FCM Programming Assignment 3

Jonathan Cushman

October 2025

1 Executive Summary

In this project we are asked to write an algorithm that computes the factorization of a matrix. The factorization algorithm is broken into three cases: no pivoting ($A = LU$), partial pivoting ($PA = LU$), and complete pivoting ($PAQ = LU$). We then empirically test this algorithm on a variety of structured matrices and their products. We then measure the relative error, growth factor, and residuals produced as the size of the matrix grows. Additionally we will look for edge cases where the matrix factorization is not possible due to restrictions in the structure or floating point arithmetic limits being reached

2 Statement of the Problem

The purpose of this assignment is to produce and implement an algorithm that computes the factorization of a matrix using Gaussian elimination and pivoting techniques. The first is a no pivoting method such that $A = LU$, where A is a full rank matrix, L is a unit lower triangular matrix, and U is an upper triangular matrix. The second method, partial pivoting yields $PA = LU$ where P is a permutation matrix that changes the rows of the matrix A. The final method is complete pivoting where $PAQ = LU$ is being computed. In this scenario all other matrices remain the same and Q is a permutation matrix that changes the columns of A.

The assignment also includes developing supporting routines for matrix permutations, forward and backward substitution, and matrix reconstruction from stored L and U factors. The result from our testing will then evaluated using metrics such as matrix factorization relative error, residual relative error, and the growth factor (γ). We will then test and validate the algorithm on variety of structured matrices and edge cases, paying special attention to numerical stability and limitations of the algorithm.

3 Description of the Mathematics

3.1 Gaussian Elimination without Pivoting

Assume A is diagonally dominant so no pivot ever vanishes

$$|a_{11}| > |a_{12}| + |a_{13}|, \quad \dots$$

where

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

Then we locate the pivot element (not the same as no pivoting method) and calculate our multiplier such that

$$\ell_{21} = \frac{a_{21}}{a_{11}}, \quad \ell_{31} = \frac{a_{31}}{a_{11}}$$

with multipliers ℓ_{21} and ℓ_{31}.

Row 2: $(a_{21}, a_{22}, a_{23}) - \ell_{21}(a_{11}, a_{12}, a_{13}) = (0, a'_{22}, a'_{23})$

Row 3: $(a_{31}, a_{32}, a_{33}) - \ell_{31}(a_{11}, a_{12}, a_{13}) = (0, a'_{32}, a'_{33})$

We then repeat this process.

Now

$$A' = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & a'_{32} & a'_{33} \end{bmatrix}$$

Then second pivot $\ell_{32} = \frac{a'_{32}}{a'_{22}}$

Then we eliminate row 3: $(0, a'_{32}, a'_{33}) - \ell_{32}(0, a'_{22}, a'_{23}) = (0, 0, a''_{33})$

Resulting in our final in-place matrix

$$A'' = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ \ell_{21} & a'_{22} & a'_{23} \\ \ell_{31} & \ell_{32} & a''_{33} \end{bmatrix}$$

So that

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a''_{33} \end{bmatrix}$$

3.2 Gaussian Elimination with Partial Pivoting

For our second method let's now take A to have a zero in its first diagonal spot such that

$$A = \begin{bmatrix} 0 & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Additionally, we will track a pre-multiplication matrix $P = I$, where

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Looking down the first column of A, suppose $|a_{21}| > |a_{31}|$, then we swap row 1 and 2 for our matrix A and P:

$$A = \begin{bmatrix} a_{21} & a_{22} & a_{23} \\ 0 & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now we locate our pivot element $m_{11} = a_{21}$, giving multipliers $\ell_{21} = \frac{0}{a_{21}} = 0$ and $\ell_{31} = \frac{a_{31}}{a_{21}}$, then we eliminate row 2 and 3 like previously.

Row 2:

$$(0, a_{12}, a_{13}) - 0 \cdot (a_{21}, a_{22}, a_{23}) = (0, a_{12}, a_{13})$$

Row 3:

$$(a_{31}, a_{32}, a_{33}) - \frac{a_{31}}{a_{21}}(a_{21}, a_{22}, a_{23}) = (0, a''_{32}, a''_{33})$$

So

$$A'' = \begin{bmatrix} a_{21} & a_{22} & a_{23} \\ 0 & a_{12} & a_{13} \\ \ell_{31} & a''_{32} & a''_{33} \end{bmatrix}$$

Then we repeat this process for Column 2.

Suppose $|a_{32}| > |a_{12}|$, we swap row 2 and 3.

$$A''' = \begin{bmatrix} a_{21} & a_{22} & a_{23} \\ \ell_{31} & a''_{32} & a''_{33} \\ 0 & a_{12} & a_{13} \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Next, $m_{22} = \frac{a_{32}}{a_{22}}$ and $\ell_{32} = \frac{a_{12}}{a''_{32}}$.
Then Row 3: $(0, a_{12}, a_{13}) - \ell_{32} \cdot (0, a''_{32}, a''_{33}) = (0, 0, a''_{13})$
So our final in place matrix is

$$\begin{bmatrix} a_{21} & a_{22} & a_{23} \\ \ell_{31} & a''_{32} & a''_{33} \\ 0 & \ell_{32} & a''_{13} \end{bmatrix}$$

Such that $PA = LU$ where

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{31} & 1 & 0 \\ 0 & l_{32} & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} a_{21} & a_{22} & a_{23} \\ 0 & a''_{32} & a''_{33} \\ 0 & 0 & a''_{13} \end{bmatrix}$$

3.3 Gaussian Elimination with complete pivoting

Assume A is a general matrix with the condition that the first entry of the first row and column is not the largest element.

