УДК 517.9

СИСТЕМА MULTIMATHFRAMEWORK ДЛЯ РОЗВ'ЯЗУВАННЯ ПОЧАТКОВО-КРАЙОВИХ ЗАДАЧ З ВИКОРИСТАННЯМ МЕТОДУ ГРАНИЧНИХ ЕЛЕМЕНТІВ ТА ПЕРЕТВОРЕННЯ ЛАҐЕРРА

А. Глова[1], **О. Кір**[2]

Львівський національний університет імені Івана Франка,
вул. Університеська, 1, Львів, 79000,
e-mail: [1]*a.hlova@gmail.com*, [2]*oleggio.kir@gmail.com*

Розглянуто загальну архітектуру та способи використання системи MultiMath-Framework (MMF) для розв'язування початково-крайових задач. Структура цієї системи відповідає основним принципам роботи програмних комплексів, що призначені для вирішення математичних проблем. На відміну від існуючих рішень, MMF фокусується на конкретному класі задач і водночас надає каркас для подальшого розширення можливостей системи з використанням базових принципів об'єктно-орієнтованого програмування. Зокрема, основними структурними одиницями є графічний інтерфейс, ядро, модулі розв'язування задачі, обробки поверхонь та аналізу даних. Графічний інтерфейс є початковою точкою взаємодії користувача з системою. Він відповідає за завантаження та демонстрацію списку задач, а також заповнення користувачем вхідних параметрів. Ядро системи визначає загальні алгоритми та структури даних, виконуючи роль бібліотеки базових класів і посередника, що керує завантаженням та ініціалізацією модуля розв'язування задачі, відповідає за його комунікацію з іншими компонентами. Модуль розв'язування реалізує набір технологічних прийомів, зокрема відомих патернів проектування, та інструкцій для вирішення математичної проблеми. Він також використовує окремий модуль для генерації, збереження та візуалізації розбиття поверхні на граничні елементи. На етапі виконання кожна задача є окремим компонентом, що динамічно підвантажується в процес на підставі даних із конфігураційного файлу. Спроектовано інтерфейси ядра системи, яким має відповідати кожен компонент. Результати роботи модуля розв'язування зберігаються в архіві. Функціональність архіву та модуля аналізу призначена для візуалізації отриманих результатів, їхнього порівняння та використання в подальших обчислювальних експериментах. Ефективність розробленої архітектури фреймворку та його програмної реалізації продемонстровано чисельними розв'язками модельних еволюційних та еліптичних задач, відзначено високу точність результатів і підтверджено порядок їхньої збіжності.

Ключові слова: початково-крайова задача, метод граничних елементів, перетворення Лаґерра, поверхневі потенціали, алгоритм, програмний модуль, компонент, інтерфейс, архітектура програмного забезпечення, бібліотека класів, базовий клас, патерн проектування, принцип програмування, об'єктно-орієнтоване програмування, наслідування, композиція, кешування.

1. Вступ

Чисельне розв'язування нестаціонарних початково-крайових задач математичної фізики в областях зі складною геометрією в $\mathbb{R}^3$ є комплексним процесом, що потребує коректної математичної моделі, ефективних числових методів та відповідного програмного забезпечення (ПЗ). Порівняно зі стаціонарними задачами, наявність додаткового виміру – часу – ставить додаткові вимоги до кожного із зазначених аспектів. Мета нашої праці – формулювання концепції фреймворку

для чисельного розв'язування зазначених задач і розробка технологічних прийомів для його програмної реалізації. Під терміном "фреймворк" розуміємо систему (сукупність), що об'єднує числові методи та відповідне програмне забезпечення, за допомогою якого можна проводити обчислювальні експерименти стосовно чисельних розв'язків і в потрібний спосіб опрацьовувати отримані результати.

Система MultiMathFramework (MMF) ґрунтується на поєднанні перетворення Лаґерра (ПЛ, див., наприклад, [1, 11]) за часовою змінною і методу граничних елементів (МГЕ). Такий підхід був успішно застосований для чисельного розв'язування різних початково-крайових задач для хвильового рівняння в $\mathbb{R}^3$ [2, 3, 7, 11–14, 17, 18]. Зазначимо, що відомий своєю універсальністю МГЕ є застосовним також і до задач в областях з меншою розмірністю. Однак у зазначених працях акцент був зроблений на обчислювальних аспектах, пов'язаних саме з 3D-геометрією. Результатом дослідженя таких нестаціонарних задач стала сукупність дискретних моделей, які мають спільні, або тісно пов'язані між собою компоненти. Це також стосується розроблених алгоритмів і програмного забезпечення. Крім того, виявлено низку інших задач, на які легко поширити розроблений підхід. Перелічені чинники пояснюють актуальність розробки єдиного фреймворку MMF, який вдосконалював розроблене раніше ПЗ і передбачав можливість подальшого ефективного розвитку для розширення на нові класи нестаціонарних задач.

Серед відомих у галузі обчислювальних систем зазначимо базу знань і набір алгоритмів WolframAlpha [20] та пакет MatLab [16]. WolframAlpha зручно використовувати для отримання простих математичних рішень (наприклад, знаходжень інтегралів чи похідних) та для розв'язування рівнянь, статистичної обробки та візуалізації даних. Також цією системою можна скористатися для перевірки окремих етапів складних методів. MatLab надає подібний сервіс. Крім того, відкриті інтерфейси MatLab можна використати для імплементації в його загальних алгоритмах додаткових блоків власного програмного коду. Однак у цьому випадку можливості дослідника обмежені за критеріями продуктивності, гнучкості та подальшого розвитку ПЗ.

Отож, для досягнення окреслених вище цілей, розробка MultiMathFramework є актуальною. Аналіз предметної області, яка окреслює задачі та підходи до їхнього чисельного розв'язування, виконано у першому розділі. На підставі такого аналізу у другому розділі розроблено архітектуру MMF, описано основні модулі та компоненти. Крім того, з'ясовано джерела вхідних даних системи й інтерфейси, які дають змогу користувачеві опрацьовувати обчислені дані. В третьому розділі детально розглянуто побудову модуля розв'язування задачі з використанням програмного каркасу, заданого ядром системи. Тут наведено діаграму класів, зазначено технологічні прийоми для забезпечення гнучкості фреймворку стосовно вибору потрібної функціональності, а також здатності до його розширення. В четвертому розділі описано модуль генерації розбиття поверхонь з використанням сучасних бібліотек. В п'ятому розділі розглянуто модуль аналізу даних, що дає змогу шукати апостеріорні похибки та досліджувати якість результатів, отриманих під час обчислювальних експериментів.

2. Математичні моделі, що лежать в основі фреймворку

У рамках фреймворку дані початково-крайових задач будемо характеризувати за допомогою чотирьох основних атрибутів:

– тип рівняння;

– тип початкових умов;
– тип крайової умови;
– геометрія області, в якій розглядаємо задачу.

Сукупність даних, пов'язаних із зазначеними атрибутами, визначає множину задач, за допомогою яких можна моделювати конкретні фізичні явища. Якщо досліджувані величини змінюються в часі, то відповідні початково-крайові задачі розглядаємо як еволюційні [9]. У цьому випадку працюємо з функціями, які залежать від дійсного аргументу $t \in \mathbb{R}_+ := (0, \infty)$ і набувають значень у відповідних просторах Соболєва у деякій області $\Omega \subset \mathbb{R}^3$. Вважаємо, що межа області Γ є ліпшицевою поверхнею. Позначимо також $\mathbb{R}_+ := (0, \infty)$, $Q := \Omega \times \mathbb{R}_+$ і $\Sigma := \Gamma \times \mathbb{R}_+$.

