Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт
До лабораторної роботи №4

На теми: “Розв’язування систем лінійних алгебраїчних рівнянь методом Гауса та методом LU-розкладу”
З дисципліни: “Чисельні методи”

Лектор:
доц. каф. ПЗ
Мельник Н.Б.
Виконав:
ст. гр. ПЗ-18
Юшкевич А.І.
Прийняв:
проф. каф. ПЗ
Гавриш В.І.
« … » … 2023 р.

∑ = ____________

Львів – 2023

Теми: розв’язування систем лінійних алгебраїчних рівнянь методом Гауса та методом LU-розкладу.
Мета: ознайомлення на практиці з методом Гауса та методом LU- розкладу розв’язування систем лінійних алгебраїчних рівнянь. Код програмної реалізації подано у додатку.

Завдання

Скласти програму розв’язування системи лінійних алгебраїчних рівнянь методами Гауса з вибором головного елемента та LU -розкладу.

22.
$$A = \begin{pmatrix} 24,67 & 3,24 & 5,45 & 4,13 \\ 4,46 & 34,86 & 3,12 & -2,43 \\ 3,87 & 6,54 & 45,44 & 3,45 \\ 2,45 & 4,25 & 5,45 & 32,72 \end{pmatrix}, \quad B = \begin{pmatrix} 80,41 \\ 85,44 \\ 187,64 \\ 152,86 \end{pmatrix}.$$

Рис. 1. Система лінійних алгебраїчних рівнянь

Метод Гаусса

Найвідомішим точним методом розв’язування систем лінійних алгебраїчних рівнянь є метод Гауса, суть якого полягає в тому, що систему рівнянь, яку необхідно розв’язати, зводять до еквівалентної системи з верхньою (або нижньою) трикутною матрицею. Невідомі знаходять послідовними підстановками, починаючи з останнього рівняння перетвореної системи. Точність результату та витрачений на його отримання час у більшості випадків залежить від алгоритму формування трикутної матриці системи. У загальному випадку алгоритм методу Гауса складається з двох етапів – прямого та зворотного ходу. Під час **прямого ходу** СЛАР перетворюють до еквівалентної системи з верхньою трикутною матрицею. **Зворотній хід** дає змогу визначити елементи вектору невідомих, починаючи з останнього рівняння системи, підставляючи послідовно відповідні елементи цього вектору, отримані на попередньому кроці.

Метод Гауса з вибором головного елемента

Серед елементів матриці A виберемо найбільший за модулем елемент, який називають головним елементом. Далі перетворюємо матрицю A так: від кожного i -го неголовного рядка віднімаємо почленно головний рядок, помножений на mi . У результаті отримуємо матрицю, у якій всі елементи k -го стовпця, за винятком pk a , дорівнюють нулеві. Відкидаючи цей стовпець і головний рядок, отримуємо нову матрицю A1 з меншою на одиницю кількістю рядків та стовпців. Такі самі дії повторюємо над матрицею A1 і отримуємо матрицю A2 і т.д. Ці перетворення продовжуємо доти, поки не отримаємо матрицю, що містить один рядок з двох елементів, який вважаємо головним. Об'єднаємо всі головні рядки, починаючи від останнього. Після деяких перестановок вони утворять трикутну матрицю, еквівалентну до початкової матриці A.

Метод LU-розкладу

Розв'язуючи систему лінійних алгебраїчних рівнянь даним методом, матрицю A коефіцієнтів системи розкладають на добуток двох матриць – нижньої трикутної матриці L , елементи головної діагоналі якої не дорівнюють нулеві та верхньої трикутної U , на головній діагоналі якої містяться одиниці. Розв'язування матричного рівняння виконуємо за два етапи: спочатку розв'язуємо матричне рівняння, а потім. Такий підхід суттєво спрощує отримання розв'язку порівняно з методом Гауса для випадку, коли маємо кілька систем рівнянь з однаковою матрицею коефіцієнтів A, оскільки матриці L та U визначають один раз. Розв'язування систем LY = B та UX = Y називають прямим та оберненим ходом відповідно. Спочатку розглянемо прямий хід методу. Завдяки трикутній формі матриці L вектор Y легко визначають. Для цього матричне рівняння перепишемо у розгорнутому вигляді. При виконанні оберненого ходу компоненти вектору X визначають зі системи рівнянь.

