Programming Assignment 2

Daniel Mazus

October 16, 2024

I Executive Summary

Developed a set of routines to compute the equation $Ax = b$ where A is decomposed into L and U and then put through two solvers, a forward substitution and then backward substitution, to numerically compute x. Computed results for both the decomposition and the solvers are compared to true results for randomly generated non-singular matrices. The norms were taken and then relatively computed by division which then were plotted over multiple dimensions to show error in all three routines.

II Statement of Problem

The problem given is to evaluate a given full-rank matrix A and view the transformation of this matrix into an upper triangular matrix U. This is done by using LU Factorization with no pivoting and with partial pivoting. The discussion of complete pivoting will also be held, but was not implemented into the algorithm. So, the evaluation that is wanted to go over is factoring A into LU where $A = LU$, without pivoting and $P_rA = LU$ for partial pivoting. We start by looking at empirical cases where we can compute and predict what will happen theoretically and then implement into the algorithm to compare the prediction with the output. This will then run into randomly generating L and U where both are non-singular and well-conditioned. The output will not only be a matrix comprised of L and U but also a vector containing the permutations if any had occurred.

III Description of Mathematics

Before implementing the algorithm, we first go over the mathematics of LU factorization and upper and lower solves and then describe how this looks in each empirical example. We want to view the transformation of $A \to U$ where A is a full-rank matrix with $A \in \mathbb{R}^{nxn}$ and U a non-singular matrix with $U \in \mathbb{R}^{nxn}$. There are three different variations of using LU factorization, no pivoting, partial pivoting, and complete pivoting. We start with no pivoting and work up to complete pivoting.

III.I No Pivoting for LU Factorization

We consider $A \in \mathbb{R}^{nxn}$ and we want to decompose A into L and U where $L \in \mathbb{R}^{nxn}$ is a unit lower triangular matrix that is non-singular and $U \in \mathbb{R}^{nxn}$ is an upper triangular matrix that is also non-singular. Looking at the equation $Ax = b$, we want to apply a transformation T^{-1} to both sides to get:

$$T^{-1}Ax = T^{-1}b \implies (T^{-1}A)x = c \implies Ux = c$$

As we see this transformation on A should lead to U, defining the factorization of A. We also let $T^{-1}b = c$. T^{-1} is defined by elementary matrices from some chosen class. Here we choose the elementary matrices to eliminate the elements below the diagonal on A, also known as *Gauss Transforms.* We now let $T^{-1} = M^{-1}$ where M^{-1} will eliminate the elements below the diagonal in the given row, i.

$$M_i^{-1} = I - l_i e_i^T$$

As we can see, M still retains the same size as A, $M \in \mathbb{R}^{nxn}$ and $l_i \in \mathbb{R}^n$. l is structured so that the elements in the current column below the (i, i) spot will eliminate the elements in A in the corresponding column below the diagonal element. The elements that do the elimination are defined as λ where $\lambda = e_j^T l_i$ and negated in M. To lay this out in an example for $i = 1$ and $n = 4$, we have (*Set 4: Linear Systems Part 2*), $A^{(1)} = M_1^{-1}A$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -\lambda_{21} & 1 & 0 & 0 \\ -\lambda_{31} & 0 & 1 & 0 \\ -\lambda_{41} & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{pmatrix}$$

$$= \left(\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 \\ \lambda_{21} \\ \lambda_{31} \\ \lambda_{41} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \right) \begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{pmatrix}$$

$$= \begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{pmatrix} - \begin{pmatrix} 0 \\ \lambda_{21} \\ \lambda_{31} \\ \lambda_{41} \end{pmatrix} \begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{41} \end{pmatrix}$$

We then go ahead and multiply out and subtract to finally get:

$$A^{(1)} = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{41} \\ 0 & \alpha_{22} - \lambda_{21}\alpha_{12} & \alpha_{23} - \lambda_{21}\alpha_{13} & \alpha_{24} - \lambda_{21}\alpha_{14} \\ 0 & \alpha_{32} - \lambda_{31}\alpha_{12} & \alpha_{33} - \lambda_{31}\alpha_{13} & \alpha_{34} - \lambda_{31}\alpha_{14} \\ 0 & \alpha_{42} - \lambda_{41}\alpha_{12} & \alpha_{43} - \lambda_{41}\alpha_{13} & \alpha_{44} - \lambda_{41}\alpha_{14} \end{pmatrix} = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ 0 & \tilde{\alpha}_{22} & \tilde{\alpha}_{23} & \tilde{\alpha}_{24} \\ 0 & \tilde{\alpha}_{32} & \tilde{\alpha}_{33} & \tilde{\alpha}_{34} \\ 0 & \tilde{\alpha}_{42} & \tilde{\alpha}_{43} & \tilde{\alpha}_{44} \end{pmatrix}$$

The matrix that is resulted from here is $A^{(1)}$. For $n = 4$ we would do this 3 more times to get the 0's in the elements below the diagonal. The sub-matrix that has its dimensions reduced by 1, we call the "active matrix". This will be useful for coding as we will only have to look at the "active matrix". Moving backwards, we see that the first row is untouched in $A^{(1)}$ and all that is updated is this sub-matrix. Since the first row is untouched and we know the corresponding column is zeroed-out, we only have to deal with the sub-matrix of the given A. This leads to seeing that the λ_{j1}'s are computed by taking α_{j1} and dividing by the diagonal element α_{ii} for the given i, i.e.,

$$\lambda_{j1} = \frac{\alpha_{j1}}{\alpha_{11}} \text{ for } j = 2, 3, 4$$

This is for the given example but we see that the general would result:

$$\lambda_{ji} = \frac{\alpha_{ji}}{\alpha_{ii}} \text{ for } j = 2, \ldots, n$$

We see that this transformation cannot continue if $\alpha_{ii} = 0$. As we are looking at no pivoting, this factorization depends on what the diagonal, pivot, elements are. Numerically, this will also cause problems if α_{ii} is near-zero, which will be discussed in how we generate our testing problems below. This can sometimes be avoided if partial and complete pivoting are implemented instead. After computing each M_i^{-1}, we have that $M_i^{-1}A = U$. Now that we have what M_i^{-1} looks like, if we want to compute $A = LU$, we can move the M_i^{-1} over and multiply by U to get,

$$A = M_1 M_2 M_3 \cdots M_{n-1} U$$

Since M_i is made up of elementary matrices that eliminated the elements of A when the inverse was taken, the result of multiplying them together gives L, the unit lower triangular matrix where the elements of L are the λ's that were computed for the *Gauss Transform* below the diagonal. Now that we have the LU factorization of A, we can now use L and U to solve the equation $Ax = b$ for x. We first would solve $Ly = b$, *forward substitution*, and then $Ux = y$, *backward substitution*.

III.II Partial Pivoting

Partial Pivoting is crucial because now we can permute rows to now have a non-zero element in the diagonal, if there was a zero in the diagonal. This now gives us the equation:

$$A = LU \implies PA = LU$$

Where P is a permutation matrix that is applied to A to give LU. We accomplish this factorization by taking the largest element in the current column and swapping that row with the current (i, i) element's row. After this is done for the current position, we then compute the *Gauss Elimination* the same way as before and this permutation is repeated for each sub-matrix we get. This process updates the permutation matrix with which rows were swapped which is how we get,

$$PA = LU$$

III.III Complete Pivoting

In complete pivoting, we now have the equation,

$$P_r A P_c = LU$$

where P_r are the row permutations and P_c are the column permutations. We are no longer looking for the maximal element in the current column of i, but rather the entire "active matrix". This element is again moved to the (i, i) element and we keep track of which rows were swapped in P_r and which columns were swapped in P_c.

III.IV Lower Triangular Solve for $Ly = b$

The next step in solving $Ax = b$ after now computing $A = LU$, is doing the triangular solve for $Ly = b$ for y. As we are given $b \in \mathbb{R}^n$ and $L \in \mathbb{R}^{n \times n}$, the output of y will be a vector of $y \in \mathbb{R}^n$. This triangular solve is also known as *forward substitution* as we start with a single variable and equation and work our way down the lower triangular matrix using the solved variable from before. If L has elements, $\alpha_{ij} > 0$ for $i > j$, $\alpha_{ii} = 1$, and 0 elsewhere. Also, let y have elements of $y_1, \dots, y_n$ that are unknowns in a column vector. Lastly, let b have elements $\xi_1, \dots, \xi_n$ with $\xi_i \in \mathbb{R}$ for $i = 1, \dots, n$. Then we have the following systems of equations after multiplication:

$$\begin{aligned}
&\alpha_{11} y_1 = \xi_1 \\
&\alpha_{21} y_1 + \alpha_{22} y_2 = \xi_2 \\
&\vdots \\
&\alpha_{n1} y_1 + \alpha_{n2} y_2 + \dots + \alpha_{nn} y_n = \xi_n
\end{aligned}$$

As we can see, the *forward substitution* comes from the idea of working from the first element, down using the elements solved in the previous step to solve for the current step. Doing this, we will want to solve for y_1 by dividing ξ_1 by α_{11}. We then take this y_1 and substitute in the next step for y_1 and then solve for y_2 as we now have all knowns except y_2. We do this for n times to get the resulting vector y. As we have y we now use this in the Upper Triangular Solve for x.

