МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **ІКНІ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 5
На тему: *"НАБЛИЖЕНІ МЕТОДИ РОЗВ'ЯЗУВАННЯ СИСТЕМ ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ."*
З дисципліни: *"Чисельні методи"*

Лектор:
доцент кафедри ПЗ
Мельник Н.Б.

Виконав:
студент групи ПЗ-18
Юшкевич А.І.

Прийняв:
професор кафедри ПЗ
Гавриш В.І.

Львів – 2023

Тема: наближені методи розв’язування систем лінійних алгебраїчних рівнянь
Мета роботи: ознайомлення на практиці з методами Якобі та Зейделя розв’язування систем лінійних алгебраїчних рівнянь.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Метод простої ітерації (Якобі)

Розглянемо систему лінійних алгебраїчних рівнянь виду:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_n. \end{cases}$$

Припустивши, що коефіцієнти a_{ii} $\neq 0$ $(i = \overline{1,n})$, розв’яжемо i-те рівняння системи відносно x_i . У результаті отримаємо таку систему:

$$\begin{cases} x_1 = \dfrac{b_1}{a_{11}} - \left(\dfrac{a_{12}}{a_{11}}x_2 + \dfrac{a_{13}}{a_{11}}x_3 + \ldots + \dfrac{a_{1n}}{a_{11}}x_n \right), \\ x_2 = \dfrac{b_2}{a_{22}} - \left(\dfrac{a_{21}}{a_{22}}x_1 + \dfrac{a_{23}}{a_{22}}x_3 + \ldots + \dfrac{a_{2n}}{a_{22}}x_n \right), \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n = \dfrac{b_n}{a_{nn}} - \left(\dfrac{a_{n1}}{a_{nn}}x_1 + \dfrac{a_{n2}}{a_{nn}}x_2 + \ldots + \dfrac{a_{n,n-1}}{a_{nn}}x_{n-1} \right). \end{cases}$$

Введемо позначення

$$\beta_i = \frac{b_i}{a_{ii}}, \quad \alpha_{ij} = -\frac{a_{ij}}{a_{ii}}, \quad i = \overline{1,n}, \quad j \neq i$$

і перепишемо систему у вигляді

$$\begin{cases} x_1 = \beta_1 + \alpha_{12}x_2 + \alpha_{13}x_3 + \ldots + \alpha_{1n}x_n, \\ x_2 = \beta_2 + \alpha_{21}x_1 + \alpha_{23}x_3 + \ldots + \alpha_{2n}x_n, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n = \beta_n + \alpha_{n2}x_2 + \alpha_{n3}x_3 + \ldots + \alpha_{nn}x_n. \end{cases}$$

Таку систему називають зведеною. У матричному вигляді запишемо її так:

$$X = \beta + \alpha X ,$$

де

$$\alpha = \begin{pmatrix} 0 & \alpha_{12} & \alpha_{13} & \dots & \alpha_{1n} \\ \alpha_{21} & 0 & \alpha_{23} & \dots & \alpha_{2n} \\ \alpha_{31} & \alpha_{32} & 0 & \dots & \alpha_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ \alpha_{n1} & \alpha_{n2} & \alpha_{n3} & \dots & 0 \end{pmatrix}, \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \cdots \\ \beta_n \end{pmatrix}.$$

За нульове наближення розв’язку системи виберемо стовпець вільних членів, тобто

$$\begin{cases} x_1^{(0)} = \beta_1, \\ x_2^{(0)} = \beta_2, \\ \dots \\ x_n^{(0)} = \beta_n \end{cases}$$

або

$$X^{(0)} = \beta .$$

Перше наближення розв’язку системи знаходимо у вигляді

$$X^{(1)} = \beta + \alpha \cdot X^{(0)}.$$

Аналогічно довільне наближення розв’язку системи визначимо співвідношенням

$$X^{(n)} = \beta + \alpha \cdot X^{(n-1)}.$$

Якщо послідовність $X^{(1)}$, $X^{(2)}$,… $X^{(n)}$,… збігається, тобто $X = \lim\limits_{n \to \infty} X^{(n)}$, то граничний вектор X є розв’язком системи рівнянь, а отже і системи.

