МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з ___________________Основи програмування____________________

(назва дисципліни)

на тему:__Розв'язання СЛАР точними методами_________________________

____

Студента (ки, ів) __1__ курсу, групи _ІП-35_
___Адаменко Арсен Богданович___________

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник
___ст. викладач,_Головченко М.М_________
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: ________________________
Національна оцінка ____________________

Члени комісії

________________ (підпис) ______________________________ (посада, вчене звання, науковий ступінь, прізвище та ініціали)

________________ (підпис) ______________________________ (посада, вчене звання, науковий ступінь, прізвище та ініціали)

Київ- 2024 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-35 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Адаменко Арсен Богданович

(прізвище, ім'я, по батькові)

1. Тема роботи Розв'язання СЛАР точними методами

2. Строк здачі студентом закінченої роботи БУДЕ ОТРИМАТНО

3. Вихідні дані до роботи Технічне завдання додаток А

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

ВСТУП, ПОСТАНОВКА ЗАДАЧІ, ТЕОРЕТИЧНІ ВІДОМОСТІ, ОПИС АЛГОРИТМІВ, ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ТЕСТУВАННЯ

ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ВИСНОВКИ, ПЕРЕЛІК ПОСИЛАНЬ, ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ, ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ

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

6. Дата видачі завдання 03 квітня 2024

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	03 квітня 2024	
2.	Підготовка ТЗ	03 квітня 2024	
3.	Пошук та вивчення літератури з питань курсової роботи	22 квітня 2024	
4.	Розробка сценарію роботи програми	06 травня 2024	
6.	Узгодження сценарію роботи програми з керівником	15 травня 2024	
5.	Розробка (вибір) алгоритму рішення задачі	06 травня 2024	
6.	Узгодження алгоритму з керівником	15 травня 2024	
7.	Узгодження з керівником інтерфейсу користувача	8 травня 2024	
8.	Розробка програмного забезпечення	22 квітня 2024	
9.	Налагодження розрахункової частини програми	15 травня 2024	
10.	Розробка та налагодження інтерфейсної частини програми	28 травня 2024	
11.	Узгодження з керівником набору тестів для контрольного прикладу	24 травня 2024	
12.	Тестування програми	25 травня 2024	
13.	Підготовка пояснювальної записки	31 травня 2024	
14.	Здача курсової роботи на перевірку	31 травня 2024	
15.	Захист курсової роботи	7 червня 2024	

Студент ________________________

(підпис)

Керівник ________________________ Головченко Максим Миколайович

(підпис) (прізвище, ім’я, по батькові)

"___" ____________________20__ р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 62 сторіна, 14 рисунків, 12 таблиць, 3 посилання. (БУДЕ ОПИСАНО)

Мета роботи: Метою курсової роботи є забезпечення надійності та коректності програмного забезпечення для розв’язання СЛАР різними точними методами.

Вивчено методи: LUP-метод, метод Гауса-Холецького, метод обертання.

Виконана програмна реалізація алгоритму LUP-метода, метода Гауса-Холецького, метода обертання.

СИСТЕМА ЛІНІЙНИХ РІВНЯНЬ, ТОЧНІ МЕТОДИ ВИРІШЕННЯ, LUP-МЕТОД, МЕТОД ГАУСА-ХОЛЕЦЬКОГО, МЕТОД ОБЕРТАННЯ.

ЗМІСТ

ВСТУП..5
1 ПОСТАНОВКА ЗАДАЧІ..6
2 ТЕОРЕТИЧНІ ВІДОМОСТІ..7
3 ОПИС АЛГОРИТМІВ..8
3.1. Загальний алгоритм..8
3.2. Алгоритм LUP-методу..8
3.3. Алгоритм методу Гауса-Холецького..8
3.4. Алгоритм методу обертання..8
4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ..9
4.1. Діаграма класів програмного забезпечення..9
4.2. Опис методів частин програмного забезпечення..9
4.2.1. Користувацькі методи..9
4.2.2. Стандартні методи..9
5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ..10
5.1. План тестування..10
5.2. Приклади тестування..10
6 ІНСТРУКЦІЯ КОРИСТУВАЧА..11
7 АНАЛІЗ РЕЗУЛЬТАТІВ..12
ВИСНОВКИ..13
ПЕРЕЛІК ПОСИЛАНЬ..14
ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ..15
ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ..18

ВСТУП

Дана робота висвітлює методи розв’язання систем лінійних алгебраїчних рівнянь для вирішення великої кількості задач та проблем у напрямках галузі інформаційних технологій, які пов’язані з такими речами, алгоритмами та методами, як графіка, штучний інтелект і машинне навчання.

Дана робота та програмне забезпечення є актуальним через використання нових підходів до розробки та супроводу програмного продукту, забезпечення додактового функціоналу для програмного забезпечення з призначенням цього виду.

Дане програмне забезпечення призначене для розв’язання систем лінійних алгебраїчних рівнянь різними методами з підрахунком практичної часової складності (кількість ітерацій), візуалізації розв’язків систем рівнянь, а також вивід розв’язків систем рівнянь до текстового файлу.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення з графічним інтерфейсом, що буде знаходити рішення для заданої СЛАР наступними методами:

а) LUP-метод;

б) метод Гауса-Холецького;

в) метод обертання.

Вхідними даними для даної роботи є СЛАР, яка задана в матричному вигляді:

$$AX = B$$

, де A – матриця коефіцієнтів, X – вектор шуканих значень (рішення системи),

B – вектор вільних членів. Програмне забезпечення повинно обробробляти матрицю коефіцієнтів та стовпець вільних членів для СЛАР розмірність яких знаходиться в межах від 1 до 10.

Вихідними даними для даної роботи являється сукупність дійсних чисел, що є розв’язками даної системи, які виводяться на екран. Програмне забезпечення повинно видавати розв’язок за умови, що для вхідних даних СЛАР метод дає коректний результат. Якщо це не так, то програма повинна вивести відповідне повідомлення. Якщо розмірність системи рівне двом невідомим, то програмне забезпечення повинно виводити графік системи та його розв’язок якщо вхідну СЛАР можна вирішити вказаним методом. Якщо система не має розв’язків, то програма повинна видати відповідне повідомлення.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Систему лінійних алгебраїчних рівнянь (далі — СЛАР) з n рівнянь можна задати наступним чином:

$$AX = B \quad (2.1)$$

де:

$$A = \begin{bmatrix} a_{11} & a_{21} & \cdots & 2 \\ a_{12} & a_{22} & \cdots & 2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \vdots & a_{nn} \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Тоді якщо ранг матриці коефцієнтів (2.1) є рівним кількості лінійних рівнянь, то система (2.1) має розв'язок та він є єдиним. Якщо система має єдиний розв'язок, то його можна знайти одним із наступних методів.

2.1. LUP-метод

Цей метод опирається на розкладання матриці коефіцієнтів A у вигляді добутку матриць L та U [1]:

$$A = LU \quad (2.2)$$

де L — нижня трикутна матриця, а U — верхня трикутна матриця, де усі діагональні елементи дорівнюють 1:

$$L = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ 0 & 1 & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \quad (2.3)$$

Тоді з цього твердження випливає, що LUP-метод розв'язання СЛАР складається з двох головних етапів:

1. етапу факторизації матриці A;
2. етапу отримання X.

Під час факторизації вектор B не змінюється, як ви бачите.

Із виразів (2.2) та (2.3) випливає, що значення всіх елементів матриці A можна записати наступним компактним чином:

$$a_{ij}=\left(\sum_{k=1}^{j-1} l_{ik} u_{kj}\right)+l_{ij}$$
$$a_{ji}=\left(\sum_{k=1}^{j-1} l_{jk} u_{ki}\right)+l_{jj} u_{ji} \qquad (2.4)$$

де $i=\overline{1,n}$, $j=\overline{1,i}$.

У виразі (2.4) записано, що i — номер рядка матриці L та номер стовпця матриці U, а j — вже номер стовпця матриці L та номер рядка матриці U.

З рівняння (2.4) випливає, що факторизація відбувається за n стадій. На кожній стадії j поточний елемент a_{ji} матриці L наступним чином:

$$l_{ij}=a_{ij}-\sum_{k=1}^{j-1} l_{ik} u_{kj}, \quad i=\overline{j,n} \ (2.5)$$

а елемент u_{ji} вже іншим:

$$u_{ji}=\frac{a_{ji}-\sum_{k=1}^{j-1} l_{jk} u_{ki}}{l_{jj}}, \quad i=\overline{j+1,n} \ (2.6)$$

Елементи двох матриць обчислюються у шаховому порядку.

За деяких умовах за формулами (2.5) та (2.6) значення елементів матриці L та U можна побати як $l_{i1}=a_{i1}, i=\overline{1,n}$, а $u_{1j}=\frac{a_{1j}}{l_{11}}, j=\overline{2,n}$.

Так як метод називається як LUP-методом, а не LU-методом, то він має ще один вбудований крок для факторизації матриці A. Так як при обрахуванні елемента u_{ji} значення l_{jj} може набувати від'ємного значення, що зробить розв’язання неможливим, або настільки близьким до нуля, що точність вирішення СЛАР може погіршитися.

Одним з методов вирішення цієї проблеми є знаходження такого рядка $k=\overline{j,n}$, для якого:

$$|a_{kj}|=|\max a_{ij}|, \quad i=\overline{j,n} \ (2.7)$$

Після чого треба зберігти вектор P, де $P_i \in \overline{1,n}, i=\overline{1,n}$ зберігає оригінальний номер рядка матриці A. По замовчуванням матриця P має наступну початкову конфігурацію перед факторизацією: $P_i=i, i=\overline{1,n}$.

Так як перед обрахуванням елементів матриці L та U в ітерації j елементи рядків $i=\overline{j,n}$ є незмінними до цього часу, то слід зробити перестановку перед поточною ітерацією j, а також обміняти місцями значення P_j та P_j, де j — поточний номер ітерації, а i — значення рядка, для якого формула (2.7) коректною.

В результаті факторизації ми маємо наступне рівняння СЛАР:

$$LUX=B \quad (2.8)$$

Далі рівняння (2.8) можна перезаписати як:

$$LY=B, Y=UX \quad (2.9)$$

Потім в нас з (2.9) з'являється наступне рівняння для розв'язку Y:

$$LY=B \quad (2.10)$$

А також (2.9) можна перетворити для розв'язку X:

$$UX=Y \quad (2.11)$$

Так як L є трикутною матрицею, то для розв'язку Y ми можемо перетворити рівняння (2.10) на наступне:

$$y_i=\frac{b_{p_{ii}}-\sum_{j=1}^{i-1} l_{ij} y_i}{l_{ii}}, \quad i=\overline{1,n}$$

де P — вектор перестановки для матриці A.

Так як U теж є трикутною матрицею, то вектор-стовпець розв'язок X можна перезаписати за допомогою формули (2.11) як:

$$x_i=y_i-\sum_{j=i+1}^{n} u_{ij} x_j, \quad i=\overline{n,1}$$

2.2. Метод Гауса-Холецького

Цей метод використовується для розв'язання СЛАР з матрицею коефіцієнтів A, яка має бути емітовою. Він грунтується на розкладанні матриці на добуток матирць L, D та $L^{\dagger}$ [1]:

$$A=LDL^{\dagger} \quad (2.12)$$

де L — нижня трикутна матриця, D — діагональна матриця, $L^{\dagger}$ — комплексно спряжененя матриця до матриці L та є верхньо трикутною до неї.

З виразу (2.12) випливає, що кожен елемент матриці A можна записати як:

$$a_{ij}=\sum_{k=1}^{i} l_{ik} d_{kk} \overline{l_{jk}}, \quad i \geq j \text{ (2.13)}$$

де $\overline{l_{jk}}$ — елемент, комплексно-спряжений до l_{jk}.

У випадку, якщо $i=j$, то можна одержати наступне рівняння:

$$d_{jj} l_{jj}^{2}=a_{jj}-\sum_{k-1}^{j-1} l_{jk}^{2} d_{kk} \text{ (2.14)}$$

Якщо $l_{ii}=1, i=\overline{1,n}$, то вираз (2.14) можна перезаписати вже ось так:

$$d_{jj}=a_{jj}-\sum_{k=1}^{j-1}\left(k_{jk}^{2} d_{kk}\right), \quad j=\overline{1,n} \text{ (2.15)}$$

Тепер якщо $i>j$, то вираз (2.14) вже буде мати наступний вигляд:

$$l_{ij}=\frac{a_{ij}-\sum_{j=1}^{j-1} l_{ik} d_{kk} \overline{l_{jk}}}{d_{jj}}, \quad j=\overline{1,n}, i=\overline{j+1,n} \text{ (2.16)}$$

Таким чином, LDL факторизація матриці відбувається за n ітерацій. Всі елементи $l_{ii}, i=\overline{1,n}$ дорвівнюють 1. На кожній ітерації j треба обчислити значення елементів d_{jj} за формулою (2.15), а потім — l_{ij} за (2.16).

Після факторизації матриці A ми отримаємо наступне рівняння:

$$LDL^{\dagger} X=B \text{ (2.17)}$$

Вираз (2.17) можна перезаписати як:

$$LY=B, \quad Y=DL^{\dagger} X \text{ (2.18)}$$

далі як:

$$DZ=Y, \quad Z=L^{\dagger} X \text{ (2.19)}$$

в кінці вже як:

$$L^{\dagger} X=Z \text{ (2.20)}$$

З рівності (2.18) випливає, що:

$$y_{i}=b_{i}-\sum_{j=1}^{i-1} l_{ij} y_{j}, \quad i=\overline{1,n}$$

З виразу (2.19):

$$z=\frac{y_i}{d_{ii}},\quad i=\overline{1,n}$$

Далі з (2.20) випливає остаточним вираз обчислення вектора X:

$$x_i=z_i-\sum_{j=i+1}^{n}\overline{l_{ji}}x_j,\quad i=\overline{n,1}$$

2.3. Метод обертання

Цей метод опирається на обнуленні значень матриці коефіцієнтів A та зміні вектора вільних членів B до верхньотрикутного вигляду та використанні етапу знаходження вектора значень змінних метода Гауса [2].

