МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА «ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ ІМЕНІ А. В. ДАБАГЯНА»

ЗВІТ

З спринту №1

«АЛГЕБРАЇЧНІ РІВНЯННЯ»

ВИКОНАВ:

студент групи КН-422ч

Максим БЕЛОШИЦЬКИЙ

ПЕРЕВІРИВ:

Завідувач каф. ІСтТ

Олена НІКУЛІНА

Харків – 2023

**ЗМІСТ**

[МЕТА 2](#_Toc164158261)

[Завдання 1.1 3](#_Toc164158262)

[Код завдання 1.1: 3](#_Toc164158263)

[cramer\_method.m 3](#_Toc164158264)

[Результати виконання програми 4](#_Toc164158265)

[Завдання 1.2 5](#_Toc164158266)

[Код завдання 1.2: 5](#_Toc164158267)

[gauss\_method.m 5](#_Toc164158268)

[Результати виконання програми 6](#_Toc164158269)

[Завдання 1.3 7](#_Toc164158270)

[Код завдання 1.3: 7](#_Toc164158271)

[seidel\_method.m 7](#_Toc164158272)

[Результати виконання програм 8](#_Toc164158273)

[Завдання 1.4 10](#_Toc164158274)

[Код завдання 1.3: 10](#_Toc164158275)

[gauss\_jordan\_method.m 10](#_Toc164158276)

[Результати виконання програми 11](#_Toc164158277)

[Завдання 1.5 12](#_Toc164158278)

[Код завдання 1.5: 12](#_Toc164158279)

[Результати виконання програми 13](#_Toc164158280)

[Завдання 1.6 15](#_Toc164158281)

[Код завдання 1.6: 15](#_Toc164158282)

[Результати виконання програми 16](#_Toc164158283)

[Завдання 1.7 17](#_Toc164158284)

[Код завдання 1.7: 17](#_Toc164158285)

[Результати виконання програми 25](#_Toc164158286)

[ВИСНОВОК 27](#_Toc164158287)

# **МЕТА**

Мільйони людей займаються математичними обчисленнями, іноді через привабливість загадок у математиці або її внутрішню красу. Однак частіше це відбувається через професійну або іншу необхідність, не кажучи вже про навчання. Багато з цих людей прагнуть перекласти деякі процеси у нашому житті на математичну мову для швидшого та ефективнішого вирішення повсякденних викликів і проблем.

Результатом цих прагнень стали рівняння. У наш час більшість людей використовують їх постійно у своєму житті, наприклад, під час покупок і, отже, порівняння цін на різні товари в магазинах, під час планування будівництва та облаштування будівель, обчислення часу подорожі до різних пунктів призначення та маршрутів.

Поступово, із еволюцією нашої цивілізації, її процеси стають більш зрілими. Плануючи покупки, потрібно було враховувати більше параметрів: ціна, якість, час доставки тощо. Різноманіття їжі створило потребу у відстеженні наявності різних поживних речовин (білки, жири, вуглеводи, вітаміни). В результаті наше суспільство прийшло до системи лінійних алгебраїчних рівнянь, але ми можемо називати її у скороченому вигляді - СЛАР.

Проте разом із зростанням складності процесу ми також збільшуємо складність розв'язку системи. На щастя, були ті, хто дуже турбувався про це питання. Це вони створили різні методи для розв'язання СЛАР, з якими можна вивчити та створити власний програмний код у цій роботі.

# Завдання 1.1

Розробити програму для методу Крамера; організувати програму у формі функції. Програму можна писати будь-якою мові програмування.

Вхідні параметри: A - матриця коефіцієнтів СЛАР, B - стовпчиковий вектор вільних членів. Вихідні параметри: X - вектор-стовпчик коренів СЛАР.

## Код завдання 1.1:

### cramer\_method.m

function X = cramer\_method()

% Prompt user for input

A = input('Enter the coefficient matrix A: ');

B = input('Enter the column vector of free terms B: ');

% Check if A is square

[n, m] = size(A);

if n ~= m

error('Coefficient matrix must be square');

end

% Check if A is singular

if abs(det(A)) < eps

error('Coefficient matrix is singular, system may not have a unique solution');

end

% Check if number of equations matches length of B

if n ~= length(B)

error('Number of equations must match the length of the vector of free terms');

end

% Solve system using Cramer's method

X = zeros(n, 1);

for i = 1:n

% Replace the ith column of A with B

Ai = A;

Ai(:, i) = B;

% Calculate the determinant of the modified matrix

det\_Ai = det(Ai);

% Calculate Xi using Cramer's rule

X(i) = det\_Ai / det(A);

end

% Display result

disp('The solution vector X is:');

disp(X);

end

## Результати виконання програми

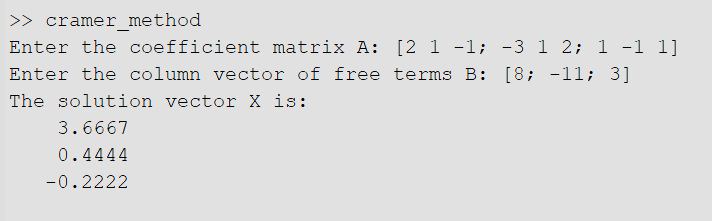


Рисунок 1.1 – «Розв’язок за Крамара»

Цей код є функцією на мові програмування MATLAB, яка реалізує метод Крамера для розв'язання систем лінійних алгебраїчних рівнянь (СЛАР). Давайте розглянемо кожен блок коду:

1. Prompt user for input: Функція спочатку просить користувача ввести коефіцієнт матриці A та вектор вільних членів B.
2. Validation of input: Перевіряється, чи матриця A є квадратною, чи вона не є сингулярною (тобто чи її детермінант не нульовий), та чи кількість рівнянь відповідає довжині вектора вільних членів B.
3. Solving the system using Cramer's method: Для кожного невідомого Xi обчислюється за допомогою правила Крамера, замінюючи i-й стовпець матриці A вектором B, обчислюючи детермінант модифікованої матриці і ділячи його на детермінант матриці A.
4. Displaying the result: Результат обчислень - вектор X, що містить корені СЛАР, виводиться на екран користувачеві.

