УДК 004.4:004.94

С.І. ГОМЕНЮК, С.М. ГРЕБЕНЮК, А.Г. КРИВОХАТА, Н.В. МАТВІЇШИНА
Запорізький національний університет

ЛІНГВІСТИЧНЕ ЗАБЕЗПЕЧЕННЯ САПР

У статті розглянуто підхід до створення лінгвістичного забезпечення систем автоматизованого проєктування, що базуються на використанні методу скінченних елементів під час моделювання складних інженерно-технічних систем. Показано, що одним із найскладніших етапів підготовки вихідних даних для розрахунку є формалізація крайових умов, навантажень та неоднорідних фізичних властивостей, особливо у разі складної або нетипової геометрії розрахункової сфери. Для автоматизації цього процесу авторами створено спеціалізовану бібліотеку класів на мові C++, яка забезпечує можливість опису фізичних і геометричних характеристик об'єкта, крайових умов та навантажень у вигляді формул, заданих на предметно-орієнтованій мові, максимально наближеній за своїм синтаксисом до мови програмування Python.

Основну увагу приділено розробленню інтерпретатора функціональних виразів, що описують фізичні параметри та крайові умови. Детально розглянуто процес трансляції арифметико-логічних виразів у абстрактне синтаксичне дерево, яке є основою для подальшого обчислення значень у вузлах скінченно-елементної моделі, які відповідають заздалегідь заданим умовам відбору. Наведено структуру класу Parser, що реалізує основні етапи трансляції: лексичний і синтаксичний аналіз, побудову та збереження дерева розбору, а також обчислення виразів. Описано алгоритм сканування вхідного коду, розпізнавання лексем – числових констант, операторів, ідентифікаторів тощо, а також формування відповідних токенів. Представлено UML-діаграму класу Node, який інкапсулює абстрактне синтаксичне дерево, і наведено приклад реалізації рекурсивного методу, що безпосередньо виконує обчислення.

Важливою особливістю створеного інтерпретатора є підтримка можливості динамічного завантаження в абстрактне синтаксичне дерево координат вузлів скінченно-елементної моделі без необхідності перекомпіляції коду, що підвищує гнучкість і розширюваність системи. Програмна реалізація парсеру орієнтована на використання у складі спеціалізованого програмного забезпечення для комп'ютерного моделювання міцності й довговічності складних інженерно-технічних систем, що проєктуються.

Ключові слова: *метод скінченних елементів, лінгвістичне забезпечення, транслятор, GIT, паралельні алгоритми, візуалізація.*

S.I. HOMENIUK, S.M. GREBENYUK, A.H. KRYVOKHATA, N.V. MATVIYISHYNA
Zaporizhzhia National University

CAD LINGUISTIC SUPPORT

The article presents an approach to developing linguistic support for computer-aided design systems based on the finite element method in modeling complex engineering and technical systems. It is shown that one of the most challenging stages in preparing input data for computation is the formalization of boundary conditions, loads, and heterogeneous physical properties, especially in cases of complex or non-standard geometry of the computational domain. To automate this process, the authors have developed a specialized C++ class library that enables the description of the physical and geometric characteristics of an object, boundary conditions, and loads in the form of formulas written in a domain-specific language whose syntax closely resembles that of the Python programming language.

The main focus is on the development of an interpreter for functional expressions that describe physical parameters and boundary conditions. The paper provides a detailed discussion of the process of translating arithmetic and logical expressions into an abstract syntax tree, which serves as the basis for further evaluation of values at the nodes of the finite element model corresponding to predefined selection conditions.

The structure of the Parser class is presented, implementing the main stages of translation: lexical and syntactic analysis, construction and storage of the parse tree, as well as expression evaluation. The algorithm for scanning the input code, recognizing lexemes—such as numeric constants, operators, and identifiers—and generating the corresponding tokens is described. A UML-diagram of class Node, which encapsulates the abstract syntax tree, is provided, along with an example implementation of a recursive method that directly performs the evaluation.

An important feature of the developed interpreter is its ability to dynamically load the coordinates of finite element model nodes into the abstract syntax tree without the need for code recompilation, which increases the system's flexibility and extensibility. The software implementation of the parser is designed for use as part of specialized software for computer modeling of the strength and durability of complex engineering and technical systems under design.

