Державний торговельно-економічний університет
Кафедра інженерії програмногозабезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗУВАННЯ ЗАДАЧ ВИЩОЇ МАТЕМАТИКИ»

Студентки 2 курсу, 2мз групи, спеціальності 121 «Інженерія програмного забезпечення» освітньої програми «Інженерія програмного забезпечення» _____ Приходько Маргарити Олександрівни

підпис студента

Науковий керівник
кандидат педагогічних наук,
доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ Жирова Тетяна Олександрівна

підпис керівника

Гарант освітньої програми
кандидат педагогічних наук,
доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ Котенко Наталія Олексіївна

підпис гаранта

КИЇВ – 2023

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Освітня програма 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«13» грудня 2022 р.

Завдання
на випускну кваліфікаційну роботу студентові

Приходько Маргариті Олександрівні
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи «Програмне забезпечення для розв'язування задач вищої математики»

Затверджена наказом ректора від «09» грудня 2022 р. № 3339

2. Строк здачі студентом закінченої роботи 1 грудня 2023

3. Цільова установка та вихідні дані до роботи

Мета роботи розробка і практична реалізація програмного забезпечення для розв'язування задач вищої математики за допомогою мови програмування Python.

Об'єкт дослідження процес розробки програмного забезпечення для розв'язування задач вищої математики.

Предмет дослідження програмне забезпечення для розв'язування задач вищої математики.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 ТЕОРЕТИКО-МЕТОДИЧНІ АСПЕКТИ РОЗВ'ЯЗУВАННЯ ЗАДАЧ ВИЩОЇ МАТЕМАТИКИ ЗА ДОПОМОГОЮ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Класифікація і аналіз основних задач вищої математики, що підлягають автоматизації

1.2. Аналіз методів та алгоритмічних підходів до розв'язування задач вищої математики з використанням програмного забезпечення

1.3. Висновки до розділу 1

РОЗДІЛ 2 АНАЛІЗ ВИБОРУ МОВИ ПРОГРАМУВАННЯ PYTHON ЯК ІНСТРУМЕНТ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗУВАННЯ ЗАДАЧ ВИЩОЇ МАТЕМАТИКИ

2.1. Аналіз переваг Python у порівнянні з іншими мовами програмування у контексті розв'язання задач вищої математики

2.2. Обґрунтування вибору Python як оптимального інструменту для програмного забезпечення

2.3. Висновки до розділу 2

РОЗДІЛ 3 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗУВАННЯ ЗАДАЧ ВИЩОЇ МАТЕМАТИКИ

3.1. Проектування архітектури програмного забезпечення

3.2. Розробка функціоналу та реалізація програмного забезпечення

3.3. Демонстрація практичного застосування програмного забезпечення

3.4. Висновок до розділу 3

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*07.11.2022*	*07.11.2022*
2.	*Розробка та затвердження завдання на роботу магістра (стац/заоч)*	*13.12.2022*	*13.12.2022*
3.	*Вступ та перелік літературних джерел*	*24.02.2023*	*24.02.2023*
4.	*Розробка технічного завдання*	*15.03.2023*	*15.03.2023*
5.	*Розділ 1. Теоретико-методичні аспекти розв'язування задач вищої математики за допомогою програмного забезпечення*	*10.04.2023*	*10.04.2023*
6.	*Розділ 2. Аналіз вибору мови програмування python як інструмент розробки програмного забезпечення для розв'язування задач вищої математики*	*24.05.2023*	*24.05.2023*
7.	*Розділ 3. Проектування та розробка програмного забезпечення для розв'язування задач вищої математики*	*06.09.2023*	*06.09.2023*
8.	*Розробка програми та методики тестування*	*18.10.2023*	*18.10.2023*
9.	*Написання наукової статті*	*17.05.2023*	*17.05.2023*
10.	*Керівництво користувача*	*25.10.2023*	*25.10.2023*
11.	*Висновки та пропозиції*	*01.11.2023*	*01.11.2023*
12.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*	*06.11.2023*	*06.11.2023*
13.	*Підготовка автореферату та презентації доповіді*	*06.11.2023*	*06.11.2023*
14.	*Попередній захист випускної кваліфікаційної роботи*	*20.11.2023 – 24.11.2023*	*20.11.2023 – 24.11.2023*
15.	*Здача зброшурованої випускної кваліфікаційної роботи*	*01.12.2023*	*01.12.2023*
16.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*29.11.2023*	*29.11.2023*
17.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*05.12.2023-06.12.2023*	*05.12.2023-06.12.2023*

7. Дата видачі завдання «13» грудня 2022 р.

8. Науковий керівник випускної кваліфікаційної роботи Жирова Т. О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Котенко Н.О.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Приходько М. О.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускної кваліфікаційної роботи

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____Приходько М. О._____

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Котенко Н.О._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В._____

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена розробці та практичній реалізації програмного забезпечення для вирішення задач вищої математики з використанням мови програмування Python. Мета дослідження полягала в автоматизації математичних обчислень, що дозволило ефективно розв’язувати типові математичні задачі і забезпечити підтримку наукової та освітньої діяльності. У роботі було здійснено аналіз існуючих підходів до автоматизації математичних задач, виокремлено та класифіковано задачі, найбільш придатні для автоматизації. Основною розробкою є програмне забезпечення з інтуїтивно зрозумілим інтерфейсом і модульною архітектурою, здатне ефективно обробляти різноманітні математичні задачі. Тестування програми продемонструвало високу надійність, точність і швидкість обчислень. Подальший розвиток проекту передбачає розширення наукових бібліотек Python, оптимізацію для роботи з великими даними та ускладненими обчислювальними моделями, а також постійне оновлення інтерфейсу для забезпечення кращої взаємодії з користувачем.

Випускна кваліфікаційна робота на тему «Програмне забезпечення для розв'язування задач вищої математики» містить 40 сторінок, 10 рисунків. Перелік використаних джерел налічує 10 найменувань.

Ключові слова: автоматизація, python, програмне забезпечення для математики, модульна архітектура, вища математика.

ABSTRACT

In accordance with the aim of the study, the work is devoted to the development and practical implementation of software for solving problems of higher mathematics using the Python programming language. The purpose of the study was to automate mathematical calculations, which allowed to effectively solve typical mathematical problems and provide support for scientific and educational activities. The paper analyzes existing approaches to the automation of mathematical tasks, identifies and classifies the tasks most suitable for automation. The main development is a software with an intuitive interface and modular architecture capable of efficiently processing various mathematical tasks. Testing of the program has demonstrated high reliability, accuracy and speed of calculations. Further development of the project involves expanding Python scientific libraries, optimization for working with big data and complex computational models, and constant updating of the interface to ensure better user interaction.

The final qualification work on "Software for solving problems of higher mathematics" contains 40 pages and 10 figures. The list of references includes 10 titles.

Keywords: automation, python, software for mathematics, modular architecture, higher mathematics.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ООП – об'єктно-орієнтований підхід

ПЗ – програмне забезпечення

GUI – графічний користувацький інтерфейс

					ДТЕУ 121 02з-14.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		19.09.23	*Програмне забезпечення для розв'язування задач вищої математики*	*Стаді*	*Аркуш*	*Аркушів*
Керівник		Жирова Т.О.		19.09.23		*ПС*	*2*	*40*
Гарант		Котенко Н.О.		19.09.23		*Факультет інформаційних технологій 2 курс, 2мз група*		
Розробив		Приходько М.О.		19.09.23	*Перелік умовних скорочень*			

ЗМІСТ

					ДТЕУ 121 02з-14.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		01.11.23	*Програмне забезпечення для розв'язування задач вищої математики*	*Стаді*	*Аркуш*	*Аркушів*
Керівник		Жирова Т. О.		01.11.23		*Зміст*	*3*	*40*
Гарант		Котенко Н.О.		01.11.23		*Факультет інформаційних технологій 2 курс, 2мз група*		
Розробив		Приходько М.О.		01.11.23	*Зміст*			

ВСТУП

В сучасному світі, де математика відіграє важливу роль у багатьох наукових і технічних галузях, розробка програмного забезпечення для розв'язування задач вищої математики стає надзвичайно актуальною. Висока складність і обсяг математичних обчислень, використання складних алгоритмів та моделей вимагають ефективних інструментів для їх розв'язування.

Одним із важливих завдань у галузі вищої математики є автоматизація процесу розв'язування складних математичних задач. Це включає в себе розробку алгоритмів, моделей та програмного забезпечення, яке забезпечує швидке і точне виконання обчислень, аналіз результатів та надійне розв'язування задач різної складності.

У цьому контексті розробка програмного забезпечення для розв'язування задач вищої математики використовуючи мову програмування Python є особливо перспективною. Python відома своєю простотою, читабельністю коду та потужними бібліотеками, що дозволяють вирішувати широкий спектр математичних задач. Вона надає зручний інтерфейс для взаємодії з чисельними методами, символьними обчисленнями, оптимізацією та моделюванням, що робить її ідеальним інструментом для розв'язування задач вищої математики.

Практичне значення розробки програмного забезпечення для розв'язування задач вищої математики полягає в тому, що воно сприяє автоматизації складних обчислень та аналізу даних, забезпечує точність

результатів і ефективність роботи, а також забезпечує можливість використання в різних галузях, де вища математика використовується для моделювання, прогнозування та оптимізації процесів.

Отже, розробка програмного забезпечення для розв'язування задач вищої математики з використанням мови програмування Python має великий потенціал і може знайти широке застосування у наукових дослідженнях, освітніх процесах та практичних галузях, де математичний аналіз є важливою складовою.

Актуальність роботи. Розробка програмного забезпечення для розв’язування задач вищої математики допоможе автоматизувати процес роботи з математичними завданнями, прискорити їх вирішення та покращити точність результатів.

Предмет дослідження: програмне забезпечення для розв’язування задач вищої математики.

Об’єкт дослідження: процес розробки програмного забезпечення для розв’язування задач вищої математики.

Мета роботи: розробка і практична реалізація програмного забезпечення для розв’язування задач вищої математики за допомогою мови програмування Python.

Для досягнення поставленої мети необхідно виконати наступні **завдання**.

1. Аналіз основних задач вищої математики, що підлягають автоматизації, та визначення їх ключових особливостей та вимог до програмного забезпечення.
2. Вивчення методів, алгоритмів та підходів, які використовуються для розв'язування задач вищої математики за допомогою програмного забезпечення.

3. Реалізація програмного забезпечення на мові програмування Python з використанням відповідних бібліотек та інструментів.

Практичне значення. Розроблене програмне забезпечення дозволяє автоматизувати обчислення, аналіз та виконання складних математичних операцій.

РОЗДІЛ 1
ТЕОРЕТИКО-МЕТОДИЧНІ АСПЕКТИ РОЗВ'ЯЗУВАННЯ ЗАДАЧ ВИЩОЇ МАТЕМАТИКИ ЗА ДОПОМОГОЮ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Класифікація і аналіз основних задач вищої математики, що підлягають автоматизації

Вища математика є важливою складовою багатьох наукових і технічних галузей. Її застосування включає розв'язування складних математичних задач, моделювання фізичних процесів, обробку даних, прогнозування та багато іншого.

Однак, розв'язування складних математичних задач часто вимагає великої кількості часу та зусиль. Тому розробка програмного забезпечення для автоматизації розв'язування задач вищої математики є актуальною та важливою задачею.

Перед тим, як приступити до розробки програмного забезпечення, необхідно провести класифікацію основних задач вищої математики, що підлягають автоматизації (рис 1.1) . Цей процес допоможе систематизувати різноманітні задачі та визначити їх ключові особливості та вимоги до програмного забезпечення.

Рис. 1.1. Класифікація задач вищої математики

Однією з типових задач є виконання операцій над матрицями. Матриці використовуються для представлення лінійних перетворень, розв'язування систем лінійних рівнянь та інших математичних і інженерних застосувань. Автоматизація операцій з матрицями дозволяє збільшувати швидкість обчислень і точність визначення результатів.

Векторна алгебра застосовується для розв'язання завдань пов'язаних з векторними просторами, що є фундаментальними в геометрії та фізиці. Це включає в себе вивчення сили, руху та напрямку векторів. Автоматизація розрахунків векторної алгебри полегшує обробку складних векторних операцій, що важливо для інженерних досліджень та комп'ютерного моделювання.

Комплексні числа знаходять широке застосування в електротехніці, квантовій механіці та інших областях науки. Вони дозволяють вирішувати рівняння, які не мають розв'язків серед дійсних чисел, та моделювати коливання та хвильові процеси. Автоматизація обчислень з комплексними числами значно спрощує аналіз та розв'язування відповідних завдань.

Ліміти є фундаментальною концепцією математичного аналізу, що має застосування в інженерії, фізиці та економіці для опису асимптотичної поведінки функцій. Автоматизація визначення лімітів допомагає аналітикам швидко оцінювати поведінку функцій в точках наближення, що є важливим для рішення багатьох прикладних задач.

Ще однією важливою задачею є обчислення інтегралів та логарифмів. Вони використовуються для обчислення площ під кривими, визначення сум, вирішення оптимізаційних задач та багатьох інших застосувань. Автоматичне виконання цих обчислень дозволяє забезпечити швидкість, точність та ефективність обробки великих обсягів даних.

Графічне представлення функцій є ключовим для візуального аналізу зміни величин і вивчення їх поведінки. Воно необхідне для підтримки інтуїтивного розуміння математичних концепцій від алгебри до складних областей математичного аналізу.

Розгляд типових задач вищої математики, що підлягають автоматизації, виявляє великий потенціал для розвитку програмного забезпечення, яке забезпечує швидке та ефективне розв'язування цих задач. Автоматизація дозволяє економити час та ресурси, забезпечує точність і надійність результатів, а також відкриває нові можливості для використання математики в різних галузях науки, техніки та інженерії. Подальший розвиток програмного забезпечення для розв'язування задач вищої математики є важливим напрямком, який впливає на розвиток сучасної науки та технологій.

Розв'язування задач вищої математики вимагає не лише глибокого розуміння математичних концепцій і методів, але і уміння використовувати їх для аналізу, моделювання та прогнозування різних явищ і процесів. Цей процес може бути вкрай складним через різноманітність задач і їхніх

особливостей. Існують деякі особливості та складнощі, з якими стикаються при розв'язуванні задач вищої математики, а також способи їх подолання.