Цей код можна використовувати для швидкого та ефективного розв'язання систем лінійних рівнянь за допомогою методу Крамера.

# Завдання 1.2

Розробити програму для методу Гауса; організувати програму у формі функції. Програму можна писати будь-якою мовою програмування.

Вхідні параметри: A - матриця коефіцієнтів СЛАР, B - стовпчиковий вектор вільних членів. Вихідні параметри: X - вектор-стовпчик коренів СЛАР.

## Код завдання 1.2:

### gauss\_method.m

function X = gauss\_method()

% Prompt user for input

A = input('Enter the coefficient matrix A: ');

B = input('Enter the column vector of free terms B: ');

% Check if A is square

[n, m] = size(A);

if n ~= m

error('Coefficient matrix must be square');

end

% Check if number of equations matches length of B

if n ~= length(B)

error('Number of equations must match the length of the vector of free terms');

end

% Augmented matrix

AB = [A, B];

% Forward elimination

for k = 1:n-1

for i = k+1:n

factor = AB(i,k) / AB(k,k);

AB(i,k:n+1) = AB(i,k:n+1) - factor \* AB(k,k:n+1);

end

end

% Back substitution

X = zeros(n,1);

X(n) = AB(n,n+1) / AB(n,n);

for i = n-1:-1:1

X(i) = (AB(i,n+1) - AB(i,i+1:n) \* X(i+1:n)) / AB(i,i);

end

% Display result

disp('The solution vector X is:');

disp(X);

end

## Результати виконання програми

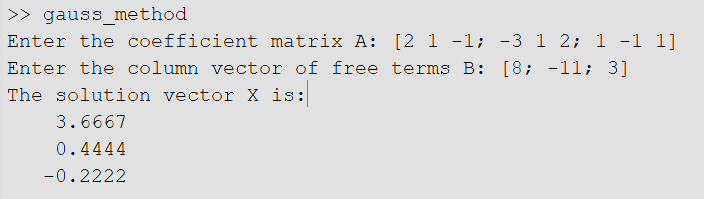


Рисунок 1.2 – «Обчислення gauss\_method()»

Цей код є функцією, яка реалізує метод Гаусса для розв'язання систем лінійних алгебраїчних рівнянь (СЛАР). Давайте розглянемо кожен блок коду:

1. Prompt user for input: Функція спочатку просить користувача ввести коефіцієнт матриці A та вектор вільних членів B.
2. Validation of input: Перевіряється, чи матриця A є квадратною, чи кількість рівнянь відповідає довжині вектора вільних членів B.
3. Forward elimination: Здійснюється пряма елімінація для отримання треугольної форми матриці, шляхом використання методу Гаусса.
4. Back substitution: Після прямої елімінації виконується обернена підстановка для знаходження коренів СЛАР.
5. Displaying the result: Результат обчислень - вектор X, що містить корені СЛАР, виводиться на екран користувачеві.

Цей код можна використовувати для швидкого та ефективного розв'язання систем лінійних рівнянь за допомогою методу Гаусса.

# Завдання 1.3

Розробити програму для методу Гаусса-Сейделя; організувати програму у формі функції. Програму можна писати будь-якою мовою програмування.

Вхідні параметри: A - матриця коефіцієнтів СЛАР, B - стовпчиковий вектор вільних членів. Вихідні параметри: X - вектор-стовпчик коренів СЛАР.

## Код завдання 1.3:

### seidel\_method.m

function X = seidel\_method()

% Prompt user for input

A = input('Enter the coefficient matrix A: ');

B = input('Enter the column vector of free terms B: ');

% Constants for convergence criteria

tolerance = 1e-9; % Tolerance for convergence

max\_iterations = 1000; % Maximum number of iterations

% Check if A is square

[n, m] = size(A);

if n ~= m

error('Coefficient matrix must be square');

end

% Check if number of equations matches length of B

if n ~= length(B)

error('Number of equations must match the length of the vector of free terms');

end

% Scale the system of equations to improve convergence

scaling\_factors = max(abs(A), [], 2);

scaled\_A = A ./ scaling\_factors;

scaled\_B = B ./ scaling\_factors;

% Initialize solution vector

X = zeros(n, 1);

% Initialize iteration counter

iterations = 0;

% Main loop

while true

iterations = iterations + 1;

% Store previous solution for convergence check

X\_old = X;

% Update solution vector using Gauss-Seidel iteration

for i = 1:n

sigma = 0;

for j = 1:n

if j ~= i

sigma = sigma + scaled\_A(i, j) \* X(j);

end

end

X(i) = (scaled\_B(i) - sigma) / scaled\_A(i, i);

end

% Check for convergence

if norm(X - X\_old, inf) < tolerance || iterations >= max\_iterations

break;

end

end

% Check for convergence

if iterations >= max\_iterations

warning('Seidel''s method did not converge within the specified maximum number of iterations');

end

% Undo scaling to obtain the final solution

X = X ./ scaling\_factors;

