Assignment: Solving a System of Linear Equations Using LU decomposition

Instructions:

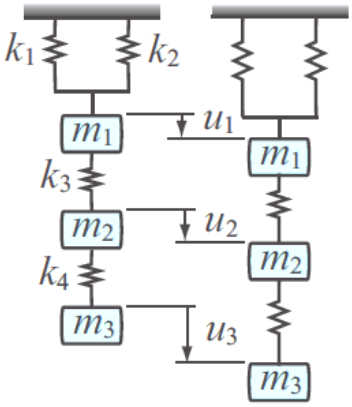
- Declare and define the numerical method functions in your header file "myNM.h", "myNM.cpp", "myMatrix.h", "myMatrix.cpp", respectively.
- The main program file, 'Assignmen_gaussElim.cpp', contains the main function and calls yours NM functions to solve the assignment problems.
- You must submit the report and the program files: "myNM.h", "myNM.cpp", "myMatrix.h", "myMatrix.cpp", "Assignmen_LU.cpp" on Hisnet.
- You can use dynamic memory allocation for 2D array and structures for matrix variables. It is not mandatory.
- Check if your function is accepting only square matrices
- You should insert exceptional/error handling(e.g. giving error message when square matrix is not used, div by zero etc)
- For this assignment, you do not need to use partial pivoting.

Problem: Solve the following linear systems of Ax=b

Q1. Determine the displacement of the three masses

They are in the equilibrium states, and u_1,u_2, u_3 are the relative displacement for each mass.

m_1=2 kg, m_2=3kg, m_3=1.5kg , g=9.81 m/s^2
k_1= 30 [N/m], k_2=25 [N/m], k_3=20 [N/m], k_4=15 [N/m]



$$(k_1 + k_2 + k_3)u_1 - k_3u_2 = m_1g$$
$$-k_3u_1 + (k_3 + k_4)u_2 - k_4u_3 = m_2g$$
$$-k_4u_2 + k_4u_3 = m_3g$$

Procedure

- Review how to define, initialize and use 2D arrays and how to pass 2D array to a function in C/C++. (See Tutorial #2)
- If you want, you can apply partial pivoting in the program.
- Add exceptional/error handling for when **A** is not square, dimension of **A**, **b** are not appropriate, division by zero and so on
- *We will only consider square matrix A (n by n) for this assignment.

1. LU decomposition without partial pivoting[20pt]. If scaled partial pivoting is applied [40 pt]

Create a C/C++ function that processes the LU decomposition

- Input: matrix **A**(n x n), vector **b**(n x1)
- Output: matrix **U**, matrix L (option)permutation matrix **P**
- Declare in “myNM.h” and define in “myNM.c”
- See appendix for help

```
void LUdecomp (Matrix A, Matrix L, Matrix U, Matrix P);
```

First write a pseudocode

Show you code here

---

2. Create a function that solves for Ax= LUx=b [20pt]. If permulation P is applied [40pt]

```
double solveLU (Matrix L, Matrix U, Matrix P, Matrix b);
or  void solveLU (Matrix L, Matrix U, Matrix P, Matrix b, Matrix x);
```

- Input: matrix **L, U, P** from LUdecomp(A,L,U,P) and vector **b**(n x1)
- Output: vector **x** (nx1)
- Declare in "myNM.h" and define in "myNM.c"

```
LUdecomp(A,L,U,P)

double solveLU(L,U,P,b)      // or     double solveLU(L,U,b)
{
        fwdsub(….)                    // what should be the input arguments to include P?
        backsub(….)          // what should be the input arguments to include P?. . .
        return vectorX
}
```

First write a pseudocode

Then, create a C function

---

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

```
double inv(Matrix A, Matrix Ainv);
```

- You can use LU decomposition, Gauss-Jordan elimination etc..
- You must check (1)A is square (nxn) and (2) it is full rank. rank(A)=n
- Check your answer by **x=Ainv*b**

4. Show the output results

5. Check your answer with the output from MATLAB

***** 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**)

2. 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.

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

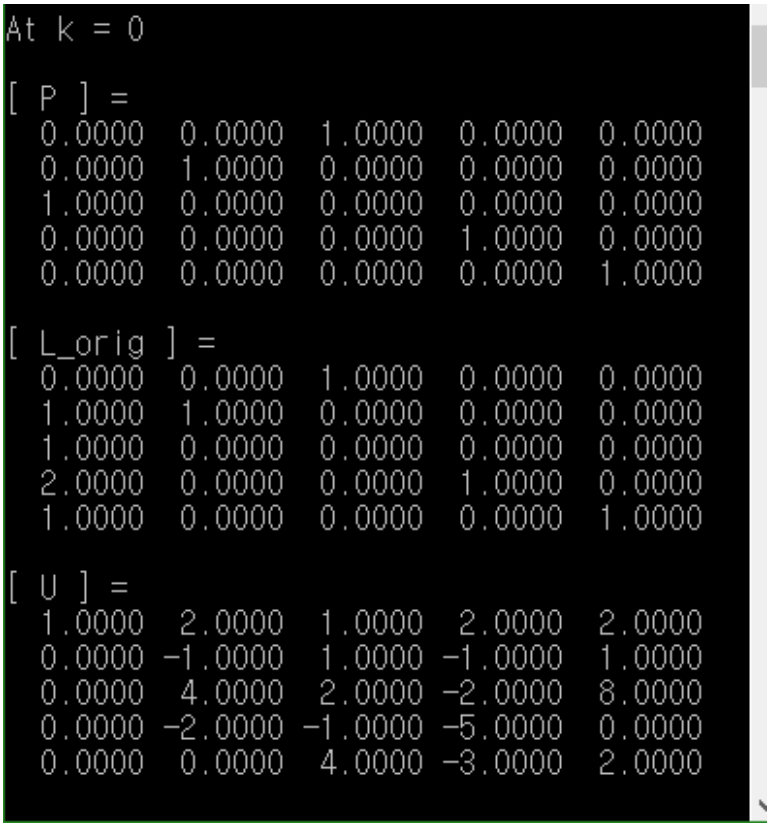
   Example) Matrix A :

```
[ matA ] =
  0.0000   4.0000   2.0000  -2.0000   8.0000
  1.0000   1.0000   2.0000   1.0000   3.0000
  1.0000   2.0000   1.0000   2.0000   2.0000
  2.0000   2.0000   1.0000  -1.0000   4.0000
  1.0000   2.0000   5.0000  -1.0000   4.0000
```

