# FCM 2 HW 3

Ryan Bausback

March 2021

## 1 Executive Summary

Our goal is to implement the LU factorization as part of solving the matrix-vector equation $Ax = b$, and then to utilize this algorithm to empirically validate predictions about the factorization of matrices with particular structure. In addition, the general trends associated with solving random matrices with particular structures and potentially large $n$ where $A \in \mathbb{R}^{n*n}$ will be explored, as well as validation of $O(n^3)$ complexity. The results of the experiments illustrated the correctness of the algorithm's implementation as well as confirming our predictions about the stability and conditioning of the factorization of particular structures

## 2 Statement of Problem

Computing the LU factorization of matrix $A$ is an important step in stably computing the solution of$Ax = b$, since directly computing $A^{-1}$ for large $n$ tends to be computational costly as well as numerically unstable. [1] Therefore, an algorithm to compute the LU factorization of $A$ and then sequential solve $Ly = b$ and $Ux = y$ will be implemented. To ensure stability, this will come in three different "flavors": the algorithm without any pivoting by permutations, with partial pivoting (row swaps only), and with complete pivoting (both row and columns swaps), each further increasing the stability of the algorithm.

Once correctness has been achieved for simple problems, it will then applied to matrices of particular structure such as diagonal, antidiagonal, tridiagonal, lower triangular, and others in order to validate predictions about the factors and permutations, as well as to investigate stability and conditioning. Finally, matrix equation for matrices with large n, some which will also have the structures above, will be solved and the stability of that part of the implementation investigated. The order of computation of $O(n^3)$for the factorization for increasing size of n will also be validated.

Based on the theoretical results in the notes, we should expect the algorithm with complete pivoting to be the most stable with the smallest growth factors, while the algorithm without pivoting to be the most unstable, with large growth factors in some cases. [1]

# 3 Description of Mathematics

The basis for the LU factorization is the application of Gauss transforms to the matrix $A \in \mathbb{R}^{n*n}$ in the equation $Ax = b$. A Gauss transform is defined as $G_i = I + \ell_i e_i^T$ where $I$ is the identity matrix, $e_i^T$ is the transpose of the i-th basis vector of $\mathbb{R}^n$, and $\ell_i$ is a vector with elements equal to 0 for index $0 \leq j \leq i$ and nonzero elements for $i+1 \leq j \leq n$. Applying $G_i^{-1}$ to $A$ amounts to eliminating the subdiagonal elements in column i while only updating the submatrix of the $i+1 \to n$ columns and $i+1 \to n$ rows, as show below:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -\lambda_{2,1} & 1 & 0 & 0 \\ -\lambda_{3,1} & 0 & 1 & 0 \\ -\lambda_{4,1} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^* & a_{23}^* & a_{24}^* \\ 0 & a_{32}^* & a_{33}^* & a_{34}^* \\ 0 & a_{42}^* & a_{43}^* & a_{44}^* \end{bmatrix}$$

where $\lambda_{i,j} = \frac{a_{i,j}}{a_{j,j}}$. If we then apply this n-1 times for each row $1 \to n-1$ to eliminate all of the elements below the diagonal, we get $G_{n-1}^{-1}...G_1^{-1}A = U$, which is the $U$ from LU factorization with an upper triangular structure. If we then divide left and right by the product of the transforms, we get $A = (G_{n-1}^{-1}...G_1^{-1})^{-1}U = (G_1...G_{n-1})U = LU$ where L is the L from the LU factorization with lower triangular structure defined in terms of the Gauss transforms. Since each Gauss transform is an elementary unit lower triangular matrix, taking the inverse just amounts to changing the signs of the subdiagonal elements.[1]

However, in some cases, $A$ may be nonsingular, meaning the inverse exists, but computing the LU factorization may fail because one of the diagonal elements either initially or after updates is 0. In order to continue to apply the Gauss Transforms and find LU, we can apply a permutation to the matrix A, in the form of a permutation matrix $P$ (indicating a row swap) or $Q$ (indicating a column swap), so that the diagonal element is no longer 0. A row swap will be multiplied from the left, while a column swap will be multiplied from the right. This gives us that $PAQ = LU$ where $P = P_{n-1}...P_1$ and $Q = Q_1...Q_{n-1}$ or the product of the individual permutations applied at each. Hence the LU is actually the factorization of the permuted $A$.[1] The LU factorization without pivoting can hence be thought of as a special case of the pivoting LU where all of the applied permutation matrices are the identity matrix $I$. In practice, permutations to the left-hand-side must also be applied to the LU of the right hand side.

Once the LU factorization is computed, and any permutations to $A$ are correspondingly applied to $b$, we can then use the L and U matrices to solve for $x$. This is a two stage process. In the first stage, we solve $Ly = b^*$, where $b^*$ is the permuted version of the original $b$. Given the lower triangular structure of L, this amounts to solving:

$$\begin{bmatrix} 1 & & \\ \ell_{21} & 1 & \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} y = \begin{bmatrix} b_1 \\ b2 \\ b3 \end{bmatrix} \iff \begin{bmatrix} y_1 = b_1 \\ y_2 = b_2 - y_1 * \ell_{21} \\ y_3 = b_3 - y_1 * \ell_{31} - y_2 * \ell_{32} \end{bmatrix}$$

which can be solved sequentially in a loop by an algorithm. Then, once we have $y$, we can then solve $Ux = y$ which uses the same method as for L but with ordering reversed.