Key words: *finite element method, linguistic support, translator, GIT, parallel algorithms, visualization.*

https://doi.org/10.32782/mathematical-modelling/2025-8-2-6 ISSN 2618-0332 (print), 2618-0340 (online)

Постановка проблеми

Розвиток машинобудування у сучасних умовах жорсткої конкуренції між різними виробниками неможливий без часткової заміни реальних фізичних випробувань дослідних зразків комп'ютерним експериментом, що дає змогу істотно зменшити витрати часу і ресурсів на проєктування нової техніки.

Під час використання чисельних методів для дослідження широкого кола інженерно-технічних задач нині застосовується спеціалізоване програмне забезпечення – системи автоматизованого проєктування і розрахунку, що, як правило, складаються з трьох основних компонентів: препроцесора, який використовується для підготовки вихідних даних для розрахунку; процесора, який безпосередньо виконує усі необхідні під час моделювання обчислення; постпроцесора, що автоматизує аналіз отриманих результатів. Досвід показує, що підготовка вихідних даних для розрахунку є найбільш складним і тривалим етапом дослідження, який зазвичай складається з двох складників: опису геометричної моделі сфери розрахунку і завдання крайових умов та навантажень. У разі складної геометрії вихідної області розрахунку формалізація крайових умов та навантажень є доволі складним завданням, яке у загальному вигляді потребує розроблення спеціальних засобів розв'язання.

Аналіз останніх досліджень і публікацій

Одним із найбільш поширених сьогодні чисельних методів дослідження широкого кола задач математичної фізики є метод скінченних елементів (МСЕ) [1]. Для його використання розроблено велику кількість спеціалізованого програмного забезпечення (ПЗ), серед якого можна виділити такі системи, як ANSYS [2], MSC Nastran [3], FreeFEM [4] та ін. [5]. Проте ускладнення завдань, що виникають у промисловості, вимагає постійного вдосконалення наявного або розроблення нового ПЗ для скінченно-елементного аналізу.

Найбільш складним етапом використання МСЕ є підготовка вихідних даних для розрахунку. Передусім це створення геометричної моделі об'єкта, що досліджується, із подальшою її дискретизацією на скінченні елементи заданого типу. А в другу чергу – опис вузлових значень граничних умов та навантажень, які можуть бути достатньо складними за нетипової форми сфери розрахунку.

Мета дослідження

Об'єктом дослідження є процес розроблення спеціалізованого ПЗ для скінченно-елементного аналізу.

Предметом дослідження є створення лінгвістичного забезпечення для систем скінченно-елементного аналізу.

Метою роботи є створення спеціалізованої бібліотеки класів мови C++, яка б давала змогу користувачу описувати фізичні та геометричні характеристики, крайові умови і навантаження для задач математичної фізики будь-якої складності, виконувати їх трансляцію в абстрактне синтаксичне дерево й здійснювати на їх основі необхідні обчислення.

Виклад основного матеріалу дослідження

Модель та програмна реалізація інтерпретатора. Розв'язання задач із використанням МСЕ, як правило, складається з таких етапів:

1) побудова скінченно-елементної (дискретної моделі) об'єкта розрахунку;

2) завдання фізичних та геометричних параметрів; крайових умов (переміщень, навантажень тощо), які загалом можуть бути різними для окремих зон вихідної області;

3) розрахунок задачі;

4) аналіз отриманих результатів (рис. 1).

https://doi.org/10.32782/mathematical-modelling/2025-8-2-6 ISSN 2618-0332 (print), 2618-0340 (online)

