

Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики

Лабораторна робота №1

Чисельні методи в інформатиці

“Розв’язок систем рівнянь за допомогою
прямих та ітераційних методів”

Варіант №4

Виконав студент групи ІПС-31
Міцкевич Костянтин Олександрович

Київ — 2025

Постановка задачі

Знайти розв'язок систем рівнянь:

1) Методом Гаусса, знайти визначник та обернену матрицю

$$\begin{array}{cccccc} 4 & 3 & 1 & 0 & x_1 & 14 \\ -2 & 2 & 6 & 1 & x_2 & 31 \\ 0 & 5 & 2 & 3 & x_3 & 33 \\ 0 & 1 & 2 & 7 & x_4 & 45 \end{array}$$

2) Методом прогонки:

$$\begin{array}{cccccc} 1 & 2 & 0 & x_1 & 5 \\ 2 & 2 & 3 & x_2 & 15 \\ 0 & 3 & 2 & x_3 & 12 \end{array}$$

3) Методом Зейделя:

$$\begin{array}{cccccc} 4 & 0 & 1 & 0 & x_1 & 7 \\ 0 & 3 & 0 & 2 & x_2 & 14 \\ 1 & 0 & 5 & 1 & x_3 & 20 \\ 0 & 2 & 1 & 4 & x_4 & 23 \end{array}$$

Теоретичні відомості та обґрунтування

Метод Гаусса:

Для зменшення обчислювальної похибки метод Гаусса дозволяє обрати головний елемент: по стовпцях, по рядках, за всією матрицею. Ми ж будемо розв'язувати СЛАР $Ax = b$ з вибором головного елемента по стовпцях.

Покладемо $A_0 = A$. Ведучим елементом обираємо максимальний по модулю елемент стовпця, $a_{lk} = \max_i |a_{ik}^{(k-1)}|$, $i = \overline{k, n}$. Для того щоб ведучий елемент зайняв відповідне місце, переставляються рядки k та l в матриці A_{k-1} за допомогою матриці перестановок: $A_k = P_k A_{k-1}$ де P_k – це матриця перестановок, отримана з одиничної матриці перестановкою k та l рядків.

Прямий хід Гаусса в матричній формі: $A_k = M_k \tilde{A}$, де M_k – матриця розмірності $n \times n$:

$$M_k = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & m_{kk} & 0 & \dots & 0 \\ 0 & 0 & \dots & m_{(k+1)k} & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & m_{nk} & 0 & \dots & 1 \end{pmatrix}$$

$$m_{kk} = \frac{1}{\tilde{a}_{kk}^{(k)}}, m_{ik} = \frac{-\tilde{a}_{ik}^{(k)}}{\tilde{a}_{kk}^{(k)}}, i = \overline{k+1, n}$$

За допомогою прямого ходу методу Гаусса в матричній формі:

$M_n P_n \dots M_2 P_2 M_1 P_1 A x = M_n P_n \dots M_2 P_2 M_1 P_1 b$ зводимо систему до вигляду:

$$\begin{aligned} x_1 + a_{12}^{(1)} x_2 + \dots + a_{1n}^{(1)} x_n &= a_{1(n+1)}^{(1)}; \\ x_2 + \dots + a_{2n}^{(2)} x_n &= a_{2(n+1)}^{(2)}; \\ x_n &= a_{n(n+1)}^{(n)} \end{aligned}$$

Розв'язок знаходимо за допомогою зворотнього ходу Гаусса:

$$x_n = a_{n(n+1)}^{(n)}, x_i = a_{i(n+1)}^{(i)} - \sum_{j=i+1}^n a_{ij}^{(i)} x_j, i = \overline{n-1, 1}$$

Складність алгоритму: $Q(n) = \frac{2}{3}n^3 + O(n^2)$

Зауваження. Методом Гаусса з вибором головного можна

знайти визначник:

$\det A = (-1)^p \tilde{a}_{11}^1 \tilde{a}_{22}^2 \dots \tilde{a}_{nn}^n = (-1)^p \tilde{a}_{11}^0 \tilde{a}_{22}^1 \dots \tilde{a}_{nn}^{n-1}$, де p – кількість перестановок.

