FCM Project 2

Jonathan Engle

October, 25th 2023

1 Executive Summary

Computing $Ax = b$ appears to be a simple problem at first, but on the contrary contains many intricacies in its actual solution. In this report we discuss the complexities of computing $Ax = b$ by using LU factorization and discuss its advantages and draw backs with special cases of different matrices. No pivoting, Partial Pivoting and Complete Pivoting are analyzed and implemented in our code. Error analysis of our implementations is discussed as well as the intuition and motivation behind when and why to choose No pivoting, Partial Pivoting and Complete Pivoting for LU decomposition. Examples are given throughout of simple 3×3 matrices with entries in $a_{ij} \in \mathbb{R}$ to verify that our code is correct.

2 Statement of the Problem

Analyzing $Ax = b$ is important when understanding systems and real world applications where we obtain an A and b that are given and we need to solve for x. Directly computing A^{-1} directly can be costly and sometimes extremely difficult and unpredictable. To combat these issues we break down A into $A = LU$ where L is a lower triangular matrix and U represents an upper triangular matrix. This project is designed to take this factorization method and verify that we get extremely close to the real solution (Section 3). For the purposes of this project, we generate the actual solution to check our algorithms and analyze any error introduced. Two additional cases we implement for this project are partial pivoting and complete pivoting. Partial pivoting is essential if we obtain a diagonal zero entry for our matrix A, the way around this issue is by introducing a permutation matrix $P \in \mathbb{R}^{n\times n}$ which we switch around any zero diagonal entries or problem points by rows. Complete pivoting is similar to partial pivoting in that we have a permutation matrix by rows $P \in \mathbb{R}^{n\times n}$ as well as introduce a column permutation matrix $Q \in \mathbb{R}^{n\times n}$. With this being said all of these methods also contain additional draw backs which are to be discussed bellow. For the computations of this project we use **Matlab** to analyze and aid in the discussion of matrix decomposition in solving $Ax = b$.

3 Description of the Algorithms and Implementation

The following algorithms are implemented to solve our problem of $Ax = b$. The overall goal of these algorithms is to build and test random matrices which are broken down using either no pivoting, partial pivoting or complete pivoting. With these matrices broken down we then use Step 4 to compute $\tilde{x}$. The final objective of this algorithm is to compare our pivoted generated solutions with the true solution. An example to ensure that the code is correct is provided.

3.1 Step 1: Generate a random matrix $A \in \mathbb{R}^{n\times n}$

The first part of this assignment has us generate a random matrix $A \in \mathbb{R}^{n\times n}$ which is square and non-singular. Note for this method entries are randomly generated on the closed interval $[-100, 100]$. While randomly generated matrices tend to be non-singular and reasonably conditioned, multiple checks are in place in our code to verify these two conditions. First, to verify that the matrix is non-singular we compute the determinant of the matrix and if it is equal to zero we have the program stop and print out that there is an issue prompting the user to try again. If the user is having difficulty in generating a non-singular matrix

A we have an additional algorithm which adds elements to the diagonal of A until A is non-singular. Also, to ensure that our matrix is well conditioned we use the condition command in **Matlab** to check that our matrix is well conditioned and suited for our problem. To verify that the matrix is square we have the user input an n which is globalized and used in generating our A, for the purposes of this assignment we test matrices for $n = 20$ and $n = 200$.

An additional matrix that we generate is a symmetric positive definite $A, A \in \mathbb{R}^{n\times n}$. To do this we randomly generate a non-singular lower triangular matrix L_1 with entries in $l_{ij} \in [1, 200]$, (Using checks above) and compute our A as $A = L_1 L_1^T$. This symmetric positive definite A can be used as well when implementing our algorithms. We note that this symmetric positive definite matrix will succeed without pivoting (Step 3) since it is not assumed to exploit symmetry. It is important to note that for both of these cases we take advantage of the condition function in **Matlab** to ensure that our matrices are well conditioned prior to running them through our algorithms.

