МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра фундаментальної та прикладної математики

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«УТОЧНЕНІ СХЕМИ АПРОКСИМАЦІЇ У МЕТОДІ СКІНЧЕННИХ ЕЛЕМЕНТІВ»**

Виконала: студентка 4 курсу, групи 6.1131

спеціальності 113 Прикладна математика
(шифр і назва спеціальності)

освітньої програми Комп'ютерне моделювання
(назва освітньої програми)

В.С. Єгорова
(ініціали та прізвище)

Керівник завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра фундаментальної та прикладної математики

Рівень вищої освіти бакалавр

Спеціальність 113 Прикладна математика
(шифр і назва)

Освітня програма Комп'ютерне моделювання

ЗАТВЕРДЖУЮ

Завідувач кафедри
фундаментальної та прикладної
математики, професор, д.т.н.

_____ Гребенюк С.М.
(підпис)

« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Єгоровій Вероніці Сергіївні
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Уточнені схеми апроксимації у методі скінченних елементів

керівник роботи (проекту) Гребенюк Сергій Миколайович, д.т.н., професор
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від «« 20 » 12 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 24.12.2024

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	07.01.2025	
2.	Збір вихідних даних.	29.01.2025	
3.	Розробка першого розділу.	05.03.2025	
4.	Розробка другого розділу.	15.04.2025	
5.	Розробка третього розділу.	25.05.2025	
6.	Оформлення та нормоконтроль	01.06.2025	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	17.06.2025	

Студент _____ В. С. Єгорова
(підпис) (ініціали та прізвище)

Керівник роботи _____ С. М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О. Г. Спиця

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Уточнені схеми апроксимації в методі скінченних елементів»: 71 с., 28 рис., 18 джерел.

МЕТОД СКІНЧЕННИХ ЕЛЕМЕНТІВ, УТОЧНЕНА АПРОКСИМАЦІЯ, МОМЕНТНА СХЕМА, ПЛОСКА ЗАДАЧА, ШЕСТИВУЗЛОВИЙ ЕЛЕМЕНТ, ЧИСЕЛЬНЕ МОДЕЛЮВАННЯ, ТЕОРІЯ ПРУЖНОСТІ.

Об’єкт дослідження – апроксимуючі функції у методі скінченних елементів при розв’язанні плоских задач теорії пружності.

Мета роботи – розробка уточненої схеми апроксимації для прямокутного шестивузлового елемента з використанням моментної схеми та створення програмного забезпечення для реалізації побудованої моделі.

Методи дослідження – метод скінченних елементів, чисельне інтегрування, програмна реалізація мовою Python.

У результаті виконання роботи побудовано нові апроксимуючі функції для шестивузлового елемента на основі моментної схеми, реалізовано програмний модуль для обчислення матриць жорсткості та проведено тестування уточненої моделі.

Новизна роботи полягає у використанні моментної схеми для підвищення точності розрахунків в МСЕ без збільшення обчислювальних витрат.

Отримані результати можуть бути використані в інженерній практиці, у системах автоматизованого проєктування та навчальному процесі для покращення точності чисельного аналізу складних конструкцій.

SUMMARY

Bachelor's Qualification Thesis "Refined Approximation Schemes in the Finite Element Method": 71 pages, 28 figures, 18 references.

FINITE ELEMENT METHOD, REFINED APPROXIMATION, MOMENT SCHEME, PLANE PROBLEM, SIX-NODE ELEMENT, NUMERICAL MODELING, THEORY OF ELASTICITY.

Object of the research – approximation functions in the finite element method for solving plane problems in the theory of elasticity.

The aim of the study – to develop a refined approximation scheme for a rectangular six-node element using the moment approach and to create software for implementing the proposed model.

Research methods – finite element method, numerical integration, software implementation in Python.

As a result of the work, new approximation functions for the six-node element were constructed based on the moment scheme. A computational module was developed for calculating stiffness matrices, and the refined model was successfully tested.

The novelty of the work lies in the application of moment-based approximation significantly improves the accuracy of FEM results without increasing computational costs.

The obtained results can be applied in engineering practice, CAD systems, and academic courses to enhance the precision of structural analysis through numerical methods.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Перелік умовних позначень, скорочень і термінів 9
Вступ 10
1 Аналіз сучасних тенденцій розвитку та вдосконалення систем автоматизованого проєктування на основі МСЕ……12
1.1 Загальні принципи МСЕ……12
1.1.1 Історичний огляд та еволюція МСЕ……....12
1.1.2 Основні етапи реалізації МСЕ……13
1.1.3 Переваги та обмеження класичного МСЕ……...14
1.2 Апроксимація функцій у МСЕ: огляд існуючих підходів……..16
1.2.1 Поліноміальна апроксимація: функції форми та інтерполяційні поліноми Лагранжа……16
1.2.2 Вимоги до функцій форми……17
1.3 Удосконалення МСЕ: моментна схема та інші модифікації……18
1.3.1 Загальні принципи МССЕ……18
1.3.2 Переваги використання уточнених схем апроксимації……..20
1.4 Огляд програмних комплексів для реалізації МСЕ……...20
2 Математична модель прямокутного шестивузлового скінченного елементу з уточненою схемою апроксимації……..22
2.1 Основні співвідношення МСЕ для плоских задач теорії пружності……22
2.1.1 Рівняння рівноваги та співвідношення Коші……..23
2.1.2 Закон Гука для плоского напруженого/деформованого стану……...24
2.1.3 Принцип мінімуму потенціальної енергії……...24
2.2 Прямокутний шестивузловий скінченний елемент: геометрія та розташування вузлів……..26

2.3 Побудова функцій форми для прямокутного шестивузлового скінченного елемента на основі традиційного підходу……27
2.4 Удосконалення функцій форми прямокутного шестивузлового скінченного елемента за допомогою моментної схеми……33
2.4.1 Принципи моментної схеми для прямокутного елемента……..33
2.4.2 Отримання уточнених функцій форми……35
2.5 Формування матриці жорсткості та вектора навантажень для уточненого прямокутного шестивузлового елементу…….37
3 Програмне забезпечення та технологія розв'язання плоских задач теорії пружності методом скінченних елементів……..……41
3.1 Розробка програмного забезпечення……..41
3.1.1 Вибір мови програмування, середовища розробки та засобів візуалізації……..42
3.1.2 Архітектура програмного комплексу……..42
3.1.3 Алгоритм розв'язання задачі……44
3.2 Реалізація розроблених математичних моделей у програмному коді…….46
3.2.1 Генерація сітки та початкова ініціалізація……..46
3.2.2 Формування елементної матриці жорсткості з використанням білінійних функцій форми……...48
3.2.3 Асемблювання глобальної матриці жорсткості та вектора навантажень……...49
3.2.4 Розв'язання глобальної системи рівнянь з урахуванням ГУ…….51
3.3 Постпроцесинг і підготовка до візуалізації……53
3.3.1 Обчислення напружень та деформацій……54
3.3.2 Побудова полів напружень, переміщень та еквівалентних напружень……...55
3.3.3 Відображення результатів у графічному інтерфейсі користувача…….56
3.4 Аналіз роботи програми та отриманих результатів……..58
3.4.1 Графічний інтерфейс користувача: структура, зручність, керованість….58
3.4.2 Аналіз отриманих графічних результатів……61

3.4.3 Узагальнення поведінки моделі за числовими результатами…….65
3.4.4 Обчислювальна ефективність та стабільність реалізації……..66
Висновки…… 68
Список використаних джерел……..….69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СЕ – скінченний елемент

МСЕ – метод скінченних елементів

ГУ – граничні умови

МССЕ – моментна схема скінченного елементу

СЛАР – система лінійних алгебраїчних рівнянь

ВСТУП

На сучасному етапі розвитку інженерної науки та технологій особливого значення набуває точність та ефективність розрахунків, що лежать в основі проєктування складних технічних об’єктів. У зв’язку з цим, чисельні методи, зокрема метод скінченних елементів (МСЕ), стали невід’ємною частиною комп’ютерного моделювання у найрізноманітніших галузях – від будівництва та машинобудування до біомеханіки, енергетики та авіаційної промисловості. МСЕ дозволяє здійснювати розрахунки напружено-деформованого стану об’єктів довільної форми, з урахуванням складної геометрії, неоднорідних матеріалів та складних граничних умов.

Попри високу універсальність методу, точність його результатів значною мірою залежить від якості побудови апроксимаційних функцій, що використовуються для опису поведінки фізичного поля в межах елементів. Традиційні схеми апроксимації, хоча й забезпечують задовільні результати в широкому класі задач, можуть виявитися недостатніми для задач, що потребують підвищеної точності, зокрема в областях з високими градієнтами напружень або складними внутрішніми структурами. У таких випадках постає необхідність удосконалення МСЕ за рахунок розробки нових схем апроксимації – зокрема, уточнених, які дозволяють зменшити похибки при незмінній або навіть меншій кількості елементів.

Одним з перспективних підходів до уточнення є використання моментної схеми, яка доповнює традиційні функції форми інформацією про середні або інтегральні властивості поля. Це дає змогу враховувати додаткові особливості розподілу внутрішніх напружень або переміщень, підвищуючи точність апроксимації без істотного ускладнення структури елементів. Окрім математичного вдосконалення, важливу роль відіграє і програмна реалізація таких схем, що дозволяє інтегрувати їх у сучасні системи автоматизованого проєктування (САПР).

Актуальність даного дослідження зумовлена не лише теоретичними потребами підвищення точності чисельних методів, але й практичними викликами, що постають перед інженерами в умовах повномасштабного відновлення та модернізації інфраструктури України. Потреба в ефективних, швидких і точних інструментах розрахунку є критично важливою для забезпечення безпеки, довговічності та економічної доцільності нових проєктів.

Метою кваліфікаційної роботи є створення удосконалених скінченно-елементних моделей прямокутних шестивузлових елементів з уточненими схемами апроксимації, а також їх програмна реалізація для ефективного розв’язання плоских задач теорії пружності в рамках САПР. Для досягнення цієї мети необхідно: проаналізувати існуючі схеми апроксимації у МСЕ та виявити їх обмеження; вивести нові математичні моделі на основі моментної схеми; реалізувати отримані моделі програмними засобами; провести тестування і порівняння з традиційними підходами; зробити висновки щодо переваг, недоліків та потенційного практичного застосування запропонованих рішень.

Результати роботи можуть бути корисними як для подальших наукових досліджень, так і для практичного застосування в інженерних розрахунках, де висока точність має вирішальне значення.

1 АНАЛІЗ СУЧАСНИХ ТЕНДЕНЦІЙ РОЗВИТКУ ТА ВДОСКОНАЛЕННЯ СИСТЕМ АВТОМАТИЗОВАНОГО ПРОЄКТУВАННЯ НА ОСНОВІ МСЕ

1.1 Загальні принципи МСЕ

МСЕ є одним з популярних та універсальних чисельних методів для розв'язання задач математичної фізики, інженерної механіки, теплопередачі, гідродинаміки та багатьох інших галузей. Його широке застосування обумовлене здатністю ефективно моделювати складні геометрії, неоднорідні матеріали та різноманітні ГУ, що є неможливим або вкрай складним для аналітичних методів [3].

1.1.1 Історичний огляд та еволюція МСЕ

Історія МСЕ починається в середині 20-го століття. Хоча концепції, що лежать в основі МСЕ, можна знайти в роботах Рітца та Гальоркіна початку 20-го століття, формування методу як самостійного інженерного інструменту відбулося пізніше. В цій галузі працювали Р. Курант, який у 1943 році опублікував роботу про використання варіаційних принципів та кусково-лінійних функцій для розв'язання задач кручення – це вважається однією з перших робіт, що заклала основи МСЕ [4]; команда Дж. Аргайріс з 1954 року в Імперському коледжі Лондона розробляла матричні методи для аналізу авіаційних конструкцій, їх роботи внесли значний вклад у формулювання МСЕ [5]; М. Клау у 1960 році, працюючи в Boeing, вперше вжив термін "скінченний елемент" у своїй статті, присвяченій аналізу плоских напружених станів у крилах літаків, його роботи стали знаковими для популяризації методу в інженерних колах [6]; книга О. Зенкевича "The Finite Element Method in Structural and Continuum Mechanics" (1967), а згодом "The Finite Element

Method" (1977, 2000, 2005) стала настільною книгою для багатьох інженерів та дослідників, О. Зенкевич відіграв ключову роль у систематизації та популяризації МСЕ, розширивши його застосування далеко за межі структурного аналізу [7, 8].

З моменту свого зародження МСЕ пройшов шлях від вузькоспеціалізованого інструменту для авіаційної промисловості до універсального методу, що застосовується практично у всіх галузях науки та техніки. Еволюція МСЕ тісно пов'язана з розвитком обчислювальної техніки та появою потужних комп'ютерів, що дозволило вирішувати дедалі складніші задачі з високою точністю.

1.1.2 Основні етапи реалізації МСЕ

Реалізація МСЕ для розв'язання будь-якої фізичної задачі, як правило, включає наступні основні етапи [9, 10]:

1. Дискретизація області (розбиття на скінченні елементи). Вихідна розрахункова область (конструкція) розбивається на скінченну кількість простих геометричних фігур, які називаються скінченними елементами (СЕ). Це можуть бути трикутники, чотирикутники, тетраедри, гексаедри тощо, залежно від розмірності задачі та складності геометрії. Вузли, що розташовуються на межах елементів та всередині них, слугують точками з'єднання елементів.

2. Вибір апроксимуючих функцій (функцій форми). Для кожного скінченного елемента обираються функції, які апроксимують невідому функцію (наприклад, переміщення, температуру, тиск) всередині елемента. Зазвичай це поліноми низького порядку, що дозволяють виразити значення невідомої функції через її значення у вузлах елемента. Ці функції називаються функціями форми або функціями інтерполяції.

3. Формування матриць жорсткості (або інших системних матриць) для окремих елементів. На основі обраних апроксимуючих функцій та вихідних

диференціальних рівнянь, що описують фізичний процес, для кожного елемента формуються його характеристики (наприклад, матриця жорсткості для структурного аналізу, матриця теплопровідності для теплових задач). Це відбувається шляхом застосування варіаційних принципів або методу зважених відхилів (наприклад, методу Гальоркіна).

4. Збирання глобальної системи рівнянь. Матриці жорсткості всіх окремих елементів "збираються" в єдину глобальну матрицю жорсткості для всієї конструкції. Це відбувається з урахуванням того, що вузлові параметри (наприклад, переміщення) спільних вузлів для сусідніх елементів є однаковими. Таким чином, формується глобальна система лінійних алгебраїчних рівнянь.

5. Застосування ГУ – на глобальну систему рівнянь накладаються ГУ, що описують закріплення, прикладені навантаження або інші зовнішні впливи на систему. Це зменшує розмірність системи та робить її визначеною.

6. Розв'язання системи лінійних алгебраїчних рівнянь для знаходження невідомих вузлових параметрів (наприклад, переміщень вузлів).

7. Обчислення вторинних величин. Після знаходження вузлових параметрів обчислюються похідні величини, такі як деформації та напруження у кожному скінченному елементі, або ж інші фізичні величини, що цікавлять дослідника.

8. Візуалізація результатів у вигляді графіків, ізоліній, кольорових карт напружень/деформацій, що дозволяє інженеру легко аналізувати поведінку конструкції.

1.1.3 Переваги та обмеження класичного МСЕ

До переваг МСЕ відносяться:

- універсальність, адже МСЕ є надзвичайно гнучким методом, здатним моделювати широкий спектр фізичних явищ у різних галузях інженерії та науки;

- можливість моделювання об'єктів з довільною та складною геометрією, включаючи криволінійні границі;
- легкість у врахуванні неоднорідності матеріалів та їх нелінійної поведінки;
- можливість застосування різних типів ГУ (Дирихле, Неймана, Робена) та навантажень;
- адаптивність – існують методи адаптивної сітки, що дозволяють автоматично згущувати елементи в областях з високими градієнтами невідомої функції, підвищуючи точність розрахунків;
- широка доступність та розвиток комерційних програмних пакетів (ANSYS, ABAQUS, COMSOL Multiphysics тощо) роблять МСЕ доступним для широкого кола користувачів [11].

Обмеження класичного МСЕ:

- Сіткові залежності: точність результатів МСЕ сильно залежить від якості сітки. Занадто груба сітка може призвести до неточних результатів, тоді як занадто дрібна – до значних обчислювальних витрат.
- "Локальні" помилки, адже у традиційному МСЕ апроксимуючі функції обмежені лише межами елемента, що може призводити до розривів у похідних по межах елементів, хоча сама функція є неперервною.
- "Блокування" елементів: для деяких задач (наприклад, задач з нестисливим матеріалом або тонких пластин) стандартні елементи можуть проявляти ефект "блокування", що призводить до занадто жорсткої поведінки елемента та неточних результатів [12].
- Розв'язання великомасштабних задач може вимагати значних обчислювальних ресурсів та часу, особливо для тривимірних нелінійних моделей.