% Display result

disp('The solution vector X is:');

disp(X);

end

## Результати виконання програм

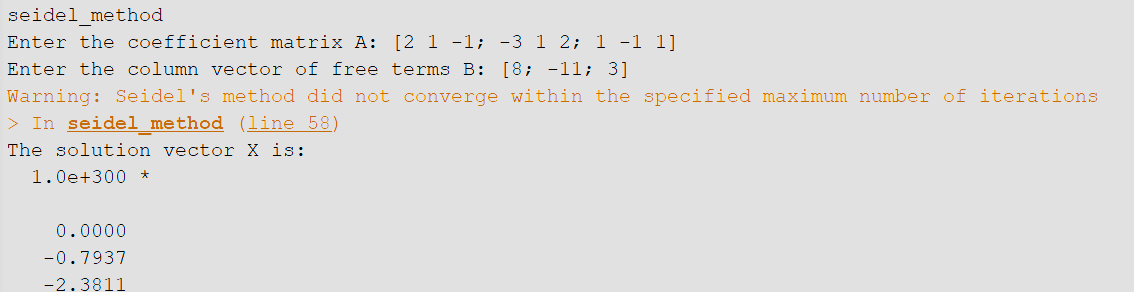


Рисунок 1.3 – «Обчислення seidel\_method»

Цей код реалізує метод Зейделя для розв'язання систем лінійних алгебраїчних рівнянь (СЛАР). Давайте розглянемо кожен блок коду:

1. Prompt user for input: Функція спочатку запитує користувача ввести коефіцієнт матриці A та стовпчик вільних членів B.
2. Constants for convergence criteria: Задаються константи для критеріїв збіжності: tolerance - допустиме значення різниці між двома послідовними ітераціями, та max\_iterations - максимальна кількість ітерацій.
3. Check if A is square: Перевіряється, чи матриця A є квадратною.
4. Check if number of equations matches length of B: Перевіряється, чи кількість рівнянь в системі відповідає довжині вектора вільних членів B.
5. Scale the system of equations to improve convergence: Система рівнянь масштабується для поліпшення збіжності методу.
6. Initialize solution vector and iteration counter: Ініціалізується вектор розв'язку X та лічильник ітерацій.
7. Main loop: Здійснюється головний цикл методу Зейделя. В кожній ітерації обчислюються нові значення невідомих шляхом ітеративного покращення.
8. Check for convergence: Перевіряється умова збіжності - якщо різниця між двома послідовними ітераціями менша за задане значення tolerance, або досягнуто максимальну кількість ітерацій, цикл завершується.
9. Display result: Виводиться отриманий вектор розв'язку X.

Цей код можна використовувати для розв'язання систем лінійних рівнянь методом Зейделя з заданими критеріями збіжності та максимальною кількістю ітерацій.

# Початок форми

# Завдання 1.4

Розробити програму для методу Гаусса-Джордана; організувати програму у формі функції. Програму можна писати будь-якою мовою програмування.

Вхідні параметри: A - матриця коефіцієнтів СЛАР, B - стовпчиковий вектор вільних членів. Вихідні параметри: X - вектор-стовпчик коренів СЛАР.

## Код завдання 1.3:

### gauss\_jordan\_method.m

function X = gauss\_jordan\_method()

% Prompt user for input

A = input('Enter the coefficient matrix A: ');

B = input('Enter the column vector of free terms B: ');

% Check if A is square

[n, m] = size(A);

if n ~= m

error('Coefficient matrix must be square');

end

% Check if number of equations matches length of B

if n ~= length(B)

error('Number of equations must match the length of the vector of free terms');

end

% Augmented matrix

AB = [A, B];

% Perform Gauss-Jordan elimination

for i = 1:n

% Find pivot row and swap rows if necessary

[~, max\_row] = max(abs(AB(i:n, i)));

max\_row = max\_row + i - 1;

if max\_row ~= i

AB([i max\_row], :) = AB([max\_row i], :);

end

% Normalize pivot row

AB(i, :) = AB(i, :) / AB(i, i);

% Eliminate non-zero elements below pivot

for j = 1:n

if j ~= i

AB(j, :) = AB(j, :) - AB(j, i) \* AB(i, :);

end

end

end

% Extract solution vector

X = AB(:, end);

end

## Результати виконання програми

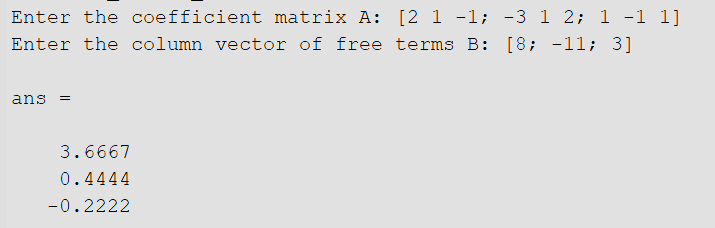


Рисунок 1.4 – «Обчислення gauss\_jordan\_method()»

Ця програма реалізує метод Гаусса-Джордана для розв'язання систем лінійних алгебраїчних рівнянь (СЛАР). Давайте розглянемо кожен блок коду та описемо його для лабораторної роботи:

1. Prompt user for input: Функція спочатку запитує користувача ввести коефіцієнт матриці A та стовпчик вільних членів B.
2. Check if A is square: Перевіряється, чи матриця A є квадратною.
3. Check if number of equations matches length of B: Перевіряється, чи кількість рівнянь в системі відповідає довжині вектора вільних членів B.
4. Augmented matrix: Створюється поширена матриця, яка об'єднує матрицю коефіцієнтів A та стовпчик вільних членів B.
5. Perform Gauss-Jordan elimination: Виконується метод Гаусса-Джордана для знаходження розв'язку СЛАР. У циклі для кожного рядка матриці:
   * Знаходиться рядок-півот та, за потреби, виконується обмін рядків для забезпечення ненульового півота.
   * Рядок-півот нормалізується.
   * Елементи нижче півота обнулюються шляхом віднімання відповідно зважених рядків.
6. Extract solution vector: Отримується вектор розв'язку з останнього стовпчика результуючої матриці.