However, because the computations that happen in the computer are for finite precision floating point numbers, the computed solution that we get from the factorization will actually not be the true factorization of $A = LU$, it will be $A + E = LU$, were $E$ is the perturbation matrix, or how much the exact arithmetic product of the computed L and U will differ from the true $A$ (the error). Since this is a matrix, we can use any norm to define a relative error $\frac{||E||}{||A||} < \omega_n \gamma_\epsilon$, where $\omega_n$ is constant defined by the size of $A$, $n$ and $\gamma_\epsilon$ is the computed growth factor. Here, $\gamma_\epsilon = \frac{|||L_\epsilon||U_\epsilon|||}{||A||}$, where the L and U are the computed L and U from the LU factorization algorithm, defines a bound on what the error can be in the worst case scenario when $A$ is ill conditioned and/or the algorithm is unstable, but it is rarely achieved.

# 4 Algorithm and Implementation

The algorithm to solve $Ax = b$ was broken up into two major functions, representing the two major stages of solving the problem: 1) factoring A into L and U, and 2) the forward and backward triangular solves of $Ly = b$ and $Ux = y$.

The LU factorization algorithm was separated into 3 distinct if-statements regions, which use a flag in order to indicate whether no pivoting, partial pivoting, or complete pivoting is to be used. However, the main workhorse of all three is a triple nested for-loop that applies the inverse Gauss transform in-place by first computing the value of the current leftmost element of the active part of the matrix divided by the current diagonal pivot element. Since this all happens in-place within the current A matrix, this will be the element of L in the corresponding index of the result (denoted $\lambda_{ik}$). This is acceptable since all elements in the current column are zeroed out by definition when the Gauss Transform is applied. A check is made direct prior to this to ensure that the pivot element is not zero, and if it is, to exit with an error message. Then $-\lambda_{ik}$ will be multiplied by each element in the pivot element's current column and subtracted from each element in the current row in order to update that row. This is seen in Figure1. The triple for-loop nested structure of the algorithm thus gives a $O(n^3)$ computational complexity, but no additional storage beyond the $O(n^2)$ required for the initial $A$.

For the cases of partial and complete pivoting, the major difference is that rows (or columns for complete) will be swapped according to the maximal element in a column or overall in the active part of the matrix respectively. This is done by first finding the maximum by comparing the magnitude of each element to the currently defined maximum in a for-loop (or a nested loop for complete) and then redefining the maximum if true. These permutations are trapped in corresponding arrays based on the index of the element swapped with the current element's index, for application in more loops later by using a temporary

```
for(int j=0; j<n-1; j++){
    if(A[j][j]==0){//check for cases when factorization fails.
        cout<<"pivot element is 0. failed to complete LU factor. "<<endl;
        return;
    }

    for(int i=j+1; i<n; i++){//row iterator for active matrix
        A[i][j] /= A[j][j];

        for(int k=j+1;k<n;k++){//column iterator for active part of matrix
            A[i][k]= A[j][k]*A[i][j]*(-1.0) +A[i][k];
```

Figure 1: Triple-Nested For-loop Structure for In-Place LU Factorization (No pivoting)

variable. Since this is done using reads and writes exclusively, it does not change the computational complexity of finding LU nor the overall order of storage as two additional arrays of size $n-1$ and a temp variable is $O(n) < O(n^2)$.

Once L and U have been found and stored in A, this 2D array and a potentially permuted $b$, with elements swapped according to the P and Q arrays as described above, are passed to the solver to do the forward and backward triangular solves. This is done in two different nested for-loops as in Section 3, resulting in $O(n^2)$ computational complexity. The only difference is that $n$ additional division happen for $Ux = y$ since the diagonal elements are not all 1. $O(n)$ additional storage is also created in order to house the solution $x$ as well as the intermediary solution $y$.

Additional utility methods were also created in order to compute the vector and matrix norms necessary to do accuracy and stability checking, each in $O(n)$ and $O(n^2)$ complexity respectively. A method was created to compute L*U in-place as well for use with the growth factor. This implements a triple-nested for-loop structure as well, giving $O(n^3)$ computations. However, because this algorithm happens in place and no additional storage beyond the $n^2$ for the output M is created (L and U are not explicitly formed), a variable indicating the minimum of the index of the two outer loops is needed in order to bound the innermost loop so that the L and U matrices do not start multiplying their own elements together.

# 5 Experimental Design and Results

## 5.1 Task 1: Structure

### 5.1.1 Diagonal

Eight different distinct matrix structures were investigated. The first type was a strictly diagonal matrix with increasing (or decreasing) positive elements:

$$\begin{bmatrix} 1 & & & \\ & 2 & & \\ & & 3 & \\ & & & 4 \end{bmatrix} \text{ or } \begin{bmatrix} 4 & & & \\ & 3 & & \\ & & 2 & \\ & & & 1 \end{bmatrix}$$

For this type of matrix, we should expect the L factor to be the identity, and the U factor to be the original matrix. This is because when applying the Gauss transform, there is nothing under the diagonal to eliminate, meaning all of the subdiagonal $\lambda$ in L will be 0. Additionally, if we only apply partial pivoting, since the algorithm only checks the elements below the diagonal in the same row for potentially larger pivot elements, we should not expect any change for either ordering. Therefore all the permutation matrices will be the identity and the array P will contain the same values as the index (meaning it is the identity). However, in the case of complete pivoting with the diagonal element at $i, i$ being $i + 1$, we should expect the output to be the same as the decreasing ordering, since it will find the maximal element (the last one) and swap the first one. This leads to a pattern where indices $0 \to n/2$ will be $n - 1 \to n/2$ inside each of the P and Q permutation arrays and the reverse for $n/2 + 1 \to n - 1$.

