**Assignment: Solving a System of Linear Equations**

**Using LU decomposition**

# What you need to submit

* Submit the report+source files as a zip file: **Assignment\_LUdecomp\_Name\_ID.zip**
* **Report:** with pseudocode, output result and source code as instructed
* **Src Code Files:**
  + - Assignment\_ludecomp\_ID.cpp
    - myNP.h, myNP.cpp, myMatrix.h, myMatrix.cpp

**Tip:**

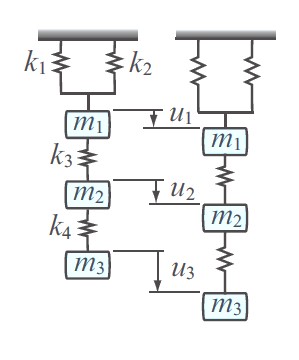
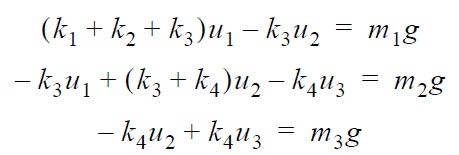
* Review how to use structure type for using 2D array
  + [Tutorial & Exercise](https://ykkim.gitbook.io/ec/c-programming/c-programming-review/dynamic-alloc)
  + [Tutorial Video](https://youtu.be/wJcecYauQvk)
* Error handling: check if user given matrix A is nxn square matrix
* Error handling: check for div by zero

# Problem: Solve the following linear systems of Ax=b

**Q1. Determine the displacement of the three masses**

They are in the equilibrium states, and *u1,u2, u3* are the relative displacement for each mass.

m1=2 kg, m2=3kg, m3=1.5kg , g=9.81 m/s^2 k1= 30 [N/m], k2=25 [N/m], k3=20 [N/m], k4=15 [N/m]



# Procedure

**1. LU decomposition without partial pivoting[20pt]. If scaled partial pivoting is applied [extra +20 pt]**

## a) Write a pseudocode for LU decomposition [5pt]

// pseudocode goes here

텍스트이(가) 표시된 사진

자동 생성된 설명

## b) Create a C function that processes the LU decomposition [15pt]

* Input: matrix **A**(n x n), vector **b**(n x1)
* Output: matrix **U**, matrix L (option)permutation matrix **P**

void LUdecomp (Matrix A, Matrix L, Matrix U); // or void LUdecomp (Matrix A, Matrix L, Matrix U, Matrix P); // for pivoting

Show you code here

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

**2. Create a function that solves for Ax= LUx=b [20pt] If permulation P is applied [ extra 20pt]**

## a) Write a pseudocode to solve Ax=b from LU decomposition [5pt]

텍스트이(가) 표시된 사진

자동 생성된 설명

## b) Create a C function [15pt]

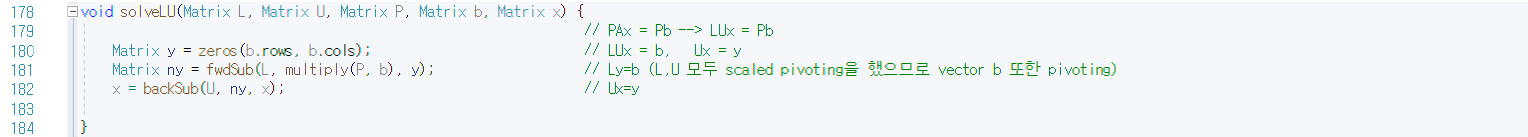
// LUdecomp(A,L,U,P);

void solveLU (Matrix L, Matrix U, Matrix b, Matrix x); // or

void solveLU (Matrix L, Matrix U, Matrix P, Matrix b, Matrix x); // for Pivoting

void fwdsub (Matrix L, Matrix b, Matrix y);

void backsub (Matrix U, Matrix y, Matrix x);



## 3. Create a function that finds the inverse of A. [20pt]

double invMat(Matrix A, Matrix Ainv);

* MUST check if A is square (nxn) and it is full rank. rank(A)=n
* Hint: check if there is any zero in diagonal terms of matrix **U**
* Confirm your answer by calculating **x=Ainv\*b**

double invMat(Matrix A, Matrix Ainv) {

if (A.rows != A.cols) {

printf("ERROR: IT IS NOT A N\*N MATRIX");

}

for (int i = 0; i < A.cols; i++) {

if (A.at[i][i] == 0) {

printf("ERROR: IT IS NOT A FULL RANK MATRIX");

return 0;