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

$$a_{ij} = \max_{p,q} |a_{pq}| \quad \text{where} \quad (p, q) \neq (1, 1)$$

We track permutation matrices P and Q. If a_{ij} has the largest magnitude, then we swap row i with 1 in A and P, and then swap column j with 1 in A and Q. Then:

$$A \to \begin{bmatrix} a_{ij} & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \quad \text{with pivot elements}$$

For $k = 2, 3$ we have:

$$\mu_{11} = a_{ij} \quad \text{and} \quad \ell_{k1} = \frac{A_{k1}}{\mu_{11}}$$

Then we eliminate row k to produce zeros in column one below the pivot. This process is then repeated with the remaining $(n-1) \times (n-1)$ portion.

At the end we have:

$$PAQ = \begin{bmatrix} a_{ij} & * & * \\ 0 & a_{mn} & * \\ 0 & 0 & * \end{bmatrix} = LU$$

4 Description of Algorithm and Implementation

One request for this assignment was that the algorithm and all its methods are stored within one function that is capable of completing each of the methods and requested subroutines. To accommodate this the function includes a "flag" that allows you to choose which method will be implemented based on the matrix that gets sent in for in place LU factorization, where we store L and U within the same matrix for efficiency. Then Prow and Pcolumn track the row and column swaps being preformed based on the selected method

For each step k within the algorithm, a pivot element is selected to eliminate entries below it by subtracting appropriate multiples of the pivot row. Each elimination can be expressed as a rank-one update to the trailing sub matrix, ensuring that we have numerical and computational efficiency.

When selecting the pivot element, each "flag" has their strategy in choosing a pivot element. In the traditional no pivoting case the element is a_{kk}. In the partial pivoting the algorithm begins search for the

largest magnitude in column k and then swaps the rows to move the largest element into the pivot position, this is done to help reduce the risk of dividing by a small pivot value. In complete pivoting, this search for the largest element is extending the the active part of the matrix, ie) A(k:n,k:n). We then swap corresponding rows and columns. The advantage of doing this is numerical stability with the expectation that we will use more computational power to find the appropriate pivoting element.

The algorithm also includes error guards that back out of the computation before continuing on in the algorithm when faced with numerically dangerous computations. We do this so we are not stuck wasting time computing something that we know will obviously be wrong and yield a high error. The value is set at $1 * e^{-12}$ to ensure time is not wasted on ill conditioned matrices.

Another consideration when implementing was the efficient use of memory. In order to do this, both L and U are stored together in place as one matrix, we are able to do this because the lower triangular matrix is unit diagonal thus it is unnecessary to store the 1's on the diagonal or the zeros included in both matrices. The store the permutation matrices, we initialize them as vectors ranging from (1,2,...,n) where Prow and Pcol record the mapping between the current and original index of our matrices.

A quick note on the complexity of the algorithms. No pivoting has no search cost and total floating point operations of $2/3 * n^3$ with an overall cost of $O(n^3)$. Both the partial and complete pivoting methods have a pivot search cost of $O(n^2)$ which brings their total respective floating point operations to $2/3 * n^3 + O(n^2)$ and $2/3 * n^3 + O(n^3)$. These methods also result in an overall cost of $O(n^3)$ for Gaussian elimination and LU factorization. from this we can gather why we have a trade off in terms of speed for numerical stability. The no pivoting method is the fastest while the complete pivoting is the slowest, but in return the complete method has the highest numerical stability while the no pivoting method yields the lowest.

Listing 1: Algorithm

```
function [M, Prow, Pcol] = LUdense(M, n, routinenum, Prow, Pcol)

    % No Pivot Method (A = LU)
    if routinenum == 1
    % Perform LU factorization without pivoting
        for k = 1:n-1
            % Error Guard
            if abs(M(k,k)) < 1e-12
                disp('Matrix is singular or nearly singular')
            end

            for i = k+1:n
            % Calculate the multiplier for each row
            M(i,k) = M(i,k) / M(k,k);
            % Update the matrix rows based on the multiplier
            M(i, k+1:n) = M(i, k+1:n) - M(i,k) * M(k, k+1:n);
            end
        end

    % Note that L and U are stored in A for computational efficiency
    end

    % Partial Pivoting Method (PA = LU)
    if routinenum == 2

        % initialize permutation vector
        Prow = (1:n)'; % P = [1,2,...,n] where n is the assciated column for the e
             vector

        for k = 1:n-1
            % find max element in column k and the associated pivot row
            max_value = 0;
```

```
            associated_row = k;
            for i = k:n
                if abs(M(i,k)) > max_value
                    max_value = abs(M(i,k));
                    associated_row = i;
                end
            end

            % Error Guard
            if max_value < 1e-12
                disp('Matrix is singular or nearly singular')
            end

            % Check for row swap
            if associated_row ~= k
                % create copy of row getting replaced
                row_copy = M(k,1:n);
                % replace with better row
                M(k,1:n) = M(associated_row,1:n);
                % replace old position of better row with the copy made
                M(associated_row,1:n) = row_copy;

                % replicate process for permutation vector
                row_copy = Prow(k);
                Prow(k) = Prow(associated_row);
                Prow(associated_row) = row_copy;
            end

            for i = k+1:n
            % Calculate the multiplier for each row
            M(i,k) = M(i,k) / M(k,k);
            % Update the matrix rows based on the multiplier
            M(i, k+1:n) = M(i, k+1:n) - M(i,k) * M(k, k+1:n);
            end
        end

    end

    % Complete Pivoting (PAQ = LU)
    if routinenum == 3

        % initalize Permutations Vectors
            Prow = (1:n)';
            Pcol = (1:n)';

        for k = 1:n-1
            max_value = 0;
            associated_row = k;
            associated_column = k;

            % find max A(i,j) in active Matrix A(k:n,k:n)
            for i = k:n
                for j = k:n
                    if abs(M(i,j)) > max_value
                        max_value = abs(M(i,j));
                        associated_row = i;
                        associated_column = j;
                    end
                end
            end
```

```
92
93              % Error Guard
94              if max_value < 1e-12
95                  disp('Matrix is singular or nearly singular')
96              end
97
98              if associated_row ~= k
99                  % create copy of row getting replaced
100                 row_copy = M(k,1:n);
101                 % replace with better row
102                 M(k,1:n) = M(associated_row,1:n);
103                 % replace old position of better row with the copy made
104                 M(associated_row,1:n) = row_copy;
105
106                 % replicate process for permutation vector
107                 row_copy = Prow(k);
108                 Prow(k) = Prow(associated_row);
109                 Prow(associated_row) = row_copy;
110             end
111
112             % Similar to before swapped indexing
113             if associated_column ~= k
114                 % Swap columns in A
115                 col_copy = M(1:n,k);
116                 M(1:n,k) = M(1:n,associated_column);
117                 M(1:n,associated_column) = col_copy;
118
119                 % Update permutation vector for columns
120                 col_copy = Pcol(k);
121                 Pcol(k) = Pcol(associated_column);
122                 Pcol(associated_column) = col_copy;
123             end
124
125             for i = k+1:n
126             % Calculate the multiplier for each row
127             M(i,k) = M(i,k) / M(k,k);
128             % Update the matrix rows based on the multiplier
129             M(i, k+1:n) = M(i, k+1:n) - M(i,k) * M(k, k+1:n);
130             end
131         end
132     end
133 end
```