Основні етапи обчислювального алгоритму для розв'язування системи лінійних алгебраїчних рівнянь методом Гауса, реалізованого у програмному продукті мовою C++

1) внесення даних в конструкторі SystemSolver() (рис. 2);
2) виклик метода Gauss() (рис. 3), що реалізує знаходження коренів системи лінійних рівнянь методом Гауса;
3) знаходження визначника заданої матриці за допомогою методу FindDeterminant() (рис. 4)
4) виклик метода GaussItself() (рис. 5), який за допомогою елементарних перетворень утворює верхню трикутну матрицю;
5) вивід результату виконання в консоль (рис. 6).
6) перевірка точності отриманого розв'язку системи лінійних рівнянь (рис. 9).

```
template <typename T>
SystemSolver::SystemSolver(T matrix, vector<ldouble> B) {
    this->matrix = CopyMatrix(matrix, GetSize(matrix));
    this->B = B;
}
```

Рис. 2. Конструктор SystemSolver()

```
vector<ldouble> SystemSolver::Gauss() {
    vector <ldouble> result(matrix[0].size());
    vector<vector<ldouble>> inside_matrix = CopyMatrix(this->matrix, matrix[0].size());
    vector<ldouble> inside_B = this->B;

    if (FindDeterminant(this->matrix) == 0) {
        cout << "Determinant is equal zero";
        return result;
    }

    GaussItself(inside_matrix, inside_B);

    for (int i = inside_matrix[0].size() - 1; i >= 0; i--) {
        result[i] = inside_B[i];
        for (int j = inside_matrix[0].size() - 1; j > i; j--) {
            result[i] -= result[j] * inside_matrix[i][j];
        }
        result[i] /= inside_matrix[i][i];
    }

    return result;

}
```

Рис. 3. Метод Gauss()

```
ldouble SystemSolver::FindDeterminant(const vector<vector<ldouble>> matrix) const {

    int index = 0;
    size_t matrix_size = matrix.size();

    if (matrix.size() == 1)
        return matrix[0][0];

    vector<vector<ldouble>> smaller_matrix = CreateMatrix(matrix_size - 1);

    ldouble determinant = 0;
    int column = 0;
    bool wrong_k_found = false;

    for (int i = 0; i < matrix_size; i++)
    {
        for (int j = 1; j < matrix_size; j++) {
            for (int k = 0; k < matrix_size; k++) {
                if (k == index) {
                    wrong_k_found = true;
                    continue;
                }

                if (wrong_k_found)
                    column = k - 1;
                else
                    column = k;

                smaller_matrix[j - 1][column] = matrix[j][k];
            }
            wrong_k_found = false;
        }

        determinant += pow(-1, i) * matrix[0][i] * FindDeterminant(smaller_matrix);
        index++;
    }

    return determinant;
}
```

