
        if (Interval.IsIntersect(v, z))
        {
            return Math.Abs(v.Width() - Interval.Intersect(v, z).Width());
        }
        else
        {
            return Math.Abs(v.Middle() - z.Middle());
        }
    }

    protected Interval GetZ(float z)
    {
        return new Interval(z * (1 - _deltaZ), z * (1 + _deltaZ));
    }

    protected Interval GetV(Interval z)
    {
        return z.Middle() * new Interval(1 - _deltaV, 1 + _deltaV);
    }

    protected Interval GetVForPlot(Interval z)
    {
        return z.Middle() * new Interval(1, 1);
    }

    public abstract Task<float> EvaluateNectar(float[] g);

    public abstract PlotData GetPlotData(float[] g, bool ignoreInitValues = false);
}

using ArtificialBeeColonyLibrary.Data;

namespace ArtificialBeeColonyLibrary.Algorithm.FoodSources;

public interface IFoodSourceEvaluator
{
    int EvaluationCount { get; }

    Task<float> EvaluateNectar(float[] g);

    void UpdateEvaluationCount(int value);

    PlotData GetPlotData(float[] g, bool ignoreInitValues = false);
}

using ArtificialBeeColonyLibrary.Algorithm.Helpers;
using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;
using Microsoft.CodeAnalysis.CSharp.Scripting;
using Microsoft.CodeAnalysis.Scripting;

namespace ArtificialBeeColonyLibrary.Algorithm.FoodSources;
```

```
public class OneDimFoodSourceNectarEvaluator : FoodSourceNectarEvaluatorBase
{
    private Interval[] _z;
    private int _k;
    private int _initK;
    private Action<int, Interval[], float[], float[,]> _func;

    public OneDimFoodSourceNectarEvaluator(OneDimFoodSourceData foodSourceData) : base(
        foodSourceData.GLength,
        foodSourceData.DeltaZ,
        foodSourceData.DeltaV,
        foodSourceData.EvaluateCountEnabled,
        foodSourceData.U)
    {
        _k = foodSourceData.Z.Length;
        _z = new Interval[_k];
        _initK = foodSourceData.InitK;
        _func = foodSourceData.Func;

        for (var i = 0; i < _k; i++)
        {
            _z[i] = GetZ(foodSourceData.Z[i]);
        }
    }

    public OneDimFoodSourceNectarEvaluator(OneDimFoodSourceIntervalData foodSourceData) : base(
        foodSourceData.GLength,
        foodSourceData.DeltaZ,
        foodSourceData.DeltaV,
        foodSourceData.EvaluateCountEnabled,
        foodSourceData.U)
    {
        _k = foodSourceData.Z.Length;
        _z = foodSourceData.Z;
        _initK = foodSourceData.InitK;
        _func = foodSourceData.Func;
    }

    public override Task<float> EvaluateNectar(float[] g)
    {
        return Task.Factory.StartNew(() =>
        {
            if (!CheckG(g.Length))
            {
                throw new Exception("G length is wrong");
            }

            if (_evaluateCountEnabled)
            {
                Interlocked.Increment(ref _evaluationCount);
            }

            var v = new Interval[_k];
            var l = new float[_k];
            for (var i = 0; i < _k; i++)
            {
                if (i < _initK)
                {
                    v[i] = GetV(_z[i]);
                }
                else
                {
                    _func(i, v, g, _u);
```

```
            }

            l[i] = CalculateLambda(v[i], _z[i]);
        }

        return MathHelper.Max(l);
    });
}

public override PlotData GetPlotData(float[] g, bool ignoreInitValues = false)
{
    var initial = new List<Interval>();
    var calculated = new List<Interval>();

    var v = new Interval[_k];
    for (var i = 0; i < _k; i++)
    {
        if (i < _initK)
        {
            v[i] = GetVForPlot(_z[i]);
        }
        else
        {
            _func(i, v, g, _u);
        }

        if (!ignoreInitValues || (ignoreInitValues && i >= _k))
        {
            initial.Add(_z[i]);
            calculated.Add(v[i]);
        }
    }

    return new PlotData()
    {
        Length = initial.Count,
        Initial = initial.ToArray(),
        Calculated = calculated.ToArray()
    };
}
}

using ArtificialBeeColonyLibrary.Algorithm.Helpers;
using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;

namespace ArtificialBeeColonyLibrary.Algorithm.FoodSources;

public class TwoDimFoodSourceNectarEvaluator : FoodSourceNectarEvaluatorBase
{
    private Interval[,] _z;
    private int _i;
    private int _initI;
    private int _j;
    private int _initJ;
    private Action<int, int, Interval[,], float[], float[,]> _func;

    public TwoDimFoodSourceNectarEvaluator(TwoDimFoodSourceData foodSourceData) : base(
        foodSourceData.GLength,
        foodSourceData.DeltaZ,
        foodSourceData.DeltaV,
        foodSourceData.EvaluateCountEnabled,
        foodSourceData.U)
```

```
{
    _i = foodSourceData.Z.GetLength(0);
    _initI = foodSourceData.InitI;
    _j = foodSourceData.Z.GetLength(1);
    _initJ = foodSourceData.InitJ;
    _z = new Interval[_i, _j];
    _func = foodSourceData.Func;

    for (var i = 0; i < _i; i++)
    {
        for (var j = 0; j < _j; j++)
        {
            _z[i, j] = GetZ(foodSourceData.Z[i, j]);
        }
    }
}

public TwoDimFoodSourceNectarEvaluator(TwoDimFoodSourceIntervalData foodSourceData) : base(
        foodSourceData.GLength,
        foodSourceData.DeltaZ,
        foodSourceData.DeltaV,
        foodSourceData.EvaluateCountEnabled,
        foodSourceData.U)
{
    _i = foodSourceData.Z.GetLength(0);
    _initI = foodSourceData.InitI;
    _j = foodSourceData.Z.GetLength(1);
    _initJ = foodSourceData.InitJ;
    _z = foodSourceData.Z;
    _func = foodSourceData.Func;
}

public override Task<float> EvaluateNectar(float[] g)
{
    return Task.Factory.StartNew(() =>
    {
        if (!CheckG(g.Length))
        {
            throw new Exception("G length is wrong");
        }

        if (_evaluateCountEnabled)
        {
            Interlocked.Increment(ref _evaluationCount);
        }

        var v = new Interval[_i, _j];
        for (var i = 0; i < _i; i++)
        {
            for (var j = 0; j < _j; j++)
            {
                if (i < _initI || j < _initJ)
                {
                    v[i, j] = GetV(_z[i, j]);
                }
            }
        }

        var l = new float[_i * _j];
        var index = 0;
        for (var i = 0; i < _i; i++)
        {
            for (var j = 0; j < _j; j++)
```

```
            {
                if (i >= _initI && j >= _initJ)
                {
                    _func(i, j, v, g, _u);
                }

                l[index] = CalculateLambda(v[i, j], _z[i, j]);
                index++;
            }
        }

        return MathHelper.Max(l);
    });
}

public override PlotData GetPlotData(float[] g, bool ignoreInitValues = false)
{
    var initial = new List<Interval>();
    var calculated = new List<Interval>();

    var v = new Interval[_i, _j];
    for (var i = 0; i < _i; i++)
    {
        for (var j = 0; j < _j; j++)
        {
            if (i < _initI || j < _initJ)
            {
                v[i, j] = GetVForPlot(_z[i, j]);
            }
        }
    }

    var l = new float[_i * _j];
    var index = 0;
    for (var i = 0; i < _i; i++)
    {
        for (var j = 0; j < _j; j++)
        {
            if (i >= _initI && j >= _initJ)
            {
                _func(i, j, v, g, _u);
            }

            if (!ignoreInitValues || (ignoreInitValues && j >= _initJ))
            {
                initial.Add(_z[i, j]);
                calculated.Add(v[i, j]);
            }


            index++;
        }
    }

    return new PlotData()
    {
        Length = initial.Count,
        Initial = initial.ToArray(),
        Calculated = calculated.ToArray(),
    };
}
}
```

```
using ArtificialBeeColonyLibrary.Algorithm.Helpers;
using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;

namespace ArtificialBeeColonyLibrary.Algorithm.FoodSources;

public class ThreeDimFoodSourceNectarEvaluator : FoodSourceNectarEvaluatorBase
{
   private Interval[][,] _z;
   private int _k;
   private int _initK;
   private int _i;
   private int _initI;
   private int _j;
   private int _initJ;
   private Action<int, int, int, Interval[][,], float[], float[,]> _func;

   public ThreeDimFoodSourceNectarEvaluator(ThreeDimFoodSourceData foodSourceData) : base(
      foodSourceData.GLength,
      foodSourceData.DeltaZ,
      foodSourceData.DeltaV,
      foodSourceData.EvaluateCountEnabled,
      foodSourceData.U)
   {
      _k = foodSourceData.Z.Length;
      _initK = foodSourceData.InitK;
      _i = foodSourceData.Z[0].GetLength(0);
      _initI = foodSourceData.InitI;
      _j = foodSourceData.Z[0].GetLength(1);
      _initJ = foodSourceData.InitJ;
      _z = new Interval[_k][,];
      _func = foodSourceData.Func;

      for (var k = 0; k < _k; k++)
      {
         _z[k] = new Interval[_i, _j];
         for (var i = 0; i < _i; i++)
         {
            for (var j = 0; j < _j; j++)
            {
               _z[k][i, j] = GetZ(foodSourceData.Z[k][i, j]);
            }
         }
      }
   }

   public ThreeDimFoodSourceNectarEvaluator(ThreeDimFoodSourceIntervalData foodSourceData) : base(
     foodSourceData.GLength,
     foodSourceData.DeltaZ,
     foodSourceData.DeltaV,
     foodSourceData.EvaluateCountEnabled,
     foodSourceData.U)
   {
      _k = foodSourceData.Z.Length;
      _initK = foodSourceData.InitK;
      _i = foodSourceData.Z[0].GetLength(0);
      _initI = foodSourceData.InitI;
      _j = foodSourceData.Z[0].GetLength(1);
      _initJ = foodSourceData.InitJ;
      _z = foodSourceData.Z;
      _func = foodSourceData.Func;
   }
```

```
public override Task<float> EvaluateNectar(float[] g)
{
    return Task.Factory.StartNew(() =>
    {
        if (!CheckG(g.Length))
        {
            throw new Exception("G length is wrong");
        }

        if (_evaluateCountEnabled)
        {
            Interlocked.Increment(ref _evaluationCount);
        }

        var v = new Interval[_k][,];
        for (var k = 0; k < _k; k++)
        {
            v[k] = new Interval[_i, _j];
        }

        for (var k = 0; k < _k; k++)
        {
            for (var i = 0; i < _i; i++)
            {
                for (var j = 0; j < _j; j++)
                {
                    if (k < _initK || i < _initI || j < _initJ)
                    {
                        v[k][i, j] = GetV(_z[k][i, j]);
                    }
                }
            }
        }

        var l = new float[_k * _i * _j];
        var index = 0;
        for (var k = 0; k < _k; k++)
        {
            for (var i = 0; i < _i; i++)
            {
                for (var j = 0; j < _j; j++)
                {
                    if (k >= _initK && i >= _initI && j >= _initJ)
                    {
                        _func(k, i, j, v, g, _u);
                    }
                    l[index] = CalculateLambda(v[k][i, j], _z[k][i, j]);
                    index++;
                }
            }
        }

        return MathHelper.Max(l);
    });
}

public override PlotData GetPlotData(float[] g, bool ignoreInitValues = false)
{
    var initial = new List<Interval>();
    var calculated = new List<Interval>();

    var v = new Interval[_k][,];
    for (var k = 0; k < _k; k++)
```

```
        {
            v[k] = new Interval[_i, _j];
        }
        for (var k = 0; k < _k; k++)
        {
            for (var i = 0; i < _i; i++)
            {
                for (var j = 0; j < _j; j++)
                {
                    if (k < _initK || i < _initI || j < _initJ)
                    {
                        v[k][i, j] = GetVForPlot(_z[k][i, j]);
                    }
                }
            }
        }

        var l = new float[_k * _i * _j];
        var index = 0;
        for (var k = 0; k < _k; k++)
        {
            for (var i = 0; i < _i; i++)
            {
                for (var j = 0; j < _j; j++)
                {
                    if (k >= _initK && i >= _initI && j >= _initJ)
                    {
                        _func(k, i, j, v, g, _u);
                    }
                    if (!ignoreInitValues || (ignoreInitValues && k >= _initK && i >= _initI && j >= _initJ))
                    {
                        initial.Add(_z[k][i, j]);
                        calculated.Add(v[k][i, j]);
                    }
                    index++;
                }
            }
        }

        return new PlotData()
        {
            Length = initial.Count,
            Initial = initial.ToArray(),
            Calculated = calculated.ToArray()
        };
    }
}

using ArtificialBeeColonyLibrary.Data;

namespace ArtificialBeeColonyLibrary.Algorithm.Helpers;

class FoodSourceGenerator
{
    private FoodSourceGeneratorData _data;

    public FoodSourceGenerator(FoodSourceGeneratorData data)
    {
        _data = data;
    }

    public FoodSource GenerateNewFoodSource()
    {
```

```
        var fs = new FoodSource(_data.FoodSourceLength);
        fs.Initialize(_data.FoodSourceLowerLimit, _data.FoodSourceUpperLimit);
        return fs;
    }

    public FoodSource[] GenerateNewFoodSources()
    {
        var newFoodSources = new FoodSource[_data.FoodSourcesCount];
        for (var i = 0; i < _data.FoodSourcesCount; i++)
        {
            newFoodSources[i] = GenerateNewFoodSource();
        }
        return newFoodSources;
    }

    public FoodSource[] GenerateNeighbourFoodSources(FoodSource[] foodSources)
    {
        var newFoodSources = new FoodSource[_data.FoodSourcesCount];
        for (var i = 0; i < _data.FoodSourcesCount; i++)
        {
            newFoodSources[i] = GenerateNeighbourFoodSource(foodSources, i);
        }
        return newFoodSources;
    }

    public FoodSource[] GenerateNeighbourFoodSources(FoodSource[] foodSources, int index, int m)
    {
        var newFoodSources = new FoodSource[m];

        for (var i = 0; i < m; i++)
        {
            newFoodSources[i] = GenerateNeighbourFoodSource(foodSources, index);
        }

        return newFoodSources;
    }

    public FoodSource GenerateNeighbourFoodSource(FoodSource[] foodSources, int fsNumber)
    {
        var nextFsNumber = GetNeighbourFoodSourceNumber(fsNumber,_data.FoodSourcesCount);
        var fs = new FoodSource(_data.FoodSourceLength);
        fs.Initialize(
            foodSources[fsNumber].G,
            foodSources[nextFsNumber].G,
            _data.FoodSourceLowerLimit,
            _data.FoodSourceUpperLimit);
        return fs;
    }

    private int GetNeighbourFoodSourceNumber(int fsNumber, int fsCount)
    {
        int number;
        do
        {
            number = Random.Shared.Next(0, fsCount);
        } while (number == fsNumber);

        return number;
    }
}

namespace ArtificialBeeColonyLibrary.Algorithm.Helpers;
```

```
static class ScoutBeeCountEvaluator
{
   public static int[] Evaluate(float[] foodSources)
   {
      var result = new int[foodSources.Length];
      var normalizedNectar = Normalize(foodSources);
      var sum = normalizedNectar.Sum(n => 1 - n);
      for (var i = 0; i < foodSources.Length; i++)
      {
         var count = foodSources.Length * (1 - normalizedNectar[i]) / sum;
         result[i] = (int)Math.Round(count);
      }

      return result;
   }

   public static int[] Evaluate2(float[] foodSources)
   {
      var result = new int[foodSources.Length];
      var sum = foodSources.Sum(n => 1 / n);
      for (var i = 0; i < foodSources.Length; i++)
      {
         var p = 1 / (foodSources[i] * sum);
         var count = foodSources.Length * p;
         result[i] = (int)Math.Round(count);
      }

      return result;
   }

   private static float[] Normalize(float[] fss)
   {
      var min = fss.Min();
      var max = fss.Max();
      return fss.Select(fs => (fs - min) / (max - min)).ToArray();
   }
}

using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Utils;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ArtificialBeeColonyLibrary.Algorithm.Helpers
{
   internal class StructuralFoodSourceGenerator
   {
      private int _foodSourceCount;
      private int _dim;
      private int _minI;
      private int _maxI;
      private List<string> _f;
      private StringBuilder _sb = new StringBuilder();

      public StructuralFoodSourceGenerator(int foodSourceCount, int minI, int maxI, int dim, List<string> f)
      {
         _foodSourceCount = foodSourceCount;
         _minI = minI;
         _maxI = maxI;
         _dim = dim;
```