5 Description of the Experimental Design and Results

The design of this experiment is based on the testing of matrices with varying structures, and calculate their metrics over varying sizes. The metrics being used in this experiment include the factorization error, growth factor, and residual error. We will test on a number of repeated tests per size and examine the mean and max values being repeatedly output by our algorithm as the size of our matrix grows. All matrix norms used for error and residual computations were the standard 2-norm. We also compute the condition number for each structured matrix to assess the relationship between matrix conditioning and the stability of the computed LU factorization.

$$E_{\text{fac}} = \frac{\|P_r A P_c - LU\|_2}{\|A\|_2} \tag{1}$$

$$\gamma = \frac{\| \, |L| \, |U| \, \|_2}{\|A\|_2} \tag{2}$$

$$E_{\text{res}} = \frac{\| b - Ax \|_2}{\|b\|_2} \tag{3}$$

$$K_2(A) = \|A\|_2 \, \|A^{-1}\|_2 \tag{4}$$

the results were summarized using mean and maximum values:

$$\bar{E}_{\text{fac}} = \frac{1}{N} \sum_{i=1}^{N} E_{\text{fac},i}, \qquad E_{\text{fac,max}} = \max_i E_{\text{fac},i}, \tag{5}$$

$$\bar{\gamma} = \frac{1}{N} \sum_{i=1}^{N} \gamma_i, \qquad \gamma_{\max} = \max_i \gamma_i, \tag{6}$$

$$\bar{E}_{\text{res}} = \frac{1}{N} \sum_{i=1}^{N} E_{\text{res},i}, \qquad E_{\text{res,max}} = \max_i E_{\text{res},i}. \tag{7}$$

5.1 Diagonal Matrix Testing Results

During testing for the first structured matrix, we see from the result in our chart below that our factorization and residual error are numerically zero, while the growth factor remained constant at the value one throughout each iteration of testing. This follows exactly as predicted by our theory that no permutation is needed to compute this matrix and we are properly preserving the structure of our matrix.

Figure 1: Diagonal Matrix Metrics

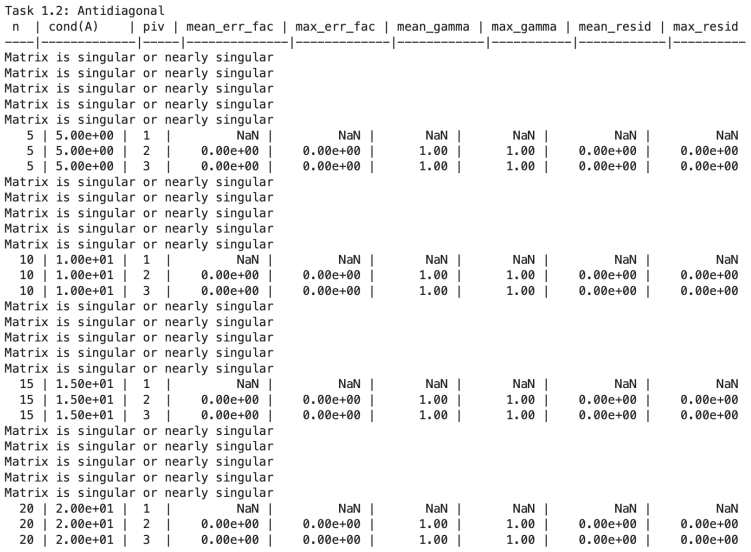
Task 1.1: Diagonal

n	cond(A)	piv	mean_err_fac	max_err_fac	mean_gamma	max_gamma	mean_resid	max_resid
5	5.00e+00	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
5	5.00e+00	2	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
5	5.00e+00	3	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
10	1.00e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
10	1.00e+01	2	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
10	1.00e+01	3	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
15	1.50e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
15	1.50e+01	2	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
15	1.50e+01	3	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
20	2.00e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
20	2.00e+01	2	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
20	2.00e+01	3	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
25	2.50e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
25	2.50e+01	2	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
25	2.50e+01	3	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
30	3.00e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
30	3.00e+01	2	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
30	3.00e+01	3	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
35	3.50e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
35	3.50e+01	2	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
35	3.50e+01	3	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
40	4.00e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
40	4.00e+01	2	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
40	4.00e+01	3	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
45	4.50e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
45	4.50e+01	2	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
45	4.50e+01	3	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
50	5.00e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
50	5.00e+01	2	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
50	5.00e+01	3	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00

