CS 330 Project 2: LU Factorization Engine

1 Introduction

For this project, you will be creating a C99 library for solving linear equations of the form $\mathbf{Ax} = \mathbf{b}$ using *LU factorization* with partial pivoting, as discussed in class. The goal is to produce a largely self-contained library that other software can use to solve problems, so you don't need to worry about reading input.

I have provided some skeleton files in Canvas. These include a header file for your library (you may change the structure as noted in Section 2), a start at the library itself, and a file to test your library. What to submit is described in Section 4.

2 LU factorization module

You will create a reusable LU factorization module in C99 with the following interface:

```
typedef struct { ... } LUfact;
LUfact *LUfactor(int N, const double **A);
void LUdestroy(LUfact*);
void LUsolve(LUfact *fact, const double *b, double *x);
```

`LUfact` : an opaque data structure that holds the necessary LU factorization information.

`LUfactor()` : this routine performs the factorization (see course notes for details) and allocates, fills, and returns a `LUfact` object. If the matrix A is singular return `NULL`. Note that the matrix A is stored as a vector of row-pointers.

`LUdestroy()` : deallocates the data allocated in `LUfactor()`.

`LUsolve()` : solves the system $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{x}$.

2.1 LU Factorization Data Structure

The `LUfactor()` function allocates and fills a `LUfact` object containing factorization information. You may implement this structure however you like (it is meant to be opaque, after all), though the following form is probably a good starting point:

```
typedef struct {
    int N;         /* Size of input NxN matrix A */
    double **LU;   /* NxN matrix holding combined L and U matrix */
    short *mutate; /* Row permutations of A */
} LUfact;
```

Note that partial pivoting means you have to keep track of how rows get swapped. Here, the mutate array is intended to hold a mapping from the row's original location to the new one.

3 A simple test case

- An example 4×4 matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 1 & 2 \\ -1 & 2 & 1 & 2 \\ 3 & 1 & 4 & 1 \\ 3 & 3 & -3 & -3 \end{bmatrix}$$

- The corresponding decomposition (without partial pivoting) is

$$\mathbf{LU} \approx \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 \\ -1.000 & 1.000 & 0.000 & 0.000 \\ 3.000 & -1.600 & 1.000 & 0.000 \\ 3.000 & -1.200 & -0.857 & 1.000 \end{bmatrix} \begin{bmatrix} 1.000 & 3.000 & 1.000 & 2.000 \\ 0.000 & 5.000 & 2.000 & 4.000 \\ 0.000 & 0.000 & 4.200 & 1.400 \\ 0.000 & 0.000 & 0.000 & -3.000 \end{bmatrix}$$

- The following decomposition results from partial pivoting and corresponds to the rows $\{2, 0, 3, 1\}$ of $\mathbf{A}$:

$$\mathbf{LU} \approx \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 \\ 0.333 & 1.000 & 0.000 & 0.000 \\ 1.000 & 0.750 & 1.000 & 0.000 \\ -0.333 & 0.875 & -0.389 & 1.000 \end{bmatrix} \begin{bmatrix} 3.000 & 1.000 & 4.000 & 1.000 \\ 0.000 & 2.667 & -0.333 & 1.667 \\ 0.000 & 0.000 & -6.750 & -5.250 \\ 0.000 & 0.000 & 0.000 & -1.167 \end{bmatrix}$$

  Please note that you must implement partial pivoting in your library. The solution without pivoting, above, is presented only for your reference in case you want to implement things without pivoting first.

4 What to submit

Submit an archive file including LUfact.c and LUfact.h, making sure you code is neatly formatted, well-commented, and identifies you, the course, and the project. You do not need to include LUtest.c If your library includes source files beyond the pair named above, include instructions for building your library.