When the algorithm was applied to the decreasing ordering with $n = 5$, the matrix came back identical, which makes sense since the 1's of the L matrix are not stored. The permutations also came back identical, indicating no rows and columns were swapped. This confirms our predictions. The growth factors for the 1-norm, max-norm, and F-norm all also came back as 1, since all the elements were positive, applying absolute value did nothing, and since $U = A$ and $L = I$, $\frac{||U||}{||A||} = \frac{||A||}{||A||} = 1$. This was also the case for the cases of no pivoting and partial pivoting for the increasing ordering with $n = 5$. However, for complete pivoting, $P = [4, 3, 2, 3] = Q$ meaning row and column 1 swap with row 5 and column 5, row/column 2 with row/column 4, and the rest identical, as predicted. The decreasing order matrix was returned in this case, as the permutation arrays would suggest. However, since ordering of elements is irrelevant to the norms, the growth factors all also came back as 1.

### 5.1.2 AntiDiagonal

The second matrix type to be considered is the antidiagonal matrix with increasing (or decreasing) positive integer elements:

$$\begin{bmatrix} & & & 1 \\ & & 2 & \\ & 3 & & \\ 4 & & & \end{bmatrix} \text{ or } \begin{bmatrix} & & & 4 \\ & & 3 & \\ & 2 & & \\ 1 & & & \end{bmatrix}$$

Although the matrix is obviously nonsingular since the columns are independent, we would expect that an LU factorization cannot be computed directly, since the diagonal elements (and in particular the first diagonal element, which remains unchanged in U) are almost all 0 no matter the choice of n. This implies that U is nonsingular with linear dependent columns since the first one at least is the zero vector, and so the LU factorization doesn't exist. However, if we allow for pivoting, we can apply row permutations, where $n - 1 - i$ swaps $i$ for $i = 0, .., n/2$, to get the matrix into the same form as in 5.1.1, which then should give identical results.

As anticipated, the algorithm outputs the "pivot element is 0. failed to complete LU factor." error message when no pivoting is applied to the matrix with $n = 5$, since proceeding to the next step would force the algorithm to divide by 0. Growth factors in this case are therefore irrelevant. For the decreasing ordering were the largest element is in the first column, both partial and complete pivoting gave back the same result, the decreasing ordering for diagonal with the same permutations of $P = [4, 3, 2, 3]$ and $Q = [0, 1, 2, 3] = I$, indicating no columns were changed, and rows 1 and 5 as well as rows 2 and 4 were swapped, as in the general pattern described above. This was the same for the opposite ordering for partial pivoting, but for complete, the algorithm swapped columns only, giving $Q = [4, 3, 2, 3]$ and $P = [0, 1, 2, 3] = I$, since swapping the rows and then swapping the rows opposite again nullifies the previous permutation. The growth factors were all 1 once again, since after permutation, multiplication is identical to the diagonal case.

### 5.1.3 X Nonzero Pattern

The third type of matrix to be investigated is the sum of the two previous types of matrices, given by the X pattern below:

$$\begin{bmatrix} 1 & & & 1 \\ & 2 & 2 & \\ & 3 & 3 & \\ 4 & & & 4 \end{bmatrix} \text{ or } \begin{bmatrix} 4 & & & 4 \\ & 3 & 3 & \\ & 2 & 2 & \\ 1 & & & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & & & 4 \\ & 2 & 3 & \\ & 2 & 3 & \\ 1 & & & 4 \end{bmatrix} \text{ or } \begin{bmatrix} 4 & & & 1 \\ & 3 & 2 & \\ & 3 & 2 & \\ 4 & & & 1 \end{bmatrix}$$

One of the assumptions for the LU factorization to exist is that A is nonsingular, since LU gives the effect of $A^{-1}$ and $A^{-1}$ can only exist if the original A has linearly independent rows and columns.[1] In this case, it is easy to see that no matter which combination of orderings from the above types we add together, we will always get a matrix in which both the columns and rows are linearly dependent. Therefore, it does not matter how much we pivot by swapping the rows and columns. We cannot permute A in order to get a matrix with a unique solution to Ax=b. Hence we predict that the algorithm will fail for all pivoting strategies and this is exactly what happened in practice. For n=5, there ended up being a situation in which the final 2x2 active matrix was all zeroes, meaning that there where not any suitable pivots and therefore factorization could not proceed, as seen below:

$$\text{5x5 array} = \begin{bmatrix} 1 & & & & 5 \\ & 2 & & 4 & \\ & & 6 & & \\ & 1 & & & \\ 1 & & & & \end{bmatrix}$$

Additionally, for complete pivoting with n odd, since $2 * ((n-1)/2+1) > n$, the middle element of the X-pattern will always be shifted to the upper leftmost

position on the first pivoting step. This unfortunately does not help to eliminate the issue of linear dependence among the rows and columns though.

However, that does not mean that the X-pattern in general cannot be factored into L and U. If instead of adding the increasing/decreasing diagonal and antidiagonal matrices above, we fill the X pattern with random positive integers between 1 and 9:

$$\begin{bmatrix} 3 & 0 & 0 & 0 & 7 \\ 0 & 2 & 0 & 6 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 5 & 0 & 8 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We then get the following for the different strategies:

$$\text{no pivoting} = \begin{bmatrix} 3 & 0 & 0 & 0 & 7 \\ 0 & 2 & 0 & 6 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 2.5 & 0 & -7 & 0 \\ 0.333 & 0 & 0 & -0 & -1.333 \end{bmatrix}, \gamma_\epsilon = \begin{bmatrix} \text{1-norm} : 2 \\ \text{Max} : 2.07692 \\ F : 1.80998 \end{bmatrix}$$