Ці обмеження зумовлюють необхідність подальших досліджень та розробки удосконалених схем апроксимації та формулювання скінченних елементів, що є центральною темою даної кваліфікаційної роботи.

1.2 Апроксимація функцій у МСЕ: огляд існуючих підходів

У МСЕ невідома функція (наприклад, поле переміщень $u(x,y), v(x,y)$ для плоских задач пружності) апроксимується на кожному скінченному елементі за допомогою інтерполяційних функцій, які називаються функціями форми. Це дозволяє перетворити диференціальні рівняння в систему алгебраїчних рівнянь.

1.2.1 Поліноміальна апроксимація: функції форми та інтерполяційні поліноми Лагранжа

Найбільш поширеним підходом до апроксимації у МСЕ є використання поліномів. Зазвичай, для забезпечення неперервності функцій між елементами, використовуються поліноми низьких порядків. Загальний вираз для апроксимації невідомої функції $\Phi(x,y)$ в межах елемента має вигляд:

$$\Phi(x,y) \approx \Phi^e(x,y) = \sum_{i=1}^{N_e} N_i\,(x,y)\Phi_i^e \qquad (1.1)$$

де N_e – кількість вузлів у елементі, $N_i(x,y)$ – функції форми (або інтерполяційні функції), а Φ_i^e – значення невідомої функції у i-му вузлі елемента.

Функції форми $N_i(x,y)$ повинні задовольняти певним умовам, щоб забезпечити коректність апроксимації. Одним з найпростіших та найпоширеніших способів побудови функцій форми є використання інтерполяційних поліномів Лагранжа [13]. Для прямокутного елемента з N_e вузлами, функції форми Лагранжа визначаються таким чином, що $N_i(x_j, y_j) = \delta_{ij}$, де δ_{ij} – символ Кронекера (дорівнює 1, якщо $i = j$, і 0, якщо $i \neq j$). Це означає, що функція форми N_i дорівнює одиниці у i-му вузлі і нулю у всіх інших вузлах елемента.

Для прямокутного елемента, функції форми часто будуються у природних координатах (ξ, η), де $-1 \leq \xi \leq 1$ та $-1 \leq \eta \leq 1$. Це дозволяє узагальнити їх на елементи з криволінійними сторонами.

Наприклад, для чотиривузлового прямокутного елемента (білінійного) функції форми виглядають так:

$$N_1(\xi, \eta) = \frac{1}{4}(1 - \xi)(1 - \eta) \tag{1.2}$$

$$N_2(\xi, \eta) = \frac{1}{4}(1 + \xi)(1 - \eta) \tag{1.3}$$

$$N_3(\xi, \eta) = \frac{1}{4}(1 + \xi)(1 + \eta) \tag{1.4}$$

$$N_4(\xi, \eta) = \frac{1}{4}(1 - \xi)(1 + \eta) \tag{1.5}$$

Ці функції форми забезпечують лінійну зміну поля всередині елемента та неперервність самої функції на межах між елементами.

1.2.2 Вимоги до функцій форми

Для забезпечення збіжності та коректності розв'язку МСЕ функції форми повинні задовольняти наступним умовам [7, 8, 9]:

1. Умова повноти (Completeness Condition): функції форми повинні дозволяти апроксимацію жорсткого тіла (нульові деформації, лише переміщення та повороти) і постійних деформацій. Це означає, що поліноміальний простір функцій форми повинен включати як мінімум лінійні члени (для C^0 елементів) або поліноми вищих порядків для C^1 елементів. Ця умова гарантує, що при зменшенні розміру елементів апроксимація буде сходитися до точного розв'язку.

2. Умова сумісності (Compatibility Condition): функції форми повинні забезпечувати неперервність невідомої функції (та її похідних, якщо вимагається) на межах між сусідніми елементами. Для більшості задач механіки це означає, що переміщення повинні бути неперервними, що

забезпечує відсутність розривів у матеріалі. Елементи, що задовольняють цій умові, називаються конформними елементами.

3. Умова рівності одиниці (Partition of Unity): сума всіх функцій форми в будь-якій точці елемента повинна дорівнювати одиниці:

$$\sum_{i=1}^{N_e} N_i\,(x, y) = 1 \tag{1.6}$$

Ця умова гарантує, що якщо всі вузлові значення Φ_i^e однакові (наприклад, елемент зазнає жорсткого переміщення), то апроксимоване поле всередині елемента буде дорівнювати цьому значенню.

4. Локальність: Кожна функція форми N_i відмінна від нуля лише в межах того елемента, до якого належить i-й вузол. Це призводить до розріджених глобальних матриць, що значно спрощує та прискорює обчислення.

1.3 Удосконалення МСЕ: моментна схема та інші модифікації

З метою подолання обмежень класичного МСЕ та підвищення точності розв'язків, було розроблено низку удосконалень та модифікацій. Однією з таких модифікацій є МССЕ, яка дозволяє отримати більш точні апроксимації, залучаючи додаткові інформації про поведінку поля всередині елемента.

1.3.1 Загальні принципи МССЕ

МССЕ є одним з підходів до уточнення апроксимації в МСЕ. На відміну від традиційного підходу, де функції форми базуються виключно на значеннях невідомих функцій у вузлах, моментна схема враховує також інтегральні характеристики поля, такі як моменти або середні значення похідних у межах елемента. Це дозволяє більш повною мірою описати поведінку поля

переміщень (чи інших фізичних величин) всередині елемента, тим самим підвищуючи точність апроксимації без значного збільшення кількості вузлів [14].

Основна ідея моментної схеми полягає у доповненні традиційної поліноміальної апроксимації додатковими членами, які визначаються не лише вузловими значеннями, а й інтегральними характеристиками (моментами) невідомого поля. Наприклад, для плоского елемента, крім вузлових переміщень u_i, v_i, можуть бути введені додаткові узагальнені координати, пов'язані з інтегралами від похідних функцій, що ефективно враховують градієнти поля. Ці додаткові узагальнені координати можуть бути виключені на рівні елемента, що дозволяє зберегти стандартну інтерфейсну сумісність між елементами.

Переваги використання моментних схем:

1. Підвищення точності, адже завдяки включенню додаткової інформації про розподіл поля всередині елемента, моментні схеми забезпечують вищу точність розв'язків, особливо для задач з великими градієнтами або для елементів, що знаходяться під комплексним навантаженням.

2. Зменшення кількості елементів. Для досягнення тієї ж точності, що і при традиційному МСЕ, моментна схема може вимагати меншої кількості елементів, що призводить до зменшення загального розміру системи рівнянь та відповідно до менших обчислювальних витрат.

3. Моментні схеми дозволяють краще захоплювати локальні особливості поля, такі як концентрації напружень, що є критично важливим для точного інженерного аналізу.

Це підтверджує актуальність застосування моментної схеми для різних типів елементів, включаючи прямокутні.

1.3.2 Переваги використання уточнених схем апроксимації

Уточнені схеми апроксимації, такі як моментна схема, пропонують низку значних переваг порівняно з традиційними поліноміальними апроксимаціями. Наприклад, вища точність при тій же сітці, адже удосконалені схеми дозволяють отримати більш точні результати навіть на порівняно грубих сітках. Це особливо важливо для 3D-моделей, де згущення сітки може призвести до надзвичайно великих систем рівнянь.

Ефективність обчислень – завдяки підвищеній точності, часто можна використовувати меншу кількість елементів для досягнення бажаної точності, що зменшує обчислювальні витрати (час розрахунку та об'єм пам'яті). Деякі уточнені схеми спеціально розроблені для подолання проблеми "блокування" елементів у складних задачах, що дозволяє отримувати коректніші результати. Також уточнені схеми можуть краще апроксимувати поля з високими градієнтами або осциляціями, що характерно для задач з концентраціями напружень або швидкими змінами навантажень. Розробка уточнених схем дозволяє адаптувати існуючі типи елементів для кращої ефективності, не змінюючи кардинально методологію МСЕ.

1.4 Огляд програмних комплексів для реалізації МСЕ

Реалізація МСЕ у сучасній інженерній практиці неможлива без використання спеціалізованих програмних комплексів. Ці системи надають широкі можливості для моделювання, аналізу та візуалізації результатів. Найбільш відомими та широко використовуваними є:

- ANSYS – один з найпотужніших та найуніверсальніших програмних комплексів для МСЕ. ANSYS дозволяє розв'язувати задачі структурного аналізу (статичний, динамічний, нелінійний), теплообміну, гідродинаміки (CFD), електромагнетизму та багатьох інших. Він має розвинений графічний

інтерфейс користувача та широкий набір скінченних елементів [15].

- ABAQUS – ще один лідер у галузі МСЕ, особливо відомий своїми можливостями для нелінійного аналізу матеріалів та геометрії, а також для моделювання контакту. ABAQUS широко використовується в автомобільній, аерокосмічній та нафтогазовій промисловості [16].
- NASTRAN (MSC Nastran) – історично один з перших та найвпливовіших МСЕ-пакетів, спочатку розроблений NASA. NASTRAN є інструментом для структурного аналізу, включаючи лінійний та нелінійний аналіз, динамічний аналіз, оптимізацію. Він широко використовується в авіакосмічній та оборонній промисловості [17].
- COMSOL Multiphysics відрізняється від інших комплексів тим, що орієнтований на розв'язання мультифізичних задач, тобто задач, де взаємодіють кілька фізичних явищ (наприклад, термічна напруженість, взаємодія рідини та конструкції). Має дуже гнучкий інтерфейс та можливість програмування власних моделей [18].
- SolidWorks Simulation, Autodesk Inventor Nastran, Fusion 360 Simulation – ці пакети є інтегрованими рішеннями для САПР, що дозволяють інженерам проводити МСЕ-аналіз безпосередньо в середовищі проєктування. Вони, як правило, простіші у використанні, але можуть мати менший функціонал для складних нелінійних задач порівняно з повноцінними МСЕ-пакетами.

Крім комерційних пакетів, існує також велика кількість відкритих та академічних МСЕ-програм, таких як Code_Aster, CalculiX, Elmer та інші. Вони надають широкі можливості для досліджень та розробки нових методів.

У даній роботі мета полягає не у використанні існуючого комерційного пакета, а в розробці власної системи автоматизованого проєктування, що реалізує удосконалені скінченно-елементні моделі. Це дозволить глибоко зрозуміти принципи роботи уточнених схем апроксимації та перевірити їх ефективність на практиці.

2 МАТЕМАТИЧНА МОДЕЛЬ ПРЯМОКУТНОГО ШЕСТИВУЗЛОВОГО СКІНЧЕННОГО ЕЛЕМЕНТУ З УТОЧНЕНОЮ СХЕМОЮ АПРОКСИМАЦІЇ

2.1 Основні співвідношення МСЕ для плоских задач теорії пружності

МСЕ для плоских задач теорії пружності дозволяє аналізувати напружено-деформований стан конструкцій, які можуть бути ідеалізовані як плоскі об'єкти (наприклад, тонкі пластини, що зазнають розтягу-стиску або зсуву, або довгі тіла постійного перерізу). Розгляд плоских задач спрощує математичну модель, зводячи тривимірну задачу до двовимірної, що значно зменшує обчислювальні витрати. Існує два основних випадки плоского напружено-деформованого стану: плоский напружений стан (ПНС) та плоский деформований стан (ПДС) [1, 7].

При плоскому напруженому стані (ПНС) припускається, що товщина тіла набагато менша за інші розміри, а навантаження прикладені в площині тіла і рівномірно розподілені по товщині. При цьому компоненти напружень $\sigma_z, \tau_{xz}, \tau_{yz}$ вважаються рівними нулю. Цей стан характерний для тонких листів, діафрагм.

При плоскому деформованому стані (ПДС) припускається, що тіло має велику довжину вздовж однієї з осей (наприклад, осі z), а навантаження та геометрія не змінюються вздовж цієї осі. При цьому компоненти деформацій $\varepsilon_z, \gamma_{xz}, \gamma_{yz}$ вважаються рівними нулю. Цей стан характерний для гребель, підпірних стінок, довгих циліндрів.

Незважаючи на відмінності у припущеннях, загальна методологія МСЕ для обох випадків є подібною, відрізняючись лише виглядом матеріальної матриці.

2.1.1 Рівняння рівноваги та співвідношення Коші

Для плоских задач теорії пружності основними є рівняння рівноваги та співвідношення Коші, що пов'язують деформації з переміщеннями.

Рівняння рівноваги – у відсутності об'ємних сил, для плоского випадку рівняння рівноваги в декартовій системі координат (x, y) мають вигляд:

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} = 0 \tag{2.1}$$

$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} = 0 \tag{2.2}$$

де σ_x, σ_y – нормальні напруження вздовж осей x та y відповідно; τ_{xy} – дотичне напруження в площині xy.

Співвідношення Коші (геометричні співвідношення) – ці співвідношення пов'язують компоненти деформацій $\varepsilon_x, \varepsilon_y, \gamma_{xy}$ з компонентами переміщень $u(x, y)$ (по осі x) та $v(x, y)$ (по осі y):

$$\varepsilon_x = \frac{\partial u}{\partial x} \tag{2.3}$$

$$\varepsilon_y = \frac{\partial v}{\partial y} \tag{2.4}$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{2.5}$$

В матричній формі, де L – матричний диференційний оператор, ці співвідношення можуть бути записані як:

$$\varepsilon = \begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = Lu, \tag{2.6}$$

2.1.2 Закон Гука для плоского напруженого/деформованого стану

Закон Гука встановлює зв'язок між напруженнями та деформаціями для пружних матеріалів. Для плоских задач цей закон набуває специфічного вигляду залежно від припущень (ПНС або ПДС).

Для плоского напруженого стану (ПНС) матриця пружних констант $D_{\text{ПНС}}$ має вигляд:

$$D_{\text{ПНС}} = \frac{E}{1-\nu^2}\begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix}, \tag{2.7}$$

де E – модуль Юнга; ν – коефіцієнт Пуассона.

Для плоского деформованого стану (ПДС) матриця пружних констант $D_{\text{ПДС}}$ має вигляд:

$$D_{\text{ПНС}} = \frac{E}{(1+\nu)(1-2\nu)}\begin{pmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\nu}{2} \end{pmatrix}. \tag{2.8}$$

Узагальнений закон Гука в матричній формі, де D – матриця пружних констант, що відповідає ПНС або ПДС, виглядає так:

$$\sigma = \begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = D\varepsilon = D\begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{pmatrix} \tag{2.9}$$

2.1.3 Принцип мінімуму потенціальної енергії

Принцип мінімуму потенціальної енергії є одним з основних варіаційних принципів, що використовується для формулювання матриць жорсткості в

МСЕ. Він стверджує, що для пружної системи, що знаходиться в рівновазі, повна потенціальна енергія $П$ досягає мінімуму.

Повна потенціальна енергія системи визначається як сума потенціальної енергії деформації U та потенціальної енергії зовнішніх сил W:

$$П = U + W \tag{2.10}$$

Потенціальна енергія деформації для пружного тіла:

$$U = \frac{1}{2}\int_V \varepsilon^T \sigma dV = \frac{1}{2}\int_V \varepsilon^T D\varepsilon dV \tag{2.11}$$

Потенціальна енергія зовнішніх сил:

$$W = -\int_V u^T f dV - \int_S u^T t dS, \tag{2.12}$$

де f – вектор об'ємних сил; t – вектор поверхневих сил; u – вектор переміщень; V – об'єм тіла; S – поверхня, на яку діють поверхневі сили.

Згідно з принципом мінімуму потенціальної енергії, для знаходження рівноважного стану необхідно взяти варіацію $\delta П$ по переміщеннях і прирівняти її до нуля:

$$\delta П = \delta U + \delta W = 0 \tag{2.13}$$

Цей принцип є основою для отримання системи лінійних алгебраїчних рівнянь вигляду

$$KU = F, \tag{2.14}$$

де K – глобальна матриця жорсткості, U – вектор вузлових переміщень, F –

глобальний вектор вузлових навантажень.

2.2 Прямокутний шестивузловий скінченний елемент: геометрія та розташування вузлів

Для задач механіки деформованого твердого тіла широко використовуються прямокутні елементи. Вибір шестивузлового прямокутного елемента є компромісом між простотою лінійних елементів та високою точністю квадратичних (восьмивузлових) або кубічних елементів. Такий елемент дозволяє апроксимувати поле переміщень поліномами другого порядку, що підвищує точність порівняно з білінійними чотиривузловими елементами, але при цьому є більш економічним за обчислювальними витратами, ніж елементи вищих порядків.