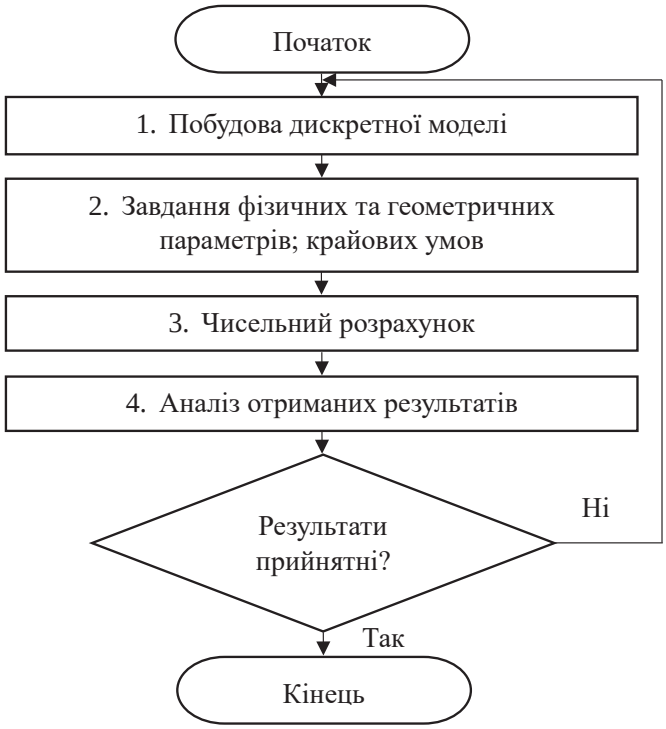

Рис. 1. Основні етапи скінченно-елементного аналізу

У статті досліджено другий етап із наведеної схеми, а саме – автоматизація завдання крайових умов і навантажень. Однією з головних проблем на цьому етапі є те, що зовнішні сили, які діють на об'єкт розрахунку, слід перевести в еквівалентні їм вузлові навантаження, що загалом є доволі нетривіальною операцією. Розглянемо, наприклад, таку модельну задачу. Нехай потрібно визначити параметри напружено-деформованого стану труби, що знаходиться під дією внутрішнього тиску P (рис. 2).

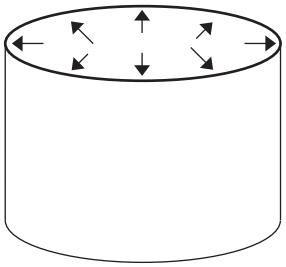

Рис. 2. Труба під дією внутрішнього тиску

Для коректного опису навантаження загалом слід задати: 1) формулу, що визначає значення навантаження; 2) логічну умову (предикат), яка визначає зону дії сили; 3) напрям дії сили.

Так, наприклад, якщо вісь труби співпадає з віссю Z, то значення складників сили P уздовж осей X та Y можна обчислити за допомогою таких співвідношень:

$$\begin{aligned} P_x &= P\cos\left(arctg\left(\frac{y}{x}\right)\right), \\ P_y &= P\sin\left(arctg\left(\frac{y}{x}\right)\right). \end{aligned} \tag{1}$$

https://doi.org/10.32782/mathematical-modelling/2025-8-2-6 ISSN 2618-0332 (print), 2618-0340 (online)

Ці сили будуть діяти у вузлах, координати яких відповідають такому предикату:

$$\left|x^2 + y^2 - R^2\right| < \varepsilon, \tag{2}$$

тут R – внутрішній радіус труби; ε – наперед задана точність пошуку вузлів дискретної моделі, координати яких відповідають указаному предикату.

Таким чином, виникає задача створення ПЗ для трансляції арифметичних виразів, що визначають значення навантаження (і за необхідності інших параметрів розрахунку), а також предикатів, які конкретизують місце їх дії. Своєю чергою, ця задача розбивається на три складники:

1) формалізація опису параметру та предикату на деякій предметно-орієнтованій мові (DSL – Domain-Specific Language) [6];

2) трансляція формул у внутрішнє представлення, наприклад AST (Abstract Syntax Tree) [7];

3) обчислення вузлових значень параметру.

