

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. Частина 2. Модульне програмування»

(назва дисципліни)

на тему: «Розв’язання СЛАР точними методами»

Студента 1 ку групи ІІІ-43  
Дутова Івана Андрійовича  
Спеціальності 121 «Інженерія програмного  
забезпечення»

Ахаладзе А.Е.

(посада, вчене звання, науковий  
ступінь, прізвище та ініціали)

Кількість балів: \_\_\_\_\_

Національна оцінка \_\_\_\_\_

Члени комісії

Головченко М.М.

\_\_\_\_\_  
(підпис)

\_\_\_\_\_  
(посада, вчене звання, науковий ступінь,  
прізвище та ініціали)

Ахаладзе А.Е.

\_\_\_\_\_  
(підпис)

\_\_\_\_\_  
(посада, вчене звання, науковий ступінь,  
прізвище та ініціали)

Київ – 2025 рік

# КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування. Курсова робота

Напрямок «ІПЗ»

Курс 1 Група ІІІ-43

Семестр 2

## ЗАВДАННЯ

на курсову роботу студента

Дутова Івана Андрійовича

(прізвище, ім'я, по батькові)

1. Тема роботи розв'язання СЛАР точними методами

2. Строк здачі студентом закінченої роботи 25 травня 2025 року

3. Вихідні дані до роботи

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

Постановка задачі, теоретичні відомості, опис алгоритмів, опис програмного, забезпечення, тестування програмного забезпечення, інструкція користувача, аналіз результатів, висновки, перелік посилань

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

6. Дата видачі завдання 04.02.2025

## КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	04.02.2025	
2.	Підготовка ТЗ	15.02.2025	
3.	Пошук та вивчення літератури з питань курсової роботи	13.03.2025	
4.	Розробка сценарію роботи програми	16.03.2025	
6.	Узгодження сценарію роботи програми з керівником	17.03.2025	
5.	Розробка (вибір) алгоритму розв'язання задач	21.03.2025	
6.	Узгодження алгоритму з керівником	24.03.2025	
7.	Узгодження з керівником інтерфейсу користувача	26.03.2025	
8.	Розробка програмного забезпечення	01.04.2025	
9.	Налагодження розрахункової частини програми	12.04.2025	
10.	Розробка та налагодження інтерфейсної частини програми	14.04.2025	
11.	Узгодження з керівником набору тестів для контрольного прикладу	18.04.2025	
12.	Тестування програми	25.04.2025	
13.	Підготовка пояснювальної записки	16.05.2025	
14.	Здача курсової роботи на перевірку	25.05.2025	
15.	Захист курсової роботи	29.05.2025	

Студент \_\_\_\_\_  
(підпис)

Керівник \_\_\_\_\_  
(підпис)

Ахаладзе А.Е.  
(прізвище, ім'я, по батькові)

«\_\_» \_\_\_\_\_ 20\_\_ р.

## АНОТАЦІЯ

Пояснювальна записка до курсової роботи: **55 сторінок, 24 рисунки, 28 таблиць, 4 посилання.**

У процесі виконання роботи було **вивчено класичні алгоритми розв'язання СЛАР**, зокрема:

**Метод Гауса**, що базується на послідовному приведенні матриці до трикутного вигляду з подальшим застосуванням зворотного ходу;

**Метод Жордана-Гауса**, що дозволяє одразу звести матрицю до діагонального вигляду, одержуючи розв'язки без необхідності зворотного проходження;

**Метод оберненої матриці**, який дає змогу знайти розв'язки шляхом множення вектора вільних членів на обернену до матриці коефіцієнтів.

Для кожного з методів **було розроблено алгоритми, реалізовані в програмному середовищі** з використанням сучасних технологій. Особливу увагу приділено **візуалізації процесу обчислення**, що реалізований у вигляді **покрокового логування** дій користувача, де кожна дія супроводжується відповідними візуальними змінами на інтерфейсі.

Програмну реалізацію створено з використанням **модерного фреймворку React**, що дозволило забезпечити **інтерактивність, адаптивність і зручну навігацію** між кроками обчислень. Реалізовано механізм переходу між кроками вперед і назад, а також режим швидкого проходження (skip-mode), що дає змогу побачити як повну історію перетворень, так і результат.

Кожен з кроків має опис, що базується на логічній структурі самого методу, а також відображає відповідні зміни в матриці коефіцієнтів. Окремо розглянуто випадки вироджених систем, багатозначності розв'язків або відсутності розв'язку.

Ключові слова: СИСТЕМА ЛІНІЙНИХ РІВНЯНЬ, МЕТОД ГАУСА, МЕТОД ЖОРДАНА-ГАУСА, МЕТОД ОБЕРНЕНОЇ МАТРИЦІ, АЛГОРИТМ, ПОКРОКОВЕ ЛОГУВАННЯ, REACT, ІНТЕРАКТИВНА ВІЗУАЛІЗАЦІЯ.

## ЗМІСТ

Вступ	6
1 Постановка задачі	7
2 Теоретичні відомості	8
2.1 Метод Гауса	8
2.1 Метод Жордана-Гауса	9
2.2 Метод оберненої матриці	10
3 Опис алгоритмів	12
3.1 Загальний алгоритм	12
3.2 Алгоритм методу Гауса	13
3.3 Алгоритм методу Жордана-Гауса	15
3.4 Алгоритм методу оберненої матриці	17
4 Опис програмного забезпечення	19
4.1 Діаграма класів програмного забезпечення	19
4.2 Опис методів частин програмного забезпечення	20
1.1.1 Стандартні методи	20
1.1.2 Користувацькі методи	21
5 Тестування програмного забезпечення	22
5.1 План тестування	22
5.2 Приклади тестування	25
6 Інструкція користувача	26
6.1 Робота з програмою	26
6.2 Формат вхідних та вихідних даних	36
6.3 Системні вимоги	36
7 Аналіз результатів	38
Висновки	41
Перелік посилань	42
Додаток А Технічне завдання	43
Додаток Б Тексти програмного коду	46

## ВСТУП

Ця курсова робота має на меті розробку якісного програмного забезпечення для розв'язання СЛАР точними методами. Розв'язання СЛАР є важливою частиною роботи інженерів, фізиків та математиків. Ця проблема постає у практичних задачах моделювання фізичних процесів, аналізі електричних кіл, розрахунку статичної та динамічної конструкцій при будівництві, комп'ютерній графіці для рендерингу та геометричних перетворень тощо.

Розглянуто такі методи розв'язання СЛАР:

- 1) Метод Гауса (класичний) – знаходження розв'язків шляхом перетворення матриці коефіцієнтів на трикутну та подальшою підстановкою.
- 2) Метод Жордана-Гауса – знаходження розв'язків шляхом перетворення матриці коефіцієнтів на одиничну
- 3) Метод оберненої матриці – знаходження розв'язків як добутку матриці, оберненої до матриці коефіцієнтів, та вектора вільних членів.

У ході виконання курсової роботи буде розроблено алгоритмічну складову, програмний інтерфейс, протестовано програмне забезпечення та написано інструкцію користувача.

## 1 ПОСТАНОВКА ЗАДАЧІ

Вхідними даними для програмного забезпечення є СЛАР, яка задана у матричному вигляді:

$$Ax = b,$$

де  $A$  – матриця коефіцієнтів,  $x$  – вектор шуканих значень (рішення системи),  $b$  – вектор вільних членів.

Проте для зручності подальших обчислень матрицю  $A$  та вектор  $b$  було вирішено об'єднати в одну розширену матрицю  $(A|B)$ .

$$(A|B) = \left[ \begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array} \right]$$

Програмне забезпечення повинно обробляти матрицю коефіцієнтів та стовпець вільних членів для СЛАР розмірність яких знаходиться в межах від 1 до 1000.

Вихідними даними для програмного забезпечення буде сукупність дійсних чисел, що є розв'язками даної системи, які виводяться на екран. Програмне забезпечення повинно завжди видавати розв'язок. Якщо система не має розв'язків або їх нескінченна кількість, то програма повинна видати відповідне повідомлення. Програма повинна видавати помилки у випадку некоректних дій користувача та у випадку виникнення виключних ситуацій.

## 2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Квадратичну систему з  $n$  лінійних рівнянь можна задати наступним чином:

$$Ax = b \quad (2.1)$$

де:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad b = [ \quad ]$$

### 2.1 Метод Гауса

Сутність методу Гауса полягає в тому, щоб звести матрицю коефіцієнтів до форми, що зручна для подальшої підстановки.

Крок 1. Матриця  $A$  шляхом елементарних перетворень на рядках (перестановок рядків, лінійних операцій) приводиться до трикутної форми:

$$[A|b] \xrightarrow{\text{елементарні операції}} U,$$

де  $U$  має вигляд:

$$U = \left[ \begin{array}{cccc|c} u_{11} & u_{12} & \dots & u_{1n} & c_1 \\ 0 & u_{22} & \dots & u_{2n} & c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & u_{nn} & c_n \end{array} \right]$$

Елементарними операціями йменують лінійні операції та перестановку рядків.

Ці ж самі перетворення застосовуються і до  $b$ , що перетворює систему на:

$$Ux = c$$

Якщо під час прямого ходу діагональний елемент виявляється нульовим, то необхідно здійснити перестановку рядків (зазвичай обирають рядок із найбільшим абсолютним значенням в тому ж стовпці).

Якщо такого рядка не знайдено, то система вироджена (тобто нескінченна кількість розв'язків, якщо відповідний член з  $c$  є нульовим, інакше жодних розв'язків). Слід враховувати усі рядки при оцінці нескінченності розв'язків, адже якщо хоча б один із рядків утворює рівняння, що не має розв'язків, відповідна СЛАР не має розв'язків



Крок 2. Зворотний хід

Починаючи з останнього рівняння:

$$u_{nn}x_n = c_n \Rightarrow x_n = \frac{c_n}{u_{nn}}$$

та піднімаючись догори, виразимо інші  $x_i$  за вже знайденими:

$$x_i = \frac{1}{u_{ii}} \left( c_i - \sum_{j=i+1}^n u_{ij}x_j \right)$$

## 2.1 Метод Жордана-Гауса

Подібно до методу Гауса, метод Жордана-Гауса послідовно застосовує елементарні перетворення із метою зведення матриці до одиничної.

Крок 1. Прямий хід.

Для кожного стовпця  $j$  знаходиться такий рядок  $i \geq j$ , де  $a_{ij} \neq 0$ , і:

- За потреби — поміняти рядки місцями (частковий вибір головного елемента).
- Поділити рядок на  $a_{ij}$  для отримання одиниці на головній діагоналі.
- Занулити всі елементи **під** цією одиницею за допомогою елементарних перетворень рядків.

Відповідні операції виконуються і на векторі  $b$ .

Після цього кроку, як і в методі Гауса, отримаємо верхню трикутну форму:

$$Ux = c,$$

проте не звичайну, а так звану рядково зведену (на діагоналі одиниці):

$$U = \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ 0 & 1 & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

Якщо хоча б один із елементів головної діагоналі нульовий, слід розглянути відповідний коефіцієнт вектора  $c$  відповідно до методу Гауса.

## Крок 2. Зворотний хід вгору

На відміну від класичного методу Гауса, у методі Жордана-Гауса відбувається зведення і над головною діагоналлю.

Відбувається це аналогічно прямому ходу у методі Гауса, проте з кінця.

Для кожного елемента  $a_{ij}$  над головною діагоналлю (тобто при  $i < j$ ), ми виконуємо:

$$R_i \leftarrow R_i - u_{ij} \cdot R_j,$$

де  $R_i$  – поточний рядок, що ми очищаємо, а  $R_j$  – той, у якому вже стоїть 1 на діагоналі у стовпці  $j$ .

## 2.2 Метод оберненої матриці

Існують різні способи реалізації методу оберненої матриці, найбільш використовуваних два – Монтанте та Жордана-Гауса.

Метод Монтанте стабільніше для цілих чисел, містить мінімальну кількість операцій, проте скоріше підходить для символьних обчислень, тоді як метод Жордана-Гауса краще справляється для операцій з дробовими числами та легший у реалізації. Тож було обрано метод оберненої матриці Жордана-Гауса.

Нагадаємо, що матричне рівняння СЛАР має вигляд:

$$Ax = b$$

Суть методу оберненої матриці полягає в знаходженні матриці  $A^{-1}$ , адже

$$A^{-1}Ax = A^{-1}b \Rightarrow x = A^{-1}b$$

Для знаходження оберненої матриці складемо розширену матрицю виду:

$$U = \left[ \begin{array}{cccc|cccc} a_{11} & a_{12} & \dots & a_{1n} & 1 & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & a_{2n} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & 0 & 0 & \dots & 1 \end{array} \right],$$

де ліва матриця містить члени матриці  $A$ , а права – одиничну матрицю розмірності  $n$ . Згодом, аналогічно до методу Жордана Гауса, ліву частину розширеної матриці до одиничної, проводячи аналогічні операції з лівою матрицею.

Отримаємо:

$$U^* = \left[ \begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & 1 & \ddots & 0 & u_{21} & u_{22} & \ddots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & u_{n1} & u_{n2} & \cdots & u_{nn} \end{array} \right]$$

Якщо на цьому етапі виявиться, що на головній діагоналі містяться нулі, то розв'язок системи не відомий (стовпця вільних членів немає, тому може бути як нескінченна кількість розв'язків, так і жодного).

Отже,

$$A^{-1} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ u_{21} & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{nn} \end{bmatrix}$$

Відповідно корені будуть

$$x = A^{-1}b = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ u_{21} & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

### 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
n	Розмірність СЛАР
Matrix	Розширена матриця системи

#### 3.1 Загальний алгоритм

1. ПОЧАТОК
2. Зчитати розмірність системи.
3. Зчитати матрицю системи та стовпець вільних членів:
  - 3.1. Зчитати розширену матрицю системи:
    - 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 елемент):
        - 3.1.1.1.1. ЯКЩО поточний елемент матриці – вірно записане число, ТО записати його в відповідну комірку matrix. ІНАКШЕ видати повідомлення про помилку та перейти до пункту 8.
  4. ЯКЩО обраний метод Гауса, ТО обробити дані згідно алгоритму методу Гауса (підрозділ 3.2)
  5. ЯКЩО обраний метод Жордана-Гауса, ТО обробити дані згідно алгоритму методу Жордана-Гауса (підрозділ 3.3)
  6. ЯКЩО обраний оберненої матриці, ТО обробити дані згідно з алгоритмом методу оберненої матриці (підрозділ 3.4).
  7. Вивести рішення системи (у тому числі його відсутність).
  8. КІНЕЦЬ

### 3.2 Алгоритм методу Гауса

#### 1. ПОЧАТОК

#### 2. Прямий хід (приведення до трикутного вигляду)

2.1. ЦИКЛ по всіх рядках матриці крім останнього (від  $sourceRow = 0$  до  $n - 2$ ):

2.1.1. Виконати частковий вибір головного елемента (перестановка рядків): Знайти  $pivotRow$ , у якому абсолютне значення елемента в стовпці  $sourceRow$  найбільше серед рядків  $sourceRow..n - 1$

2.1.1.1. Ініціалізувати  $pivotRow = sourceRow$

2.1.1.2. ЦИКЛ для всіх рядків  $i$  від  $sourceRow + 1$  до  $n - 1$ :

2.1.1.2.1.ЯКЩО  $|Matrix[i][sourceRow]| >$  поточного максимуму ( $|Matrix[i][pivotRow]|$ )

2.1.1.2.1.1. Оновити  $pivotRow$

2.1.1.2.2.ЯКЩО головний елемент  $Matrix[pivotRow][sourceRow] \approx 0$ , ТО пропустити перестановку. ІНАКШЕ, ЯКЩО  $pivotRow \neq sourceRow$ , ТО:

2.1.1.2.2.1. переставити  $pivotRow$  і  $sourceRow$

2.1.2. Знищення елементів нижче головного (елімінація)

2.1.2.1. ЦИКЛ для всіх рядків  $eliminationRow = sourceRow + 1$  до  $n - 1$ :

2.1.2.1.1.Ініціалізувати  $pivot = Matrix[sourceRow, sourceRow]$

2.1.2.1.2.ЯКЩО  $pivot \approx 0$  ТО пропустити

2.1.2.1.3.Обчислити множник, що нівелює елемент в стовпці  $sourceRow$  поточного  $eliminationRow$

$$multiplier = \frac{Matrix[sourceRow, eliminationRow]}{pivot}.$$

2.1.2.1.4. ЦИКЛ по стовпцях до кінця матриці, починаючи з діагоналі від  $col = sourceRow$  до  $n - 1$

2.1.2.1.4.1. Присвоюємо нове значення клітинці рядка, що обнуляється:  $Matrix[eliminationRow, col] = Matrix[eliminationRow, col] + Matrix[sourceRow, col] * multiplier$

3. Аналіз форми матриці після прямого ходу: Визначити ранг матриці (без останнього стовпця)

3.1.1. Ініціалізувати  $rank = 0$

3.1.2. ЦИКЛ по кожному рядку матриці:

3.1.2.1. Перевірити, чи всі коефіцієнти в рядку дорівнюють  $\approx 0$

3.1.2.2. ЯКЩО так ТА правий бік  $\neq 0$  ТО система несумісна (ПОВЕРНУТИ «Розв'язки відсутні»)

3.1.2.3. ЯКЩО хоча б один дорівнює  $\neq 0$ , то збільшити  $rank$ .

3.1.3. ЯКЩО  $rank < n$ , то ПОВЕРНУТИ «Нескінченна кількість розв'язків»

4. Зворотний хід (обернене підстановлення)

4.1. Ініціалізувати масив  $roots$  довжиною  $n$ .

4.2. ЦИКЛ  $row$  від останнього  $n - 1$  до 0 (знизу вгору):

4.2.1. Ініціалізувати  $rhs = Matrix[row, cols - 1]$

4.2.2. ЦИКЛ  $col$  від  $row + 1$  до  $n$ : Переносимо члени з лівої частини рівнянь у праву, використовуючи попередньо знайдені корені.

4.2.2.1.  $rhs = rhs - Matrix[row, col] \cdot roots[col]$

4.2.3. Ініціалізувати  $pivot = Matrix[row, row]$

4.2.4. ЯКЩО  $pivot \approx 0$ , ТО ПОМИЛКА «Неочікуваний нульовий головний елемент»

4.2.5. Знаходимо наступний корінь  $roots[row] = \frac{rhs}{pivot}$

4.3. ПОВЕРНУТИ  $roots$

### 3.3 Алгоритм методу Жордана-Гауса

#### 1. ПОЧАТОК

#### 2. Прямий хід (приведення до трикутного вигляду)

2.1. ЦИКЛ по всіх рядках матриці крім останнього (від  $sourceRow = 0$  до  $n - 2$ ):

2.1.1. Виконати частковий вибір головного елемента (перестановка рядків): Знайти  $pivotRow$ , у якому абсолютне значення елемента в стовпці  $sourceRow$  найбільше серед рядків  $sourceRow..n - 1$

2.1.1.1. Ініціалізувати  $pivotRow = sourceRow$

2.1.1.2. ЦИКЛ для всіх рядків  $i$  від  $sourceRow + 1$  до  $n - 1$ :

2.1.1.2.1.ЯКЩО  $|Matrix[i][sourceRow]| >$  поточного максимуму ( $|Matrix[i][pivotRow]|$ )

2.1.1.2.1.1. Оновити  $pivotRow$

2.1.1.2.2.ЯКЩО головний елемент  $Matrix[pivotRow][sourceRow] \approx 0$ , ТО пропустити перестановку. ІНАКШЕ, ЯКЩО  $pivotRow \neq sourceRow$ , ТО:

2.1.1.2.2.1. переставити  $pivotRow$  і  $sourceRow$

2.1.2. Масштабування елементи рядка, щоб головний елемент став 1

2.1.2.1. Ініціалізувати  $pivot = Matrix[sourceRow, sourceRow]$

2.1.2.2.  $multiplier = \frac{1}{pivot}$

2.1.2.3. ЯКЩО  $multiplier \approx 0$  ПРОПУСТИТИ

2.1.2.4. ЦИКЛ  $col$  від  $sourceRow$  до  $n$ :

2.1.2.4.1.  $Matrix[sourceRow][columnIndex] =$   
 $Matrix[sourceRow][columnIndex] * multiplier$

2.1.3. Знищення елементів нижче головного (елімінація)

2.1.3.1. ЦИКЛ для всіх рядків  $eliminationRow = sourceRow + 1$  до  $n - 1$ :

2.1.3.1.1. Ініціалізувати  $pivot =$   
 $Matrix[sourceRow, sourceRow]$

2.1.3.1.2. ЯКЩО  $pivot \approx 0$  ТО пропустити

2.1.3.1.3. Обчислити множник, що нівелює елемент в стовпці  
 $sourceRow$  поточного  $eliminationRow$   
 $multiplier = \frac{Matrix[sourceRow, eliminationRow]}{pivot}$ .

2.1.3.1.4. ЦИКЛ по стовпцях до кінця матриці, починаючи з  
діагоналі від  $col = sourceRow$  до  $n - 1$

2.1.3.1.4.1. Присвоюємо нове значення клітинці рядка, що  
обнуляється:  $Matrix[eliminationRow, col] =$   
 $Matrix[eliminationRow, col] +$   
 $Matrix[sourceRow, col] * multiplier$

3. Масштабуємо останній рядок матриці за кроком 1.1.2.4.1 цього  
алгоритму.

4. Зворотний хід (приведення матриці до одиничної)

4.1. ЦИКЛ  $sourceRow$  від  $n - 1$  до 0

4.1.1. ЦИКЛ  $eliminationRow$  від 0 до  $sourceRow$

4.1.1.1. ЯКЩО  $eliminationRow = sourceRow$  ПРОПУСТИТИ

4.1.1.1.1. Ініціалізувати  $pivot =$   
 $Matrix[sourceRow, sourceRow]$

4.1.1.1.2. ЯКЩО  $pivot \approx 0$  ТО пропустити

4.1.1.1.3. Обчислити множник, що нівелює елемент в стовпці  
 $sourceRow$  поточного  $eliminationRow$   
 $multiplier = \frac{Matrix[sourceRow, eliminationRow]}{pivot}$ .

4.1.1.1.4. ЦИКЛ по стовпцях до кінця матриці, починаючи з  
діагоналі від  $col = sourceRow$  до  $n - 1$

4.1.1.1.4.1. Присвоюємо нове значення клітинці рядка, що  
обнуляється:  $Matrix[eliminationRow, col] =$



$$Matrix[eliminationRow, col] + \\ Matrix[sourceRow, col] * multiplier$$

5. Аналіз форми розширеної матриці після приведення її до одиничної:  
Визначити ранг матриці (без останнього стовпця)

- 5.1.1. Ініціалізувати  $rank = 0$

- 5.1.2. ЦИКЛ по кожному рядку матриці:

- 5.1.2.1. ЯКЩО  $Matrix[row][row]$  близький до нуля ТО

- 5.1.2.1.1. ЯКЩО вільний член  $Matrix[row, n]$  близький до нуля ТО

- 5.1.2.1.1.1. ПОВЕРНУТИ «Розв'язку нема»

- 5.1.2.1.2. ІНАКШЕ  $rank = rank + 1$

- 5.1.3. ЯКЩО  $rank < n$ , то ПОВЕРНУТИ «Нескінченна кількість розв'язків»

6. Зворотний хід (обернене підстановлення)

- 6.1. Ініціалізувати масив  $roots$  довжиною  $n$ .

