Programming Assignment 2 Applied Linear Algebra 2 Spring 2024

The solutions are due on Canvas by 11:59 PM on Friday, February 16, 2024

General Task

This programming assignment implements and empirically evaluates the transformation of a full-rank matrix $A \in \mathbb{R}^{n\times n}$ with $n \geq n$ to an upper triangular form, i.e.,

$$T_L^{-1} A \rightarrow \begin{pmatrix} U \\ 0 \end{pmatrix}$$

where $U \in \mathbb{R}^{n\times n}$ and nonsingular. You must produce code to use Gauss transformations and row permutations to achieve this transformation

$$P_r A = LU$$

where $P_r \in \mathbb{R}^{n\times n}$ is a permutation matrix that interchanges rows, columns, $L \in \mathbb{R}^{n\times n}$ is a unit lower triangular matrix, and $U \in \mathbb{R}^{n\times n}$ is a nonsingular upper triangular matrix.

This assignment empirically evaluates the predictions you make about the structure of the factors and the general performance of the method across a large set of randomly generated problems and selected specific problems.

The Codes

1. Implement a code that computes the LU factorization of a matrix A without pivoting first and then add partial pivoting (rows). You should use the immediate update form of algorithm based on rank-one updates. The final code should be capable of performing these two tasks, based on a user selection parameter. 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. 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 or partial 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 matrix P_r when partial pivoting is used. This permutation matrix should not be stored as as a full matrix within any array. The matrix or equivalently the set of elementary permutations that are its factors, can be represented by at most n integers.

2. Your code should be based on subroutines similar to the ones discussed below.

3. You will also need various test routines designed to evaluate and validate the correctness of the code and accomplish the tasks described below. You may code in Matlab, Python (as long as you write basic code that exposes data structure details and vector or scalar loop constructs) or 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 implementing the computations described above. (They may be used for the validation codes.)

4. For both the factorization, triangular solutions and testing codes you should make use of your solutions to programming assignment 1. If you were unable to get any of them working reliably then you are free to use or convert my solutions. The testing drivers from my solution should also be adatped or used to inspire your test driver's design for this assignment.

Suggested Subroutines for the factorization:

1. FindPivot This routine shoud determine the position of the desired pivot element based on the row partial pivoting scheme when it is used. Note it is also possible to determine the pivot element for step $i+1$ in the rank-one update routine when it is producing the updated active part of the matrix (i.e., the i-th Schur complement) on step i. In which case, this routine would only be used to find the first pivot element, i.e., since there is no step 0 rank-one update.

2. Permutation**:** This routine takes the desired pivot element's indices, defines the appropriate elementary permutations P_{r_i}, stores their parameters appropriately, and applies them to the current active portion of the matrix.

3. FormGauss**:** This routine examines the appropriate part of the permuted active part of the matrix and determines the parameters determining M_i. You may find it convenient to return the parameters in a work vector and then place the values in the appropriate positions of the array in which A is being transformed and the elements of L and U are created.

4. ApplyGauss**:** This routine applies M_i^{-1} to the active part of the matrix. **It is recommended but not required that this routine also determine the next pivot element in the updated active part when row partial pivoting is used.** If this is not done then you can use the separate pivot search routine mentioned above

after the update is done to begin the next step. The index r_{i+1} should be returned to the calling routine for use in the next step of elimination. Note that you should be able to determine the index in the same pass through the active part of the matrix that performs the rank-1 update defined by M_i^{-1}, i.e., you should not update the matrix on one pass through and then make a second pass over the candidate pivot set to determine the $i+1$ pivot element. Clearly, this step should be surpressed when applying M_{n-1}^{-1} since there is no following step n.

Routines to support the evaluation of the factorization:

1. Implement a routine that accepts as input the 1-dimensional array specifying P_r and applies it to a matrix stored in a 2-dimensional arrays and returns the result in a separate 2-dimensional array, i.e., $A_2 \leftarrow P_r A_1$. This matrix of course will be a subarray of the array used to produced L and U.