$$\text{partial} = \begin{bmatrix} 3 & 0 & 0 & 0 & 7 \\ 0 & 5 & 0 & 8 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0.4 & 0 & 2.8 & 0 \\ 0.333 & 0 & 0 & 0 & -1.333 \end{bmatrix}, \gamma_\epsilon = \begin{bmatrix} \text{1-norm} : 1 \\ \text{Max} : 1 \\ F : 1.03223 \end{bmatrix}$$

$$\text{complete} = \begin{bmatrix} 8 & 0 & 5 & 0 & 0 \\ 0 & 7 & 0 & 0 & 3 \\ 0.75 & 0 & -1.75 & 0 & 0 \\ 0 & 0 & -0 & 1 & 0 \\ 0 & 0.143 & -0 & 0 & 0.571 \end{bmatrix}, \gamma_\epsilon = \begin{bmatrix} \text{1-norm} : 1 \\ \text{Max} : 1 \\ F : 1.06684 \end{bmatrix}$$

where partial and complete pivoting applied significant permutations on every step according to maximal elements. This seems to indicate that the issue was not the X pattern in general, but the fact that we were adding the diagonal and antidiagonal matrices with identical elements together.

### 5.1.4 Unit Lower Triangular

The fourth matrix structure to be investigated is a unit lower triangular structure with the magnitude all of the subdiagonal elements strictly less than 1, for example:

$$\begin{bmatrix} 1 & & & \\ 0.4 & 1 & & \\ -0.1 & 0.3 & 1 & \\ 0.7 & 0.8 & 0.2 & 1 \end{bmatrix}$$

Since the condition that the L factor be unit lower triangular is already met, if we want to create another factor U such that A=LU, then the obvious choice to take U as the identity. Therefore, we should expect the algorithm to output the same matrix that we input and this was exactly the case for all strategies. Since the diagonal is all ones, the $\lambda$'s in the Gauss transform will exactly be the subdiagonal elements of A, so no change will happen. This also makes sense for complete and partial pivoting. Because the magnitude of all of the subdiagonal elements is less than the diagonal elements, we do not need to apply any permutations to get the maximal element on the diagonal. Therefore, all the P and Q are also the identity. Additionally, because A=LU=L, the growth factors were all 1 since A and L are identical and U is the identity.

### 5.1.5 Lower Triangular

The fifth matrix structure investigated is a lower triangular structure, but this time, the magnitude of the elements was restricted to be greater than 1 and positive, for instance:

$$\begin{bmatrix} 2 & & & \\ 3 & 2 & & \\ 4 & 3 & 2 & \\ 5 & 4 & 3 & 2 \end{bmatrix}$$

This time, without pivoting, we should still expect the structure to be preserved since there are not any above diagonal elements that can be updated. Therefore, each subdiagonal will just be the original value divided by the value along the diagonal in the corresponding column. This means that since U holds the upper triangular portion, that since the diagonal is not updated, it will be $\lambda_{ii} * I$ where $\lambda_{ii}$ is the value on the diagonal. This was confirmed, resulting in growth factors of 1 for all norms, since division by 2 can be represented and then reconstructed finitely.

However, for the cases of partial and complete pivoting with the above structure, since the element of greatest magnitude at step 1 is in the lower left, it will always swap rows with row 1, thereby destroying the lower triangular structure. The permutation for partial pivoting is $P = [4, 4, 4, 4]$ while the permutations for complete pivoting were $P = [4, 1, 2, 3]$ and $Q = [0, 2, 3, 4]$. This makes sense for partial pivoting because after permutation, the last element of each column has a zero so therefore since there is less positive to counteract the negative in the inverse Gauss transform, it will have the largest magnitude. For complete pivoting, the max magnitude is in the lower left so no column permutation is required but after successive eliminations, the max magnitude ends up being in the next column over due to the initial zeroes leading to permutation. The growth factors were all close to 1 for these strategies as well.

### 5.1.6 Tridiagonal

The next type of matrix to be factored is a tridiagonal matrix, where only the diagonals directly above and directly below the main diagonal are nonzero:

$$\begin{bmatrix} a_1 & c_1 & & \\ b_2 & a_2 & c_2 & \\ & b_3 & a_3 & c_3 \\ & & b_4 & a_4 \end{bmatrix}$$

The first tridiagonal case is a matrix that is diagonally dominant, meaning that $a_i > b_i$ and $a_i > c_i$ by both rows and columns. For such a matrix, we predict that L and U will also be banded for the cases of no pivoting and partial pivoting, since all of the elements below the diagonal will be less than the diagonal element, so there will not be any valid pivots. The zeros below the diagonal will simply remain 0 in L while upper diagonal positions in U will not be updated since those elements will multiply by the 1's of the identity in the inverse Gauss Transform. The upper diagonal zeros will also not be updated due to the lower triangular structure of the Gauss Transform. This is indeed what our experiments showed. However, with complete pivoting, there were nontrivial permutations that destroyed the structure, since the $a_i$'s were not constrained to be identical or decreasing order. The permutations were $P = [1, 4, 4, 4]$ and $Q = [1, 4, 4, 4]$, but this does not represent a general trend of swapping the final row because the different random matrix generated gave a different set of permutations, $P = [0, 3, 3, 4]$ and $Q = [0, 3, 3, 4]$. This does indicate that since the maximal element will always be on the diagonal, the P and Q swaps will both be identical. The growth factors for all three strategies were all approximately 1 for all 3 norms.

