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  задана таблицею своїх значень: , . Потрібно знайти многочлен фіксованого степеня m, для якого похибка апроксимації - середньоквадратичне відхилення (СКВ)



мінімальне.

Так як многочлен  визначається своїми коефіцієнтами, то фактично треба підібрати набір коефіцієнтів , який мінімізує функцію



Використовуючи необхідну умову екстремуму , , отримуємо так звану ***нормальну систему*** методу найменших квадратів:

, .

Отримана система - це ***система алгебраїчних рівнянь*** відносно невідомих . 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 m = 0 многочлен прийме вигляд:

.

Для знаходження невідомого коефіцієнта  маємо рівняння:

.

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

Якщо ж використовується многочлен другого порядку

,

то нормальна система рівнянь набуде вигляду:



**Індивідуальне завдання**

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

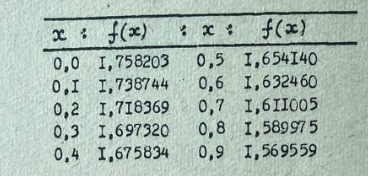


Рис. 1. Таблично задана функція.

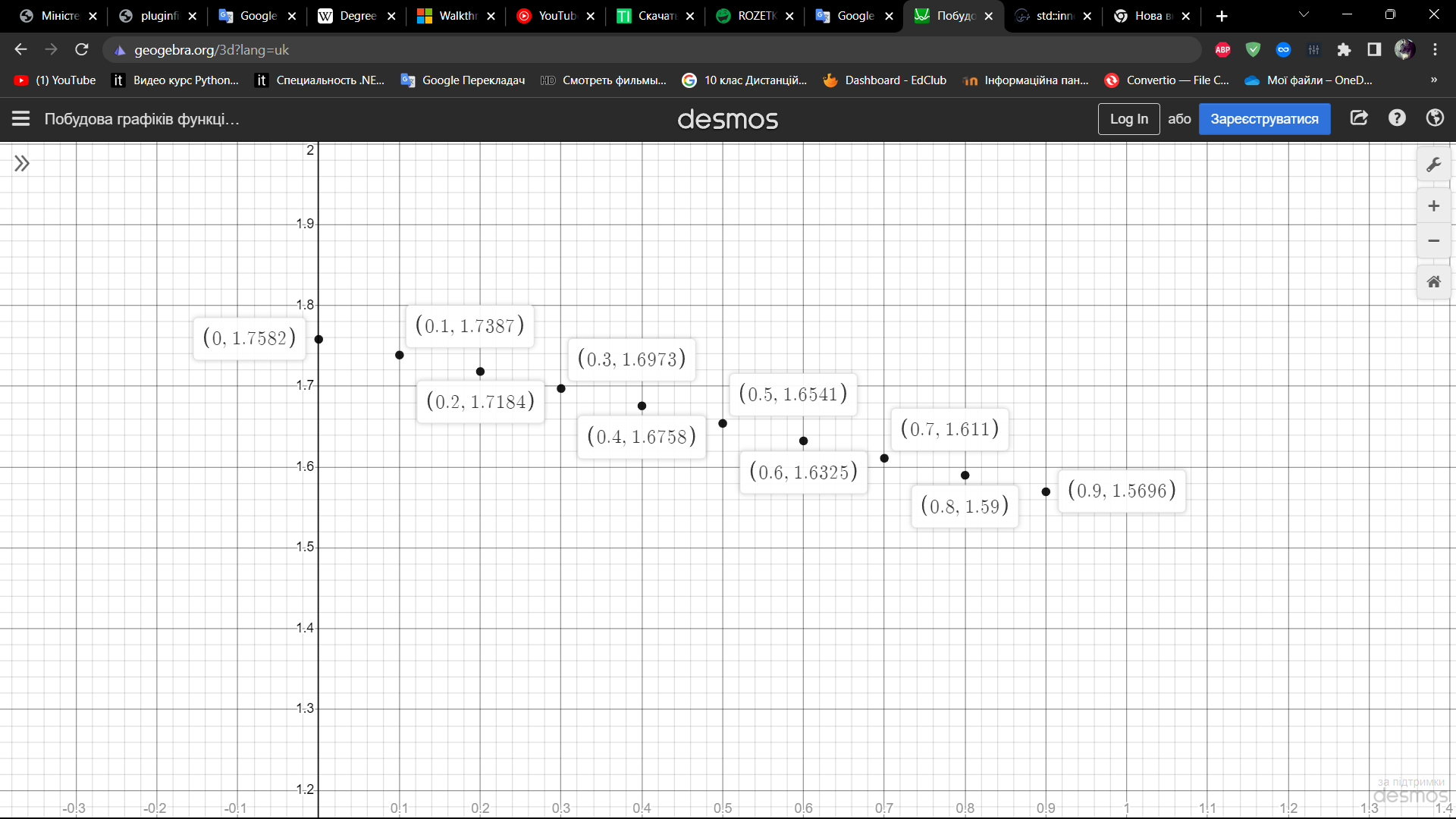


Рис. 2. Геометричне зображення таблично заданої функції

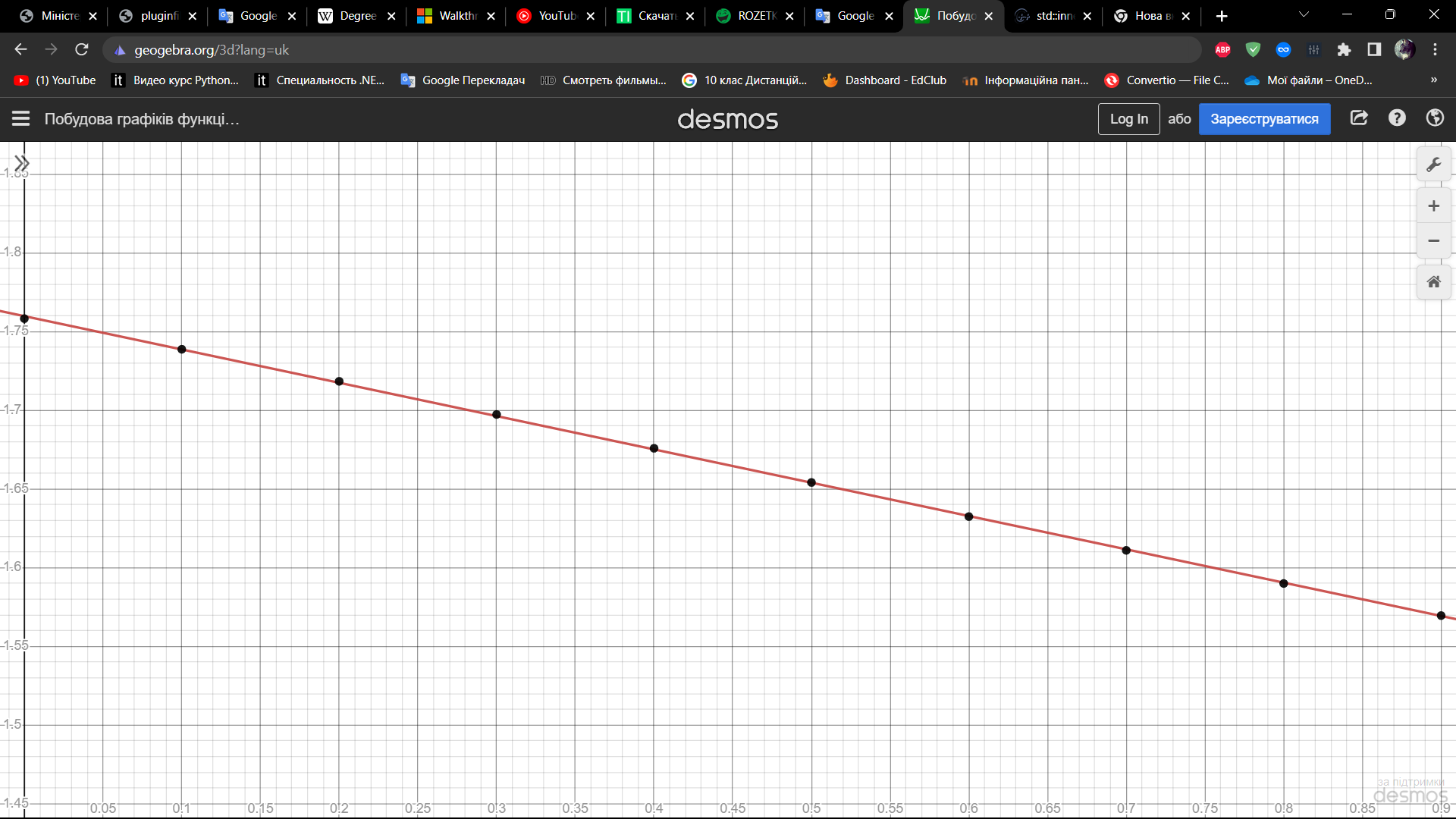
****

Рис. 3. Геометричне зображення лінійного апроксимаційного поліному



Рис. 4. Геометричне зображення квадратичного апроксимаційного поліному

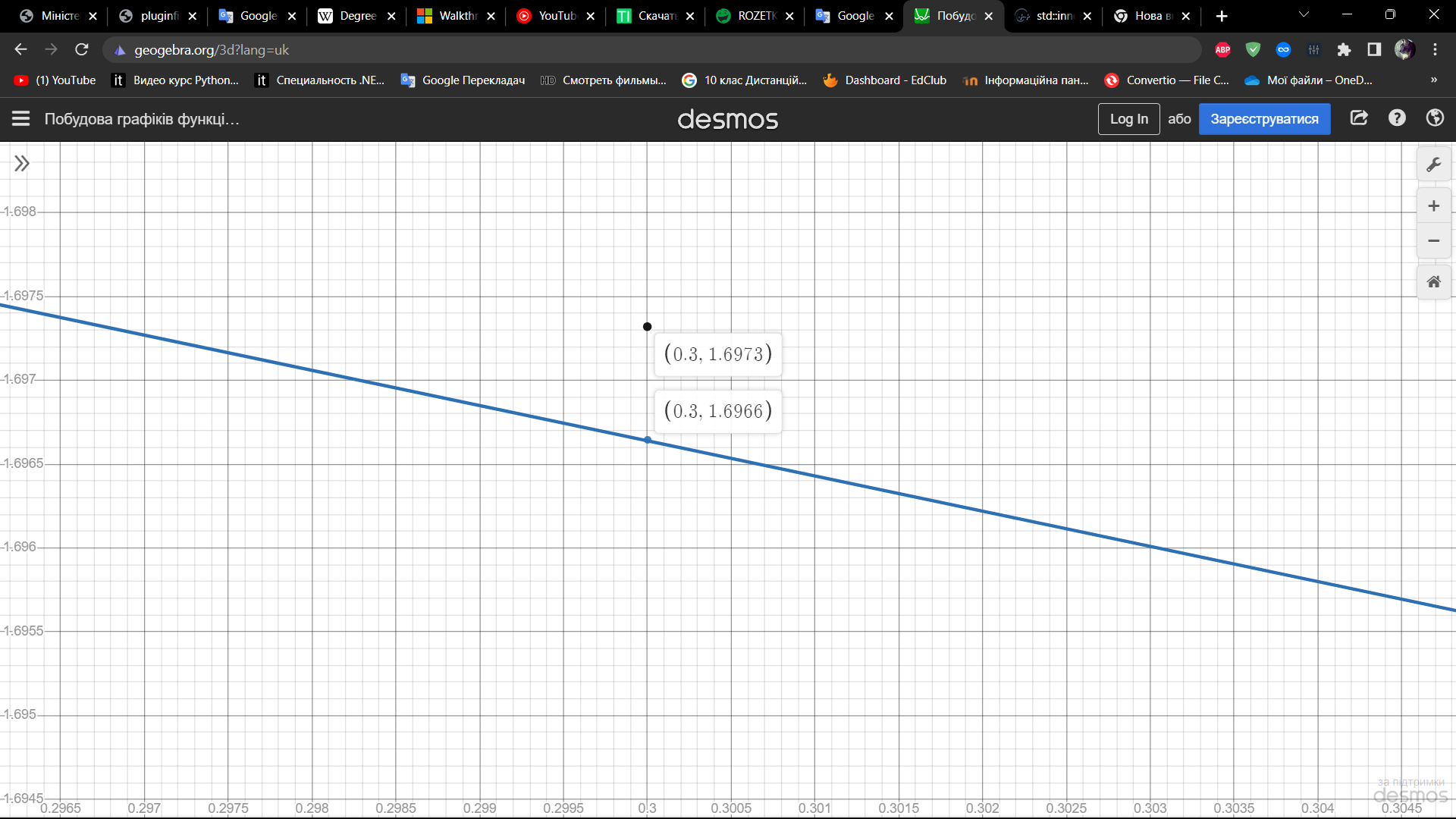


Рис. 5. Геометричне зображення квадратичного апроксимаційного поліному (його точки)