- 6.2. ЦИКЛ  $row$  від 0 до  $n - 1$

- 6.2.1.  $roots[row] = matrix[row][n]$

- 6.3. ПОВЕРНУТИ  $roots$

### 3.4 Алгоритм методу оберненої матриці

1. ПОЧАТОК

2. Присвоїти  $adjustedMatrix$  матрицю  $A$  (коефіцієнти)

3. Присвоїти  $inverseMatrix$  одиничну матрицю розмірності  $n$

4. Прямий хід (приведення до трикутного вигляду) здійснюємо АНАЛОГІЧНО ДО кроку 2 методу Жордана-Гауса, до того ж повторюючи операції на  $inverseMatrix$ , проте користуючись множниками, отриманими від  $adjustedMatrix$

5. Масштабуємо останній рядок матриці за кроком 3 алгоритму Жордана-Гауса, повторюючи операції на *inverseMatrix* , проте користуючись множниками, отриманими від *adjustedMatrix*
6. Зворотний хід (приведення матриці до одиничної) кроком 4 алгоритму Жордана-Гауса, повторюючи операції на *inverseMatrix* , проте користуючись множниками, отриманими від *adjustedMatrix*
7. Аналіз форми квадратної допоміжної матриці після приведення її до одиничної: Визначити ранг матриці
  - 7.1.1. Ініціалізувати *rank* = 0
  - 7.1.2. ЦИКЛ по кожному рядку матриці:
    - 7.1.2.1. ЯКЩО всі коефіцієнти в рядку дорівнюють  $\approx 0$  ТА правий бік  $\neq 0$  ТО ПОВЕРНУТИ «Систему неможливо вирішити»
8. Зворотний хід (обернене підстановлення)
  - 8.1. Ініціалізувати масив *roots* довжиною *n*.
  - 8.2. ЦИКЛ *row* від 0 до *n* – 1: Множимо обернену матрицю на вектор вільних членів
    - 8.2.1. ЦИКЛ *col* від 0 до *n* – 1
      - 8.2.1.1.  $roots[i] = roots[i] + inverseMatrix[row][col] * Matrix[row][n]$
  - 8.3. ПОВЕРНУТИ *roots*

Рисунок Опис програмного забезпечення.1 – Діаграма класів

## 4.2 Опис методів частин програмного забезпечення

### 1.1.1 Стандартні методи

У таблиці 4.1 наведено методи, що надаються мовою програмування, бібліотекою, чи фреймворком, що були використані задля виконання курсової роботи

Таблиця 4.1 – Стандартні методи

№ п/п	Назва методу, функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
	React.useState	Встановлення змінної для подальшого використання, що викликає ререндеринг при зміні	Початкове значення стану	Значення стану та функція для реактивного оновлення
	React.useEffect	Виконання дії за зміни значень, викладених за допомогою useState	функція, що викликається при зміні параметрів, масив параметрів	Відсутні
	React.useRef	Створення контейнера, що зберігає значення (як змінна)		

### 1.1.2 Користувацькі методи

У таблиці 4.2 наведено ...

Таблиця 4.2 – Користувацькі методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл

## 5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 5.1 План тестування

Мета плану – перевірити працездатність роботи UI та правильність та достатню точність реалізованих методів розв’язання СЛАР. Досліджено на здатність розробленого програмного забезпечення до виняткових ситуацій.

Покладемо:

$$\varepsilon_{min} = 10^{-6}, \varepsilon_{max} = 10^6, size_{max} = 1000, size_{min} = 1$$

- 1) Тестування правильності введених значень.
  - 1) Введення некоректних символів у поля розміру СЛАР
  - 2) Введення значень у клітинки матриці, точність яких неможливо гарантувати (не в межах по модулю від  $\varepsilon_{min}$  до  $\varepsilon_{max}$ )
  - 3) Введення нецілого розміру матриці.
  - 4) Введення неприпустимого розміру матриці (менше  $size_{min}$  більше  $size_{max}$ )
  - 5) Введення некоректних символів у поля діапазону для генерації СЛАР.
  - 6) Введення значень, точність яких неможливо гарантувати (не в межах по модулю від  $\varepsilon_{min}$  до  $\varepsilon_{max}$ )
  - 7) Введення некоректного діапазону для генерації СЛАР («від» більше-рівне «до»)
- 2) Перевизначення розмірів матриці
  - 1) При введенні меншого розміру обрізає СЛАР (лишаючи певні коефіцієнти)
  - 2) При введенні більшого розміру розширює СЛАР
- 3) Очищення матриці
  - 1) Очищує
  - 2) Не обнуляє матрицю при методі, що працює.
- 4) Тестування генерації
  - 1) Генерація СЛАР з коректними значеннями генерації призведе до

відображення згенерованих значень у СЛАР у заданому користувачем діапазоні.

2) Неможливо згенерувати під час проходження алгоритму.

5) Тестування коректності роботи методів розв'язання СЛАР

1) Для кожного методу розв'язання СЛАР:

5.1.1. Знаходження коренів СЛАР, що унікальні та існують.

5.1.2. Знаходження розв'язку СЛАР, що не має розв'язків

5.1.3. Знаходження розв'язку СЛАР з нескінченною кількістю розв'язків.

2) Лише для методу оберненої матриці:

5.2.1. Перевірка коректності отриманих матриць коефіцієнтів та оберненої матриці.

6) Тестування коректності роботи переходу між кроками розв'язання СЛАР.

6.1.1. Натискання кнопки «Старт» розпочинає роботу методу.

6.1.2. Натискання кнопки «Стоп» зупиняє роботу методу.

6.1.3. Зміна напрямку

6.1.3.1. Назад

6.1.3.2. Вперед

6.1.3.3. Вперед-назад двічі

6.1.4. Встановлення швидкості

6.1.4.1. Під час роботи методу

6.1.4.2. Поча роботою методу

6.1.5. Натискання кнопки переходу на один крок.

6.1.5.1. При напрямку вперед

6.1.5.2. При напрямку назад

6.1.6. Перехід до кінця напрямку

6.1.6.1. При напрямку вперед

6.1.6.2. При напрямку назад

#### 6.1.7. Перехід на початок («reset»)

##### 6.1.7.1. При русі

##### 6.1.7.2. Поза рухом

6.1.8. Неможливість зміни значень СЛАР під час виконання або за наявності вже виконаних кроків у методі.

6.1.9. При чинному розв'язанні СЛАР методом оберненої матриці, можливо вільно переглядати (але не змінювати) матрицю коефіцієнтів та обернену матрицю.

#### 7) Редагування СЛАР

- 1) Неможливо, змінити, вводячи некоректні значення.
- 2) Перевірка дійсності зміни СЛАР шляхом запуску методу.
- 3)

#### 8) Файлова система

- 1) Імпортування коректної СЛАР з коректним роздільником із CSV
- 2) Імпортування некоректної СЛАР з CSV
- 3) Експортування чинної СЛАР як матриці у форматі CSV.

#### 9) Текстові операції

- 1) Копіювання матриці користувачем
- 2) Вставлення правильно відформатованої СЛАР (з певним роздільником)
- 3) Вставлення неправильно відформатованої СЛАР.

#### 10) Зміна теми

- 1) Змінити тему на світлу
- 2) Змінити тему на темну
- 3) Змінити тему на системну

#### 11) Обрання режиму програми

- 1) Перехід до режиму побудови графіків
- 2) Перехід до режиму розв'язання СЛАР



## 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
Вхідні дані	
Схема проведення тесту	Поелементне заповнення матриці коефіцієнтів
Очікуваний результат	Повідомлення про помилку формату даних
Стан програми після проведення випробувань	Видано помилку «Введіть дійсне число»

## 6 ІНСТРУКЦІЯ КОРИСТУВАЧА

### 6.1 Робота з програмою

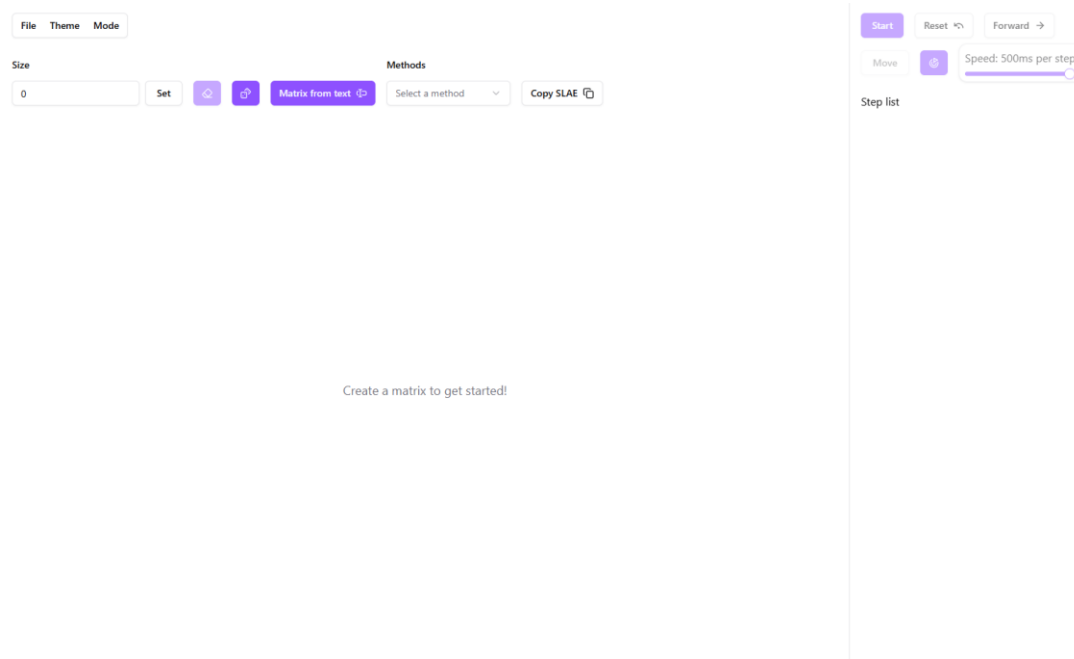


Рисунок 6.1 – Головне вікно програми

Далі за допомогою лічильника з назвою «Size» шляхом введення числа з клавіатури необхідно виставити розмір системи, що буде оброблятися програмою та натиснути на кнопку «Set» (рисунок 6.2):

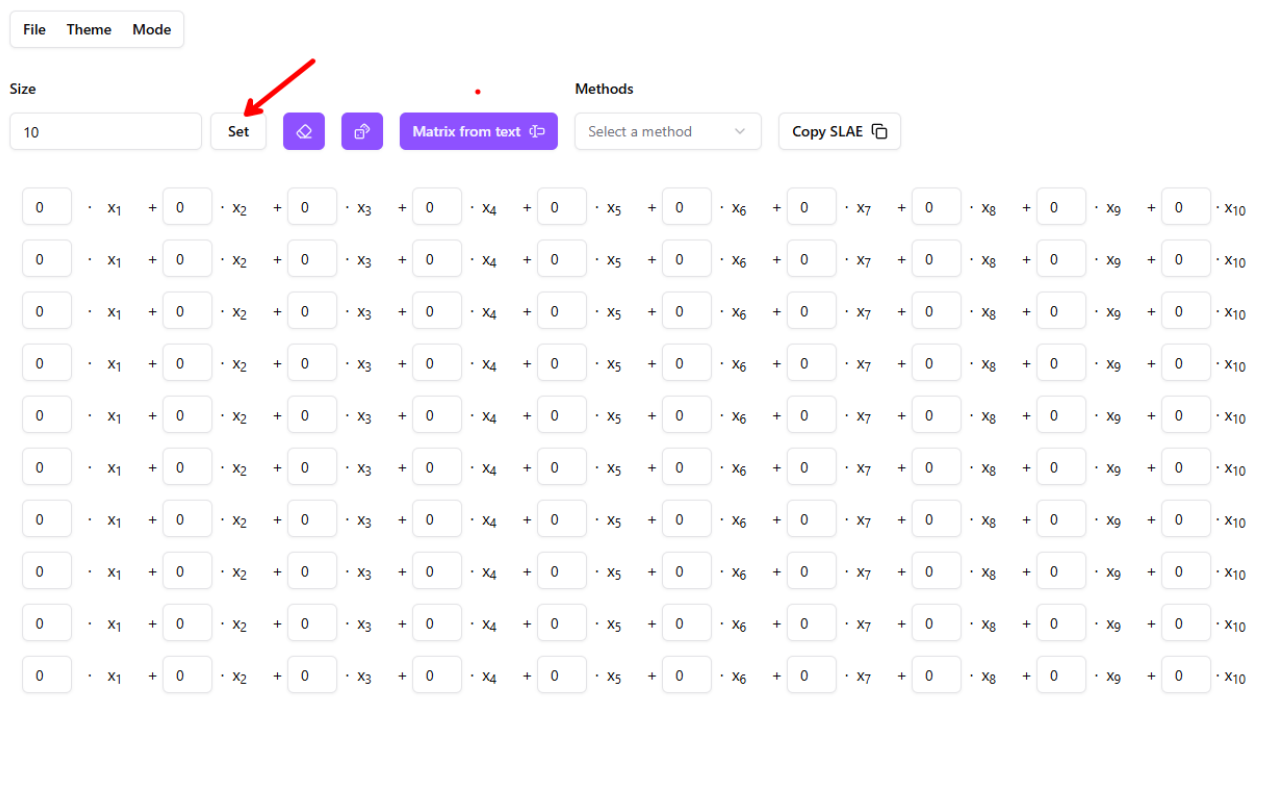
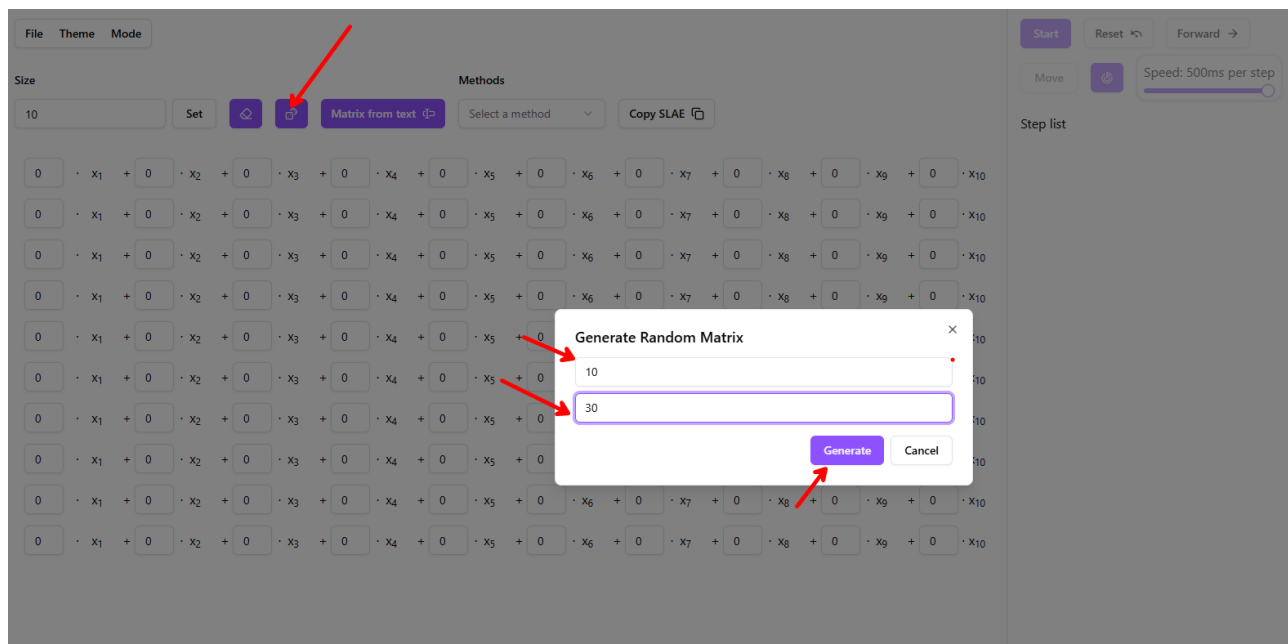





Рисунок 6.2 – Вибір необхідного розміру системи



Користувач може замостійно ввести або випадково згенерувати матрицю натисканням на іконку у вигляді гральних кісток і ввівши необхідні дані для генерації (початок та кінець)



Згодом користувач обирає певний метод із зазначених:

File Theme Mode

Size: 10 Set   Matrix from text 




Methods: Select a method  Copy SLAE 

Method: Gauss, Gauss-Jordan, Inverse Matrix


26.58020 ·x<sub>1</sub> + 10.41575 ·x<sub>2</sub> + 19.65189 ·x<sub>3</sub> + 23.51490 ·x<sub>4</sub> + 14.51917 ·x<sub>5</sub> + 27.29979 ·x<sub>6</sub> + 18.43475 ·x<sub>7</sub> + 27.04800 ·x<sub>8</sub> + 17.07028 ·x<sub>9</sub> + 17.50553 ·x<sub>10</sub> = 19.51537 ·x<sub>1</sub> + 15.70818 ·x<sub>2</sub> + 24.24211 ·x<sub>3</sub> + 22.53265 ·x<sub>4</sub> + 11.48887 ·x<sub>5</sub> + 24.82815 ·x<sub>6</sub> + 25.34045 ·x<sub>7</sub> + 24.91994 ·x<sub>8</sub> + 17.69133 ·x<sub>9</sub> + 26.71829 ·x<sub>10</sub> = 26.50843 ·x<sub>1</sub> + 12.83304 ·x<sub>2</sub> + 23.52060 ·x<sub>3</sub> + 21.64367 ·x<sub>4</sub> + 14.92560 ·x<sub>5</sub> + 20.66876 ·x<sub>6</sub> + 12.35236 ·x<sub>7</sub> + 28.20976 ·x<sub>8</sub> + 20.70887 ·x<sub>9</sub> + 13.10172 ·x<sub>10</sub> = 14.04215 ·x<sub>1</sub> + 16.35923 ·x<sub>2</sub> + 10.45136 ·x<sub>3</sub> + 12.03691 ·x<sub>4</sub> + 13.93308 ·x<sub>5</sub> + 14.26942 ·x<sub>6</sub> + 15.38021 ·x<sub>7</sub> + 29.50704 ·x<sub>8</sub> + 13.43761 ·x<sub>9</sub> + 10.85429 ·x<sub>10</sub> = 15.16417 ·x<sub>1</sub> + 11.50464 ·x<sub>2</sub> + 26.66397 ·x<sub>3</sub> + 26.52260 ·x<sub>4</sub> + 24.94776 ·x<sub>5</sub> + 17.04657 ·x<sub>6</sub> + 27.54957 ·x<sub>7</sub> + 27.33823 ·x<sub>8</sub> + 16.78831 ·x<sub>9</sub> + 25.68307 ·x<sub>10</sub> = 14.13167 ·x<sub>1</sub> + 20.76610 ·x<sub>2</sub> + 11.17416 ·x<sub>3</sub> + 18.09166 ·x<sub>4</sub> + 23.12222 ·x<sub>5</sub> + 16.05254 ·x<sub>6</sub> + 27.08051 ·x<sub>7</sub> + 11.50806 ·x<sub>8</sub> + 17.98263 ·x<sub>9</sub> + 12.94574 ·x<sub>10</sub> = 22.60295 ·x<sub>1</sub> + 16.28067 ·x<sub>2</sub> + 22.62899 ·x<sub>3</sub> + 13.13524 ·x<sub>4</sub> + 28.74822 ·x<sub>5</sub> + 15.65586 ·x<sub>6</sub> + 18.10914 ·x<sub>7</sub> + 14.08341 ·x<sub>8</sub> + 24.80927 ·x<sub>9</sub> + 18.58511 ·x<sub>10</sub> = 26.55101 ·x<sub>1</sub> + 23.87904 ·x<sub>2</sub> + 27.70544 ·x<sub>3</sub> + 25.63244 ·x<sub>4</sub> + 27.21282 ·x<sub>5</sub> + 19.55448 ·x<sub>6</sub> + 25.84939 ·x<sub>7</sub> + 27.50051 ·x<sub>8</sub> + 17.37749 ·x<sub>9</sub> + 27.81693 ·x<sub>10</sub> = 26.06094 ·x<sub>1</sub> + 16.96974 ·x<sub>2</sub> + 20.27712 ·x<sub>3</sub> + 13.13611 ·x<sub>4</sub> + 21.29277 ·x<sub>5</sub> + 12.30317 ·x<sub>6</sub> + 29.70951 ·x<sub>7</sub> + 12.63299 ·x<sub>8</sub> + 27.70394 ·x<sub>9</sub> + 20.97759 ·x<sub>10</sub> = 13.95238 ·x<sub>1</sub> + 24.40616 ·x<sub>2</sub> + 27.06736 ·x<sub>3</sub> + 28.91436 ·x<sub>4</sub> + 16.16062 ·x<sub>5</sub> + 12.12912 ·x<sub>6</sub> + 12.69069 ·x<sub>7</sub> + 24.56332 ·x<sub>8</sub> + 15.22532 ·x<sub>9</sub> + 14.39970 ·x<sub>10</sub> =




Користувач має змогу розпочинати та зупиняти покрокове проходження натисненням кнопки «Start/Stop»:


File Theme Mode

Size: 10 Set   Matrix from text 

Methods: Gauss, Gauss-Jordan, Inverse Matrix

Copy SLAE 

Start  Reset  Forward 

Move  Speed: 500ms per step

Step list

26.58020 ·x<sub>1</sub> + 10.41575 ·x<sub>2</sub> + 19.65189 ·x<sub>3</sub> + 23.51490 ·x<sub>4</sub> + 14.51917 ·x<sub>5</sub> + 27.29979 ·x<sub>6</sub> + 18.43475 ·x<sub>7</sub> + 27.04800 ·x<sub>8</sub> + 17.07028 ·x<sub>9</sub> + 17.50553 ·x<sub>10</sub> = 19.51537 ·x<sub>1</sub> + 15.70818 ·x<sub>2</sub> + 24.24211 ·x<sub>3</sub> + 22.53265 ·x<sub>4</sub> + 11.48887 ·x<sub>5</sub> + 24.82815 ·x<sub>6</sub> + 25.34045 ·x<sub>7</sub> + 24.91994 ·x<sub>8</sub> + 17.69133 ·x<sub>9</sub> + 26.71829 ·x<sub>10</sub> = 26.50843 ·x<sub>1</sub> + 12.83304 ·x<sub>2</sub> + 23.52060 ·x<sub>3</sub> + 21.64367 ·x<sub>4</sub> + 14.92560 ·x<sub>5</sub> + 20.66876 ·x<sub>6</sub> + 12.35236 ·x<sub>7</sub> + 28.20976 ·x<sub>8</sub> + 20.70887 ·x<sub>9</sub> + 13.10172 ·x<sub>10</sub> = 14.04215 ·x<sub>1</sub> + 16.35923 ·x<sub>2</sub> + 10.45136 ·x<sub>3</sub> + 12.03691 ·x<sub>4</sub> + 13.93308 ·x<sub>5</sub> + 14.26942 ·x<sub>6</sub> + 15.38021 ·x<sub>7</sub> + 29.50704 ·x<sub>8</sub> + 13.43761 ·x<sub>9</sub> + 10.85429 ·x<sub>10</sub> = 15.16417 ·x<sub>1</sub> + 11.50464 ·x<sub>2</sub> + 26.66397 ·x<sub>3</sub> + 26.52260 ·x<sub>4</sub> + 24.94776 ·x<sub>5</sub> + 17.04657 ·x<sub>6</sub> + 27.54957 ·x<sub>7</sub> + 27.33823 ·x<sub>8</sub> + 16.78831 ·x<sub>9</sub> + 25.68307 ·x<sub>10</sub> = 14.13167 ·x<sub>1</sub> + 20.76610 ·x<sub>2</sub> + 11.17416 ·x<sub>3</sub> + 18.09166 ·x<sub>4</sub> + 23.12222 ·x<sub>5</sub> + 16.05254 ·x<sub>6</sub> + 27.08051 ·x<sub>7</sub> + 11.50806 ·x<sub>8</sub> + 17.98263 ·x<sub>9</sub> + 12.94574 ·x<sub>10</sub> = 22.60295 ·x<sub>1</sub> + 16.28067 ·x<sub>2</sub> + 22.62899 ·x<sub>3</sub> + 13.13524 ·x<sub>4</sub> + 28.74822 ·x<sub>5</sub> + 15.65586 ·x<sub>6</sub> + 18.10914 ·x<sub>7</sub> + 14.08341 ·x<sub>8</sub> + 24.80927 ·x<sub>9</sub> + 18.58511 ·x<sub>10</sub> = 26.55101 ·x<sub>1</sub> + 23.87904 ·x<sub>2</sub> + 27.70544 ·x<sub>3</sub> + 25.63244 ·x<sub>4</sub> + 27.21282 ·x<sub>5</sub> + 19.55448 ·x<sub>6</sub> + 25.84939 ·x<sub>7</sub> + 27.50051 ·x<sub>8</sub> + 17.37749 ·x<sub>9</sub> + 27.81693 ·x<sub>10</sub> = 26.06094 ·x<sub>1</sub> + 16.96974 ·x<sub>2</sub> + 20.27712 ·x<sub>3</sub> + 13.13611 ·x<sub>4</sub> + 21.29277 ·x<sub>5</sub> + 12.30317 ·x<sub>6</sub> + 29.70951 ·x<sub>7</sub> + 12.63299 ·x<sub>8</sub> + 27.70394 ·x<sub>9</sub> + 20.97759 ·x<sub>10</sub> = 13.95238 ·x<sub>1</sub> + 24.40616 ·x<sub>2</sub> + 27.06736 ·x<sub>3</sub> + 28.91436 ·x<sub>4</sub> + 16.16062 ·x<sub>5</sub> + 12.12912 ·x<sub>6</sub> + 12.69069 ·x<sub>7</sub> + 24.56332 ·x<sub>8</sub> + 15.22532 ·x<sub>9</sub> + 14.39970 ·x<sub>10</sub> =