Перш за все, задачі вищої математики можуть бути дуже абстрактними і комплексними. Вони часто вимагають високого рівня абстрактного мислення та уміння встановлювати логічні зв'язки між різними математичними об'єктами. Розв'язування таких задач вимагає глибокого розуміння математичних концепцій, теорем і методів, а також їхнього використання для формулювання логічних аргументів і доведення результуючих тверджень.

Крім того, однією з особливостей задач вищої математики є їхня не лінійність. Багато задач мають нелінійні залежності і складні структури, що ускладнює їх розв'язування. Іноді вирішення таких задач потребує застосування чисельних методів, ітераційних процесів та числових розрахунків. Важливо враховувати ці особливості під час розробки програмного забезпечення для розв'язування задач вищої математики, щоб забезпечити точність і надійність результатів.

Також варто відзначити, що розв'язування задач може вимагати багато часу та ресурсів. Деякі задачі вимагають великої обчислювальної потужності і великого обсягу пам'яті для їх ефективного вирішення. Це може ставити виклики перед програмним забезпеченням і вимагати оптимізації алгоритмів і структур даних.

Безперечно, при розв'язуванні задач вищої математики виникають труднощі і технічні виклики. Проте, розробка програмного забезпечення для їх автоматизації може суттєво полегшити процес, збільшити ефективність і точність рішень, а також забезпечити можливість швидкого перегляду і аналізу результатів.

1.2. Аналіз методів та алгоритмічних підходів до розв'язування задач вищої математики з використанням програмного забезпечення

Математика відіграє ключову роль у численних наукових і технічних дисциплінах, надаючи нам алгоритмічні методи для ефективного розв'язування широкого спектру задач. Для початку розглянемо визначення і опис основних алгоритмічних методів, які застосовуються для рішення складних математичних проблем.

Матриці і операції над ними становлять основу багатьох математичних і інженерних аналізів. Особливу увагу приділимо трьом фундаментальним операціям: додаванню, відніманню та множенню матриць.

Додавання та віднімання матриць — це елементарні операції, які виконуються за елементами. Ці операції можливі лише для матриць однакового розміру і полягають у покомпонентному додаванні чи відніманні відповідних елементів двох матриць. Результатом є нова матриця, в якій кожен елемент є сумою чи різницею відповідних елементів вихідних матриць.

Множення матриць є складнішою операцією і вимагає більш глибокого розуміння. Не кожні дві матриці можна помножити одна на одну. Для цього необхідно, щоб кількість стовпців у першій матриці відповідала кількості рядків у другій. Результатом множення матриці A розміру $m \times n$ на матрицю B розміру $n \times p$ є нова матриця C розміру $m \times p$, де кожен елемент обчислюється як скалярний добуток i-го рядка матриці A на j-й стовпець матриці B.

Ці операції є основними будівельними блоками для більш складних математичних задач, включаючи системи лінійних рівнянь, перетворення векторних просторів, обчислення визначників та власних значень матриць. Маючи можливість виконувати ці операції, математики та інженери можуть конструювати більш складні математичні моделі та розв'язувати задачі, що

зустрічаються в фізиці, комп'ютерних науках, економіці та багатьох інших дисциплінах.

Векторна алгебра є важливим розділом математики, який досліджує вектори та операції, що можна з ними виконувати. Вектори володіють властивостями напрямку та величини та можуть бути представлені у тривимірному просторі за допомогою координат x, y та z. Основні операції над векторами включають додавання, віднімання, скалярний та векторний добуток та знаходження модуля вектору.

Додавання векторів відбувається шляхом покомпонентного додавання їх координат, тоді як для віднімання векторів відповідні координати одного вектора віднімаються від координат іншого. Результатом є нові вектори, які вказують на суму чи різницю напрямків та величин оригінальних векторів.

Скалярний добуток двох векторів дає скалярну величину, яка є результатом множення відповідних координат векторів та їхнього подальшого сумування. Цей результат відображає не лише спільний напрям векторів, а й кут між ними, і часто використовується для визначення ортогональності векторів.

Модуль вектора, який також відомий як його довжина чи норма, визначається як корінь квадратний із суми квадратів його координат. Це дає відстань від початку координат до точки, яку вказує вектор, і є важливою мірою в контексті векторної величини.

Векторна алгебра відіграє ключову роль в розрахунках, що стосуються сил, руху та напрямків у тривимірному просторі, а також у тривимірному моделюванні та інших застосуваннях, де необхідно зрозуміти та маніпулювати напрямками та величинами у просторі.

Комплексні числа знаходять широке застосування в різних областях, включаючи фізику, інженерію та електроніку. Вони розширюють наше

розуміння числового простору, дозволяючи враховувати і вирішувати більше різноманітних математичних та фізичних задач.

Комплексні числа представляють собою комбінацію дійсної та уявної частини, і записуються у вигляді a + bi, де "a" - це дійсна частина, "b" - уявна частина, а "i" - уявна одиниця, що визначається як √(-1). За допомогою комплексних чисел можна представляти числа, які не можуть бути точно виражені в реальних числах. Основні операції над комплексними числами включають додавання, віднімання, множення, ділення та знаходження модуля числа.

Додавання та віднімання комплексних чисел відбувається покомпонентно. Для того щоб знайти суму комплексних чисел треба дійсні та уявні частини додати окремо, а уявні частини додаються окремо. Наприклад, для a + bi і c + di, сума буде (a + c) + (b + d)i.

Множення та ділення комплексних чисел виконується за правилами розподільності та використання властивостей уявної одиниці "i". Множення дійсних та уявних частин відбувається згідно з формулою (a + bi) * (c + di) = (ac - bd) + (ad + bc)i. Для ділення комплексного числа a + bi на b + di, результатом є (a + bi) / (b + di), яке можна обчислити за допомогою складних числових операцій.

Модуль комплексного числа a + bi визначається як $|a + bi| = \sqrt{(a^2 + b^2)}$ і вказує на його відстань від початку координат в комплексному числовому просторі.

Операції над комплексними числами дозволяють розв'язувати багато математичних рівнянь, що не мають розв'язків в реальних числах, і мають застосування в різних областях математики та науки. Вони також є потужним інструментом для аналізу і моделювання складних фізичних процесів і явищ.

Логарифми є математичними функціями, які використовуються для перетворення звичайних арифметичних операцій на операції з показниками.

Це математична операція, яка визначає, до якого показника потрібно підняти певне число, щоб отримати інше число. Основний вигляд логарифма має вигляд $\log_b(a)$, де "a" - це число, для якого ми обчислюємо логарифм, а "b" - це основа логарифма. Логарифми є важливим інструментом в математиці та науці та застосовуються для розв'язання різних математичних і наукових завдань, де необхідно знаходити показник або експоненту, до якої підноситься число, щоб отримати інше число.

Інтеграли, які використовуються для обчислення площ під кривими, об'ємів тіл і загальних інтегральних значень. Для обчислення інтегралів застосовуються алгоритми, які засновані на чисельних методах. Методи трапецій, Сімпсона та Монте-Карло є широко відомими та застосовуваними в практиці. Вони дозволяють апроксимувати інтеграл шляхом побудови спеціальних формул або використання випадкових вибірок.

Останнім елементом розгляду є ліміти, що використовуються для опису зміни функцій та їх поведінки у межах певних точок. Ліміти відіграють важливу роль в математичному аналізі та застосовуються в різних галузях науки та інженерії для вирішення різноманітних завдань. Ліміт функції вказує, до якого значення збігається функція, коли її аргумент наближається до певної точки чи збіжності. Для розрахунку лімітів застосовуються різні методи, включаючи алгоритми прямого обчислення та використання правил Лопіталя. Ці методи дозволяють вивчати та аналізувати властивості функцій та їх залежності від зміни параметрів.

Описані алгоритмічні методи є основою для вирішення широкого спектру задач вищої математики і їх застосування в різних наукових і технічних дисциплінах.

Оцінка можливостей реалізації алгоритмів за допомогою мов програмування також є важливою складовою. Кожна мова програмування має свої переваги та обмеження щодо реалізації різних алгоритмів. Вибір

мови програмування може впливати на продуктивність, швидкодію, доступність і легкість розробки програмного забезпечення для розв'язання задач вищої математики.

Важливо брати до уваги специфіку задачі та особливості мов програмування. Наприклад, для швидкодії та обробки великих обсягів даних мають бути відповідні мови програмування, які пропонують високу продуктивність, такі як C++ або Fortran. З іншого боку, для швидкої розробки алгоритмів можуть підійти мови, що мають велику кількість наукових бібліотек, такі як Python або MATLAB [1].

Варто зазначити також доступність і підтримку мов програмування, їх екосистему та інструментарій, що можуть спростити процес розробки та підтримки реалізації алгоритмів. Це може включати готові бібліотеки, інтегровані середовища розробки або спільноту розробників.

Враховуючи ці фактори, важливо ретельно оцінювати можливості реалізації алгоритмів за допомогою різних мов програмування. Кожна мова має свої переваги та обмеження, і вибір мови повинен відповідати конкретним вимогам та характеристикам задачі.

1.3. Висновки до розділу 1

У даному розділі проведено аналіз та класифікацію ключових задач вищої математики, що підлягають автоматизації. Вища математика відіграє важливу роль у різних галузях науки та техніки. Автоматизація процесу розв'язування її складних завдань має великий науковий та практичний потенціал.

Проведена класифікація охопила такі типи задач, як операції над матрицями, векторна алгебра, комплексні числа, ліміти, інтеграли, логарифми та графічне представлення функцій. Кожна з цих задач має свої специфічні характеристики та вимоги до програмного забезпечення.

Важливими особливостями задач є їхній абстрактний характер, нелінійність та значна обчислювальна складність. Враховуючи нелінійність, вирішення цих завдань може вимагати використання чисельних методів, ітераційних процесів та числових розрахунків. Також важливо раціонально управляти часом та ресурсами, необхідними для вирішення таких задач, та оптимізувати алгоритми та структури даних для досягнення максимальної ефективності.

Автоматизація процесу розв'язування задач вищої математики має значний потенціал для розвитку програмного забезпечення, яке допомагатиме математикам та науковцям працювати більш продуктивно та точно. Це також відкриває нові горизонти для використання математики в різних галузях науки, техніки та інженерії. Подальший розвиток ПЗ для автоматизації розв'язування задач вищої математики є важливим напрямом, який сприяє розвитку сучасної науки та технологій. Урахування специфіки завдань, доступності ресурсів, а також вибір оптимальної мови програмування для реалізації алгоритмів є ключовими аспектами в роботі над таким програмним забезпеченням.

.

РОЗДІЛ 2
АНАЛІЗ ВИБОРУ МОВИ ПРОГРАМУВАННЯ PYTHON ЯК ІНСТРУМЕНТ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗУВАННЯ ЗАДАЧ ВИЩОЇ МАТЕМАТИКИ

2.1. Аналіз переваг Python у порівнянні з іншими мовами програмування у контексті розв'язання задач вищої математики

Python - мова програмування, яка здобула широку популярність у наукових та математичних галузях завдяки своїй простоті, ефективності та гнучкості. Існують деякі структурні та синтаксичні особливості Python, які допомагають спростити розробку ПЗ для розв'язання задач вищої математики [2].

Однією з основних переваг Python є його зрозуміла синтаксична структура. Мова має просту та читабельну синтаксичну граматику, що дозволяє розробникам легко розуміти та розробляти код. Python використовує відступи для визначення блоків коду, що полегшує сприйняття логіки програми і робить код більш структурованим. Ця особливість сприяє покращенню читабельності програми, особливо в математичних обчисленнях, де точність та зрозумілість коду є важливими факторами.

Python також володіє широким спектром вбудованих типів даних та структур даних, що робить його потужним інструментом для розв'язання математичних задач. Наприклад, списки, множини та словники дозволяють зручно та ефективно працювати з колекціями даних. Також, Python надає

велику кількість вбудованих функцій та операцій для обробки даних, таких як сортування, фільтрація та статистичні обчислення.

Ще однією важливою особливістю Python є його підтримка об'єктно-орієнтованого програмування. Об'єктно-орієнтований підхід дає змогу створювати класи та об'єкти, що відповідають математичним концепціям та об'єктам. Це дозволяє реалізовувати математичні алгоритми та моделі у вигляді об'єктів з властивостями та методами. Підтримка ООП у Python робить його ефективним інструментом для створення складних математичних структур та алгоритмів.

Python також володіє великою екосистемою модулів та пакетів, які сприяють розв'язанню задач вищої математики. Наукові та математичні бібліотеки, такі як NumPy, SciPy та SymPy, надають готові реалізації математичних функцій, алгоритмів та інструментів для чисельних обчислень, символьної математики, оптимізації та візуалізації даних. Ці бібліотеки значно спрощують розробку математичного програмного забезпечення та дозволяють використовувати потужні математичні можливості Python [3].

Продуктивність є важливим аспектом при виборі мови програмування для розв'язання задач вищої математики. В цьому контексті, порівняємо Python з іншими мовами програмування, а також розглянемо його переваги та обмеження.

Перш за все, варто зазначити, що Python є інтерпретованою мовою, що може вплинути на її швидкодію порівняно з компільованими мовами, такими як C++ або Fortran. Однак, завдяки використанню оптимізованих бібліотек та JIT-компіляції, Python може досягати значної продуктивності у випадках, коли використовуються чисельні обчислення [4].

У сфері чисельних обчислень Python набув значної популярності завдяки бібліотеці NumPy, яка надає високопродуктивні масиви та операції над ними, що дозволяють виконувати чисельні обчислення у

векторизованому вигляді. Це дозволяє отримати значне прискорення в порівнянні з використанням циклів у звичайному Python коді. Крім того, існує також можливість використовувати оптимізовані бібліотеки, які мають написану на C-подібній мові ядро, такі як SciPy, які дозволяють виконувати багатоопераційні обчислення з високою швидкодією.

Також важливо зазначити, що Python має велику спільноту розробників та базу пакетів, що дозволяє розширити його можливості та досягнути високої продуктивності. Завдяки цьому, якщо необхідність у високій продуктивності стає критичною, можна використовувати спеціалізовані бібліотеки, написані на мовах з високою продуктивністю, такі як C++ або Fortran, та інтегрувати їх з Python.