   Final Output of P, L, U Matrix :

```
[ P ] =
  0.0000   0.0000   1.0000   0.0000   0.0000
  0.0000   1.0000   0.0000   0.0000   0.0000
  0.0000   0.0000   0.0000   1.0000   0.0000
  1.0000   0.0000   0.0000   0.0000   0.0000
  0.0000   0.0000   0.0000   0.0000   1.0000
[ L ] =
  1.0000   0.0000   0.0000   0.0000   0.0000
  1.0000   1.0000   0.0000   0.0000   0.0000
  2.0000   2.0000   1.0000   0.0000   0.0000
  0.0000  -4.0000  -2.0000   1.0000   0.0000
  1.0000   0.0000  -1.3333   0.5833   1.0000
[ U ] =
  1.0000   2.0000   1.0000   2.0000   2.0000
  0.0000  -1.0000   1.0000  -1.0000   1.0000
  0.0000   0.0000  -3.0000  -3.0000  -2.0000
  0.0000   0.0000   0.0000 -12.0000   8.0000
  0.0000   0.0000   0.0000   0.0000  -5.3333
```

Output of P, L_orig, U at each iteration number :



```
At k = 0

[ P ] =
  0.0000  0.0000  1.0000  0.0000  0.0000
  0.0000  1.0000  0.0000  0.0000  0.0000
  1.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  1.0000  0.0000
  0.0000  0.0000  0.0000  0.0000  1.0000

[ L_orig ] =
  0.0000  0.0000  1.0000  0.0000  0.0000
  1.0000  1.0000  0.0000  0.0000  0.0000
  1.0000  0.0000  0.0000  0.0000  0.0000
  2.0000  0.0000  0.0000  1.0000  0.0000
  1.0000  0.0000  0.0000  0.0000  1.0000

[ U ] =
  1.0000  2.0000  1.0000  2.0000  2.0000
  0.0000 -1.0000  1.0000 -1.0000  1.0000
  0.0000  4.0000  2.0000 -2.0000  8.0000
  0.0000 -2.0000 -1.0000 -5.0000  0.0000
  0.0000  0.0000  4.0000 -3.0000  2.0000
```

```
At k = 1

[ P ] =
  0.0000  0.0000  1.0000  0.0000  0.0000
  0.0000  1.0000  0.0000  0.0000  0.0000
  1.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  1.0000  0.0000
  0.0000  0.0000  0.0000  0.0000  1.0000

[ L_orig ] =
  0.0000 -4.0000  1.0000  0.0000  0.0000
  1.0000  1.0000  0.0000  0.0000  0.0000
  1.0000  0.0000  0.0000  0.0000  0.0000
  2.0000  2.0000  0.0000  1.0000  0.0000
  1.0000  0.0000  0.0000  0.0000  1.0000

[ U ] =
  1.0000  2.0000  1.0000  2.0000  2.0000
  0.0000 -1.0000  1.0000 -1.0000  1.0000
  0.0000  0.0000  6.0000 -6.0000 12.0000
  0.0000  0.0000 -3.0000 -3.0000 -2.0000
  0.0000  0.0000  4.0000 -3.0000  2.0000
```

```
At k = 2

[ P ] =
  0.0000  0.0000  1.0000  0.0000  0.0000
  0.0000  1.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  1.0000  0.0000
  1.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  0.0000  1.0000

[ L_orig ] =
  0.0000 -4.0000 -2.0000  1.0000  0.0000
  1.0000  1.0000  0.0000  0.0000  0.0000
  1.0000  0.0000  0.0000  0.0000  0.0000
  2.0000  2.0000  1.0000  0.0000  0.0000
  1.0000  0.0000 -1.3333  0.0000  1.0000

[ U ] =
  1.0000  2.0000  1.0000  2.0000  2.0000
  0.0000 -1.0000  1.0000 -1.0000  1.0000
  0.0000  0.0000 -3.0000 -3.0000 -2.0000
  0.0000  0.0000  0.0000-12.0000  8.0000
  0.0000  0.0000  0.0000 -7.0000 -0.6667
```

```
At k = 3

[ P ] =
  0.0000  0.0000  1.0000  0.0000  0.0000
  0.0000  1.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  1.0000  0.0000
  1.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  0.0000  1.0000

[ L_orig ] =
  0.0000 -4.0000 -2.0000  1.0000  0.0000
  1.0000  1.0000  0.0000  0.0000  0.0000
  1.0000  0.0000  0.0000  0.0000  0.0000
  2.0000  2.0000  1.0000  0.0000  0.0000
  1.0000  0.0000 -1.3333  0.5833  1.0000

[ U ] =
  1.0000  2.0000  1.0000  2.0000  2.0000
  0.0000 -1.0000  1.0000 -1.0000  1.0000
  0.0000  0.0000 -3.0000 -3.0000 -2.0000
  0.0000  0.0000  0.0000-12.0000  8.0000
  0.0000  0.0000  0.0000  0.0000 -5.3333
```