Формули перепишемо у розгорнутому вигляді

$$\begin{cases} x_i^{(0)} = \beta_i \\ x_i^{(k)} = \beta_i + \sum\limits_{j=1}^{i-1} \alpha_{ij} x_j^{(k-1)} + \sum\limits_{j=i+1}^{n} \alpha_{ij} x_j^{(k-1)}, \end{cases} \qquad i = \overline{1,n}, \quad k = 1,2,\dots$$

Збіжність ітераційного процесу

Збіжність ітераційного процесу залежить від величини коефіцієнтів матриці 〈. Тому не обов’язково за нульове наближення розв’язку вибирати стовпець вільних членів. За початкове наближення $X^{(0)}$ можна також вибирати

– вектор, усі координати якого $x_i^{(0)} = 0$ $(i = 1,n)$;

– вектор, усі координати якого $x_i^{(0)} = 1$ $(i = 1,n)$;

– вектор $X^{(0)}$, отриманий у результаті аналізу особливостей об’єкту дослідження та задачі, яку розв’язують.

Теорема (про збіжність ітераційного процесу). Якщо елементи матриці 〈 системи рівнянь задовольняють одну з умов:

$$\sum_{j=1}^{n}\left|\frac{a_{ij}}{a_{ii}}\right|<1, \quad i=\overline{1,n},$$

$$\sum_{i=1}^{n}\left|\frac{a_{ij}}{a_{ii}}\right|<1, \quad j=\overline{1,n},$$

$$\sum_{i,j=1}^{n}\left(\frac{a_{ij}}{a_{ii}}\right)^2<1,$$

то система рівнянь має єдиний розв'язок X^*, який не залежить від початкового наближення $X^{(0)}$.

Критерії припинення ітераційного процесу

Якщо задана похибка ∑ наближеного розв’язку, то критерієм припинення ітераційного процесу вважають виконання однієї з умов:

- модуль різниці між наступним та попереднім наближенням розв’язку повинен бути меншим за ∑
$$\left|X^{(k)}-X^{(k-1)}\right|=\sqrt{\sum_{i=1}^{n}\left(x_i^{(k)}-x_i^{(k-1)}\right)^2}<\varepsilon, \quad k=1,2,\ldots;$$
- максимальне значення модуля різниць між відповідними компонентами наступного та попереднього наближення розв’язку повинно бути меншим за ∑
$$\max_i\left|x_i^{(k)}-x_i^{(k-1)}\right|<\varepsilon, \quad i=\overline{1,n}, \quad k=1,2,\ldots;$$
- максимальне значення модуля відносних різниць між відповідними компонентами наступного та попереднього наближення розв’язку повинно бути меншим за ∑
$$\max_i\left|\frac{x_i^{(k)}-x_i^{(k-1)}}{x_i^{(k)}}\right|<\varepsilon, \quad |x_i|>>1, \quad i=\overline{1,n}, \quad k=1,2,\ldots .$$

Для попередження необґрунтованих витрат машинного часу для випадку, коли ітераційний процес не є збіжним для конкретної СЛАР, використовують лічильник кількості ітерацій, і при досягненні нею деякого заданого числа N обчислення припиняють.

Метод Зейделя

Даний метод є модифікацією методу простої ітерації. Основна його ідея полягає в тому, що при обчисленні чергового k-го наближення розв’язку $x_i^{(k)}$ (i =1,n)використовують вже знайдені значення k-го наближеного розв’язку $x_1^{(k)}$, $x_2^{(k)}$, …, $x_{i-1}^{(k)}$. Ітераційні формули методу Зейделя мають вигляд

$$x_i^{(k)}=\beta_i+\sum_{j=1}^{i-1}\alpha_{ij}x_j^{(k)}+\sum_{j=i+1}^{n}\alpha_{ij}x_j^{(k-1)}, \quad i=\overline{1,n}, \quad k=1,2,\ldots$$

Умова збіжності і критерій припинення ітераційного процесу за методом Зейделя є такими ж, як і для методу простої ітерації.

За методом Зейделя отримують кращу збіжність, ніж за методом Якобі, але доводиться проводити більш громіздкі обчислення. Крім того, ітераційний процес, проведений за методом Зейделя іноді буває збіжним у тому випадку, коли він є розбіжним за методом простої ітерації і навпаки.

Індивідуальне завдання

1. Ознайомитись з теоретичними відомостями.
2. Розв'язати систему нелінійних рівнянь з точністю eps = 10^{-3} методом ітерацій та методом Ньютона.