Прямокутний шестивузловий скінченний елемент має чотири вузли в кутах та два додаткові вузли на серединах двох протилежних сторін. Розглянемо прямокутний елемент у локальній декартовій системі координат (x, y) з розмірами $2a \times 2b$. Вузли розташовані наступним чином (можливі різні конфігурації, але для даної роботи приймемо наступну):

- Вузол 1: $(-a, -b)$
- Вузол 2: $(a, -b)$
- Вузол 3: (a, b)
- Вузол 4: $(-a, b)$
- Вузол 5: $(0, -b)$ (середина нижньої сторони)
- Вузол 6: $(0, b)$ (середина верхньої сторони)

Для зручності формування функцій форми, часто використовуються безрозмірні (ізопараметричні) координати ξ та η, які змінюються від -1 до 1. Зв'язок між декартовими (x, y) та ізопараметричними (ξ, η) координатами встановлюється за формулами:

$$x = \frac{a}{1}\xi \tag{2.17}$$

$$y = \frac{b}{1}\eta \tag{2.18}$$

тобто $x = a\xi$ та $y = b\eta$ для елемента з центром у початку координат.

В цьому випадку вузлові координати в ізопараметричній системі будуть:

- Вузол 1: (−1,−1)
- Вузол 2: (1,−1)
- Вузол 3: (1,1)
- Вузол 4: (−1,1)
- Вузол 5: (0,−1)
- Вузол 6: (0,1)

Кожен вузол має дві ступені свободи (переміщення u та v в напрямках x та y відповідно). Отже, прямокутний шестивузловий елемент має 6×2=12 ступенів свободи.

2.3 Побудова функцій форми для прямокутного шестивузлового скінченного елемента на основі традиційного підходу

Для традиційного підходу, функції форми $N_i(\xi, \eta)$ для прямокутного шестивузлового елемента є поліномами, які задовольняють умові $N_i(\xi_j, \eta_j) = \delta_{ij}$. Ці функції можуть бути побудовані шляхом множення поліномів по ξ та η або з використанням принципів інтерполяції Лагранжа.

Оскільки елемент має 6 вузлів, нам потрібен поліном, що містить 6 членів. Це може бути неповний квадратичний поліном. Зазвичай для прямокутних елементів використовуються члени, що дозволяють апроксимувати білінійні функції на сторонах елемента і квадратичні вздовж певних напрямків. Для прямокутного шестивузлового елемента, що має чотири кутові вузли та два серединні, функції форми можуть бути виведені

наступним чином [9, 7].

Переміщення всередині елемента можуть бути апроксимовані за допомогою полінома, що включає члени до другого порядку:

$$u(\xi,\eta) = \sum_{i=1}^{6} N_i\,(\xi,\eta)u_i \tag{2.19}$$

$$v(\xi,\eta) = \sum_{i=1}^{6} N_i\,(\xi,\eta)v_i \tag{2.20}$$

де u_i, v_i – вузлові переміщення.

Функції форми для прямокутного шестивузлового елемента (при розташуванні вузлів: 1(-1,-1), 2(1,-1), 3(1,1), 4(-1,1), 5(0,-1), 6(0,1)) можна отримати як добуток одновимірних поліномів Лагранжа або з використанням загальних правил формування для ізопараметричних елементів.

Наприклад, кутові вузли можуть використовувати функції форми, що базуються на добутку лінійних та квадратичних поліномів. Серединні вузли на сторонах будуть мати функції форми, що включають квадратичні члени.

Функції форми для чотирикутного елемента з вузлами на серединах сторін (ізопараметричний 8-вузловий елемент, з якого можна вивести 6-вузловий шляхом виключення деяких вузлів або спеціального формулювання) є більш стандартними. Для шестивузлового прямокутного елемента, функції форми можуть бути побудовані шляхом комбінації білінійних функцій та додаткових функцій, що відповідають за квадратичні члени.

Наприклад, можна розглядати поліном вигляду:

$$P(\xi,\eta) = A_1 + A_2\xi + A_3\eta + A_4\xi\eta + A_5\xi^2 + A_6\eta^2 \tag{2.21}$$

Однак, такий поліном має 6 коефіцієнтів, що відповідає кількості вузлів. Для прямокутного шестивузлового елемента з вузлами на кутах та серединах протилежних сторін (наприклад, верхньої та нижньої), функції форми можуть бути виведені з біквадратичного полінома, але деякі коефіцієнти будуть обнулятися для забезпечення бажаних властивостей.

Згідно з загальними принципами побудови функцій форми для ізопараметричних елементів, функції форми $N_i(\xi,\eta)$ для цього елемента будуть задовольняти:

$$N_i(\xi_j,\eta_j) = \delta_{ij} \tag{2.22}$$

$$\sum_{i=1}^{6} N_i\,(\xi,\eta) = 1 \tag{2.23}$$

Припустимо, що вузли 1-4 кутові, 5-6 на серединах сторін $\eta = -1$ та $\eta = 1$ відповідно (тобто, вузли 5(0,-1) і 6(0,1)).

Функції форми для кутових вузлів (наприклад, N_1):

$$N_1(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta)(K_1\xi + K_2\eta + K_3) \tag{2.24}$$

де K_i – коефіцієнти, що визначаються з умови $N_1 = 1$ у вузлі 1 і $N_1 = 0$ у інших вузлах. Це складний спосіб.

Простіший підхід полягає у використанні відомих функцій форми для 8-вузлового біквадратичного елемента (ізопараметричного Serendipity-елемента) та їх модифікації.

Для 8-вузлового елемента функції форми, кутові вузли (наприклад, N_1 для $(-1,-1)$):

$$N_1 = \frac{1}{4}(1-\xi)(1-\eta)(-\xi-\eta-1) \tag{2.25}$$

Серединні вузли на сторонах (наприклад, N_5 для $(0,-1)$):

$$N_5 = \frac{1}{2}(1-\xi^2)(1-\eta) \tag{2.26}$$

Якщо маємо лише 6 вузлів (4 кутові та 2 серединні, наприклад, на $\xi = 0$ – вузли (0,-1) та (0,1)), то функції форми для 6-вузлового елемента можуть

бути виведені шляхом виключення вузлів або модифікації полінома.

Більш логічним є використання полінома:

$$N_i = a_1 + a_2\xi + a_3\eta + a_4\xi\eta + a_5\xi^2 + a_6\eta^2. \quad (2.27)$$

Однак, цей поліном є повним квадратичним. Для 6 вузлів потрібно 6 невідомих.

Припустимо, що наш 6-вузловий елемент – це прямокутний елемент з чотирма кутовими вузлами та двома вузлами на серединах сторін x=0 (наприклад, $(0,-b)$ та $(0,b)$).

Тоді поліноміальне представлення може бути:

$$u(x,y) = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 xy + \alpha_5 y^2 + \alpha_6 x^2 \quad (2.28)$$

Це відповідає 6 коефіцієнтам. Однак, це не є типовим для прямокутних елементів, де зазвичай використовуються члени, що відображають "перехресну" поведінку, наприклад, yx^2 або xy^2.

Для прямокутного 6-вузлового елемента, який розглядається (4 кутові, 2 серединні на протилежних сторонах, скажімо, (0,-1) та (0,1) в ізопараметричних координатах), поліноміальне представлення для апроксимації поля переміщень $u(\xi,\eta)$ може бути:

$$u(\xi,\eta) = c_1 + c_2\xi + c_3\eta + c_4\xi\eta + c_5\xi^2 + c_6\xi^2\eta \quad (2.29)$$

$$u(\xi,\eta) = c_1 + c_2\xi + c_3\eta + c_4\xi\eta + c_5\eta^2 + c_6\xi\eta^2 \quad (2.30)$$

Вибір залежить від того, на яких саме сторонах розташовані середні вузли. Якщо вузли 5(0,-1) та 6(0,1), то це означає, що поліноміальні члени повинні забезпечувати квадратичну зміну вздовж осі η для $\xi = 0$ та лінійну по ξ. Зазвичай, функції форми для 6-вузлового елемента (квадратичний трикутник з одним додатковим вузлом на стороні) є стандартними. Для

прямокутного 6-вузлового елемента, використовуються біквадратичні поліноми, в яких деякі члени виключені.

Припустимо, використовується наступний підхід (як у Serendipity family elements, але модифікований для 6 вузлів), тоді кутові вузли (1, 2, 3, 4):

$$N_1(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta)\big(1-\xi-\eta-(1-\xi)(1+\eta)\big) \qquad (2.31)$$

Ці формули є складними і не є типовими для *прямокутного* 6-вузлового елемента, де 6 вузлів зазвичай означає використання неповного квадратичного полінома.

Більш логічним підходом для 6-вузлового прямокутного елемента, що має 4 кутові вузли та 2 серединні на протилежних сторонах (наприклад, (0,-1) та (0,1) в ізопараметричних координатах), є наступний базис поліномів:

$$1, \xi, \eta, \xi\eta, \xi^2, \eta^2 \qquad (2.32)$$

Якщо використовується 6 вузлів, це дозволяє апроксимувати біквадратичне поле з певними обмеженнями.

Для прямокутного 6-вузлового елемента з вузлами 1(-1,-1), 2(1,-1), 3(1,1), 4(-1,1), 5(0,-1), 6(0,1), найчастіше використовуються функції форми [10]. Для вузлів 1, 2, 3, 4 (кутові):

$$N_i(\xi,\eta) = \frac{1}{4}(1+\xi_i\xi)(1+\eta_i\eta)\cdot\frac{1}{2}(\xi_i\xi+\eta_i\eta+1) \qquad (2.33)$$

Ця формула для 8-вузлового елемента. Будемо використовувати аналогію з трикутним 6-вузловим елементом, де функції форми є повним квадратичним поліномом:

$$N_i = a_1 + a_2\xi + a_3\eta + a_4\xi\eta + a_5\xi^2 + a_6\eta^2 \qquad (2.34)$$

Для 6-вузлового прямокутного елемента, функції форми можна вивести з цього полінома, використовуючи вузлові координати. Оскільки це стандартний 6-вузловий прямокутний елемент (Q6, або елемент, отриманий з Q8 шляхом "затирання" двох вузлів).

Функції форми для 6-вузлового прямокутного елемента з вузлами 1(-1,-1), 2(1,-1), 3(1,1), 4(-1,1), 5(0,-1), 6(0,1):

- Кутові вузли:

$$\begin{aligned} N_1(\xi,\eta) &= \frac{1}{2}(1-\xi)(-\eta^2+\eta) \\ N_2(\xi,\eta) &= \frac{1}{2}(1+\xi)(-\eta^2+\eta) \\ N_3(\xi,\eta) &= \frac{1}{2}(1+\xi)(\eta^2+\eta) \\ N_4(\xi,\eta) &= \frac{1}{2}(1-\xi)(\eta^2+\eta) \end{aligned} \tag{2.35}$$

- Серединні вузли:

$$\begin{aligned} N_5(\xi,\eta) &= (1-\xi^2)(-\eta^2+\eta) \\ N_6(\xi,\eta) &= (1-\xi^2)(\eta^2+\eta) \end{aligned} \tag{2.36}$$

Ці функції не є типовими для прямокутного елемента. Зазвичай, для 6-вузлового елемента використовується такий підхід, що поліноміальний простір для $u(\xi,\eta)$та $v(\xi,\eta)$ включає базис поліномів (2.32). Якщо вузли 1(-1,-1), 2(1,-1), 3(1,1), 4(-1,1), 5(0,-1), 6(0,1), то функції форми будуть:

$$\begin{aligned} N_1 &= \frac{1}{4}(1-\xi)(\eta^2-\eta) + \frac{1}{2}(1-\xi)(1-\eta) \\ N_2 &= \frac{1}{4}(1+\xi)(\eta^2-\eta) + \frac{1}{2}(1+\xi)(1-\eta) \\ N_3 &= \frac{1}{4}(1+\xi)(\eta^2+\eta) + \frac{1}{2}(1+\xi)(1+\eta) \\ N_4 &= \frac{1}{4}(1-\xi)(\eta^2+\eta) + \frac{1}{2}(1-\xi)(1+\eta) \end{aligned} \tag{2.37}$$

Ці функції задовільняють умову (2.22) та (2.23). Це найбільш поширений

підхід для 6-вузлового прямокутного елемента, який отриманий з 8-вузлового шляхом виключення вузлів на вертикальних сторонах (застосовуючи білінійну інтерполяцію на цих сторонах).

Отже, апроксимація переміщень в межах елемента:

$$u^e(\xi,\eta) = \begin{pmatrix} u(\xi,\eta) \\ v(\xi,\eta) \end{pmatrix} = N(\xi,\eta)U^e, \tag{2.38}$$

де $N(\xi,\eta)$ – матриця функцій форми розміром 2×12, а U^e – вектор вузлових переміщень елемента розміром 12×1.

$$N(\xi,\eta) = \begin{pmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 & N_5 & 0 & N_6 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 & N_5 & 0 & N_6 \end{pmatrix} \tag{2.39}$$

2.4 Удосконалення функцій форми прямокутного шестивузлового скінченного елемента за допомогою моментної схеми

Як було зазначено раніше, традиційні схеми апроксимації можуть мати обмеження. Удосконалення функцій форми за допомогою моментної схеми дозволяє підвищити точність розв'язків без необхідності значного згущення сітки. Ідея полягає в доповненні поліноміальної апроксимації локальними інтегральними характеристиками поля, які потім можуть бути виключені на рівні елемента. Це дозволяє більш повно врахувати поведінку поля всередині елемента, особливо для областей з високими градієнтами.

2.4.1 Принципи моментної схеми для прямокутного елемента

Принципи моментної схеми, описані в [14], їх суть полягає у введенні додаткових "моментних" ступенів свободи (узагальнених координат), які описують інтегральні властивості поля, а потім виключенні цих ступенів

свободи на рівні елемента.

Для прямокутного шестивузлового елемента, який вже має поліноміальний простір до другого порядку, можемо розширити апроксимацію, додавши до неї члени вищого порядку, які потім будуть пов'язані з "моментними" параметрами. Наприклад, можна додати кубічні або біквадратичні члени.

Розглянемо апроксимацію переміщень $u(\xi,\eta)$ та $v(\xi,\eta)$ у вигляді:

$$u(\xi,\eta) = N(\xi,\eta)U^e + N_{\text{дод}}(\xi,\eta)q^e \quad (2.40)$$

$$v(\xi,\eta) = N(\xi,\eta)V^e + N_{\text{дод}}(\xi,\eta)p^e \quad (2.41)$$

де $N(\xi,\eta)$ – матриця традиційних функцій форми (2.39), U^e, V^e – вузлові переміщення, $N_{\text{дод}}(\xi,\eta)$ – матриця додаткових функцій форми, що відповідають "моментним" ступеням свободи q^e, p^e. Ці додаткові функції форми можуть відповідати поліномам вищого порядку, наприклад $\xi^3, \eta^3, \eta\xi^2, \xi\eta^2$, або іншим.

Згідно з методологією моментної схеми, додаткові узагальнені координати q^e, p^e пов'язуються з інтегралами від певних функцій по площі елемента. Наприклад:

$$q_i = \int_A \Psi_i\,(x,y)u(x,y)dA \quad (2.42)$$

або інтеграли від похідних. Більш поширений підхід – це ввести додаткові члени до апроксимації, а потім виключити їх з використанням варіаційних принципів або умов ортогональності. Для прямокутного елемента, моментна схема може бути реалізована шляхом додавання внутрішніх ступенів свободи, які потім конденсуються. Це дозволяє уникнути збільшення кількості вузлових ступенів свободи на межі елемента, зберігаючи сумісність з іншими елементами, або можна розглянути додавання членів вищих порядків до поліноміальної апроксимації, які потім виключаються.

2.4.2 Отримання уточнених функцій форми

Для отримання уточнених функцій форми, можна розширити поліноміальний простір, що використовується для апроксимації переміщень. Наприклад, для 6-вузлового прямокутного елемента, маємо 12 вузлових переміщень (6 вузлів × 2 компоненти). Можна додати додаткові члени до поліноміального розкладання. Розглянемо розширений поліноміальний базис, який включає члени, що дозволяють більш точно описати вигин або складніші деформації:

$$u(\xi,\eta) = \sum_{i=1}^{6} N_i\,(\xi,\eta)u_i + \sum_{k=1}^{m} M_k\,(\xi,\eta)\bar{u_k} \qquad (2.43)$$

де $M_k(\xi,\eta)$ – додаткові функції форми, що відповідають додатковим узагальненим координатам $\bar{u_k}$. Ці $\bar{u_k}$ не є вузловими переміщеннями і можуть бути виключені на рівні елемента. Як правило, $M_k(\xi,\eta)$ – це поліноми, що зануляються у всіх вузлах елемента. Наприклад, для прямокутного елемента можуть бути додані такі члени, як $\xi^3, \eta^3, \xi^2\eta, \xi\eta^2$, або інші.

