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

Звіт
Про виконання лабораторної роботи № 9

На тему:
«Наближення функції методом найменших квадратів»

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

Тема роботи: Наближення функції методом найменших квадратів.
Мета роботи: Ознайомлення на практиці з методом апроксимації (наближення) функції методом найменших квадратів.

Теоретичні відомості

Нехай функція $y = f(x)$ задана таблицею своїх значень: $y_i = f(x_i)$, $i = \overline{0,n}$. Потрібно знайти многочлен фіксованого степеня m, для якого похибка апроксимації - середньоквадратичне відхилення (СКВ)

$$\sigma = \sqrt{\frac{1}{n+1}\sum_{i=0}^{n}\left(P_m(x_i) - y_i\right)^2}$$

мінімальне.

Так як многочлен $P_m(x) = a_0 + a_1 x + a_2 x^2 ... + a_m x^m$ визначається своїми коефіцієнтами, то фактично треба підібрати набір коефіцієнтів $a_0, a_1, a_2, ...a_m$, який мінімізує функцію

$$\Phi(a_0, a_1, a_2, ...a_m) = \sum_{i=0}^{n}\left(P_m(x_i) - y_i\right)^2 = \sum_{i=0}^{n}\left(\sum_{j=0}^{m} a_j x_i^{\ j} - y_i\right)^2$$

Використовуючи необхідну умову екстремуму $\frac{\partial\Phi}{\partial a_k} = 0$, $k = \overline{0,m}$, отримуємо так звану ***нормальну систему*** методу найменших квадратів:

$$\sum_{j=0}^{m}\left(\sum_{i=0}^{n} x_i^{\ j+k}\right) a_j = \sum_{i=0}^{n} y_i x_i^{\ k}, \qquad k = \overline{0,m}.$$

Отримана система - це ***система алгебраїчних рівнянь*** відносно невідомих $a_0, a_1, a_2, ...a_m$. Можна показати, що визначник цієї системи відмінний від нуля, тобто розв’язок існує і єдиний. Однак при високих степенях m система є погано обумовленою. Тому метод найменших квадратів застосовують для знаходження многочленів, ступінь яких не вищий, ніж 5. Розв’язок нормальної системи можна знайти, наприклад, методом Гаусса.

Нормальна система методу найменших квадратів

Запишемо нормальну систему найменших квадратів для двох простих випадків: m = 0 і m = 2.

При <u>m = 0</u> многочлен прийме вигляд:

$$P_0(x) = a_0 .$$

Для знаходження невідомого коефіцієнта a_0 маємо рівняння:

$$(n+1)a_0 = \sum_{i=0}^{n} y_i .$$

Отримуємо, що коефіцієнт a_0 дорівнює середньому арифметичному значень функції в заданих точках.

Якщо ж використовується многочлен другого порядку

$$P_2(x) = a_0 + a_1 x + a_2 x^2 ,$$

то нормальна система рівнянь набуде вигляду:

$$\begin{cases} (n+1)\,a_0 + \left(\sum_{i=0}^{n} x_i\right) a_1 + \left(\sum_{i=0}^{n} x_i^{\,2}\right) a_2 = \sum_{i=0}^{n} y_i \\ \left(\sum_{i=0}^{n} x_i\right) a_0 + \left(\sum_{i=0}^{n} x_i^{\,2}\right) a_1 + \left(\sum_{i=0}^{n} x_i^{\,3}\right) a_2 = \sum_{i=0}^{n} y_i x_i \\ \left(\sum_{i=0}^{n} x_i^{\,2}\right) a_0 + \left(\sum_{i=0}^{n} x_i^{\,3}\right) a_1 + \left(\sum_{i=0}^{n} x_i^{\,4}\right) a_2 = \sum_{i=0}^{n} y_i x_i^{\,2} \end{cases}$$

Індивідуальне завдання

Методом найменших квадратів побудувати лінійний, квадратичний і кубічний апроксимаційні поліноми для таблично заданої функції.

x	f(x)	x	f(x)
0,0	I,758203	0,5	I,654I40
0,I	I,738744	0,6	I,632460
0,2	I,7I8369	0,7	I,6II005
0,3	I,697320	0,8	I,589975
0,4	I,675834	0,9	I,569559

Рис. 1. Таблично задана функція.

