Programming Assignment 2

Jenny Petrova

October 16th, 2024

1 Executive Summary

This report presents a method to efficiently evaluate the lower-upper (LU) factorization of a nonsingular matrix $A \in \mathbb{R}^{n\times n}$. This factorization is achieved by changing the basis of the full-rank matrix A to an upper triangular form U. Nonsingular transformations T_i^{-1} are applied such that

$$T_i^{-1}A \to U$$

where $U \in R^{n\times n}$ and nonsingular, and the product of the transformations T_i form the unit lower triangular matrix $L \in \mathbb{R}^{n\times n}$. Gaussian transformations and permutation arrays (P_r, P_c) are applied to achieve the transformation

$$P_r A P_c = LU$$

where the indices of $P_r \in \mathbb{R}^{n\times n}$ indicate the row order of A, the indices of $P_c \in \mathbb{R}^{n\times n}$ indicate the column order of A, $L \in \mathbb{R}^{n\times n}$ is a unit lower triangular matrix, and $U \in \mathbb{R}^{n\times n}$ is a nonsingular upper triangular matrix.
The factorization routines avoid the usage of high-level matrix libraries; solutions are developed by performing in-place updates to the matrix A, minimizing memory usage.
The accuracy and efficiency of the implemented LU factorizations are evaluated across various test matrices, included randomly generated matrices and specific structured matrices to highlight numerical stability issues.

2 Statement of the Problem

Computing the inverse of a matrix is computationally expensive, costing $O(n^3)$ for an $n \times n$ matrix. Direct inversion can lead to potential numerical instability, including round-off errors, especially for large or ill-conditioned matrices. Given a linear system of equations $Ax = b$, for a matrix $A \in \mathbb{R}^{n\times n}$ and a vector $b \in \mathbb{R}^{n\times 1}$, where A^{-1} exists, we want to implement an algorithm to efficiently solve for the vector x. We apply LU factorization to solve the system $Ax = b$:

1. Decompose A into LU, where L and U are stored in A

2. Solve $(LU)x = b$ through forward substitution (solve $Ly = b$ for y) and backward substitution (solve $Ux = y$ for x)

By transforming A to a triangular matrix, we preserve efficiency and stability.

3 Description of the Mathematics

Assume $A \in \mathbb{R}^{n\times n}$ where A is nonsingular, and $b \in \mathbb{R}^{n\times 1}$. Let $\alpha_{ij} \in A$ denote the (i, j) entry of A. Assume $L \in \mathbb{R}^{n\times n}$ is a unit lower triangular matrix and $U \in \mathbb{R}^{n\times n}$ is an upper triangular matrix.

3.1 Method for Factorization of A

We change the basis of the problem $Ax = b$ using a nonsingular transformation matrix T^{-1}:

$$\begin{aligned} M^{-1}(Ax) &= M^{-1}b \\ (M^{-1}A)x &= c \\ Ux &= c \end{aligned}$$

where $T^{-1}A = U$. Many T_i^{-1} are possible, and T^{-1} is constructed as the product of Gaussian transformations, which eliminate the sub-diagonal elements of A, one column at a time. Each T_i^{-1} corresponds to a step i in the Gaussian elimination process, where the sub-diagonal elements of column i of the current transformed matrix are eliminated, and all other elements of the current matrix are updated. This results in a rank-1 update of the lower submatrix of order $n - 1$. After $n - 1$ transformations,

$$T_{n-1}^{-1} \dots T_2^{-1} T_1^{-1} A = U, \tag{1}$$

therefore

$$A = T_1 T_2 \dots T_{n-1} U \tag{2}$$

where

$$T_1 T_2 \dots T_{n-1} = L. \tag{3}$$

Thus the matrix A is decomposed into LU, stored in place of the original entries of A.

3.1.1 Gaussian Elimination

Consider the matrix A:

$$A = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix}$$

We want to reduce A to a row echelon form U by Gaussian elimination, keeping track of the multipliers used to introduce the leading coefficients and the multipliers used to introduce the zeroes below the leading coefficients. We first compute the multiplier $\lambda_{j1} = \alpha_{ji}/\alpha_{11}$ for the first diagonal entry. Apply the first Gaussian transformation T_1^{-1} to eliminate the elements below α_{11} in the first column:

$$T_1^{-1}A = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} - \lambda_{21}\alpha_{11} & \alpha_{22} - \lambda_{21}\alpha_{12} & \alpha_{23} - \lambda_{21}\alpha_{13} \\ \alpha_{31} - \lambda_{31}\alpha_{11} & \alpha_{32} - \lambda_{31}\alpha_{12} & \alpha_{33} - \lambda_{31}\alpha_{13} \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ 0 & \alpha'_{22} & \alpha'_{23} \\ 0 & \alpha'_{32} & \alpha'_{33} \end{bmatrix}.$$