III.V Upper Triangular Solve for $Ux = y$

Using the y in the previous step, we now have the Upper Triangular Solve $Ux = y$ where we want to solve for x to get the solution to $Ax = b$. We know U from LU factorization output and we just solved for y. We now use this solve, also known as backward substitution. Instead of working from the top of the matrix, the first row, we work from the last row up since we will start with one variable equation which we can find the solution of and use that solution in the next step and work backwards up U to solve x. We now have the elements of U to be $\alpha_{ij} > 0$ for $i \leq j$ and $\alpha_{ij} = 0$ for $i > j$. This gives us an Upper Triangular Matrix. The elements of y have the form of $y_1, \dots, y_n$ with $y_i \in \mathbb{R}$ for $i = 1, \dots, n$ and has the values of the Lower Triangular Solve above. Lastly, let x have elements of $x_1, \dots, x_n$ with $x_i \in \mathbb{R}$ for $i = 1, \dots, n$ and are unknown values in a column vector. This results in the following systems of equations:

$$\begin{aligned}
\alpha_{11}x_1 + \alpha_{12}x_2 + \cdots + \alpha_{1n}x_n &= y_1 \\
\alpha_{22}x_2 + \cdots + \alpha_{2n}x_n &= y_2 \\
\ddots \quad &\vdots \\
\alpha_{n-1,n-1}x_{n-1} + \alpha_{n-1,n}x_n &= y_{n-1} \\
\alpha_{nn}x_n &= y_n
\end{aligned}$$

As we can see, we have the α_{nn} value from U, and y_n value from the previous solve. We can solve for x_n and then use the result to compute the next step up or the previous step, hence *backward substitution*. We start from the bottom and work our way back up instead of starting from the first row and working down the the n'th row. This occurs because at the n'th row, we have an equation with only one unknown, which is able to be solved. After doing this process, we will have the solution vector x which is the solution to $Ax = b$ when A and b are given.

IV Description of Algorithm and Implementation

IV.I Overview of Algorithm

The task was to evaluate the transformation given a full-rank matrix A where we can decompose A into L and U and then pass the decomposed matrix through the solvers. The first step was to develop the LU factorization function. Briefly, the function takes in a matrix A, an n, and routine number and outputs the corresponding matrix depending on routine given. The solvers take in a matrix and vector which then outputs a vector after completing the corresponding solve. Only briefly bring this up since each routine will be gone into more detail below.

IV.I.1 LU Factorization Routine

The first thought of being built into the this function was having a user input whether partial pivoting would like to be done or no pivoting. We first start with the latter case, no pivoting, as this is the simpler of the two.

1. As we have seen from the mathematics, we know we first have to check to make sure the diagonal element is under a certain threshold, 10^{-12} which is near zero. In the algorithm, this is known as *alpha* and above we see it is also α. Since we know that we are not doing pivoting, this is crucial as this will not allow LU factorization to continue as this is near zero, or is 0, and will either be poorly conditioned or break, respectively. If this does occur, there is a warning that is printed to the user that tells them that the value is below the threshold in the diagonal and LU factorization cannot continue. This returns $None$. After checking the diagonal element, (i, i), and if the threshold is not hit, the function then continues into an inner for loop which is only looped over the rows below the current i. There is then a calculation and storage of the λ's which is then used in an intermediate loop that performs the reduction on the "active matrix". We take the elements of the "active matrix" and update them with the λ's that were calculated and stored. This is seen in the mathematics portion of the document. Once this is done for the matrix, we return A which has L and U stored in a single $2D$-array where the 1's and 0's for L and U are not stored.

2. Now if the user wants to run partial pivoting, they will enter 2 for routine when prompted. This will let the function know which routine to run. The start of the function comes from initializing a P which is a Permutation vector that stores the indices of the rows. Now walking step by step we see:

   (a) Step 1: Inside the first for loop we initialize the *max_row* equal to i along with the *max_value* to the absolute value of the diagonal element.

   (b) Step 2: A second for loop now iterates through the rows of the current column to see if the absolute value of the element is greater than the *max_value* initialized and if it is, we update *max_value* equal to that element and set the row to that row. There is then an if statement that checks to see if the *max_row* value is equal to 0 and if it is, a warning is issued, exits, and returns *None* and the Permutation vector P.

(c) Step 3: If there is a maximal element found that is not i, then we swap the rows of A and the indices corresponding with those in P. A print statement then tells the user which rows and were swapped with each other.

(d) Step 4: Now that the pivoting is done for the given row, LU factorization can now proceed. First, α is set to the diagonal element, (i, i) and is then checked to see if the value is under a certain threshold, 10^{-12}, which is near zero. If this happens, LU factorization does not continue as this leads to poor conditioned problems and high error numerically.

(e) Step 5: λ is now computed and stored as before in the no pivoting case.

(f) Step 6: An intermediate loop is used and is used to compute the "active matrix" the same as above.

(g) Step 7: Once this has been completed for the matrix, the output is A decomposed into L and U stored in a single $2D$-array and the permutation vector, P, which is updated with the swapped rows.

IV.I.2 Lv Solver Routine

This is much simpler than the factorization routine, but is needed when solving $Ax = b$. We look at this one first since the output of this function is the input for the Uv solver. The function name for this is *solve_Lb* where it has three input parameters. These parameters are a matrix either randomly generated, a decomposed matrix that is stored in a single $2D$-array, or any matrix that is composed of L and U. We use this solver to solve the equation, $Ly = b$ for y, also known as forward substitution. Walking through the algorithm is:

1. Initialize vector $y \in \mathbb{R}^n$.
2. Set the first element of y equal to the first element of b since L is a unit lower triangular matrix.
3. Initialize a temporary sum variable that is used in the following step to accumulate the sum of the terms.
4. Multiply the corresponding elements of L with the "variables" of y. This is shown in the mathematics section of how forward substitution works. This is then accumulated into the *temp_sum* variable.
5. Solve for the y value by subtracting the b value and the *temp_sum* values. When this is over, this function returns y.

As we can see, this is simpler than the LU factorization routine but takes in the matrix that comes from the LU factorization routine, which is why this is secondly discussed and not first.

IV.II Uv Solver Routine

The Uv solver routine is very similar to the Lv solver routine except that this does backwards substitution rather than forward. The name of this function is *solve_Ux* and again has three parameters as inputs. These include:

- The same matrix that is used in the input of the Lv solver but uses the upper triangular part of the matrix, U.
- The output vector from the Lv solver, which is denoted y.
- An integer n which specifies the dimensions of the matrix and vectors.

Here is the step-by-step explanation of the algorithm:

1. Initialize the vector x which will store the output of $Ux = y$. This x is filled with zeroes and will be updated.

2. Runs through a for loop that starts from the last row, $n-1$, to the first row 0, or in python, -1, hence why this is called backward substitution. A *temp_sum* variable is initialized again to store the values for the next loop.

3. We now look at the values above the diagonal in this loop which now multiplies the corresponding elements in the rows with the "variables" x that can be seen above. This is accumulated in the *temp_sum* variable that was created.

4. Solve for the $x[i]$ value that is computed by:

$$x[i] = \frac{y[i] - temp\_sum}{LU[i][i]}$$

   This is essentially isolating $x[i]$ to be able to store the values in x for the output.

5. Return the x vector which represents the solution in $Ux = y$ and also corresponds to the solution vector in $Ax = b$.

V Description of Experimental Design and Results

V.I Empirical Test Design (Special Cases) Predictions and Results

To make sure the code works correctly and should theoretically match the mathematics, we look at several Empirical Tests, which are special cases of different matrices to see how they will behave. Predictions for the output for L, U, P_r, P_c will be given for each test and the results discussed after the *Generation of Problems/Tester Description* section. All matrices tested and thought of for empirical tests had $n = 5$. The results from the output will be included at the end of the document in the *Extra Figures* section for readability purposes. These will be screenshots of the output for the matrices and growth factors for the corresponding tests.

