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

Факультет прикладної математики

Кафедра прикладної математики

Звіт

із лабораторної роботи

із дисципліни «АЛГОРИТМИ І СИСТЕМИ КОМП'ЮТЕРНОЇ МАТЕМАТИКИ 1.МАТЕМАТИЧНІ АЛГОРИТМИ»

на тему:

Розв’язання систем лінійних рівнянь

Виконав:
студент групи КМ-13
Онищенко В.С.

Перевірила:
асистент кафедри ПМА
Ковальчук-Химюк Л.О.

Київ — 2024

ЗМІСТ

1 ВСТУП **3**

2 Порядок виконання роботи **4**

3 Основна частина **5**

3.1 Типовий перелік вимог до ПЗ 6

3.2 Опис методів 8

3.2.1 Метод Гауса 8

3.2.2 Метод LU-факторизації 9

3.3 Блок-схеми алгоритмів методів 11

3.3.1 Метод Гауса 11

3.3.2 Метод LU-факторизації 12

3.4 Верифікація розробленої програми 13

3.4.1 Python 13

3.4.2 Octave 15

3.5 Програмно отримані розв'язки 16

3.5.1 Python 16

3.5.2 Octave 17

4 ВИСНОВКИ **19**

5 СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ **21**

6 ДОДАТОК А (код роботи програм) **22**

1 ВСТУП

Мета - дослідження методів розв'язання систем лінійних алгебраїчних рівнянь (СЛАР) за допомогою чисельних методів. Основна увага приділяється вивченню правил використання програмних засобів для факторизації матриць, таких як метод Гауса та метод трикутної декомпозиції. У лабораторній роботі розглядаються можливості застосування цих методів у різних умовах, враховуючи особливості матриці коефіцієнтів та обчислювальні ресурси.

Актуальність даної роботи зумовлена широким використанням систем лінійних рівнянь в прикладних задачах математичного моделювання та інженерних обчислень. Різноманітність методів розв'язання дозволяє обирати найбільш оптимальний підхід залежно від структури системи та вимог до точності розрахунків.

У процесі виконання лабораторної роботи студенти ознайомляться з основними теоретичними відомостями, навчаться реалізовувати алгоритми чисельного розв'язання СЛАР з використанням сучасних комп'ютерних засобів та проводити порівняльний аналіз результатів. Робота сприяє розвитку навичок програмування та аналітичного мислення, необхідних для вирішення прикладних задач у галузі прикладної математики та інформатики.

Для розробки було вискористано мови Python та Octave[1]

2 Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом і методичними вказівками до роботи.

2. Одержати варіант завдання /задану систему рівнянь, використовувані підпрограми, необхідну точність розв'язку/.

3. Скласти блок-схему алгоритму розв'язання поставленої задачі.

4. Написати, налагодити і виконати програму розв'язання систем лінійних алгебраїчних рівнянь із використанням підпрограм СКМ.

5. Розв'язати задану систему лінійних алгебраїчних рівнянь і оцінити точність отриманого розв'язку по координатах.

$$\delta = \max \left| x^i - x_i^* \right|, \quad i = 1, \ldots, N;$$

де x^i — координати точного розв'язання; x_i^* — координати чисельного розв'язання; N — число невідомих.

3 Основна частина

Варіант 15:

$$\begin{pmatrix} 10.9 & 1.2 & 2.1 \\ 1.2 & 11.2 & 1.5 \\ 2.1 & 1.5 & 9.8 \\ 0.9 & 2.5 & 12.1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -0.7 \\ 5.3 \\ 10.3 \\ 24.6 \end{pmatrix} \tag{1}$$

Оскільки створене програмне забезпечення видає помилку при обробці цього варіанту через те, що матриця A не є квадратною, було обрано найбільш подібний варіант із квадратною матрицею.

Варіант 14:

$$\begin{pmatrix} 1.02 & -0.25 & -0.30 \\ -0.41 & 1.13 & -0.15 \\ -0.25 & -0.14 & 1.21 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0.515 \\ 1.555 \\ 2.780 \end{pmatrix} \tag{2}$$

3.1 Типовий перелік вимог до ПЗ

Програмне забезпечення, розроблюване в рамках виконання кожної лабораторної роботи, повинно задовольняти низку вимог, які можна розділити на *обов'язкові* (які ПЗ повинно задовольняти незалежно від лабораторної роботи) та *варіативні* (які для кожної лабораторної роботи унікальні). До обов'язкових вимог належать:

- У програмі повинно бути передбачено перевірки на некоректне введення для всіх полів введення, зокрема:

    1. порожнє введення;
    2. синтаксично некоректне введення (наприклад, літери в полі для числових коефіцієнтів);
    3. уведення спеціальних символів;
    4. уведення чисел, які перевищують максимальний розмір для чисел відповідного типу даних (для перевірки на переповнення розрядної сітки).