Ключовим елементом математичної моделі є відповідне диференційне рівняння. Його отримують із загального рівняння

$$a_1 \frac{\partial^2 u}{\partial t^2} + a_2 \frac{\partial u}{\partial t} + a_3 u - \Delta u = 0 \quad \text{в} \quad Q, \tag{1}$$

де $\Delta = \sum\limits_{i=1}^{3} \partial^2 / \partial x_i^2$ – оператор Лапласа, шляхом задання сталих $a_i, i = \overline{1,3}$. При $a_2 = a_3 = 0$ маємо однорідне хвильове рівняння, а при $a_1 = a_3 = 0$ – рівняння теплопровідності.

Аналогічно, за допомогою сталих $b_i, i = \overline{1,3}$ визначають крайову умову з загального співвідношення

$$b_1 \partial_{\boldsymbol{\nu}(x)} u + b_2 \partial_t u + b_3 u = g \quad \text{на} \quad \Sigma. \tag{2}$$

Тут $\boldsymbol{\nu}(x)$ – одиничний вектор нормалі до Γ в точці $x \in \Gamma$; а g – задана функція (функціонал). У рамках фреймворку сьогодні розглядають такі крайові умови:

– Діріхле:

$$u = g \quad \text{на} \quad \Sigma; \tag{3}$$

– Неймана:

$$\partial_{\boldsymbol{\nu}(\cdot)} u = g \quad \text{на} \quad \Sigma, \tag{4}$$

– Робіна:

$$\partial_{\boldsymbol{\nu}(\cdot)} u - b_2 \partial_t u = g \quad \text{на} \quad \Sigma. \tag{5}$$

Початково-крайову задачу, яка полягає у знаходженні узагальненого розв'язку рівняння (1), що задовольняє відповідні однорідні початкові умови та крайову умову (3), будемо називати задачею Діріхле. Якщо ж крайовою умовою є (4) чи (5), то таку задачу називатимемо задачею Неймана чи Робіна, відповідно.

При $a_1 = a_2 = 0$ еволюційне рівняння (1) вироджується в еліптичне, коли невідомі величини залежать лише від просторових координат. У цьому випадку за допомогою фрейморку можна чисельно розв'язувати крайові еліптичні задачі. За ними збережено назви, визначені крайовими умовами на поверхні Γ. У випадку задачі Робіна крайову умову задаємо так:

$$\partial_{\boldsymbol{\nu}(\cdot)} u + b_3 u = g \quad \text{на} \quad \Gamma. \tag{6}$$

Зазначимо, що коректність задач (1)-(2) досліджено у працях [4–6,8,11,14,17,18].

Чисельне розв'язування задач (1)-(2) ґрунтується на поданні їхніх розв'язків за допомогою одного з відповідних поверхневих потенціалів простого $u = \mathcal{S}\mu$ чи подвійного $u = \mathcal{D}\lambda$ шару, або їхньої комбінації

$$u = -\mathcal{S}\mu + \mathcal{D}\lambda \quad \text{в} \quad Q. \tag{7}$$

Тут через $\lambda := \gamma_0 u$ і $\mu := \gamma_1 u$ позначено дані Коші розв'язку на поверхні Γ в довільний момент часу $t \in \mathbb{R}_+$, γ_0 і γ_1 – оператори сліду і нормальної похідної, відповідно. У випадку хвильового рівняння це потенціали з запізненням, а для рівняння теплопровідності – теплові потенціали. Розв'язки еліптичних рівнянь задаємо аналогічно. У цьому випадку поверхневі потенціали не залежать від часу, за ними залишено таке саме позначення, що й у попередньому випадку.

Використання поверхневих потенціалів дає змогу звести (з врахуванням крайових умов) початково-крайову задачу до залежного від часу ГІР. Дослідження таких інтегральних рівнянь у задачах для хвильового рівняння див. [2, 3, 5, 6, 11, 13, 19]. Хоч зазначені ГІР мають меншу розмірність, ніж початково-крайові задачі, їхнє чисельне розв'язування ресурсно-затратне. У фреймворку за допомогою перетворення Лаґерра таким залежним від часу ГІР поставлено у відповідність послідовності ГІР з рекурентно залежними правими частинами, а інтегральний оператор у цих рівняннях той самий для усіх рівнянь послідовності. Цей факт зумовлює ефективне поєднання методу граничних елементів з перетворення Лаґерра. У перелічених вище публікаціях описано підходи та наведено розрахункові формули для алгоритмів дискретизації.

Результатом застосування ПЛ до залежних від часу поверхневих потенціалів є нескінченні послідовності $\mathbf{u} := (u_0, u_1, ..., u_k, ...)^\top$, які залежать від розв'язків відповідних послідовностей ГІР [2,11,13]. Якщо N перших компонентів цих розв'язків знайдено, то засобами фреймворку обчислюють апроксимації $(u_0^h, u_1^h, ..., u_N^h)^\top$ і наближений розв'язок задачі (1)-(2) рахують за формулою

$$\widetilde{u}^{N,h}(x,t) := \sum_{k=0}^{N} u_k^h(x) l_k(t), \quad (x,t) \in Q. \tag{8}$$

Тут

$$l_n(t) = \sqrt{\sigma} L_n(\sigma t) e^{-\frac{\beta}{2}t}, \; t \in \mathbb{R}_+, \; n \in \mathbb{N}_0, \tag{9}$$

де L_k, $k \in \mathbb{N}_0$, – поліноми Лаґерра, $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$, $\mathbb{N}$ – множина натуральних чисел, $\sigma > 0$ і $\beta \geq 0$ – параметри.

Зазначимо, що дискретні оператори, які відповідають ГІР для гіперболічного і параболічного випадків, відрізняються лише ядрами. Крім того, у фреймворку такі самі дискретні моделі відповідають задачам (1)-(2) у дещо іншому підході, який передбачає безпосереднє застосування ПЛ до початково-крайових задач. У цьому випадку еволюційній задачі ставиться у відповідність крайова задача для нескінченної трикутної системи еліптичних рівнянь стосовно послідовності $\mathbf{u} := (u_0, u_1, ..., u_k, ...)^\top$. У [18] для таких систем розроблено відповідні поверхневі потенціали і досліджено відповідні послідовності ГІР. Як і у попередньому випадку, після чисельного розв'язування послідовності ГІР наближений розв'язок еволюційної задачі обчислюють за формулою (8).

Оскільки еволюційні задачі та пов'язані з ними залежні від часу ГІР відображаються за допомогою перетворення Лаґерра в еліптичні задачі і відповідні ГІР, то у фреймворку є окремий інтерфейс для еліптичних задач. Він дає змогу досліджувати їхні чисельні розв'язки окремими процесами та для проведення обчислювальних експериментів з підбору параметрів числових методів у еволюційних задачах. Для цього можна вибрати реалізації МГЕ, які ґрунтуються на методі Гальоркіна, або на методі колокації. В обох методах поверхня Γ та невідомі густини μ і λ апроксимуються за допомогою кусково-сталих $\left\{\varphi_l^0\right\}_{l=1}^{M}$ чи кусково-лінійних $\left\{\varphi_l^1\right\}_{l=1}^{M^*}$ базисних

функцій. Після розв'язування відповідних систем алгебричних рівнянь засобами фреймворку можна оперувати компонентами апроксимацій густин μ_k^h і λ_k^h, де h – параметр дискретизації граничної поверхні.

Описані вище основні атрибути і підходи до розв'язування еволюційних та еліптичних задач становлять предметну область для розробки архітектури фреймворку з урахуванням спільних ознак і відмінностей кожного типу задачі.