Ця програма може бути використана для розв'язання СЛАР методом Гаусса-Джордана. Користувачеві надається можливість ввести власні дані, а результат обчислень виводиться на екран.

# Завдання 1.5

Розробити програму для методу Якобі; організувати програму у формі функції. Програму можна писати будь-якою мовою програмування.

Вхідні параметри: A - матриця коефіцієнтів СЛАР, B - стовпчиковий вектор вільних членів. Вихідні параметри: X - вектор-стовпчик коренів СЛАР.

## Код завдання 1.5:

function X = jacobi\_method()

% Prompt user for input

A = input('Enter the coefficient matrix A: ');

B = input('Enter the column vector of free terms B: ');

% Constants for convergence criteria

tolerance = 1e-9; % Tolerance for convergence

max\_iterations = 1000; % Maximum number of iterations

% Check if A is square

[n, m] = size(A);

if n ~= m

error('Coefficient matrix must be square');

end

% Check if number of equations matches length of B

if n ~= length(B)

error('Number of equations must match the length of the vector of free terms');

end

% Initialize solution vector

X = zeros(n, 1);

% Initialize iteration counter

iterations = 0;

% Main loop

while true

iterations = iterations + 1;

% Store previous solution for convergence check

X\_old = X;

% Update solution vector using Jacobi iteration

for i = 1:n

sigma = 0;

for j = 1:n

if j ~= i

sigma = sigma + A(i, j) \* X\_old(j);

end

end

X(i) = (B(i) - sigma) / A(i, i);

end

% Check for convergence

if norm(X - X\_old, inf) < tolerance || iterations >= max\_iterations

break;

end

end

% Check for convergence

if iterations >= max\_iterations

warning('Jacobi''s method did not converge within the specified maximum number of iterations');

end

end

## Результати виконання програми

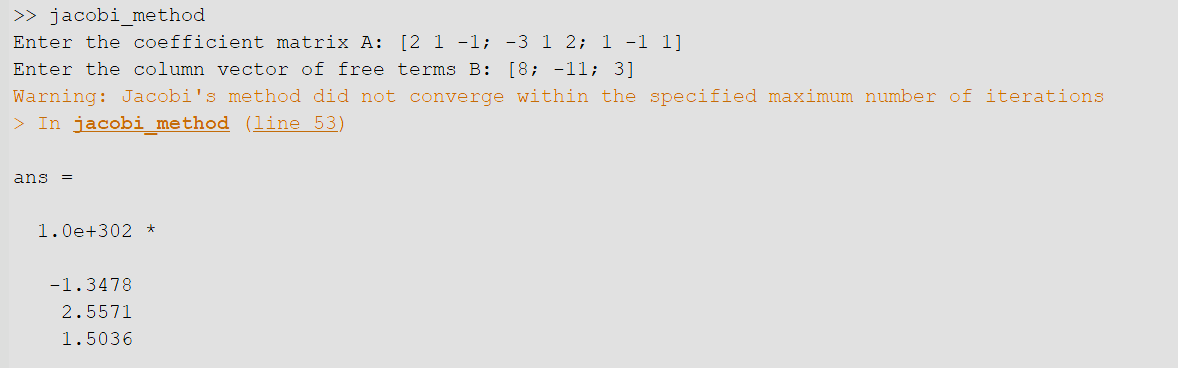


Рисунок 1.5 – «Обчислення jacobi\_method.m»

Ця програма реалізує метод Якобі для розв'язання систем лінійних алгебраїчних рівнянь (СЛАР). Розглянемо кожен блок коду та опишемо його для лабораторної роботи:

1. Prompt user for input: Функція спочатку запитує користувача ввести коефіцієнт матриці A та стовпчик вільних членів B.
2. Constants for convergence criteria: Задаються константи для критеріїв збіжності: tolerance - допустиме значення різниці між двома послідовними ітераціями, та max\_iterations - максимальна кількість ітерацій.
3. Check if A is square: Перевіряється, чи матриця A є квадратною.
4. Check if number of equations matches length of B: Перевіряється, чи кількість рівнянь в системі відповідає довжині вектора вільних членів B.
5. Initialize solution vector and iteration counter: Ініціалізується вектор розв'язку X та лічильник ітерацій.
6. Main loop: Здійснюється головний цикл методу Якобі. У кожній ітерації обчислюються нові значення невідомих на основі попередніх ітерацій.
7. Check for convergence: Перевіряється умова збіжності - якщо різниця між двома послідовними ітераціями менша за задане значення tolerance, або досягнуто максимальну кількість ітерацій, цикл завершується.
8. Display warning if maximum iterations reached: Якщо максимальна кількість ітерацій досягнута, виводиться попередження про незбіжність методу.

Ця програма може бути використана для розв'язання СЛАР методом Якобі. Користувачеві надається можливість ввести власні дані, а результат обчислень виводиться на екран.

# Завдання 1.6

Розробити програму для перевірки знайдених коренів СЛАР, організувати програму у вигляді функції.

Вхідні параметри: X - вектор-стовпець коренів СЛАР. Вихідні параметри: flag - булева змінна (true \ false).