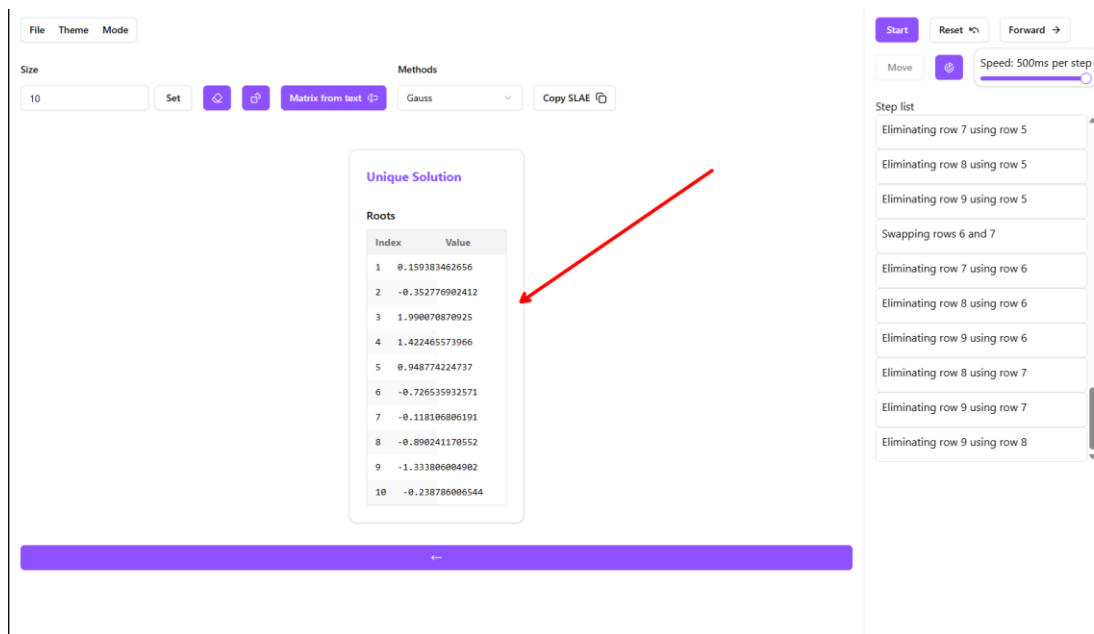
Проходження в такому випадку йтиме за таймером, зі швидкістю, зазначеною в «Speed» слайдері.

Ось приклад того, як може закінчитися покроковий обхід:

Результатом виконання програми є покрокове проходження заданої СЛАР, який видається у вигляді таблиці кожне число якої записане з точністю до 6-х знаків після коми.

Нині користувачу стає доступним перехід до панелі розв'язків, у якій і міститься рішення цієї системи.

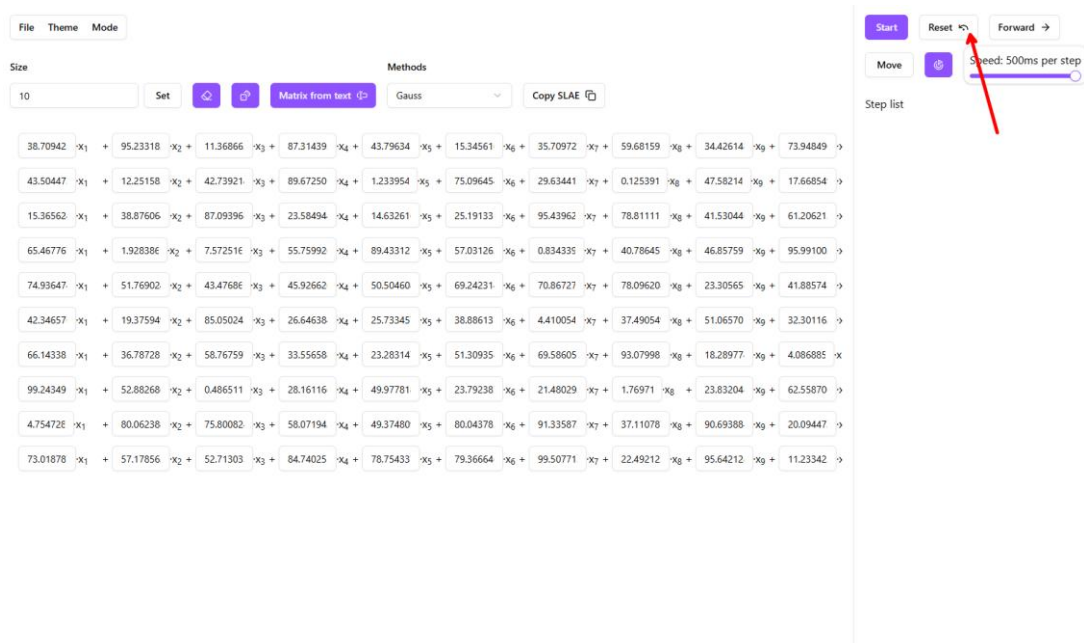
Ось результат переходу натисканням на кнопку. У випадку, коли унікальних розв'язків не існує, виведеться відповідне повідомлення.



The screenshot shows the software interface with a 'Unique Solution' window open. The window displays a table of roots with 10 rows and 2 columns: 'Index' and 'Value'. A red arrow points to the 'Value' column.

Index	Value
1	0.159383462656
2	-0.352776902412
3	1.990070870925
4	1.422465573966
5	0.948774224737
6	-0.726535932571
7	-0.118186806191
8	-0.890241170552
9	-1.333806004902
10	-0.238786006544

Розпочати з початкової матриці можна натисканням кнопки «Reset»



The screenshot shows the software interface with the 'Reset' button highlighted by a red arrow. The main window displays a large grid of numerical values, likely representing the initial matrix.

Рис. 6.2 Результат натискання кнопки «Reset»

Іще однією важливою особливістю програми є можливість зміни напрямку проходження кроків.

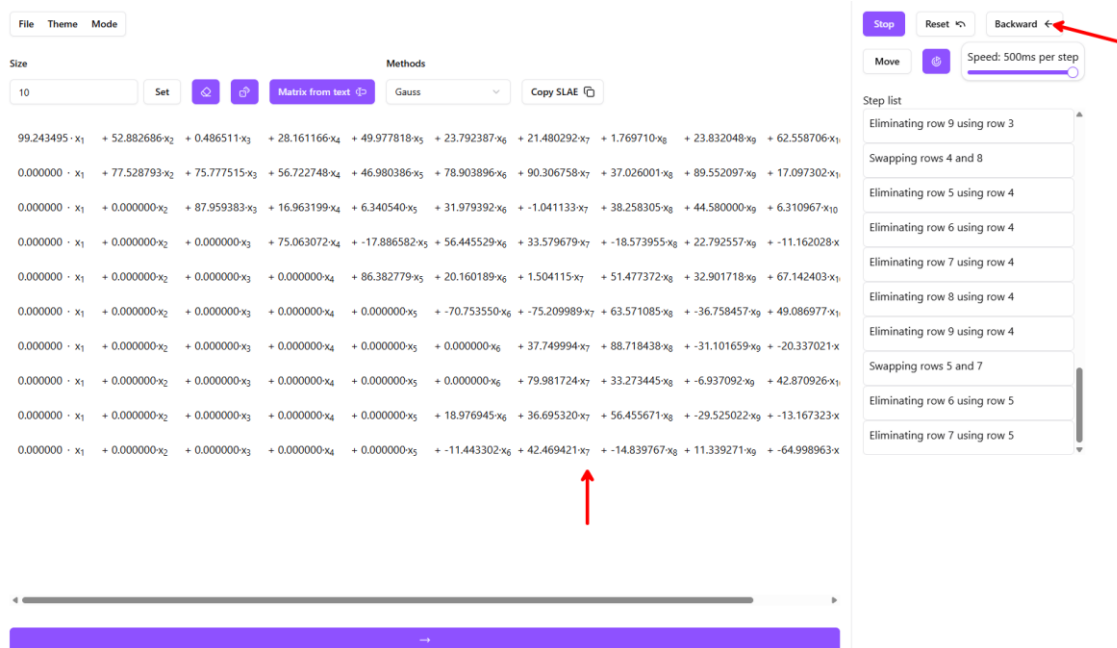
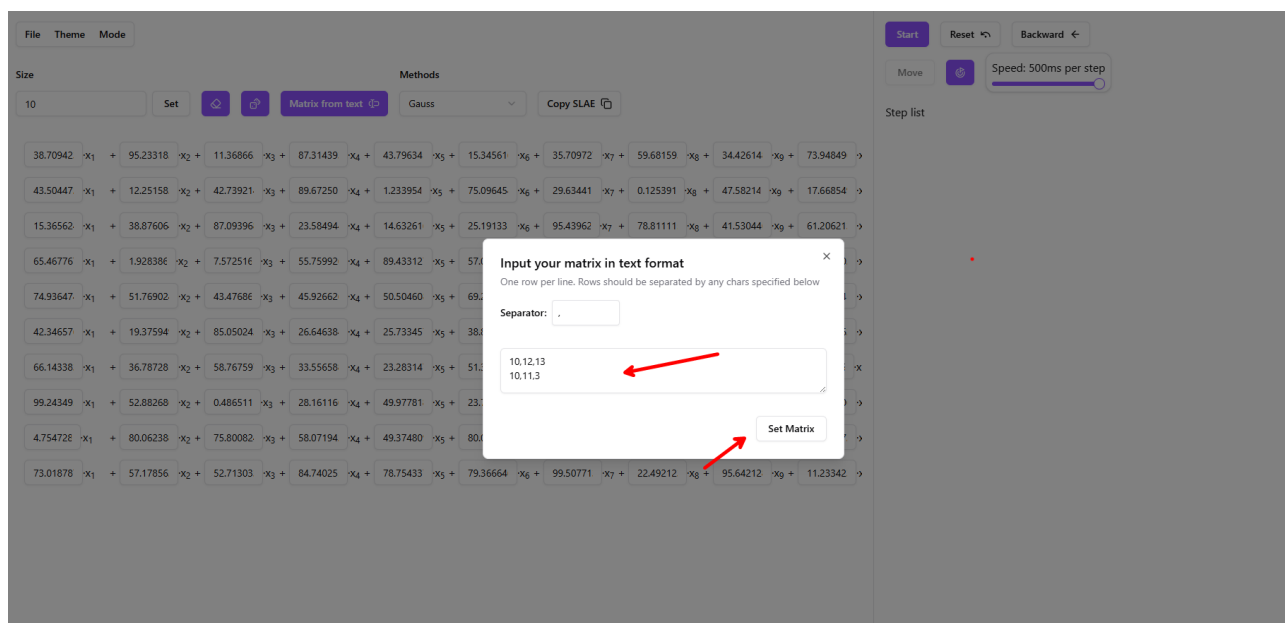


Рис. 6.2. Відбуваються зворотні зміни в попередньо трикутній матриці СЛАР.

Також можливе перетворення в СЛАР з текстового вигляду шляхом оброблення тексту з роздільниками (CSV-подібно)



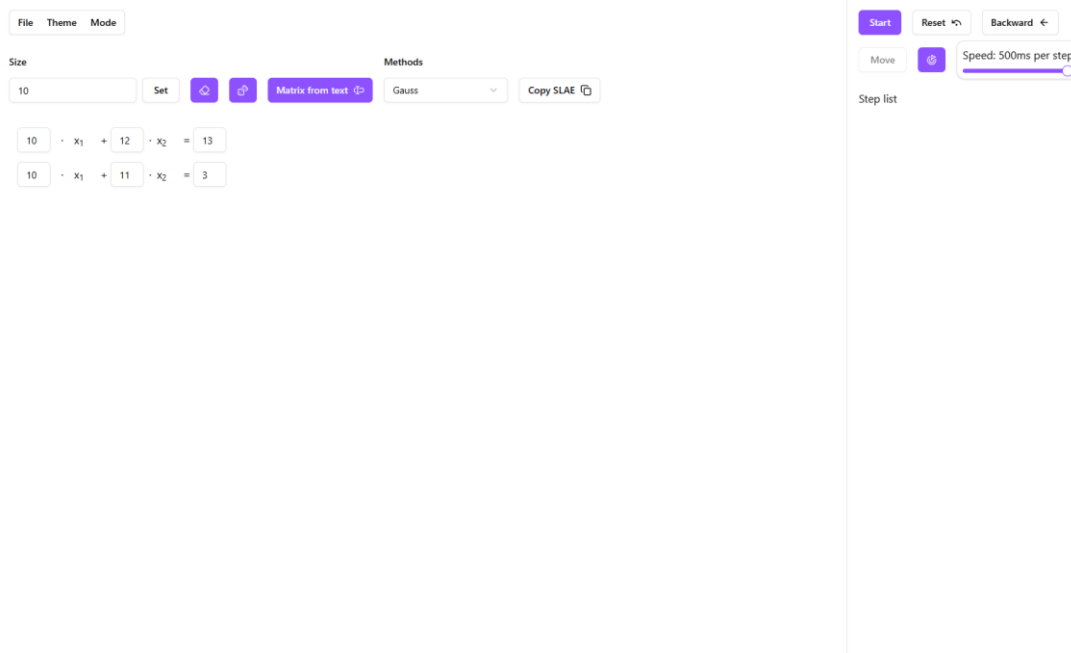
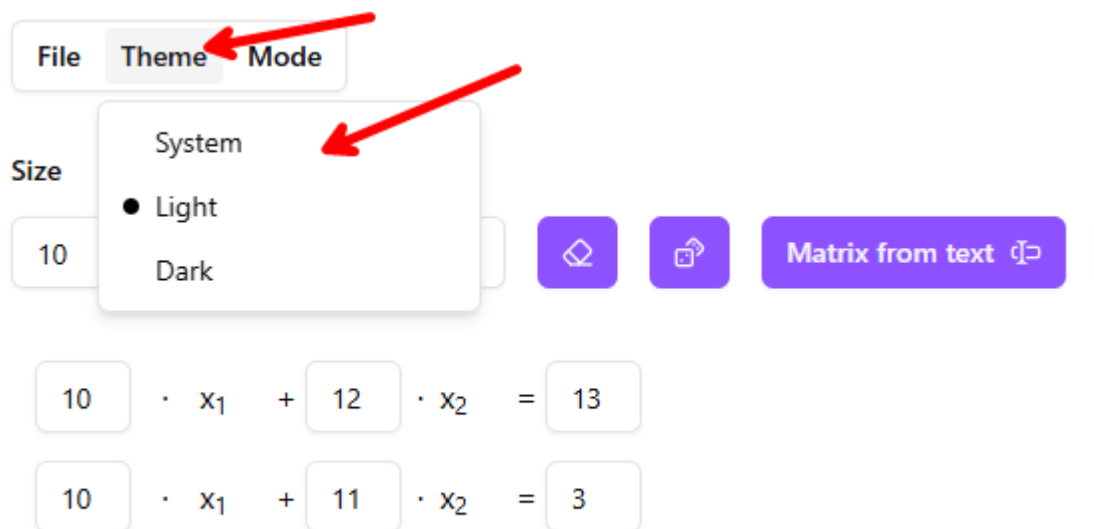
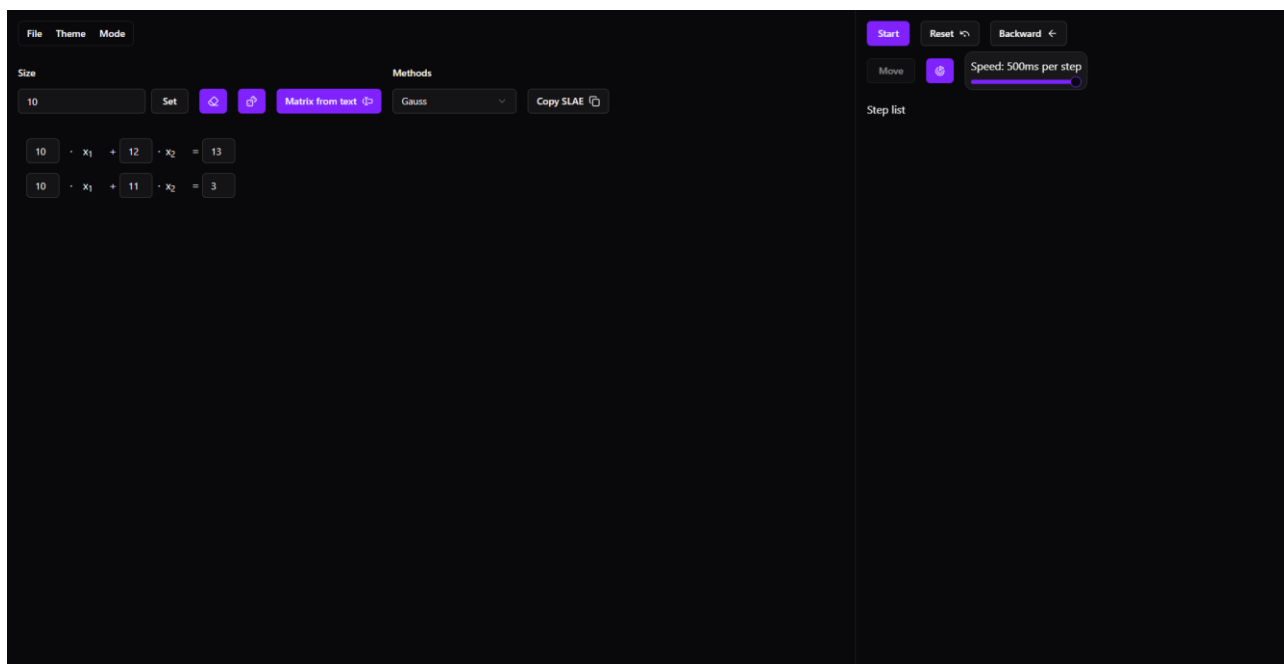


Рис. 6.2. Успішне виконання операції

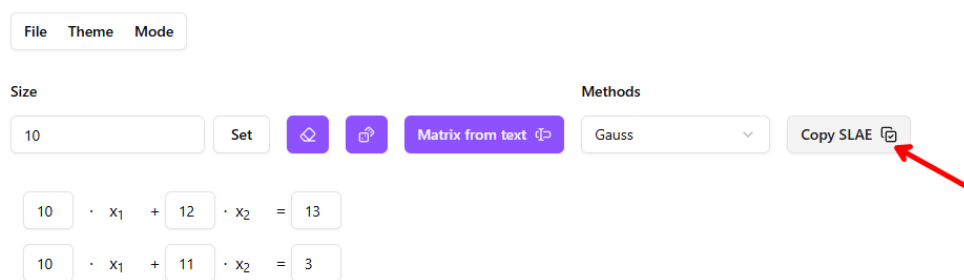
Якщо користувачу більше імпонує темна тема, він без проблем може обрати.



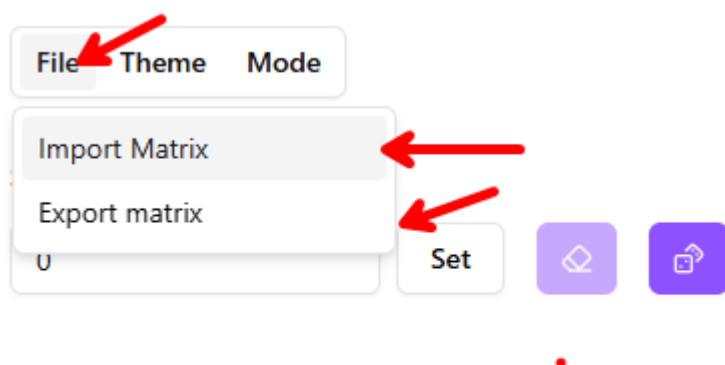




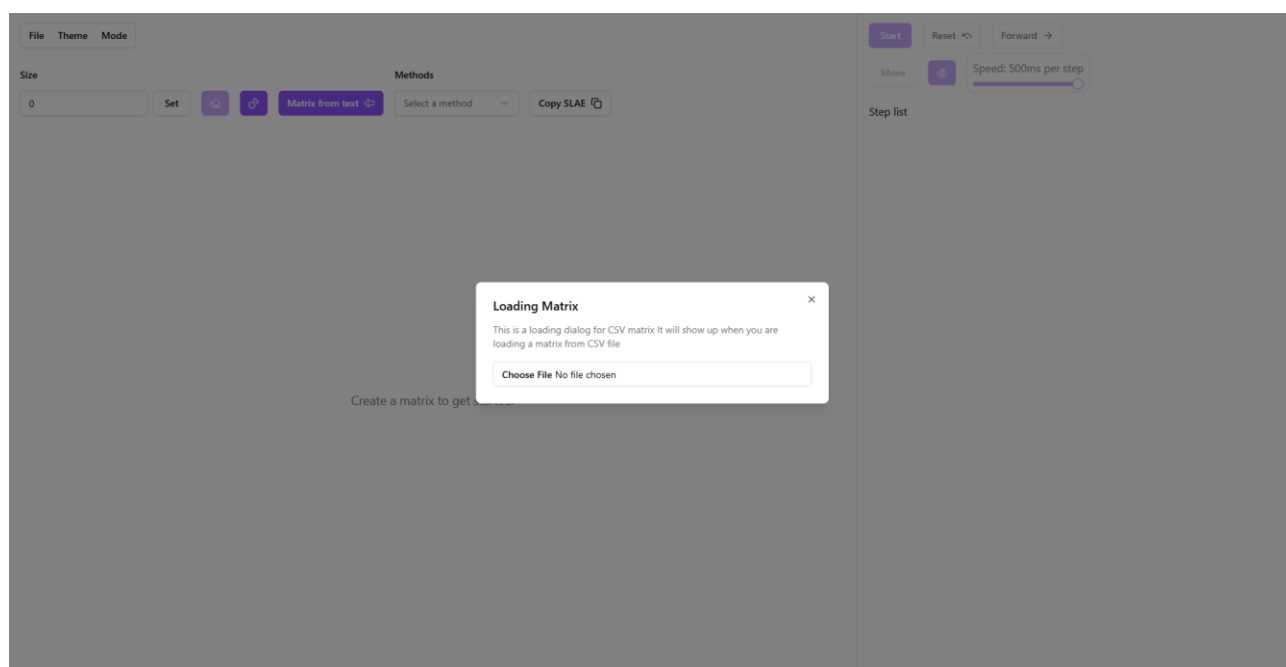
Додатково користувач може скопіювати матрицю до буфера обміну, у форматі CSV:



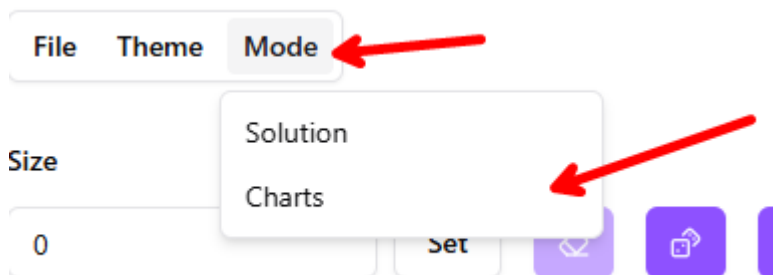
Користувач також може імпортувати та експортувати матриці через меню,



обравши файл у діалозі, що з'явиться



Також, користувач, суто у навчальних цілях, може перейти в режим графіків.





