НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Факультет робототехніки та приладобудування
Кафедра комп'ютерно-інтегрованих оптичних танавігаційних систем

ДО ЗАХИСТУ ДОПУЩЕНО
Завідувач кафедри
_____ Надія БУРАУ
«___»_____ 2026 р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Комп'ютерно-інтегровані системи та технології в приладобудуванні»
спеціальності 174
«Автоматизація, комп'ютерно-інтегровані технології та робототехніка»
на тему: «Додаток для автоматизації математичних обчислень»

Виконав:
студент ІІІ курсу, групи ПГ-п31
Довбиш Д. Ю. _____

Керівник:
Асистент, к.т.н. Рупіч С. С. _____

Рецензент:
Доцент, к.т.н, Безугла Н. В. _____

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

Київ – 2026

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Факультет робототехніки та приладобудування

Кафедра Комп'ютерно-інтегрованих оптичних та навігаційних систем
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
(код і назва)

Освітньо-професійна програма – «Комп'ютерно-інтегровані системи та технології в приладобудуванні»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Н.І. Бурау
(підпис) (ініціали, прізвище)
«___»_____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту
Довбишу Дмитру Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема дипломної роботи: Додаток для автоматизації математичних обчислень

Науковий керівник: Рупіч Сергій Сергійович, к.т.н. _____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» _____ 20__ р. № _____

2. Термін подання студентом роботи «02» червня 2026 р.

3. Вихідні дані до роботи: реалізувати додаток для виконання математичних обчислень; розробити функціонал для введення формул, виведення результату обчислень, побудови графіка функції; розробити архітектуру програмної реалізації; розроблений додаток має бути забезпечений можливістю встановлення на носій користувача.

4. Питання, що мають бути розроблені:

1. Зробити огляд та аналіз актуальності проблеми.

2. Провести огляд існуючих рішень та підходів для виконання математичних обчислень.

3. Визначити вимоги до застосунку.

4. Розробити архітектуру та структуру програмної реалізації, основні модулі та підсистеми.

5. Описати функціональність та особливості системи.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): таблиці, графіки, рисунки тощо.

6. Дата видачі завдання 01.03.2026

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Огляд стану проблеми	10.05.2026	
2.	Огляд існуючих рішень та підходів	15.05.2026	
3.	Визначення вимог. Побудова архітектури та структури проекту	18.05.2026	
4.	Розробка програмного забезпечення	28.05.2026	
5	Опис функціональності та особливостей системи	30.06.2026	
6.	Оформлення пояснювальної записки. Підготовка до захисту	02.06.2026	

Студент _____ Дмитро ДОВБИШ

Керівник дипломної роботи _____ Сергій РУПІЧ

АНОТАЦІЯ

Кваліфікаційна бакалаврська робота присвячена проєктуванню та розробці автономного мобільного додатка для виконання комплексних математичних обчислень і візуального геометричного моделювання.

У роботі проведено аналіз існуючих систем комп'ютерної алгебри, на основі якого обґрунтовано доцільність створення портативного математичного інструментарію. Значну увагу приділено проєктуванню програмної архітектури: розглядається застосування модульного підходу для ефективної ізоляції ресурсомістких аналітичних процесів від графічної оболонки.

Практична частина роботи описує процес розробки додатка в середовищі ігрового рушія Unity з використанням мови C#. У ній детально висвітлено алгоритми роботи обчислювального ядра на базі інтегрованої системи AngouriMath та механізми обробки вводу багатошарових математичних виразів через бібліотеку MathLive. Розглядається логіка реалізації ключових підсистем додатка: алгебраїчного спрощення, аналітичного знаходження змінних, параметричного розрахунку величин, а також генерації інтерактивних двовимірних та тривимірних графіків і нерівностей.

Результатом роботи є повністю функціонуючий програмний продукт, архітектура та функціонал якого всебічно протестовані й описані у відповідних розділах.

Робота складається з 94 сторінок та 51 рисунка.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

Кросплатформність – властивість програмного забезпечення повноцінно функціонувати на різних апаратних платформах та операційних системах без необхідності суттєвого переписування базового вихідного коду.

Плагін (Plug-in) – незалежний програмний модуль, що інтегрується в основну систему чи середовище розробки для розширення їхніх базових функціональних можливостей.

Рендеринг – процес генерації зображення на основі створеної математичної, програмної або геометричної моделі за допомогою обчислювальних потужностей пристрою.

Шейдер (Shader) – спеціалізована комп'ютерна програма, що виконується графічним процесором і визначає фінальні візуальні параметри пікселів або вершин: колір, освітлення, тіні, прозорість та накладання матеріалів під час рендерингу 2D та 3D об'єктів.

Mesh (Полігональна сітка) – сукупність вершин, ребер та площин, що формують геометричний каркас тривимірного об'єкта.

ВСТУП

Математичні обчислення всю історію займали велике місце у життєдіяльності суспільства, дозволяючи реалізовувати будь які потреби та задачі в різноманітних сферах – від сільського господарства та навігації до машинного виробництва та будівництва, архітектури. Їх важливість важко переоцінити, їхня цінність у створення сучасного світу є основоположною. Неможливо відділити математичні обчислення від процесу автоматизації – настільки ж важливої та широкої діяльності, реалізація якої є необхідною для існування будь якого колективу людей. Вони є абсолютно нероздільними і розглядати одне без іншого не має сенсу.

Історичний розвиток автоматизації повсякденного життя та математичних обчислень відбувався паралельно, взаємно стимулюючи та доповнюючи одна одну [1]. Зростання потреб людства у точних і масових обчисленнях для потреб навігації, господарства, виробництва та інженерії спонукало перехід від ручної праці до використання спеціалізованих інструментів. Перші кроки до інтеграції математики та автоматизації були створені ще в епоху створення ранніх механічних обчислювальних машин, таких як «Калькулятор Паскаля» [2] чи ступінчастий обчислювач Готфріда Лейбніца [3]. Ці винахідники продемонстрували революційну ідею: абстрактні математичні операції можна змоделювати за допомогою законів кінематики. Перші механізми були відносно складними системами зубчастих колес, циліндрів та важелів, що своєю взаємодією виконували прості математичні операції. Наприклад, десяткова система числення була реалізована через колеса з десятьма зубцями, де повний оберт одного колеса механічно викликав зсув сусіднього на одну позицію, тим самим автоматизуючи процес перенесення десяткового розряду. Подальшим розвитком механічної автоматизації обчислень стала «Аналітична машина» Чарльза Беббіджа [4]. Вона вперше в історії передбачала наявність пам'яті для зберігання проміжних результатів, арифметичного пристрою та системи введення інструкцій за допомогою перфокарт. Ця машина стала однією з перших, що реалізовувала автоматизацію не лише поодиноких арифметичних дії, а й комплексних послідовних обчислень.

Якісний стрибок відбувся в середині двадцятого століття завдяки переходу від кінематичних механізмів до електронних елементів. Поява перших електронно-обчислювальних машин, а згодом і стрімкий розвиток мікропроцесорної техніки назавжди пов'язали поняття математичних обчислень і автоматизації. Апаратна реалізація логічних операцій на базі напівпровідників дозволила збільшити швидкість обробки інформації на багато порядків. Складні математичні моделі, багатовимірні матричні обчислення та системи диференціальних рівнянь, розв'язання яких раніше вимагало колосальних людських ресурсів і тривалого часу, почали виконуватися комп’ютерним шляхом за секунди. Таке стрімке зростання обчислювальної потужності дало змогу вивести автоматизацію за межі простих циклічних дій. Машинний розрахунок складних алгоритмів став основою для автоматичного керування високотехнологічними виробничими лініями, оптичними системами та, з нещодавно, нейромережевими моделями різного призначення.

На поточному етапі розвитку математика та автоматизація вийшли за межі суто фізичної праці і перейшла у складну інтелектуальну діяльність. Сучасна автоматизація охоплює всі рівні виробничих та інформаційних процесів: від первинного збору даних до багатокритеріальної оптимізації роботи складних технологічних процесів. Такі багаторівневі системи цілком і повністю базуються на фундаментальному математичному аналізі, теорії автоматичного керування та статистичних методах.

Особливу роль у розширенні функціональних меж сучасної автоматизації відіграють перспективні алгоритмічні рішення, серед яких значний потенціал демонструють штучні нейронні мережі. Їхнє використання як спеціалізованого математичного інструменту дозволяє автоматизувати складні процеси, які важко піддаються традиційній логіці програмування. Вони принципово змінюють можливості існуючих систем, даючи можливість реалізувати такий складний принцип, як прийняття рішень. Потенційно, майбутні системи стануть надстійкими, через можливість реакції технологій на будь що, продумане заздалегідь та абсолютно нове.

Тривалий шлях розвитку від найпростіших механічних лічильників до сучасних багаторівневих систем керування яскраво демонструє глибинний зв'язок між цими

науковими дисциплінами. Автоматизація в її сучасному вигляді перестала бути просто супутником математики, перетворившись на її неминуче технологічне продовження. Кожна нова математична концепція знаходить своє практичне втілення у цифрових пристроях, що забезпечує безперервний розвиток систем автоматизації у всіх сферах людської діяльності.

РОЗДІЛ 1

ОГЛЯД СТАНУ ПРОБЛЕМИ

1.1. Проблеми складності сучасних математичних обчислень

Еволюція засобів для автоматизації математичних обчислень виникла не з проста, вона відображає процес постійного ускладнення необхідних до вирішення задач. На ранніх етапах розвитку математики домінувала дискретна арифметика, де операції мали лінійний характер і передбачали отримання результату за скінченну, прийнятну для людини кількість кроків. Такі розрахунки, хоч і вимагали точності, залишалися в межах реальності відтворення особисто математиком, оскільки структура виразів не містила глибоких рівнів вкладеності та неосяжних розрахунків.

З розвитком алгебри та математичного аналізу характер задач якісно змінився. Замість прямого пошуку числового значення виникла необхідність роботи з функціональними залежностями та багаторівневими системами. Це призвело до появи математичних конструкцій, що містять ірраціональні вирази, логарифмічні функції, багатоповерхові дроби та інші фрагменти, обчислення яких є особливо складною та довготривалою для людини задачею, якщо не є одиничною. Формалізація таких виразів вимагає від дослідника одночасного утримання в пам’яті як загального виду формули, так і особливо складних операцій, що значно підвищує ймовірність помилки при ручній обробці.

Особливим етапом став перехід до чисельних методів та ітераційних процесів. Більшість сучасних прикладних задач – від апроксимації кривих до розв’язання диференціальних рівнянь не мають елегантного розв’язку у закритому вигляді, не існує універсальної формули. Натомість вони базуються на багаторазовому повторенні одних і тих самих обчислювальних циклів для наближення до істинного значення. Кожна така ітерація є ланкою в ланцюгу, де результат попереднього кроку стає вхідними

даними для наступного. При виконанні ручного обчислення цей процес створює накопичувальну помилку: найменша неточність на першому етапі ітерації призводить до недопустимого відхилення отриманого результату від дійсного.

Яскравою ілюстрацією безнадійності ручного обчислення складних математичних задач є класичні задачі чисельного аналізу, для яких аналітичного розв'язку не існує принципово. Згідно з фундаментальною теоремою Абеля-Руффіні, алгебраїчні рівняння п'ятого та вищих степенів у загальному вигляді неможливо розв'язати в радикалах. Єдиним шляхом знаходження їхніх коренів є використання ітераційних обчислювальних алгоритмів [5], таких як метод Ньютона (метод дотичних). Цей алгоритм вимагає на кожному кроці обчислювати значення функції та її похідної для знаходження наступного, точнішого наближення. Для досягнення прийнятної точності необхідно виконати серію таких вкладених розрахунків із багатозначними дробовими числами. Людина, виконуючи цей процес вручну, з високою ймовірністю допустить помилку, а через залежність наступної операції від попередньої, унеможливить отримати точний розв’язок. Верифікація ж кожної операції іншими людьми кратно збільшує необхідну кількість роботи на отримання результату.

Іншим показовим прикладом є алгоритми розв'язання звичайних диференціальних рівнянь, які лягають в основу моделювання багатьох фізичних та технологічних процесів. Оскільки знайти точну первісну для систем із нелінійними компонентами аналітично неможливо, застосовуються алгоритми, наприклад метод Рунге-Кутта четвертого порядку [6]. Щоб просунути розв'язок лише на один крок уперед, цей алгоритм вимагає послідовного розрахунку чотирьох проміжних коефіцієнтів, кожен з яких спирається на результат попереднього. Побудова моделі навіть на короткому інтервалі вимагає десятків або сотень таких циклів. Єдина механічна помилка у знаку чи розряді числа на початкових етапах інтегрування призводить до лавиноподібного накопичення похибки, перетворюючи фінальний розрахунок на беззмістовні дані.

Аналогічний бар'єр спостерігається і при роботі з багатовимірними структурами даних. Стандартний алгоритм Гауса для розв'язання систем лінійних алгебраїч-

них рівнянь має кубічну обчислювальну складність. Це означає, що ручний аналіз системи всього з десяти параметрів вимагає безпомилкового виконання кількох сотень взаємопов'язаних операцій додавання та множення дробів.

Таким чином, математичні алгоритми досягли такого рівня детермінованої складності, за якого їхнє відтворення без засобів автоматизації стає не просто трудомістким, а принципово неможливим. Сучасні обчислення перетворилися з інструменту статичного опису на динамічну послідовність вкладених процедур, що і зумовлює необхідність створення систем автоматизованого розпізнавання та розв'язання.

1.2. Розвиток арифметико-логічних пристроїв для моделювання та розрахунків

Описані вище причини спонукали розвиток комп’ютерних технологій. Без них усі дослідження математичних розв’язань стали б філософськими розвагами, а не прикладними рішеннями вже після 50-100 людино-годин вирішення однієї задачі. Історичний розвиток засобів математичних обчислень зазнав якісного стрибка з появою електронно-обчислювальних машин (ЕОМ), що дозволило відмовитися від кінематичних механізмів на користь керування потоками електронів. Цей перехід створив нову галузь цифрових обчислень, яка у своєму розвитку пройшла кілька ключових технологічних етапів, кожен з яких розширював межі математичного моделювання і значно розвивається по сьогоднішній день.

Першим значним проривом стали розробки на базі електронних вакуумних ламп у 1940-х роках (наприклад, система ENIAC) [7]. Заміна електромеханічних реле на електронні перемикачі дозволила збільшити швидкість виконання арифметичних операцій на порядки. Це вперше зробило практично можливим застосування чисельних методів для складних задач (наприклад, розрахунку балістичних таблиць чи розв'язання систем лінійних рівнянь), хоча такі системи залишалися надзвичайно громіздкими, енергозатратними та складними у перепрограмуванні.

Винахід транзистора в середині XX століття змінив елементну базу обчислювальних систем, замінивши крихкі, об’ємні та енергоємні вакуумні лампи на напівпровідникові компоненти. Це забезпечило не лише фізичну мініатюризацію пристроїв, а й значне зниження енергоспоживання разом із суттєвим підвищенням надійності. Впровадження транзисторної логіки дозволило створювати складніші арифметико-логічні пристрої, здатні безперервно виконувати мільйони операцій без критичних апаратних збоїв. Саме стабільність нової елементної бази створила передумови для реалізації масштабних математичних розрахунків, які раніше переривалися через технічну недосконалість компонентів.

Важливим кроком на цьому етапі стало не лише апаратне вдосконалення, а й поява принципово нового програмного інструментарію. До цього моменту програмування здійснювалося на рівні машинних кодів або низькорівневих процесорних мов, що вимагало від математика глибоких знань архітектури конкретного процесора та робило процес перенесення алгоритмів надзвичайно трудомістким. Вченому, окрім власних знань цільової проблеми, необхідні були глибокі знання комп’ютерних технологій, що є іншою галуззю знань. Виникнення високорівневих мов програмування, серед яких ключове місце посів FORTRAN, здійснило революцію в методах взаємодії людини з машиною. Назва мови красномовно підкреслювала її цільове призначення – трансляцію математичних формул у машинний код. FORTRAN дозволив науковцям записувати складні алгоритми у вигляді, максимально наближеному до традиційної математичної нотації. Замість маніпулювання регістрами пам'яті та запам’ятовування системних викликів процесора, розробники отримали можливість оперувати масивами, вкладеними циклами та комплексними числами на рівні мовних конструкцій [8]. Це значно прискорило розробку математичного забезпечення, оскільки фокус уваги змістився з технічних нюансів реалізації на логіку самого обчислювального методу. Надання такого функціоналу дозволило автоматизувати розв’язання задач, обсяг яких раніше вважався недосяжним.

Поява великих інтегральних схем у 1970-х роках стала основою переходу від громіздких багатоплатних обчислювальних структур до компактних мікропроцесорних систем. Ця технологія дозволила розмістити мільйони транзисторів на одному

кристалі кремнію, що призвело до створення перших універсальних мікропроцесорів. Однак архітектура ранніх центральних процесорів була оптимізована переважно для роботи з цілочисельними даними та базовою логікою. Виконання складних математичних операцій із числами з рухомою комою потребувало програмної емуляції, що суттєво сповільнювало обчислювальний процес і робило наукові розрахунки вкрай неефективними.

Для задоволення потреб прикладної математики та інженерного моделювання критично важливим етапом стала розробка та впровадження спеціалізованих математичних співпроцесорів, відомих як Floating Point Unit (FPU) [9]. Основна перевага таких елементів полягала у апаратному виконанні ресурсомістких обчислень. Замість виконання десятків програмних інструкцій для однієї операції, FPU реалізовував обчислення тригонометричних, логарифмічних та експоненціальних функцій на рівні мікрокоду та схемотехніки. Це не лише радикально розвантажило центральний процесор, звільнивши його для задач загального керування, а й прискорило наукові розрахунки у десятки разів, наблизивши потужність персональних систем до рівня тогочасних міні-ЕОМ.

Прийняття стандарту IEEE 754 у 1985 році уніфікувало правила арифметики з рухомою комою для всієї обчислювальної техніки [10]. До цього моменту кожен виробник використовував власні формати представлення чисел та правила округлення, що призводило до розбіжностей у результатах одних і тих самих розрахунків на різних машинах. Стандарт IEEE 754 чітко визначив формати одинарної та подвійної точності, методи обробки виняткових ситуацій, таких як ділення на нуль або нечислові значення, та встановив жорсткі правила математичних операцій. Це забезпечило повну переносність програмного забезпечення та стабільність математичних алгоритмів, заклавши основу для подальшого розвитку систем автоматизації та сучасного прикладного програмного забезпечення.

На початку XXI століття класична архітектура центральних процесорів зіткнулася з термодинамічними та квантовими обмеженнями. Екстенсивне нарощування тактової частоти як основного методу підвищення швидкодії призвело до критичного

зростання тепловиділення, що унеможливило подальше лінійне прискорення обчислень. Цей технологічний бар'єр змусив індустрію перейти від парадигми послідовного виконання інструкцій до збільшення кількості потоків. Для прикладної математики це означало трансформацію механізму виконання методів: замість оптимізації одного потоку обчислень, складні задачі почали декомпозувати на тисячі незалежних мікроопераций, які стало можливо виконувати одночасно, паралельно.

Залучення графічних процесорів для неграфічних математичних розрахунків значно цьому сприяло. Архітектура GPU, що початково розроблялася для одночасного розрахунку координат та кольору пікселів на екрані, адаптували для базових задач лінійної алгебри. На відміну від центрального процесора, який містить кілька потужних ядер для обробки складної розгалуженої логіки, графічний прискорювач складається з тисяч простіших арифметико-логічних пристроїв. Оскільки більшість сучасних алгоритмів – від перетворення Фур'є до розв'язання систем лінійних рівнянь базуються саме на матричній та векторній алгебрі, GPU перетворилися на ідеальні апаратні матричні обчислювачі.

Повноцінному використанню нових апаратних можливостей сприяло створення програмних архітектур, таких як CUDA (Compute Unified Device Architecture) [11] та OpenCL [12]. Ітераційні процеси, чисельне інтегрування багатовимірних систем диференціальних рівнянь, криптографічний аналіз великих чисел та обробка гігантських статистичних вибірок – задачі, розв'язання яких раніше вимагало доступу до унікальних, складнодоступних комп’ютерів, стали виконуватися за лічені секунди навіть на базових робочих станціях, забезпечивши новий, значно вищий рівень автоматизації складних розрахунків.

Стрімкий розвиток технологій машинного навчання сприяв розвитку нового апаратного забезпечення. Математика штучних нейронних мереж базується переважно на багатовимірній лінійній алгебрі та тензорному численні. Головним обчислювальним процесом у таких моделях є масове виконання операції множення із накопиченням. Під час прямого проходження сигналу через нейронну мережу вхідні вектори даних перемножуються на багатовимірні матриці вагових коефіцієнтів із подальшим

додаванням векторів зміщення. Для навчання ж цих моделей використовується алгоритм зворотного поширення помилки, який вимагає мільйонів операцій знаходження частинних похідних та оптимізації багатовимірних функцій методами стохастичного градієнтного спуску [13].

Виконання таких масивів однотипних обчислень на традиційних центральних або навіть універсальних графічних процесорах супроводжується значними затримками через постійне пересилання даних між обчислювальними ядрами та оперативною пам'яттю [14]. Для подолання цього обмеження були створені тензорні ядра, що реалізують концепцію систолічних масивів. У такій архітектурі дані протікають через сітку обчислювальних вузлів безперервним потоком, що дозволяє виконувати математичні операції над цілими матрицями апаратно, за мінімальну кількість тактів і без проміжних записів у регістри [15].

1.3. Особливості мов програмування для виконання математичних алгоритмів

Паралельно з удосконаленням апаратної частини відбувався стрімкий розвиток мов програмування та програмних структур, що дозволило значно підвищити рівень абстракції при описі математичних алгоритмів. Це виростило окрему галузь програмування, відносно відокремлену від апаратних спеціалістів. Розробник програмного забезпечення отримав окремі інструменти, і знання з механічної будови комп’ютера, адреси регістрів та електроніки не є йому необхідними.