After this transformation, the first column below α_{11} becomes zero, and α'_{ij} are the updated matrix elements after applying the first Gaussian transformation. Next, we eliminate the elements below α'_{22}. We compute the multiplier $\lambda_{j2} = \alpha'_{j2}/\alpha'_{12}$ for the second column and apply a second Gaussian transformation T_2^{-1}:

$$T_2^{-1}T_1^{-1}A = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ 0 & \alpha'_{22} & \alpha'_{23} \\ 0 & \alpha'_{32} - \lambda_{32}\alpha'_{22} & \alpha'_{33} - \lambda_{32}\alpha'_{23} \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ 0 & \alpha'_{22} & \alpha'_{23} \\ 0 & 0 & \alpha''_{33} \end{bmatrix} = U,$$

where α''_{ij} is the updated matrix element after the second Gaussian transformation. Each transformation is essentially a rank-1 update of the lower right submatrix of A of order $n-1$. We transform A to upper triangular matrix form and the mulipliers λ'_{ij} for each transformation form the elements of L. We have

$$T_2^{-1}T_1^{-1}A = U. \tag{4}$$

Then $A = T_1T_2U$ and

$$L = T_1T_2. \tag{5}$$

Therefore for a matrix $A \in \mathbb{R}^{3\times 3}$, we apply $3-1=2$ Gaussian eliminations to factorize A into L and U. Matrix A becomes

$$A = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \lambda_{21} & 1 & 0 \\ \lambda_{31} & \lambda'_{21} & 1 \end{bmatrix} \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ 0 & \alpha'_{22} & \alpha'_{23} \\ 0 & 0 & \alpha''_{33} \end{bmatrix}.$$

3.1.2 Pivoting

It is important to note that A is nonsingular does not imply $A = LU$. A problem arises if we encounter a zero pivot α_{ii}, since the algorithm will break down by attempting to divide by zero. If a pivot element is near-zero, then the value of the multiplier will be very large, which can lead to rounding errors and loss of numerical stability. To avoid these potential problems, we consider the following lemmas:

Lemma 1. *If A^{-1} exists then there exists a product of elementary row permutation matrices $P = P_{n-1}\dots P_1$, a unit lower triangular matrix L, and an upper triangular matrix U such that*

$$PA = LU.$$

Lemma 2. *If A^{-1} exists then there exists a product of elementary row permutation matrices $P = P_{n-1} \dots P_1$, a product of elementary column permutation matrices $Q = Q_{n-1} \dots Q_1$, a unit lower triangular matrix L, and an upper triangular matrix U such that*

$$PAQ = LU.$$

To guarantee the existence of and stability of the factorization, we perform pivoting; i.e. interchanging the matrix rows and columns. In row pivoting ("partial" pivoting), we swap rows to move the element largest in magnitude (in the current column) to the current pivot position. In row and column pivoting ("complete" pivoting) we swap rows and columns to move the element largest in magnitude (in the current submatrix) to the current pivot position.

4 Description of the Algorithm and Implementation

The algorithms are implemented in Python, on the PyCharm integrated development environment. A driver code holds the routines, which are constructed as defined Python functions. A solver code calls the routines from the driver code, and runs several tests to assess the corectness of the routines. Note that Python indices begin at 0, and a range of "0 to n" values in Python is *not* n inclusive.

4.1 Routine 1: LU Factorization

This routine decomposes A into the product of a unit lower triangular matrix L and an upper triangular matrix U. For an $n \times n$ matrix A, the routine will perform $n-1$ Gaussian transformations (run $n-1$ loops):

1. **Input:** $A \in \mathbb{R}^{n \times n}$, dimension n, and flag to indicate whether the routine will perform no pivoting, partial pivoting, or complete pivoting.

2. **Initialization:** Array $P_r \in \mathbb{R}^{n \times 1}$ to store row permutations and array $P_c \in \mathbb{R}^{n \times 1}$ to store column permutations.

3. **Pivoting:** At each loop $i \in \{1, \dots, n-1\}$ the routine will first detect whether factorization can proceed for a pivot element, depending on the flag. If small values ($<$ 1e$-$10) are detected, the routine pauses and returns **None**. If no pivoting (flag $= 1$) will be performed, the pivot element (i.e. the diagonal entry α_{ii}) cannot be a small value. If partial pivoting (flag $= 2$), then all the entries in the i^{th} column $(\alpha_{0i}, \dots, \alpha_{ni})$ are checked for small values. If there is at least one non-small value, then the routine proceeds to find the row with the element of maximum magnitude in the i^{th} column. The row with the maximum element is interchanged with the first row of the current transformation matrix, and the permutation is

stored in P_r. If complete pivoting (flag = 3), all the entries in the current submatrix are checked for small values. If there is at least one non-small value, then the routine proceeds to find the position of the element with the largest magnitude in the matrix. The row and column containing the maximum element are swapped with the first row and first column of the current transformation matrix, and the permutations are stored in P_r and P_c, respectively.

4. **Factorization:** At each loop $i \in \{1, \ldots, n-1\}$ the routine computes

$$\lambda = \frac{\alpha_{ji}}{\alpha_{ii}} \tag{6}$$

for all $j \in i+1, \ldots n$. Each λ is stored in the corresponding sub-diagonal entry α_{ji} of A. These entries become the elements of L. The remaining super-diagonal elements are updated by

$$\alpha_{jk} = \alpha_{jk} - (\lambda \alpha_{ik}). \tag{7}$$

These entries become the elements of U.

5. **Output:** Return the updated matrix $\tilde{A}$ and the corresponding permutation vectors. The composition of A now becomes

$$\tilde{A} = \begin{bmatrix} \mu_{11} & \mu_{12} & \cdots & \mu_{1n} \\ \lambda_{21} & \mu_{22} & \cdots & \mu_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{n1} & \lambda_{n2} & \cdots & \mu_{nn} \end{bmatrix} \tag{8}$$

where $\lambda_{ij} \in L$ and $\mu_{ij} \in U$.