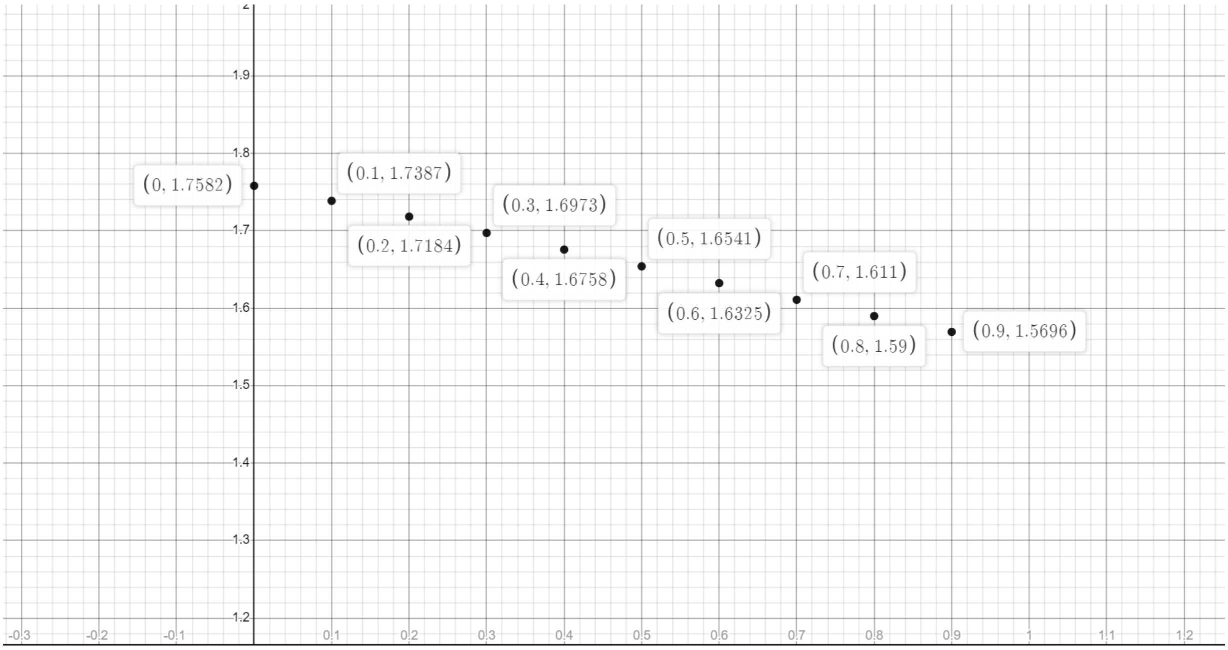

Рис. 2. Геометричне зображення таблично заданої функції.

