Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота №2

з дисципліни «Спеціальні розділи математики»

на тему

«Розв’язання систем лінійних алгебраїчних рівнянь (СЛАР) прямими методами.

Звичайний метод Гауса та метод квадратних коренів»

Виконав:

студент гр. ІС-02

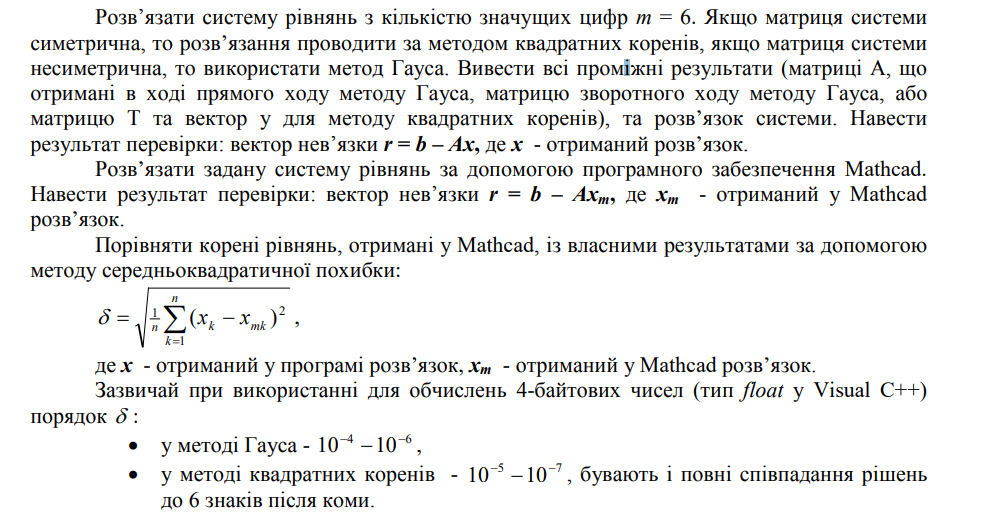
Плостак Ілля

Викладач:

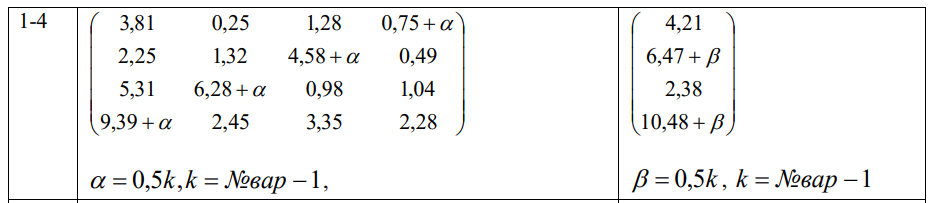
доц. Рибачук Л.В.