2. Implement (or use your code from Program 1) a matrix multiplicaiton routine that accepts as input the 2-dimensional array containing the information specifying the L and U matrices and returns in a separate 2-dimensional array the product $M = LU$ **or** the product $M = |L||U|$ based on user selection indicated by an input flag. (This second function requires an addition to your routines from Program 1 if you use them.) Note that these matrix multiplications 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.

3. The test routines should include the computation of $\|W\|$ where W is a matrix. The norms used should be finite in computation, i.e., $\|W\|_1$, $\|W\|_\infty$, or $\|W\|_F$. You may use library routines to compute $\|W\|_2$ if you wish but you are definitely not expected to generate the code for the 2-norm.

Library Codes

You may use libaries and external routines in your test drivers to generate solutions for comparisons, to generate historgrams, graphs and any other useful summary display mechanisms. Make sure when using library routines as part of your empirical analysis, you carefully check, e.g., the P_r, L and U generated by your routines and those generated by the library, e.g., MATLAB. These factors are not unique given that pivoting choices are not unique in general. So they may not match. The products LU however should match the matrices PA generated by your routine and a library respectively.

Metrics

There are several important metrics to use when assessing the code's correctness. These metrics should be computed in double precision espeicially if you have run your routines in

single precision. (The ability to run in both is not required for this assignment but is easily done if you use a compiled and typed language.)

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. Check the factorization accuracy

$$\frac{\|P_r A - LU\|}{\|A\|}$$

   where $\|A\| \geq 1$, i.e., relative error for large A.

3. If the solution is known by design of the problem check

$$\frac{\|x - \tilde{x}\|}{\|x\|}$$

   where $\tilde{x}$ is the computed solution and $\|x\| \geq 1$.

4. 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.

5. If you have access to a standard library you may also use the results of its LU factorization algorithms. However, as noted above care must be taken since details of pivoting strategies may yield differences in the factors and permutations. The library routines or your Program 1 solutions 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 computed solution.

6. You should compute the growth factor

$$\gamma_\epsilon = \frac{\| \ |L_\epsilon||U_\epsilon| \ \|}{\|A\|}$$

   where $L_\epsilon U_\epsilon = P_r A$ is the **computed** factorization of $P_r A$ using the selected pivoting strategy. This is important in that is used to assess the numerical stability of solving systems but it is also useful for checking the correctness of structured problems such as the one in the study questions with the large growth of elements.

Generation of Test Problems

A key consideration in this assignment is the generation of test problems. They must have full rank, i.e., linearly independent columns.

Some suggestions follow:

1. The matrices should be generated using double precision storage and computation. You can then coerce the type to single when storing the matrix before calling your single precision code if you decide to check both single and double precision execution of your codes. This is, however, not required but it is encouraged that you do this at some point.

2. Generate L and U so they are nonsingular unit lower triangular and upper triangular matrices respectively. Evaluate their product to define A. By their structure these matrices have linearly independent columns and therefore their product has full-rank.

   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 have a reasonable range in magnitudes. The off-diagonal elements in U should not be significantly larger in magnitude than the diagonal elements of U. Recall, it was suggested to do this in Program 1 problems.

3. Remember even if A is generated from L and U when you run your routine with partial pivoting, nontrivial permutation matrices P_r may result and you may return $\tilde{L}$ and $\tilde{U}$ such that $P_r A = \tilde{L}\tilde{U}$.

4. Square randomly generated matrices tend may be nonsingular and reasonably conditioned. Random triangular matrices tend not to be well-conditioned, as you have seen. You can make sure any matrix is nonsingular by adding to the diagonal elements until the matrix is diagonally dominant by rows, columns or both.[1] This guarantees success of the factorization if it is 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, A you can apply random permutations $\tilde{P}$ and $\hat{P}$ to generate a new test matrix $\tilde{A} = \tilde{P}A\hat{P}$ that will not be diagonally dominant but will still be nonsingular. Of course, $\tilde{P}$ and $\hat{P}$ are not necessarily the permuations that will be generated by applying partial or complete pivoting to $\tilde{A}$.

