Program 2 Foundations of Computational Math 2 Spring 2019

Due date: 11:59PM on Monday 18 February 2019

General Task

Consider the case where $A \in \mathbb{R}^{n \times n}$ is nonsingular and the system $Ax = b$ is to be solved via LU factorization without pivoting, using partial pivoting or using complete pivoting. Implement a code that is capable of performing these three tasks, based on a user selection, and one or more test routines designed to evaluate and validate the correctness of the code. Your code should also detect situations where the factorization may not proceed and exit gracefully. Clearly, this is the case when the candidate pivot set contains no acceptable value. In exact arithmetic this means they are all 0. Your code should also allow the detection of a set of candidate pivots that are all "too small" and warn the user. You may code in any compiled and typed language you wish although C, C++, Julia, and Fortran are preferred. In all cases, however, you may not use standard libraries such as LAPACK or built-in matrix routines for pieces of your routines with the exception of the test routines.

Code Structure

Your factorization routine, which must be a separate routine from the driver/tester routine, should accept the matrix A stored in a simple 2-dimensional array-like data structure and other relevant parameters such as n and a flag indicating what form of the factorization should be attempted, e.g., no pivoting, partial pivoting, or complete pivoting. The routine should return the matrices L and U stored within the array that contained A on input, i.e., you should implement the in-place algorithm strategy described in the class notes and lectures. You should also keep a copy of A in an additional data structure for correctness checking. The 1 values on the diagonal of L, the 0 values in the upper triangular portion of L and the 0 values in the lower triangular portion of U should not be stored at any point in the code. The routine should also return the permutation matrices P and/or Q appropriate to the pivoting strategy used. These permutation matrices should not be stored as full matrices within any array. The matrices, or equivalently the set of elementary permutations that are their factors, can be represented by at most n integers each.

Your solution routine should accept as input a vector b stored in a simple 1-dimensional array. The routine should accept as input the 2-dimensional array containing the information specifying the L and U matrices and the 1 dimensional arrays specifying P and Q if appropriate. The forward and backward solves $Ly = b$ and $Ux = y$ must also work only with the information specificed in the 2-dimensional array and not expanded version of L and U. The routine should return the solution x to $Ax = b$ in the same array that the vector b was input, i.e., the solution should overwrite b in its data structure.

In order to evaluate your code's function you must implement one or more tests. These will require various support routines. These include a routine that accepts as input the 2-dimensional array containing the information specifying the L and U matrices and return in a separate 2-dimensional array the product $M = LU$. Note that this matrix multiplication must use the information specifying L and U within the data structure. It **must not** expand them into separate arrays that include the 1 and 0 values that are not stored in the input array. You will also need a routine that accepts as input the 1-dimensional arrays specifying P and Q and applies them to vectors and matrices stored in 1 dimensional and 2 dimensional arrays.

Note that the textbook has relevant MATLAB coding examples. Programs 1, 2, and 3 in Chapter 3 implement triangular system solving when L and U are given in a 2-dimensional array. Note that some thought must be given when adapting these to the current assignment since L and U will be stored together in a single 2-dimensional array resulting from the factorization algorithm.

Programs 4, 5, and 6 give in-place versions of LU without pivoting. This style is the form expected for the solution to the assignment, i.e., no matrix operations are implemented directly in terms of full dense matrices. These routines do not solve the assignment, however, since they lack the required pivoting capabilities.

Program 9 implements complete pivoting and therefore performs one of the assigned tasks. However, its style is one that yields clarity and ease of implementation. It is **not acceptable** for the solution to the assignment since it is wasteful of computations and space. Notice how an entire matrix is used to represent the accumulated row and column permutations. The accumulation of Gauss transforms and permutations is accomplished by full matrix multiplication, each requiring $2n^3 + O(n^2)$ operations despite the fact that the matrices have structure that we have seen to reduced complexity substantially. A solution program using this style will not receive credit.

Test Problems

A key consideration in this assignment is the generation of test problems. Some suggestions follow:

1. Generate L and U so they are nonsingular unit lower triangular and upper triangular matrices respectively. Evaluate their product as A. This is useful for both small and large values of n. For small values you can also constrain L and U to have integer values so A will have integer values. Take care with the magnitude of the elements of L and U. Nicely conditioned problems tend to be specified by off-diagonal elements in L smaller than 1 in magnitude and diagonal elements in U that are not too small compared to off-diagonal elements in U.