Отже, при оцінці продуктивності Python для розв'язання задач вищої математики важливо враховувати контекст та особливості конкретної задачі. Python забезпечує зручну розробку, має широкий вибір наукових бібліотек та може бути досить продуктивним для чисельних обчислень. При необхідності високої швидкодії, можна використовувати оптимізовані бібліотеки або поєднувати Python з іншими мовами програмування [5].

Як було зазначено, Python має широкий вибір наукових та математичних бібліотек, які значно спрощують розробку та виконання математичних розрахунків.

Одна з найбільш відомих наукових бібліотек для Python - NumPy. Вона надає потужні функціональні можливості для маніпуляції масивами даних та векторизованих обчислень. NumPy дозволяє виконувати ефективні обчислення на масивах даних, такі як матричні операції, обчислення статистичних характеристик та генерування випадкових чисел [6].

Ще одна важлива бібліотека - SciPy, яка розширює функціональність NumPy та надає багато готових функцій для чисельних обчислень. Вона містить методи розв'язування диференціальних рівнянь, оптимізації,

інтерполяції, обробки сигналів та багато іншого. SciPy дозволяє розв'язувати складні математичні задачі швидко та ефективно [7].

Для візуалізації даних та побудови графіків Python має багато інструментів, але один з найпопулярніших - Matplotlib. Ця бібліотека дозволяє створювати різноманітні типи графіків, від простих лінійних графіків до складних тривимірних візуалізацій. Matplotlib дозволяє відображати дані у зручний та професійний спосіб, що сприяє аналізу та виведенню результатів.

Для наукових обчислень та символьної математики часто використовується бібліотека SymPy. Вона дозволяє виконувати символьні обчислення, розв'язувати рівняння, виконувати диференціювання та інтегрування, а також працювати з виразами та символьними функціями. SymPy розширює можливості Python у математичній області та дозволяє проводити складні аналітичні розрахунки.

Крім зазначених бібліотек, Python також має ряд інших корисних інструментів для математичного моделювання та розв'язання задач. До них входять Pandas для роботи з даними, Scikit-learn для машинного навчання, Statsmodels для статистичного моделювання та багато інших.

Аналіз різних наукових та математичних бібліотек Python демонструє багатогранність мови та її потенціал у розв'язанні задач вищої математики. Ці бібліотеки дозволяють розв'язувати складні математичні задачі, проводити чисельні обчислення, візуалізувати дані та виконувати символьні обчислення. Їх використання спрощує розробку програмного забезпечення для розв'язування задач вищої математики та допомагає досягти точних та надійних результатів.

Python є потужним інструментом для розробки програмного забезпечення в області розв'язання задач вищої математики. Структурні та

синтаксичні особливості Python забезпечують зрозумілість та простоту кодування, що полегшує розробку та розуміння математичних алгоритмів.

Використання Python сприяє точним та ефективним обчисленням, що робить його цінним інструментом для математичних досліджень та практичного використання в наукових та технічних дисциплінах.

2.2. Обґрунтування вибору Python як оптимального інструменту для програмного забезпечення

Python є оптимальним інструментом для розробки програмного забезпечення з розв'язування задач вищої математики.

По-перше, Python має простий та зрозумілий синтаксис, що дозволяє легко виразити математичні концепції та алгоритми. Він підтримує читабельність коду, що робить його зрозумілим для розробників та сприяє легкості супроводу та розширення програмного забезпечення.

По-друге, Python має багату екосистему бібліотек, які надають потужні інструменти для роботи з числовими обчисленнями, символьними обчисленнями та візуалізацією даних.

По-третє, Python є мовою програмування з відкритим вихідним кодом, що сприяє активному співробітництву та обміну ідеями у спільноті розробників. Це означає, що нові алгоритми, методи та рішення для задач вищої математики можуть бути швидко впроваджені та вдосконалені у вигляді нових пакетів та бібліотек. Багато розробників активно внесли свій внесок у розвиток Python-екосистеми для математичного програмування.

Загалом, Python має ряд переваг у контексті розв'язання задач вищої математики, завдяки своєму зрозумілому синтаксису, науковим бібліотекам та відкритому характеру спільноти розробників. Використання Python сприяє зручності та ефективності роботи з математичним програмним забезпеченням.

Python володіє високою сумісністю з існуючими системами та технологіями, роблячи його оптимальним вибором для розв'язання задач вищої математики. Його здатність взаємодіяти з іншими мовами програмування, такими як C, C++, Fortran та інші, дозволяє використовувати швидкодію і можливості цих мов для обчислювально-інтенсивних операцій, використовуючи Python як зручний інтерфейс та контрольну оболонку.

Python також має високу сумісність з базами даних, дозволяючи легко взаємодіяти з реляційними та нереляційними системами. Це надає можливість зберігати, обробляти та аналізувати великі обсяги даних, що є необхідним у вищої математики.

Крім того, Python підтримує інтеграцію з існуючими науковими та математичними пакетами, що використовуються у вищій математиці. Це дозволяє зручно обмінюватися даними та використовувати функціонал цих пакетів у своїх проектах.

Python також підтримує різноманітні стандартні інтерфейси та протоколи, такі як REST API, SOAP, JSON, XML та інші [8]. Це дозволяє легко інтегрувати програмне забезпечення, написане на Python, з іншими системами та сервісами, що є необхідним у розв'язанні складних математичних задач.

Крім того, Python має можливість використовувати багатопоточність та паралельні обчислення за допомогою бібліотек, таких як concurrent.futures, multiprocessing, або використання GPU через бібліотеки, наприклад, TensorFlow або PyTorch. Це дозволяє ефективно виконувати обчислення на багатоядерних процесорах або розподілені обчислення на кластерах.

Також важливо враховувати розмір пам'яті та використання ресурсів, особливо при обробці великих обсягів даних. Це дозволяє зменшити використання пам'яті та покращити ефективність програми.

Враховуючи всі ці фактори, можна стверджувати, що Python, хоча і не є найшвидшою мовою програмування, але завдяки своїм оптимізаціям, можливостям багатопотоковості та паралельних обчислень, а також ефективному використанню пам'яті, є оптимальним вибором для розв'язання задач вищої математики.

2.3 Висновки до розділу 2

В даному розділі було проведено аналіз, який надає обґрунтування вибору мови програмування Python як оптимального інструменту для розробки програмного забезпечення з метою розв'язування задач вищої математики. Python має низку важливих переваг, які роблять його ідеальним вибором у цьому контексті.

По-перше, простий та зрозумілий синтаксис Python сприяє зрозумілості та легкості розробки математичних алгоритмів та концепцій. Це полегшує розробку, супровід та розширення програмного забезпечення.

По-друге, Python має багатий вибір наукових бібліотек, таких як NumPy, SciPy, SymPy, Matplotlib, які надають потужні інструменти для чисельних обчислень, символьних обчислень та візуалізації даних.

По-третє, Python має високу сумісність з існуючими системами та технологіями, можливість взаємодії з іншими мовами програмування, інтеграцію з базами даних та зовнішніми інтерфейсами, а також підтримку багатопотоковості та паралельних обчислень.

Враховуючи ці переваги, можна вважати Python оптимальним інструментом для розв'язання задач вищої математики. Його структурні та синтаксичні особливості, наукові бібліотеки, підтримка спільноти розробників та підтримка інших систем та технологій роблять його оптимальним вибором для програмування в цій галузі.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗУВАННЯ ЗАДАЧ ВИЩОЇ МАТЕМАТИКИ

3.1. Проектування архітектури програмного забезпечення

Програмне забезпечення для розв'язування задач вищої математики, є комплексним інструментом, який надає користувачам можливість виконувати широкий спектр обчислень. Основною ціллю проектування такого програмного забезпечення є створення інтуїтивно зрозумілого, водночас гнучкого інструменту, який би відповідав потребам як освітнього сектору, так і професійної сфери, зокрема інженерії, наукових досліджень, та аналітики.

Кожна категорія математичних задач реалізується у вигляді окремого класу, який є нащадком класу **ttk.Frame** з бібліотеки Tkinter. Такий підхід дозволяє ізолювати логіку різних математичних операцій в окремих модулях, забезпечуючи високий рівень абстракції та спрощення подальшої підтримки коду [9].

Діаграма класів ПЗ для розв'язування задач вищої математики, ілюструє як центральний клас MathCalculatorApp виступає як оркестратор взаємодії користувача з додатком, управляючи відображенням різних вікон із математичними операціями (Рис. 3.1). Вихідний вузол StarttPage слугує вступним екраном, який переводить користувача на StartPage, де розташовані навігаційні кнопки до основних математичних модулів.

Кожен модуль, такий як ComplexNumbers чи MatrixOperations, представлений окремим класом, який надає спеціалізовані функції для конкретного типу обчислень. Модульна структура діаграми відображає гнучкість системи та її здатність до масштабування, забезпечуючи легке додавання нових математичних інструментів згідно з вимогами користувачів.

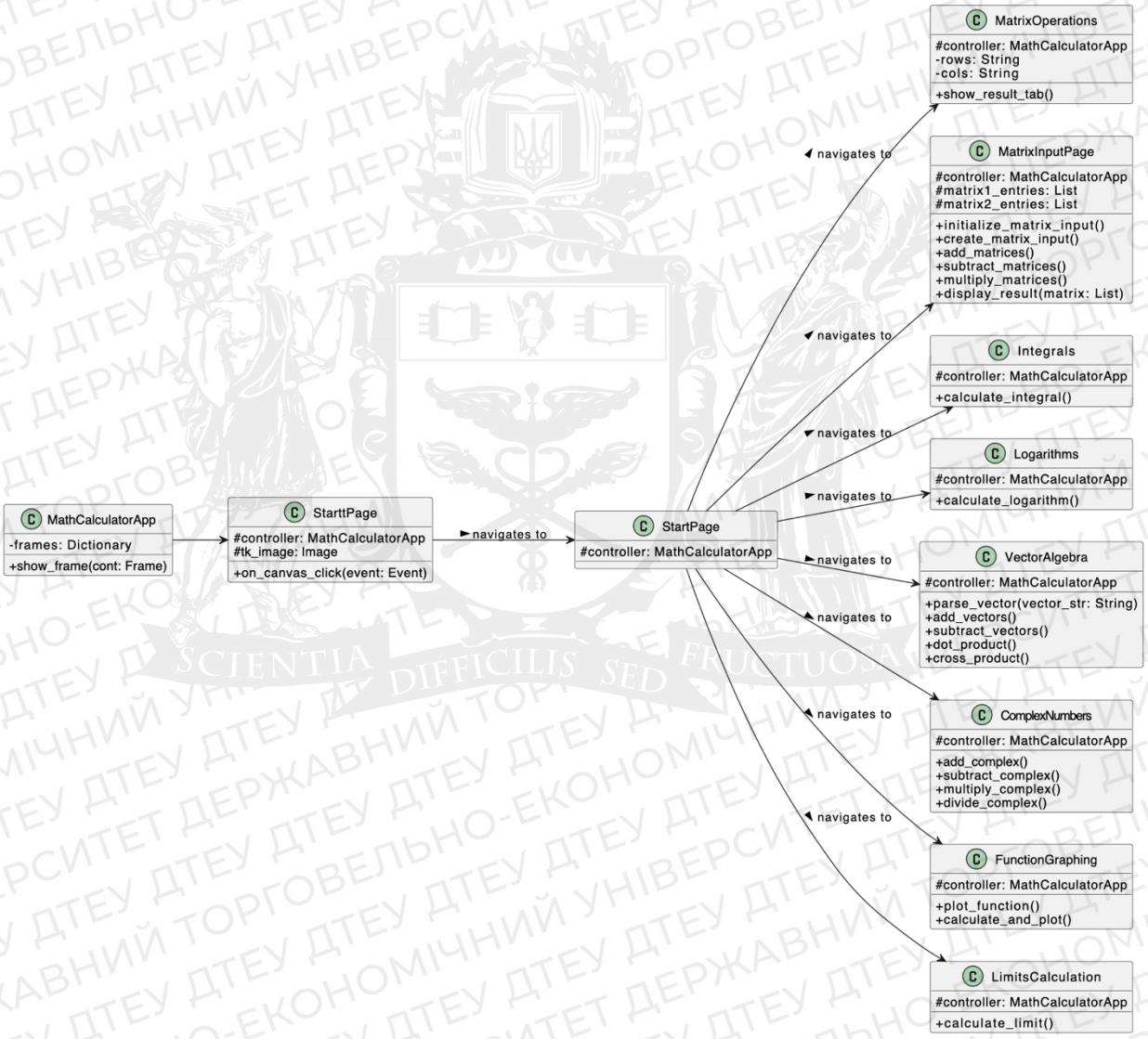


Рис. 3.1. Діаграма класів програмного забезпечення

Діаграма компонентів, представлена у додатку Б, відображає архітектурну структуру програмного забезпечення для вирішення математичних задач. Кожен компонент на діаграмі представляє окремий

модуль програми, що відповідає за певний набір функцій. Головний компонент "MathCalculatorApp" виступає як центральний контролер, який координує взаємодію між всіма частинами програми. Інші компоненти, такі як "StartPage", "MatrixOperations", "Integrals" та інші, з'єднані з головним компонентом за допомогою зв'язків "uses", що вказують на їх використання головним контролером для виконання специфічних операцій. Ця діаграма ілюструє високий рівень модульності та зручності розширення програми завдяки чітко визначеним залежностям і взаємодіям між компонентами.

Інтерфейс користувача будується на основі віджетів Tkinter, що включає в себе текстові поля для введення даних, кнопки для виконання обчислень і поля для відображення результатів. Завдяки Tkinter, інтерфейс є легко адаптованим під різні операційні системи, а також підтримує кастомізацію візуального стилю.

Діаграма активностей, представлена у додатку В, відображає процес взаємодії користувача з програмним забезпеченням для розв'язування задач вищої математики. Запуск програми ініціюється з показу головного меню, де користувачеві пропонується вибір операції. У разі позитивної відповіді програма пропонує серію математичних операцій, таких як робота з матрицями, векторами, комплексними числами, графічне представлення функцій, обчислення інтегралів та логарифмів. Кожна опція передбачає введення вхідних даних, виконання обчислень та відображення результатів. Варіант виходу з програми доступний користувачеві на будь-якому етапі взаємодії, що забезпечує гнучкість та зручність користування додатком.