## Код завдання 1.6:

function flag = check\_roots(method)

% Prompt user for input

A = input('Enter the coefficient matrix A: ');

B = input('Enter the column vector of free terms B: ');

% Constants

tolerance = 1e-9; % Tolerance for convergence

% Call the specified method to obtain roots

switch method

case 'cramer'

X = cramer\_method(A, B);

case 'gauss'

X = gauss\_method(A, B);

case 'seidel'

X = seidel\_method(A, B, tolerance, 1000); % Maximum iterations set to 1000

case 'jacobi'

X = jacobi\_method(A, B, tolerance, 1000); % Maximum iterations set to 1000

case 'gauss\_jordan'

X = gauss\_jordan\_method(A, B);

otherwise

error('Invalid method specified');

end

% Check if A is square

[n, m] = size(A);

if n ~= m

error('Coefficient matrix must be square');

end

% Check if number of equations matches length of B

if n ~= length(B)

error('Number of equations must match the length of the vector of free terms');

end

% Calculate residual vector

residual = A \* X - B;

% Check if residuals are within tolerance

flag = all(abs(residual) < tolerance);

end

## Результати виконання програми

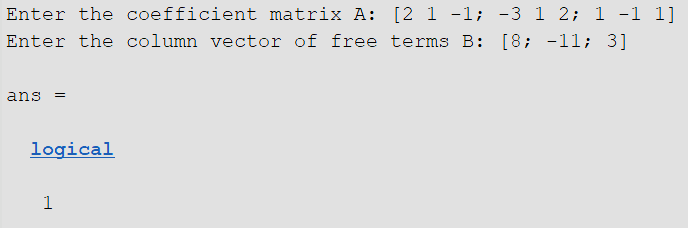


Рисунок 1.6 – «Обчислення check\_roots()»

У цій програмі створено функцію **check\_roots**, яка перевіряє коректність коренів системи лінійних алгебраїчних рівнянь (СЛАР), що розв'язується за допомогою різних числових методів.

Програма запитує в користувача введення матриці коефіцієнтів *A* та вектора вільних членів *B*. Після цього вона викликає вказаний метод (зазначений як аргумент **method**) для отримання коренів СЛАР. Метод може бути вибраний зі списку доступних методів, таких як метод Крамера, метод Гаусса, метод Сейделя, метод Якобі та метод Гауса-Жордана.

Отримані корені використовуються для обчислення залишкового вектора, який представляє різницю між вектором вільних членів *B* та добутком матриці коефіцієнтів *A* на вектор коренів *X*. Після цього програма перевіряє, чи всі елементи залишкового вектора менші за встановлене значення допустимої похибки (тобто, чи всі вони знаходяться в межах допустимої похибки).

Функція повертає логічне значення **flag**, яке показує, чи всі корені системи СЛАР були знайдені з вказаною точністю. Якщо **flag** дорівнює true, це означає, що всі корені були знайдені з потрібною точністю, в іншому випадку - false.

# Завдання 1.7

Для створення графічного інтерфейсу програмного забезпечення реалізуйте можливість вибору будь-якого методу, а також зчитування початкових даних з текстових файлів та клавіатури. Зміст текстових файлів має виглядати так: n - кількість рівнянь (невідомих), A - матриця коефіцієнтів системи лінійних алгебраїчних рівнянь, B - стовпчиковий вектор.

## Код завдання 1.7:

classdef MatrixCalculator < matlab.apps.AppBase

% Properties that correspond to app components

properties (Access = public)

UIFigure matlab.ui.Figure

InputdatafromtxtfileButton matlab.ui.control.Button

XTextArea matlab.ui.control.TextArea

XTextAreaLabel matlab.ui.control.Label

SolveButton matlab.ui.control.Button

InputyourdataTextArea matlab.ui.control.TextArea

InputyourdataTextAreaLabel matlab.ui.control.Label

ChoosemethodDropDown matlab.ui.control.DropDown

ChoosemethodDropDownLabel matlab.ui.control.Label

end

methods (Access = public)

function methodChoose = mapToEnum(~, value, SolverMethod)

switch value

case 'Cramer''s Method'

methodChoose = SolverMethod.CramersMethod;

case 'Gauss Method'

methodChoose = SolverMethod.GaussMethod;

case 'Seidel Method'

methodChoose = SolverMethod.SeidelMethod;

case 'Gauss-Jordan Method'

methodChoose = SolverMethod.GaussJordanMethod;

case 'Jacobi Method'

methodChoose = SolverMethod.JacobiMethod;

otherwise

methodChoose = []; % Handle unknown method

end

end

function X = cramerMethod(~, coefficientMatrix, vector)

% Check if coefficientMatrix is square

[rows, cols] = size(coefficientMatrix);

if rows ~= cols

error('Coefficient matrix must be square for Cramer''s method.');

end

% Check if coefficientMatrix is non-singular

if abs(det(coefficientMatrix)) < eps

error('Coefficient matrix must be non-singular for Cramer''s method.');

end

% Get the number of equations

n = rows;

% Initialize solution vector X

X = zeros(n, 1);

% Calculate determinant of coefficientMatrix

detA = det(coefficientMatrix);

% Solve for each variable using Cramer's rule

for i = 1:n

% Create a copy of coefficientMatrix and replace i-th column with the vector

tempMatrix = coefficientMatrix;

tempMatrix(:, i) = vector;

% Calculate determinant of modified matrix

detAi = det(tempMatrix);

% Calculate the i-th component of the solution vector

X(i) = detAi / detA;

end

end

function X = seidelMethod(~, coefficientMatrix, vector)

% Define constants for maximum iterations and tolerance

MAX\_ITERATIONS = 1000;

TOLERANCE = 1e-6;

% Check if coefficientMatrix is square

[rows, cols] = size(coefficientMatrix);

if rows ~= cols

error('Coefficient matrix must be square for Seidel method.');

end

% Check if coefficientMatrix is diagonally dominant

diagonal = abs(diag(coefficientMatrix));

offDiagonal = sum(abs(coefficientMatrix), 2) - diagonal;

if any(diagonal <= offDiagonal)

warning('Coefficient matrix is not diagonally dominant. Convergence may not be guaranteed.');

end

% Initialize solution vector X

X = zeros(rows, 1);