Метод Прогонки:

Нехай маємо систему вигляду:

$$\begin{aligned} -c_0 y_0 + b_0 y_1 &= -f_0 \\ \dots \dots \dots \\ a_i y_{i-1} - c_i y_i + b_i y_{i+1} &= -f_i \\ \dots \dots \dots \\ a_n y_{n-1} - c_n y_n &= -f_n \end{aligned}$$

Достатня умова стійкості. Нехай коефіцієнти $a_0, b_0 = 0$; $c_0, c_n \neq 0$; $a_i, b_i, c_i = 0$; $i = \overline{1, n-1}$. Якщо виконуються умови:

$$1) |c_i| \geq |a_i| + |b_i|, i = \overline{0, n}$$

$$2) \exists i: c_i > a_i + b_i, \text{ то метод є стійким: } |\alpha_i| \leq 1, |z| > 1, i = \overline{1, n}.$$

Прямий хід метода Гаусса в методі прогонки відповідає знаходженню прогонкових коефіцієнтів:

$$\alpha_1 = \frac{b_0}{c_0} \quad \beta_1 = \frac{a_0}{c_0} \quad a_{i_1} = \frac{b_i}{z_i}$$

$$\beta_{i+1} = \frac{f_i + a_i \beta_i}{z_i} \quad z_i = c_i - \alpha_i a_i \quad i = \overline{1, n-1}$$

Зворотній хід:

$$y_n = \frac{f_n + a_n \beta_n}{z_n}, \quad y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1}, \quad i = \overline{n-1, 0}$$

Складність методу прогонки: $Q(n) = 8n - 2$

Метод Зейделя:

Ітераційний метод для розв'язання СЛАР типу $Ax=b$. Розв'язок знаходимо із заданою точністю ε . Початкове наближення обираємо довільним чином.

$$\text{Ітераційний процес має вигляд: } x_i^{k+1} = - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{k+1} - \sum_{j=i+1}^n \frac{a_{ij}}{a_{ii}} x_j^k + \frac{b_i}{a_{ii}}$$

Достатня умова збіжності 1: Якщо $\forall i: i = \overline{1, n}$ виконується нерівність

$$|a_{ii}| \geq \sum_{j=1, j \neq i}^n |a_{ij}|, \text{ то ітераційний процес методу Зейделя збігається з лінійною}$$

швидкістю.

Достатня умова збіжності 2: Якщо $A=A^T > 0$, то ітераційний процес методу Зейделя збігається з лінійною швидкістю.

Умова припинення: $\|x^n - x^{n-1}\| \leq \varepsilon$

Необхідні і достатні умови збіжності: Для $\forall x^0$ ітераційний процес методу Зейделя збігається тоді і тільки тоді, коли $\lambda < 1$, де λ - корені нелінійного

рівняння:
$$\det \begin{pmatrix} \lambda a_{11} & a_{12} & \dots & a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ \lambda a_{n1} & \lambda a_{n2} & \dots & \lambda a_{nn} \end{pmatrix} = 0$$

Хід роботи

Мова програмування – Python;

Використані бібліотеки: numpy (робота з масивами, векторизовані обчислення многочлена, прості перетворення), math (для розрахунку апріорної кількості ітерацій).

Метод Гаусса:

Розпочинаємо метод з ініціалізації матриці та початку прямого ходу пошуку коренів, паралельно створюємо одиничну матрицю для обрахунку оберненої.

```
2025-10-13 22:37:55 - gauss_method.py - INFO - =====
2025-10-13 22:37:55 - gauss_method.py - INFO - RUN: Gaussian elimination with partial pivoting
2025-10-13 22:37:55 - gauss_method.py - INFO - =====
2025-10-13 22:37:55 - gauss_method.py - INFO - Initial augmented matrix (A|b||I):
2025-10-13 22:37:55 - gauss_method.py - INFO -
[ 4.00000, 3.00000, 1.00000, 0.00000] | 14.00000 || [ 1.00000, 0.00000, 0.00000, 0.00000]
[ -2.00000, 2.00000, 6.00000, 1.00000] | 31.00000 || [ 0.00000, 1.00000, 0.00000, 0.00000]
[ 0.00000, 5.00000, 2.00000, 3.00000] | 33.00000 || [ 0.00000, 0.00000, 1.00000, 0.00000]
[ 0.00000, 1.00000, 2.00000, 7.00000] | 45.00000 || [ 0.00000, 0.00000, 0.00000, 1.00000]
2025-10-13 22:37:55 - gauss_method.py - INFO - Step 1: working column = 0, looking for max |element| in rows 0..3
2025-10-13 22:37:55 - gauss_method.py - INFO - Largest |element| in column 0 is in row 0 (value = 4.000000)
2025-10-13 22:37:55 - gauss_method.py - INFO - No swap needed (largest element already in position).
2025-10-13 22:37:55 - gauss_method.py - INFO - For row 1: multiplier = A[1][0] / A[0][0] = -2.000000 / 4.000000 = -0.500000
2025-10-13 22:37:55 - gauss_method.py - INFO - After subtracting -0.500000*row(0) from row 1:
2025-10-13 22:37:55 - gauss_method.py - INFO -
[ 4.00000, 3.00000, 1.00000, 0.00000] | 14.00000 || [ 1.00000, 0.00000, 0.00000, 0.00000]
[ 0.00000, 3.50000, 6.50000, 1.00000] | 38.00000 || [ 0.50000, 1.00000, 0.00000, 0.00000]
[ 0.00000, 5.00000, 2.00000, 3.00000] | 33.00000 || [ 0.00000, 0.00000, 1.00000, 0.00000]
[ 0.00000, 1.00000, 2.00000, 7.00000] | 45.00000 || [ 0.00000, 0.00000, 0.00000, 1.00000]
2025-10-13 22:37:55 - gauss_method.py - INFO - For row 2: multiplier = A[2][0] / A[0][0] = 0.000000 / 4.000000 = 0.000000
2025-10-13 22:37:55 - gauss_method.py - INFO - After subtracting 0.000000*row(0) from row 2:
2025-10-13 22:37:55 - gauss_method.py - INFO -
[ 4.00000, 3.00000, 1.00000, 0.00000] | 14.00000 || [ 1.00000, 0.00000, 0.00000, 0.00000]
[ 0.00000, 3.50000, 6.50000, 1.00000] | 38.00000 || [ 0.50000, 1.00000, 0.00000, 0.00000]
[ 0.00000, 5.00000, 2.00000, 3.00000] | 33.00000 || [ 0.00000, 0.00000, 1.00000, 0.00000]
[ 0.00000, 1.00000, 2.00000, 7.00000] | 45.00000 || [ 0.00000, 0.00000, 0.00000, 1.00000]
2025-10-13 22:37:55 - gauss_method.py - INFO - For row 3: multiplier = A[3][0] / A[0][0] = 0.000000 / 4.000000 = 0.000000
2025-10-13 22:37:55 - gauss_method.py - INFO - After subtracting 0.000000*row(0) from row 3:
2025-10-13 22:37:55 - gauss_method.py - INFO -
[ 4.00000, 3.00000, 1.00000, 0.00000] | 14.00000 || [ 1.00000, 0.00000, 0.00000, 0.00000]
[ 0.00000, 3.50000, 6.50000, 1.00000] | 38.00000 || [ 0.50000, 1.00000, 0.00000, 0.00000]
[ 0.00000, 5.00000, 2.00000, 3.00000] | 33.00000 || [ 0.00000, 0.00000, 1.00000, 0.00000]
[ 0.00000, 1.00000, 2.00000, 7.00000] | 45.00000 || [ 0.00000, 0.00000, 0.00000, 1.00000]
2025-10-13 22:37:55 - gauss_method.py - INFO - End of step 1. Matrix after elimination in column 0:
2025-10-13 22:37:55 - gauss_method.py - INFO -
[ 4.00000, 3.00000, 1.00000, 0.00000] | 14.00000 || [ 1.00000, 0.00000, 0.00000, 0.00000]
[ 0.00000, 3.50000, 6.50000, 1.00000] | 38.00000 || [ 0.50000, 1.00000, 0.00000, 0.00000]
[ 0.00000, 5.00000, 2.00000, 3.00000] | 33.00000 || [ 0.00000, 0.00000, 1.00000, 0.00000]
[ 0.00000, 1.00000, 2.00000, 7.00000] | 45.00000 || [ 0.00000, 0.00000, 0.00000, 1.00000]
```

Повторюємо кроки допоки не зведемо матрицю до верхньої трикутної форми, паралельно продовжуючи обраховувати обернену матрицю, а також перевіряти чи на діагональних елементах у нас нема нулів.