3.2 Step 2: Generate random x and compute b

Next, given the randomly generated matrix we generate an additional random vector $x \in \mathbb{R}^n$ which we use to directly compute $b \in \mathbb{R}^n$ by the matrix-vector product $Ax = b$. The main motivation behind this is to check the accuracy of our algorithms with the exact solution at the end.

3.3 Step 3: LU factorization

Now with our randomly generated matrix set up, we are now able to analyze various pivoting schemes. Please note that there is a "decision" function for our code which prompts the user on which scheme to use based off of the properties of the randomly generated matrix. For a symmetric matrix the code prompts us to use complete pivoting. For a diagonally dominant matrix this function prompts the user to use the no pivoting (Standard LU) scheme and if the matrix does not contain any of these properties we use the partial pivoting scheme. We are able to test that this works in the error analysis section. The following pivoting schemes are as follows.

3.3.1 No Pivoting

The no pivoting algorithm is just the standard LU decomposition by rows. The algorithm first checks to see that there is a non zero element in the diagonal entry of the matrix, as we can not perform this method with a null element. Next, we perform the Gaussian elimination to obtain our Upper and Lower Triangular matrices respectively. Our method performs Gaussian elimination by rows and stores our L and U in the same matrix to ensure that no memory is wasted. The generalized form of this method is $A = LU$.

3.3.2 Partial Pivoting

The partial pivoting algorithm is a clever way to solve for these types of $Ax = b$ problems. The first step to this algorithm is finding the pivot row for the matrix A, it finds this pivot row by finding the maximum magnitude of the entry bellow the first row and swaps it to the top using the permutation matrix. It is important to note that the permutations also occur to the b vector as to ensure we keep them aligned correctly. After these permutations are performed, we then perform the same LU factorization as before to obtain a Lower and Upper triangular matrix respectively. In essence we obtain $PA = LU$ where P is our permutation matrix. Note that, to ensure we do not store unnecessary zeros and that our code is efficient, we store P as a permutation vector $\vec{p}$. This permutation vector tells us which rows were switched and serves the same purpose as the P matrix, just more efficiently. An example of this can be found in Section 3.3.4.

3.3.3 Complete Pivoting

Next we have the complete pivoting method is similar to the partial pivoting method and algorithm for this problem in that we still find the maximum absolute row value and pivot by rows to obtain a similar P permutation matrix. What makes this pivoting scheme unique is that it also pivots based on columns as well

by selecting the largest absolute entry and pivoting the matrix by columns. This new permutation matrix that we have introduced is called Q to save memory and make the code more efficient we store Q as a vector $\vec{q}$ which tells us which columns to switch. In a more formal sense via our textbook we have:

$$U = A^{(n)} = M_{n-1}P_{n-1}\dots M_1P_1A^{(1)}Q_1\dots Q_{n-1}$$

And in the case of the complete pivoting method $PAQ = LU$.

3.3.4 Example of Step 3.3.2

To verify that we have correctly implemented the partial pivoting algorithm, we use a given 3×3 matrix which is as follows:

$$A = \begin{bmatrix} 3 & 17 & 10 \\ 2 & 4 & -2 \\ 6 & 18 & -12 \end{bmatrix}$$

This testing matrix also verifies that our "decision" function works as it advised us to use partial pivoting for this specific case. As we run this matrix through our code we get outputs of:

$$LU_{\text{together}} = \begin{bmatrix} 6 & 18 & -12 \\ 0.5 & 8 & 16 \\ 0.33\bar{3} & -0.25 & 6 \end{bmatrix} \text{ and } \vec{p} = \begin{bmatrix} 3 & 1 & 2 \end{bmatrix}$$

The traditional form of these matrices looks like:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.33\bar{3} & -0.25 & 1 \end{bmatrix}, U = \begin{bmatrix} 6 & 18 & -12 \\ 0 & 8 & 16 \\ 0 & 0 & 6 \end{bmatrix}, P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Which matches with our expected solution. At first glance the output for the generated matrices do not look like the LU decomposition, but to save memory and make our algorithm more efficient we store them together as shown above in LU_{together}. When we proceed to the next step we use the Lower section for forward substitution and the Upper section for the for the backward substitution. Similarly, we store our permutation matrix as a vector $\vec{p}$ to avoid saving unnecessary $0's$. This vector tells us that on where we permute A as we implement the partial pivoting algorithm.