```
        _f = f;
    }

    public StructuralFoodSource[] GenerateNewFoodSources()
    {
        var result = new StructuralFoodSource[_foodSourceCount];
        for (var i = 0; i < _foodSourceCount; i++)
        {
            var indexes = GetRandomIndexes(new SortedSet<int>(), GetRandomI());
            result[i] = CreateNewFoodSource(indexes);
        }
        return result;
    }

    public StructuralFoodSource GenerateNewFoodSource()
    {
        var indexes = GetRandomIndexes(new SortedSet<int>(), GetRandomI());
        return CreateNewFoodSource(indexes);
    }

    internal StructuralFoodSource[] GenerateNeighbourFoodSources(StructuralFoodSource[] foodSources)
    {
        var newFoodSources = new StructuralFoodSource[_foodSourceCount];

        var minDelta = foodSources.Min(fs => fs.FoodSource.Delta);

        for (var i = 0; i < _foodSourceCount; i++)
        {
            newFoodSources[i] = GenerateNeighbourFoodSource(foodSources[i], minDelta);
        }

        return newFoodSources;
    }

    internal StructuralFoodSource[] GenerateNeighbourFoodSources(StructuralFoodSource[] foodSources, int index, int
m)
    {
        var newFoodSources = new StructuralFoodSource[m];

        var minDelta = foodSources.Min(fs => fs.FoodSource.Delta);

        for (var i = 0; i < m; i++)
        {
            newFoodSources[i] = GenerateNeighbourFoodSource(foodSources[index], minDelta);
        }

        return newFoodSources;
    }

    private StructuralFoodSource GenerateNeighbourFoodSource(StructuralFoodSource foodSource, float minDelta)
    {
        var tempN = Math.Round((1 - minDelta / foodSource.FoodSource.Delta) * foodSource.Indexes.Count);
        var n = tempN == 0 ? 1 : (int)tempN;
        var indexesToRemove = SelectIndexesToRemove(foodSource.Indexes, n);
        var indexes = GetRandomIndexes(new SortedSet<int>(foodSource.Indexes), foodSource.Indexes.Count + n);
        indexes.RemoveAll(indexesToRemove.Contains);
        return CreateNewFoodSource(indexes);
    }

    private List<int> SelectIndexesToRemove(List<int> indexes, int n)
    {
        if (indexes.Count == n)
            return new List<int>(indexes);
```

```
    var indexesToRemove = new List<int>();
    while (indexesToRemove.Count < n)
    {
        var index = indexes[Random.Shared.Next(indexes.Count)];
        if (!indexesToRemove.Contains(index))
        {
            indexesToRemove.Add(index);
        }
    }
    return indexesToRemove;
}

private StructuralFoodSource CreateNewFoodSource(List<int> indexes)
{
    _sb.Clear();
    _sb.Append(GetV());
    for (var j = 0; j < indexes.Count; j++)
    {
        _sb.Append($"+G[{j + 1}]*{_f[indexes[j]]}");
    }
    return new StructuralFoodSource()
    {
        Indexes = indexes,
        Function = _sb.ToString()
    };
}

private int GetRandomI()
{
    return Random.Shared.Next(_minI, _maxI + 1);
}

private List<int> GetRandomIndexes(SortedSet<int> sortedSet, int count)
{
    if (count > _f.Count)
    {
        return sortedSet.ToList();
    }
    while (sortedSet.Count < count)
    {
        var index = Random.Shared.Next(_f.Count);
        if (!sortedSet.Contains(index))
        {
            sortedSet.Add(index);
        }
    }
    return sortedSet.ToList();
}

private string GetV()
{
    switch (_dim)
    {
        case 1: return "V[k]=G[0]";
        case 2: return "V[i,j]=G[0]";
        case 3: return "V[k][i,j]=G[0]";
        default: return string.Empty;
    }
}

private int GetNeighbourFoodSourceNumber(int fsNumber, int fsCount)
{
    int number;
```

```
            do
            {
                number = Random.Shared.Next(0, fsCount);
            } while (number == fsNumber);

            return number;
        }
    }
}


using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;
using ManagedCuda;

namespace ArtificialBeeColonyLibrary.Algorithm.NectarEvaluators;

public class CudaNectarEvaluator<T> : NectarEvaluatorBase where T : struct, IKernelFoodSource
{
    private CudaContext _cudaContext;
    private CudaKernel _evaluateNectar;
    private FoodSourceDataBase _foodSourceData;
    private int _threadsPerBlock;
    private byte[] _ptx;

    public CudaNectarEvaluator(FoodSourceDataBase foodSourceData, byte[] ptx, int threadsPerBlock)
        : base(foodSourceData)
    {
        _foodSourceData = foodSourceData;
        _threadsPerBlock = threadsPerBlock;
        _ptx = ptx;
        _cudaContext = new CudaContext();
        if (_ptx != null)
        {
            _evaluateNectar = _cudaContext.LoadKernelPTX(_ptx, "EvaluateNectar");
        }
    }

    public override async Task Evaluate(FoodSource[] foodSources)
    {
        if (foodSources.Length == 0) return;
        _foodSourceEvaluator.UpdateEvaluationCount(foodSources.Length);
        var nectar = await Task.Factory.StartNew(() =>
        {
            return EvaluateInternal(foodSources);
        });
        for (var i = 0; i < foodSources.Length; i++)
        {
            foodSources[i].SetNectar(nectar[i]);
        }
    }

    private unsafe float[] EvaluateInternal(FoodSource[] foodSources)
    {
        _cudaContext.SetCurrent();
        var n = foodSources.Length;
        var h_A = new T[n];
        for (var i = 0; i < n; i++)
        {
            for (var j = 0; j < _foodSourceData.GLength; j++)
            {
                h_A[i].SetG(j, foodSources[i].G[j]);
            }
```

```
        }

        CudaDeviceVariable<T> d_Fss = h_A;
        CudaDeviceVariable<float> d_Za = _foodSourceData.GetFlatZ();
        CudaDeviceVariable<float> d_Ua = _foodSourceData.GetFlatU();
        CudaDeviceVariable<float> d_R = new CudaDeviceVariable<float>(n);

        _evaluateNectar.BlockDimensions = _threadsPerBlock;
        _evaluateNectar.GridDimensions = (n + _threadsPerBlock - 1) / _threadsPerBlock;

        _evaluateNectar.Run(d_Fss.DevicePointer, d_Za.DevicePointer, d_Ua.DevicePointer,
_foodSourceData.IsIntervalData, d_R.DevicePointer, n);

        float[] h_R = d_R;

        d_Fss.Dispose();
        d_Za.Dispose();
        d_Ua.Dispose();
        d_R.Dispose();

        return h_R;
    }

    public override void Dispose()
    {
        base.Dispose();
        _cudaContext.Dispose();
    }
}

using ArtificialBeeColonyLibrary.Data.FoodSourceData;
using ArtificialBeeColonyLibrary.Data;

namespace ArtificialBeeColonyLibrary.Algorithm.NectarEvaluators
{
    internal static class CudaNectarEvaluatorFactory
    {
        public static INectarEvaluator Create(BeeColonyData beeColonyData, FoodSourceDataBase foodSourceData)
        {
            switch (foodSourceData.GLength)
            {
                case 1:
                    return new CudaNectarEvaluator<KernelFoodSource1>(foodSourceData, beeColonyData.Ptx,
beeColonyData.CoresNumber);
                case 2:
                    return new CudaNectarEvaluator<KernelFoodSource2>(foodSourceData, beeColonyData.Ptx,
beeColonyData.CoresNumber);
                case 3:
                    return new CudaNectarEvaluator<KernelFoodSource3>(foodSourceData, beeColonyData.Ptx,
beeColonyData.CoresNumber);
                case 4:
                    return new CudaNectarEvaluator<KernelFoodSource4>(foodSourceData, beeColonyData.Ptx,
beeColonyData.CoresNumber);
                case 5:
                    return new CudaNectarEvaluator<KernelFoodSource5>(foodSourceData, beeColonyData.Ptx,
beeColonyData.CoresNumber);
                case 6:
                    return new CudaNectarEvaluator<KernelFoodSource6>(foodSourceData, beeColonyData.Ptx,
beeColonyData.CoresNumber);
                case 7:
                    return new CudaNectarEvaluator<KernelFoodSource7>(foodSourceData, beeColonyData.Ptx,
beeColonyData.CoresNumber);
                case 8:
```

```
                return new CudaNectarEvaluator<KernelFoodSource8>(foodSourceData, beeColonyData.Ptx,
beeColonyData.CoresNumber);
            case 9:
                return new CudaNectarEvaluator<KernelFoodSource9>(foodSourceData, beeColonyData.Ptx,
beeColonyData.CoresNumber);
            case 10:
                return new CudaNectarEvaluator<KernelFoodSource10>(foodSourceData, beeColonyData.Ptx,
beeColonyData.CoresNumber);
            default:
                throw new Exception("NectarEvaluator creating failed.");
        }
    }
  }
}

using ArtificialBeeColonyLibrary.Data;

namespace ArtificialBeeColonyLibrary.Algorithm.NectarEvaluators;

public interface INectarEvaluator : IDisposable
{
  int EvaluationCount { get; }
  Task Evaluate(FoodSource[] foodSources);
  Task EvaluateSingle(FoodSource foodSource);
  PlotData GetPlotData(float[] g, bool ignoreInitValues = false);
}

using ArtificialBeeColonyLibrary.Algorithm.FoodSources;
using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;

namespace ArtificialBeeColonyLibrary.Algorithm.NectarEvaluators;

public abstract class NectarEvaluatorBase : INectarEvaluator
{
  protected readonly IFoodSourceEvaluator _foodSourceEvaluator;

  protected NectarEvaluatorBase(FoodSourceDataBase foodSourceData)
  {
    _foodSourceEvaluator = FoodSourceEvaluatorFactory.Create(foodSourceData);
  }

  public int EvaluationCount => _foodSourceEvaluator.EvaluationCount;

  public virtual void Dispose()
  {
  }

  public abstract Task Evaluate(FoodSource[] foodSources);

  public async Task EvaluateSingle(FoodSource foodSource)
  {
    var nectar = await _foodSourceEvaluator.EvaluateNectar(foodSource.G);
    foodSource.SetNectar(nectar);
  }

  public PlotData GetPlotData(float[] g, bool ignoreInitValues = false)
  {
    return _foodSourceEvaluator.GetPlotData(g, ignoreInitValues);
  }
}

using ArtificialBeeColonyLibrary.Data;
```

```
using ArtificialBeeColonyLibrary.Data.FoodSourceData;

namespace ArtificialBeeColonyLibrary.Algorithm.NectarEvaluators;

internal static class NectarEvaluatorFactory
{
   public static INectarEvaluator Create(BeeColonyData beeColonyData, FoodSourceDataBase foodSourceData)
   {
      switch (beeColonyData.ParallelOption)
      {
         case ParallelOption.No:
            return new SequentialNectarEvaluator(foodSourceData);
         case ParallelOption.Cpu:
            return new ParallelNectarEvaluator(foodSourceData, beeColonyData.CoresNumber);
         case ParallelOption.Cuda:
            return CudaNectarEvaluatorFactory.Create(beeColonyData, foodSourceData);
         default:
            throw new Exception("NectarEvaluator creating failed.");
      }
   }
}

using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;

namespace ArtificialBeeColonyLibrary.Algorithm.NectarEvaluators;

public class ParallelNectarEvaluator : NectarEvaluatorBase
{
   private int _coresCount;

   public ParallelNectarEvaluator(FoodSourceDataBase foodSourceData, int coresCount)
      : base(foodSourceData)
   {
      _coresCount = coresCount;
   }

   public override async Task Evaluate(FoodSource[] foodSources)
   {
      var po = new ParallelOptions() { MaxDegreeOfParallelism = _coresCount };
      var size = foodSources.Length;
      var chunkSize = (int)Math.Ceiling((float)size / _coresCount);
      await Parallel.ForAsync(0, _coresCount, po, async (i, token) =>
      {
         for (var j = i * chunkSize; j < (i + 1) * chunkSize; j++)
         {
            if (j < size)
            {
               var nectar = await _foodSourceEvaluator.EvaluateNectar(foodSources[j].G);
               foodSources[j].SetNectar(nectar);
            }
         }
      });
   }
}

using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;

namespace ArtificialBeeColonyLibrary.Algorithm.NectarEvaluators;

public class SequentialNectarEvaluator : NectarEvaluatorBase
{
```

```
    public SequentialNectarEvaluator(FoodSourceDataBase foodSourceDataBase)
        : base(foodSourceDataBase)
    {
    }

    public override async Task Evaluate(FoodSource[] foodSources)
    {
        foreach (var fs in foodSources)
        {
            var nectar = await _foodSourceEvaluator.EvaluateNectar(fs.G);
            fs.SetNectar(nectar);
        }
    }
}

using ArtificialBeeColonyLibrary.Algorithm.Helpers;
using ArtificialBeeColonyLibrary.Algorithm.NectarEvaluators;
using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;
using ArtificialBeeColonyLibrary.Utils;
using System.Threading;

namespace ArtificialBeeColonyLibrary.Algorithm;

public class BeeColony : IDisposable
{
    private BeeColonyData _beeColonyData;
    private FoodSourceGenerator _foodSourceGenerator;
    private INectarEvaluator _nectarEvaluator;
    private BeeColonyResult _beeColonyResult;
    private FoodSourceDataBase _foodSourceData;
    private FoodSource[] _foodSources;

    public BeeColony(
        BeeColonyData beeColonyData,
        FoodSourceDataBase foodSourceData,
        FoodSourceGeneratorData foodSourceGeneratorData)
    {
        _beeColonyData = beeColonyData;
        foodSourceGeneratorData.FoodSourcesCount = beeColonyData.S;
        foodSourceGeneratorData.FoodSourceLength = foodSourceData.GLength;
        _foodSourceGenerator = new FoodSourceGenerator(foodSourceGeneratorData);
        _beeColonyResult = new BeeColonyResult();
        _foodSourceData = foodSourceData;
    }

    public async Task<bool> InitializeAsync()
    {
        var populatedWithFunc = await ModelBuilder.PopulateWithFunc(_foodSourceData, _beeColonyData.Func);
        if (!populatedWithFunc)
        {
            return false;
        }

        if (_beeColonyData.ParallelOption == ParallelOption.Cuda)
        {
            var populatedWithPtx = await CudaModelBuilder.PopulateWithPtx(_beeColonyData, _foodSourceData);
            if (!populatedWithPtx) return false;
        }

        _nectarEvaluator = NectarEvaluatorFactory.Create(_beeColonyData, _foodSourceData);
        return true;
    }
```

```
    public async Task<BeeColonyResult> Run(CancellationToken cancellationToken)
    {
        try
        {
            var runUntilFound = _beeColonyData.Mcn == -1;
            await InitializeFoodSources();
            var bestFoodSource = FindBestFoodSource();
            var mcn = 0;
            while (runUntilFound || mcn < _beeColonyData.Mcn)
            {
                mcn++;
                await SendEmployedBees();
                if (cancellationToken.IsCancellationRequested)
                {
                    break;
                }

                await SendOnlookerBees();
                if (cancellationToken.IsCancellationRequested)
                {
                    break;
                }

                var newBestFoodSource = FindBestFoodSource();
                if (newBestFoodSource.Id != bestFoodSource.Id && Math.Abs(newBestFoodSource.Delta -
bestFoodSource.Delta) < _beeColonyData.RelativeDeltaCap)
                {
                    break;
                }

                bestFoodSource = newBestFoodSource;
                if (bestFoodSource.Delta <= _beeColonyData.DeltaCap)
                {
                    break;
                }
                await SendScoutBees();
                if (cancellationToken.IsCancellationRequested)

                {
                    break;
                }
            }

            _beeColonyResult.Cycle = mcn;
            _beeColonyResult.FoodSource = bestFoodSource;
            _beeColonyResult.EvaluationCount = _nectarEvaluator.EvaluationCount;

            return _beeColonyResult;
        }
        catch (Exception ex)
        {
            _beeColonyResult.ErrorMessage = ex.Message;
            return _beeColonyResult;
        }
    }

    private async Task InitializeFoodSources()
    {
        using (AbcLogger.LogTime("Initialization phase:"))
        {
            _foodSources = _foodSourceGenerator.GenerateNewFoodSources();
            await _nectarEvaluator.Evaluate(_foodSources);
```

```
    }
}

private FoodSource FindBestFoodSource()
{
    var bestFoodSource = _foodSources[0];
    for (var i = 1; i < _beeColonyData.S; i++)
    {
        if (_foodSources[i].Delta < bestFoodSource.Delta)
        {
            bestFoodSource = _foodSources[i];
        }
    }