Давайте розглянемо систему лінійних рівнянь наступного вигляду:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ \cdots & & \cdots & & & & \cdots & & \cdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{nn}x_n & = & b_n \end{array} \quad (2.21)$$

Нехай c_1 і s_1 — значення, відмінні від нуля. Нехай вони будуть рівними наступним значенням:

$$c_1=\frac{a_{11}}{\sqrt{a_{11}^2+a_{21}^2}},\quad s_1=\frac{a_{21}}{\sqrt{a_{11}^2+a_{21}^2}} \quad (2.22)$$

Нехай значення матриці коефіцієнтів та вектора вільних членів будуть мати наступні значення:

$$\begin{array}{ll} a_{1j}^{(1)}=c_1a_{1j}+s_1a_{2j} & j=\overline{1,n}, b_1^{(1)}=c_1b_1+s_1b_2 \\ a_{2j}^{(1)}=-s_1a_{1j}+c_1a_{2j} & j=\overline{2,n}, b_2^{(1)}=-s_1b_1+c_1b_2 \end{array} \quad (2.23)$$

Другий елемент першого рівняння матриці коефіцієнтів буде мати наступне значення під час розкриття значення змінних:

$$a_{21}=-s_1a_{1j}+c_1a_{2j}=-a_{21}k\,a_{11}+a_{11}k\,a_{21}=k(a_{11}a_{21}-a_{11}a_{21})=0$$
$$k=\frac{1}{\sqrt{a_{11}^2+a_{21}^2}}$$

Під час підстановки нових значень матриці коефіцієнтів та вектора вільних членів (2.21) по формулам (2.22) та (2.23), система лінійних алгебраїчних рівнянь буде мати наступний вигляд:

$$\begin{array}{ccccccccc}
a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \dots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)} \\
 & & a_{22}^{(1)}x_2 & + & \dots & + & a_{2n}^{(1)}x_n & = & b_2^{(1)} \\
\dots & & \dots & & & & \dots & & \dots \\
a_{n1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{nn}x_n & = & b_n
\end{array}$$

Далі рівняння системи (2.24) замінюємо новим, але операція буде проводитися вже для першого та третього рівняння:

$$\begin{array}{ccccccccc}
a_{11}^{(2)}x_1 & + & a_{12}^{(2)}x_2 & + & \dots & + & a_{1n}^{(2)}x_n & = & b_1^{(2)} \\
 & & a_{22}^{(1)}x_2 & + & \dots & + & a_{2n}^{(1)}x_n & = & b_2^{(1)} \\
 & & a_{32}^{(1)}x_2 & + & \dots & + & a_{3n}^{(1)}x_n & = & b_2^{(1)} \\
\dots & & \dots & & & & \dots & & \dots \\
a_{n1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{nn}x_n & = & b_n
\end{array} \quad (2.25)$$

Для першого та третього рівнняння (2.25) будуть використані наступні формули:

$$c_2 = \frac{a_{11}^{(1)}}{\sqrt{a_{11}^{(1)2} + a_{31}^2}}, \quad s_2 = \frac{a_{31}}{\sqrt{a_{11}^{(1)2} + a_{31}^2}}$$

$$a_{1j}^{(2)} = c_2 a_{1j}^{(1)} + s_2 a_{3j} \quad j = \overline{1,n}, \quad b_1^{(2)} = c_2 b_1^{(1)} + s_2 b_3$$
$$a_{3j}^{(1)} = -s_2 a_{1j}^{(1)} + c_2 a_{3j} \quad j = \overline{2,n}, \quad b_3^{(1)} = -s_2 b_1^{(1)} + c_2 b_3$$

Після двох ітерацій можна почати перетворювати систему рівнянь для четвертого рівняння і наступних. Після наступних ітерацій ми отримаємо підсистему рівнянь з (2.25) наступного вигляду:

$$\begin{array}{ccccccc}
a_{22}^{(1)}x_2 & + & \dots & + & a_{2n}^{(1)}x_n & = & b_2^{(1)} \\
\dots & & & & \dots & & \dots \\
a_{n2}^{(1)}x_2 & + & \dots & + & a_{nn}^{(1)}x_n & = & b_n^{(1)}
\end{array} \quad (2.26)$$

Тепер залишається лише перетворити рівняння (2.26) за минулим алгоритмом. В результаті ми отримаємо наступну систему рівнянь:

$$\begin{array}{cccccccc}
a_{11}^{(n-1)}x_1 & + & a_{12}^{(n-1)} & \dots & + & a_{1n}^{(n-1)} & = & b_1^{(n-1)} \\
 & & a_{22}^{(n-1)} & \dots & + & a_{2n}^{(n-1)} & = & b_2^{(n-1)} \\
 & & & & & \vdots & & \vdots \\
 & & & & & a_{nn}^{(n-1)}x_n & = & b_n^{(n-1)}
\end{array} \quad (2.27)$$

Після чого ми можемо отримати розв’язки системи лінійних рівнянь з остаточної системи рівнянь (2.27) за допомогою зворотнього методу Гауса.

3 ОПИС АЛГОРИТМІВ

Перелік всіх основних змінних та їхнє призначення наведено в таблиці 3.1.

Таблиця 3.1 — Основні змінні та їхні призначення.

Змінна	Призначення
A	Матриця коефіцієнтів
B	Вектор вільних коефіцієнтів
IsSolvable	Ознака можливості вирішення СЛАР
i	Лічильник циклу
j	Лічильник циклу
k	Лічильник циклу
s	Сума ряду
X	Вектор розв’язків СЛАР

3.1 Загальний алгоритм

1. ПОЧАТОК
2. Зчитати розмірність системи.
3. Зчитати матрицю системи та стовпець вільних членів:
   3.1. Зчитати матрицю коефіцієнтів:
      3.1.1. Цикл проходу по всіх рядках матриці системи (a_i — поточний рядок):
         3.1.1.1. Цикл проходу всіх стовпцях матриці системи (a_{ij} — поточний рядок):

3.1.1.1.1. ЯКЩО поточний елемент матриці – вірно записане число, ТО записати його в відповідну комірку А. ІНАКШЕ видати повідомлення про помилку та перейти до пункту 10.

3.2. Зчитати вектор вільних коефіцієнтів:

3.2.1. Цикл проходу по всіх елементах стовпця вільного членів:

3.2.1.1. Якщо поточний елемент вектора вільних коефіцієнтів – вірно записане число, ТО записати його в відповідну комірку В. ІНАКШЕ видати повідомлення про помилку та перейти до пункту 10.

4. ЯКЩО детермінант матриці A рівен 0 ТО

4.1. вивсети повідомлення про нульовий детермінант мтариці коефіцієнтів.

5. ЯКЩО обраний LUP-метод, ТО обробити дані згідно алгоритму методу Якобі (підрозділ 3.2).
6. ЯКЩО обраний метод Гауса-Зейделя, ТО обробити дані згідно алгоритму методу Якобі (підрозділ 3.3).
7. ЯКЩО обраний метод обертання, ТО обробити дані згідно алгоритму методу Якобі (підрозділ 3.4).
8. ЯКЩО IsSolvable дорівнює правді, ТО:

8.1. ЯКЩО обрана система має дві невідомих, ТО побудувати та вивести графік системи рівнянь.

8.2. Вивести рішення системи з вектора X.

8.3. Записати систему та її рішення у файл.

9. ІНАКШЕ вивсети повідомлення про неможливість розв’язання СЛАР вказаним методом.
10. КІНЕЦЬ

3.2 Алгоритм LUP-методу

1. ПОЧАТОК
2. Отримати розмірність матриці *A* як n.
3. Створити новий вектор *P* розміром n.
4. Заповнення вектора *P* початковими значеннями:

   4.1. ЦИКЛ по всім індексам нової матриці P лічильником i:

   4.1.1. Встановити значення p_i як i.
5. Створити нову матрицю *NA* розмірністю в n рядків та n стовпців.
6. Копіювання матриці *A* до матриці *NA*:

   6.1. ЦИКЛ по всім рядкам нової матриці *NA* лічильником i:

   6.1.1. ЦИКЛ по всім стовпцям нової матриці *NA* лічильником j:

   6.1.1.1. Встановити значення na_{ij} як у елемента a_{ij}.
7. Знайти факторизацію матриці *NA*:

   7.1. Обміняти значення двох рядків матриці *NA* для забезпечення можливості розв'язання у випадку, коли $na_{jj}=0$:

   7.1.1. ЦИКЛ для змінної j від *1* до n:

   7.1.1.1. Знайти рядок з максимальним значенням i, де $|na_{ij}|=|\max na_{kj}|,\ j=\overline{j,n},\ i=\overline{j,n}$:

   7.1.1.1.1. Встановити значення для змінної *MaxColumnValue* як a_{jj}.

   7.1.1.1.2. Встановити значення для змінної *MaxColumnIndex* як j.

   7.1.1.1.3. ЦИКЛ для змінної i від j до n:

   7.1.1.1.3.1. ЯКЩО $|na_{ij}|>|na_{MaxColumnIndex,\,j}|$, ТО

   7.1.1.1.3.2. Встановити значення для змінної *MaxColumnValue* як na_{ij}.

   7.1.1.2. ЯКЩО $MaxColumnValue=0$ є правдою, ТО

   7.1.1.2.1. Встановити змінну *IsSolvable* як хиба.

   7.1.1.2.2. Перейти до пункту 11.

7.1.2. ЦИКЛ для змінної *i* від *1* до *n*:

7.1.2.1. Обміняти елементи матриці *NA* $a_{j,i}$ та $na_{MaxColumnIndex,i}$ місцями.

7.1.3. Обміняти елементи вектора *P* p_j та $p_{MaxColumnIndex}$ місцями.

7.2. Ітерація факторизації матриці *NA*:

7.2.1. ЦИКЛ для змінної *i* від *j* до *n*:

7.2.1.1. Встановити змінну *s* як суму циклу з початковим значенням 0.

7.2.1.2. ЦИКЛ для змінної *k* від *1* до *j - 1*:

7.2.1.2.1. Збільшити значення змінної *s* як суми циклу на $na_{ik} a_{kj}$.

7.2.1.3. Встановити значення елементу матриці a_{ij} як $a_{ij} - s$.

7.2.1.4. ЯКЩО $i > j$, ТО

7.2.1.4.1. Встановити змінну *s* як суму циклу з початковим значенням 0.

7.2.1.4.2. ЦИКЛ для змінної *k* від *0* до *j - 1*:

7.2.1.4.2.1. Збільшити значення змінної *s* як суми циклу на $na_{jk} a_{ki}$.

7.2.1.4.3. Встановити значення елементу матриці a_{ji} як $\frac{a_{ji} - s}{a_{jj}}$.

7.3. Створити нову матрицю *L* розмірністю в *n* рядків та *n* стовпців.

7.4. Створити нову матрицю *U* розмірністю в *n* рядків та *n* стовпців.

7.5. Заповнити матрицю *L* матрицею *NA*:

7.5.1. ЦИКЛ по всім рядкам нової матриці *L* лічильником *j*:

7.5.1.1. ЦИКЛ для змінної *i* від *1* до *j*:

7.5.1.1.1.1. Встановити значення l_{ji} як na_{ji}.

7.6. Заповнити матрицю *L* матрицею *NA*:

7.6.1. ЦИКЛ по всім рядкам нової матриці *U* лічильником *j*:

7.6.1.1. ЦИКЛ для змінної *i* від *j* до *n*:

7.6.1.1.1. ЯКЩО $i > j$, ТО

7.6.1.1.1.1. Встановити значення u_{ji} як a_{ji}.

7.6.1.1.2. ІНАКШЕ

7.6.1.1.2.1. Встановити значення u_{ji} як 1.

8. Обчислити значення вектора *Y*:
   8.1. Створити новий вектор *Y* розміру *n*.
   8.2. ЦИКЛ для змінної *i* від *1* до *n*:
      8.2.1. Встановити змінну *s* як суму циклу з початковим значенням 0.
      8.2.2. ЦИКЛ для змінної *k* від *0* до *i - 1*:
         8.2.2.1. Збільшити значення змінної *s* як суми циклу на $l_{ik} y_k$.
      8.2.3. Встановити значення вектора y_i як $\frac{b_{p_i} - s}{l_{ii}}$.
9. Обчислити значення вектора *X*:
   9.1. Створити новий вектор *Y* розміру *n*.
   9.2. ЦИКЛ для змінної *i* від *n* до *1* донизу:
      9.2.1. Встановити змінну *s* як суму циклу з початковим значенням 0.
      9.2.2. ЦИКЛ для змінної *k* від *i + 1* до *n*:
         9.2.2.1. Збільшити значення змінної *s* як суми циклу на $u_{ik} x_k$.
      9.2.3. Встановити значення розв’язка x_i як $y_i - s$.
10. Встановити змінну *IsSolvable* як правда.
11. КІНЦЕЬ

3.3 Алгоритм методу Гауса-Холецького

1. ПОЧАТОК
2. Задати змінній n значення розміру матриці A.
3. Обчислення розкладу Холецького:

3.1. Створити квадратну матрицю L розміром n.

3.2. Створити квадратну матрицю D розміром n.

3.3. ЦИКЛ для змінної j від 1 до n:

3.3.1. Встановити значення елементу матриці $L_{j,j}$ як 1.

3.3.2. Встановити значення для змінної суми s значення 0.

3.3.3. ЦИКЛ для змінної k від 1 до $j-1$:

3.3.3.1. Збільшити змінну суми s на $L_{j,k}^2 D_{k,k}$.

3.3.4. Встановити значення елементу матриці $D_{j,j}$ як $A_{j,j}-s$.

3.3.5. ЦИКЛ для змінної i від $j+1$ до n:

3.3.5.1. ЯКЩО $D_{j,j}$ дорівнює 0:

3.3.5.1.1. Встановити змінну *IsSolvable* як хиба.

3.3.5.1.2. Перейти до пункту 9.

3.3.5.2. Встановити значення для змінної суми s значення 0.

3.3.5.3. ЦИКЛ для змінної k від 1 до $j-1$:

3.3.5.3.1. Збільшити змінну суми s на $L_{i,k} D_{k,k} L_{j,k}$.

3.3.5.4. Встановити значення елементу матриці $L_{i,j}$ як $\frac{A_{i,j}-s}{D_{j,j}}$.

4. Обчислення стовбця Y:

4.1. Створити стовпець-вектор Y розміром n.

4.2. ЦИКЛ для змінної i від 1 до n:

4.2.1. Встановити значення для змінної суми s значення 0.

4.2.2. ЦИКЛ для змінної k від 1 до $i-1$:

4.2.2.1. Збільшити змінну суми s на $L_{i,j} Y_j$.

4.2.3. Встановити значення елементу вектора Y_i як B_i-s.

5. Обчислення стовбця Z:

5.1. Створити стовпець-вектор Z розміром n.

5.2. ЦИКЛ для змінної i від 1 до n:

5.2.1. ЯКЩО $D_{j,j}$ дорівнює 0:

5.2.1.1. Встановити змінну *IsSolvable* як хиба.

5.2.1.2. Перейти до пункту 9.

5.2.2. Встановити значення елементу вектора Z_i як $\frac{Y_i}{D_{i,i}}$.

6. Обчислення стовбця X:

6.1. Створити стовпець-вектор X розміром n.

6.2. ЦИКЛ для змінної i від n до 1 змінюючи змінну на -1:

6.2.1. Встановити значення для змінної суми s значення 0.

6.2.2. ЦИКЛ для змінної k від $i+1$ до n:

6.2.2.1. Збільшити змінну суми s на $L_{j,i} X_k$.

6.2.3. Встановити значення елементу вектора X_i як $Z_i - s$.