Перші високорівневі мови, такі як FORTRAN, ALGOL та згодом C, базувалися на імперативній та процедурній парадигмах. Їхньою основою була лінійна послідовність інструкцій, де програма розглядалася як набір кроків, що змінюють стан глобальних даних. Математичний алгоритм декомпозувався на окремі підпрограми (функції або процедури), які приймали вхідні параметри та повертали результат [16]. Цей підхід був надзвичайно ефективним для реалізації послідовних чисельних методів,

розрахунку траєкторій чи розв'язання систем рівнянь. Проте зі зростанням масштабів систем автоматизації виявився критичний недолік процедурного підходу: глобальний стан даних ставав некерованим. Коли складні математичні моделі почали інтегруватися з інтерфейсами користувача та системами збору даних з датчиків, код перетворювався на заплутану, некеровану конструкцію, що ускладнювало локалізацію помилок та модернізацію алгоритмів.

Вирішенням проблеми масштабованості стала поява та масове впровадження об'єктно-орієнтованого підходу до програмування (ООП). Ця парадигма докорінно змінила підхід до проєктування систем: замість написання послідовних інструкцій розробники почали моделювати предметну область у вигляді взаємодіючих об'єктів [16]. Для математики та автоматизації це мало колосальне значення. Абстрактні математичні концепти – матриці, вектори, комплексні числа або навіть цілі диференціальні рівняння – почали описуватися як незалежні класи зі своїми внутрішніми даними та алгоритмами обробки. ООП запровадило стандартизовані принципи створення коду та роботи з ним, що значно спростило підготовку нових спеціалістів, створення користувацьких бібліотек та інтеграцію нових технологій.

У сучасній комерційній розробці використання коду напряму відходить на другий план, посуваючись фреймворкам. Фреймворк – це вже створений іншими розробниками каркас, архітектура майбутньої системи [17]. Його задача – зняти з програмістів задачу зі створення базового функціоналу та зосередити їхні зусилля на безпосередній розробці вирішення цільової проблеми. Фреймворк бере на себе задачі з оптимізації, інтеграції з апаратним забезпеченням, організації роботи та інші, що не відносяться напряму до конкретної програми.

Вибір мови програмування для реальних проєктів нерозривно пов'язаний із вибором відповідної екосистеми фреймворків, які спеціалізуються на різних предметних областях. Наприклад, для побудови високонавантажених корпоративних систем традиційно використовуються мови C# та Java з їхніми потужними інфраструктурними рішеннями, такими як .NET (зокрема ASP.NET Core) [18] та Spring Boot [19]. У сфері веброзробки та створення користувацьких інтерфейсів домінують JavaScript та TypeScript завдяки фреймворкам Angular, React та Vue.js [17].

Для галузей, що потребують інтенсивних обчислень, також розроблені числені архітектури. Важкі математичні задачі сьогодні притаманні сферам машинного навчання та моделювання інтерактивних систем. У сфері штучних нейронних мереж беззаперечним лідером є мова Python, навколо якої побудовано інструментарій потужних математичних фреймворків, таких як TensorFlow та PyTorch [20]. Водночас у розробці інтерактивних систем, симуляторів та фізичних моделей активно застосовуються спеціалізовані рушії-фреймворки (наприклад, Unity на базі C# або Unreal Engine на базі C++), які беруть на себе складні задачі з рендерингу, фізичних симуляцій та інтеграції з технологіями 2D та 3D моделювання [21]. Сучасний інженер програмного забезпечення не займається реалізацією базових елементів програмної системи, він більше схожий на конструктора, який збирає складні системи з готових, високонадійних програмних блоків, що значно пришвидшує життєвий цикл розробки та підвищує стабільність кінцевого продукту.

Автоматизація комп'ютерних обчислень є нескінченною задачею. Адже постійний розвиток математичних підходів та задач вимагає постійного прогресу у апаратних та програмних елементах, постійну адаптацію до нових підходів та технологій. Нові виклики, нові підходи підбурюють інженерів комп'ютерних компонентів створювати нові системи, а розробників програмного забезпечення – реалізовувати новий функціонал. Наприклад, стрибок у розвитку штучних нейронних мереж за останні пару років підтверджує, що цей шлях ще далекий від завершення, і, неминуче, він призведе до подальших відкриттів та винаходів.

1.4. Огляд сучасних програмних реалізацій для математичних обчислень

На даний момент існує багато варіантів реалізації автоматизації математичних обчислень. Різноманіття сучасних цифрових технологій дозволяє відтворити задачу будь-яким чином, використовуючи різний підхід та архітектуру цифрової системи.

Веб-технології, десктопні програми, використання нейромереж – все це використовується аналогами розроблюваної системи для відтворення автоматизації математики. Розглянемо декілька з них:

1.4.1. MATLAB

MATLAB – це обчислювальна платформа, яка використовується для інженерних та наукових застосувань, таких як аналіз даних, обробки сигналів і зображень, систем керування, бездротового зв'язку і робототехніки. MATLAB включає мову програмування, інтерактивні додатки, вузькоспеціалізовані бібліотеки для інженерних застосувань та інструменти для автоматичної генерації вбудованого коду. MATLAB також є основою для Simulink, середовища блок-схем для моделювання складних багатодоменних систем [22].

Це середовище є професійним програмний коплексом, що дозволяє виконувати операції лінійної алгебри, розв’язувати системи диференціальних та алгебраїчних рівнянь, здійснювати статистичний аналіз, а також розробляти власні комплексні алгоритми. Важливою складовою системи є розширений інструментарій для побудови двовимірних та тривимірних графіків, що забезпечує наочну графічну інтерпретацію отриманих результатів і аналіз емпіричних даних.

Ключовою перевагою інформаційної системи MATLAB є обчислювані можливості, функціональність та модульність системи. Під більшість інженерно-наукових задач створений окремий модуль з необхідними програмними реалізаціями. Різноманітний інструментарій MATLAB підходить для вирішення задач будь-якої складності.

Головним же недоліком є складність системи. Вона орієнтована на професійне інженерно-наукове моделювання і вимагає від користувача знання власної мови складання алгоритмів та навичок роботи із неінтуїтивним інтерфейсом. Це робить її неефективної для швидкого рішення прикладних математичних задач нижчого рівня складності. На рис. 1.1 показано інтерфейс основного середовища розробки, на рис. 1.2 – інтерфейс компонента для аналізу динамічних систем Simulink.

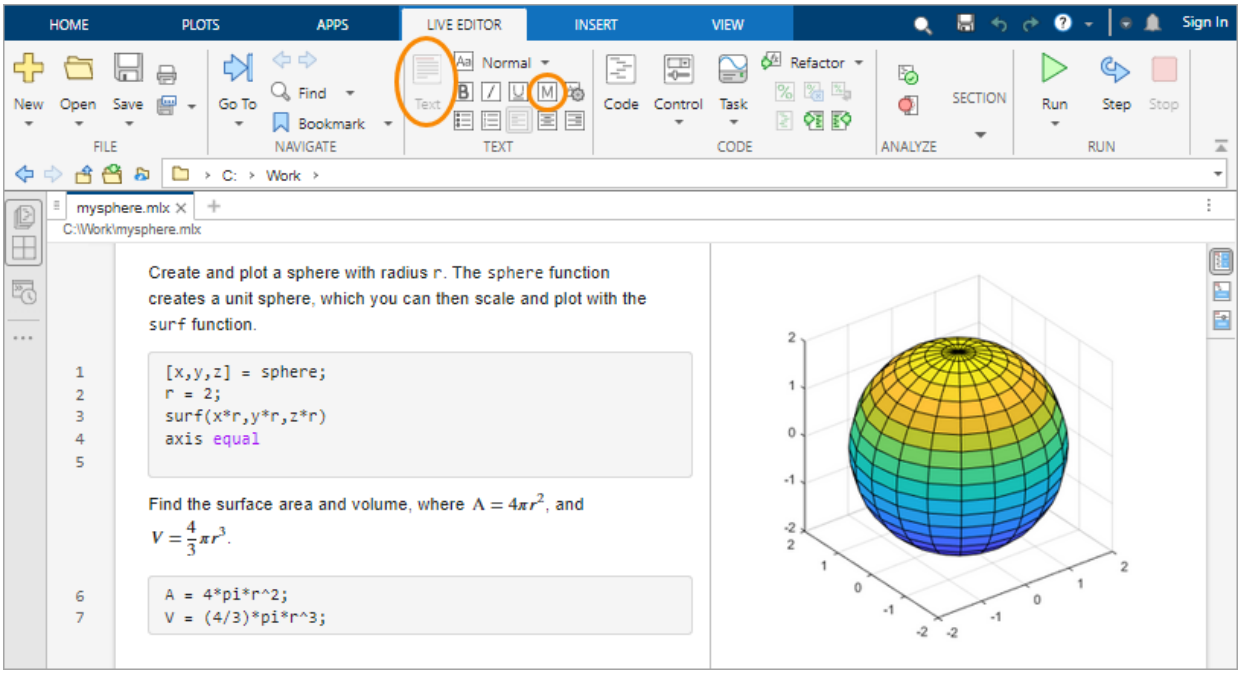

Рис. 1.1. Інтерфейс програми MATLAB.

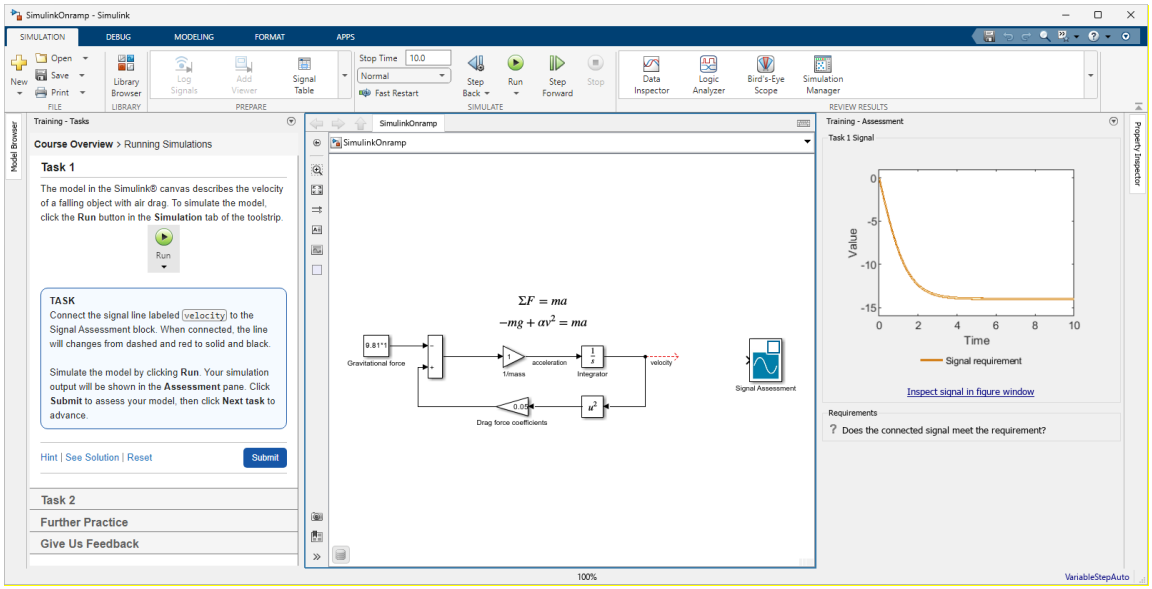

Рис. 1.2. Інтерфейс модуля Simulink.

1.4.2. WolframAlpha

WolframAlpha – це обчислювальна платформа, розроблена компанією Wolfram Research. Спеціалізується на математичних обчисленнях, але здатна виконувати і інші задачі. На відміну від інших обчислювальних програм, ця система динамічно генерує відповіді та виконує розрахунки на основі власної гігантської бази структурованих

даних, потужної системи комп'ютерної алгебри та технологій розпізнавання природної мови [23].

Головною перевагою є розпізнавання природньої людської мови використовуючи системи нейронних мереж, генерація в реальному часі відповідей на питання користувача. Через це система є привітною до нового користувача, ніяких додаткових знань та навичок для її використання не потрібно.

Це ж є і головним недоліком. Окрім очевидно необхідного підключення до інтернет мережі, адже подібні обчислення вимагають набагато більших обчислювальних потужностей, ніж є на середньостатистичному комерційному пристрої, існує і таке явище, як нейромережеві галюцинації та артефакти. Технологія розпізнавання людської мови є достатньо новою на даний момент і помилки розуміння та втрата сенсу можуть сильно вплинути на правильність отриманого результату.

Інтерфейс веб-системи WolframAlpha наведено нарис. 1.3. На рис. 1.4 показано використання системи: вирішення виразу $3x^2 + 5y^2 = 10$.

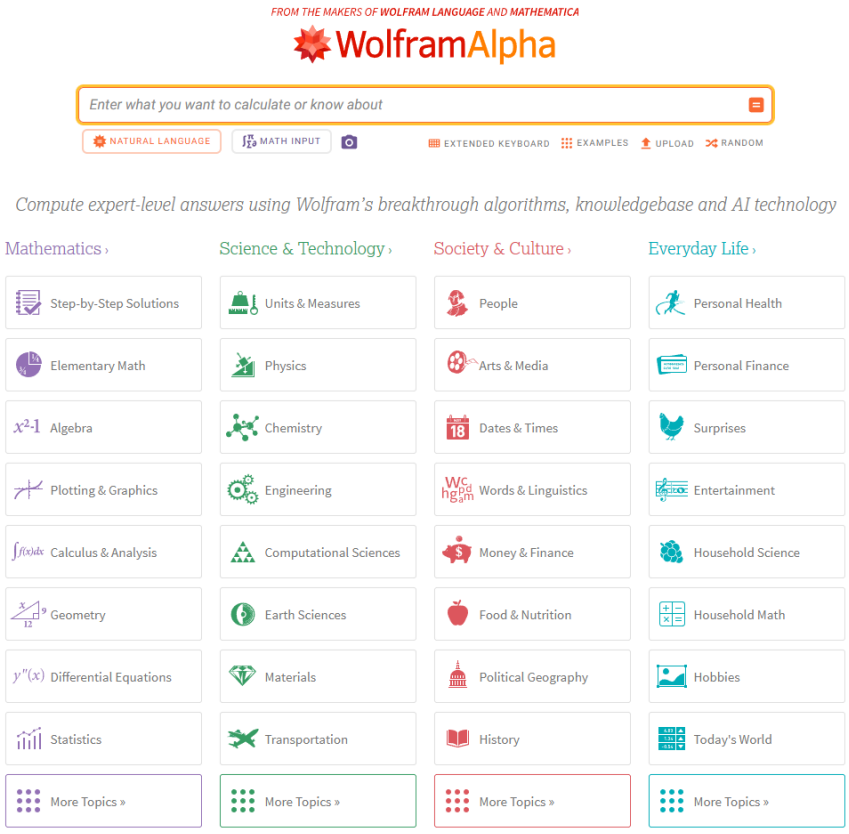

Рис. 1.3. Інтерфейс веб-системи WolframAlpha.

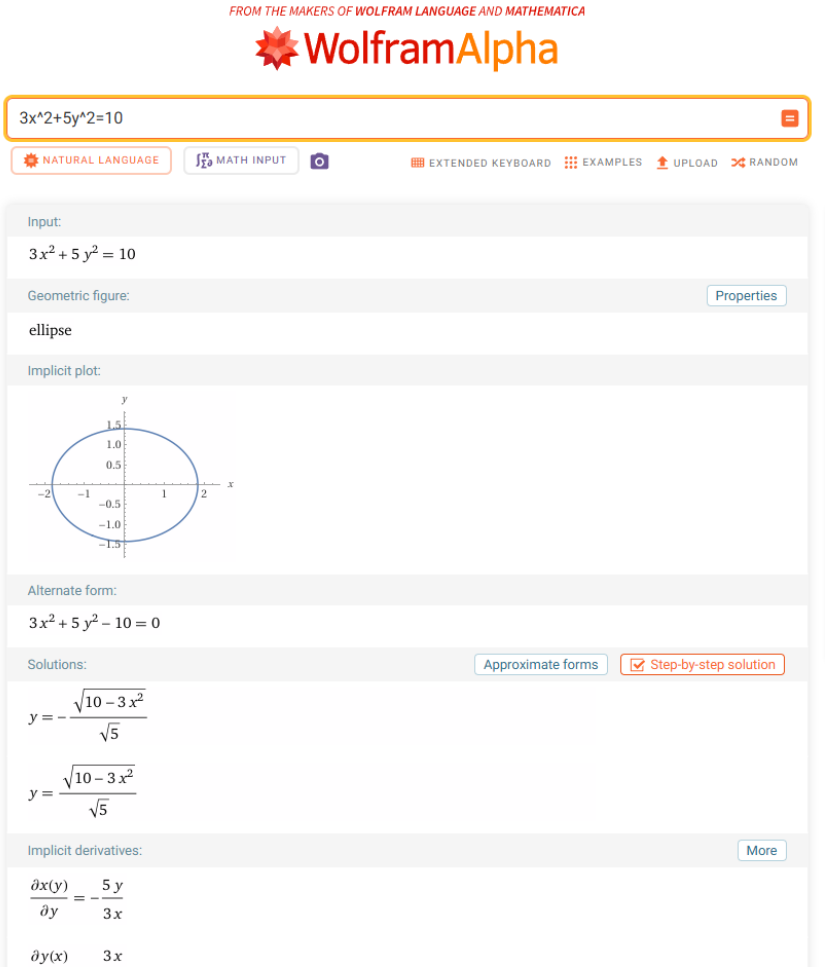

Рис. 1.4. Приклад функціоналу покрокового розв’язання WolframAlpha.

1.4.3. Photomath

Photomath – це мобільний застосунок, який використовує технології оптичного розпізнавання символів та алгоритми комп'ютерного зору для миттєвого зчитування друкованих або рукописних математичних виразів через камеру пристрою з їх подальшим автоматичним розв'язанням [24]. Також можливе ручне введення (рис. 1.5). Створений для вирішення математичних завдань шкільного рівня та рівня початкової вищої освіти. Перевагами є реалізація покрокового вирішення для освітніх цілей та сприйняття виразів з фото, що значно спрощує процес введення даних. Єдиним мінусом є те, що проєкт, як і усі попередні є комерційним, через що розблокування повного функціоналу є платним, що часто не є доступним цільовому користувачу.

Результат роботи мобільного додатку, а саме вирішення виразу $3x^2 + 5y^2 = 10$ можна побачити на рис. 1.6.

Рис 1.5. Ручне введення математичної задачі у додатку Photomath.

Рис. 1.6. Приклад функціоналу покрокового розв’язання Photomath.

1.4.4. Desmos

Desmos – це потужний графічний калькулятор, реалізований у вигляді веб-платформи та мобільного застосунку, який забезпечує миттєву візуалізацію складних математичних функцій (рис. 1.7.), систем рівнянь, нерівностей та параметричних кривих у двовимірній системі координат [25]. Введення математичних виразів здійснюється виключно вручну за допомогою спеціалізованої віртуальної клавіатури або комп'ютерної миші. Інструмент широко використовується в освітніх цілях від шкільного рівня до університетського. Головними перевагами Desmos є надзвичайно висока швидкодія побудови графіків у реальному часі, можливість інтерактивного маніпулювання параметрами для наочного дослідження поведінки функцій, а також те, що основний функціонал платформи є абсолютно безкоштовним. Основним недоліком системи, особливо у контексті оперативної роботи, є відсутність модуля комп'ютерного зору для автоматичного зчитування формул із камери, а також те, що програма лише візуалізує результат, але не надає користувачеві алгебраїчного розв'язання задачі.

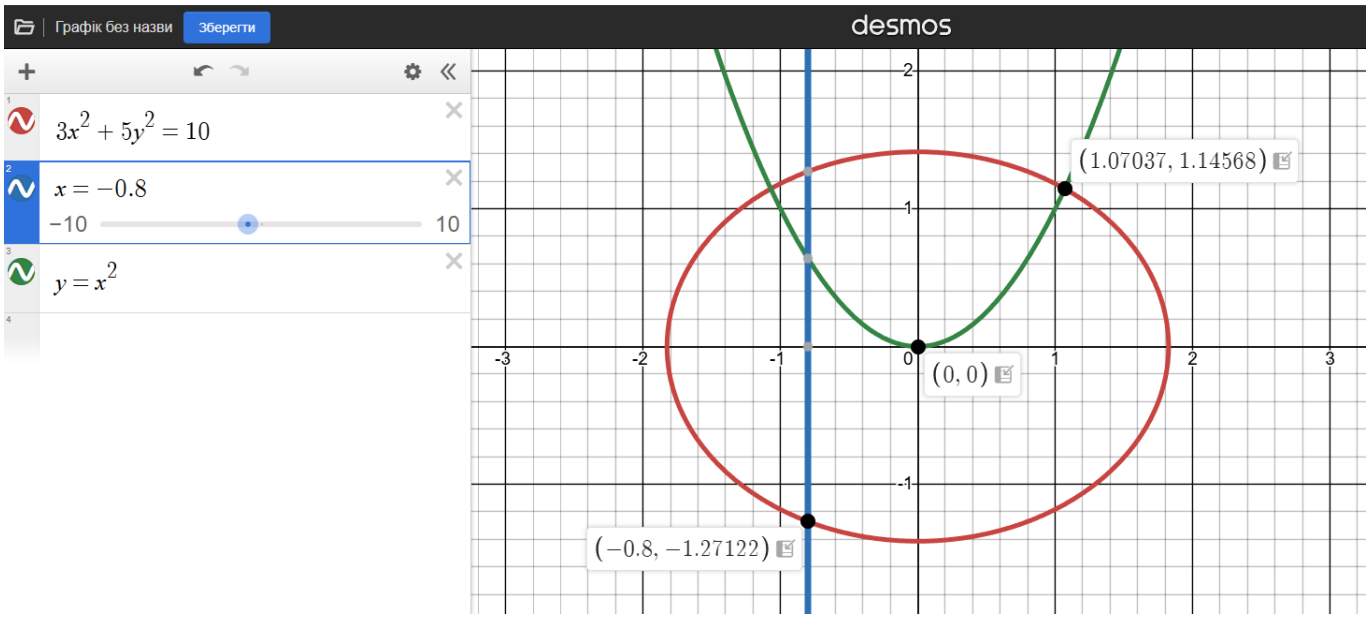

Рис. 1.7. Вирішення виразу за допомогою веб-платформи Desmos

Незважаючи на наявність потужних систем комп'ютерної алгебри та популярних мобільних застосунків, ніша швидких, компактних та повністю автономних засобів залишається відкритою. Більшість сучасних аналогів критично залежать від хмарної інфраструктури, вимагаючи стабільного підключення до мережі Інтернет для обробки зображень та виконання розрахунків. Інші ж системи є занадто громіздкими, комерційно закритими або вимагають високих апаратних потужностей для підтримки глибоких нейромереж чи симуляції систем для розрахунків. У зв'язку з цим, розробка нового мобільного додатка є технологічно та практично обґрунтованою. Головною концептуальною перевагою створюваної системи є її абсолютна автономність та алгоритмічна компактність.