1. Empirical Test 1: For the first empirical test, we are examining a diagonal matrix $A \in \mathbb{R}^{n\times n}$ where the diagonal elements are positive, i.e. $\alpha_{ii} > 0$. For two examples for $n = 5$ we have that the diagonal elements are $1, 2, 3, 4, 5$ and the other matrix is $5, 4, 3, 2, 1$ and all other positions are 0. These look like:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{pmatrix} \text{ and } A = \begin{pmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

   - With no pivoting, we can easily see that L will be the unit lower triangular matrix with 1's on the diagonal and U will be the Upper triangular matrix with the values of the diagonal elements of A as the values on the diagonal of U.
   - Considering Partial Pivoting, there will be no change as all the elements are already the maximal elements of the given columns for the respective sub-matrices for the corresponding step. This also makes sense because the we have already eliminated the elements below the diagonal, which is what the *Gauss Elimination* accomplishes, so nothing will change, and P_r will just be the identity matrix.
   - Now consider complete pivoting. We would predict that in both matrices, the elements of L will still be the identity matrix but U will always have the elements decreasing on the diagonal, $5, 4, 3, 2, 1$ since we are taking the maximal element out of the whole sub-matrix.
   - Theoretically, the growth factor should be equal to 1 as diagonal matrices with non zero elements on the diagonal will always equal 1. Also, if no pivoting, partial pivoting, and complete pivoting result in the same structure with the same values, then the growth factors should all be the same.

We can see that the prediction holds true to the result of our code. This is shown in Figure 7(a) and Figure 7(b) respectively. This checks to make sure our code works properly to be able to use in our tester for general and random L and U. For the growth factor, we do get 1 for both no pivoting and partial pivoting retaining our theoretical prediction.

2. Empirical Test 2: The second empirical test involved examining anti-diagonal matrices where the entries were positive and all other entries not on the anti-diagonal were 0. The two matrices involved were:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } A = \begin{pmatrix} 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

   - No Pivoting: Clearly, we can tell that these matrices will fail right away with no pivoting because of having 0 in the α_{11} spot. So we cannot continue LU factorization.
   - Partial Pivoting: We will be able to see that for partial pivoting (row pivoting) that these matrices will end up looking the same as the matrices described above which will then give the same results for L and U. P_r in this case will look like the identity matrix
   - Complete Pivoting: Complete pivoting will theoretically be the same case as partial pivoting and A will result in the matrix from Empirical Test 1 with the diagonal elements decreasing, $5, 4, 3, 2, 1$. We can see then that L will be the identity matrix, I, and U will be the upper triangular matrix that holds the values of A. P_r will interchange the rows and P_c will interchange the columns where both will be anti-diagonal identity matrices.
   - We expect the growth factor to be the same as the no pivoting case and partial pivoting case should result in the same values, maybe not in the equal indices, but values of the matrix retain equivalence which will make the growth factor the same. We also expect this value to be 1 as the growth factor for diagonal matrices with non-zeros entries on the diagonal are always equal to 1.

   Again, we see our prediction and theoretical input hold true to the numerical output of running the code. We do not see complete pivoting due to the fact that this was no implemented into the function due to time constraints. We do see for partial pivoting and no pivoting in Figure 7(c) and Figure 7(d). We also see that the growth factor holds up to what we expected it to be, which is 1 due to the diagonal dominance of the matrix.

3. Empirical Test 3: The third empirical test dealt with matrices that looked like an X where one matrix is a diagonal matrix, D, and the other matrix is an anti-diagonal matrix, A, where when added together, form an X. An example of this is,

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } D = \begin{pmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, A + D = \begin{pmatrix} 5 & 0 & 0 & 0 & 1 \\ 0 & 4 & 0 & 2 & 0 \\ 0 & 0 & 6 & 0 & 0 \\ 0 & 4 & 0 & 2 & 0 \\ 5 & 0 & 0 & 0 & 1 \end{pmatrix}$$

   Examining the matrices for no pivoting, we can predict that this will not be able to succeed as $A+D$ is singular. We see that this is singular because of the repeated rows, leading us to have linearly dependent rows, implying a determinant of 0, making this matrix singular. So, $A + D$ cannot be passed through LU factorization with no pivoting. This will actually make Partial Pivoting and Complete Pivoting fail as LU factorization requires the matrix to be non-singular but in any case, this matrix has row dependencies, i.e. linearly dependent rows, making this matrix fail in any case. If we consider the general case, this matrix will only be able to work if there are no identical rows or columns due to the matrix $A + D$ needing to be full-rank.

   After passing this matrix through the LU factorization routine, we can verify that this prediction and theory is true that $A + D$ is singular and fails LU factorization for both no pivoting and partial pivoting for the given examples. We would see that if there would be no identical rows or the matrix was full-rank, LU factorization would be able to happen. There is also the result of when the matrix $A + D$ is full-rank given in Figure 7(e) where we do end up with an output given that no rows were duplicates, or linearly dependent. This holds to our theoretical input of the code succeeding when this holds true.

4. Empirical Test 4: We look at matrices of the form that A is a unit lower triangular matrix where $|\lambda_{ij}| < 1$ for $i > j$ and 0 for $i < j$. Since A is unit lower triangular, this also means that 1's are on the diagonal of A. The example we look at here is:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0.1 & 1 & 0 & 0 & 0 \\ 0.3 & 0.7 & 1 & 0 & 0 \\ 0.7 & 0.9 & 0.4 & 1 & 0 \\ 0.9 & 0.2 & 0.1 & 0.6 & 1 \end{pmatrix}$$

   When considering no pivoting, this matrix should always be able to factor as it is non-singular and L should be the matrix A with U being the identity matrix. This is because as we eliminate the values below the diagonal, those λ's become L and the upper triangular matrix, U becomes the identity. If we were to multiply L and U together, we would see that the result would be A. This would be the same for both partial pivoting and complete pivoting as the largest element is always on the diagonal as the magnitude of the elements below are always less than 1.

   RESULT: In Figure 7(f) we see once again our prediction and theory matches numerical output where when partial pivoting is ran, no permutations are required and U is the identity matrix, I_n for a given n. This is because A is already in the form of unit lower triangular which is the output of LU factorization and the L being spit out.

5. Empirical Test 5: For empirical test 5, we look at matrices that are lower triangular, but not unit lower triangular, i.e. 1 cannot be in the α_{ii} spots. The α_{ii} values must be positive and the values of the elements below the diagonal greater than 1. The example that I examined was:

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

   - Thinking about this test with no pivoting, we can predict that U again will only hold the diagonal elements since all elements above the diagonal are 0. The diagonal elements of U will actually be the diagonal elements A which is 2. We can also see that the elements of L will be just the division of the value in the given spot by the diagonal element. For example, $\lambda_{21} = \frac{3}{2} = 1.5$, $\lambda_{31} = \frac{4}{2} = 2.0$, etc. This holds for each column below the diagonal. This is because...
   - Considering partial pivoting, we first see that we would swap the first and last row. We then proceed with LU factorization. This would lead to having the maximal element in absolute value to be the last row again, so we swap the second row and last row. Each time we swap the last row with the current row due to the maximal element always being in the final row due to the setup of the matrix given. We can then generalize this of matrices of the form given by the example that where the row with the maximal element is, that row will be swapped and the largest element from computing the given step will again be in that row and swaps will continue to happen with that row. We see that L will hold the values to eliminate the elements below the diagonal and U will be the values that result from the factorization. L's elements will also always be positive while U may have both positive and negative elements. P_r takes the last row and then counts down, i.e. instead of $P_r = [0, 1, 2, 3, 4]$, $P_r = [4, 0, 1, 2, 3]$. It changes the order, if using indirect indexing.
   - Looking at complete pivoting, we would see that P_r will still retain the same structure but P_c may look different based on where the maximal elements will be.

In Figure 8(a) we see that the result holds true to our prediction for this empirical task. The given matrix outputs L with elements as the ratios of the element below the diagonal in given column by the diagonal element. Partial pivoting does change this and we see that the elements of L are always positive while U is always negative in our given example. We see that the growth factor also shrinks as the elements are smaller than the elements in the no pivoting case.

6. Empirical Test 6: For empirical test 6, matrices take the form of tridiagonal and non-singular where there are elements in the first super- and sub-diagonals and the diagonal with all other elements being 0. We will then look at if the matrix is strictly diagonally dominant by rows and columns. For this test, I used the matrix:

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

This matrix is non-singular because the determinant is not 0 and is obviously tridiagonal. This is also diagonally dominant by rows and columns.