7. Перевірити коректність результиту виконання методу:

7.1. Перевірити, чи є результат алгоритму коректним:

7.1.1. Створити новий вектор стовпець $NewB$ розміром n.

7.1.2. ЦИКЛ для змінної y від 1 до n:

7.1.2.1. ЦИКЛ для змінної x від 1 до n:

7.1.2.1.1. Встановити значення елементу матриці $NewB_y$ як $A_{y,x} X_x$

.

7.1.3. ЦИКЛ для змінної y від 1 до n:

7.1.3.1. ЯКЩО $|NewB_y - New_x|$ не дорівнює 0:

7.1.3.1.1. Встановити змінну *IsSolvable* як хиба.

7.1.3.1.2. Перейти до пункту 9.

7.2. Перевірити, чи є результат алгоритму лише одним з рішенням СЛАР:

7.2.1. ЯКЩО n менше 2:

7.2.1.1. Перейти до пункту 8.

7.2.2. Встановити змінній *IsAmbigious* значення хиби.

7.2.3. ЦИКЛ для змінної i від 1 до n:

7.2.3.1. ЯКЩО B_i не дорівнює 0 або X_i не дорівнює 0:

7.2.3.1.1. Встановити змінній *IsAmbigious* значення правди.

7.2.4. ЯКЩО значення *IsAmbigious* дорівнює хибі.

8. Встановити змінну *IsSolvable* як правда.
9. КІНЕЦЬ

3.4 Загальний методу обертання

1. ПОЧАТОК
2. Отримати розмірність матриці *A* як *n*.
3. Створити нову матрицю *NA* розмірністю в *n* рядків та *n + 1* стовпців.
4. Заповити нову матрицю *NA* значеннями з матриці *A* та *B*.

4.1. ЦИКЛ по всім рядкам нової матриці лічильником *i*:

4.1.1. ЦИКЛ по всім стовпцям нової матриці лічильником *j*:

4.1.1.1. ЯКЩО вираз $j <= n$ є правдивим, ТО

4.1.1.1.1. Встановити значення na_{ij} як у елемента a_{ij}.

4.1.1.2. ІНАКШЕ

4.1.1.2.1. Встановити значення na_{ij} як у елемента b_i.

5. Обчислити розкладання нової матриці NA:

5.1. ЦИКЛ для змінної *i* від *1* до *n - 1*:

5.1.1. ЦИКЛ для змінної *j* від *i + 1* до *n*:

5.1.1.1. Встановити значення для змінної b як na_{ji}.

5.1.1.2. Встановити значення для змінної a як na_{ii}.

5.1.1.3. ЯКЩО значення a^2+b^2 не є позитивним, ТО

5.1.1.3.1. Встановити змінну *IsSolvable* як хиба.

5.1.1.3.2. Перейти до пункту 8.

5.1.1.4. Встановити значення для змінної c як $\frac{a}{\sqrt{(a^2+b^2)}}$.

5.1.1.5. Встановити значення для змінної b як $\frac{b}{\sqrt{(a^2+b^2)}}$.

5.1.1.6. ЦИКЛ для змінної *k* від *i* до *n + 1*:

5.1.1.6.1. Встановити значення для змінної t як na_{ik}.

5.1.1.6.2. Встановити значення елементу матриці na_{ik} як $ca_{ik}+sa_{jk}$.

5.1.1.6.3. Встановити значення елементу матриці na_{jk} як $-st+ca_{jk}$.

6. Обчислити значення вектора X довжиною *n*:

6.1. ЦИКЛ для змінної *i* від *n* до *1* донизу:

6.1.1. Встановити змінну s сумми цикла як 0.

6.1.2. ЦИКЛ для змінної *j* від *i + 1* до *n*:

6.1.2.1. Збільшити значення *s* змінної суми цикла на $a_{ij} x_j$.

6.1.2.2. ЯКЩО значення a_{ii} є нулем, ТО

6.1.2.2.1. Встановити змінну *IsSolvable* як хиба.

6.1.2.2.2. Перейти до пункту 8.

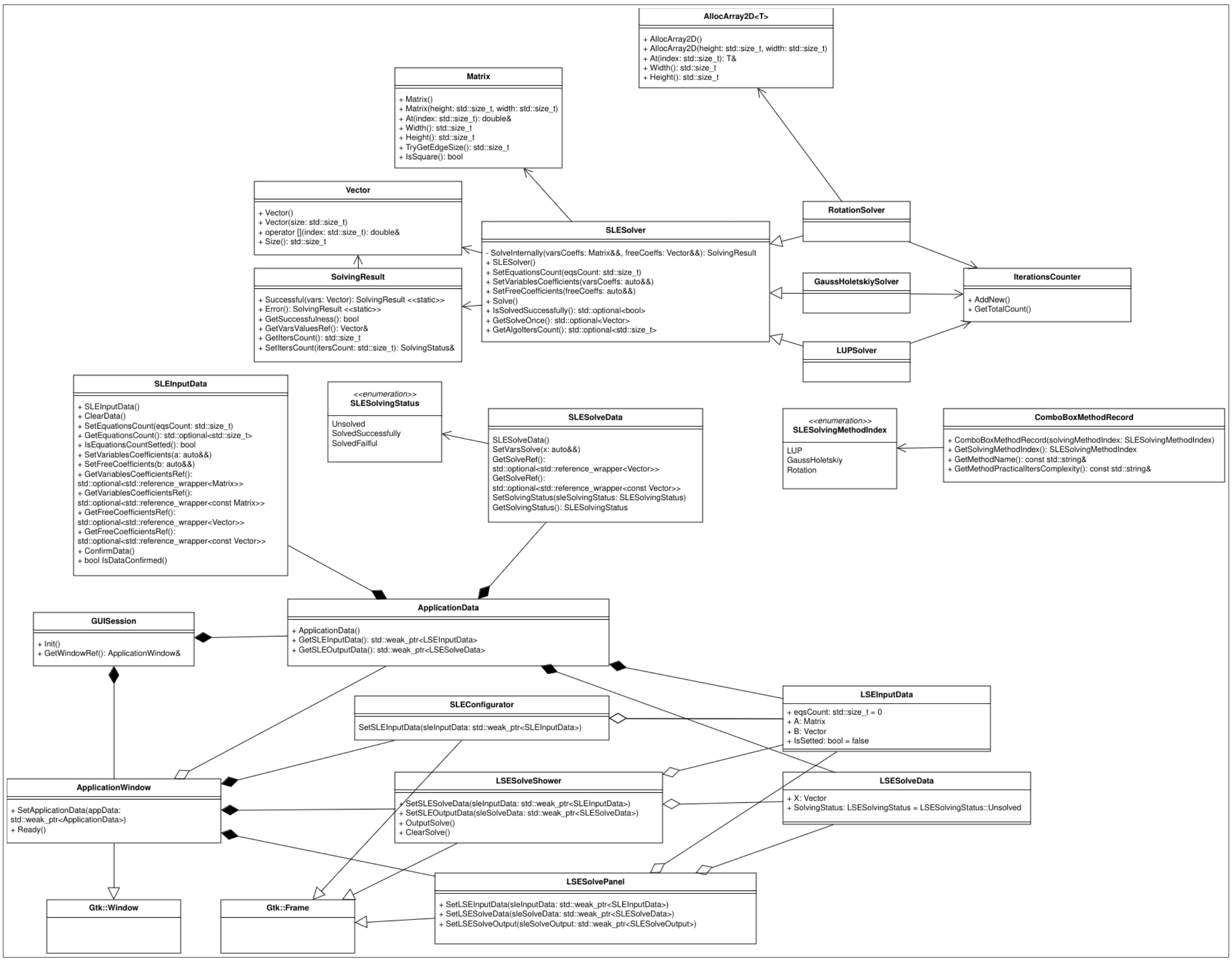
6.1.2.3. Встановити значення розв’язку x_i як $\frac{a_{in}-s}{a_{ii}}$.

7. Встановити змінну *IsSolvable* як правда.

8. КІНЕЦЬ

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Діаграма класів програмного забезпечення



4.2 Опис методів частин програмного забезпечення

4.2.1 Користувацькі методи

У таблиці 4.1 наведено більшість методів, які були спроектовані та реалізовані самостійно під час створення даного програмного забезпечення.

Таблиця 4.1 — Користувацькі методи

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
1.	Vector	Vector	Створити новий пустий вектор			Vector.hpp
2.	Vector	Vector	Створити новий вектор довжини size	size		Vector.hpp
3.	Vector	operator[]	Отримати посилання на елемент вектора	index		Vector.hpp
4.	Vector	Size	Отримати довжину вектора			
5.	Matrix	Matrix	Створити нову пусту матрицю			Matrix.hpp
6.	Matrix	Matrix	Створити матрицю розмірністю width на height	height, width		Matrix.hpp
7.	Matrix	At	Отримати	y, x		Matrix.hp

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
			посилання на елемент матриці			p
8.	Matrix	Width	Отримати кількість стовпців матриці			Matrix.hpp
9.	Matrix	Height	Отримати кількість рядків матриці			Matrix.hpp
10.	Matrix	TryGetEdgeSize	Спробувати отримати розмірність квадратної матриці по її ширині			Matrix.hpp
11.	Matrix	IsSquare	Чи є матриця квадратною			Matrix.hpp
12.	AllocArray2D<T>	AllocArray2D	Створини пустий новий двовимірний масив			AllocArray2D.inc.hpp

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
13.	AllocArray2D<T>	AllocArray2D	Створини новий двовимірний масив шириною width та висотою height	height, width		AllocArray2D.inc.hpp
14.	AllocArray2D<T>	At	Отримати посилання на елемент двовимірного масиву по ширині x та висоті y	y, x		AllocArray2D.inc.hpp
15.	AllocArray2D<T>	Width	Отримати ширину двовимірний масиву			AllocArray2D.inc.hpp
16.	AllocArray2D<T>	Height	Отримати висоту двовимірний масиву			AllocArray2D.inc.hpp

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
17.	SolvingResult	Successful	Отримати об’єкт типу класу цього методу. Об’єкт репрезентує, що статус значення є успішним при обробці			LSESolver.hpp
18.	SolvingResult	Error	Отримати об’єкт типу класу цього методу. Об’єкт репрезентує, що статус значення не є успішним при обробці			LSESolver.hpp
19.	SolvingResult	GetSuccessfulness	Отримати статус, чи є результат обрахунків			

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
			успішним			
20.	SolvingResult	GetVarsValuesRef	Отримати посилання на значення невідомих змінних СЛАР			
21.	SolvingResult	GetItersCount	Отримати кількість ітерацій алгоритму			
22.	SolvingResult	SetItersCount	Встановити кількість ітерацій при виконанні алгоритмів та вернути оригінальний об’єкт	itersCount		LSESolver.hpp
23.	SLESolver	LSESolver	Конструктор стану об’єкта цього типу			LSESolver.hpp
24.	SLESolver	~LSESolve	Віртуальний			LSESolve

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
		r	деструктор для класів-нащадків			r.hpp
25.	SLESolver	SetEquationsCount	Встановити кількість лінійних рівнянь єдиножди	eqsCount		LSESolver.hpp
26.	SLESolver	SetVariablesCoefficients	Встановити матрицю коефіцієнтів	varsCoeffs		LSESolver.hpp
27.	SLESolver	SetFreeCoefficients	Встановити вектор вільних членів	freeCoeffs		LSESolver.hpp
28.	SLESolver	Solve	Почати процес розв’язку встановленої СЛАР			LSESolver.hpp
29.	SLESolver	IsSolvedSuccessfully	Отримати статус виконання			LSESolver.hpp

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
			алгоритму розв'язку			
30.	SLESolver	GetSolveOnce	Отримати розв'язок встановленої СЛАР єдиножди			LSESolver.hpp
31.	SLESolver	GetAlgoItersCount	Отримати кількість ітерацій при викоанні алогритму			LSESolver.hpp
32.	SLEInputData	ClearData	Скинути стан об'єку даних СЛАР			
33.	SLEInputData	SetEquationsCount	Встановити кількість рівнянь	eqsCount		
34.	SLEInputData	IsEquationsCountSetted	Перевірити, чи встановлено кількість рівнянь			

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
35.	SLEInputData	SetVariablesCoefficients	Встановити матрицю коефіцієнтів	a		
36.	SLEInputData	SetFreeCoefficients	Встановити вільні коефіцієнти	b		
37.	SLEInputData	GetVariablesCoefficients	Отримати посилання на матрицю коефіцієнтів			
38.	SLEInputData	GetFreeCoefficients	Отримати посилання на вільні коефіцієнти			
39.	SLEInputData	ConfirmData	Підтвердити дані СЛАР			
40.	SLEInputData	IsDataConfirmed	Перевірити, чи є слар підтверджено ю			
41.	SLESolveData	SetVarsSolve	Встановити розв’язок СЛАР	x		

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
42.	SLESolveData	GetSolveRef	Отримтаи посилання на розв'язок СЛАР			
43.	SLESolveData	SetSolvingStatus	Встановити статус вирішення СЛАР	sleSolvingStatus		
44.	SLESolveData	GetSolvingStatus	Отримати статус вирішення СЛАР			
45.	GUISession	GUISession	Створити об'єкт типу сесії графічного інтерфейсу			GUI.hpp
46.	GUISession	Init	Ініціалізувати вікно графічної сесії аргументами командного			GUI.hpp

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
			рядка			
47.	GUISession	GetWindow	Отримати посилання на вікно програми			GUI.hpp
48.	ApplicationData	GetSLEInputData	Отримати посилання на вхідні дані СЛАР			GUI.hpp
49.	ApplicationData	GetSLEOutputData	Отримати посилання на дані розв’язку СЛАР			GUI.hpp
50.	ApplicationWindow	SetApplicationData	Встановити посилання на об’єкт, що зберігає дані програми	appData		GUI.hpp
51.	ApplicationWindow	Ready	Встановити стан вікна як готового для опрацювання			GUI.hpp

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
52.	SLEConfigurator	SetLSEInputData	Встановити місце для запису конфігурації СЛАР	sleInputData		GUI.hpp
53.	SLESolveShower	SetSLEOutputData	Встановити місце для зчитання даних СЛАР	sleSolveData		GUI.hpp
54.	SLESolveShower	SetSLEInputData	Встановити місце для зчитання даних про розв'язок СЛАР	sleInputData		GUI.hpp
55.	SLESolveShower	OutputSolve	Показати розв'язки СЛАР			GUI.hpp
56.	SLESolveShower	ClearSolve	Приховати розв'язки СЛАР			GUI.hpp
57.	SLESolvePanel	SetSLEInputData	Встановити місце для	sleInputData		GUI.hpp

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл
			зчитання даних СЛАР			
58.	SLESolvePanel	SetSLESolveData	Встановити місце для запису даних про розв'язок СЛАР	sleSolveData		GUI.hpp
59.	SLESolvePanel	SetSLESolveOutput	Встановити віджет для відображення розв'язків СЛАР	sleSolveOutput		GUI.hpp

4.2.2 Стандартні методи

У таблиці 4.2 наведено більшість вбудованих, стандартизованих та зовнішніх методів з бібліотек, які були використані під час створення даного програмного забезпечення.