1.5. Функціональні та нефункціональні вимоги до застосунку

Успішна реалізація будь-якого програмного продукту вимагає попереднього формування чітких критеріїв та специфікацій, яким він повинен відповідати. Для розроблюваної мобільної системи математичних обчислень усі вимоги логічно поділяються на дві основні категорії:

- функціональні – визначають безпосередню поведінку та можливості системи;
- нефункціональні – описують атрибути якості, технічні обмеження і властивості системи в цілому.

До базових функціональних вимог належить забезпечення користувача надійним інструментарієм для введення, редагування та обробки математичних даних. Додаток повинен мати інтегрований візуальний редактор для зручного вводу багатошарових формул із підтримкою математичної нотації. Обчислювальне ядро системи має забезпечувати автоматичне розпізнавання тексту, виконання точних алгебраїчних спрощень, аналітичне розв'язання рівнянь відносно заданої змінної, а також парамет-

ричний розрахунок виразів за відомих вхідних даних. Окремим блоком функціональних вимог є наявність розвиненої графічної підсистеми: програма повинна будувати інтерактивні двовимірні графіки функцій та нерівностей з автоматичним маркуванням критичних точок, а також генерувати тривимірні просторові моделі з можливістю їх огляду.

Нефункціональні вимоги диктуються специфікою цільового середовища розгортання. Додаток розробляється виключно для мобільних пристроїв під управлінням операційної системи Android, що вимагає адаптації графічного інтерфейсу під сенсорні екрани різних діагоналей. Усі інструменти взаємодії (масштабування, обертання, панорамування) повинні реагувати на загальноприйняті для цільової платформи жести користувача без затримок. Важливою вимогою є оптимізація споживання апаратних ресурсів смартфона: рендеринг графіки та виконання складних символьних обчислень не повинні призводити до критичного падіння частоти кадрів або перегріву пристрою.

З огляду на необхідність забезпечення високої портативності, встановлюється жорстке обмеження на обсяг кінцевого інсталяційного файлу. Вага зібраного пакета для встановлення додатка не повинна перевищувати 70 Мб. Дотримання цієї вимоги мінімізує використання дискового простору на носії користувача. Крім того, додаток має відповідати вимозі повної автономності, тобто виконувати всі математичні перетворення та геометричні побудови локально на пристрої, без необхідності підключення до мережі Інтернет.

РОЗДІЛ 2

СТВОРЕННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ

2.1. Архітектура створюваної системи

Будь-який процес розробки складної програмно-алгоритмічної системи починається з проєктування її архітектури. Архітектура програмної системи – це базова структура, яка визначає її ключові компоненти та принципи взаємодії між собою. Від того, наскільки грамотно та стратегічно її спроєктовано на початкових етапах життєвого циклу розробки, безпосередньо залежать критично важливі характеристики кінцевого продукту: швидкість його розробки, загальна швидкодія, масштабованість, відмовостійкість та придатність до подальшої модернізації чи підтримки.

У контексті розробки мобільного додатка для автоматизації математичних обчислень етап архітектурного проєктування набуває особливого важливого значення. На відміну від десктопних або серверних систем, розроблюваний додаток функціонує в умовах жорстких апаратних обмежень мобільних пристроїв (обмежений обсяг оперативної пам'яті, лімітований заряд акумулятора, різна продуктивність мобільних процесорів, різна архітектура їхніх систем). Оскільки система має виконувати ресурсомісткі завдання з обробки зображень, моделювання двохвимірних та трьохвимірних поверхонь, комп'ютерної алгебри виключно локально, до створення архітектури системи вимагає якісного планування.

З ціллю спрощення розробки, полегшення тестування та можливого подальшого масштабування системи в даному випадку було прийнято рішення строго дотримуватися принципу модульності [26], який забезпечує високу зв'язність логіки всередині окремих підсистем та мінімальну залежність їх одна від одної.

На базі цього принципу було спроєктовано загальну архітектуру програмної системи (рис. 2.1).

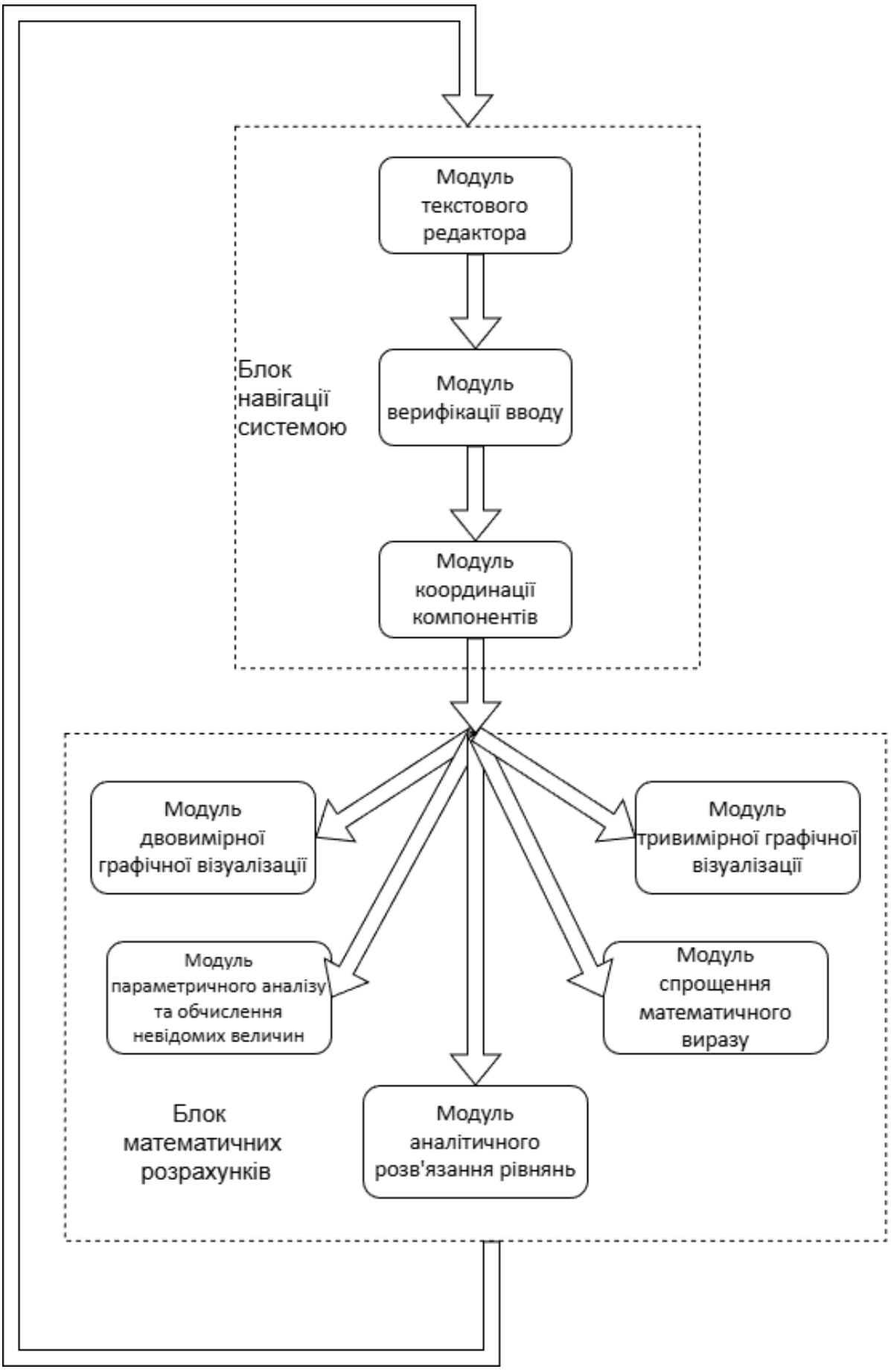

Рис. 2.1. Схема зв’язку модулів системи

Структурно розроблена архітектура поділяється на два основні блоки: блок навігації системою та блок математичних розрахунків.

Блок навігації системою відповідає за первинний прийом даних від користувача, їх перевірку та подальшу координацію процесів. До його складу входять наступні елементи:

- модуль текстового редактора;
- модуль верифікації вводу;
- модуль координації компонентів.

Блок математичних розрахунків виступає обчислювальним ядром програми, яке ізолює в собі всі спеціалізовані алгоритми обробки даних. Цей блок включає такі компоненти:

- модуль спрощення математичного виразу;
- модуль аналітичного розв'язання рівнянь;
- модуль двовимірної графічної візуалізації;
- модуль тривимірної графічної візуалізації;
- модуль параметричного аналізу та обчислення невідомих величин.

Застосування модульного підходу дозволило створити гнучку та стійку до помилок архітектуру, що повністю відповідає вимогам до сучасних мобільних додатків. Її головна перевага полягає у строгому ізолюванні ресурсомістких математичних обчислень від роботи графічного інтерфейсу. Це гарантує безперебійну передачу даних між компонентами та забезпечує стабільну, швидку реакцію візуальної оболонки навіть під час виконання складних аналітичних операцій.

Спроєктована модульна база слугує надійним каркасом, на основі якого було сформовано загальний алгоритм роботи додатка (рис. 2.2). Він наочно описує логічну послідовність кроків: від моменту введення математичного виразу до кінцевої візуалізації результатів на екрані.

Розглянемо складові модулі архітектури додатку та їх особливості.

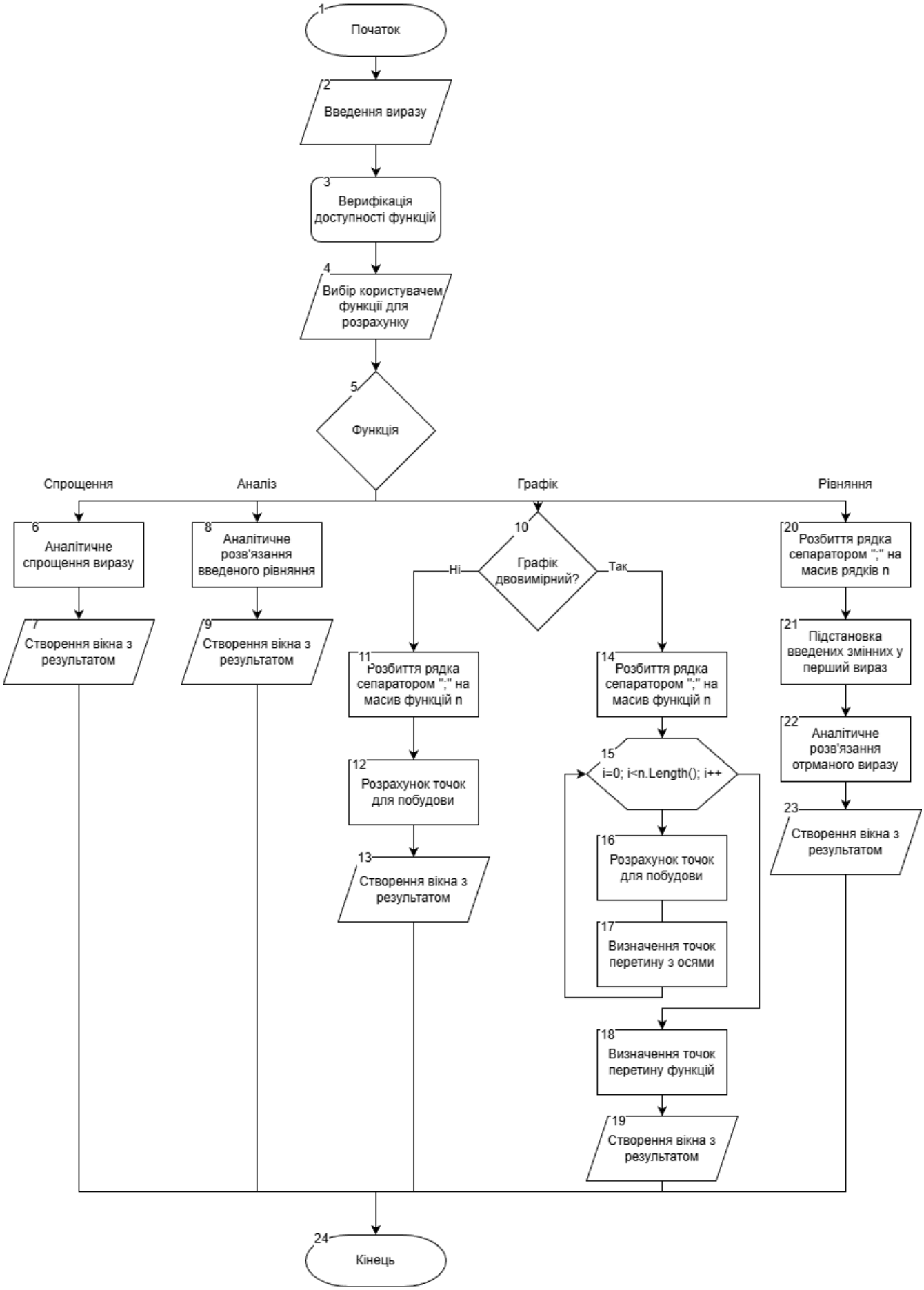

Рис. 2.2. Блок схема алгоритму роботи створеної програмної системи

2.1.1. Модуль текстового редактора

Першим компонентом програмної системи, з якою взаємодіятиме користувач, є модуль введення математичного виразу, з яким будуть працювати усі інші модулі. Його головним призначенням є забезпечення зручного інтуїтивно зрозумілого інтерфейсу для формування або коригування початкових математичних умов задачі.

Необхідність виділення цього редактора в окремий архітектурний блок, ізольований від інших, зумовлена специфічною складністю математичної нотації. На відміну від лінійного тексту, математичні вирази є двовимірними ієрархічними структурами, які містять багаторівневі дроби, системи рівнянь, матриці, а також верхні та нижні індекси. Стандартні системні компоненти введення мобільних операційних систем не мають вбудованих механізмів для підтримки такого просторового форматування.

Робота цього модуля слабо відрізняється від роботи звичайного текстового поля для користувача. Стандартна робота з текстом вимагає від користувача роботи з набором примітивів для формування складніших конструкцій. Проте, на відміну від звичайного тексту, де примітивами виступають букви, а конструкціями – слова, в математичній нотації базовими елементами є змінні, числа, операції та інші фрагменти обчислювального виразу, а складальними конструкціями вищого рівня – завершені вирази, що сформовані по строгим правилам (виконання яких забезпечує інший модуль, що буде описано нижче).

Модуль виконує декілька дій, які забезпечують реалізацію його головного призначення: приймає від користувача введені математичні символи, реалізовує відображення їх на екрані у коректній нотації та зберігає введене значення для подальшого використання іншими модулями у форматі, що підлягає розрахунку комп’ютерним методом. Таким чином, він виступає інформаційним шлюзом між користувачем та іншими модулями програмної системи.

Відносно окремою, автономною функцією архітектурного модуля є трансформація результатів обчислень з програмної мови у математичну нотацію. Більшість фу-

нкцій додатку мають демонструвати користувачу результат обчислень у вигляді іншого алгебраїчного виразу, який також необхідно відобразити правильно, згідно строгих правил запису математичних виразів. Так як модуль вже вміє це робити для введених користувачем даних, ця задача також лягає на нього.

2.1.2. Модуль спрощення математичного виразу

Спрощення є базовою операцією, що зменшує розмір запису математичного виразу, жодним чином не впливаючи на його алгебраїчний зміст. Цей процес є необхідним для виконання будь яких розрахунків, особливо в області фізики. По своїй суті спрощення є достатньо простою операцією, але складно піддається комп’ютерній автоматизації через свою абстрактність та різноманіття підходів.

Основною функцією модуля є виконання операції спрощення з введеним математичним виразом. Для цього йому необхідно отримати від текстового редактора введені дані, провести з ним усі потрібні операції та вивести у зручному, зрозумілому вигляді результат для користувача, також в рамках математичної нотації. Поряд із результатом обчислень знаходиться і введений вираз, для візуальної зрозумілості проведених модулем дій.

2.1.3. Модуль аналітичного розв'язання рівнянь

Аналітичне розв'язання рівнянь є важливою математичною операцією, що спрямована на знаходження точних символьних значень невідомих змінних, при яких заданий вираз перетворюється на істинну рівність. На відміну від чисельних методів, що дають лише наближений результат, аналітичний підхід зберігає абсолютну математичну точність, що є критично необхідним для теоретичного моделювання у фізико-технічних та інженерних дисциплінах. По своїй суті цей процес базується на строгих законах алгебри, однак він вкрай складно піддається комп'ютерній автоматизації. Це пов'язано з нелінійністю алгоритмів: система повинна не просто послідовно

виконувати інструкції, а обирати оптимальний шлях розв'язку з безлічі можливих (наприклад, застосування специфічних замін змінних або методів розкладання на множники) залежно від унікальної форми кожного конкретного рівняння.

Основною функцією модуля є виконання повного циклу знаходження коренів для введеного користувачем математичного виразу. Для цього йому необхідно отримати від текстового редактора формалізовані вхідні дані, провести їх класифікацію, застосувати відповідний ланцюжок алгебраїчних перетворень та вивести результат у зручному, зрозумілому вигляді для користувача в рамках звичної математичної нотації. Поряд із фінальною відповіддю на екрані відображається вихідний вираз. Таке компонування забезпечує максимальну візуальну прозорість, дозволяючи користувачеві не лише побачити кінцевий результат, а й повністю усвідомити логіку проведених модулем дій.

2.1.4. Модуль двовимірної графічної візуалізації

Цей модуль архітектури призначений для трансформації аналітичного запису математичних виразів у геометричні представлення функцій на декартовій площині. Компонент забезпечує наочний аналіз поведінки функціональних залежностей, що дозволяє користувачеві якісно оцінювати характер досліджуваних процесів.

Модуль забезпечує повне відображення графіків, через їхню часто нескінченну протяжність зображує тільки фрагмент, певну область, позицію та масштаб якої можна динамічно змінювати інтуїтивними природніми для мобільних пристроїв жестами. Обчислювальний алгоритм адаптивно підлаштовується під поточні координаційні межі, що гарантує цілісне відображення ліній функції у будь-якій обраній точці математичного простору без втрати деталізації фрагментів кривої.

Важливою особливістю цього модуля є його інтеграція з аналітичним обчислювальним рушієм системи. Блок не просто виконує дискретизацію та графічний рендеринг, а проводить попередній математичний аналіз кривих. Це дозволяє автоматично локалізувати та графічно виділити критичні точки графіка: точки перетину з осями абсцис та ординат. Крім того, у разі одночасного виведення кількох функціональних

залежностей, модуль здатний аналітично розраховувати точні координати точок їхнього взаємного перетину. Це реалізується шляхом алгебраїчного розв'язання відповідних систем рівнянь на символьному рівні, що виключає похибки, притаманні наближеним чисельним методам.

2.1.5. Модуль тривимірної графічної візуалізації

Окремим інструментом розроблюваної системи є модуль просторової візуалізації, призначений для побудови тривимірних поверхонь на основі функцій двох змінних виду $z = f(x, y)$. Головна відмінність цього компонента від двовимірної підсистеми полягає у зміні самої парадигми відображення: замість аналізу плоских кривих ліній система має рендерити цілісні об'ємні геометричні фігури. Це дозволяє користувачеві досліджувати топологію складних просторових об'єктів, таких як параболоїди, сідлові точки чи хвильові поверхні.

Архітектура та логіка роботи 3D-модуля жорстко продиктовані апаратними обмеженнями мобільних пристроїв. Розрахунок точних координат для тривимірної поверхні вимагає зведення багатовимірної матриці значень, що експоненціально збільшує навантаження на процесор порівняно з 2D-графікою. У зв'язку з цим система не використовує механізми безперервного динамічного перерахунку сітки під час взаємодії користувача з екраном. Натомість алгоритм генерує велику фіксовану полігональну сітку для заздалегідь визначеної широкої області координат лише один раз – у момент ініціалізації графіка.

З огляду на такий підхід до оптимізації ресурсів в автономному режимі, функціонал модуля має об'єктивні відмінності: тут відсутній плаваючий курсор для зчитування точних значень у конкретних координатах, а також не виконується обчислення складних просторових ліній перетину кількох різних поверхонь. Точний аналітичний пошук значень на тривимірній площині є надто ресурсомістким для локального виконання на мобільних процесорах без суттєвих затримок інтерфейсу.

Натомість інтерактивна взаємодія в цьому модулі повністю зосереджена на просторових трансформаціях згенерованої області. Користувач має можливість вільно

обертати отриману модель навколо всіх трьох координатних осей, а також змінювати загальний масштаб відображення. Таким чином, основна мета модуля тривимірної графіки зводиться не до знаходження конкретних числових параметрів, а до забезпечення якісного візуального розуміння форми, симетрії та загального характеру поведінки просторової функції.

2.1.6. Модуль параметричного аналізу та обчислення невідомих величин

Наступним компонентом розроблюваної архітектури є модуль параметричного аналізу. Його прямим призначенням є автоматизація процесу знаходження однієї невідомої змінної у багатопараметричній формулі за умови задання числових значень для всіх інших її компонентів. Цей інструмент орієнтований на розв'язання прикладних задач з фізики, геометрії та інженерних дисциплін, де базові закономірності описуються сталими рівняннями, з яких необхідно аналітично виразити шукану величину.

Отримавши масив вхідних даних від текстового редактора, в якому вказаний і сам вираз, і значення змінних, які є відомими для розрахунку, модуль виконує операцію символьної підстановки. Ключовим етапом є подальша алгебраїчна ізоляція невідомої величини. Спираючись на базові алгоритми математичного рушія, система здійснює тотожні перетворення вихідної формули таким чином, щоб єдина невідома змінна опинилася з одного боку рівності. Завершальним кроком є арифметичне обчислення або алгебраїчне спрощення отриманого виразу з урахуванням підставлених значень.

Практична цінність цього модуля полягає в усуненні необхідності ручного перетворення математичних формул. Його використання мінімізує ризик виникнення механічних помилок на етапах перенесення доданків чи роботи з дробами, забезпечуючи високу надійність та значно збільшує швидкість інженерних або навчальних розрахунків.