4.2 Evaluation of Factorization

4.2.1 In-Place Matrix Product

Since A is factorized to store L and U, a routine to perform the in-place outer product $L * U$ is necessary to evaluate the accuracy of the factorization.

1. **Input:** Factorized matrix $\tilde{A}$, dimension n
2. **Initialization:** Matrix $M \in \mathbb{R}^{n \times n}$
3. **Outer Product Summation:** Matrix A represents the outer product of L and U. Matrix M is initialized using the first row of A. Then $i \in \{1, \ldots, n-1\}$ iterations are performed. Each iteration i loops over i to n outer products, where each outer product contributes to a partial sum that updates the trailing submatrix $A(i : n, i : n)$. The diagonal elements of M are given by μ_{ii}, since A holds a unit triangular matrix where $\lambda_{ii} = 1$. This includes the final entry μ_{nn}.
4. **Output:** Return $L * U$ stored in M

4.2.2 Application of Permutation Arrays to A

When decomposing A into LU, we generated the permutation matrices P_r and P_c. To validate $L * U$ computed above, we apply these permutation matrices back to the original matrix A, i.e. we want to show $P_r * A * P_c = L * U$.

1. **Input:** Original matrix A, row permutation array P_r, column permutation array P_c, and the chosen flag
2. **Pivoting:** First, the indices in P_r indicate the row order for A. The rows of A are rearranged according to P_r. Then, the indices in P_c are used to rearrange the columns of A.
3. **Output:** Permuted matrix A, stored in A_2

Note that when no pivoting (flag = 1) is performed, the indices of P_r and P_c store the original row and column orders of A. Hence $A_2 = P_r A P_c = A$. When partial pivoting (flag = 2) is performed, only the indices of P_r reflect potential row permutations. The array P_c holds the original column order of A. Hence $A_2 = P_r A P_c = P_r A$. For complete pivoting (flag = 3), the indices of both P_r and P_c store potential row and column permutations.

4.3 Routine 2: Solve Ax = b for Vector x

4.3.1 Forward Substitution

A forward substitution algorithm is applied to solve $Ly = b$ for y, where L is stored in the decomposed matrix $\tilde{A}$. The lower triangular system $Ly = b$ is treated as a system of linear equations

$$\sum_{j=1}^{i} \lambda_{ij} y_j = b_i \quad \text{for } i = 2, 3, \ldots, n.$$

The algorithm works top-down to solve

$$y_n = \frac{b_n - \sum_{j=1}^{n-1} \lambda_{nj} y_j}{\lambda_{nn}} \quad \text{for } i = 2, 3, \ldots, n.$$

4.3.2 Backward Substitution

A backward substitution algorithm is applied to solve $Ux = y$ for x, where U is stored in the decomposed matrix $\tilde{A}$, and y is obtained from the forward substitution. The upper triangular system is treated as a system of linear equations

$$\sum_{j=i}^{n} \mu_{ij} x_j = b_i, \quad \text{for } i = 1, 2, \ldots, n.$$

The algorithm works bottom-up to solve

$$x_i = \frac{y_i - \sum_{j=i+1}^{n} \mu_{ij} x_j}{\mu_{ii}}, \quad \text{for } i = n-1, n-2, \ldots, 1.$$

4.4 Methods for Error Analysis

4.4.1 Factorization Accuracy

The relative factorization error can be assessed by comparing the outputs discussed in **4.2.1** and **4.2.2**. We compute

$$\frac{||P_r A P_c - LU||}{||A||} \tag{9}$$

where $||A|| \geq 1$. If the factorization error = 0, then the computed matrices L and U exactly reproduce A. In practice, the factorization error may be small due to round-off errors.

4.4.2 Solution Accuracy

Python SciPy packages are implemented to obtain the true solution x. We check the accuracy of the computed $\tilde{x}$ from Routine 2 (**4.3**), by evaluating the relative error

$$\frac{||x - \tilde{x}||}{||x||} \tag{10}$$

where $||x|| \geq 1$, so we assume the norm of x is sufficiently large.

4.4.3 Residual Accuracy

Python NumPy packages are implemented to compute $A\tilde{x}$. The residual values $b - A\tilde{x}$ measure how well the computed solution $\tilde{x}$ from Routine 2 (**4.3**) satisfies the original equation $Ax = b$. We evaluate the residual error

$$\frac{||b - A\tilde{x}||}{b} \tag{11}$$

where $||b|| \geq 1$, so we assume the norm of b is sufficiently large. Small residual error indicates that the computed solution $\tilde{x}$ is close to satisfying $Ax = b$, even if there is some relative error in $\tilde{x}$.

4.4.4 Condition Number

Python NumPy packages are implemented to compute the condition number

$$\kappa(A) = ||A^{-1}||\,||A||. \tag{12}$$

The condition number indicates how well-conditioned the matrix A is for numerical computations. If the condition number is large, then a small change to the input b may result in large changes to the solution x, and the matrix A is close to being singular. If the condition number is smalll, then a small change to the input b will cause only small changes to the solution x.

4.4.5 Growth Factor

The algorithm computes the growth factor. The numerical stability of the LU decomposition depends on the growth factor

$$\gamma_\epsilon = \frac{||L_\epsilon U_\epsilon||}{||A||} \tag{13}$$

where A is the original matrix and $L_\epsilon U_\epsilon = P_r A$ is the computed factorization of a row-permuted matrix $P_r A$, based on the selected pivoting strategy. Ideally, γ_ϵ should be close to 1, meaning the elements of the matrix do not significantly change in magnitude during the calculation.

