**Міністерство освіти і науки України**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”**

**Кафедра прикладної математики**

«Постановка задачі

КУРСОВОЇ РОБОТИ»

з дисципліни: «Програмування» 2-й семестр

на тему: «**Факторизація матриць**»

Виконав: Байда О.Є.

Група КМ-01, факультет ФПМ

Керівник: Любашенко Н.Д.

**Київ**

**2021**

ЗМІСТ

[ВСТУП 3](#_Toc73650158)

[1 Постановка задачі 4](#_Toc73650159)

[2 Вибір методу розв’язання задачі 5](#_Toc73650160)

[2.1 Теорія обраного методу 5](#_Toc73650161)

[3 Алгоритм обраного методу 7](#_Toc73650162)

[4 Опис програми 10](#_Toc73650163)

[5 Результати. Контрольні приклади 11](#_Toc73650164)

[ВИСНОВКИ 14](#_Toc73650165)

[Література та посилання 15](#_Toc73650166)

[ДОДАТОК А (Текст програми) 16](#_Toc73650167)

# ВСТУП

**Мета роботи:** засвоєння теоретичних та практичних навичок при проектуванні та написані програм на мові С.

**Об’єкт дослідження:** матриці яка вводиться в програмі поелементно

**Предмет дослідження:** види факторизації заданої матриці розкладами LU, QR та Холецького.

# 1 Постановка задачі

Формат введення: матриця яку требу факторизувати у вигляді масиву масивів. Введення матриці у програмі.

Формат виведення: факторизовані матриці у вигляді масивів які містять масиви. Вивід результатів у програмі.

Задачею буде реалізувати такі види розкладу матриць:

* LU – розклад
* Рангова факторизація
* Розклад Холецького
* QR – розклад

Додаткові завдання до курсової роботи:

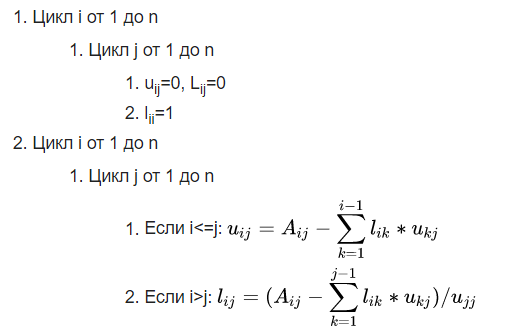
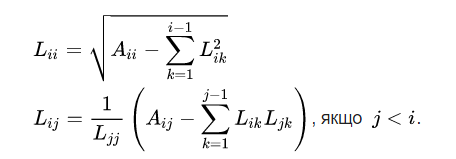
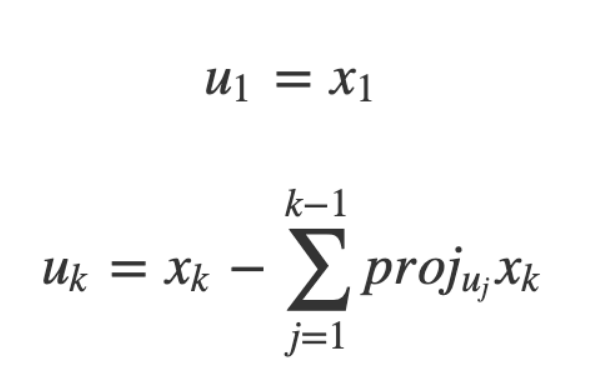
* Динамічна бібліотека на С, яка використовується в іншій програмі (Python)
* Порівняння продуктивності виконаного коду на С із аналогічним на іншій (Python)