Рис. 4. Метод FindDeterminant()

```
void SystemSolver::GaussItself(vector<vector<ldouble>>& matrix, vector<ldouble>& B) {
    int index_of_row_with_max_element{ 0 };
    ldouble max_element{ 0 };
    size_t size_of_matrix = matrix.size();
    while (size_of_matrix > 1) {
        size_t current_column = matrix[0].size() - size_of_matrix;
        for (int i = current_column; i < matrix[0].size(); i++) {
            if (fabs(matrix[i][current_column]) > max_element) {
                index_of_row_with_max_element = i;
                max_element = matrix[i][current_column];
            }
        }

        if (fabs(max_element) > 1e-13) {
            if (index_of_row_with_max_element != current_column) {
                vector<ldouble> temp_row(size_of_matrix);
                ldouble temp_B = B[index_of_row_with_max_element];
                B[index_of_row_with_max_element] = B[current_column];
                B[current_column] = temp_B;

                for (int i = current_column; i < size_of_matrix; i++) {
                    temp_row[i] = matrix[index_of_row_with_max_element][i];
                    matrix[index_of_row_with_max_element][i] = matrix[current_column][i];
                    matrix[current_column][i] = temp_row[i];
                }
            }

            for (int i = current_column + 1; i < matrix.size(); i++) {
                ldouble multiplier = matrix[i][current_column] / matrix[current_column][current_column];

                matrix[i][current_column] = 0;
                for (int j = current_column + 1; j < matrix.size(); j++) {
                    matrix[i][j] -= matrix[current_column][j] * multiplier;
                }
                B[i] -= B[current_column] * multiplier;
            }
        }

        index_of_row_with_max_element = 0;
        max_element = 0;
        size_of_matrix--;
    }
}
```

Рис. 5. Метод GaussItself()

```
Roots by Gauss are:

Root №1:    1.59999
Root №2:    2.19985
Root №3:    3.40001
Root №4:    3.6999
```

Рис. 6. Результат виконання програмної реалізації методу Гауса

Основні етапи обчислювального алгоритму для розв’язування системи лінійних алгебраїчних рівнянь методом LU-розкладу, реалізованого у програмному продукті мовою C++

1) внесення даних в конструкторі SystemSolver() (рис. 2);
2) виклик метода LU() (рис. 7), що реалізує знаходження коренів системи лінійних рівнянь методом LU-розкладу;
3) вивід результату виконання в консоль (рис. 8).
4) перевірка точності отриманого розв’язку системи лінійних рівнянь (рис. 9).

```
vector<ldouble> SystemSolver::LU() {
    size_t size = matrix[0].size();

    vector<vector<ldouble>> l(size, vector<ldouble>(size));
    vector<vector<ldouble>> u(size, vector<ldouble>(size));


    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            l[i][j] = 0;
            u[i][j] = (i == j) ? 1 : 0;
        }
    }

    for (int i = 0; i < size; i++)
        l[i][0] = matrix[i][0];

    for (int index = 1, switcher = 0; index < size; index++, switcher++) {
        if (switcher % 2) {
            for (int i = index; i < size; i++) {
                for (int k = 0; k < index; k++)
                    l[i][index] += l[i][k] * u[k][index];
                l[i][index] = matrix[i][index] - l[i][index];
            }
        }
        else {
            for (int i = index - 1, j = index; j < size; j++) {
                for (int k = 0; k < index - 1; k++) {
                    u[i][j] += l[i][k] * u[k][j];
                }
                u[i][j] = (matrix[i][j] - u[i][j]) / l[i][i];
            }
            index--;
        }
    }

    vector<ldouble> y(size);
    for (int i = 0; i < size; i++)
        y[i] = 0;

    for (int i = 0; i < size; i++) {
        for (int k = 0; k < i; k++)
            y[i] += y[k] * l[i][k];

        y[i] = (B[i] - y[i]) / l[i][i];
    }

    vector<ldouble> result(size);
    for (int i = 0; i < size; i++)
        result[i] = 0;

    for (int i = size - 1; i >=0; i--) {
        for (int k = size - 1; k > i; k--)
            result[i] += result[k] * u[i][k];

        result[i] = (y[i] - result[i]) / u[i][i];
    }

    return result;
}
```

Рис. 7. Метод LU ()

```
Roots by LU are:

Root №1:    1.59999
Root №2:    2.19985
Root №3:    3.40001
Root №4:    3.6999
```

Рис. 8. Результат виконання програмної реалізації методу LU-розкладу

```
Accuracy of the solution of the system:

method result    free term
 80.41            80.41
 85.44            85.44
 187.84           187.84
 152.86           152.86
```

Рис. 9. Перевірка точності отриманого розв’язку системи лінійних рівнянь

Висновки

У результаті виконання лабораторної роботи розробив програму розв’язування системи лінійних алгебраїчних рівнянь методами Гауса з вибором головного елемента та LU -розкладу для заданої системи лінійних алгебраїчних рівнянь.