Київ – 2021

**1. Завдання**

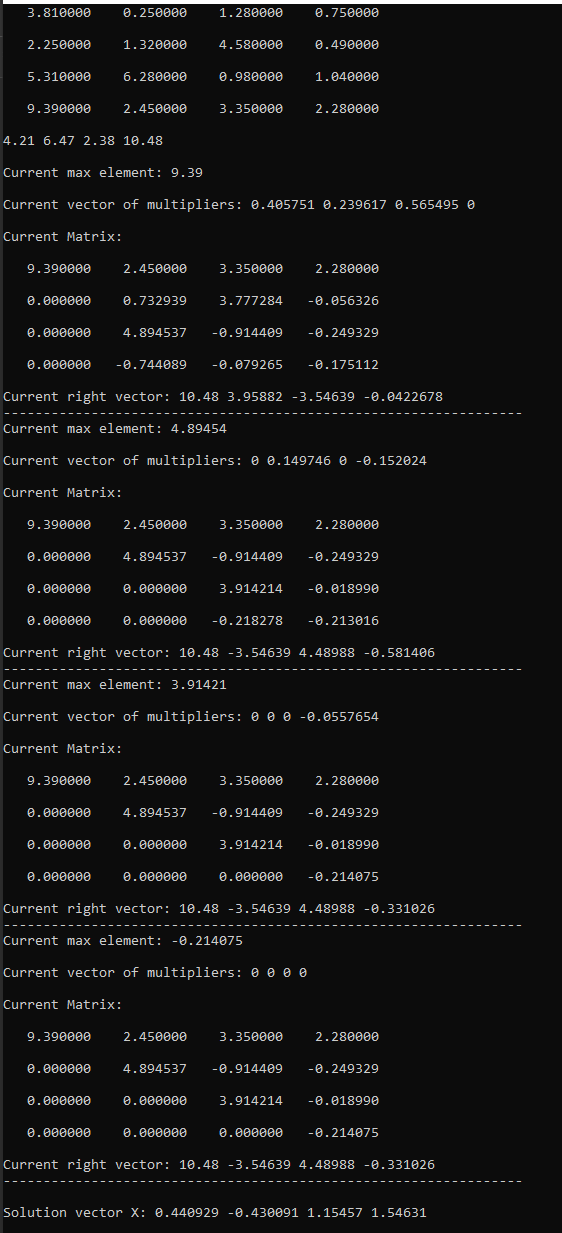


**2. Система рівнянь**



Варіант 1; k=0

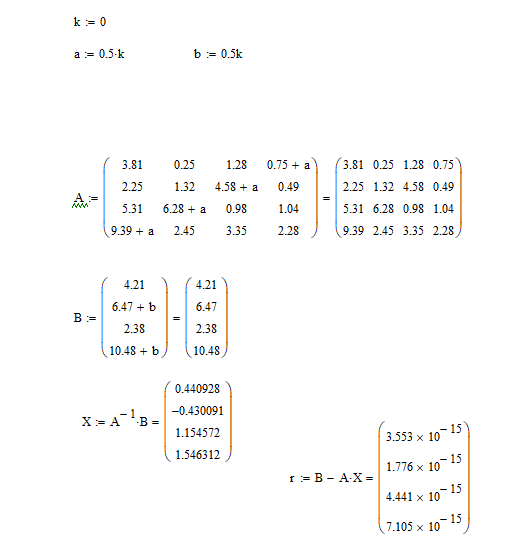
**3. Проміжні результати та кінцевий результат**



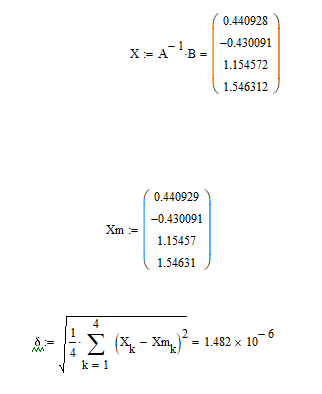
**4. Вектор нев’язки**



**5. Копія розв’язку задачі у Mathcad; вектор нев’язки для цього розв’язку**



**6. Порівняння власного розв’язку та розв’язку, отриманого у Mathcad**



**7. Лістинг програми**

[main.cpp](https://github.com/feedblackg44/kpilabs2/blob/master/CHM/Lab1/Lab1/main.cpp):

#include <iostream>

#include "functions.h"

using namespace std;

int main()

{

int size = 4;

int\* numbersX = new int[size];

fillInSeq(numbersX, size);

float\*\* A = createMatrix(size);

//fillRandom(A, size);

matrix4Init(A);

print(A, size);

float\* B = createVec(size);

//fillRandom(B, size);

vector4Init(B);

print(B, size);

float\* X = createVec(size);

gaussAlgorithmTriangle(A, B, numbersX, size);

gaussAlgorithmGetSolution(A, B, numbersX, X, size);

cout << "\n\nSolution vector X: ";

print(X, size);

matrix4Init(A);

vector4Init(B);

cout << "\nr = b - Ax:\n";

print(vecDelta(A, B, X, size), size);

system("pause");

return 0;