- No pivoting will always work as this matrix is non-singular and diagonally dominant by rows and columns. L would be made up of the elements below the diagonal divided by the diagonal elements as in earlier examples but only have 1's on the diagonal and one sub-diagonal. This can also be viewed as the ratio's of the sub-diagonal elements to the corresponding diagonal elements. This sub-diagonal will have elements that should get smaller and smaller as the sub-matrices get smaller and smaller through the process which reflects the effects of the matrix being diagonally dominant. U will be made up of the results from the Gaussian Elimination, eliminating zeros below the diagonal. The diagonal elements of U should be the same for the first two entries, exhibiting again, the effects of the diagonal dominance, no matter how big the matrix and then decrease. This will cause U to have one super diagonal where the first entry of this will be 0 and then slowly increase as the process of LU factorization moves on.
- There should be no change in what happens between no pivoting, partial pivoting, and complete pivoting because of the matrix being diagonally dominant throughout, which is saying that the largest element is always on the diagonal no matter what happens through the process. L, U should be the same with P_r and P_c being equal to the identity matrix since no pivoting has occurred.
- When considering the growth factor, since all cases should theoretically be the same so should the growth factor for each of them. This growth factor should also be relatively small due to the elements be small and the matrix diagonally dominant and well-conditioned.

In Figure 8(b) we see the prediction holds true with the behavior of the matrix under LU factorization due to the strict diagonal dominance of A. We note that the elements of the sub-diagonal of L increase as we go farther down the sub-diagonal and the elements of the super-diagonal of U increase closer to the diagonal element with the first two elements being equal to each other. This also occurs in the sub-diagonal of L. As we said no pivoting and partial pivoting should result in the same output, we see that is also true which leads to the growth factor also being equal to each other since no permutations occurred and the elements of both routines are the same.

7. Empirical Test 7: Now consider a matrix A that is again $A \in \mathbb{R}^{n \times n}$ where elements below the diagonal, $\alpha_{ij}, i < j$ are equal to -1, the diagonal elements, $\alpha_{ii} = 1$ and the elements of the last column of the matrix, $\alpha_{in} = 1$. All other elements are set to 0. The example used and analyzed was:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

- For no pivoting, this matrix should be able to be factored in the form given. Analyzing what L will look like is that it will retain the same structure as A with the same values below the diagonal and 0's above the diagonal. Since we have values of 1 in the last column, the LU factorization of this A will lead to an accumulation in the final column of the previous element. Each time the Gaussian Elimination occurs, the last column is update with double the previous step. For example, in the first step of Gaussian elimination for the matrix above, the last column will be 2 except for the first element. For the second iteration of Gaussian Elimination all elements in the last column below the current element will now be 4, or double the previous. This occurs all the way through the elimination giving us the result of the column of U is 2^n.
- Partial Pivoting should result in the same as no pivoting due to the maximal element in the given column will always be the diagonal element. So P_r is the identity matrix again.
- In complete pivoting this will change as we are now taking the maximal element in each sub-matrix where the last column will always be the maximal element. We do not permute rows, this will only permute columns as the maximal element will be in the last column of A and any sub-matrix.
- The growth factor for this should be small due to the elements being small except for the final column. The growth factor should also be the same for no pivoting and partial pivoting as the elements should be identical.

In Figure 8(c) we see that the last column in U does behave as expected with the elements being equal to 2^n where n is the dimension of the square matrix. We also note that L does not change as expected and both no pivoting and partial pivoting are equal setting the growth factor equal to each other since all elements are equal in both cases.

8. Empirical Test 8: For the final empirical test of special cases takes the form of $A \in \mathbb{R}^{n\times n}$ that is symmetric positive definite which is generated from a lower triangular matrix, we call $\tilde{L}$ where the diagonal elements are positive and does not always have to be 1. For symmetric we have $A = \tilde{L}\tilde{L}^T$. We see that A is non-singular as the diagonal elements are greater than zero so the factorization can happen with no pivoting. We can analyze what the L, U, and $\tilde{L}$ will look like after the LU factorization. The example looked at is:

$$A = \begin{pmatrix} 16 & 20 & 4 & 56 & 60 \\ 20 & 169 & 41 & 178 & 99 \\ 4 & 41 & 59 & 118 & 63 \\ 56 & 178 & 118 & 759 & 541 \\ 60 & 99 & 63 & 541 & 758 \end{pmatrix}$$

The relationship between L, U, and $\tilde{L}$ comes from the difference between LU factorization and *Cholesky Factorization* where Cholesky Factorization is defined for a symmetric positive definite matrix in which the symmetry can be exploited and the output is an $\tilde{L}$ where $\tilde{L}\tilde{L}^T = A$ which will also be symmetric positive definite. But, the LU code does not exploit this symmetry resulting in two different matrices, L and U where L is normalized in LU factorization and U are the elements resulting from Gaussian elimination and will not be normalized. We can also not guarantee that the elements of U will always be positive in the output.

We see from the results that this holds true with L being normalized compared to the diagonal elements, showing diagonal dominance and U is the result of the Gaussian Elimination. In this example, elements of U are all positive but this is not guaranteed in the output. We can see that L and U^T are not equal to each other which we see that it does not retain the symmetry. The product only retains symmetry if the chosen $L = U^T$ but it does not seem to be the case here.

V.II Empirical Results

Before moving into how the tester works and was developed, we first examine the Empirical Tests that were posed earlier in Description of Mathematics section. Since complete pivoting was not examined in the code, there will be no results for complete pivoting but will be briefly discussed on how it would be.

1.

V.III Generation of Problems/Tester Description

Now that we have seen that the LU factorization routine is correct, we can now move on to the tester for testing matrices of size n. The first step in defining the tester, we had to set the inputs for the function. The following are the inputs for the function which are all specified by the user upon running/calling the function:

- *dim_min*: This is the minimum dimension that is to be ran, which is started at this dimension.
- *dim_max*: The maximum dimension to indicate the upper range of tests ran.
- *num_tests*: This is the number of tests that are ran per dimension
- a: Lower Bound for the matrix/vector values
- c: Upper Bound for the matrix/vector values
- *routine*: Specifies which routine is to be run (1 for no pivoting or 2 for partial pivoting.
- *ratio*: A ratio factor for conditioning the Upper triangular matrix U on elements above the diagonal based off of the diagonal elements.

The tester runs from a minimum dimension to a maximum dimension with a given number of tests for each dimension. The values for all inputs within the tester are set by a and c. The routine to be specified is also passed through to tell whether no pivoting or partial pivoting is to be run. Along with a and c as the bounds, the ratio factor is passed through to control U's off-diagonal elements. This will be explained further when the inputs for each routine are explained. All of these inputs are determined by the user when running the function. The motivation setting it this was so a user would have full control of the tests that he/she would want to run or if they wanted to be able to run with their own parameters. This tester outputs four different plots showing different metrics to show how the factorization and solver completed. This was accomplished by using custom functions from the previous assignment, the current assignment and small custom functions for specific purposes. Walking step by step through the tester for the idea and generation behind it is:

1. First run a statement that tests to make sure the routine was either 1 or 2 on input and if not exits and tells the user to pick a specific routine for no pivoting and partial pivoting.
2. Create an *if* statement that checks which routine was chosen, where if routine 1 was chosen, no pivoting, the following would run:
   (a) Initialize a set of lists that will hold the values for each dimension tested
   (b) Run a *for* loop that runs over n in *num_tests* to test however many times per dimension and initialize a set of lists to hold the values for each run n
   (c) Create an inner *for* loop that runs each n and does all of the calculations for the factorization and error holding.
      i. Now create the variables that hold the generation of the matrices L, U, and x, where x will be used to compute b in the solver to be able to test the solver routine. Both L and U are conditioned and non-singular matrices. U is conditioned by taking the diagonal elements and restricting them to not hold values between -1 and 1 and then setting a variable *max_value* that takes the absolute value of the diagonal element and multiplies it by the given ratio set by the user on input. This is then put into the *random.uniform* function and the bounds are the negative of *max_value* and positive *max_value.* L is conditioned by setting the magnitude of the elements below the diagonal less than 1, i.e, $|\lambda_{ij}| < 1$ for $i > j$. Within this parameter section, LU is created by combining L and U into a single $2D$-array where 1's and 0's are not stored and then uses LU to compute the A from programming assignment 1, subroutine 4.
      ii. After all of the initialization, we now compute the LU factorization for the given routine, here is no pivoting. We also create the absolute value of the output, A_factor to use later on in computing the growth factor.
      iii. Since no pivoting can cause problems if the matrix has 0 on the diagonal, we check to make sure that if this happens, the program lets the user know that this happened and then exits that test and continues to the next one.

iv. We now compute the factorization error. This is useful as it compares how close our $L \cdot U$ is to the original A. So, we first recompute the multiplication of L and U from A_factor and then take the difference from the original A to the recomputed A, which is called C in the tester. Then the norm is taken of the difference matrix and the original A and then divided to yield the relative factorization error. This is then appended to the list in the *for* loop.

v. The solver is now used to compute $Ax = b$ where we first compute b from the randomly generated x and original A.

vi. First solves $Ly = b$ with forward substitution with the function, *solve_Lb* where we use the factored A, A_factor and the computed b.

vii. Use the output, y, from the *solve_Lb* function to compute $Ux = y$ for x where this x is named *x_comp* for readability. This uses the function *solve_Ux*.

viii. Now computes the error for the solver by taking the difference of the randomly generated x and *x_comp* and takes the 2-norm of that difference vector and divide that by the 2-norm of x to get the relative error which is appended to the list in the *for* loop.

ix. After having these done, we can now compute the residual and relative error of residual where we have now a new variable, *b_comp* which computes the Matrix-Vector product of the original A and *x_comp*. Now take the difference vector of b and *b_comp* where b was computed above and *b_comp* was computed in this step. Again, take the 2-norm of this difference and 2-norm of the b and divide them to get the relative residual.

x. The final metric computed is the growth factor where we take the absolute value of A_factor and run this through the in-place multiplication from programming assignment 1. Now take the Frobenius Norm of this multiplied A_factor and divide it by the Frobenius Norm of the original A. This is then appended to the list.