    У випадку некоректного введення повинно з'являтися діалогове вікно з відповідним повідомленням.

- Для всіх полів введення повинно бути визначено гранично допустиму кількість символів (для числових полів — гранично допустимі значення).

- У програмі повинно відслідковуватися переповнення розрядної сітки під час виконання обчислень. У випадку переповнення повинно з'являтися діалогове вікно з відповідним повідомленням.

- У графічному інтерфейсі користувача повинно бути передбачено можливість гнучкого налаштування розмірності розв'язуваної задачі (наприклад, можливість зміни розмірності матриці чи кількості складів у транспортній задачі).

До варіативних вимог належать вимоги щодо перевірки на коректне опрацювання виключних ситуацій, які можуть виникати під час застосування заданого методу до розв'язання поставленої задачі (наприклад, коли сума заявок не збігається з сумою ресурсів у транспортній задачі, нижня межа інтегрування перевищує верхню тощо).

3.2 Опис методів

3.2.1 Метод Гауса

Метод Гауса[2] використовується для розв'язання систем лінійних алгебраїчних рівнянь виду:

$$A\mathbf{x} = \mathbf{b}$$

де A — квадратна матриця розміру $n \times n$, **x** — вектор невідомих, і **b** — вектор правих частин.

Основні етапи методу:

1. Прямий хід На кожному кроці k (де $k = 1, 2, \ldots, n-1$) перетворюємо елементи нижче діагоналі на нулі:

$$a_{ij} = a_{ij} - \frac{a_{ik}}{a_{kk}} \cdot a_{kj} \quad \text{для } i > k,\ j = k+1, \ldots, n$$

Вектор правих частин також змінюється:

$$b_i = b_i - \frac{a_{ik}}{a_{kk}} \cdot b_k$$

Якщо на діагоналі елемент $a_{kk} = 0$, то рядки переставляються для уникнення ділення на нуль.

2. Зворотний хід Знаходження значень невідомих починаючи з останнього рівняння:

$$x_n = \frac{b_n}{a_{nn}}$$

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}}, \quad i = n-1, n-2, \ldots, 1$$

Таким чином, метод Гауса дозволяє знайти точні значення змінних $x_1, x_2, \ldots, x_n$ шляхом поетапного підстановки результатів на кожному кроці.

3.2.2 Метод LU-факторизації

Метод LU-факторизації[2] передбачає розкладання матриці A на добуток двох трикутних матриць: нижньої трикутної матриці L та верхньої трикутної матриці U, тобто:

$$A = LU$$

де:

- L — нижня трикутна матриця з одиницями на головній діагоналі,
- U — верхня трикутна матриця.

Етапи обчислення LU-факторизації

1. Запис матриці L та U На кожному кроці для $k = 1, 2, \ldots, n$ виконуємо:

- для матриці U:

$$u_{kj} = a_{kj} - \sum_{p=1}^{k-1} l_{kp} u_{pj}, \quad j = k, k+1, \ldots, n$$

- для матриці L:

$$l_{ik} = \frac{a_{ik} - \sum_{p=1}^{k-1} l_{ip} u_{pk}}{u_{kk}}, \quad i = k+1, k+2, \ldots, n$$

2. Розв'язування системи Після того як ми маємо $A = LU$, розв'язок системи $A\mathbf{x} = \mathbf{b}$ зводиться до двох етапів:

- Знаходимо проміжний вектор **y** з рівняння:

$$L\mathbf{y} = \mathbf{b}$$

який розв'язується послідовним підстановленням знизу вверх.

- Потім розв'язуємо $U\mathbf{x} = \mathbf{y}$ для знаходження вектора $\mathbf{x}$ також послідовним підстановленням.

LU-факторизація є ефективним методом для розв'язування систем лінійних рівнянь, особливо при багаторазовому обчисленні з різними векторами $\mathbf{b}$, оскільки дозволяє уникнути повторного розкладання матриці A.