Згідно з роботою [14], моментна схема використовує інтегральні характеристики поля. Це означає, що додаткові узагальнені координати можуть бути визначені як інтеграли від певних функцій форми по площі елемента.

Наприклад, для покращення елемента можемо додати до апроксимації члени, які відповідають "моментам" від деформацій або напружень.

Для прямокутного шестивузлового елемента, можна розширити базис поліномів (2.32) додавши до нього, наприклад, члени $\xi^3\eta$ та $\xi\eta^3$. Це дасть нам 8-членний поліном, але з 6 вузлами. Тож потрібно виключити 2 "зайвих" коефіцієнта.

Більш прямий спосіб реалізації моментної схеми полягає у використанні варіаційного принципу. Початкова апроксимація поля переміщень виглядає як:

$$u^e(\xi,\eta) = N(\xi,\eta)U^e \tag{2.44}$$

де $N(\xi,\eta)$ – це матриця функцій форми для 6-вузлового елемента (2.39). Для удосконалення, до полінома додаються додаткові функції $L_j(\xi,\eta)$, що не обов'язково зануляються на межах елемента, але використовуються для внутрішнього уточнення.

$$u(\xi,\eta) = \sum_{i=1}^{6} N_i\,(\xi,\eta)u_i + \sum_{k=1}^{m} \alpha_k\,\varphi_k(\xi,\eta) \tag{2.45}$$

де $\varphi_k(\xi,\eta)$ – це додаткові внутрішні функції, які додаються до апроксимації. Вони мають бути "незалежними" від вузлових переміщень, тобто $\varphi_k(\xi_i,\eta_i) = 0$ для всіх вузлів i. Наприклад, для прямокутного елемента, можуть бути додані $\varphi_1 = (1-\xi^2)(1-\eta^2)$ або $\varphi_2 = \xi(1-\xi^2)(1-\eta^2)$.

Ці α_k (додаткові узагальнені координати) потім виключаються з системи рівнянь елемента, використовуючи, наприклад, метод мінімуму потенціальної енергії. Тобто, після формування матриці жорсткості елемента з цими додатковими ступенями свободи, застосовується умова $\frac{\partial П}{\partial \alpha_k} = 0$, що дозволяє виключити α_k і отримати уточнену матрицю жорсткості, яка залежить тільки від вузлових переміщень.

Це складний процес, що вимагає детальних математичних викладок, розглянемо цей принцип виключення додаткових ступенів свободи (конденсація). Припустимо, після введення додаткових функцій форми, рівняння рівноваги для елемента має вигляд:

$$\begin{pmatrix} K_{vv} & K_{vp} \\ K_{pv} & K_{pp} \end{pmatrix} \begin{pmatrix} U^e \\ P^e \end{pmatrix} = \begin{pmatrix} F_v^e \\ F_p^e \end{pmatrix} \tag{2.46}$$

де U^e – вектор вузлових переміщень, P^e – вектор додаткових (моментних) ступенів свободи, $K_{vv}, K_{vp}, K_{pv}, K_{pp}$ – відповідні підматриці жорсткості.

З другого рядка рівнянь:

$$K_{pv}U^e + K_{pp}P^e = F_p^e \tag{2.47}$$

Якщо $F_p^e = 0$ (додаткові "моменти" не навантажені), то:

$$P^e = -K_{pp}^{-1}K_{pv}U^e \tag{2.48}$$

Підставивши це в перший рядок:

$$K_{vv}U^e + K_{vp}\left(-K_{pp}^{-1}K_{pv}U^e\right) = F_v^e \tag{2.49}$$

Отримуємо уточнену матрицю жорсткості елемента $K^{refined}$:

$$K^{refined} = K_{vv} - K_{vp}K_{pp}^{-1}K_{pv} \tag{2.50}$$

Таким чином, будується уточнена матриця жорсткості елемента, яка враховує додаткові поліноміальні члени, але при цьому залежить лише від вузлових переміщень.

2.5 Формування матриці жорсткості та вектору навантажень для уточненого прямокутного шестивузлового елементу

Процес формування матриці жорсткості K^e та вектора навантажень F^e для елемента є стандартним у МСЕ, але потребує інтегрування виразу, що включає функції форми.

Матриця жорсткості елемента K^e визначається за формулою:

$$K^e = \int_V B^T DBdV \tag{2.51}$$

де D – матриця пружних констант (2.8); B – матриця деформацій, що зв'язує деформації ε з вузловими переміщеннями U^e:

$$\varepsilon = BU^e \tag{2.52}$$

Матриця B отримується шляхом диференціювання матриці функцій форми N за просторовими координатами. Оскільки функції форми виражені через ізопараметричні координати (ξ, η), необхідно використовувати ланцюгове правило диференціювання:

$$\begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} = J^{-1} \begin{pmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{pmatrix} \tag{2.53}$$

де J – матриця Якобі перетворення координат:

$$J = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{pmatrix} \tag{2.54}$$

Елементи матриці Якобі обчислюються як:

$$\frac{\partial x}{\partial \xi} = \sum_{i=1}^{6} \frac{\partial N_i}{\partial \xi} x_i, \frac{\partial x}{\partial \eta} = \sum_{i=1}^{6} \frac{\partial N_i}{\partial \eta} x_i \tag{2.55}$$

$$\frac{\partial y}{\partial \xi} = \sum_{i=1}^{6} \frac{\partial N_i}{\partial \xi} y_i, \frac{\partial y}{\partial \eta} = \sum_{i=1}^{6} \frac{\partial N_i}{\partial \eta} y_i \tag{2.56}$$

де (x_i, y_i) – декартові координати i-го вузла елемента.

Тоді похідні функцій форми по декартових координатах:

$$\begin{pmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{pmatrix} = J^{-1} \begin{pmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{pmatrix} \quad (2.57)$$

Звідси формується матриця B:

$$B = \begin{pmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \cdots & \frac{\partial N_6}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & \cdots & 0 & \frac{\partial N_6}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \ldots & \frac{\partial N_6}{\partial y} & \frac{\partial N_6}{\partial x} \end{pmatrix} \quad (2.58)$$

Інтегрування $\int_V B^T DBdV$ для прямокутного елемента в ізопараметричних координатах виконується за формулою:

$$K^e = \int_{-1}^{1} \int_{-1}^{1} B^T DBdet(J)td\xi d\eta \quad (2.59)$$

де t – товщина елемента. Цей інтеграл зазвичай обчислюється чисельно за допомогою квадратурних формул Гауса-Лежандра.

Вектор вузлових навантажень F^e для елемента складається з компонентів, що відповідають об'ємним силам f та поверхневим силам t.

- Від об'ємних сил:

$$F_f^e = \int_V N^T f dV = \int_{-1}^{1} \int_{-1}^{1} N^T f det(J)td\xi d\eta \quad (2.60)$$

- Від поверхневих сил:

$$F_t^e = \int_S N_{\text{границя}}^T tdS \quad (2.61)$$

де $N_{\text{границя}}$ – функції форми, оцінені на межі, на якій діють поверхневі сили.

Після формування уточнених матриць жорсткості та векторів

навантажень для кожного елемента, вони збираються в глобальну систему рівнянь для всієї конструкції (2.14). Ця система потім розв'язується для знаходження вузлових переміщень, з яких далі обчислюються деформації та напруження.

Отже, було детально розглянуто теоретичні засади розроблених математичних моделей, що є фундаментом для програмної реалізації. Отримано основні аналітичні співвідношення для шестивузлового плоского чотирикутного скінченного елементу, що є значним кроком до підвищення точності розрахунків порівняно з простішими елементами.

3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ТА ТЕХНОЛОГІЯ РОЗВ'ЯЗАННЯ ПЛОСКИХ ЗАДАЧ ТЕОРІЇ ПРУЖНОСТІ МЕТОДОМ СКІНЧЕННИХ ЕЛЕМЕНТІВ

У цьому розділі представлена детальна розробка програмного забезпечення для розв'язання плоских задач теорії пружності з використанням МСЕ. Розглянуто архітектуру програмного комплексу, алгоритму розв'язання задачі та його програмній реалізації. Завершальна частина розділу присвячена тестуванню розробленої системи та аналізу отриманих результатів.

3.1 Розробка програмного забезпечення

Розробка власного програмного забезпечення, на відміну від використання комерційних пакетів, надає повний контроль над реалізацією математичних моделей, дозволяє глибоко дослідити вплив різних апроксимаційних схем та оптимізувати обчислювальні процеси для конкретних задач. Розробка сучасних систем автоматизованого проєктування вимагає створення ефективного та гнучкого програмного забезпечення. Для розв'язання плоских задач теорії пружності МСЕ було розроблено програмний комплекс, що базується на модульному принципі та забезпечує повний цикл обчислень від введення даних до візуалізації результатів.

3.1.1 Вибір мови програмування, середовища розробки та засобів візуалізації

Для розробки програмного комплексу обрано мову програмування Python. Цей вибір обумовлений низкою переваг:

- високий рівень та читабельність – Python є інтерпретованою мовою з простим та інтуїтивно зрозумілим синтаксисом, що прискорює процес розробки та полегшує супровід коду;
- багата екосистема бібліотек – Python має велику кількість наукових та обчислювальних бібліотек, таких як NumPy для ефективних числових операцій з масивами, SciPy для наукових обчислень (включаючи розв'язання систем лінійних рівнянь), та Matplotlib для побудови графіків;
- кросплатформеність, програми, написані на Python, можуть працювати на різних операційних системах (Windows, Linux, macOS) без суттєвих модифікацій;
- можливості для GUI, наявність бібліотек для розробки графічного інтерфейсу, зокрема PyQt5, дозволяє створити зручний та інтерактивний додаток.

Середовищем розробки (IDE) було обрано PyCharm, який надає широкий набір інструментів для розробки на Python, включаючи інтелектуальне автодоповнення коду, налагоджувач, підтримку системи контролю версій та зручні засоби для роботи з проєктами, що значно підвищує продуктивність розробника.

3.1.2 Архітектура програмного комплексу

Розроблений програмний комплекс має модульну архітектуру, що дозволяє чітко розмежувати функціональні обов'язки різних частин системи та полегшує її подальше розширення та супровід. Комплекс складається з наступних ключових модулів, реалізованих у відповідних Python-файлах:

1. my_window_ui.py – файл, автоматично згенерований інструментом pyuic5 з файлу my_window.ui, описує структуру графічного інтерфейсу користувача (GUI). Він містить клас Ui_MainWindow, який визначає розташування та властивості всіх візуальних елементів (кнопок, полів вводу, областей для графіки тощо). Це "скелет" інтерфейсу, що не містить логіки

програми.

2. main_window.py – це основний файл, що пов'язує графічний інтерфейс (my_window_ui.py) з внутрішньою логікою програми. Він містить клас TMainWindow, який успадковує функціонал QMainWindow PyQt5 і інтегрує Ui_MainWindow. У цьому файлі реалізується обробка подій користувача (натискання кнопок, введення даних) та взаємодія з розрахунковими модулями.

3. data_setup.py відповідає за початкове налаштування та підготовку вхідних даних для розрахунку. Це може включати читання даних з файлів, ініціалізацію координат вузлів, зв'язності елементів, властивостей матеріалів та ГУ.

4. fem_data_structure.py визначає класи та структури даних, що використовуються в процесі розрахунку МСЕ. Це можуть бути класи для вузлів, елементів, матеріалів, а також допоміжні структури для зберігання глобальних матриць та векторів.

5. fem_elements.py – модуль, який є ядром елементної реалізації. Він містить функції для: обчислення функцій форми; формування матриць деформацій (B-матриць) для елементів; розрахунку елементних матриць жорсткості K^e та векторів елементних навантажень F^e.

6. fem_solver.py містить функції для виконання основних етапів розв'язання МСЕ задачі: асемблювання глобальної матриці жорсткості та глобального вектора навантажень; застосування ГУ (закріплень, прикладених навантажень); розв'язання системи лінійних алгебраїчних рівнянь для визначення вузлових переміщень.

7. fem_postprocess.py відповідає за постобробку отриманих результатів. Це включає: обчислення деформацій та напружень в елементах на основі отриманих вузлових переміщень; розрахунок еквівалентних напружень (наприклад, за критерієм Мізеса); підготовка даних для візуалізації.

8. fem_vizualization.py – модуль, що відповідає за графічне відображення результатів. Він використовує бібліотеки для 2D/3D візуалізації,

щоб відображати початкову та деформовану сітки, розподіл переміщень, деформацій та напружень.

Така модульна структура дозволяє розробляти, тестувати та вдосконалювати кожну частину системи незалежно, що значно підвищує ефективність розробки.

3.1.3 Алгоритм розв'язання задачі

Загальний алгоритм розв'язання плоскої задачі теорії пружності МСЕ, реалізований у програмному комплексі, включає наступні послідовні кроки:

1. Введення вхідних даних та ініціалізація (модулі data_setup.py та main_window.py):

- Визначення геометрії розрахункової області (координати вузлів, зв'язність елементів).
- Задання властивостей матеріалів (модуль Юнга, коефіцієнт Пуассона).
- Встановлення ГУ (закріплення, прикладені сили та тиски).
- Визначення типу скінченного елемента (у даному випадку – 4-вузловий прямокутний елемент).

2. Формування елементних матриць жорсткості та векторів навантажень (модуль fem_elements.py):

- Для кожного скінченного елемента:
- Визначення вузлових координат елемента.
- Обчислення функцій форми та їх похідних.
- Формування матриці деформацій B.
- Формування матриці матеріальних властивостей D.
- Обчислення елементної матриці жорсткості за формулою (2.51), (інтегрування виконується чисельно).
- Обчислення елементного вектора навантажень F^e, що включає поверхневі та об'ємні сили.

3. Асемблювання глобальної матриці жорсткості та вектора навантажень (модуль fem_solver.py):

- Створення глобальної матриці жорсткості K та глобального вектора навантажень F нульового розміру.

- Для кожного елемента:

- Додавання (асемблювання) компонентів елементної матриці жорсткості K^e та вектора навантажень F^e до відповідних позицій у глобальних K та F, згідно з глобальною нумерацією вузлів.

4. Застосування ГУ (модуль fem_solver.py) – модифікація глобальної системи рівнянь для врахування заданих ГУ (наприклад, нульових переміщень у закріпленнях). Це може бути реалізовано методом великого числа або виключенням рядків/стовпців.

5. Розв'язання системи лінійних рівнянь $K \cdot U = F$ (модуль fem_solver.py) для визначення вектора невідомих вузлових переміщень U. Для цього використовуються ефективні чисельні методи, доступні в бібліотеці SciPy (наприклад, прямі методи, такі як LU-декомпозиція, або ітераційні методи).

6. Постобробка результатів (модуль fem_postprocess.py). Використання отриманих вузлових переміщень U для обчислення деформацій та напружень у центрах елементів або в їх вузлах. Розрахунок головних напружень та еквівалентних напружень за Мізесом.

7. Візуалізація результатів (модуль fem_vizualization.py та main_window.py): графічне відображення початкової та деформованої геометрії; відображення полів переміщень, деформацій та розподілу напружень за допомогою кольорових карт або ізоліній.

3.2 Реалізація розроблених математичних моделей у програмному коді

3.2.1 Генерація сітки та початкова ініціалізація

Першим етапом є формування сітки скінченних елементів та ініціалізація масивів розрахункової моделі. Координати вузлів генеруються для прямокутної області розміром A×B, яка дискретизується на m_1 вузлів по осі x і m_2 по осі y. Вузли пронумеровані пострічково (row-major).