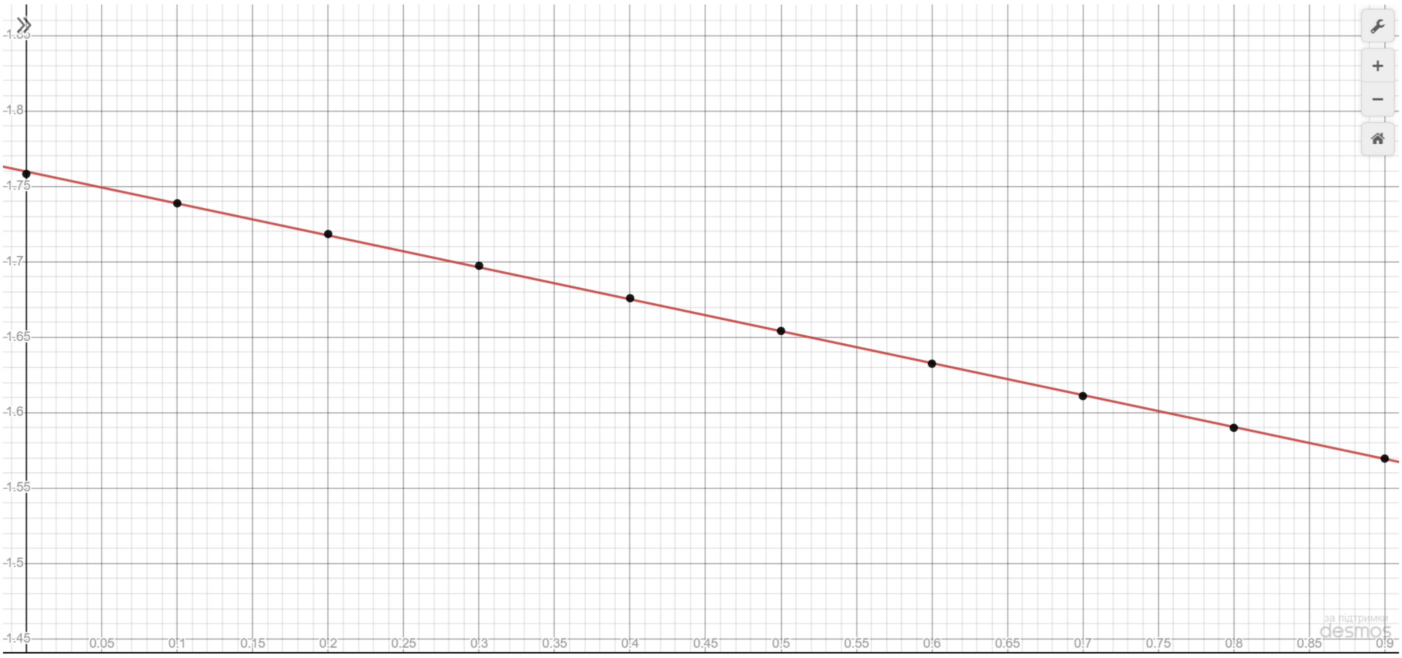

Рис. 3. Геометричне зображення лінійного апроксимаційного поліному.