3.3 Блок-схеми алгоритмів методів

3.3.1 Метод Гауса

На фігурі 3.3.1.1 зображено діаграму методу LU-факторизації

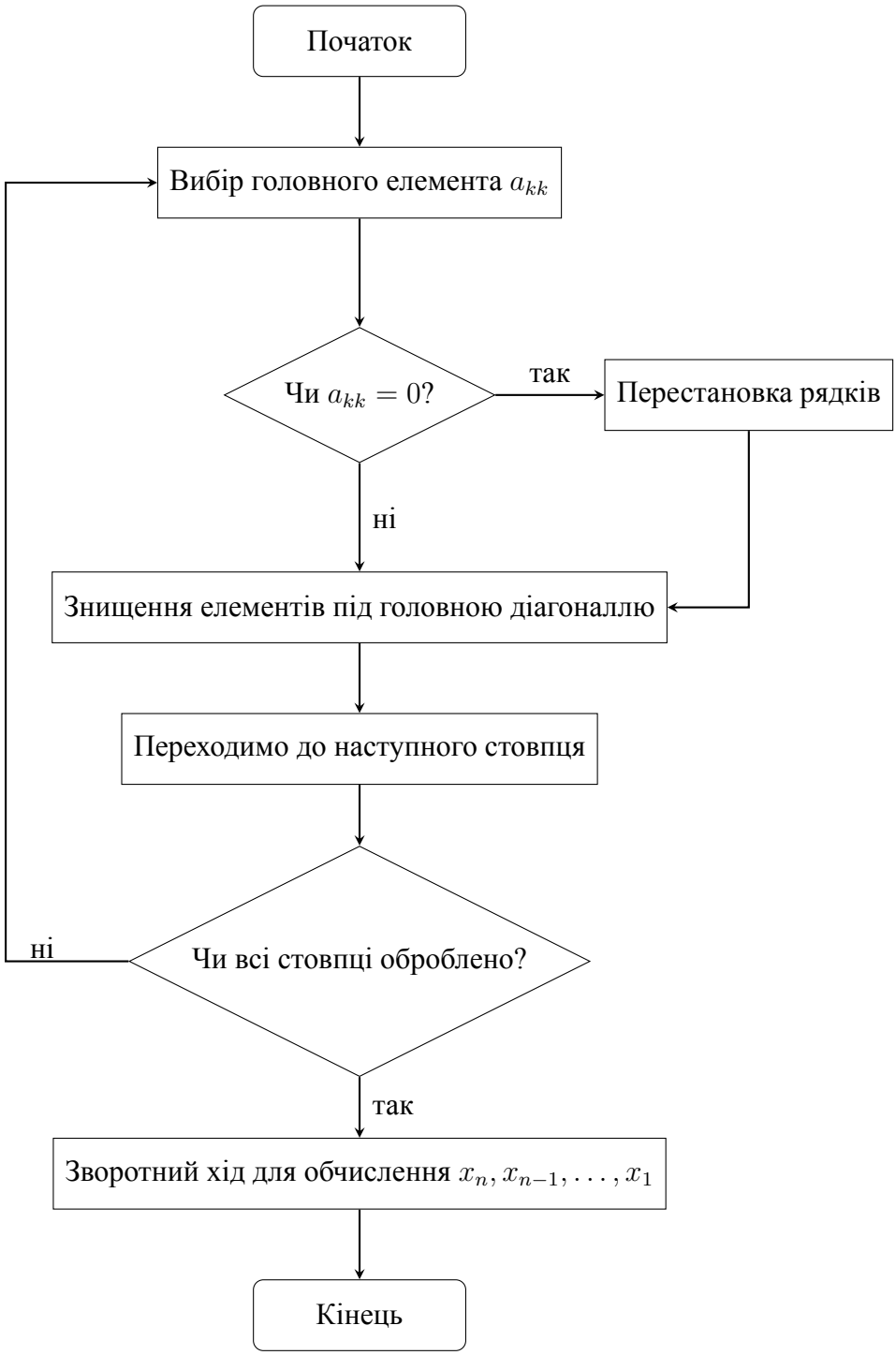

Рисунок 3.3.1.1 -- Діаграма методу Гауса

3.3.2 Метод LU-факторизації

На фігурі 3.3.2.1 зображено діаграму методу LU-факторизації

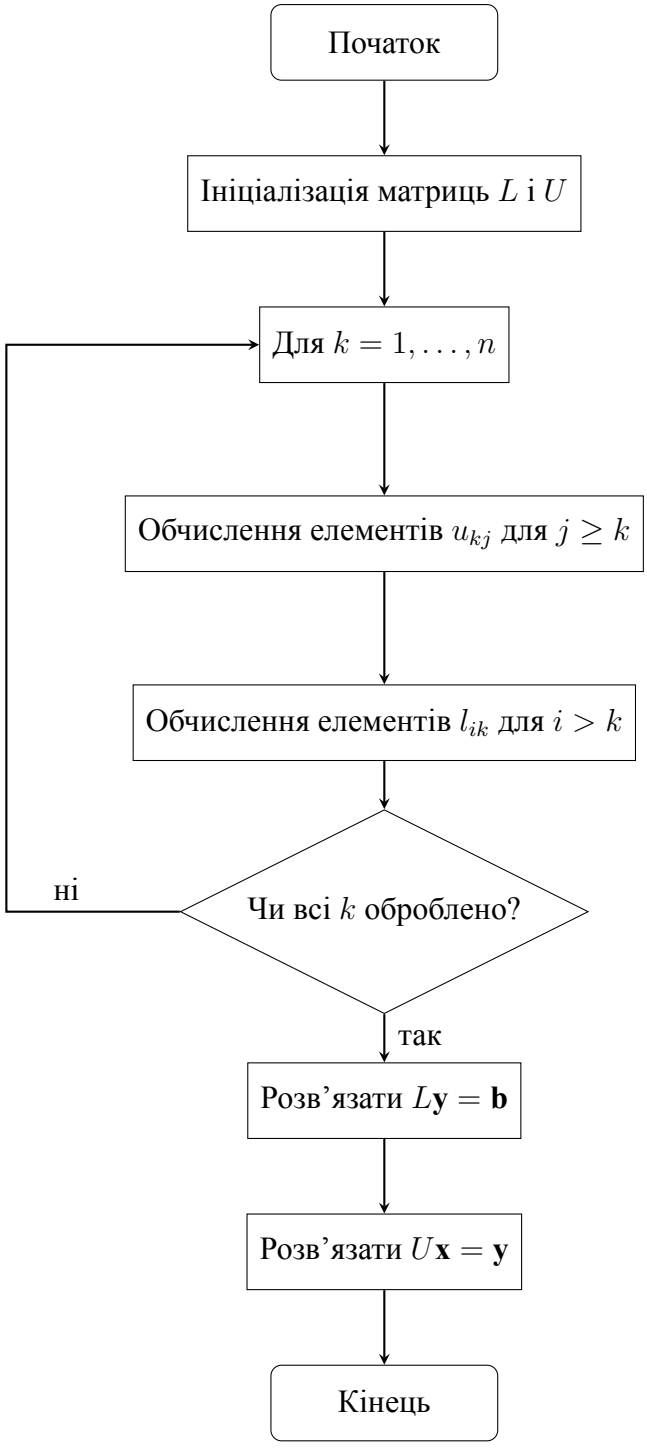

Рисунок 3.3.2.1 -- Діаграма методу LU-факторизації

3.4 Верифікація розробленої програми

3.4.1 Python

Для успішного розв'язання системи рівнянь методом Гауса та LU-факторизації необхідно виконати валідацію вхідних даних. Валідація включає наступні кроки:

1. **Перевірка типів даних.** Вхідні матриця A та вектор b мають бути представлені у вигляді масивів бібліотеки `numpy`. Якщо вхідні дані не є `numpy` масивами, алгоритм видасть помилку, оскільки не всі операції будуть доступними. (Див. рис. 3.4.1.1)

   isinstance(A, np.ndarray) and isinstance(b, np.ndarray)

2. **Перевірка на числові значення.** Елементи матриці A та вектора b мають бути числовими (дійсними або цілими числами). Якщо вхідні дані містять нечислові значення, методи обчислення не зможуть коректно працювати. (Див. рис. 3.4.1.2)

   np.issubdtype(A.dtype, np.number) and np.issubdtype(b.dtype, np.number)

3. **Перевірка розмірів матриці та вектора.** Матриця A повинна бути квадратною, а кількість елементів у векторі b повинна відповідати кількості рядків у матриці A. Ця перевірка гарантує, що система рівнянь є узгодженою для розв'язання. (Див. рис. 3.4.1.3)

   $$A.shape[0] = A.shape[1] \quad \text{та} \quad A.shape[0] = b.shape[0]$$

4. **Перевірка на виродженість матриці.** Визначник матриці A не повинен

дорівнювати нулю, щоб система мала єдиний розв'язок. Якщо матриця вироджена (детермінант $\det(A) = 0$), система може не мати унікального розв'язку. (Див. рис. 3.4.1.4)

$$\det(A) \neq 0$$

```
--------------------------------------------------

Тест 5 пройдено: Некоректний тип даних
Вхідні дані:
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
b = [0.515, 1.555, 2.78]

--------------------------------------------------
```