$$\begin{bmatrix} 72 & 5 & 0 & 0 & 0 \\ 3 & 24 & 1 & 0 & 0 \\ 0 & 1 & 8 & 7 & 0 \\ 0 & 0 & 5 & 40 & 6 \\ 0 & 0 & 0 & 2 & 16 \end{bmatrix} \rightarrow \begin{bmatrix} 72 & 0 & 5 & 0 & 0 \\ 0 & 40 & 0 & 6 & 5 \\ 0.04167 & 0 & 23.7917 & 0 & 1 \\ 0 & 0.05 & 0 & 15.7 & -0.25 \\ 0 & 0.175 & 0.04203 & -0.0668 & 7.066 \end{bmatrix}$$

If we impose the opposite condition, that the elements in the main or upper diagonal be strictly smaller than any element on either of lower diagonal, partial row pivoting will be required at every step to produce a stable algorithm. We should expect the permutations to be $P = [1, 2, 3, 4]$ for $i = 0 \rightarrow 3$ indicating that each row swapped with the one underneath of it, and that is exactly what the experiment showed. U should remain banded but with the diagonal containing the new maximal elements from the lower diagonal and with the former main and upper diagonals shifted up by 1 row respectively. L should have a completely different structure, however. Because of the row swap on every step, the results of eliminations of the subdiagonals on each step will push all of the formerly lower diagonal elements to the bottom, resulting in L being the identity plus row n-1 of nonzero elements, as seen below:

$$\begin{bmatrix} 5 & 8 & 0 & 0 & 0 \\ 32 & 4 & 8 & 0 & 0 \\ 0 & 48 & 6 & 6 & 0 \\ 0 & 0 & 32 & 4 & 2 \\ 0 & 0 & 0 & 8 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 32 & 4 & 8 & 0 & 0 \\ 0 & 48 & 6 & 6 & 0 \\ 0 & 0 & 32 & 4 & 2 \\ 0 & 0 & 0 & 8 & 1 \\ 0.15625 & 0.1536 & -0.06787 & -0.0813 & 0.217 \end{bmatrix}$$

### 5.1.7 "Almost" Lower Triangular

The next matrix structure to be investigated is a special structure, in which the diagonal and last column are all 1's, while all of the subdiagonal elements are -1, as seen here for n=5:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}$$

We should not expect the L factor to be any different for the strategies of no pivoting and partial pivoting, because the lower half of A is already unit lower triangular, with elements smaller than 1 below a diagonal of all 1's. Therefore, there should not be any change in the rows and hence P should be the identity, which was confirmed as experimentation gave $P = [0, 1, 2, 3]$ for all strategies.

However, as we see if 2.8, because the application of the inverse Gauss transform takes the opposite sign of the subdiagonal column elements over the diagonal, the $\lambda$'s will all be 1.[3] Hence when the last column is applied to the transform, the current values in the last column of the active part of the matrix will be effectively multiplied by 2. This results in the U factor being the identity, with the last column being replace by $2^i$ were $i$ is the row index.

As a result of this, we can see a pattern for the Q permutation, were after the first step, the final column will swap with the current pivot column of the active matrix as 2¿1, resulting in $Q = [0, 4, 4, 4]$ for n=5. The identical pattern was confirmed for n=8. After step 2, this will change to -2 on the diagonal for U as a result of the eliminations, with 1 in the upper diagonal after the swaps. This will also change L since elimination of the 2's and then -2's below the diagonal swaps the sign to 1 instead of -1, as we need -1 in the Gauss transform. This was confirmed for both n=5 and n=8, as seen below:

$$\text{complete data struct} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 2 & 1 & 0 & 0 \\ -1 & 1 & -2 & 1 & 0 \\ -1 & 1 & 1 & -2 & 1 \\ -1 & 1 & 1 & 1 & -2 \end{bmatrix}$$

As the vectors below show, for n=8, the growth factors for partial pivoting and without pivoting are very large compared to the size of n, which was also true

for n=5. This can attributed to the accumulation that occurs in the final column of U, which increases with n. This seems to suggest that the growth factor in these cases is a function of the size of the matrix and hence is proportional to n. However, for complete pivoting, the growth factor remained less than 3, indicating that it is perhaps the stability of the algorithms above that caused the huge growth factor, and not necessarily the ill conditioning of the problem.

$$\gamma_\epsilon(\text{w/out and partial}) = \begin{bmatrix} \text{1-norm} : 62.75 \\ \text{Max} : 32.75 \\ F : 44.826 \end{bmatrix}$$

$$\gamma_\epsilon(\text{complete}) = \begin{bmatrix} \text{1-norm} : 2.75 \\ \text{Max} : 2.75 \\ F : 2.49185 \end{bmatrix}$$

### 5.1.8 Symmetric Positive Definite

The final class of matrix to be investigated is a symmetric positive definite matrix, generated by taking the matrix-matrix product of $A = \tilde{L}\tilde{L}^T$ where L is a lower triangular matrix. For instance, with n=5:

$$\begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 & 0 \\ 8 & 4 & 5 & 0 & 0 \\ 1 & 5 & 6 & 1 & 0 \\ 5 & 6 & 8 & 1 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 16 & 12 & 32 & 4 & 20 \\ 12 & 13 & 32 & 13 & 27 \\ 32 & 32 & 105 & 58 & 104 \\ 4 & 13 & 58 & 63 & 84 \\ 20 & 27 & 104 & 84 & 162 \end{bmatrix}$$