3. Архітектура MultiMathFramework

Розглянутий підхід до чисельного розв'язування початково-крайових задач накладає вимоги до архітектури MMF. Еволюційні задачі мають низку властивостей, які можна використувати спільно, а також унікальні алгоритми, які доцільно винести в окремі структури. Тому MMF є багаторівневою системою, що передбачає наявність центрального компонента – ядра, яке утворюють програмні конструкції, загальні для всіх типів задач. Реалізацію алгоритмів та інше ПЗ, які стосуються розв'язування лише конкретних початково-крайової задач чи специфічних завдань, організовано в окремі модулі.

MMF розглядаємо як ПЗ для розв'язування не лише окреслених початково-крайових задач. Натомість це програмний каркас, який є базою для вирішення нових математичних проблем. Важливо, що цей каркас реалізований з можливістю подальшого розширення фундаментальних алгоритмів у класах модулів. Водночас він визначає відповідну зручну структуру для проектування усіх компонентів, що взаємодіють із системою.

На рис. 1 зображено архітектуру MMF. Внутрішньо вона поділена на три рівні (відділено пунктиром):

– рівень подання;
– рівень ядра;
– рівень розв'язування задачі.

Такий поділ зумовлений особливістю описаних процесів розв'язування еволюційних та еліптичних задач (1)-(2). Кожен рівень складається з окремих компонентів, що виконують чітко визначені функції. Отож, досягається принцип єдиного обов'яку (single responsibility principle) на архітектурному рівні та забезпечується слабка зв'язність компонентів системи. Наприклад, якщо потрібно додати новий алгоритм до ядра чи модуля розв'язування, то достатньо лише змінити програмний код на відповідному рівні без додаткової компіляції інших компонентів.

Рівень подання необхідний для взаємодії з користувачем і введення початкових параметрів задачі (наприклад, меж інтервалів для часової та просторової змінних x і t, параметрів σ і β, що використовуються у функціях Лаґерра (9) та ін.). Основним компонентом цього рівня є Graphical User Interface (GUI). Він відповідає за завантаження списку розв'язуваних задач, що міститься в конфігураційному файлі в форматі xml, та відображення його у вікні для користувача. Кожен елемент зі списку задач має визначений ряд атрибутів, таких як назва задачі, опис (демонструються користувачу на екрані), посилання на модуль розв'язування, конфігураційний файл конкретної задачі та форму введення даних.

Всередині компонента GUI міститься клас TaskWrapper, який ініціалізується при виборі конкретної задачі із завантаженого списку. Фактично, властивості, зчитані з конфігураційного файлу, інкапсулюються всередині об'єкта класу TaskWrapper, який звертається до рівня ядра системи для завантаження модуля розв'язування задачі. Отож, TaskWrapper виконує роль посередника, що з'єднує GUI з ядром

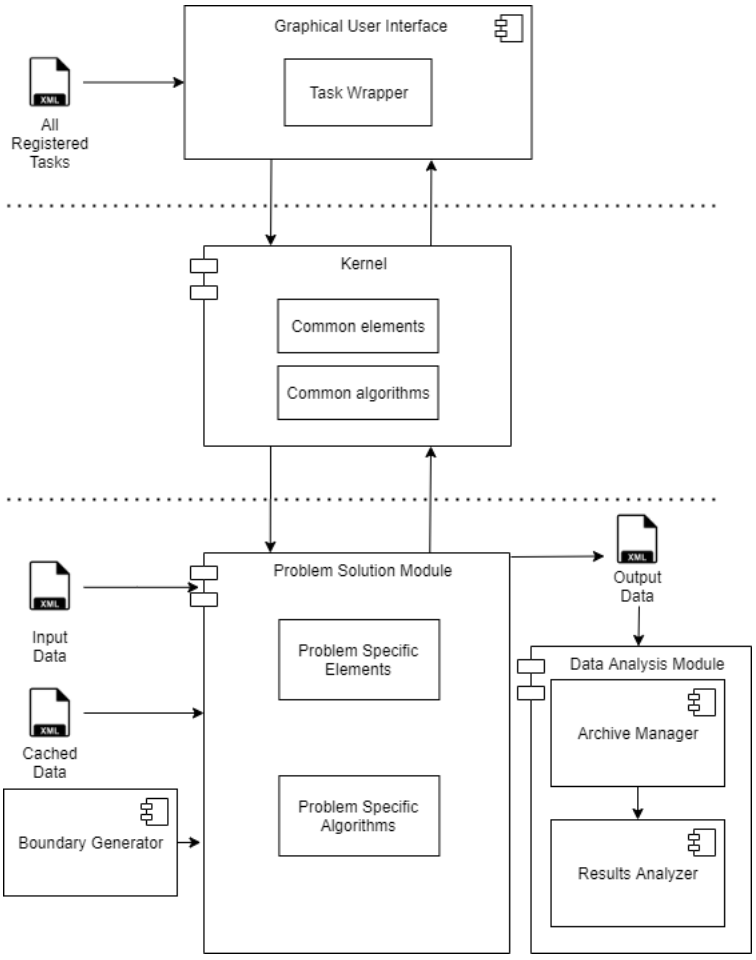

Рис. 1. Архітектура MultiMathFramework

системи, або контроллера, якщо розглядати систему MMF як видозмінену реалізацію архітектурного патерну Model-View-Controller (в такому випадку трактуватимемо ядро системи і рівень розв'язування задачі як модель).

Наступний рівень системи MMF – ядро (kernel на рис. 1). З одного боку, кожну еволюційну задачу чи еліптичну задачу (1)-(2) можна розв'язувати за допомогою окремого модуля без сполучного компонента у вигляді окремої бібліотеки елементів. Ідея ядра як посередника для розв'язування задачі виникає як наслідок спільних властивостей класу подібних задач і дає змогу зменшити кількість дублювання програмного коду для їхнього розв'язування (дотримання принципу DRY – Don't Repeat Yourself). Позаяк загальна схема розв'язування передбачає використання ПЛ та МГЕ, важливо реалізувати логіку цих методів в ядрі, щоб кожна наступна задача могла з легкістю використати їх без повторного написання коду.

Основними функціями рівня ядра як частини системи MMF є:

– завантаження та ініціалізація компонента розв'язування задачі;
– визначення інтерфейсів і базових класів, які є основою програмного каркасу

MMF;
- визначення класів для отримання значень потенціалів простого та подвійного шару після застосування ПЛ;
- визначення класів для обчислення значень ядер потенціалів $e(\cdot,\cdot)$ для різних типів задач, наприклад, однорідного хвильового рівняння чи рівняння теплопровідності;
- визначення кусково-лінійних і кусково-сталих базисних функцій для апроксимації густин потенціалів;
- визначення класів для знаходження поверхневих інтегралів, що виникають внаслідок застосування методу Гальоркіна до послідовності ГІР;
- визначення допоміжних компонентів для обчислення суми ряду, інтегрування та ін.

Отож, рівень ядра є фундаментальною частиною MMF, яку можна розглядати в аспекті бібліотеки класів і центрального сервісу, що формує програмний каркас і керує виконанням процесу розв'язування задачі.

Останнім, третім рівнем системи MMF, є рівень розв'язування задачі. Тут реалізовано специфічні алгоритми й елементи, які відповідають загальним вимогам інтерфейсів визначених в ядрі. Головною відмінністю кожної задачі здебільшого є ліва та права частина СЛАР, тому заповнення цих складових блоків для розв'язування задачі відбувається саме у цьому модулі, не порушуючи загального шаблону.

Одним із атрибутів кожної задачі є геометрія поверхні. Згенерувати розбиття поверхні можна за допомогою компонента Boundary Generator, написаного мовою Python. Вибір цієї мови програмування зумовлений використанням бібліотек, які добре підходять для візуалізації.