(d) Finally appends the original list with the mean of each of the dimensions.

3. If routine 2, partial pivoting, was chosen, the tester generally remains the same and will only highlight the differences between the two,

   (a) As L and U are both non-singular and well-conditioned, there had to be random permutation applied to A before running through this routine to ensure that pivoting would occur. If this was not done, A would not pivot because of the conditioning and making the matrix diagonally dominant.

   (b) After the inputs were initialized, the LU factorization routine now not only outputs the matrix but also the permutation vector P that stores the indices of the permutation matrix.

   (c) Now permute A with the permutation vector and recompute A, called C, from the factorization routine output. This is for the factorization accuracy and now subtracts C and *A_perm* which is the permuted A that just occurred. The rest is calculated the same as above for the relative error.

   (d) For the inputs of the solver, we now multiply the original randomly permuted A and randomly generated x to get b. We then pass this b through the row permutation that uses the P and vector b because we now have the equation $PAx = Pb$.

   (e) The solver now takes in Pb instead of b but calculates the same way as above for both lower and upper solves.

   (f) Relative error for x is calculated the same as above in the steps.

   (g) To calculate the relative residual error, we compute with *A_perm* and the computed x, *x_comp* instead of A. This is the difference from above, along with instead of b, we use Pb to calculate the difference vector. The norm is then taken for the difference and Pb.

   (h) The growth factor is computed the same as above with using the output of the factorization and taking the absolute value of that output.

4. After the routine chosen is ran through, the plots are generated to plot the means for each dimension tested for the number of tests for the respective dimension. There are four subplots with the title of the overall graphic showing what the ratio factor tested was to show differences between choices in ratio

factors. The plots will show the relative factorization error, the relative solver error, relative residual, and growth factor. The motivation of doing this was to show the errors under certain conditions and how the impact of conditioned matrices impacts the results.

V.IV Results

For computing the results, we will first discuss the dimensions tested along with the bounds and ratio factor. For each routine, the minimum dimension tested was 10 going up to a max dimension of 200 computing 50 tests per dimension. The bounds used were -50 to 50 encompassing a range a values that were not only positive but also negative. There were different ratio factors tested for each routine under these conditions to be able to examine what happens as a matrix becomes less and less diagonally dominant and potentially closer to singular. We first analyze the results for the no pivoting routine.

V.IV.1 No Pivoting Results

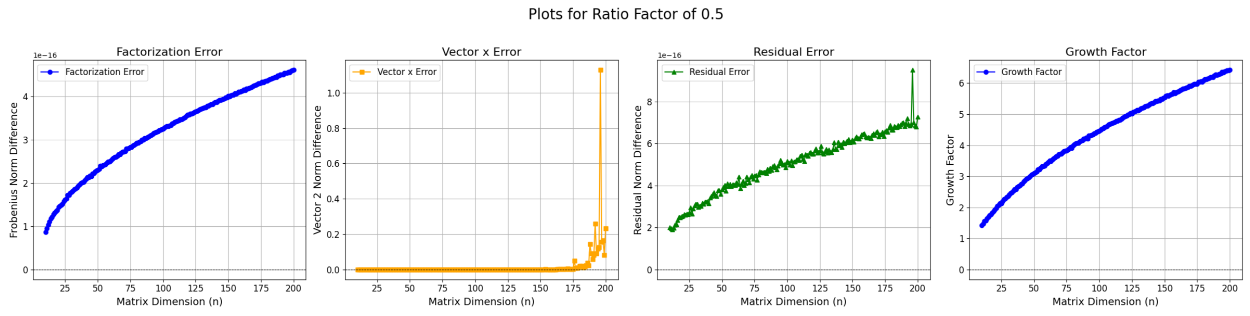


Figure 1: No Pivoting Mean Metrics for a Ratio Factor of 0.5

Examining the results in these four graphs of the four different metrics used, we can see some significant results. The Factorization Error and Residual Error seem to both be under 10^{-16} which is most likely due to round off error on the machine as the machine only stores up to 16 bits of memory. As the matrices get larger and larger, we see, that in the Factorization Error and Growth Factor, they tend to increase logarithmic which indicates that as the matrix size is growing, the floating point precision and rounding error tend to be more pronounced. The Residual error tends to be more linear than the others with a spike at the end which could be due to a matrix that was ill-conditioned or happened to have small enough values that did not reach the threshold set at 10^{-12}. The big error comes in the solver error, the second graph which seems to have spikes as the matrix size goes between 180 and 200. This could also be due to some ill-conditioned matrices sneaking through the threshold of having values very close to it which causes numerical instability and high error. But, we see that error leading up to that seems to be very small with the latter spiking.

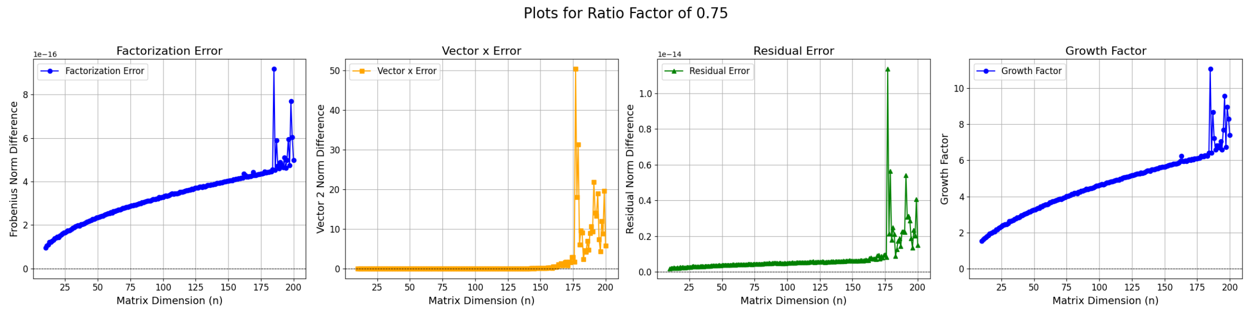


Figure 2: No Pivoting Metrics for a Ratio Factor of 0.75

Now we increase the ratio factor to 0.75 which leads to not as diagonally dominant upper triangular matrices, or closer to singular. We can see the differences here in each of the graphs as now factorization error has spiked at the end unlike above. We also see that residual error has increased significantly to above by now having max values of 10^{-14} which may not seem alot but is about 100 times as bad as above. We see growth factor spike at the end which attributes to how the factorization error, solver, and residual error all look and behave. The solver error also increases earlier, maybe around dimensions of 150 and higher with much higher spikes. The max error being about 50 compared to just over 1 above. We can see just from these two sets of plots that as the matrix becomes less and less conditioned, the higher errors and more numerical instability happens. What happens if there is a ratio of 1, i.e. the elements are not scaled down or up?