For this type of matrix, since by definition it is nonsingular, pivoting is not required in order to get a stable result. Hence, we can determine a relationship between the factors L, U and $\tilde{L}$ using a strategy without pivoting. Since the Gauss transform will eliminate the elements below the diagonal by dividing by the corresponding diagonal element, we should expect the L factor should be $L = \tilde{L}D$, where D is a diagonal matrix composed of the multiplicative inverses of the diagonal elements of $\tilde{L}$. Additionally, U should be $U = (\tilde{L}D^{-1})^T$, where $D^{-1}$ is the inverse of the diagonal matrix D containing the reciprocals of all of the diagonal elements, as application of the Gauss transform will strip the upper elements of most of accumulated effect of the original matrix multiplication to get symmetric positive definiteness, apart from multiplication by the original diagonal. This can be seen in the example below for the random lower triangular matrix, n=5:

$$\tilde{L}D = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 & 0 \\ 8 & 4 & 5 & 0 & 0 \\ 1 & 5 & 6 & 1 & 0 \\ 5 & 6 & 8 & 1 & 6 \end{bmatrix} \begin{bmatrix} 0.25 & & & & \\ & 0.5 & & & \\ & & 0.2 & & \\ & & & 1 & \\ & & & & 0.17 \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ 0.75 & 1 & & & \\ 2 & 2 & 1 & & \\ 0.25 & 2.5 & 1.2 & 1 & \\ 1.25 & 3 & 1.6 & 1 & 1 \end{bmatrix} = L$$

$$(\tilde{L}D^{-1})^T = \left( \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 & 0 \\ 8 & 4 & 5 & 0 & 0 \\ 1 & 5 & 6 & 1 & 0 \\ 5 & 6 & 8 & 1 & 6 \end{bmatrix} \begin{bmatrix} 4 & & & & \\ & 2 & & & \\ & & 5 & & \\ & & & 1 & \\ & & & & 6 \end{bmatrix} \right)^T = \begin{bmatrix} 16 & 12 & 32 & 4 & 20 \\ & 4 & 8 & 10 & 12 \\ & & 25 & 30 & 40 \\ & & & 1 & 1 \\ & & & & 36 \end{bmatrix} = U$$

## 5.2 Task 2: General Trends

### 5.2.1 LU Factorization Order of Computations

In order to confirm the $O(n^3)$ computations for the LU factorization algorithm discussed in section 4, the factorization was run for randomly generated symmetric positive definite matrices without pivoting for $n = 100 \rightarrow 1,000$ by incremental increases of 50. The symmetric positive definite structure was chosen because by definition the matrix is nonsingular and factorization will thus automatically succeed without pivoting. The algorithm was then timed with the same method as previously implemented for the DFT. As Figure 2 shows, when plotting the size of the matrix against the time for the function to run, a polynomial of best-fit, order 3, intersects every point, as shown by $R^2 = 1$, thus confirming the theorized order of computations.

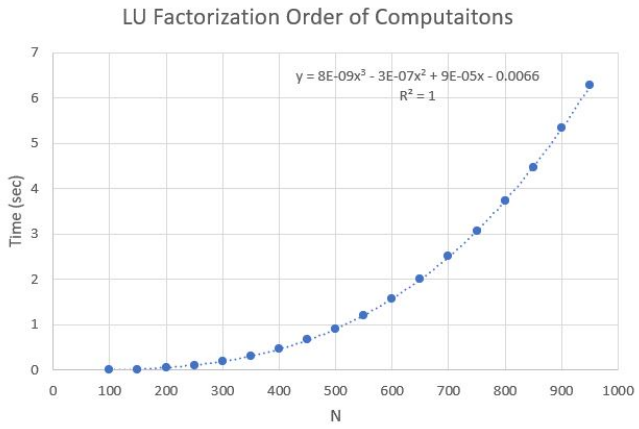


Figure 2: Empirical Validation of $O(n^3)$ of LU factorization

### 5.2.2 Without Pivoting

In order to further validate the correctness of the entire algorithm for factorization and solving the full system $Ax = b$, the three different factorization strategies will be tested against different structured problems sampled from the above section for sizes $n = 5 \rightarrow 50$. For each size $n$, 1000 randomly generated

matrices of floats satisfying the structure will be created, as well as 1000 random floating point solutions $x$. Then $b$ will be found by performing the matrix-vector multiplication $Ax$, and then the system will be solved by the algorithm using LU as well as the forward and backward solves. This will allow comparison between the true and computed solution. Then, once the solutions and error metrics have been computed, the results of the 1000 samples will be averaged and plotted against the size.

In order to test the strategy without pivoting, the diagonally dominant by row matrix structure was chosen, as this structure is nonsingular by design and is guaranteed to succeed without pivoting.