5.2 Anti-Diagonal Matrix Testing Results

Looking at the results from our second set of structured matrices we notice that for the no pivoting method in our algorithm, the matrix factorization was unable to produce any results. This is different from our 2nd and 3rd method which both mirror the results from our first set of matrices. This matches what we would expect in result because without pivoting the algorithm is always going to fail. This is due to the 0 entry on the first diagonal element making the matrix singular and non computable. This however is not the case in both partial and complete pivoting, where the element of highest magnitude in the active column or matrix is moved to evade this issue. Overall this leads us to believe that the algorithm works for matrices of this structure when using partial and complete pivoting.

Figure 2: Anti-Diagonal Matrix Metrics

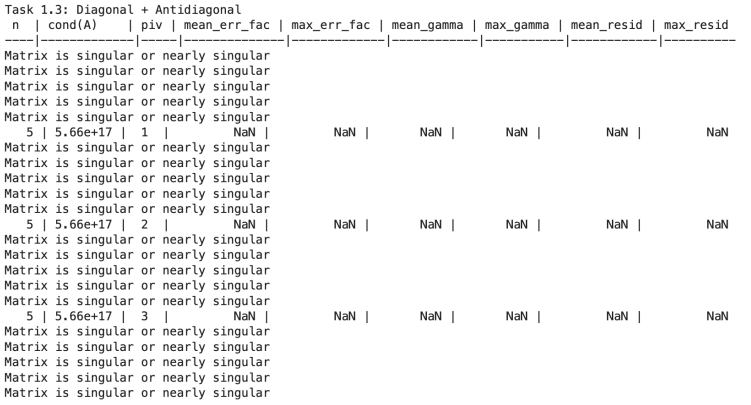


```
Task 1.2: Antidiagonal
 n  | cond(A)    | piv | mean_err_fac | max_err_fac | mean_gamma | max_gamma | mean_resid | max_resid
----|------------|-----|--------------|-------------|------------|-----------|------------|----------
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
   5 | 5.00e+00 |  1  |         NaN |         NaN |       NaN |       NaN |         NaN |         NaN
   5 | 5.00e+00 |  2  |    0.00e+00 |    0.00e+00 |      1.00 |      1.00 |    0.00e+00 |    0.00e+00
   5 | 5.00e+00 |  3  |    0.00e+00 |    0.00e+00 |      1.00 |      1.00 |    0.00e+00 |    0.00e+00
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
  10 | 1.00e+01 |  1  |         NaN |         NaN |       NaN |       NaN |         NaN |         NaN
  10 | 1.00e+01 |  2  |    0.00e+00 |    0.00e+00 |      1.00 |      1.00 |    0.00e+00 |    0.00e+00
  10 | 1.00e+01 |  3  |    0.00e+00 |    0.00e+00 |      1.00 |      1.00 |    0.00e+00 |    0.00e+00
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
  15 | 1.50e+01 |  1  |         NaN |         NaN |       NaN |       NaN |         NaN |         NaN
  15 | 1.50e+01 |  2  |    0.00e+00 |    0.00e+00 |      1.00 |      1.00 |    0.00e+00 |    0.00e+00
  15 | 1.50e+01 |  3  |    0.00e+00 |    0.00e+00 |      1.00 |      1.00 |    0.00e+00 |    0.00e+00
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
  20 | 2.00e+01 |  1  |         NaN |         NaN |       NaN |       NaN |         NaN |         NaN
  20 | 2.00e+01 |  2  |    0.00e+00 |    0.00e+00 |      1.00 |      1.00 |    0.00e+00 |    0.00e+00
  20 | 2.00e+01 |  3  |    0.00e+00 |    0.00e+00 |      1.00 |      1.00 |    0.00e+00 |    0.00e+00
```

5.3 Diagonal + Anti-Diagonal Matrix Testing Results

Looking at results from our third set of structured matrices, we can that algorithm is not producing any results, why? One thing we can notice from the results, was that our tolerance factor was hit for every method within our algorithm meaning that the matrix was not able able to be factorized and if it was, our algorithm would have output poor results. I even attempted to lower the lower but each time I was unable to yield any results. From this information we conclude that several rows are linearly dependent, leading to small pivot points which our algorithm is protected against, yielding the results we observed.

Figure 3: Diagonal + Anti-Diagonal Matrix Metrics



```
Task 1.3: Diagonal + Antidiagonal
 n  | cond(A)    | piv | mean_err_fac | max_err_fac | mean_gamma | max_gamma | mean_resid | max_resid
----|------------|-----|--------------|-------------|------------|-----------|------------|----------
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
   5 | 5.66e+17 |  1  |         NaN |         NaN |       NaN |       NaN |         NaN |         NaN
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
   5 | 5.66e+17 |  2  |         NaN |         NaN |       NaN |       NaN |         NaN |         NaN
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
   5 | 5.66e+17 |  3  |         NaN |         NaN |       NaN |       NaN |         NaN |         NaN
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
Matrix is singular or nearly singular
```

5.4 Unit Lower Triangular Matrix Testing Results

Looking at the results from our 4th set of structured matrices, a unit lower triangular matrix yields very good error results for every method tested on it. This matches what we would expect theoretically as this case should yield $A = L * I$, where U is now the identity matrix, and no pivoting is needed. This can also help to explain why our growth factor remains consistent throughout each method (methods 1,2,3) because no pivoting is needed. The growth factor also grows at a modest pace in this algorithm due to the restrictions of floating point arithmetic.