2.1.7. Модуль координації компонентів

Оскільки розроблена архітектура базується на принципі суворої ізоляції компонентів, де кожен алгоритмічний блок є самостійним і не залежить від інших, виникає об'єктивна необхідність у механізмі їхньої інтеграції. Цю роль виконує центральний координаційний модуль – компактний керуючий компонент, який забезпечує маршрутизацію процесів розробленої системи.

Координатор не виконує жодних математичних перетворень самостійно, натомість він слугує комунікаційним мостом між призначеним для користувача інтерфейсом та обчислювальними підсистемами. Залежно від обраного користувачем інструменту, координатор спрямовує вхідні дані до відповідного вузла: модуля спрощення, аналітичного розв'язання чи підсистем графічної візуалізації. Модуль створений для централізації усіх керуючих викликів всередині однієї структури, а не винесення їх по окремим іншим модулям для кожної функції окремо.

2.1.8. Модуль верифікації вводу

Завершальним компонентом програмної архітектури є модуль верифікації доступності математичних операцій. Його головне призначення полягає у попередньому логічному фільтруванні запитів: система має визначити, чи відповідає структура введеного виразу критеріям обраного користувачем типу розрахунку.

Необхідність цього блоку зумовлена тим, що система пропонує п'ять різних сценаріїв обробки даних (спрощення, аналітичне розв'язання, побудова 2D або 3D графіків, параметричний аналіз), вимоги до яких суттєво відрізняються. Наприклад, побудова двовимірного графіка вимагає наявності щонайменше однієї або двох змінних, тоді як аналітичне розв'язання потребує наявності оператора рівності. Спроба ініціювати розрахунок, який не відповідає типу виразу (наприклад, запит на побудову 3D-графіка для звичайного числа без змінних), призведе до математичної колізії та збою обчислювального рушія.

Робота модуля базується на алгоритмах лексичного та структурного аналізу. Отримавши формалізований рядок, алгоритм сканує його та виділяє ключові маркери: загальну кількість унікальних змінних, наявність знаків рівності чи нерівності, а також тип операторів. Після цього система зіставляє отримані параметри із закладеною матрицею правил допуску для кожного з п'яти обчислювальних блоків.

Результат роботи цього модуля безпосередньо інтегрується з інтерфейсом користувача. На основі сформованого логічного висновку система динамічно керує станом елементів керування: активує кнопки доступних операцій та блокує ті, виконання яких є математично некоректним для поточного виразу. Такий превентивний підхід унеможливлює відправку хибних запитів до процесора, знижує ризик виникнення програмних помилок під час виконання та робить взаємодію користувача з додатком інтуїтивнішою.

2.2. Компоненти реалізації мобільного додатку

Створення сучасного мобільного додатка, здатного виконувати ресурсомісткі математичні обчислення в автономному режимі, вимагає ретельного підбору інструментарію. Обраний технологічний стек має забезпечувати не лише кросплатформність (для можливого подальшого розширення) та продуктивність, а й гнучкість інтеграції зовнішніх бібліотек для роботи з комп'ютерною алгеброю та веб-відображенням. Базовими компонентами для реалізації спроєктованої архітектури стали середовище Unity [27], бібліотека MathLive [28], плагін WebViewPlugin [29] та математичний рушій AngouriMath [30].

2.2.1. Рушій Unity

У якості базової платформи та центрального координаційного середовища для розробки мобільного додатка було обрано ігровий рушій Unity. Незважаючи на те, що

історично ця платформа орієнтована на створення інтерактивних 3D- та 2D-ігор, на сучасному етапі розвитку технологій Unity являє собою потужний універсальний фреймворк для розробки будь-якого кросплатформного програмного забезпечення.

Основним фактором на користь вибору середовища Unity стала простота та гнучкість взаємодії з двовимірними та тривимірними просторами, на яких він спеціалізується, що є критично необхідною умовою для якісної реалізації модулів графічної візуалізації математичних функцій. Програмна побудова 2D та 3D графіків за допомогою базових нативних фреймворків мобільних операційних систем є надзвичайно складною та ресурсомісткою інженерною задачею. Вона вимагає глибокого розуміння низькорівневого програмування графічних інтерфейсів ручної реалізації алгоритмів відображення у просторі, керування матрицями проєкцій, складного налаштування віртуальних камер та обчислення глибини для об'ємних об'єктів. Натомість Unity, як рушій, що від початку архітектурно спроєктований для роботи з багатовимірною геометрією, автоматизує більшість цих процесів, беручи на себе всю складну математику рендерингу. Наявність потужних вбудованих підсистем просторового позиціонування, готових компонентів для оптимізованого створення векторних ліній (компонент LineRenderer) та алгоритмічної генерації складних полігональних сіток (компонент MeshFilter) дозволило абстрагуватися від апаратних нюансів графічного виведення і зосередити увагу безпосередньо на логіці математичного моделювання та аналітичному розрахунку координат, перекладаючи задачу ефективного, плавного та масштабованого відображення згенерованих кривих і поверхонь на надійно відпрацьоване графічне ядро фреймворку, що в результаті значно скоротило час розробки та підвищило стабільність роботи додатка.

Окрім потужних графічних інструментів, значним фактором для успішної реалізації проєкту в фреймворці Unity стала вбудована в обране середовище компонентно-орієнтована архітектура. Парадигма програмування, заснована на взаємодії сутностей та ізольованих скриптів-компонентів (MonoBehaviour), ідеально наклалася на спроєктовану архітектуру додатка. На практиці це дало змогу повністю уникнути написання монолітного коду на користь гнучкої системи слабко зв'язаних елементів. У

рамках створеної програмної системи кожен логічний вузол, кожен модуль було реалізовано як окремий програмний компонент зі своєю вузькою зоною відповідальності. Призначення цих скриптів незалежним об'єктам дозволило чітко розмежувати логіку обчислень та логіку відображення, яку майже повністю на себе взяв обраний рушій. Такий підхід суттєво спростив процес відлагодження на етапі написання коду та забезпечив високу стабільність готового продукту, сформувавши надійну базу для подальшого масштабування системи без ризику руйнування існуючих архітектурних зв'язків.

2.2.2. Мова програмування системи

Органічною сполучною ланкою всієї компонентної архітектури стала мова програмування C#. Сучасне середовище Unity використовує її як єдиний безальтернативний стандарт для розробки програмної логіки. Це забезпечує абсолютну однорідність кодової бази комплексу, дозволивши повною мірою використати потужні інструменти C# – строгу типізацію, автоматичне керування пам'яттю та механізми асинхронного виконання для реалізації складних математичних алгоритмів у межах єдиної екосистеми.

Важливою архітектурною особливістю розробленого додатка є одночасне використання двох концептуально різних типів зовнішніх розширень: чистих C# .NET бібліотек та нативних плагінів Unity.

C# .NET плагіни (до яких належить інтегрований математичний рушій AngouriMath) являють собою стандартизовані бібліотеки класів, що містять виключно алгоритмічну та математичну логіку. Вони є цілком абстрактними, апаратно та платформо незалежними. Їхня взаємодія з проєктом відбувається виключно на рівні середовища виконання, простіше кажучи, вони є інструментарієм мови програмування. Оскільки Unity має глибоку інтеграцію з екосистемою .NET, рушій здатний безпосередньо імпортувати ці бібліотеки та компілювати їхній код разом із основною логікою додатка. Це дозволяє виконувати складні маніпуляції з абстрактними синта-

ксичними деревами та символьними обчисленнями максимально швидко, використовуючи лише оперативну пам'ять та процесорні потужності, без жодних звернень до специфічних функцій операційної системи.

Натомість нативні плагіни Unity (такі як використаний WebViewPlugin) виконують функцію апаратного або системного комунікаційного моста. Вони застосовуються тоді, коли додатку необхідно отримати доступ до специфічних прикладних інтерфейсів конкретної мобільної ОС, які недоступні у стандартному інструментарії Unity. Такі плагіни створені для розширення можливостей та інтеграції конкретних окремих сутностей до стандартної версії рушія.

2.2.3. Оптимізація роботи рушія Unity

Попри всі архітектурні переваги, варто визнати, що середовище Unity є об'єктивно надлишковим та занадто потужним для реалізації утилітарного математичного додатка. Оскільки цей рушій історично проєктувався для розробки складних інтерактивних ігор з інтенсивною графікою, його базова конфігурація містить величезну кількість вбудованих підсистем, які є абсолютно непотрібними для роботи математичного програмного продукту. Невикористані модулі в активному стані неминуче призвело б до критичного збільшення розміру кінцевого інсталяційного файлу, надмірного споживання оперативної пам'яті смартфона та стрімкого виснаження заряду акумулятора через виконання непотрібних фонових розрахунків. З огляду на це, невід'ємним і вкрай важливим етапом розробки стала глибока оптимізація проєкту шляхом цілеспрямованого відсікання надлишкового функціоналу на рівні налаштувань ядра та конвеєра компіляції. Зокрема, з проєкту були повністю виключені важкі модулі тривимірної та двовимірної фізики, оскільки додаток не оперує поняттями гравітації чи колізій твердих тіл. Також була деактивована система обробки звуку та вимкнені підсистеми складного рендерингу: глобальне освітлення, розрахунок динамічних тіней і системи частинок. Для відображення інтерфейсу та математичних поверхонь застосовано найпростіші шейдери без урахування освітлення. Завдяки такому жорст-

кому урізанню базового інструментарію вдалося успішно розв'язати проблему надмірного функціоналу рушія, перетворивши Unity на компактне, максимально енергоефективне та швидкодіюче середовище, здатне спрямовувати всі апаратні ресурси мобільного пристрою виключно на виконання обчислювальних алгоритмів та побудову графіків.

Поряд із загальною структурною оптимізацією рушія, критичним аспектом забезпечення стабільної роботи додатка стала правильна організація обчислювальних процесів у часі. Оскільки аналітичне розв'язання складних багатоступеневих рівнянь, парсинг об'ємних виразів або розрахунок щільної координатної сітки для тривимірних графіків є вкрай ресурсомісткими операціями, їх синхронне виконання у межах головного потоку є архітектурно неприпустимим. Завантаження основного потоку такими задачами неминуче призвело б до зависання програми, повної втрати відгуку графічного інтерфейсу до моменту завершення обчислень. Для розв'язання цієї проблеми у проєкті було глибоко інтегровано парадигму асинхронного програмування. Завдяки використанню вбудованих механізмів платформи .NET та специфічної системи співпрограм Unity, усі найскладніші математичні обчислення були винесені у незалежні фонові задачі. Таке архітектурне рішення дозволило системі безперервно відмальовувати графічний інтерфейс із заданою частотою кадрів та миттєво реагувати на дії користувача (наприклад, плавно відображати анімацію завантаження або пересуватися площиною побудованого графіка), поки у фоновому режимі, без жодної шкоди для чуйності додатка, виконується складний математичний аналіз введених даних.

2.2.4. Структурна організація проєкту в середовищі Unity

В основу роботи додатка в середовищі Unity покладено концепцію односценної архітектури. На відміну від класичних проєктів, де кожен новий рівень або вікно завантажується як окрема сцена, створена програма функціонує в межах єдиного безперервного програмного простору. Усі переходи між режимами (від введення фор-

мули у веб-редакторі до виведення покрокового розв'язку чи побудови графіка) відбуваються шляхом динамічного вмикання, вимикання або трансформації відповідних груп об'єктів. Це повністю усуває затримки на завантаження додаткових ресурсів і забезпечує миттєвий відгук інтерфейсу. Створена архітектура проєкту Unity показана на рис. 2.3.

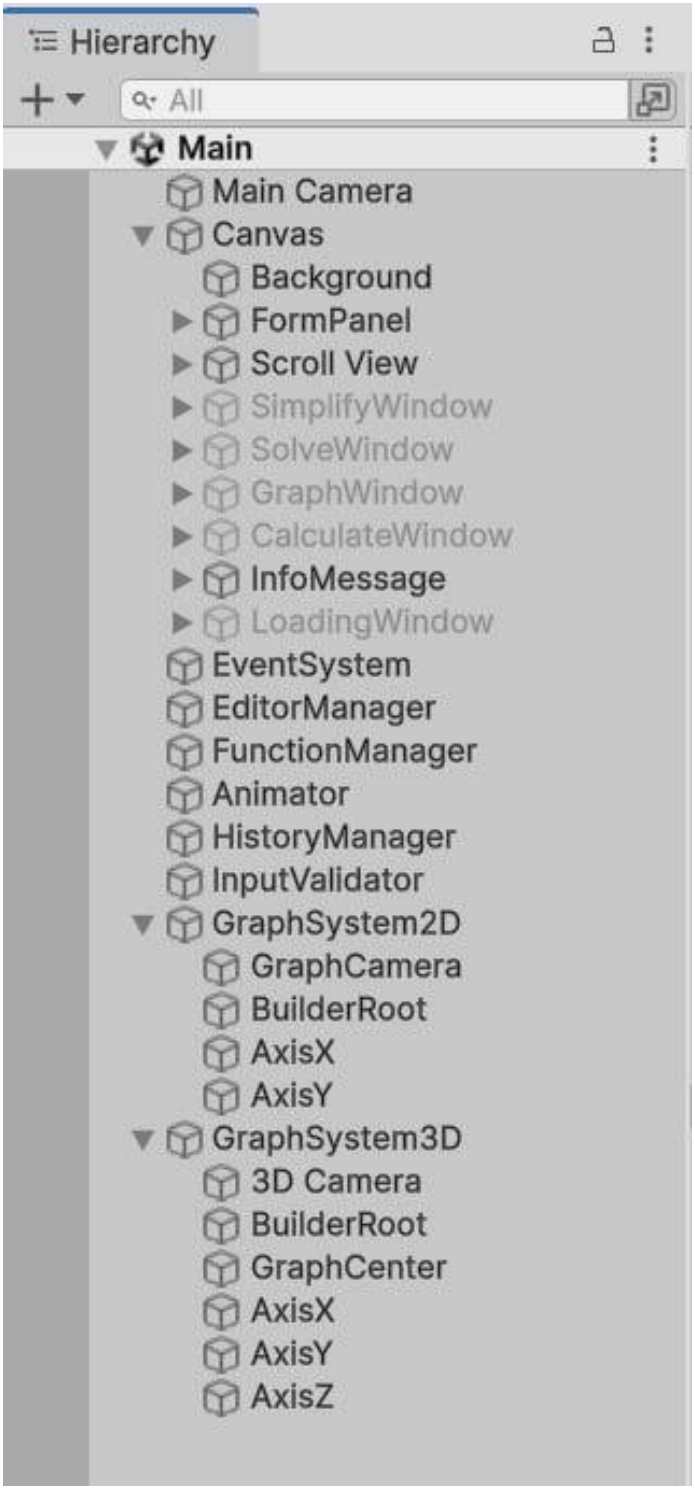

Рис. 2.3. Архітектура створеного проєкту у середовищі Unity

Програмний простір сцени логічно та візуально розділений на три основні групи об’єктів: графічний інтерфейс користувача, група функціональних об’єктів та геометричні візуалізатори графіків функцій (однієї та двох змінних).

2.2.4.1. Інтерфейс користувача.

Центральним елементом побудови графічного інтерфейсу користувача є компонент Canvas – базова підсистема рендерингу двовимірних об'єктів у середовищі Unity. Вона виконує роль абстрактного віртуального екрана, поверх якого проєктуються всі видимі елементи керування та текстової інформації. Налаштування компонента наведено на рис. 2.4.

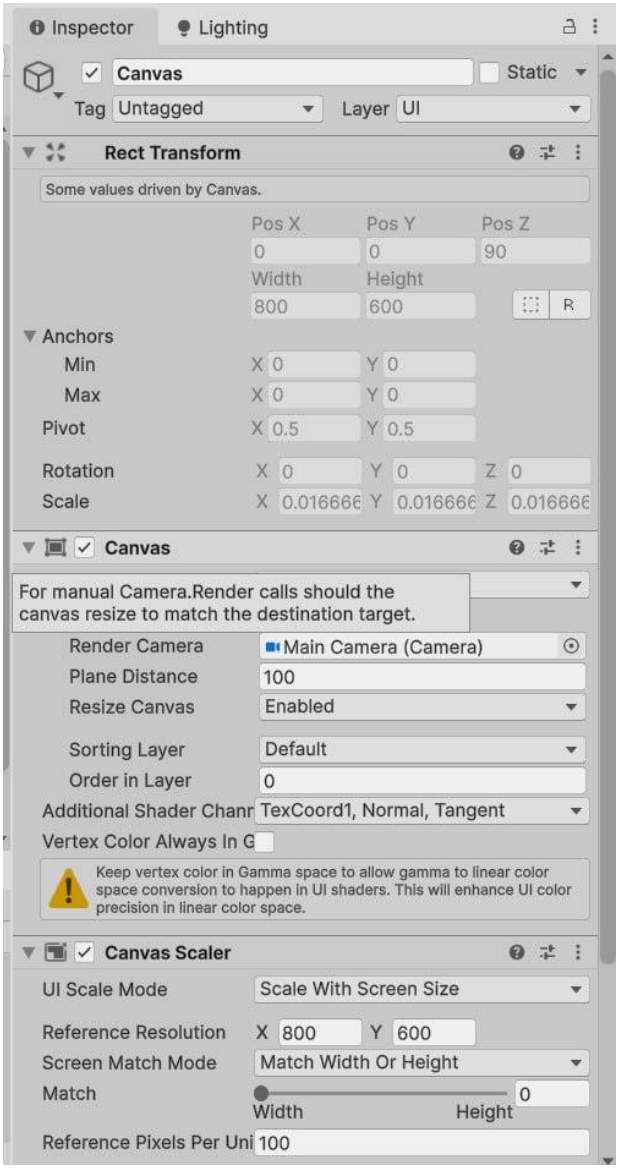

Рис. 2.4. Налаштування об’єкта з компонентом Canvas, що відповідає за відображення інтерфейсу

Механіка роботи цього компонента базується на використанні системи гнучких якорів та автоматичного масштабування, що гарантує збереження заданих пропорцій і коректне позиціонування інтерфейсу незалежно від фізичної діагоналі чи співвідношення сторін дисплея мобільного пристрою. У рамках розроблюваного комплексу архітектура Canvas має глибоко продуману ієрархію, логічно розділену на кілька робочих зон. Першою є панель інтерактивного графічного редактора (FormPanel), яка виступає системним контейнером для розгортання WebView-компонента та введення первинних математичних виразів (рис. 2.3). Поряд із нею інтегровано динамічний контейнер прокручування – Scroll View та дочірні (рис. 2.5), у якому розміщено ергономічне меню з кнопками виклику цільових функцій (спрощення, аналітичне розв'язання тощо), а також компактна панель історії. Ця панель автоматично фіксує та зберігає останні введені вирази користувача, дозволяючи миттєво повертатися до раніше введених даних без необхідності їх повторного набору. Важливим оптимізаційним рішенням є реалізація вікон для виведення результатів обчислень (об’єкти з назвами «…Window»). Вони заздалегідь розміщені в ієрархії Canvas, проте за замовчуванням перебувають у деактивованому стані (рис. 2.3). Вони не споживають ресурсів на графічне відмальовування і миттєво активуються керуючим скриптом лише після того, як математичний рушій завершує обробку запиту. Це дозволяє зберігати інтерфейс візуально чистим на етапі введення даних та забезпечує плавну зміну контексту під час демонстрації готового розв'язку.

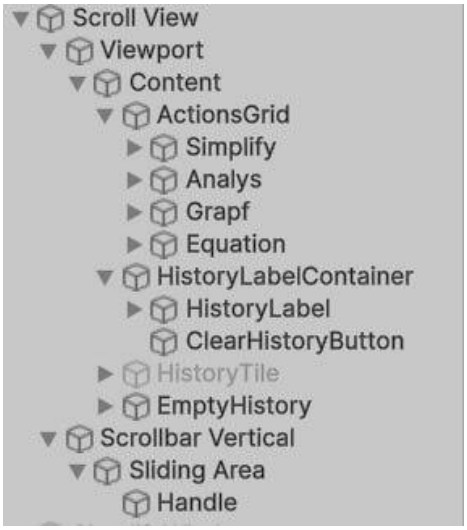

Рис. 2.5. Ієрархія компонента ScrollView, що відповідає за графічні елементи

На рис. 2.6 показано початкове меню додатку. На ньому можна побачити графічне зображення описаної вище архітектури. Блок зверху з написом «Формула» та полем для вводу тексту – об’єкт FormPanel, кнопки виклику функцій та область із історією введених виразів – об’єкт ScrollView. На рис. 2.6. прокручування недоступне, адже при початку роботи з додатком панель історії порожня. При її наповненні інтерфейс динамічно підлаштується під новий розмір.