5 Description of the Experimental Design and Results

5.1 Testing Specific Matrix Structures

If the elements of an entry are not specified, assume the entry is 0. We use I to denote the identity matrix.

1. Let $A \in \mathbb{R}^{n \times n}$ be diagonal with positive elements. Suppose $\alpha_{ii} = i + 1$, and let $\alpha_{ij} = 0$ for all $i \neq j$. For no pivoting, the factorization of A is

$$\tilde{A} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 2 & \cdots & 0 \\ \vdots & & & \\ 0 & 0 & \cdots & n \end{bmatrix} = A,$$

   meaning $L = I$ and $U = A$. The same result is true for partial pivoting, and the permutation array P_r is unchanged ($P_r = I$). Example output for partial pivoting when $n = 5$:

```
A:
 [[1. 0. 0. 0. 0.]
 [0. 2. 0. 0. 0.]
 [0. 0. 3. 0. 0.]
 [0. 0. 0. 4. 0.]
 [0. 0. 0. 0. 5.]]
A Factorized into LU:
 [[1. 0. 0. 0. 0.]
 [0. 2. 0. 0. 0.]
 [0. 0. 3. 0. 0.]
 [0. 0. 0. 4. 0.]
 [0. 0. 0. 0. 5.]]
Row Permutation Array P:
 [0 1 2 3 4]
Column Permutation Array P:
 [0 1 2 3 4]
```

For complete pivoting, the factorization of A is

$$\tilde{A} = \begin{bmatrix} n & 0 & \cdots & 0 \\ 0 & n-1 & \cdots & 0 \\ \vdots & & & \\ 0 & 0 & \cdots & 1 \end{bmatrix},$$

meaning $L = I$ and $U = \tilde{A}$. The rows and columns are permuted in the same way, so the indices of $P_r = P_c = [n, n-1, \ldots, 1]$ are in reverse order. The growth factor $= 1$ for each pivoting choice, because the diagonal entries do not change in magnitude. Example output for complete pivoting when $n = 5$:

```
A:
 [[1. 0. 0. 0. 0.]
 [0. 2. 0. 0. 0.]
 [0. 0. 3. 0. 0.]
 [0. 0. 0. 4. 0.]
 [0. 0. 0. 0. 5.]]
A Factorized into LU:
 [[5. 0. 0. 0. 0.]
 [0. 4. 0. 0. 0.]
 [0. 0. 3. 0. 0.]
 [0. 0. 0. 2. 0.]
 [0. 0. 0. 0. 1.]]
Row Permutation Array P:
 [4 3 2 1 0]
Column Permutation Array P:
 [4 3 2 1 0]
```

2. Let $A \in \mathbb{R}^{n \times n}$ be anti-diagonal with positive elements, where $\alpha_{n0} = n$ and $\alpha_{0n} = 1$. For no pivoting, A cannot be factorized, since the first diagonal element $= 0$. Example output for no pivoting when $n = 5$:

```
A:
 [[0. 0. 0. 0. 1.]
 [0. 0. 0. 2. 0.]
 [0. 0. 3. 0. 0.]
 [0. 4. 0. 0. 0.]
 [5. 0. 0. 0. 0.]]
A Factorized into LU:
 None
Row Permutation Array P:
 None
Column Permutation Array P:
 None
```

For partial pivoting, the factorization of A yields the diagonal matrix

$$\tilde{A} = \begin{bmatrix} n & 0 & \cdots & 0 \\ 0 & n-1 & \cdots & 0 \\ \vdots & & & \\ 0 & 0 & \cdots & 1 \end{bmatrix},$$

meaning $L = I$, $U = \tilde{A}$, and the indices of the permutation array P_r are reversed in order (i.e. $P_r = [n, n-1, \ldots, 1]$). The same result is true for complete pivoting, and the permutation array $P_c = I$ is unchanged. The growth factor = 1 for each pivoting choice, because the diagonal entries do not change in magnitude. Example output for complete pivoting when $n = 5$:

```
A:
 [[0. 0. 0. 0. 1.]
 [0. 0. 0. 2. 0.]
 [0. 0. 3. 0. 0.]
 [0. 4. 0. 0. 0.]
 [5. 0. 0. 0. 0.]]
A Factorized into LU:
 [[5. 0. 0. 0. 0.]
 [0. 4. 0. 0. 0.]
 [0. 0. 3. 0. 0.]
 [0. 0. 0. 2. 0.]
 [0. 0. 0. 0. 1.]]
Row Permutation Array P:
 [4 3 2 1 0]
Column Permutation Array P:
 [0 1 2 3 4]
```

3. Let $A \in \mathbb{R}^{n \times n}$ be the sum of a diagonal matrix and an anti-diagonal matrix. In theory, partial and complete pivoting can help manage instability in the factorization. In practice, however, the factorization does not proceed for either pivoting strategy, as zero elements are encountered in the diagonal during factorization. Example output for complete pivoting when $n = 5$:

```
A:
 [[1. 0. 0. 0. 1.]
 [0. 2. 0. 2. 0.]
 [0. 0. 6. 0. 0.]
 [0. 4. 0. 4. 0.]
 [5. 0. 0. 0. 5.]]
A Factorized into LU:
 None
Row Permutation Array P:
 None
Column Permutation Array P:
 None
```

