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

Звіт
Про виконання лабораторної роботи № 6
На тему:
«Розв’язування перевизначених систем лінійних алгебраїчних рівнянь»

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

∑ = ____________

Львів – 2023

Тема: розв’язування перевизначених систем лінійних алгебраїчних рівнянь.

Мета: ознайомлення на практиці з методами розв’язування перевизначених систем лінійних алгебраїчних рівнянь.

Завдання

Варіант 22

Розв’язати перевизначену систему лінійних алгебраїчних рівнянь методом найменших квадратів. Отриману відповідну нормальну систему розв’язати методом квадратного кореня.

Код програмної реалізації подано у додатку.

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

Метод найменших квадратів для розв'язування перевизначених СЛАР

Розглянемо систему лінійних алгебраїчних рівнянь, у якій кількість рівнянь є більшою за кількість невідомих

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1m}x_m = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2m}x_m = b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nm}x_m = b_n, \end{cases}$$

(рис. 1)

де $n > m$.

Знайдемо розв’язок системи x_1, x_2, ..., x_m наближено, але щоб він задовольняв усі рівняння системи (рис. 1), а саме

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1m}x_m - b_1 = \varepsilon_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2m}x_m - b_2 = \varepsilon_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nm}x_m - b_n = \varepsilon_n. \end{cases}$$

(рис. 2)

Розв’язок системи (рис. 2) будемо знаходити з використанням умови мінімізації суми квадратів відхилень, тобто з умов мінімізації функції

$$S(x_1, x_2, ..., x_m) = \sum_{j=1}^{m} \varepsilon_i^2$$

(рис. 3)

і вимагатимемо, щоб виконувалась умова

$$\sum_{i=1}^{m} \varepsilon_i^2 \to \min .$$

(рис. 4)

Проведемо деякі перетворення над системою (рис. 2), використовуючи умову (рис. 4). Розглянемо функцію

$$S(x_1, x_2, \ldots, x_m) = \sum_{i=1}^{n}\left(\sum_{j=1}^{m} a_{ij}x_j - b_j\right)^2 .$$
(рис. 5)

Необхідною умовою мінімуму функції від багатьох змінних є рівність нулеві її частинних похідних. Використаємо цей факт і продиференціюємо функцію (рис. 5) за змінними і x (i =1,m). У результаті отримаємо

$$\frac{\partial S}{\partial x_k} = 2\sum_{i=1}^{n} a_{ik}\left(\sum_{j=1}^{m} a_{ij}x_j - b_j\right), \qquad k = \overline{1,m}.$$
(рис. 6)

Прирівнявши вирази (рис. 6) до нуля, отримаємо нормальну систему рівнянь

$$\sum_{i=1}^{n} a_{ik}\left(\sum_{j=1}^{m} a_{ij}x_j - b_j\right) = 0, \qquad k = \overline{1,m},$$
(рис. 7)

в якій кількість рівнянь системи дорівнює кількості невідомих. Нормальні системи лінійних алгебраїчних рівнянь характеризуються тим, що матриці їх коефіцієнтів завжди є симетричними, а діагональні елементи - додатніми.

Систему (рис. 7) розв'язують довільними прямими або ітераційними методами. Якщо матриця коефіцієнтів системи рівнянь (рис. 7) є додатньо визначеною (визначник матриці є більшим за нуль), то рекомендують для її розв’язування використовувати метод квадратного кореня.

Запишемо систему лінійних алгебраїчних рівнянь (рис. 1) у матричному вигляді

$$AX = B,$$

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & & & \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}, \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}, \quad B = \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_m \end{pmatrix},$$
(рис. 8)

де A - матриця коефіцієнтів системи розмірності $m \times n$, X – матриця-стовпець невідомих розмірності $m \times 1$, B - матриця-стовпець вільних членів системи розмірності $m \times 1$.

Матричне рівняння (рис. 8) помножимо на транспоновану матрицю A^T до матриці A . У результаті отримаємо матричне рівняння

$$NX = C$$ (рис. 9)

де N – матриця коефіцієнтів нормальної системи

$$N = A^T A,$$ (рис. 10)

C -стовпець вільних членів

$$C = A^T B.$$ (рис. 11)

Розв’язавши нормальну систему лінійних алгебраїчних рівнянь, отримаємо її точний розв’язок (якщо використано прямі методи) або наближений розв’язок (якщо використано ітераційні методи). Отриманий розв’язок буде наближеним для СЛАР (рис. 1).