Рисунок 3.4.1.1 -- Перевірка типів даних

```
Тест 6 пройдено: Некоректні дані у матриці A
Вхідні дані:
A = [['1.02' '-0.25' '-0.3']
 ['-0.41' 'invalid' '-0.15']
 ['-0.25' '-0.14' '1.21']]
b = [0.515 1.555 2.78 ]

--------------------------------------------------

Тест 7 пройдено: Некоректні дані у векторі b
Вхідні дані:
A = [[ 1.02 -0.25 -0.3 ]
 [-0.41  1.13 -0.15]
 [-0.25 -0.14  1.21]]
b = ['0.515' 'invalid' '2.78']
```

Рисунок 3.4.1.2 -- Перевірка на числові значення

```
Тест 2 пройдено: Некоректний розмір вектора b
Вхідні дані:
A = [[ 1.02 -0.25 -0.3 ]
 [-0.41  1.13 -0.15]
 [-0.25 -0.14  1.21]]
b = [0.515 1.555]

--------------------------------------------------

Тест 3 пройдено: Неквадратна матриця A
Вхідні дані:
A = [[ 1.02 -0.25]
 [-0.41  1.13]
 [-0.25 -0.14]]
b = [0.515 1.555 2.78 ]
```

Рисунок 3.4.1.3 -- Перевірка розмірів матриці та вектора

```
--------------------------------------------------

Тест 4 пройдено: Вироджена матриця A
Вхідні дані:
A = [[1 2 3]
 [4 5 6]
 [7 8 9]]
b = [0.515 1.555 2.78 ]

--------------------------------------------------
```

Рисунок 3.4.1.4 -- Перевірка на виродженість матриці

3.4.2 Octave

У програмі, розробленій на GNU Octave, перевірка правильності введення даних не передбачена, оскільки вона орієнтована на швидке обчислення результатів за умови, що користувач вводить коректні дані.

3.5 Програмно отримані розв'язки

В результаті виконання методів Гауса та LU-факторизації за допомогою мов програмування Python та Octave були отримані розв'язки системи рівнянь разом з похибками та покоординатною точністю. Результати для кожної мови програмування наведені нижче.

3.5.1 Python

На мові Python були отримані наступні розв'язки та похибки для методів Гауса і LU-факторизації:

- **Розв'язок системи методом Гауса:** $[2.0, 2.5, 3.0]$
- **Похибка методу Гауса:** $5.551115123125783 \times 10^{-16}$
- **Покоординатна точність методу Гауса:** $[4.4408921 \times 10^{-16}, 0, 0]$
- **Розв'язок системи методом LU-факторизації:** $[2.0, 2.5, 3.0]$
- **Похибка методу LU-факторизації:** $4.577566798522237 \times 10^{-16}$
- **Покоординатна точність методу LU-факторизації:** $[0, 0, 4.4408921 \times 10^{-16}]$

Зображення з результатами, отриманими на мові Python, наведено на рис. 3.5.1.1.

```
Розв'язок системи методом Гауса: [2.  2.5 3. ]
Похибка методу Гауса: 5.551115123125783e-16
Покоординатна точність методу Гауса: [4.4408921e-16 0.0000000e+00 0.0000000e+00]
Розв'язок системи методом LU-факторизації: [2.  2.5 3. ]
Похибка методу LU-факторизації: 4.577566798522237e-16
Покоординатна точність методу LU-факторизації: [0.0000000e+00 0.0000000e+00 4.4408921e-16]
```

Рисунок 3.5.1.1 -- Результати виконання програмного коду на мові Python

3.5.2 Octave

На мові Octave були отримані наступні розв'язки та похибки для методів Гауса і LU-факторизації:

- **Solution using Gauss method:** $[2.0, 2.5, 3.0]$
- **Error of Gauss method:** 0.000000
- **Coordinate-wise accuracy of Gauss method:** $[4.4409 \times 10^{-16}, 0, 0]$
- **Solution using LU factorization:** $[2.0, 2.5, 3.0]$
- **Error of LU factorization:** 0.000000
- **Coordinate-wise accuracy of LU factorization:** $[0, 0, 4.4409 \times 10^{-16}]$

Зображення з результатами, отриманими на мові Octave, наведено на рис. 3.5.2.1.