Рис. 2.6. Стартовий екран додатку

2.2.4.2. Функціональні об’єкти

Частину ієрархії проєкту займають об’єкти, єдиною задачею який є запустити один виконуваний скрипт. Наприклад, об’єкт EventSystem є системний, згенерованим рушієм, відповідає за обробку подій взаємодії користувача з екраном (натискання на кнопки, ідентифікація жестів та інших дотиків). Він нерозривно поєднаний з самим Canvas, один без одного працювати не буде. Контролем елементів з рис. 2.6 займаються 3 об’єкти та відповідні їм класи: EditorManager, клас FormulaEditorBridge (відповідає за редактор формул); FunctionManager, клас ExecFunction (відповідає за виклик функцій при натисканні на кнопки) та HistoryManager з однойменним класом (займається записом, видаленням, відображенням історії введення та завантаженням виразу у поле редагування). Код класу ExecFunction:

```
public class ExecFunction : MonoBehaviour
{
   public SimplifyFunction f1;
   public SolveEquationFunction f2;
   public GraphWindow f3;
   public CalculateFunction f4;
   public void ClickOnFunctionButton(int id)
   {
      AnimatorScript.Instance.StartLoading();

      switch (id)
      {
         case 1:
            {
               f1.Exec();
            }break;
         case 2:
            {
               f2.Exec();
            }break;
         case 3:
            {
               f3.Exec();
            }break;
         case 4:
            {
               f4.Exec();
            }break;
      }
   }
}
```

Найпростіший клас створеної системи, повністю займається обробкою натискань на кнопки виклику функцій, та направляє виконання до іншого відповідного натиснутій кнопці класу, яку він визначає по назначеному кожній кнопці ідентифікатору. Також у ньому викликається AnimatorScript.Instance.StartLoading(), що звертається до іншого класу AnimatorScript (відповідає за анімацію завантаження, поки інші компоненти виконують розрахунки та анімацію копіювання результату обчислень у буфер пристрою). Код цього класу:

```
public class AnimatorScript : MonoBehaviour
{
   public static AnimatorScript Instance;

   private void Awake()
   {
      Instance = this;
   }

   [SerializeField] private GameObject loadingScreen;
   [SerializeField] private GameObject InfoNote;

   public void StartLoading()
   {
      Debug.Log("StartLoading");
      loadingScreen.gameObject.SetActive(true);
   }

   public void StopLoading()
   {
      Debug.Log("StopLoading");
      loadingScreen.gameObject.SetActive(false);
   }

   public void ShowInfo(string message)
   {
      InfoNote.GetComponentInChildren<TMP_Text>().text = message;
      float startY = InfoNote.transform.localPosition.y;
      LeanTween.moveLocalY(InfoNote, startY + 200, 0.2f).setOnComplete(() =>
      {
         LeanTween.moveLocalY(InfoNote, startY, 0.2f).setDelay(2f);
      });
   }
}
```

В класі окрім використання спрощеного патерну «Одинак» («Singleton») [31] можна побачити використання пакету LeanTween [32]. Він додає до рушія інструме-

нтарій простих маніпуляцій з об’єктами в асинхронному режимі, що використовується замість ручного прописування змін у положенні об’єктів кожен кадр. Це значно спрощує поставлену задачу реалізації простої анімації завантаження. Вона ж в свою чергу є необхідною для того, аби дати користувачу розуміння, що програма займається обчисленнями і її поточний стан є запланованим.

Окреме місце в ієрархії робочої сцени займає об’єкт InputValidator, що відповідає за верифікацію вхідних даних, реалізовану класом InputCheck. На відміну від раніше описаних об'єктів графічного інтерфейсу чи підсистем рендерингу, цей контролер не бере безпосередньої участі у візуальному відображенні інформації на екрані, а функціонує виключно на рівні керування внутрішньою логікою та станами системи. Працюючи у фоновому режимі як глобальна точка доступу, об'єкт безперервно перехоплює поточні текстові рядки, що надходять від користувача, та проводить їхній первинний структурний аналіз. За допомогою інтегрованих методів математичного рушія AngouriMath скрипт тестує введений вираз на предмет загальної синтаксичної коректності, після чого перевіряє його відповідність специфічним критеріям кожної з доступних операцій: підраховує кількість розділювачів, ідентифікує наявність буквених змінних та операторів відношення. Спираючись на результати цього багатокрокового парсингу, контролер динамічно керує станом активності навігаційних кнопок. Наприклад, він блокує можливість ініціювати аналітичне розв'язання, якщо у виразі відсутній знак рівності, або деактивує кнопку параметричного розрахунку, якщо алгоритм виявляє, що після підстановки заданих умов кількість невідомих змінних не дорівнює одиниці. Такий превентивний підхід забезпечує надійний захист обчислювального ядра від математичних колізій, фізично унеможливлюючи відправку некоректних запитів до процесора ресурсомістких алгоритмів.

2.2.4.3. Геометричні ієрархії

Група ігрових об'єктів, що відповідає за геометричну візуалізацію двовимірних графіків, принципово відрізняється від інших. Вони знаходяться далеко від розміщеного об’єкту Canvas, що в жодній ситуації не дозволить користувачу побачити величезні об’єкти графіків поза задуманим вікном, не дивлячись на їх знаходження на тій

самій сцені. Це архітектурне рішення дозволяє повністю виключити графічний конвеєр із робочого процесу, суттєво заощаджуючи апаратні ресурси мобільного пристрою, та завантажувати його виключно за цільовим запитом користувача на побудову графіка. Для функцій однієї змінної створена власна камера, що є спеціально налаштованою працювати в ортографічному режимі проєктування. Такий режим є критично важливим для 2D графіки, оскільки він гарантує повну відсутність перспективних спотворень, забезпечуючи математично точне та рівномірне відображення масштабів у будь-якій точці площини. Згенероване цією камерою зображення не виводиться на дисплей пристрою безпосередньо, натомість воно програмно транслюється у спеціальну текстуру рендерингу. Далі ця текстура проєктується на елемент графічного інтерфейсу, створюючи ізольоване інтерактивне вікно, у якому користувач може масштабувати та переміщувати робочу область. Базовий геометричний простір всередині цієї області формується окремим набором ігрових об'єктів, які відповідають за генерацію координатних осей (абсцис та ординат) із відповідною динамічною розміткою та сіткою для спрощення орієнтації по площині графіка. Для безпосереднього відображення самих математичних функцій застосовуються спеціалізовані порожні об'єкти, оснащені компонентом LineRenderer. Отримавши від аналітичного рушія дискретний масив обчислених координат точок для поточної області видимості, цей компонент за допомогою графічного процесора з'єднує їх векторними відрізками. У результаті формується неперервна суцільна лінія, що точно повторює траєкторію досліджуваної кривої на площині.

Архітектурна реалізація підсистеми тривимірних графіків побудована за аналогічним принципом ізольованих ігрових об'єктів, проте використовує інший набір базових компонентів Unity. Центральним елементом цього вузла є окрема віртуальна камера, яка налаштована виключно в режимі перспективи, що є критично необхідним для створення ефекту глибини та коректного відображення об'єму просторових фігур. Зображення з цієї камери, як і у двовимірному варіанті, перенаправляється на текстуру рендерингу на відповідне вікно інтерфейсу Canvas. Безпосереднє формування 3D-поверхні в просторі сцени відбувається за допомогою порожнього об'єкта, на який призначено два системних компоненти: MeshFilter та MeshRenderer. Керуючий

скрипт цього об'єкта реалізує процедурну генерацію полігональної сітки (Mesh). Він отримує статичний масив обчислених координат і транслює його у специфічні для Unity структури даних: масив просторових вершин (масив типу Vector3) та масив індексів, які вказують графічному рушію, як саме ці вершини з'єднуються у трикутники. Компонент MeshFilter діє як контейнер для збереження згенерованої геометрії, тоді як MeshRenderer відповідає за її візуальне відмальовування, накладаючи матеріал та шейдер для заливки площин. Можливість візуального огляду графіка користувачем реалізована без перерахунку самої сітки – шляхом прямого звернення до компонента Transform ігрового об'єкта, змінюючи його параметри обертання (поворот за допомогою кватерніонів) та локального масштабування у відповідь на сенсорне введення.

2.2.5. Бібліотека MathLive та плагін WebViewPlugin

Ключовим інструментом системи є спеціалізована бібліотека MathLive. Це потужний веб-компонент відкритого коду, розроблений на базі JavaScript та орієнтований на інтерактивне введення, редагування і високоякісний рендеринг складних математичних формул. Необхідність його залучення до проєкту зумовлена об'єктивними архітектурними обмеженнями базового функціоналу рушія Unity: стандартні елементи текстового введення (навіть такі просунуті як TextMeshPro – сучасний стандарт текстових елементів Unity) здатні оперувати лише послідовним однорядковим текстом і не підтримують багатошарову математичну нотацію, яка є критично необхідною для коректного відображення алгебраїчних дробів, вкладених радикалів тощо. Спроба розробки власного візуального редактора математичних виразів нативного рівня вимагала б невиправданих витрат часу та системних ресурсів, адже написання власного текстового рушія – задача складніша сама по собі, ніж створювана система.

MathLive елегантно вирішує цю проблему, надаючи користувачеві інтуїтивно зрозуміле середовище із власною вбудованою інтерактивною математичною клавіатурою. У межах розробленої архітектури додатка цей інструмент не взаємодіє з Unity безпосередньо, а розгортається всередині ізольованого веб-контейнера, який проєк-

тується поверх основного інтерфейсу користувача. Під час роботи з додатком користувач візуально конструює математичний вираз за допомогою звичних символів, а бібліотека миттєво, у режимі реального часу, відмальовує його з дотриманням усіх класичних стандартів математичного форматування.

Головна ж прикладна цінність інструменту MathLive для внутрішньої логіки додатка полягає в його здатності здійснювати двосторонню трансляцію даних. Після того як користувач завершує конструювання виразу, вбудовані алгоритми бібліотеки автоматично конвертують побудовану візуальну геометрію формули у строгий формалізований машинний рядок (використовуючи стандарт LaTeX або синтаксис чистого тексту). Цей стандартизований рядок через комунікаційний міст експортується з веб-середовища у простір Unity, де перехоплюється модулем InputCheck для подальшої верифікації, лексичного аналізу та безпосередньої передачі до аналітичного обчислювального рушія.

Технологічною основою для практичної реалізації зазначеного комунікаційного моста став спеціалізований компонент WebViewPlugin. Його інтеграція дозволила повністю усунути проблему архітектурної несумісності між ігровим рушієм Unity та веб-технологіями, на яких базується інтерфейс MathLive. Програмний механізм цього плагіна забезпечує динамічне розгортання нативного вікна веб-перегляду операційної системи безпосередньо поверх відмальованого графічного полотна Unity Canvas. У цей ізольований контейнер додаток завантажує локальний веб-скрипт із інтегрованим математичним редактором.

Основна функція WebViewPlugin у структурі розробленого комплексу полягає в організації стабільного інтерфейсу взаємодії для двосторонньої асинхронної передачі даних між середовищем виконання JavaScript та об'єктно-орієнтованим кодом на C#. При будь-якій зміні символьного вмісту в полі введення MathLive, внутрішня JavaScript-функція автоматично транслює поточний стан формули і передає згенерований LaTeX-рядок у комунікаційний канал плагіна. WebViewPlugin перехоплює цей текстовий пакет, виконує його внутрішнє перекодування в сумісний формат STRING мови C# та миттєво маршрутизує до керуючого модуля InputCheck. Цим процесом керує вищезгаданий об’єкт EditorManager та клас FormulaEditorBridge (Додаток А).

Таке інженерне рішення дозволило забезпечити абсолютну цілісність передачі інформації, унеможливило виникнення помилок при декодуванні специфічних математичних символів та дозволило реалізувати швидку реакцію обчислювальних модулів системи на дії користувача в реальному часі.

Завдяки інтеграції цих двох компонентів у проєкт, вдалося реалізувати інтуїтивний редактор математичних виразів. Схематичну взаємодію між бібліотекою MathLive, плагіном WebViewPlugin та іншими компонентами системи показано на рис. 2.7а, а зовнішній вигляд отриманого редактора на рис. 2.7б.

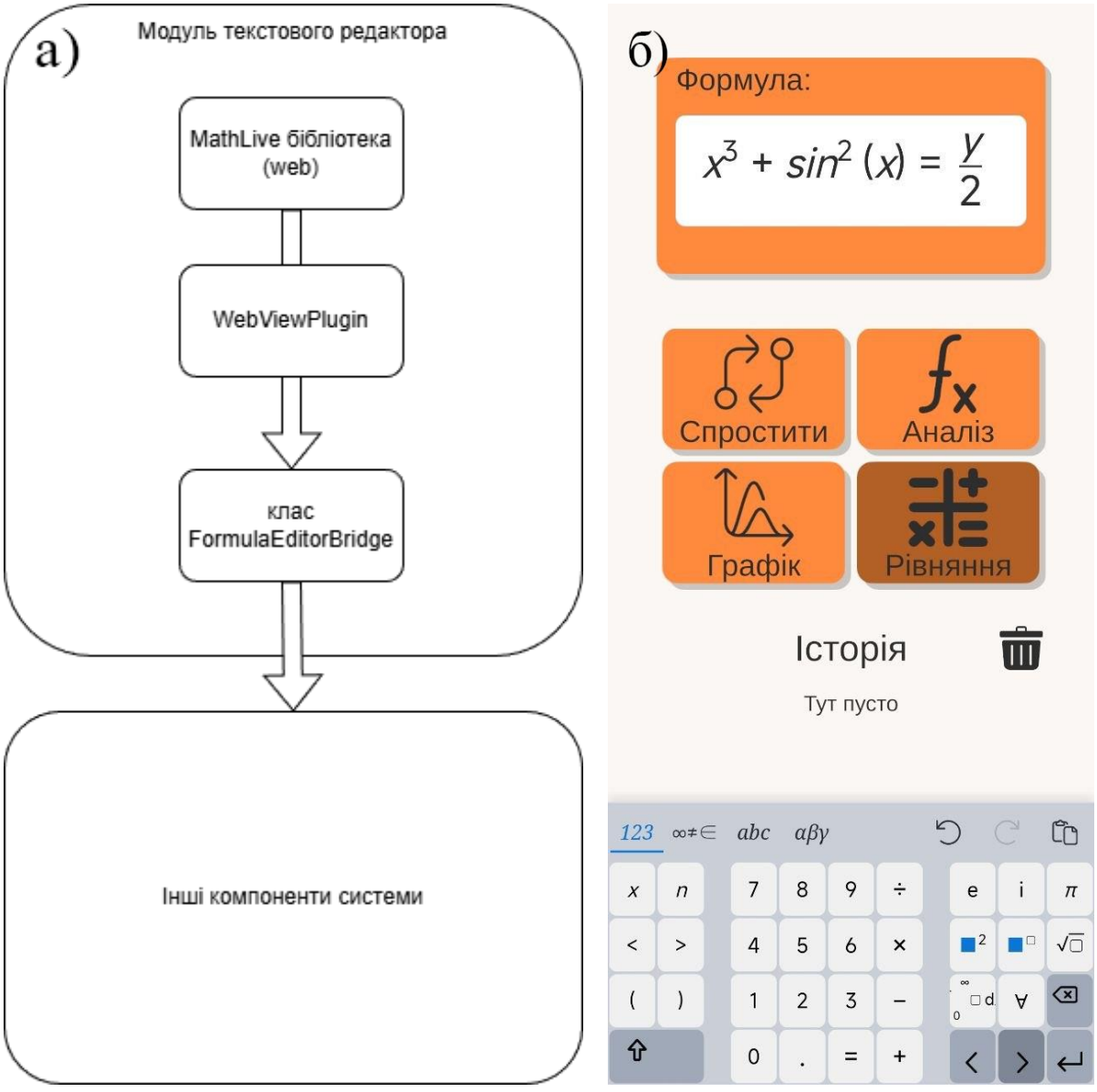

Рис. 2.7. Текстовий редактор математичних виразів: а) схема взаємодії; б) зовнішній вигляд

2.2.6. Бібліотека AngouriMath

Середовище Unity формує візуально-координаційний каркас додатка, математичні ж обчислення розробленого програмної системи виконує відкрита математична бібліотека AngouriMath. За своєю суттю, це повноцінна система комп'ютерної алгебри, написана мовою C#. Її інтеграція була необхідною для створеного продукту, оскільки класичні мови програмування здатні виконувати лише чисельні розрахунки, оперуючи готовими числами з плаваючою комою. Натомість специфіка розробленого додатка вимагає здатності обробляти математику на символьному рівні: система має рахувати змінні без конкретних значень, вміти розкривати дужки, знаходити похідні та зберігати ірраціональні корені у їхньому точному аналітичному вигляді без попереднього округлення до десяткових дробів.

Вибір саме AngouriMath серед інших існуючих математичних рушіїв був зумовлений кількома критичними факторами. По-перше, абсолютна нативна сумісність із платформою .NET дозволила імпортувати цю складну систему як звичайну динамічну бібліотеку (DLL) безпосередньо у проєкт Unity. Це виключає необхідність створення складних мостів до бібліотек на C++ чи Python, гарантуючи найвищу швидкість взаємодії між об'єктами сцени та математичним процесором. По-друге, AngouriMath забезпечує повну автономність роботи додатка: всі символьні обчислення, незалежно від їхньої складності, виконуються локально на процесорі смартфона. Користувачу не потрібне підключення до інтернету для відправки запитів на віддалений сервер, що є величезною перевагою для програмної системи мобільного пристрою.

Практичне застосування AngouriMath у додатку охоплює всі етапи життєвого циклу обробки даних. Процес розпочинається з етапу валідації введеного значення: коли центральний координатор отримує від MathLive формалізований текстовий рядок, він передає його до методу MathS.FromString(). На цьому етапі рушій виконує лексичний аналіз тексту і будує з нього дерево абстрактного синтаксису – ієрархічну програмну структуру, де кожен вузол представляє певну математичну сутність (базо-

вий клас Entity у термінології бібліотеки), таку як змінна, константа, оператор чи функція. В класі InputCheck, функції CheckOnMath() результати цього аналізу використовуються для валідації введеного користувачем виразу:

```
private bool CheckOnMath()
{
    if (string.IsNullOrWhiteSpace(currentInput)) return false;

    string[] parts = currentInput.Split(';');

    foreach (string part in parts)
    {
        if (string.IsNullOrWhiteSpace(part)) continue;

        try
        {
            Entity parsedEntity = MathS.FromString(part);
            if (parsedEntity == null) return false;
        }
        catch (Exception)
        {
            return false;
        }
    }

    return true;
}
```

Другим, і найбільш складним рівнем використання рушія, є безпосереднє виконання алгебраїчних перетворень. Для модуля спрощення виразів викликаються внутрішні алгоритми .Simplify(), які проходять по розкладеному на компоненти виразу і застосовують до нього набір математичних аксіом і правил: зведення подібних доданків, скорочення дробів, застосування тригонометричних тотожностей. Для модуля аналітичного розв'язання використовується функціонал .Solve(), який здатний алгебраїчно ізолювати задану змінну і повернути точну множину коренів (зокрема комплексних). Окремої уваги заслуговує роль AngouriMath у роботі підсистем геометричної візуалізації (2D та 3D графіків). Оскільки для плавної побудови графічної сітки потрібно обчислити значення функції у тисячах дискретних точок щосекунди, пряма символьна підстановка значень у вираз була б занадто повільною. Для вирішення цієї проблеми використано вбудований у бібліотеку механізм компіляції на льоту. За до-

помогою методу .Compile() рушій бере символьний вираз і перетворює його на оптимізований машинний делегат C#. Цей скомпільований код виконує математичні операції з такою ж швидкістю, як якби формула була жорстко прописана ще до запуску додатка. Таке поєднання символьної алгебри для аналізу та нативної швидкості для розрахунку сітки дозволив досягти ідеальної плавності рендерингу графіків навіть на мобільних пристроях із низькою обчислювальною потужністю.

Висновки до 2 розділу

У другому розділі було реалізовано програмну автоматизовану систему – мобільний додаток, що охоплює комплекс робіт від побудови архітектурного каркаса до безпосередньої програмної реалізації та імплементації всіх ключових модулів.

Фундаментом створеної системи стала спроєктована та впроваджена модульна архітектура, яка забезпечила гнучкість, стійкість до помилок та надійну ізоляцію обчислювальних процесів від роботи графічного інтерфейсу. На базі цієї архітектури було фізично реалізовано логіку роботи ключових підсистем. Для безпечного прийому та маршрутизації даних написано та інтегровано модулі текстового редактора, верифікації вводу та координації компонентів. Обчислювальне та візуальне ядро системи сформовано шляхом програмної реалізації модулів спрощення математичних виразів, аналітичного розв'язання рівнянь, параметричного аналізу та обчислення невідомих величин, а також підсистем двовимірної та тривимірної графічної візуалізації.

Практичне втілення проєкту здійснено на базі ігрового рушія Unity з використанням об'єктно-орієнтованої мови програмування C#. У процесі розробки було вибудовано чітку структурну організацію проєкту в середовищі Unity та застосовано конкретні алгоритмічні методи оптимізації роботи рушія, що гарантували високу продуктивність системи на ресурсах мобільного смартфона. Для забезпечення повноцінної роботи додатка успішно виконано інтеграцію зовнішніх програмних інструментів.

Зокрема, налаштовано міст зв'язку через WebViewPlugin для вбудовування бібліотеки MathLive, що дозволило реалізувати зручне візуальне введення багатошарових математичних формул, імплементовано систему комп'ютерної алгебри AngouriMath для виконання складних символьних обчислень локально на пристрої.

Таким чином, результатом виконання даного розділу стало створення повноцінної та цілісної кодової бази застосунку. Усі розроблені підсистеми були успішно інтегровані в єдину інформаційну систему, яка фізично реалізує закладену логіку математичних функцій («Спростити», «Аналіз», «Графік» і «Рівняння»).

РОЗДІЛ 3

ДЕМОНСТРАЦІЯ ФУНКЦІОНАЛУ

Для підтвердження практичної працездатності створеної програмної системи та оцінки ефективності алгоритмічних рішень було проведено серію тестувань. У даному підрозділі продемонстровано результати роботи системи на прикладі обробки тестової вибірки різнотипних математичних задач. Демонстрація охоплює повний цикл користувацької взаємодії: від введення початкових виразів за допомогою візуального редактора до отримання фінального результату у вигляді спрощених формул, аналітичних розв'язань та побудованих двовимірних і тривимірних графіків. Наведені нижче приклади ілюструють коректність відпрацювання внутрішньої логіки додатка, адекватність реакції графічного інтерфейсу та загальну стабільність координації обчислювальних модулів в умовах реального використання.

3.1. Функція «Спростити»

Результатом виконання функції має бути спрощений математичний вираз, тотожний початковому.

Приклад 1. Початковий вираз: $\frac{1}{2} + \frac{1}{3} - \frac{1}{6}$ (рис. 3.1а).

Очікуваний результат: $\frac{2}{3}$.

Отриманий результат: $\frac{2}{3}$ (рис. 3.1б).

Час виконання: 1,2 с.