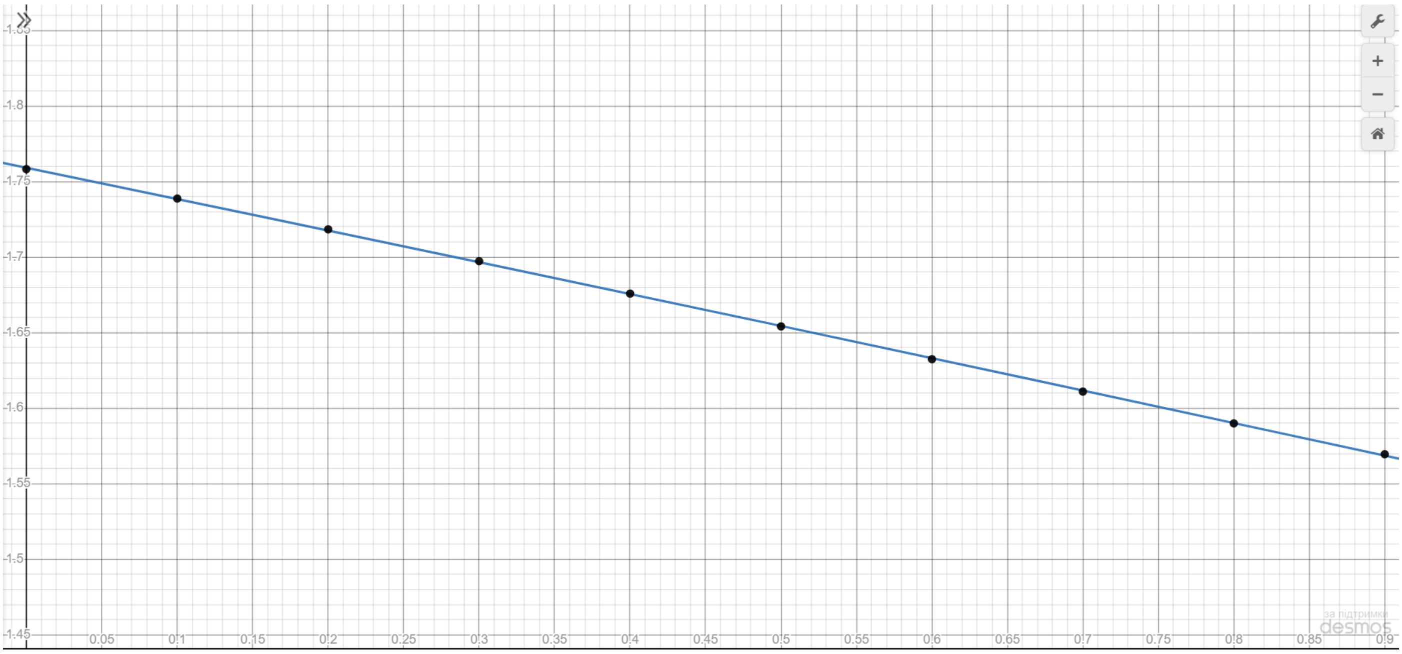

Рис. 4. Геометричне зображення квадратичного апроксимаційного поліному.

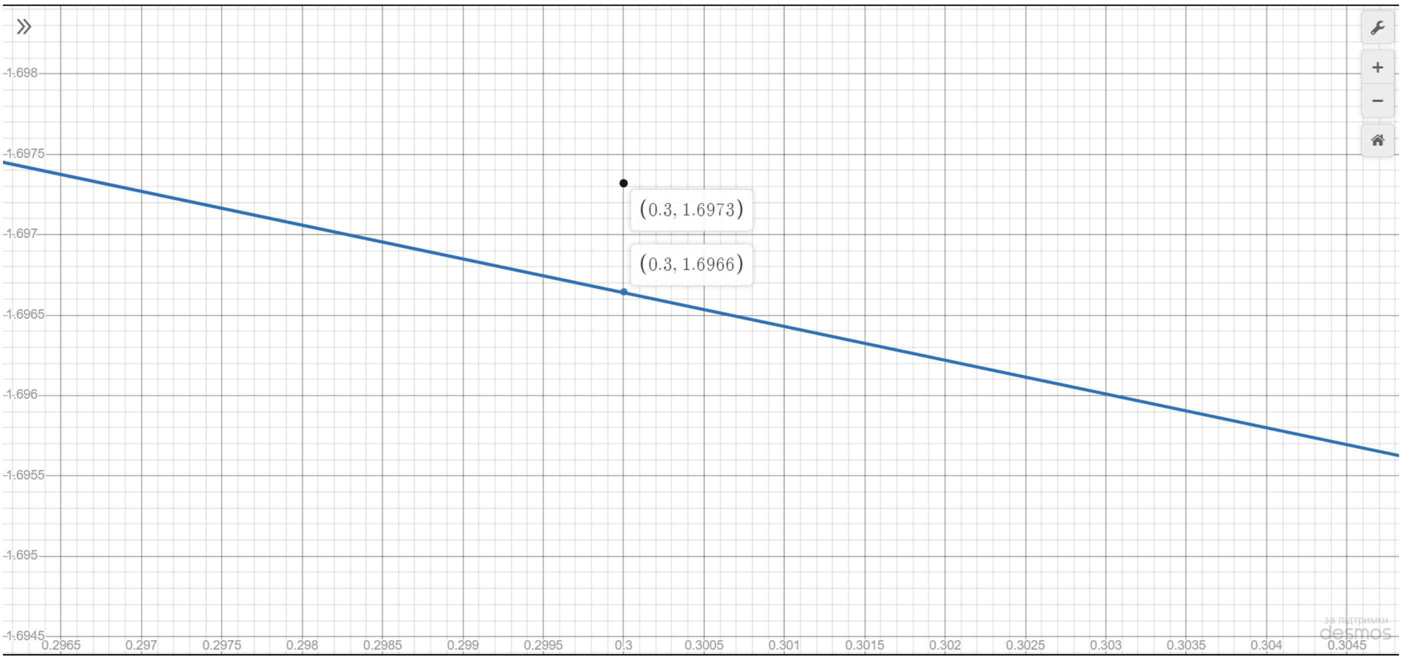

Рис. 5. Геометричне зображення наближення квадратичного апроксимаційного поліному (його точки).

Рис. 6. Геометричне зображення кубічного апроксимаційного поліному.