При проектуванні було приділено особливу увагу обробці помилок. У випадку введення користувачем нечинних даних, система відображає зрозуміле повідомлення про помилку. Це забезпечує зручність користування та допомагає уникнути неправильного інтерпретування результатів.

Архітектура розроблена з урахуванням потенційного розширення програми. Нові модулі для обчислення, наприклад, похідних можуть бути додані з мінімальними змінами в існуючому коді.

Архітектура програмного забезпечення створена з акцентом на масштабованість, надійність і зручність використання, що робить її ідеальною для широкого спектру задач вищої математики і забезпечує потужну основу для майбутнього розвитку.

3.2. Розробка функціоналу та реалізація програмного забезпечення.

Програмне забезпечення для розв'язування задач вищої математики представляє собою ефективний інструмент, призначений для вирішення широкого спектру математичних задач. Розглянемо детальний аналізі загальних характеристик даного програмного забезпечення

Графічний користувацький інтерфейс (GUI), реалізований на основі бібліотеки Tkinter, демонструє поєднання естетичних та функціональних аспектів. Оснащення темною темою оформлення не лише кореспондує сучасним трендам інтерфейсного дизайну, а й сприяє зниженню візуальної напруги користувача, що є критичним при проведенні тривалих аналітичних сесій [10].

Структурна організація програмного забезпечення відзначається логічним розділенням функціоналу на модулі, що відображені окремими сторінками. Такий підхід забезпечує не лише модульність та масштабованість рішення, але й оптимізує навігацію та взаємодію користувача з різноманітними компонентами системи.

Функціональний можливості додатку охоплюють широкий спектр математичних інструментів, таких як операції над матрицями, векторну алгебру, комплексні числа, графічне представлення функцій, обчислення лімітів, інтегралів та логарифмів. При цьому, ініціюючи роботу з додатком,

користувач має можливість вибору відповідної математичної дії через інтуїтивно зрозуміле головне меню, що значно спрощує процес взаємодії з програмою (Рис. 3.2).

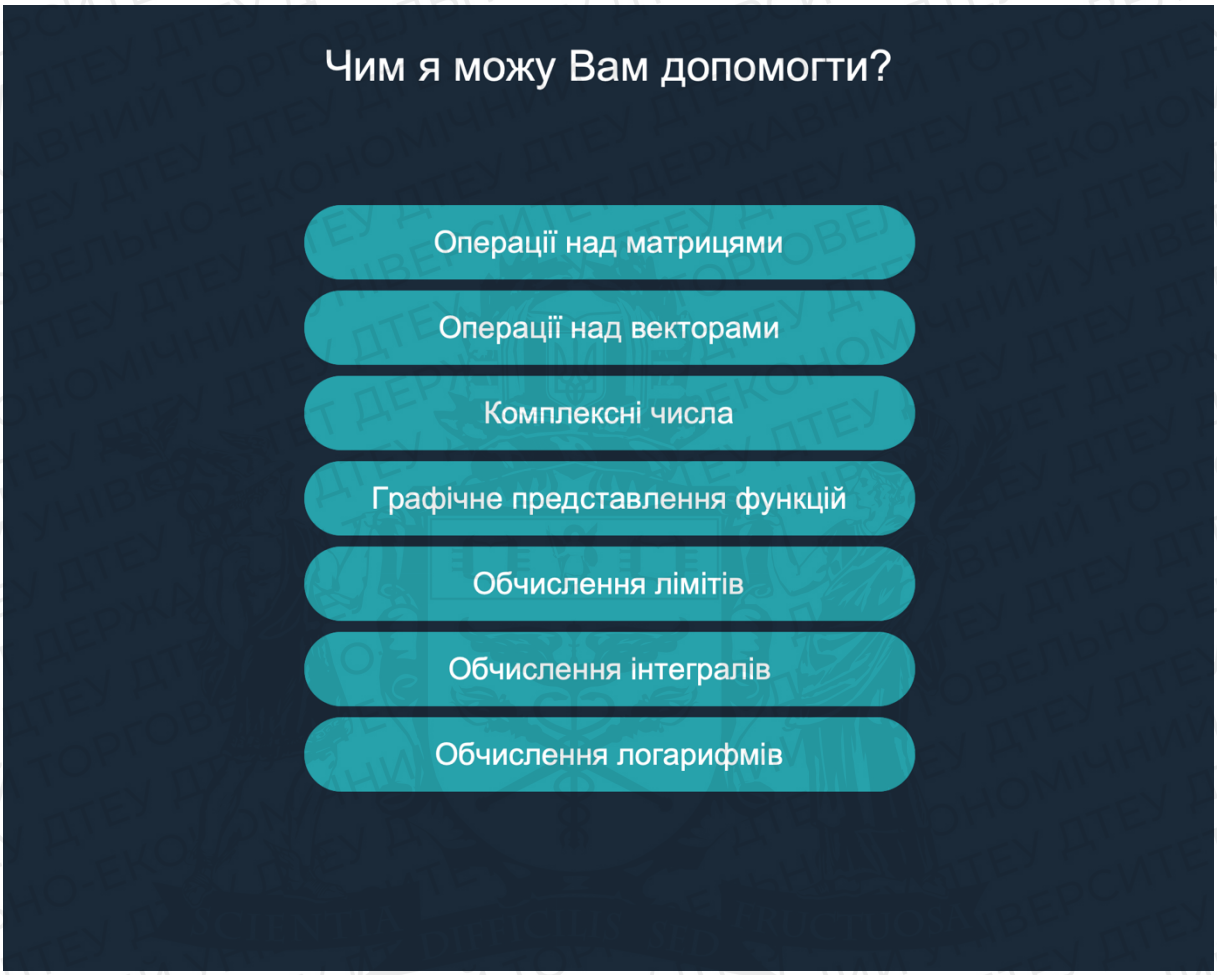


Рис. 3.2. Головне меню додатку

Розглянемо функціонал двох модулів «Операції над матрицями» та «Операції над векторами»

Модуль "Операції над матрицями" є важливою частиною розробленої системи, який забезпечує користувачам потужний інструмент для роботи з матрицями. Його інтуїтивно зрозумілий інтерфейс та надійна система обробки помилок забезпечують високу ефективність та зручність використання.

Введення розмірів матриць відбувається через випадаючі списки (combobox), де користувач може вибрати від 1 до 5 рядків та стовпців (Рис. 3.3). Таке обмеження розміру обрано з міркувань оптимізації візуального відображення та зручності користувача.

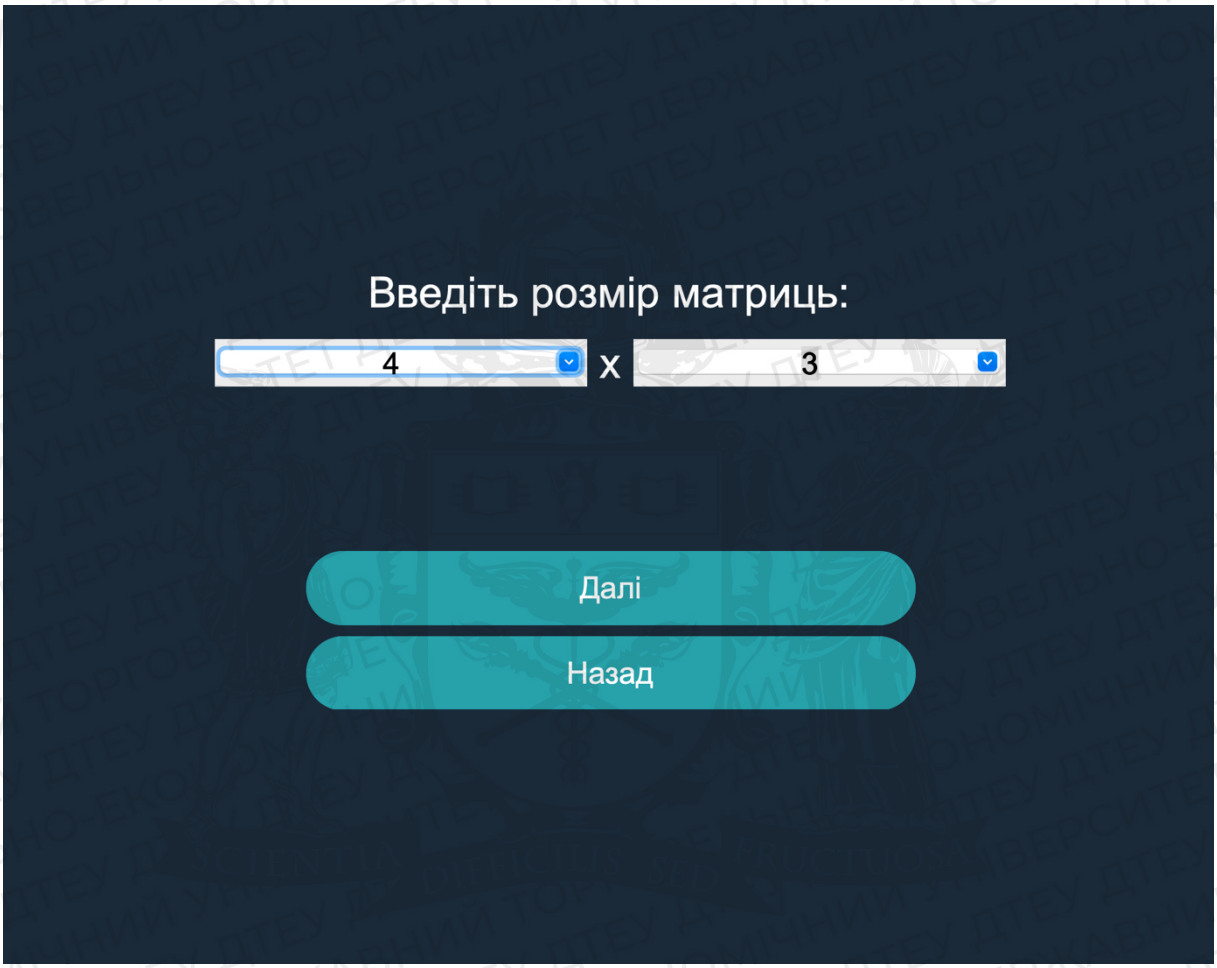


Рис. 3.3. Введення розміру матриць

Програмний модуль дозволяє виконувати такі основні операції над матрицями, як додавання, віднімання та множення (Рис. 3.4).

Рис. 3.4. Операції над матрицями

Результати відображаються у вигляді нової матриці безпосередньо у графічному інтерфейсі, що забезпечує зворотний зв'язок для користувача.

Обробка помилок в модулі реалізована через систему інформаційних повідомлень. Якщо, наприклад, користувач ввів текст замість числа користувач отримує відповідне інформаційне повідомлення (Рис. 3.5.).

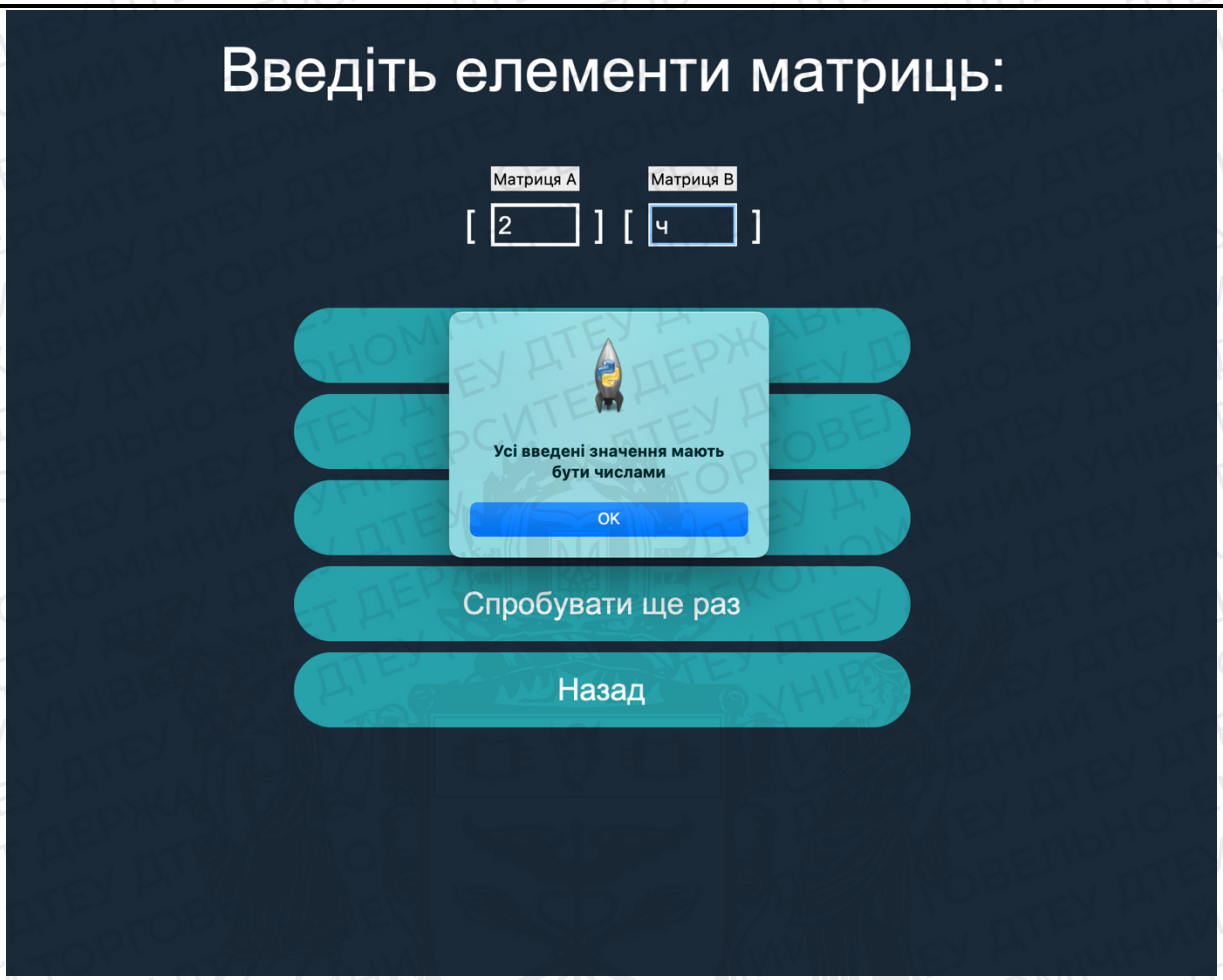


Рис. 3.5. Інформаційне повідомлення про помилку

Модуль "Операції над векторами" є компонентом програмного забезпечення, який забезпечує швидкий і точний розрахунок векторних операцій. Для роботи користувачу необхідно ввести координати векторів A і B. Введення координат здійснюється через текстові поля **ttk.Entry**, розташовані в блоку вводу векторів. Користувачу потрібно вказати значення для кожної з осей (x, y, z), відокремлюючи їх пробілами. Після введення даних у текстові поля, можливо їх перевірити та підтвердити, що всі координати введено вірно (Рис. 3.6). Такий метод введення даних є інтуїтивним і зручним, оскільки він дозволяє швидко ввести вектори, що мають до трьох вимірів, і забезпечує високу точність введення за рахунок безпосереднього контролю над кожною координатою.