}

[functions.h](https://github.com/feedblackg44/kpilabs2/blob/master/CHM/Lab1/Lab1/functions.h):  
#pragma once

float\*\* createMatrix(int size);

float\* createVec(int size);

void matrix4Init(float\*\* M);

void vector4Init(float\* V);

void fillInSeq(int\* Vector, int size);

void fillRandom(float\*\* Matrix, int size);

void fillRandom(float\* Vector, int size);

void print(float\*\* Matrix, int size);

void print(float\* Vector, int size);

void print(float number);

int\* getIndexOfMaxElement(float\*\* Matrix, int size, int n);

float\* getMultipliersArray(float\*\* Matrix, int size, int\* key, int n);

void swapRows(float\*\* Matrix, float\* rightVector, int size, int numRow1, int numRow2);

void swapCols(float\*\* Matrix, int\* numbersX, int size, int numCol1, int numCol2);

void gaussStep(float\*\* Matrix, float\* rightVector, int\* keys, float\* multipl, int\* numbersX, int size, int n);

void gaussAlgorithmTriangle(float\*\* Matrix, float\* rightVector, int\* numbersX, int size);

void gaussAlgorithmGetSolution(float\*\* Matrix, float\* rightVector, int\* numbersX, float\* X, int size);

float Solution(float\*\* Matrix, float\* X, int n, int size);

float\* MatrixVecMult(float\*\* matrix1, float\* matrix2, int size);

float\* vecDelta(float\*\* Matrix, float\* rightVec, float\* X, int size);

[function.cpp](https://github.com/feedblackg44/kpilabs2/blob/master/CHM/Lab1/Lab1/functions.cpp):

#include <iostream>

#include <iomanip>

#include "functions.h"

using namespace std;

float\*\* createMatrix(int size)

{

float\*\* matrix = new float\* [size];

for (int i = 0; i < size; i++)

{

matrix[i] = new float[size];

}

return matrix;

}

float\* createVec(int size)

{

float\* vector = new float[size];

return vector;

}

void matrix4Init(float\*\* M)

{

M[0][0] = 3.81; M[0][1] = 0.25; M[0][2] = 1.28; M[0][3] = 0.75;

M[1][0] = 2.25; M[1][1] = 1.32; M[1][2] = 4.58; M[1][3] = 0.49;

M[2][0] = 5.31; M[2][1] = 6.28; M[2][2] = 0.98; M[2][3] = 1.04;

M[3][0] = 9.39; M[3][1] = 2.45; M[3][2] = 3.35; M[3][3] = 2.28;

}

void vector4Init(float\* V)

{

V[0] = 4.21;

V[1] = 6.47;

V[2] = 2.38;

V[3] = 10.48;

}

void fillInSeq(int\* Vector, int size)

{

for (int i = 0; i < size; i++)

{

Vector[i] = i;

}

}

void fillRandom(float\*\* Matrix, int size)

{

srand(time(NULL));

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

Matrix[i][j] = rand() % 201 - 100;

}

}

}

void fillRandom(float\* Vector, int size)

{

srand(time(NULL));

for (int i = 0; i < size; i++)

{

Vector[i] = rand() % 201 - 100;

}

}

void print(float\*\* Matrix, int size)

{

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

cout << fixed << setw(11) << setprecision(6) << Matrix[i][j] << " ";

}

cout << "\n" << endl;

}

cout << defaultfloat;

}

void print(float\* Vector, int size)

{

for (int i = 0; i < size; i++)

{

cout << Vector[i] << " ";

}

cout << endl;

}

void print(float number)

{

cout << number << endl;

}

int\* getIndexOfMaxElement(float\*\* Matrix, int size, int n)

{

int\* keys = new int[2];

float Max = abs(Matrix[n][n]);

keys[0] = n;

keys[1] = n;

for (int i = n; i < size; i++)

{

for (int j = n; j < size; j++)

{

if (abs(Matrix[i][j]) > Max)

{

Max = abs(Matrix[i][j]);

keys[0] = i;

keys[1] = j;

}

}

}

return keys;