% Perform Seidel iteration

for iter = 1:MAX\_ITERATIONS

% Store the current solution vector

X\_old = X;

% Update each component of X using the latest available values

for i = 1:rows

sigma = coefficientMatrix(i, 1:i-1) \* X(1:i-1) + coefficientMatrix(i, i+1:end) \* X\_old(i+1:end);

X(i) = (vector(i) - sigma) / coefficientMatrix(i, i);

end

% Check for convergence

if norm(X - X\_old, inf) < TOLERANCE

fprintf('Converged after %d iterations.\n', iter);

break;

end

end

% Check for maximum iterations reached

if iter == MAX\_ITERATIONS

warning('Maximum iterations reached without convergence.');

end

end

function X = gaussMethod(~, coefficientMatrix, vector)

% Check if coefficientMatrix is square

[rows, cols] = size(coefficientMatrix);

if rows ~= cols

error('Coefficient matrix must be square for Gauss method.');

end

% Check if coefficientMatrix is non-singular

if abs(det(coefficientMatrix)) < eps

error('Coefficient matrix must be non-singular for Gauss method.');

end

% Get the number of equations

n = rows;

% Augment the coefficient matrix with the vector

augmentedMatrix = [coefficientMatrix, vector];

% Perform forward elimination

for i = 1:n-1

% Partial pivoting

[maxVal, maxIndex] = max(abs(augmentedMatrix(i:n,i)));

maxIndex = maxIndex + i - 1;

if maxIndex ~= i

augmentedMatrix([i, maxIndex], :) = augmentedMatrix([maxIndex, i], :);

end

% Eliminate coefficients below pivot

for j = i+1:n

factor = augmentedMatrix(j,i) / augmentedMatrix(i,i);

augmentedMatrix(j,:) = augmentedMatrix(j,:) - factor \* augmentedMatrix(i,:);

end

end

% Perform back substitution

X = zeros(n, 1);

X(n) = augmentedMatrix(n,n+1) / augmentedMatrix(n,n);

for i = n-1:-1:1

X(i) = (augmentedMatrix(i,n+1) - augmentedMatrix(i,i+1:n) \* X(i+1:n)) / augmentedMatrix(i,i);

end

end

function X = gaussJordanMethod(~, coefficientMatrix, vector)

% Check if coefficientMatrix is square

[rows, cols] = size(coefficientMatrix);

if rows ~= cols

error('Coefficient matrix must be square for Gauss-Jordan method.');

end

% Check if coefficientMatrix is non-singular

if abs(det(coefficientMatrix)) < eps

error('Coefficient matrix must be non-singular for Gauss-Jordan method.');

end

% Get the number of equations

n = rows;

% Augment the coefficient matrix with the vector

augmentedMatrix = [coefficientMatrix, vector];

% Perform forward elimination

for i = 1:n

% Partial pivoting

[maxVal, maxIndex] = max(abs(augmentedMatrix(i:n,i)));

maxIndex = maxIndex + i - 1;

if maxIndex ~= i

augmentedMatrix([i, maxIndex], :) = augmentedMatrix([maxIndex, i], :);

end

% Eliminate coefficients above and below pivot

for j = [1:i-1, i+1:n]

factor = augmentedMatrix(j,i) / augmentedMatrix(i,i);

augmentedMatrix(j,:) = augmentedMatrix(j,:) - factor \* augmentedMatrix(i,:);

end

end

% Perform back substitution

for i = 1:n

augmentedMatrix(i,:) = augmentedMatrix(i,:) / augmentedMatrix(i,i);

for j = [1:i-1, i+1:n]

augmentedMatrix(j,:) = augmentedMatrix(j,:) - augmentedMatrix(j,i) \* augmentedMatrix(i,:);

end

end

% Extract the solution vector

X = augmentedMatrix(:, end);

end

function X = jacobiMethod(~, coefficientMatrix, vector)

% Define constants for maximum iterations and tolerance

MAX\_ITERATIONS = 1000;

TOLERANCE = 1e-6;

% Check if coefficientMatrix is square

[rows, cols] = size(coefficientMatrix);

if rows ~= cols

error('Coefficient matrix must be square for Jacobi method.');

end

% Check if coefficientMatrix is diagonally dominant

diagonal = abs(diag(coefficientMatrix));

offDiagonal = sum(abs(coefficientMatrix), 2) - diagonal;

if any(diagonal <= offDiagonal)

warning('Coefficient matrix is not diagonally dominant. Convergence may not be guaranteed.');

end

% Initialize solution vector X

X = zeros(rows, 1);

% Perform Jacobi iteration

for iter = 1:MAX\_ITERATIONS

% Store the current solution vector

X\_old = X;

% Compute the new solution vector

for i = 1:rows

sigma = 0;

for j = 1:cols

if j ~= i

sigma = sigma + coefficientMatrix(i, j) \* X\_old(j);

end

end

X(i) = (vector(i) - sigma) / coefficientMatrix(i, i);

end

% Check for convergence

if norm(X - X\_old, inf) < TOLERANCE

fprintf('Converged after %d iterations.\n', iter);

break;

end

end

% Check for maximum iterations reached

if iter == MAX\_ITERATIONS

warning('Maximum iterations reached without convergence.');

end

end

function [coefficientMatrix, vector, numEquations] = parseTextAreaData(app)

% Get the content of the TextArea

inputData = app.InputyourdataTextArea.Value;