Алгоритм переходу в режим графіків


У цьому режимі користувач може обрати методи та розміри, що бажає дослідити з алгоритмічної точки зору. Виведеться статистика стосовно 3 характеристик цих алгоритмів.

File Theme Mode


Methods


Gauss 


Gauss-Jordan 

Inverse Matrix 

Sizes

10 

100 

Select sizes 

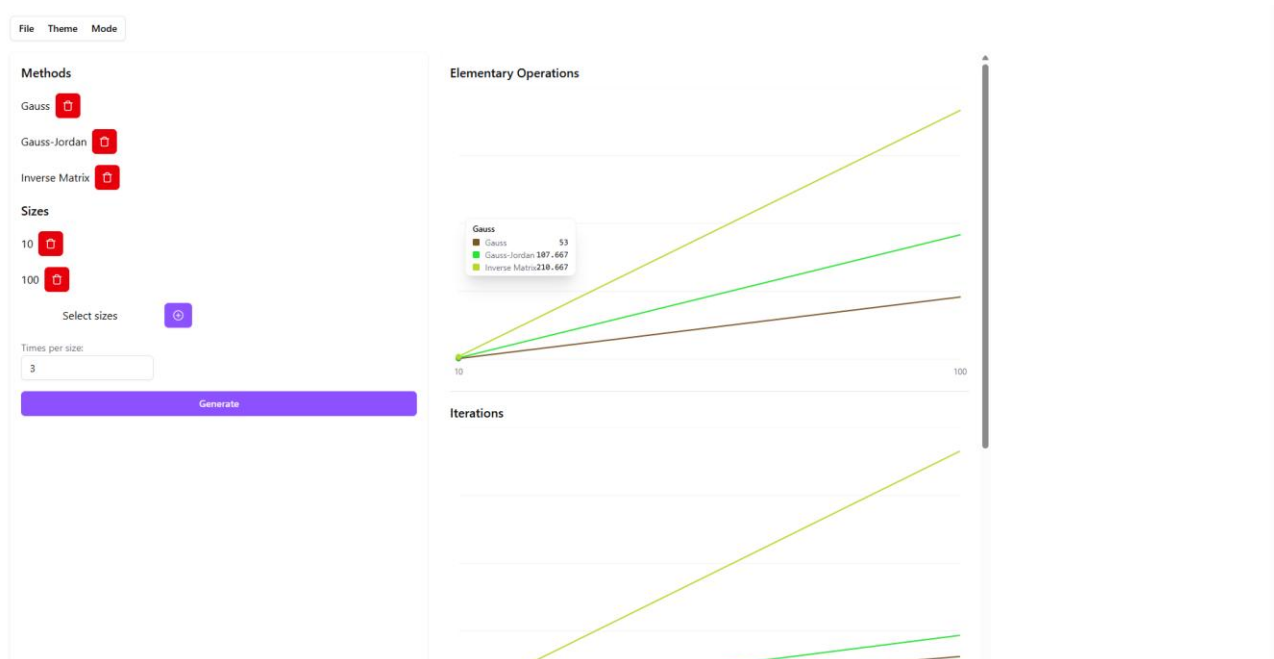
Times per size:

3

Generate

Вікно вибору методів та розмірів (можна обрати шляхом натискання на «Select Sizes» або «Select methods», якщо прибрати деякі методи.

Після натискання кнопки «Generate» побачимо графіки за різними характеристиками.



## 6.2 Формат вхідних та вихідних даних

Користувачем на вхід програми подається СЛАР у форматі розширеної матриці, тобто задається за допомогою матриці системи та стовпця вільних членів, числа яких дійсні з точністю не більше, ніж 6 знаків після коми (якщо точність більша, то програма видасть повідомлення користувачу про неможливість обрання заданої точності).

## 6.3 Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги програмного забезпечення

	Мінімальні	Рекомендовані
Підтримувані браузери	Internet Explorer 11 Microsoft Edge (версії 80+) Google Chrome 49+ Mozilla Firefox 52+	Microsoft Edge (останні версії) Google Chrome (останні версії) Mozilla Firefox (останні

		версії) Safari 13+
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Core i3 або AMD Ryzen 3
Оперативна пам'ять	2 GB RAM	4 GB RAM або більше
Відеоадаптер	Вбудований відеоадаптер з підтримкою WebGL / дискретна відеокарта	
Дисплей	1024x768	1366×768 або вище

Продовження таблиці 6.1

	Мінімальні	Рекомендовані
Прилади введення	Клавіатура, комп'ютерна миша	
Додаткове програмне забезпечення	Сучасні браузері з повною підтримкою ESNext React 18+ Zustand 4+ Comlink останніх версій Shadcn/UI для якісного UI Підтримка Web Workers для асинхронних завдань	

## 7 АНАЛІЗ РЕЗУЛЬТАТІВ

Головною задачею курсової роботи була реалізація програми для розв’язання СЛАР наступними методами: Якобі, ...

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, ретельно перевіряються на валідність і лише потім подаються на обробку програмі.

Для перевірки та доведення достовірності результатів виконання програмного забезпечення скористаюся MS Excel:

а) Метод Якобі.

Результат виконання методу Якобі наведено на рисунку 7.1:

Рисунок 7.1 – Результат виконання методу Якобі

Оскільки результат виконання збігається з результатом в MS Excel (рисунок 7.2), то даний метод працює вірно.

	A	B	C	D	E	F	G	H
1								
2		Матриця системи			Стовпець вільних членів		Рішення системи	
3			3	1,4		14,06		5,363
4			10	23,457		19,65		-1,448
5								

Рисунок 7.2 – Перевірка методу Якобі в MS Excel 2010

б) Метод Гауса-Зейделя.

...

Для проведення тестування ефективності програми було створено матриці наступного вигляду:

$$\begin{pmatrix} 2n & 01 & 01 & 01 & \dots & 01 \\ 01 & 2n & 01 & 01 & \dots & 01 \\ 01 & 01 & 2n & 01 & \dots & 01 \\ 0: & 0: & 0: & 0: & \vdots & 0: \\ 01 & 01 & 01 & 01 & \dots & 2n \end{pmatrix} (7.1),$$

де  $n$  – розмірність системи.

Матриця (7.1) для довільного додатного  $n$  є симетричною, додатньо визначеною та має домінантну головну діагональ.

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

**! Це лише приклад**

**Таблиця 7.1 – Тестування ефективності методів**

Розмірність системи	Параметри тестування	Метод		
		Якобі	Гауса-Зейделя	Градінтного спуску
1000	Кількість ітерацій			
	Кількість елементарних операцій (млн.)			
	Власний критерій			
2500	Кількість ітерацій			
	Кількість елементарних операцій (млн.)			
5000	Кількість ітерацій			
	Кількість елементарних операцій (млн.)			
10000	Кількість ітерацій			
	Кількість елементарних операцій (млн.)			
15000	Кількість ітерацій			

	Кількість елементарних операцій (млн.)			

Візуалізація результатів таблиці 7.1 наведено на рисунку 7.1:

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

Необхідні графіки по усіх критеріях.

За результатами тестування можна зробити такі висновки:

а) Всі розглянуті методи дозволяють знаходити розв'язки великих та надвеликих СЛАР.

б) Складність всіх розглянутих методів є квадратичною, тобто –  $O(k * n^2)$ , де  $k$  – кількість ітерацій виконаних методом,  $n$  – розмір СЛАР.

в) З розглянутих методів найоптимальнішим для практичного використання є метод Гауса-Зейделя, оскільки він виконується найшвидше та має такі умови сходимості, що охоплюють найширший спектр СЛАР.



## **ВИСНОВКИ**

Коротко описати, що було виконано в кожному розділі. Можливо описати шляхи покращення розробленого програмного забезпечення.

## **ПЕРЕЛІК ПОСИЛАНЬ**

**ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ**

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра  
інформатики та програмної інженерії

Затвердив

Керівник Головченко М.М.

«» \_\_\_\_\_ 202\_ р.

Виконавець:

Студент <ПІБ>

«\_\_» \_\_\_\_\_ 202\_ р.

**ТЕХНІЧНЕ ЗАВДАННЯ**

на виконання курсової роботи

на тему: «Розв’язання СЛАР точними методами»

з дисципліни:

«Основи програмування. Курсова робота»

1. *Мета:* Метою курсової роботи є забезпечення коректного розв'язання СЛАР точними методами та унаочнення процесу знаходження розв'язків СЛАР.
2. *Дата початку роботи:* «1» квітня 2025 р.
3. *Дата закінчення роботи:* «25» травня 2025 р.
4. *Вимоги до програмного забезпечення.*

1) Функціональні вимоги:

- Можливість введення розмірів матриці
- Можливість введення даних у матрицю
- Можливість вставлення елементів матриці, написаних через певний роздільник
- Можливість випадкового створення будь-якої матриці для СЛАР
- Можливість випадкового створення матриці для СЛАР, що має розв'язки.
- Можливість завантаження матриці з файлу CSV
- Можливість експорту матриці до файлу CSV
- Забезпечення правильного розрахунку розв'язків СЛАР (у тому числі виявлення розв'язків із нескінченною кількістю членів та СЛАР, що розв'язків не мають)
- Можливість вибору методу розв'язання СЛАР з трьох варіантів:
  - o метод Гауса (за замовчуванням)
  - o метод Жордана-Гауса
  - o матричний метод (за допомогою оберненої матриці)
- Можливість перевірки допустимості значення, що вводяться у матрицю (заборона вводу буквених виразів, великих чисел, чисел із плаваючою точкою, точність яких неможливо гарантувати).
- Можливість візуалізації кроків розв'язання СЛАР, з додатковими поясненнями.

- Можливість руху вперед та назад вперед і назад кроками розв'язання матриці
- Можливість моментального розв'язання СЛАР (без перегляду кроків розв'язання).
- Можливість зміни теми програми (світла/темна)
- Можливість зміни мови програми (українська/англійська)

2) Нефункціональні вимоги:

- Підтримка браузерів з движками Chromium та Gecko.
- Використовуваний обсяг пам'яті < 20 Мб при незначних розмірах СЛАР.
- Код на Typescript та React
- Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:  
ДСТУ 3008 - 2015 - Розробка технічної документації.

5. *Стадії та етапи розробки програмного забезпечення:*

- 1) Розробка алгоритмічної складової програмного забезпечення (до \_\_.\_\_.202\_р.)
- 2) Об'єктно-орієнтований аналіз предметної області завдання (до \_\_.\_\_.202\_р.)
- 3) Об'єктно-орієнтоване проєктування програмного забезпечення (до \_\_.\_\_.202\_р.)
- 4) Розробка програмного забезпечення (до \_\_.\_\_.202\_р.)
- 5) Тестування розробленого програмного забезпечення (до \_\_.\_\_.202\_р.)
- 6) Демонстрація та захист програмного забезпечення (до \_\_.\_\_.202\_р.)

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  
обернення матриць

---

(Найменування програми (документа))

*GitHub репозиторій*

---

(Вид носія даних)

**182 арк, 124 Кб**

---

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

студента групи ІП-XX / курсу

**Гуменського В.Л.**

**action-sidebar.tsx**

```

import { useMatrixStore } from "@/store/matrix";
import { useSolutionRunner } from "../../hooks/use-solution-runner";
import { useSolutionStore } from "@/store/solution";
import { useState, useEffect } from "react";
import { toast } from "sonner";
import StepControls from "../step-controls";
import StepList from "../step-list";
import { useInterval } from "../../hooks/use-interval";
import type { Direction } from "../action";
import { Skeleton } from "../ui/skeleton";
import {
  Drawer,
  DrawerTrigger,
  DrawerContent,
  DrawerHeader,
  DrawerTitle,
  DrawerFooter,
  DrawerClose,
} from "../ui/drawer";
import { Menu } from "lucide-react";
import { Button } from "../ui/button";

export default function ActionSidebar({
  showStepListInDrawer = false,
}: { showStepListInDrawer?: boolean } = {}) {
  const slae = useMatrixStore((s) => s.slae);
  const matrixState = useMatrixStore((s) => s.matrixConfiguration);
  const setMatrix = useMatrixStore((s) => s.setMatrixConfiguration);
  const setLoadingMatrix = useMatrixStore((s) => s.setIsLoadingMatrix);
  const method = useSolutionStore((s) => s.method);
  const result = useSolutionStore((s) => s.solutionResult);
  const setResult = useSolutionStore((s) => s.setSolutionResult);
  const setCurrentTargetRow = useMatrixStore((s) => s.setCurrentTargetRow);

  const setIsActive = useSolutionStore((s) => s.setIsActive);

  const [isRunning, setRunning] = useState(false);
  const [direction, setDirection] = useState<Direction>("forward");
  const [speed, setSpeed] = useState(500);
  const [isMobile, setIsMobile] = useState(false);
  const wasUpdated = useMatrixStore((s) => s.wasUpdated);
  const stopUpdating = useMatrixStore((s) => s.stopUpdating);

  const handleStop = () => setRunning(false);

```

```

const { steps, index, move, skipAndFinish, loadingSteps, reset } =
  useSolutionRunner(
    method,
    slae,
    matrixState,
    setMatrix,
    setResult,
    setCurrentTargetRow,
    setIsActive,
    handleStop,
    setLoadingMatrix,
    wasUpdated,
    stopUpdating
  );

useInterval(
  () => {
    if (isRunning) move(direction);
  },
  isRunning ? speed : null
);

const handleStart = () => {
  if (!method) return toast.error("Select a method first.");
  if (!matrixState) {
    return toast.error("Matrix is empty or invalid. Please enter a matrix.");
  }
  if (direction === "backward" && index <= 0) {
    return toast.error("Cannot move backward from the first step.");
  }
  if (direction === "forward" && index !== -1 && index > steps.length - 1) {
    return toast.error("Cannot move forward from the last step.");
  }
  }

  setRunning(true);
  move(direction);
};

const handleReset = () => {
  if (!method) return toast.error("Select a method first.");
  if (!matrixState) {
    return toast.error("Matrix is empty or invalid. Please enter a matrix.");
  }
  setRunning(false);
  reset();
}

```



```

    setResult(null);
  };

  useEffect(() => {
    const checkMobile = () => setIsMobile(window.innerWidth < 768);
    checkMobile();
    window.addEventListener("resize", checkMobile);
    return () => window.removeEventListener("resize", checkMobile);
  }, []);

  return (
    <div className="h-full fixed">
      {isMobile ? (
        <>
          <Drawer>
            <DrawerTrigger asChild>
              <button className="fixed z-50 bottom-4 right-4 bg-primary text-primary-foreground rounded-full shadow-lg p-4 md:hidden">
                <Menu />
                <span className="sr-only">Open Step Controls</span>
              </button>
            </DrawerTrigger>
            <DrawerContent className="p-4">
              <DrawerHeader>
                <DrawerTitle>Step Controls</DrawerTitle>
              </DrawerHeader>
              <StepControls
                isRunning={isRunning}
                handleStart={handleStart}
                handleStop={handleStop}
                handleReset={handleReset}
                direction={direction}
                setDirection={setDirection}
                moveOne={move}
                skipAndFinish={skipAndFinish}
                speed={speed}
                setSpeed={setSpeed}
                isFirstStep={index === -1}
                isLastStep={result !== null}
                canUse={!method && !matrixState}
              />
              {showStepListInDrawer && (
                <div className="mt-4">
                  <h1>Step list</h1>
                  {loadingSteps ? (
                    <div className="flex items-center justify-center h-full">

```

```

        {Array.from({ length: 5 }).map((_, i) => (
          <Skeleton key={i} className="w-full h-10 mb-2" />
        ))}
      </div>
    ) : (
      <StepList steps={steps} index={index} />
    )}
  </div>
)}
<DrawerFooter>
  <DrawerClose asChild>
    <Button variant="outline">Close</Button>
  </DrawerClose>
</DrawerFooter>
</DrawerContent>
</Drawer>
{!showStepListInDrawer && (
  <div className="mt-4">
    <h1>Step list</h1>
    {loadingSteps ? (
      <div className="flex items-center justify-center h-full">
        {Array.from({ length: 5 }).map((_, i) => (
          <Skeleton key={i} className="w-full h-10 mb-2" />
        ))}
      </div>
    ) : (
      <StepList steps={steps} index={index} />
    )}
  </div>
)}
</>
) : (
  <
    <StepControls
      isRunning={isRunning}
      handleStart={handleStart}
      handleStop={handleStop}
      handleReset={handleReset}
      direction={direction}
      setDirection={setDirection}
      moveOne={move}
      skipAndFinish={skipAndFinish}
      speed={speed}
      setSpeed={setSpeed}
      isFirstStep={index === -1}
      isLastStep={result !== null}
    >

```

```

    canUse={!method && !!matrixState}
  />
  <h1>Step list</h1>
  {loadingSteps ? (
    <div className="flex items-center justify-center h-full">
      {Array.from({ length: 5 }).map((_, i) => (
        <Skeleton key={i} className="w-full h-10 mb-2" />
      ))}
    </div>
  ) : (
    <StepList steps={steps} index={index} />
  )}
</>
)}
</div>
);
}

```

#### **action.ts**

```
export type Direction = "forward" | "backward";
```

#### **step-controls.tsx**

```

import { Button } from "../ui/button";
import { ArrowLeft, ArrowRight, Goal, UndoDot } from "lucide-react";
import { Card } from "../ui/card";
import { Slider } from "../ui/slider";
import { cn } from "@lib/utls";
import type { Direction } from "../action";

```

```

interface StepControlsProps {
  isRunning: boolean;
  isFirstStep: boolean;
  isLastStep: boolean;
  handleStart: () => void;
  handleStop: () => void;
  handleReset: () => void;
  direction: Direction;
  setDirection: (value: Direction) => void;
  moveOne: (direction: Direction) => void;
  skipAndFinish: (direction: Direction) => void;
  speed: number;
  setSpeed: (value: number) => void;
  canUse: boolean;
}

```

```

}

const StepControls = ({
  isRunning,
  handleStart,
  handleStop,
  handleReset,
  direction,
  setDirection,
  moveOne,
  skipAndFinish,
  speed,
  setSpeed,
  canUse,
  isFirstStep,
  isLastStep,
}: StepControlsProps) => {
  const impossibleToMoveForward = isLastStep && direction === "forward";
  const impossibleToMoveBackward = isFirstStep && direction === "backward";

  const toggleDirection = () => {
    if (direction === "forward") {
      setDirection("backward");
    } else {
      setDirection("forward");
    }
  };

  return (
    <div
      className={cn(
        "flex flex-col gap-2 mb-4 w-full",
        !canUse && "opacity-50 pointer-events-none select-none"
      )}
    >
      {/* First row: Start/Stop, Reset, Direction */}
      <div className="flex gap-4 items-center w-full">
        {isRunning ? (
          <Button onClick={handleStop}>Stop</Button>
        ) : (
          <Button onClick={handleStart}>Start</Button>
        )}
        <Button onClick={handleReset} variant="outline">
          Reset <UndoDot />
        </Button>
        <Button onClick={toggleDirection} variant="outline">

```

```

    {direction === "forward" ? (
      <>
        Forward
        <ArrowRight />
      </>
    ) : (
      <>
        Backward
        <ArrowLeft />
      </>
    )}
  </Button>
</div>
{/* Second row: Move, Complete, Speed */}
<div className="flex gap-4 items-center w-full">
  <Button
    onClick={() => moveOne(direction)}
    variant={"outline"}
    disabled={
      !canUse || impossibleToMoveBackward || impossibleToMoveForward
    }
  >
    Move
  </Button>
  <Button onClick={() => skipAndFinish(direction)}>
    <Goal />
  </Button>
  <Card className="w-full p-2">
    <div className="flex flex-col gap-2 items-center justify-between">
      <span>Speed: {speed}ms per step</span>
      <Slider
        value={[speed]}
        min={10}
        max={300}
        step={10}
        onValueChange={({val}) => setSpeed(val)}
      />
    </div>
  </Card>
</div>
);
};

```

```
export default StepControls;
```

**step-list.tsx**

```

import React from "react";
import { useVirtualizer } from "@tanstack/react-virtual";
import {
  getStepDescription,
  type StepMetadata,
} from "@lib/steps/step-metadata";

interface StepListProps {
  steps: StepMetadata[];
  index: number;
}

const StepList: React.FC<StepListProps> = ({ steps, index }) => {
  const containerRef = React.useRef<HTMLDivElement | null>(null);
  const virtualizer = useVirtualizer({
    count: index + 1,
    getScrollElement: () => containerRef.current,
    estimateSize: () => 50,
    overscan: 5,
  });

  React.useEffect(() => {
    virtualizer.scrollToIndex(index, { align: "center", behavior: "smooth" });
  }, [index, virtualizer]);

  return (
    <div ref={containerRef} className="relative h-[500px] overflow-auto">
      <div
        style={{
          height: `${virtualizer.getTotalSize()}px`,
          position: "relative",
        }}
      >
        {virtualizer.getVirtualItems().map((virtualItem) => (
          <div
            key={virtualItem.key}
            className="p-2 border rounded-md absolute w-full"
            style={{
              transform: `translateY(${virtualItem.start}px)`,
              height: `${virtualItem.size}px`,
            }}
          >
            {getStepDescription(steps[virtualItem.index])}
          </div>
        ))}
      </div>
    </div>
  )
}

```

```

    </div>
  </div>
);
};

export default StepList;

charts/algorithm-chart.tsx
import { CartesianGrid, Line, LineChart, XAxis } from "recharts";

import {
  ChartContainer,
  ChartTooltip,
  ChartTooltipContent,
} from "@components/ui/chart";

import type { ChartDataEntry, ChartGroup } from "./chart";

export function AlgorithmChart({
  chartGroup,
  chartData,
  valueKey = "elementaryOperations",
}: {
  chartGroup: ChartGroup;
  chartData: ChartDataEntry[];
  valueKey?: "elementaryOperations" | "iterations" | "backSubstitution";
}) {
  const processedData = chartData.map((entry) => {
    const entryRecord = entry as Record<string, number>;
    const newEntry: { [key: string]: number } = { size: entry.size };
    Object.keys(chartGroup).forEach((method) => {
      if (valueKey === "elementaryOperations") {
        newEntry[method] = entryRecord[method];
      } else if (valueKey === "iterations") {
        newEntry[method] = entryRecord[method + "_iterations"];
      } else if (valueKey === "backSubstitution") {
        newEntry[method] = entryRecord[method + "_backSubstitution"];
      }
    });
    return newEntry;
  });

  return (
    <div>
      <ChartContainer config={chartGroup}>
        <LineChart data={processedData} margin={{ left: 12, right: 12 }}>

