HW2

Kevin Smith

October 2023

1 Executive Summary

In this report, we consider the LU-Decomposition of a matrix A in $\mathbb{R}^{nxn}$. We will examine the different methods for finding this LU, including the use of different pivoting methods.

2 Statement of the Problem

The LU-Decomposition is often used in numerical mathematics as a way to approximate x when $Ax = b$. This is useful whenever A^{-1} is difficult to find. L is defined as a non-singular unit-lower triangular matrix and U is an upper triangular matrix. The process of transforming A into these matrices creates several errors, and can lead to approximated solutions x. In general these errors are introduced by,

1. **Pivot Selection**: The choice of pivots is a critical aspect of GEM. If the pivot elements are poorly selected, such as very small values (close to zero) or very large values, it can lead to inaccuracies. Small pivot values can introduce numerical instability, and large pivot values may result in loss of precision. This can be exacerbated in computer-based computations due to finite precision arithmetic, resulting in round-off errors.

2. **Ill-Conditioned Matrices**: GEM can magnify the problems associated with ill-conditioned matrices. An ill-conditioned matrix has a high

condition number, indicating that small changes in coefficients or constants can lead to significant changes in the solution. GEM might not effectively mitigate these issues, leading to approximated solutions that may not accurately represent the actual problem due to the inherent numerical instability.

3. **Round-off Errors**: Computer-based numerical computations rely on finite precision arithmetic. During the elimination process, round-off errors can accumulate. These errors result from the limitations in representing real numbers as floating-point values with a finite number of significant digits. When numerous arithmetic operations are performed, round-off errors can accumulate and affect the accuracy of the solution. This is especially problematic when dealing with matrices that have large condition numbers.

3 Description of the Algorithms and Implementation

U and L are both found using the Gauss elimination method (GEM). In general, GEM takes a matrix

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

And reduces the lower off-diagonal elements into zero with several elementary row operations:

1. Start with the first row (Row 1) and eliminate the coefficients below the first element (a_{11}) to create zeros in the off-diagonal positions. This is done by performing row operations on the subsequent rows as follows:

   (a) Scale Row 1 by a factor (k) and subtract it from the corresponding elements in Rows 2 through n to create zeros:

$$\begin{aligned} R_2 &\leftarrow R_2 - kR_1 \\ R_3 &\leftarrow R_3 - kR_1 \\ &\vdots \\ R_n &\leftarrow R_n - kR_1 \end{aligned}$$

2. Move to the second row (Row 2) and create zeros below the second element (a_{22}) by performing similar row operations on the subsequent rows:

   (a) Scale Row 2 by a factor (k) and subtract it from the corresponding elements in Rows 3 through n:

$$\begin{aligned} R_3 &\leftarrow R_3 - kR_2 \\ &\vdots \\ R_n &\leftarrow R_n - kR_2 \end{aligned}$$

3. Continue this process, moving from Row 3 to Row $n-1$, creating zeros in the off-diagonal elements of each row.

4. Once you reach the last row (Row $n-1$), the matrix will be in upper triangular form with zeros in the off-diagonal elements. You can then proceed to solve the system of equations by back-substitution.

This upper diagonal matrix is the U of the LU-decomposition. L is found by performing the same operations on I^{nxn}. Using both L & U, you solve the following equations:

1. $Ly = b$

2. $Uy = x$

To perform these operations, I used Python as well as the math and random packages within. I generated matrices of size n=200 and n=20 with elements ranging from [-100,100]. I used the following alogrithms to generate the matrix A

1. generate_spd_matrix(n)

   This created a matrix that was symmetric, by ensuring $A_{i,j} = A_{j,i}$. And ensured positive definiteness by adding the size of the matrix to the diagonal entries, creating diagonal dominance.

2. generate_diagonally_dominant_matrix(n)

   This created a diagonally dominant matrix, finding the row sum and adding it to the diagonal entries.

I also used several functions to solve the matrix depending on the pivoting technique being used.

1. lu_decomposition_partial_pivot(A)
2. lu_decomposition_complete_pivot(A)
3. lu_decomposition_no_pivot(A)

All of these functions returned the permutation matrices P,Q as well as the LU decomposition which was performed with the in-place algorithm strategy.

To test my algorithm, I used a test matrix A and test vector b:

$$A = \begin{bmatrix} 2 & 1 & 0 \\ -4 & 0 & 4 \\ 2 & 5 & -10 \end{bmatrix}$$

$$b = \begin{bmatrix} 3 \\ 0 \\ 17 \end{bmatrix}$$

I found the LU decomposition with partial pivoting as:

Matrix LU: Permutation Vector P: Solution Vector x:

$$A = \begin{bmatrix} -4 & 0 & 4 \\ -0.5 & 5.0 & -8.0 \\ -0.5 & 0.2 & 3.6 \end{bmatrix} \qquad P = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \qquad x = \begin{bmatrix} -0.1111 \\ 3.2222 \\ -0.1111 \end{bmatrix}$$

And with complete pivoting as:

Matrix A:

$$A = \begin{bmatrix} 2 & 5 & -10 \\ -2.0 & 10.0 & -16.0 \\ 1.0 & -0.4 & 3.59 \end{bmatrix}$$

Permutation Vector P:

$$P = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

Permutation Vector Q:

$$Q = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

Solution Vector x:

$$x = \begin{bmatrix} -0.11111111111111083 \\ 3.2222222222222228 \\ -0.11111111111111073 \end{bmatrix}$$

Which verifies that my algorithm is computing correctly.