Створення нової проблемно-орієнтованої мови є окремою задачею, що виходить за межі даного дослідження. Тому у цій статті буде розглядатися трансляція в AST функціональних співвідношень, записаних згідно із синтаксисом мови Python [8]. Тоді вирази, що визначають значення навантажень та предикату, заданих співвідношеннями (1), (2), можуть бути записані так:

$$\begin{aligned} &\mathrm{P*cos}\left(\mathrm{atan2}\left(\mathrm{y},\mathrm{x}\right)\right),\\ &\mathrm{P*sin}\left(\mathrm{atan2}\left(\mathrm{y},\mathrm{x}\right)\right), \end{aligned} \tag{3}$$

$$\mathrm{abs}\left(\mathrm{x**2+y**2-1.99**2}\right) <= \mathrm{eps}. \tag{4}$$

Таким чином, задача полягає у тому, щоб автоматизувати трансляцію функціональних виразів, подібних до (3), (4), у AST із подальшим обчисленням їхніх значень у заданих вузлах скінченно-елементної моделі.

Трансляцією називається процес перекладу початкового коду, написаного однією мовою програмування, у еквівалентний код іншою мовою [9]. Відповідно, транслятором називають програму, що здійснює такий переклад. Зазвичай розрізняють два основні типи трансляторів: інтерпретатор та компілятор. Перший крок за кроком виконує трансляцію початкового коду й миттєво його виконує. Компілятор же перетворює весь початковий код на деяку цільову мову, якою, як правило, виступає машинний або байт-код. Компілятор на відміну від інтерпретатора не бере участі у виконанні програми.

Головними перевагами інтерпретатора є його відносно проста реалізація та багатоплатформність, оскільки один і той самий код можна запускати без змін у різних платформах, якщо для них було реалізовано відповідний інтерпретатор. Вагомою перевагою компілятора є швидкість виконання створених ним програм, оскільки вони, як правило, реалізовані на машинній мові та на етапі трансляції можлива оптимізація програмного коду. Проте практична реалізація компілятора набагато складніша. Тому у цій статті буде розглянута реалізація інтерпретатора функціональних співвідношень, які описують крайові умови, фізичні параметри та умови їх застосування під час використання МСЕ.

Процес інтерпретації зазвичай складається з таких етапів, як: 1) зчитування початкового коду; 2) лексичний аналіз, коли код розбивається на лексеми або токени (ключові слова, ідентифікатори, константи тощо); 3) синтаксичний аналіз, коли множина токенів перетворюється на AST; 4) семантичний аналіз – перевірки на коректність програми з погляду мови (другий, третій та четвертий етапи трансляції у простих випадках можна об’єднати); 5) виконання програми, коли інтерпретатор обходить AST і безпосередньо виконує команди (рис. 3).

https://doi.org/10.32782/mathematical-modelling/2025-8-2-6 ISSN 2618-0332 (print), 2618-0340 (online)

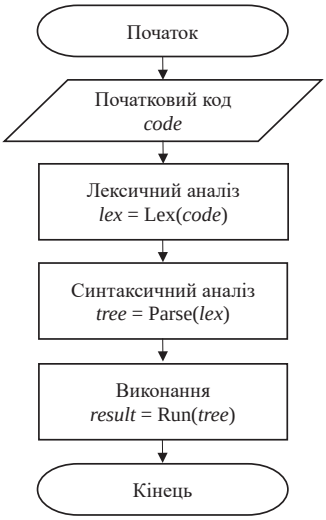

Рис. 3. Стандартні етапи роботи транслятора

На початку своєї роботи інтерпретатор сканує початковий код функціонального співвідношення (змінна *code*), який попадає на вхід до лексичного аналізатора, на виході з якого (у разі коректності початкового коду) буде множина лексем (змінна *lex*). Далі отримана множина лексем оброблюється синтаксичним аналізатором, результатом роботи якого є дерево розбору (змінна *tree*), яке в подальшому може оброблятися з використанням паралельних розрахунків.

Алгоритм трансляції арифметико-логічних виразів у AST можна представити так (рис. 4).