```

```

    <CartesianGrid vertical={false} />
    <XAxis
      dataKey="size"
      tickLine={false}
      axisLine={false}
      tickMargin={8}
    />
    <ChartTooltip cursor={false} content={<ChartTooltipContent />} />
    {Object.entries(chartGroup).map(([method, config]) => (
      <Line
        key={method}
        dataKey={method}
        type="monotone"
        stroke={config.color}
        name={config.label}
        strokeWidth={2}
        dot={false}
        connectNulls
      />
    ))}
  </LineChart>
</ChartContainer>
</div>
);
}

```

### **chart-generator.tsx**

```

import { useEffect, useRef, useState } from "react";
import { MethodType } from "@lib/methods/IMethod";
import { methodToString } from "../solution/preferences/method";
import { MultiSelector } from "../ui/multi-selector";
import { Input } from "../ui/input";
import { AlgorithmChart } from "../algorithm-chart";
import type { ChartDataEntry, ChartGroup } from "../chart";
import { Button } from "../ui/button";
import { createChartWorker } from "@workers/chart.worker-wrapper";
import { proxy } from "comlink";
import { toast } from "sonner";
import { Separator } from "../ui/separator";

const ChartGenerator = () => {
  const [methods, setMethods] = useState<MethodType[]>([
    "Gauss",
    "GaussJordan",
    "InverseMatrix",
  ]);
};

```



```

const [methodToAdd, setMethodToAdd] = useState<MethodType | null>(null);

const [sizes, setSizes] = useState<number[]>([10, 100]);
const [sizeToAdd, setSizeToAdd] = useState<number | null>(null);

const [timesPerSize, setTimesPerSize] = useState<number>(1);

const [, setIsGenerating] = useState<boolean>(false);
const [wasGenerating, setWasGenerating] = useState<boolean>(false);
const [chartGroup, setChartGroup] = useState<ChartGroup>();
const [chartData, setChartData] = useState<ChartDataEntry[]>([]);
const workerRef = useRef<ReturnType<typeof createChartWorker> | null>(null);

useEffect(() => {
  // Initialize the worker
  workerRef.current = createChartWorker();
  return () => {
    workerRef.current?.terminate();
    workerRef.current = null;
  };
}, []);

async function startGeneration() {
  if (!workerRef.current) {
    toast.error("Worker is not initialized");
    return;
  }

  setIsGenerating(true);
  setWasGenerating(true);
  console.log("Starting generation...");

  const chartGroup: ChartGroup = {};
  for (const method of methods) {
    chartGroup[method] = {
      label: methodToString[method],
      color: getRandomColor(),
    };
  }
  setChartGroup(chartGroup);

  const newChartData: ChartDataEntry[] = [];
  const workerApi = workerRef.current;

  for (const size of sizes) {
    const data: ChartDataEntry & Record<string, number> = { size };

```

```

const promises = methods.map((methodType) => {
  return new Promise<void>((resolve) => {
    let totalOps = 0;
    let totalIterations = 0;
    let totalBackSub = 0;
    let callbackCount = 0;

    const callback = proxy(
      (result: {
        elementaryOperations: number;
        iterations: number;
        backSubstitutionOperations?: number;
      }) => {
        totalOps += result.elementaryOperations;
        totalIterations += result.iterations;
        if (typeof result.backSubstitutionOperations === "number") {
          totalBackSub += result.backSubstitutionOperations;
        }
        callbackCount++;

        if (callbackCount >= timesPerSize) {
          data[methodType] = totalOps / timesPerSize;
          data[methodType + "_iterations"] =
            totalIterations / timesPerSize;
          data[methodType + "_backSubstitution"] =
            totalBackSub / timesPerSize;
          resolve();
        }
      }
    );

    workerApi.runOneTillEndWithCallback(
      methodType,
      size,
      timesPerSize,
      { from: 1, to: 10 },
      callback
    );
  });
});

await Promise.all(promises).then(() => {
  newChartData.push(data);
});
}

```

```

console.log("Final chart data:", newChartData);
setChartData(newChartData);
setIsGenerating(false);
}
// Helper: generate a random hex color string like "#A1B2C3"
function getRandomColor(): string {
  return (
    "#" +
    Math.floor(Math.random() * 0xffffffff)
      .toString(16)
      .padStart(6, "0")
  );
}

return (
  <div className="flex gap-4 min-h-full">
    <div className="flex flex-col gap-4 w-1/3 min-w-[320px] h-full">
      <div className="bg-card p-4 rounded-lg shadow flex flex-col gap-4 h-full">
        <MultiSelector<MethodType>
          title="Methods"
          selectedItems={methods}
          allItems={Object.keys(methodToString) as MethodType[]}
          itemToAdd={methodToAdd}
          setItemToAdd={setMethodToAdd}
          setSelectedItems={setMethods}
          toString={(m: MethodType) => methodToString[m]}
        />
        <MultiSelector<number>
          title="Sizes"
          selectedItems={sizes}
          allItems={[10, 100, 1000, 10000]}
          itemToAdd={sizeToAdd}
          setItemToAdd={setSizeToAdd}
          setSelectedItems={setSizes}
          toString={(s: number) => s.toString()}
        />
      <div>
        <span className="text-sm text-muted-foreground">
          Times per size:
        </span>
        <Input
          type="number"
          value={timesPerSize}
          onChange={(e) => setTimesPerSize(Number(e.target.value))}
          placeholder="Times per size"

```

```

        className="w-48"
      />
    </div>
    <Button onClick={startGeneration}>Generate</Button>
  </div>
</div>
<div className="flex-1 h-full overflow-y-auto max-w-[800px]">
  {wasGenerating && chartGroup && (
    <div className="bg-card p-4 rounded-lg shadow overflow-y-auto flex flex-col gap-4">
      <div className="min-h-[160px]">
        <h3 className="text-lg font-semibold mb-2">
          Elementary Operations
        </h3>
        <AlgorithmChart chartGroup={chartGroup} chartData={chartData} />
      </div>
      <Separator />
      <div className="min-h-[160px]">
        <h3 className="text-lg font-semibold mb-2">Iterations</h3>
        <AlgorithmChart
          chartGroup={chartGroup}
          chartData={chartData}
          valueKey="iterations"
        />
      </div>
      <Separator />
      <div className="min-h-[160px]">
        <h3 className="text-lg font-semibold mb-2">
          Back Substitution Operations
        </h3>
        <AlgorithmChart
          chartGroup={chartGroup}
          chartData={chartData}
          valueKey="backSubstitution"
        />
      </div>
    </div>
  )}
</div>
</div>
);
};

```

```
export default ChartGenerator;
```

#### **charts/chart.ts**

```
import type { MethodType } from "@lib/methods/IMethod";
```

```

type MethodValues = {
  [K in MethodType]?: number;
};

export type ChartDataEntry = {
  size: number;
} & MethodValues;

type ChartGivenConfig = {
  label: string;
  color: string;
};

export type ChartGroup = {
  [K in MethodType]?: ChartGivenConfig;
};

```

#### **charting-mode.tsx**

```

import AppMenubar from "../app-menubar";
import ChartGenerator from "../chart-generator";

const ChartingMode = () => {
  return (
    <div className="flex flex-col gap-4 h-screen p-4">
      <div className="inline-flex">
        <AppMenubar />
      </div>
      <ChartGenerator />
    </div>
  );
};

export default ChartingMode;

```

#### **solution/preferences/clear-matrix-preferences.tsx**

```

import { Button } from "@components/ui/button";
import { useMatrixStore } from "@store/matrix";
import { Eraser } from "lucide-react";
import { toast } from "sonner";

const ClearMatrixPreferences = () => {
  const setSlae = useMatrixStore((state) => state.setSlae);
  const slae = useMatrixStore((state) => state.slae);

```

```

const clearSlae = () => {
  if (!slae) {
    toast.error("SLAE is not set.");
    return;
  }
  const rows = slae?.length;
  const cols = slae[0]?.length;
  setSlae(new Array(rows).fill(0).map(() => new Array(cols).fill(0)));
};

return (
  <Button disabled={!slae} onClick={clearSlae}>
    <Eraser />
  </Button>
);
};

export default ClearMatrixPreferences;

```

#### **solution/preferences/copy-matrix.tsx**

```

import { Button } from "@/components/ui/button";
import { useMatrixStore } from "@/store/matrix";
import { CopyCheckIcon, CopyIcon } from "lucide-react";
import React from "react";
import { toast } from "sonner";

const CopyMatrix = () => {
  const matrix = useMatrixStore((state) => state.slae);
  const [copied, setCopied] = React.useState(false);

  const copyToClipboard = async () => {
    if (!matrix) {
      toast.error("No matrix to copy");
      return;
    }

    try {
      let content = "";
      for (let i = 0; i < matrix.length; i++) {
        content += matrix[i].join(",") + "\n";
      }
      await navigator.clipboard.writeText(content);
      setCopied(true);
      setTimeout(() => setCopied(false), 2000);
    } catch (err) {
      console.error("Failed to copy!", err);
    }
  }

```

```

    };
    return (
      <Button onClick={copyToClipboard} variant="outline">
        Copy SLAE {copied ? <CopyCheckIcon /> : <CopyIcon />}
      </Button>
    );
  };

```

```
export default CopyMatrix;
```

### **method-preferences.tsx**

```

import React from "react";
import {
  Select,
  SelectContent,
  SelectGroup,
  SelectItem,
  SelectLabel,
  SelectTrigger,
  SelectValue,
} from "@components/ui/select";
import { Label } from "@components/ui/label.tsx";
import { MethodType } from "@lib/methods/IMethod.ts";
import { useSolutionStore } from "@store/solution";
import { methodToString } from "./method";

const MethodPreferences = () => {
  const method = useSolutionStore((state) => state.method);
  const setStoreMethod = useSolutionStore((state) => state.setMethod);
  const worker = useSolutionStore((state) => state.worker);

  const setMethod = (value: MethodType) => {
    setStoreMethod(value);
    if (worker) {
      worker.setMethod(value);
    }
  };

  return (
    <div className="flex flex-col gap-4">
      <Label htmlFor="methods">Methods</Label>

      <Select
        value={method ?? undefined}
        onValueChange={(value) => setMethod(value as MethodType)}
      >

```

```

    <SelectTrigger className="w-[180px]">
      <SelectValue placeholder="Select a method" />
    </SelectTrigger>
    <SelectContent>
      <SelectGroup>
        <SelectLabel>Method</SelectLabel>
        { Object.entries(methodToString).map(([key, value]) => (
          <SelectItem value={key} key={key}>
            { value }
          </SelectItem>
        ))}
      </SelectGroup>
    </SelectContent>
  </Select>
</div>
);
};

```

```
export default MethodPreferences;
```

#### **method.ts**

```
import type { MethodType } from "@lib/methods/IMethod";
```

```

type MethodsDropdown = {
  [K in MethodType]: string;
};

```

```

export const methodToString: MethodsDropdown = {
  Gauss: "Gauss",
  GaussJordan: "Gauss-Jordan",
  InverseMatrix: "Inverse Matrix",
};

```

#### **random-matrix-generator.tsx**

```

import { Button } from "@components/ui/button";
import {
  Dialog,
  DialogTrigger,
  DialogContent,
  DialogHeader,
  DialogTitle,
  DialogFooter,
  DialogClose,
} from "@components/ui/dialog";
import { useMatrixStore } from "@store/matrix";
import { toast } from "sonner";

```



```

import { useState } from "react";
import { Input } from "@/components/ui/input";
import { Dices } from "lucide-react";
import { useSolutionStore } from "@/store/solution";
import {
  MAX_AFTER_DOT,
  useSafeNumericInput,
} from "@/hooks/use-safe-numeric-input";

const RandomMatrixGenerator = () => {
  const setIsLoadingMatrix = useMatrixStore(
    (state) => state.setIsLoadingMatrix
  );
  const setSlae = useMatrixStore((state) => state.setSlae);
  const matrix = useMatrixStore((state) => state.slae);
  const worker = useSolutionStore((state) => state.worker);
  const [open, setOpen] = useState(false);

  const [from, setFrom] = useState(0);
  const [to, setTo] = useState(100);

  const { safeInput: fromInput, onSafeInputChange: onFromChange } =
    useSafeNumericInput(
      from,
      (num) => setFrom(num),
      undefined, // or { min: -999, max: 999 }
      false,
      true
    );

  const { safeInput: toInput, onSafeInputChange: onToChange } =
    useSafeNumericInput(to, (num) => setTo(num), undefined, false, true);

  const setRandomMatrix = async () => {
    if (!matrix) {
      toast.error("Please set the matrix size first.");
      return;
    }

    if (from >= to) {
      toast.error("Invalid range: 'from' must be less than 'to'.");
      return;
    }

    setIsLoadingMatrix(true);
    try {

```

```

if (!worker) {
  toast.error("Worker not initialized");
  setIsLoadingMatrix(false);
  return;
}

const result = await worker.generateRandomMatrix(
  matrix.length,
  matrix[0].length,
  from,
  to,
  MAX_AFTER_DOT
);

setSlae(result);
await worker.setMatrix(result);
setIsLoadingMatrix(false);
setOpen(false);
} catch (err) {
  setIsLoadingMatrix(false);
  console.error("Error generating matrix:", err);
  toast.error("Failed to generate matrix");
}
};

return (
<Dialog open={open} onOpenChange={setOpen}>
  <DialogTrigger asChild>
    <Button>
      <Dices />
    </Button>
  </DialogTrigger>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Generate Random Matrix</DialogTitle>
    </DialogHeader>
    <div className="flex flex-col gap-2">
      <Input
        value={fromInput}
        onChange={(e) => onFromChange(e.target.value)}
        placeholder="From"
        type="text"
      />
      <Input
        value={toInput}
        onChange={(e) => onToChange(e.target.value)}
        placeholder="To"

```

```

        type="text"
      />
    </div>
    <DialogFooter>
      <Button onClick={setRandomMatrix}>Generate</Button>
      <DialogClose asChild>
        <Button variant="outline">Cancel</Button>
      </DialogClose>
    </DialogFooter>
  </DialogContent>
</Dialog>
);
};

export default RandomMatrixGenerator;

```

### **set-matrix-from-input.tsx**

```

import { useEffect, useState } from "react";
import { Button } from "@/components/ui/button";
import { useMatrixStore } from "@/store/matrix";
import { TextCursorInput } from "lucide-react";
import { toast } from "sonner"; // or any toast/error UI you prefer
import {
  Dialog,
  DialogDescription,
  DialogHeader,
  DialogTitle,
  DialogContent,
  DialogTrigger,
} from "@/components/ui/dialog";
import { Textarea } from "@/components/ui/textarea";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";

const parseMatrix = (separator: string, text: string): number[][] => {
  const lines = text.trim().split("\n");
  const rowCount = lines.length;
  const expectedColCount = rowCount + 1;

  const parsedMatrix = lines.map((line, rowIndex) => {
    const values = line
      .trim()
      .split(separator)
      .map((str) => {
        const num = Number(str);
        if (isNaN(num)) {

```

```

        throw new Error(`Invalid number "${str}" on line ${rowIndex + 1}`);
    }
    return num;
});

if (values.length !== expectedColCount) {
    throw new Error(
        `Line ${rowIndex + 1} must have ${expectedColCount} values (found ${
            values.length
        })`
    );
}

return values;
});
return parsedMatrix;
};

const convertMatrixToText = (matrix: number[][], separator: string): string => {
    return matrix
        .map((row) => row.map((num) => num.toString()).join(separator))
        .join("\n");
};

const SetMatrixFromInput = ({
    open,
    setOpen,
}: {
    open: boolean;
    setOpen: (open: boolean) => void;
}) => {
    const matrix = useMatrixStore((state) => state.slae);
    const setMatrix = useMatrixStore((state) => state.setMatrixConfiguration);
    const [separator, setSeparator] = useState<string>(",");
    const [textareaDraft, setTextareaDraft] = useState(matrix?.join("\n") || "");

    useEffect(() => {
        if (
            separator === "" ||
            separator === "\n" ||
            separator === "." ||
            separator === "-"
        ) {
            toast.error("Invalid separator. Please use a valid character.");
            setSeparator(",");
        }
    });

```

```

    if (matrix) {
      setTextareaDraft(convertMatrixToText(matrix, separator));
    }
  }, [matrix, separator]);

const onSubmit = () => {
  try {
    const parsedMatrix = parseMatrix(separator, textareaDraft);
    setMatrix({
      type: "standard",
      matrix: parsedMatrix,
    });
    setOpen(false);
  } catch (error: unknown) {
    toast.error("Matrix parsing failed", {
      description: (error as Error).message,
    });
  }
};

return (
  <Dialog open={open} onOpenChange={setOpen}>
    <DialogTrigger asChild>
      <Button className="w-fit">
        Matrix from text <TextCursorInput />
      </Button>
    </DialogTrigger>
    <DialogContent>
      <DialogHeader>
        <DialogTitle>Input your matrix in text format</DialogTitle>
        <DialogDescription>
          One row per line. Rows should be separated by any chars specified
          below
        </DialogDescription>
      </DialogHeader>
      <div className="flex items-center mb-4">
        <Label className="mr-2">Separator:</Label>
        <Input
          type="text"
          value={separator}
          onChange={(e) => setSeparator(e.target.value)}
          className="border rounded px-2 py-1 w-24"
        />
      </div>
      <Textarea
        wrap="off"

```

```

        className="w-full h-full overflow-auto "
        value={textareaDraft}
        onChange={(e) => {
            setTextareaDraft(e.target.value);
        }}
    />
    <div className="mt-4 flex justify-end">
        <Button type="submit" onClick={onSubmit} variant="outline">
            Set Matrix
        </Button>
    </div>
</DialogContent>
</Dialog>
);
};

```

```
export default SetMatrixFromInput;
```

### size-input.tsx

```

import { Button } from "@components/ui/button";
import { Input } from "@components/ui/input";
import { useSafeNumericInput } from "@hooks/use-safe-numeric-input";
import React, { useState } from "react";

const SizeInput = ({
    size,
    setSize,
}): {
    size: number;
    setSize: (size: number) => void;
} => {
    const [inputValue, setInputValue] = useState(String(size));
    const { safeInput, onSafeInputChange, isValid } = useSafeNumericInput(
        Number(inputValue),
        (num) => setInputValue(String(num)),
        { min: 1, max: 1000 },
        true,
        false
    );

    return (
        <div className="flex gap-2">
            <Input
                value={safeInput}
                type="text"
                onChange={(e) => onSafeInputChange(e.target.value)}
            />

```

```

    />
    <Button
      variant="outline"
      onClick={() => setSize(Number(safeInput))}
      disabled={!isValid}
    >
      Set
    </Button>
  </div>
);
};

```

```
export default SizeInput;
```

### **size-preferences.tsx**

```

import { Label } from "@components/ui/label";
import { useMatrixStore } from "@store/matrix";
import SizeInput from "../size-input";

const SizePreferences = () => {
  const matrix = useMatrixStore((state) => state.slae);
  const setSize = useMatrixStore((state) => state.resize);

  return (
    <div className="flex flex-col gap-4">
      <Label htmlFor="size">Size</Label>
      <SizeInput size={matrix?.length ?? 0} setSize={setSize} />
    </div>
  );
};

```

```
export default SizePreferences;
```

### **size-input.tsx**

```

import { Button } from "@components/ui/button";
import { Input } from "@components/ui/input";
import { useSafeNumericInput } from "@hooks/use-safe-numeric-input";
import React, { useState } from "react";

const SizeInput = ({
  size,
  setSize,
}: {
  size: number;
  setSize: (size: number) => void;
}) => {

```

```

const [inputValue, setInputValue] = useState(String(size));
const { safeInput, onSafeInputChange, isValid } = useSafeNumericInput(
  Number(inputValue),
  (num) => setInputValue(String(num)),
  { min: 1, max: 1000 },
  true,
  false
);

return (
  <div className="flex gap-2">
    <Input
      value={safeInput}
      type="text"
      onChange={(e) => onSafeInputChange(e.target.value)}
    />
    <Button
      variant="outline"
      onClick={() => setSize(Number(safeInput))}
      disabled={!isValid}
    >
      Set
    </Button>
  </div>
);
};

```

```
export default SizeInput;
```

### **size-preferences.tsx**

```

import { Label } from "@components/ui/label";
import { useMatrixStore } from "@store/matrix";
import SizeInput from "../size-input";

const SizePreferences = () => {
  const matrix = useMatrixStore((state) => state.slae);
  const setSize = useMatrixStore((state) => state.resize);

  return (
    <div className="flex flex-col gap-4">
      <Label htmlFor="size">Size</Label>
      <SizeInput size={matrix?.length ?? 0} setSize={setSize} />
    </div>
  );
};

```



```
export default SizePreferences;
```

### **solution/inverse-method-slae.tsx**

```
import React, { useState } from "react";
import { Button } from "../ui/button";
import SlaeDisplay from "../slae-display";
import { useMatrixStore } from "@store/matrix";
import { toast } from "sonner";

const InverseMethodSlae = ({
  matrix,
  isLoadingMatrix,
  currentTargetRow,
  isRunning,
  setCell,
}: {
  matrix: number[][] | null;
  isLoadingMatrix: boolean;
  currentTargetRow: number | null;
  isRunning: boolean;
  setCell: ((row: number, column: number, value: number) => void) | null;
}) => {
  const matrixState = useMatrixStore((state) => state.matrixConfiguration);
  const originalMatrix = useMatrixStore((state) => state.slae);

  const [inverseMethodMatrixView, setInverseMethodMatrixView] = useState<
    "original" | "adjusted" | "inverse"
  >("original");

  const getNextView = (current: "original" | "adjusted" | "inverse") => {
    if (current === "original") return "adjusted";
    if (current === "adjusted") return "inverse";
    return "original";
  };

  const toggleInverseMethodMatrixView = () => {
    if (!matrixState) {
      toast.error("Matrix is empty or invalid. Please enter a matrix.");
      return;
    }
    setInverseMethodMatrixView((prev) => getNextView(prev));
  };

  const getButtonLabel = () => {
    if (inverseMethodMatrixView === "original") return "View Adjusted";
    if (inverseMethodMatrixView === "adjusted") return "View Inverse";
```

```

    return "View Original";
  };

  const getMatrixForView = () => {
    if (inverseMethodMatrixView === "original") {
      return originalMatrix ?? matrix;
    }
    if (inverseMethodMatrixView === "adjusted") {
      return matrixState?.type === "inverse" ? matrixState.adjusted : null;
    }
    // inverse
    return matrixState?.type === "inverse" ? matrixState.inverse : null;
  };

  // Determine if the matrix should be editable
  const isEditable = inverseMethodMatrixView === "original";

  // SlaeDisplay expects isEnterable and setCell props for editability
  return (
    <>
    <Button onClick={toggleInverseMethodMatrixView}>
      {getButtonLabel()}
    </Button>
    <SlaeDisplay
      matrix={getMatrixForView()}
      isLoadingMatrix={isLoadingMatrix}
      currentTargetRow={currentTargetRow}
      emptyText="Inverse method wasn't started yet."
      isEnterable={isEditable}
      isRunning={isRunning}
      setCell={setCell}
    />
  </>
  );
};

export default InverseMethodSlae;

```

### **slae-display.tsx**

```

import { useCallback, useEffect, useRef } from "react";
import SolutionCell from "@components/solution/solution-cell.tsx";
import { useVirtualizer } from "@tanstack/react-virtual";
import { Skeleton } from "../ui/skeleton";

const SlaeDisplay = ({
  matrix,