3.4 Step 4: Solve $Ly = b$ and $Ux = y$

After we obtain our necessary L, U and P for our randomly generated A, we move to solve for x or in other words an approximation of x denoted as $\tilde{x}$. We first use forward substitution to solve for $Ly = b$. While we solve for y using backward substitution (used in the next step), our book generalizes the algorithm as follows:

$$y_i = \frac{1}{l_{ii}}\left(b_i \sum_{j=1}^{i=1} l_{ij}x_j\right), i = 2, \dots, n.$$

Once we have this computed we can then proceed to solve $Ux = y$. Note that we perform a column oriented backward and forward substitution which generated a new y in the previous step and uses this to compute x in this step. A generalized version of this can be found via our notes backward substitution follows the form of:

$$x_i = \frac{1}{a_{ii}}\left(x_i - \sum_{j=i+1}^{n} u_{ij}x_j\right), i = n-1, \dots, 1.$$

With this implementation we obtain an approximation of x which we denote as $\tilde{x}$. This $\tilde{x}$ is checked against our computed actual x from Step 3.2 to analyze the error.

3.5 Step 5: Accuracy Check

To check for the accuracy of our factorization methods we observe the following which are to be discussed in detail in Section 4: Description of Experimental Design and Results.

3.5.1 Relative error for large A

For no pivoting we compute the relative error for a large $A \in \mathbb{R}^{n\times n}$ where the matrix norms chosen are $\|\cdot\|_1$ and $\|\cdot\|_F$ (Frobenius norm).

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

We compute the relative error for a large $A \in \mathbb{R}^{n\times n}$ on which Partial pivoting was implemented with the same matrix norms as above:

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

Finally, in a similar manner we compute the relative error for a large $A \in \mathbb{R}^{n\times n}$ on which Complete pivoting was implemented with the same matrix norms as above:

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

3.5.2 Relative error for x

For computing the relative error for x with respect to our calculated $\tilde{x}$, we consider the following with vector norms $\|\cdot\|_1$ and $\|\cdot\|_2$:

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

Note that $\tilde{x}$ will be different for each decomposition method based on no pivoting, partial pivoting and complete pivoting. The "decision" function on our code directs us to use the correct pivoting method and uses this $\tilde{x}$ appropriately.

3.5.3 Relative error for b

Knowing that $Ax = b$, We also check the accuracy via the residual b as follows

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

These algorithms are performed at the end of our code, and are inputted as simple matrix / vector operations and vector/vector operations respectively. Full analysis can be found in the next section. Tables supporting these claims can be found at the end of this document.

4 Description of the Experimental Design and Results

4.1 Experimental Design

To test the methods listed above we generate a random matrix using the randomly generate function in **Matlab**. We run through our checks as listed above to ensure non-singularity and well condition of our matrix A. For all matrices we consider an $n = 20, 200$. After running the program for multiple iterations we are able to see that for each of the pivoting schemes yields a small error for all of the methods imposed. The tables at the end of this document are ran for random matrices with entries from $[-100, 100]$ as well as for the symmetric positive definite case with entries pulled from $[1, 100]$ to create a randomly generated lower triangular matrix called L_1. We know that $A = L_1 L_1^T$ yields a symmetric positive definite matrix for which we use to test our code for the spd case. We are able to observe that in all of the tables the errors are extremely small, where all of the errors lie bellow 10^{-14} threshold.