Крім цього рахуємо кількість перестановок, щоб порахувати визначник.

В кінці прямого ходу отримуємо наступний результат:

```
2025-10-13 22:37:55 - gauss_method.py - INFO - Forward elimination complete. Upper-triangular form (augmented A|b||I'):
2025-10-13 22:37:55 - gauss_method.py - INFO -
[ 4.00000, 3.00000, 1.00000, 0.00000] | 14.00000 || [ 1.00000, 0.00000, 0.00000, 0.00000]
[ 0.00000, 5.00000, 2.00000, 3.00000] | 33.00000 || [ 0.00000, 0.00000, 1.00000, 0.00000]
[ 0.00000, 0.00000, 5.10000, -1.10000] | 14.90000 || [ 0.50000, 1.00000, -0.70000, 0.00000]
[ 0.00000, 0.00000, 0.00000, 6.74510] | 33.72549 || [-0.15686, -0.31373, 0.01961, 1.00000]
```

Маючи верхню трикутну матрицю, можемо знайти детермінант:

```
2025-10-13 22:37:55 - gauss_method.py - INFO - Determinant (product of diagonals x sign): det = -688.0000000000
```

$\text{Det}(A) = -688$;

Тепер можемо почати зворотній хід для знаходження коренів СЛАР.

Паралельно передаємо в метод зворотнього ходу незавершену обернену матрицю:

```
2025-10-13 22:37:55 - gauss_method.py - INFO - Determinant (product of diagonals x sign): det = -688.0000000000
2025-10-13 22:37:55 - gauss_method.py - INFO - Back substitution (solving system and computing inverse):
2025-10-13 22:37:55 - gauss_method.py - INFO - x[3] = (b[3] - Σ) / A[3][3] = (33.72549 - 0.00000) / 6.74510 = 5.0000000000
2025-10-13 22:37:55 - gauss_method.py - INFO - x[2] = (b[2] - Σ) / A[2][2] = (14.90000 - -5.50000) / 5.10000 = 4.0000000000
2025-10-13 22:37:55 - gauss_method.py - INFO - x[1] = (b[1] - Σ) / A[1][1] = (33.00000 - 23.00000) / 5.00000 = 2.0000000000
2025-10-13 22:37:55 - gauss_method.py - INFO - x[0] = (b[0] - Σ) / A[0][0] = (14.00000 - 10.00000) / 4.00000 = 1.0000000000
2025-10-13 22:37:55 - gauss_method.py - INFO - Solution x: [ 1.00000, 2.00000, 4.00000, 5.00000]
2025-10-13 22:37:55 - gauss_method.py - INFO - Computing inverse matrix (back substitution for each column):
2025-10-13 22:37:55 - gauss_method.py - INFO - Inverse matrix A-1:
2025-10-13 22:37:55 - gauss_method.py - INFO - [ 0.24419, -0.01163, -0.15552, 0.06831]
2025-10-13 22:37:55 - gauss_method.py - INFO - [ -0.02326, -0.04651, 0.25291, -0.10174]
2025-10-13 22:37:55 - gauss_method.py - INFO - [ 0.09302, 0.18605, -0.13663, 0.03198]
2025-10-13 22:37:55 - gauss_method.py - INFO - [ -0.02326, -0.04651, 0.00291, 0.14826]
2025-10-13 22:37:55 - gauss_method.py - INFO - Check: A x A-1 =
2025-10-13 22:37:55 - gauss_method.py - INFO - [ 1.00000, 0.00000, 0.00000, 0.00000]
2025-10-13 22:37:55 - gauss_method.py - INFO - [ 0.00000, 1.00000, 0.00000, 0.00000]
2025-10-13 22:37:55 - gauss_method.py - INFO - [ 0.00000, 0.00000, 1.00000, 0.00000]
2025-10-13 22:37:55 - gauss_method.py - INFO - [ 0.00000, 0.00000, 0.00000, 1.00000]
2025-10-13 22:37:55 - gauss_method.py - INFO - SUMMARY:
2025-10-13 22:37:55 - gauss_method.py - INFO - Determinant det(A) = -688.0000000000
2025-10-13 22:37:55 - gauss_method.py - INFO - Solution x = [ 1.00000, 2.00000, 4.00000, 5.00000]
2025-10-13 22:37:55 - gauss_method.py - INFO - =====
2025-10-13 22:37:55 - main.py - INFO - Gaussian elimination completed successfully
```