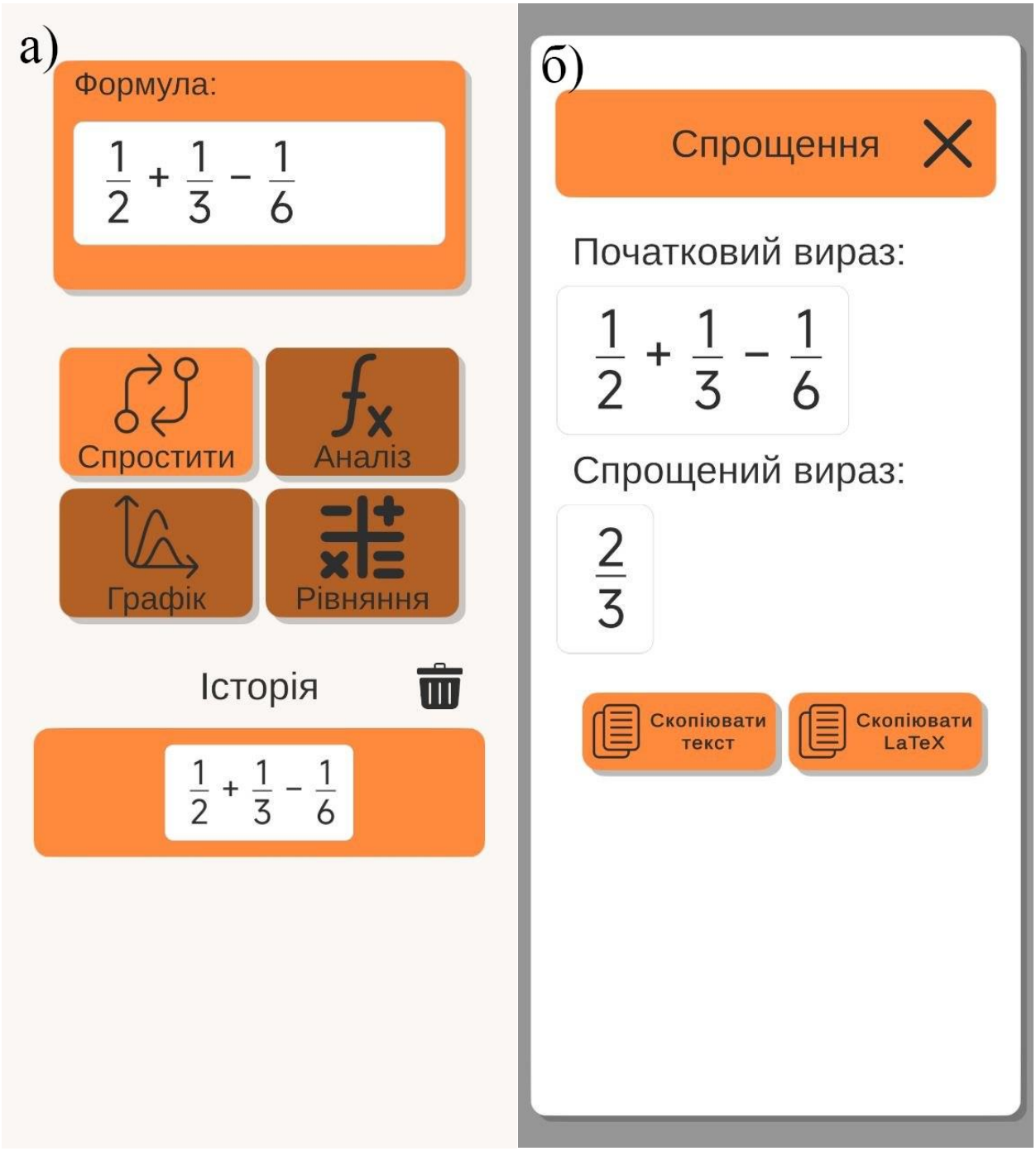

Рис. 3.1. Результат виконання функції «Спростити»: а) введення в додаток математичного виразу $\frac{1}{2}+\frac{1}{3}-\frac{1}{6}$; б) результат спрощення для введеного виразу

Приклад 2. Початковий вираз: $3x + 5y - x + 2y$ (рис. 3.2а).

Очікуваний результат: $2x + 7y$.

Отриманий результат: $2x + 7y$ (рис. 3.2б).

Час виконання: 0,8 с.

Рис. 3.2. Спрощення другого тестового виразу: а) введення в додаток математичного виразу $3x + 5y - x + 2y$; б) результат спрощення введеного виразу

Приклад 3. Початковий вираз: $(x - 2)(x + 2) + 4$ (рис. 3.3а).

Очікуваний результат: x^2.

Отриманий результат: x^2 (рис. 3.3б).

Час виконання: 1,8 с.

Рис. 3.3. Спрощення третього тестового виразу: а) введення в додаток математичного виразу $(x - 2)(x + 2) + 4$; б) результат спрощення введеного виразу

Приклад 4. Початковий вираз: $sin(x)^2 + cos(x)^2 + x$ (рис. 3.4а).

Очікуваний результат: $x + 1$.

Отриманий результат: $1 + x$ (рис. 3.4б).

Час виконання: 1,5 с.

Рис. 3.4. Спрощення четвертого тестового виразу: а) введення в додаток математичного виразу $sin(x)^2 + cos(x)\text{^}2 + x$; б) результат спрощення введеного виразу

Як свідчать результати тестування, модуль спрощення математичних виразів функціонує коректно, а отриманий вивід програми повністю збігається з очікуваними значеннями. Алгоритми успішно розпізнають структуру введених формул та безпомилково застосовують відповідні правила алгебраїчних перетворень. На основі цього можна зробити висновок щодо правильності та стабільності роботи даної функції.

3.2. Функція «Аналіз»

Результатом виконання даної функції має бути аналітичне представлення шуканої змінної, виражене з вихідного рівняння. Проте, для коректної роботи математичного аналізатора та уникнення неоднозначностей під час розрахунків, користувач має дотримуватися специфічного синтаксису введення.

Логіка роботи алгоритму аналітичного розв'язання безпосередньо залежить від кількості змінних у заданому виразі. Якщо користувач вводить рівняння лише з однією невідомою змінною (наприклад, $2x + 4 = 0$), обчислювальний рушій здатен самостійно ідентифікувати її як цільову та автоматично знайти розв'язок відносно неї. Однак, додаток спроєктовано таким чином, щоб ефективно обробляти і складніші функції багатьох змінних (наприклад, $3x - 4y = 12$). Саме для створення такого функціоналу у системі передбачено використання спеціального синтаксичного розділювача – крапки з комою «;». Після введення основного рівняння користувач може поставити цей символ і вказати цільову змінну (формат введення виглядатиме як «$3x - 4y = 12;\ y$ »). Цей маркер виступає командою рушію, що розв'язок має бути знайдений виключно відносно змінної «y», тоді як усі інші літерні символи у виразі (у даному випадку «x») повинні оброблятися алгоритмом як звичайні числові параметри або константи. Якщо для виразу з декількома змінними не вказано конкретної, відносно якої необхідно отримати результат, алгоритм розв’яже вираз відносно першої змінної, яку зустріне у виразі у порядку введення.

Нижче представлено перевірку виконання операцій даної функції за декількома тестовими прикладами.

Приклад 1. Початковий вираз: $2x + 4 = 0$ (рис. 3.5а).

Очікуваний результат: $x = -2$.

Отриманий результат: $x = -2$ (рис. 3.5б).

Час виконання: 1,2 с.

Рис. 3.5. Вирішення відносно змінної першого тестового виразу: а) введення в додаток математичного виразу $3x - 4y = 12;\ y$; б) результат аналізу для введеного виразу

Приклад 2. Початковий вираз: $3x - 4y = 12;\ y$ (рис. 3.6а).

Очікуваний результат: $y = \frac{3x-12}{4}$.

Отриманий результат: $y = \frac{3}{4}x - 3$ (рис. 3.6б).

Час виконання: 1,6 с.

Рис. 3.6. Вирішення відносно змінної другого тестового виразу: а) введення в додаток математичного виразу $3x - 4y = 12;\ y$; б) результат аналізу введеного виразу

Приклад 3. Початковий вираз: $P * V = n * R * T;\ T$ (рис. 3.7а).

Очікуваний результат: $T = \frac{P * V}{n * R}$.

Отриманий результат: $T = \frac{PV}{nR}$ (рис. 3.7б).

Час виконання: 1,5 с.

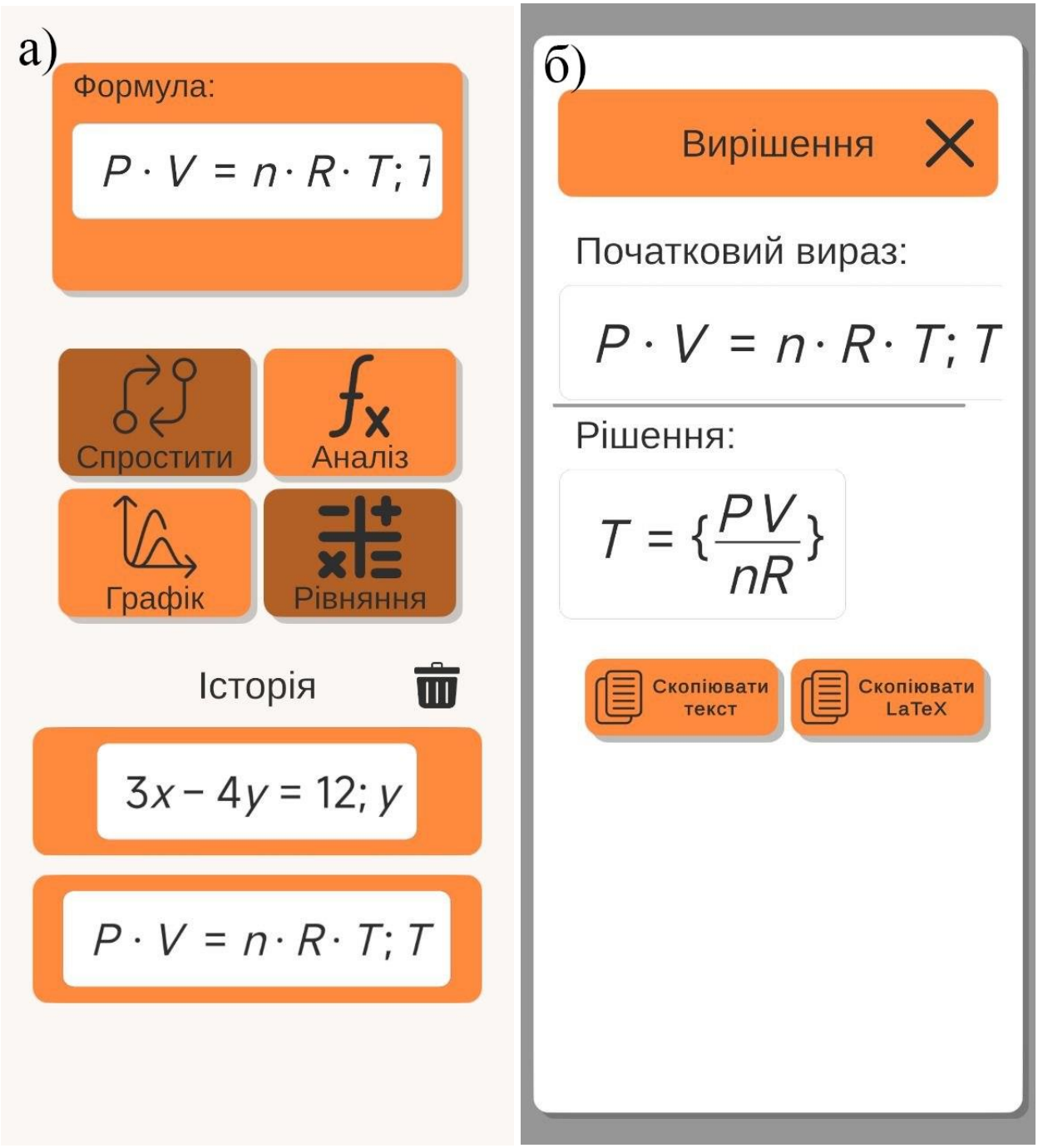

Рис. 3.7. Вирішення відносно змінної третього тестового виразу: а) введення в додаток математичного виразу $P * V = n * R * T;\ T$; б) результат аналізу введеного виразу

Приклад 4. Початковий вираз: $x^2 + y^2 = 25;\ x$ (рис. 3.8а).

Очікуваний результат: $x = \sqrt{25 - y^2}$, $x = -\sqrt{25 - y^2}$.

Отриманий результат: $x = i\sqrt{y^2 - 25},\ x = -i\sqrt{y^2 - 25}$ (рис. 3.8б).

Час виконання: 2 с.

Рис.3.8. Вирішення відносно змінної четвертого тестового виразу: а) введення в додаток математичного виразу $x^2 + y^2 = 25;\ x$; б) результат аналізу введеного виразу

Приклад 5. Початковий вираз: $s = vt + \frac{(at^2)}{2};\ a$ (рис. 3.9а).

Очікуваний результат: $a = \frac{2(s-tv)}{t^2}$.

Отриманий результат: $a = \frac{2(s-tv)}{t^2}$ (рис. 3.9б).

Час виконання: 2 с.

Рис. 3.9. Вирішення відносно змінної п’ятого тестового виразу: а) введення в додаток математичного виразу $s = vt + \frac{(at^2)}{2}; a$; б) результат виконання аналізу введеного виразу

Із результатів тестування функції можна зробити висновок, що модуль аналітичного розв'язання рівнянь функціонує коректно, успішно справляючись із задачею ізоляції шуканої змінної. Здебільшого згенеровані програмою відповіді повністю збігаються з очікуваннями. Проте в окремих випадках фінальний результат візуально

відрізнявся від класичного формату запису (приклад 2 та 4). Такі розбіжності не є програмними дефектами, а безпосередньо зумовлені внутрішньою специфікою роботи системи комп'ютерної алгебри AngouriMath. Будь який математичний вираз можна представити безліччю форм, і алгоритм не може точно знати, яка із них зовнішньо є простішою. У обох прикладах результатом обчислень є вираз, еквівалентний до очікуваного. У прикладі 2 він представлений іншою, але однаково спрощеною формою, а от у прикладі 4 можна побачити комплексні числа, які не є необхідними для представлення рішення. Алгоритм вирішив, що комплексне число на початку виразу є простішим, ніж множення змінної на мінус. З цим можна посперечатись, проте з фундаментальної математичної точки зору обидва варіанти запису є абсолютно правильними. З огляду на це можна зробити обґрунтований висновок, що попри певні візуальні особливості машинного форматування, логіка роботи функції аналізу є математично правильною, стабільною та повною мірою виконує поставлене завдання з аналітичного розв'язання виразів.

3.3. Функція «Графік»

Результатом виконання має стати вікно з інтерактивним графіком заданої функції або системи виразів, де користувачу надається повний набір інструментів для візуального дослідження. У режимі двовимірної графіки на площині мають автоматично візуалізуватися ключові аналітичні вузли, а саме точки перетину графіка з осями координат та точки взаємного перетину кількох кривих. Для зручної ідентифікації систем функцій має відображатися легенда, що зіставляє кожну лінію з її вихідним виразом за допомогою унікального кольору. При натисканні на будь-яку точку графіка мають відобразитись її координати.

При тестуванні функціоналу побудови нерівностей очікується специфічна візуальна реакція системи. Мають чітко відрізнятись типи граничної кривої: програма має

відмальовувати суцільну лінію для нестрогих відношень (<, >) та пунктирну для строгих (≤, ≥). Множина розв'язків нерівності повинна відображатися у вигляді напівпрозорого затінення відповідної частини площини. У випадку обробки системи нерівностей має візуально ідентифікуватись спільна зону розв'язку як область геометричного перетину цих затінених шарів.

У режимі тривимірної графіки тестується взаємодія з просторовими моделями. Очікується, що користувач отримає доступ до відображеної об'ємної поверхні, за допомогою жестів має бути можливо обертати модель у всіх площинах, змінювати кут огляду та масштабувати фігуру для детального вивчення її рельєфу.

Перевірка функціоналу наведено нижче за декількома тестовими прикладами.

Приклад 1. Початковий вираз:$y = \frac{(x^2 - 4)}{x}$ (рис. 3.10а).

Очікуваний результат: 2 гілки гіперболи, що не перетинають вісь y.

Отриманий результат показано на рис. 3.10б.

Час виконання: 1,5с.

Приклад 2. Початковий вираз: $y \leq 4 - x^2;\ y > x$ (рис. 3.11а).

Очікуваний результат: перевернута парабола з правильним відображенням множини розв’язків з внутрішньої сторони, піднята над віссю x, та лінія, що проходить через центр координат, множина розв’язків зверху.

Отриманий результат показано на рис. 3 11б.

Час виконання: 2,2с.

Приклад 3. Початковий вираз: $x^2 + y^2 = 25; x$ (рис. 3.12а).

Очікуваний результат: Коло в центрі координат та лінія, що пересікає його та проходить крізь центр координат.

Отриманий результат показано на рис. 3.12б.

Час виконання: 2,1с.

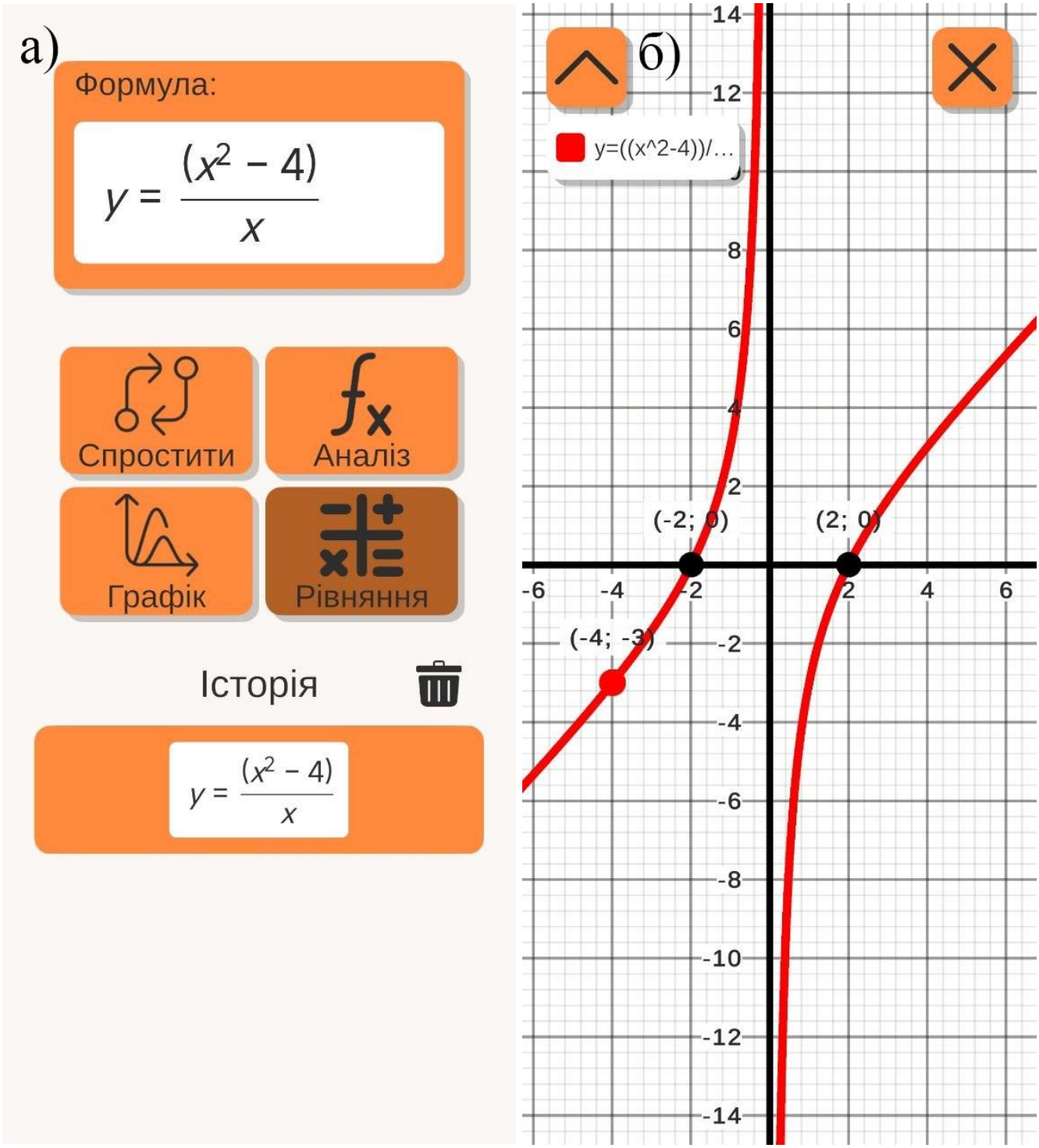

Рис. 3.10. Побудова графіка функції для першого тестового виразу: а) введення в додаток математичного виразу $y = \frac{(x^2 - 4)}{x}$; б) результат побудови графіка введеного виразу

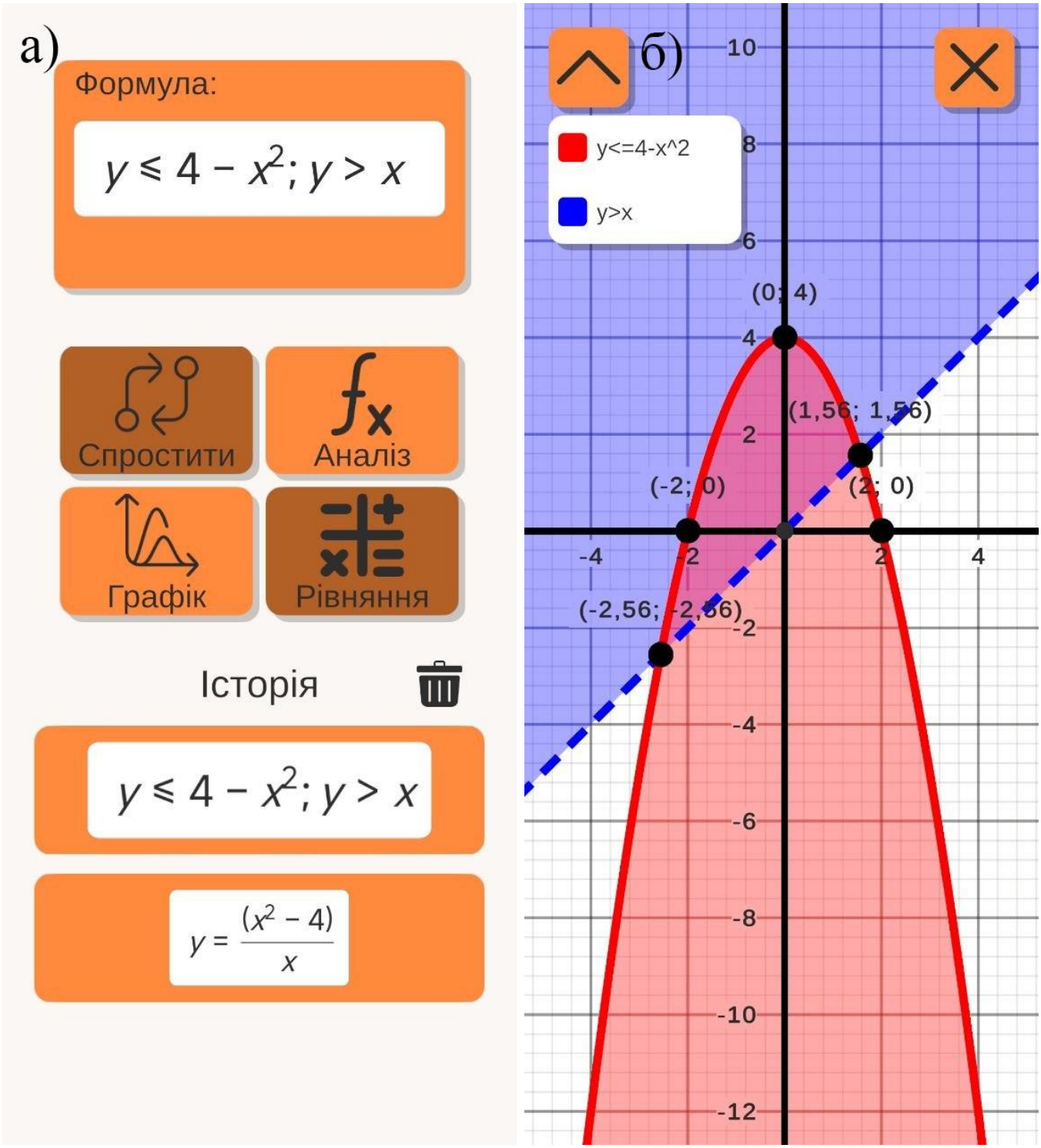

Рис. 3.11. Побудова графіка функції для другого тестового виразу: а) введення в додаток математичного виразу $y \leq 4 - x^2;\ y > x$; б) результат побудови графіка введеного виразу