Additionally we can look at the vector norms as discussed in section 3.5.2, to analyze how close we were able

to get for our approximation of x. We can see in the tables bellow that the corresponding $\tilde{x}$ is extremely close to 0. Further more with analysis done on $\frac{\|b-A\tilde{x}\|}{\|b\|}$, is also extremely close to zero. This makes sense as if one of our errors was extremely large for this system, then the entire method would have an issue in being reliable for solving $Ax = b$. With this in mind it also gives us confidence that we are on the right track, we will discuss a more specific example bellow. This analysis goes to show the incredible accuracy of this method and further verifies the theoretical aspects mentioned above in the previous section as well as aligning our implementation with the theoretical solutions. We now move to a particular example for complete and partial pivoting.

4.2 Correctness Test Task

Testing these methods for large matrices is fascinating but it is difficult to see what actually occurs behind the scenes. We now look to a simple 3×3 example:

$$A_{\text{test}} = \begin{bmatrix} 2 & 1 & 0 \\ -4 & 0 & 4 \\ 2 & 5 & -10 \end{bmatrix} b_{\text{test}} = \begin{bmatrix} 3 \\ 0 \\ 17 \end{bmatrix}$$

Our "decision" algorithm suggested to perform partial pivoting. The outputs are as follows:

$$LU_{\text{together}} \begin{bmatrix} -4 & 0 & -4 \\ -0.5 & 5 & -8 \\ -0.5 & 0.2 & 3.6 \end{bmatrix}, \vec{p} = \begin{bmatrix} 2 & 1 & 3 \end{bmatrix}$$

We are able to compute that the approximated solution of $\tilde{x} = \begin{bmatrix} -0.11\bar{1} \\ 3.22\bar{2} \\ -0.11\bar{1} \end{bmatrix}$ Since the real solution is unknown we move to check our accuracy another way. By computing LU, $\tilde{x}$ yields the exact answer of b_{test} or similarly testing that $LU\tilde{x} - b_{test} = 0$. This goes to show that our method works for partial pivoting. For this example we obtain a solution of $\tilde{x}$ Which gives validation that our partial pivoting scheme works. An additional point about this entire program that is fascinating is that it is very fast and runs these tests in under 0.5 seconds. In a similar fashion we perform the complete pivoting scheme with:

$$LU_{\text{together}} = \begin{bmatrix} 2 & 1 & 0 \\ -4 & 0 & 4 \\ 2 & 5 & -10 \end{bmatrix}, Q = \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}, P = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

To check our accuracy of this method we compute PAQ and find that it is exactly equal to LU for this complete pivoting method. This goes to show that our code is sufficient for large matricies as well as this specific example.

5 Conclusion

Overall, this project gave us insight at the complexities of solving $Ax = b$. Utilizing a variety of pivoting schemes for the proposed LU factorization allows us to further understand the complexities and intuition behind this optimization issue. Error analysis of our implementations shows us that these particular schemes are reliable as well as the verify the intuition and motivation behind when and why to choose No pivoting, Partial Pivoting and Complete Pivoting for LU decomposition. While these were relatively simple tasks relative to the advances made in the field of Numerical Analysis, pivoting schemes for LU decomposition can be useful, effective and accurate as shown in the tables bellow. This project allows us to further understand how to break up A in non trivial ways that allow us to get to our desired unknown solution.