Figure 4: Unit Lower Triangular Matrix Metrics

Task 1.4: Unit Lower Triangular

n	cond(A)	piv	mean_err_fac	max_err_fac	mean_gamma	max_gamma	mean_resid	max_resid
5	4.52e+00	1	0.00e+00	0.00e+00	1.25	1.25	0.00e+00	0.00e+00
5	4.52e+00	2	0.00e+00	0.00e+00	1.25	1.25	0.00e+00	0.00e+00
5	4.52e+00	3	0.00e+00	0.00e+00	1.25	1.25	0.00e+00	0.00e+00
10	1.45e+01	1	0.00e+00	0.00e+00	1.35	1.35	8.54e-17	8.54e-17
10	1.45e+01	2	0.00e+00	0.00e+00	1.35	1.35	8.54e-17	8.54e-17
10	1.45e+01	3	0.00e+00	0.00e+00	1.35	1.35	8.54e-17	8.54e-17
15	4.55e+01	1	0.00e+00	0.00e+00	1.70	1.70	6.45e-17	6.45e-17
15	4.55e+01	2	0.00e+00	0.00e+00	1.70	1.70	6.45e-17	6.45e-17
15	4.55e+01	3	0.00e+00	0.00e+00	1.70	1.70	6.45e-17	6.45e-17
20	1.73e+02	1	0.00e+00	0.00e+00	1.77	1.77	1.32e-16	1.32e-16
20	1.73e+02	2	0.00e+00	0.00e+00	1.77	1.77	1.32e-16	1.32e-16
20	1.73e+02	3	0.00e+00	0.00e+00	1.77	1.77	1.32e-16	1.32e-16
25	2.67e+02	1	0.00e+00	0.00e+00	1.89	1.89	1.16e-16	1.16e-16
25	2.67e+02	2	0.00e+00	0.00e+00	1.89	1.89	1.16e-16	1.16e-16
25	2.67e+02	3	0.00e+00	0.00e+00	1.89	1.89	1.16e-16	1.16e-16
30	2.12e+02	1	0.00e+00	0.00e+00	1.99	1.99	7.50e-17	7.50e-17
30	2.12e+02	2	0.00e+00	0.00e+00	1.99	1.99	7.50e-17	7.50e-17
30	2.12e+02	3	0.00e+00	0.00e+00	1.99	1.99	7.50e-17	7.50e-17
35	1.66e+03	1	0.00e+00	0.00e+00	1.96	1.96	2.58e-16	2.58e-16
35	1.66e+03	2	0.00e+00	0.00e+00	1.96	1.96	2.58e-16	2.58e-16
35	1.66e+03	3	0.00e+00	0.00e+00	1.96	1.96	2.58e-16	2.58e-16
40	1.12e+03	1	0.00e+00	0.00e+00	2.32	2.32	2.94e-16	2.94e-16
40	1.12e+03	2	0.00e+00	0.00e+00	2.32	2.32	2.94e-16	2.94e-16
40	1.12e+03	3	0.00e+00	0.00e+00	2.32	2.32	2.94e-16	2.94e-16
45	7.41e+03	1	0.00e+00	0.00e+00	2.49	2.49	2.98e-16	2.98e-16
45	7.41e+03	2	0.00e+00	0.00e+00	2.49	2.49	2.98e-16	2.98e-16
45	7.41e+03	3	0.00e+00	0.00e+00	2.49	2.49	2.98e-16	2.98e-16
50	1.33e+04	1	0.00e+00	0.00e+00	2.62	2.62	2.63e-16	2.63e-16
50	1.33e+04	2	0.00e+00	0.00e+00	2.62	2.62	2.63e-16	2.63e-16
50	1.33e+04	3	0.00e+00	0.00e+00	2.62	2.62	2.63e-16	2.63e-16

5.5 Lower Triangular Matrix Testing Results

Similar to the section above we are evaluating a lower triangular matrix structure, with the difference being the diagonal entries being greater than zero but not all one, and elements below the diagonal are greater than one. The results from our testing yielded a low error for the mean and max residual and factorization error along with a slowly growing growth factor. One pattern we can notice is that when preforming no pivoting, our growth factor stays consistent at one but with the other methods it stays equal for each iteration while growing at the same rate as well. We can attribute this to no large multiplier being introduced. For the other methods, even though it is not necessary, the arithmetic within the process of comparisons yields a higher growth rate.