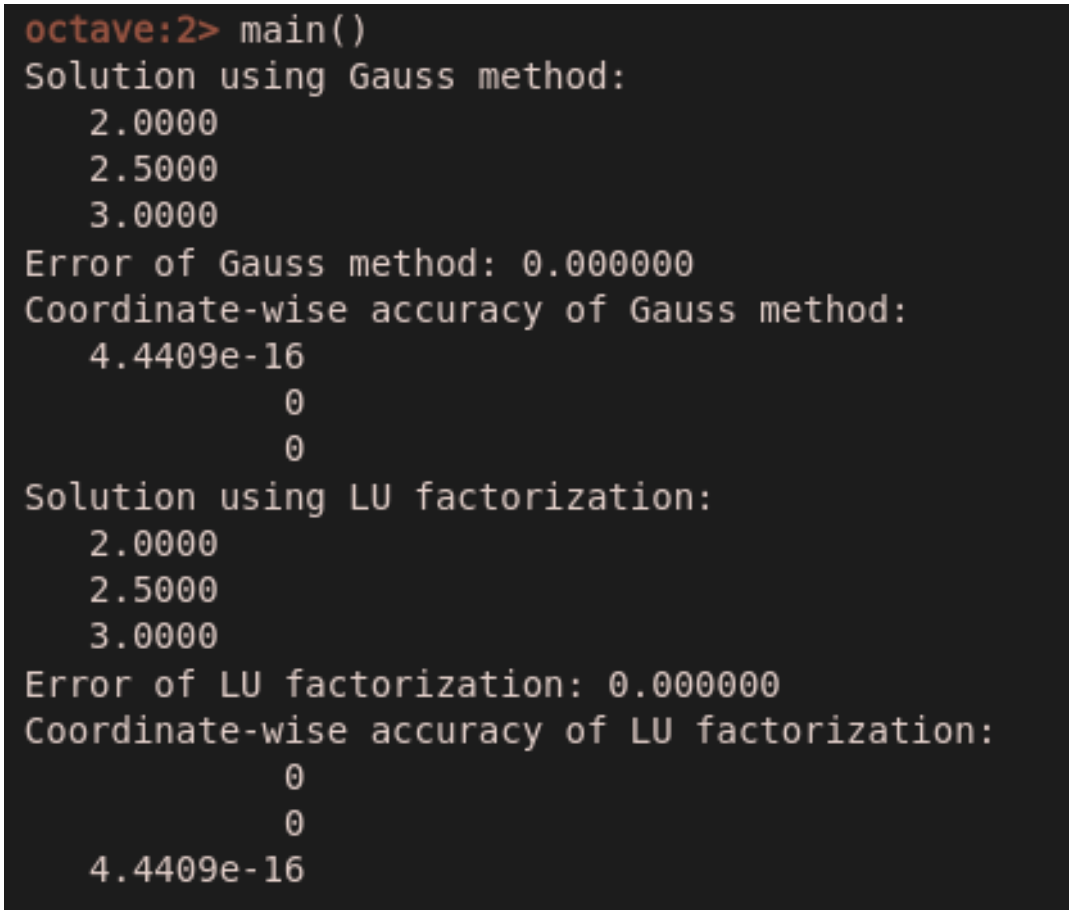

Рисунок 3.5.2.1 -- Результати виконання програмного коду на мові Octave

4 ВИСНОВКИ

Метою даної лабораторної роботи було дослідження методів розв'язання систем лінійних алгебраїчних рівнянь (СЛАР) за допомогою чисельних методів. Зокрема, були розглянуті методи Гауса та LU-факторизації для знаходження розв'язків СЛАР, реалізовані мовами програмування Python та Octave.

В результаті роботи було отримано наступні основні висновки:

- Обидва методи (Гауса та LU-факторизації) успішно знаходять розв'язок для заданої системи рівнянь. Розв'язок у кожному випадку є однаковим: $[2.0, 2.5, 3.0]$, що свідчить про правильність реалізації обох методів на обох мовах програмування.

- Похибка обчислень в обох методах дуже мала, знаходиться в межах 10^{-16}, що свідчить про високу точність обчислень для даної системи рівнянь. Однак, варто зазначити, що метод Гауса на мові Octave показав нульову похибку, що може бути пов'язано з особливостями чисельної реалізації та округлення в Octave порівняно з Python.

- Покоординатна точність також підтверджує, що розв'язки, отримані методами Гауса та LU-факторизації, є дуже близькими до точного розв'язку. Найбільша покоординатна похибка знаходиться на рівні 4.4409×10^{-16}, що також вказує на високу стабільність обчислень обох методів.