4. Let $A \in \mathbb{R}^{n \times n}$ be unit lower triangular and let the sub-diagonal entries have magnitude < 1. Since the each diagonal entry = 1 and is greater than the magnitude of all the sub-diagonal entries, all three pivoting strategies yield the same result (since no pivoting will occur). The factorization of A yields $\tilde{A} = L$, and thus $U = I$. No permutations occur, so $P_r = P_c = I$ and the indices remain unchanged. Example output for complete pivoting when $n = 5$:

```
A:
 [[ 1.          0.          0.          0.          0.        ]
 [-0.25091976  1.          0.          0.          0.        ]
 [ 0.98142861  0.46398788  1.          0.          0.        ]
 [ 0.19731697 -0.68796272 -0.68801096  1.          0.        ]
 [-0.88383278  0.73235229  0.20223002  0.41614516  1.        ]]
A Factorized into LU:
 [[ 1.          0.          0.          0.          0.        ]
 [-0.25091976  1.          0.          0.          0.        ]
 [ 0.98142861  0.46398788  1.          0.          0.        ]
 [ 0.19731697 -0.68796272 -0.68801096  1.          0.        ]
 [-0.88383278  0.73235229  0.20223002  0.41614516  1.        ]]
Row Permutation Array P:
 [0 1 2 3 4]
Column Permutation Array P:
 [0 1 2 3 4]
```

5. Let $A \in \mathbb{R}^{n \times n}$ be lower triangular. Let the diagonal entries be positive and $\neq 1$ and let the sub-diagonal entries be > 1. Similar to the results above, if no pivoting, then $\tilde{A} = L$ and $U = I$, but the factorization becomes unstable when there are large off-diagonal entries. Since the off-diagonal entries can be larger than the diagonal entries, then partial and complete pivoting may occur. Complete pivoting yields the most stable factorization and reduces the growth factor. Example output for no pivoting when $n = 5$:

```
A:
 [[4. 0. 0. 0. 0.]
 [8. 4. 0. 0. 0.]
 [9. 7. 4. 0. 0.]
 [3. 2. 9. 4. 0.]
 [2. 4. 3. 4. 4.]]
A Factorized into LU:
 [[4.   0.   0.   0.   0.  ]
 [2.   4.   0.   0.   0.  ]
 [2.25 1.75 4.   0.   0.  ]
 [0.75 0.5  2.25 4.   0.  ]
 [0.5  1.   0.75 1.   4.  ]]
Row Permutation Array P:
 [0 1 2 3 4]
Column Permutation Array P:
 [0 1 2 3 4]
Condition Number:
 32.59951972171953
Growth Factor:
 2.024034139323784
```

Compare to output when complete pivoting is performed on the same matrix A:

```
A:
 [[4. 0. 0. 0. 0.]
 [8. 4. 0. 0. 0.]
 [9. 7. 4. 0. 0.]
 [3. 2. 9. 4. 0.]
 [2. 4. 3. 4. 4.]]
A Factorized into LU:
 [[ 9.          4.          0.          7.          0.        ]
 [ 0.33333333  7.66666667  0.         -0.33333333  4.        ]
 [ 0.22222222  0.27536232  4.          2.53623188  2.89855072]
 [ 0.44444444 -0.23188406  0.         -3.1884058   0.92753623]
 [ 0.88888889 -0.46376812  0.          0.74545455  1.16363636]]
Row Permutation Array P:
 [2 3 4 0 1]
Column Permutation Array P:
 [0 2 4 1 3]
Condition Number:
 32.59951972171953
Growth Factor:
 1.5005983831901861
```

6. Let $A \in \mathbb{R}^{n \times n}$ be tri-diagonal with nonzero elements on the super-diagonal, diagonal, and sub-diagonal entries. Suppose A is strictly diagonally dominant by rows and columns. Since A is diagonally dominant, then no row pivoting will occur, since the diagonal element has the largest magnitude per column. Hence the factorization of A will yield the same results when no pivoting or partial pivoting are performed, and the indices of P_r remain unchanged. The tri-diagonal structure of A is preserved in the result $\tilde{A}$. Example output for partial pivoting when $n = 5$:

```
A:
 [[12.52982362  3.7909153   0.          0.          0.        ]
 [ 4.86811936 14.34791532  1.86435798  0.          0.        ]
 [ 0.          3.188929   17.79382922  4.90509782  0.        ]
 [ 0.          0.          4.89073744 11.97685075  1.02492102]
 [ 0.          0.          0.          3.85926397 18.62993236]]
A Factorized into LU:
 [[12.52982362  3.7909153   0.          0.          0.        ]
 [ 0.38852258 12.87505915  1.86435798  0.          0.        ]
 [ 0.          0.24768267 17.33206006  4.90509782  0.        ]
 [ 0.          0.          0.28217866 10.59273683  1.02492102]
 [ 0.          0.          0.          0.36433115 18.2565217 ]]
Row Permutation Array P:
 [0 1 2 3 4]
Column Permutation Array P:
 [0 1 2 3 4]
```