Для його реалізації було розроблено клас Parser, який за допомогою псевдокоду можна описати так (фрагмент).

class Parser
  private fields:
    code : string – початковий код виразу
    tree : Node – дерево розбору виразу
    token_type: integer – тип лексеми
    token : string – поточна лексема
private methods:
method tokenOr(*tree*)
input: *tree* – дерево розбору виразу
hold ← Node()
  tokenAnd(*tree*)
while tokenType = OR **do**
    getToken()
    tokenAnd(*hold*)
    tree ← Node(*tree*, OR, *hold*)
  end while
end method

procedure getToken(*code*)
input: *code* – початковий текст виразу
  output: *token* – чергова лексема
  token_type – тип лексеми
position ← 0
token ← []

https://doi.org/10.32782/mathematical-modelling/2025-8-2-6 ISSN 2618-0332 (print), 2618-0340 (online)

```
while position < length(code) do
char ← code[position]
// Пропуск пробілів та табуляцій
if char is whitespace or tabulation then
position ← position + 1
continue
end if
// Обробка числа
if char is digit or dot then
begin ← position
while char is digit or dot do
position ← position + 1
if position >= length(code) break
char ← code[position]
end while
token ← code[begin: position]
token_type ← NUMBER
continue
end if
// Обробка ідентифікатора
if char is alpha then
begin ← position
while char is alpha or digit or underscore do
if position >= length(code) break
char ← code[position]
end while
token ← code[begin: position]
token_type ← IDENT
continue
end if
// Обробка операторів
if char is plus or minus or mul or div or ... then
token ← char
else
syntaxError()
break
end if
position ← position + 1
end while
return token, token_type
end procedure
public method:
method setExpression(expression)
input: expression – рядок початкового коду
  code ← expression
  tokenOr(tree)
  end method
  method run()
   return tree.value()
  end method
  …
end class
```

https://doi.org/10.32782/mathematical-modelling/2025-8-2-6 ISSN 2618-0332 (print), 2618-0340 (online)

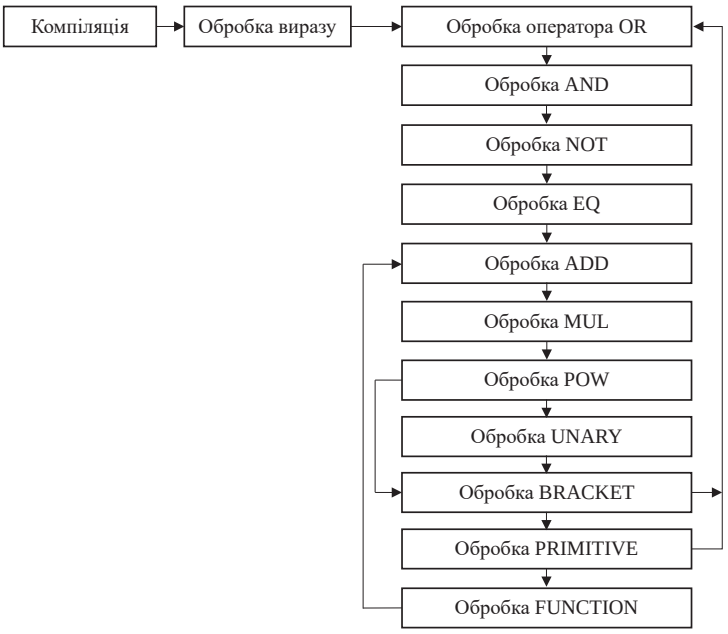

Рис. 4. Алгоритм трансляції арифметико-логічних виразів у AST

Тут публічний метод setExpression() завантажує в об'єкт класу рядок початкового коду і виконує його трансляцію у AST. Приватний метод tokenOr() здійснює обробку логічного оператору OR шляхом виклику методів tokenAnd() для обробки лівого і правого операндів диз'юнкції. Аналогічно метод tokenAnd() викликає методи tokenNot() і т. д. Слід зазначити, що обробка гілок вузла дерева розбору може бути запущена у окремих потоках виконання (threads), які будуть працювати паралельно, що істотно підвищить швидкість трансляції складних виразів.

Важливим складником будь-якого парсеру є сканер лексем. У класі Parser він реалізований за допомогою приватного методу getToken(). Результатом його роботи буде рядок, що містить чергову лексему, а також її тип (або повідомлення про помилку).

Для збереження дерева розбору виразу (поле *tree* класу Parser) було реалізовано клас Node, кожен екземпляр якого описує певний вузол AST. Його UML-діаграма має такий вигляд (рис. 5).