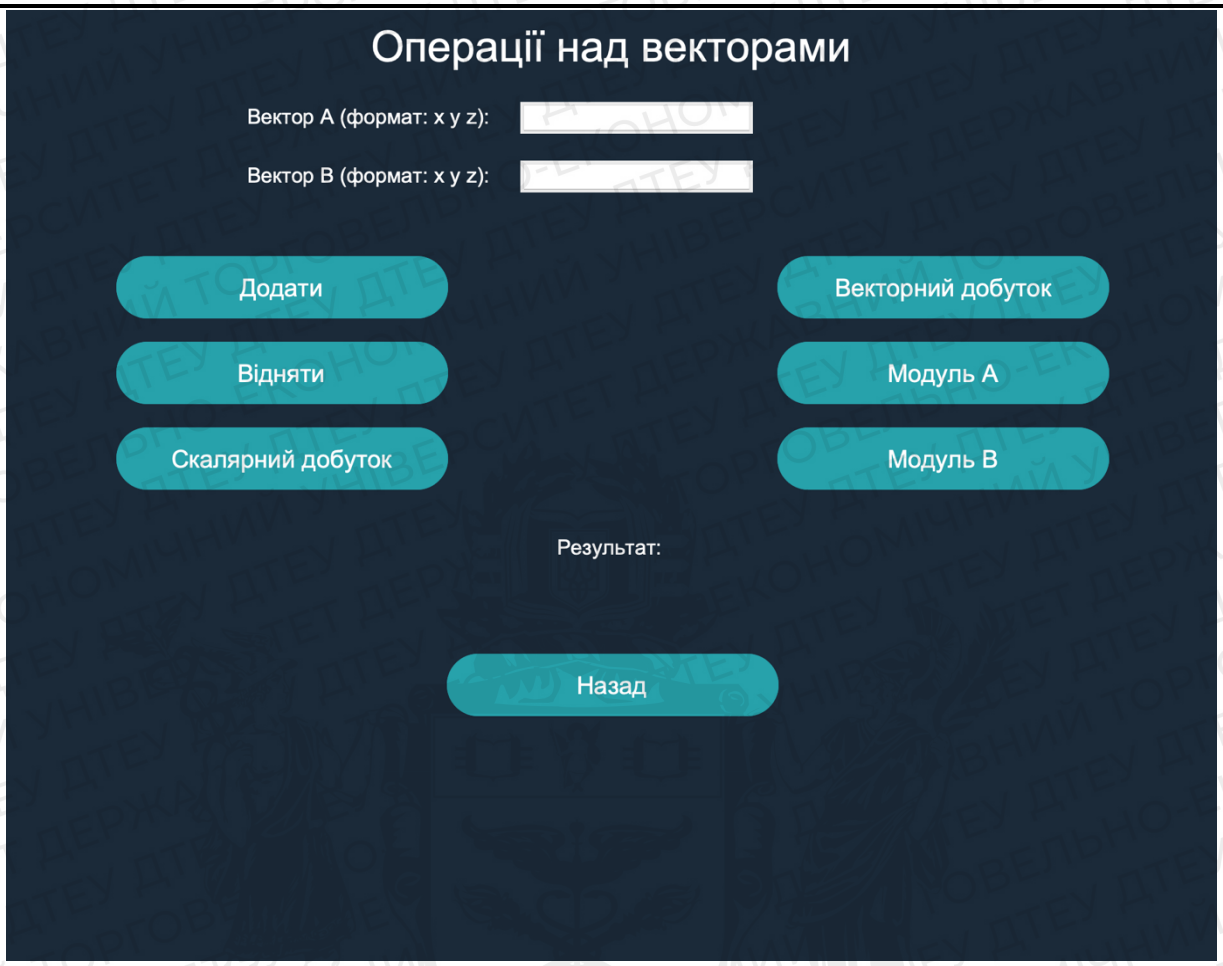


Рис. 3.6. Форма для введення даних.

Користувачі мають можливість виконувати різноманітні операції над векторами, такі як додавання, віднімання, знаходження скалярного та векторного добутків, а також обчислення модуля вектора. Для ініціації виконання операції користувач має натиснути на відповідну кнопку на центральній панелі. Після цього програма обчислює результат, використовуючи введені користувачем дані та обрану операцію, і відображає результат на мітці **self.result_label**. Це дозволяє отримати миттєвий зворотний зв'язок та забезпечує зручність при аналізі та порівнянні векторів. Такий підхід забезпечує як швидкість обчислень, так і їх наглядність для користувача.

3.3. Демонстрація практичного застосування програмного забезпечення

Розглянемо практичне застосування програмного забезпечення, на прикладах типових математичних задач, які можуть бути вирішені за допомогою даного програмного продукту. Для кожного прикладу описано початкові умови задачі, процес її розв'язування за допомогою розроблених модулів, а також представлено отримані результати.

Приклад 1: Необхідно обчислити визначений інтеграл функції $f(x)=x^2$ на інтервалі від 1 до 3.

Процес розв'язання: Для обчислення інтегралу користувач вводить у відповідні поля програмного модуля функцію $f(x)$, нижню межу інтегрування (1) та верхню межу інтегрування (3) (Рис. 3.7).

Рис. 3.7. Введення даних

Результат: Після проведення обчислень програма виводить результат, який буде еквівалентний значенню 8.6667. Цей результат відображається у користувальницькому інтерфейсі, забезпечуючи користувача зрозумілим і точним розв'язком задачі (Рис. 3.8).

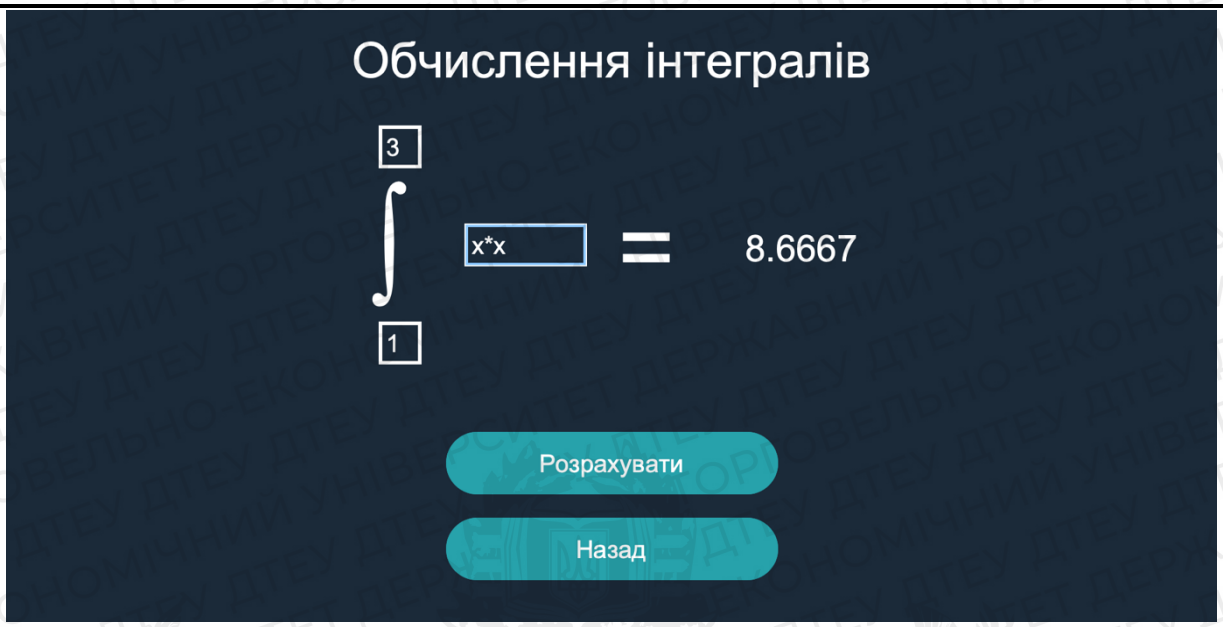


Рис. 3.8. Виведення результату обчислень

Демонстрація можливості програмного забезпечення з обчислення визначених інтегралів підкреслює його придатність для застосування в академічних цілях та у розв'язанні реальних інженерних проблем, де такі обчислення є необхідністю.

Приклад 2: Необхідно візуалізувати графік функції f(x)=(x+2)*x*x, де x = 10.

Процес розв'язання: Відкриваємо модуль для графічного представлення функцій у програмному забезпеченні. Вводимо вираз функції (x+2)*x*x у поле для введення математичних виразів. Обираємо діапазон змінної x. Натискаємо кнопку для побудови графіку.

Результат: На екрані з'являється графік функції (Рис. 3.9)

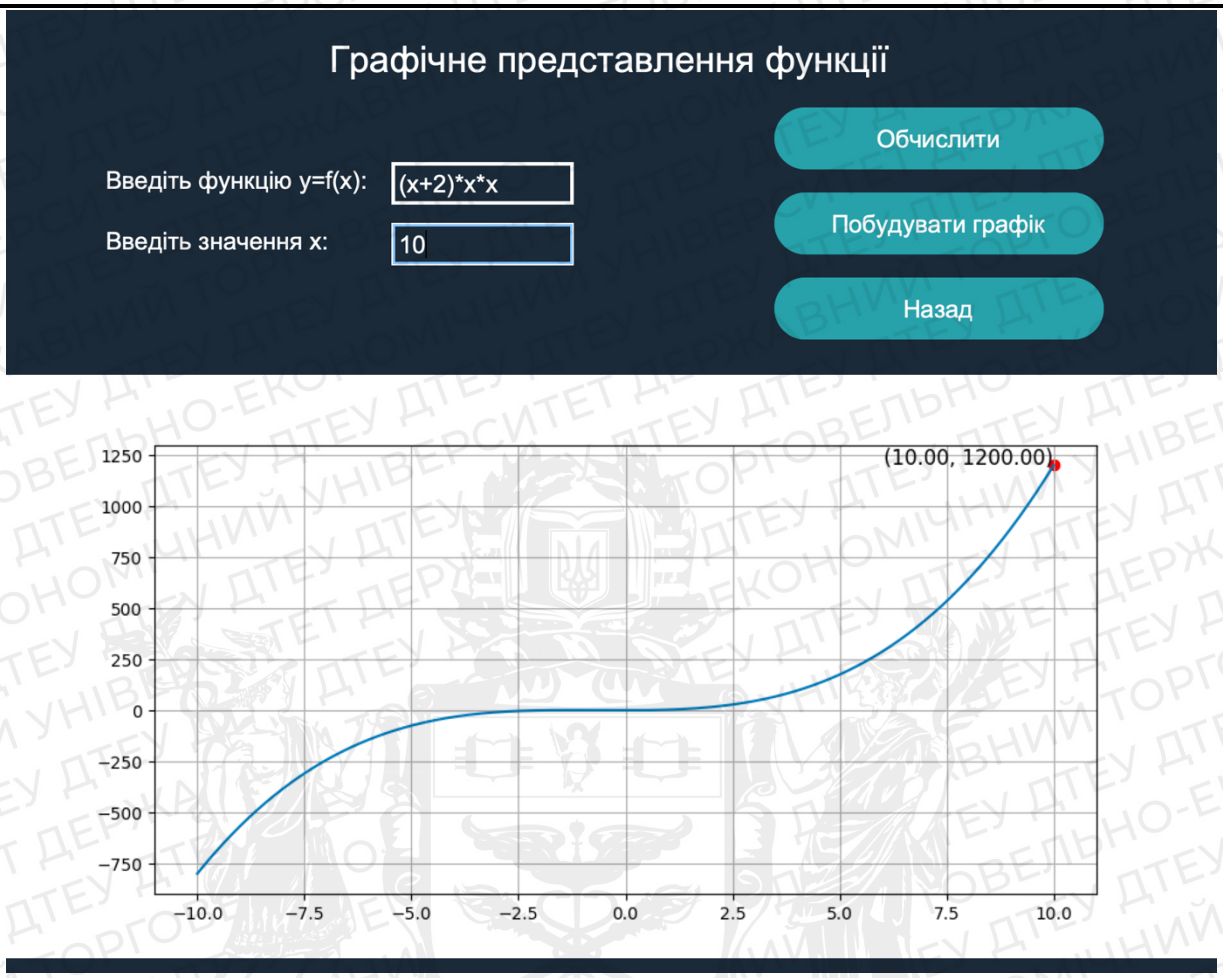


Рис. 3.9. Графік функції

Цей приклад показує, як програмне забезпечення може бути використане для візуального аналізу складних функцій та їх властивостей. Це особливо корисно при розв'язанні математичних та інженерних задач, коли необхідно зрозуміти форму та особливості графіків різних функцій.

3.4. Висновок до розділу 3

У даному розділі було розглянуто процес проектування та розробки програмного забезпечення, призначеного для розв'язання математичних задач.