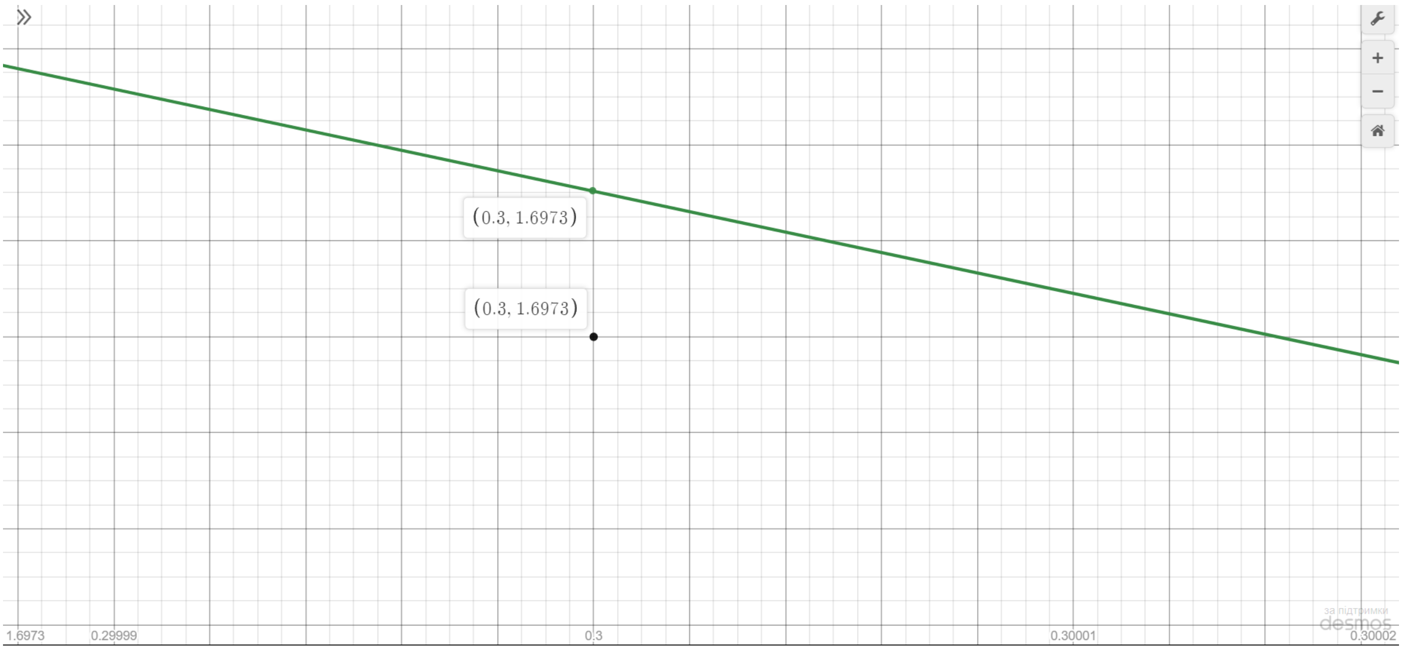

Рис. 7. Геометричне зображення наближення кубічного апроксимаційного поліному (його точки).