Таблиця 4.2 — Стандартні методи

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовочний файл

1.	std::vector<T>	vector	Конструктор порожнього динамічного масиву			vector
2.	std::vector<T>	vector	Конструктор динамічного масиву з size стандартних елементів	size		vector
3.	std::vector<T>	operator[]	Отримати посилання на елемент	index	T&	vector
4.	std::vector<T>	size	Отримати довжину		std::size_t	vector
5.	std::istringstream	istringstream	Новий рядковий вхідних потік з вхідних рядком __str	__str		sstream
6.	std::istringstream	imbue	Встановити локаль	__loc	std::locale	sstream
7.	std::istringstream	operator>>	Вивести форматовані дані з потоку до змінної	__f	std::istream	sstream
8.	std::istringstream	fail	Перевірити, чи потік		bool	sstream

			зазнав помилки			
9.	std::istringstream	eof	Перевірити, чи є потік у кінці файла		bool	sstream
10.	std::unique_ptr<T>	unique_ptr	Конструктор нового унікального розумного покажчика			memory
11.	std::unique_ptr<T>	reset	Перевстановити покажчик	__p		memory
12.	std::unique_ptr<T>	operator *	Отримати посилання на об’єкт покажчика		T&	memory
13.	std::shared_ptr<T>	shared_ptr	Конструктор нового спільного покажчика			memory
14.	std::shared_ptr<T>	operator *	Отримати посилання на об’єкт покажчика		T&	memory
15.	std::weak_ptr<T>	weak_ptr	Конструктор нового слабкого			memory

			покажчика			
16.	std::weak_ptr<T>	lock	Отримати новий спільний покажчик, який посилається так же, як і слабкий покажчик		std::shared_ptr<T>	memory
17.	std::optional<T>	optional	Конструктор нового опціонального значення, яке буде сконструйоване пустим конструктором			optional
18.	std::optional<T>	optional	Конструктор нового опціонального значення, яке буде тримати значення об’єкта __t	__t		optional

19.	std::optional <T>	optional	Конструктор нового опціонального значення, яке не буде мати об’єкта, буде зберігати статус відсутності об’єкта	<<немає назви>>		optional
20.	std::optional <T>	value	Отрмати посилання на об’єкт опціонального об’єкта		T&	optional
21.	std::optional <T>	has_value	Перевірити, чи має опціональний об’єкт об’єкт		bool	optional
22.	Gtk::Window	Window	Конструктор вікна			gtkmm.h
23.	Gtk::Window	set_title	Встновити підпис до вікна	title		gtkmm.h
24.	Gtk::Window	set_default_size	Встановити мінімальний розмір	width, height		gtkmm.h

25.	Gtk::Window	set_resizable	Перемкнути можливість змінювати розмір вікна	resizable		gtkmm.h
26.	Gtk::Window	set_border_width	Встановити внутрішній відступ від рамки всіх внутрішніх віджетів вікна	border_width		gtkmm.h
27.	Gtk::Window	set_sensitive	Перемкнути можливість взаємодії з внутрішніми елементами вікна	sensitive		gtkmm.h
28.	Gtk::Frame	Frame	Конструктор іменованої рамки			gtkmm.h
29.	Gtk::Frame	set_label	Встановити видиму назву рамки	label		gtkmm.h
30.	Gtk::Frame	set_size_request	Встановити мінімальний розмір	width, height		gtkmm.h
31.	Gtk::Frame	add	Додати єдиний	widget		gtkmm.h

			елемент як дитячий віджет			
32.	Gtk::Frame	set_border_width	Встановити внутрішній відступ від рамки всіх внутрішніх віджетів	border_width		gtkmm.h
33.	Gtk::Frame	show	Показати даний віджет			gtkmm.h
34.	Gtk::Frame	show_all_children	Показати всі віджети цього віджета			gtkmm.h
35.	Gtk::Fixed	Fixed	Конструктор поля для встановки віджетів по координатам			gtkmm.h
36.	Gtk::Fixed	put	Додати віджет по вказаним координатам відносно даного віджета	__v		gtkmm.h
37.	Gtk::Fixed	show	Показати			gtkmm.h

			даний выджет			
38.	Gtk::Box	Box	Конструктор поля для встановки віджетів один за одним			gtkmm.h
39.	Gtk::Box	set_orientation	Встновити орієнтацію списка віджетів	orientation		gtkmm.h
40.	Gtk::Box	set_spacing	Встановити відстань між віджетами	spacing		gtkmm.h
41.	Gtk::Box	pack_start	Додати до списку новий віджет	column		gtkmm.h
42.	Gtk::Box	show	Показати даний віджет	child		gtkmm.h
43.	Gtk::Box	show_all_children	Показати всі віджети цього віджета			gtkmm.h
44.	Gtk::Grid	Grid	Конструктор сітки для відображення віджетів			gtkmm.h
45.	Gtk::Grid	attach	Додати новий віджет до	child, left, top, width,		gtkmm.h

			сітки по координатам з вказаним розміром в клітинках	height		
46.	Gtk::Grid	remove_column	Прибрати один рядок	position		gtkmm.h
47.	Gtk::Grid	hide	Сховати даний віджет			gtkmm.h
48.	Gtk::Grid	show	Показати даний віджет			gtkmm.h
49.	Gtk::Grid	show_all_children	Показати всі віджети цього віджета			gtkmm.h
50.	Gtk::Label	Label	Конструктор віджета відображення тексту			gtkmm.h
51.	Gtk::Label	Label	Конструктор віджета відображення тексту з встановленим текстом	text		gtkmm.h
52.	Gtk::Label	set_text	Встановити текст для вдображення	str		gtkmm.h

53.	Gtk::Label	set_width_chars	Встановити розмір віджета у символах	n_chars		gtkmm.h
54.	Gtk::Label	set_size_request	Встановити мінімальний розмір	width, height		gtkmm.h
55.	Gtk::Entry	Entry	Конструктор поля ввода			gtkmm.h
56.	Gtk::Entry	set_placeholder_text	Встановити текст зо замовчування	text		gtkmm.h
57.	Gtk::Entry	get_text	Отримати введений текст		Glib::ustring	gtkmm.h
58.	Gtk::Entry	set_width_chars	Встановити ширину поля у символах	n_chars		gtkmm.h
59.	Gtk::Entry	set_max_length	Встановити максимальну дожину введеного тексту	max		gtkmm.h
60.	Gtk::Entry	set_size_request	Встановити мінімальний розмір	width, height		gtkmm.h

61.	Gtk::Button	Button	Конструктор кнопки			gtkmm.h
62.	Gtk::Button	Button	Конструктор кнопки з встановленим текстом	text		gtkmm.h
63.	Gtk::Button	signal_clicked	Встановити логіку для події на натискання	slot_	Glib::SignalProxy<void>	gtkmm.h
64.	Gtk::ComboBox	ComboBox	Конструктор поля зі списком			gtkmm.h
65.	Gtk::ComboBox	append	Додати новий пункт до списку	id, text		gtkmm.h
66.	Gtk::DrawingArea	DrawingArea	Конструктор холста для програмного малювання			gtkmm.h
67.	Gtk::DrawingArea	set_size_request	Встановити мінімальний розмір	width, height		gtkmm.h
68.	Gtk::DrawingArea	signal_draw	Встановити логіку для події на малювання	slot_	Glib::SignalProxy<void>	gtkmm.h

69.	Gtk::DrawingArea	queue_draw	Принудово відправити об'єкт до черги для відображення			gtkmm.h
70.	Gtk::Allocation	Allocation	Конструктор об'єкта, що відображає геометричні параметри віджета			gtkmm.h
71.	Gtk::Allocation	get_width	Отримати реальну ширину		int	gtkmm.h
72.	Gtk::Allocation	get_height	Отримати реальну висоту		int	gtkmm.h
73.	Cairo::Context	Context	Конструктор об'єкта, який є інтерфейсом для малювання та отримання даних Gtk::DrawingArea			gtkmm.h

74.	Cairo::Context	get_allocation	Отримати об'єкт типу Gtk::Allocation		Gtk::Allocation	gtkmm.h
75.	Cairo::Context	set_source_rgb	Встановити колір для малювання	red, green, blue		gtkmm.h
76.	Cairo::Context	rectangle	Намалювати прямокутник	x, y, iwdth, height		gtkmm.h
77.	Cairo::Context	fill	Заповнити весь холст один коліром			gtkmm.h
78.	Cairo::Context	stroke	Очистити буфер координат			gtkmm.h
79.	Cairo::Context	move_to	Додати нову координату до буфера ліній	x, y		gtkmm.h
80.	Cairo::Context	line_to	Намалювати лінію з останньої встановленої координати	x, y		gtkmm.h
81.	Cairo::Context	set_font_size	Встановити розмір шрифта	size		gtkmm.h

82.	Cairo::Context	show_text	Намалювати текст по раніше встановленим параметрам	utf8		gtkmm.h

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 План тестування

Даний розділ посв’ячений тестуванню програмного забезпечення на наявність помилок, недоліків та інших відхилень від нормальної та коректної роботи програмного забезпечення.

У таблиці 5.1-5.8 показано тестування програмного забезпечення на різних етапах його виконання та його поведінку при різних маніпуляціях зцим програмним додатком.

У списку нижче показано короткий зміст та весь план тестування цієї програми:

а) Тестування правильності введених значень.
   1) Тестування при введенні некоректних символів.
   2) Тестування при введенні замалих та завеликих значень.
б) Тестування коректної роботи при введені систем, що не мають коренів.
   1) Тестування роботи програми при нульовому значенні визначника.
в) Тестування коректності роботи методів 1, 2, 3 з довільними коефіцієнтами:
   1) Перевірка коректності роботи методу 1.
   2) Перевірка коректності роботи методу 2.
   3) Перевірка коректності роботи методу 3.
г) Тестування побудови графіків.

5.2 Приклади тестування

Таблиця 5.1 — Приклад роботи програми при введенні некоректних символів.

Мета тесту	Перевірити можливість введення некоректних даних
Початковий стан програми	Відкрите вікно програми та кількість рівнянь (3) встановлена
Вхідні дані	1 2 3 8; 4 c 6 5; 7 8 9 100
Схема проведення тесту	Поелементне заповнення матриці коефіцієнтів та вектора вільних членів
Очікуваний результат	Повідомлення про помилку формату даних
Стан програми після проведення випробувань	Видано помилку «Введіть дійсне число»

Таблиця 5.2 — Приклад роботи програми при введенні замалих та завеликих значень.

Мета тесту	Перевірити можливість введення занадто великих або замалих чисел
Початковий стан програми	Відкрите вікно програми та кількість рівнянь (2) встановлена
Вхідні дані	100 300 990; -670 480 30000
Схема проведення тесту	Поелементне заповнення матриці коефіцієнтів та вектора вільних членів
Очікуваний результат	Повідомлення про помилку діапазону значень даних
Стан програми після проведення випробувань	Видано помилку «Комірка B[рівняння №2] не є в діапазоні [-10'000; 10'000]»

Таблиця 5.3 — Приклад роботи програми при нульовому значенні визначника.

Мета тесту	Перевірити можливість розв’язку СЛАР з нульовим визначником
Початковий стан програми	Відкрите вікно програми та кількість рівнянь (3) встановлена, матриця коефіцієнтів та вектор вільних членів встановлені
Вхідні дані	1 3 2 5; 4 8 9 9; 0 0 0 1
Схема проведення тесту	Поелементне заповнення матриці коефіцієнтів та вектора вільних членів
Очікуваний результат	Повідомлення про помилку вирішення СЛАР
Стан програми після проведення випробувань	Видано помилку «Детермінант матриці коеф. рівен 0»

Таблиця 5.4 — Приклад перевірки коректності роботи методу 1.

Мета тесту	Перевірити коректність роботи методу 1 на довільних даних
Початковий стан програми	Відкрите вікно програми та кількість рівнянь (3) встановлена, матриця коефіцієнтів та вектор вільних членів встановлені, метод вирішення встановлений
Вхідні дані	146 136 102 3602; 136 155 100 3525; 102 100 114 3258
Схема проведення тесту	Поелементне заповнення матриці коефіцієнтів та вектора вільних членів
Очікуваний результат	Повідомлення про розв’язок «X1: 10; X2: 3; X3: 17»
Стан програми після проведення	Видано розв’язок «X1: 10; X2: 3; X3:

випробувань	17»

Таблиця 5.5 — перевірки коректності роботи методу 2.

Мета тесту	Перевірити коректність роботи методу 2 на довільних даних
Початковий стан програми	Відкрите вікно програми та кількість рівнянь (3) встановлена, матриця коефіцієнтів та вектор вільних членів встановлені, метод вирішення встановлений
Вхідні дані	
Схема проведення тесту	Поелементне заповнення матриці коефіцієнтів та вектора вільних членів
Очікуваний результат	Повідомлення про розв’язок СЛАР «X1: 10; X2: 3; X3: 17»
Стан програми після проведення випробувань	Видано розв’язок «X1: 9.999999999999998; X2: 3.0000000000000027; X3: 16.999999999999996»

Таблиця 5.6 — перевірки коректності роботи методу 3.

Мета тесту	Перевірити коректність роботи методу 3 на довільних даних
Початковий стан програми	Відкрите вікно програми та кількість рівнянь (3) встановлена, матриця коефіцієнтів та вектор вільних членів встановлені, метод вирішення встановлений
Вхідні дані	8 7 5 138; 9 3 2 181; 6 4 1 224

Схема проведення тесту	Поелементне заповнення матриці коефіцієнтів та вектора вільних членів
Очікуваний результат	Повідомлення про розв'язок «X1: 10; X2: 3; X3: 17»
Стан програми після проведення випробувань	Видано розв'язок «X1: 9.999999999999996; X2: 2.9999999999999996; X3: 17.000000000000004»

Таблиця 5.7 — Приклад роботи графіку системи рівнянь.