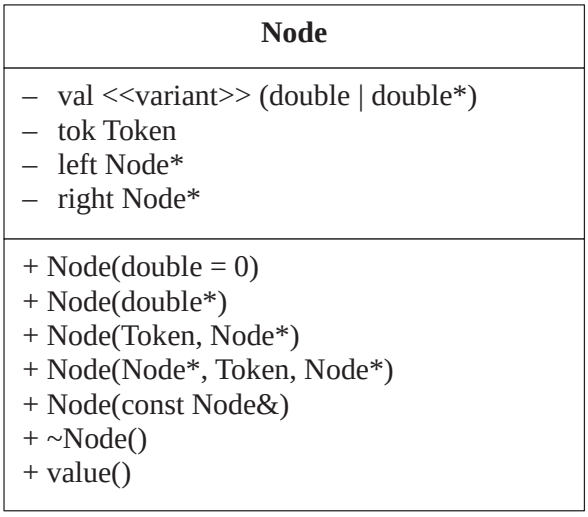

Рис. 5. UML-діаграма класу, що реалізує AST

https://doi.org/10.32782/mathematical-modelling/2025-8-2-6 ISSN 2618-0332 (print), 2618-0340 (online)

Полями Node є *val* – варіантна структура даних, що зберігає або значення дійсної константи, або покажчик на дійсне число (для реалізації можливості завантаження необхідних координат вузлів геометричної моделі в AST без необхідності перекомпіляції виразу); *tok* – код операції; *left*, *right* – покажчики на об'єкти класу Node, де зберігаються вирази, що утворюють ліву та праву гілки бінарної операції (у разі унарної операції буде використано лише одну з них).

Клас Node має декілька реалізацій конструктора, які дають змогу створювати об'єкти, що описують вузли AST будь-якого типу.

Найбільш важливим тут є рекурсивний метод value(), який залежно від реалізації може повертати або програмний код, що реалізує відповідний вираз, або значення цього виразу. Під час створення інтерпретаторів частіше за все використовується саме другий варіант.

Один із можливих варіантів реалізації цього методу може бути описаний так:

```
class Node
  private fields:
    val : <<variant>> (double | double*)
    tok : Token
    left : Node*
    right : Node*

  public method:
method value()
switch tok
        case NUMBER
                return val.get<double>()
        case PLUS
                return left.value() + right.value()
        case SIN
                return sin(left.value())
        …
end switch
end method
    …
end class
```

Повний початковий код класів, що реалізують транслятор для обчислення арифметико-логічних виразів, можна завантажити з репозиторію GitHub за посиланням [10].

Використання цього інтерпретатора у програмі на мові C++ має такий вигляд:

```
…
Parser parser;
try {
parser.setExpression(«y >= 1 and y< 2»);
double result = parser.run();
…
}
catch (…) {
    …
}
```

https://doi.org/10.32782/mathematical-modelling/2025-8-2-6 ISSN 2618-0332 (print), 2618-0340 (online)

У наведеному прикладі спочатку створюється екземпляр класу Parser (змінна *parser*), потім у нього завантажується початковий код виразу, що підлягає обробці. При цьому автоматично відбувається трансляція вихідного виразу в AST. Якщо під час цієї операції виникне помилка, буде ініційована виняткова ситуація. У разі відсутності помилок отримати значення виразу можна за допомогою метода run(), який повертає дійсне значення. Під час оброблення логічних умов слід ураховувати, що у разі якщо вираз є істинним, то run() поверне ненульове значення, і навпаки.

Розглянемо такий приклад. Нехай деяка конструкція має змінний модуль Юнга, який задається таким законом:

$$\mathrm{E} = \begin{cases} 10^{10}\ \text{Па}, y < 1, \\ 2 \cdot 10^{10}\ \text{Па}, 1 \le y < 2, \\ 3 \cdot 10^{10}\ \text{Па}, 2 \le y < 3, \\ 4 \cdot 10^{10}\ \text{Па}, y \ge 4. \end{cases} \tag{5}$$

Під час практичної реалізації МСЕ для моделювання неоднорідних фізичних характеристик можуть застосовуватися різні алгоритми розподілу значень параметра по скінченному елементу, наприклад вибір однакових значень для всіх вузлів, що відповідають значенню у центрі елемента.