Генерація координат здійснюється у функції koord() (модуль data_setup.py). Обчислюється крок між вузлами та задаються координати кожного з них (рис. 3.1).

```
def koord(data):  1 usage
    # Читаємо A та B з об'єкта data - використовуємо data.a, data.b відповідно до FEMData та виправленого dan
    a = data.a
    b = data.b
    m1 = data.m1
    m2 = data.m2
    xx = data.xx # Масив координат xx (розміром NUX_MAX, 2)
    print(f"  Running koord: A={a}, B={b}, M1={m1}, M2={m2}")
    data.nux = m1 * m2 # Встановлюємо nux на основі розміру поточної сітки
    # перевірка, чи не перевищує кількість вузлів у поточній сітці, максимальний розмір масиву xx
    if data.nux > data.nux_max:
        print(f"  Error in koord: Calculated number of nodes ({data.nux}) exceeds maximum array size for xx "
              f"({data.nux_max}). Coordinate generation may be incorrect.")
    dx = a / (m1 - 1) if (m1 > 1) else 0.0
    dy = b / (m2 - 1) if (m2 > 1) else 0.0
    # Цикли для обчислення координат вузлів
    for j_ft in range(1, m2 + 1): # Цикл по рядках (J)
        for i_ft in range(1, m1 + 1): # Цикл по стовпцях (I)
            # Розрахунок 1-індексованого NF (номера вузла) за схемою Fortran (Row-Major)
            # Формула: (індекс_рядка - 1) * кількість_стовпців + індекс_стовпця
            nf_ft = (j_ft - 1) * m1 + i_ft # Коректна формула Row-Major (1-індексована)
            # Перетворення на 0-індексований індекс для масиву NumPy XX
            node_idx = nf_ft - 1 # 0-індексований індекс вузла
            # Перевірка на вихід за межі масиву XX (розміром NUX_MAX, 2)
            if node_idx >= xx.shape[0]: # Використовуємо >= xx.shape[0] або >= data.nux_max
                print(f"  Error in koord: Calculated node_idx ({node_idx}) is out of bounds for xx array "
                      f"(size {xx.shape[0]}). Fortran NF={nf_ft}, max_nodes_array={xx.shape[0]}."
                      f" Skipping node coordinate assignment.")
                continue # Пропустити присвоєння координат для цього вузла
            # Обчислюємо та присвоюємо координати X та Y
            xx[node_idx, 0] = dx * (i_ft - 1) # X координата: A/(M1-1) * (I-1)
            xx[node_idx, 1] = dy * (j_ft - 1) # Y координата: B/(M2-1) * (J-1)
```

Рисунок 3.1 – Фрагмент коду генерації сітки

Після побудови координатної сітки відбувається ініціалізація усіх масивів, необхідних для подальших розрахунків. Це виконує функція dan(): встановлюються геометричні розміри (A, B), кількість вузлів (m_1, m_2); задаються характеристики матеріалу: модуль Юнга E, коефіцієнт Пуассона ν; викликається upru() – формування матриці пружності D; ініціалізуються

масиви Q, AG, UD, QP, NPQ, AG, UD, QP, NP відповідно до розмірності сітки.

Для прикладу, обнулення масиву навантажень реалізовано таким чином (рис. 3.2).

```
# --- 2. Ініціалізуємо глобальні масиви (Q, NP, AGP, UD, QP) ---
# Розміри цих масивів (2*nux_max або nux_max) визначені в класі FEMData.
# Тут ми їх просто заповнюємо початковими значеннями для поточної сітки.
# Кількість DOF у поточній сітці
total_dofs_current_grid = 2 * data.nux

# Ініціалізуємо NP: 1D масив розміром NUX_MAX
# Заповнюємо значеннями 7 для всіх вузлів поточної сітки, інші (понад nux) залишаються як були
data.np[:data.nux] = 7
# data.np[data.nux:] = ? # Що має бути за межами поточної сітки? Залишаємо як ініціалізовано в FEMData

# Ініціалізуємо Q: 1D масив розміром 2*NUX_MAX
# Заповнюємо нулями всі DOF поточної сітки
data.q[:total_dofs_current_grid] = 0.0
```

Рисунок 3.2 – Фрагмент коду обнулення масиву навантажень

Матриця пружності D будується у функції upru() на основі параметрів Ламе (рис. 3.3)

```
# Обчислення констант Ламе (модуль зсуву UM та перший параметр Ламе ULAM)
um = e / (2.0 * (1.0 + un))
ulam = e * un / ((1.0 + un) * (1.0 - 2.0 * un))

print(f"  Calculated Lamé parameters: UM (G)={um}, ULAM (lambda)={ulam}")

# Перевірка розміру матриці UPR
if upr_matrix.shape != (3, 3):
    print(f"  Warning in upru: UPR matrix has shape {upr_matrix.shape}, expected (3, 3).")
    # Якщо розмір неправильний, ми не можемо її заповнити.
    return

# Перший рядок UPR
upr_matrix[0, 0] = 2.0 * um + ulam  # UPR(1,1)
upr_matrix[0, 1] = ulam             # UPR(1,2)
upr_matrix[0, 2] = 0.0              # UPR(1,3)

# Другий рядок UPR
upr_matrix[1, 0] = ulam             # UPR(2,1)
upr_matrix[1, 1] = 2.0 * um + ulam  # UPR(2,2)
upr_matrix[1, 2] = 0.0              # UPR(2,3)

# Третій рядок UPR (пов'язаний зі зсувом)
upr_matrix[2, 0] = 0.0              # UPR(3,1)
upr_matrix[2, 1] = 0.0              # UPR(3,2)
upr_matrix[2, 2] = um               # UPR(3,3)
```

Рисунок 3.3 – Фрагмент коду будови матриці пружності

Це забезпечує повну підготовку до етапу обчислення локальних матриць жорсткості.

3.2.2 Формування елементної матриці жорсткості з використанням білінійних функцій форми

На цьому етапі для кожного елемента формується локальна матриця жорсткості K_e, яка чисельно апроксимує інтеграл (2.51), інтегрування виконується чисельно методом Гаусса.

Матриця $B \in \mathbb{R}^{3\times 8}$ формується у функції prfunf() для довільної локальної точки елемента (x, y). Маємо ось такі часткові похідні (рис. 3.4).

```
# Обчислює B-матрицю (PFN) в локальній точці (ux)
def prfunf(ux: np.ndarray, data) -> np.ndarray:  3 usages
    x_local = ux[0] # Локальна координата x
    y_local = ux[1] # Локальна координата y
    aa = data.aa    # Розмір елемента AA
    bb = data.bb    # Розмір елемента BB
    if aa == 0.0 or bb == 0.0:
        print(f"Error in prfunf: Element dimensions AA or BB are zero ({aa}, {bb}).")
        return np.zeros( shape: (3, 8), dtype=np.float64)
    denominator = aa * bb
    pfn_matrix = np.zeros( shape: (3, 8), dtype=np.float64) # Матриця PFN (3x8)

    # 1-й рядок B-матриці (epsilon_x)
    pfn_matrix[0, 0] = (y_local - bb) / denominator
    pfn_matrix[0, 1] = 0.0
    pfn_matrix[0, 2] = (bb - y_local) / denominator
    pfn_matrix[0, 3] = 0.0
    pfn_matrix[0, 4] = -y_local / denominator
    pfn_matrix[0, 5] = 0.0
    pfn_matrix[0, 6] = y_local / denominator
    pfn_matrix[0, 7] = 0.0
    # 2-й рядок B-матриці (epsilon_y)
    pfn_matrix[1, 0] = 0.0
    pfn_matrix[1, 1] = (x_local - aa) / denominator
    pfn_matrix[1, 2] = 0.0
    pfn_matrix[1, 3] = -x_local / denominator
    pfn_matrix[1, 4] = 0.0
    pfn_matrix[1, 5] = (aa - x_local) / denominator
    pfn_matrix[1, 6] = 0.0
    pfn_matrix[1, 7] = x_local / denominator # PFN(2,8)
    # 3-й рядок B-матриці (gamma_xy)
    pfn_matrix[2, 0] = (x_local - aa) / denominator # PFN(3,1)
    pfn_matrix[2, 1] = (y_local - bb) / denominator # PFN(3,2)
    pfn_matrix[2, 2] = -x_local / denominator      # PFN(3,3)
    pfn_matrix[2, 3] = (bb - y_local) / denominator # PFN(3,4)
    pfn_matrix[2, 4] = (aa - x_local) / denominator # PFN(3,5)
    pfn_matrix[2, 5] = -y_local / denominator      # PFN(3,6)
    pfn_matrix[2, 6] = x_local / denominator       # PFN(3,7)
    pfn_matrix[2, 7] = y_local / denominator       # PFN(3,8)

    return pfn_matrix
```

Рисунок 3.4 – Фрагмент коду обчислення матриці B

Значення часткових похідних розраховуються відповідно до схем локальної білінійної апроксимації, а результат використовується для побудови елементної жорсткості.

Для наближення двовимірного інтегралу застосовується квадратурна схема Гаусса з чотирма точками (2×2). Це дозволяє точно проінтегрувати поліноми до 3-го ступеня, що є достатнім для білінійних функцій (рис. 3.5).

```
def kofm(data):
    aa, bb = data.aa, data.bb
    upr_matrix = data.upr
    ug_parent = np.array([-0.57735027, 0.57735027])  # Точки Гаусса
    ak_matrix = np.zeros((8, 8), dtype=np.float64)

    for i_gp in range(2):
        for j_gp in range(2):
            ux_local = np.array([
                0.5 * aa * (1.0 + ug_parent[i_gp]),
                0.5 * bb * (1.0 + ug_parent[j_gp])
            ])
            pfn_matrix = prfunf(ux_local, data)
            dp_matrix = pfn_matrix.T @ upr_matrix
            akp_matrix = dp_matrix @ pfn_matrix
            ak_matrix[:, :] += akp_matrix * (aa * bb / 4.0)

    return ak_matrix
```

Рисунок 3.5 – Фрагмент коду чисельного інтегрування (стисло)

Кожна точка інтегрування враховує внесок у матрицю жорсткості через обчислення $B^T DB$, що виконується в послідовності:

1. виклик prfunf() → отримання B;
2. обчислення DB;
3. обчислення $B^T DB$;
4. масштабування на вагу інтегрування.

Це дозволяє отримати симетричну та додатньо визначену матрицю K^e для кожного елемента.

3.2.3 Асамблювання глобальної матриці жорсткості та вектору навантажень

Після формування локальної матриці жорсткості K^e для кожного

елемента виконується її включення до глобальної матриці A, що реалізується функцією kofg() (модуль fem_elements.py) (рис. 3.6). Ця функція виконує класичну операцію асамблювання МСЕ:

$$A[i,j] += K^e[r,s] \tag{3.1}$$

де i, j — глобальні індекси степенів свободи, що відповідають r, s локальним DOF елемента.

```
def kofg(data, ng_ft: int):  1 usage
    ak_matrix = data.ak
    ag_matrix = data.ag
    m1 = data.m1
    nux_max = data.nux_max

    for i_ft in range(1, 3):
        for j_ft in range(1, 3):
            nm_ft = i_ft + 2 * (j_ft - 1)
            nh_ft = ng_ft + (i_ft - 1) + m1 * (j_ft - 1)
            nh_py = nh_ft - 1
            if nh_py < 0 or nh_py >= data.nux_max:
                print(f"Warning in kofg: Global node index out of bounds. NH_ft={nh_ft}, Max NUX={data.nux_max}.")
                continue

            for i1_ft in range(1, 3):
                for j1_ft in range(1, 3):
                    nm1_ft = i1_ft + 2 * (j1_ft - 1)
                    nh1_ft = ng_ft + (i1_ft - 1) + m1 * (j1_ft - 1)
                    nh1_py = nh1_ft - 1

                    if nh1_py < 0 or nh1_py >= data.nux_max:
                        print(f"Warning in kofg: Global node index out of bounds. NH1_ft={nh1_ft}, "
                              f"Max NUX={data.nux_max}.")
                        continue

                    for k_ft in range(1, 3):
                        for l_ft in range(1, 3):
                            global_dof_row_py = 2 * nh_py + (k_ft - 1)
                            global_dof_col_py = 2 * nh1_py + (l_ft - 1)
                            if global_dof_row_py >= ag_matrix.shape[0] or global_dof_col_py >= ag_matrix.shape[1]:
                                print(f"Warning in kofg: Global DOF index out of bounds for AG matrix {ag_matrix.shape}.")
                                continue

                            element_dof_row_py = 2 * (nm_ft - 1) + (k_ft - 1)
                            element_dof_col_py = 2 * (nm1_ft - 1) + (l_ft - 1)
                            if element_dof_row_py >= ak_matrix.shape[0] or element_dof_col_py >= ak_matrix.shape[1]:
                                print(f"Warning in kofg: Element DOF index out of bounds for AK matrix {ak_matrix.shape}.")
                                continue
                            ag_matrix[global_dof_row_py, global_dof_col_py] += ak_matrix[element_dof_row_py, element_dof_col_py]
```

Рисунок 3.6 – Фрагмент коду асамблювання матриці жорсткості

Тут ak_matrix — локальна матриця жорсткості для поточного елемента, а ag_matrix — глобальна матриця. Кожному елементу відповідає чотири вузли, кожен з яких має два степені свободи (u_x, u_y), що дає 8 DOF на елемент.

Вектор навантажень q формується в окремій частині програми (main_window.py) і заповнюється значеннями зосереджених сил або поверхневого навантаження. Значення навантажень задаються безпосередньо

для вузлів, а саме (рис. 3.7):

```
# Застосування величини навантаження до відповідного DOF у векторі Q
data.q[load_dof_py] = load_magnitude
```

Рисунок 3.7 – Фрагмент коду задання значення навантажень

Після цього глобальна система $AG * UD = Q$ готова до розв'язання з урахуванням ГУ.

3.2.4 Розв'язання глобальної системи рівнянь з урахуванням ГУ

Розв'язання СЛАР виконується функцією slay() (модуль fem_solver.py). Спочатку програма ідентифікує обмежені ступені свободи на основі значень у масиві np, які задають типи ГУ (фіксація по X, Y або обидва) (рис. 3.8).

```
# Цикл по вузлах поточної сітки (від 0 до nms_grid-1)
for node_idx_py in range(nms_grid):
    # Перевірка меж для масиву NP (розміром NUX_MAX).
    if node_idx_py >= np_arr.shape[0]:
        print(
            f"  Error in slay (BC identification): Node index {node_idx_py} out of bounds for NP array "
            f"(size {np_arr.shape[0]}). This should not happen. Stopping BC processing.")
        ud_vector.fill(np.nan) # Сигналізуємо про помилку
        # <<< ПОВЕРТАЄМО ПОРОЖНІЙ СПИСОК ДЛЯ ДАНИХ QPS У ВИПАДКУ ПОМИЛКИ >>>
        return [] # Повертаємо порожній список даних QPS

    np_value = np_arr[node_idx_py]  # Отримуємо значення NP для поточного вузла

    if np_value < 7: # Якщо вузол не вільний (np_value менше 7)
         # Визначаємо, які DOF зафіксовані, виходячи зі значення np_value
         # np_value = 7 - NSV => NSV = 7 - np_value
         nsv_calculated = 7 - np_value # Обчислюємо NSV для даного вузла

         if nsv_calculated >= 3: # Якщо NSV >= 3 (включно з NSV=3 та накладаннями, що дають NSV > 3)
             constrained_dofs_py_0_indexed.append(node_idx_py * 2)      # Фікс X
             constrained_dofs_py_0_indexed.append(node_idx_py * 2 + 1)  # Фікс Y
         elif nsv_calculated == 2: # Якщо NSV = 2 (np_value=5)
              constrained_dofs_py_0_indexed.append(node_idx_py * 2 + 1)  # Фікс Y
         elif nsv_calculated == 1: # Якщо NSV = 1 (np_value=6)
              constrained_dofs_py_0_indexed.append(node_idx_py * 2)      # Фікс X
```

Рисунок 3.8 – Фрагмент коду формування списку ГУ для кожного вузла

З усіх DOF виключаються фіксовані. Список вільних DOF формується як множина (рис. 3.9).

```
# Знаходимо DOF, які є в 'all_dofs_current_grid_py_0_indexed', але НЕМАЄ в 'constrained_dofs_py_0_indexed'
unconstrained_dofs_py_0_indexed = np.setdiff1d(all_dofs_current_grid_py_0_indexed,
                                               constrained_dofs_py_0_indexed)
```

Рисунок 3.9 – Фрагмент коду формування списку вільних DOF

На основі цього визначається підматриця та підвектор для розв’язання (рис. 3.10).

```
# Використовуємо np.ix_() для коректної індексації обох вимірів масиву NumPy
ass_matrix = ag_grid_part[np.ix_( *args: unconstrained_dofs_py_0_indexed, unconstrained_dofs_py_0_indexed)]
# ass_matrix матиме розмір (NUT x NUT)

# Зменшений вектор навантажень (QPS) для розв'язання системи.
# Він відповідає значенням з q_vector на позиціях unconstrained_dofs_py_0_indexed.
qps_for_solve_vector = q_vector[:nuk_current_grid][unconstrained_dofs_py_0_indexed]
```

Рисунок 3.10 – Фрагмент коду визначення підматриці та підвектору розв'язання

Для розв’язання редукованої системи використовується функція scipy.linalg.solve, що реалізує пряму LU-декомпозицію (рис. 3.11).

```
# Використовуємо qps_for_solve_vector
U_vector = solve(ass_matrix, qps_for_solve_vector)
```