Мета тесту	Перевірити коректність виведення графіку системи рівнянь
Початковий стан програми	Відкрите вікно програми та кількість рівнянь (3) встановлена, матриця коефіцієнтів та вектор вільних членів встановлені, метод вирішення встановлений, СЛАР була вирішена успішно
Вхідні дані	1 2 5; 3 4 6 LUP-метод
Схема проведення тесту	Поелементне заповнення матриці коефіцієнтів та вектора вільних членів, натиснення на кнопки вирішення СЛАР
Очікуваний результат	Графік системи рівнять, де пересічення прямих знаходиться в точці «$M(-4, 4.5)$»
Стан програми після проведення	Пересічення прямих у точці «

випробувань	$M\left(-4.000\mathrm{e}{+}00, 4.500\mathrm{e}{+}00\right)$»

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1 Робота з програмою

Після запуску виконавчого файлу з розширенням *.exe, відкривається головне вікно програми:

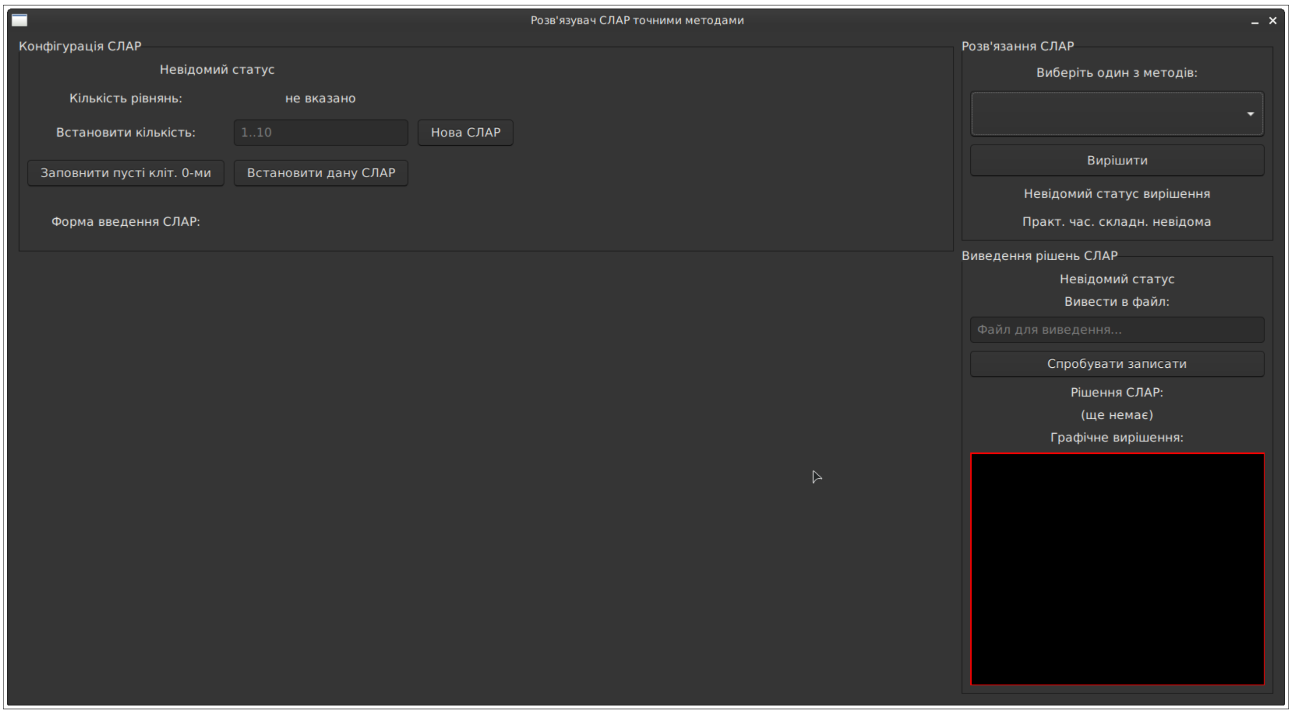


Рисунок 6.1 — Головне вікно програми

Далі треба встановити розмірність системи лінійних алгебраїчних рівнянь (далі — СЛАР) біля таблички «Встановити кількість», потім натиснути на клавішу «Нова СЛАР» щоб встановити нову СЛАР, яка буде оброблятися програмою (рисунок 6.2).

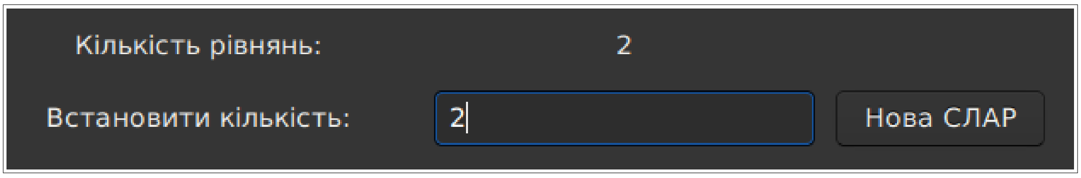


Рисунок 6.2 — Вибір розміру системи

Після чого треба встановити всі коефіцієнти матриці та вільні коефіцієнти до всіх комірок форми введення (рисунок 6.3). На рисунку 6.4 зображено форма введення СЛАР, яка повністю заповнена.

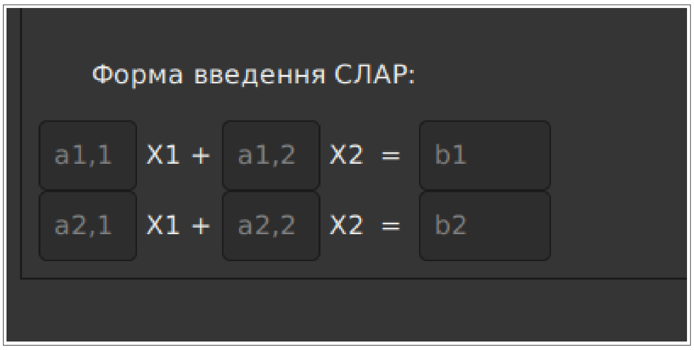


Рисунок 6.3 — Нова форма введення СЛАР

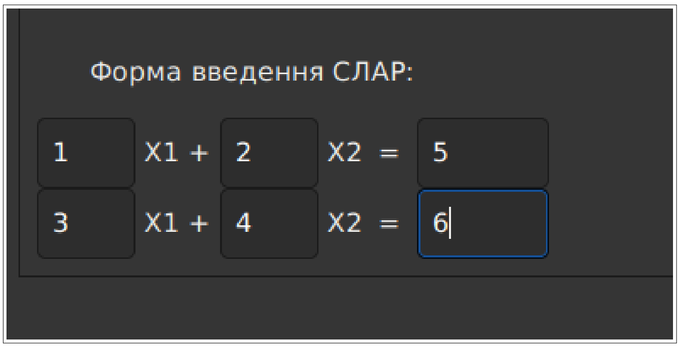


Рисунок 6.4 — Заповнена форма введення СЛАР

Потім треба натиснути на клавішу «Встановити дану СЛАР», що ви згодні встановити дану введену СЛАР до програми для подальшої обробки програмою (рисунок 6.5).

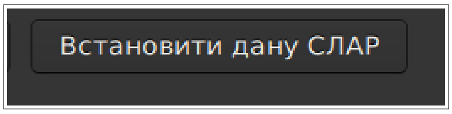


Рисунок 6.5 — Встановлення СЛАР для обробки програмою

Якщо значення комірок форми введення СЛАР не є числами, то буде виведена помилка про немождивість підтвердити дану СЛАР (рисунок 6.6). Якщо СЛАР введена правильно користувачем, то буде відображено час підтвердження СЛАР (рисунок 6.7).

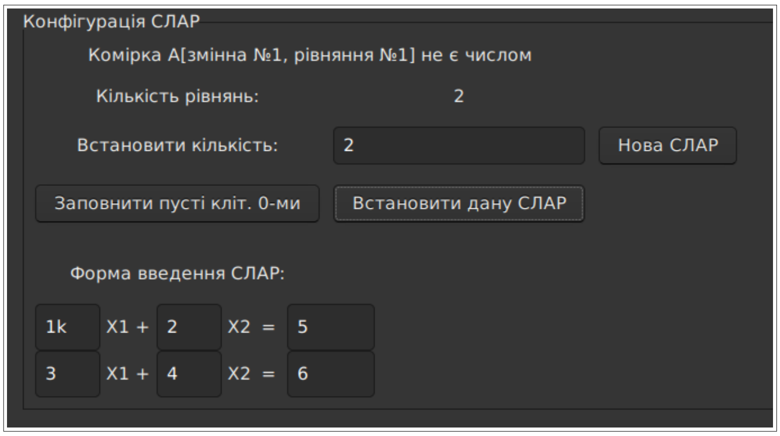


Рисунок 6.6 — Помилка підтвердження СЛАР

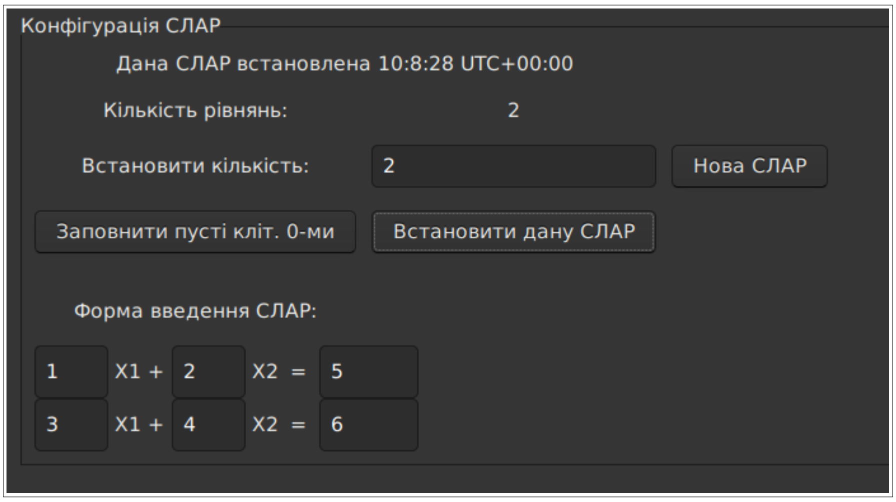


Рисунок 6.7 — Час затвердження СЛАР

Далі користувач має вказати один з методів розв'язання введеної СЛАР до програми за допомогою вибору одного з методів у списку (рисунок 6.8) у підменю «Розв'язання СЛАР». Також можна побачити практичну складність обраного методу.

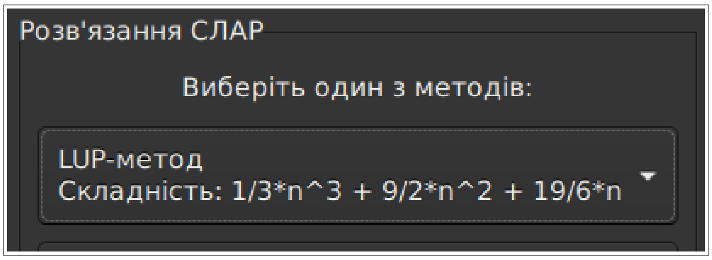


Рисунок 6.8 — Список вибору методу розв'язання СЛАР

Потім треба натиснути на клавішу «Вирішити» (рисунок 6.9) для запущення процесу розв'язання введеної СЛАР користувачем. Якщо детермінант матриці коефіцієнтів СЛАР є нульовим, то буде показана помилка про цю властивість матриці коефіцієнтів (рисунок 6.10).

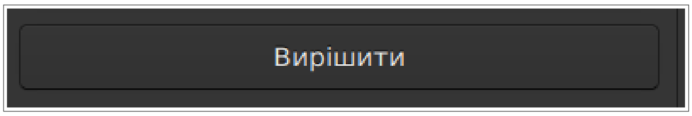


Рисунок 6.9 — Кнопка для початку процеси вирішення СЛАР

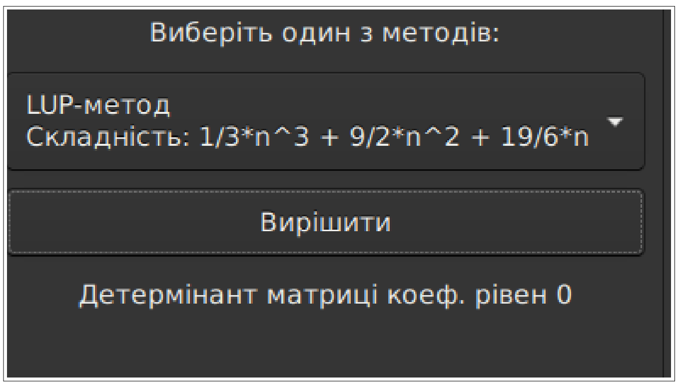


Рисунок 6.10 — Помилка про нульовий детермінант матриці коефіцієнтів

Якщо обраний метод не може вирішити встановлену СЛАР, то буде виведено повідомлення про цю подію (рисунок 6.11).

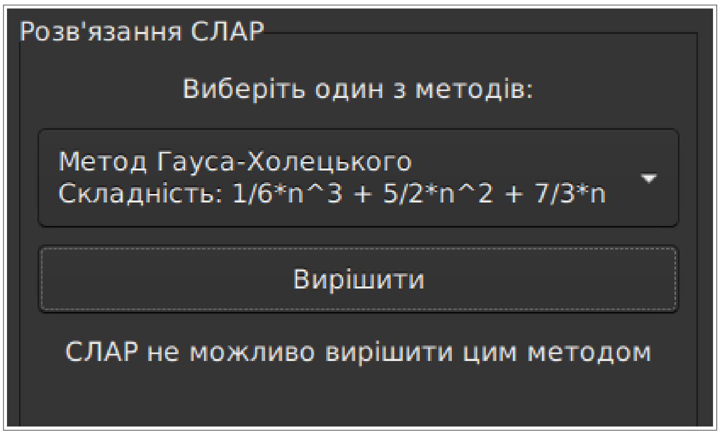


Рисунок 6.11 — Повідомлення про неможливість вирішити СЛАР обраним методом

Якщо встановлена СЛАР для обробки була вирішена успішно, то буде виведена таблиця значень невідомих коефіцієнтів цієї СЛАР (рисунок 6.13). Також буде виведена практична складність та час вирішення СЛАР обраним методом (рисунок 6.12).

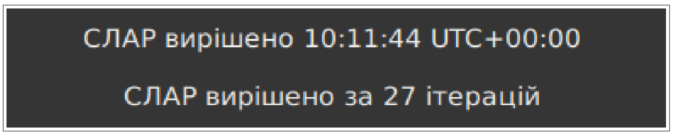


Рисунок 6.12 — Практична складність методу та час вирішення СЛАР

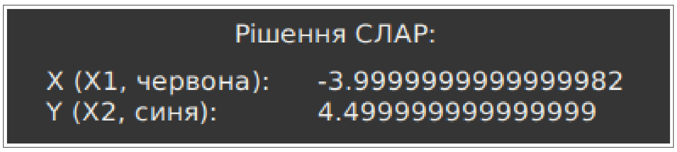


Рисунок 6.13 — Табличний метод відображення розв’язків СЛАР

Якщо кількість рівнянь встановленої СЛАР рівна 2, то буде виведено її графічний розв’язок у вікні програми (рисунок 6.14).