- Порівнюючи реалізації на мовах Python та Octave, можна відзначити, що:

  – Реалізація методів на мові Python є більш гнучкою та універсальною, оскільки Python надає широкий спектр бібліотек для наукових обчислень та обробки даних. Вона також демонструє надійні результати з мінімальними похибками.

  - Octave є зручною мовою для чисельних розрахунків, подібною до MATLAB, і показує високу точність обчислень із незначними похибками. Однак, на відміну від Python, Octave має більш вузькоспеціалізовану екосистему.

- Обидва підходи до реалізації чисельних методів є ефективними для розв’язання СЛАР невеликого розміру та дозволяють швидко отримати точний розв'язок.

Загалом, обидва методи та обидві мови програмування продемонстрували високу точність та надійність при розв'язанні системи лінійних рівнянь. Python надає більше можливостей для подальшого використання і обробки результатів, тоді як Octave є ефективним інструментом для швидких чисельних обчислень, що може бути корисним для задач, де важлива простота і швидкість реалізації.

5 СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Eaton, J. W., Bateman, D., Hauberg, S. & Wehbring, R. *GNU Octave 4.0 Reference Manual: Free Your Numbers* ISBN: 978-9888381050 (Free Software Foundation, 2015).
2. Костюшко, І. А., Любашенко, Н. Д. & Третиник, В. В. *Методи обчислень* Бібліографія: с. 241–242, 243 (Вид-во «Політехніка», КПІ ім. Ігоря Сікорського, Київ, 2021).

6 ДОДАТОК А (код роботи програм)

Python

```python
import numpy as np

def validate_input(A, b):
    # Перевірка типів даних
    if not (isinstance(A, np.ndarray) and isinstance(b, np.ndarray)):
        raise ValueError("Вхідні дані мають бути у вигляді numpy масивів")
    if not (np.issubdtype(A.dtype, np.number)
        and np.issubdtype(b.dtype, np.number)):
        raise ValueError("Матриця A і вектор b мають \
        містити лише числові значення")

    # Перевірка розмірів матриці A та вектора b
    if A.shape[0] != A.shape[1]:
        raise ValueError("Матриця A повинна бути квадратною")
    if A.shape[0] != b.shape[0]:
        raise ValueError("Кількість рядків у A повинна \
        дорівнювати кількості елементів у b")

    # Перевірка на виродженість матриці
    if np.linalg.det(A) == 0:
        raise ValueError("Матриця A є виродженою, \
        і система може не мати унікального розвязку'")

def gauss_method(A, b):
    validate_input(A, b)
    n = len(b)

    # Копіюємо A та b для уникнення зміни вхідних даних
    A = A.astype(float).copy()
```

```
    b = b.astype(float).copy()

    # Прямий хід
    for k in range(n):
        if abs(A[k, k]) < 1e-10:
            raise ValueError("Нульовий елемент на діагоналі; \
            система можливо вироджена або вимагає перестановки рядків")

        for i in range(k + 1, n):
            factor = A[i, k] / A[k, k]
            A[i, k:] -= factor * A[k, k:]
            b[i] -= factor * b[k]

    # Зворотний хід
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i]

    return x

def lu_factorization(A, b):
    validate_input(A, b)
    n = A.shape[0]

    # Ініціалізація матриць L та U
    L = np.eye(n)
    U = A.astype(float).copy()

    # LU факторизація
    for k in range(n):
        if abs(U[k, k]) < 1e-10:
            raise ValueError("Нульовий елемент на діагоналі; \
```

```
            система можливо вироджена або вимагає перестановки рядків")

        for i in range(k + 1, n):
            factor = U[i, k] / U[k, k]
            L[i, k] = factor
            U[i, k:] -= factor * U[k, k:]

    # Розв'язування L * y = b
    y = np.zeros(n)
    for i in range(n):
        y[i] = b[i] - np.dot(L[i, :i], y[:i])

    # Розв'язування U * x = y
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (y[i] - np.dot(U[i, i+1:], x[i+1:])) / U[i, i]

    return x

def calculate_error(A, x, b):
    # Обчислення похибки як норма вектора нев'язки
    residual = np.dot(A, x) - b
    error = np.linalg.norm(residual)
    return error

def calculate_coordinate_accuracy(exact_solution, approx_solution):
    # Обчислення покоординатної точності
    coordinate_errors = np.abs(exact_solution - approx_solution)
    return coordinate_errors

# Вхідні дані
A = np.array([
```

```
    [1.02, -0.25, -0.30],
    [-0.41, 1.13, -0.15],
    [-0.25, -0.14, 1.21]
])
b = np.array([0.515, 1.555, 2.780])

# Точний розвязок'
exact_solution = np.linalg.solve(A, b)

# Перевірка вхідних даних
try:
    validate_input(A, b)
except ValueError as e:
    print("Помилка у вхідних даних:", e)
else:
    # Виконання методу Гауса
    try:
        solution_gauss = gauss_method(A, b)
        error_gauss = calculate_error(A, solution_gauss, b)
        coord_accuracy_gauss = calculate_coordinate_accuracy(
            exact_solution, solution_gauss)
        print("Розвязок' системи методом Гауса:", solution_gauss)
        print("Похибка методу Гауса:", error_gauss)
        print("Покоординатна точність методу Гауса:", coord_accuracy_gauss
    except ValueError as e:
        print("Помилка метод( Гауса):", e)

    # Виконання методу LUфакторизації-
    try:
        solution_lu = lu_factorization(A, b)
        error_lu = calculate_error(A, solution_lu, b)
        coord_accuracy_lu = calculate_coordinate_accuracy(
```

```
            exact_solution, solution_lu)
        print("Розвязок' системи методом LUфакторизації-:", solution_lu)
        print("Похибка методу LUфакторизації-:", error_lu)
        print("Покоординатна точність методу LUфакторизації-:",
              coord_accuracy_lu)
    except ValueError as e:
        print("Помилка метод( LUфакторизації-):", e)
```