```

```

isLoadingMatrix,
currentTargetRow,
emptyText,
isEnterable,
isRunning,
setCell,
}: {
  matrix?: number[][] | null;
  isLoadingMatrix: boolean;
  currentTargetRow: number | null;
  emptyText?: string;
  isEnterable: boolean;
  isRunning: boolean;
  setCell: ((row: number, column: number, value: number) => void) | null;
}) => {
  const containerRef = useRef<HTMLDivElement>(null);
  const rows = matrix ? matrix.length : 0;
  const columns = matrix && matrix.length !== 0 ? matrix[0].length : 0;

  const getScrollElement = useCallback(() => containerRef.current, []);

  const columnVirtualizer = useVirtualizer({
    count: columns,
    horizontal: true,
    getScrollElement,
    estimateSize: () => 120,
    overscan: 5,
  });

  const rowVirtualizer = useVirtualizer({
    count: rows,
    getScrollElement,
    estimateSize: () => 50,
    overscan: 5,
  });

  useEffect(() => {
    if (currentTargetRow === null || currentTargetRow < 0) return;
    rowVirtualizer.scrollToIndex(currentTargetRow, {
      behavior: "smooth",
      align: "center",
    });
  }, [currentTargetRow]);

  const columnItems = columnVirtualizer.getVirtualItems();
  const rowItems = rowVirtualizer.getVirtualItems();

```

```

const [before, after] =
  columnItems.length > 0
    ? [
      columnItems[0].start,
      columnVirtualizer.getTotalSize() -
        columnItems[columnItems.length - 1].end,
    ]
    : [0, 0];

return !matrix || matrix.length === 0 ? (
  <div className="w-full h-full flex items-center justify-center text-lg text-muted-foreground">
    {emptyText || "Create a matrix to get started!"}
  </div>
) : (
  <div
    ref={containerRef}
    className="overflow-auto"
    style={{ width: "100%", height: "100%", position: "relative" }}
  >
    <div
      style={{
        height: `${rowVirtualizer.getTotalSize()}px`,
        width: `${columnVirtualizer.getTotalSize()}px`,
        position: "relative",
      }}
    >
      {rowItems.map((row) => (
        <div
          key={row.key}
          style={{
            position: "absolute",
            top: 0,
            left: 0,
            transform: `translateY(${row.start}px)`,
            display: "flex",
          }}
        >
          <div style={{ width: `${before}px` }} />

          {columnItems.map((column) => (
            <div
              key={column.key}
              style={{
                width: `${column.size}px`,
                height: `${row.size}px`,

```

```

        boxSizing: "border-box",
      }}
    >
    {isLoadingMatrix ? (
      <Skeleton
        key={` ${row.index} - ${column.index} `}
        className="absolute bg-muted rounded-md"
        style={{
          top: row.start + 4,
          left: column.start + 4,
          width: column.size - 8,
          height: row.size - 8,
        }}
      />
    ) : (
      <SolutionCell
        contents={matrix[row.index][column.index]}
        rowIndex={row.index}
        columnIndex={column.index}
        rowLength={columns}
        matrix={matrix}
        isRunning={isRunning}
        setCell={setCell}
        isEnterable={isEnterable}
      />
    )}
  </div>
))}
<div style={{ width: `${after}px` }} />
</div>
)))}
</div>
</div>
);
};

```

```
export default SlaeDisplay;
```

#### **solution/preferences/solution-cell.tsx**

```

import { Input } from "@/components/ui/input.tsx";
import {
  MAX_AFTER_DOT,
  useSafeNumericInput,
} from "@/hooks/use-safe-numeric-input";
import { toast } from "sonner";

```

```
interface SolutionCellProps {
  rowIndex: number;
  columnIndex: number;
  rowLength: number;
  contents: number;
  matrix: number[][] | null;
  isRunning: boolean;
  isEnterable: boolean;
  setCell: ((row: number, column: number, value: number) => void) | null;
}
```

```
function SolutionCell({
  columnIndex,
  rowIndex,
  rowLength,
  contents,
  matrix,
  isRunning,
  setCell,
  isEnterable,
}: SolutionCellProps) {
  const isEnding = columnIndex === rowLength - 1;
  const isStarting = columnIndex === 0;

  const { safeInput, onSafeInputChange } = useSafeNumericInput(
    contents,
    (num) => {
      if (isRunning) {
        toast.error("Cannot change cell value while running.");
        return;
      }
      if (!matrix) return;
      if (setCell) {
        const rounded = Number(num.toFixed(MAX_AFTER_DOT));
        console.log(
          "Setting cell in solution:",
          rowIndex,
          columnIndex,
          rounded
        );
        setCell(rowIndex, columnIndex, rounded);
      }
    }
  );
```

```
const charCount = safeInput.length > 0 ? safeInput.length : 1;
```

```

const inputWidth = `min(calc(${charCount}ch + 1.2rem), 11ch)`;

return (
  <div className="flex items-center justify-center p-1">
    <div className="flex items-center justify-between w-full h-full px-2">
      <div className="flex items-center gap-1">
        {!isStarting && (
          <span className="latex-symbol">{isEnding ? "=" : "+"}</span>
        )}
        {isEnterable ? (
          <Input
            disabled={isRunning}
            className="latex-input"
            style={{ width: inputWidth, minWidth: "48px" }}
            value={safeInput}
            onChange={(e) => onSafeInputChange(e.target.value)}
          />
        ) : (
          <span className="latex-symbol">
            {Number(contents).toFixed(MAX_AFTER_DOT)}
          </span>
        )}
      </div>
      {!isEnding && (
        <math>
          <span className="latex-symbol">&middot;</span>
          <span className="latex-symbol">
            x<sub>{columnIndex + 1}</sub>
          </span>
        </math>
      )}
    </div>
  </div>
);
}

```

```
export default SolutionCell;
```

#### **solution/solution-display.tsx**

```

import { useSolutionStore } from "@store/solution";
import SlaeDisplay from "../slae-display";
import SolutionResult from "../solution-result-display";
import { Button } from "../ui/button";
import { MoveLeft, MoveRight } from "lucide-react";
import { useState } from "react";
import InverseMethodSlae from "../inverse-method-slae";

```

```

import { useMatrixStore } from "@store/matrix";

function SolutionDisplay() {
  const solutionResult = useSolutionStore((state) => state.solutionResult);
  const [currentTab, setCurrentTab] = useState<"display" | "result">("display");
  const method = useSolutionStore((state) => state.method);
  const matrix = useMatrixStore((state) => state.slae);
  const setSlaeCell = useMatrixStore((state) => state.setMatrixCell);
  const worker = useSolutionStore((state) => state.worker);
  const setSlae = useMatrixStore((state) => state.setSlae);
  const isRunning = useSolutionStore((state) => state.isActive);
  const isLoadingMatrix = useMatrixStore((state) => state.isLoadingMatrix);
  const currentTargetRow = useMatrixStore((state) => state.currentTargetRow);
  const isActive = useSolutionStore((state) => state.isActive);

  const [areChanges, setAreChanges] = useState(false);
  const [matrixBeforeChanges, setMatrixBeforeChanges] = useState<
    number[][] | null
  >(null);

  const setCell = async (row: number, column: number, value: number) => {
    if (!worker) {
      console.error("Worker is not initialized");
      return;
    }
    if (!areChanges) {
      setMatrixBeforeChanges(matrix ? matrix.map((r) => [...r]) : null);
    }

    setSlaeCell(row, column, value);
    setAreChanges(true);
  };

  const applyChanges = async () => {
    if (!matrix || !worker) {
      console.error("Matrix or worker is not initialized");
      return;
    }

    try {
      setAreChanges(false);
      setMatrixBeforeChanges(null);
      await worker.setMatrix(matrix);
      setSlae(matrix);
    } catch (error) {
      console.error("Error applying changes:", error);
    }
  };
}

```



```

    }
  };

  const resetChanges = () => {
    setSlae(matrixBeforeChanges || []);
    setMatrixBeforeChanges(null);
    setAreChanges(false);
  };

  return currentTab === "result" && solutionResult ? (
    <>
    <SolutionResult result={solutionResult} />
    <Button onClick={() => setCurrentTab("display")}>
      <MoveLeft />
    </Button>
  </>
) : (
  <>
  {method === "InverseMatrix" ? (
    <InverseMethodSlae
      matrix={matrix}
      isLoadingMatrix={isLoadingMatrix}
      currentTargetRow={currentTargetRow}
      isRunning={isRunning}
      setCell={setCell}
    />
  ) : (
    <SlaeDisplay
      matrix={matrix}
      isLoadingMatrix={isLoadingMatrix}
      currentTargetRow={currentTargetRow}
      isEnterable={!isActive}
      isRunning={isRunning}
      setCell={setCell}
    />
  )}
  { /* Apply/Reset changes controls */ }
  {areChanges && (
    <div className="flex gap-2 mt-4">
      <Button variant="default" onClick={applyChanges}>
        Apply Changes
      </Button>
      <Button variant="outline" onClick={resetChanges}>
        Reset Changes
      </Button>
    </div>
  )}

```

```

    })
    {solutionResult && (
      <Button
        size="icon"
        variant="outline"
        onClick={() => setCurrentTab("result")}
      >
        <MoveRight />
      </Button>
    )}
  </>
);
}

```

```
export default SolutionDisplay;
```

### **solution-mode.tsx**

```

import AppMenubar from "@components/app-menubar";
import SolutionDisplay from "@components/solution/solution-display.tsx";
import SolutionPreferences from "@components/solution/solution-preferences";
import ActionSidebar from "@components/action/action-sidebar";
import {
  ResizableHandle,
  ResizablePanel,
  ResizablePanelGroup,
} from "@components/ui/resizable";
import { useEffect, useState } from "react";
import { createSolutionWorker } from "@workers/solution.worker-wrapper";
import { toast } from "sonner";
import { useSolutionStore } from "@store/solution";

```

```

const SolutionMode = () => {
  const [isMobile, setIsMobile] = useState(false);
  const setWorker = useSolutionStore((state) => state.setWorker);
  useEffect(() => {
    const worker = createSolutionWorker();
    if (!worker) {
      toast.error("Failed to create solution worker.");
      return;
    }
    setWorker(worker);
  }, [setWorker]);
  useEffect(() => {
    const checkMobile = () => setIsMobile(window.innerWidth < 768);
    checkMobile();
    window.addEventListener("resize", checkMobile);

```

```

    return () => window.removeEventListener("resize", checkMobile);
  }, []);

  if (isMobile) {
    return (
      <div className="flex flex-col gap-8 h-screen p-4">
        <div className="inline-flex">
          <AppMenubar />
        </div>
        <SolutionPreferences />
        <SolutionDisplay />
        <ActionSidebar showStepListInDrawer />
      </div>
    );
  }

  return (
    <ResizablePanelGroup direction="horizontal">
      <ResizablePanel defaultSize={66} minSize={40}>
        <div className="flex flex-col gap-8 h-screen p-4">
          <div className="inline-flex">
            <AppMenubar />
          </div>

          <SolutionPreferences />
          <SolutionDisplay />
        </div>
      </ResizablePanel>
      <ResizableHandle />
      <ResizablePanel defaultSize={34} minSize={16}>
        <div className="h-full p-4">
          <ActionSidebar />
        </div>
      </ResizablePanel>
    </ResizablePanelGroup>
  );
};

export default SolutionMode;

```

### **solution-preferences.tsx**

```

import SizePreferences from "../preferences/size-preferences";
import MethodPreferences from "../preferences/method-preferences";
import RandomMatrixGenerator from "../preferences/random-matrix-generator";
import CopyMatrix from "../preferences/copy-matrix";
import SetMatrixFromInput from "../preferences/set-matrix-from-input";

```

```

import { useState } from "react";
import ClearMatrixPreferences from "../preferences/clear-matrix-preferences";

function SolutionPreferences() {
  const [open, setOpen] = useState(false);
  return (
    <div
      className="flex gap-4 items-end flex-wrap"
      style={{ position: "relative" }}
    >
      <SizePreferences />
      <ClearMatrixPreferences />
      <RandomMatrixGenerator />
      <SetMatrixFromInput open={open} setOpen={setOpen} />
      <MethodPreferences />
      <CopyMatrix />
    </div>
  );
}

export default SolutionPreferences;

```

#### **solution-result-display.tsx**

```

import type { SolutionResult } from "@lib/solution/solution-result";
import { SolutionResultType } from "@lib/solution/solution-result-type";
import { Card, CardContent, CardHeader, CardTitle } from "../ui/card";
import * as React from "react";
import { useVirtualizer } from "@tanstack/react-virtual";

const ROW_HEIGHT = 36;

const SolutionResultDisplay = ({ result }: { result: SolutionResult }) => {
  const roots = result?.roots || [];
  const parentRef = React.useRef<HTMLDivElement>(null);

  const rowVirtualizer = useVirtualizer({
    count: roots.length,
    getScrollElement: () => parentRef.current,
    estimateSize: () => ROW_HEIGHT,
    overscan: 8,
  });

  return (
    <Card className="max-w-2xl mx-auto mt-6">
      <CardHeader>
        <CardTitle>

```

```

<span className="text-primary font-bold text-lg">
  {result.result === SolutionResultType.Unique && "Unique Solution"}
  {result.result === SolutionResultType.Infinite &&
    "Infinite Solutions"}
  {result.result === SolutionResultType.None && "No Solution"}
  {result.result === SolutionResultType.NoneOrInfinite &&
    "No or Infinite Solutions"}
</span>
</CardTitle>
</CardHeader>
<CardContent className="flex flex-col gap-6">
  {roots.length > 0 && (
    <div>
      <h3 className="font-semibold mb-2">Roots</h3>
      <div
        ref={parentRef}
        className="overflow-x-auto overflow-y-auto border rounded"
        style={{ maxHeight: "60vh", minHeight: 80, position: "relative" }}
      >
        <table className="min-w-[200px] text-sm w-full">
          <thead className="sticky top-0 bg-muted z-10">
            <tr>
              <th className="px-3 py-2 text-left bg-muted text-muted-foreground">
                Index
              </th>
              <th className="px-3 py-2 text-left bg-muted text-muted-foreground">
                Value
              </th>
            </tr>
          </thead>
          <tbody
            style={{
              position: "relative",
              display: "block",
              height: rowVirtualizer.getTotalSize(),
            }}
          >
            {rowVirtualizer.getVirtualItems().map((virtualRow) => {
              const i = virtualRow.index;
              return (
                <tr
                  key={i}
                  className={{"even:bg-muted/50"}}
                  style={{
                    position: "absolute",
                    top: 0,

```

```

        left: 0,
        width: "100%",
        transform: `translateY(${virtualRow.start}px)`,
      }}
    >
    <td className="px-3 py-2 font-mono">{i + 1}</td>
    <td className="px-3 py-2 font-mono">
      {roots[i].toFixed(12)}
    </td>
  </tr>
);
}}
</tbody>
</table>
</div>
</div>
)}
{result.description && (
  <div className="bg-muted rounded p-4 text-sm text-muted-foreground">
    {result.description}
  </div>
)}
</CardContent>
</Card>
);
};

```

```
export default SolutionResultDisplay;
```

### **uploads/import-dialog.tsx**

```

import React from "react";
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogTitle,
} from "../ui/dialog";
import CsvParserWorker from "@workers/csv-parser.worker?worker";
import { useMatrixStore } from "@store/matrix";
import { toast } from "sonner";
import { Input } from "../ui/input";

type MatrixLoadingDialogProps = {
  open: boolean;
  setOpen: (isOpen: boolean) => void;
};

```

```

const ImportDialog = ({ open, setOpen }: MatrixLoadingDialogProps) => {
  const workerRef = React.useRef<Worker | null>(null);
  const setMatrix = useMatrixStore((state) => state.setSlae);

  const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];
    if (!file) return;

    if (!workerRef.current) {
      workerRef.current = new CsvParserWorker();

      workerRef.current.onmessage = (event: MessageEvent) => {
        if (event.data.success) {
          setMatrix(event.data.data);
          toast.success("Matrix loaded successfully");
        } else {
          console.error("Worker error:", event.data.error);
          toast.error("Error loading matrix: " + event.data.error);
          setOpen(false);
        }
      };
    }

    workerRef.current.postMessage(file);
  };

  return (
    <Dialog onOpenChange={ (open) => !open } open={ open }>
      <DialogContent>
        <DialogTitle>Loading Matrix</DialogTitle>
        <DialogDescription>
          This is a loading dialog for CSV matrix It will show up when you are
          loading a matrix from CSV file
        </DialogDescription>
        <Input type="file" accept=".csv" onChange={ handleFileChange } />;
      </DialogContent>
    </Dialog>
  );
};

export default ImportDialog;

```

#### **app-menubar.tsx**

```

import {
  Menubar,
  MenubarMenu,

```

```

    MenubarContent,
    MenubarItem,
    MenubarTrigger,
  } from "@components/ui/menubar";
import { type Theme, useTheme } from "@components/theme-provider.tsx";
import ImportDialog from "../uploads/import-dialog";
import { useModeStore } from "@store/mode";
import { useState } from "react";

type Themes = {
  label: string;
  value: Theme;
}[];

const AppMenubar = () => {
  const appMode = useModeStore((state) => state.mode);
  const setAppMode = useModeStore((state) => state.setMode);
  const { theme, setTheme } = useTheme();
  const themes: Themes = [
    {
      label: "System",
      value: "system",
    },
    {
      label: "Light",
      value: "light",
    },
    {
      label: "Dark",
      value: "dark",
    },
  ];

  const [importModal, setImportModal] = useState<"CSV" | null>(null);
  return (
    <>
      <Menubar>
        <MenubarMenu>
          <MenubarTrigger>File</MenubarTrigger>
          <MenubarContent>
            <MenubarItem onClick={() => setImportModal("CSV")}>
              Import Matrix
            </MenubarItem>
            <MenubarItem>Export matrix</MenubarItem>
          </MenubarContent>
        </MenubarMenu>
      </>
    </>
  );

```



```

<MenubarMenu>
  <MenubarTrigger>Theme</MenubarTrigger>
  <MenubarContent>
    {themes.map(({ label, value }) => (
      <MenubarItem
        key={value}
        onSelect={() => setTheme(value)}
        className="flex items-center gap-2"
      >
        <span
          className="w-2 h-2 rounded-full dark:bg-white"
          style={{
            backgroundColor: theme === value ? "black" : "transparent",
          }}
        />
        {label}
      </MenubarItem>
    ))}
  </MenubarContent>
</MenubarMenu>
<MenubarMenu>
  <MenubarTrigger>Mode</MenubarTrigger>
  <MenubarContent>
    <MenubarItem onSelect={() => setAppMode("solution")}>
      Solution
    </MenubarItem>
    <MenubarItem onSelect={() => setAppMode("charts")}>
      Charts
    </MenubarItem>
  </MenubarContent>
</MenubarMenu>
</Menubar>
{importModal && <ImportDialog open={importModal} />}
</>
);
};

```

```
export default AppMenubar;
```

### **theme-provider.tsx**

```
import { createContext, useContext, useEffect, useState } from "react";
```

```
export type Theme = "dark" | "light" | "system";
```

```
type ThemeProviderProps = {
  children: React.ReactNode;
```

```

defaultTheme?: Theme;
storageKey?: string;
};

type ThemeProviderState = {
  theme: Theme;
  setTheme: (theme: Theme) => void;
};

const initialState: ThemeProviderState = {
  theme: "system",
  setTheme: () => null,
};

const ThemeProviderContext = createContext<ThemeProviderState>(initialState);

export function ThemeProvider({
  children,
  defaultTheme = "system",
  storageKey = "vite-ui-theme",
  ...props
}: ThemeProviderProps) {
  const [theme, setTheme] = useState<Theme>(
    () => (localStorage.getItem(storageKey) as Theme) || defaultTheme
  );

  useEffect(() => {
    const root = window.document.documentElement;

    root.classList.remove("light", "dark");

    if (theme === "system") {
      const systemTheme = window.matchMedia("(prefers-color-scheme: dark)")
        .matches
        ? "dark"
        : "light";

      root.classList.add(systemTheme);
      return;
    }

    root.classList.add(theme);
  }, [theme]);

  const value = {
    theme,

```

```

    setTheme: (theme: Theme) => {
      localStorage.setItem(storageKey, theme);
      setTheme(theme);
    },
  };

  return (
    <ThemeProviderContext.Provider {...props} value={value}>
      {children}
    </ThemeProviderContext.Provider>
  );
}

export const useTheme = () => {
  const context = useContext(ThemeProviderContext);

  if (context === undefined)
    throw new Error("useTheme must be used within a ThemeProvider");

  return context;
};

```

#### **hooks/use-interval.tsx**

```

import { useEffect, useRef } from "react";

export function useInterval(callback: () => void, delay: number | null) {
  const savedCallback = useRef(callback);

  useEffect(() => {
    savedCallback.current = callback;
  }, [callback]);

  useEffect(() => {
    if (delay === null) return;
    const id = setInterval(() => savedCallback.current(), delay);
    return () => clearInterval(id);
  }, [delay]);
}

```

#### **hooks/use-safe-numeric-input.ts**

```

import { toast } from "sonner";
import { useEffect, useState } from "react";

export const MAX_BEFORE_DOT = 6;
export const MAX_AFTER_DOT = 6;

export const useSafeNumericInput = (
  input: number,

```

```

setInput: (num: number) => void,
bounds?: { min?: number; max?: number },
mustBeInteger: boolean = false,
checkValidPrecision: boolean = true
) => {
  const [safeInput, setValue] = useState(input.toString());
  const [isValid, setIsValid] = useState(true);

  // Sync internal state with external input changes
  useEffect(() => {
    const inputAsString = input.toString();
    if (inputAsString !== safeInput) {
      setValue(inputAsString);
      setIsValid(true);
    }
  }, [input]);

  function validateValue(newValue: string): string | null {
    if (newValue === "" || newValue === "-") return null;

    const parsed = Number(newValue);
    if (isNaN(parsed)) return "Invalid number";

    if (bounds) {
      const { min, max } = bounds;
      if (min !== undefined && parsed < min) return "Value is below minimum";
      if (max !== undefined && parsed > max) return "Value is above maximum";
    }

    if (mustBeInteger && !Number.isInteger(parsed))
      return "Value must be an integer";

    if (checkValidPrecision) {
      const [beforeDot, afterDot] = newValue.split(".");
      if (
        (beforeDot && beforeDot.length > MAX_BEFORE_DOT) ||
        (afterDot && afterDot.length > MAX_AFTER_DOT)
      ) {
        return "Invalid precision";
      }
    }

    return null;
  }

  function onSafeInputChange(newValue: string) {

```

```

    setValue(newValue);

    const validationResult = validateValue(newValue);
    if (validationResult === null) {
        setIsValid(true);
        setInput(Number(newValue));
    } else {
        setIsValid(false);
        toast.error(validationResult);
    }
}

return { safeInput, onSafeInputChange, isValid };
};

```

#### **hooks/use-solution-runner.ts**

```

import type { Direction } from "@components/action/action";
import type { MethodType } from "@lib/methods/IMethod";
import type { SolutionResult } from "@lib/solution/solution-result";
import type { StepMetadata } from "@lib/steps/step-metadata";
import type { MatrixConfiguration } from "@store/matrix";
import { useSolutionStore } from "@store/solution";
import { useEffect, useRef, useState } from "react";
import { toast } from "sonner";

export function useSolutionRunner(
    method: MethodType | null,
    slae: number[][] | null,
    configuration: MatrixConfiguration | null,
    setMatrix: (contents: MatrixConfiguration) => void,
    setResult: (result: SolutionResult | null) => void,
    setCurrentTargetRow: (row: number | null) => void,
    setIsActive: (isRunning: boolean) => void,
    stop: () => void,
    setLoadingMatrix: (loading: boolean) => void,
    wasUpdated: boolean,
    stopUpdating: () => void
) {
    const [steps, setSteps] = useState<StepMetadata[]>([]);
    const [loadingSteps, setLoadingSteps] = useState(false);
    const [index, setIndex] = useState(-1);
    const isFirstStep = index === -1;

    const worker = useSolutionStore((state) => state.worker);