Варіант 22

22. $$A = \begin{pmatrix} 24{,}67 & 3{,}24 & 5{,}45 & 4{,}13 \\ 4{,}46 & 34{,}86 & 3{,}12 & -2{,}43 \\ 3{,}87 & 6{,}54 & 45{,}44 & 3{,}45 \\ 2{,}45 & 4{,}25 & 5{,}45 & 32{,}72 \end{pmatrix}, \quad B = \begin{pmatrix} 80{,}41 \\ 85{,}44 \\ 187{,}84 \\ 152{,}86 \end{pmatrix}.$$

Результат виконання програми

```
Jacobi

x1            x2            x3            x4
1.24221       1.98961       3.14875       3.4208

1.72984       2.24866       3.48193       3.79584

1.55942       2.18259       3.37464       3.67019

1.61284       2.20524       3.4082        3.7094

1.59589       2.19814       3.39742       3.69687

1.6013        2.2004        3.40083       3.70086

1.59958       2.19968       3.39975       3.69959

1.60013       2.1999        3.40009       3.69999

1.59995       2.19983       3.39998       3.69987

1.60001       2.19985       3.40002       3.69991

Final result:
x1: 1.60001
x2: 2.19985
x3: 3.40002
x4: 3.69991
```

Рис. 1. Зображення ітераційного процесу методу Якобі

```
Seidel

x1            x2            x3            x4
1.24221       2.2477        3.34981       3.72883

1.59996       2.20636       3.39688       3.69957

1.59988       2.20012       3.4           3.69987

1.59996       2.19985       3.40001       3.6999

1.59999       2.19985       3.40001       3.6999

Final result:
x1: 1.59999
x2: 2.19985
x3: 3.40001
x4: 3.6999
```

Рис. 2. Зображення ітераційного процесу методу Зейделя

Висновки

У результаті виконання лабораторної роботи визначено дійсні корені заданої СЛАР з заданою точністю 10^{-3} методом Якобі та методом Зейделя. Розв'язок отриманий із заданою точністю, методом Якобі за 10 кроків ітерації, а методом Зейделя за 5.

Код програми

SystemSolver_2_Header.h:

```
#pragma once

#include <iostream>

#include <vector>
#include <string>
#include <cmath>

using namespace std;

typedef long double ldouble;


class SystemSolver_2 {
public:

	SystemSolver_2() = delete;

	template <class T>
	SystemSolver_2(const T matrix, const size_t matrix_size, const
vector<ldouble> free_terms);

	template <class T>
	SystemSolver_2(const T matrix, const size_t matrix_size, const
vector<ldouble> free_terms, const ldouble epsilon);

	vector<ldouble> Jacobi();
	vector<ldouble> Seidel();

private:
	vector<vector<ldouble>> m_matrix;
	vector<ldouble> m_free_terms;
	size_t m_matrix_size;
	bool m_is_roots_found;
	ldouble m_epsilon;

	template <typename T>
	vector<vector<ldouble>> CopyMatrix(const T matrix, const size_t size);

	bool IsMatrixConvergent() const;

	void DivideByMaxElement();

	void SetHollowMatrix(vector<vector<ldouble>>& hollow_matrix,
vector<ldouble>& new_free_terms) const;

	void CheckIfRootsFound(const vector<ldouble> roots, const vector<ldouble>
previous_roots);

	vector<ldouble> FindCloserJacobi(const vector<vector<ldouble>>
hollow_matrix, const vector<ldouble> new_free_terms, vector<ldouble>
previous_roots);
```

```
	vector<ldouble> FindCloserSeidel(const vector<vector<ldouble>>
hollow_matrix, const vector<ldouble> new_free_terms, vector<ldouble>
previous_roots);

};

template <class T>
SystemSolver_2::SystemSolver_2(const T matrix, const size_t matrix_size, const
vector<ldouble> free_terms) {
	m_matrix = CopyMatrix(matrix, matrix_size);
	m_free_terms = free_terms;
	m_matrix_size = matrix_size;
	m_is_roots_found = false;
	m_epsilon = 0.001;
}
template <class T>
SystemSolver_2::SystemSolver_2(const T matrix, const size_t matrix_size, const
vector<ldouble> free_terms, const ldouble epsilon) {
	m_matrix = CopyMatrix(matrix, matrix_size);
	m_free_terms = free_terms;
	m_matrix_size = matrix_size;
	m_is_roots_found = false;
	m_epsilon = epsilon;
}

template <typename T>
vector<vector<ldouble>> SystemSolver_2::CopyMatrix(const T matrix, const size_t
size) {
	vector<vector<ldouble>> new_vector(size, vector<ldouble>(size));

	do {
		if (new_vector.empty())
			break;

		for (int i = 0; i < size; i++) {
			for (int j = 0; j < size; j++) {
				new_vector[i][j] = matrix[i][j];
			}
		}

	} while (false);

	return new_vector;
}
```