Додаток

Header.h:

```
#pragma once
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

typedef long double ldouble;

class SystemSolver {
private:
	vector<ldouble> B;
	vector<vector<ldouble>> matrix;

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

```
		size_t GetSize(const T matrix) const;
		vector<vector<ldouble>> CreateMatrix(const size_t size) const;
		ldouble FindDeterminant(const vector<vector<ldouble>> matrix) const;
		void GaussItself(vector<vector<ldouble>>& matrix, vector<ldouble>& B);

public:

		template <typename T>
		SystemSolver(T matrix, vector<ldouble> B);

		vector<ldouble> Gauss();
		vector<ldouble> LU();

};

template <typename T>
size_t SystemSolver::GetSize(const T matrix) const {
		size_t result{ 0 };
		if(matrix != nullptr)
				result = sizeof(matrix[0]) / sizeof(matrix[0][0]);

		return result;
}

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

		do {
				new_vector = CreateMatrix(size);

				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;
}

template <typename T>
SystemSolver::SystemSolver(T matrix, vector<ldouble> B) {
		this->matrix = CopyMatrix(matrix, GetSize(matrix));
		this->B = B;
}
```

Functions.cpp:

```
#include "Header.h"

vector<vector<ldouble>> SystemSolver::CreateMatrix(const size_t size) const{
	vector<vector<ldouble>> new_matrix(size, vector<ldouble>(size));

	return new_matrix;
}
ldouble SystemSolver::FindDeterminant(const vector<vector<ldouble>> matrix) const {

	int index = 0;
	size_t matrix_size = matrix.size();

	if (matrix.size() == 1)
		return matrix[0][0];

	vector<vector<ldouble>> smaller_matrix = CreateMatrix(matrix_size - 1);

	ldouble determinant = 0;
	int column = 0;
	bool wrong_k_found = false;

	for (int i = 0; i < matrix_size; i++)
	{
		for (int j = 1; j < matrix_size; j++) {
			for (int k = 0; k < matrix_size; k++) {
				if (k == index) {
					wrong_k_found = true;
					continue;
				}

				if (wrong_k_found)
					column = k - 1;
				else
					column = k;

				smaller_matrix[j - 1][column] = matrix[j][k];
			}
			wrong_k_found = false;
		}

		determinant += pow(-1, i) * matrix[0][i] * FindDeterminant(smaller_matrix);
		index++;
	}

	return determinant;
}

vector<ldouble> SystemSolver::Gauss() {
	vector <ldouble> result(matrix[0].size());
	vector<vector<ldouble>> inside_matrix = CopyMatrix(this->matrix, matrix[0].size());
	vector<ldouble> inside_B = this->B;

	if (FindDeterminant(this->matrix) == 0) {
		cout << "Determinant is equal zero";
		return result;
	}

	GaussItself(inside_matrix, inside_B);
```

```
	for (int i = inside_matrix[0].size() - 1; i >= 0; i--) {
		result[i] = inside_B[i];
		for (int j = inside_matrix[0].size() - 1; j > i; j--) {
			result[i] -= result[j] * inside_matrix[i][j];
		}
		result[i] /= inside_matrix[i][i];
	}

	return result;