```

```

const startingMatrixRef = useRef<number[][] | null>(null);

useEffect(() => {
  if (!method || !slae || slae.length === 0) return;
  if (!startingMatrixRef.current) {
    startingMatrixRef.current = slae.map((row) => [...row]);
  } else {
    reset();
  }
}, [method]);

useEffect(() => {
  if (!method || !slae || slae.length === 0) return;
  startingMatrixRef.current = slae.map((row) => [...row]);
  reset();
}, [wasUpdated]);

const reset = async () => {
  if (!worker) {
    toast.error("Worker not initialized.");
    return;
  }
  if (!startingMatrixRef.current) {
    toast.error("Starting matrix is not set.");
    return;
  }
  if (!configuration) {
    toast.error("Matrix configuration is not set.");
    return;
  }
  if (!method) {
    toast.error("Method is not selected.");
    return;
  }

  setResult(null);
  setSteps([]);
  setIndex(-1);
  if (configuration?.type === "standard") {
    setMatrix({
      type: "standard",
      matrix: startingMatrixRef.current.map((row) => [...row]),
    });
  } else if (configuration?.type === "inverse") {
    setMatrix({
      type: "inverse",

```

```

    adjusted: [],
    inverse: [],
  });
}
await worker.reset();
await worker.setMethod(method!);
await worker.setMatrix(startingMatrixRef.current.map((row) => [...row]));
stopUpdating();
setIsActive(false);
};

// Move forward or backward one step
const move = async (direction: Direction) => {
  if (!worker) {
    toast.error("Worker not initialized.");
    return;
  }

  if (!configuration) {
    toast.error("Matrix is not set. Please enter a matrix first.");
    return;
  }

  if (direction === "forward") {
    await forwardOne();
  } else if (direction === "backward") {
    backwardOne();
  }
};

// Move forward one step via worker
const forwardOne = async () => {
  if (!worker) return;

  const step = await worker.getNextStep();
  if (!step) {
    const result = await worker.getResult();
    setResult(result);
    setIsActive(false);
    toast.success("Reached the end!");
    return;
  }

  const updatedMatrix = await worker.getCurrentMatrix();
  if (!updatedMatrix) {
    toast.error("Failed to get updated matrix.");
  }
};

```

```

    return;
  }

  setSteps((prev) => [...prev, step]);
  setMatrix(updatedMatrix);

  setCurrentTargetRow(step.targetRow);
  setIndex((i) => i + 1);
  setIsActive(true);
};

// Move backward one step locally using history steps
const backwardOne = async () => {
  if (isFirstStep) {
    toast.error("Already at the beginning.");
    stop();
    setIsActive(false);
    return;
  }
  if (!worker || !steps || steps.length === 0) {
    toast.error("No previous steps available.");
    return;
  }

  const prevIndex = index - 1;
  const prevStep = await worker.getPreviousStep();
  if (!prevStep) {
    toast.error("No previous step found.");
    return;
  }

  const newMatrix = await worker.getCurrentMatrix();
  if (!newMatrix) {
    toast.error("Failed to get updated matrix.");
    return;
  }

  setMatrix(newMatrix);
  setCurrentTargetRow(steps[prevIndex].targetRow);
  setIndex(prevIndex);
  setIsActive(true);
};

// Skip to end or reset
const skipAndFinish = async (direction: Direction) => {
  if (!worker) {

```



```

    toast.error("Worker not initialized.");
    return;
}

if (!method) {
    toast.error("Select a method first.");
    return;
}

if (!configuration) {
    toast.error("Matrix is empty or invalid. Please enter a matrix.");
    return;
}

setIsActive(true);
if (direction === "forward") {
    await skipAndFinishForward();
} else if (direction === "backward") {
    if (index <= 0) {
        toast.error("Already at the first step.");
        return;
    }
    await skipAndFinishBackward();
}
setIsActive(false);
};

// Skip forward all steps via worker
const skipAndFinishForward = async () => {
    stop();

    if (!worker || !configuration) return;

    setLoadingMatrix(true);
    setLoadingSteps(true);

    const res = await worker.skipAndFinishForward();
    if (!res) {
        toast.error("Failed to skip and finish forward.");
        setLoadingSteps(false);
        setLoadingMatrix(false);
        return;
    }

    const { results, steps, matrix: updatedMatrix } = res;

```

```

    if (results) setResult(results);
    setMatrix(updatedMatrix);
    setSteps(steps);
    setIndex(steps.length - 1);

    setLoadingSteps(false);
    setLoadingMatrix(false);
  };

const skipAndFinishBackward = async () => {
  if (!startingMatrixRef.current) {
    toast.error("Starting matrix is not set.");
    return;
  }
  if (!method) {
    toast.error("Select a method first.");
    return;
  }

  stop();
  if (!worker || !configuration) return;

  setSteps([]);
  setIndex(-1);
  const newMatrix = await worker.skipAndFinishBackward();
  if (!newMatrix) {
    toast.error("Failed to skip and finish backward.");
    return;
  }

  setMatrix(newMatrix);
  setResult(null);
  setCurrentTargetRow(null);
  await worker.reset();
  await worker.setMethod(method);
  await worker.setMatrix(startingMatrixRef.current.map((row) => [...row]));
};

return {
  steps,
  index,
  move,
  skipAndFinish,
  stop,
  reset,
  setSteps,

```

```

    setIndex,
    startingMatrixRef,
    loadingSteps,
  };
}

```

### **lib/math/Matrix.ts**

```

import { isNearZero } from "../utils";

export class Matrix {
  public readonly rows: number;
  public readonly cols: number;
  private _contents: number[][];

  public get contents(): number[][] {
    return this._contents;
  }

  public set contents(value: number[][] ) {
    this._contents = value.map((row) => row.map((v) => v));
  }

  constructor(height: number, width: number);
  constructor(data: number[][]);
  constructor(param1: number | number[][], param2?: number) {
    if (typeof param1 === "number" && typeof param2 === "number") {
      this.rows = param1;
      this.cols = param2;
      this._contents = Array.from({ length: this.rows }, () =>
        Array(this.cols).fill(0)
      );
    } else if (Array.isArray(param1)) {
      this.rows = param1.length;
      this.cols = this.rows === 0 ? 0 : param1[0].length;
      this._contents = param1.map((row) => row.map((value) => value));
    } else {
      throw new Error("Invalid constructor arguments");
    }
  }

  get(row: number, col: number): number {
    return this._contents[row][col];
  }

  set(x: number, y: number, value: number): void {

```

```

    this._contents[x][y] = value;
  }

  swapRows(fromRow: number, toRow: number): void {
    [this._contents[fromRow], this._contents[toRow]] = [
      this._contents[toRow],
      this._contents[fromRow],
    ];
  }

  multiplyByVector(vector: number[]): number[] {
    if (this.cols !== vector.length) {
      throw new Error("Incompatible matrix and vector dimensions");
    }
    const result: number[] = Array(this.rows).fill(0);
    for (let i = 0; i < this.rows; i++) {
      for (let j = 0; j < this.cols; j++) {
        result[i] += this.get(i, j) * vector[j];
      }
    }
    return result;
  }

  toArray(): number[][] {
    return this._contents.map((row) => row.map((value) => value));
  }

  public isZeroRowCoefficients(row: number): boolean {
    for (let col = 0; col < this.cols - 1; col++) {
      if (!isNearZero(this.get(row, col))) return false;
    }
    return true;
  }
}

export function generateRandomMatrix(size: number): number[][] {
  return Array.from({ length: size }, () =>
    Array.from({ length: size + 1 }, () => Math.floor(Math.random() * 100))
  );
}

```

### **SlaeMatrix.ts**

```

import { Matrix } from "../Matrix";
import { SquareMatrix } from "../SquareMatrix";
import { isNearZero } from "../utils";

```

```

export class SlaeMatrix extends Matrix {
  private _size: number = this.rows;
  public get size(): number {
    return this._size;
  }
  constructor(size: number) {
    super(size, size + 1);
    this._size = size;
  }

  static fromNumbers(array: number[][]): SlaeMatrix {
    if (array.length === 0 || array[0].length === 0) {
      throw new Error("Input array must not be empty");
    }
    if (array.some((row) => row.length !== array[0].length)) {
      throw new Error("All rows in the input array must have the same length");
    }

    const matrix = new SlaeMatrix(array.length);
    for (let i = 0; i < array.length; i++) {
      for (let j = 0; j < array[i].length; j++) {
        matrix.set(i, j, array[i][j]);
      }
    }

    return matrix;
  }

  public toSquareMatrix(): SquareMatrix {
    const squareMatrix = new SquareMatrix(this.rows);
    for (let i = 0; i < this.rows; i++) {
      for (let j = 0; j < this.cols - 1; j++) {
        squareMatrix.set(i, j, this.get(i, j));
      }
    }
    return squareMatrix;
  }
}

```

#### **lib/math/SquareMatrix.ts**

```

import { Matrix } from "../Matrix";

export class SquareMatrix extends Matrix {
  private _size: number;
  public get size(): number {

```

```

    return this._size;
  }
  constructor(size: number) {
    super(size, size);
    this._size = size;
  }

  static identity(size: number): SquareMatrix {
    const identity = new SquareMatrix(size);
    for (let i = 0; i < size; i++) {
      for (let j = 0; j < size; j++) {
        identity.set(i, j, i === j ? 1 : 0);
      }
    }
    return identity;
  }
}

```

#### **lib/math/utils.ts**

```

import { maxPrecision } from "../constants";

const minJsValue = Math.pow(10, -maxPrecision);

export function isNearZero(
  value: number,
  epsilon: number = minJsValue
): boolean {
  return Math.abs(value) < epsilon;
}

```

#### **lib/methods/gauss-method.ts**

```

import type { SolutionResult } from "../solution/solution-result";
import { SolutionResultType } from "../solution/solution-result-type";
import { Method } from "../Method";
import { isNearZero } from "../math/utils";
import { StepSwapRows } from "../steps/step-swap-rows";
import { StepEliminate } from "../steps/step-eliminate";
import { SlaeMatrix } from "../math/slae-matrix";
import type { Step } from "../steps/step";

export class GaussMethod extends Method {
  *getForwardSteps(): IterableIterator<Step> {
    if (!this.matrix) {
      throw new Error("Matrix not initialized");
    }
    const augmentedMatrix = this.matrix;

```

```

for (let sourceRow = 0; sourceRow < augmentedMatrix.rows - 1; sourceRow++) {
  yield* this.performPivotSwap(augmentedMatrix, sourceRow);
  yield* this.performRowElimination(augmentedMatrix, sourceRow);
}
}

```

```

private *performPivotSwap(augmentedMatrix: SlaeMatrix, sourceRow: number) {
  const pivotRow = this.findPivotRow(augmentedMatrix, sourceRow);
  if (isNearZero(augmentedMatrix.get(pivotRow, sourceRow))) {
    return;
  }
  if (pivotRow !== sourceRow) {
    const step = new StepSwapRows(sourceRow, pivotRow);
    step.perform(augmentedMatrix);
    this.methodMetadata.elementaryOperations++;
    this.methodMetadata.iterations += step.iterations;
    yield step;
  }
}

```

```

private *performRowElimination(
  augmentedMatrix: SlaeMatrix,
  sourceRow: number
) {
  for (
    let eliminationRow = sourceRow + 1;
    eliminationRow < augmentedMatrix.rows;
    eliminationRow++
  ) {
    const step = new StepEliminate(sourceRow, eliminationRow);
    if (!step.perform(augmentedMatrix, false)) {
      continue;
    }
    this.methodMetadata.elementaryOperations++;
    this.methodMetadata.iterations += step.iterations;
    yield step;
  }
}

```

```

private findPivotRow(augmentedMatrix: SlaeMatrix, sourceRow: number) {
  let pivotRow = sourceRow;
  for (let i = sourceRow + 1; i < augmentedMatrix.rows; i++) {
    if (
      Math.abs(augmentedMatrix.get(i, sourceRow)) >
      Math.abs(augmentedMatrix.get(pivotRow, sourceRow))
    )

```

```

    ) {
        pivotRow = i;
    }
    this.methodMetadata.iterations++;
}
return pivotRow;
}

backSubstitute(): SolutionResult {
    if (!this.matrix) {
        throw new Error("Matrix not initialized");
    }

    const solutionType = this.analyzeEchelonForm(this.matrix);
    if (solutionType !== SolutionResultType.Unique) {
        return {
            result: solutionType,
            description:
                solutionType === SolutionResultType.Infinite
                ? "General solution exists"
                : undefined,
        };
    }

    const roots = this.solveUpperTriangular(this.matrix);
    return {
        result: SolutionResultType.Unique,
        roots: roots,
    };
}

private solveUpperTriangular(matrix: SlaeMatrix): number[] {
    const roots = new Array<number>(matrix.rows);

    for (let row = matrix.rows - 1; row >= 0; row--) {
        let rhs = matrix.get(row, matrix.cols - 1);

        for (let col = row + 1; col < matrix.cols - 1; col++) {
            const coeff = matrix.get(row, col);
            rhs -= coeff * roots[col];
            this.methodMetadata.backSubstitutionOperations++;
        }

        const pivot = matrix.get(row, row);
        if (isNearZero(pivot)) {
            throw new Error("Unexpected zero pivot during back-substitution");
        }
    }
}

```



```

    }

    roots[row] = rhs / pivot;
  }

  return roots;
}

private analyzeEchelonForm(matrix: SlaeMatrix): SolutionResultType {
  let rank = 0;
  const rows = matrix.rows;
  const cols = matrix.cols - 1;

  for (let row = 0; row < rows; row++) {
    const isZeroRow = matrix.isZeroRowCoefficients(row);
    this.methodMetadata.backSubstitutionOperations += cols;
    const rhs = matrix.get(row, matrix.cols - 1);

    if (isZeroRow && !isNearZero(rhs)) return SolutionResultType.None;
    if (!isZeroRow) rank++;
  }

  return rank < cols
    ? SolutionResultType.Infinite
    : SolutionResultType.Unique;
}
}

```

### **IMethod.ts**

```

import type { Step } from "../steps/step";
import type { SolutionResult } from "../solution/solution-result";
import { GaussMethod } from "../gauss-method";
import { JordanGaussMethod } from "../jordan-gauss-method";
import { InverseMethod } from "../inverse-method";
import type { SlaeMatrix } from "../math/slae-matrix";

export const MethodType = {
  Gauss: "Gauss",
  GaussJordan: "GaussJordan",
  InverseMatrix: "InverseMatrix",
} as const;

export type MethodType = (typeof MethodType)[keyof typeof MethodType];

export const createSolutionMethodFromType = (
  type: MethodType,

```

```

    matrix: SlaeMatrix
  ) => {
    switch (type) {
      case MethodType.Gauss:
        return new GaussMethod(matrix);
      case MethodType.GaussJordan:
        return new JordanGaussMethod(matrix);
      case MethodType.InverseMatrix:
        return new InverseMethod(matrix);
      default:
        throw new Error(`Unknown method type: ${type}`);
    }
  };

export const getMethodTypeFromClass = (method: IMethod): MethodType => {
  if (method instanceof GaussMethod) {
    return MethodType.Gauss;
  } else if (method instanceof JordanGaussMethod) {
    return MethodType.GaussJordan;
  } else if (method instanceof InverseMethod) {
    return MethodType.InverseMatrix;
  } else {
    throw new Error("Invalid method");
  }
};

// IMethod interface in TypeScript
export interface IMethod {
  getForwardSteps(): IterableIterator<Step>;
  backSubstitute(): SolutionResult;
  runToTheEnd(): Step[];
  matrix: SlaeMatrix | null;
}

```

#### **inverse-method.ts**

```

import type { Step } from "../steps/step";
import type { SolutionResult } from "../solution/solution-result";
import { Method } from "../Method";
import type { SlaeMatrix } from "../math/slae-matrix";
import { SquareMatrix } from "../math/SquareMatrix";
import { JordanGaussStepper } from "../jordan-gauss-stepper";
import { SolutionResultType } from "../solution/solution-result-type";

export class InverseMethod extends Method {
  private _adjustedMatrix: SquareMatrix;
  private _inverseMatrix: SquareMatrix;
}

```

```

private _matrixStepper: JordanGaussStepper;
private _rhs: number[];

public get inverseMatrix(): SquareMatrix {
  return this._inverseMatrix;
}

public get adjustedMatrix(): SquareMatrix {
  return this._adjustedMatrix;
}

constructor(matrix: SlaeMatrix) {
  super(matrix);
  this._adjustedMatrix = matrix.toSquareMatrix();
  this._inverseMatrix = SquareMatrix.identity(matrix.rows);
  this._matrixStepper = new JordanGaussStepper(
    this._adjustedMatrix,
    this.methodMetadata
  );
  this._rhs = new Array(matrix.rows);
  for (let i = 0; i < matrix.rows; i++) {
    this._rhs[i] = matrix.get(i, matrix.cols - 1);
  }
}

getForwardSteps(): IterableIterator<Step> {
  if (!this._adjustedMatrix) {
    throw new Error("Matrix not initialized");
  }
  const innerIterator = this._matrixStepper.getForwardSteps();
  const inverseMatrix = this._inverseMatrix;
  const metadata = this.methodMetadata;

  function* wrapper(): IterableIterator<Step> {
    for (const step of innerIterator) {
      step.perform(inverseMatrix);
      metadata.elementaryOperations++;
      metadata.iterations += step.iterations;
      yield step;
    }
  }
  this.iterator = wrapper();
  return this.iterator;
}

```

```

backSubstitute(): SolutionResult {
  if (!this._adjustedMatrix) {
    throw new Error("Matrix not initialized");
  }

  const solutionType = this.analyzeEchelonForm(this._adjustedMatrix);
  if (solutionType !== SolutionResultType.Unique) {
    return {
      result: solutionType,
      description:
        solutionType === SolutionResultType.NoneOrInfinite
          ? "Couldn't find inverse matrix. Cannot solve."
          : undefined,
    };
  }

  const roots = this._inverseMatrix.multiplyByVector(this._rhs);
  this.methodMetadata.backSubstitutionOperations +=
    this._inverseMatrix.rows * this._inverseMatrix.cols;

  return {
    result: SolutionResultType.Unique,
    roots: roots,
  };
}

public analyzeEchelonForm(matrix: SquareMatrix): SolutionResultType {
  const rows = matrix.rows;

  for (let row = 0; row < rows; row++) {
    const isZeroRow = this._adjustedMatrix.get(row, row) === 0;
    this.methodMetadata.iterations++;
    if (isZeroRow) {
      return SolutionResultType.NoneOrInfinite;
    }
  }

  return SolutionResultType.Unique;
}

```

#### **lib/methods/jordan-gauss-method.ts**

```

import { SlaeMatrix } from "../math/slae-matrix";
import { SolutionResultType } from "../solution/solution-result-type";
import type { SolutionResult } from "../solution/solution-result";
import { Method } from "../Method";

```

```

import { JordanGaussStepper } from "../jordan-gauss-stepper";
import type { Step } from "../steps/step";
import { isNearZero } from "../math/utlis";

export class JordanGaussMethod extends Method {
  private _stepper: JordanGaussStepper;
  constructor(matrix: SlaeMatrix) {
    super(matrix);
    this._stepper = new JordanGaussStepper(matrix, this.methodMetadata);
  }

  public getForwardSteps(): IterableIterator<Step> {
    if (!this.matrix) {
      throw new Error("Matrix not initialized");
    }
    this.iterator = this._stepper.getForwardSteps();
    return this.iterator;
  }

  backSubstitute(): SolutionResult {
    if (!this.matrix) {
      throw new Error("Matrix not initialized");
    }

    const solutionType = this.analyzeEchelonForm();
    if (solutionType !== SolutionResultType.Unique) {
      return {
        result: solutionType,
        description:
          solutionType === SolutionResultType.Infinite
            ? "General solution exists"
            : undefined,
      };
    }

    const roots = new Array<number>(this.matrix.rows);
    for (let i = 0; i < this.matrix.rows; i++) {
      roots[i] = this.matrix.get(i, this.matrix.cols - 1);
      this.methodMetadata.backSubstitutionOperations++;
    }

    return {
      result: SolutionResultType.Unique,
      roots,
    };
  }
}

```

```

public analyzeEchelonForm(): SolutionResultType {
  let rank = 0;
  const matrix = this.matrix;
  const rows = matrix.rows;
  const cols = matrix.cols - 1;

  for (let row = 0; row < rows; row++) {
    this.methodMetadata.iterations++;
    const isZeroRow = isNearZero(matrix.get(row, row));
    const rhs = matrix.get(row, cols);

    if (isZeroRow && !isNearZero(rhs)) return SolutionResultType.None;
    if (!isZeroRow) rank++;
  }

  return rank < cols
    ? SolutionResultType.Infinite
    : SolutionResultType.Unique;
}

```

#### **lib/methods/jordan-gauss-stepper.ts**

```

import type { Matrix } from "../math/Matrix";
import { isNearZero } from "../math/utils";
import type { Step } from "../steps/step";
import { StepEliminate } from "../steps/step-eliminate";
import { StepScaleAfterPivot } from "../steps/step-scale";
import { StepSwapRows } from "../steps/step-swap-rows";
import type { MethodMetadata } from "../Method";

export class JordanGaussStepper {
  public matrix: Matrix;
  public metadata: MethodMetadata;

  constructor(matrix: Matrix, metadata: MethodMetadata) {
    this.matrix = matrix;
    this.metadata = metadata;
  }

  *getForwardSteps(): IterableIterator<Step> {
    if (!this.matrix) {
      throw new Error("Matrix not initialized");
    }
    const augmentedMatrix = this.matrix;

```

```

for (let sourceRow = 0; sourceRow < augmentedMatrix.rows - 1; sourceRow++) {
  yield* this.performPivotSwap(augmentedMatrix, sourceRow);
  yield* this.performScaling(augmentedMatrix, sourceRow);
  yield* this.performRowElimination(augmentedMatrix, sourceRow, "down");
}

```

```

yield* this.performScaling(augmentedMatrix, augmentedMatrix.rows - 1);

```

```

for (
  let sourceRow = augmentedMatrix.rows - 1;
  sourceRow >= 0;
  sourceRow--
) {
  yield* this.performRowElimination(augmentedMatrix, sourceRow, "up");
}

```

```

private *performPivotSwap(augmentedMatrix: Matrix, sourceRow: number) {
  const pivotRow = this.findPivotRow(augmentedMatrix, sourceRow);
  if (isNearZero(augmentedMatrix.get(pivotRow, sourceRow))) {
    return;
  }
  if (pivotRow !== sourceRow) {
    const step = new StepSwapRows(sourceRow, pivotRow);
    step.perform(augmentedMatrix);
    this.metadata.elementaryOperations++;
    this.metadata.iterations += step.iterations;
    yield step;
  }
}

```

```

private *performScaling(augmentedMatrix: Matrix, sourceRow: number) {
  const step = new StepScaleAfterPivot(sourceRow);
  if (step.perform(augmentedMatrix, true)) {
    this.metadata.elementaryOperations++;
    this.metadata.iterations += step.iterations;
    yield step;
  }
}

```

```

private *performRowElimination(
  augmentedMatrix: Matrix,
  sourceRow: number,
  direction: "up" | "down"
) {
  const start = direction === "down" ? sourceRow + 1 : 0;

```

```

const end = direction === "down" ? augmentedMatrix.rows : sourceRow;
const stepInc = direction === "down" ? 1 : 1;
for (
  let eliminationRow = start;
  eliminationRow < end;
  eliminationRow += stepInc
) {
  if (eliminationRow === sourceRow) continue;
  const step = new StepEliminate(sourceRow, eliminationRow);
  if (!step.perform(augmentedMatrix, false)) {
    continue;
  }
  this.metadata.iterations += step.iterations;
  this.metadata.elementaryOperations++;
  yield step;
}
}