Тоді візуалізація розподілу модуля Юнга по двотавровій балці з отворами згідно з (5) буде мати такий вигляд (рис. 6).

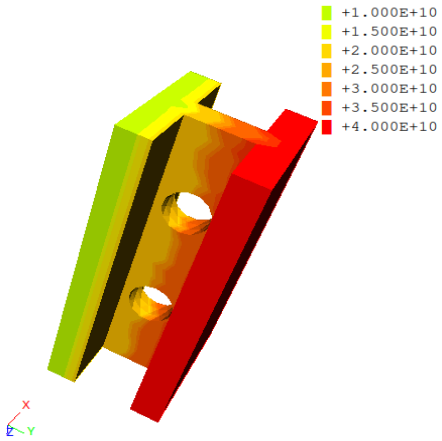

Рис. 6. Розподіл значення модуля Юнга по конструкції відповідно до закону (5)

Візуалізацію навантажень на трубу, що знаходиться під дією внутрішнього тиску (рис. 2), отриману з використанням запропонованого транслятора арифметико-логічних співвідношень, наведено на рис. 7.

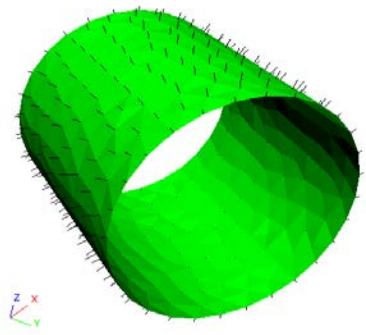

Рис. 7. Візуалізація навантажень, що діють на конструкцію згідно із законом (1)

https://doi.org/10.32782/mathematical-modelling/2025-8-2-6 ISSN 2618-0332 (print), 2618-0340 (online)

Висновки

У роботі представлено програмну реалізацію транслятора арифметико-логічних співвідношень із відкритим початковим кодом, якій можна використовувати у системах скінченно-елементного аналізу для формалізації складних крайових умов та фізичних властивостей.

Запропоновано алгоритми і структури даних, що реалізують інтерпретатор, який здійснює трансляцію арифметико-логічних та функціональних виразів у абстрактне синтаксичне дерево з подальшим обчисленням їхніх значень. Транслятор дає змогу користувачу описувати функціональні та логічні залежності у зручній формі, сумісній із синтаксисом мови Python, що суттєво спрощує підготовку вихідних даних для моделювання складних фізичних процесів.

Розроблена бібліотека класів на мові C++ забезпечує універсальність та розширюваність системи, даючи змогу інтегрувати її у програмне забезпечення для скінченно-елементного аналізу різного рівня складності. Застосування інтерпретатора показало ефективність підходу під час моделювання неоднорідних фізичних характеристик і складних навантажень, зокрема під час аналізу напружено-деформованого стану конструкцій під дією внутрішнього тиску.

Отримані результати підтверджують доцільність використання створеного інструментарію для підвищення автоматизації та гнучкості процесу формування вхідних даних у системах скінченно-елементного аналізу, що в перспективі дасть змогу зменшити трудомісткість підготовчого етапу та підвищити точність розрахунків.

Як варіант подальшого вдосконалення запропонованої бібліотеки передбачається розширення її функціональності та швидкодії за рахунок застосування паралельних обчислень і JIT-компіляції.

Список використаної літератури

1. Zienkiewicz O.C., Taylor R.L., Zhu J.Z. The Finite Element Method: Its Basis and Fundamentals. Sixth edition. Butterworth-Heinemann, 2016. 753 p.
2. Ansys. Engineering Simulation Software. URL: https://www.ansys.com/ (дата звернення: 23.08.2025).
3. MSC Nastran – Multidisciplinary Structural Analysis. URL: http://surl.li/schezo (дата звернення: 23.08.2025).
4. FreeFEM – An open-source PDE Solver using the Finite Element Method. URL: https://freefem.org/ (дата звернення: 23.08.2025).
5. Best Open-Source Finite Element Analysis Software. URL: http://surl.li/vmblnr (дата звернення: 23.08.2025).
6. Calçado P. Domain-Specific Languages (Unfinished Draft). URL: http://philcalcado.com/content/research_on_domain_specific_languages.html (дата звернення: 31.08.2025).
7. Cooper K.D., Torczon L. Engineering a Compiler. 3rd Edition. Morgan Kaufmann, 2022. 848 p.
8. Python.org. URL: https://www.python.org/ (дата звернення: 01.09.2025).
9. Aho A., Ullman J., Sethi R., Lam M. Compilers: Principles, Techniques, and Tools 2nd Edition. Addison Wesley, 2006. 1040 p.
10. QFEM/core/parser/. URL: https://surl.lt/pdmjly (дата звернення: 23.08.2025).