Octave

```
function main()
    % Input data
    A = [1.02, -0.25, -0.30; -0.41, 1.13, -0.15; -0.25, -0.14, 1.21];
    b = [0.515; 1.555; 2.780];

    % Exact solution
    exact_solution = A \ b;

    % Execute Gauss method
    gauss_solution = gauss_method(A, b);
    gauss_error = calculate_error(A, gauss_solution, b);
    gauss_coord_accuracy = calculate_coordinate_accuracy(exact_solution, g

    fprintf("Solution using Gauss method:\n");
    disp(gauss_solution);
    fprintf("Error of Gauss method: %.6f\n", gauss_error);
    fprintf("Coordinate-wise accuracy of Gauss method:\n");
    disp(gauss_coord_accuracy);

    % Execute LU factorization method
    lu_solution = lu_factorization(A, b);
    lu_error = calculate_error(A, lu_solution, b);
```

```
    lu_coord_accuracy = calculate_coordinate_accuracy(exact_solution, lu_s

    fprintf("Solution using LU factorization:\n");
    disp(lu_solution);
    fprintf("Error of LU factorization: %.6f\n", lu_error);
    fprintf("Coordinate-wise accuracy of LU factorization:\n");
    disp(lu_coord_accuracy);
endfunction

function gauss_solution = gauss_method(A, b)
    n = length(b);
    % Forward elimination
    for k = 1:n
        for i = k+1:n
            factor = A(i, k) / A(k, k);
            A(i, k:n) = A(i, k:n) - factor * A(k, k:n);
            b(i) = b(i) - factor * b(k);
        end
    end

    % Back substitution
    gauss_solution = zeros(n, 1);
    for i = n:-1:1
        gauss_solution(i) = (b(i) - A(i, i+1:n) * gauss_solution(i+1:n)) /
    end
endfunction

function lu_solution = lu_factorization(A, b)
    n = size(A, 1);
    L = eye(n);
    U = A;

```

```
    % LU factorization
    for k = 1:n
        for i = k+1:n
            factor = U(i, k) / U(k, k);
            L(i, k) = factor;
            U(i, k:n) = U(i, k:n) - factor * U(k, k:n);
        end
    end

    % Solving L * y = b
    y = zeros(n, 1);
    for i = 1:n
        y(i) = b(i) - L(i, 1:i-1) * y(1:i-1);
    end

    % Solving U * x = y
    lu_solution = zeros(n, 1);
    for i = n:-1:1
        lu_solution(i) = (y(i) - U(i, i+1:n) * lu_solution(i+1:n)) / U(i,
    end
endfunction

function error = calculate_error(A, x, b)
    % Calculate error as the norm of the residual vector
    residual = A * x - b;
    error = norm(residual);
endfunction

function coord_accuracy = calculate_coordinate_accuracy(exact_solution, ap
    % Calculate coordinate-wise accuracy
    coord_accuracy = abs(exact_solution - approx_solution);
endfunction
```