В кінці перевіряємо обернену матрицю, помноживши її на початкову.

Метод Прогонки:

Починаємо метод прогонки з перевірки достатніх умов стійкості:

```
2025-10-13 22:50:46 - main.py - INFO - User selected: Thomas algorithm
2025-10-13 22:50:46 - thomas_method.py - INFO - =====
2025-10-13 22:50:46 - thomas_method.py - INFO - RUN: Thomas algorithm (tridiagonal method)
2025-10-13 22:50:46 - thomas_method.py - INFO - =====
2025-10-13 22:50:46 - thomas_method.py - INFO - 
===== TEST 1 =====
2025-10-13 22:50:46 - thomas_method.py - INFO - Matrix A:
[[1. 2. 0.]
 [2. 2. 3.]
 [0. 3. 2.]]
2025-10-13 22:50:46 - thomas_method.py - INFO - Vector b: [ 5.00000, 15.00000, 12.00000]
2025-10-13 22:50:46 - thomas_method.py - INFO - === Checking tridiagonality ===
2025-10-13 22:50:46 - thomas_method.py - INFO - Matrix is tridiagonal: True
2025-10-13 22:50:46 - thomas_method.py - INFO - === Checking stability conditions ===
2025-10-13 22:50:46 - thomas_method.py - INFO - i=0: |c_i| = 1.000000, |a_i| + |b_i| = 2.000000
2025-10-13 22:50:46 - thomas_method.py - INFO - i=1: |c_i| = 2.000000, |a_i| + |b_i| = 5.000000
2025-10-13 22:50:46 - thomas_method.py - INFO - i=2: |c_i| = 2.000000, |a_i| + |b_i| = 3.000000
2025-10-13 22:50:46 - thomas_method.py - WARNING - Condition |c_i| >= |a_i| + |b_i| does not hold for all i.
2025-10-13 22:50:46 - thomas_method.py - WARNING - No i found where |c_i| > |a_i| + |b_i|, method may be unstable.
2025-10-13 22:50:46 - thomas_method.py - INFO -
```

Бачимо, що умови не пройшли, а отже розв'язок може бути не стійким, тому було прийнято рішення перевірити коректність коду на СЛАР з методички.

```
===== TEST 2 =====
2025-10-13 22:50:46 - thomas_method.py - INFO - Matrix A:
[[1. 1. 0.]
 [1. 3. 2.]
 [0. 1. 2.]]
2025-10-13 22:50:46 - thomas_method.py - INFO - Vector b: [ 1.00000, 1.00000, 1.00000]
2025-10-13 22:50:46 - thomas_method.py - INFO - === Checking tridiagonality ===
2025-10-13 22:50:46 - thomas_method.py - INFO - Matrix is tridiagonal: True
2025-10-13 22:50:46 - thomas_method.py - INFO - === Checking stability conditions ===
2025-10-13 22:50:46 - thomas_method.py - INFO - i=0: |c_i| = 1.000000, |a_i| + |b_i| = 1.000000
2025-10-13 22:50:46 - thomas_method.py - INFO - i=1: |c_i| = 3.000000, |a_i| + |b_i| = 3.000000
2025-10-13 22:50:46 - thomas_method.py - INFO - i=2: |c_i| = 2.000000, |a_i| + |b_i| = 1.000000
2025-10-13 22:50:46 - thomas_method.py - INFO - Condition |c_i| >= |a_i| + |b_i| holds for all i (possible borderline stability)
.
2025-10-13 22:50:46 - thomas_method.py - INFO - There exists i where |c_i| > |a_i| + |b_i|, Thomas algorithm is stable
```