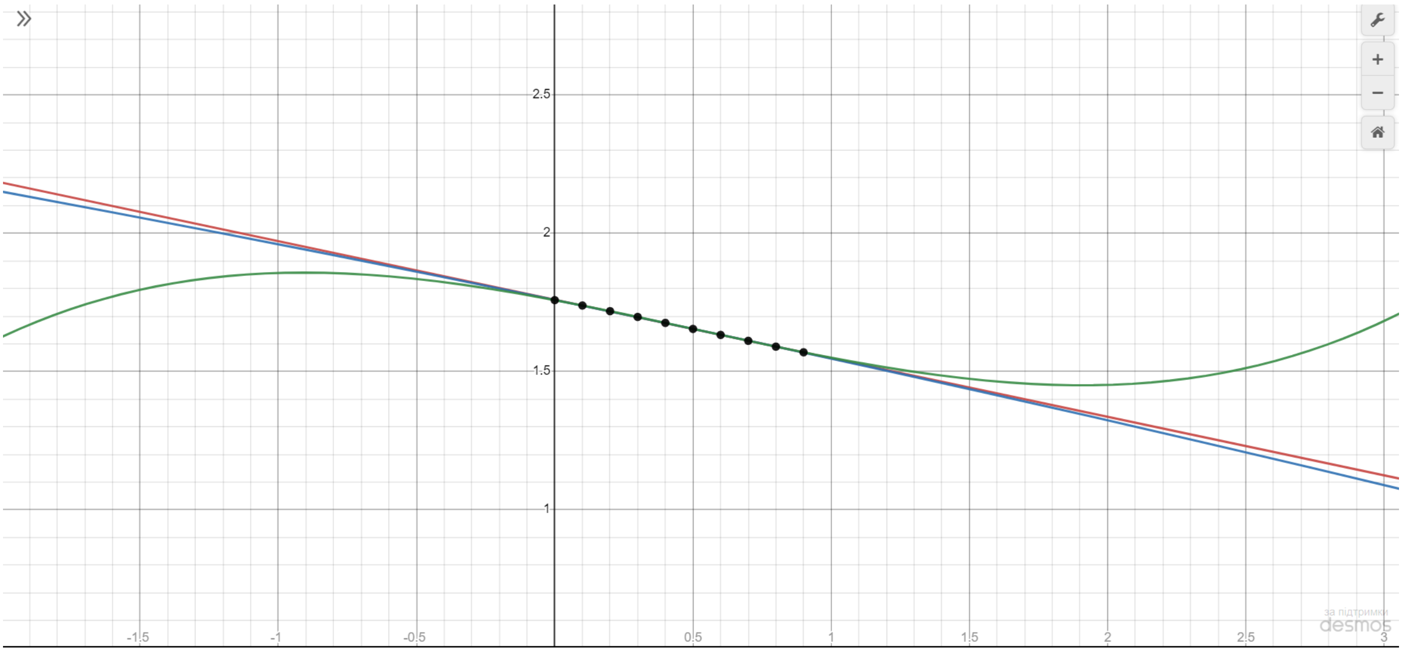

Рис. 8. Геометричне зображення всіх знайдених апроксимаційних поліномів у лабораторній роботі.

Результат виконання програми

```
Linear: +1.7598x^0 -0.211653x^1
Precision is: 0.00101116


Quadratic:      +1.75914x^0 -0.206676x^1 -0.0055303x^2
Precision is: 0.000679345


Cubic:  +1.75822x^0 -0.1898x^1 -0.054948x^2 +0.0366057x^3
Precision is: 1.52005e-06
```

Висновки

У результаті виконання лабораторної роботи, реалізовано програму побудови лінійного, квадратичного і кубічного апроксимаційних поліномів для таблично заданої функції методом найменших квадратів. Знайдено похибки апроксимації, для лінійного апроксимаційного поліному – 0.00101116, для квадратичного – 0.000679345, для кубічного – $1.52005 * 10^{-6}$. Нормальну систему рівнянь для визначення коефіцієнтів апроксимаційних поліномів розв’язано методом LU - розкладу.

Додаток

LeastSquares.h:

```
#pragma once

#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>

using namespace std;

	vector<double> Find(vector<double> x, vector<double> y, unsigned int m);
```

LeastSquares.cpp:

```
#include "LeastSquares.h"
#include "..//Methods_Lib/Methods_Lib_Header.h"

vector<double> Find(vector<double> x, vector<double> y, unsigned int m) {

	vector<double> working_x(x.size());
	vector<double> coefficients;
	vector<vector<double>> matrix_coefficients(m + 1, vector<double>(m + 1));
	vector<double> free_terms;

	for (int i = 0; i <= m * 2; i++) {

		copy(x.begin(), x.end(), working_x.begin());

		for (double& element : working_x) {
			element = pow(element, i);
		}

		coefficients.push_back(accumulate(working_x.begin(), working_x.end(),
0.0));

		if (i <= m) {
			free_terms.push_back(inner_product(y.begin(), y.end(),
working_x.begin(), 0.0));
		}
	}
```

```
	for (int i = 0; i <= m; i++) {
		copy(coefficients.begin() + i, coefficients.begin() + m + 1 + i, 
matrix_coefficients[i].begin());
	}

	SystemSolver holder(matrix_coefficients, free_terms);

	return  holder.LU();
}
```