% Split the input by newline characters

lines = split(inputData, '\n');

% Extract the number of equations

numEquations = str2double(lines{1});

% Initialize variables for the coefficient matrix and vector

coefficientMatrix = zeros(numEquations, numEquations);

vector = zeros(numEquations, 1);

% Parse the input data to populate the coefficient matrix and vector

for i = 2:numEquations+1

% Split each line by spaces

elements = str2double(strsplit(lines{i}));

% Check if the number of elements is correct

if numel(elements) ~= numEquations + 1

error('Invalid input format: each line should contain %d coefficients and one vector element.', numEquations);

end

% Extract the coefficients (excluding the last element which is the vector entry)

coefficientMatrix(i-1, :) = elements(1:end-1);

% Extract the vector entry

vector(i-1) = elements(end);

end

end

end

% Callbacks that handle component events

methods (Access = private)

% Value changed function: ChoosemethodDropDown

function ChoosemethodDropDownValueChanged(app, event)

end

% Button pushed function: SolveButton

function SolveButtonPushed(app, event)

[coefficientMatrix, vector, numEquations] = parseTextAreaData(app);

% Check if the input data is valid

if isnan(numEquations) || numEquations <= 0 || numel(vector) ~= numEquations || any(isnan(coefficientMatrix(:)))

% Display a warning message if the input data is invalid

warning('Invalid input data.');

return;

end

% Get the selected method from the dropdown menu

selectedMethod = app.ChoosemethodDropDown.Value;

% Validate the selected method

supportedMethods = {'Cramer''s Method', 'Gauss Method', 'Seidel Method', 'Gauss-Jordan Method', 'Jacobi Method'};

if ~ismember(selectedMethod, supportedMethods)

% Display a warning message if the selected method is not supported

warning('Selected method is not supported.');

return; % Exit the function

end

% Now you have the coefficient matrix, vector, and selected method, you can proceed with solving the equations

% Call the corresponding solver method based on the selected method

switch selectedMethod

case 'Cramer''s Method'

% Call the Cramer's method solver function

X = cramerMethod(app, coefficientMatrix, vector);

case 'Gauss Method'

% Call the Gauss method solver function

X = gaussMethod(app, coefficientMatrix, vector);

case 'Seidel Method'

% Call the Seidel method solver function

X = seidelMethod(app, coefficientMatrix, vector);

case 'Gauss-Jordan Method'

% Call the Gauss-Jordan method solver function

X = gaussJordanMethod(app, coefficientMatrix, vector);

case 'Jacobi Method'

% Call the Jacobi method solver function

X = jacobiMethod(app, coefficientMatrix, vector);

end

% Format the solution as a string in vertical order

solutionStr = ''; % Initialize an empty string

for i = 1:numel(X)

% Convert the number to a string

numStr = num2str(X(i));

% Append the number to the solution string

solutionStr = [solutionStr numStr newline];

end

% Set the solution string as the value of the XTextArea

app.XTextArea.Value = solutionStr;

end

% Button pushed function: InputdatafromtxtfileButton

function InputdatafromtxtfileButtonPushed(app, event)

% Open a file dialog to select a text file

[filename, path] = uigetfile({'\*.txt','Text files (\*.txt)'},'Select Text File');

% Check if the user canceled the selection

if isequal(filename, 0)

disp('User canceled the operation.');

return;

end

% Read the contents of the selected file

try

fileContent = fileread(fullfile(path, filename));

catch

disp('Error reading file.');

return;

end

% Display the file content in the InputyourdataTextArea

app.InputyourdataTextArea.Value = fileContent;

end

end

% Component initialization

methods (Access = private)

% Create UIFigure and components

function createComponents(app)

% Create UIFigure and hide until all components are created

app.UIFigure = uifigure('Visible', 'off');

app.UIFigure.Color = [1 1 1];

colormap(app.UIFigure, 'cool');

app.UIFigure.Position = [100 100 640 480];

app.UIFigure.Name = 'MATLAB App';

% Create ChoosemethodDropDownLabel

app.ChoosemethodDropDownLabel = uilabel(app.UIFigure);

app.ChoosemethodDropDownLabel.HorizontalAlignment = 'right';

app.ChoosemethodDropDownLabel.Position = [210 444 90 22];

app.ChoosemethodDropDownLabel.Text = 'Choose method';

% Create ChoosemethodDropDown

app.ChoosemethodDropDown = uidropdown(app.UIFigure);

app.ChoosemethodDropDown.Items = {'Cramer''s Method', 'Gauss Method', 'Seidel Method', 'Gauss-Jordan Method', 'Jacobi Method'};

app.ChoosemethodDropDown.ValueChangedFcn = createCallbackFcn(app, @ChoosemethodDropDownValueChanged, true);

app.ChoosemethodDropDown.Position = [315 444 147 22];

app.ChoosemethodDropDown.Value = 'Cramer''s Method';

% Create InputyourdataTextAreaLabel

app.InputyourdataTextAreaLabel = uilabel(app.UIFigure);

app.InputyourdataTextAreaLabel.HorizontalAlignment = 'center';

app.InputyourdataTextAreaLabel.Position = [278 349 85 22];

app.InputyourdataTextAreaLabel.Text = 'Input your data';

% Create InputyourdataTextArea

app.InputyourdataTextArea = uitextarea(app.UIFigure);

app.InputyourdataTextArea.HorizontalAlignment = 'center';

app.InputyourdataTextArea.Placeholder = 'Put the data as the instruction recomends';

app.InputyourdataTextArea.Position = [210 233 222 105];