Functions.cpp:

```
#include "SystemSolver_2_Header.h"

vector<ldouble> SystemSolver_2::Jacobi() {

	if (!IsMatrixConvergent())
		DivideByMaxElement();

	vector<vector<ldouble>> hollow_matrix(m_matrix_size,
vector<ldouble>(m_matrix_size));
	vector<ldouble> new_free_terms(m_matrix_size);

	SetHollowMatrix(hollow_matrix, new_free_terms);

	vector<ldouble> roots(new_free_terms);
	do {
		roots = FindCloserJacobi(hollow_matrix, new_free_terms, roots);
	} while (!m_is_roots_found);

	m_is_roots_found = false;
	return roots;
}
```

```cpp
vector<ldouble> SystemSolver_2::Seidel() {
	if (!IsMatrixConvergent())
		DivideByMaxElement();

	vector<vector<ldouble>> hollow_matrix(m_matrix_size,
vector<ldouble>(m_matrix_size));
	vector<ldouble> new_free_terms(m_matrix_size);

	SetHollowMatrix(hollow_matrix, new_free_terms);


	vector<ldouble> roots(new_free_terms);
	do {
		roots = FindCloserSeidel(hollow_matrix, new_free_terms, roots);
	} while (!m_is_roots_found);

	m_is_roots_found = false;
	return roots;
}

bool SystemSolver_2::IsMatrixConvergent() const {

	//Sum of elements of each row less than 1:
	ldouble sum_of_row{ 0 };
	bool is_all_rows_less_than_1{true};
	for (int i = 0; i < m_matrix_size; i++) {
		for (int j = 0; j < m_matrix_size; j++) {
			sum_of_row += m_matrix[i][j];
		}

		if (sum_of_row > 1) {
			is_all_rows_less_than_1 = false;
			break;
		}

		sum_of_row = 0;
	}

	if (is_all_rows_less_than_1)
		return true;

	//Sum of elements of each column less than 1:
	ldouble sum_of_column{ 0 };
	bool is_all_columns_less_than_1{true};
	for (int i = 0; i < m_matrix_size; i++) {
		for (int j = 0; j < m_matrix_size; j++) {
			sum_of_column += m_matrix[j][i];
		}

		if (sum_of_column > 1) {
			is_all_columns_less_than_1 = false;
			break;
		}

		sum_of_column = 0;
	}

	if (is_all_columns_less_than_1)
		return true;

	//Sum of elements of each row less than diagonal element:
	ldouble sum_of_incompete_row{ 0 };
	bool is_all_rows_less_than_diagonal{ true };
	for (int i = 0; i < m_matrix_size; i++) {
		for (int j = 0; j < m_matrix_size; j++) {
			if (i != j)
				sum_of_incompete_row += m_matrix[i][j];
		}

		if (sum_of_incompete_row > m_matrix[i][i]) {
			is_all_rows_less_than_diagonal = false;
```

```
				break;
			}

			sum_of_incompete_row = 0;
		}

		if (is_all_rows_less_than_diagonal)
			return true;


		//Sum of elements of each column less than diagonal element:
		ldouble sum_of_incompete_column{ 0 };
		bool is_all_columns_less_than_diagonal{ true };
		for (int i = 0; i < m_matrix_size; i++) {
			for (int j = 0; j < m_matrix_size; j++) {
				if (i != j)
					sum_of_incompete_column += m_matrix[i][j];
			}

			if (sum_of_incompete_column > m_matrix[i][i]) {
				is_all_columns_less_than_diagonal = false;
				break;
			}

			sum_of_incompete_column = 0;
		}

		if (is_all_columns_less_than_diagonal)
			return true;


		return false;
	}

	void SystemSolver_2::DivideByMaxElement() {
		ldouble max_element{ 0 };
		for (int i = 0; i < m_matrix_size; i++) {
			for (int j = 0; j < m_matrix_size; j++) {
				if (m_matrix[i][j] > max_element)
					max_element = m_matrix[i][j];
			}
		}

		max_element *= m_matrix_size + 1;

		for (int i = 0; i < m_matrix_size; i++) {
			for (int j = 0; j < m_matrix_size; j++) {
				m_matrix[i][j] /= max_element;
			}
			m_free_terms[i] /= max_element;
		}

	}

	void SystemSolver_2::SetHollowMatrix(vector<vector<ldouble>>& hollow_matrix,
	vector<ldouble>& new_free_terms) const {

		for (int i = 0; i < m_matrix_size; i++) {
			new_free_terms[i] = m_free_terms[i] / m_matrix[i][i];
			hollow_matrix[i][i] = 0;

			for (int j = 0; j < m_matrix_size; j++) {
				if (i == j)
					continue;

				hollow_matrix[i][j] = -(m_matrix[i][j] / m_matrix[i][i]);
			}
		}
	}
```

```cpp
vector<ldouble> SystemSolver_2::FindCloserJacobi(
	const vector<vector<ldouble>> hollow_matrix,
	const vector<ldouble> new_free_terms,
	vector<ldouble> previous_roots) {

	vector<ldouble> roots(m_matrix_size);

	for (int i = 0; i < m_matrix_size; i++) {
		roots[i] = 0;
		for (int j = 0; j < m_matrix_size; j++) {
			roots[i] += hollow_matrix[i][j] * previous_roots[j];
		}
		roots[i] += new_free_terms[i];
	}

	for (int i = 0; i < roots.size(); i++) {
		cout << roots[i] << "\t\t";
	}
	cout << endl << endl;

	CheckIfRootsFound(roots, previous_roots);

	return roots;
}

vector<ldouble> SystemSolver_2::FindCloserSeidel(
	const vector<vector<ldouble>> hollow_matrix,
	const vector<ldouble> new_free_terms,
	vector<ldouble> previous_roots) {

	vector<ldouble> roots(m_matrix_size);
	vector<ldouble> previous_roots_copy(previous_roots);

	for (int i = 0; i < m_matrix_size; i++) {
		roots[i] = 0;
		for (int j = 0; j < m_matrix_size; j++) {
			roots[i] += hollow_matrix[i][j] * previous_roots_copy[j];
		}
		roots[i] += new_free_terms[i];
		previous_roots_copy[i] = roots[i];
	}

	for (int i = 0; i < roots.size(); i++) {
		cout << roots[i] << "\t\t";
	}
	cout << endl << endl;

	CheckIfRootsFound(roots, previous_roots);

	return roots;
}
void SystemSolver_2::CheckIfRootsFound(const vector<ldouble> roots, const
vector<ldouble> previous_roots) {
	// first condition
	ldouble random_bullshit{ 0 };
	for (int i = 0; i < m_matrix_size; i++) {
		random_bullshit += pow(roots[i] - previous_roots[i], 2);
	}
	random_bullshit = pow(random_bullshit, 0.5);

	if (random_bullshit < m_epsilon)
		m_is_roots_found = true;


	// second condition
	random_bullshit = 0;
	for (int i = 0; i < m_matrix_size; i++) {
		if (fabs(roots[i] - previous_roots[i]) > random_bullshit)
			random_bullshit = fabs(roots[i] - previous_roots[i]);
	}
```

```
        if (random_bullshit < m_epsilon)
            m_is_roots_found = true;

}
```