4 Description of the Experimental Design and Results

I began by generating the A, as well as x. This allowed me to directly find b in order to best examine the accuracy of our results. I then began implementing my algorithm with different types of pivoting with the two different n's mentioned above. I performed error analysis using these conditions on SPD matrices as well as diagonally dominant matrices. I performed LU decompositions with each different method ten times and found the average of each error.

I found that in general, the factorization error ($\frac{||PAQ-LU||}{||A||}$) was 0 for all cases except for when a pivoting technique was used. While the pivoting did not lead to the exact factorization, it did lead to a more accurate prediction of x. This could come from the fact that pivoting helps prevent the diagonal entries of our matrix from becoming less than one. This prevents larger errors in rounding. The disparity between the errors of SPD and diagonally dominant can easily be explained by examining the condition numbers of both matrices prior to factorization. The SPD matrices I have generated have a much higher condition number than the diagonally dominant matrices. This suggests that the matrix has a higher sensitivity to changes in our data. In fact, I observed that my generated SPD matrices have a much more inconsistent condition number compared to the diagonally dominant matrix. This may be generating by the larger diagonal entries being generated in generate_diagonal_dominant_matrix.

It can also be observed that the complete pivoting of a matrix created much higher relative errors in x ($\frac{||x-\tilde{x}||}{||x||}$) compared to the other pivoting techniques. I believe this may be caused by round off errors, as there are many more operations being performed on the matrices, leading to the entries in our factorization being inaccurate; creating an inaccurate x. All of the errors become larger as the size of the matrix increases, this likely occurs for the same reason as the errors coming from complete pivoting. More operations leads to more errors because of roundoff.

The residual errors (($\frac{||b-A\tilde{x}||}{||b||}$)) followed the same general trends as the relative error in x. with lower errors in smaller diagonally-dominant matrices.

I have compiled a table containing the errors I calculated in section 6.

5 Conclusions

Our results indicate that the LU-factorization is an effective way to compute x in $Ax = b$. While there are obvious inaccuracies generated from using numerical methods, the results are within a good margin of error. These methods can of course be made better by fine tuning the algorithm, including increasing the number of iterations and changing the matrix A in order for it to be more easily transformed using GEM. The results also indicate that the diagonally-dominant matrix is by far more accurate in terms of numerics. This is simply due to the fact that SPD matrices are much more sensitive to changes in the data. Pivoting techniques, while decreasing the factorization accuracy, increase the accuracy of the result x due to it ensuring that the matrices diagonal entries are no smaller than 1 which could lead to much larger numbers when division is applied. LU-factorization is extremely useful in the case where the inverse of A is not easily found, like when n is large.

6 Tables and Figures

n=200, Diagonally Dominant Matrix						
Pivot type	Factorization Error using 1 norm	Factorization Error using Forbenius norm	Relative Error in x using one norm	Relative Error in x using two norm	Residual Error using one norm	Residual Error using two norm
none	0	0	0.9999718434	1.000129124	0.05040638598	0.05718624956
partial	0	0	0.9990848695	0.9992198423	0.04902267378	0.0568578946144245
complete	1.002092216	1.414083122	784.9406701	2736.440316	561.6474883	695.2160883

n=200, SPD Matrix						
Pivot type	Factorization Error using 1 norm	Factorization Error using Forbenius norm	Relative Error in x using one norm	Relative Error in x using two norm	Residual Error using one norm	Residual Error using two norm
none	0	0	7475.86578	8012.510022	117.4651873	57.20988322
partial	1.504686569	1.401987044	84.23030016	92.36156426	4.843635484	5.07980626
complete	1.43901541	1.414662742	6965.047432	7444.051585	230.6371116	230.4742387

n=20, Diagonally Dominant Matrix						
Pivot type	Factorization Error using 1 norm	Factorization Error using Forbenius norm	Relative Error in x using one norm	Relative Error in x using two norm	Residual Error using one norm	Residual Error using two norm
none	0	0	1.012871891	1.019952924	0.1511537736	0.1754739127
partial	0	0	1.033636718	1.040320075	0.1765277698	0.1998665883
complete	1.010065138	1.412042316	80.37665578	159.8089866	298.7922459	326.6313241

n=20, SPD Matrix						
Pivot type	Factorization Error using 1 norm	Factorization Error using Forbenius norm	Relative Error in x using one norm	Relative Error in x using two norm	Residual Error using one norm	Residual Error using two norm
none	0	0	109.6928502	109.9773577	13.88771516	14.40261874
partial	1.436147137	1.376916629	6.252133736	6.73427537	2.381155848	2.319181796
complete	1.260150448	1.412234866	247.6277868	251.8568198	6.882763599	6.214888664