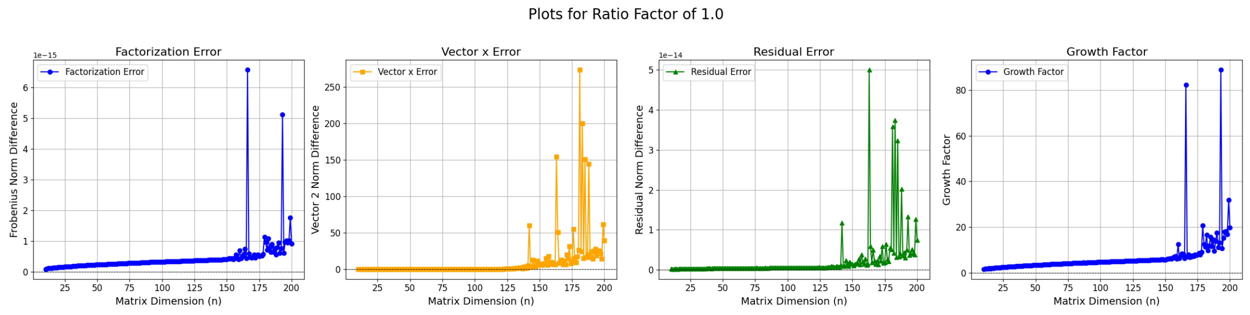


Figure 3: No Pivoting Metrics for a Ratio Factor of 1.0

As the ratio factor becomes one we see much bigger spikes in not only factorization error but all other metrics. Growth factor is now spiking to above 80 where before it was between 6-10 for high dimension matrices. There wasn't huge change in residual error but the solver error also jumps to a max of close to 300 and the error starts accumulating even earlier than before, at around a dimension size of 100-120.

After examining these for no pivoting, partial pivoting should act better under the same ratio factors given that we can manipulate rows to have maximal elements in the diagonals.

V.IV.2 Partial Pivoting Results

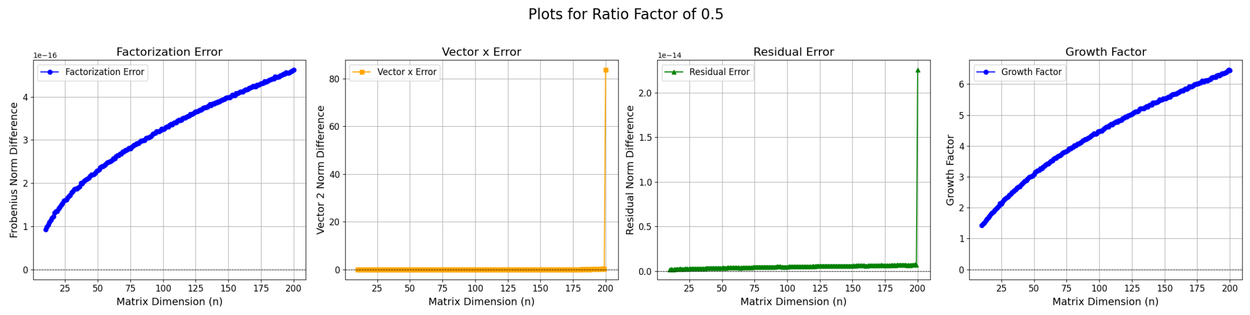


Figure 4: Partial Pivoting Metrics for a Ratio Factor of 0.5

We figured that the partial pivoting results for the same ratio factor should be better than no pivoting due to the ability to permute rows to have maximal elements in the diagonals. This is seen above as now the solver error and residual error only have a spike at the very end. Residual error for a ratio factor of 0.5 actually did worse than the error for no pivoting which could be due to ill-conditioned matrices even with pivoting and the generation of random values differing. The growth factor stayed about the same and we can see that the factorization error follows the growth factor, as expected because of how the growth factor is computed. As matrix size increases, the round off error can become more pronounced and the magnitudes can become larger since there are more elements in the matrix to be computed.

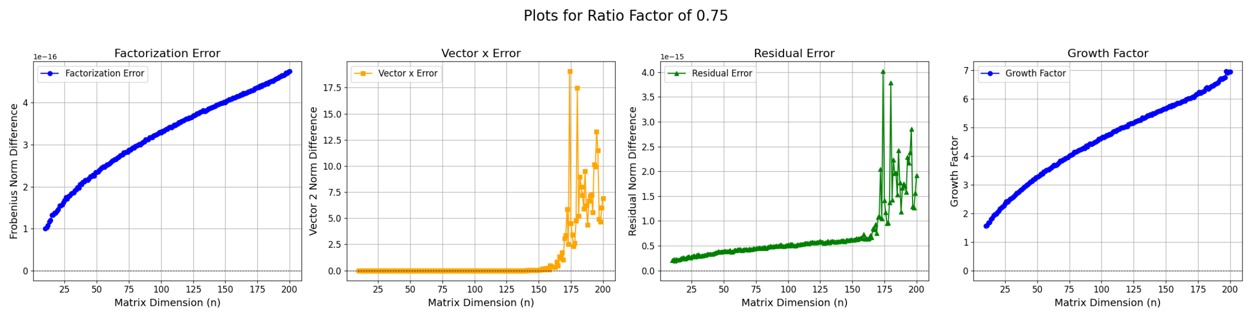


Figure 5: Partial Pivoting Metrics for a Ratio Factor of 0.75

First comparing the ratio factor of 0.75 to 0.5 with partial pivoting, we see that there is more error in smaller dimensions than the ratio factor 0.5 but the max is less than the single spike above. We see a max solver error of just over 17.5 with the error beginning to accumulate at dimensions of 150 and above. We can see the growth factor is starting to become less logarithmic but does not increase much compared to above with a max growth factor of just under 7 while before it was just above 6. Residual error improves from before with lower error but can be seen maybe spiking earlier. We can also see the factorization error starting to become more linear which follows from how the growth factor is graphed and shown.

Comparing the partial pivoting with a ratio factor of 0.75 to no pivoting with the same ratio, we see that partial pivoting performs much better with lower error across all plots. The same dimensions seem to spike at the same time but these increases are much less than the no pivoting case. Both in factorization error and growth factor we see that there are no spikes compared to no pivoting, which is a much better situation as we want them to not have high accumulation of error.

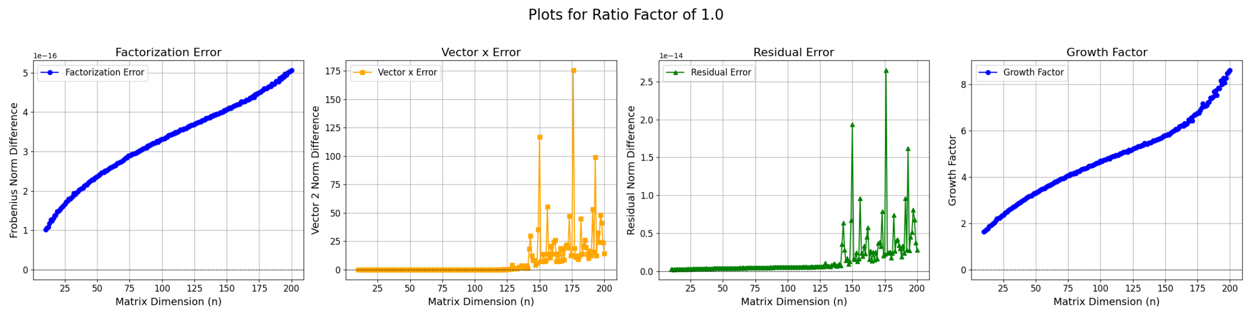


Figure 6: Partial Pivoting Metrics for a Ratio Factor of 1.0

Lastly, we compare ratio factor of 1.0 to the above graphs with partial pivoting first and then consider the comparison to no pivoting. As we expected the error will be larger than the two above as there is no scaling, less conditioning, going on in the matrices. The solver error seems to spike around 175 and residual error also increases with more variance in the spikes. We see that the error starts to increase again earlier at around dimensions of 125 and higher. Factorization error is starting to become more linear, maybe exponential if higher dimensions would be tested which can visually be seen in the growth factor graph. With growth factor almost looking cubic but now growing close to exponential at around dimension 125 and higher which can be seen in the errors as well with the increases starting to become more pronounced then.

Comparing this to the no pivoting case for ratio factor of 1.0, we see our prediction holds that this is much better than no pivoting as the spikes for factorization error and growth factor have been eliminated and the increases in solver and residual error tend to be less than the no pivoting errors. We also see the dramatic drop in growth factor from no pivoting to partial pivoting which makes sense as we are taking the maximal elements each time in the *for* loop.

VI Conclusion

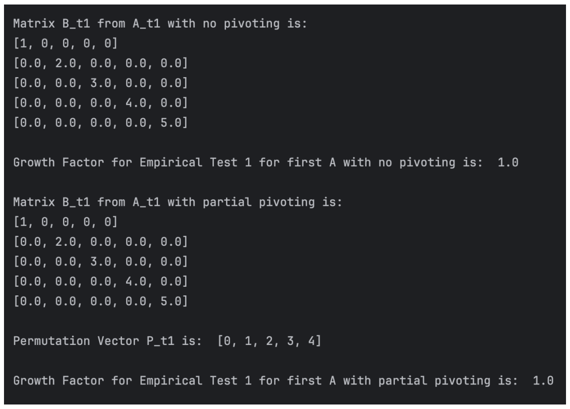
Within the report, we have seen how LU factorization works and is derived and the use of Upper and Lower Triangular solvers all for solving the equation $Ax = b$. LU factorization was tested under special cases to see how L and U can behave under certain structures of matrices all with no pivoting, partial pivoting, and complete pivoting. These special cases were not only examined theoretically but also verified numerically through running the matrices through the function and comparing the prediction to the results. In all cases, it performed as anticipated in theory, leading to testing matrices with dimensions of size $n \times n$.