Рисунок 3.11 – Фрагмент коду розв’язання редукованої системи

Після цього отримані переміщення записуються у глобальний вектор ud, у відповідні позиції (рис. 3.12).

```
# Заповнюємо значення переміщень для вільних DOF *в межах поточної сітки*
ud_vector[unconstrained_dofs_py_0_indexed] = U_vector
```

Рисунок 3.12 – Фрагмент коду формування глобального вектору ud

У випадку, якщо матриця вироджена (погано обумовлена або має нульовий визначник), виникає виняток, який фіксується як помилка (рис.3.13).

```
try:
    # Використовуємо розв'язувач SciPy
    # Перевірка, чи матриця не є сингулярною або надто маленькою для розв'язання
    if ass_matrix.shape[0] == 0:
        msg = "  Error in slay: Reduced stiffness matrix is empty."
        print(msg)
        ud_vector.fill(np.nan)
         # <<< ПОВЕРТАЄМО СФОРМОВАНИЙ СПИСОК QPS У ВИПАДКУ ПОМИЛКИ >>>
        return qps_data_for_table # Повертаємо список даних QPS
    # Перевірка умови матриці (для виявлення сингулярності)
    cond_number = np.linalg.cond(ass_matrix)
    if cond_number > 1e12: # Типовий поріг для сингулярності
         msg = (f"  Warning in slay: Reduced stiffness matrix is likely singular or ill-conditioned (size "
                f"{ass_matrix.shape[0]}). Condition number: {cond_number:.2e}. Solution may be inaccurate or fail.")
         print(msg)

    # Використовуємо qps_for_solve_vector
    U_vector = solve(ass_matrix, qps_for_solve_vector)

    # Перевірка, чи вектор U_vector має коректний розмір (NUT)
    if U_vector.shape[0] != nut:
        msg = f"  Error in slay: Solver returned a vector of size {U_vector.shape[0]}, expected NUT ({nut})."
        print(msg)
        ud_vector.fill(np.nan)  # Позначаємо помилку в UD
        return qps_data_for_table # Повертаємо список даних QPS

except Exception as e:
    # Ловимо винятки від розв'язувача (наприклад, LinAlgError для сингулярної матриці)
    msg = f"  Error during solution step: {e}"
    print(msg)
    print(f"  SciPy error details: {e}")
    ud_vector.fill(np.nan)  # Позначаємо помилку в UD
    return qps_data_for_table # Повертаємо список даних QPS
```

Рисунок 3.13 – Фрагмент коду обробки помилок та винятків

Це запобігає подальшим помилкам у постпроцесінгу та сигналізує, що розв’язання некоректне.

Отже, функція slay() забезпечує стійке та контрольоване розв’язання глобальної системи рівнянь. Реалізація дозволяє автоматично враховувати довільні ГУ, ефективно виключати фіксовані DOF, забезпечити обробку вироджених випадків, а також отримати повний вектор переміщень, придатний до подальшого аналізу. Цей підхід є типовим для реалізації МСЕ і відповідає вимогам інженерного розв’язувача для плоских задач теорії пружності.

3.3 Постпроцесинг і підготовка до візуалізації

Після обчислення вузлових переміщень u необхідно визначити розподіл деформацій та напружень у межах кожного елемента. Цей процес виконується

функцією napr() (модуль fem_postprocess.py) та ґрунтується на класичній схемі обчислення за МКЕ:

$$\sigma_e = D \cdot B \cdot u_e \tag{3.2}$$

де u_e – вектор вузлових переміщень поточного елемента; B – матриця деформацій у локальній точці елемента; D – матриця пружності.

3.3.1 Обчислення напружень та деформацій

Функція napr() циклічно обробляє кожен елемент сітки, визначає його розміри a, b, формує локальний вектор переміщень udd (розмір 8×1), обчислює B-матрицю у центрі елемента та розраховує напруження (рис. 3.14).

```
pfn_matrix = prfunf(ux_local_for_prfunf, data)  # B-матриця
dp_matrix = data.upr @ pfn_matrix               # D * B
sig_vector = dp_matrix @ udd_vector             # σ = D * B * u
```

Рисунок 3.14 – Фрагмент коду визначення розподілу деформацій та напружень у межах кожного елемента

Координати вузлів та DOF елемента визначаються за глобальними індексами, відповідно до розташування вузлів (рис. 3.15).

Як результат, для кожного елемента обчислюється вектор напружень за формулою (2.9), ці дані зберігаються у структурованому списку або записуються у файл.

```
# Порядок має відповідати B-матриці prfunf: BL, BR, TR, TL
global_node_indices_for_udd = [gn_bl, gn_br, gn_tr, gn_tl]

udd_vector = np.zeros( shape: (8, 1), dtype=np.float64)
error_in_extraction = False

# Цикл по локальних вузлах елемента в стандартному порядку (0..3)
for local_node_idx in range(4):
    current_global_node_py = global_node_indices_for_udd[local_node_idx]

    # Обчислюємо 0-індексовані глобальні DOF індекси для цього вузла (X та Y)
    global_dof_x_py = 2 * current_global_node_py
    global_dof_y_py = global_dof_x_py + 1

    # Перевірка меж для глобальних DOF в ud_vector (розмір ud_vector = 2*data.nux_max)
    # Перевіряємо проти актуальної заповненої частини data.ud (2*data.nux)
    if global_dof_y_py >= 2 * data.nux: # Перевірка проти 2 * data.nux
        msg = (f"  Error in napr (UD extraction): Global DOF index {global_dof_y_py} out of bounds "
               f"for UD (size {ud_vector.shape[0]}) during extraction at element ({ii},{jj}), "
               f"local node {local_node_idx}. Max NUX={data.nux_max}, Current Grid DOF={2*data.nux}. "
               f"Skipping extraction.")
        output_messages.append(msg)
        # Заповнюємо UDD значеннями NaN, щоб сигналізувати про помилку для цього елемента
        udd_vector.fill(np.nan)
        error_in_extraction = True
        break # Виходимо з циклу по локальних вузлах

    # Збираємо переміщення у UDD вектор (X, потім Y для кожного локального вузла)
    # Перевіряємо, що індекси в ud_vector дійсні (global_dof_x_py та global_dof_y_py)
    # Ця перевірка по суті зайва при коректній перевірці вище, але не зашкодить.
    if global_dof_x_py < 2 * data.nux and global_dof_y_py < 2 * data.nux:
        udd_vector[local_node_idx * 2, 0] = ud_vector[global_dof_x_py] # u_x
        udd_vector[local_node_idx * 2 + 1, 0] = ud_vector[global_dof_y_py] # u_y
    else:
        # Ця гілка не має виконуватись при коректній логіці.
        msg = (f"  Internal Warning in napr: DOF indices {global_dof_x_py}, {global_dof_y_py} seem out "
               f"of bounds for UD within current grid size {2*data.nux} at element ({ii},{jj}), "
               f"local node {local_node_idx}. Skipping extraction.")
        output_messages.append(msg)
        udd_vector.fill(np.nan) # Забезпечуємо, що UDD містить NaN
        error_in_extraction = True
        break # Виходимо з циклу локальних вузлів
```

Рисунок 3.15 – Фрагмент коду визначення координат вузлів та DOF елемента

3.3.2 Побудова полів напружень, переміщень та еквівалентних напружень

Модуль fem_vizualization.py реалізує функції для візуалізації початкової та деформованої сітки, векторного поля переміщень, розподілу напружень ($\sigma_x, \sigma_y, \tau_{xy}$) та поля еквівалентних напружень (наприклад, за критерієм Мізеса):

$$\sigma_{eq} = \sqrt{\sigma_x^2 - \sigma_x\sigma_y + \sigma_y^2 + 3\tau_{xy}^2} \tag{3.3}$$

Побудова виконується з використанням бібліотек matplotlib, numpy та PyQtGraph. Візуалізація реалізована так, щоб інтерактивно оновлювати графічне вікно при зміні вхідних параметрів у GUI (main_window.py).

3.3.3 Відображення результатів у графічному інтерфейсі користувача

Усі функції візуалізації результатів реалізовано у вигляді інтерактивного графічного інтерфейсу користувача на базі бібліотеки PyQt5, а графіка побудована з використанням бібліотеки Matplotlib. Інтерфейс дозволяє:

- вводити параметри задачі;
- запускати обчислення;
- переглядати графічні результати за допомогою окремого візуалізатора MainWindow, який побудований вручну і використовує OpenGL для 3D-графіки(переміщення, напруження, сітка).

Інтерфейс визначено у файлі my_window_ui.py, який є автогенерованим з QtDesigner через pyuic5. Він містить клас Ui_MainWindow, де декларативно описано усі елементи GUI (рис. 3.16).

```
class Ui_MainWindow(object):  2 usages
    def setupUi(self, MainWindow):  1 usage
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1311, 1623)
        MainWindow.setTabShape(QtWidgets.QTabWidget.Rounded)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayout_main = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout_main.setObjectName("verticalLayout_main")
        self.scrollArea = QtWidgets.QScrollArea(self.centralwidget)
        self.scrollArea.setWidgetResizable(True)
        self.scrollArea.setObjectName("scrollArea")
        self.scrollAreaWidgetContents = QtWidgets.QWidget()
        self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents")
        self.verticalLayout_scroll_content = QtWidgets.QVBoxLayout(self.scrollAreaWidgetContents)
        self.verticalLayout_scroll_content.setObjectName("verticalLayout_scroll_content")
        self.horizontalLayout_input_row1 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_input_row1.setObjectName("horizontalLayout_input_row1")
        spacerItem = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_input_row1.addItem(spacerItem)
        self.groupBox_GeometryMesh = QtWidgets.QGroupBox(self.scrollAreaWidgetContents)
```

Рисунок 3.16 – Фрагмент коду створення GUI додатку

Графічний інтерфейс користувача під час роботи програми має вигляд (рис. 3.17).

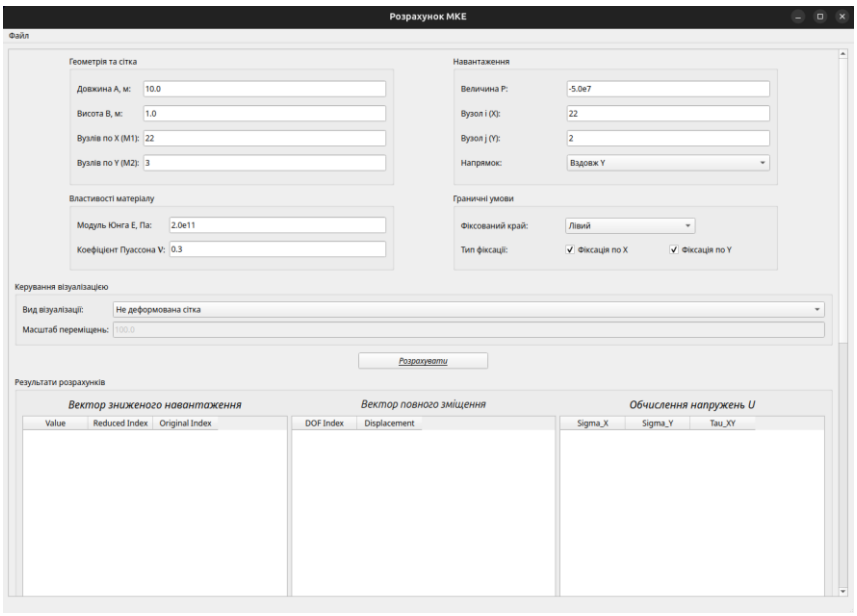


Рисунок 3.17 – Вигляд GUI при старті програми

Файл main_window.py містить клас MainWindow, який відповідає за основну логіку програми та реалізацію логіки головного вікна та підключає логіку до елементів інтерфейсу. Основна робота виконується у методах класу (рис. 3.18).

```python
# --- Клас Головного Вікна ---
class MainWindow(QMainWindow, Ui_MainWindow):  1 usage
    # Конструктор класу
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # --- PyVista Integration ---
        # Знаходимо віджет-плейсхолдер для візуалізації за об'єктним ім'ям з .ui файлу
        self.frame_visualization_placeholder = self.findChild(QFrame, "frame_visualization")

        # Перевіряємо, чи знайдено віджет-плейсхолдер
        if not self.frame_visualization_placeholder:...
        else:...


        # --- Зберігання даних розрахунку для візуалізації ---
        self.mesh = None # Об'єкт сітки PyVista (не деформована)
        self.displacements_data = None # Масив переміщень (3D) для кожного вузла
        self.stress_data = None # Словник з масивами напружень по елементах


        # --- Валідатор для поля масштабу переміщень (тільки числа) ---
        # self.lineEdit_DisplacementScale.setValidator(QtGui.QDoubleValidator()) # Закоментовано/видалено

        # --- Підключення Сигналів (обробників подій) ---
        self.pushButton_Calculate.clicked.connect(self.calculate_fem) # Кнопка "Розрахувати"
        self.action_Exit.triggered.connect(self.close_application) # Дія меню "Вийти"
        self.action_SaveResults.triggered.connect(self.save_results) # Дія меню "Зберегти результати..."
        self.pushButton_Clear.clicked.connect(self.clear_tables) # Кнопка "Очистити таблиці"
```

Рисунок 3.18 – Фрагмент коду ініціалізації GUI

Метод update_plot перемальовує сцену залежно від вибраного типу даних.

```
def update_plot(self):
    visualization_type = self.comboBox_VisualizationType.currentText()
    self.interactor.clear()

    if visualization_type == "Displacement":
        plot_displacement(self.interactor, self.data)
    elif visualization_type == "Sigma_x":
        plot_stress(self.interactor, self.data, component="sx")
    elif visualization_type == "Mises":
        plot_mises(self.interactor, self.data)
```

Рисунок 3.19 – Фрагмент коду реалізації відображення сцени в залежності від вибору користувача

Це дозволяє динамічно оновлювати зображення відповідно до результатів розрахунку та вибору користувача, а саме здійснюється підтримка кількох типів візуалізації.

У випадаючому списку plot_combo реалізовано підтримку кількох типів полів: “Не деформована сітка”, “Деформована сітка”, “Контури напружень (Мізес)”, “Контури напружень (Sigma_X)”, “Контури напружень (Sigma_Y)”, “Контури напружень (Tau_XY)”, “Контури переміщень (величина)”.

Залежно від обраного пункту, викликається відповідна функція побудови.

3.4 Аналіз роботи програми та отриманих результатів

3.4.1 Графічний інтерфейс користувача: структура, зручність, керованість

Розроблене програмне забезпечення містить повноцінний графічний інтерфейс, реалізований за допомогою бібліотеки PyQt5 на основі інтерфейсного шаблону, створеного в середовищі Qt Designer. Його компонування побудоване у відповідності до логіки постановки та розв’язання задачі МСЕ. Інтерфейс інтегрує всі ключові етапи — від введення параметрів до перегляду результатів – у єдиному вікні, що значно підвищує зручність роботи користувача.

Верхня частина вікна програми присвячена введенню вхідних даних (рис. 3.20). Ліворуч розташовано блок для задання геометричних параметрів області та параметрів сітки: довжини та висоти прямокутної області, а також кількості вузлів по горизонталі та вертикалі. Праворуч від нього міститься група полів для визначення навантаження – його величини, вузла прикладання та напрямку дії. Безпосередньо під цими полями користувач задає характеристики матеріалу, зокрема модуль пружності Юнга та коефіцієнт Пуассона. Ще нижче розташовано блок для задання ГУ: обирається один із фіксованих країв (лівий, правий, верхній або нижній), а також визначається, чи має бути фіксація по напрямках X і/або Y.