    AbcLogger.Log($"Best delta: {bestFoodSource.Delta}");

    if (_beeColonyData.ResetLimit)
    {
        bestFoodSource.ResetLimit();
    }

    return bestFoodSource;
}

private async Task SendEmployedBees()
{
    using (AbcLogger.LogTime("Employed Bee phase:"))
    {
        var newFoodSources = _foodSourceGenerator.GenerateNeighbourFoodSources(_foodSources);
        await _nectarEvaluator.Evaluate(newFoodSources);

        for (var i = 0; i < _beeColonyData.S; i++)
        {
            if (newFoodSources[i].Delta < _foodSources[i].Delta)
            {
                _foodSources[i] = newFoodSources[i].DeepClone();
            }
        }
    }
}

private async Task SendOnlookerBees()
{
    using (AbcLogger.LogTime("Onlooker Bee phase:"))
    {
        var scoutBeeCount = ScoutBeeCountEvaluator.Evaluate(_foodSources.Select(fs => fs.Delta).ToArray());

        var listOfSource = new List<FoodSource>();

        for (var i = 0; i < _foodSources.Length; i++)
        {
            var m = scoutBeeCount[i];
            _foodSources[i].M = m;
            if (m <= 0) continue;
            listOfSource.AddRange(_foodSourceGenerator.GenerateNeighbourFoodSources(_foodSources, i, m));
        }

        var preparedFoodSources = listOfSource.ToArray();
        await _nectarEvaluator.Evaluate(preparedFoodSources);

        var counter = 0;
        for (var i = 0; i < _foodSources.Length; i++)
        {
```

```
                var m = _foodSources[i].M;
                if (m <= 0) continue;

                var wasChanged = false;
                for (var j = 0; j < m; j++)
                {
                    if (preparedFoodSources[counter].Delta < _foodSources[i].Delta)
                    {
                        wasChanged = true;
                        _foodSources[i] = preparedFoodSources[counter].DeepClone();
                    }
                    counter++;
                }

                if (!wasChanged)
                {
                    _foodSources[i].IncrementLimit();
                }
            }
        }
    }

    private async Task SendScoutBees()
    {
        using (AbcLogger.LogTime($"Scout Bee phase:"))
        {
            for (var i = 0; i < _foodSources.Length; i++)
            {
                if (_foodSources[i].Limit > _beeColonyData.Limit)
                {
                    _foodSources[i] = _foodSourceGenerator.GenerateNewFoodSource();
                    await _nectarEvaluator.EvaluateSingle(_foodSources[i]);
                }
            }
        }
    }

    public void Dispose()
    {
        _nectarEvaluator?.Dispose();
    }
}


using ArtificialBeeColonyLibrary.Data.FoodSourceData;
using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Algorithm.NectarEvaluators;
using ArtificialBeeColonyLibrary.Utils;
using System.Threading;
using ArtificialBeeColonyLibrary.Algorithm.Helpers;

namespace ArtificialBeeColonyLibrary.Algorithm
{
    public class StructuralBeeColony : IDisposable
    {
        private StructuralBeeColonyData _structuralBeeColonyData;
        private BeeColonyData _parametricBeeColonyData;
        private FoodSourceDataBase _foodSourceData;
        private FoodSourceGeneratorData _foodSourceGeneratorData;
        private StructuralFoodSourceGenerator _structuralFoodSourceGenerator;
        private BeeColonyResult _beeColonyResult;

        private StructuralFoodSource[] _foodSources;
```

```
    public StructuralBeeColony(
        StructuralBeeColonyData structuralBeeColonyData,
        BeeColonyData parametricBeeColonyData,
        FoodSourceDataBase foodSourceData,
        FoodSourceGeneratorData foodSourceGeneratorData)
    {
        _structuralBeeColonyData = structuralBeeColonyData;
        _parametricBeeColonyData = parametricBeeColonyData;
        _foodSourceData = foodSourceData;
        _foodSourceGeneratorData = foodSourceGeneratorData;
        _structuralFoodSourceGenerator = new StructuralFoodSourceGenerator(
            _structuralBeeColonyData.S,
            _structuralBeeColonyData.MinI,
            _structuralBeeColonyData.MaxI,
            _structuralBeeColonyData.Dim,
            _structuralBeeColonyData.F
            );
        _beeColonyResult = new BeeColonyResult();
    }

    public async Task<BeeColonyResult> Run(CancellationToken cancellationToken)
    {
        try
        {
            var runUntilFound = _structuralBeeColonyData.Mcn == -1;
            await InitializeFoodSources(cancellationToken);
            if (cancellationToken.IsCancellationRequested)
            {
                return _beeColonyResult;
            }
            var bestFoodSource = FindBestFoodSource();
            var mcn = 0;
            while (runUntilFound || mcn < _structuralBeeColonyData.Mcn)
            {
                if (cancellationToken.IsCancellationRequested)
                {
                    break;
                }

                mcn++;
                await SendEmployedBees(cancellationToken);
                if (cancellationToken.IsCancellationRequested)
                {
                    break;
                }

                await SendOnlookerBees(cancellationToken);
                if (cancellationToken.IsCancellationRequested)
                {
                    break;
                }

                var newBestFoodSource = FindBestFoodSource();
                if (newBestFoodSource.Id != bestFoodSource.Id && Math.Abs(newBestFoodSource.FoodSource.Delta -
bestFoodSource.FoodSource.Delta) < _structuralBeeColonyData.RelativeDeltaCap)
                {
                    break;
                }

                bestFoodSource = newBestFoodSource;
                if (bestFoodSource.FoodSource.Delta <= _structuralBeeColonyData.DeltaCap)
                {
```

```
                break;
            }
            await SendScoutBees(cancellationToken);
        }

        _beeColonyResult.Cycle = mcn;
        _beeColonyResult.StructuralFoodSource = bestFoodSource;

        return _beeColonyResult;
      }
      catch (Exception ex)
      {
        _beeColonyResult.ErrorMessage = ex.Message;
        return _beeColonyResult;
      }

    }

    private async Task SendOnlookerBees(CancellationToken cancellationToken)
    {
      using (AbcLogger.LogTime("Onlooker Bee phase:"))
      {
        var scoutBeeCount = ScoutBeeCountEvaluator.Evaluate(_foodSources.Select(fs =>
fs.FoodSource.Delta).ToArray());

        var listOfSource = new List<StructuralFoodSource>();

        for (var i = 0; i < _foodSources.Length; i++)
        {
          var m = scoutBeeCount[i];
          _foodSources[i].M = m;
          if (m <= 0) continue;
          listOfSource.AddRange(_structuralFoodSourceGenerator.GenerateNeighbourFoodSources(_foodSources, i,
m));
        }

        var preparedFoodSources = listOfSource.ToArray();
        for (var i = 0; i < preparedFoodSources.Length; i++)
        {
          var result = await RunBeeColony(preparedFoodSources[i].Function, preparedFoodSources[i].Indexes.Count +
1, cancellationToken);
          if (cancellationToken.IsCancellationRequested)
          {
            return;
          }
          preparedFoodSources[i].SetNectar(result.FoodSource);
        }

        var counter = 0;
        for (var i = 0; i < _foodSources.Length; i++)
        {
          var m = _foodSources[i].M;
          if (m <= 0)
          {
            _foodSources[i].IncrementLimit();
            continue;
          }

          var wasChanged = false;
          for (var j = 0; j < m; j++)
          {
            if (preparedFoodSources[counter].FoodSource.Delta < _foodSources[i].FoodSource.Delta)
            {
```

```
                    wasChanged = true;
                    _foodSources[i] = preparedFoodSources[counter].DeepClone();
                }
                counter++;
            }

            if (!wasChanged)
            {
                _foodSources[i].IncrementLimit();
            }
        }
    }

    private async Task SendScoutBees(CancellationToken cancellationToken)
    {
        using (AbcLogger.LogTime($"Scout Bee phase:"))
        {
            for (var i = 0; i < _foodSources.Length; i++)
            {
                if (_foodSources[i].Limit > _structuralBeeColonyData.Limit)
                {
                    _foodSources[i] = _structuralFoodSourceGenerator.GenerateNewFoodSource();
                    var result = await RunBeeColony(_foodSources[i].Function, _foodSources[i].Indexes.Count + 1,
cancellationToken);
                    if (cancellationToken.IsCancellationRequested)
                    {
                        return;
                    }
                    _foodSources[i].SetNectar(result.FoodSource);
                }
            }
        }
    }

    private StructuralFoodSource FindBestFoodSource()
    {
        var bestFoodSource = _foodSources[0];
        for (var i = 1; i < _foodSources.Length; i++)
        {
            if (_foodSources[i].FoodSource.Delta < bestFoodSource.FoodSource.Delta)
            {
                bestFoodSource = _foodSources[i];
            }
        }

        return bestFoodSource;
    }

    private async Task InitializeFoodSources(CancellationToken cancellationToken)
    {
        using (AbcLogger.LogTime("Initialization phase:"))
        {
            _foodSources = _structuralFoodSourceGenerator.GenerateNewFoodSources();
            for (var i = 0; i < _foodSources.Length; i++)
            {
                var result = await RunBeeColony(_foodSources[i].Function, _foodSources[i].Indexes.Count + 1,
cancellationToken);
                if (cancellationToken.IsCancellationRequested)
                {
                    return;
                }
                _foodSources[i].SetNectar(result.FoodSource);
```

```
            }
        }

    }

    private async Task<BeeColonyResult> RunBeeColony(string function, int gLength, CancellationToken
cancellationToken)
    {
        _parametricBeeColonyData.Func = function;
        _foodSourceData.GLength = gLength;
        using (var beeColony = new BeeColony(_parametricBeeColonyData, _foodSourceData,
_foodSourceGeneratorData))
        {
            await beeColony.InitializeAsync();
            return await beeColony.Run(cancellationToken);
        }
    }

    private async Task SendEmployedBees(CancellationToken cancellationToken)
    {
        using (AbcLogger.LogTime("Employed Bee phase:"))
        {
            var newFoodSources = _structuralFoodSourceGenerator.GenerateNeighbourFoodSources(_foodSources);
            for (var i = 0; i < newFoodSources.Length; i++)
            {
                var result = await RunBeeColony(newFoodSources[i].Function, newFoodSources[i].Indexes.Count + 1,
cancellationToken);
                if (cancellationToken.IsCancellationRequested)
                {
                    return;
                }
                newFoodSources[i].SetNectar(result.FoodSource);
            }

            for (var i = 0; i < _structuralBeeColonyData.S; i++)
            {
                if (newFoodSources[i].FoodSource.Delta < _foodSources[i].FoodSource.Delta)
                {
                    _foodSources[i] = newFoodSources[i].DeepClone();
                }
            }
        }
    }

    public void Dispose()
    {
    }
  }
}

namespace ArtificialBeeColonyLibrary.Data.FoodSourceData;

public abstract class FoodSourceDataBase
{
  public float[,] U { get; private set; }
  public int GLength { get; internal set; }
  public float DeltaZ { get; }
  public float DeltaV { get; }
  public bool EvaluateCountEnabled { get; }
  public abstract bool IsIntervalData { get; }

  protected FoodSourceDataBase(int gLength, float deltaZ, float deltaV, bool evaluateCountEnabled)
  {
```

```
        GLength = gLength;
        DeltaZ = deltaZ;
        DeltaV = deltaV;
        EvaluateCountEnabled = evaluateCountEnabled;
    }

    public void SetControlParams(float[,] u)
    {
        U = u;
    }

    public float[] GetFlatU()
    {
        if (U == null)
        {
            return new float[] { 1 };
        };

        var rows = U.GetLength(0);
        var cols = U.GetLength(1);

        var flatU = new float[rows * cols];

        int index = 0;
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                flatU[index] = U[i, j];
                index++;
            }
        }

        return flatU;
    }

    public abstract float[] GetFlatZ();
}

namespace ArtificialBeeColonyLibrary.Data.FoodSourceData;

public class OneDimFoodSourceData : FoodSourceDataBase
{
    public float[] Z { get; }
    public int InitK { get; }
    public Action<int, Interval[], float[], float[,]> Func { get; internal set; }

    public override bool IsIntervalData => false;

    public OneDimFoodSourceData(
        int gLength,
        float deltaZ,
        float deltaV,
        float[] z,
        int initK,
        bool evaluateCountEnabled = false) : base(gLength, deltaZ, deltaV, evaluateCountEnabled)
    {
        Z = z;
        InitK = initK;
    }

    public override float[] GetFlatZ()
    {
```

```
        return Z;
    }
}

namespace ArtificialBeeColonyLibrary.Data.FoodSourceData;

public class OneDimFoodSourceIntervalData : FoodSourceDataBase
{
    public Interval[] Z { get; }
    public int InitK { get; }
    public Action<int, Interval[], float[], float[,]> Func { get; internal set; }

    public override bool IsIntervalData => true;

    public OneDimFoodSourceIntervalData(
        int gLength,
        float deltaZ,
        float deltaV,
        Interval[] z,
        int initK,
        bool evaluateCountEnabled = false) : base(gLength, deltaZ, deltaV, evaluateCountEnabled)
    {
        Z = z;
        InitK = initK;
    }

    public override float[] GetFlatZ()
    {
        var flatZ = new float[Z.Length * 2];
        var index = 0;
        foreach (var z in Z)
        {
            flatZ[index++] = z.Left;
            flatZ[index++] = z.Right;
        }
        return flatZ;
    }
}

namespace ArtificialBeeColonyLibrary.Data.FoodSourceData;

public class ThreeDimFoodSourceData : FoodSourceDataBase
{
    public float[][,] Z { get; }
    public int InitK { get; }
    public int InitI { get; }
    public int InitJ { get; }
    public Action<int, int, int, Interval[][,], float[], float[,]> Func { get; internal set; }

    public override bool IsIntervalData => false;

    public ThreeDimFoodSourceData(
        int gLength,
        float deltaZ,
        float deltaV,
        float[][,] z,
        int initK,
        int initI,
        int initJ,
        bool evaluateCountEnabled = false) : base(gLength, deltaZ, deltaV, evaluateCountEnabled)
    {
        Z = z;
        InitK = initK;
```

```
        InitI = initI;
        InitJ = initJ;
    }

    public override float[] GetFlatZ()
    {
        var tables = Z.Length;
        var rows = Z[0].GetLength(0);
        var cols = Z[0].GetLength(1);

        var flatZ = new float[tables * rows * cols];

        int index = 0;
        for (int k = 0; k < tables; k++)
        {
            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    flatZ[index] = Z[k][i, j];
                    index++;
                }
            }
        }

        return flatZ;
    }
}


namespace ArtificialBeeColonyLibrary.Data.FoodSourceData;

public class ThreeDimFoodSourceIntervalData : FoodSourceDataBase
{
    public Interval[][,] Z { get; }
    public int InitK { get; }
    public int InitI { get; }
    public int InitJ { get; }
    public Action<int, int, int, Interval[][,], float[], float[,]> Func { get; internal set; }

    public override bool IsIntervalData => true;

    public ThreeDimFoodSourceIntervalData(
        int gLength,
        float deltaZ,
        float deltaV,
        Interval[][,] z,
        int initK,
        int initI,
        int initJ,
        bool evaluateCountEnabled = false) : base(gLength, deltaZ, deltaV, evaluateCountEnabled)
    {
        Z = z;
        InitK = initK;
        InitI = initI;
        InitJ = initJ;
    }

    public override float[] GetFlatZ()
    {
        var tables = Z.Length;
        var rows = Z[0].GetLength(0);
        var cols = Z[0].GetLength(1);
```