Розроблено інтуїтивно зрозумілий інтерфейс, який реалізований за допомогою бібліотеки Tkinter і надає змогу користувачам зручно навігувати між різними функціями програми.

Архітектура програмного забезпечення, ілюстрована діаграмами, підкреслює організованість і модульність системи. Основний клас MathCalculatorApp забезпечує управління каркасом програми та ініціалізацію головного вікна. Від цього класу відгалужуються спеціалізовані модулі, такі як ComplexNumbers, MatrixOperations, VectorAlgebra та інші, кожен з яких відповідає за окремий сегмент обчислень і має унікальний користувацький інтерфейс. Це дає можливість забезпечити легке масштабування та додавання нових функцій у майбутньому.

Велике значення мають функціональні можливості програми, що включають в себе роботу з матрицями, векторами, комплексними числами та іншими математичними об'єктами. Інтуїтивно зрозуміле головне меню та логічна структура дозволяють користувачам ефективно використовувати програмне забезпечення для розв'язання складних задач.

Застосування програми було продемонстровано на прикладі розв'язування типових математичних задач, таких як обчислення визначених інтегралів. Точність і швидкість обчислень, які були показані в цих прикладах, свідчать про високу ефективність та надійність розробленого продукту.

Отже, створене програмне забезпечення представляє собою потужний інструмент, який забезпечує значну підтримку в роботі математиків, інженерів, науковців, і студентів. Його розробка враховує найсучасніші технічні вимоги та потреби користувачів, забезпечуючи зручність, швидкість і точність обчислень, що робить його цінним у сфері освітніх та наукових програмних рішень.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Метою роботи була розробка та практична реалізація програмного забезпечення для розв’язування задач вищої математики за допомогою мови програмування Python. В результаті було визначено ключові аспекти автоматизації математичних обчислень, класифіковано та проаналізовано типи задач, що підлягають автоматизації, та розглянуто особливості їх реалізації в програмному забезпеченні. Мова програмування Python була обрана як інструмент розробки на підставі її широких можливостей, зокрема простоти синтаксису, багатої бібліотеки наукових пакетів, високої сумісності з іншими технологіями та зручності розширення функціоналу.

В процесі проектування та розробки програмного забезпечення було створено інтуїтивно зрозумілий інтерфейс, забезпечено модульність архітектури та продемонстровано ефективність програми на прикладі розв'язання типових математичних задач. Подальші тести продукту підтвердили його надійність, точність та швидкість обчислень, підкреслюючи його практичну цінність як для наукової, так і для освітньої сфери.

Розвиток такого програмного забезпечення не може бути статичним, оскільки технології та потреби користувачів постійно змінюються. Тому важливо не лише підтримувати, але й розвивати програму, збільшуючи її можливості. Це можна досягти за рахунок розширення наукових бібліотек Python, розробки додаткових модулів для роботи з новими типами математичних завдань, і постійного вдосконалення інтерфейсу користувача для забезпечення високого рівня зручності та інтуїтивної взаємодії.

Також важливо враховувати потреби користувачів, які працюють з великими обсягами даних або складними обчислювальними моделями, оптимізуючи продуктивність програми для забезпечення плавної роботи. Регулярне тестування та зворотній зв'язок з користувачами допоможуть виявити та виправити помилки, підвищуючи надійність програми. Особлива увага повинна бути приділена створенню докладних навчальних матеріалів та документації, щоб зробити програму доступною широкому колу користувачів з різним рівнем підготовки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 12 Top Data Science Programming Languages 2023 | Built In. – 2023. [Електронний ресурс]. – Режим доступу : https://builtin.com/data-science/data-science-programming-languages

2. Підручник з Python. – 2023. [Електронний ресурс]. – Режим доступу : https://docs.python.org/uk/3/tutorial/index.html

3. Jake VanderPlas. Python Data Science Handbook. – 2016. [Електронний ресурс]. – Режим доступу : https://jakevdp.github.io/PythonDataScienceHandbook/

4. Is Python Slow? Use Cases and Comparison to Other Languages. – 2023. [Електронний ресурс]. – Режим доступу : https://www.monterail.com/blog/is-python-slow

5. Advantages Of Python Over Other Programming Languages. – 2023. [Електронний ресурс]. – Режим доступу : https://elearningindustry.com/advantages-of-python-programming-languages

6. NumPy. – 2023. [Електронний ресурс]. – Режим доступу : https://numpy.org/

7. SciPy. – 2023. [Електронний ресурс]. – Режим доступу : https://scipy.org/

8. Consuming HTTP Services in Python Course. – 2023. [Електронний ресурс]. – Режим доступу : https://training.talkpython.fm/courses/explore_http_reset_client_course/consuming-http-and-soap-services-in-python-with-json-xml-and-screen-scraping

9. Python Tkinter Frame reference, organisation with classes. – 2023. [Електронний ресурс]. – Режим доступу : https://stackoverflow.com/questions/64516247/python-tkinter-frame-reference-organisation-with-classes

10. Light or Dark Theme Changer using Tkinter. – 2023. [Електронний ресурс]. – Режим доступу : https://www.geeksforgeeks.org/light-or-dark-theme-changer-using-tkinter/

ТЕХНІЧНЕ ЗАВДАННЯ

1. Загальні відомості

1.1. Калькулятор з вищої математики

2. Мета та призначення створення системи

2.1. Розробити програмне забезпечення, призначене для розв'язання задач вищої математики.

3. Основні функції програми

3.1. Введення математичних виразів:

Програма повинна дозволяти користувачам вводити математичні вирази та рівняння в зручному для них форматі.

3.2. Обчислення результатів:

Програма повинна обробляти введені вирази та рівняння і виводити користувачам точні результати.

3.3. Графічний інтерфейс:

Програма повинна мати інтуїтивно зрозумілий графічний інтерфейс, який дозволить користувачам легко працювати з математичними моделями.

3.4. Підтримка різних розділів математики:

Програма повинна підтримувати широкий спектр математичних задач, включаючи операції над матрицями, векторами, комплексними числами, графічне представлення функцій, обчислення лімітів, інтегралів та логарифмів.

					ДТЕУ 121 02-14.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		06.09.23	*Програмне забезпечення для розв'язування задач вищої математики*	*Стадія*	*Аркуш*	*Арку*
Керівник		Жирова Т.О.		06.09.23		*ТЗ*	*41*	*40*
Гарант		Котенко Н.О.		06.09.23		*Факультет інформаційних технологій 2 курс, 2мз група*		
Розробив		Приходько М.О.		06.09.23	*Технічне завдання*			

4. Технічні вимоги

4.1. Мова програмування:

Для розробки програми рекомендується використовувати мову програмування Python.

4.2. Бібліотеки та фреймворки:

Програма повинна інтегрувати математичні бібліотеки, такі як NumPy та SciPy.

4.3. Графічний інтерфейс:

Для створення графічного інтерфейсу можна використовувати бібліотеку Tkinter або PyQt.

5. Завдання для виконання

5.1. Проектування та розробка інтерфейсу користувача.

5.2. Вибір та інтеграція необхідних математичних бібліотек.

5.3. Реалізація алгоритмів розв'язання математичних задач.

5.4. Тестування функціональності та надійності програми.

ТЕСТУВАННЯ ПЗ ДЛЯ РОЗВ'ЯЗУВАННЯ ЗАДАЧ ВИЩОЇ МАТЕМАТИКИ

Метою тестування є перевірка програми з метою забезпечення її функціональності, надійності та відповідності технічним вимогам і очікуванням користувачів. Завдання тестування включають:

1. Визначення правильності обчислень: Перевірка того, чи розв'язання математичних операцій відповідають очікуваним результатам.

2. Перевірка функціональності: Перевірка того, чи відповідає програма вимогам щодо введення та обробки математичних операцій.

3. Оцінка продуктивності: Визначення того, наскільки швидко програма виконує обчислення та реагує на дії користувачів.

4. Тестування користувацького інтерфейсу: Оцінка зручності та інтуїтивності взаємодії з графічним інтерфейсом.

5. Валідація введення і виведення даних: Перевірка коректності обробки введених математичних виразів та відображення результатів.

6. Тестування на надійність: Виявлення та виправлення можливих помилок, виключення можливих ситуацій аварійного завершення роботи програми.

7. Забезпечення відповідності технічним вимогам: Перевірка, чи програма відповідає всім технічним вимогам, включаючи обрану мову програмування та використані бібліотеки.

Тестування функціональності отримання даних:

1. Введення математичних виразів:

Виконати тести для перевірки коректного отримання введених математичних виразів від користувача.

2. Перевірка коректності введення даних:

Виконати тести для перевірки наявності і обробки помилок при введенні некоректних математичних виразів.

3. Перевірка обробки спеціальних символів:

Перевірити, як програма обробляє спеціальні символи, такі як літери, коми тощо.

Тестування функціональності аналізу та відображення даних:

1. Обчислення результатів:

Перевірити, чи програма правильно обчислює результати математичних виразів та рівнянь.

Виконати тести для різних типів обчислень.

2. Відображення результатів:

Перевірити, чи програма правильно відображає обчислені результати.

Переконатися, що результати відображаються в зрозумілому для користувача форматі.

3. Підтримка різних математичних об'єктів:

Виконати тести для роботи з різними математичними об'єктами, такими як матриці, вектори, комплексні числа тощо.

Перевірити, чи правильно програма обробляє операції з цими об'єктами та відображає результати.

Після проведення комплексного тестування програми "Калькулятор з вищої математики", були отримані наступні результати:

1. Функціональність отримання даних:

Введення математичних виразів працює без помилок та відповідає очікуванням.

Коректно виявляє та обробляє помилки при введенні некоректних даних.

2. Функціональність аналізу та відображення даних:

Обчислення математичних виразів відповідають правильним результатам.

Результати вірно відображаються для користувача в зрозумілому форматі.

Підтримка різних математичних об'єктів (матриці, вектори, комплексні числа) працює правильно.

3. Тестування на надійність:

В ході тестування не було виявлено недоліків в обробці типів даних, які ведуть до аварійного завершення програми.

4. Тестування продуктивності:

Програма продемонструвала хорошу продуктивність при обчисленнях, проте в окремих випадках може виникати затримка при роботі з великими об'ємами даних. Рекомендується оптимізувати алгоритми обчислень.

ДОДАТКИ

Додаток А

Лістинг програми

Клас MathCalculatorApp центральний контрлер додатку:

```
class MathCalculatorApp(ThemedTk):
    def __init__(self, *args, **kwargs):
        ThemedTk.__init__(self, *args, **kwargs)
        self.title("Калькулятор з вищої математики")
        self.geometry("1000x800")
        self.configure(bg="#1E2A38")
        self.frames = {}

        for F in (StarttPage, StartPage, MatrixOperations, MatrixInputPage, Integrals, Logarithms, VectorAlgebra,
                  ComplexNumbers, FunctionGraphing, LimitsCalculation):
            frame = F(self, self)
            self.frames[F] = frame
            frame.pack(side="top", fill="both", expand=True)
        self.show_frame(StarttPage)

    def show_frame(self, cont):
        for frame in self.frames.values():
            frame.pack_forget()
        self.frames[cont].pack(side="top", fill="both", expand=True)
```

Клас StarttPage вступне вікно додатку:

```
class StarttPage(tk.Frame):
    def __init__(self, parent, controller):
        ttk.Frame.__init__(self, parent)
        self.controller = controller
        canvas = tk.Canvas(self, bg="#1E2A38", highlightthickness=0)
        canvas.pack(expand=True, fill=tk.BOTH)
        button_bg_color = "#4FA0A9"
        width, height = 300, 60
        center_x = 500
        y_position = 580
        x1, y1 = center_x - width // 2, y_position - height // 2
        x2, y2 = center_x + width // 2, y_position + height // 2
        radius = 30
        canvas.create_rectangle(x1 + radius, y1, x2 - radius, y2, outline=button_bg_color, fill=button_bg_color)
        canvas.create_oval(x1, y1, x1 + radius * 2, y1 + radius * 2, outline=button_bg_color, fill=button_bg_color)
        canvas.create_oval(x2 - radius * 2, y1, x2, y1 + radius * 2, outline=button_bg_color, fill=button_bg_color)
        canvas.create_oval(x1, y2 - radius * 2, x1 + radius * 2, y2, outline=button_bg_color, fill=button_bg_color)
        canvas.create_oval(x2 - radius * 2, y2 - radius * 2, x2, y2, outline=button_bg_color, fill=button_bg_color)
        text = canvas.create_text(center_x, y_position, text="Почати", fill="white", font=("Arial", 25))
        canvas.bind("<Button-1>", self.on_canvas_click)
        image_path = "im/2638186.png"
        image = Image.open(image_path)
        new_width = 300
        new_height = 300
        image = image.resize((new_width, new_height), Image.LANCZOS)
        self.tk_image = ImageTk.PhotoImage(image)
        image_x = 350
        image_y = 150
        canvas.create_image(image_x, image_y, anchor=tk.NW, image=self.tk_image)

    def on_canvas_click(self, event):
        x1, y1 = 200, 520
        x2, y2 = 500, 580
        radius = 30


        if x1 + radius <= event.x <= x2 - radius and y1 <= event.y <= y2:
            self.controller.show_frame(StartPage)
            return
```

```
        for corner_x, corner_y in [(x1, y1), (x2, y1), (x1, y2), (x2, y2)]:
            distance = ((event.x - corner_x) ** 2 + (event.y - corner_y) ** 2) ** 0.5
            if distance <= radius:
                self.controller.show_frame(StartPage)
                return
```

Клас StartPage головне меню додатку:

```
class StartPage(ttk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.configure(bg="#1E2A38")
        self.grid_columnconfigure(0, weight=1)
        label = tk.Label(self, text="Чим я можу Вам допомогти?", font=("Arial", 35), background='#1E2A38', foreground='#FFFFFF')
        label.grid(row=0, column=0, pady=(30, 0))
        buttons = [
            ("Операції над матрицями", MatrixOperations),
            ("Операції над векторами", VectorAlgebra),
            ("Комплексні числа", ComplexNumbers),
            ("Графічне представлення функцій", FunctionGraphing),
            ("Обчислення лімітів", LimitsCalculation),
            ("Обчислення інтегралів", Integrals),
            ("Обчислення логарифмів", Logarithms),
        ]

        canvas_height = len(buttons) * 70 + 100
        canvas = tk.Canvas(self, bg="#1E2A38", highlightthickness=0, height=canvas_height)
        canvas.grid(row=1, column=0, sticky="nsew")

        for i, (text, frame) in enumerate(buttons, start=1):
            button_y_position = i * 70 + 50
            RoundedButton(canvas, 500, button_y_position, text,
                          lambda f=frame: controller.show_frame(f))
```