Figure 5: Lower Triangular Matrix Metrics

Task 1.5: Lower Triangular

n	cond(A)	piv	mean_err_fac	max_err_fac	mean_gamma	max_gamma	mean_resid	max_resid
5	1.71e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
5	1.71e+01	2	1.65e-17	1.65e-17	1.29	1.29	1.20e-16	1.20e-16
5	1.71e+01	3	0.00e+00	0.00e+00	1.29	1.29	1.20e-16	1.20e-16
10	5.33e+01	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
10	5.33e+01	2	3.59e-17	3.59e-17	1.32	1.32	1.52e-16	1.52e-16
10	5.33e+01	3	3.73e-17	3.73e-17	1.32	1.32	1.69e-16	1.69e-16
15	1.08e+02	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
15	1.08e+02	2	6.20e-17	6.20e-17	1.34	1.34	2.78e-16	2.78e-16
15	1.08e+02	3	5.48e-17	5.48e-17	1.34	1.34	1.93e-16	1.93e-16
20	1.82e+02	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
20	1.82e+02	2	4.91e-17	4.91e-17	1.35	1.35	2.13e-16	2.13e-16
20	1.82e+02	3	6.00e-17	6.00e-17	1.35	1.35	2.78e-16	2.78e-16
25	2.75e+02	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
25	2.75e+02	2	6.03e-17	6.03e-17	1.36	1.36	1.86e-16	1.86e-16
25	2.75e+02	3	8.81e-17	8.81e-17	1.36	1.36	1.43e-16	1.43e-16
30	3.87e+02	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
30	3.87e+02	2	5.01e-17	5.01e-17	1.37	1.37	3.93e-16	3.93e-16
30	3.87e+02	3	7.33e-17	7.33e-17	1.37	1.37	1.75e-16	1.75e-16
35	5.18e+02	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
35	5.18e+02	2	5.60e-17	5.60e-17	1.37	1.37	3.62e-16	3.62e-16
35	5.18e+02	3	8.48e-17	8.48e-17	1.37	1.37	2.61e-16	2.61e-16
40	6.68e+02	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
40	6.68e+02	2	6.01e-17	6.01e-17	1.38	1.38	2.61e-16	2.61e-16
40	6.68e+02	3	7.84e-17	7.84e-17	1.38	1.38	2.40e-16	2.40e-16
45	8.37e+02	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
45	8.37e+02	2	6.37e-17	6.37e-17	1.38	1.38	4.09e-16	4.09e-16
45	8.37e+02	3	8.75e-17	8.75e-17	1.38	1.38	3.31e-16	3.31e-16
50	1.02e+03	1	0.00e+00	0.00e+00	1.00	1.00	0.00e+00	0.00e+00
50	1.02e+03	2	6.10e-17	6.10e-17	1.38	1.38	4.22e-16	4.22e-16
50	1.02e+03	3	8.89e-17	8.89e-17	1.38	1.38	2.43e-16	2.43e-16

5.6 Tri-Diagonally, Diagonally Dominant Matrix Testing Results

The structure for the 6th set of Matrices requires that non zero elements are only on the diagonal and first super and sub diagonal as well. additionally the main diagonal will have value greater than the super and sub diagonal. This structure leads to a well conditioned matrix and is confirmed by our testing results where we have very low error across the board along with a consistent growth factor of one for each trial and iteration.

Figure 6: Tri-Diagonally, Diagonally Dominant Matrix Metrics

Task 1.6: Tridiagonal (Diagonally Dominant)

n	cond(A)	piv	mean_err_fac	max_err_fac	mean_gamma	max_gamma	mean_resid	max_resid
5	2.53e+00	1	0.00e+00	0.00e+00	1.00	1.00	1.20e-16	1.20e-16
5	2.53e+00	2	0.00e+00	0.00e+00	1.00	1.00	1.20e-16	1.20e-16
5	2.53e+00	3	0.00e+00	0.00e+00	1.05	1.05	0.00e+00	0.00e+00
10	2.84e+00	1	1.88e-17	1.88e-17	1.00	1.00	9.05e-17	9.05e-17
10	2.84e+00	2	1.88e-17	1.88e-17	1.00	1.00	9.05e-17	9.05e-17
10	2.84e+00	3	3.35e-18	3.35e-18	1.08	1.08	4.53e-17	4.53e-17
15	2.92e+00	1	1.86e-17	1.86e-17	1.00	1.00	7.57e-17	7.57e-17
15	2.92e+00	2	1.86e-17	1.86e-17	1.00	1.00	7.57e-17	7.57e-17
15	2.92e+00	3	4.31e-18	4.31e-18	1.09	1.09	6.31e-18	6.31e-18
20	2.96e+00	1	1.86e-17	1.86e-17	1.00	1.00	8.29e-17	8.29e-17
20	2.96e+00	2	1.86e-17	1.86e-17	1.00	1.00	8.29e-17	8.29e-17
20	2.96e+00	3	4.42e-18	4.42e-18	1.09	1.09	3.32e-17	3.32e-17
25	2.97e+00	1	1.85e-17	1.85e-17	1.00	1.00	7.65e-17	7.65e-17
25	2.97e+00	2	1.85e-17	1.85e-17	1.00	1.00	7.65e-17	7.65e-17
25	2.97e+00	3	4.36e-18	4.36e-18	1.09	1.09	5.97e-18	5.97e-18
30	2.98e+00	1	1.85e-17	1.85e-17	1.00	1.00	4.47e-17	4.47e-17
30	2.98e+00	2	1.85e-17	1.85e-17	1.00	1.00	4.47e-17	4.47e-17
30	2.98e+00	3	4.54e-18	4.54e-18	1.10	1.10	6.59e-17	6.59e-17
35	2.98e+00	1	1.85e-17	1.85e-17	1.00	1.00	7.42e-17	7.42e-17
35	2.98e+00	2	1.85e-17	1.85e-17	1.00	1.00	7.42e-17	7.42e-17
35	2.98e+00	3	4.45e-18	4.45e-18	1.10	1.10	5.63e-17	5.63e-17
40	2.99e+00	1	1.85e-17	1.85e-17	1.00	1.00	6.20e-17	6.20e-17
40	2.99e+00	2	1.85e-17	1.85e-17	1.00	1.00	6.20e-17	6.20e-17
40	2.99e+00	3	4.45e-18	4.45e-18	1.10	1.10	5.97e-17	5.97e-17
45	2.99e+00	1	1.85e-17	1.85e-17	1.00	1.00	6.50e-17	6.50e-17
45	2.99e+00	2	1.85e-17	1.85e-17	1.00	1.00	6.50e-17	6.50e-17
45	2.99e+00	3	4.52e-18	4.52e-18	1.10	1.10	5.49e-17	5.49e-17
50	2.99e+00	1	1.85e-17	1.85e-17	1.00	1.00	8.83e-17	8.83e-17
50	2.99e+00	2	1.85e-17	1.85e-17	1.00	1.00	8.83e-17	8.83e-17
50	2.99e+00	3	4.50e-18	4.50e-18	1.10	1.10	5.49e-17	5.49e-17