```
        var flatZ = new float[tables * rows * cols * 2];

        int index = 0;
        for (int k = 0; k < tables; k++)
        {
            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    flatZ[index++] = Z[k][i, j].Left;
                    flatZ[index++] = Z[k][i, j].Right;
                }
            }
        }

        return flatZ;
    }
}

namespace ArtificialBeeColonyLibrary.Data.FoodSourceData;

public class TwoDimFoodSourceData : FoodSourceDataBase
{
    public float[,] Z { get; set; }
    public int InitI { get; set; }
    public int InitJ { get; set; }
    public Action<int, int, Interval[,], float[], float[,]> Func { get; set; }
    public override bool IsIntervalData => false;

    public TwoDimFoodSourceData(
        int gLength,
        float deltaZ,
        float deltaV,
        float[,] z,
        int initI,
        int initJ,
        bool evaluateCountEnabled = false) : base(gLength, deltaZ, deltaV, evaluateCountEnabled)
    {
        Z = z;
        InitI = initI;
        InitJ = initJ;
    }

    public override float[] GetFlatZ()
    {
        var rows = Z.GetLength(0);
        var cols = Z.GetLength(1);

        var flatZ = new float[rows * cols];

        int index = 0;
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                flatZ[index] = Z[i, j];
                index++;
            }
        }

        return flatZ;
    }
```

```
}

namespace ArtificialBeeColonyLibrary.Data.FoodSourceData;

public class TwoDimFoodSourceIntervalData : FoodSourceDataBase
{
    public Interval[,] Z { get; set; }
    public int InitI { get; set; }
    public int InitJ { get; set; }
    public Action<int, int, Interval[,], float[], float[,]> Func { get; internal set; }
    public override bool IsIntervalData => true;

    public TwoDimFoodSourceIntervalData(
        int gLength,
        float deltaZ,
        float deltaV,
        Interval[,] z,
        int initI,
        int initJ,
        bool evaluateCountEnabled = false) : base(gLength, deltaZ, deltaV, evaluateCountEnabled)
    {
        Z = z;
        InitI = initI;
        InitJ = initJ;
    }

    public override float[] GetFlatZ()
    {
        var rows = Z.GetLength(0);
        var cols = Z.GetLength(1);

        var flatZ = new float[rows * cols * 2];

        var index = 0;
        for (var i = 0; i < rows; i++)
        {
            for (var j = 0; j < cols; j++)
            {
                flatZ[index++] = Z[i, j].Left;
                flatZ[index++] = Z[i, j].Right;
            }
        }

        return flatZ;
    }
}

namespace ArtificialBeeColonyLibrary.Data;

public class BeeColonyData
{
    public int S { get; }
    public int Mcn { get; } // -1 - run until solution found
    public int Limit { get; }
    public int CoresNumber { get; }
    public ParallelOption ParallelOption { get; }
    public bool ResetLimit { get; }
    public float DeltaCap { get; }
    public float RelativeDeltaCap { get; }
    public string Func { get; internal set; }
    internal byte[] Ptx { get; set; }

    public BeeColonyData(
```

```
        int s,
        int mcn,
        int limit,
        float deltaCap,
        float relativeDeltaCap,
        int coresNumber,
        ParallelOption parallelOption,
        bool resetLimit,
        string func)
    {
        S = s;
        Mcn = mcn;
        Limit = limit;
        DeltaCap = deltaCap;
        RelativeDeltaCap = relativeDeltaCap;
        if (parallelOption == ParallelOption.Cuda)
        {
            CoresNumber = coresNumber;
        }
        else
        {
            CoresNumber = coresNumber > Environment.ProcessorCount || coresNumber < 2
                ? Environment.ProcessorCount
                : coresNumber;
        }
        ParallelOption = parallelOption;
        ResetLimit = resetLimit;
        Func = func;
    }
}

namespace ArtificialBeeColonyLibrary.Data;

public class BeeColonyResult
{
    public FoodSource FoodSource { get; internal set; }
    public StructuralFoodSource StructuralFoodSource { get; internal set; }
    public int Cycle { get; internal set; }
    public int EvaluationCount { get; internal set; }
    public string ErrorMessage { get; internal set; }
}

using System.Runtime.InteropServices;

namespace ArtificialBeeColonyLibrary.Data;

public interface IKernelFoodSource
{
    public void SetG(int index, float value);
};

[StructLayout(LayoutKind.Sequential)]
unsafe struct KernelFoodSource1 : IKernelFoodSource
{
    public fixed float G[1];

    public void SetG(int index, float value)
    {
        G[index] = value;
    }
};

[StructLayout(LayoutKind.Sequential)]
```

```
unsafe struct KernelFoodSource2 : IKernelFoodSource
{
   public fixed float G[2];

   public void SetG(int index, float value)
   {
      G[index] = value;
   }
};

[StructLayout(LayoutKind.Sequential)]
unsafe struct KernelFoodSource3 : IKernelFoodSource
{
   public fixed float G[3];

   public void SetG(int index, float value)
   {
      G[index] = value;
   }
};

[StructLayout(LayoutKind.Sequential)]
unsafe struct KernelFoodSource4 : IKernelFoodSource
{
   public fixed float G[4];

   public void SetG(int index, float value)
   {
      G[index] = value;
   }
};

[StructLayout(LayoutKind.Sequential)]
unsafe struct KernelFoodSource5 : IKernelFoodSource
{
   public fixed float G[5];

   public void SetG(int index, float value)
   {
      G[index] = value;
   }
};

[StructLayout(LayoutKind.Sequential)]
unsafe struct KernelFoodSource6 : IKernelFoodSource
{
   public fixed float G[6];

   public void SetG(int index, float value)
   {
      G[index] = value;
   }
};

[StructLayout(LayoutKind.Sequential)]
unsafe struct KernelFoodSource7 : IKernelFoodSource
{
   public fixed float G[7];

   public void SetG(int index, float value)
   {
      G[index] = value;
   }
```

```
};

[StructLayout(LayoutKind.Sequential)]
unsafe struct KernelFoodSource8 : IKernelFoodSource
{
    public fixed float G[8];
    public void SetG(int index, float value)
    {
        G[index] = value;
    }
};

[StructLayout(LayoutKind.Sequential)]
unsafe struct KernelFoodSource9 : IKernelFoodSource
{
    public fixed float G[9];

    public void SetG(int index, float value)
    {
        G[index] = value;
    }
};

[StructLayout(LayoutKind.Sequential)]
unsafe struct KernelFoodSource10 : IKernelFoodSource
{
    public fixed float G[10];

    public void SetG(int index, float value)
    {
        G[index] = value;
    }
};

using System.Globalization;
using System.Text;

namespace ArtificialBeeColonyLibrary.Data;

public class FoodSource
{
    public int M { get; set; }
    public float Limit { get; private set; }
    public float Delta { get; private set; }
    public float[] G { get; }
    public Guid Id { get; }

    public FoodSource(int dimension)
    {
        G = new float[dimension];
        Id = Guid.NewGuid();
    }

    public FoodSource(float[] g)
    {
        G = g;
    }

    public void Initialize(float[] lowerBound, float[] upperBound)
    {
        for (var i = 0; i < G.Length; i++)
        {
            G[i] = lowerBound[i] + Random.Shared.NextSingle() * (upperBound[i] - lowerBound[i]);
```

```
    }
}

public void Initialize(float[] g, float[] neighborG, float[] lowerBound, float[] upperBound)
{
    var dim = Random.Shared.Next(0, G.Length);

    // Amount of impact is between -1 to 1
    var phi = -1 + Random.Shared.NextSingle() * 2;

    var newParam1 = g[dim] + phi * (g[dim] - neighborG[dim]);
    var newParam2 = g[dim] - phi * (g[dim] - neighborG[dim]);
    var newParam = newParam1 > upperBound[dim] || newParam1 < lowerBound[dim] ? newParam2 : newParam1;

    for (var i = 0; i < G.Length; i++)
    {
        if (i == dim)
        {
            G[i] = newParam;
        }
        else
        {
            G[i] = g[i];
        }
    }
}

public void SetNectar(float nectar)
{
    Delta = nectar;
}

public void IncrementLimit()
{
    Limit++;
}

public void ResetLimit()
{
    Limit = 0;
}

public string GetString()
{
    var sb = new StringBuilder();
    var i = 0;
    foreach (var p in G)
    {
        sb.AppendFormat("g{0}={1} ", i++, p.ToString(CultureInfo.InvariantCulture));
    }

    sb.AppendFormat("| delta={0}", Delta.ToString(CultureInfo.InvariantCulture));
    return sb.ToString();
}


private FoodSource(FoodSource other)
{
    M = other.M;
    Limit = other.Limit;
    Delta = other.Delta;
    Id = other.Id;
```

```
        // Copy the array
        G = new float[other.G.Length];
        Array.Copy(other.G, G, other.G.Length);
    }

    public FoodSource DeepClone()
    {
        return new FoodSource(this);
    }
}

namespace ArtificialBeeColonyLibrary.Data;

public class FoodSourceGeneratorData
{
    public int FoodSourceLength { get; internal set; }
    public int FoodSourcesCount { get; internal set; }
    public float[] FoodSourceUpperLimit { get; }
    public float[] FoodSourceLowerLimit { get; }

    public FoodSourceGeneratorData(
        float[] foodSourceLowerLimit,
        float[] foodSourceUpperLimit)
    {
        FoodSourceLowerLimit = foodSourceLowerLimit;
        FoodSourceUpperLimit = foodSourceUpperLimit;
    }
}

using ArtificialBeeColonyLibrary.Algorithm.Helpers;

namespace ArtificialBeeColonyLibrary.Data;

public struct Interval
{
    public float Left { get; }
    public float Right { get; }

    public Interval(float left, float right)
    {
        if (float.IsNegativeInfinity(left)) {
            left = -999999999;
        }

        if (float.IsPositiveInfinity(left)) {
            left = 999999999;
        }

        if (float.IsNegativeInfinity(right)) {
            right = -999999999;
        }

        if (float.IsPositiveInfinity(right)) {
            right = 999999999;
        }

        Left = left;
        Right = right;
    }

    public static Interval operator +(Interval a, Interval b)
    {
        var numbers = new[]
```

```
    {
        a.Left + b.Left,
        a.Right + b.Right
    };
    return new Interval(MathHelper.Min(numbers), MathHelper.Max(numbers));
}

public static Interval operator +(float a, Interval b)
{
    return b + a;
}

public static Interval operator +(Interval a, float b)
{
    return new Interval(a.Left + b, a.Right + b);
}

public static Interval operator -(Interval a, Interval b)
{
    var numbers = new[]
    {
        a.Left - b.Right,
        a.Right - b.Left
    };
    return new Interval(MathHelper.Min(numbers), MathHelper.Max(numbers));
}

public static Interval operator -(float a, Interval b)
{
    return new Interval(a - b.Left, a - b.Right);
}

public static Interval operator -(Interval a, float b)
{
    return new Interval(a.Left - b, a.Right - b);
}

public static Interval operator *(Interval a, Interval b)
{
    var numbers = new[]
    {
        a.Left * b.Left,
        a.Left * b.Right,
        a.Right * b.Left,
        a.Right * b.Right,
    };
    return new Interval(MathHelper.Min(numbers), MathHelper.Max(numbers));
}

public static Interval operator *(float a, Interval b)
{
    return b * a;
}

public static Interval operator *(Interval a, float b)
{
    return new Interval(a.Left * b, a.Right * b);
}

public static Interval operator /(Interval a, Interval b)
{
    var numbers = new[]
    {
```

```
            a.Left / b.Left,
            a.Left / b.Right,
            a.Right / b.Left,
            a.Right / b.Right,
        };
        if (numbers.Any(n => float.IsNaN(n)))
        {
            // TODO
        }
        return new Interval(MathHelper.Min(numbers), MathHelper.Max(numbers));
    }

    public static Interval operator /(float a, Interval b)
    {
        return new Interval(a / b.Left, a / b.Right);
    }

    public static Interval operator /(Interval a, float b)
    {
        return new Interval(a.Left / b, a.Right / b);
    }

    public float Width()
    {
        return Math.Abs(Left - Right);
    }

    public float Middle()
    {
        return (Left + Right) / 2;
    }

    public static Interval Intersect(Interval a, Interval b)
    {
        if (!IsIntersect(a, b))
        {
            throw new ArgumentException("Intervals are not overlapping.");
        }

        return new Interval(Math.Max(a.Left, b.Left), Math.Min(a.Right, b.Right));
    }

    public static bool IsIntersect(Interval a, Interval b)
    {
        return a.Left < b.Right && a.Right > b.Left;
    }
}

namespace ArtificialBeeColonyLibrary.Data;

public enum ParallelOption
{
    No,
    Cpu,
    Cuda
}

namespace ArtificialBeeColonyLibrary.Data
{
    public class PlotData
    {
        public int Length {  get; set; }
        public Interval[] Initial { get; set; }
```

```
        public Interval[] Calculated { get; set; }
    }
}

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

namespace ArtificialBeeColonyLibrary.Data
{
    public class StructuralBeeColonyData
    {
        public int S { get; }
        public int Mcn { get; } // -1 - run until solution found
        public int Limit { get; }
        public float DeltaCap { get; }
        public float RelativeDeltaCap { get; }
        public int MinI { get; }
        public int MaxI { get; }
        public List<string> F {  get; }
        public int Dim { get; }

        public StructuralBeeColonyData(int s, int mcn, int limit, float deltaCap, float relativeDeltaCap, int minI, int maxI, int
dim, List<string> f)
        {
            S = s;
            Mcn = mcn;
            Limit = limit;
            DeltaCap = deltaCap;
            RelativeDeltaCap = relativeDeltaCap;
            MinI = minI;
            MaxI = maxI;
            Dim = dim;
            F = f;
        }
    }
}

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

namespace ArtificialBeeColonyLibrary.Data
{
    public class StructuralFoodSource
    {
        public float Limit { get; private set; }
        public FoodSource FoodSource { get; private set; }
        public Guid Id { get; }

        public List<int> Indexes { get; set; }
        public string Function { get; set; }
        public int M { get; internal set; }

        public StructuralFoodSource()
        {
            Id = Guid.NewGuid();
        }
```

```
        public void SetNectar(FoodSource foodSource)
        {
            FoodSource = foodSource;
        }

        public void IncrementLimit()
        {
            Limit++;
        }

        public string GetString()
        {
            var sb = new StringBuilder();
            sb.AppendLine(Function);
            sb.AppendLine(FoodSource?.GetString());
            return sb.ToString();
        }

        private StructuralFoodSource(StructuralFoodSource other)
        {
            M = other.M;
            Limit = other.Limit;
            Id = other.Id;
            Function = other.Function;
            Indexes = new List<int>(other.Indexes);
            FoodSource = other.FoodSource.DeepClone();
        }

        internal StructuralFoodSource DeepClone()
        {
            return new StructuralFoodSource(this);
        }
    }
}

using System;
using System.Globalization;
using System.Text.RegularExpressions;
using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;
using ManagedCuda;
using ManagedCuda.NVRTC;

namespace ArtificialBeeColonyLibrary.Utils
{
    public static class CudaModelBuilder
    {
        private const string gLengthPH = "gLength";
        private const string kPH = "K";
        private const string iPH = "I";
        private const string jPH = "J";
        private const string initKPH = "initK";
        private const string initIPH = "initI";
        private const string initJPH = "initJ";
        private const string deltaZPH = "deltaZ";
        private const string deltaVPH = "deltaV";
        private const string funcPH = "func";
        private const string uLenght0 = "Ul0";
        private const string uLenght1 = "Ul1";

        public static bool IsCudaAvailable()
        {
            try
```

```
    {
        var deviceCount = CudaContext.GetDeviceCount();
        return deviceCount > 0;
    }
    catch { return false; }
}

public static byte[] CreateOneDimPtx(
    string projName,
    int gLength,
    int k,
    int ul0,
    int ul1,
    int initK,
    float deltaZ,
    float deltaV,
    string func)
{
    string filename = "abc_one_dim_kernel.cu";
    string fileToCompileTemplate = File.ReadAllText(filename);

    var placeholderValues = new Dictionary<string, string>()
    {
        [gLengthPH] = gLength.ToString(),
        [kPH] = k.ToString(),
        [initKPH] = initK.ToString(),
        [deltaZPH] = ConvertToFloatString(deltaZ),
        [deltaVPH] = ConvertToFloatString(deltaV),
        [funcPH] = func,
        [uLenght0] = ul0.ToString(),
        [uLenght1] = ul1.ToString(),
    };
    var fileToCompile = fileToCompileTemplate.ReplacePlaceholders(placeholderValues);

    return Compile(fileToCompile, projName);
}

private static string ConvertToFloatString(float value)
{
    if (value == 0)
    {
        return $"0.0f";
    }

    return $"{value.ToString(CultureInfo.InvariantCulture)}f";
}

public static byte[] CreateTwoDimPtx(
    string projName,
    int gLength,
    int i,
    int j,
    int ul0,
    int ul1,
    int initI,
    int initJ,
    float deltaZ,
    float deltaV,
    string func)
{
    string filename = "abc_two_dim_kernel.cu";
    string fileToCompileTemplate = File.ReadAllText(filename);
```

```
    var placeholderValues = new Dictionary<string, string>()
    {
        [gLengthPH] = gLength.ToString(),
        [iPH] = i.ToString(),
        [jPH] = j.ToString(),
        [initIPH] = initI.ToString(),
        [initJPH] = initJ.ToString(),
        [deltaZPH] = ConvertToFloatString(deltaZ),
        [deltaVPH] = ConvertToFloatString(deltaV),
        [funcPH] = func.ModifyToCPP(),
        [uLenght0] = ul0.ToString(),
        [uLenght1] = ul1.ToString(),
    };

    var fileToCompile = fileToCompileTemplate.ReplacePlaceholders(placeholderValues);

    return Compile(fileToCompile, projName);
}

public static byte[] CreateThreeDimPtx(
 string projName,
 int gLength,
 int k,
 int i,
 int j,
 int ul0,
 int ul1,
 int initK,
 int initI,
 int initJ,
 float deltaZ,
 float deltaV,
 string func)
{
    string filename = "abc_three_dim_kernel.cu";
    string fileToCompileTemplate = File.ReadAllText(filename);

    var placeholderValues = new Dictionary<string, string>()
    {
        [gLengthPH] = gLength.ToString(),
        [kPH] = k.ToString(),
        [iPH] = i.ToString(),
        [jPH] = j.ToString(),
        [initKPH] = initK.ToString(),
        [initIPH] = initI.ToString(),
        [initJPH] = initJ.ToString(),
        [deltaZPH] = ConvertToFloatString(deltaZ),
        [deltaVPH] = ConvertToFloatString(deltaV),
        [funcPH] = func.ModifyToCPP(),
        [uLenght0] = ul0.ToString(),
        [uLenght1] = ul1.ToString(),
    };

    var fileToCompile = fileToCompileTemplate.ReplacePlaceholders(placeholderValues);

    return Compile(fileToCompile, projName);
}

private static byte[] Compile(string fileToCompile, string outputName)
{
    var rtc = new CudaRuntimeCompiler(fileToCompile, outputName);
    try
    {
```