Після виконання алгоритму розв'язування вихідні дані серіалізуються у xml файл і зберігаються на диск, що забезпечує постійність даних. Далі основна робота виконується в модулі аналізу даних (Data Analysis Module) за допомогою компонента під назвою Archive Manager (архів). Його функція полягає у зчитуванні результатів задачі зі згенерованого файлу й експортування їх у зручному для користувача вигляді (MS Excel, Tex та ін.). Також Archive Manager передбачає можливість візуалізації даних. Важливо, що цей компонент не залежить від інших компонентів системи і може виконуватись самостійно в окремому процесі операційної системи, працюючи лише з вихідними даними задачі.

Окремим компонентом системи, що викликається безпосередньо з Archive Manager, є аналізатор результатів (Results Analyzer на рис. 1). Він дає змогу завантажити декілька різних результатів задачі разом з аналітичним розв'язком та порівняти їх між собою, зокрема, знайти абсолютні та відносні похибки, порядок збіжності та ін.

4. Побудова модуля розв'язування задачі

Модуль розв'язування задачі – це окрема сутність MMF, що відповідає за процес знаходження розв'язку. Під поняттям "модуль" розуміємо певну частину системи, яка призначена для досягнення конкретних цілей і логічно відділена від інших структур. Отже, зміни до процесу розв'язування однієї задачі, не потребують змін в інших частинах системи.

Модуль розв'язування має реалізовуватись на основі інтерфейсів і базових класів, що визначені в ядрі MMF. Якщо ця вимога не буде дотримана, то ядро не зможе на етапі виконання завантажити модуль. Це виключає можливість під'єднання

задач, що не підлягають загальній концепції системи, а також вводить певний рівень абстракції між ядром і модулем розв'язування задачі.

Розглянемо основні базові класи та інтерфейси ядра MMF, на підставі яких відбувається побудова модуля розв'язування задачі. На рис. 2 зображено класи Task, SingleSeriesTask і DoubleSeriesTask. Вони містять метод Run, який описує загальний алгоритм розв'язування початково-крайової задачі. Оскільки ми використовуємо ПЛ і МГЕ, то основні кроки алгоритму наперед визначені і в такому випадку зручно застосувати патерн проектування під назвою "шаблонний метод". Цей метод складається з послідовності таких кроків як ініціалізація даних, обчислення ряду й опрацювання результатів і всі вони перевизначаються в класі, що походить від одного з базових SingleSeriesTask або DoubleSeriesTask.

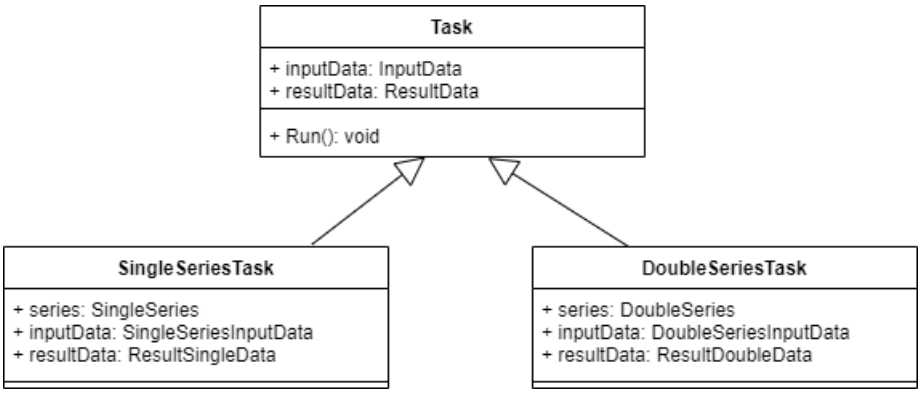

Рис. 2. Ієрархія класів Task для основного алгоритму розв'язування

У підсумку, для реалізації окремої задачі потрібно вибрати правильний базовий клас, який розташований в ядрі MMF. Головна відмінність SingleSeriesTask і DoubleSeriesTask – тип об'єкта ряду, що пов'язаний з цими класами відношенням композиції і використовується для обчислення розв'язку задачі. На рис. 3 можемо побачити, що ряд SingleSeries використовує один клас фабрики, на відміну від DoubleSeries. Це зумовлено тим, що DoubleSeries дає змогу обчислити ряд, доданками якого є добутки двох функцій. Отож, DoubleSeries зручно використовувати для еволюційних задач (1)-(2), оскільки наближений розв'язок таких задач є сумою добутків просторових компонент і відповідних функцій Лаґерра.

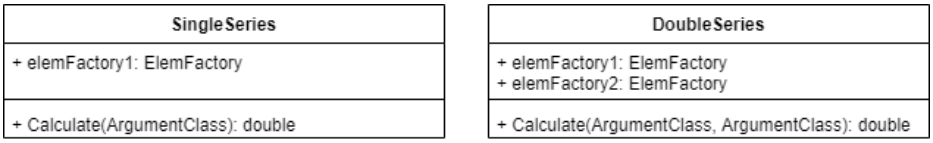

Рис. 3. Порівняння класів SingleSeries і DoubleSeries

Для знаходження суми ряду потрібно отримати об'єкти функцій, які використовуються для обчислення доданків. Класи SingleSeries і DoubleSeries містять посилання на об'єкти класу ElemFactory (рис. 4), що представляє відомий патерн проектування "фабричний метод". Цей патерн робить систему максимально незалежною від процесу створення нових об'єктів і дає змогу користувачу легко

розширювати її. Для еволюційних задач (1)-(2) у ядрі MMF реалізовано фабрику для знаходження функцій Лаґерра. Отож, користувачу, який вводить нову задачу в систему MMF, залишається правильно реалізувати фабрику для отримання просторових компонент розв'язку u_i.

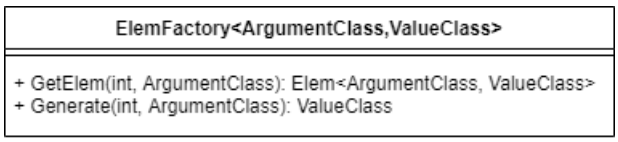

Рис. 4. Клас ElemFactory, що представляє патерн "фабричний метод"

Обчислення просторових компонет u_i у еволюційних задачах вимагає розв'язування СЛАР. Для цього потрібно створити клас фабрики UnFactory, де відбуватиметься заповнення лівої і правої частин СЛАР. Всередині фабрики достатньо використати об'єкт класу MatrixSolver, що визначений у ядрі системи (рис. 5). Цей об'єкт складається з двох стратегій для лівої і правої частин, відповідно. Патерн "стратегія" дає змогу використовувати різні варіації алгоритму всередині одного об'єкта. Це зручно, коли, наприклад, залежно від параметру, ліва частина може рахуватись звичайним способом, обчислюючи інтеграли для кожного з елементів матриці, і використовуючи, наприклад, алгоритм адаптивної перехресної апроксимації. Крім того, можна ввести додаткове кешування лівої та правої частин у файли для того, щоб не перераховувати обчислені елементи.

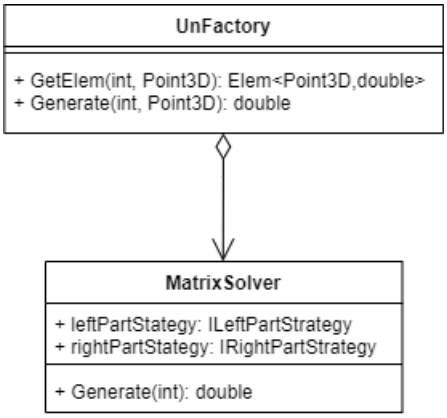

Рис. 5. Клас UnFactory та MatrixSolver для знаходження u_i

Заповнення матриць, що виникають при переході до СЛАР, відбувається за допомогою класів обчислення подвійних інтегралів. Всі вони розташовані в ядрі MMF і реалізовані як фабрики для отримання елементів у відповідному рядку та стовпці матриці. Ядро потенціалів може відрізнятись для різного типу задач (наприклад, для рівняння теплопровідності та хвильового рівняння), саме тому кожен клас, що представляє обчислення інтеграла приймає параметр фабрики для обчислення значень ядра потенціалів. Це надає системі гнучкості, адже достатньо замінити лише фабрику для генерування ядра потенціалу і підставити її у відповідний клас обчислення інтегралів.