Дана система стабільна, може приступати до пошуку її коренів зі знаходження прогонкових коефіцієнтів:

```
2025-10-13 22:50:46 - thomas_method.py - INFO - === Forward sweep (transformation) ===
2025-10-13 22:50:46 - thomas_method.py - INFO - i=0:  $\alpha_0 = -1.000000$ ,  $\beta_0 = 1.000000$ ,  $z_0 = -1.000000$ 
2025-10-13 22:50:46 - thomas_method.py - INFO - i=1:  $\alpha_1 = -1.000000$ ,  $\beta_1 = -0.000000$ ,  $z_1 = -2.000000$ 
2025-10-13 22:50:46 - thomas_method.py - INFO - i=2:  $\alpha_2 = 0.000000$ ,  $\beta_2 = 1.000000$ ,  $z_2 = -1.000000$ 
```

Знайшовши коефіцієнти, можемо розпочати зворотній хід:

```
2025-10-13 22:50:46 - thomas_method.py - INFO - === Backward substitution (solving) ===
2025-10-13 22:50:46 - thomas_method.py - INFO -  $y_2 = 1.000000$ 
2025-10-13 22:50:46 - thomas_method.py - INFO -  $y_1 = \alpha_1 * y_2 + \beta_1 = -1.000000$ 
2025-10-13 22:50:46 - thomas_method.py - INFO -  $y_0 = \alpha_0 * y_1 + \beta_0 = 2.000000$ 
2025-10-13 22:50:46 - thomas_method.py - INFO - Solution y2: [ 2.00000, -1.00000, 1.00000]
2025-10-13 22:50:46 - thomas_method.py - INFO - =====
2025-10-13 22:50:46 - main.py - INFO - Thomas algorithm completed successfully
```

Метод Зейделя:

Як і в методі прогонки, починаємо метод Зейделя з перевірки умов збіжності:

```
2025-10-13 22:56:35 - seidel_method.py - INFO - Matrix has strict diagonal dominance: Gauss-Seidel method converges
2025-10-13 22:56:35 - seidel_method.py - INFO - Matrix is symmetric and positive definite: Gauss-Seidel method converges
2025-10-13 22:56:35 - seidel_method.py - INFO - Eigenvalues of the iteration matrix G: [0. 0. 0.04266156 0.39067178]
2025-10-13 22:56:35 - seidel_method.py - INFO - Spectral radius: 0.390672
2025-10-13 22:56:35 - seidel_method.py - INFO - Necessary and sufficient convergence condition satisfied: method converges for any x0
```