% Create SolveButton

app.SolveButton = uibutton(app.UIFigure, 'push');

app.SolveButton.ButtonPushedFcn = createCallbackFcn(app, @SolveButtonPushed, true);

app.SolveButton.BackgroundColor = [0 0.4471 0.7412];

app.SolveButton.FontColor = [1 1 1];

app.SolveButton.Position = [271 37 100 23];

app.SolveButton.Text = 'Solve';

% Create XTextAreaLabel

app.XTextAreaLabel = uilabel(app.UIFigure);

app.XTextAreaLabel.HorizontalAlignment = 'center';

app.XTextAreaLabel.Position = [240 145 25 22];

app.XTextAreaLabel.Text = 'X';

% Create XTextArea

app.XTextArea = uitextarea(app.UIFigure);

app.XTextArea.Editable = 'off';

app.XTextArea.HorizontalAlignment = 'center';

app.XTextArea.Placeholder = 'Result';

app.XTextArea.Position = [275 113 126 86];

% Create InputdatafromtxtfileButton

app.InputdatafromtxtfileButton = uibutton(app.UIFigure, 'push');

app.InputdatafromtxtfileButton.ButtonPushedFcn = createCallbackFcn(app, @InputdatafromtxtfileButtonPushed, true);

app.InputdatafromtxtfileButton.Position = [254 396 134 23];

app.InputdatafromtxtfileButton.Text = 'Input data from .txt file';

% Show the figure after all components are created

app.UIFigure.Visible = 'on';

end

end

% App creation and deletion

methods (Access = public)

% Construct app

function app = MatrixCalculator

% Create UIFigure and components

createComponents(app)

% Register the app with App Designer

registerApp(app, app.UIFigure)

if nargout == 0

clear app

end

end

% Code that executes before app deletion

function delete(app)

% Delete UIFigure when app is deleted

delete(app.UIFigure)

end

end

end

## Результати виконання програми

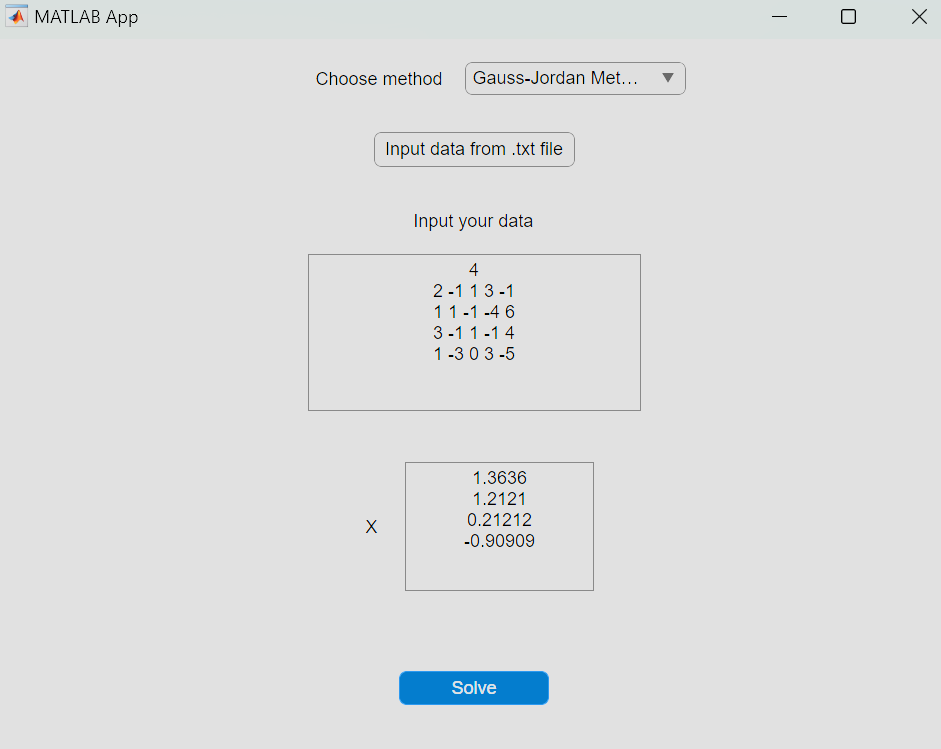


Рисунок 1.7 – «Обчислення MatrixCalculator»

У цьому завданні було створено графічний інтерфейс для програми, яка вирішує систему лінійних алгебраїчних рівнянь (СЛАР) за допомогою різних числових методів, таких як метод Крамера, метод Гаусса, метод Сейделя, метод Гауса-Жордана та метод Якобі.

Графічний інтерфейс надає користувачеві можливість вибору методу для розв'язання СЛАР, а також зчитування вхідних даних з текстових файлів або введення їх з клавіатури. Крім того, в програмі реалізовані перевірки на введення коректних даних, таких як перевірка на квадратність матриці коефіцієнтів, співпадіння розмірності матриці з вектором вільних членів тощо.

В кожній функції рішення методу також реалізована перевірка на діагональну домінантність матриці, яка відображається як попередження користувачу у разі її відсутності, що допомагає покращити збіжність числових методів.

Цей графічний інтерфейс спрощує використання числових методів для розв'язання СЛАР, забезпечуючи зручний інтерфейс для введення даних та отримання результатів.

# **ВИСНОВОК**

Спринт з передбачає створення програмного інтерфейсу, який дозволяє користувачу вирішувати системи лінійних алгебраїчних рівнянь за допомогою різних методів. Цей інтерфейс дозволяє користувачу вибирати метод розв'язку, вводити початкові дані з клавіатури або зчитувати їх з файлів. Після введення даних програма розв'язує систему за допомогою обраного методу та виводить результат у текстовому форматі. Такий підхід дозволяє зручно та ефективно вирішувати чисельні задачі з лінійної алгебри.