Because the diagonal elements of A vary in magnitudes, complete pivoting may occur. Since the diagonal elements will be shifted to locations on the diagonal, the same indices will change for P_r and P_c (i.e. $P_r = P_c$ when complete pivoting). Example output for complete pivoting when $n = 5$:

```
A:
 [[12.52982362  3.7909153   0.          0.          0.        ]
 [ 4.86811936 14.34791532  1.86435798  0.          0.        ]
 [ 0.          3.188929   17.79382922  4.90509782  0.        ]
 [ 0.          0.          4.89073744 11.97685075  1.02492102]
 [ 0.          0.          0.          3.85926397 18.62993236]]
A Factorized into LU:
 [[18.62993236  0.          0.          0.          3.85926397]
 [ 0.         17.79382922  3.188929    0.          4.90509782]
 [ 0.          0.10477554 14.01379358  4.86811936 -0.51393425]
 [ 0.          0.          0.27051314 11.21293337  0.13902597]
 [ 0.05501475  0.27485582 -0.06254521  0.02715414 10.3804204 ]]
Row Permutation Array P:
 [4 2 1 0 3]
Column Permutation Array P:
 [4 2 1 0 3]
```

7. Let $A \in \mathbb{R}^{n \times n}$ and let the sub-diagonal entries $= -1$, diagonal entries $= 1$, and all elements in the last column $= 1$. When no pivoting or partial pivoting are performed, the factorization of A yields

$$\tilde{A} = \begin{bmatrix} 1 & 0 & \cdots & 1 \\ -1 & 1 & \cdots & 2 \\ -1 & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & 2^i \end{bmatrix}.$$

The $[0 : n-1]$ columns of A are preserved in the output $\tilde{A}$. The entries in the last column n of $\tilde{A}$ becomes $\alpha_{in} = 2^i$. The row permutation array P_r is unchanged. Example output for partial pivoting when $n = 5$:

```
A:
 [[ 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.]]
A Factorized into LU:
 [[ 1.  0.  0.  0.  1.]
 [-1.  1.  0.  0.  2.]
 [-1. -1.  1.  0.  4.]
 [-1. -1. -1.  1.  8.]
 [-1. -1. -1. -1. 16.]]
Row Permutation Array P:
 [0 1 2 3 4]
Column Permutation Array P:
 [0 1 2 3 4]
Condition Number:
 2.2239234404758794
Growth Factor:
 1.883209726302261
```

When complete pivoting is performed, the factorization of A yields a slightly different output: the first column remains unchanged, the last column is moved to the second column, and the rest of the columns are moved one column to the right. The column permutation array becomes $P_c = [0, n, 1, 2, \ldots, n-1]$. The first diagonal entry $\alpha_{00} = 1$, the second diagonal entry $\alpha_{11} = 2$, and the remaining diagonal entries $= -2$. Example output for complete pivoting when $n = 5$:

```
A:
 [[ 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.]]
A Factorized into LU:
 [[ 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.]]
Row Permutation Array P:
 [0 1 2 3 4]
Column Permutation Array P:
 [0 4 1 2 3]
Condition Number:
 2.2239234404758794
Growth Factor:
 1.7870501915438117
```

As the dimensions n increase to 100, the growth factor approaches 2 for all three pivoting strategies.

8. Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite (SPD) generated from a lower triangular matrix $\tilde{L}$ with positive diagonal elements via $A = \tilde{L}\tilde{L}^T$. This is known as the Cholesky decomposition of A. Every SPD matrix has a unique Cholesky decomposition. Note that the Cholesky decomposition is more efficient than LU decomposition for SPD matrices because it only involves working with half the matrix (since A is symmetric) and reduces the computational cost by about half, compared to LU. Since A is SPD, the LU decomposition takes the form of a Cholesky factorization, where $L = \tilde{L}$ and $U = \tilde{L}^T$. Example output when $n = 5$:

```
L̃ (Lower triangular matrix):
[[ 1.075  0.     0.     0.     0.   ]
 [-0.568  1.601  0.     0.     0.   ]
 [ 0.559 -0.605  1.168  0.     0.   ]
 [ 0.195 -0.982 -0.227  1.733  0.   ]
 [-0.128  0.898  0.573  0.733  1.408]]
L̃ Transpose:
[[ 1.075 -0.568  0.559  0.195 -0.128]
 [ 0.     1.601 -0.605 -0.982  0.898]
 [ 0.     0.     1.168 -0.227  0.573]
 [ 0.     0.     0.     1.733  0.733]
 [ 0.     0.     0.     0.     1.408]]

A (Symmetric Positive Definite matrix):
[[ 1.155625 -0.6106    0.600925  0.209625 -0.1376  ]
 [-0.6106    2.885825 -1.286117 -1.682942  1.510402]
 [ 0.600925 -1.286117  2.04273   0.437979  0.054422]
 [ 0.209625 -1.682942  0.437979  4.057167  0.233422]
 [-0.1376    1.510402  0.054422  0.233422  3.67087 ]]
Form of Factorization:
 (1) No Pivoting
 (2) Partial Pivoting
 (3) Complete Pivoting
Enter Factorization Type: 1
A Factorized into LU:
 [[ 1.155625   -0.6106      0.600925    0.209625   -0.1376    ]
 [-0.52837209  2.563201   -0.968605   -1.572182    1.437698  ]
 [ 0.52       -0.37788882  1.364224   -0.265136    0.669264  ]
 [ 0.18139535 -0.61336665 -0.19434932  3.003289    1.270289  ]
 [-0.11906977  0.56089944  0.49058219  0.42296595  1.982464  ]]
Row Permutation Array P:
 [0 1 2 3 4]
Column Permutation Array P:
 [0 1 2 3 4]
```

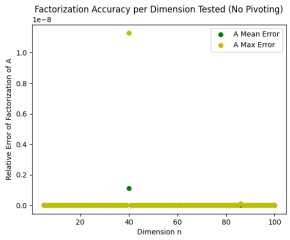
We see that $\tilde{L}\tilde{L}^T = A = LU$.