# 2 Вибір методу розв’язання задачі

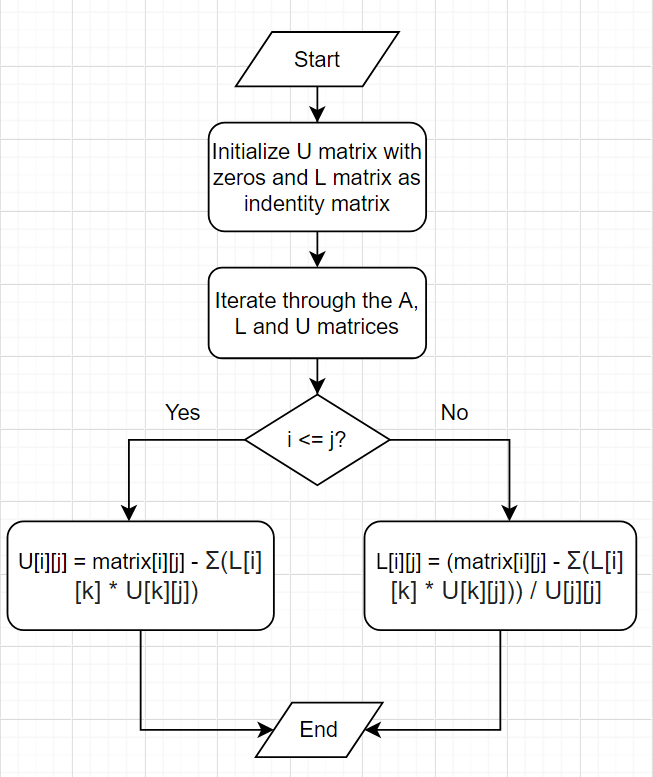
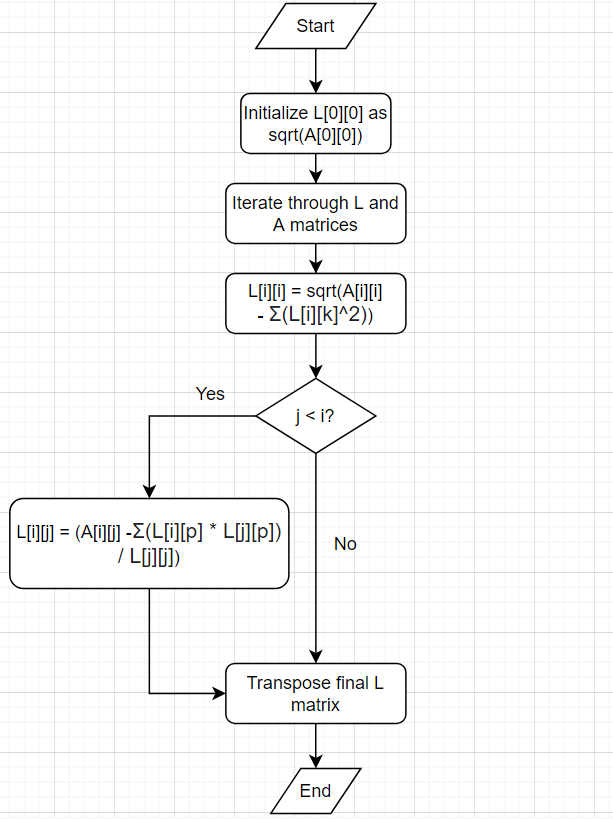
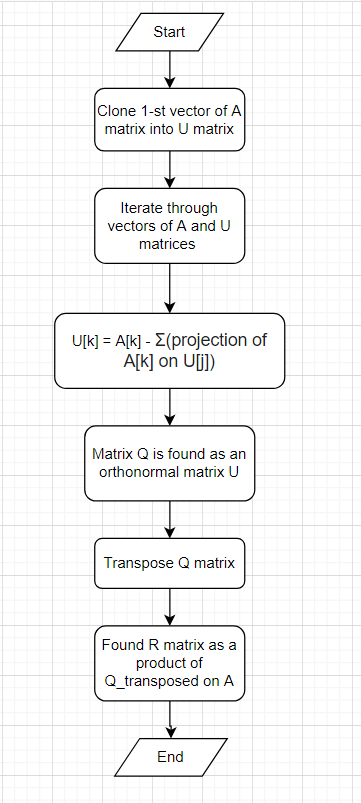
Для розв’язання задачі декомпозиції обрано наступні методи:

1. LU розклад – один з різновидів методу Гаусса.
2. Розклад Холецького – за формулами Холецького
3. QR розклад – для знаходження матриці Q використовується процес Грама-Шмідта, матриці R за формулою Q^T \* A. (Q^T – транспонована матриця Q, A – вихідна матриця.)

# 2.1 Теорія обраного методу

1. LU decomposition: розклад матриці A у вигляд L \* U, де L – нижня трикутна матриця, а U – верхня трикутна матриця. Ці матриці знаходяться за наступним алгоритмом:  
     
   Результати перевіряються перемноженням матриць L та U
2. Kholetskys\_decomposition – представлення симетричної додатноозначеної матриці у вигляді добутку трикутної матриці з додатними елементами на діагоналі на себе транспоновану (L \* L^T). Розклад виконується за формулами:  
     
   
3. QR – decomposition: представлення матриці у вигляді добутку унітарної та правої трикутної матриці. (A = Q \* R). Метод розв’язання базований на методі Грама – Шмідта  
     
   Де u – вектори проміжної матриці U, яка містить ортогональні вектори. x – вектори початкової матриці A. Proj – проекції вектора x на u. Після знаходження матриці U знаходиться матриця Q як ортонормована U. Матриця R знаходиться за формулою R = Q^T \* A.