Важливим етапом розв'язування задачі є генерація поверхні розбиття і обробка вхідних параметрів. Для цього клас, похідний від SingleSeriesTask або DoubleSeriesTask, має перевизначити крок ініціалізації у шаблонному методі Run. На цьому кроці відбувається десеріалізація поверхні розбиття з формату xml, згенерованого окремим модулем Boundary Generator, у об'єкт класу ядра BoundarySurface (рис. 6), що потім використовуватиметься при обчисленні інтегралів. Крім того, крок ініціалізації передбачає запис даних з xml файлу в об'єкт inputData (рис. 2). Ядро надає лише базові параметри задачі, такі як інтервали по часу та простору і кількість доданків в ряді Фур'є-Лаґерра. Користувач може створити похідний клас для передачі власних параметрів при ініціалізації.

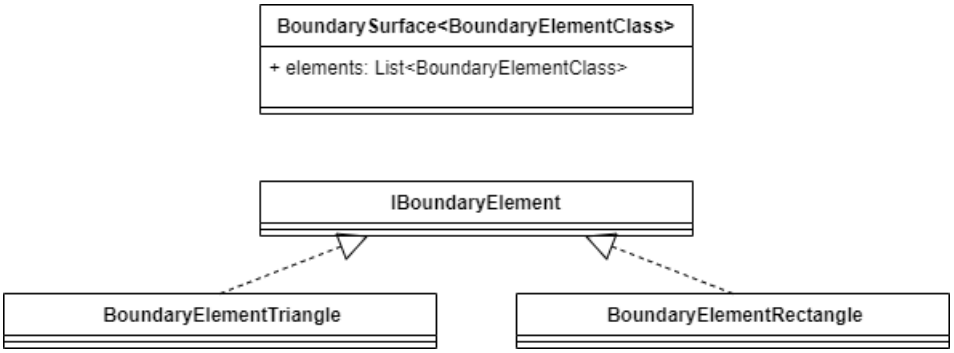

Рис. 6. Класи ядра для представлення робиття поверхні

Результат обчислення ряду, а також параметри, що використовувались для отримання розв'язку, записуються в об'єкт класів похідних від ResultData (рис. 7). Вибір класу залежить від типу розв'язуваної задачі, для еволюційних найліпше застосовувати ResultDoubleData, для еліптичних – ResultSingleData. Надалі система використовує цей об'єкт для збереження вихідних даних у різних форматах з подальшою можливістю їх обробки в архіві та аналізаторі результатів.

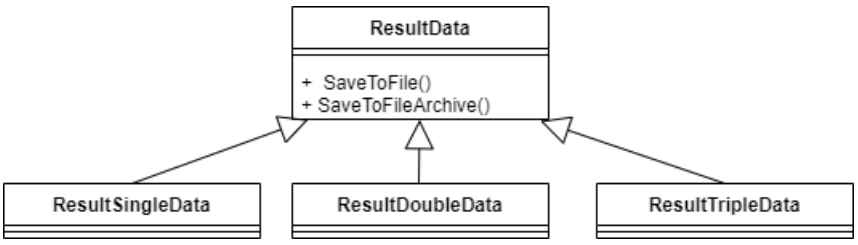

Рис. 7. Ієрархія класів для збереження результату

Отож, головна ідея побудови модуля розв'язування задачі полягає у правильному застосуванні програмного каркасу, наданого ядром MMF, та перевизначенні окремих кроків алгоритму розв'язування з використанням об'єктно-орієнтованого підходу.

5. Модуль генерування граничних елементів

Для розв'язування початково-крайових задач типу (1)-(2) треба вибрати розбиття поверхні на граничні елементи. З ними у фреймворку асоціюються алгоритми задання крайових умов вихідної задачі, а також формування множин функцій бази скінченно-вимірних просторів для дискретизації інтегральних операторів. За генерування розбиття у MMF відповідає модуль Boundary Generator. Результатом роботи цього модуля є xml-файл, який містить усю інформацію про структуру розбиття, наприклад, просторові точки, нормалі та ін. Структура xml-файлу відповідає вимогам модуля розв'язування задачі, який десеріалізує цей файл і використовує розбиття для обчислення інтегралів у процесах формування матриць дискретних операторів.

Для реалізації Boundary Generator було використано засоби мови Python. Причини такого вибору – підтримка Qt-фреймворку для реалізації користувацького інтерфейсу, легкість інтеграції з основною системою, наявність зручних бібліотек для візуалізації поверхонь і різноманітних математичних операцій, простота у розробці.

Головні функції модуля:

– генерація поверхні заданого типу (куб, сфера, конус, циліндр);
– генерація кількох поверхонь різних типів;
– перегляд згенерованих поверхонь;
– експорт поверхні в xml формат;
– експорт поверхні у вигляді зображення.

Модуль використовується для генерації розбиття чотирьох типів поверхонь – куба, сфери, циліндра та конуса. Цей список може бути розширений користувачем. Всередині модуля застосовано паттерн проектування "стратегія". Користувачу достатньо додати нову реалізацію класу SurfaceGenerator (рис. 8) і зареєструвати її у відповідному місці в коді.

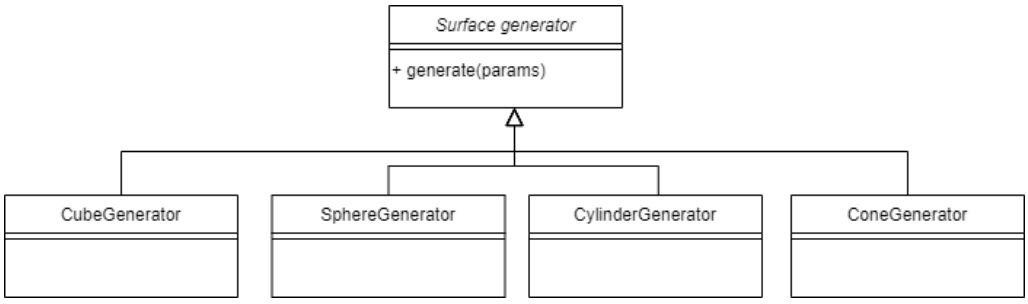

Рис. 8. Діаграма класів генераторів

Для задання параметрів поверхні використовується SurfaceLayout. Він наслідується від системного класу QGridLayout. QGridLayout відповідає за коректне розміщення різних UI-компонент у вікні програми. Для задання різних типів поверхонь модуль використовує різні реалізації SurfaceLayout. Кожна поверхня потребує свого набору параметрів для генерації розбиття. Наприклад, для куба необхідно задати координати центру, довжину ребра і кількість сегментів розбиття

ребра, а для конусу – координати центру кола, координати вершини, кількість сегментів розбиття кола і висоти.

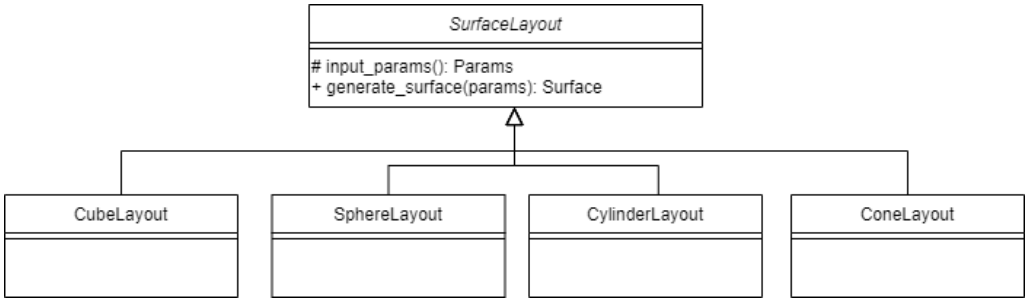

Рис. 9. Діаграма UI-компонент

Для кожної поверхні можна задати назву або цифровий ідентифікатор. Це потрібно в тому випадку, коли ми використовуємо кілька поверхонь для задання граничної умови.