Далі обираємо $\varepsilon = 0.001$ (обирається у кодї), опісля чого розпочинаємо ітерації для пошуку коренів СЛАР з заданою точністю:

```
2025-10-13 22:56:35 - seidel_method.py - INFO - === Starting Gauss-Seidel method ===
2025-10-13 22:56:35 - seidel_method.py - INFO - Initial guess: x0 = [0. 0. 0. 0.]
2025-10-13 22:56:35 - seidel_method.py - INFO - Iteration 1:
2025-10-13 22:56:35 - seidel_method.py - INFO - x_0^1 = 1.750000
2025-10-13 22:56:35 - seidel_method.py - INFO - x_1^1 = 4.666667
2025-10-13 22:56:35 - seidel_method.py - INFO - x_2^1 = 3.650000
2025-10-13 22:56:35 - seidel_method.py - INFO - x_3^1 = 2.504167
2025-10-13 22:56:35 - seidel_method.py - INFO - ||x^1 - x^0||_∞ = 4.666667
2025-10-13 22:56:35 - seidel_method.py - INFO - Iteration 2:
2025-10-13 22:56:35 - seidel_method.py - INFO - x_0^2 = 0.837500
2025-10-13 22:56:35 - seidel_method.py - INFO - x_1^2 = 2.997222
2025-10-13 22:56:35 - seidel_method.py - INFO - x_2^2 = 3.331667
2025-10-13 22:56:35 - seidel_method.py - INFO - x_3^2 = 3.418472
2025-10-13 22:56:35 - seidel_method.py - INFO - ||x^2 - x^1||_∞ = 1.669444
2025-10-13 22:56:35 - seidel_method.py - INFO - Iteration 3:
2025-10-13 22:56:35 - seidel_method.py - INFO - x_0^3 = 0.917083
2025-10-13 22:56:35 - seidel_method.py - INFO - x_1^3 = 2.387685
2025-10-13 22:56:35 - seidel_method.py - INFO - x_2^3 = 3.132889
2025-10-13 22:56:35 - seidel_method.py - INFO - x_3^3 = 3.772935
2025-10-13 22:56:35 - seidel_method.py - INFO - ||x^3 - x^2||_∞ = 0.609537
.....
2025-10-13 22:56:35 - seidel_method.py - INFO - Iteration 10:
2025-10-13 22:56:35 - seidel_method.py - INFO - x_0^10 = 0.999882
2025-10-13 22:56:35 - seidel_method.py - INFO - x_1^10 = 2.000538
2025-10-13 22:56:35 - seidel_method.py - INFO - x_2^10 = 3.000185
2025-10-13 22:56:35 - seidel_method.py - INFO - x_3^9 = 3.999193
2025-10-13 22:56:35 - seidel_method.py - INFO - ||x^9 - x^8||_∞ = 0.002148
2025-10-13 22:56:35 - seidel_method.py - INFO - Iteration 10:
2025-10-13 22:56:35 - seidel_method.py - INFO - x_0^10 = 0.999882
2025-10-13 22:56:35 - seidel_method.py - INFO - x_1^10 = 2.000538
2025-10-13 22:56:35 - seidel_method.py - INFO - x_2^10 = 3.000185
2025-10-13 22:56:35 - seidel_method.py - INFO - Iteration 10:
2025-10-13 22:56:35 - seidel_method.py - INFO - x_0^10 = 0.999882
2025-10-13 22:56:35 - seidel_method.py - INFO - x_1^10 = 2.000538
2025-10-13 22:56:35 - seidel_method.py - INFO - x_2^10 = 3.000185
2025-10-13 22:56:35 - seidel_method.py - INFO - x_3^10 = 3.999685
2025-10-13 22:56:35 - seidel_method.py - INFO - ||x^10 - x^9||_∞ = 0.000839
2025-10-13 22:56:35 - seidel_method.py - INFO - x_3^10 = 3.999685
2025-10-13 22:56:35 - seidel_method.py - INFO - ||x^10 - x^9||_∞ = 0.000839
2025-10-13 22:56:35 - seidel_method.py - INFO - Stopping: target accuracy ε = 0.001 reached
2025-10-13 22:56:35 - seidel_method.py - INFO - Solution x = [0.99988152 2.00053815 3.00018514 3.99968464]
2025-10-13 22:56:35 - seidel_method.py - INFO - Approximate solution of the system: [0.99988152 2.00053815 3.00018514 3.99968464]
2025-10-13 22:56:35 - seidel_method.py - INFO - ||x^10 - x^9||_∞ = 0.000839
2025-10-13 22:56:35 - seidel_method.py - INFO - Stopping: target accuracy ε = 0.001 reached
2025-10-13 22:56:35 - seidel_method.py - INFO - Solution x = [0.99988152 2.00053815 3.00018514 3.99968464]
2025-10-13 22:56:35 - seidel_method.py - INFO - Approximate solution of the system: [0.99988152 2.00053815 3.00018514 3.99968464]
2025-10-13 22:56:35 - seidel_method.py - INFO - Stopping: target accuracy ε = 0.001 reached
2025-10-13 22:56:35 - seidel_method.py - INFO - Solution x = [0.99988152 2.00053815 3.00018514 3.99968464]
2025-10-13 22:56:35 - seidel_method.py - INFO - Approximate solution of the system: [0.99988152 2.00053815 3.00018514 3.99968464]
2025-10-13 22:56:35 - main.py - INFO - Gauss-Seidel method completed successfully
```

Отримані результати показують, що реалізований алгоритм методу Зейделя

забезпечив збіжність до розв'язку з заданою точністю ε . Ймовірні корені:

$$x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4$$