}

}

Matrix L = zeros(A.rows, A.cols);

Matrix U = zeros(A.rows, A.cols);

Matrix P = eye(A.rows, A.cols);

Matrix I = eye(A.rows, A.cols);

Matrix x = zeros(A.rows, 1);

Matrix AI = zeros(A.rows, 1);

Matrix Uc = zeros(A.rows, A.cols);

Matrix Lc = zeros(A.rows, A.cols);

Matrix result = zeros(A.rows, A.cols);

LUdecomp(A, L, U, P);

for (int j = 0; j < A.cols; j++) {

for (int i = 0; i < A.cols; i++) {

AI.at[i][0] = I.at[i][j];

} // I를 열 벡터로 만들기

x = backSub(U, AI, x); // U \* x = AI 를 back substitution으로 x 벡터 구하기

for (int i = 0; i < A.cols; i++) {

Uc.at[i][j] = x.at[i][0];

}initMat(x, 0); // 각 x벡터들을 Uc에 할당

}

for (int j = 0; j < A.cols; j++) {

for (int i = 0; i < A.cols; i++) {

AI.at[i][0] = I.at[i][j];

} // I를 열 벡터로 만들기

x = fwdSub(L, AI, x); // L \* x = AI 를 back substitution으로 x 벡터 구하기

for (int i = 0; i < A.cols; i++) {

Lc.at[i][j] = x.at[i][0];

}initMat(x, 0); // 각 x벡터들을 Lc에 할당

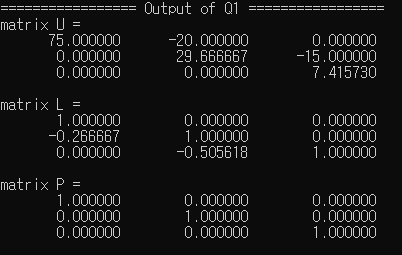
}

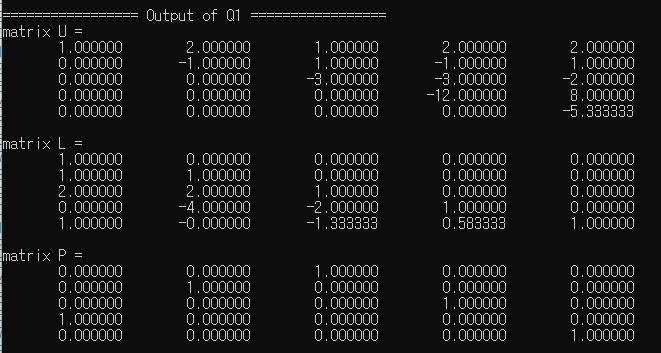
copyVal(multiply(Uc, Lc), Ainv); // copyVal을 통해 Ainv에 저장, Uc \* Lc

return 0;

}

**4. Capture the output results to solve Q1 [5pt]**





< Checking Pivoting >

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

## 5. Check your answer with the output from MATLAB [5pt]

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**

*\*\* For LU with pivoting, check your process with the values shown in Appendix. TA will check your library with test matrices that need pivoting.*

# Appendix

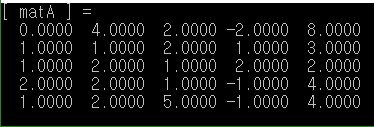
1. When do you use the permutation matrix **P**? For LU, y=fwdsub(**L**, **P**\***b**) → x=backsub(**U**,**y**)

1. How to update the permutation matrix **P** during the elimination process?

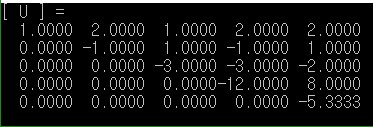
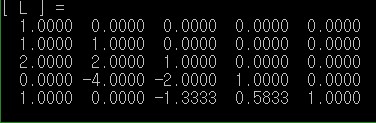
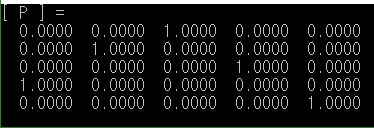
One method could be using index 1-D array Pidx=[1,2,3,4] → Pidx=[3,2,1,4] etc.

1. Matrix elements for debugging the process of finding P, L, U using LU decomposition.

Example) Matrix A :



Final Output of P, L, U Matrix :



Output of P, L\_orig, U at each iteration number :