Клас MatrixOperations – вікно для введення розміру майбутніх матриць:

```
class MatrixOperations(ttk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.configure(bg="#1E2A38")
        center_frame = tk.Frame(self, bg="#1E2A38")
        center_frame.place(relx=0.5, rely=0.5, anchor='center')
        label = tk.Label(center_frame, text="Введіть розмір матриць:", background="#1E2A38", foreground="#FFFFFF", font=("Arial", 35))
        label.grid(row=0, column=0, columnspan=3, pady=5)
        self.rows = tk.StringVar(value='1')
        self.cols = tk.StringVar(value='1')
        size_options = ['1', '2', '3', '4', '5']
        rows_menu = ttk.Combobox(center_frame, values=size_options, textvariable=self.rows, state='readonly', font=("Arial", 25),
justify="center")
        rows_menu.grid(row=1, column=0, pady=5, padx=5)
        x_label = tk.Label(center_frame, text="x", background="#1E2A38", foreground="#FFFFFF", font=("Arial", 35))
        x_label.grid(row=1, column=1, pady=5)
        cols_menu = ttk.Combobox(center_frame, values=size_options, textvariable=self.cols, state='readonly', font=("Arial", 25), justify="center")
        cols_menu.grid(row=1, column=2, pady=5, padx=5)
        button_canvas = tk.Canvas(center_frame, bg="#1E2A38", width=600, height=250, highlightthickness=0)
        button_canvas.grid(row=2, column=0, columnspan=3, pady=5)
        RoundedButton(button_canvas, 300, 150, "Далі", self.show_result_tab)
        RoundedButton(button_canvas, 300, 220, "Назад", lambda: controller.show_frame(StartPage))

    def show_result_tab(self):
        self.controller.show_frame(MatrixInputPage)
        self.controller.frames[MatrixInputPage].initialize_matrix_input()
```

Клас MatrixInputPage – вікно введення значень та обчислення матриць:

```
class MatrixInputPage(ttk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.configure(bg="#1E2A38")
        self.label_frame = tk.Frame(self, bg="#1E2A38")
        self.matrix_frame = tk.Frame(self, bg="#1E2A38")
        self.button_frame = tk.Frame(self, bg="#1E2A38")
        self.label_frame.pack(pady=20)
```

```
        self.matrix_frame.pack(pady=20)
        self.button_frame.pack(pady=20)

        label = tk.Label(self.label_frame, text="Введіть елементи матриць:", bg="#1E2A38", fg="white", font=("Arial", 50))
        label.pack()

        self.initialize_matrix_input()

    def create_matrix_input(self, start_row, entries_list, matrix_name, matrix_position_offset):
        ttk.Label(self.matrix_frame, text=matrix_name).grid(row=start_row, column=matrix_position_offset, columnspan=self.cols + 2, pady=5)
        start_row += 1
        for i in range(self.rows):
            tk.Label(self.matrix_frame, text="[", foreground="white", font=("Arial", 30), background="#1E2A38", width=1).grid(row=start_row + i,
column=matrix_position_offset)
            row_entries = []
            for j in range(self.cols):
                entry = tk.Entry(self.matrix_frame, font=("Arial", 20), foreground="white", background="#1E2A38", width=5)
                entry.grid(row=start_row + i, column=j + 1 + matrix_position_offset, padx=5, pady=5)
                row_entries.append(entry)
            entries_list.append(row_entries)
            tk.Label(self.matrix_frame, text="]", foreground="white", font=("Arial", 30), background="#1E2A38", width=1).grid(row=start_row + i,
column=self.cols + 1 + matrix_position_offset)
    def initialize_matrix_input(self):
        for widget in self.matrix_frame.winfo_children():
            widget.destroy()
        for widget in self.button_frame.winfo_children():
            widget.destroy()
        self.rows = int(self.controller.frames[MatrixOperations].rows.get())
        self.cols = int(self.controller.frames[MatrixOperations].cols.get())
        self.matrix1_entries = []
        self.matrix2_entries = []
        self.create_matrix_input(1, self.matrix1_entries, "Матриця A", 0)
        self.create_matrix_input(1, self.matrix2_entries, "Матриця B", self.cols + 3)
        self.create_buttons(self.rows + 2, 0)

    def add_matrices(self):
        matrix1 = self.get_matrix(self.matrix1_entries)
        matrix2 = self.get_matrix(self.matrix2_entries)

        if not matrix1 or not matrix2:
            return

        if len(matrix1) != len(matrix2) or any(len(matrix1[i]) != len(matrix2[i]) for i in range(len(matrix1))):
            messagebox.showerror("Помилка", "Матриці мають бути однакового розміру для додавання")
            return
        result_matrix = []
        for i in range(len(matrix1)):
            row = []
            for j in range(len(matrix1[0])):
                row.append(matrix1[i][j] + matrix2[i][j])
            result_matrix.append(row)

        self.display_result(result_matrix)

    def subtract_matrices(self):
        matrix1 = self.get_matrix(self.matrix1_entries)
        matrix2 = self.get_matrix(self.matrix2_entries)

        if not matrix1 or not matrix2:
            return

        if len(matrix1) != len(matrix2) or any(len(matrix1[i]) != len(matrix2[i]) for i in range(len(matrix1))):
            messagebox.showerror("Помилка", "Матриці мають бути однакового розміру для віднімання")
            return

        result_matrix = []
        for i in range(len(matrix1)):
            row = []
            for j in range(len(matrix1[0])):
                row.append(matrix1[i][j] - matrix2[i][j])
            result_matrix.append(row)

        self.display_result(result_matrix)

    def multiply_matrices(self):
        matrix1 = self.get_matrix(self.matrix1_entries)
        matrix2 = self.get_matrix(self.matrix2_entries)

        if not matrix1 or not matrix2:
```

```
            return

        if len(matrix1[0]) != len(matrix2):
            messagebox.showerror("Помилка",
                                 "Кількість стовпців першої матриці має бути рівною кількості рядків другої матриці для множення")
            return
        result_matrix = []
        for i in range(len(matrix1)):
            row = []
            for j in range(len(matrix2[0])):
                total = 0
                for k in range(len(matrix1[0])):
                    total += matrix1[i][k] * matrix2[k][j]
                row.append(total)
            result_matrix.append(row)
        self.display_result(result_matrix)
    def get_matrix(self, entries):
        matrix = []
        for row_entries in entries:
            row = []
            for entry in row_entries:
                try:
                    num = float(entry.get())
                    row.append(num)
                except ValueError:
                    messagebox.showerror("Помилка", "Усі введені значення мають бути числами")
                    return
            matrix.append(row)
        return matrix

    def display_result(self, matrix):
        for widget in self.matrix_frame.winfo_children():
            widget.destroy()
        result_label = tk.Label(self.matrix_frame, text="Результат виконання операції:", font=("Arial", 30, ), background="#1E2A38",
                                foreground="#FFFFFF")
        result_label.grid(row=0, column=0, columnspan=len(matrix[0]), pady=5)
        for i in range(len(matrix)):
            for j in range(len(matrix[0])):
                entry_label = tk.Label(self.matrix_frame, text=str(matrix[i][j]), width=5, font=("Arial", 20),
                                       background="#1E2A38", foreground="#FFFFFF")
                entry_label.grid(row=i + 1, column=j, padx=5, pady=5)

    def create_buttons(self, start_row, start_col):
        button_canvas = tk.Canvas(self.button_frame, bg="#1E2A38", width=520, height=800, highlightthickness=0)
        button_canvas.grid(row=start_row + self.rows + 2, column=start_col, columnspan=3, pady=5)
        RoundedButton(button_canvas, 250, 30, "Додати", self.add_matrices)
        RoundedButton(button_canvas, 250, 100, "Відняти", self.subtract_matrices)
        RoundedButton(button_canvas, 250, 170, "Помножити", self.multiply_matrices)
        RoundedButton(button_canvas, 250, 240, "Спробувати ще раз", self.go_back2)
        RoundedButton(button_canvas, 250, 310, "Назад", self.go_back)

    def go_back(self):
        self.controller.show_frame(MatrixOperations)

    def go_back2(self):
        self.initialize_matrix_input()

    def create_result_display(self, start_row, start_col):
        for i in range(self.rows):
            row_labels = []
            for j in range(self.cols):
                label = ttk.Label(self, text="0", width=5)
                label.grid(row=start_row + i, column=j + start_col, padx=5, pady=5)
                row_labels.append(label)
            self.result_labels.append(row_labels)
```

Клас Integrals – вікно введення значень та обчислення інтегралів:

```
class Integrals(ttk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.configure(bg="#1E2A38")
        tk.Label(self, text="Обчислення інтегралів", foreground="white", font=("Arial", 40), background="#1E2A38").pack(pady=20)
        tk.Label(self, text="∫", foreground="white", font=("Arial", 100), background="#1E2A38").place(x=300, y=140)
        self.function_entry = tk.Entry(self, font=("Arial", 20), foreground="white", background="#1E2A38")
        self.function_entry.place(x=380, y=180, width=100)
        self.lower_bound_entry = tk.Entry(self, font=("Arial", 20), foreground="white", background="#1E2A38")
```

```
        self.lower_bound_entry.place(x=310, y=260, width=35)
        tk.Label(self, text="=", foreground="white", font=("Arial", 80), background="#1E2A38").place(x=500, y=150)
        self.upper_bound_entry = tk.Entry(self, font=("Arial", 20), foreground="white", background="#1E2A38")
        self.upper_bound_entry.place(x=310, y=100, width=35)
        self.result_label = tk.Label(self, text="", foreground="white", font=("Arial", 30), background="#1E2A38")
        self.result_label.place(x=580, y=180, width=150)
        center_frame = tk.Frame(self, bg="#1E2A38")
        center_frame.place(x=0,y=300)
        button_canvas = tk.Canvas(center_frame, bg="#1E2A38", width=1000, height=200, highlightthickness=0)
        button_canvas.grid(row=2, column=0, columnspan=3, pady=5)
        RoundedButton2(button_canvas, 500, 70, "Розрахувати", self.calculate_integral)
        RoundedButton2(button_canvas, 500, 140, "Назад", lambda: controller.show_frame(StartPage))

    def calculate_integral(self):
        function_str = self.function_entry.get()
        a = float(self.lower_bound_entry.get())
        b = float(self.upper_bound_entry.get())
        try:
            f = lambda x: eval(function_str)
            result, error = integrate.quad(f, a, b)
            self.result_label.config(text=f"{result:.4f}")
        except Exception as e:
            self.result_label.config(text="Помилка: перевірте введені дані.")
```

Клас Logarithms – вікно введення значень та обчислення логарифмів:

```
class Logarithms(ttk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.configure(bg="#1E2A38")
        tk.Label(self, text="Розрахунок логарифмів", foreground="white", font=("Arial", 40), background="#1E2A38").pack(pady=20)
        tk.Label(self, text="log", foreground="white", font=("Arial", 40), background="#1E2A38").place(x=300, y=120)
        self.base_entry = tk.Entry(self, font=("Arial", 20), foreground="white", background="#1E2A38")
        self.base_entry.place(x=370, y=165, width=50)
        tk.Label(self, text="=", font=("Arial", 35), foreground="white",background="#1E2A38").place(x=550, y=125)
        self.number_entry = tk.Entry(self, font=("Arial", 20), foreground="white", background="#1E2A38")
        self.number_entry.place(x=430, y=135, width=100)
        self.result_label = tk.Label(self, font=("Arial", 35),foreground="white",background="#1E2A38")
        self.result_label.place(x=570, y=125, width=200)
        center_frame = tk.Frame(self, bg="#1E2A38")
        center_frame.place(x=0, y=200)
        button_canvas = tk.Canvas(center_frame, bg="#1E2A38", width=1000, height=500, highlightthickness=0)
        button_canvas.grid(row=2, column=0, columnspan=3, pady=5)
        RoundedButton2(button_canvas, 500, 50, "Розрахувати", self.calculate_logarithm)
        RoundedButton2(button_canvas, 500, 120, "Назад", lambda: controller.show_frame(StartPage))

    def calculate_logarithm(self):
        number_str = self.number_entry.get()
        base_str = self.base_entry.get()
        try:
            number = float(number_str)
            if base_str:
                base = float(base_str)
                result = math.log(number, base)
            else:
                result = math.log(number)
            self.result_label.config(text=f"{result:.4f}")
        except Exception as e:
            self.result_label.config(text="Помилка")
```