}

float\* getMultipliersArray(float\*\* Matrix, int size, int\* key, int n)

{

float\* arr = new float[size];

for (int i = 0; i < size; i++)

{

if (i != key[0] && i >= n)

{

arr[i] = Matrix[i][key[1]]/Matrix[key[0]][key[1]];

}

else

{

arr[i] = 0;

}

}

return arr;

}

void swapRows(float\*\* Matrix, float\* rightVector, int size, int numRow1, int numRow2)

{

for (int i = 0; i < size; i++)

{

float temp = Matrix[numRow1][i];

Matrix[numRow1][i] = Matrix[numRow2][i];

Matrix[numRow2][i] = temp;

}

float temp = rightVector[numRow1];

rightVector[numRow1] = rightVector[numRow2];

rightVector[numRow2] = temp;

}

void swapCols(float\*\* Matrix, int\* numbersX, int size, int numCol1, int numCol2)

{

for (int i = 0; i < size; i++)

{

float temp = Matrix[i][numCol1];

Matrix[i][numCol1] = Matrix[i][numCol2];

Matrix[i][numCol2] = temp;

temp = numbersX[numCol1];

numbersX[numCol1] = numbersX[numCol2];

numbersX[numCol2] = temp;

}

}

void gaussStep(float\*\* Matrix, float\* rightVector, int\* keys, float\* multipl, int\* numbersX, int size, int n)

{

for (int i = n; i < size; i++)

{

for (int j = n; j < size; j++)

{

Matrix[i][j] -= multipl[i] \* Matrix[keys[0]][j];

}

rightVector[i] -= multipl[i] \* rightVector[keys[0]];

}

swapRows(Matrix, rightVector, size, n, keys[0]);

swapCols(Matrix, numbersX, size, n, keys[1]);

}

void gaussAlgorithmTriangle(float\*\* Matrix, float\* rightVector, int\* numbersX, int size)

{

for (int n = 0; n < size; n++)

{

int\* maxKeys = getIndexOfMaxElement(Matrix, size, n);

float maxA = Matrix[maxKeys[0]][maxKeys[1]];

cout << "\nCurrent max element: ";

print(maxA);

if (maxA == 0)

{

cout << "This system has no solutions!" << endl;

break;

}

float\* m = getMultipliersArray(Matrix, size, maxKeys, n);

cout << "\nCurrent vector of multipliers: ";

print(m, size);

gaussStep(Matrix, rightVector, maxKeys, m, numbersX, size, n);

cout << "\nCurrent Matrix:\n\n";

print(Matrix, size);

cout << "Current right vector: ";

print(rightVector, size);

cout << "-----------------------------------------------------------------";

}

}

void gaussAlgorithmGetSolution(float\*\* Matrix, float\* rightVector, int\* numbersX, float\* X, int size)

{

for (int n = size - 1; n >= 0; n--)

{

X[n] = (rightVector[n] - Solution(Matrix, X, n, size)) / Matrix[n][n];

}

for (int i = 0; i < size; i++)

{

float temp = X[i];

X[i] = X[numbersX[i]];

X[numbersX[i]] = temp;

}

}

float Solution(float\*\* Matrix, float\* X, int n, int size)

{

float output = 0;

for (int i = size - 1; i > n; i--)

{

output += Matrix[n][i] \* X[i];

}

return output;

}

float\* MatrixVecMult(float\*\* matrix1, float\* matrix2, int size)

{

float\* outMatrix = new float[size];

for (int i = 0; i < size; i++)

{

outMatrix[i] = 0;

for (int k = 0; k < size; k++)

{

outMatrix[i] += matrix1[i][k] \* matrix2[k];

}

}

return outMatrix;

}

float\* vecDelta(float\*\* Matrix, float\* rightVec, float\* X, int size)

{

float\* Ax = MatrixVecMult(Matrix, X, size);

float\* r = new float[size];

for (int i = 0; i < size; i++)

{

r[i] = rightVec[i] - Ax[i];

}

return r;

}