These tests allowed for user-inputs of specified parameters of their choosing with a parameter that included how well- or ill-conditioned U would be. Notably, the results and conclusions showed that no pivoting with three different ratio factors, how well- or ill-conditioned U would be, performed worse than partial pivoting under the same or similar situations. We also saw within the results of partial pivoting that there seemed to be unique structural characteristics that formed in the empirical tasks and the errors. We saw that errors were able to retain their function form while no pivoting caused this form to break in higher dimensional matrices. These results highlight the importance of conditioning and how choosing routines impacts the errors in LU factorization and the solvers.

VII Program Listing

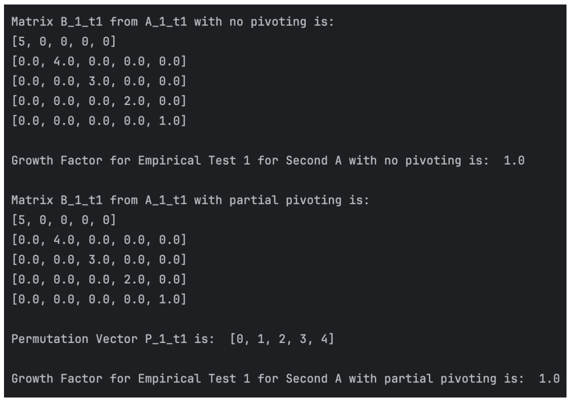
All files can be compiled in any IDE that can read python. There are three separate functions for the LU factorization, Lv solver, and Uv solver. Any other functions called from *my_package* will be included and can be found by the function name in the file. There is also a test driver that runs the LU factorization and solvers that the user must input the Minimum Dimension, Maximum Dimensions, Number of Tests per Dimensions, Lower Bound for Floating Points, Upper Bound for Floating Points, and a Ratio of the off-diagonal elements of U that is used for conditioning purposes as described in earlier sections. Each function/file contains a short description of the function and file that describes the use, the parameters, and the output.

VIII Extra Figures



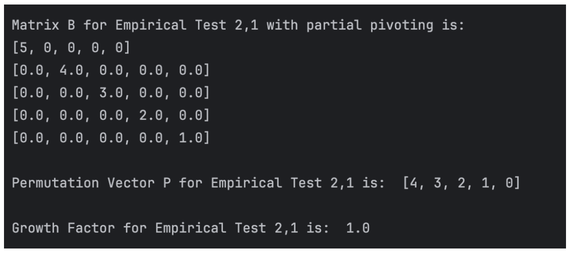
```
Matrix B_t1 from A_t1 with no pivoting is:
[1, 0, 0, 0, 0]
[0.0, 2.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 4.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 5.0]

Growth Factor for Empirical Test 1 for first A with no pivoting is:  1.0

Matrix B_t1 from A_t1 with partial pivoting is:
[1, 0, 0, 0, 0]
[0.0, 2.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 4.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 5.0]

Permutation Vector P_t1 is:  [0, 1, 2, 3, 4]

Growth Factor for Empirical Test 1 for first A with partial pivoting is:  1.0
```

(a) Empirical Test 1 for Given A



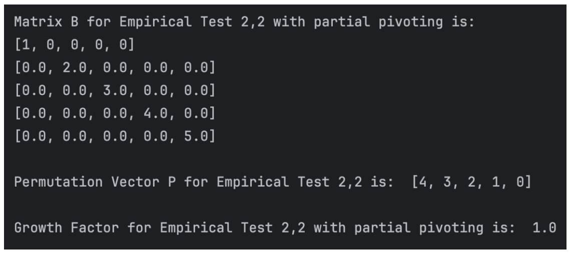
```
Matrix B_1_t1 from A_1_t1 with no pivoting is:
[5, 0, 0, 0, 0]
[0.0, 4.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 2.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 1.0]

Growth Factor for Empirical Test 1 for Second A with no pivoting is:  1.0

Matrix B_1_t1 from A_1_t1 with partial pivoting is:
[5, 0, 0, 0, 0]
[0.0, 4.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 2.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 1.0]

Permutation Vector P_1_t1 is:  [0, 1, 2, 3, 4]

Growth Factor for Empirical Test 1 for Second A with partial pivoting is:  1.0
```

(b) Empirical Test 1 for Given A_1



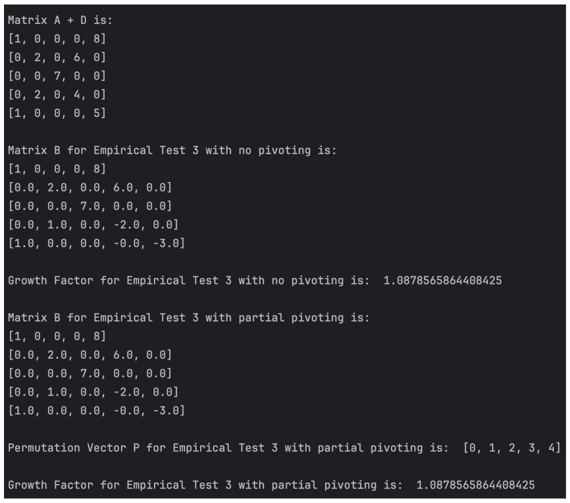
```
Matrix B for Empirical Test 2,1 with partial pivoting is:
[5, 0, 0, 0, 0]
[0.0, 4.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 2.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 1.0]

Permutation Vector P for Empirical Test 2,1 is:  [4, 3, 2, 1, 0]

Growth Factor for Empirical Test 2,1 is:  1.0
```

(c) Empirical Test 2 for Given A



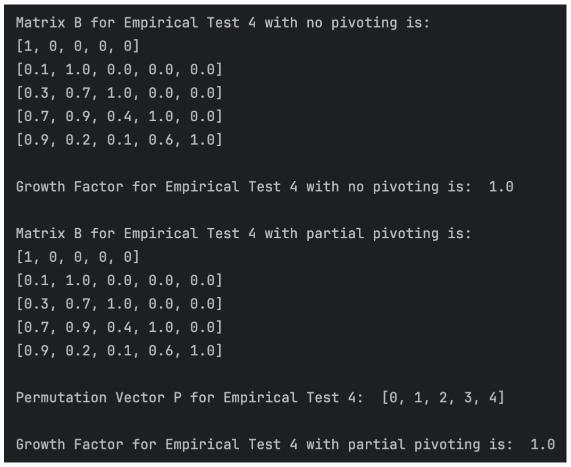
```
Matrix B for Empirical Test 2,2 with partial pivoting is:
[1, 0, 0, 0, 0]
[0.0, 2.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 3.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 4.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 5.0]

Permutation Vector P for Empirical Test 2,2 is:  [4, 3, 2, 1, 0]

Growth Factor for Empirical Test 2,2 with partial pivoting is:  1.0
```

(d) Empirical Test 2 for Given A_1



```
Matrix A + D is:
[1, 0, 0, 0, 8]
[0, 2, 0, 6, 0]
[0, 0, 7, 0, 0]
[0, 2, 0, 4, 0]
[1, 0, 0, 0, 5]

Matrix B for Empirical Test 3 with no pivoting is:
[1, 0, 0, 0, 8]
[0.0, 2.0, 0.0, 6.0, 0.0]
[0.0, 0.0, 7.0, 0.0, 0.0]
[0.0, 1.0, 0.0, -2.0, 0.0]
[1.0, 0.0, 0.0, -0.0, -3.0]

Growth Factor for Empirical Test 3 with no pivoting is:  1.0878565864408425

Matrix B for Empirical Test 3 with partial pivoting is:
[1, 0, 0, 0, 8]
[0.0, 2.0, 0.0, 6.0, 0.0]
[0.0, 0.0, 7.0, 0.0, 0.0]
[0.0, 1.0, 0.0, -2.0, 0.0]
[1.0, 0.0, 0.0, -0.0, -3.0]

Permutation Vector P for Empirical Test 3 with partial pivoting is:  [0, 1, 2, 3, 4]

Growth Factor for Empirical Test 3 with partial pivoting is:  1.0878565864408425
```