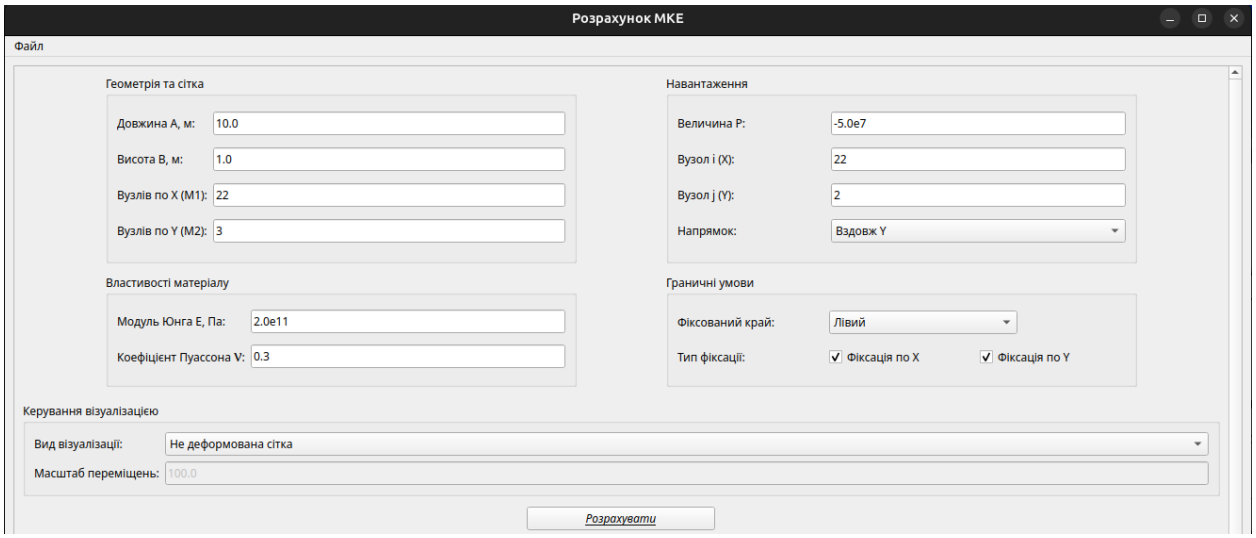


Рисунок 3.20 – Інтерфейс взаємодії з користувачем

Під полями введення розміщено секцію керування візуалізацією. Вона дозволяє обрати один із варіантів відображення результатів – недеформована сітка, деформована форма, поле переміщень або контури напружень за складовими. За потреби можна вказати масштаб переміщень для графічного відображення деформацій конструкції.

Після виконання розрахунку в нижній частині вікна з’являються результати у вигляді трьох таблиць (рис. 3.21). Перша таблиця відображає вектор зниженого навантаження (QPS), що містить реакції у ступенях свободи, виключених з глобальної системи рівнянь внаслідок накладення ГУ. Друга

таблиця – вектор повного переміщення (UD), в якому наведено обчислені переміщення по кожному DOF. Третя таблиця містить компоненти напружень, обчислені для кожного елемента у центрі – σ_x, σ_y та τ_{xy}. Таблиці автоматично очищаються або оновлюються при зміні вхідних даних або натисканні відповідних кнопок керування.

Інтерфейс підтримує повноцінний обмін між обчислювальним ядром і візуальною частиною програми. Зміна будь-якого параметра (наприклад, геометрії або сітки) автоматично враховується при повторному розрахунку. Кнопка “Розрахувати” ініціює весь процес моделювання, від побудови сітки до обчислення переміщень і напружень, тоді як кнопка “Очистити таблиці” дозволяє швидко обнулити результати та повернутися до початкового стану.

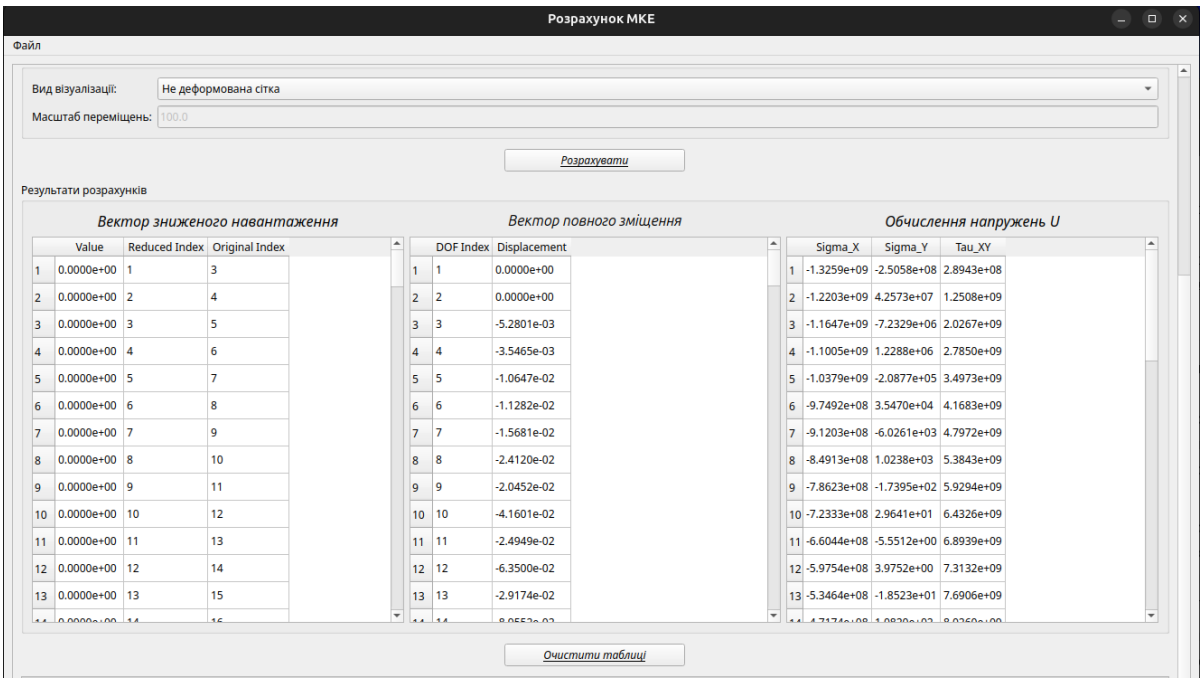


Рисунок 3.21 – Візуалізація результатів розрахунку

У цілому інтерфейс є інтуїтивно зрозумілим і забезпечує користувачеві повний контроль над постановкою задачі, її розв’язанням і візуалізацією результатів. Така інтеграція всіх етапів у єдиному вікні значно полегшує експлуатацію програмного забезпечення та дозволяє зосередитися на аналізі поведінки об’єкта моделювання, а не на технічних деталях взаємодії з програмою.

3.4.2 Аналіз отриманих графічних результатів

Після виконання розрахунку візуалізаційне поле програми дозволяє переглядати як геометричну конфігурацію (початкову або деформовану), так і результати обчислень у вигляді контурних карт напружень та переміщень. Для аналізу було розглянуто приклад розтягування вузької прямокутної пластини розміром $A = 10$ м, $B = 1$ м, з прикладеним навантаженням у від'ємному напрямку осі Y до вузла правого краю. Визначено реакції, переміщення та напруження для кожного елемента сітки 22×3.

Графічне поле дозволяє перемикатися між варіантами відображення результатів. У режимі "Не деформована сітка" видно, що початкова геометрія елементів правильна, прямокутна, без перекручування – що свідчить про коректну генерацію сітки та розрахункових координат (рис. 3.22).

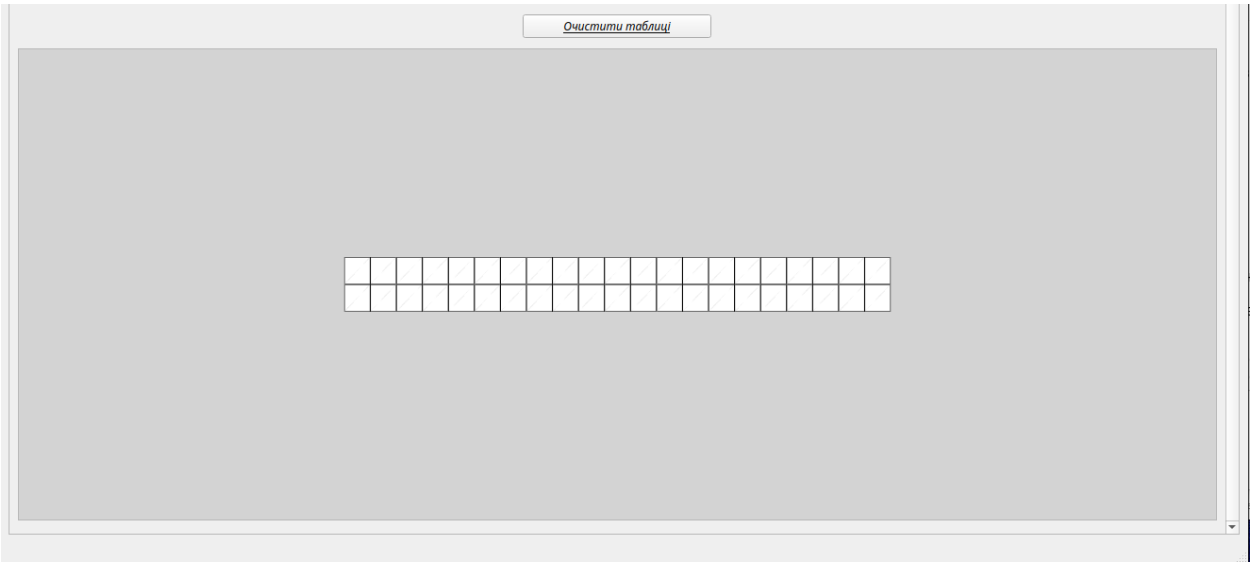


Рисунок 3.22 – Графічна візуалізація "Не деформованої сітки"

У режимі "Деформована сітка" форма області деформується відповідно до напрямку прикладеного навантаження. Спостерігається плавний вигин пластини в нижньому правому куті, характерний для випадку одностороннього затиснення та вертикального тиску. Значення переміщень масштабуються для графічного подання (у прикладі масштаб = 100), а реальні

значення переміщень можна побачити в таблиці “Вектор повного зміщення”. Максимальні переміщення зосереджені на правому краї, де навантаження було прикладене, що фізично обґрунтовано (рис. 3.23).

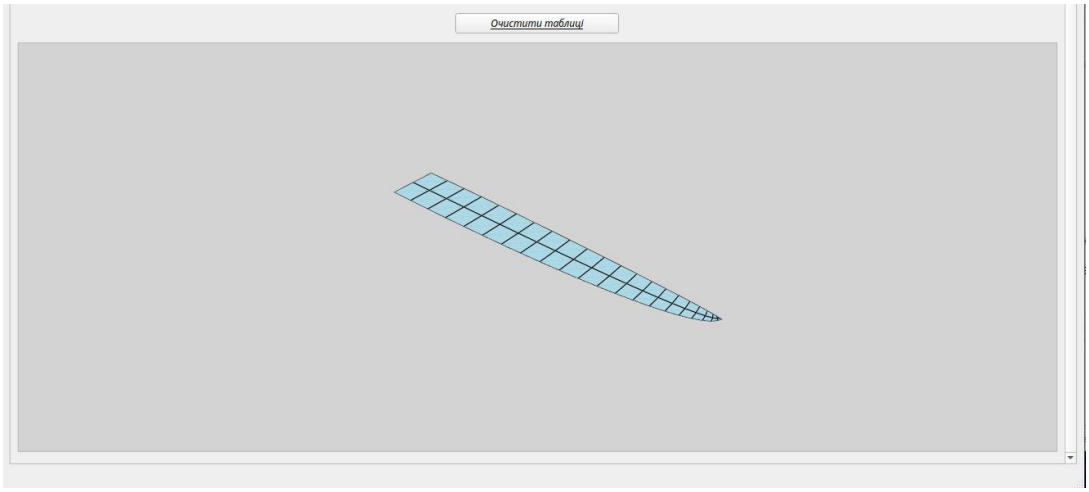


Рисунок 3.23 – Графічна візуалізація “Деформованої сітки”

Розподіл напружень оцінювався за трьома компонентами: $\sigma_x, \sigma_y, \tau_{xy}$, а також за еквівалентним напруженням Мізеса. Контурна карта напружень σ_x демонструє максимальні значення у верхній частині правого краю та поступове зменшення до лівого фіксованого краю. Мінімальні значення спостерігаються у лівому верхньому куті – це типова картина для згинального навантаження при жорсткому закріпленні (рис. 3.24).

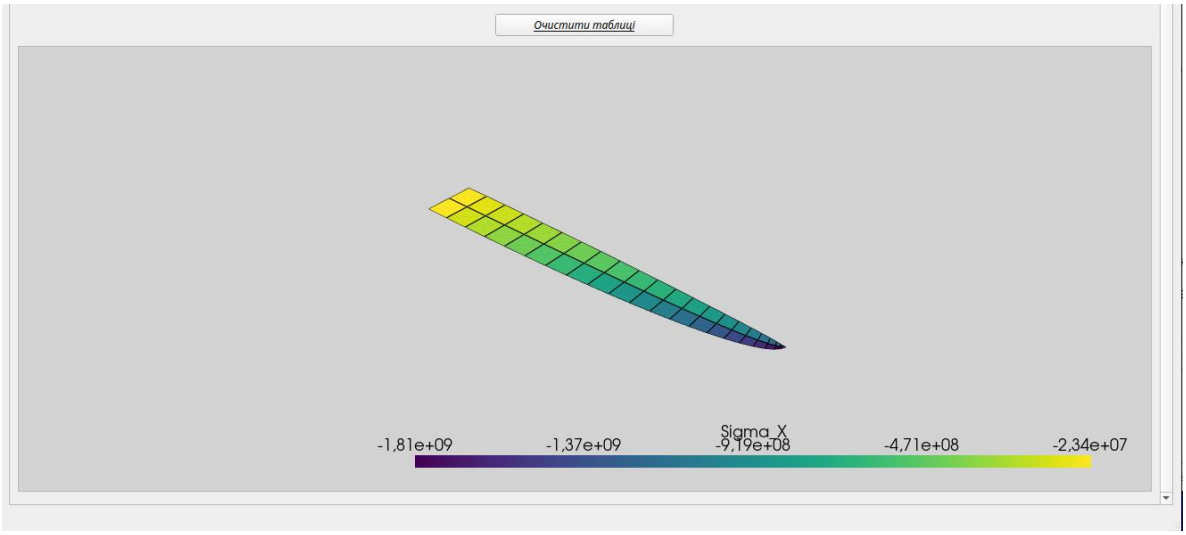


Рисунок 3.24 – Графічна візуалізація “Контури напружень σ_x”

Аналогічно, розподіл σ_y показує зростання напружень до правого краю, де прикладено вертикальну силу. Значення є майже симетричними по вертикалі, що свідчить про правильне врахування ГУ та рівномірність прикладеного навантаження (рис. 3.25).

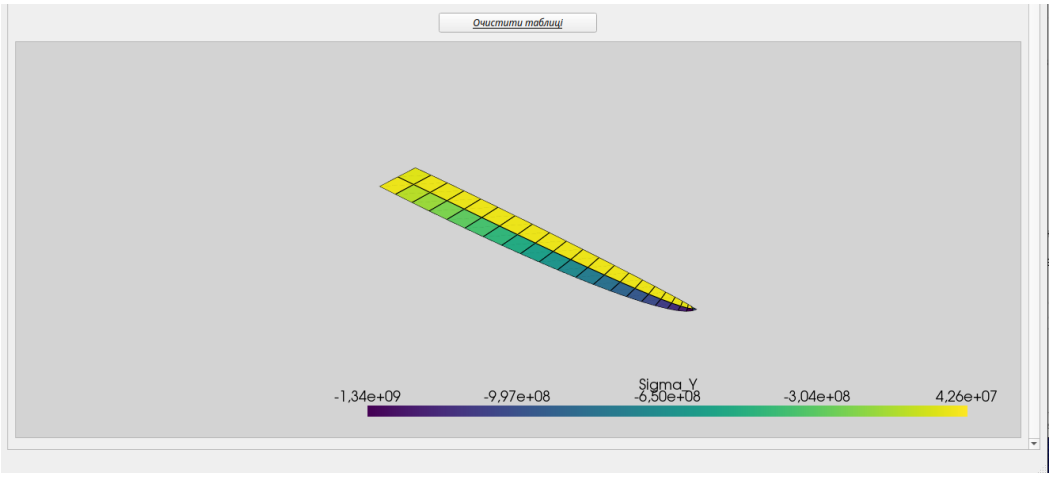


Рисунок 3.25 – Графічна візуалізація “Контури напружень σ_y”

Компонента зсувного напруження τ_{xy} має характерне концентроване поле в центральній частині пластини, поблизу зони деформації. Це також узгоджується з очікуваною поведінкою матеріалу при згині та частковому розтягу (рис. 3.26).

