Homework 6 (25 points)

LU decomposition

March 3, 2026

1 HW6 contents

This file comes zipped with three files:

1. A simple object matrix library matrix.cpp (optimized for simplicity, not quickness).
2. Header file matrix.h for the matrix library (contains mostly declarations of the class and functions).
3. Demo program using the library mainMatrix.h.

Unzip all three into the same directory. Create a new project in your IDE, add the $*$.h and both $*$.cpp files to the project. Rebuild and run.

2 General information

The matrix class from HW3 has been updated as follows:

1. The element type is aliased T, which is defined to be **float** at the beginning of the header matrix.h. It can be easily changed to **double** or **long double** by one the following modifications of the **typedef** definition:

   ```
   typedef double T;
   typedef long double T;
   ```

   If you change the type, do not forget to rebuild the project before running it again. For the tasks below please use the type **float**.

2. The class implements LU decomposition with rescaling and row pivoting. The matrices **L**, **U** and the rescaling and permutation data are pointed to internally by the following pointers:

   - vector<vector<T> > $*$luMatrix;
     which is a pointer to the decomposed **LU** matrix. Both **L** and **U** matrices are stored in this single matrix: **U** above and in the diagonal, while **L** below it (as the matrix **L** has only 1s on the diagonal). See the functions getLMatrix() and getUMatrix().

- vector<T> * sclVector;
  which is a pointer to the scaling vector. The matrix rows are scaled with respect to the sums of the absolute values of the row elements. That is, for example, the scaling vector $\{0.1, 0.1, 0.1\}$ means that all row-sums were 10.
- vector<**int**> * prmVector;
  which is a pointer to the permutation vector. For example, the vector $\{1, 2, 0\}$ means that during the decomposition 0th row went to the 2nd row, 1st row went to the 0th row and 2nd row went to the 1st row: $\{\text{row}_0, \text{row}_1, \text{row}_2\} \longrightarrow \{\text{row}_1, \text{row}_2, \text{row}_0\}$.

The LU data are initialized, deleted (when the original matrix is modified) or copied from another matrix (in the copy constructor and the assignment operator) by the following private functions:

```
void initialiseLUMatrices(void);
void deleteLUMatrices(void);
void copyLUMatrices(const matrix & m);
```

LU decomposition is run automatically whenever necessary (see for example matrix.cpp, lines 162, 173, etc.) by calling the private function luDecompose(**void**), which makes use of

- **void** luRescale(**void**);
  to rescale and copy the original matrix to the *luMatrix and to save the scaling information in *sclVector;
- **void** luSwapRows(**int** r1, **int** r2);
  to swap the rows no. r1 and r2 of *luMatrix and to store the pivoting information in *prmVector.

The six following public functions

```
int getLUPermutationVector(int r);
T getLUScalingVector(int r);
matrix getLMatrix(void);
matrix getUMatrix(void);
matrix getLUPermutationVector(void);
matrix getLUScalingVector(void);
```

are used to get the **LU** data. If the matrix is not yet decomposed, they call the luDecompose() function.

3. The unary minus operator is defined,

   ```
   matrix operator- (const matrix & m);
   ```

   to make possible verification of the results without depriving you of your fun with HW3, where you are expected to write binary **operator**−() and **operator**−=().

4. Two new member function are declared (in matrix.h) and partially defined (in matrix.cpp):

- matrix solveLS(**const** matrix& b);
  which is meant to solve (if posible) a linear system $\mathbf{Ax} = \mathbf{b}$ and to return the computed solution vector $\mathbf{x}$;
- matrix det(**void**);
  which is meant to compute (if possible) the determinant of the matrix.

The demo program (mainMatrix.cpp) generates a random 4×4 matrix $\mathbf{A}$, decomposes it into $\mathbf{L}$ and $\mathbf{U}$, prints the results and verifies them by back permutating and rescaling $\mathbf{LU}$ to compare it with the original matrix $\mathbf{A}$.

3 Your tasks

1. (5 points) Fill the gaps in the det() function (matrix.cpp file, line 266). Notice that to compute the determinant it is enough to multiply the diagonal elements of the $\mathbf{U}$ matrix (since the $\mathbf{L}$ matrix has 1s on the diagonal) and to divide them by the row scaling factors (stored in *sclVector). Take into account the permutation parity (stored in the variable permutationParity, which is either -1 or 1). Use the hints in the comments.

2. (12 points) Fill the gaps in the solveLS() function (matrix.cpp file, line 237). Use the hints in the comments. Modify the main() function (mainMatrix.cpp) to demonstrate that your function really works (for example uncomment the lines 32–41 in the main() function).

3. (3 points) Define (in the mainMatrix.cpp file) the following matrices:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \end{bmatrix}, \qquad \mathbf{b}_1 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \qquad \mathbf{b}_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

   Clearly, $\mathbf{A}$ is a singular matrix. Compute its determinant and use your function solveLS() to solve two linear systems $\mathbf{Ax}_1 = \mathbf{b}_1$ and $\mathbf{Ax}_2 = \mathbf{b}_2$. Check the residua $\mathbf{r}_1 = \mathbf{b}_1 - \mathbf{Ax}_1$ and $\mathbf{r}_2 = \mathbf{b}_2 - \mathbf{Ax}_2$. What are the results and why? Redo the task using the type **double** instead of **float** (line 10 of matrix.h, do not forget to rebuild the project). Are there any differences? What are the exact analytical solutions?

4. (5 points) Modify the solveLS() function so that it accepts not only vectors (that is, the case of B.getColNo()==1), but full matrices as arguments. The return value should hence be such a matrix $\mathbf{X}$ (with B.getColNo() columns), that $\mathbf{AX} = \mathbf{B}$.