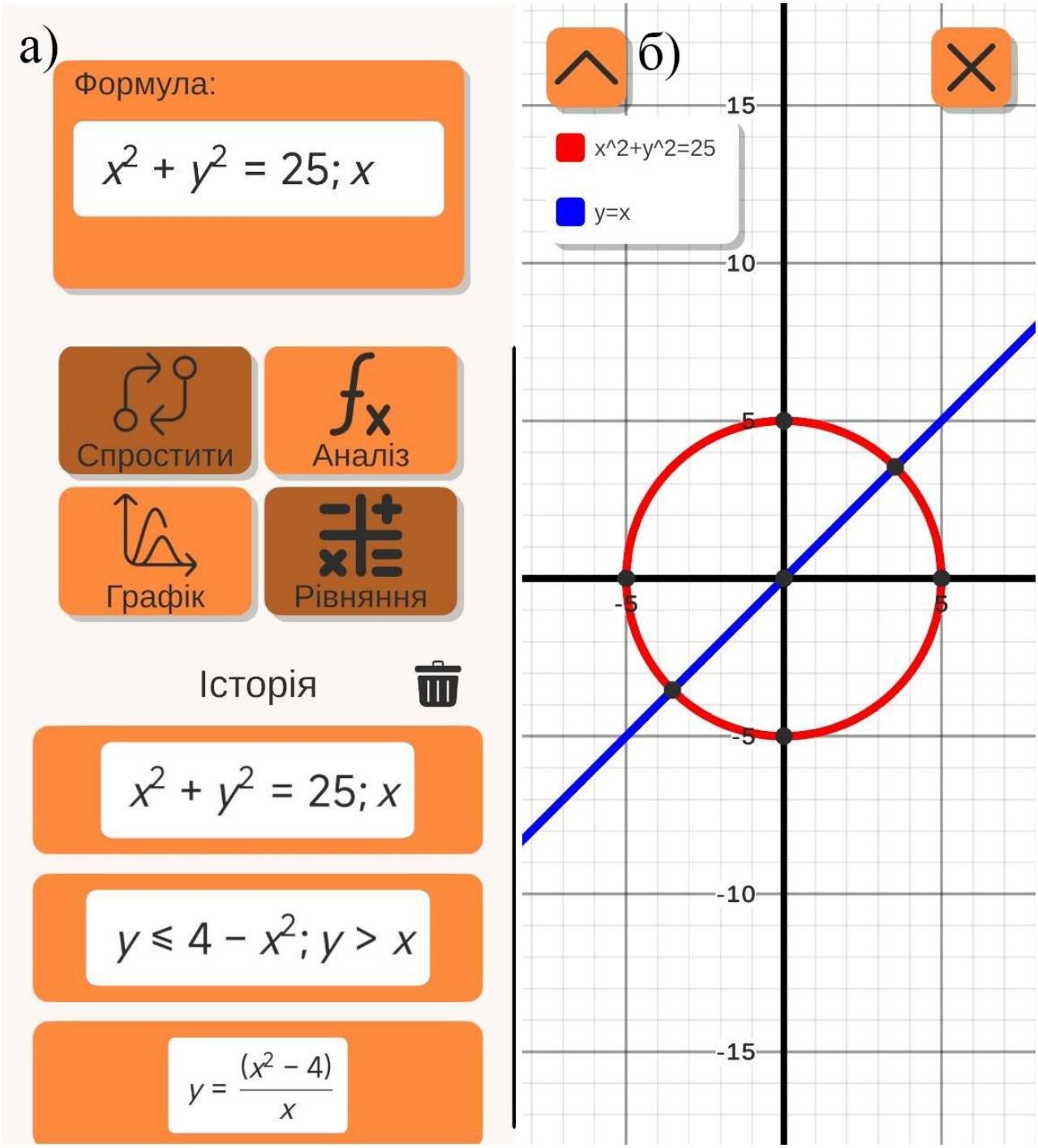

Рис. 3.12. Побудова графіка функції для третього тестового виразу: а) введення в додаток математичного виразу $x^2 + y^2 = 25; x$; б) результат побудови графіка введеного виразу

Приклад 4. Початковий вираз: $z = sin(\sqrt{(x^2 + y^2)})$ (рис. 3.13а).

Очікуваний результат: Тривимірна площина, схожа на хвилі на воді.

Отриманий результат показано на рис. 3.13б.

Час виконання: 1,5с.

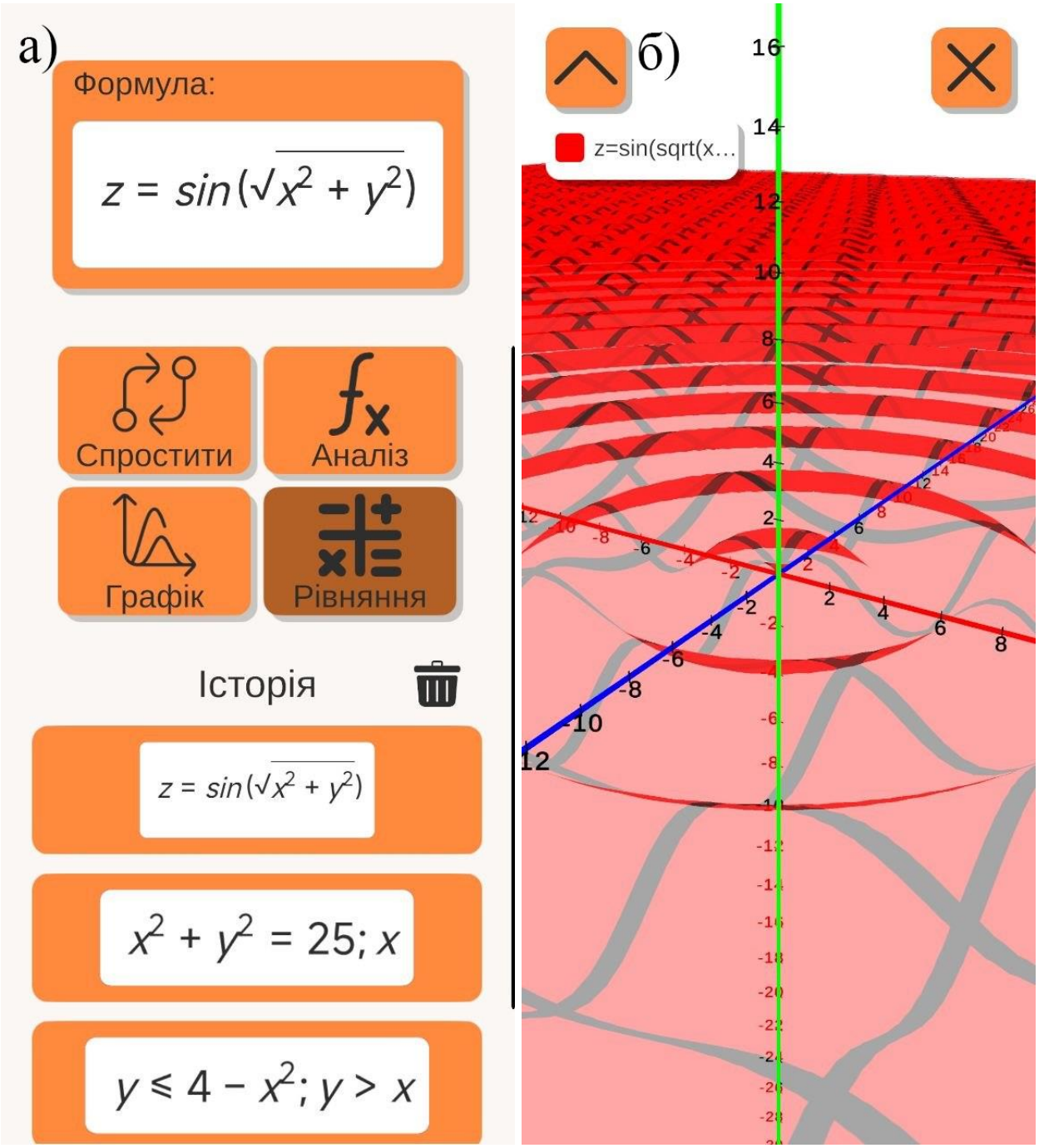

Рис. 3.13. Побудова графіка функції для четвертого тестового виразу: а) введення в додаток математичного виразу $z = sin(\sqrt{(x^2 + y^2)})$; б) результат побудови графіка введеного виразу

Приклад 5. Початковий вираз: $z = x^2 - y^2$ (рис. 3.14а).

Очікуваний результат: Тривимірна площина, схожа на сідло.

Отриманий результат показано на рис. 3.14б.

Час виконання: 1,5с.

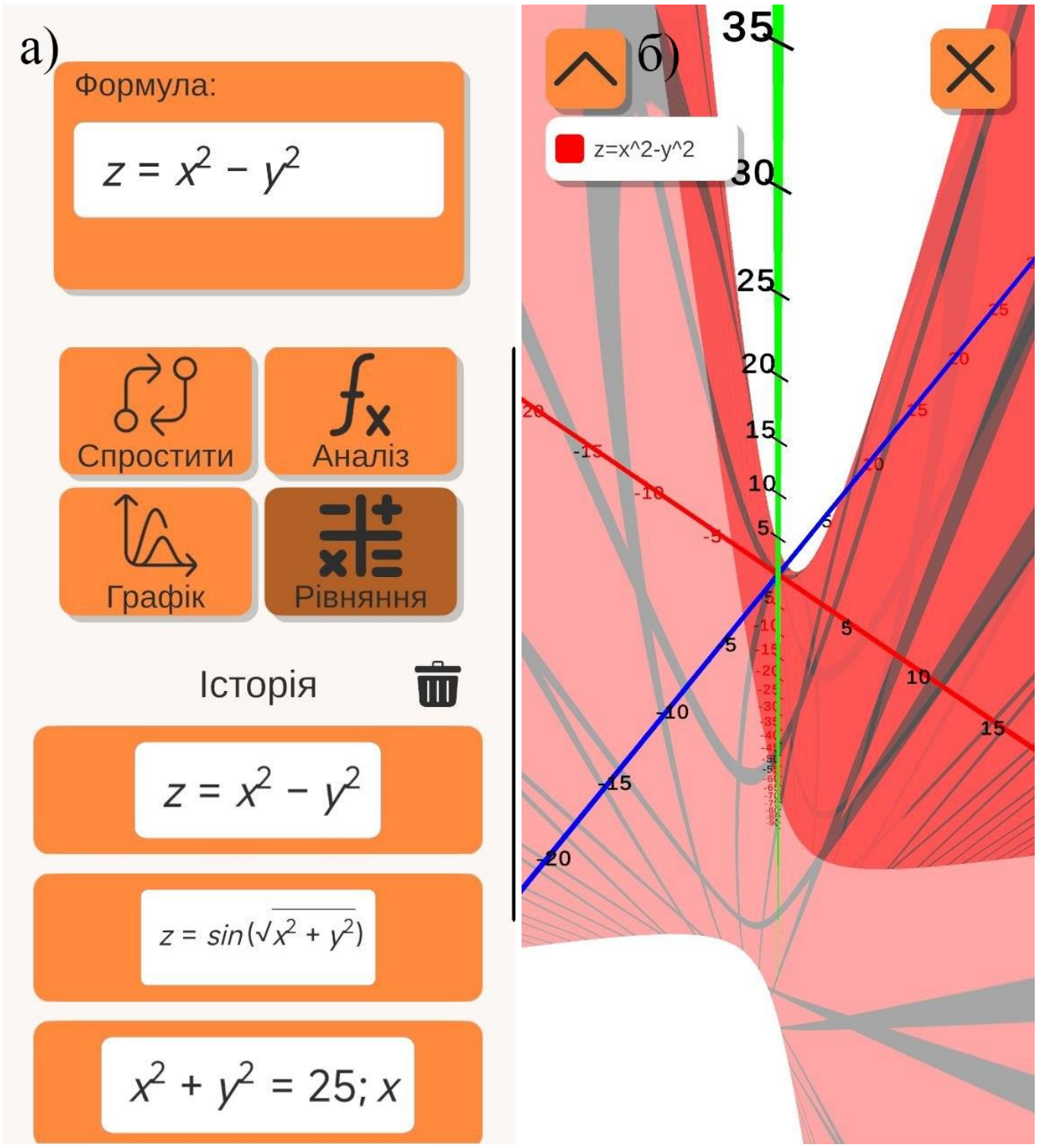

Рис. 3.14. Побудова графіка функції для п’ятого тестового виразу: а) введення в додаток математичного виразу $z = x^2 - y^2$; б) результат побудови графіка введеного виразу

За результатами проведеного тестування модуля геометричної візуалізації можна зробити висновок про його стабільну роботу та повну відповідність поставленим

вимогам. Перевірка двовимірного режиму підтвердила, що програма коректно будує лінії заданих функцій, точно обчислює та маркує критичні точки перетину, а також забезпечує миттєвий відгук усіх інструментів інтерактивної взаємодії. Алгоритм побудови нерівностей успішно візуалізував області розв'язку та граничні лінії. Тривимірні площини відображаються без геометричних артефактів та плавно реагують на сенсорні жести обертання і масштабування. Отже, розроблений графічний модуль успішно виконує всі визначені завдання, надаючи користувачеві надійний та зручний інструментарій для візуального дослідження математичних функцій.

3.4. Функція «Рівняння»

Результатом виконання розрахунку має бути кінцеве значення єдиної шуканої змінної шляхом підстановки числових параметрів у вхідний вираз.

Для ініціалізації розрахунку необхідно дотримуватися строгого синтаксису: спочатку вводиться базова формула, після чого через крапку з комою ініціалізуються відомі змінні. Наприклад, для розрахунку відстані формат запису матиме вигляд: $S = v \cdot t; v = 5; t = 10$. Під час тестування перевіряється здатність системи коректно розпізнати цей запис, виконання алгебраїчної підстановки та отримання рішення. Очікуваним результатом є виведення на екран окремого вікна з фінальним значенням цієї змінної.

Аналогічно до попередніх функцій, нижче представлено перевірку функціоналу на тестових прикладах.

Приклад 1. Початковий вираз: $P = 2\,(a + b);\ P = 24;\ a = 5$ (рис. 3.15а).

Очікуваний результат: $b = 7$.

Отриманий результат: $b = 7$ (рис. 3.15б).

Час виконання: 1,5 с.

Рис. 3.15. Знаходження значення невідомої змінної першого тестового виразу: а) введення в додаток математичного виразу $P = 2\,(a + b);\ P = 24;\ a = 5$; б) результат знаходження значення невідомої змінної для введеного виразу

Приклад 2. Початковий вираз: $U = IR;\ U = 12;\ I = 2$ (рис. 3.16а).

Очікуваний результат: $R = 6$.

Отриманий результат: $R = 6$ (рис. 3.16б).

Час виконання: 1,4 с.

Рис. 3.16. Знаходження значення невідомої змінної другого тестового виразу: а) введення в додаток математичного виразу $U = IR;\ U = 12;\ I = 2$; б) результат знаходження значення невідомої змінної для введеного виразу

Приклад 3. Початковий вираз: $E = \frac{(m * v^2)}{2};\ E = 400;\ m = 8$ (рис. 3.17а).

Очікуваний результат: $v = 10, -10$.

Отриманий результат: $v = 10, -10$ (рис. 3.17б).

Час виконання: 1,4 с.

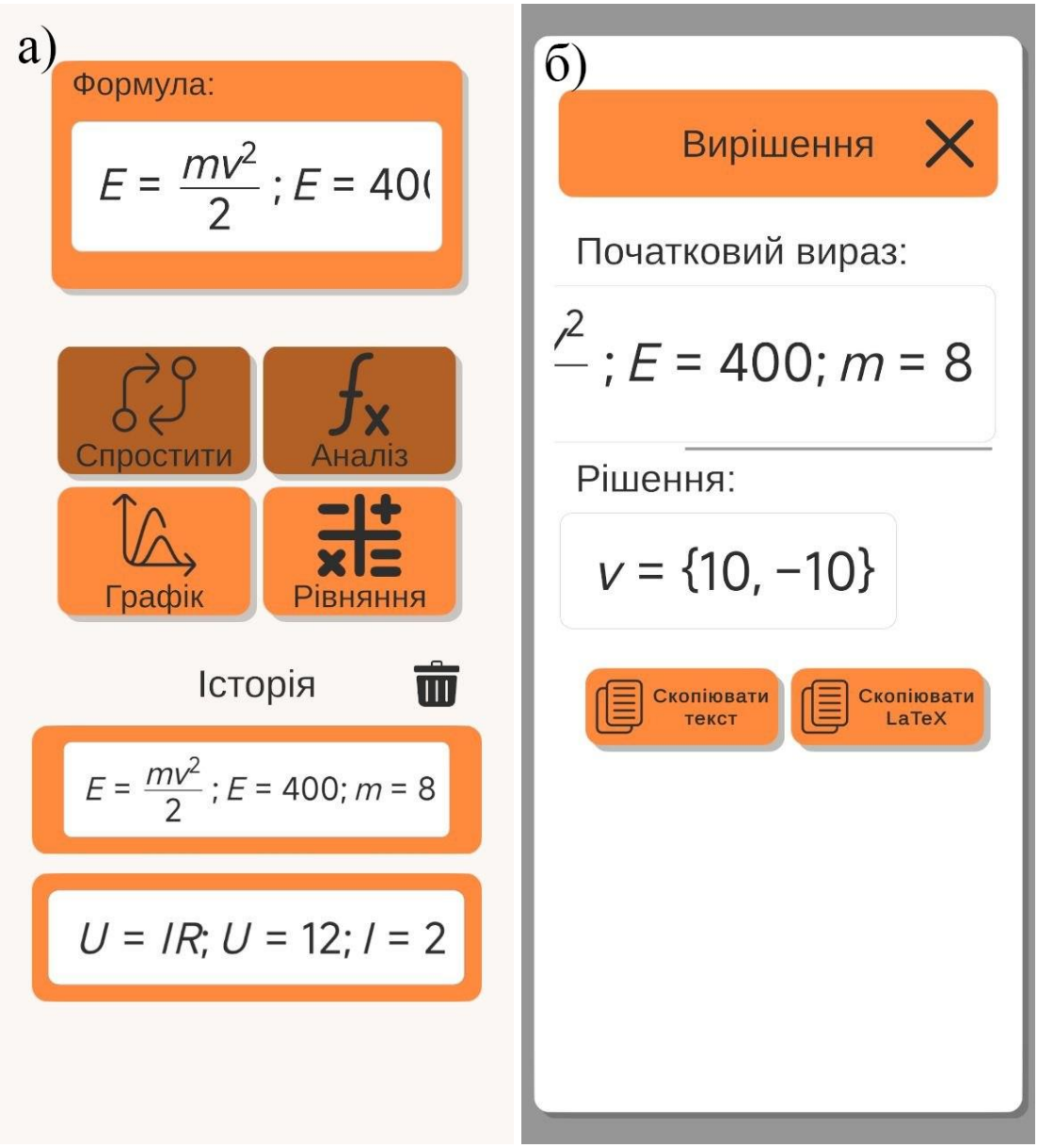

Рис. 3.17. Знаходження значення невідомої змінної третього тестового виразу: а) введення в додаток математичного виразу $E = \frac{(m * v^2)}{2}$; $E = 400$; $m = 8$; б) результат знаходження значення невідомої змінної для введеного виразу

Тестування функції «Рівняння» підтвердило її коректну та стабільну роботу. Програмний модуль безпомилково розпізнає синтаксис користувацького вводу, відділяючи базову формулу від блоку ініціалізованих змінних, та виконує їх точну алгебраїчну підстановку. Механізм валідації надійно блокує некоректні запити, а у всіх

правильних тестових сценаріях алгоритм успішно обчислював єдину шукану величину та виводить на екран правильний фінальний результат, що повністю доводить працездатність даного функціоналу.

3.5. Компіляція та формування інсталяційного пакета додатка

Після успішного завершення етапу комплексного тестування всіх функціональних модулів фінальним кроком розробки стала підготовка програмного продукту до розгортання та встановлення на кінцеві пристрої користувачів. Цільовою платформою розробленого застосунку є операційна система Android, тому вихідний код, графічні матеріали та підключені бібліотеки було перетворено у єдиний портативний інсталяційний файл формату APK (Android Package) (рис. 3.18).