# 3 Алгоритм обраного методу

1. LU-decomposition  
   
2. Kholetskys\_decomposition  
   
3. QR-decomposition  
   

# 4 Опис програми

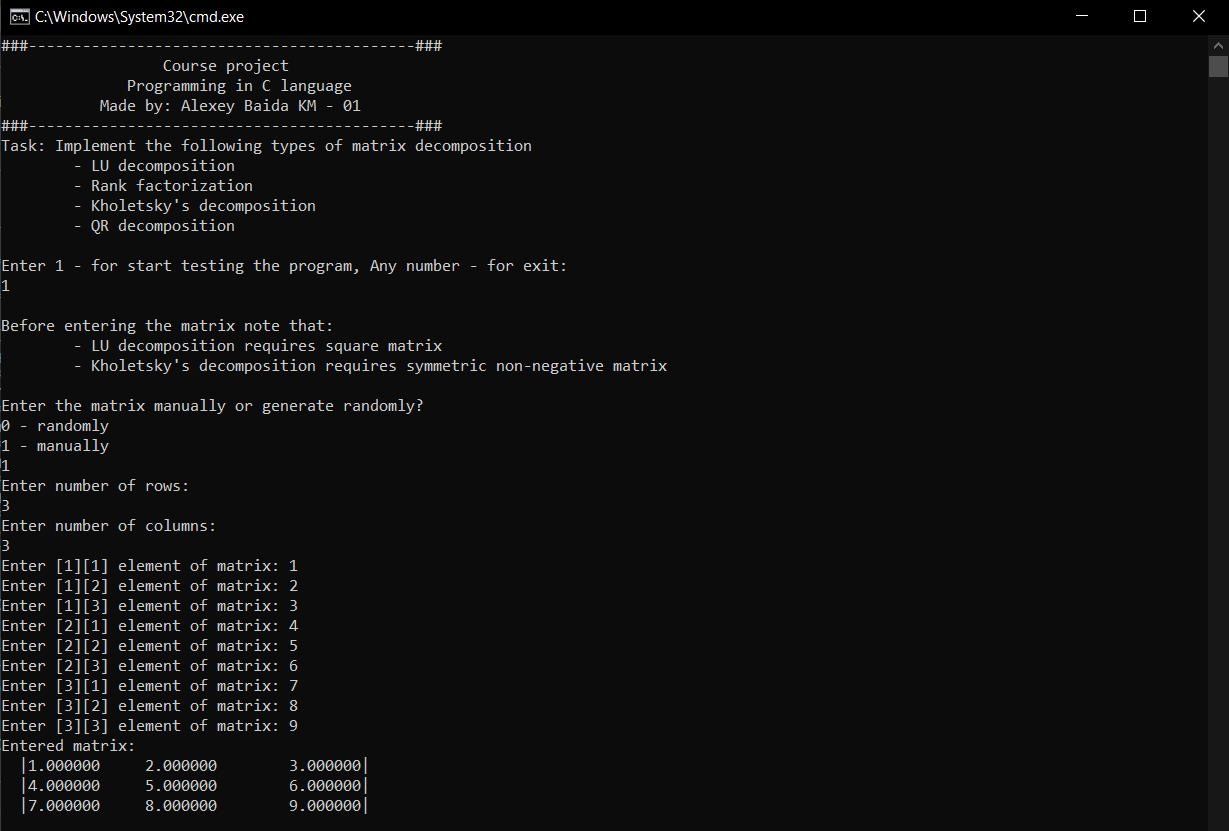