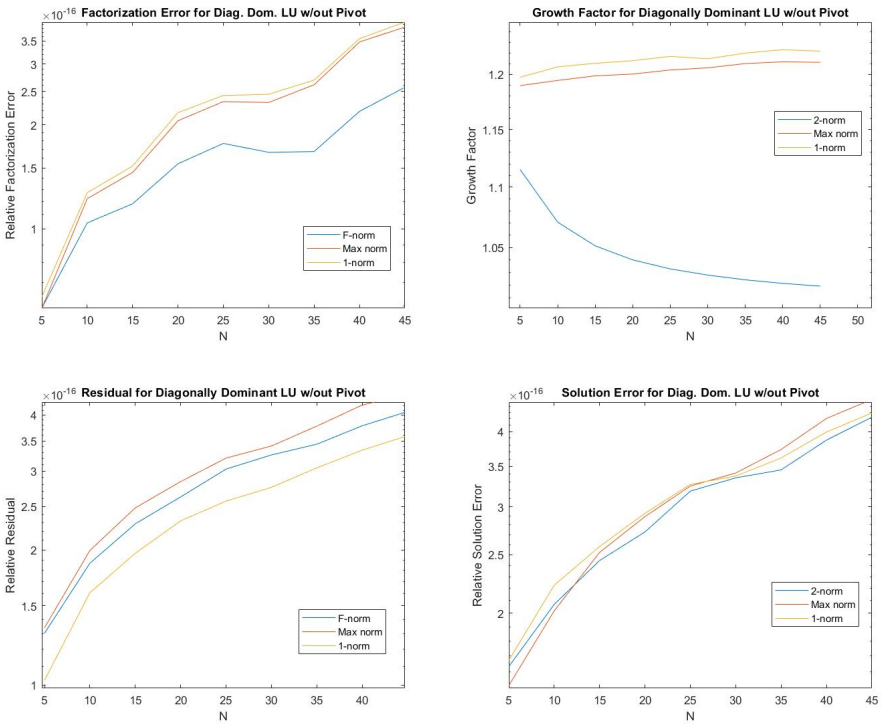


Figure 3: Empirical Validation of Correctness for Strategy without Pivoting

As Figure 3 shows, although the error metrics tend to increase for increasing $n$, they all are close to $10^{-16}$ or approximately 0 for floating point, indicating that the strategy is able to correctly determine the factorization, that the computed solution closely matches the correct solution, and that computed solution actually satifies the matrix-vector multiplication as intended for all three discrete norms. Additionally, we can see that the problem remains very stable as N increases, as the computed growth factor remains very close to 1 for all of the norms

### 5.2.3 Partial and Complete Pivoting

In order to further validate the correctness of the partial and complete pivoting strategies, random matrices were generated with the same diagonally dominant structure as above. Then, randomly generated permutations were applied to the matrix, thereby ensuring that the resulting matrix actually required pivoting to be solved, while still remaining nonsingular. One-thousand random sample for each of the sizes $n = 5 \rightarrow 50$ were generated and the metrics averaged by size as above for all three norms.

In order to first test that the generated matrices actually required pivoting in order to be solved, the strategy of factorization without pivoting was applied to a identically generated set using the max norm.

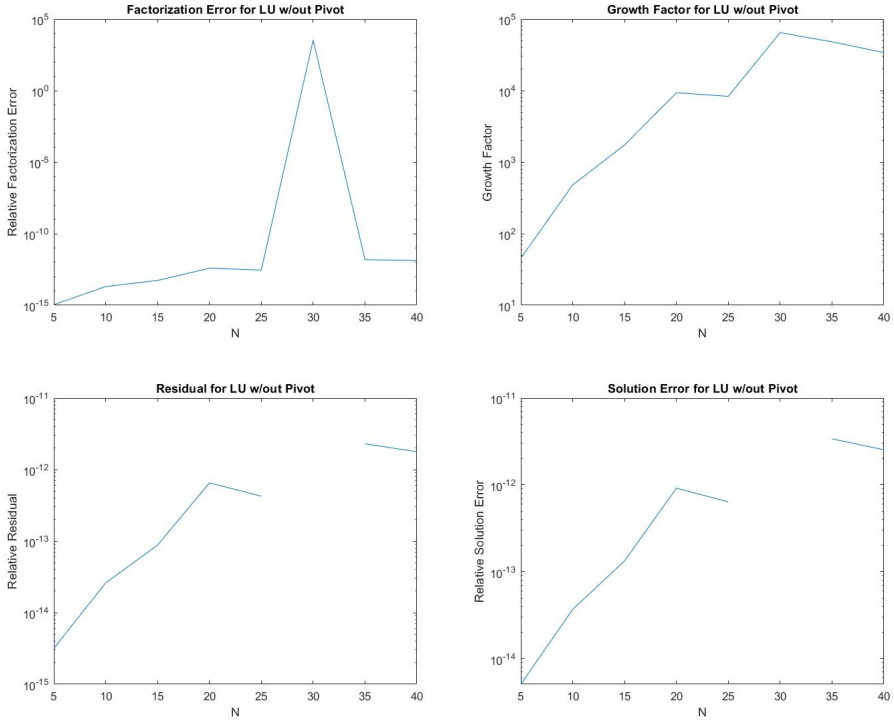


Figure 4: Empirical Validation of Need for Pivoting

As Figure 4 shows, while in many cases the factorization was able to proceed without much significant error, for several different cases in the samples for n=30, the factorization completely broke and was unable to continue due to the zeros introduced on the diagonal by the elimination. This is represented by the huge breaks in the residual and solution error plots, as no solution was able to be found at all, and relative factorization error of over 1000, since only part of the matrix was factorized. Additionally, we see that the growth factor, even

when there is not much actual error for n=5, is around 100 and increases to just under 100,000 for n=40, illustrating the need for pivoting.