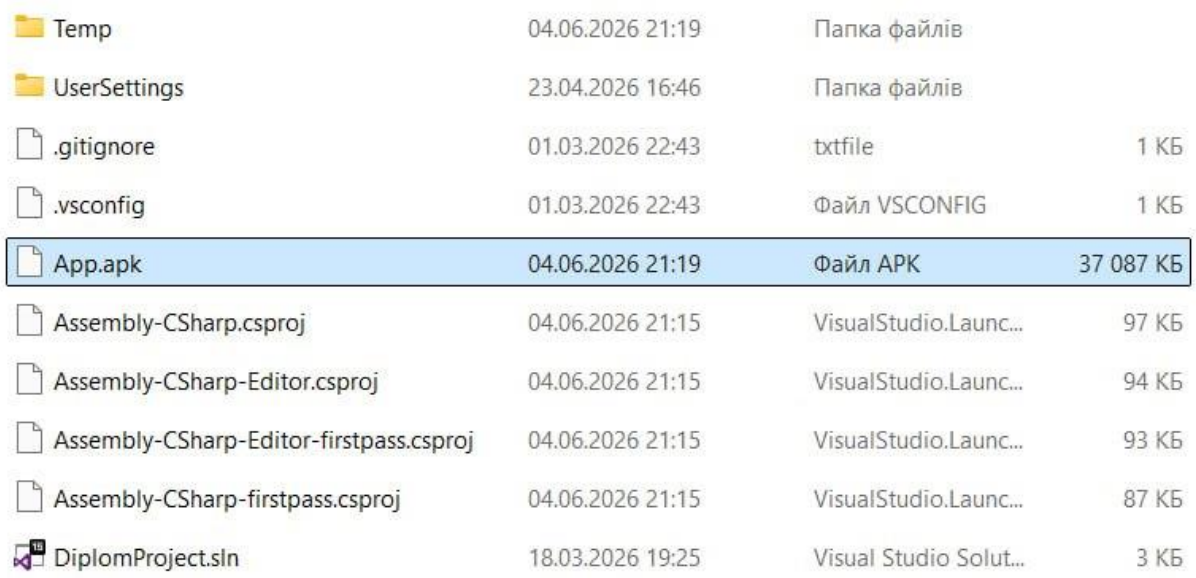

Рис. 3.18. Створений інсталяційний файл App.apk

Зовнішній вигляд встановленого додатка на робочому столі пристроя зображено на рис. 3.20.

Рис. 3.19. Зовнішній вигляд встановленого додатка на робочому столі пристроя

Процес компіляції здійснювався вбудованими інструментами середовища Unity. На етапі конфігурації проєкту було задано глобальні параметри додатка: унікальний ідентифікатор пакета, версію продукту та мінімально підтримувану версію API операційної системи Android, що гарантує широку сумісність із більшістю сучасних смартфонів. Для забезпечення максимальної продуктивності обчислювального ядра на мобільних процесорах як скриптовий бекенд було обрано сучасний стандарт

Unity – технологію IL2CPP. Вона автоматично перекладає C# код додатка у C++, після чого компілює його в нативний машинний код для архітектур ARMv7 та ARM64. Це не лише суттєво пришвидшує виконання важких математичних операцій, але й підвищує рівень захисту вихідного коду системи.

Особливу увагу під час підготовки до компіляції було приділено оптимізації розміру кінцевого пакета. Вагу фінального інсталяційного файлу вдалося довести до 37 мб (рис. 3.18), що для сучасних смартфонів є майже непомітним об’ємом пам’яті. Для досягнення цього показника було застосовано вбудовані алгоритми відсікання невикористовуваного коду, стиснення текстур графічного інтерфейсу та мінімізацію підключених зовнішніх модулів.

У результаті виконання процесу компіляції було згенеровано оптимізований самостійний інсталяційний пакет. Створений файл містить усі необхідні ресурси, скрипти, систему комп'ютерної алгебри AngouriMath та компоненти MathLive для локальної роботи. Розроблений додаток є повністю портативним, не потребує дозавантаження додаткових файлів чи наявності інтернет-з'єднання і готовий до безпосереднього встановлення та використання на мобільному носії користувача.

Висновки до 3 розділу

У третьому розділі було проведено комплексне тестування розробленого мобільного додатка та виконано фінальні етапи підготовки програмного продукту до розгортання на пристроях кінцевих користувачів.

Головною метою етапу тестування стала практична перевірка коректності, стабільності та швидкодії алгоритмів обчислювального і графічного ядра системи. У ході роботи було наочно продемонстровано та підтверджено безпомилкову взаємодію всіх підсистем на реальних вибірках математичних виразів. Зокрема, успішно верифіковано роботу ключових функцій: «Спростити» (виконання точних алгебраїчних перетворень), «Аналіз» (знаходження значення шуканої змінної аналітично), «Графік»

(динамічна процедурна побудова двовимірних і тривимірних геометричних моделей) та «Рівняння» (алгебраїчна підстановка та параметричний розрахунок невідомих величин). Результати перевірки засвідчили, що механізми верифікації вводу надійно відпрацьовують некоректні запити, а інтегрована математична система видає правильні результати.

Заключним етапом стало формування вихідного коду, графічних матеріалів та підключених бібліотек проєкту у самостійний інсталяційний файл для операційної системи Android. За допомогою інструментарію рушія Unity та скриптового бекенду IL2CPP було здійснено оптимізовану компіляцію програми в машинний код. Завдяки застосуванню алгоритмів відсікання невикористовуваного коду та стиснення ресурсів вдалося успішно виконати закладені нефункціональні вимоги – розмір сформованого інсталяційного пакета не перевищує 45 Мб.

Таким чином, результатом виконання даного розділу є всебічно перевірений, стабільний та оптимізований інсталяційний пакет (APK). Створений програмний продукт є повністю автономним, здійснює всі обчислення безпосередньо на апаратних потужностях смартфона і є абсолютно готовим до встановлення та практичного використання.

ВИСНОВКИ

В результаті виконання бакалаврської роботи було повністю досягнуто поставленої мети та виконано всі завдання, визначені технічним завданням. Робота охоплює повний цикл створення програмного продукту – від теоретичного дослідження предметної області до практичної реалізації та тестування готового мобільного додатка для виконання математичних обчислень.

Підсумовуючи результати проведеної роботи, можна зробити наступні висновки:

- Проведено огляд та аналіз актуальності проблеми. Доведено, що розробка мобільних систем комп'ютерної алгебри є актуальною задачею, і відповідає сучасним запитам користувачів.
- Виконано огляд існуючих рішень та підходів. Аналіз наявних на ринку програмних продуктів дозволив виявити їхні ключові переваги та недоліки, що стало базисом для проєктування вдосконаленого застосунку.
- Сформовано вимоги до застосунку. На основі проведеного аналізу було чітко визначено функціональні та нефункціональні вимоги до системи, які гарантували дотримання балансу між обчислювальною потужністю рушія та загальною оптимізацією для мобільних пристроїв.
- Розроблено архітектуру та структуру програмної реалізації. Застосовано строгий модульний підхід, який дозволив розділити систему на незалежні блоки (навігації та математичних розрахунків). Спроєктовано ключові підсистеми: модулі верифікації вводу, координації, спрощення, аналітичного розв'язання, параметричного аналізу та графічної візуалізації. Це забезпечило надійну ізоляцію складних математичних операцій від роботи графічного інтерфейсу.
- Реалізовано функціонал для введення формул, виведення результату та побудови графіків. У середовищі Unity з використанням мови C# було написано та інтегровано всі необхідні програмні скрипти. Завдяки імплементації бібліотеки

MathLive через технологію WebViewPlugin реалізовано інтуїтивно зрозуміле візуальне введення складних формул. Інтеграція системи комп'ютерної алгебри AngouriMath забезпечила точне аналітичне розв'язання та перетворення виразів. Створено потужну геометричну підсистему для побудови інтерактивних двовимірних і тривимірних графіків функцій та нерівностей.

- Описано функціональність та особливості системи. У ході роботи було проведено комплексне тестування розробленого функціоналу на практичних вибірках та підтверджено математичну точність і стабільність роботи всіх компонентів.
- Забезпечено можливість встановлення на носій користувача. Розроблений додаток успішно скомпільовано, оптимізовано під архітектуру мобільних платформ та підготовлено у вигляді самостійного інсталяційного пакета, що повністю відповідає вихідним вимогам до портативності програмного продукту.

Отже, розроблена автоматизована система є успішно завершеним програмним продуктом. Створений мобільний застосунок виступає повноцінним, автономним і надійним інструментом, що забезпечує користувача потужним функціоналом для виконання комплексних символьних обчислень, аналітичного аналізу рівнянь та детального геометричного моделювання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Історія розвитку обчислювальної техніки // Електронний підручник з Інформатики. URL: https://studyit1.blogspot.com/2017/08/3.html (дата звернення: 28.04.2026).

2. Підсумовуюча машина Паскаля // Galanix. URL: https://galanix.com/ua/articles/pidsumovuyucha-mashyna-paskalya (дата звернення: 29.04.2026).

3. Рахункова машина Лейбниця // Galanix. URL: https://galanix.com/ua/articles/rahunkova-mashyna-leybnycya (дата звернення: 29.04.2026).

4. Аналітична машина Беббіджа – прообраз комп'ютера // Galanix. URL: https://galanix.com/ua/articles/analitychna-mashyna-bebbidzha-proobraz-kompyutera (дата звернення: 29.04.2026).

5. Дрозд Ю. А. Теорія Галуа : навчальний посібник. Київ : РВЦ «Київський університет», 1997. 35 с. URL: https://imath.kiev.ua/~drozd/Galois.pdf (дата звернення: 10.05.2026).

6. Метод Рунге-Кутта Покроково: Як Він Обчислює Наближене Розв’язання? // Mathos. URL: https://www.mathros.net.ua/rozvjazuvannja-zvychajnyh-dyferencialnyh-rivnjan-metodom-runge-kutta.html (дата звернення: 10.05.2026).

7. ENIAC: що розраховував для військових перший електронний комп’ютер // Universe Magazine. URL: https://universemagazine.com/eniac-shho-rozrahovuvav-dlya-vijskovyh-pershyj-elektronnyj-kompyuter/?srsltid=AfmBOorMFyoRWBrlIUDjl5LJT8rHFKharyONGx8SS0hoBF-xx0wAaZrF (дата звернення: 10.05.2026).

8. Fortran High-performance parallel programming language // Fortran. URL: https://fortran-lang.org/ (дата звернення: 10.05.2026).

9. Мікропроцесор. Організація пам’яті // Informatics. URL: https://informatics.dp.ua/mikroprotsesor-orhanizatsiya-pamyati/ (дата звернення: 10.05.2026).

10. Стандарт 754-1985 – IEEE Standard for Binary Floating-Point Arithmetic // IEEE Xplore. URL: https://ieeexplore.ieee.org/document/30711 (дата звернення: 10.05.2026).

11. Nvidia CUDA // NVIDIA Developer. URL: https://developer.nvidia.com/cuda (дата звернення: 10.05.2026).

12. OpenCL // NVIDIA Developer. URL: https://developer.nvidia.com/opencl (дата звернення: 10.05.2026).

13. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge : MIT Press, 2016. 800 p. URL: https://pzs.dstu.dp.ua/DataMining/bibl/Deep%20Learning.pdf (дата звернення: 09.05.2026).

14. Von Neumann bottleneck // TechTarget WhatIs. URL: https://www.techtarget.com/whatis/definition/von-Neumann-bottleneck (дата звернення: 10.05.2026).

15. Штучний інтелект на стероїдах // De Novo. URL: https://denovo.ua/blog/tensore-core-gpu-for-ai (дата звернення: 10.05.2026).

16. Парадигми програмування // W3Schools. URL: https://w3schoolsua.github.io/hyperskill/paradigms.html#gsc.tab=0 (дата звернення: 10.05.2026).

17. Фреймворки у мовах програмування: навіщо служать та як їх вибрати // IT Step Academy. URL: https://cloud.itstep.org/blog/frameworks-in-programming-languages-what-are-they-for-and-how-to-choose-them (дата звернення: 10.05.2026).

18. Overview of ASP.NET Core // Microsoft Learn. URL: https://learn.microsoft.com/uk-ua/aspnet/core/overview?view=aspnetcore-10.0 (дата звернення: 10.05.2026).

19. Overview of Spring Boot // Spring. URL: https://spring.io/projects/spring-boot (дата звернення: 10.05.2026).

20. Бібліотеки машинного навчання: TensorFlow, PyTorch та Scikit-learn // Hostragons. URL: https://www.hostragons.com/uk/%D0%B1%D0%BB%D0%BE%D0%B3/%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%B5-%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F-tensorflow-pytorch/#TensorFlow_ve_PyTorch_Temel_Farklar (дата звернення: 10.05.2026).

21. Огляд переваг та недоліків ігрових рушіїв // ePrints ЖДУ ім. І. Франка. URL: https://eprints.zu.edu.ua/40326/1/4444.pdf (дата звернення: 10.05.2026).

22. Офіційний сайт продукту MATLAB // MathWorks. URL: https://www.mathworks.com/products/matlab.html (дата звернення: 30.04.2026).

23. Офіційний сайт продукту WolframAlpha // Wolfram|Alpha. URL: https://www.wolframalpha.com (дата звернення: 30.04.2026).

24. Офіційний сайт продукту Photomath // Photomath. URL: https://photomath.com/ (дата звернення: 30.04.2026).

25. Офіційний сайт продукту Desmos // Desmos. URL: https://www.desmos.com/calculator (дата звернення: 10.05.2026).

26. Що таке модульна архітектура продукту? // Visure Solutions. URL: https://visuresolutions.com/uk/%D0%BF%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA-%D0%B7-PLM/%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%B0-%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D1%83/ (дата звернення: 24.05.2026).

27. Unity Documentation // Unity. URL: https://docs.unity.com/en-us (дата звернення: 24.05.2026).

28. The Math Editor for the Web. MathLive Documentation // MathLive. URL: https://mathlive.io/mathfield/ (дата звернення: 24.05.2026).

29. Unity WebView Open Source Project // GitHub. URL: https://github.com/gree/unity-webview (дата звернення: 24.05.2026).

30. What is AngouriMath? // AngouriMath. URL: https://am.angouri.org/ (дата звернення: 24.05.2026).

31. Одинак. Також відомий як: Singleton // Refactoring.Guru. URL: https://refactoring.guru/uk/design-patterns/singleton (дата звернення: 24.05.2026).

32. LeanTween Documentation // Dented Pixel. URL: https://dentedpixel.com/LeanTweenDocumentation/classes/LeanTween.html (дата звернення: 24.05.2026).

ДОДАТОК А

Клас FormulaEditorBridge

```
using UnityEngine;
using TMPro;

public class FormulaEditorBridge : MonoBehaviour
{
    public string currentAsciiMath = "";
    public string currentLatexMath = "";
    public RectTransform containerTile;
    public Canvas mainCanvas;
    public event System.Action<Texture2D, float> OnEquationImageReceived;
    public event System.Action<Texture2D, float> OnResultImageReceived;

    private WebViewObject webViewObject;
    private bool isExpanded = false;
    private int[] lastMargins = new int[4] { -1, -1, -1, -1 };

    private string CleanMathInput(string input)
    {
        if (string.IsNullOrWhiteSpace(input)) return "";
        return input
            .Replace("a r c s i n", "arcsin")
            .Replace("a r c c o s", "arccos")
            .Replace("a r c t a n", "arctan")
            .Replace("s i n", "sin")
            .Replace("c o s", "cos")
            .Replace("t a n", "tan")
            .Replace("c o t", "cot")
            .Replace("l o g", "log")
            .Replace("l n", "ln")
            .Replace("s q r t", "sqrt");
    }
    void Start()
    {
        webViewObject = (new GameObject("WebViewObject")).AddComponent<WebViewObject>();

        webViewObject.Init(
            cb: (msg) =>
            {
                if (msg == "READY")
                {
                    ForceSync();
                    return;
                }

                if (msg == "EXPAND")
                {
                    isExpanded = true;
                    ForceSync();
                    return;
                }

                if (msg == "SHRINK")
                {
                    isExpanded = false;
                    ForceSync();
                    return;
                }
```

```
                if (msg == "CLOSE_EDITOR")
                {
                    webViewObject.EvaluateJS("if(window.mathVirtualKeyboard)  window.mathVirtualKeyboard.hide();
document.activeElement.blur();");

                    SetEditorVisibility(false);
                    return;
                }

                if (msg.StartsWith("LATEX:"))
                {
                    currentLatexMath = CleanMathInput(msg.Substring(6));
                    return;
                }

                if (msg.StartsWith("ASCII:"))
                {
                    currentAsciiMath = CleanMathInput(msg.Substring(6));
                    InputCheck.Instance.CheckForValidation(currentAsciiMath);
                    return;
                }

                if (msg.StartsWith("IMAGE:"))
                {
                    string payload = msg.Substring(6);
                    int firstColon = payload.IndexOf(':');
                    float scale = 2f;

                    if (firstColon > 0)
                    {
                        string scaleStr = payload.Substring(0, firstColon);
                        float.TryParse(scaleStr,  System.Globalization.NumberStyles.Float,  System.Globalization.Cul-
tureInfo.InvariantCulture, out scale);

                        string base64Full = payload.Substring(firstColon + 1);
                        string base64String = base64Full.Substring(base64Full.IndexOf(',') + 1);

                        byte[] imageBytes = System.Convert.FromBase64String(base64String);

                        Texture2D equationTexture = new Texture2D(2, 2, TextureFormat.RGBA32, false);
                        equationTexture.LoadImage(imageBytes);
                        equationTexture.filterMode = FilterMode.Bilinear;
                        equationTexture.anisoLevel = 1;

                        OnEquationImageReceived?.Invoke(equationTexture, scale);
                    }
                    return;
                }

                if (msg.StartsWith("RESULT_IMAGE:"))
                {
                    string payload = msg.Substring(13);
                    int firstColon = payload.IndexOf(':');
                    float scale = 2f;

                    if (firstColon > 0)
                    {
                        string scaleStr = payload.Substring(0, firstColon);
                        float.TryParse(scaleStr,  System.Globalization.NumberStyles.Float,  System.Globalization.Cul-
tureInfo.InvariantCulture, out scale);

                        string base64Full = payload.Substring(firstColon + 1);
                        string base64String = base64Full.Substring(base64Full.IndexOf(',') + 1);
```

```
                    byte[] imageBytes = System.Convert.FromBase64String(base64String);

                    Texture2D resultTexture = new Texture2D(2, 2, TextureFormat.RGBA32, false);
                    resultTexture.LoadImage(imageBytes);
                    resultTexture.filterMode = FilterMode.Bilinear;
                    resultTexture.anisoLevel = 1;

                    OnResultImageReceived?.Invoke(resultTexture, scale);
                }
                return;
            }
        },
        err: (msg) => Debug.LogError(msg),
        transparent: true,
        enableWKWebView: true
    );

    string url = "";
#if UNITY_EDITOR || UNITY_IOS
    url = "file://" + Application.streamingAssetsPath + "/editor.html";
#elif UNITY_ANDROID
    url = "file:///android_asset/editor.html";
#endif
    webViewObject.LoadURL(url);
    webViewObject.SetVisibility(true);

    ForceSync();
}

public void RequestEquationImage()
{
    if (webViewObject != null)
    {
        string jsCommand = "window.CaptureEquation();";
        webViewObject.EvaluateJS(jsCommand);
    }
}

public void ExecJsToEditor(string js)
{
    webViewObject.EvaluateJS(js);
}

public void RequestResultImage(string latex)
{
    if (webViewObject != null)
    {
        string jsCommand = $"window.RenderResultImage('{latex.Replace("\\", "\\\\").Replace("'", "\\'")}');";
        webViewObject.EvaluateJS(jsCommand);
    }
}

public void ResetKeyboardState()
{
    if (webViewObject != null)
    {
        string jsCommand = "if(window.mathVirtualKeyboard) window.mathVirtualKeyboard.hide(); document.activeElement.blur();";
        webViewObject.EvaluateJS(jsCommand);
    }
}
```

```
    void Update()
    {
        if (webViewObject != null)
        {
            SyncWebViewToContainer();
        }
    }

    private void ForceSync()
    {
        lastMargins[0] = -1;
        SyncWebViewToContainer();
    }

    private void SyncWebViewToContainer()
    {
        if (containerTile == null || mainCanvas == null || isOffscreen) return;

        Vector3[] corners = new Vector3[4];
        containerTile.GetWorldCorners(corners);

        Camera cam = mainCanvas.renderMode == RenderMode.ScreenSpaceOverlay ? null : mainCanvas.worldCamera;

        Vector2 bottomLeft = RectTransformUtility.WorldToScreenPoint(cam, corners[0]);
        Vector2 topLeft = RectTransformUtility.WorldToScreenPoint(cam, corners[1]);
        Vector2 topRight = RectTransformUtility.WorldToScreenPoint(cam, corners[2]);

        int leftMargin = Mathf.RoundToInt(bottomLeft.x);
        int topMargin = Mathf.RoundToInt(Screen.height - topLeft.y);
        int rightMargin = Mathf.RoundToInt(Screen.width - topRight.x);
        int bottomMargin = isExpanded ? 0 : Mathf.RoundToInt(bottomLeft.y);

        if (leftMargin != lastMargins[0] || topMargin != lastMargins[1] || rightMargin != lastMargins[2] || bottomMargin
!= lastMargins[3])
        {
            webViewObject.SetMargins(0, topMargin, 0, bottomMargin);

            string js = $"if(window.SetHorizontalPadding) window.SetHorizontalPadding({leftMargin}, {rightMargin});";
            webViewObject.EvaluateJS(js);

            lastMargins[0] = leftMargin;
            lastMargins[1] = topMargin;
            lastMargins[2] = rightMargin;
            lastMargins[3] = bottomMargin;
        }
    }

    private void ProcessFormula(string latex)
    {
        Debug.Log(latex);
    }

    public void SetEditorVisibility(bool isVisible)
    {
        if (webViewObject != null)
        {
            webViewObject.SetVisibility(isVisible);
        }
    }

    void OnDestroy()
    {
        if (webViewObject != null)
```

```
        {
            Destroy(webViewObject.gameObject);
        }
    }

    public bool isOffscreen = false;

    public void MoveOffscreen()
    {
        isOffscreen = true;
        if (webViewObject != null)
        {
            webViewObject.SetMargins(9999, 9999, -9999, -9999);
        }
    }

    public void RestorePosition()
    {
        isOffscreen = false;
        ForceSync();
    }

    public void HideEditorInstantly()
    {
        SetEditorVisibility(false);
        isExpanded = false;
        ForceSync();
    }
}
```