6. Модуль аналізу даних

Результати виконання модуля розв'язування задачі зберігаються в об'єкті класу ResultData. Вони складаються зі значень розв'язку в точках області у заданій множині значень часової змінної, а також з набору вхідних параметрів задачі. GUI надає користувачу можливість опрацювання отриманих результатів: додавати до архіву, візуалізувати розв'язки у вигляді графіків, конвертувати у спеціальні формати для подальшої роботи з іншими програмами, наприклад, MS Excel чи LaTeX-фреймворками.

Модуль аналізу даних складається з двох компонетів (див. рис. 10): Archive Manager (архів результатів) та Results Analyzer (аналізатор результатів).

Archive Manager відповідає за завантаження збережених результатів задачі та їх обробку відповідно до вибраного типу представлення. Головні функції архіву:

- завантаження списку доступних результатів для подальшого дослідження;
- перегляд результатів у форматі 2D та 3D;
- експорт даних у MS Excel та html формати;
- експорт даних до MatLab (генерація файлів з даними та кодом, які можна виконати в цьому середощі);
- експорт даних до системи Mathematica;
- генерування файлів для отримання анімації розв'язку в часі з подальшим виконанням у MatLab;
- взаємодія з аналізатором результатів.

Така функціональність свідчить про широкий спектр можливостей для користувача у роботі з вихідними даними. Архітектура архіву спроектована так, що не потрібно перезавантажувати чи зупиняти модуль роз'язування задачі у випадку отримання нових даних. Достатньо лише зберегти їх у каталозі, що відповідає архіву, та оновити список доступних результатів. Представлення результатів задач є основною властивістю архіву. Зокрема, підтримуються MS Excel та xml-формати (для простого аналізу), а також експортування до програмних систем MatLab і

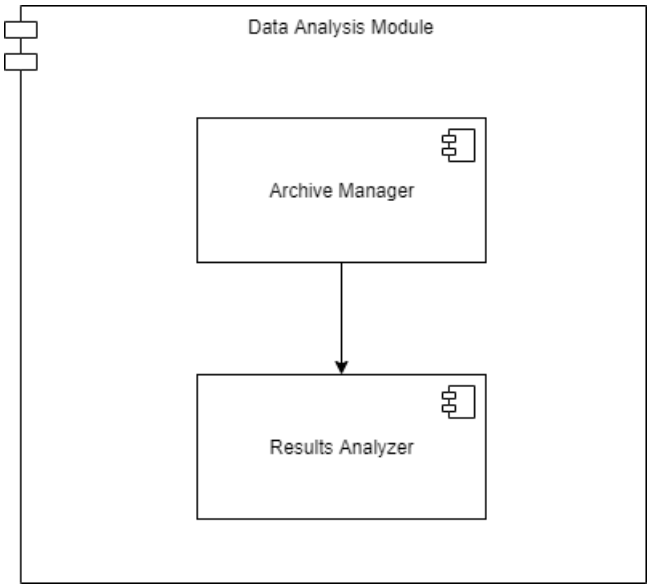

Рис. 10. Модуль аналізу даних

Matematica. Архів не обмежує користувача у виборі формату і якісно спрощує процес представлення даних.

Другим важливим компонентом модуля є Results Analyzer (аналізатор даних). Його головне завдання полягає у дослідженні апостеріорних похибок чисельного розв'язку. Для оцінки точності отриманих просторових коефіцієнтів u_k^h для еволюційних задач використовують апостеріорні похибки

$$\delta_k^h := ||u_k^h - u_k||_{L^2(\Omega_{a,b})}, \;\; \epsilon_k^h := \delta_k^h / ||u_k||_{L^2(\Omega_{a,b})} * 100\ \%, \;\; k \in \mathbb{N}_0. \tag{10}$$

де u_k – відомий аналітичний розв'язок. Тут під $\Omega_{a,b}$ розуміємо деякий відрізок з кінцями в точках a і b, вздовж якого розміщуються точки спостереження. Крім того, важливою є величина передбачуваного порядку збіжності (estimated order of convergence)

$$eoc_k := \frac{\ln(\delta_k^{h_{j-1}} / \delta_k^{h_j})}{\ln(h_{j-1}/h_j)}, \;\; k \in \mathbb{N}_0, \tag{11}$$

де h_{j-1} і h_j – параметри двох послідовних розбиттів граничної поверхні на граничні елементи.

Для нестаціонарних чисельних розв'язків можна знайти такі апостеріорні похибки:

$$\widetilde{\delta}^{N,h} := ||\widetilde{u}^{N,h} - \widetilde{u}^N||_{L^2_\sigma(\mathbb{R}_+;L^2(\Omega_{a,b}))}, \;\; \widetilde{\epsilon}^{N,h} := \widetilde{\delta}^{N,h} / ||\widetilde{u}^N||_{L^2_\sigma(\mathbb{R}_+;L^2(\Omega_{a,b}))} * 100\ \%.$$

Тут під $\Omega_{a,b}$ розуміємо відрізок (без кінцевих точок) у просторі поза кубом. Для послідовних значень параметра h величина

$$\widetilde{eoc}^{N,h} := \frac{\ln(\widetilde{\delta}^{N,h_{j-1}} / \widetilde{\delta}^{N,h_j})}{\ln(h_{j-1}/h_j)}.$$

Графічний інтерфейс цього компонента дає змогу завантажити один або кілька xml-файлів з розв'язками (наприклад, для різних розбиттів поверхні) і порівняти їх з відомим аналітичним розв'язком, який попередньо має бути обчисленим у потрібній множині точок. Для коректного порівняння результатів треба дотримуватися лише однакових за своєю внутрішньою структурою форматів даних. Це досягається використанням об'єктів класу ResultData і механізмів серіалізації у платформі .NET.

Результатом обробки даних у аналізаторі є таблиці з абсолютною та відносною похибками, а також дані стосовно порядків збіжності на послідовності множин зі зростаючою кількістю граничних елементів на поверхні Γ. Крім того, аналізатор візуалізує графіки похибок для зручності порівняння отриманих чисельних розв'язків з відомим аналітичним. Така можливість зручна також при тестуванні нового ПЗ.