Клас VectorAlgebra – вікно введення значень та обчислення векторів:

```
class VectorAlgebra(ttk.Frame):
  def __init__(self, parent, controller):
    tk.Frame.__init__(self, parent)
    self.controller = controller
    self.configure(bg="#1E2A38")
    self.grid_columnconfigure(0, weight=1)
    self.grid_columnconfigure(1, weight=2)

    tk.Label(self, text="Операції над векторами", foreground="white", font=("Arial", 35), background="#1E2A38").grid(row=0, column=0,
columnspan=2, pady=10, padx=10)
    tk.Label(self, text="Вектор A (формат: x y z):", foreground="white", font=("Arial", 17), background="#1E2A38").grid(row=1, column=0,
padx=10, pady=10, sticky="e")
    self.vector_a = ttk.Entry(self)
    self.vector_a.grid(row=1, column=1, padx=10, pady=10, sticky="w")
    tk.Label(self, text="Вектор B (формат: x y z):", foreground="white", font=("Arial", 17), background="#1E2A38").grid(row=2, column=0,
padx=10, pady=10, sticky="e")
    self.vector_b = ttk.Entry(self)
```

```
        self.vector_b.grid(row=2, column=1, padx=10, pady=10, sticky="w")
        center_frame = tk.Frame(self, bg="#1E2A38")
        center_frame.place(x=0, y=200)
        button_canvas = tk.Canvas(center_frame, bg="#1E2A38", width=1000, height=500, highlightthickness=0)
        button_canvas.grid(row=2, column=0, columnspan=3, pady=5)
        RoundedButton2(button_canvas, 230, 25, "Додати", self.add_vectors)
        RoundedButton2(button_canvas, 230, 95, "Відняти", self.subtract_vectors)
        RoundedButton2(button_canvas, 230, 165, "Скалярний добуток", self.dot_product)
        RoundedButton2(button_canvas, 770, 25, "Векторний добуток", self.cross_product)
        RoundedButton2(button_canvas, 770, 95, "Модуль A", lambda: self.vector_magnitude(self.vector_a))
        RoundedButton2(button_canvas, 770, 165, "Модуль B", lambda: self.vector_magnitude(self.vector_b))
        self.result_label = tk.Label(self, text="Результат:", foreground="white", font=("Arial", 17), background="#1E2A38")
        self.result_label.place(x=450, y=430)
        RoundedButton2(button_canvas, 500, 350, "Назад", lambda: controller.show_frame(StartPage))

    def parse_vector(self, vector_str):
        return np.array(list(map(float, vector_str.split())))

    def format_result(self, result_array):
        return "{" + ", ".join(map(str, result_array)) + "}"

    def add_vectors(self):
        vector_a = self.parse_vector(self.vector_a.get())
        vector_b = self.parse_vector(self.vector_b.get())
        result = np.add(vector_a, vector_b)
        self.result_label.config(text=f"Результат: {self.format_result(result)}")

    def subtract_vectors(self):
        vector_a = self.parse_vector(self.vector_a.get())
        vector_b = self.parse_vector(self.vector_b.get())
        result = np.subtract(vector_a, vector_b)
        self.result_label.config(text=f"Результат: {self.format_result(result)}")

    def dot_product(self):
        vector_a = self.parse_vector(self.vector_a.get())
        vector_b = self.parse_vector(self.vector_b.get())
        result = np.dot(vector_a, vector_b)
        self.result_label.config(text=f"Результат: {result}")

    def cross_product(self):
        vector_a = self.parse_vector(self.vector_a.get())
        vector_b = self.parse_vector(self.vector_b.get())
        if len(vector_a) == 3 and len(vector_b) == 3:
            result = np.cross(vector_a, vector_b)
            self.result_label.config(text=f"Результат: {self.format_result(result)}")
        else:
            self.result_label.config(text="Помилка: Векторний добуток можливий тільки для тривимірних векторів")

    def vector_magnitude(self, vector_entry):
        vector = self.parse_vector(vector_entry.get())
        result = np.linalg.norm(vector)
        self.result_label.config(text=f"Результат: {result}")
```

Клас ComplexNumbers – вікно введення значень та обчислення комлексних чисел:

```
class ComplexNumbers(ttk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.configure(bg="#1E2A38")
        self.grid_columnconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=2)

        tk.Label(self, text="Комплексні числа", foreground="white", font=("Arial", 35), background="#1E2A38").grid(row=0, column=0,
columnspan=2, pady=10, padx=10)
        tk.Label(self, text="Комплексне число A (формат: a+bj):", foreground="white", font=("Arial", 17), background="#1E2A38").grid(row=1,
column=0, padx=10, pady=10, sticky="e")
        self.complex_a = ttk.Entry(self)
        self.complex_a.grid(row=1, column=1, padx=10, pady=10, sticky="w")
        tk.Label(self, text="Комплексне число B (формат: a+bj):", foreground="white", font=("Arial", 17), background="#1E2A38").grid(row=2,
column=0, padx=10, pady=10, sticky="e")
        self.complex_b = ttk.Entry(self)
        self.complex_b.grid(row=2, column=1, padx=10, pady=10, sticky="w")
        center_frame = tk.Frame(self, bg="#1E2A38")
        center_frame.place(x=0, y=200)
        button_canvas = tk.Canvas(center_frame, bg="#1E2A38", width=1000, height=500, highlightthickness=0)
        button_canvas.grid(row=2, column=0, columnspan=3, pady=5)
```

```
        RoundedButton2(button_canvas, 230, 25, "Додати", self.add_complex)
        RoundedButton2(button_canvas, 230, 95, "Відняти", self.subtract_complex)
        RoundedButton2(button_canvas, 230, 165, "Помножити", self.multiply_complex)
        RoundedButton2(button_canvas, 770, 25, "Поділити", self.divide_complex)
        RoundedButton2(button_canvas, 770, 95, "Модуль A", lambda: self.complex_magnitude(self.complex_a))
        RoundedButton2(button_canvas, 770, 165, "Модуль B", lambda: self.complex_magnitude(self.complex_b))
        self.result_label = tk.Label(self, text="Результат:", foreground="white", font=("Arial", 17),background="#1E2A38")
        self.result_label.place(x=450, y=430)
        RoundedButton2(button_canvas, 500, 350, "Назад", lambda: controller.show_frame(StartPage))

    def parse_complex(self, complex_str):
        return complex(complex_str)

    def add_complex(self):
        complex_a = self.parse_complex(self.complex_a.get())
        complex_b = self.parse_complex(self.complex_b.get())
        result = complex_a + complex_b
        self.result_label.config(text=f"Результат: {result}")

    def subtract_complex(self):
        complex_a = self.parse_complex(self.complex_a.get())
        complex_b = self.parse_complex(self.complex_b.get())
        result = complex_a - complex_b
        self.result_label.config(text=f"Результат: {result}")

    def multiply_complex(self):
        complex_a = self.parse_complex(self.complex_a.get())
        complex_b = self.parse_complex(self.complex_b.get())
        result = complex_a * complex_b
        self.result_label.config(text=f"Результат: {result}")

    def divide_complex(self):
        complex_a = self.parse_complex(self.complex_a.get())
        complex_b = self.parse_complex(self.complex_b.get())
        if complex_b == 0:
            self.result_label.config(text="Помилка: Ділення на нуль")
        else:
            result = complex_a / complex_b
            self.result_label.config(text=f"Результат: {result}")

    def complex_magnitude(self, complex_entry):
        complex_number = self.parse_complex(complex_entry.get())
        result = abs(complex_number)
        self.result_label.config(text=f"Результат: {result}")

    def complex_argument(self, complex_entry):
        complex_number = self.parse_complex(complex_entry.get())
        result = cmath.phase(complex_number)
        self.result_label.config(text=f"Результат: {result} радіан")

    def complex_conjugate(self, complex_entry):
        complex_number = self.parse_complex(complex_entry.get())
        result = complex_number.conjugate()
        self.result_label.config(text=f"Результат: {result}")
```

Клас FunctionGraphing – вікно введення та графічного представлення функції:

```
class FunctionGraphing(ttk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.configure(bg="#1E2A38")
        tk.Label(self, text="Графічне представлення функції", foreground="white", font=("Arial", 30), background="#1E2A38").pack(pady=20)
        tk.Label(self, text="Введіть функцію y=f(x):", foreground="white", font=("Arial", 20), background="#1E2A38").place(x=80, y=125)
        self.function_entry = tk.Entry(self, font=("Arial", 20), foreground="white", background="#1E2A38")
        self.function_entry.place(x=320, y=125, width=150)
        tk.Label(self, text="Введіть значення x:", foreground="white", font=("Arial", 20), background="#1E2A38").place(x=80, y=175)
        self.x_value_entry = tk.Entry(self, font=("Arial", 20), foreground="white", background="#1E2A38")
        self.x_value_entry.place(x=320, y=175, width=150)
        self.fig, self.ax = plt.subplots()
        self.canvas = FigureCanvasTkAgg(self.fig, master=self)
        self.canvas_widget = self.canvas.get_tk_widget()
        self.canvas_widget.place(x=0, y=300, width=1000)
        center_frame = tk.Frame(self, bg="#1E2A38")
        center_frame.place(x=500, y=75)
        button_canvas = tk.Canvas(center_frame, bg="#1E2A38", width=500, height=200, highlightthickness=0)
        button_canvas.grid(row=2, column=0, columnspan=3, pady=5)
```

```
        RoundedButton2(button_canvas, 270, 25, "Обчислити", self.calculate_and_plot)
        RoundedButton2(button_canvas, 270, 95, "Побудувати графік", self.plot_function)
        RoundedButton2(button_canvas, 270, 165, "Назад", lambda: controller.show_frame(StartPage))

    def plot_function(self):
        function_str = self.function_entry.get()
        try:
            x = np.linspace(-10, 10, 400)
            y = eval(function_str, {"x": x, "np": np})
            self.ax.clear()
            self.ax.plot(x, y)
            self.ax.grid(True)
            self.canvas.draw()
        except Exception as e:
            ttk.Label(self, text="Помилка: перевірте введену функцію").pack(pady=10)

    def calculate_and_plot(self):
        function_str = self.function_entry.get()
        x_value_str = self.x_value_entry.get()
        try:
            x_value = float(x_value_str)
            y_value = eval(function_str, {"x": x_value, "np": np})
            self.ax.scatter([x_value], [y_value], color='red')
            self.ax.text(x_value, y_value, f'({x_value:.2f}, {y_value:.2f})', fontsize=12, ha='right')
            self.canvas.draw()
        except Exception as e:
            ttk.Label(self, text="Помилка: перевірте введені дані").pack(pady=10)
```

Клас LimitsCalculation – вікно введення та обчислення лімітів:

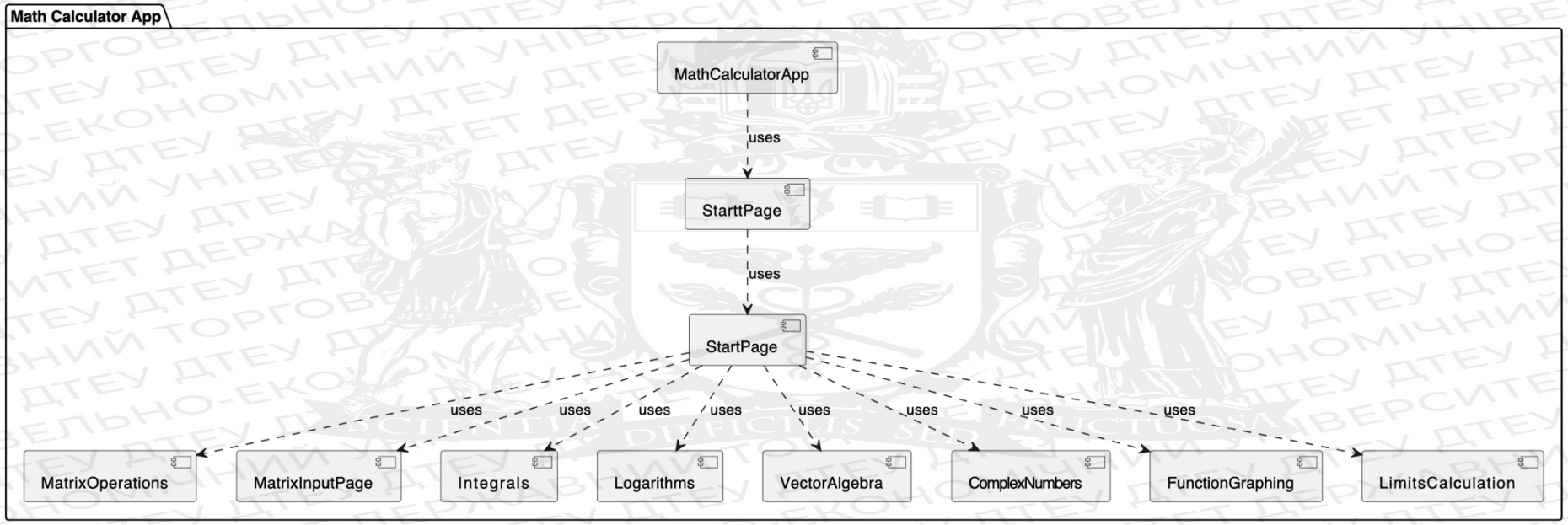
```
class LimitsCalculation(ttk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.configure(bg="#1E2A38")
        tk.Label(self, text="Розрахунок лімітів", foreground="white", font=("Arial", 40), background="#1E2A38").pack(
            pady=20)
        tk.Label(self, text="lim", foreground="white", font=("Arial", 40), background="#1E2A38").place(x=300, y=120)
        self.function_entry = tk.Entry(self, font=("Arial", 20), foreground="white", background="#1E2A38")
        self.function_entry.place(x=430, y=135, width=100)
        tk.Label(self, text="x → ", foreground="white", font=("Arial", 40), background="#1E2A38").place(x=270, y=170)
        self.x_value_entry = tk.Entry(self, font=("Arial", 20), foreground="white", background="#1E2A38")
        self.x_value_entry.place(x=350, y=185, width=50)
        tk.Label(self, text="=", font=("Arial", 45), foreground="white", background="#1E2A38").place(x=550, y=125)
        self.result_label = tk.Label(self, font=("Arial", 35), foreground="white", background="#1E2A38")
        self.result_label.place(x=570, y=125, width=200)
        center_frame = tk.Frame(self, bg="#1E2A38")
        center_frame.place(x=0, y=300)
        button_canvas = tk.Canvas(center_frame, bg="#1E2A38", width=1000, height=500, highlightthickness=0)
        button_canvas.grid(row=2, column=0, columnspan=3, pady=5)
        RoundedButton2(button_canvas, 500, 50, "Обчислити ліміт", self.calculate_limit)
        RoundedButton2(button_canvas, 500, 120, "Назад", lambda: controller.show_frame(StartPage))

    def calculate_limit(self):
        function_str = self.function_entry.get()
        x_value_str = self.x_value_entry.get()

        try:
            x = sp.symbols('x')
            function = sp.sympify(function_str)
            x_value = float(x_value_str)

            limit = sp.limit(function, x, x_value)
            if limit.is_real:
                if limit % 1 == 0:
                    display_limit = int(limit)
                else:
                    display_limit = round(float(limit), 3)
            else:
                display_limit = limit
            self.result_label['text'] = f'{display_limit}'
        except Exception as e:
            self.result_label['text'] = "Помилка"
app = MathCalculatorApp()
app.mainloop()
```

Діаграма компонентів



Додаток В

Діаграма активностей