```

```

private findPivotRow(augmentedMatrix: Matrix, sourceRow: number) {
  let pivotRow = sourceRow;
  for (let row = sourceRow + 1; row < augmentedMatrix.rows; row++) {
    if (
      Math.abs(augmentedMatrix.get(row, sourceRow)) >
      Math.abs(augmentedMatrix.get(pivotRow, sourceRow))
    ) {
      pivotRow = row;
    }
  }
  return pivotRow;
}
}

```

### **lib/methods/Method.ts**

```

import { SlaeMatrix } from "../math/slae-matrix";
import type { SolutionResult } from "../solution/solution-result";
import type { Step } from "../steps/step";
import type { IMethod } from "../IMethod";

```

```

export type MethodMetadata = {
  iterations: number;
  elementaryOperations: number;
  backSubstitutionOperations: number;
};

```

```

export abstract class Method implements IMethod {
  abstract getForwardSteps(): IterableIterator<Step>;

```



```

abstract backSubstitute(): SolutionResult;
protected iterator?: IterableIterator<Step>;

public methodMetadata: MethodMetadata = {
  iterations: 0,
  elementaryOperations: 0,
  backSubstitutionOperations: 0,
};

constructor(matrix: SlaeMatrix) {
  this.matrix = matrix;
}

public matrix: SlaeMatrix;

public runToTheEnd() {
  if (!this.iterator) {
    throw new Error("Method not initialized. Call run() first.");
  }

  const steps: Step[] = [];
  for (const step of this.iterator) {
    steps.push(step);
  }
  return steps;
}

```

#### **lib/solution/solution-preferences.ts**

```

import type { MethodType } from "@lib/methods/IMethod.ts";

export type SolutionPreferencesType = {
  size: number,
  method: MethodType
}

```

#### **lib/solution/solution-result-type.ts**

```

export enum SolutionResultType {
  Unique = "Unique",
  Infinite = "Infinite",
  None = "None",
  NoneOrInfinite = "NoneOrInfinite",
}

```

#### **lib/solution/solution-result.ts**

```

import { SolutionResultType } from "../solution-result-type";

```

```
// SolutionResult record in TypeScript
export interface SolutionResult {
  result: SolutionResultType;
  roots?: number[];
  description?: string;
}
```

### **steps/step-eliminate.ts**

```
import type { Matrix } from "../math/Matrix";
import { isNearZero } from "../math/utils";
import { Step } from "../step";
import type { StepMetadata } from "../step-metadata";

export class StepEliminate extends Step {
  toMetadata(): StepMetadata {
    return {
      type: "eliminate",
      sourceRow: this.sourceRow,
      targetRow: this.targetRow,
      multiplier: this._multiplier,
    };
  }

  perform(matrix: Matrix, isStartingFromBeginning: boolean = true): boolean {
    return this.eliminateRow(matrix, isStartingFromBeginning);
  }

  private eliminateRow(
    augmentedMatrix: Matrix,
    isStartingFromBeginning: boolean = true
  ): boolean {
    const sourceRow = this.sourceRow;
    const targetRow = this.targetRow;
    const pivot = augmentedMatrix.get(sourceRow, sourceRow);

    if (this._multiplier === undefined) {
      if (isNearZero(pivot)) return false;
      this._multiplier = -augmentedMatrix.get(targetRow, sourceRow) / pivot;
    }

    for (
      let col = isStartingFromBeginning ? 0 : sourceRow;
      col < augmentedMatrix.cols;
      col++
    ) {

```

```

    const value =
      augmentedMatrix.get(targetRow, col) +
      augmentedMatrix.get(sourceRow, col) * this._multiplier;
    augmentedMatrix.set(targetRow, col, value);
    this.iterations++;
  }

  return true;
}

private _multiplier?: number;
public get multiplier(): number | undefined {
  return this._multiplier;
}

constructor(sourceRow: number, targetRow: number) {
  super(sourceRow, targetRow);
}

inverse(matrix: number[][]): number[][] {
  const sourceRow = this.sourceRow;
  const targetRow = this.targetRow;
  const multiplier = this._multiplier!;

  if (
    sourceRow < 0 ||
    targetRow < 0 ||
    sourceRow >= matrix.length ||
    targetRow >= matrix.length
  ) {
    throw new Error("Invalid row indices for inverse operation.");
  }

  const numCols = matrix[0].length;
  const result = matrix.map((row) => [...row]);
  for (let col = 0; col < numCols; col++) {
    result[targetRow][col] =
      result[targetRow][col] - multiplier * result[sourceRow][col];
  }
  return result;
}
}

```

#### **steps/step-metadata.ts**

```

type StepType = "eliminate" | "scale" | "swap";

```

```

export type StepMetadata = {
  sourceRow: number;
  targetRow: number;
  type: StepType;
  multiplier?: number;
};

export function getStepDescription(step: StepMetadata): string {
  switch (step.type) {
    case "eliminate":
      return `Eliminating row ${step.targetRow} using row ${step.sourceRow}`;
    case "scale":
      return `Scaling row ${step.sourceRow} by a factor of ${step.multiplier}`;
    case "swap":
      return `Swapping rows ${step.sourceRow} and ${step.targetRow}`;
    default:
      return "Unknown step type";
  }
}

```

#### **steps/step-scale.ts**

```

import type { Matrix } from "../math/Matrix";
import { isNearZero } from "../math/utils";
import { Step } from "../step";
import type { StepMetadata } from "../step-metadata";

export class StepScaleAfterPivot extends Step {
  toMetadata(): StepMetadata {
    return {
      type: "scale",
      sourceRow: this.sourceRow,
      targetRow: this.targetRow,
      multiplier: this._multiplier,
    };
  }

  private _multiplier?: number;

  perform(matrix: Matrix, isStartingFromSource: boolean = false): boolean {
    const sourceRow = this.sourceRow;
    const pivot = matrix.get(sourceRow, sourceRow);

    if (this._multiplier === undefined) {
      if (isNearZero(pivot)) return false;
      this._multiplier = 1 / pivot;
    }
  }
}

```

```

    if (isNearZero(this._multiplier)) return false;
    for (
      let columnIndex = isStartingFromSource ? sourceRow : 0;
      columnIndex < matrix.cols;
      columnIndex++
    ) {
      matrix.set(
        sourceRow,
        columnIndex,
        matrix.get(sourceRow, columnIndex) * this._multiplier!
      );
      this.iterations++;
    }
    return true;
  }
}

```

```

constructor(sourceRow: number) {
  super(sourceRow, sourceRow);
}

```

```

inverse(matrix: number[][]): number[][] {
  for (let col = 0; col < matrix[0].length; col++) {
    matrix[this.sourceRow][col] =
      matrix[this.sourceRow][col] * this._multiplier!;
  }
  return matrix;
}
}

```

#### **lib/steps/step-swap-rows.ts**

```

import { Step } from "../step";
import type { Matrix } from "../math/Matrix";
import type { StepMetadata } from "../step-metadata";

```

```

export class StepSwapRows extends Step {
  toMetadata(): StepMetadata {
    return {
      type: "swap",
      sourceRow: this.sourceRow,
      targetRow: this.targetRow,
    };
  }
  constructor(sourceRow: number, targetRow: number) {
    super(sourceRow, targetRow);
  }
}

```

```
iterations = 1;
```

```
perform(matrix: Matrix): boolean {
  matrix.swapRows(this.sourceRow, this.targetRow);
  return true;
}
```

```
performOnNumbers(matrix: number[][]): number[][] {
  const temp = matrix[this.sourceRow];
  matrix[this.sourceRow] = matrix[this.targetRow];
  matrix[this.targetRow] = temp;
  return matrix;
}
```

```
inverse(matrix: number[][]): number[][] {
  return this.performOnNumbers(matrix);
}
}
```

#### **lib/steps/step.ts**

```
import type { Matrix } from "../math/Matrix";
import type { StepMetadata } from "../step-metadata";

export abstract class Step {
  constructor(sourceRow: number, targetRow: number) {
    this.sourceRow = sourceRow;
    this.targetRow = targetRow;
  }

  public iterations = 0;

  public sourceRow: number;
  public targetRow: number;

  abstract perform(matrix: Matrix): boolean;

  abstract inverse(matrix: number[][]): number[][];

  abstract toMetadata(): StepMetadata;
}
```

#### **lib/utils.ts**

```
import { clsx, type ClassValue } from "clsx"
import { twMerge } from "tailwind-merge"

export function cn(...inputs: ClassValue[]) {
```

```

    return twMerge(clsx(inputs))
  }

```

### store/matrix.ts

```
import { create } from "zustand";
```

```
export type MatrixConfiguration =
```

```

  | { type: "standard"; matrix: number[][] }
  | { type: "inverse"; adjusted: number[][]; inverse: number[][] };

```

```
export type MatrixStore = {
```

```

  isLoadingMatrix: boolean;
  slae: number[][] | null;
  setSlae: (slae: number[][]) => void;
  matrixConfiguration: MatrixConfiguration | null;
  resize: (newSize: number) => void;
  setIsLoadingMatrix: (isLoading: boolean) => void;
  setMatrixConfiguration: (matrix: MatrixConfiguration) => void;
  setMatrixCell: (row: number, col: number, value: number) => void;
  currentTargetRow: number | null;
  setCurrentTargetRow: (row: number | null) => void;
  wasUpdated: boolean; // Optional, used to trigger reset in hooks
  stopUpdating: () => void; // Optional, used to trigger reset in hooks
};

```

```
export const useMatrixStore = create<MatrixStore>((set) => ({
```

```

  isLoadingMatrix: false,
  matrixConfiguration: null,
  slae: null,

```

```

  resize: (size: number) => {

```

```

    set((state) => {
      const slae = new Array(size)
        .fill(0)
        .map(() => new Array(size + 1).fill(0));
      if (state.slae) {
        // If there is an existing SLAE, copy its values into the new matrix
        const rows = Math.min(size, state.slae.length);
        const cols = Math.min(size + 1, state.slae[0].length);

```

```

        for (let i = 0; i < rows; i++) {
          for (let j = 0; j < cols; j++) {
            slae[i][j] = state.slae[i][j];
          }
        }
      }
    });
  }

```

```

    return {
      matrixConfiguration: {
        type: "standard",
        matrix: slae,
      },
      slae,
    };
  });
},
setIsLoadingMatrix: (isLoading) => set({ isLoadingMatrix: isLoading }),
setMatrixConfiguration: (matrix: MatrixConfiguration) =>
  set(() => {
    if (matrix.type === "standard") {
      return { matrixConfiguration: matrix, slae: matrix.matrix };
    }
    // Inverse state goes to inverse place
    return { matrixConfiguration: matrix };
  }),
setSlae: (slae) =>
  set((state) => {
    if (!state.matrixConfiguration) return {};
    if (state.matrixConfiguration.type !== "standard") return {};

    state.matrixConfiguration.matrix = slae;
    return {
      matrixConfiguration: { type: "standard", matrix: slae },
      slae,
      wasUpdated: true,
    };
  }),
setMatrixCell: (row, col, value) =>
  set((state) => {
    if (!state.matrixConfiguration) return {};
    if (state.matrixConfiguration.type !== "standard") return {};

    state.matrixConfiguration.matrix[row][col] = value;
    return { matrixConfiguration: state.matrixConfiguration };
  }),
setCurrentTargetRow: (row) => set({ currentTargetRow: row }),
currentTargetRow: null,
wasUpdated: false,
stopUpdating: () => set({ wasUpdated: false }),
}));

```

#### **store/mode.ts**

```
import { create } from "zustand";
```



```

type ModeStore = {
  mode: "solution" | "charts";
  setMode: (mode: "solution" | "charts") => void;
};

export const useModeStore = create<ModeStore>((set) => ({
  mode: "charts",
  setMode: (mode) => set({ mode }),
}));

```

#### **store/solution.ts**

```

import { type MethodType } from "@lib/methods/IMethod";
import type { SolutionResult } from "@lib/solution/solution-result";
import type { createSolutionWorker } from "@workers/solution.worker-wrapper";
import { create } from "zustand";

```

```

type SolutionState = {
  method: MethodType | null;
  setMethod: (method: MethodType) => void;
  solutionResult: SolutionResult | null;
  setSolutionResult: (result: SolutionResult | null) => void;
  currentStepIndex: number;
  setCurrentStepIndex: (index: number) => void;
  isActive: boolean;
  setIsActive: (isRunning: boolean) => void;
  worker: ReturnType<typeof createSolutionWorker> | null;
  setWorker: (worker: ReturnType<typeof createSolutionWorker> | null) => void;
};

```

```

export const useSolutionStore = create<SolutionState>((set) => ({
  method: null,
  setMethod: (method: MethodType) =>
    set({
      method: method,
      currentStepIndex: 0,
      solutionResult: null,
    }),
  solutionResult: null,
  setSolutionResult: (result) => set({ solutionResult: result }),
  currentStepIndex: 0,
  setCurrentStepIndex: (index) => set({ currentStepIndex: index }),
  isActive: false,
  setIsActive: (isRunning) => set({ isActive: isRunning }),
  worker: null,
  setWorker: (worker) => set({ worker }),
}));

```

```
));
```

#### **workers/chart.worker-wrapper.ts**

```
import { wrap } from "comlink";
import Worker from "../chart.worker.ts?worker";
import type { ChartWorker } from "../chart.worker";
```

```
export const createChartWorker = () => {
  return wrap<ChartWorker>(new Worker());
};
```

#### **workers/chart.worker.ts**

```
import { SlaeMatrix } from "@lib/math/slae-matrix";
import {
  createSolutionMethodFromType,
  type MethodType,
} from "@lib/methods/IMethod";
import type { MethodMetadata } from "@lib/methods/Method";
import { expose } from "comlink";
```

```
export type ChartWorker = {
  runOneTillEndWithCallback: (
    methodType: MethodType,
    size: number,
    timesPerSize: number,
    generationOptions: SlaeGenerationOptions,
    onStep: (stepResult: MethodMetadata) => void
  ) => Promise<void>;
  generateRandomMatrix(
    rows: number,
    cols: number,
    from: number,
    to: number
  ): number[][];

  terminate: () => void;
};
```

```
type SlaeGenerationOptions = {
  from: number;
  to: number;
};
```

```
export type ChartRunConfiguration = {
  method: MethodType;
  size: number;
```

```

    timesPerSize: number;
};

const chartWorker: ChartWorker = {
  runOneTillEndWithCallback: async (
    methodType: MethodType,
    size: number,
    timesPerSize: number,
    generationOptions: SlaeGenerationOptions,
    onStep: (stepResult: MethodMetadata) => void
  ) => {
    console.log(
      `Running method ${methodType} for size ${size} with ${timesPerSize} iterations`
    );
    for (let i = 0; i < timesPerSize; i++) {
      const matrix = SlaeMatrix.fromNumbers(
        chartWorker.generateRandomMatrix(
          size,
          size,
          generationOptions.from,
          generationOptions.to
        )
      );
      const method = createSolutionMethodFromType(methodType, matrix);
      const iterator = method.getForwardSteps();
      // eslint-disable-next-line @typescript-eslint/no-unused-vars
      for (const _ of iterator) {
        continue;
      }
      method.backSubstitute();

      console.log("Results on webworker", method.methodMetadata);
      onStep(method.methodMetadata);
    }
  },

  generateRandomMatrix(
    rows: number,
    cols: number,
    from: number,
    to: number
  ): number[][] {
    const matrix: number[][] = [];
    for (let i = 0; i < rows; i++) {
      const row: number[] = [];
      for (let j = 0; j < cols; j++) {

```

```

        const randomValue = Math.random() * (to - from) + from;
        row.push(randomValue);
    }
    matrix.push(row);
}
return matrix;
},
terminate: () => {
    console.log("Chart worker terminated.");
    self.close();
},
};

expose(chartWorker);

import Papa from "papaparse";

self.onmessage = (e: MessageEvent<File>) => {
    const file = e.data;

    Papa.parse(file, {
        header: false,
        skipEmptyLines: true,
        complete: (results) => {
            const data: number[][] = (results.data as string[][]).map((row) =>
                row.map((cell) => parseFloat(cell))
            );

            self.postMessage({ success: true, data });
        },
        error: (err) => {
            self.postMessage({ success: false, error: err.message });
        },
    });
};

export {};

```

#### **workers/csv-parser.worker.ts**

```

import Papa from "papaparse";

self.onmessage = (e: MessageEvent<File>) => {
    const file = e.data;

    Papa.parse(file, {
        header: false,

```

```

skipEmptyLines: true,
complete: (results) => {
  const data: number[][] = (results.data as string[][]).map((row) =>
    row.map((cell) => parseFloat(cell))
  );

  self.postMessage({ success: true, data });
},
error: (err) => {
  self.postMessage({ success: false, error: err.message });
},
});
};

export {};

```

#### **workers/solution.worker-wrapper.ts**

```

import { wrap } from "comlink";
import Worker from "@workers/solution.worker.ts?worker";
import type { SolutionWorker } from "./solution.worker";

```

```

export const createSolutionWorker = () => {
  return wrap<SolutionWorker>(new Worker());
};

```

#### **workers/solution.worker.ts**

```

import { expose } from "comlink";
import {
  createSolutionMethodFromType,
  getMethodTypeFromClass,
  type IMethod,
  type MethodType,
} from "@lib/methods/IMethod";
import type { Step } from "@lib/steps/step";
import type { SolutionResult } from "@lib/solution/solution-result";
import { SlaeMatrix } from "@lib/math/slae-matrix";
import type { StepMetadata } from "@lib/steps/step-metadata";
import { Matrix } from "@lib/math/Matrix";
import { InverseMethod } from "@lib/methods/inverse-method";
import type { MatrixConfiguration } from "@store/matrix";

```

```

export type SolutionWorker = {
  setMethod: (method: MethodType) => void;
  getNextStep(): StepMetadata | null;
  getPreviousStep(): StepMetadata | null;
  getCurrentMatrix(): MatrixConfiguration | null;

```

```

getSteps(): StepMetadata[];
skipAndFinishForward(): {
  results: SolutionResult | null;
  steps: StepMetadata[];
  matrix: MatrixConfiguration;
} | null;
skipAndFinishBackward(): MatrixConfiguration | null;
getResult(): SolutionResult | null;
reset(): void;
generateRandomMatrix(
  rows: number,
  cols: number,
  from: number,
  to: number,
  precision: number
): Promise<number[][]>;
setMatrix(matrix: number[][]): Promise<void>;
};

```

```

let methodInstance: IMethod | null = null;
let methodType: MethodType | null = null;
let matrixInstance: Matrix | null = null;
let iterator: Iterator<Step> | null = null;
let appliedSteps: Step[] = [];

```

```

const solutionWorker: SolutionWorker = {
  setMethod(method: MethodType): void {
    if (methodInstance) {
      const type = getMethodTypeFromClass(methodInstance);
      if (type === method) return;
    }

```

```

    if (!matrixInstance) {
      methodType = method;
      return;
    }

```

```

    methodInstance = createSolutionMethodFromType(
      method,
      matrixInstance as SlaeMatrix
    );

```

```

    iterator = methodInstance.getForwardSteps();
    appliedSteps = [];
  },

```

```

getNextStep(): StepMetadata | null {
  console.log("WEBWORKER: Getting next step.");
  console.log(iterator, matrixInstance, methodInstance);
  if (!iterator || !matrixInstance || !methodInstance) return null;

  const next = iterator.next();
  if (next.done) return null;

  appliedSteps.push(next.value);
  return next.value.toMetadata();
},

getPreviousStep(): StepMetadata | null {
  if (!matrixInstance || appliedSteps.length === 0) return null;

  const lastStep = appliedSteps.pop()!;
  const revertedMatrix = lastStep.inverse(matrixInstance.contents);

  matrixInstance.contents = revertedMatrix;
  return lastStep.toMetadata();
},

getCurrentMatrix(): MatrixConfiguration | null {
  if (methodInstance instanceof InverseMethod) {
    return {
      type: "inverse",
      adjusted: methodInstance.adjustedMatrix!.contents,
      inverse: methodInstance.inverseMatrix!.contents,
    };
  }
  return {
    type: "standard",
    matrix:
      matrixInstance?.contents.map((row) => row.map((value) => value)) ?? [],
  };
},

getSteps(): StepMetadata[] {
  return appliedSteps.map((step) => step.toMetadata());
},

skipAndFinishForward(): {
  results: SolutionResult | null;
  steps: StepMetadata[];
  matrix: MatrixConfiguration;
} | null {

```

```
if (!iterator || !methodInstance || !matrixInstance) return null;
```

```
while (true) {
  const next = iterator.next();
  if (next.done) break;
  appliedSteps.push(next.value);
}
```

```
return {
  results: methodInstance.backSubstitute(),
  steps: appliedSteps.map((step) => step.toMetadata()),
  matrix:
    methodInstance instanceof InverseMethod
    ? {
      type: "inverse",
      adjusted: methodInstance.adjustedMatrix!.contents,
      inverse: methodInstance.inverseMatrix!.contents,
    }
    : {
      type: "standard",
      matrix: matrixInstance.contents,
    },
};
```

```
skipAndFinishBackward(): MatrixConfiguration | null {
  if (!iterator || !methodInstance || !matrixInstance) return null;
```

```
for (let i = appliedSteps.length - 1; i >= 0; i--) {
  const step = appliedSteps[i];
  const revertedMatrix = step.inverse(matrixInstance.contents);
  matrixInstance.contents = revertedMatrix;
  appliedSteps.pop();
}
```

```
return methodInstance instanceof InverseMethod
  ? {
    type: "inverse",
    adjusted: methodInstance.adjustedMatrix!.contents.map((row) =>
      row.map((value) => value)
    ),
    inverse: methodInstance.inverseMatrix!.contents.map((row) =>
      row.map((value) => value)
    ),
  }
  : {
```



```

        type: "standard",
        matrix: matrixInstance.contents,
    };
},

getResult(): SolutionResult | null {
    return methodInstance?.backSubstitute() ?? null;
},

reset(): void {
    iterator = null;
    methodInstance = null;
    matrixInstance = null;
    appliedSteps = [];
},

async generateRandomMatrix(
    rows: number,
    cols: number,
    from: number,
    to: number,
    precision: number
): Promise<number[][]> {
    const matrix: number[][] = [];
    for (let i = 0; i < rows; i++) {
        const row: number[] = [];
        for (let j = 0; j < cols; j++) {
            const randomValue = parseFloat(
                (Math.random() * (to - from) + from).toFixed(precision)
            );
            row.push(randomValue);
        }
        matrix.push(row);
    }
    return matrix;
},

async setMatrix(matrix: number[][]): Promise<void> {
    matrixInstance = new Matrix(matrix);
    if (!methodType) return;
    solutionWorker.setMethod(methodType);
},
};

expose(solutionWorker);

```

**App.tsx**

```
import ChartingMode from "../components/charts/charting-mode";
import SolutionMode from "../components/solution/solution-mode";
import { useModeStore } from "../store/mode";

const App = () => {
  const mode = useModeStore((state) => state.mode);
  return mode === "solution" ? <SolutionMode /> : <ChartingMode />;
};

export default App;
```

**main.tsx**

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import "../index.css";
import App from "../App.tsx";
import RootLayout from "../RootLayout.tsx";

createRoot(document.getElementById("root")!).render(
  <StrictMode>
    <RootLayout>
      <App />
    </RootLayout>
  </StrictMode>
);
```

**RootLayout.tsx**

```
import { type ReactNode } from "react";
import { ThemeProvider } from "../components/theme-provider";
import { Toaster } from "../components/ui/sonner";

function RootLayout({ children }: { children: ReactNode }) {
  return (
    <ThemeProvider>
      {children}
      <Toaster />
    </ThemeProvider>
  );
}

export default RootLayout;
```