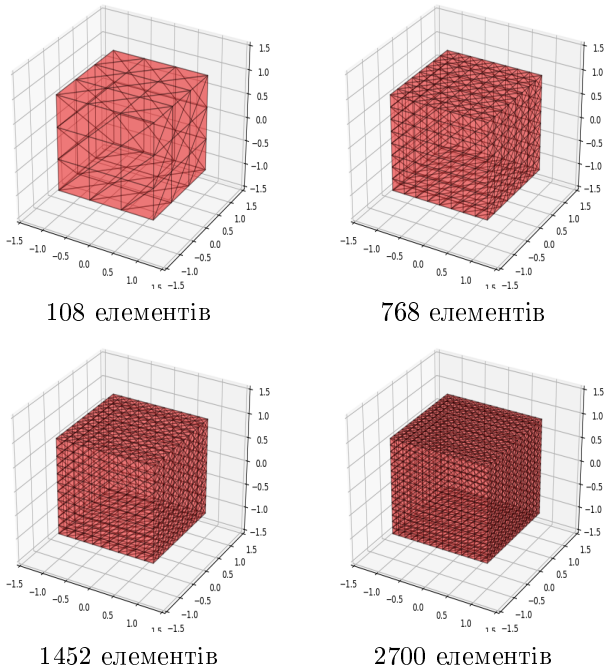

Рис. 11. Згенеровані розбиття поверхні куба з різною кількістю граничних елементів

7. Приклади використання MultiMathFramework

На цей момент у рамках MMF реалізовано чисельне розв'язування таких еволюційних задач для однорідного хвильового рівняння:

1) задача Діріхле з використанням потенціалу простого шару [11];
2) задача Діріхле з використанням формули Кірхгофа [10];
3) задача Неймана з використанням потенціалу подвійного шару [13];
4) задача Неймана з використанням потенціалу простого шару [3];
5) задача Неймана з використанням формули Кірхгофа [3];
6) задача Робіна з використанням формули Кірхгофа [2];
7) задача Коші [7];

8) задача Діріхле у півпросторі з включеннями [10];

а також задач Діріхле, Неймана і Робіна для еліптичних рівнянь та систем [15, 17].

Продемонструємо на практиці застосування MMF для чисельного розв'яззування початково-крайових задач з використанням ПЛ та МГЕ. Розглянемо типові приклади.

Таблиця 1

Аналіз збіжності для u_0^h, u_{10}^h і $\widetilde{u}^{N,h}$ де $\alpha = 3.5$, $\beta = 0.5$, $N = 10$ при збільшенні кількості граничних елементів

M	u_0^h			u_{10}^h			$\widetilde{u}^{N,h}$		
	δ_0^h	eoc_0	ϵ_0^h	$\delta_1 0^h$	eoc_{10}	$\epsilon_1 0^h$	$\widetilde{\delta}^{N,h}$	$\widetilde{eoc}^{N,h}$	$\widetilde{\epsilon}^{N,h}$
108	$4.37 \cdot 10^{-4}$		4.65	$6.73 \cdot 10^{-4}$		4.38	$9.14 \cdot 10^{-3}$		1.77
192	$1.26 \cdot 10^{-4}$	4.32	1.34	$1.98 \cdot 10^{-4}$	4.25	1.29	$4.09 \cdot 10^{-3}$	2.79	0.79
300	$1.07 \cdot 10^{-4}$	0.73	1.14	$1.29 \cdot 10^{-4}$	1.93	0.84	$2.68 \cdot 10^{-3}$	1.89	0.52
432	$7.12 \cdot 10^{-5}$	2.24	0.76	$9.70 \cdot 10^{-5}$	1.55	0.63	$2.02 \cdot 10^{-3}$	1.56	0.39
588	$5.57 \cdot 10^{-5}$	1.60	0.59	$8.00 \cdot 10^{-5}$	1.25	0.52	$1.62 \cdot 10^{-3}$	1.40	0.31
768	$4.67 \cdot 10^{-5}$	1.32	0.50	$6.77 \cdot 10^{-5}$	1.26	0.44	$1.36 \cdot 10^{-3}$	1.33	0.26
972	$3.88 \cdot 10^{-5}$	1.58	0.41	$5.86 \cdot 10^{-5}$	1.22	0.38	$1.17 \cdot 10^{-3}$	1.29	0.23

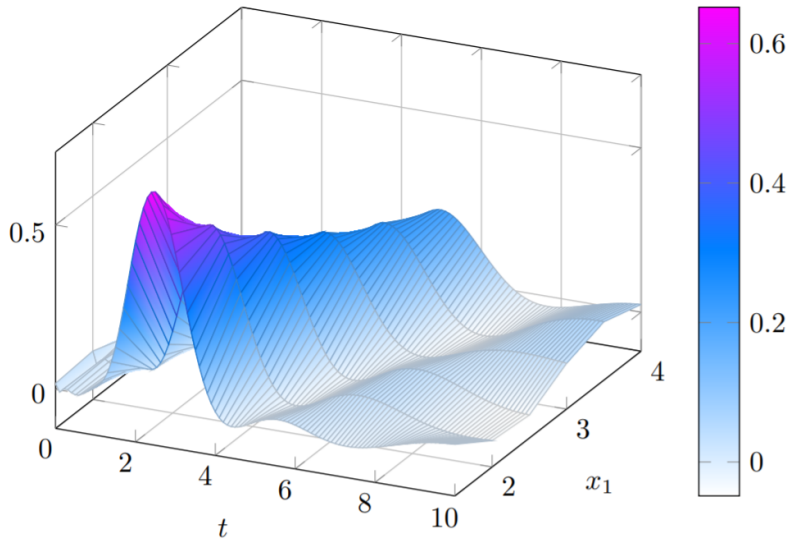

Рис. 12. Зміна чисельного розв'язку $\widetilde{u}^{N,h}(x,t)$ в часі для точок $(x_1, 0, 0)$, де $x_1 \in [1.5, 4]$

Приклад 1. *Згенерувати та візуалізувати розбиття поверхні куба з центром в точці* $x^* = (0, 0, 0)$ *та довжиною сторони* $l = 2$ *на* $M = 108, 768, 1452, 2700$ *елементів для розв'язування задач, використовуючи компонент Boundary Generator системи MMF.*

Для генерації поверхонь оберемо тип "куб". Визначимо такі параметри: координати центру – (0, 0, 0); довжина ребра – 2; кількість сегментів розбиття ребра –

3 (для 108-елементного розбиття поверхні); 8 (для 768-елементного розбиття); 11 (для 1452-елементного розбиття); 15 (для 2700-елементного розбиття). Згенеруємо розбиття з обраними параметрами, що можна побачити на рис. 11. Після цього експортуємо розбиття в xml-файл для подальшого використання в системі MMF.

Для задання модельної еволюційної задачі розглянемо "сферичний імпульс"

$$w(x,t) := \frac{a}{|x|} w^*(t - |x| + a),\ (x,t) \in \mathbb{R}^3 \setminus \{0\} \times \mathbb{R}_0, \tag{12}$$

де функція w^* є кубічним B-сплайном [13].

Приклад 2. *Знайти чисельний розв'язок $\widetilde{u}^{N,h}(x,t)$ задачі Робіна при $N = 10$ для значення параметра імпедансу $b_2 = 1$, коли функція g в крайовій умові (5) задана за формулою $g = -\partial_{\boldsymbol{\nu}(\cdot)} w + b_2 \partial_t w$, розглядаючи розбиття поверхні куба на граничні елементи для послідовності значень M, а також дослідити апостеріорну похибку і швидкість збіжності u_0^h, u_{10}^h та $\widetilde{u}^{N,h}(x,t)$.*

Використаємо систему MMF для знаходження коефіцієнтів Фур'є-Лаґерра $u_0^h(x)$, u_{10}^h та чисельного розв'язку $\widetilde{u}^{N,h}(x,t)$ задачі Робіна. Для реалізації було розроблено модуль розв'язування для задачі Робіна шляхом додавання нової фабрики для обчислення просторових компонентів u_i, відповідні розрахункові формули подано у [2].

Обчислені відповідно до завдань прикладу 2 результати бути завантажені до архіву та візуалізовані графічно у відповідних елементах інтерфейсу користувача. Також за допомогою аналізатора результатів дані було відформатовано для безпосереднього використання їх засобами TeX для побудови табл. 1 та графіку зміни розв'язку в часі на рис. 12.

Отримані результати модельної задачі демонструють зменшення абсолюної і відносної похибок як окремих коефіцієнтів Фур'є-Лаґерра і самого чисельного розв'язку задачі Робіна при збільшенні кількості елементів розбиття M. Цей факт свідчить про ефективність поєднання ПЛ і МГЕ, а також характеризує можливості MMF.