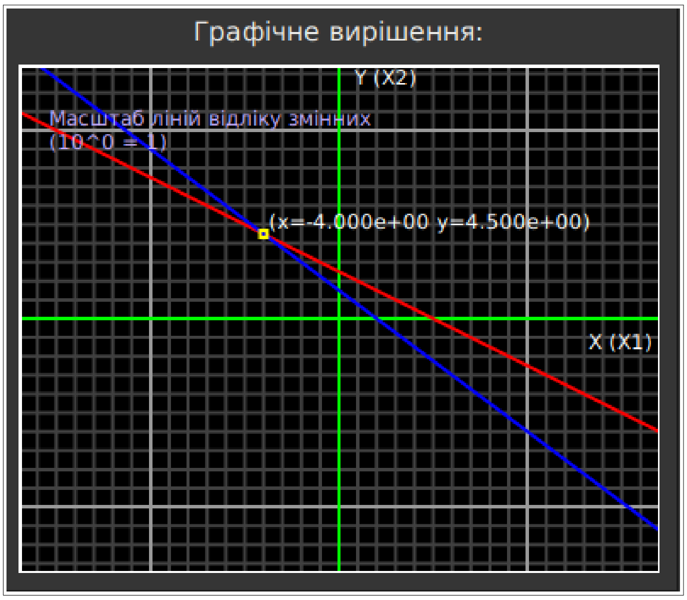


Рисунок 6.14 — Графічне вирішення СЛАР

Також є можливість вивести розв’язок СЛАР до текстового файлу, вказавши його ім’я та натиснувши на кнопку «Спробувати записати» для запущення спроби запису розв’язку до текстового файлу (рисунок 6.15).

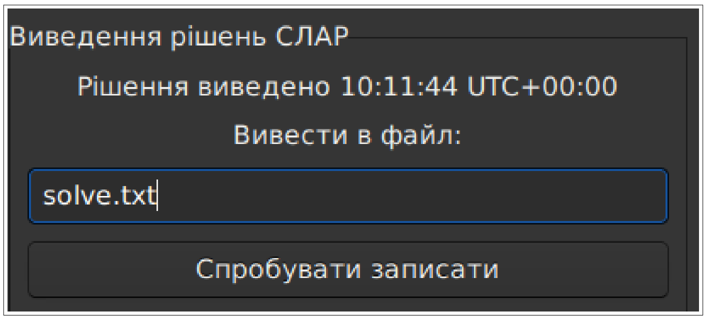


Рисунок 6.15 — Виведення розв’язку СЛАР до текстового файлу

Якщо запис до текстового файлу є успішним, то його змістом буде розв’язок встановленої СЛАР вказаним методом вирішення (рисунок 6.16).

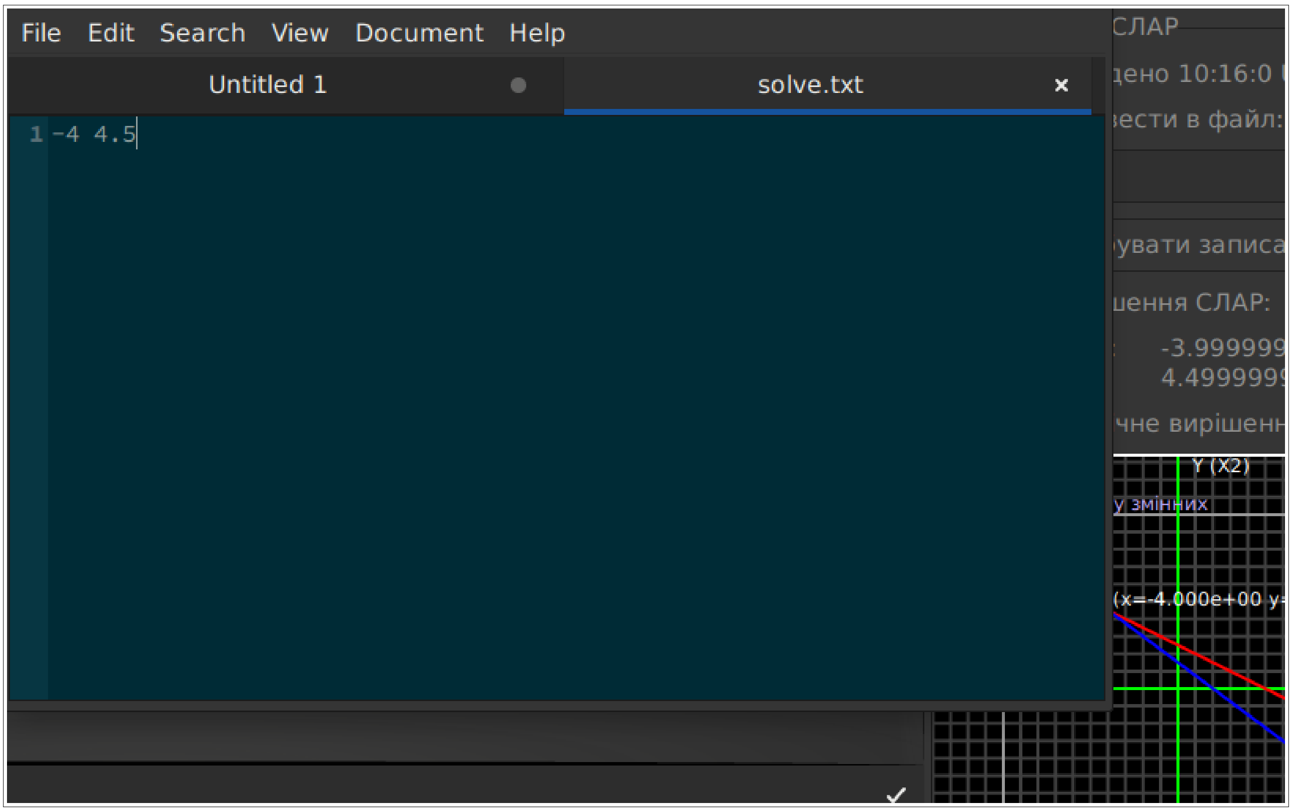


Рисунок 6.16 — Зміст файлу з розв’язком СЛАР

6.2 Формат вхідних даних

Користувачу на вхід програми подається СЛАР у матричному вигляді, тобто задається за допомогою матриці коефіцієнтів та стовпця вільних членів, числа яких є дійсними. Значення кожного члена матриці коефіцієнтів має знаходитися в діапазоні $[-1000; 1000]$, а значення кожного вільного члена — $[-10000; 10000]$. При обробці вхідних даних введеної СЛАР користувачем у формі введення, числа коефіцієнтів матриці та стовпя вільних членів будуть округлені до 6 знака після точки включно.

Результатом виконання програми є розв’язок встановленої СЛАР. У вікні програми невідомі змінні встановленої СЛАР будуть відображатися вертикальною таблицею значень розв’язків встановленої СЛАР. Якщо розв’язок буде виведено до файлу, то його змістом буде список дійсних чисел розміром вирішеної СЛАР, де числа розділені одним пробілом, а ціла частина числа та її дробове значення буде відділене точкою.

6.3 Системні вимоги

	Мінімальні	Рекомендовані
Операційна система	ArchLinux (з останніми обновленнями) з ядром «6.9.1-arch1-1»	ArchLinux (з останніми обновленнями) з ядром «6.9.1-zen1-1-zen»
Процесор	Intel i7-6700HQ (8) @ 3.500GHz	
Оперативна пам'ять	8 GB RAM	16 GB RAM
Відеоадаптер	Intel HD Graphics 530	
Дисплей	1920x1080	1920x1080 або вище
Прилади введення	Клавіатура, комп'ютерна миша	
Додаткове програмне забезпечення	Пакет «gtkmm3»	

7 АНАЛІЗ РЕЗУЛЬАТІВ

Головною задачею курсової роботи була реалізація програми для розв'язання СЛАР наступними методами: LUP, Гауса-Холецького, обертання.

Критичні ситуації у роботі програми виявлені не були. Під час тестування було виявлено, що більшість помилок виникало тоді, коли користувачем вводилися не числові вхідні дані. Тому всі дані, які вводить користувач, перевіряються на коректність перед обробкою.

Для перевірки та доведення достовірності результатів виконання програмного забезпечення скористаюся LibreOffice Calc:

а) LUP-метод.

Результат виконання LUP-методу наведено на рисунку 7.1:

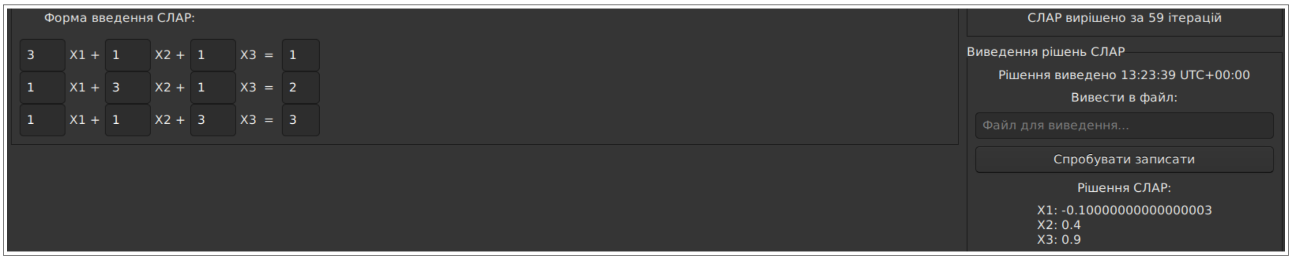


Рисунок 7.1 – Результат виконання LUP-методу

Оскільки результат виконання (7.1) збігається з результатом в LibreOffice Calc (рисунок 7.2), то даний метод працює вірно.

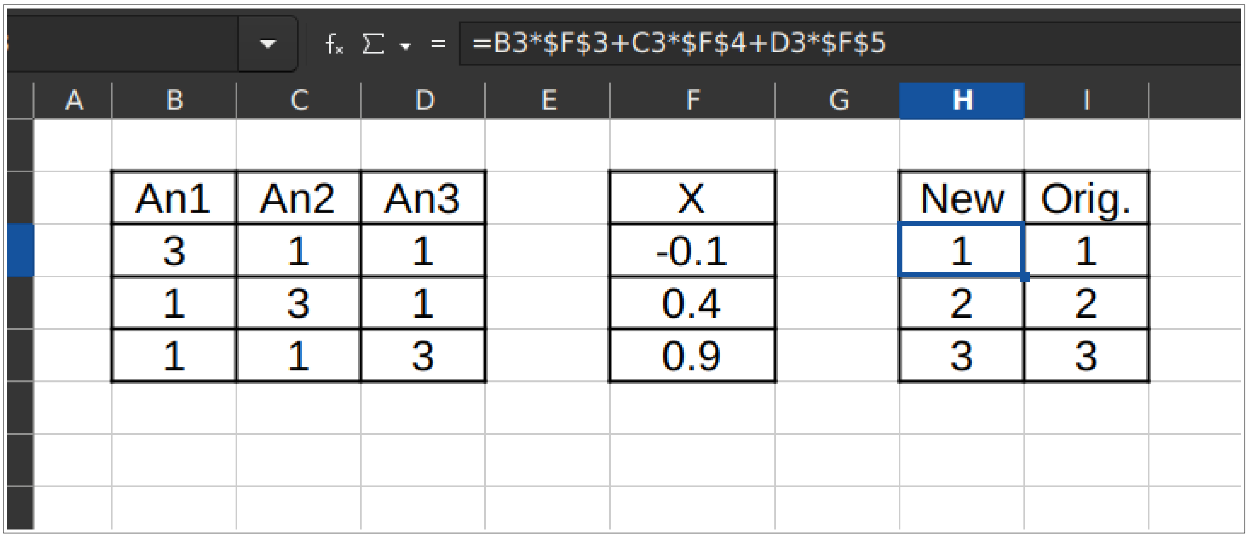


Рисунок 7.2 – Перевірка методу Якобі в LibreOffice Calc

б) Метод Гауса-Холецького.

Результат виконання методу Гауса-Холецького наведено на рисунку 7.2:

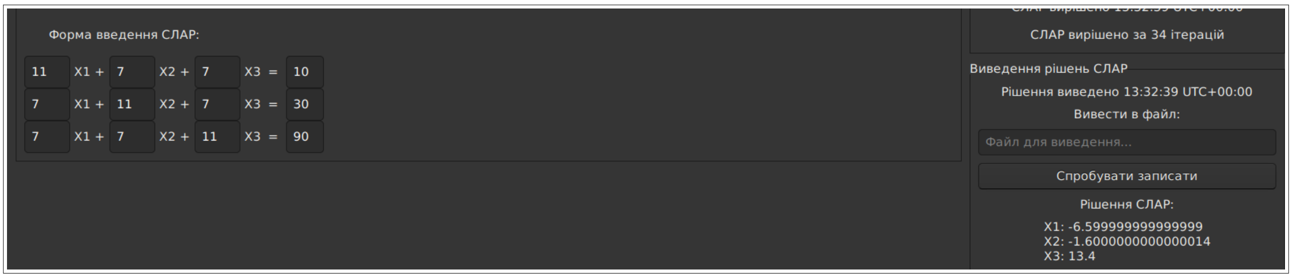


Рисунок 7.3 – Результат виконання методу Гауса-Холецького

Оскільки результат виконання (7.3) збігається з результатом в LibreOffice Calc (рисунок 7.4), то даний метод працює вірно.

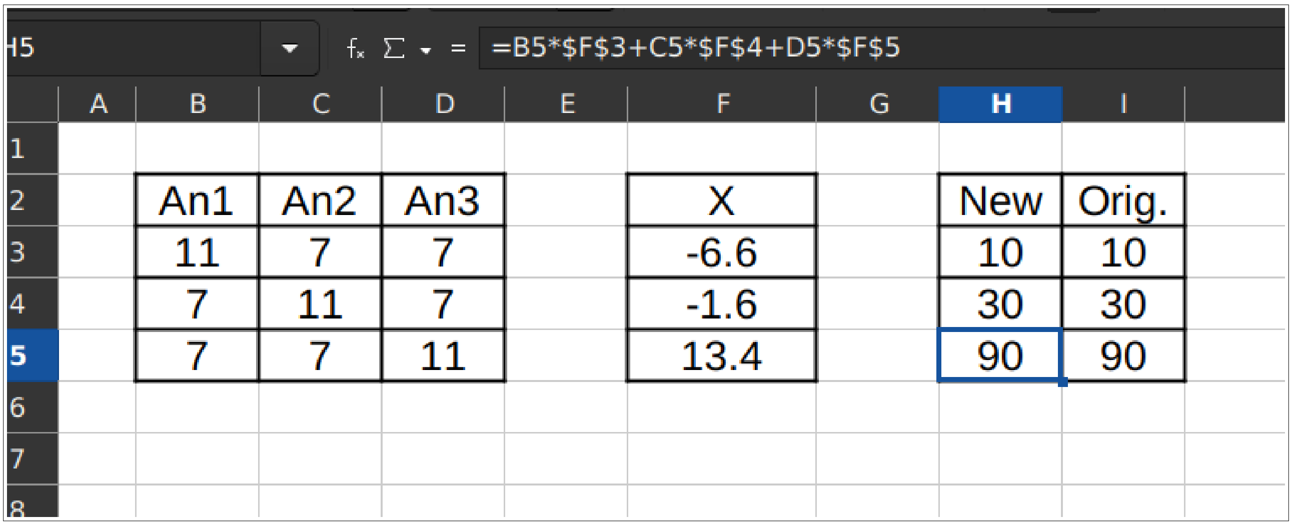


Рисунок 7.4 – Перевірка методу методу Гауса-Холецького в LibreOffice Calc

в) Метод обертання.