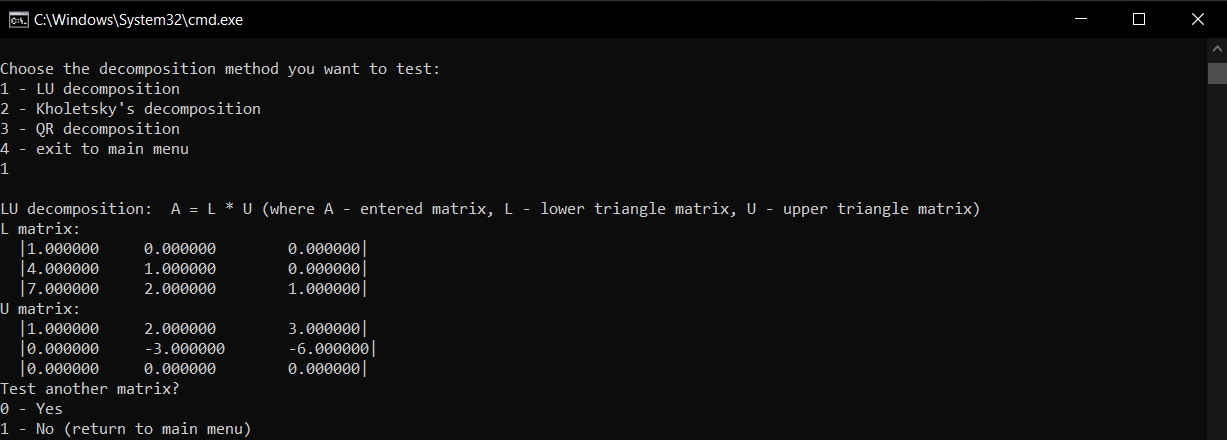
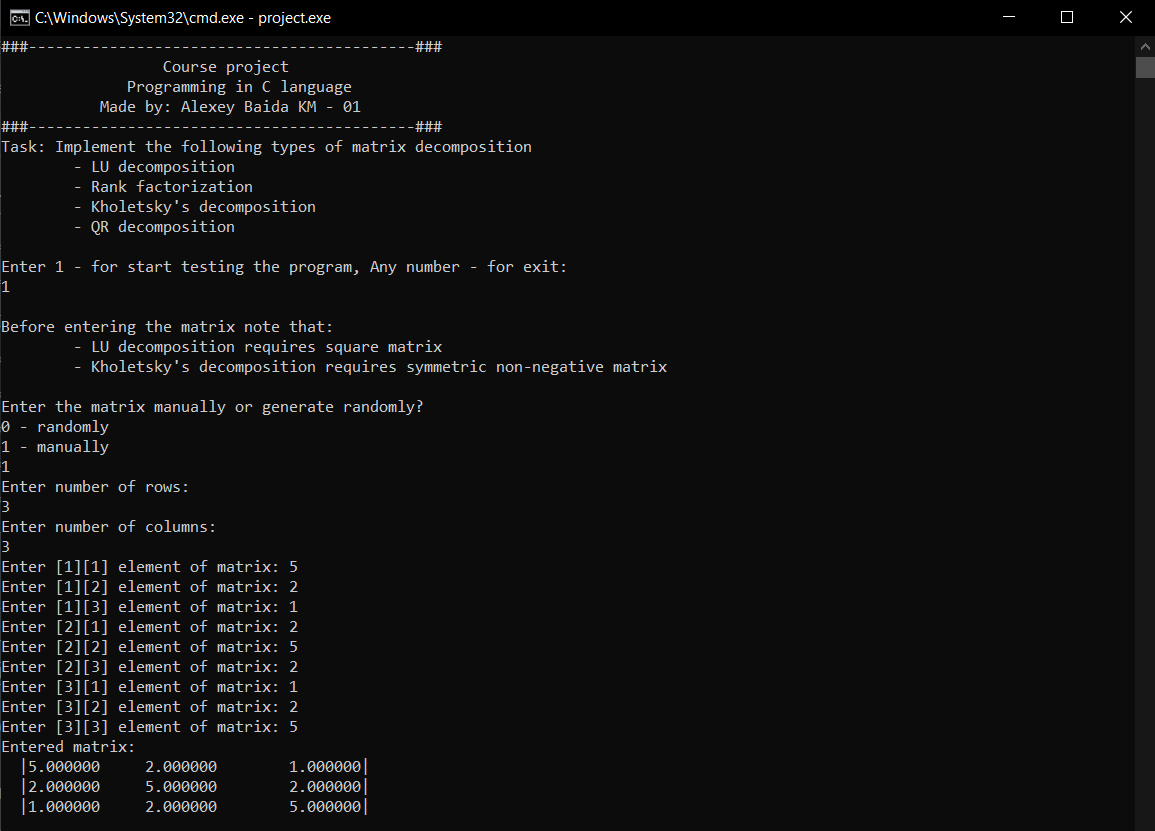
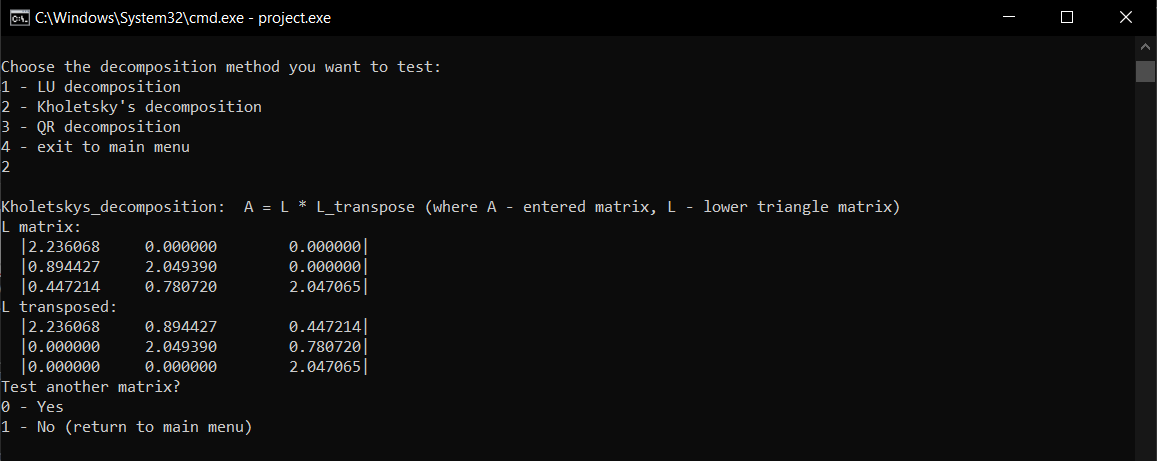
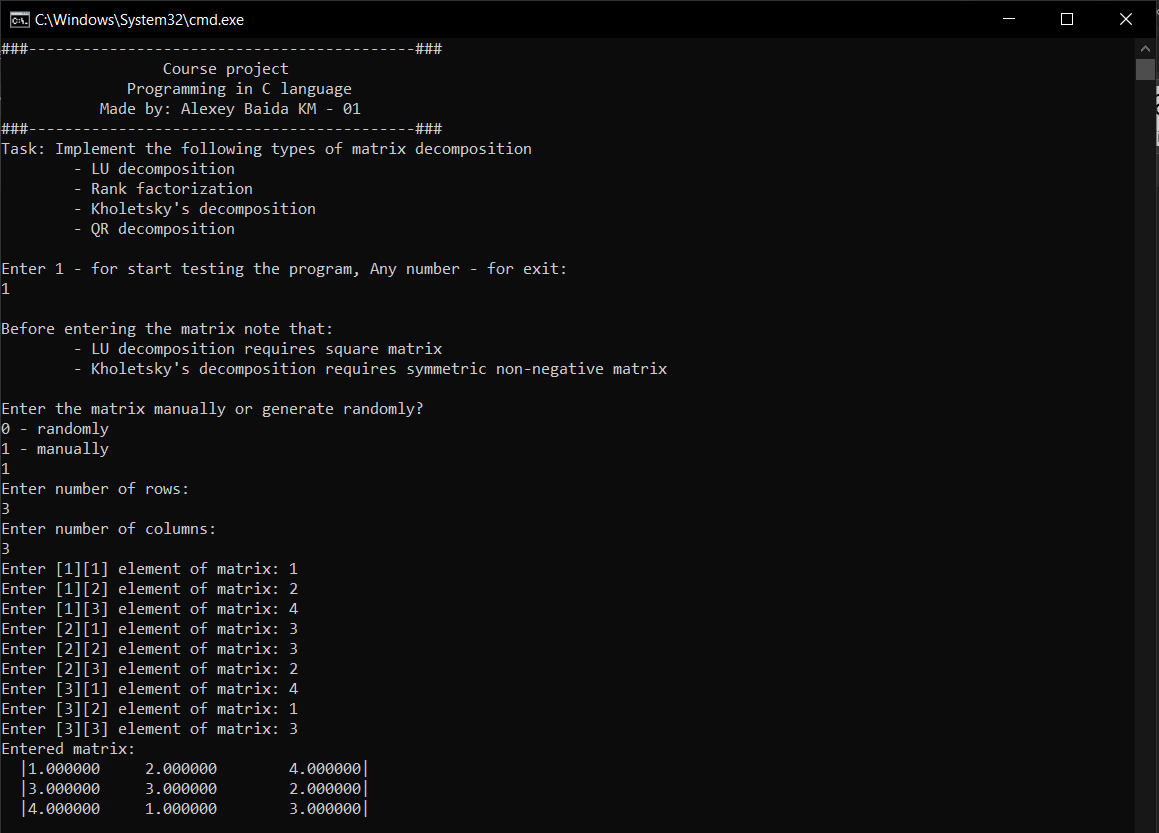
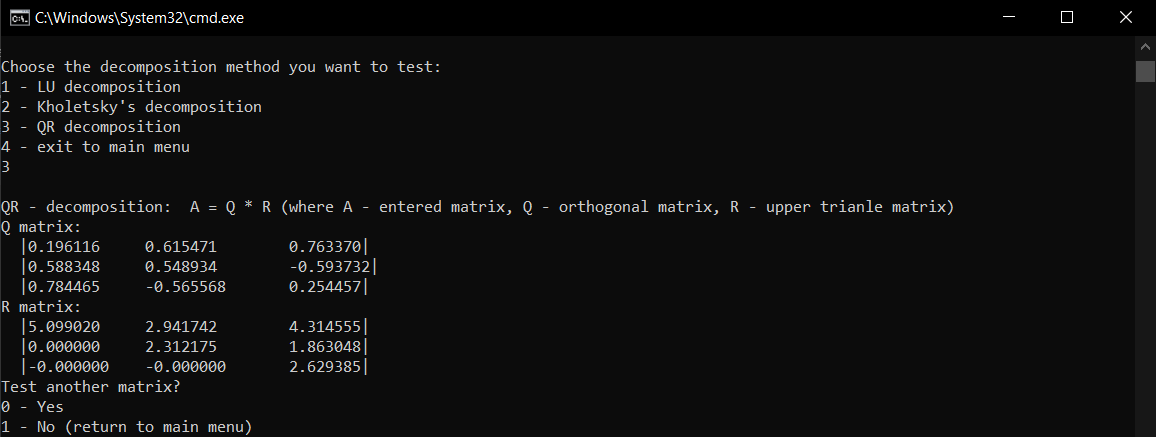
Програму виконано у форматі меню з функціями різних типів факторизацій матриць. На початку програма запитує користувача ввести матрицю яку треба факторизувати. Ввід матриць реалізовано 2-ма методами:

1. Поелементно ввести матрицю розміру n\*k
2. Згенерувати псевдорандомну матрицю розміру n\*k

Після вводу матриці користувачу пропонується вибір з трьох можливих типів декомпозиції. В кожного типу є обмеження для вхідної матриці, це описано перед початком вводу матриці. Кожному типу декомпозиції відповідає число, при натисканні на яке викликається відповідна функція, обрахує матриці розкладу та виведе їх на екран. В кінці програма запитує користувача, протестувати іншу матрицю, або вийти з програми.

# 5 Результати. Контрольні приклади

Скріншоти тестувань різних розкладів матриць:

1. LU - decomposition  
     
     
   Рис. 1.1, 1.2 – тестування LU розкладу
2. Kholetsky’s - decomposition  
     
     
   Рис. 2.1, 2.2 – тестування розкладу Холецького
3. QR – decomposition  
     
     
   Рис. 3.1, 3.2 – тестування QR розкладу

Результати розкладів перевірено перемноженням рокзладених матриць та порівняно з результатами онлайн калькуляторів.

# ВИСНОВКИ

У ході виконання курсової роботи було здобуто навички програмування мовою C з дотриманням style guide CS50. Вивчено математичну складову різних алгоритмів декомпозиції. Здобуто навички роботи з Git. Реалізовано програму з меню здатну обраховувати різні типи факторизацій матриць.

# Література та посилання

1. <https://ru.wikipedia.org/wiki/LU-%D1%80%D0%B0%D0%B7%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5#%D0%92%D1%8B%D0%B2%D0%BE%D0%B4_%D1%84%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D1%8B>
2. <https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D0%BA%D0%BB%D0%B0%D0%B4_%D0%A5%D0%BE%D0%BB%D0%B5%D1%86%D1%8C%D0%BA%D0%BE%D0%B3%D0%BE>
3. <https://uk.wikipedia.org/wiki/QR-%D1%80%D0%BE%D0%B7%D0%BA%D0%BB%D0%B0%D0%B4_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%96>
4. <https://nuancesprog.ru/p/7321/>
5. <https://matworld.ru/matrix/QR-decomposition.php>
6. А. Левитин – “Алгоритмы. Введение в разработку и анализ”