8. Висновки

Розроблена архітектура MultiMathFramework та його програмна реалізація робить цей інструмент ефективним для знаходження чисельних розв'язків початково-крайових задач в 3D областях відповідно до підходу, який грунтується на комбінації ПЛ і МГЕ. Використані технологічні рішення забезпечують його високу швидкодію, гнучкість і зручність у використанні, а також можливість досягнення високої точності результатів в обчислювальних експериметах.

Список використаної літератури

1. *Галазюк В. А.* Метод поліномів Чебишева – Лагера в змішаній задачі для лінійного диференціального рівняння другого порядку з постійними коефіцієнтами / В. А. Галазюк // Доп. АН УРСР. – 1981. – № 1. – С. 3–7.
2. *Глова А. Р.* Чисельне розв'язування початково-крайових задач для хвильового рівняння із застосуванням формули Кірхгофа та перетворення Лаґерра / А. Р. Глова, С. В. Літинський, Ю. А. Музичук, А. О. Музичук // Вісник ЛНУ. Серія прикл. матем. та інформ. – 2019. – Вип. 27. – С. 18–33.

3. *Глова А. Р.* Про застосування потенціалів із запізненням в початково-крайових задачах для хвильового рівняння / А. Р. Глова, С. В. Літинський, Ю. А. Музичук // Матеріали XXIV Всеукраїнської наукової конференції "Сучасні проблеми прикладної математики та інформатики. (APAMCS-2018)" 26-28 вересня 2018, Львів. – Львів: Вид-во Тараса Сороки, 2018. – С. 39–42.
4. *Музичук Ю.* Про чисельне розв'язування внутрішніх крайових задач для нескінченних систем еліптичних рівнянь / Ю. Музичук // Вісник ЛНУ. Серія прикл. матем. та інформ. – 2013. – Вип. 20. – С. 49–56.
5. *Bamberger A.* Formulation variationnelle espace-temps pour le calcul par potentiel retarde de la diffraction d'une onde acoustique (I) / A. Bamberger, T. Ha Duong // Math. Methods Appl. Sci. – 1986. – No 8 (3). – P. 405–435.
6. *Bamberger A.* Formulation variationnelle pour le calcul de la diffraction d'une onde acoustique par une surface rigide / A. Bamberger, T. Ha Duong // Math. Methods Appl. Sci. – 1986. – No 8. – P. 598–608.
7. *Chapko R.* Wave propagation from lateral Cauchy data using a boundary element method / R. Chapko, B. Tomas Johansson, Yuriy Muzychuk, Andriy Hlova // Wave Motion. – November, 2019. – Vol. 91. – Article 102385. – https:// doi.org /10.1016 /j.wavemoti. 2019.102385.
8. *Costabel M.* Boundary integral operators for the heat equation / M. Costabel // Integral Equations Operator Theory. – 1990. – Vol. 13 (4). – P. 498–552.
9. *Dautray R.* Mathematical analysis and numerical methods for science and technology / R. Dautray, J. L. Lions // Evolution problems I. – Berlin: Springer-Verlag, 1992. – Vol. 5.
10. *Hlova A. R.* Coupling of Laguerre Transform and Fast BEM for solving Dirichlet initial-boundary value problems for the wave equation / A. R. Hlova, S. V. Litynskyy, Yu. A. Muzychuk, A. O. Muzychuk // J. of Computational and Appl. Math. – 2018. – No 2 (128). – P. 42–60.
11. *Litynskyy S.* Solving of the initial-boundary value problems for the wave equation by the use of retarded potential and the Laguerre transform / S. Litynskyy, A. Muzychuk // Matematychni Studii. – Vol. 44, No 2. – P. 185–203.
12. *Litynskyy S.* On the generalized solution of the initial-boundary value problems with Neumann condition for the wave equation by the use of retarded double layer potential and the Laguerre transform / S. Litynskyy, A. Muzychuk // J. of Computational and Appl. Math. – 2016. – No 2 (122). – P. 21–39.
13. *Litynskyy S.* On the numerical solution of the initial boundary value problem with Neumann condition for the wave equation by the use of the laguerre transform and boundary elements method / S. Litynskyy, Y. Muzychuk, A. Muzychuk // Acta Mechanica et Automatica, The J. of Bialystok Techn. Univ. – 2016. – Vol. 10, No 4. – P. 285–290.
14. *Litynskyy S.* Combination of the Laguerre Transform with BEM for the solution of integral equations with retarded kernel / S. Litynskyy, Y. Muzychuk, A. Muzychuk // J. of Math.Science. – Vol. 236, № 1. – DOI 10.1007/s10958-018-4100-x.
15. *Litynskyy S.* On weak solution of a boundary value problem for an infinite triangular system of elliptic equations with Robin boundary conditions / S. Litynskyy, Yu. Muzychuk // Proceedings of XV International Seminar/Workshop on Direct and Inverse problems of Electromagnetic and Acoustic Wave Theory (DIPED-2010). – Tbilisi, 2010. – P. 192–195.
16. MatLab. – https://www.mathworks.com/products/matlab.html
17. *Muzychuk Yu.* On the boundary integral equations method for Robin boundary value problems received as a result of the Laguerre transform of mixed problems for evolution equations / Yu. Muzychuk // Odesa National University Herald. Math. and Mech. – 2013. – Vol. 18, Num. 4 (20). – P. 38–49.
18. *Muzychuk Yu. A.* On variational formulations of inner boundary value problems for infinite systems of elliptic equations of special kind / Yu. A. Muzychuk, R. S. Chapko // Matematychni Studii. – 2012. – Vol. 38. – P. 12–34.

19. *Sayas F. J.* Retarded potentials and time domain boundary integral equations: a road map / F. J. Sayas. – Springer Intern. Publ., 2016. – 241 p.
20. WolframAlpha Computational Intelligence. – https://www.wolframalpha.com

Стаття: надійшла до редколегії 24.09.2020
доопрацьована 02.10.2020
прийнята до друку 07.10.2020

MULTIMATHFRAMEWORK SYSTEM FOR SOLVING INITIAL-BOUNDARY VALUE PROBLEMS USING BOUNDARY ELEMENT METHOD AND LAGUERRE TRANSFORM

A. Hlova[1], O. Kir[2]

Ivan Franko National University of Lviv,
Universytetska str., 1, Lviv, 79000,
e-mail: [1] *a.hlova@gmail.com*. [2] *oleggio.kir@gmail.com*

General architecture and application of the MultiMathFramework system (MMF) for solving initial-boundary value problems are considered. The structure of the system corresponds to basic principles of software dedicated for solving mathematical problems. In contrast to existing products, MMF focuses on specific class of the tasks and at the same time provides framework for further extension of system capabilities using basic principles of object-oriented programming. In particular, the main structural units are graphical user interface, kernel, solver module, boundary generator and data analysis module. Graphical user interface is the first point of user interaction with the system. It is responsible for loading and demonstrating list of the tasks with ability to fill input data of the problem. System kernel defines general algorithms, acting like extensible base class library and mediator that loads, initializes solver module and manages its communication with other components. Solver module implements a set of technical decisions, in particular design patterns, and instructions for solving mathematical problem. It also uses separate module for surface generation and visualization. Every task is considered as a separate component at runtime that is dynamically loaded into the process based on data from configuration file. The structure of such component must correspond to interface defined by system kernel. The results of solver module are stored as output data in archive. Functionality of archive and analysis module is designed for visualization, comparison and usage in further experiments. Efficiency of application of this architecture is proved by numerical results of evolutionary and elliptic problems with high accuracy and proved estimated order of convergence.

Key words: initial-boundary value problem, boundary element method, Laguerre transform, surface potentials, algorithm, program module, component, interface, software architecture, class library, base class, design pattern, programming principle, object-oriented programming, inheritance, composition, caching.