Lab_09_NM.cpp:

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

int main()
{
	vector<double> x{ 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 };
	vector<double> y{ 1.758203, 1.738744, 1.718369, 1.697320, 1.675834, 1.654140, 
1.632460, 1.611005, 1.589975, 1.569559 };

	vector<double> func_0 = Find(x, y, 1);
	vector<double> func_1 = Find(x, y, 2);
	vector<double> func_2 = Find(x, y, 3);

	double difference{ 0 };

	cout << "Linear:\t";
	for(int i = 0; i < func_0.size(); i++)
	{
		if (func_0[i] >= 0)
			cout << "+";
		cout << func_0[i] << "x^" << i << " ";

		difference += func_0[i] * pow(x[3], i);
	}
	difference -= y[3];
	cout << "\n\nPrecision is: " << fabs(difference) << "\n\n\n\n";

	difference = 0;
	cout << "Quadratic:\t";
	for(int i = 0; i < func_1.size(); i++)
	{
		if (func_1[i] >= 0)
			cout << "+";
		cout << func_1[i] << "x^" << i << " ";

		difference += func_1[i] * pow(x[3], i);
	}
	difference -= y[3];
	cout << "\n\nPrecision is: " << fabs(difference) << "\n\n\n\n";

	difference = 0;
	cout << "Cubic:\t";
	for(int i = 0; i < func_2.size(); i++)
	{
		if (func_2[i] >= 0)
			cout << "+";
		cout << func_2[i] << "x^" << i << " ";

		difference += func_2[i] * pow(x[3], i);
	}
	difference -= y[3];
	cout << "\n\nPrecision is: " << fabs(difference) << "\n\n\n\n";
}
```

Methods_Lib_Header.h:

```
#pragma once
```

```cpp
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

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

	template <typename T>
	vector<vector<double>> CopyMatrix(const T matrix, const size_t size);
	template <typename T>
	size_t GetSize(const T matrix) const;
	vector<vector<double>> CreateMatrix(const size_t size) const;
	double FindDeterminant(const vector<vector<double>> matrix) const;
	void GaussItself(vector<vector<double>>& matrix, vector<double>& B);

public:

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

	SystemSolver(vector<vector<double>> matrix, vector<double> B) {
		this->matrix = matrix;
		this->B = B;
	}

	vector<double> Gauss();
	vector<double> 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<double>> SystemSolver::CopyMatrix(const T matrix, const size_t size) {
	vector<vector<double>> new_vector(size, vector<double>(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<double> B) {
	this->matrix = CopyMatrix(matrix, GetSize(matrix));
	this->B = B;
}
```

```
void Show(vector<vector<double>> matrix, string name);
```

Methods_Lib.cpp:

```
#include "Methods_Lib_Header.h"

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

	return new_matrix;
}
double SystemSolver::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 = CreateMatrix(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<double> SystemSolver::Gauss() {
	vector <double> result(matrix[0].size());
	vector<vector<double>> inside_matrix = CopyMatrix(this->matrix,
matrix[0].size());
	vector<double> 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<double>>& matrix, vector<double>& B) {
	int index_of_row_with_max_element{ 0 };
	double 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<double> temp_row(size_of_matrix);
				double 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++) {
				double 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<double> SystemSolver::LU() {
	size_t size = matrix[0].size();

	vector<vector<double>> l(size, vector<double>(size));
	vector<vector<double>> u(size, vector<double>(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--;
		}
	}

	Show(l, string("L"));
	Show(u, string("U"));

	vector<double> 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];
	}

	cout << "Free term is: " << endl << endl;
	for (int i = 0; i < size; i++)
		cout << B[i] << "\t";
	cout << endl << endl;

	cout << "Y is: " << endl << endl;
	for (int i = 0; i < size; i++)
		cout << y[i] << "\t";
	cout << endl << endl;

	vector<double> 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;
}


//////////

void Show(vector<vector<double>> matrix, string name) {
	cout << name << " matrix: " << endl << endl;
	for (int i = 0; i < matrix.size(); i++) {
		for (int j = 0; j < matrix.size(); j++) {
			cout << matrix[i][j] << "\t";
		}
		cout << endl;
	}
	cout << endl << endl;
}
```