5.7 Growth Factor Matrix Results

For the seventh set if structured matrices, we look into a matrix with ones on the diagonal and in every position of the last column, additionally we have values of -1 underneath the unit diagonal. The purpose of testing a matrix like this is to analyze the growth factor. We are able to notice from our results that the growth factor blows up astronomically quick when using the first and second method within our algorithm, but when we use the complete pivoting method our growth factor remains stable and grows at a slow pace in comparison. We can tell from this that row permutations are not enough to maintain numerical stability and the more robust method (complete pivoting) is better for situations like this.

Figure 7: Growth Factor Matrix Metrics

Task 1.7: Growth Factor Matrix

n	cond(A)	piv	mean_err_fac	max_err_fac	mean_gamma	max_gamma	mean_resid	max_resid
5	2.22e+00	1	0.00e+00	0.00e+00	11.26	11.26	0.00e+00	0.00e+00
5	2.22e+00	2	0.00e+00	0.00e+00	11.26	11.26	0.00e+00	0.00e+00
5	2.22e+00	3	0.00e+00	0.00e+00	2.97	2.97	0.00e+00	0.00e+00
10	4.38e+00	1	0.00e+00	0.00e+00	190.40	190.40	0.00e+00	0.00e+00
10	4.38e+00	2	0.00e+00	0.00e+00	190.40	190.40	0.00e+00	0.00e+00
10	4.38e+00	3	0.00e+00	0.00e+00	3.06	3.06	0.00e+00	0.00e+00
15	6.60e+00	1	0.00e+00	0.00e+00	4052.70	4052.70	0.00e+00	0.00e+00
15	6.60e+00	2	0.00e+00	0.00e+00	4052.70	4052.70	0.00e+00	0.00e+00
15	6.60e+00	3	0.00e+00	0.00e+00	3.05	3.05	0.00e+00	0.00e+00
20	8.83e+00	1	0.00e+00	0.00e+00	96912.46	96912.46	0.00e+00	0.00e+00
20	8.83e+00	2	0.00e+00	0.00e+00	96912.46	96912.46	0.00e+00	0.00e+00
20	8.83e+00	3	0.00e+00	0.00e+00	3.05	3.05	0.00e+00	0.00e+00
25	1.11e+01	1	0.00e+00	0.00e+00	2473963.22	2473963.22	0.00e+00	0.00e+00
25	1.11e+01	2	0.00e+00	0.00e+00	2473963.22	2473963.22	0.00e+00	0.00e+00
25	1.11e+01	3	0.00e+00	0.00e+00	3.04	3.04	0.00e+00	0.00e+00
30	1.33e+01	1	0.00e+00	0.00e+00	65830866.53	65830866.53	0.00e+00	0.00e+00
30	1.33e+01	2	0.00e+00	0.00e+00	65830866.53	65830866.53	0.00e+00	0.00e+00
30	1.33e+01	3	0.00e+00	0.00e+00	3.04	3.04	0.00e+00	0.00e+00
35	1.56e+01	1	0.00e+00	0.00e+00	1802644652.01	1802644652.01	0.00e+00	0.00e+00
35	1.56e+01	2	0.00e+00	0.00e+00	1802644652.01	1802644652.01	0.00e+00	0.00e+00
35	1.56e+01	3	0.00e+00	0.00e+00	3.03	3.03	0.00e+00	0.00e+00
40	1.78e+01	1	0.00e+00	0.00e+00	50407485251.57	50407485251.57	0.00e+00	0.00e+00
40	1.78e+01	2	0.00e+00	0.00e+00	50407485251.57	50407485251.57	0.00e+00	0.00e+00
40	1.78e+01	3	0.00e+00	0.00e+00	3.03	3.03	0.00e+00	0.00e+00
45	2.01e+01	1	0.00e+00	0.00e+00	1432280894243.95	1432280894243.95	0.00e+00	0.00e+00
45	2.01e+01	2	0.00e+00	0.00e+00	1432280894243.95	1432280894243.95	0.00e+00	0.00e+00
45	2.01e+01	3	0.00e+00	0.00e+00	3.03	3.03	0.00e+00	0.00e+00
50	2.23e+01	1	0.00e+00	0.00e+00	412133156271127.97	412133156271127.97	0.00e+00	0.00e+00
50	2.23e+01	2	0.00e+00	0.00e+00	412133156271127.97	412133156271127.97	0.00e+00	0.00e+00
50	2.23e+01	3	0.00e+00	0.00e+00	3.02	3.02	0.00e+00	0.00e+00