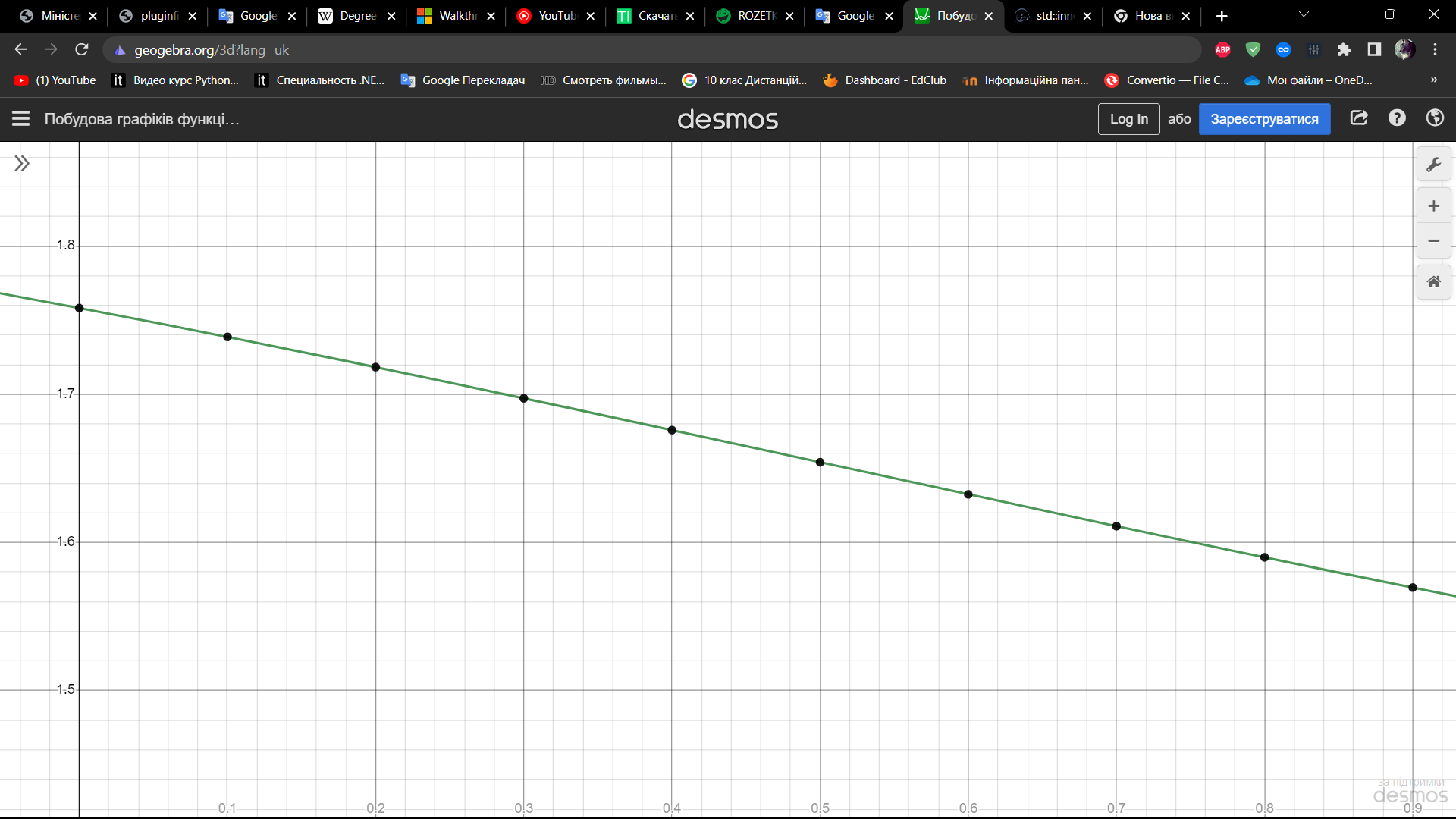
****

Рис. 6. Геометричне зображення кубічного апроксимаційного поліному

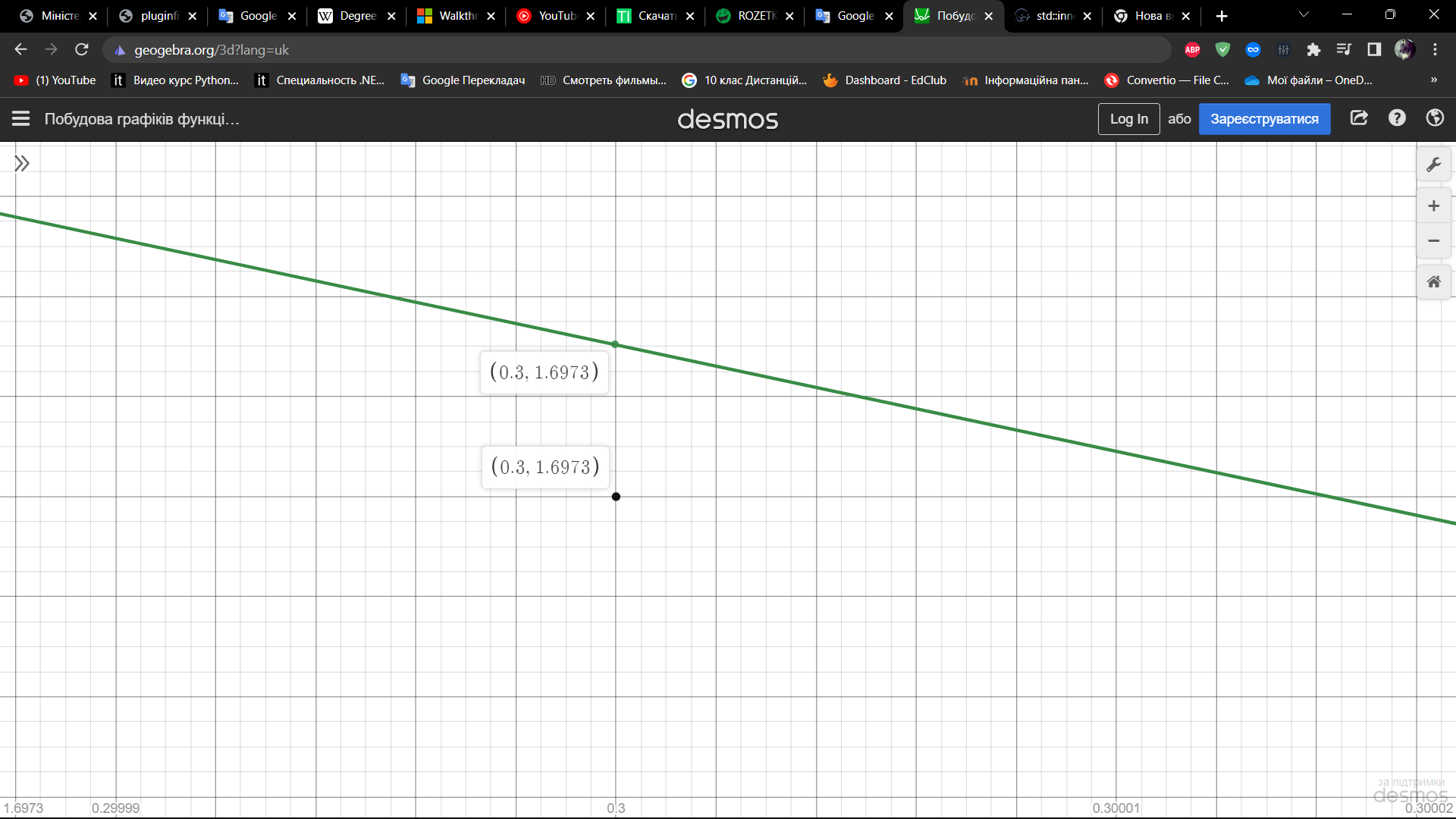
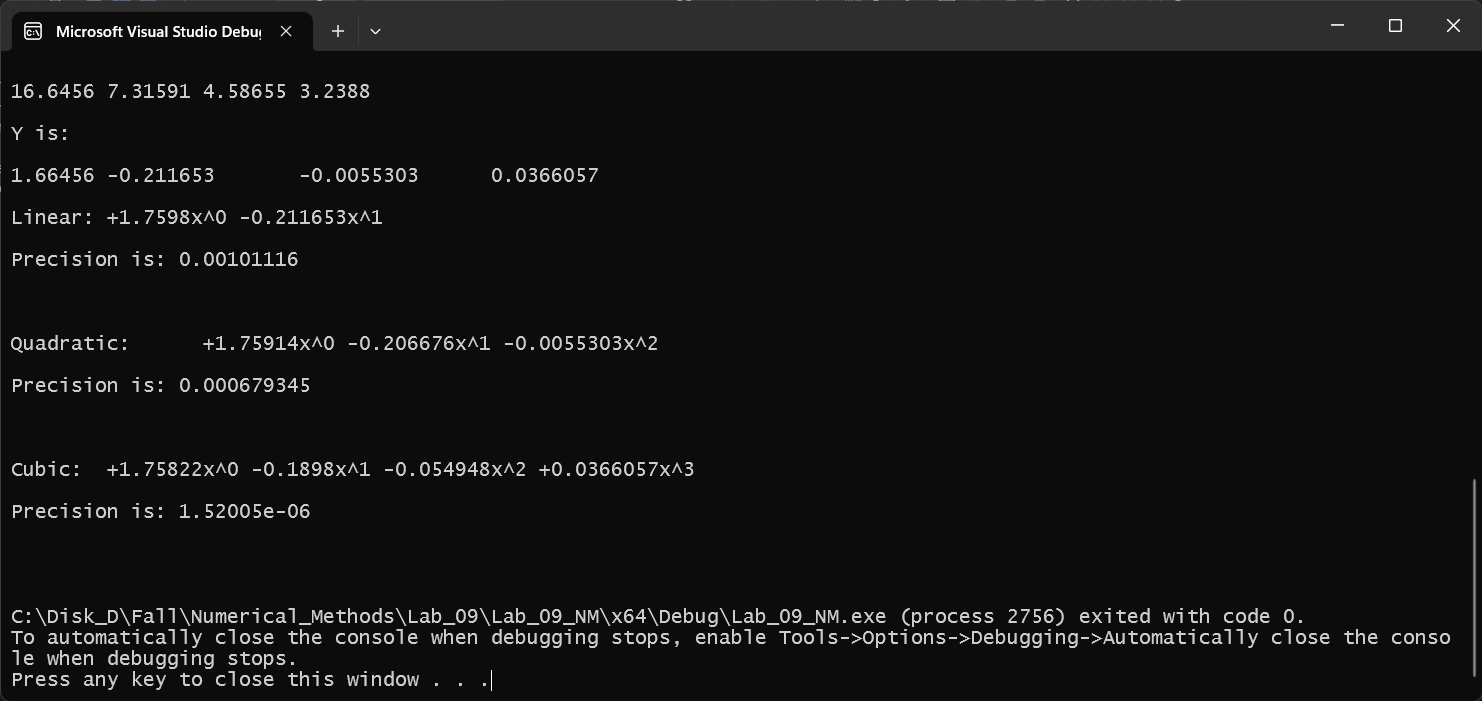
****

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

****

Рис. 8. Геометричне зображення всіх знайдених апроксимаційних поліномів, зокрема лінійний – червоний, квадратичний – синій та кубічний – зелений, у лабораторній роботі

**Результат виконання програми**

****

**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 та для кубічного – . 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 << "Сubic:\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

#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;

}