# ДОДАТОК А (Текст програми)

// Program implements different matrix decompositions

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

void header(void)

{

printf("###-------------------------------------------###\n");

printf(" Course project\n", 35);

printf(" Programming in C language\n");

printf(" Made by: Alexey Baida KM - 01\n");

printf("");

printf("###-------------------------------------------###\n");

printf("Task: Implement the following types of matrix decomposition\n");

printf(" - LU decomposition\n");

printf(" - Rank factorization\n");

printf(" - Kholetsky's decomposition\n");

printf(" - QR decomposition\n\n");

}

// Check correctness of entered integer, returns valid integer

int check\_int(void)

{

int num;

char term;

while (1)

{

if(scanf("%d%c", &num, &term) != 2 || term != '\n')

{

printf("Invalud input. Please enter an integer value\n");

fflush (stdin);

}

else

break;

}

return num;

}

// Prints given matrix

void show\_matrix(int nrows, int ncols, float matrix[][ncols])

{

int i, j;

for(i = 0; i < nrows; i++)

{

printf(" |");

for(j = 0; j < ncols; j++)

{

if (j == ncols - 1){

printf("%f", matrix[i][j]);

}

else{

printf("%f\t", matrix[i][j]);

}

}

printf("|\n");

}

}

// Impelements LU decomposition, outputs results

void lu\_decomposition(int nrows, int ncols, float matrix[][ncols])

{

float L[nrows][ncols];

float U[nrows][ncols];

int i, j, k;

printf("\nLU decomposition: A = L \* U (where A - entered matrix, L - lower triangle matrix, U - upper triangle matrix)\n");

for (i = 0; i < nrows; i++)

{

for (j = 0; j < ncols; j++)

{

U[i][j] = 0;

L[i][j] = 0;

L[i][i] = 1;

}

}

float s, s1;

for (i = 0; i < nrows; i++)

{

for (j = 0; j < ncols; j++)

{

if (i <= j)

{

s = 0;

for (k = 0; k <= i - 1; k++)

{

s += L[i][k] \* U[k][j];

}

U[i][j] = matrix[i][j] - s;

}

else if (i > j)

{

s1 = 0;

for (k = 0; k <= j - 1; k++)

{

s1 += L[i][k] \* U[k][j];

}

L[i][j] = (matrix[i][j] - s1)/U[j][j];

}

}

}

// Show Lower matrix

printf("L matrix:\n");

show\_matrix(nrows, ncols, L);

// Show Upper matrix

printf("U matrix:\n");

show\_matrix(nrows, ncols, U);

}

// Impelements Kholetsky's decomposition, outputs results

void Kholetskys\_decomposition(int nrows, int ncols, float matrix[][ncols])

{

int i, j, p;

float L[nrows][ncols];

float L\_transpose[nrows][ncols];

float s, s1;

printf("\nKholetskys\_decomposition: A = L \* L\_transpose (where A - entered matrix, L - lower triangle matrix)\n");

L[0][0] = sqrt(matrix[0][0]);

for (i = 0; i < nrows; i++)

{

for (j = 0; j < ncols; j++)

{

s = 0;

for (p = 0; p <= i - 1; p++)

{

s += L[i][p] \* L[i][p];

}

L[i][i] = sqrt(matrix[i][i] - s);

if (j < i)

{

s1 = 0;

for (p = 0; p <= j - 1; p++)

{

s1 += L[i][p] \* L[j][p];

}

L[i][j] = (matrix[i][j] - s1)/L[j][j];

}

}

}

for (i = 0; i < nrows; i++)

{

for (j = 0; j < ncols; j++)

{

L\_transpose[j][i] = L[i][j];

}

}

printf("L matrix:\n");

show\_matrix(nrows, ncols, L);

printf("L transposed:\n");

show\_matrix(nrows, ncols, L\_transpose);

}

// Impelements QR decomposition, outputs results

void qr\_decomposition(int nrows, int ncols, float matrix[][ncols])

{

int i, j, a, k;

float Q[nrows][ncols];

float Q\_transpose[nrows][ncols];

float R[nrows][ncols];

float U[nrows][ncols];

printf("\nQR - decomposition: A = Q \* R (where A - entered matrix, Q - orthogonal matrix, R - upper trianle matrix)\n");

/\* Calculates U matrix \*/

// Clone 1-st vector of A matrix into U

for (i = 0; i < nrows; i++)