}
void SystemSolver::GaussItself(vector<vector<ldouble>>& matrix, vector<ldouble>& B) {
	int index_of_row_with_max_element{ 0 };
	ldouble max_element{ 0 };
	size_t size_of_matrix = matrix.size();
	while (size_of_matrix > 1) {
		size_t current_column = matrix[0].size() - size_of_matrix;
		for (int i = current_column; i < matrix[0].size(); i++) {
			if (fabs(matrix[i][current_column]) > max_element) {
				index_of_row_with_max_element = i;
				max_element = matrix[i][current_column];
			}
		}

		if (fabs(max_element) > 1e-13) {
			if (index_of_row_with_max_element != current_column) {
				vector<ldouble> temp_row(size_of_matrix);
				ldouble temp_B = B[index_of_row_with_max_element];
				B[index_of_row_with_max_element] = B[current_column];
				B[current_column] = temp_B;

				for (int i = current_column; i < size_of_matrix; i++) {
					temp_row[i] = matrix[index_of_row_with_max_element][i];
					matrix[index_of_row_with_max_element][i] =
matrix[current_column][i];
					matrix[current_column][i] = temp_row[i];
				}
			}

			for (int i = current_column + 1; i < matrix.size(); i++) {
				ldouble multiplier = matrix[i][current_column] /
matrix[current_column][current_column];

				matrix[i][current_column] = 0;
				for (int j = current_column + 1; j < matrix.size(); j++) {
					matrix[i][j] -= matrix[current_column][j] * multiplier;
				}
				B[i] -= B[current_column] * multiplier;
			}
		}

		index_of_row_with_max_element = 0;
		max_element = 0;
		size_of_matrix--;
	}
}
vector<ldouble> SystemSolver::LU() {
	size_t size = matrix[0].size();

	vector<vector<ldouble>> l(size, vector<ldouble>(size));
	vector<vector<ldouble>> u(size, vector<ldouble>(size));
```

```
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			l[i][j] = 0;
			u[i][j] = (i == j) ? 1 : 0;
		}
	}

	for (int i = 0; i < size; i++)
		l[i][0] = matrix[i][0];

	for (int index = 1, switcher = 0; index < size; index++, switcher++) {
		if (switcher % 2) {
			for (int i = index; i < size; i++) {
				for (int k = 0; k < index; k++)
					l[i][index] += l[i][k] * u[k][index];
				l[i][index] = matrix[i][index] - l[i][index];
			}
		}
		else {
			for (int i = index - 1, j = index; j < size; j++) {
				for (int k = 0; k < index - 1; k++) {
					u[i][j] += l[i][k] * u[k][j];
				}
				u[i][j] = (matrix[i][j] - u[i][j]) / l[i][i];
			}
			index--;
		}
	}

	vector<ldouble> y(size);
	for (int i = 0; i < size; i++)
		y[i] = 0;

	for (int i = 0; i < size; i++) {
		for (int k = 0; k < i; k++)
			y[i] += y[k] * l[i][k];

		y[i] = (B[i] - y[i]) / l[i][i];
	}

	vector<ldouble> result(size);
	for (int i = 0; i < size; i++)
		result[i] = 0;

	for (int i = size - 1; i >=0; i--) {
		for (int k = size - 1; k > i; k--)
			result[i] += result[k] * u[i][k];

		result[i] = (y[i] - result[i]) / u[i][i];
	}

	return result;
}
```

Lab_04_NM.cpp:

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

using namespace std;

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 };

	SystemSolver ss(matrix, B);

	vector<ldouble> result = ss.Gauss();

	cout << "Roots by Gauss are: " << endl << endl;
	for (int i = 0; i < result.size(); i++) {
		cout << "Root №" << i + 1 << ":   " << result[i] << endl;
	}
	cout << endl << endl << endl;

	result = ss.LU();

	cout << "Roots by LU are: " << endl << endl;
	for (int i = 0; i < result.size(); i++) {
		cout << "Root №" << i + 1 << ":   " << result[i] << endl;
	}
	cout << endl << endl << endl;

	ldouble sum{ 0 };
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			sum += matrix[i][j] * result[j];
		}

		cout << sum << "\t" << B[i] << endl;
		sum = 0;
	}



	return 0;
}
```