5.2 General Trends for Randomly Generated Matrices

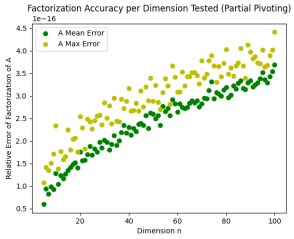
Singular matrices have either none or infinitely many solutions, which complicates the computations. If a matrix is full rank (i.e. is nonsingular) then the system $Ax = b$ has a unique solution. Test matrices with full rank are generated to access the accuracy of the factorization. We generate nicely conditioned nonsingular matrices L and U. The unit lower triangular matrix L is specified by uniform randomly generated values in the off-diagonal entries. The diagonal entries are set to 1, and the off-diagonal entries are uniformly generated on the range (-1, 1) and thus have magnitude < 1. The upper triangular matrix U is specified by uniform randomly generated values in the off-diagonal elements,

generated on the range (-10, 10). The diagonal elements of U are randomly generated on the ranges (-10, -5) and (5, 10), and the off-diagonal elements in L and U are scaled by the size of the matrix, to ensure U is diagonally dominant. Double point precision is used to maintain numerical accuracy. Since the product of two nonsingular matrices is also a nonsingular matrix, we evaluate the product of L and U to define our test matrix A for each experiment. However, no pivoting will occur if the matrix A is diagonally dominant. Before testing, we apply random permutations to the rows and columns of A, so A is not diagonally dominant but still nonsingular. The input matrix b is specified by uniform randomly generated values in the range (-10, 10).

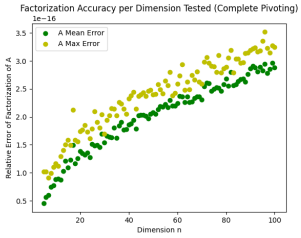
Experiments are conducted on the randomly generated nonsingular matrices A from $n = 5$ to $n = 100$ dimensions, with step size = 1. At each dimension, we generate 10 test problems (i.e. 10 matrices). A total of $(100 - 5) \times 10 = 950$ experiments are ran for each form of factorization (no pivoting, partial pivoting, and complete pivoting). Each experiment computes and stores the relative factorization error, solution error, residual error, condition number, and growth factor for the tested matrix. The average and maximum values of these metrics are stored at each dimension (since 10 matrices are tested per dimension), and scatterplots are generated to display the effect of increasing dimensions on these metrics.



(a) LU Factorization Accuracy (No Pivoting)

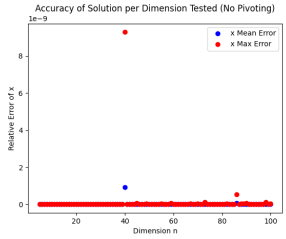


(b) LU Factorization Accuracy (Partial Pivoting)

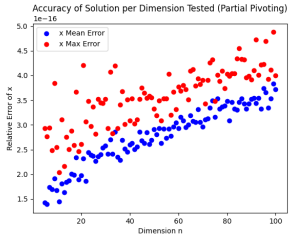


(c) LU Factorization Accuracy (Complete Pivoting)

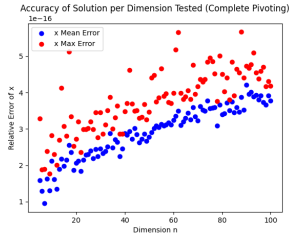
In the scatterplots above, we examine the average error from the LU factorization of A remains consistently low across all pivoting types. When no pivoting is performed, we see low error of order 1×10^{-12}. The error decreases when partial or complete pivoting are performed. The scatterplots show the error changes from order 10^{-17} to 10^{-16} as n increases from 5 to 100, and thus remains consistently low. Across all pivoting strategies, the errors are close to the machine epsilon for double-precision floating point arithmetic. This suggests the approximation errors may be largely attributed to rounding in floating point number systems, meaning our test matrices are well-conditioned and numerically stable for factorization.



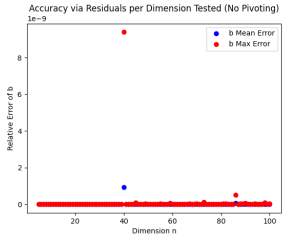
(a) Solving for x (No Pivoting)



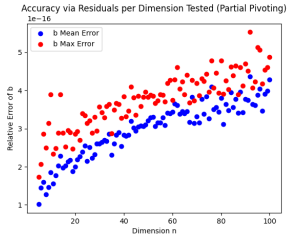
(b) Solving for x (Partial Pivoting)



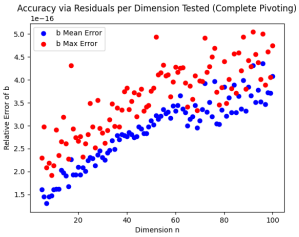
(c) Solving for x (Complete Pivoting)



(a) Accuracy via b (No Pivoting)

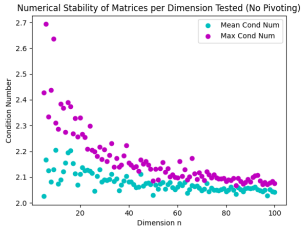


(b) Accuracy via b (Partial Pivoting)