2. Remember even if A is generated from L and U when you run your routine with partial or complete pivoting nontrivial permutation matrices P and/or Q may result and you may return $\tilde{L}$ and $\tilde{U}$ such that $PAQ = \tilde{L}\tilde{U}$.

3. Randomly generated matrices tend to be nonsingular and reasonably conditioned. You can make sure any matrix is nonsingular by adding to the diagonal elements until the matrix is diagonally dominant. This guarantees success of the factorization if run without pivoting. As noted above, if you allow pivoting the routine may so do even for a diagonally dominant matrix depending on the form of dominance and the pivoting strategy used. It is also useful to note that after generating such a matrix you can apply random permutations P and Q to generate a new matrix A that will not be diagonally dominant but will still be nonsingular.

4. You can generate a symmetric positive definite A from a lower triangular L via $A = LL^T$. A is nonsingular by definition and will succeed without pivoting. Note that your code will still produce an LU factorization since it is not assumed to exploit symmetry. However, if no pivoting is used $A = LU = LDL^T$ where D is diagonal and positive. So you can check if your computed U satisfies this compatability condition with L.

5. Matrices with known structure that influences the structure or magnitude pattern of their factorizations are also useful. This is especially true if the patterns scale in a known way with n. Recall the example in Homework 2. Consider what should happen when no pivoting is used, partial pivoting and complete pivoting for various n values. Also, nonzero patterns such as banded matrices for A should produce specific nonzero patterns in L and U. More interesting structure such as those seen in Homework 3 with and without pivoting can be used to validate a code.

6. Given a matrix one can also generate a vector b based on a chosen solution x. That is given A choose x and generate b via the matrix vector product Ax.

7. Make sure that the matrices you use to check pivoting **actually require some pivoting when factored**.

8. You should run a range of problem sizes for each algorithm and problem type you evaluate.

9. Do not simply report the computed solution and/or the factors for a small number of small systems. Think about how you would report the results of testing each of the routines with many matrices including those of sizes too large to display for any useful effect.

Metrics

There are several important metrics to use when assessing the code's correctness. Some suggestions follow:

1. When comparing matrices use more than one matrix norm, e.g., the finitely computable ones, $\|M\|_1$, $\|M\|_\infty$ and $\|M\|_F$.

2. When comparing vectors use more than one norm, e.g., $\|v\|_1$, $\|v\|_\infty$ and $\|v\|_2$.

3. Check the factorization accuracy
$$\frac{\|PAQ - LU\|}{\|A\|}$$
where $\|A\| \geq 1$, i.e., relative error for large A.

4. If the solution is known by design of the problem check
$$\frac{\|x - \tilde{x}\|}{\|x\|}$$
where $\tilde{x}$ is the computed solution and $\|x\| \geq 1$.

5. You should check the accuracy via the residual $b - A\tilde{x}$ and
$$\frac{\|b - A\tilde{x}\|}{\|b\|}$$
assuming $\|b\| \geq 1$ for all attempts to solve a system, i.e., whether or not you know the true solution.

6. If you have access to a standard library you may also use the results of its LU factorization algorithms. However, care must be taken since details of pivoting strategies may yield differences in the factors and permutations. The library routines are very useful when you generate a system by choosing A and b and need a reliable way of generating x to compare with your routine's solution.

Submission of Results

Expected results comprise:

- A document describing your solutions as prescribed in the notes on writing up a programming solution posted on the class website.
- The source code, makefiles, and instructions on how to compile and execute your code including the Math Department machine used, if applicable.
- Code documentation should be included in each routine.
- All text files that do not contain code or makefiles must be PDF files. **Do not send Microsoft word files of any type**.

These results should be submitted by 11:59 PM on the due date. Submission of results is to be done via FSU Dropbox at `https://dropbox.fsu.edu`. Drop the files off for Zhifeng Deng using his MyFSU email `zd16d@my.fsu.edu`. (Note: **do not use `zdeng@math.fsu.edu` to drop off the files**) You should login to FSU Dropbox using your MyFSU login. If for some reason you cannot use FSU Dropbox please email the files to Zhifeng at the email address above.