```
            rtc.Compile(new string[] { }); //"-use_fast_math"
        }
        catch (Exception e)
        {
            var a = rtc.GetLogAsString();
            return null;
        }

        byte[] ptx = rtc.GetPTX();

        rtc.Dispose();
        return ptx;
    }

    private static string ReplacePlaceholders(this string input, Dictionary<string, string> placeholderValues)
    {
        string pattern = @"\${(\w+)\}";

        string result = Regex.Replace(input, pattern, match =>
        {
            string placeholderName = match.Groups[1].Value;

            if (placeholderValues.TryGetValue(placeholderName, out var replacement))
            {
                return replacement;
            }
            else
            {
                return match.Value;
            }
        });

        return result;
    }

    private static string ModifyToCPP(this string input)
    {
        return Regex.Replace(input, @",", @"][");
    }

    internal static async Task<bool> PopulateWithPtx(BeeColonyData data, FoodSourceDataBase foodSourceData)
    {
        byte[] ptx = null;
        if (foodSourceData is OneDimFoodSourceData oneDimFoodSourceData)
        {
            ptx = await Task.Run(() => CreateOneDimPtx(
                        null,
                        oneDimFoodSourceData.GLength,
                        oneDimFoodSourceData.Z.Length,
                        oneDimFoodSourceData.U?.GetLength(0) ?? 1,
                        oneDimFoodSourceData.U?.GetLength(1) ?? 1,
                        oneDimFoodSourceData.InitK,
                        oneDimFoodSourceData.DeltaZ,
                        oneDimFoodSourceData.DeltaV,
                        data.Func));
        }

        if (foodSourceData is OneDimFoodSourceIntervalData oneDimFoodSourceIntervalData)
        {
            ptx = await Task.Run(() => CreateOneDimPtx(
                        null,
                        oneDimFoodSourceIntervalData.GLength,
                        oneDimFoodSourceIntervalData.Z.Length,
```

```
            oneDimFoodSourceIntervalData.U?.GetLength(0) ?? 1,
            oneDimFoodSourceIntervalData.U?.GetLength(1) ?? 1,
            oneDimFoodSourceIntervalData.InitK,
            oneDimFoodSourceIntervalData.DeltaZ,
            oneDimFoodSourceIntervalData.DeltaV,
            data.Func));
}

if (foodSourceData is TwoDimFoodSourceData twoDimFoodSourceData)
{
   ptx = await Task.Run(() => CudaModelBuilder.CreateTwoDimPtx(
            null,
            twoDimFoodSourceData.GLength,
            twoDimFoodSourceData.Z.GetLength(0),
            twoDimFoodSourceData.Z.GetLength(1),
            twoDimFoodSourceData.U?.GetLength(0) ?? 1,
            twoDimFoodSourceData.U?.GetLength(1) ?? 1,
            twoDimFoodSourceData.InitI,
            twoDimFoodSourceData.InitJ,
            twoDimFoodSourceData.DeltaZ,
            twoDimFoodSourceData.DeltaV,
            data.Func));
}

if (foodSourceData is TwoDimFoodSourceIntervalData twoDimFoodSourceIntervalData)
{
   ptx = await Task.Run(() => CudaModelBuilder.CreateTwoDimPtx(
            null,
            twoDimFoodSourceIntervalData.GLength,
            twoDimFoodSourceIntervalData.Z.GetLength(0),
            twoDimFoodSourceIntervalData.Z.GetLength(1),
            twoDimFoodSourceIntervalData.U?.GetLength(0) ?? 1,
            twoDimFoodSourceIntervalData.U?.GetLength(1) ?? 1,
            twoDimFoodSourceIntervalData.InitI,
            twoDimFoodSourceIntervalData.InitJ,
            twoDimFoodSourceIntervalData.DeltaZ,
            twoDimFoodSourceIntervalData.DeltaV,
            data.Func));
}

if (foodSourceData is ThreeDimFoodSourceData threeDimFoodSourceData)
{
   ptx = await Task.Run(() => CudaModelBuilder.CreateThreeDimPtx(
            null,
            threeDimFoodSourceData.GLength,
            threeDimFoodSourceData.Z.Length,
            threeDimFoodSourceData.Z[0].GetLength(0),
            threeDimFoodSourceData.Z[0].GetLength(1),
            threeDimFoodSourceData.U?.GetLength(0) ?? 1,
            threeDimFoodSourceData.U?.GetLength(1) ?? 1,
            threeDimFoodSourceData.InitK,
            threeDimFoodSourceData.InitI,
            threeDimFoodSourceData.InitJ,
            threeDimFoodSourceData.DeltaZ,
            threeDimFoodSourceData.DeltaV,
            data.Func));
}

if (foodSourceData is ThreeDimFoodSourceIntervalData threeDimFoodSourceIntervalData)
{
   ptx = await Task.Run(() => CudaModelBuilder.CreateThreeDimPtx(
           null,
           threeDimFoodSourceIntervalData.GLength,
```

```
                    threeDimFoodSourceIntervalData.Z.Length,
                    threeDimFoodSourceIntervalData.Z[0].GetLength(0),
                    threeDimFoodSourceIntervalData.Z[0].GetLength(1),
                    threeDimFoodSourceIntervalData.U?.GetLength(0) ?? 1,
                    threeDimFoodSourceIntervalData.U?.GetLength(1) ?? 1,
                    threeDimFoodSourceIntervalData.InitK,
                    threeDimFoodSourceIntervalData.InitI,
                    threeDimFoodSourceIntervalData.InitJ,
                    threeDimFoodSourceIntervalData.DeltaZ,
                    threeDimFoodSourceIntervalData.DeltaV,
                    data.Func));
            }

            data.Ptx = ptx;
            return ptx != null;
        }
    }
}

using System.Xml.Linq;

namespace ArtificialBeeColonyLibrary.Utils
{
    public static class ElementsGenerator
    {
        private class Element
        {
            public string Value { get; set; }
            public SortedSet<int> Order { get; set; } = new SortedSet<int>();
        }


        private const string OneDimElementFormat = "V[k-{0}]";
        private const string TwoDimElementFormat = "V[i-{0},j-{1}]";
        private const string ThreeDimElementFormat = "V[k-{0}][i-{1},j-{2}]";
        private const string MultiplicationFormat = "{0}*{1}";
        private const string DivisionFormat = "{0}/({1})";

        private static void GenerateMultiplication(List<Element> result, List<Element> initialElements, int power)
        {
            for (var i = 0; i < power - 1; i++)
            {
                var multElements = new List<Element>();
                foreach (var element in result)
                {
                    foreach (var initElement in initialElements)
                    {
                        if (initElement.Order.Max >= element.Order.Max)
                        {
                            var me = new Element() { Value = string.Format(MultiplicationFormat, element.Value,
initElement.Value), Order = new SortedSet<int>(element.Order) };
                            me.Order.Add(initElement.Order.Max);
                            multElements.Add(me);
                        }
                    }
                }
                result.AddRange(multElements);
            }
        }

        private static void GenerateDivision(List<Element> result)
        {
            var oneDivElements = new List<Element>();
```

```
            foreach (var initElement in result)
            {
                oneDivElements.Add(new Element() { Value = string.Format(DivisionFormat, 1, initElement.Value) });
            }


            var divElements = new List<Element>();
            foreach (var element in result)
            {
                foreach (var element2 in result)
                {
                    var notContains = true;
                    foreach (var eOrder in element2.Order)
                    {
                        notContains = notContains && !element.Order.Contains(eOrder);
                    }

                    if (notContains)
                    {
                        var me = new Element() { Value = string.Format(DivisionFormat, element.Value, element2.Value) };
                        divElements.Add(me);
                    }
                }
            }
            result.AddRange(oneDivElements);
            result.AddRange(divElements);
        }

        public static List<string> GenerateOneDimElements(int orderK, int power, bool multiplication = false, bool division =
false)
        {
            var result = new List<Element>();

            var initialElements = new List<Element>();
            for (var i = 1; i <= orderK; i++)
            {
                var e = new Element() { Value = string.Format(OneDimElementFormat, i) };
                e.Order.Add(i);
                initialElements.Add(e);
            }

            result.AddRange(initialElements);

            if (multiplication)
            {
                GenerateMultiplication(result, initialElements, power);
            }

            if (division)
            {
                GenerateDivision(result);
            }

            return result.Select(e => e.Value).ToList();
        }

        public static List<string> GenerateTwoDimElements(int orderI, int orderJ, int power, bool multiplication = true, bool
division = false)
        {
            var jconst = 100;
            var result = new List<Element>();

            var initialElements = new List<Element>();
```

```
            for (var i = 0; i <= orderI; i++)
            {
                for (var j = 0; j <= orderJ; j++)
                {
                    if (i == 0 && j == 0) continue;
                    var e = new Element() { Value = string.Format(TwoDimElementFormat, i, j) };
                    e.Order.Add(i * jconst + j);
                    initialElements.Add(e);
                }
            }
            result.AddRange(initialElements);

            if (multiplication)
            {
                GenerateMultiplication(result, initialElements, power);
            }

            if (division)
            {
                GenerateDivision(result);
            }

            return result.Select(e => e.Value).ToList();
        }

        public static List<string> GenerateThreeDimElements(int orderK, int orderI, int orderJ, int power, bool multiplication
= true, bool division = false)
        {
            var jconst = 100;
            var kconst = 1000;
            var result = new List<Element>();

            var initialElements = new List<Element>();
            for (var k = 0; k <= orderK; k++)
            {
                for (var i = 0; i <= orderI; i++)
                {
                    for (var j = 0; j <= orderJ; j++)
                    {
                        if (i == 0 && j == 0 && k == 0) continue;
                        var e = new Element() { Value = string.Format(ThreeDimElementFormat, k, i, j) };
                        e.Order.Add(k * kconst + i * jconst + j);
                        initialElements.Add(e);
                    }
                }
            }
            result.AddRange(initialElements);

            if (multiplication)
            {
                GenerateMultiplication(result, initialElements, power);
            }

            if (division)
            {
                GenerateDivision(result);
            }

            return result.Select(e => e.Value).ToList();
        }
    }
}
```

```
using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;
using Microsoft.CodeAnalysis.CSharp.Scripting;
using Microsoft.CodeAnalysis.Scripting;
using System.Reflection;

namespace ArtificialBeeColonyLibrary.Utils;

public static class ModelBuilder
{
   private static ScriptOptions scriptOptions = ScriptOptions.Default
      .WithReferences(
         Assembly.GetExecutingAssembly().Location
      )
      .WithImports("System");

   public static async Task<Action<int, Interval[], float[], float[,]>> CreateOneDimAction(string func)
   {
      string fullCode = $"({typeof(int)} k, {typeof(Interval)}[] V, {typeof(float)}[] G, {typeof(float)}[,] U) => {{ {func}; }}";
      return await Compile<Action<int, Interval[], float[], float[,]>> (fullCode);
   }

   public static async Task<Action<int, int, Interval[,], float[], float[,]>> CreateTwoDimAction(string func)
   {
      string fullCode = $"({typeof(int)} i, {typeof(int)} j, {typeof(Interval)}[,] V, {typeof(float)}[] G, {typeof(float)}[,] U) => {{ {func}; }}";
      return await Compile<Action<int, int, Interval[,], float[], float[,]>>(fullCode);
   }

   public static async Task<Action<int, int, int, Interval[][,], float[], float[,]>> CreateThreeDimAction(string func)
   {
      string fullCode = $"({typeof(int)} k, {typeof(int)} i, {typeof(int)} j, {typeof(Interval)}[][,] V, {typeof(float)}[] G, {typeof(float)}[,] U) => {{ {func}; }}";
      return await Compile<Action<int, int, int, Interval[][,], float[], float[,]>>(fullCode);
   }

   private static async Task<T> Compile<T>(string code)
   {
      var script = CSharpScript.Create<T>(code, scriptOptions);
      try
      {
         var compilation = await script.RunAsync();
         return compilation.ReturnValue;
      }
      catch
      {
         return default(T);
      }
   }

   internal static async Task<bool> PopulateWithFunc(FoodSourceDataBase foodSourceData, string func)
   {
      if (foodSourceData is OneDimFoodSourceData oneDimFoodSourceData)
      {
         var action = await CreateOneDimAction(func);
         if (action == null) return false;
         oneDimFoodSourceData.Func = action;
         return true;
      }

      if (foodSourceData is OneDimFoodSourceIntervalData oneDimFoodSourceIntervalData)
      {
```

```
            var action = await CreateOneDimAction(func);
            if (action == null) return false;
            oneDimFoodSourceIntervalData.Func = action;
            return true;
        }

        if (foodSourceData is TwoDimFoodSourceData twoDimFoodSourceData)
        {
            var action = await CreateTwoDimAction(func);
            if (action == null) return false;
            twoDimFoodSourceData.Func = action;
            return true;
        }

        if (foodSourceData is TwoDimFoodSourceIntervalData twoDimFoodSourceIntervalData)
        {
            var action = await CreateTwoDimAction(func);
            if (action == null) return false;
            twoDimFoodSourceIntervalData.Func = action;
            return true;
        }

        if (foodSourceData is ThreeDimFoodSourceData threeDimFoodSourceData)
        {
            var action = await CreateThreeDimAction(func);
            if (action == null) return false;
            threeDimFoodSourceData.Func = action;
            return true;
        }

        if (foodSourceData is ThreeDimFoodSourceIntervalData threeDimFoodSourceIntervalData)
        {
            var action = await CreateThreeDimAction(func);
            if (action == null) return false;
            threeDimFoodSourceIntervalData.Func = action;
            return true;
        }

        return false;
    }
}
```

ArtificialBeeColonyApp.cu

```
using ArtificialBeeColonyLibrary.Algorithm;
using ArtificialBeeColonyLibrary.Data;
using System.Diagnostics;
using System.Text;

namespace ArtificialBeeColonyApp.ABC
{
    internal class ABCRunner
    {
        private TextBox _resultTextBox;
        private Stopwatch sp;

        public FoodSource Result { get; private set; }
        public string FuncResult { get; private set; }

        public ABCRunner(TextBox resultTextBox)
        {
            _resultTextBox = resultTextBox;
            sp = new Stopwatch();
```

```
    }

    public void SetResult(float[] g)
    {
        Result = new FoodSource(g);
    }

    public async Task Run(ABCPreparer abcPreparer, CancellationToken cancellationToken)
    {
        if (abcPreparer.IsStructural())
        {
            await RunStructural(abcPreparer, cancellationToken);
        }
        else
        {
            await RunParametric(abcPreparer, cancellationToken);
        }
    }

    private async Task RunStructural(ABCPreparer abcPreparer, CancellationToken cancellationToken)
    {
        var parametricBeeColonyData = abcPreparer.GetBeeColonyData();
        var structuralBeeColonyData = abcPreparer.GetStructuralBeeColonyData();
        var foodSourceData = abcPreparer.GetFoodSourceData();
        var foodSourceGeneratorData = abcPreparer.GetGeneratorData();

        var sb = new StringBuilder();
        using (var beeColony = new StructuralBeeColony(structuralBeeColonyData, parametricBeeColonyData,
foodSourceData, foodSourceGeneratorData))
        {
            var result = await beeColony.Run(cancellationToken);
            if (result.StructuralFoodSource != null)
            {
                sb.AppendLine(result.StructuralFoodSource.GetString());
                Result = result.StructuralFoodSource.FoodSource;
                FuncResult = result.StructuralFoodSource.Function;
            }
            sb.AppendLine($"Cycle = {result.Cycle}");
        }
        _resultTextBox.Text = sb.ToString();
    }

    private async Task RunParametric(ABCPreparer abcPreparer, CancellationToken cancellationToken)
    {
        var totalTime = 0.0;
        var totalCycles = 0.0;
        var totalEvalCount = 0.0;

        var beeColonyData = abcPreparer.GetBeeColonyData();
        var foodSourceData = abcPreparer.GetFoodSourceData();
        var foodSourceGeneratorData = abcPreparer.GetGeneratorData();

        var sb = new StringBuilder();
        for (var i = 0; i < abcPreparer.NumberOfRuns; i++)
        {
            sp.Restart();

            using (var beeColony = new BeeColony(beeColonyData, foodSourceData, foodSourceGeneratorData))
            {
                if(!await beeColony.InitializeAsync())
                {
                    sb.AppendLine($"Error: initialization failed.");
                    break;
```