5.8 Symmetric Positive Definite Matrix Testing Results

The final set of structured matrices to test are symmetric positive definite structured and generated from a lower triangular matrix. From the results we see below, the error and growth factor metrics are both stable. More importantly we want to analyze the relationship between L, U, and L_{tilde}. For a positive symmetric definite matrix $A = L_{tilde} * L_{tilde}^T$ we are still able to run the algorithm. In this case, $U = DL^T$, where D is the diagonal containing the diagonal entries of U. From this the the matrix $L_{tilde} = LD^{1/2}$. therefore our decomposition can then be interpreted as a scaled version, with L providing the same elimination structure as $D^{1/2}$, which would appropriately scale to recover L_{tilde}

Figure 8: Symmetric Positive Definite Matrix Metrics

Task 1.8: Symmetric Positive Definite Matrix (A = L~L^T)

n	cond(A)	piv	mean_err_fac	max_err_fac	mean_gamma	max_gamma	mean_resid	max_resid
5	4.93e+01	1	0.00e+00	0.00e+00	1.00	1.00	1.20e-16	1.20e-16
5	4.93e+01	2	4.66e-17	4.66e-17	2.12	2.12	1.47e-16	1.47e-16
5	4.93e+01	3	1.16e-17	1.16e-17	1.00	1.00	6.69e-17	6.69e-17
10	4.28e+02	1	0.00e+00	0.00e+00	1.00	1.00	1.67e-16	1.67e-16
10	4.28e+02	2	2.33e-16	2.33e-16	3.37	3.37	1.24e-16	1.24e-16
10	4.28e+02	3	2.54e-17	2.54e-17	1.00	1.00	8.47e-17	8.47e-17
15	1.80e+03	1	0.00e+00	0.00e+00	1.00	1.00	2.45e-16	2.45e-16
15	1.80e+03	2	1.26e-16	1.26e-16	2.14	2.14	2.08e-16	2.08e-16
15	1.80e+03	3	2.00e-17	2.00e-17	1.00	1.00	2.95e-16	2.95e-16
20	5.38e+03	1	0.00e+00	0.00e+00	1.00	1.00	3.60e-16	3.60e-16
20	5.38e+03	2	1.03e-16	1.03e-16	1.97	1.97	7.29e-16	7.29e-16
20	5.38e+03	3	2.46e-17	2.46e-17	1.00	1.00	3.89e-16	3.89e-16
25	1.28e+04	1	0.00e+00	0.00e+00	1.00	1.00	5.05e-16	5.05e-16
25	1.28e+04	2	1.29e-16	1.29e-16	1.82	1.82	3.13e-16	3.13e-16
25	1.28e+04	3	3.53e-17	3.53e-17	1.00	1.00	3.90e-16	3.90e-16
30	2.58e+04	1	0.00e+00	0.00e+00	1.00	1.00	5.54e-16	5.54e-16
30	2.58e+04	2	1.28e-16	1.28e-16	1.95	1.95	6.31e-16	6.31e-16
30	2.58e+04	3	2.85e-17	2.85e-17	1.00	1.00	3.87e-16	3.87e-16
35	4.71e+04	1	0.00e+00	0.00e+00	1.00	1.00	4.39e-16	4.39e-16
35	4.71e+04	2	1.45e-16	1.45e-16	1.99	1.99	4.30e-16	4.30e-16
35	4.71e+04	3	3.20e-17	3.20e-17	1.00	1.00	4.85e-16	4.85e-16
40	7.98e+04	1	0.00e+00	0.00e+00	1.00	1.00	3.83e-16	3.83e-16
40	7.98e+04	2	1.58e-16	1.58e-16	1.73	1.73	5.54e-16	5.54e-16
40	7.98e+04	3	3.49e-17	3.49e-17	1.00	1.00	5.73e-16	5.73e-16
45	1.28e+05	1	0.00e+00	0.00e+00	1.00	1.00	5.09e-16	5.09e-16
45	1.28e+05	2	1.19e-16	1.19e-16	1.82	1.82	7.66e-16	7.66e-16
45	1.28e+05	3	3.83e-17	3.83e-17	1.00	1.00	9.00e-16	9.00e-16
50	1.94e+05	1	0.00e+00	0.00e+00	1.00	1.00	4.97e-16	4.97e-16
50	1.94e+05	2	1.27e-16	1.27e-16	1.84	1.84	5.69e-16	5.69e-16
50	1.94e+05	3	3.61e-17	3.61e-17	1.00	1.00	4.21e-16	4.21e-16

5.9 Conditioning Number Analysis

The condition number $K_2(A)$ was also computed for each structured matrix to evaluate how conditioning affects numerical stability during the various factorization methods. Matrices that were diagonally dominant or triangular exhibited small condition numbers. This helps us indicate that they are well conditioned, producing stable factorizations with very minimal errors. Matrix structures such as combining diagonal and anti diagonal elements or the growth factor test matrix produced very large condition numbers pointing towards the matrix being ill conditioning. Ill condition matrices yielded high factorization and residual errors, or the matrix was unable to be computed due to computations bounds implemented into the algorithm.

6 Conclusion

Using the results collected from testing we can indicate that our algorithm is correctly producing the Matrix factorization using no pivoting, partial pivoting, and complete pivoting. We can ensure these results due to the low error received through testing and the validation of our theoretical outputs being computed. Although the algorithm is limited by certain edge cases, we can be certain that it will run properly for matrices that have a structure we can preserve. Most of the error in the computations can be attributed to floating point arithmetic and its limitations when computing large matrices and their products. The metrics evaluated also helped to give us an insight into how the matrices we being affected through the factorization and what methods have certain advantages under certain restrictions

7 Program Files

The assignment includes 4 files: LUdense, TestDriverProgram3, Lvsolve , and Uvsolve . The test driver runs every test at once and takes less than 30 seconds to do so.