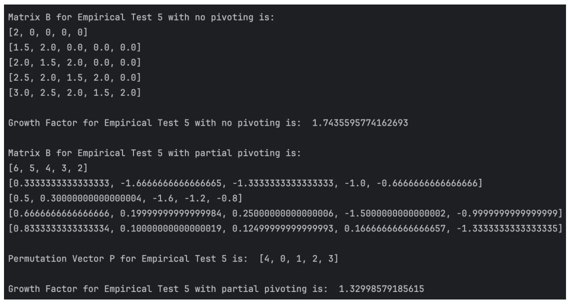
(e) Empirical Test 3 for Given A



```
Matrix B for Empirical Test 4 with no pivoting is:
[1, 0, 0, 0, 0]
[0.1, 1.0, 0.0, 0.0, 0.0]
[0.3, 0.7, 1.0, 0.0, 0.0]
[0.7, 0.9, 0.4, 1.0, 0.0]
[0.9, 0.2, 0.1, 0.6, 1.0]

Growth Factor for Empirical Test 4 with no pivoting is:  1.0

Matrix B for Empirical Test 4 with partial pivoting is:
[1, 0, 0, 0, 0]
[0.1, 1.0, 0.0, 0.0, 0.0]
[0.3, 0.7, 1.0, 0.0, 0.0]
[0.7, 0.9, 0.4, 1.0, 0.0]
[0.9, 0.2, 0.1, 0.6, 1.0]

Permutation Vector P for Empirical Test 4:  [0, 1, 2, 3, 4]

Growth Factor for Empirical Test 4 with partial pivoting is:  1.0
```

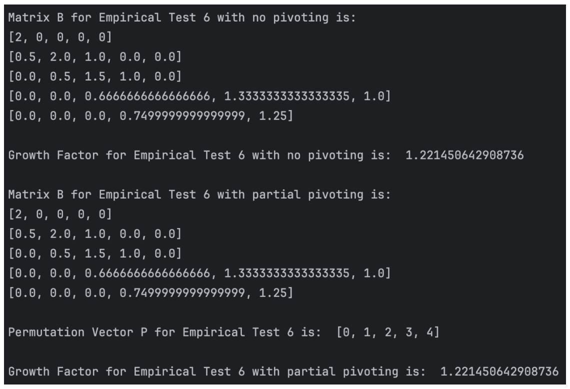
(f) Empirical Test 4 for Given A

Figure 7: Empirical Test Results - Part 1



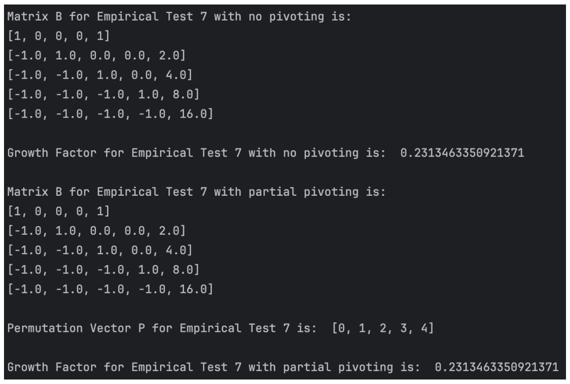
```
Matrix B for Empirical Test 5 with no pivoting is:
[2, 0, 0, 0, 0]
[1.5, 2.0, 0.0, 0.0, 0.0]
[2.0, 1.5, 2.0, 0.0, 0.0]
[2.5, 2.0, 1.5, 2.0, 0.0]
[3.0, 2.5, 2.0, 1.5, 2.0]

Growth Factor for Empirical Test 5 with no pivoting is:  1.7435595774162693

Matrix B for Empirical Test 5 with partial pivoting is:
[6, 5, 4, 3, 2]
[0.3333333333333333, -1.6666666666666665, -1.3333333333333333, -1.0, -0.6666666666666666]
[0.5, 0.30000000000000004, -1.6, -1.2, -0.8]
[0.6666666666666666, 0.19999999999999984, 0.25000000000000006, -1.5000000000000002, -0.9999999999999999]
[0.8333333333333334, 0.10000000000000019, 0.12499999999999993, 0.16666666666666657, -1.3333333333333335]

Permutation Vector P for Empirical Test 5 is:  [4, 0, 1, 2, 3]

Growth Factor for Empirical Test 5 with partial pivoting is:  1.32998579185615
```

(a) Empirical Test 5 for Given A



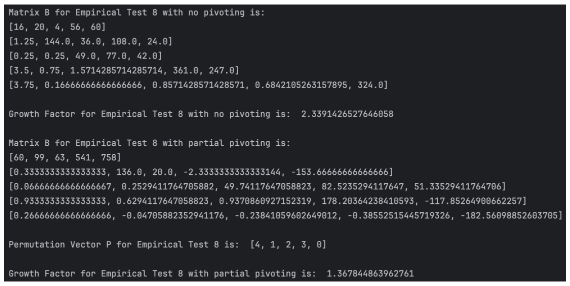
```
Matrix B for Empirical Test 6 with no pivoting is:
[2, 0, 0, 0, 0]
[0.5, 2.0, 1.0, 0.0, 0.0]
[0.0, 0.5, 1.5, 1.0, 0.0]
[0.0, 0.0, 0.6666666666666666, 1.3333333333333335, 1.0]
[0.0, 0.0, 0.0, 0.7499999999999999, 1.25]

Growth Factor for Empirical Test 6 with no pivoting is:  1.221450642908736

Matrix B for Empirical Test 6 with partial pivoting is:
[2, 0, 0, 0, 0]
[0.5, 2.0, 1.0, 0.0, 0.0]
[0.0, 0.5, 1.5, 1.0, 0.0]
[0.0, 0.0, 0.6666666666666666, 1.3333333333333335, 1.0]
[0.0, 0.0, 0.0, 0.7499999999999999, 1.25]

Permutation Vector P for Empirical Test 6 is:  [0, 1, 2, 3, 4]

Growth Factor for Empirical Test 6 with partial pivoting is:  1.221450642908736
```

(b) Empirical Test 6 for Given A



```
Matrix B for Empirical Test 7 with no pivoting is:
[1, 0, 0, 0, 1]
[-1.0, 1.0, 0.0, 0.0, 2.0]
[-1.0, -1.0, 1.0, 0.0, 4.0]
[-1.0, -1.0, -1.0, 1.0, 8.0]
[-1.0, -1.0, -1.0, -1.0, 16.0]

Growth Factor for Empirical Test 7 with no pivoting is:  0.2313463350921371

Matrix B for Empirical Test 7 with partial pivoting is:
[1, 0, 0, 0, 1]
[-1.0, 1.0, 0.0, 0.0, 2.0]
[-1.0, -1.0, 1.0, 0.0, 4.0]
[-1.0, -1.0, -1.0, 1.0, 8.0]
[-1.0, -1.0, -1.0, -1.0, 16.0]

Permutation Vector P for Empirical Test 7 is:  [0, 1, 2, 3, 4]

Growth Factor for Empirical Test 7 with partial pivoting is:  0.2313463350921371
```

(c) Empirical Test 7 for Given A



```
Matrix B for Empirical Test 8 with no pivoting is:
[16, 20, 4, 56, 60]
[1.25, 144.0, 36.0, 108.0, 24.0]
[0.25, 0.25, 49.0, 77.0, 42.0]
[3.5, 0.75, 1.5714285714285714, 361.0, 247.0]
[3.75, 0.16666666666666666, 0.8571428571428571, 0.6842105263157895, 324.0]

Growth Factor for Empirical Test 8 with no pivoting is:  2.3391426527646058

Matrix B for Empirical Test 8 with partial pivoting is:
[60, 99, 63, 541, 758]
[0.3333333333333333, 136.0, 20.0, -2.3333333333333144, -153.66666666666666]
[0.06666666666666667, 0.2529411764705882, 49.74117647058823, 82.5235294117647, 51.33529411764706]
[0.9333333333333333, 0.6294117647058823, 0.9370860927152319, 178.20364238410593, -117.85264900662257]
[0.26666666666666666, -0.04705882352941176, -0.23841059602649012, -0.38552515445719326, -182.56098852603705]

Permutation Vector P for Empirical Test 8 is:  [4, 1, 2, 3, 0]

Growth Factor for Empirical Test 8 with partial pivoting is:  1.367844863962761
```

(d) Empirical Test 8 for Given A

Figure 8: Empirical Test Results - Part 2

IX Citations

- In the description of mathematics, there was a section that was taken from "*Set 4: Linear Systems Part 2*" from Dr. Gallivan that was used in class. The specific section used was on Gauss Transformations Derivation on slides 22-25.
- Consulted with Charles Ohanian about algorithm development and logic. This was done through pseudocode being computed together.