{

U[i][0] = matrix[i][0];

}

for (k = 1; k < ncols; k++)

{

// Initialize sum vector

float summ[nrows];

for (a = 0; a < nrows; a++)

{

summ[a] = 0;

}

// Calculates sum of projections

for (j = 0; j <= k - 1; j++)

{

float sk, sk1, proj[nrows];

//proj uj xk = (xk,uj)/(uj,uj)\*uj

sk = 0;

for (a = 0; a < nrows; a++)

{

sk += matrix[a][k] \* U[a][j];

}

sk1 = 0;

for (a = 0; a < nrows; a++)

{

sk1 += U[a][j] \* U[a][j];

}

for (a = 0; a < nrows; a++)

{

proj[a] = (sk / sk1) \* U[a][j];

}

for (a = 0; a < nrows; a++)

{

summ[a] += proj[a];

}

}

// Performs Ak vector minus sum of projections

for (a = 0; a < nrows; a++)

{

U[a][k] = matrix[a][k] - summ[a];

}

}

/\* End of calculation U matrix \*/

/\* Calculates Q matrix (based on U matrix) \*/

for (k = 0; k < ncols; k++)

{

float sum\_of\_squares, norm;

sum\_of\_squares = 0;

for (a = 0; a < nrows; a++)

{

sum\_of\_squares += U[a][k] \* U[a][k];

}

norm = sqrt(sum\_of\_squares);

for (a = 0; a < nrows; a++)

{

Q[a][k] = U[a][k] / norm;

}

}

/\* Calculates Q\_transposed\*/

for (i = 0; i < nrows; i++)

{

for (j = 0; j < ncols; j++)

{

Q\_transpose[j][i] = Q[i][j];

}

}

/\* Calculates R matrix as product of Q\_transposed and A(entered matrix) \*/

for (i = 0; i < nrows; i++)

{

for (j = 0; j < ncols; j++)

{

R[i][j] = 0;

for (k = 0; k < nrows; k++)

{

R[i][j] += Q\_transpose[i][k] \* matrix[k][j];

}

}

}

printf("Q matrix:\n");

show\_matrix(nrows, ncols, Q);

printf("R matrix:\n");

show\_matrix(nrows, ncols, R);

}

int main(void)

{

int answer;

answer = 5; // any non zero value :)

system("cls");

header(); // print header

while(1)

{

printf("Enter 1 - for start testing the program, Any number - for exit:\n");

answer = check\_int();

if (answer == 1)

{

while (1)

{

printf("\nBefore entering the matrix note that:\n");

printf(" - LU decomposition requires square matrix\n");

printf(" - Kholetsky's decomposition requires symmetric non-negative matrix\n\n");

printf("Enter the matrix manually or generate randomly?\n0 - randomly\n1 - manually\n");

answer = check\_int();

int nrows, ncols, i, j;

printf("Enter number of rows:\n");

nrows = check\_int();

printf("Enter number of columns:\n");

ncols = check\_int();

float matrix[nrows][ncols];

// Generate randomly

if (answer == 0)

{

for(i = 0; i < nrows; i++)

{

for(j = 0; j < ncols; j++)

{

//Generates pseudorandom float between 0.1 and 9.9

matrix[i][j] = (float) rand() / RAND\_MAX \* (9.9 - 0.1) + 0.1;

}

}

}

// Enter manually

else if (answer == 1)

{

for(i = 0; i < nrows; i++)

{

for(j = 0; j < ncols; j++)

{

printf("Enter [%d][%d] element of matrix: ", i + 1, j + 1);

scanf("%f", &matrix[i][j]);

}

}

}

// Show entered matrix

printf("Entered matrix:\n");

show\_matrix(nrows, ncols, matrix);

printf("\nChoose the decomposition method you want to test:\n1 - LU decomposition\n");

printf("2 - Kholetsky's decomposition\n3 - QR decomposition\n4 - exit to main menu\n");

answer = check\_int();

if (answer == 1)

{

lu\_decomposition(nrows, ncols, matrix);

}

else if (answer == 2)

{

Kholetskys\_decomposition(nrows, ncols, matrix);

}

else if (answer == 3)

{

qr\_decomposition(nrows, ncols, matrix);

}

else if (answer == 4)

{

break;

}

printf("Test another matrix?\n0 - Yes\n1 - No (return to main menu)\n");

answer = check\_int();

if (answer == 1)

{

break;

}

}

}

else

{

break;

}

}

printf("---Thanks for using!---\n");

}