Lab_05.cpp:

```
#include <iostream>
#include "SystemSolver_2_Header.h"

int main()
{

    const size_t size{ 4 };
    ldouble matrix[size][size] = { { 24.67, 3.24,  5.45, 4.13},
                                   { 4.46,  34.86,  3.12, -2.43},
                                   { 3.87,  6.54,  45.44, 3.45},
                                   { 2.45,  4.25,  5.45, 32.72} };

    vector<ldouble> B{ 80.41, 85.44, 187.84, 152.86 };

    const ldouble epsilon = 0.0001;

    SystemSolver_2 solver(matrix, size, B, epsilon);

    cout << "Jacobi" << endl << endl;
    cout << "x1\t\tx2\t\tx3\t\tx4\n";
    vector<ldouble> result_jacobi = solver.Jacobi();
    cout << "Final result:\n";
    for (int i = 0; i < result_jacobi.size(); i++) {
        cout << "x" << i + 1 << ": " << result_jacobi[i] << endl;
    }
    cout << endl << endl;

    cout << "Seidel" << endl << endl;
    cout << "x1\t\tx2\t\tx3\t\tx4\n";
    vector<ldouble> result_seidel = solver.Seidel();
    cout << "Final result:\n";
    for (int i = 0; i < result_seidel.size(); i++) {
        cout << "x" << i + 1 << ": " << result_seidel[i] << endl;
    }
    cout << endl << endl;
}
```