```
                }
                var result = await beeColony.Run(cancellationToken);

                sp.Stop();
                totalTime += sp.ElapsedMilliseconds;
                totalCycles += result.Cycle;
                totalEvalCount += result.EvaluationCount;
                Result = result.FoodSource;

                if (!string.IsNullOrEmpty(result.ErrorMessage))
                {
                    sb.AppendLine($"Error: {result.ErrorMessage}");
                    break;
                }
            }
        }

        sb.AppendLine($"Avg Cycle = {totalCycles / abcPreparer.NumberOfRuns}");
        sb.AppendLine($"Avg Eval Count = {totalEvalCount / abcPreparer.NumberOfRuns}");
        sb.AppendLine($"Avg Time = {totalTime / abcPreparer.NumberOfRuns}");
        sb.AppendLine(Result?.GetString());

        _resultTextBox.Text = sb.ToString();
    }
  }
}

using ArtificialBeeColonyApp.Data;
using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Data.FoodSourceData;
using ArtificialBeeColonyLibrary.Utils;

namespace ArtificialBeeColonyApp.ABC
{
  internal class ABCPreparer
  {

    private AbcProjectData _abcProjectData;

    public int NumberOfRuns => _abcProjectData.NumberOfRuns;

    public void SetProjectData(AbcProjectData abcProjectData)
    {
        _abcProjectData = abcProjectData;
    }

    public bool IsStructural()
    {
        return _abcProjectData?.StructuralData?.IsEnabled ?? false;
    }

    public BeeColonyData GetBeeColonyData()
    {
        var parallelOption = _abcProjectData.ParallelOption == ParallelOption.Cuda
            ? ParallelOption.Cuda
            : (_abcProjectData.ThreadsNumber == 1 ? ParallelOption.No : ParallelOption.Cpu);
        return new BeeColonyData(
            _abcProjectData.S,
            _abcProjectData.Mcn,
            _abcProjectData.Limit,
            _abcProjectData.DeltaCap,
            _abcProjectData.RelativeDeltaCap,
            _abcProjectData.ThreadsNumber,
```

```
            parallelOption,
            _abcProjectData.ResetLimit,
            _abcProjectData.Func);
        }

        public FoodSourceDataBase GetFoodSourceData(bool test = false)
        {
            FoodSourceDataBase result;
            switch (_abcProjectData.DataGridModel.Dim)
            {
                case 1:
                    {
                        result = _abcProjectData.DataGridModel.IsInterval
                            ? new OneDimFoodSourceIntervalData(
                            _abcProjectData.GLength,
                            _abcProjectData.DeltaZ,
                            _abcProjectData.DeltaV,
                            test ? _abcProjectData.TestDataGridModel.GetOneDimIntervalData() :
_abcProjectData.DataGridModel.GetOneDimIntervalData(),
                            _abcProjectData.InitK,
                            _abcProjectData.EvaluationCount)

                            : new OneDimFoodSourceData(
                            _abcProjectData.GLength,
                            _abcProjectData.DeltaZ,
                            _abcProjectData.DeltaV,
                            test ? _abcProjectData.TestDataGridModel.OneDimData :
_abcProjectData.DataGridModel.OneDimData,
                            _abcProjectData.InitK,
                            _abcProjectData.EvaluationCount);
                        break;
                    }
                case 2:
                    {
                        result = _abcProjectData.DataGridModel.IsInterval
                            ? new TwoDimFoodSourceIntervalData(
                            _abcProjectData.GLength,
                            _abcProjectData.DeltaZ,
                            _abcProjectData.DeltaV,
                            test ? _abcProjectData.TestDataGridModel.GetTwoDimIntervalData() :
_abcProjectData.DataGridModel.GetTwoDimIntervalData(),
                            _abcProjectData.InitI,
                            _abcProjectData.InitJ,
                            _abcProjectData.EvaluationCount)

                            : new TwoDimFoodSourceData(
                            _abcProjectData.GLength,
                            _abcProjectData.DeltaZ,
                            _abcProjectData.DeltaV,
                            test ? _abcProjectData.TestDataGridModel.TwoDimData :
_abcProjectData.DataGridModel.TwoDimData,
                            _abcProjectData.InitI,
                            _abcProjectData.InitJ,
                            _abcProjectData.EvaluationCount);
                        break;
                    }
                case 3:
                    {
                        result = _abcProjectData.DataGridModel.IsInterval
                            ? new ThreeDimFoodSourceIntervalData(
                            _abcProjectData.GLength,
                            _abcProjectData.DeltaZ,
                            _abcProjectData.DeltaV,
```

```
                    test ? _abcProjectData.TestDataGridModel.GetThreeDimIntervalData() :
_abcProjectData.DataGridModel.GetThreeDimIntervalData(),
                    _abcProjectData.InitK,
                    _abcProjectData.InitI,
                    _abcProjectData.InitJ,
                    _abcProjectData.EvaluationCount)

                    : new ThreeDimFoodSourceData(
                    _abcProjectData.GLength,
                    _abcProjectData.DeltaZ,
                    _abcProjectData.DeltaV,
                    test ? _abcProjectData.TestDataGridModel.ThreeDimData :
_abcProjectData.DataGridModel.ThreeDimData,
                    _abcProjectData.InitK,
                    _abcProjectData.InitI,
                    _abcProjectData.InitJ,
                    _abcProjectData.EvaluationCount);
                break;
            }
        default: throw new NotImplementedException();
      }

      result.SetControlParams(test ? _abcProjectData.TestControlDataGridModel?.GetControlParams() :
_abcProjectData.ControlDataGridModel?.GetControlParams());
      return result;
    }

    public FoodSourceGeneratorData GetGeneratorData()
    {
      return new FoodSourceGeneratorData(_abcProjectData.LowerBounds, _abcProjectData.UpperBounds);
    }

    public StructuralBeeColonyData GetStructuralBeeColonyData()
    {
      return new StructuralBeeColonyData(
        _abcProjectData.StructuralData.S,
        _abcProjectData.StructuralData.Mcn,
        _abcProjectData.StructuralData.Limit,
        _abcProjectData.StructuralData.DeltaCap,
        _abcProjectData.StructuralData.RelativeDeltaCap,
        _abcProjectData.StructuralData.MinI,
        _abcProjectData.StructuralData.MaxI,
        _abcProjectData.DataGridModel.Dim,
        _abcProjectData.StructuralData.Elements);
    }
  }
}

using ArtificialBeeColonyApp.Models;
using CsvHelper.Configuration;
using CsvHelper;
using System.Data;
using System.Globalization;

namespace ArtificialBeeColonyApp.Controllers
{
  internal class DataGridController
  {
    private DataGridView _dataGridView;
    public DataGridModel DataGridModel { get; private set; }

    public DataGridController(DataGridView dataGridView)
    {
```

```
        _dataGridView = dataGridView;
    }

    public void Populate(DataGridModel dataGridModel)
    {
        if (dataGridModel == null) return;

        DataGridModel = dataGridModel;
        switch (dataGridModel.Dim)
        {
            case 1:
                {
                    SetOneDimDataTable(dataGridModel);
                    break;
                }
            case 2:
                {
                    SetTwoDimDataTable(dataGridModel);
                    break;
                }
            case 3:
                {
                    SetThreeDimDataTable(dataGridModel);
                    break;
                }
        }
        Restrict();
    }

    private void Restrict()
    {
        foreach (DataGridViewColumn column in _dataGridView.Columns)
        {
            column.SortMode = DataGridViewColumnSortMode.NotSortable;
        }
    }

    private void SetOneDimDataTable(DataGridModel data)
    {
        if (data == null) return;
        if (data.IsInterval && (data.OneDimData == null || data.OneDimDataU == null)) return;
        if (!data.IsInterval && data.OneDimData == null) return;

        var dataTable = new DataTable();
        dataTable.Columns.Add(new DataColumn("k"));
        for (var i = 0; i < data.OneDimData.Length; i++)
        {

            var row = dataTable.NewRow();
            row[0] = data.IsInterval ?
$"[{data.OneDimData[i].ToString(CultureInfo.InvariantCulture)};{data.OneDimDataU[i].ToString(CultureInfo.InvariantC
ulture)}]" : data.OneDimData[i].ToString(CultureInfo.InvariantCulture);
            dataTable.Rows.Add(row);
        }

        _dataGridView.DataSource = dataTable;

        for (var i = 0; i < _dataGridView.Rows.Count; i++)
        {
            _dataGridView.Rows[i].HeaderCell.Value = $"k{i}";
        }
    }
```

```
        private void SetTwoDimDataTable(DataGridModel data)
        {
            if (data == null) return;
            if (data.IsInterval && (data.TwoDimData == null || data.TwoDimDataU == null)) return;
            if (!data.IsInterval && data.TwoDimData == null) return;

            var dataTable = new DataTable();
            var rowsCount = data.TwoDimData.GetLength(0);
            var columnsCount = data.TwoDimData.GetLength(1);
            for (var i = 0; i < columnsCount; i++)
            {
                dataTable.Columns.Add(new DataColumn($"j{i}"));
            }

            for (var i = 0; i < rowsCount; i++)
            {
                var row = dataTable.NewRow();
                for (var j = 0; j < columnsCount; j++)
                {
                    row[j] = data.IsInterval ? $"[{data.TwoDimData[i,
j].ToString(CultureInfo.InvariantCulture)};{data.TwoDimDataU[i, j].ToString(CultureInfo.InvariantCulture)}]" :
data.TwoDimData[i, j].ToString(CultureInfo.InvariantCulture);
                }
                dataTable.Rows.Add(row);
            }

            _dataGridView.DataSource = dataTable;

            for (var i = 0; i < _dataGridView.Rows.Count; i++)
            {
                _dataGridView.Rows[i].HeaderCell.Value = $"i{i}";
            }
        }

        private void SetThreeDimDataTable(DataGridModel data)
        {
            if (data == null) return;
            if (data.IsInterval && (data.ThreeDimData == null || data.ThreeDimDataU == null)) return;
            if (!data.IsInterval && data.ThreeDimData == null) return;

            var dataTable = new DataTable();
            var kCount = data.ThreeDimData.Length;
            var rowsCount = data.ThreeDimData[0].GetLength(0);
            var columnsCount = data.ThreeDimData[0].GetLength(1);
            for (var i = 0; i < columnsCount; i++)
            {
                dataTable.Columns.Add(new DataColumn($"j{i}"));
            }

            for (var k = 0; k < kCount; k++)
            {
                dataTable.Rows.Add(dataTable.NewRow());
                for (var i = 0; i < rowsCount; i++)
                {
                    var row = dataTable.NewRow();
                    for (var j = 0; j < columnsCount; j++)
                    {
                        row[j] = data.IsInterval ? $"[{data.ThreeDimData[k][i,
j].ToString(CultureInfo.InvariantCulture)};{data.ThreeDimDataU[k][i, j].ToString(CultureInfo.InvariantCulture)}]" :
data.ThreeDimData[k][i, j].ToString(CultureInfo.InvariantCulture);
                    }
                    dataTable.Rows.Add(row);
                }
```

```
    }

    _dataGridView.DataSource = dataTable;

    var index = 0;
    for (var k = 0; k < kCount; k++)
    {
        _dataGridView.Rows[index++].HeaderCell.Value = $"k{k}";
        for (var i = 0; i < rowsCount; i++)
        {
            _dataGridView.Rows[index++].HeaderCell.Value = $"i{i}";
        }
    }
}

public void PopulateFromCsv(int dim, bool isInterval, string filePath)
{
    var dataTable = new DataTable();

    var config = new CsvConfiguration(CultureInfo.InvariantCulture)
    {
        HasHeaderRecord = false,
    };

    using (var reader = new StreamReader(filePath))
    using (var csv = new CsvReader(reader, config))
    {
        switch (dim)
        {
            case 1:
                {
                    ReadOneDimCsv(csv, isInterval);
                    break;
                }
            case 2:
                {
                    ReadTwoDimCsv(csv, isInterval);
                    break;
                }
            case 3:
                {
                    ReadThreeDimCsv(csv, isInterval);
                    break;
                }
        }

        Populate(DataGridModel);
    }
}

private void ReadOneDimCsv(CsvReader csv, bool isInterval)
{
    if (isInterval)
    {
        var records = csv.GetRecords<dynamic>().ToArray();

        int rowCount = records.Length;
        var key = ((IDictionary<string, object>)records[0]).Keys.First();
        var format = key.Remove(key.Length - 1);
        int colCount = ((IDictionary<string, object>)records[0]).Values.Count;

        var resultL = new float[rowCount];
        var resultU = new float[rowCount];
```

```
            for (int i = 0; i < rowCount; i++)
            {
               for (int j = 0; j < colCount; j++)
               {
                  var str = (string)Convert.ChangeType(((IDictionary<string, object>)records[i])[$"{format}{j + 1}"],
typeof(string));
                  if (j == 0)
                  {
                     resultL[i] = float.Parse(str, CultureInfo.InvariantCulture);
                  }
                  else if (j == 1)
                  {
                     resultU[i] = float.Parse(str, CultureInfo.InvariantCulture);
                  }
               }
            }
            DataGridModel = new DataGridModel() { Dim = 1, IsInterval = isInterval, OneDimData = resultL,
OneDimDataU = resultU };
         }
         else
         {
            var records = csv.GetRecords<float>().ToArray();
            DataGridModel = new DataGridModel() { Dim = 1, IsInterval = isInterval, OneDimData = records };
         }
      }

      private void ReadTwoDimCsv(CsvReader csv, bool isInterval)
      {
         if (isInterval)
         {
            var records = csv.GetRecords<dynamic>().ToArray();

            int rowCount = records.Length;
            var key = ((IDictionary<string, object>)records[0]).Keys.First();
            var format = key.Remove(key.Length - 1);
            int colCount = ((IDictionary<string, object>)records[0]).Values.Count;

            var resultL = new float[rowCount, colCount/2];
            var resultU = new float[rowCount, colCount/2];

            for (int i = 0; i < rowCount; i++)
            {
               var realJ = 0;
               for (int j = 0; j < colCount; j++)
               {
                  var str = (string)Convert.ChangeType(((IDictionary<string, object>)records[i])[$"{format}{j + 1}"],
typeof(string));
                  resultL[i, realJ] = float.Parse(str, CultureInfo.InvariantCulture);

                  j++;

                  var strU = (string)Convert.ChangeType(((IDictionary<string, object>)records[i])[$"{format}{j + 1}"],
typeof(string));
                  resultU[i, realJ] = float.Parse(strU, CultureInfo.InvariantCulture);

                  realJ++;
               }
            }

            DataGridModel = new DataGridModel() { Dim = 2, IsInterval = isInterval, TwoDimData = resultL,
TwoDimDataU = resultU };
         }
```

```
        else
        {
            var records = csv.GetRecords<dynamic>().ToArray();

            int rowCount = records.Length;
            var key = ((IDictionary<string, object>)records[0]).Keys.First();
            var format = key.Remove(key.Length - 1);
            int colCount = ((IDictionary<string, object>)records[0]).Values.Count;

            var result = new float[rowCount, colCount];

            for (int i = 0; i < rowCount; i++)
            {
                for (int j = 0; j < colCount; j++)
                {
                    var str = (string)Convert.ChangeType(((IDictionary<string, object>)records[i])[$"{format}{j + 1}"],
typeof(string));
                    result[i, j] = float.Parse(str, CultureInfo.InvariantCulture);
                }
            }

            DataGridModel = new DataGridModel() { Dim = 2, IsInterval = isInterval, TwoDimData = result };
        }
    }

    private void ReadThreeDimCsv(CsvReader csv, bool isInterval)
    {
        if (isInterval)
        {
            var records = csv.GetRecords<dynamic>().ToArray();

            int rowCount = records.Length;
            var key = ((IDictionary<string, object>)records[0]).Keys.First();
            var format = key.Remove(key.Length - 1);
            int colCount = ((IDictionary<string, object>)records[0]).Values.Count;

            var tablesCount = 0;
            int? rowsCount = null;
            for (int i = 0; i < rowCount; i++)
            {
                var str = (string)Convert.ChangeType(((IDictionary<string, object>)records[i])[$"{format}{1}"],
typeof(string));

                if (str == "-")
                {
                    tablesCount++;
                    if (!rowsCount.HasValue)
                    {
                        rowsCount = i;
                    }
                }
            }

            var resultL = new float[tablesCount][,];
            var resultU = new float[tablesCount][,];
            for (var k = 0; k < tablesCount; k++)
            {
                resultL[k] = new float[rowsCount.Value, colCount / 2];
                resultU[k] = new float[rowsCount.Value, colCount / 2];
            }

            int row = 0;
            for (var k = 0; k < tablesCount; k++)
```

```
            {
                for (int i = 0; i < rowsCount; i++)
                {
                    var realJ = 0;
                    for (int j = 0; j < colCount; j++)
                    {
                        var str = (string)Convert.ChangeType(((IDictionary<string, object>)records[row])[$"{format}{j + 1}"],
typeof(string));
                        resultL[k][i, realJ] = float.Parse(str, CultureInfo.InvariantCulture);
                        j++;

                        var strU = (string)Convert.ChangeType(((IDictionary<string, object>)records[row])[$"{format}{j + 1}"],
typeof(string));
                        resultU[k][i, realJ] = float.Parse(strU, CultureInfo.InvariantCulture);

                        realJ++;
                    }
                    row++;
                }
                row++;
            }