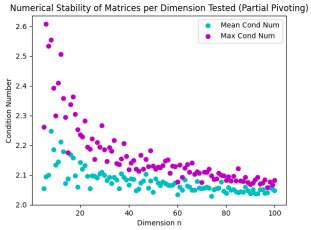


(c) Accuracy via b (Complete Pivoting)

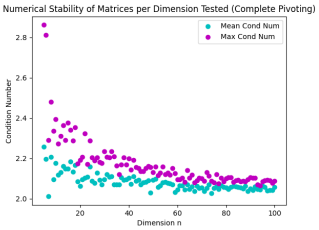
Visually, the plots for the relative errors and the residual errors (shown above) mirror the behavior for the LU factorization accuracy. When partial pivoting or complete pivoting are performed, the algorithm demonstrates improved accuracy in computing the solution x. By selecting pivot elements with the largest magnitude, we minimize round-off errors impacting the solution accuracy, and hence see a lower difference between the true versus computed values of b. Consequently, the relative errors in computing x are also significantly lower (as compared to no pivoting), since the computations are less susceptible to numerical instability.



(a) Condition Number (No Pivoting)

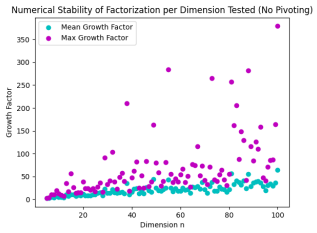


(b) Condition Number (Partial Pivoting)

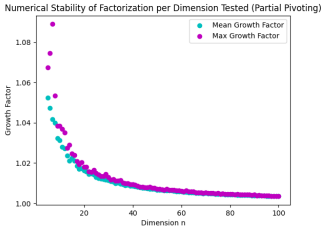


(c) Condition Number (Complete Pivoting)

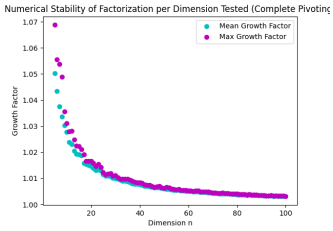
The figures above show that, for no pivoting, smaller matrices ($n < 40$) have larger variations in condition numbers. We see slightly less variation in the condition numbers for smaller matrices when partial pivoting is performed. We observe the least amount of variation in the condition numbers for small matrices when complete pivoting is used, since complete pivoting shifts elements with the largest magnitude to the diagonal entries. Hence the matrix is more well-conditioned. Across all pivoting strategies, as the dimensions increase, the condition numbers converge to ≈ 2. This suggests large matrices are more stable. This is due to our matrix conditioning strategy, where we scale the off diagonal entries of our test matrices by a factor of $\frac{1}{n}$. Thus larger matrices will be more diagonally dominant.



(a) Growth Factor (No Pivoting)



(b) Growth Factor (Partial Pivoting)



(c) Growth Factor (Complete Pivoting)

The figures above show the growth factors for the different pivoting strategies. On average, when no pivoting is performed, the growth factor is ≈ 50, with certain test matrices yielding maximum growth factors between 100 and 400. This indicates relative instability in the factorization. We see improved numerical stability when partial or complete pivoting are performed. The growth factor decreases as n increases, converging to a value close to 1 for dimensions $n > 40$, meaning the elements of large matrices do not significantly change in magnitude during the factorization. The growth factor converges to 1 more quickly when complete pivoting is performed, which confirms that complete pivoting yields the most stable factorization.

6 Conclusions

The results show that partial pivoting provides sufficient numerical stability for most test cases. Both the factorization and residual errors remain small, even as matrix dimensions increase. Complete pivoting, while computationally more expensive, yields the lowest errors and highest numerical stability, making

it effective when precision is critical, such as for ill-conditioned matrices. No pivoting is less stable, with larger errors in the factorization accuracy. The factorization can fail in the presence of zero or small pivot elements. Additionally, the growth factor indicates that partial and complete pivoting better stabilize the factorization, with complete pivoting providing the smallest increase in magnitude during decomposition. In contrast, no pivoting leads to higher growth factors. This signals potential instability.

Overall, for practical implementations, partial pivoting offers a balance between computational efficiency and numerical accuracy. However, complete pivoting is the most reliable option for high-dimensional matrices, where numerical precision is important. In the future, we may extend the experimental design to assess larger dimensions of matrices to better understand the practical limits of these methods.

References

[1] Dr. Kyle A. Gallivan, "Set 4: Factorization for Linear Systems Part 2", Class Lecture, MAD5403: Foundations of Computational Mathematics I, Florida State University, September 18, 2024.

[2] Wikipedia contributors, "LU Decomposition," *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/LU_decomposition, [Online; accessed 11-October-2024].

[3] Wikipedia contributors, "Condition Number," *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Condition_number, [Online; accessed 11-October-2024].

[4] Wikipedia contributors, "Diagonally Dominant Matrix," *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Diagonally_dominant_matrix, [Online; accessed 11-October-2024].

[5] Wikipedia contributors, "Gaussian Elimination," *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Gaussian_elimination, [Online; accessed 11-October-2024].

[6] Nicholas J. Higham, "What is the Growth Factor for Gaussian Elimination?", https://nhigham.com/2020/07/14/what-is-the-growth-factor-for-gaussian-elimination/, July 14, 2020. [Online; accessed 11-October-2024].

[7] University of Illinois at Urbana-Champaign, "CS357 - Lecture Notes on Linear Systems," https://courses.physics.illinois.edu/cs357/sp2022/notes/ref-9-linsys.html, [Online; accessed 11-October-2024].