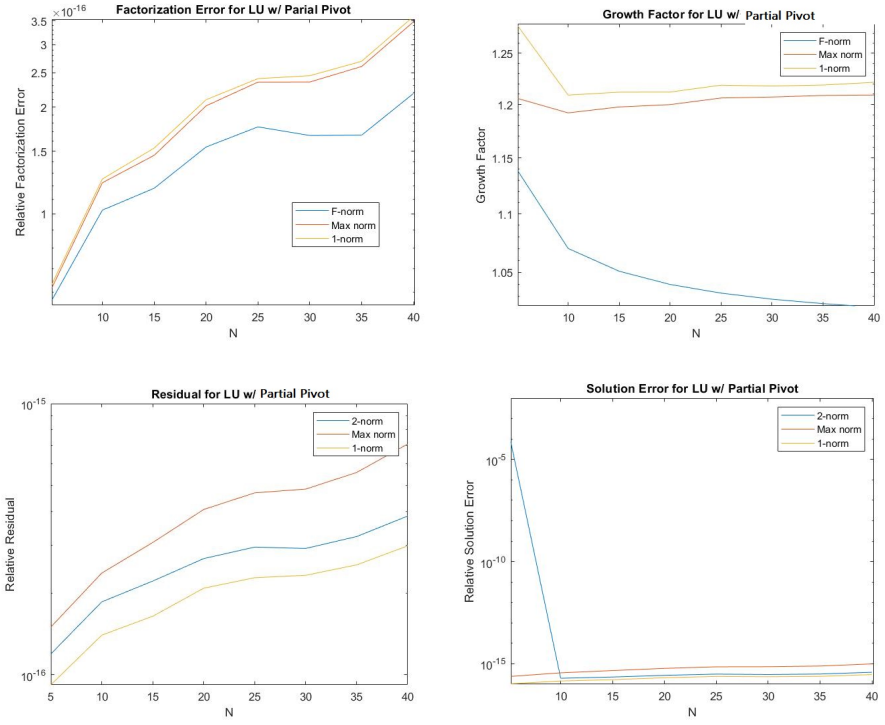


Figure 5: Empirical Validation of Correctness of Partial Pivoting

However, when pivoting of both types was implemented, the system was able to be successfully factored and solved for all 1000 samples for all sizes 5 to 50. As Figures 5 and 6 illustrate, all error metrics remained very close to machine epsilon for double of $10^{-16}$ and the growth factor stayed close to 1 for all three norms. One discrepancy is however, is in the solution error for partial pivoting where n=5, which is much higher than expected. This may be attributed to cancellations during the backward and forward solves that involved the subtractions of similar numbers that were subsequently chopped during division. This was probably a single example and since the averages are taken, it outweighed the $10^{-15}$ solution errors for n=5 and therefore boosted the error. It is near $10^{-6}$ which is still relatively small and should not be included in a determination of the correctness of the partial pivoting strategy, but in a failure of the random generation of the matrix A.

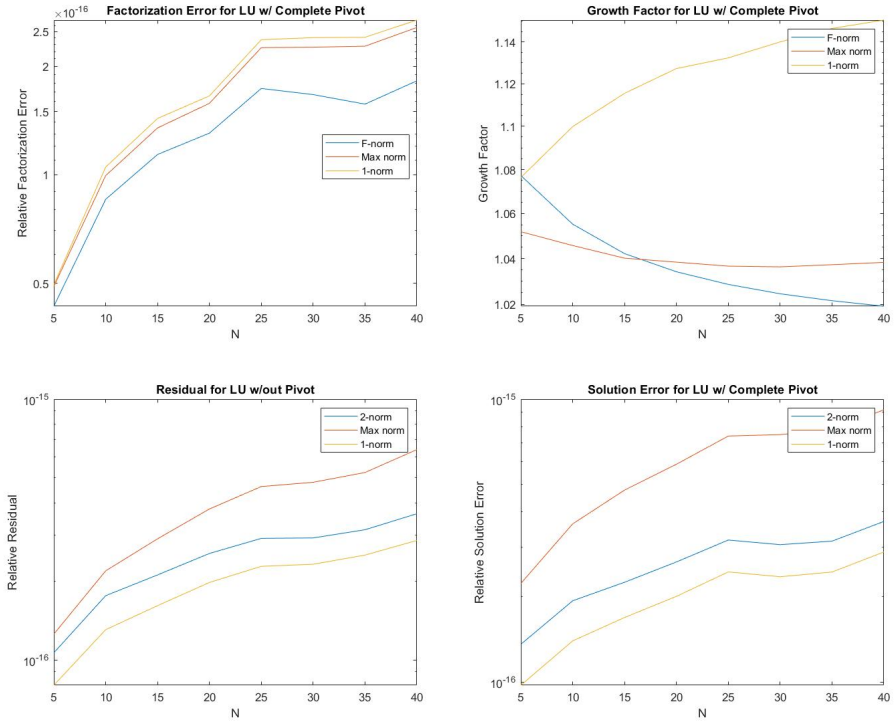


Figure 6: Empirical Validation of Correctness of Complete Pivoting

# 6 Conclusion

Overall, we can see that the general trend is an increase in the residual error, the solution error, and the factorization error on average for all norms as the size of the matrix A increases. Additionally, we confirm that for stable problems, the growth factor tends to remain fairly constant, confirming theory, while for problems that are ill conditioned and/or combined with an overall unstable algorithm, the growth factor tends to increase with n or worse, increase exponentially, such as with the strategy without pivoting applied to the randomly permuted diagonally dominate matrix.

Predictions about the factorization of matrices with particular structure were also able to be empirically validated. The algorithm with complete pivoting once again turned out to be the most robust, as it was able to produce a low growth factor bound for all classes of problems, except for the X-pattern matrix, where the inherent nonsingularity of both rows and columns was not able to be overcome by swapping rows and columns. Hence, it was shown that as long as A satisfies the nonsingularity assumption and $A^{-1}$ actually exists, complete pivoting can stably produce a result. On the other hand, we showed that for

for the algorithm without pivoting, having A be nonsingular is not enough and total diagonal dominance is required in order to ensure a result is obtained.

# 7 Program Files

The algorithms were compiled in a single C++ source file (.cpp) using "g++ -o lu LUfactor.cpp" using the Powershell on Windows, after being written on Notepad++. The timeing for $O(n^3)$ was done on the local machine. Graphs were produced using either excel for trendline or matlab.

# References


[1] www.math.fsu.edu/ gallivan/courses/FCM2new/Locked/Sets/set4.pdf

[2] www.math.fsu.edu/ gallivan/courses/FCM2new/Locked/Sets/set6.pdf

[3] www.math.fsu.edu/ gallivan/courses/FCM2new/Locked/Solutions/solhw2.pdf