            DataGridModel = new DataGridModel() { Dim = 3, IsInterval = isInterval, ThreeDimData = resultL,
ThreeDimDataU = resultU };
        }
        else
        {
            var records = csv.GetRecords<dynamic>().ToArray();

            int rowCount = records.Length;
            var key = ((IDictionary<string, object>)records[0]).Keys.First();
            var format = key.Remove(key.Length - 1);
            int colCount = ((IDictionary<string, object>)records[0]).Values.Count;

            var tablesCount = 0;
            int? rowsCount = null;
            for (int i = 0; i < rowCount; i++)
            {
                var str = (string)Convert.ChangeType(((IDictionary<string, object>)records[i])[$"{format}{1}"],
typeof(string));

                if (str == "-")
                {
                    tablesCount++;
                    if (!rowsCount.HasValue)
                    {
                        rowsCount = i;
                    }
                }
            }

            var result = new float[tablesCount][,];
            for (var k = 0; k < tablesCount; k++)
            {
                result[k] = new float[rowsCount.Value, colCount];
            }
            int row = 0;
            for (var k = 0; k < tablesCount; k++)
            {
                for (int i = 0; i < rowsCount; i++)
                {
                    for (int j = 0; j < colCount; j++)
                    {
```

```
                    var str = (string)Convert.ChangeType(((IDictionary<string, object>)records[row])[$"{format}{j + 1}"],
typeof(string));
                    result[k][i, j] = float.Parse(str, CultureInfo.InvariantCulture);
                }
                row++;
            }
            row++;
        }

        DataGridModel = new DataGridModel() { Dim = 3, IsInterval = isInterval, ThreeDimData = result };
    }
  }
}

using ArtificialBeeColonyApp.UserControls;
using ArtificialBeeColonyLibrary.Data;
using ScottPlot;
using ScottPlot.WinForms;
using System.Globalization;

namespace ArtificialBeeColonyApp.Controllers
{
  internal class PlotController
  {
    private FormsPlot _plot;
    private Panel _panel;
    private List<ParameterInputTextField> _parameters;

    public PlotController(FormsPlot plot, Panel panel)
    {
      _plot = plot;
      _panel = panel;
      _parameters = new List<ParameterInputTextField>();

      _plot.Plot.Axes.Bottom.Label.Text = "X";
      _plot.Plot.Axes.Left.Label.Text = "Y";
    }

    public void AddParameters(float[] g)
    {
      AddParameters(g.Length);

      for (var i = 0; i < g.Length; i++)
      {
        _parameters[i].SetParameter(g[i].ToString(CultureInfo.InvariantCulture));
      }
    }

    public void AddParameters(int gLength)
    {
      foreach (var control in _parameters)
      {
        _panel.Controls.Remove(control);
      }
      _parameters.Clear();

      for (var i = 0; i < gLength; i++)
      {
        var gInput = new ParameterInputTextField();
        gInput.ParameterName = $"G{i}";
        gInput.ParameterDescription = $"G{i}";
```

```
            gInput.Location = new Point(0, 26 + (2 + i) * 31);

            _panel.Controls.Add(gInput);
            _parameters.Add(gInput);
        }
    }

    public float[] GetParameters(ref bool success)
    {
        var result = new float[_parameters.Count];
        for (var i = 0; i < result.Length; i++)
        {
            var value = _parameters[i].TryGetFloat(ref success);
            if (!success) return result;
            result[i] = value;
        }
        return result;
    }

    public void DrawPlot(PlotData plotData, string xAxis, string yAxis)
    {
        _plot.Plot.Clear();
        _plot.Refresh();
        if (plotData != null)
        {
            var x = new double[plotData.Length];
            var y1 = new double[plotData.Length];
            var y2 = new double[plotData.Length];
            var y3 = new double[plotData.Length];

            for (var i = 0; i < plotData.Length; i++)
            {
                x[i] = i;
                y1[i] = plotData.Initial[i].Right;
                y2[i] = plotData.Initial[i].Left;
                y3[i] = plotData.Calculated[i].Middle();
            }

            var s1 = _plot.Plot.Add.Signal(y1);
            s1.Color = ScottPlot.Color.FromHex("#FF0000");
            s1.Label = "Interval limits of measured values";

            var s2 = _plot.Plot.Add.Signal(y2);
            s2.Color = ScottPlot.Color.FromHex("#FF0000");

            var s3 = _plot.Plot.Add.Signal(y3);
            s3.Color = ScottPlot.Color.FromHex("#008000");
            s3.Label = "Modeled values";

            if (!string.IsNullOrWhiteSpace(xAxis))
            {
                _plot.Plot.Axes.Bottom.Label.Text = xAxis;
            }

            if (!string.IsNullOrWhiteSpace(yAxis))
            {
                _plot.Plot.Axes.Left.Label.Text = yAxis;
            }

            _plot.Plot.ShowLegend(Alignment.UpperLeft);

            _plot.Plot.Axes.AutoScale();
            _plot.Refresh();
```

```
        }
      }
    }
}

using ArtificialBeeColonyApp.Models;
using ArtificialBeeColonyLibrary.Data;
using Newtonsoft.Json;

namespace ArtificialBeeColonyApp.Data
{
    internal class AbcProjectData
    {
        public int S { get; set; }
        public int Mcn { get; set; }
        public int Limit { get; set; }
        public float DeltaCap { get; set; }
        public float RelativeDeltaCap { get; set; }
        public ParallelOption ParallelOption { get; set; }
        public int ThreadsNumber { get; set; }
        public bool ResetLimit { get; set; }
        public float[] LowerBounds { get; set; }
        public float[] UpperBounds { get; set; }
        public int NumberOfRuns { get; set; }
        public int GLength { get; set; }
        public float DeltaZ { get; set; }
        public float DeltaV { get; set; }
        public int InitK { get; set; }
        public int InitI { get; set; }
        public int InitJ { get; set; }
        public string Func { get; set; }
        public bool EvaluationCount { get; set; }
        public bool MeasureTime { get; set; }
        public DataGridModel DataGridModel {  get; set; }
        public DataGridModel TestDataGridModel {  get; set; }
        public DataGridModel ControlDataGridModel { get; set; }
        public DataGridModel TestControlDataGridModel { get; set; }
        public float[] Result { get; set; }
        public StructuralData StructuralData { get; set; }
    }
}

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

namespace ArtificialBeeColonyApp.Data
{
    internal class StructuralData
    {
        public bool IsEnabled { get; set; }
        public int S { get; set; }
        public int Mcn { get; set; }
        public int Limit { get; set; }
        public float DeltaCap { get; set; }
        public float RelativeDeltaCap { get; set; }
        public int OrderK { get; set; }
        public int OrderI { get; set; }
        public int OrderJ { get; set; }
        public int Power { get; set; }
        public int MinI { get; set; }
```

```
        public int MaxI { get; set; }
        public bool Multiplication { get; set; }
        public bool Division { get; set; }
        public List<string> Elements { get; set; } = new List<string>();

    }
}

using ArtificialBeeColonyLibrary.Data;

namespace ArtificialBeeColonyApp.Models
{
    public class DataGridModel
    {
        public int Dim { get; set; }
        public bool IsInterval { get; set; }
        public float[] OneDimData { get; set; }
        public float[] OneDimDataU { get; set; }
        public float[,] TwoDimData { get; set; }
        public float[,] TwoDimDataU { get; set; }
        public float[][,] ThreeDimData { get; set; }
        public float[][,] ThreeDimDataU { get; set; }

        public Interval[] GetOneDimIntervalData()
        {
            var result = new Interval[OneDimData.Length];
            for (var i = 0; i < OneDimData.Length; i++)
            {
                result[i] = new Interval(OneDimData[i], OneDimDataU[i]);
            }
            return result;
        }

        public Interval[,] GetTwoDimIntervalData()
        {
            var I = TwoDimData.GetLength(0);
            var J = TwoDimData.GetLength(1);
            var result = new Interval[I, J];
            for (var i = 0; i < I; i++)
            {
                for (var j = 0; j < J; j++)
                {
                    result[i, j] = new Interval(TwoDimData[i, j], TwoDimDataU[i, j]);
                }
            }
            return result;
        }

        public Interval[][,] GetThreeDimIntervalData()
        {
            var K = ThreeDimData.Length;
            var I = ThreeDimData[0].GetLength(0);
            var J = ThreeDimData[0].GetLength(1);
            var result = new Interval[K][, ];
            for (var k = 0; k < K; k++)
            {
                result[k] = new Interval[I, J];
                for (var i = 0; i < I; i++)
                {
                    for (var j = 0; j < J; j++)
                    {
                        result[k][i, j] = new Interval(ThreeDimData[k][i, j], ThreeDimDataU[k][i, j]);
                    }
```

```
            }
        }
        return result;
    }

    public float[,] GetControlParams()
    {
        if (Dim == 2)
        {
            return TwoDimData;
        }

        return null;

    }
  }
}


using ArtificialBeeColonyApp.ABC;
using ArtificialBeeColonyApp.Controllers;
using ArtificialBeeColonyApp.Data;
using ArtificialBeeColonyApp.UserControls;
using ArtificialBeeColonyApp.Utils;
using ArtificialBeeColonyLibrary.Data;
using ArtificialBeeColonyLibrary.Utils;
using Newtonsoft.Json;
using System.Globalization;
using System.Text;
using System.Xml.Linq;

namespace ArtificialBeeColonyApp;

public partial class MainWindow : Form
{
  private OverlayPanel overlay;
  private ABCRunner abcRunner;
  private ABCPreparer abcPreparer;
  private ABCChecker abcChecker;

  private DataGridController inputDataGridController;
  private DataGridController testDataGridController;
  private DataGridController controlParamsDataGridController;
  private DataGridController testControlParamsDataGridController;
  private PlotController plotController;

  private string projectName = "DefaulProjectName";

  public MainWindow()
  {
    InitializeComponent();
    InitializeOverlay();

    inputDataGridController = new DataGridController(dataGridView);
    testDataGridController = new DataGridController(testDataGridView);
    controlParamsDataGridController = new DataGridController(controlParamsDataGridView);
    testControlParamsDataGridController = new DataGridController(testControlParamsdataGridView);

    if (!CudaModelBuilder.IsCudaAvailable())
    {
      parallelParameterInput.DisableRadioButtonTwo();
    }
```

```
        plotController = new PlotController(plot, plotPanel);

        abcRunner = new ABCRunner(resultTextBox);
        abcPreparer = new ABCPreparer();
        abcChecker = new ABCChecker(plotResultTextBox);
    }

    private void InitializeOverlay()
    {
        overlay = new OverlayPanel();
        this.Controls.Add(overlay);
        overlay.BringToFront();
    }

    private void openProjectToolStripMenuItem_Click(object sender, EventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog
        {
            Filter = "Json files (*.json)|*.json",
            RestoreDirectory = true,
        };

        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            string filePath = openFileDialog.FileName;
            var json = File.ReadAllText(filePath);
            var abcProjectData = JsonConvert.DeserializeObject<AbcProjectData>(json);
            SetProjectData(abcProjectData);
            projectName = Path.GetFileNameWithoutExtension(filePath);
        }
    }

    private void SetProjectData(AbcProjectData abcProjectData)
    {
        sParameterInput.SetParameter(abcProjectData.S.ToString(CultureInfo.InvariantCulture));
        mcnParameterInput.SetParameter(abcProjectData.Mcn.ToString(CultureInfo.InvariantCulture));
        limitParameterInput.SetParameter(abcProjectData.Limit.ToString(CultureInfo.InvariantCulture));
        deltaCapParameterInput.SetParameter(abcProjectData.DeltaCap.ToString(CultureInfo.InvariantCulture));

relativeDeltaCapParameterInput.SetParameter(abcProjectData.RelativeDeltaCap.ToString(CultureInfo.InvariantCulture));
        parallelParameterInput.SetParameter(abcProjectData.ParallelOption);

threadsNumberParameterInput.SetParameter(abcProjectData.ThreadsNumber.ToString(CultureInfo.InvariantCulture));
        resetLimitParameterInput.SetParameter(abcProjectData.ResetLimit);
        lowerBoundsParameterInput.SetParameter(abcProjectData.LowerBounds.ToStringAbc());
        upperBoundsParameterInput.SetParameter(abcProjectData.UpperBounds.ToStringAbc());
        numberOfRunsParameterInput.SetParameter(abcProjectData.NumberOfRuns.ToString(CultureInfo.InvariantCulture));
        gLengthParameterInput.SetParameter(abcProjectData.GLength.ToString(CultureInfo.InvariantCulture));
        deltaZParameterInput.SetParameter(abcProjectData.DeltaZ.ToString(CultureInfo.InvariantCulture));
        deltaVParameterInput.SetParameter(abcProjectData.DeltaV.ToString(CultureInfo.InvariantCulture));

        SetInitAvailability(abcProjectData.DataGridModel.Dim);

        if (abcProjectData.DataGridModel.Dim == 1 || abcProjectData.DataGridModel.Dim == 3)
        {
            initKParameterInput.SetParameter(abcProjectData.InitK.ToString(CultureInfo.InvariantCulture));
        }
        if (abcProjectData.DataGridModel.Dim == 2 || abcProjectData.DataGridModel.Dim == 3)
        {
            initIParameterInput.SetParameter(abcProjectData.InitI.ToString(CultureInfo.InvariantCulture));
            initJParameterInput.SetParameter(abcProjectData.InitJ.ToString(CultureInfo.InvariantCulture));
        }
```

```
        funcParameterInput.SetParameter(abcProjectData.Func);
        evaluationCountParameterInput.SetParameter(abcProjectData.EvaluationCount);

        inputDataGridController.Populate(abcProjectData.DataGridModel);
        dimensionParameterInput.SetParameter(abcProjectData.DataGridModel.Dim.ToString(CultureInfo.InvariantCulture));
        isIntervalCheckBox.SetParameter(abcProjectData.DataGridModel.IsInterval);
        testDataGridController.Populate(abcProjectData.TestDataGridModel);

        controlParamsDataGridController.Populate(abcProjectData.ControlDataGridModel);
        testControlParamsDataGridController.Populate(abcProjectData.TestControlDataGridModel);

        if (abcProjectData.Result != null)
        {
            var sb = new StringBuilder();
            var i = 0;
            foreach (var p in abcProjectData.Result)
            {
                sb.AppendFormat("g{0}={1} ", i++, p);
            }

            resultTextBox.Text = sb.ToString();

            abcRunner.SetResult(abcProjectData.Result);
        }

        if (abcProjectData.StructuralData != null)
        {
            if (abcProjectData.DataGridModel.Dim == 1 || abcProjectData.DataGridModel.Dim == 3)
            {

orderKParameterInput.SetParameter(abcProjectData.StructuralData.OrderK.ToString(CultureInfo.InvariantCulture));
            }
            if (abcProjectData.DataGridModel.Dim == 2 || abcProjectData.DataGridModel.Dim == 3)
            {

orderIParameterInput.SetParameter(abcProjectData.StructuralData.OrderI.ToString(CultureInfo.InvariantCulture));

orderJParameterInput.SetParameter(abcProjectData.StructuralData.OrderJ.ToString(CultureInfo.InvariantCulture));
            }

            multParameterInput.SetParameter(abcProjectData.StructuralData.Multiplication);
            divParameterInput.SetParameter(abcProjectData.StructuralData.Division);
            powerParameterInput.SetParameter(abcProjectData.StructuralData.Power.ToString(CultureInfo.InvariantCulture));
            structuralSParameterInput.SetParameter(abcProjectData.StructuralData.S.ToString(CultureInfo.InvariantCulture));

structuralMCNParameterInput.SetParameter(abcProjectData.StructuralData.Mcn.ToString(CultureInfo.InvariantCulture));

structuralLimitParameterInput.SetParameter(abcProjectData.StructuralData.Limit.ToString(CultureInfo.InvariantCulture));

structuralDeltaCapParameterInput.SetParameter(abcProjectData.StructuralData.DeltaCap.ToString(CultureInfo.InvariantC
ulture));

structuralRelativeDeltaCapParameterInput.SetParameter(abcProjectData.StructuralData.RelativeDeltaCap.ToString(Cultur
eInfo.InvariantCulture));

structuralIMinParameterInput.SetParameter(abcProjectData.StructuralData.MinI.ToString(CultureInfo.InvariantCulture));

structuralIMaxParameterInput.SetParameter(abcProjectData.StructuralData.MaxI.ToString(CultureInfo.InvariantCulture));
            structuralIdentificationEnabled.SetParameter(abcProjectData.StructuralData.IsEnabled);

            gLengthParameterInput.Enabled = false;
            gLengthParameterInput.SetParameter((abcProjectData.StructuralData.MaxI +
1).ToString(CultureInfo.InvariantCulture));
```