Метод квадратного кореня

Якщо матриця невироджена та симетрична, то ми можемо розкласти її на добуток матриць $A = LDL^T$, де L — одинична нижня трикутна матриця; D — діагональна матриця.

Отримаємо систему:

$LDL^T * x = b$

Розв'язок отримаємо послідовно розв'язавши дві трикутні СЛАР:

$LD * y = b$ та

$L^T * x = y$.

Порівняно з загальнішими методами (метод Гауса чи LU-розклад матриці) він стійкіший і потребує вдвічі менше арифметичних операцій.

Результат виконання завдання

```
Initial matrix:

 1       3      -2               -5
-1       2       1                1
 3      -2      -2               -5
 3       1      -3                1
 1      -1      -7                5


Result:

x1: -1.7163
x2: -0.456919
x3: -0.866864
```

Рис. 12. Результат обчислення СЛАР методом найменших квадратів

Висновки

У результаті виконання лабораторної роботи створено обчислювальний алгоритм, за допомогою мови C++, для розв’язування перевизначеної системи лінійних алгебраїчних рівнянь методом найменших квадратів для заданої системи лінійних алгебраїчних рівнянь.

Додаток

LeastSquares.h:

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

using namespace std;

class LeastSquares
{
public:
	LeastSquares() = delete;

	template <class T>
	LeastSquares(const T matrix, const size_t num_of_variables, const size_t num_of_equations, const vector<double> free_terms);
```

```
	vector<double> Find();

private:
	vector<vector<double>> m_matrix;
	vector<double> m_free_terms;
	size_t m_num_of_variables;
	size_t m_num_of_equation;

	template <typename T>
	vector<vector<double>> CopyMatrix(const T matrix, const size_t
num_of_variables, const size_t num_of_equasions) const;

	vector<vector<double>> TranspondMatrix(const vector<vector<double>> matrix)
const;

	vector<vector<double>> MultiplyMatrixes(const vector<vector<double>>
first_matrix, const vector<vector<double>> second_matrix) const;

	vector<double> MultiplyMatrixAndColumn(const vector<vector<double>> matrix,
const vector<double> column) const;

	double FindDeterminant(const vector<vector<double>> matrix) const;

	vector<vector<double>> SplitMatrixIntoToTransponded(const
vector<vector<double>> matrix) const;

	vector<double> GetY(const vector<vector<double>> matrix, const
vector<double> new_free_terms) const;

	vector<double> GetX(const vector<vector<double>> matrix, const
vector<double> y) const;

};


template <typename T>
vector<vector<double>> LeastSquares::CopyMatrix(const T matrix, const size_t
num_of_variables, const size_t num_of_equasions) const {
	vector<vector<double>> new_vector(num_of_equasions,
vector<double>(num_of_variables));

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

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

	} while (false);

	return new_vector;
}

template <class T>
LeastSquares::LeastSquares(const T matrix, const size_t num_of_variables, const
size_t num_of_equations, const vector<double> free_terms) {

	this->m_num_of_equation = num_of_equations;
	this->m_num_of_variables = num_of_variables;
	this->m_matrix = CopyMatrix(matrix, num_of_variables, num_of_equations);
	this->m_free_terms = free_terms;
}
```

Lab_06_NM.cpp:

```
#include "LeastSquares.h"

int main()
{
	const double result_accuracy = 0.0001;
	const size_t num_of_equations{ 5 };
	const size_t num_of_variables{ 3 };
	double matrix[num_of_equations][num_of_variables] ={{ 1,  3, -2},

	{-1,  2,  1},

	{ 3, -2, -2},

	{ 3,  1, -3},

	{ 1, -1, -7} };

	vector<double> free_terms{ -5, 1, -5, 1, 5 };



	LeastSquares ls(matrix, num_of_variables, num_of_equations, free_terms);
	vector<double> result = ls.Find();

	cout << "Initial matrix:\n\n";
	for (int i = 0; i < num_of_equations; i++) {
		for (int j = 0; j < num_of_variables; j++) {
			if (matrix[i][j] >= 0)
				cout << " ";
			cout << matrix[i][j] << "\t";
		}
		cout << "\t";
		if (free_terms[i] >= 0)
			cout << " ";
		cout << free_terms[i] << endl;
	}
	cout << "\n\n";

	cout << "Result:\n\n";
	for (int i = 0; i < result.size(); i++) {
		cout << "x" << i + 1 << ": " << result[i] << "\n";
	}
	cout << endl;
}
```

LeastSquares.cpp:

```
#include "LeastSquares.h"

vector<vector<double>> LeastSquares::TranspondMatrix(const vector<vector<double>>
matrix) const {
	vector<vector<double>> transponded(matrix[0].size(),
vector<double>(matrix.size()));

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

	return transponded;
}

vector<double> LeastSquares::MultiplyMatrixAndColumn(const vector<vector<double>>
matrix, const vector<double> column) const {
	vector<double> local_column(column.size());

	double sum = 0;
```

```
	for (int i = 0; i < matrix.size(); i++) {
		for (int j = 0; j < matrix[0].size(); j++) {
			local_column[i] += matrix[i][j] * column[j];
		}
	}

	return local_column;
}

double LeastSquares::FindDeterminant(const vector<vector<double>> matrix) const {

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

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

	vector<vector<double>> smaller_matrix(matrix_size - 1,
vector<double>(matrix_size - 1));

	double 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<vector<double>> LeastSquares::MultiplyMatrixes(const vector<vector<double>>
first_matrix, const vector<vector<double>> second_matrix) const {
	vector<vector<double>> multiplication(first_matrix.size(),
vector<double>(second_matrix[0].size()));

	for (int i = 0; i < first_matrix.size(); i++) {
		for (int j = 0; j < second_matrix[0].size(); j++) {
			multiplication[i][j] = 0;
			for (int k = 0; k < first_matrix[0].size(); k++) {
				multiplication[i][j] += first_matrix[i][k] *
second_matrix[k][j];
			}
		}
	}

	return multiplication;
}

vector<vector<double>> LeastSquares::SplitMatrixIntoToTransponded(const
vector<vector<double>> matrix) const {
```

```
	vector<vector<double>> square_matrix(matrix.size(),
vector<double>(matrix.size()));

	for (int i = 0; i < matrix.size(); i++) {

		square_matrix[i][i] = matrix[i][i];
		for (int j = 0; j < i; j++) {
			square_matrix[i][i] -= pow(square_matrix[j][i], 2);
		}
		square_matrix[i][i] = sqrt(square_matrix[i][i]);

		for (int j = i + 1; j < matrix.size(); j++) {

			square_matrix[i][j] = matrix[i][j];
			for (int k = 0; k < i; k++) {
				square_matrix[i][j] -= square_matrix[k][i] *
square_matrix[k][j];
			}
			square_matrix[i][j] /= square_matrix[i][i];
		}
	}

	return square_matrix;
}

vector<double> LeastSquares::GetY(const vector<vector<double>> matrix, const
vector<double> new_free_terms) const {
	vector<double> y(matrix.size());

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

		y[i] = (new_free_terms[i] - y[i]) / matrix[i][i];
	}

	return y;
}

vector<double> LeastSquares::GetX(const vector<vector<double>> matrix, const
vector<double> y) const {
	vector<double> x(matrix.size());

	for (int i = matrix.size() - 1; i >= 0; i--) {
		for (int k = matrix.size() - 1; k > i; k--)
			x[i] += x[k] * matrix[i][k];

		x[i] = (y[i] - x[i]) / matrix[i][i];
	}

	return x;
}

vector<double> LeastSquares::Find() {

	vector<vector<double>> transponded_matrix = TranspondMatrix(m_matrix);
	vector<vector<double>> multiplied = MultiplyMatrixes(transponded_matrix,
m_matrix);
	vector<double> new_free_terms = MultiplyMatrixAndColumn(transponded_matrix,
m_free_terms);

	if (!FindDeterminant(multiplied)) {
		cout << "Determinant is equal zero, try another method";
		return vector<double>(multiplied.size());
	}

	vector<vector<double>> splited_upper =
SplitMatrixIntoToTransponded(multiplied);
	vector<vector<double>> splited_lower = TranspondMatrix(splited_upper);

	vector<double> y = GetY(splited_lower, new_free_terms);
```

```
        vector<double> x = GetX(splited_upper, y);

        return x;
}
```