References

1. Zienkiewicz, O.C., Taylor, R.L., Zhu, J.Z. (2016). The Finite Element Method: Its Basis and Fundamentals. Sixth edition. Butterworth-Heinemann [in English].
2. Ansys. Engineering Simulation Software. Retrieved from https://www.ansys.com/ [in English].
3. MSC Nastran – Multidisciplinary Structural Analysis. Retrieved from http://surl.li/schezo [in English].
4. FreeFEM – An open-source PDE Solver using the Finite Element Method. Retrieved from https://freefem.org/ [in English].

https://doi.org/10.32782/mathematical-modelling/2025-8-2-6 ISSN 2618-0332 (print), 2618-0340 (online)

5. Best Open-Source Finite Element Analysis Software. Retrieved from http://surl.li/vmblnr [in English].
6. Calçado, P. Domain-Specific Languages (Unfinished Draft). Retrieved from http://philcalcado.com/content/research_on_domain_specific_languages.html [in English].
7. Cooper, K.D., Torczon, L. (2022). Engineering a Compiler. 3rd Edition. Morgan Kaufmann [in English].
8. Python.org. Retrieved from https://www.python.org/ [in English].
9. Aho, A., Ullman, J., Sethi, R., Lam, M. (2006). Compilers: Principles, Techniques, and Tools. 2nd Edition. Addison Wesley [in English].
10. QFEM/core/parser/. Retrieved from https://surl.lt/pdmjly [in English].

Гоменюк Сергій Іванович – д.т.н., професор, декан математичного факультету Запорізького національного університету. E-mail: gserega71@gmail.com, ORCID: 0000-0001-7340-5947.

Гребенюк Сергій Миколайович – д.т.н., професор, завідувач кафедри фундаментальної та прикладної математики Запорізького національного університету. E-mail: gsm1212@ukr.net, ORCID: 0000-0002-5247-9004.

Кривохата Анастасія Григорівна – к.ф.-м.н., доцент кафедри програмної інженерії Запорізького національного університету. E-mail: krivohata@gmail.com, ORCID: 0000-0001-9664-0659.

Матвіїшина Надія Вікторівна – к.т.н., доцент, доцент кафедри комп'ютерних наук Запорізького національного університету. E-mail: mnv2902@gmail.com, ORCID: 0000-0001-7938-4622.

Homeniuk Sergii Ivanovych – Doctor of Technical Sciences, Professor, Dean of the Department of Mathematics of the Zaporizhzhia National University. E-mail: gserega71@gmail.com, ORCID: 0000-0001-7340-5947.

Grebenyuk Sergii Mykolaiovych – Doctor of Technical Sciences, Professor, Head of the Department of Fundamental and Applied Mathematics of the Zaporizhzhia National University. E-mail: gsm1212@ukr.net, ORCID: 0000-0002-5247-9004.

Kryvokhata Anastasiia Hryhorivna – Candidate of Physical and Mathematical Sciences, Associate Professor at the Department of Software engineering of the Zaporizhzhia National University. E-mail: krivohata@gmail.com, ORCID: 0000-0001-9664-0659.

Matviyishyna Nadiya Viktorivna – Candidate of Technical Sciences, Associate Professor, Associate Professor at the Department of Computer Science of the Zaporizhzhia National University. E-mail: mnv2902@gmail.com, ORCID: 0000-0001-7938-4622.

Отримано: 27.10.2025
Рекомендовано: 09.12.2025
Опубліковано: 30.12.2025