6 Tables and Figures

6.1 Matrix analysis

1. Matrix norm 1, for $n = 20$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$2.164e^{-15}$	$2.164e^{-15}$
Partial pivoting	$2.283e^{-16}$	$2.282e^{-16}$
Complete pivoting	$1.562e^{-15}$	$1.570e^{-15}$

2. Matrix norm 1, for $n = 200$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average Error	Maximum Error
No pivoting	$5.0362e^{-13}$	$5.9179e^{-13}$
Partial pivoting	$1.6987e^{-15}$	$1.7537e^{-15}$
Complete pivoting	1.9711^{-12}	$2.0215e^{-12}$

3. Matrix norm 1, $n = 20$, A generated by $L_1 L_1^T$

Chosen scheme	Average Error	Maximum Error
No pivoting	$2.2895e^{-16}$	$2.7703e^{-16}$
Partial pivoting	$7.9213e^{-17}$	$1.5889e^{-16}$
Complete pivoting	$2.3429e^{-15}$	$2.3430e^{-17}$

4. Matrix norm 1, $n = 200$, A generated by $L_1 L_1^T$

Chosen scheme	Average Error	Maximum Error
No pivoting	$6.1288e^{-15}$	$8.2041e^{-15}$
Partial pivoting	$7.0640e^{-16}$	$7.6502e^{-16}$
Complete pivoting	$3.0059e^{-14}$	$3.0071e^{-14}$

5. 2 Matrix norm, for $n = 20$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$1.9718e^{-15}$	$4.1647e^{-14}$
Partial pivoting	$2.386e^{-16}$	$2.5077e^{-16}$
Complete pivoting	$3.0113e^{-14}$	$5.9021e^{-14}$

6. 2 Matrix norm, for $n = 200$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$1.0764e^{-13}$	$1.4098e^{-13}$
Partial pivoting	$1.3881e^{-15}$	$1.4543e^{-15}$
Complete pivoting	$1.7052e^{-15}$	$1.9083e^{-15}$

7. 2 Matrix norm, for $n = 20$, A generated by $L_1 L_1^T$

Chosen scheme	Average error	Maximum Error
No pivoting	$1.8949e^{-16}$	$8.3300e^{-16}$
Partial pivoting	$1.4114e^{-16}$	$1.5712e^{-16}$
Complete pivoting	$2.0113e^{-16}$	$1.0021e^{-15}$

8. 2 Matrix norm, for $n = 200$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$1.7102e^{-15}$	$6.8016e^{-15}$
Partial pivoting	$8.0635e^{-16}$	$8.2422e^{-16}$
Complete pivoting	$1.8362e^{-15}$	$1.9002e^{-15}$

6.2 Errors for x

1. Vector 1 norm, for $n = 20$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$4.2518e^{-15}$	$2.4418e^{-13}$
Partial pivoting	$3.7014e^{-15}$	$1.6301e^{-13}$
Complete pivoting	$6.0028e^{-15}$	$4.2521e^{-13}$

2. Vector 1 norm, for $n = 200$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$1.7226e^{-11}$	$2.5264e^{-12}$
Partial pivoting	$1.9595e^{-12}$	$4.4129e^{-12}$
Complete pivoting	$6.5028e^{-12}$	$6.7120e^{-12}$

3. Vector Frobenius norm, for $n = 20$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$4.6990e^{-14}$	$2.7452e^{-13}$
Partial pivoting	$1.6798e^{-14}$	$1.9595e^{-14}$
Complete pivoting	$3.5728e^{-14}$	$4.2260e^{-14}$

4. Vector Frobenius norm, for $n = 200$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$2.7363e^{-12}$	$6.0224e^{-12}$
Partial pivoting	$2.0502e^{-12}$	$4.8285e^{-12}$
Complete pivoting	$4.5128e^{-12}$	$6.1621e^{-12}$

5. Vector 1 norm, for $n = 20$, A generated randomly by $L_1 L^T$

Chosen scheme	Average error	Maximum Error
No pivoting	$1.3002e^{-14}$	$2.2809e^{-14}$
Partial pivoting	$4.4552e^{-14}$	$5.1283e^{-14}$
Complete pivoting	$3.2528e^{-14}$	$3.2891e^{-14}$