```
        elementsTextBox.Text = string.Join(Environment.NewLine, abcProjectData.StructuralData.Elements);
    }
}

private void SetInitAvailability(int dim)
{
    if (dim == 1)
    {
        initKParameterInput.Enabled = true;
        initIParameterInput.Enabled = false;
        initJParameterInput.Enabled = false;

        orderKParameterInput.Enabled = true;
        orderIParameterInput.Enabled = false;
        orderJParameterInput.Enabled = false;
    }
    else if (dim == 2)
    {
        initKParameterInput.Enabled = false;
        initIParameterInput.Enabled = true;
        initJParameterInput.Enabled = true;

        orderKParameterInput.Enabled = false;
        orderIParameterInput.Enabled = true;
        orderJParameterInput.Enabled = true;
    }
}

private void saveProjectToolStripMenuItem_Click(object sender, EventArgs e)
{
    var abcProjectData = GetProjectData();

    if (abcProjectData == null) return;

    SaveFileDialog saveFileDialog = new SaveFileDialog();

    saveFileDialog.DefaultExt = "json";
    saveFileDialog.Filter = "Json files (*.json)|*.json";

    DialogResult result = saveFileDialog.ShowDialog();

    if (result == DialogResult.OK)
    {
        string fileName = saveFileDialog.FileName;
        var json = JsonConvert.SerializeObject(abcProjectData, Formatting.Indented);
        File.WriteAllText(fileName, json);
    }
}

private AbcProjectData? GetProjectData()
{
    bool success = true;
    var abcProjectData = new AbcProjectData();

    abcProjectData.DataGridModel = inputDataGridController.DataGridModel;
    abcProjectData.TestDataGridModel = testDataGridController.DataGridModel;

    abcProjectData.ControlDataGridModel = controlParamsDataGridController.DataGridModel;
    abcProjectData.TestControlDataGridModel = testControlParamsDataGridController.DataGridModel;

    abcProjectData.S = sParameterInput.TryGetInt(ref success);
    abcProjectData.Mcn = mcnParameterInput.TryGetInt(ref success);
```

```
abcProjectData.Limit = limitParameterInput.TryGetInt(ref success);
abcProjectData.DeltaCap = deltaCapParameterInput.TryGetFloat(ref success);
abcProjectData.RelativeDeltaCap = relativeDeltaCapParameterInput.TryGetFloat(ref success);
abcProjectData.ParallelOption = parallelParameterInput.GetParameter();
abcProjectData.ThreadsNumber = threadsNumberParameterInput.TryGetInt(ref success);
abcProjectData.ResetLimit = resetLimitParameterInput.GetParameter();
abcProjectData.NumberOfRuns = numberOfRunsParameterInput.TryGetInt(ref success);
abcProjectData.GLength = gLengthParameterInput.TryGetInt(ref success);
abcProjectData.DeltaZ = deltaZParameterInput.TryGetFloat(ref success);
abcProjectData.DeltaV = deltaVParameterInput.TryGetFloat(ref success);

if (abcProjectData.DataGridModel.Dim == 1 || abcProjectData.DataGridModel.Dim == 3)
{
   abcProjectData.InitK = initKParameterInput.TryGetInt(ref success);
}
if (abcProjectData.DataGridModel.Dim == 2 || abcProjectData.DataGridModel.Dim == 3)
{
   abcProjectData.InitI = initIParameterInput.TryGetInt(ref success);
   abcProjectData.InitJ = initJParameterInput.TryGetInt(ref success);
}

var lowerBounds = lowerBoundsParameterInput.GetParameter().ToArrayAbc();
if (lowerBounds.Length != abcProjectData.GLength)
{
   lowerBoundsParameterInput.SetError();
   success = false;
}
else
{
   abcProjectData.LowerBounds = lowerBounds;
}

var upperBounds = upperBoundsParameterInput.GetParameter().ToArrayAbc();
if (upperBounds.Length != abcProjectData.GLength)
{
   upperBoundsParameterInput.SetError();
   success = false;
}
else
{
   abcProjectData.UpperBounds = upperBounds;
}

var func = funcParameterInput.GetParameter();
if (string.IsNullOrEmpty(func))
{
   funcParameterInput.SetError();
   success = false;
}
else
{
   abcProjectData.Func = func;
}

abcProjectData.EvaluationCount = evaluationCountParameterInput.GetParameter();

var structuralEnabled = structuralIdentificationEnabled.GetParameter();

if (structuralEnabled)
{
   abcProjectData.StructuralData = new StructuralData();
   abcProjectData.StructuralData.IsEnabled = structuralEnabled;
   if (abcProjectData.DataGridModel.Dim == 1 || abcProjectData.DataGridModel.Dim == 3)
```

```
            {
                abcProjectData.StructuralData.OrderK = orderKParameterInput.TryGetInt(ref success);
            }
            if (abcProjectData.DataGridModel.Dim == 2 || abcProjectData.DataGridModel.Dim == 3)
            {
                abcProjectData.StructuralData.OrderI = orderIParameterInput.TryGetInt(ref success);
                abcProjectData.StructuralData.OrderJ = orderJParameterInput.TryGetInt(ref success);
            }
            abcProjectData.StructuralData.Multiplication = multParameterInput.GetParameter();
            abcProjectData.StructuralData.Division = divParameterInput.GetParameter();
            abcProjectData.StructuralData.Power = powerParameterInput.TryGetInt(ref success);
            abcProjectData.StructuralData.Elements = elementsTextBox.Text.Split(Environment.NewLine).ToList();

            abcProjectData.StructuralData.S = structuralSParameterInput.TryGetInt(ref success);
            abcProjectData.StructuralData.Mcn = structuralMCNParameterInput.TryGetInt(ref success);
            abcProjectData.StructuralData.Limit = structuralLimitParameterInput.TryGetInt(ref success);
            abcProjectData.StructuralData.DeltaCap = structuralDeltaCapParameterInput.TryGetFloat(ref success);
            abcProjectData.StructuralData.RelativeDeltaCap = structuralRelativeDeltaCapParameterInput.TryGetFloat(ref
success);
            abcProjectData.StructuralData.MinI = structuralIMinParameterInput.TryGetInt(ref success);
            abcProjectData.StructuralData.MaxI = structuralIMaxParameterInput.TryGetInt(ref success);
        }

        return success ? abcProjectData : null;
    }

    private async void calculateButton_Click(object sender, EventArgs e)
    {
        var cts = new CancellationTokenSource();
        overlay.ShowLoadingAnimation(cts);
        var abcProjectData = GetProjectData();
        if (abcProjectData == null)
        {
            overlay.HideLoadingAnimation();
            return;
        }
        abcPreparer.SetProjectData(abcProjectData);

        await abcRunner.Run(abcPreparer, cts.Token);
        overlay.HideLoadingAnimation();
    }

    private async void plotButton_Click(object sender, EventArgs e)
    {
        var success = true;
        var g = plotController.GetParameters(ref success);
        var ignoreInitValues = ignoreInitValuesCheckBox.GetParameter();
        var xAxis = xAsixParameterInput.GetParameter();
        var yAxis = yAsixParameterInput.GetParameter();

        if (!success) return;
        plotController.DrawPlot(await abcChecker.Run(abcPreparer, g, abcRunner.FuncResult, ignoreInitValues), xAxis,
yAxis);
    }

    private void loadCsvButton_Click(object sender, EventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog
        {
            Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*",
            RestoreDirectory = true,
        };
```

```
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        string filePath = openFileDialog.FileName;

        bool success = true;
        var dim = dimensionParameterInput.TryGetInt(ref success);
        var isInterval = isIntervalCheckBox.GetParameter();
        if (!success) return;

        inputDataGridController.PopulateFromCsv(dim, isInterval, filePath);
        SetInitAvailability(inputDataGridController.DataGridModel.Dim);
    }
}

private void fromDataButton_Click(object sender, EventArgs e)
{
    testDataGridController.Populate(inputDataGridController.DataGridModel);
}

private void loadTestDataButton_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog
    {
        Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*",
        RestoreDirectory = true,
    };

    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        string filePath = openFileDialog.FileName;

        bool success = true;
        var dim = dimensionParameterInput.TryGetInt(ref success);
        var isInterval = isIntervalCheckBox.GetParameter();
        if (!success) return;

        testDataGridController.PopulateFromCsv(dim, isInterval, filePath);
    }
}

private void transferToPlotButton_Click(object sender, EventArgs e)
{
    if (abcRunner.Result != null)
    {
        plotController.AddParameters(abcRunner.Result.G);
    }
    else
    {
        var s = true;
        plotController.AddParameters(gLengthParameterInput.TryGetInt(ref s));
    }

    tabControl.SelectedIndex = 6;
}

private void loadControlParamsButton_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog
    {
        Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*",
        RestoreDirectory = true,
    };
```

```
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        string filePath = openFileDialog.FileName;
        controlParamsDataGridController.PopulateFromCsv(2, false, filePath);
    }
}

private void copyFromControlBtn_Click(object sender, EventArgs e)
{
    testControlParamsDataGridController.Populate(controlParamsDataGridController.DataGridModel);
}

private void loadTestControlParamsButton_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog
    {
        Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*",
        RestoreDirectory = true,
    };

    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        string filePath = openFileDialog.FileName;
        testControlParamsDataGridController.PopulateFromCsv(2, false, filePath);
    }
}

private void generateElementsButton_Click(object sender, EventArgs e)
{
    bool success = true;

    var dim = dimensionParameterInput.TryGetInt(ref success);
    var orderK = 0;
    if (dim == 1 || dim == 3)
    {
        orderK = orderKParameterInput.TryGetInt(ref success);
    }
    var orderI = 0;
    var orderJ = 0;
    if (dim == 2 || dim == 3)
    {
        orderI = orderIParameterInput.TryGetInt(ref success);
        orderJ = orderJParameterInput.TryGetInt(ref success);
    }

    var power = powerParameterInput.TryGetInt(ref success);
    var mult = multParameterInput.GetParameter();
    var div = divParameterInput.GetParameter();

    if (success)
    {
        var elements = new List<string>();
        switch (dim)
        {
            case 1:
                elements = ElementsGenerator.GenerateOneDimElements(orderK, power, mult, div);
                break;
            case 2:
                elements = ElementsGenerator.GenerateTwoDimElements(orderI, orderJ, power, mult, div);
                break;
            case 3:
                elements = ElementsGenerator.GenerateThreeDimElements(orderK, orderI, orderJ, power, mult, div);
                break;
```

```
            default:
                return;
        }
        generatedElementsTextBox.Text = string.Join(Environment.NewLine, elements);
    }
}

private void structuralIMaxParameterInput_OnParameterChanged()
{
    bool s = true;
    var maxI = structuralIMaxParameterInput.TryGetInt(ref s) + 1;
    gLengthParameterInput.SetParameter(maxI.ToString(CultureInfo.InvariantCulture));
}
}
```

ДОДАТОК Б. СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

1. M. Dyvak, P. Tyande, O. Kindzerskyi Mathematical Model of a Social Network User Profile Based on Interval Data Analysis, International Journal of Computing, vol. 24, no. 3, pp. 452-459, Oct. 2025. ISSN: 2312-538, doi:10.47839/ijc.24.3.4182
Url: https://computingonline.net/computing/article/view/4182

2. М. Дивак, О. Кіндзерський Архітектура програмного забезпечення структурної та параметричної ідентифікації на основі алгоритму штучної бджолиної колонії з використанням технології NVIDIA CUDA, НаукПраці ВНТУ, вип. 2, Чер 2025, ISSN: 2307-5376, doi:10.31649/2307-5376-2025-2-41-50
Url: https://praci.vntu.edu.ua/index.php/praci/article/view/837

3. М. Дивак, О. Кіндзерський Дослідження ефективності паралельної обчислювальної схеми ідентифікації інтервальних дискретних моделей на основі ройового інтелекту Том 331 № 1 (2024): Вісник Хмельницького національного університету. Серія: Технічні науки, с.29-37, doi:10.31891/2307-5732-2024-331-3, ISSN: 2307-5732,
Url: https://heraldts.khmnu.edu.ua/index.php/heraldts/issue/view/2

4. M. Dyvak, O. Kindzerskyi “Implementation of the structural identification for interval models based on the behavioral model of an artificial bee colony,” 2025 15th International Conference on Advanced Computer Information Technologies (ACIT), Sibenik, Croatia, 2025, pp.98-101, doi:10.1109/ACIT65614.2025.11185828, ISSN: 2770-5218

5. M. Dyvak, N. Petryshyn, O. Kindzerskyi, O. Papa, Y. Franko and O. Opalko, "Modeling of the Efficiency of Electricity Generation Processes by a Solar Power Plant Research Using the Example of a 570 W Model," 2025 15th International Conference on Advanced Computer Information Technologies (ACIT), Sibenik, Croatia, 2025, pp. 92-97, doi:10.1109/ACIT65614.2025.11185608, ISNN: 2770-5218

6. M. Dyvak, O. Kindzerskyi “Implementation of Parallel Computation for Identification of Interval Models based on Multi-core Parallelism and CUDA

Technology," 2024 14th International Conference on Advanced Computer Information Technologies (ACIT), 2024, pp.72-76, doi:10.1109/ACIT62333.2024.10712545. ISSN: 2770-5218

7. M. Dyvak, I. Spivak, T. Dyvak, O. Kindzerskyi "Modeling the Interaction of Unmanned Aerial Vehicles in a Swarm as an Object with Distributed Parameters," 2024 14th International Conference on Advanced Computer Information Technologies (ACIT), 2024, pp.60-66., doi:10.1109/ACIT62333.2024.10712502. ISSN: 2770-5218

8. M. Dyvak, O. Kindzerskyi, L. Dostalek, M. Stetsko and J. Nowak "Parallel Computations in the Problem of Identification of Interval Discrete Models based on Swarm Intelligence of a Bee Colony," 2023 13th International Conference on Advanced Computer Information Technologies (ACIT), 2023, pp. 23-28, doi:10.1109/ACIT58437.2023.10275695, ISSN: 2770-5218

ДОДАТОК В. АКТИ ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОГО ДОСЛІДЖЕННЯ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАУКОВО-ДОСЛІДНА ЧАСТИНА
46009, Україна, м. Тернопіль, вул. Львівська, буд. 5А, тел. (0352) 51-75-52

№ 21/104-2025 «22» 12 2025 року

ДОВІДКА
про участь у виконанні НДР

Видана **КІНДЗЕРСЬКОМУ Олександру Віталійовичу** про участь у виконанні науково-дослідних робіт Західноукраїнського національного університету. Зокрема, у 2024 році на посаді молодшого наукового співробітника він прийняв участь у виконанні молодіжного держбюджетного дослідження «Математичне та програмне забезпечення прототипу біогазової установки з підвищеною ефективністю функціонування» (державний реєстраційний номер 0124U000076); у 2024-2025 роках долучився до виконання НДР в межах основного робочого часу професорсько-викладацького персоналу, докторантів, аспірантів та здобувачів наукового ступеня кафедри комп'ютерних наук «Методи та програмні засоби для ідентифікації інтервальних моделей складних систем» (державний реєстраційний номер 0122U000627).

Начальник
науково-дослідної частини Віта СЕМАНЮК

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

вул. Львівська, 11, м. Тернопіль, 46009; тел./факс +380 (352) 51-75-75;
www.wunu.edu.ua; rektor@wunu.edu.ua; ідентифікаційний код за ЄДРПОУ 33680120

№ 126-27/2941 29 грудня 2025 р.

ЗАТВЕРДЖУЮ
Проректор з науково-педагогічної роботи
Західноукраїнського національного
університету, к.е.н., доцент
_____ Віктор ОСТРОВЕРХОВ

АКТ
про впровадження в освітній процес Західноукраїнського національного університету результатів дисертаційної роботи
Кіндзерського Олександра Віталійовича
«Ідентифікація інтервальних моделей систем програмними агентами бджолиної колонії у середовищі NVIDIA CUDA»

Даний акт складений про те, що результати дисертаційної роботи здобувача ступеня доктора філософії Кіндзерського Олександра Віталійовича на тему: «Ідентифікація інтервальних моделей систем програмними агентами бджолиної колонії у середовищі NVIDIA CUDA» використані в освітньому процесі кафедри комп'ютерних наук факультету комп'ютерних інформаційних технологій Західноукраїнського національного університету, зокрема під час підготовки студентів спеціальності 121 – Інженерія програмного забезпечення. При викладанні дисциплін «Архітектура та проєктування програмного забезпечення» та «Об'єктно-орієнтоване програмування» розглядається архітектура програмних засобів паралельної ідентифікації інтервальних моделей систем, що поєднує об'єктно-орієнтовану структуру компонентів із модульною організацією обчислювальних ядер CUDA, що забезпечує гнучкість, масштабованість та ефективне використання GPU-ресурсів, тоді як у рамках

курсу «Паралельні та розподілені обчислення» розглядаються практичні аспекти агентної взаємодії на спеціалізованому апаратному забезпеченні, а також методи паралельної обробки даних.

Декан факультету комп'ютерних
інформаційних технологій,
к.т.н., доцент Ігор ЯКИМЕНКО

Завідувач кафедри комп'ютерних наук,
д.т.н., професор Андрій ПУКАС