Результат виконання методу обертання наведено на рисунку 7.5:

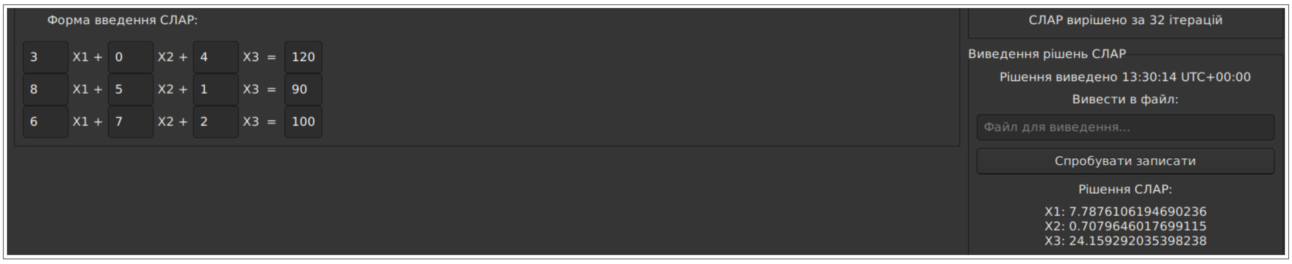


Рисунок 7.5 – Результат виконання методу обертання

Оскільки результат виконання (7.5) збігається з результатом в LibreOffice Calc (рисунок 7.6), то даний метод працює вірно.

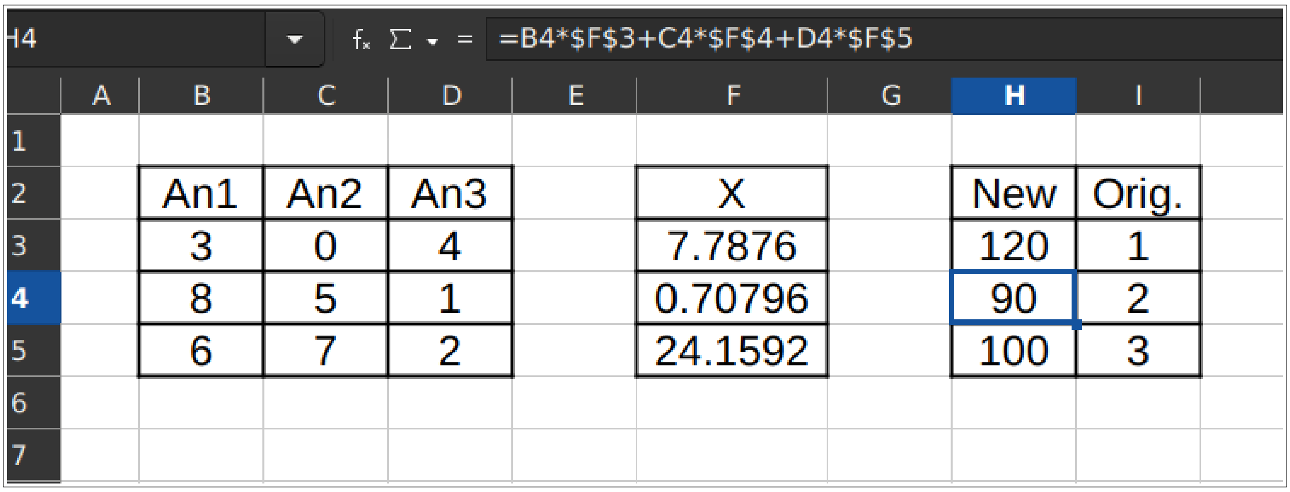


Рисунок 7.6 – Перевірка методу методу обертання в LibreOffice Calc

Для проведення тестування ефективності програми було створено матриці наступного вигляду:

$$R=\begin{pmatrix} n & 1 & 1 & \cdots & 1 \\ 1 & n & 1 & \cdots & 1 \\ 1 & 1 & n & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & n \end{pmatrix}, \quad A=R^T R=\begin{pmatrix} n^2+2 & 2n+1 & 2n+1 & \cdots & 2n+1 \\ 2n+1 & n^2+2 & 2n+1 & \cdots & 2n+1 \\ 2n+1 & n^2+2 & n^2+2 & \cdots & 2n+1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2n+1 & 2n+1 & 2n+1 & \cdots & n^2+2 \end{pmatrix} \quad (7.1)$$

де n — розмірність системи.

Матриця A (7.1) має підходити для всіх методів вирішення СЛАР, включно для методу Гауса-Холецького [3].

Результати тестування ефективності алгоритмів розв’язання СЛАР наведено в таблиці 7.1:

Таблиця 7.1 – Тестування ефективності методів

Розмірність системи	Параметри тестування	Метод		
		LUP	Гауса-Холецького	обертання
100	Кількість ітерацій	378650	191900	358350

Розмірність системи	Параметри тестування	Метод		
		LUP	Гауса-Холецького	обертання
	циклів			
250	Кількість ітерацій циклів	5490375	2761000	5364625
500	Кількість ітерацій циклів	42793250	21459500	42291750
1000	Кількість ітерацій циклів	337836500	169169000	335833500

Візуалізація результатів табилиці 7.1 наведено на рисунку 7.1:

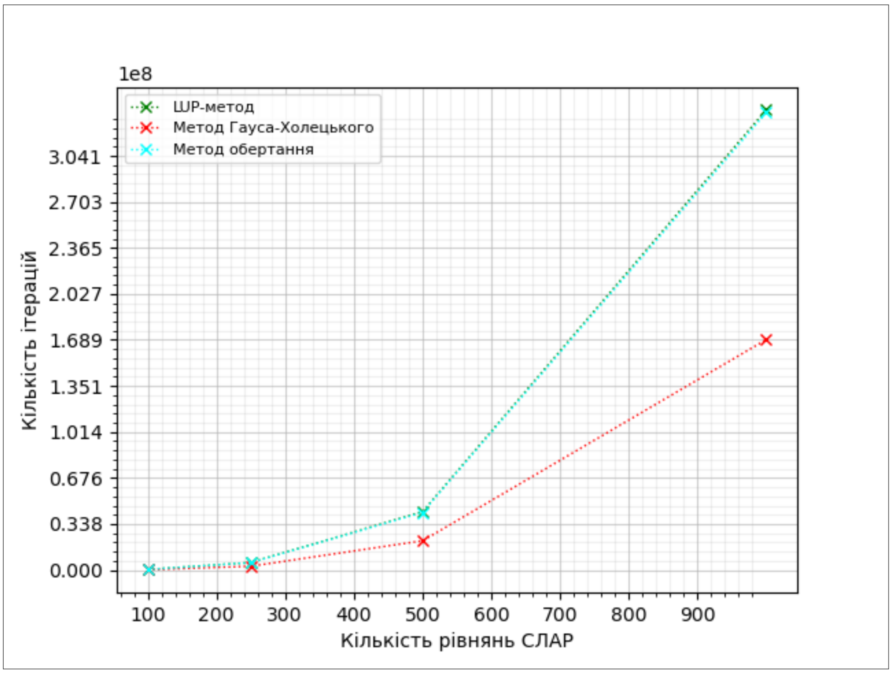


Рисунок 7.1 – Графік залежності кількості ітерацій методу від розміру вхідної системи

Теоретична складність трьох метод вирішення СЛАР має бути $O(n^3)$, бо всі алгоритми використовують потрійно вкладені цикли з кількістю ітерацій $O(n)$. Далі це твердження буде перевірено за допомогою практичної складності всіх методів.

Так як розмірність системи збільшується приблизно в 2 рази, то кількість ітерацій має збільшитися приблизно в 8 разів, оскільки $\frac{(2n)^3}{n^3}=\frac{8n^3}{n^3}=8$. Щоб приблизно визначити степінь члена з найбільшим степеневим коефіцієнтом поліноміального виразу, можна визначити наступну формулу:

$$\begin{aligned}&\lim_{x\to\infty}\frac{jn^p+kn^y+\ldots+\ln^z}{jn^p}=1,\quad p>y>\ldots>z,\quad\Rightarrow\\&\Rightarrow\frac{(an)^p}{n^p}=b\Rightarrow a^p=b\Rightarrow p=\log_a b\end{aligned}\qquad(7.1)$$

, де a — ділення між розмірностями систем лінійних рівнянь, b — ділення між кількістю ітерацій.

Якщо взяти пратичну часову складність LUP-методу (таблиця 7.1), то вийдуть наступні результати (таблиця 7.2) за формулою 7.1.

Таблиця 7.2 — підтвердження теоретичної складності LUP-методу.

Номери сусідніх розмірів СЛАР та кількостех ітерацій	Ділення сусідніх розмірів СЛАР	Ділення сусідніх кількостей ітерацій	Ступінь найстаршого члена поліному
1 та 2	2.5	14.49986795	2.918440021
2 та 3	2	7.794230813	2.962406655
3 та 4	2	7.894621231	2.980870050

Якщо взяти пратичну часову складність методу Гауса-Холецького (таблиця 7.1), то вийдуть наступні результати (таблиця 7.3) за формулою 7.1.

Таблиця 7.3 — підтвердження теоретичної складності методу Гауса-Холецького.

Номери сусідніх розмірів СЛАР та кількостех ітерацій	Ділення сусідніх розмірів СЛАР	Ділення сусідніх кількостей ітерацій	Ступінь найстаршого члена поліному
1 та 2	2.5	14.38770193	2.909964836
2 та 3	2	7.772365085	2.958353669
3 та 4	2	7.883175284	2.978776854

Якщо взяти пратичну часову складність методу обертання (таблиця 7.1), то вийдуть наступні результати (таблиця 7.4) за формулою 7.1.

Таблиця 7.4 — підтвердження теоретичної складності методу обертання.

Номери сусідніх розмірів СЛАР та кількостех ітерацій	Ділення сусідніх розмірів СЛАР	Ділення сусідніх кількостей ітерацій	Ступінь найстаршого члена поліному
1 та 2	2.5	14.97035022	2.953289277
2 та 3	2	7.883449449	2.978827027
3 та 4	2	7.940874993	2.989297985

Якщо подивитися на таблиці 7.2-7.4, то можна зробити висновок, що практична складність методів вирішення СЛАР є $\lim_{x \to \infty} O(f(x)) = O(n^3)$.

За результатами практичного тестування можна зробити наступні висновки про основні алгоритми вирішення СЛАР:

а) Всі розглянуті методи дозволяю знаходити розвязки великих та надвеликих СЛАР.

б) Складність всіх розглянутих методів є кубічною, тобто — $O(n^3)$, де n — кількість рівнянь СЛАР.

в) З розглянутих методів найоптимальнішим для вузького використання є метод Гауса-Холецького з точки зору кількості ітерацій циклів.

г) Найбільш оптимальним для широкого практичного використання є метод обертання, бо його точність обрахунків при його використанні вища, ніж у LUP-метода.

ВИСНОВКИ

Було вивчено та реалізовано три точних методи вирішення СЛАР, які є головними алгоритмами в програмному додатку. Також було додано кілька корисних функціональних можливостей для програмного забезпечення.

Було реалізовано практичний графічний інтерфейс для користувача. Форма введення системи лінійних рівнянь є інтуїтивно зрозумілою для користувача, а також інші елементи керування програмним забезпеченням.

Було проаналізовано практичну часову складність алгоритмів на СЛАР різного розміру на різних алгоритмах точного вирішення СЛАР за різними критеріями оцінювання ефективності виконання алгоритмів. Під час аналізування результатів було видвинуті наступні твердження:

1) Всі методи вирішення СЛАР можуть вирішувати будь-які системи лінійних рівнянь за розміром від 1.
2) З точки зору кількості ітерацій, метод Гауса-Холецького є найбільш ефективним.
3) Метод Гауса-Холецького не є універсальним методом для точного вирішення СЛАР, бо він працює лише для додатньо визначеної матриці коефіцієнтів.
4) З точки зору точності результів, метод обертання є найбільш точним, а ніж інші методи.
5) Всі методи вирішення СЛАР працюють достатньо швидко з точки зору кількості ітерацій циклів.
6) Всі алгоритми працюють з теоретичною складністю $O(n^3)$.

ПЕРЕЛІК ПОСИЛАНЬ

[1] Прокопенко, Ю. В., Татарчук, Д. Д., & Казміренко, В. А. (2003). Обчислювальна математика.

[2] Шахно, С. М., Дудикевич, А. Т., & Левицька, С. М. (2009). ББК В 162.73 я73 Ш81.

[3] Ayres Jr, F. (1962). Theory and problems of matrices. McGraw-Hill, 134.

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив
Керівник __Головченко М.М.____
«_03»___квітня______2024 р.

Виконавець:
Студент __Адаменко А.Б.__
«_03»____квітня________2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання курсової роботи
на тему: «Розв'язання СЛАР точними методами»
з дисципліни:
«Основи програмування»

Київ 2024

7.1 *Мета*: Метою курсової роботи є забезпечення надійності та коректності програмного забезпечення для розв’язання СЛАР різними точними методами.

7.2 *Дата початку роботи*: «_03»___квітня____2024 р.

7.3 *Дата закінчення роботи*: «___»___________ 2024 р.

7.4 *Вимоги до програмного забезпечення*.

1) Функціональні вимоги:

- Можливість ввести розмірність СЛАР.
- Можливість задавати вільні коефіцієнти та коефіцієнти невідомих змінних СЛАР.
- Можливість обирати один з методів розв’язання СЛАР.
- Розв’язати СЛАР обраним методом (LUP-методом, метод обертання, метод Гауса-Холецького (квадратного кореня)).
- Можливість зберігати результати розв’язання СЛАР у текстовий файл.
- Можливість графічного розв’язання СЛАР у випадку розмірності СЛАР у два рівняння.
- Відображати значення практичної складності алгоритмів розв’язання СЛАР.
- Нормальна обробка та реакція на некоректні дії користувача.
- Можливість відображення практичної складності алгоритму.

2) Нефункціональні вимоги:

- Можливість запускати та працювати з програмним забезпеченням на операційній системі «Linux».
- Все програмне забезпечення та супроводжувача технічна документація повинні задовольняти наступним ДЕСТам:

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 2015 - Розробка технічної документації.

7.5 *Стадії та етапи розробки*:

1) Об'єктно-орієнтований аналіз предметної області задачі (до 15.05.2024 р.)
2) Об'єктно-орієнтоване проектування архітектури програмної системи (до 24.05.2024 р.)
3) Розробка програмного забезпечення (до 28.05.2024 р.)
4) Тестування розробленої програми (до 26.05.2024 р.)
5) Розробка пояснювальної записки (до __.__.2024 р.).
6) Захист курсової роботи (до __.__.2024 р.).