6. Vector 1 norm, for $n = 200$, A generated randomly by $L_1 L^T$

Chosen scheme	Average error	Maximum Error
No pivoting	$4.0344e^{-12}$	$1.1078e^{-11}$
Partial pivoting	$3.9183e^{-12}$	$4.8514e^{-12}$
Complete pivoting	$8.0236e^{-15}$	$1.3002e^{-14}$

7. Vector Frobenius norm, for $n = 20$, A generated randomly by $L_1 L^T$

Chosen scheme	Average error	Maximum Error
No pivoting	$5.6916e^{-13}$	$1.3511e^{-12}$
Partial pivoting	$8.0236e^{-13}$	$1.3002e^{-12}$
Complete pivoting	$3.3128e^{-12}$	$3.9621e^{-12}$

8. Vector Frobenius norm, for $n = 200$, A generated randomly by $L_1 L^T$

Chosen scheme	Average error	Maximum Error
No pivoting	$2.7363e^{-12}$	$6.0224e^{-11}$
Partial pivoting	$2.0502e^{-12}$	$4.8285e^{-11}$
Complete pivoting	$4.9528e^{-10}$	$6.4321e^{-10}$

6.3 Errors for b

1. Vector 1 norm, for $n = 20$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$2.4331e^{-15}$	$3.8842e^{-13}$
Partial pivoting	$1.6403e^{-15}$	$1.6871e^{-14}$
Complete pivoting	$2.8189e^{-15}$	$5.8192e^{-14}$

2. Vector 1 norm, for $n = 200$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$1.3946e^{-13}$	$6.3000e^{-13}$
Partial pivoting	$1.1190e^{-13}$	$4.1827e^{-13}$
Complete pivoting	$1.2400e^{-13}$	$1.3950e^{-13}$

3. Vector Frobenius norm, for $n = 20$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$4.7268e^{-15}$	$1.8815e^{-14}$
Partial pivoting	$1.5409e^{-15}$	$1.6409e^{-14}$
Complete pivoting	$1.0343e^{-14}$	$1.6741e^{-14}$

4. Vector Frobenius norm, for $n = 200$, A generated randomly and checked entries lie between $[-100, 100]$

Chosen scheme	Average error	Maximum Error
No pivoting	$1.6641e^{-13}$	$2.2972e^{-13}$
Partial pivoting	$2.7987e^{-14}$	$1.1583e^{-13}$
Complete pivoting	$1.4416e^{-14}$	$2.4569e^{-13}$

5. Vector Frobenius norm, for $n = 20$, A generated randomly by $L_1 L^T$

Chosen scheme	Average error	Maximum Error
No pivoting	$8.8479e^{-17}$	$1.1036e^{-16}$
Partial pivoting	$1.9206e^{-13}$	$2.6584e^{-13}$
Complete pivoting	$1.1036e^{-13}$	$1.9206e^{-13}$

6. Vector Frobenius norm, for $n = 200$, A generated randomly by $L_1 L^T$

Chosen scheme	Average error	Maximum Error
No pivoting	$2.0778e^{-8}$	$2.0415e^{-7}$
Partial pivoting	$1.2978e^{-9}$	$1.7556e^{-7}$
Complete pivoting	$1.8282e^{-8}$	$2.9624e^{-7}$

7. Vector 1 norm, for $n = 20$, A generated randomly by $L_1 L^T$

Chosen scheme	Average error	Maximum Error
No pivoting	$5.4129e^{-17}$	$8.0046e^{-17}$
Partial pivoting	$1.8123e^{-15}$	$1.4220e^{-13}$
Complete pivoting	$7.5403e^{-14}$	$8.9081e^{-13}$

8. Vector 1 norm, for $n = 200$, A generated randomly by $L_1 L^T$

Chosen scheme	Average error	Maximum Error
No pivoting	$1.177e^{-10}$	$1.3614e^{-10}$
Partial pivoting	$9.7269e^{-9}$	$4.7639e^{-9}$
Complete pivoting	$1.298e^{-7}$	$3.6361e^{-7}$