5. You can generate a symmetric positive definite $A \in \mathbb{R}^{n\times n}$ from a lower triangular $\tilde{L}$ with positive diagonal elements (not necessarily 1) via $A = \tilde{L}\tilde{L}^T$. A is nonsingular by

[1] A matrix is strictly diagonally dominant by rows (columns) if the magnitude of each diagonal element is strictly larger than the sum of the magnitudes of all off-diagonal elements in the same row (column). A matrix may of course be diagonally dominant by rows and columns simultaneously.

definition and factorization will succeed without pivoting. Note that your code will still produce an LU factorization since your factorization routine is not designed to exploit symmetry. However, there is a relationship between L, U and $\tilde{L}$. This is probed in the first set of structured factorization tasks.

6. Matrices with known structures that influence 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 the homework problems that has large elements in U. Consider what should happen when no pivoting or partial pivoting is used for various n values. Also, nonzero patterns such as banded matrices for A should produce specific nonzero patterns in L and U. Structure is the subject of the first set of empirical tasks discussed below.

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 the factors or accuracy of the factorization 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.

Empirical Tasks Set 1 : Structure

Consider the following structured matrices, predict the structure of their factors and factorization, and verify them empirically. Your solutions must include an explanation of your observations and justification for the conclusion that your predictions are correct. This could include solutions to related homework or study questions.

1. Consider a matrix $A \in \mathbb{R}^{n \times n}$ that is unit lower triangular and the elements in the strict lower part all have magnitude less than 1, i.e., $|\lambda_{ij}| < 1$ for $i > j$, $\lambda_{ij} = 0$ for $i < j$, and $\lambda_{ii} = 1$. Consider factoring with no pivoting and partial row pivoting. What can be said about the L, U, and P_r factors? (Note that your code does not know that A is unit lower triangular and will eliminate the elements below the diagonal by applying Gauss transformations.)

2. Consider a lower triangular matrix A again but this time let the diagonal elements be positive and not 1 and the elements in the strictly lower triangular part be larger than 1, e.g.,

$$A = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 & 0 \\ 4 & 3 & 2 & 0 & 0 \\ 5 & 4 & 0 & 2 & 0 \\ 6 & 5 & 4 & 3 & 2 \end{pmatrix}.$$

   Consider factoring with no pivoting and partial row pivoting. What can be said about the L, U, and P_r factors?

3. Consider a matrix $A \in \mathbb{R}^{n \times n}$ that is tridiagonal with all elements on the main diagonal and the first super and subdiagonals nonzero, i.e., $\alpha_{ii} \neq 0$, $\alpha_{i+1,i} \neq 0$, and $\alpha_{i,i+1} \neq 0$. Of course, these must be such that the matrix is nonsingular.

   Suppose A is, additionally, strictly diagonally dominant by rows and columns. Consider factoring with no pivoting and partial row pivoting. What can be said about the L, U, and P_r factors?

4. Suppose $A \in \mathbb{R}^{n \times n}$ is a symmetric positive definite generated from a lower triangular $\tilde{L}$ with positive diagonal elements (not necessarily 1) via $A = \tilde{L}\tilde{L}^T$. A is nonsingular by definition and factorization will succeed without pivoting. Note that your code will still produce an LU factorization since your factorization routine is not designed to exploit symmetry. What is the relationship between L, U and $\tilde{L}$?

Empirical Tasks Set 2 : General Trends

As with the assignment for Program 1, this set requries the generation of a large set of problems grouped and empirically analyzed by problem size n and, if appropriate the class of matrix problem considered.

For each problem size and class of problem, generate many example problems and evaluate the various metrics discussed earlier You should present your results in a form appropriate to characterize these metrics over a large data set, i.e., too large to look at each problem individually. This can be done, for example, by graphs and histograms. The latter is particularly useful for detecting outliers in the performance such as large factorization error. These outliers can be discussed in more detail and explained if you wish.