7.6 *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду програмного забезпечення вирішення задачі знаходження розв’язків системи лінійних рівнянь

(Найменування програми (документа))

GIT

(Вид носія даних)

https://github.com/adamenko-arsen/SLE-Accurate-Solver

(Обсяг програми (документа), арк., байти)

студента групи ІП-35 І курсу

Адаменко А.Б.

Файл «Vector.hpp»:

```
#pragma once

#include <vector>

class Vector
{
public:
    Vector();

    explicit Vector(std::size_t vectorSize);

    double operator[](std::size_t index) const;
    double& operator[](std::size_t index);

    std::size_t Size() const noexcept;

private:
    std::vector<double> numbersVector{};
};
```

Файл «Matrix.hpp»:

```
#pragma once

#include <cstdint>

#include <vector>

class Matrix
{
public:
    Matrix();

    explicit Matrix(std::size_t height, std::size_t width);

    double At(std::size_t y, std::size_t x) const;
    double& At(std::size_t y, std::size_t x);

    std::size_t Width() const noexcept;
    std::size_t Height() const noexcept;

    std::size_t TryGetEdgeSize() const noexcept;

    bool IsSquare() const noexcept;
```

```
private:
    std::size_t width = 0, height = 0;

    std::vector<double> flattenMatrixVH{};

    double flatMtxElemRef(std::size_t y, std::size_t x) const;
    double& flatMtxElemRef(std::size_t y, std::size_t x);
};
```

Файл «LUPSolver.cpp»:

```
#include "LUPSolver.hpp"

#include <cmath>

bool LUPSolver::isCloseToZero(double x)
{
    return std::fabs(x) < 1e-9;
}

std::size_t LUPSolver::maxDiagLine(const Matrix& A, std::size_t baseColumn)
{
    auto maxDiagValue = std::fabs(A.At(baseColumn, baseColumn));
    std::size_t indexOfMax = baseColumn;

    for (std::size_t curColumn = 0; curColumn < A.TryGetEdgeSize(); curColumn++)
    {
        auto newDiagValue = std::fabs(A.At(curColumn, baseColumn));

        if (newDiagValue > maxDiagValue)
        {
            maxDiagValue = newDiagValue;
            indexOfMax = curColumn;
        }
    }

    return indexOfMax;
}

std::optional<LUPDecResult>    LUPSolver::lupDecompose(Matrix    A,    IterationsCounter&
itersCounter)
{
    auto n = A.TryGetEdgeSize();

    std::vector<std::size_t> P(n);

    for (std::size_t i = 0; i < n; i++)
```

```
{
    P[i] = i;

    itersCounter.AddNew();
}

for (std::size_t j = 0; j < n; j++)
{
    auto maxDiagColumn = maxDiagLine(A, j);

    if (isCloseToZero(A.At(maxDiagColumn, j)))
    {
        return std::nullopt;
    }

    for (std::size_t r = 0; r < n; r++)
    {
        std::swap(A.At(j, r), A.At(maxDiagColumn, r));

        itersCounter.AddNew();
    }

    std::swap(P[j], P[maxDiagColumn]);

    for (std::size_t i = j; i < n; i++)
    {
        {
            double sum = 0;

            for (std::size_t k = 0; k < j; k++)
            {
                sum += A.At(i, k) * A.At(k, j);

                itersCounter.AddNew();
            }

            A.At(i, j) -= sum;

            itersCounter.AddNew();
        }

        if (i >= j + 1)
        {
            double sum = 0;

            for (std::size_t k = 0; k < j; k++)
```

```
                {
                    sum += A.At(j, k) * A.At(k, i);

                    itersCounter.AddNew();
                }

                A.At(j, i) -= sum;

                if (isCloseToZero(A.At(j, j)))
                {
                    return std::nullopt;
                }

                A.At(j, i) /= A.At(j, j);

                itersCounter.AddNew();
            }
        }
    }

    Matrix L(n, n);
    Matrix U(n, n);

    for (std::size_t y = 0; y < n; y++)
    {
        for (std::size_t x = 0; x < n; x++)
        {
            L.At(y, x) = 0;

            itersCounter.AddNew();
        }
        U.At(y, y) = 1;

        itersCounter.AddNew();
    }

    for (std::size_t j = 0; j < n; j++)
    {
        for (std::size_t i = 0; i < j + 1; i++)
        {
            L.At(j, i) = A.At(j, i);

            itersCounter.AddNew();
        }
    }
```

```
    for (std::size_t j = 0; j < n; j++)
    {
        for (std::size_t i = j + 1; i < n; i++)
        {
            U.At(j, i) = A.At(j, i);

            itersCounter.AddNew();
        }
    }

    return LUPDecResult {
            .L = L
        , .U = U
        , .P = P
    };
}

std::optional<Vector> LUPSolver::solveY(
        const Matrix& L
    , const std::vector<std::size_t>& P
    , const Vector& B
    , IterationsCounter& itersCounter
)
{
    auto n = B.Size();

    Vector Y(n);

    for (std::size_t i = 0; i < n; i++)
    {
        if (isCloseToZero(L.At(i, i)))
        {
            return std::nullopt;

            itersCounter.AddNew();
        }
    }

    for (std::size_t i = 0; i < n; i++)
    {
        double sum = 0;

        for (std::size_t k = 0; k < i; k++)
        {
            sum += L.At(i, k) * Y[k];
```

```
            itersCounter.AddNew();
        }

        Y[i] = (
            B[P[i]] - sum
        )
        / L.At(i, i);

        itersCounter.AddNew();
    }

    return Y;
}

Vector LUPSolver::solveX(const Matrix& U, const Vector& Y, IterationsCounter&
itersCounter)
{
    auto n = Y.Size();

    Vector X(n);

    for (std::ptrdiff_t i = n - 1; i >= 0; i--)
    {
        double sum = 0;

        for (std::size_t k = i + 1; k < n; k++)
        {
            sum += U.At(i, k) * X[k];

            itersCounter.AddNew();
        }

        X[i] = Y[i] - sum;

        itersCounter.AddNew();
    }

    return X;
}

SolvingResult LUPSolver::SolveInternally(Matrix&& A, Vector&& B)
{
    IterationsCounter itersCounter{};

    auto mayLUPDecRes = lupDecompose(A, itersCounter);
```

```
    if (! mayLUPDecRes.has_value())
    {
        return SolvingResult::Error();
    }

    const auto& lup = mayLUPDecRes.value();

    auto mayY = solveY(lup.L, lup.P, B, itersCounter);

    if (! mayY.has_value())
    {
        return SolvingResult::Error();
    }

    auto X = solveX(lup.U, mayY.value(), itersCounter);

                                                                        return
SolvingResult::Successful(std::move(X)).SetItersCountChainly(itersCounter.GetTotalCount()
);
}
```

Файл «GaussHoletskiySolver.cpp»:

```
#include "GaussHoletskiySolver.hpp"

#include <cmath>

bool GaussHoletskiySolver::isCloseToZero(double x)
{
    return std::fabs(x) < 1e-9;
}

bool GaussHoletskiySolver::isCloseToZeroForSolves(double x)
{
    return std::fabs(x) < 1e-9;
}

bool GaussHoletskiySolver::isCloseToZeroForAmbigiousCheck(double x)
{
    return std::fabs(x) < 1e-9;
}

bool  GaussHoletskiySolver::isSolveSuitable(const  Matrix&  A,  const  Vector&  B,  const
Vector& X)
{
    auto n = B.Size();
```

```
    Vector NewB(n);

    for (std::size_t y = 0; y < n; y++)
    {
        NewB[y] = 0;

        for (std::size_t x = 0; x < n; x++)
        {
            NewB[y] += A.At(y, x) * X[x];
        }
    }

    for (std::size_t y = 0; y < n; y++)
    {
        if (! isCloseToZeroForSolves(B[y] - NewB[y]))
        {
            return false;
        }
    }

    return true;
}

#include <iostream>

bool GaussHoletskiySolver::isSolveNotAmbigious(const Vector& B, const Vector& X)
{
    if (! (B.Size() >= 2))
    {
        return true;
    }

    for (std::size_t i = 0; i < B.Size(); i++)
    {
        std::cout << B[i] << std::endl;

        if (! isCloseToZeroForAmbigiousCheck(B[i]))
        {
            return true;
        }
    }

    for (std::size_t i = 0; i < X.Size(); i++)
    {
        std::cout << X[i] << std::endl;
```

```
        if (! isCloseToZeroForAmbigiousCheck(X[i]))
        {
            return true;
        }
    }

    return false;
}

std::optional<LDLDecResult>    GaussHoletskiySolver::ldlDecompose(const    Matrix&    A,
IterationsCounter& itersCounter)
{
    auto n = A.TryGetEdgeSize();

    Matrix L(n, n);
    Matrix D(n, n);

    for (std::size_t y = 0; y < n; y++)
    {
        for (std::size_t x = 0; x < n; x++)
        {
            L.At(y, x) = 0;
            D.At(y, x) = 0;

            itersCounter.AddNew();
        }
    }

    for (std::size_t j = 0; j < n; j++)
    {
        L.At(j, j) = 1;

        {
            double sum = 0;

            for (std::size_t k = 0; k < j; k++)
            {
                sum += std::pow(L.At(j, k), 2) * D.At(k, k);

                itersCounter.AddNew();
            }

            D.At(j, j) = A.At(j, j) - sum;

            itersCounter.AddNew();
        }
```

```
        for (std::size_t i = j + 1; i < n; i++)
        {
            auto diagElem = D.At(j, j);

            if (isCloseToZero(diagElem))
            {
                return std::nullopt;
            }

            double sum = 0;

            for (std::size_t k = 0; k < j; k++)
            {
                sum += L.At(i, k) * D.At(k, k) * L.At(j, k);

                itersCounter.AddNew();
            }

            L.At(i, j) = (A.At(i, j) - sum) / D.At(j, j);

            itersCounter.AddNew();
        }
    }

    return LDLDecResult
    {
            .L = L
        , .D = D
    };
}

Vector GaussHoletskiySolver::solveY(const Matrix& L, const Vector& B, IterationsCounter&
itersCounter)
{
    auto n = B.Size();

    Vector Y(n);

    for (std::size_t i = 0; i < n; i++)
    {
        double sum = 0;

        for (std::size_t j = 0; j < i; j++)
        {
            sum += L.At(i, j) * Y[j];
```

```
            itersCounter.AddNew();
        }

        Y[i] = B[i] - sum;

        itersCounter.AddNew();
    }

    return Y;
}

std::optional<Vector> GaussHoletskiySolver::solveZ(const Matrix& D, const Vector& Y,
IterationsCounter& itersCounter)
{
    auto n = Y.Size();

    Vector Z(n);

    for (std::size_t i = 0; i < n; i++)
    {
        auto diagElem = D.At(i, i);

        if (isCloseToZero(diagElem))
        {
            return std::nullopt;

            itersCounter.AddNew();
        }

        Z[i] = Y[i] / diagElem;

        itersCounter.AddNew();
    }

    return Z;
}

Vector GaussHoletskiySolver::solveX(const Matrix& L, const Vector& Z, IterationsCounter&
itersCounter)
{
    auto n = Z.Size();

    Vector X(n);

    for (std::ptrdiff_t i = n - 1; i >= 0; i--)
```

```
    {
        double sum = 0;

        for (std::size_t j = i + 1; j < n; j++)
        {
            sum += L.At(j, i) * X[j];

            itersCounter.AddNew();
        }

        X[i] = Z[i] - sum;

        itersCounter.AddNew();
    }

    return X;
}

SolvingResult GaussHoletskiySolver::SolveInternally(Matrix&& A, Vector&& B)
{
    IterationsCounter itersCounter{};

    auto mayLDL = ldlDecompose(A, itersCounter);

    if (! mayLDL.has_value())
    {
        return SolvingResult::Error();
    }

    const auto& ldl = mayLDL.value();

    auto Y = solveY(ldl.L, B, itersCounter);
    auto mayZ = solveZ(ldl.D, Y, itersCounter);

    if (! mayZ.has_value())
    {
        return SolvingResult::Error();
    }

    auto X = solveX(ldl.L, mayZ.value(), itersCounter);

    if (! isSolveSuitable(A, B, X))
    {
        return SolvingResult::Error();
    }
```

```
    if (! isSolveNotAmbigious(B, X))
    {
        return SolvingResult::Error();
    }

                                                                                              return
SolvingResult::Successful(X).SetItersCountChainly(itersCounter.GetTotalCount());
}
```

Файл «RotationSolver.cpp»:

```
#include "RotationSolver.hpp"

#include "../Containers/AllocArray2D.inc.hpp"

#include <cmath>

bool RotationSolver::isCloseToZero(double x)
{
    return std::fabs(x) < 1e-9;
}

SolvingResult RotationSolver::SolveInternally(Matrix&& A, Vector&& B)
{
    IterationsCounter itersCounter{};

    auto n = B.Size();

    RTArray2D<double> AB(n + 1, n);

    for (std::size_t y = 0; y < n; y++)
    {
        for (std::size_t x = 0; x < n; x++)
        {
            AB.At(y, x) = A.At(y, x);

            itersCounter.AddNew();
        }

        AB.At(y, n) = B[y];

        itersCounter.AddNew();
    }

    for (std::size_t i = 0; i < n - 1; i++)
    {
        for (std::size_t j = i + 1; j < n; j++)
```

```
    {
        auto b = AB.At(j, i);
        auto a = AB.At(i, i);

        auto squaresSum = a*a + b*b;

        if (! (squaresSum > 0))
        {
            return SolvingResult::Error();
        }

        auto sqrtedSquaresSum = std::sqrt(squaresSum);

        if (isCloseToZero(sqrtedSquaresSum))
        {
            return SolvingResult::Error();
        }

        auto c = a / sqrtedSquaresSum;
        auto s = b / sqrtedSquaresSum;

        for (std::size_t k = i; k < n + 1; k++)
        {
            auto t = AB.At(i, k);

            AB.At(i, k) = c * AB.At(i, k) + s * AB.At(j, k);
            AB.At(j, k) = -s * t + c * AB.At(j, k);

            itersCounter.AddNew();
        }

        itersCounter.AddNew();
    }
}

Vector X(n);

for (std::size_t i = 0; i < n; i++)
{
    if (isCloseToZero(AB.At(i, i)))
    {
        return SolvingResult::Error();
    }
}

for (std::ptrdiff_t i = n - 1; i >= 0; i--)
```

```
    {
        double membersSum = 0;

        for (std::size_t j = i + 1; j < n; j++)
        {
            membersSum += AB.At(i, j) * X[j];

            itersCounter.AddNew();
        }

        X[i] = (AB.At(i, n) - membersSum) / AB.At(i, i);

        itersCounter.AddNew();
    }

                                                                                            return
SolvingResult::Successful(std::move(X)).SetItersCountChainly(itersCounter.GetTotalCount()
);
}
```