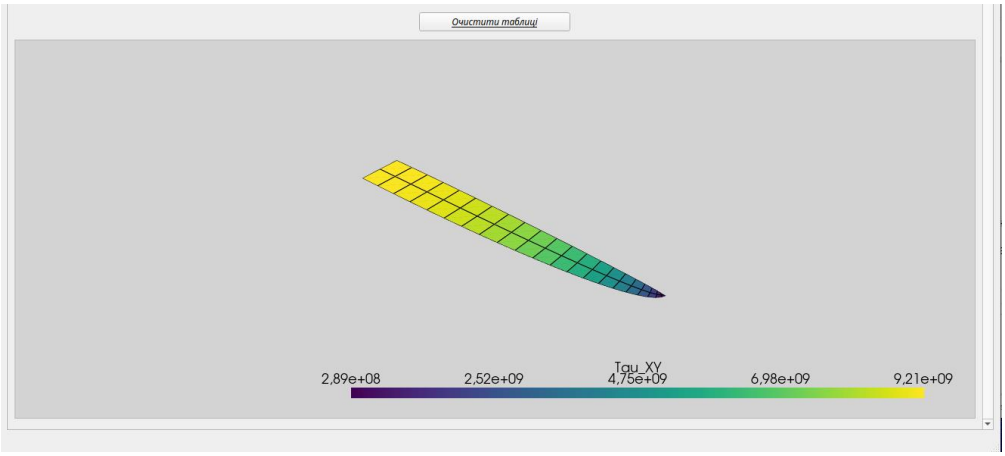


Рисунок 3.26 – Графічна візуалізація “Контури напружень τ_{xy}”

Карта еквівалентного напруження за критерієм Мізеса надає узагальнену оцінку напруженого стану. Максимальні значення

спостерігаються поблизу точок прикладання навантаження, що дозволяє виявити потенційно критичні зони конструкції. Мінімальні значення розміщені на фіксованому краю, де відсутні переміщення (рис. 3.27).

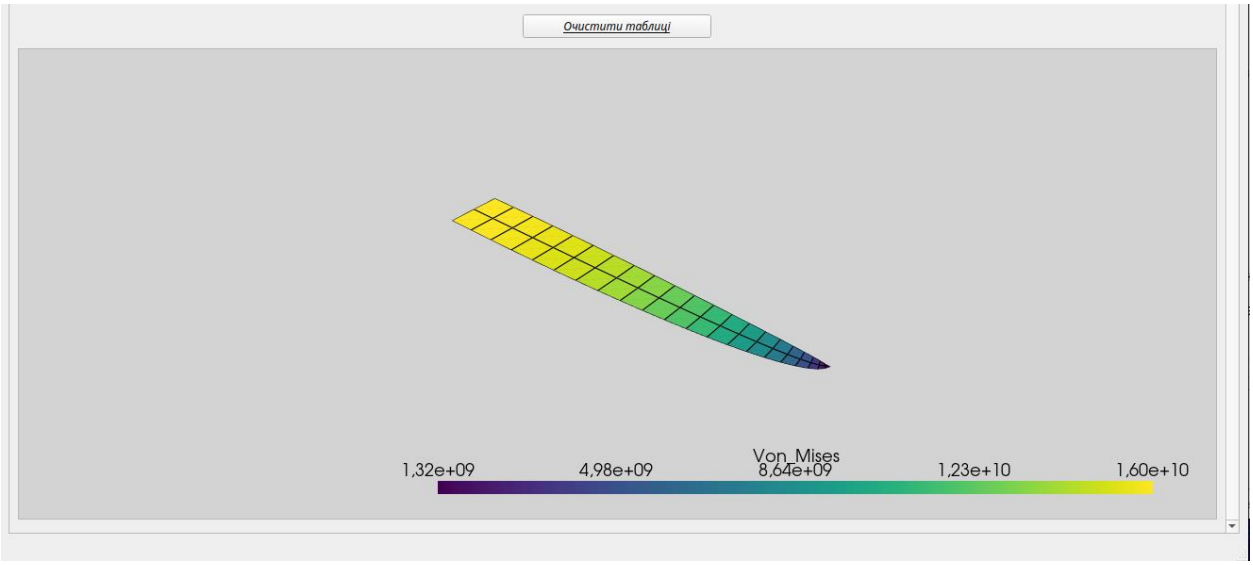


Рисунок 3.27 – Графічна візуалізація “Контури напружень Мізес”

Контурна карта повної величини переміщень (Displacement Magnitude) дозволяє оцінити максимальні відхилення вузлів від початкового положення. Очевидно, що найбільші зміщення відбуваються у верхньому правому куті – там, де відсутні граничні обмеження і прикладене навантаження. Це відповідає очікуваній поведінці деформівного тіла (рис. 3.28).

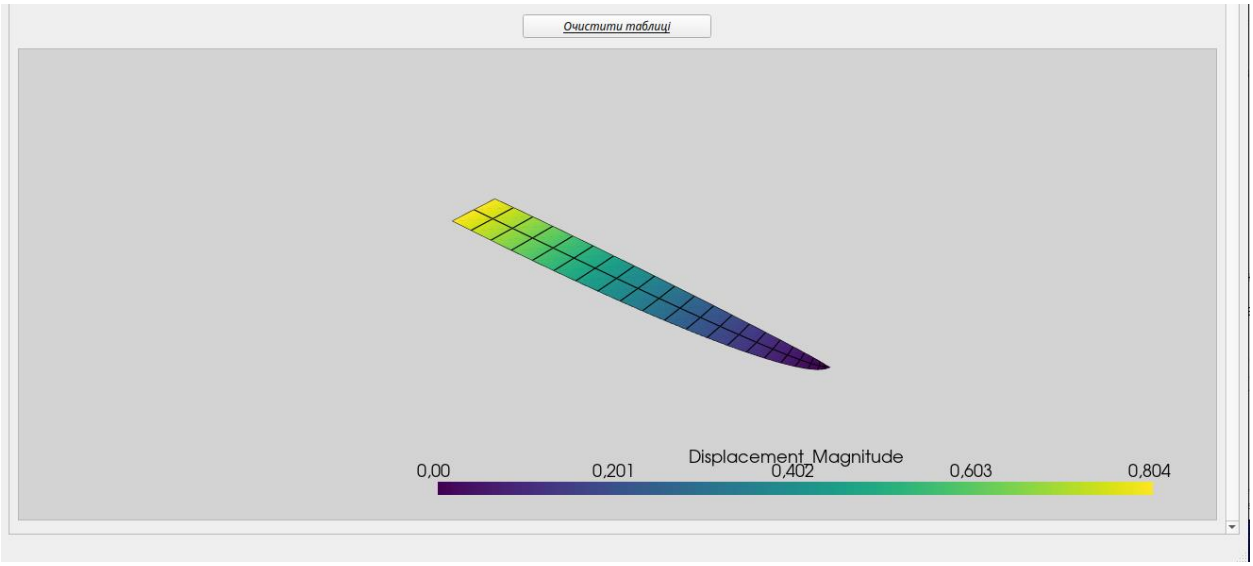


Рисунок 3.28 – Графічна візуалізація “Контури переміщень”

Усі графічні зображення супроводжуються шкалами значень, що дозволяє оцінити порядок величин результатів, а також зіставити їх із табличними даними. Інтерфейс дає змогу динамічно перемикатися між різними варіантами візуалізації, що забезпечує зручність аналізу.

3.4.3 Узагальнення поведінки моделі за числовими результатами

Файл результатів, який можна зберегти використовуючи меню “Файл”, містить три основні частини: вектор зниженого навантаження (QPS), вектор повного переміщення (UD) та масив обчислених напружень по кожному елементу. Це дозволяє провести кількісну оцінку напружено-деформованого стану конструкції та перевірити фізичну обґрунтованість моделі. Зробимо аналіз отриманих даних.

У розділі Full Displacement Vector (UD) наведено значення переміщень по всіх ступенях свободи. Максимальні відхилення спостерігаються у DOF 88, який відповідає точці прикладання навантаження (правий край, вузол $i = 22, j = 2$). Переміщення за напрямком Y у цій точці становить приблизно $-8.02 \cdot 10^{-1}$ м, що фізично відповідає очікуваному згину довгої тонкої пластини з лівим закріпленням і правостороннім навантаженням.

Симетричність поля переміщень підтверджується поступовим спаданням значень від правого до лівого краю, де переміщення зафіксовано на рівні нуля, що узгоджується з накладеними ГУ ($np = 0$).

Масив Element Stresses містить три компоненти напруженого стану для кожного з елементів: $\sigma_x, \sigma_y, \tau_{xy}$. Найбільші значення осьових напружень $\sigma_x \approx -1.81 \cdot 10^9$ Па зосереджені у правій верхній частині конструкції – це зона, де прикладено силу, і де формується максимальний згинальний момент.

Еквівалентне напруження за критерієм Мізеса (не виводиться явно у файлі, але може бути обчислене з даних) досягає порядку $\sim 1.6 \cdot 10^{10}$ Па, що також підтверджується на контурній карті. Спостерігається плавний розподіл напружень уздовж довжини пластини з посиленою концентрацією в точках

вигину – зокрема у верхньому правому секторі сітки.

Варто відзначити, що компоненти зсувного напруження τ_{xy} також демонструють зростання від фіксованого краю до зони дії навантаження, досягаючи значень понад $9 \cdot 10^9$ Па. Така поведінка характерна для згину та підтверджує правильність реалізації апроксимаційних функцій і обчислення B-матриці.

У векторі зниженого навантаження (Reduced Load Vector) єдине ненульове значення припадає на DOF з індексом 88, що відповідає прикладеній силі $P = -5 \cdot 10^7$ Н у напрямку Y. Усі інші значення дорівнюють нулю, що відповідає умовам задачі з єдиним зосередженим навантаженням. Відсутність паразитних реакцій або неочікуваних концентрацій підтверджує коректність складання глобального вектора q та застосування ГУ.

Отже, числові результати повністю узгоджуються з аналітичними очікуваннями для довгої прямокутної балки під дією згинаючого навантаження при лівосторонньому закріпленні. Вектор переміщень, розподіл напружень та контурні діаграми дозволяють відстежити повну картину напружено-деформованого стану. Плавність переходів, відсутність осередків неузгодженості та концентрації в зоні закріплення підтверджують правильність реалізації уточненої апроксимації у скінченних елементах, чисельного інтегрування та умов узгодження вузлів.

3.4.4 Обчислювальна ефективність та стабільність реалізації

Програмне забезпечення було протестовано на декількох сітках різної густини – від грубої 4×4 до щільної 22×3, яка використовувалась у прикладі попереднього аналізу. Навіть при збільшенні кількості вузлів до кількох сотень система показує стабільну роботу без помітного зростання часу обчислень. Завдяки використанню бібліотеки NumPy для векторизованих операцій та функції scipy.linalg.solve() для розв’язання системи лінійних рівнянь, досягається висока продуктивність при збереженні точності.

Формування глобальної матриці жорсткості реалізоване через послідовне додавання локальних елементних матриць у відповідні позиції глобальної матриці. Оскільки всі основні операції над матрицями виконуються у форматі масивів NumPy, їх обчислювальна складність лишається лінійною від кількості елементів при фіксованій кількості DOF на елемент. Це дозволяє масштабувати розрахунки для великих сіток без суттєвих втрат продуктивності.

Окремої уваги заслуговує стабільність розв’язувача при обробці ГУ. У реалізації функції slay() виконується редукція глобальної системи рівнянь через вилучення закріплених DOF із матриці жорсткості та вектору навантажень. Такий підхід дозволяє уникнути чисельної нестійкості, яка часто виникає при застосуванні методу великого числа, зберігаючи при цьому симетрію матриці та її позитивну визначеність. У разі виродженості матриці реалізовано обробку винятків з повідомленням про помилку, що забезпечує контроль за коректністю розрахунку.

Після розв’язання задачі результати виводяться у графічному вигляді з використанням PyVista та QtInteractor. Побудова контурних карт напружень та переміщень виконується за лічені частки секунди, навіть для сіток із кількістю елементів понад 100. Завдяки цьому користувач має змогу одразу після розрахунку перейти до аналізу результатів, не чекаючи на вивід зображень. При цьому масштаб переміщень та тип поля задаються інтерактивно, що підвищує гнучкість інтерфейсу та зручність експлуатації.

Отже, реалізований програмний комплекс демонструє хорошу масштабованість, стабільність при розв’язанні задач із фіксованими або змішаними ГУ, а також достатню швидкодію для інтерактивного використання в рамках інженерного аналізу. Це дозволяє розглядати його як ефективний інструмент для розв’язання задач плоскої деформації з використанням уточнених схем скінченноелементного аналізу.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи досліджено та реалізовано підхід до розв’язання плоских задач теорії пружності на основі уточнених схем апроксимації у МСЕ. Проведено комплексний аналіз класичних апроксимаційних функцій, а також запропоновано модифікований підхід на основі моментної схеми для прямокутного шестивузлового елемента. Побудовано відповідну математичну модель, що враховує основні вимоги до сумісності та точності функцій форми.

На основі розробленої моделі створено програмне забезпечення з інтерактивним графічним інтерфейсом користувача, яке дозволяє повністю автоматизувати процес постановки задачі, її чисельного розв’язання та візуалізації результатів. Реалізовано функції генерації сітки, побудови елементних і глобальних матриць жорсткості, застосування ГУ, розв’язання СЛАР, обчислення переміщень і напружень, а також їх графічного представлення.

Проведене чисельне моделювання на прикладі прямокутної пластини із жорстким закріпленням і зосередженим навантаженням підтвердило правильність реалізації та точність побудованої схеми. Отримані результати демонструють фізично обґрунтовану поведінку об’єкта, зокрема характерний розподіл напружень і переміщень.

Розроблений програмний комплекс є універсальним, масштабованим, а його відкрита архітектура дозволяє подальше розширення функціональності: реалізацію нових типів елементів, підтримку тривимірних задач, нелінійних моделей матеріалів та розширеного постпроцесингу. Результати роботи можуть бути використані в навчальному процесі, наукових дослідженнях та прикладних інженерних розрахунках.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Єгорова В. С. Підвищення точності апроксимацій функцій в методі скінчених елементів. Запоріжжя: ЗНУ, 2024. 8 с.

2. Єгорова В. С., Зубко К. В. Удосконалення систем автоматизації проєктувальних робіт за допомогою уточнення математичних моделей. Запоріжжя: ЗНУ, 2023. 23 с.

3. Zienkiewicz O. C., Taylor R. L. The Finite Element Method. Vol. 1: The Basis. Butterworth-Heinemann, 2000. 708 p.

4. Courant R. Variational methods for the solution of problems of equilibrium and vibrations. Bulletin of the American Mathematical Society, 1943. 23 p.

5. Argyris J. H. Energy theorems and structural analysis; a generalised formulation with applications to aircraft structures. Butterworths, 1954. 61 p.

6. Clough R. W. *The finite element method in plane stress analysis*: proceedings of the 2nd Conference on Electronic Computation, ASCE, Pittsburgh, 1960. P. 345-378.

7. Zienkiewicz O. C., Taylor R. L. The Finite Element Method for Solid and Structural Mechanics. Butterworth-Heinemann, 2005. 476 p.

8. Ern A., Guermond J.-L. Finite elements: Approximation and interpolation. Springer, 2021. 288 p.

9. Gupta R. K. Numerical Methods: Fundamentals and Applications. Cambridge: Cambridge University Press, 2019. 83 p.

10. Bathe K. J. Finite Element Procedures. New Jersey: Prentice hall, 2014. 1052 p.

11. Ільченко О. В. Посібник з курсу “Математичний аналіз” для студентів ННІ «Інститут геології». Київ: КНУ ім. Тараса Шевченка, 2021. 65 с.

12. Szabó B., Babuška I. Finite element analysis: Method, verification and validation. Hoboken: Wiley, 2021. 369 p.

13. Wang L., Hu M., Zhong Z., Yang F. *Stabilized Lagrange Interpolation*

Collocation Method: A meshfree method incorporating the advantages of finite element method: Computer Methods in Applied Mechanics and Engineering, 2023. P. 300-324.

14. Киричевський В. В., Дохняк Б. М., Гребенюк С. М. Трикутні скінченні елементи у механіці руйнування еластомерів. *Опір матеріалів і теорія споруд*. 2000. Вип. 67. С. 72–83.

15. ANSYS. URL: https://goo.su/v7hTMP (дата звернення: 01.03.2025).

16. Dassault Systèmes. Abaqus. URL: https://goo.su/lIbVL (дата звернення: 10.03.2025).

17. MSC Software. Nastran. URL: https://goo.su/NZFo (дата звернення: 05.03.2025).

18. COMSOL Multiphysics. URL: https://goo.su/KhfCPl (дата звернення: 07.03.2025).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Єгорова Вероніка Сергіївна

студентка 4 курсу, денної форми навчання, математичного факультету, спеціальності Прикладна математика, адреса електронної пошти evs9990@gmail.com, – підтверджую, що написана мною кваліфікаційна робота бакалавра на тему «Уточнені схеми апроксимації у методі скінченних елементів» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлена;

– заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

– згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Студент

_____	_____	_____
(дата)	(підпис)	(ініціали, прізвище)

Науковий керівник

_____	_____	_____
(дата)	(підпис)	(ініціали, прізвище)