On reducing the time complexity of LU Factorisation and other applications.

Yajush Dev Rai Bhatia

WP/01/2023-24/IPM 2022-27 Batch
December 2023

A New Algorithm for LU Factorization through Fast Matrix Multiplication

Yajush Dev Rai Bhatia

December 23, 2023

Abstract

This paper presents a novel algorithm for computing the LU factorization of a square matrix in $O(n^{log_2 7})$ time complexity, significantly improving upon existing algorithms with $O(n^3)$ time complexity. The proposed algorithm has the potential to revolutionize various computational tasks relying on LU factorization.

1.1 Introduction

With the LUP decomposition, the matrix determinant can be computed in polynomial time, thanks to the English computer scientist Alan Turing. The matrix determinant has the same asymptotic complexity as matrix multiplication between two matrices of the same order.

Theorem 1.1

Let $M(n)$ be the time required to multiply two $n \times n$ matrices over some ring, and A is an $n \times n$ matrix. Then, we can compute the determinant of A in $O(M(n))$ steps.

Various methods exist for multiplying n-order matrices, each with different complexities. The standard matrix multiplication (schoolbook method) [3] has a complexity of $O(n^3)$, while the Strassen algorithm [1] and the Coppersmith-Winograd algorithm [2] have complexities of $O(n^{2.807})$ and $O(n^{2.376})$, respectively. Optimized CW-like algorithms achieve an $O(n^{2.373})$ complexity [4, 5, 6], which is the same as that of matrix determinant computation using rapid matrix multiplication.

Different approaches to compute the matrix determinant are available, each with its complexity. The Laplace enlargement algorithm has an $O(n!)$ complexity, making it less efficient for larger matrices. The Division-free $O(n^4)$ algorithm [7] also suffers from high complexity. However, the LUP decomposition method achieves an $O(n^3)$ complexity, making it a more efficient option.

Additionally, the Bareiss algorithm and Fast-matrix multiplication offer $O(n^3)$ [8] and $O(n^{2.373})$ [4] complexities, respectively, for determining the matrix determinant.

The fast matrix multiplication is a galactical algorithm, making it impractical on existing datasets. Strassen proposed a method for determinant calculation [1, 9], but it isn't extensively used. I propose a simple recursive algorithm to calculate the LU decomposition, which not only brings the determinant but also other applications such as linear equation solving. Since it's not an LUP factorization, not all matrices can be factorized, but the ones which don't require a P matrix, such as variance-covariance matrix, can be solved by partially pivoting all, as performed in Algorithm 2.2.2.

2.1 LU Factorization

The LU factorization is a fundamental technique in linear algebra that decomposes a square matrix into the product of a lower triangular matrix and an upper triangular matrix. It has wide-ranging applications in various fields, including solving systems of linear equations and calculating matrix determinants.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} = \begin{bmatrix} l_{1,1} & 0 & 0 & 0 \\ l_{2,1} & l_{2,2} & 0 & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} & 0 \\ l_{4,1} & l_{4,2} & l_{4,3} & l_{4,4} \end{bmatrix} \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} \\ 0 & 0 & u_{3,3} & u_{3,4} \\ 0 & 0 & 0 & u_{4,4} \end{bmatrix}$$

Now, LU Factorization is not always unique, but the algorithm will always result in a unique solution. This is because the diagonals of the lower matrix are fixed as 1 ($l_{ii} = 1$), making the factorization unique.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{2,1} & 1 & 0 & 0 \\ l_{3,1} & l_{3,2} & 1 & 0 \\ l_{4,1} & l_{4,2} & l_{4,3} & 1 \end{bmatrix} \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} \\ 0 & 0 & u_{3,3} & u_{3,4} \\ 0 & 0 & 0 & u_{4,4} \end{bmatrix}$$

2.1.1 Lemma

If all n leading principal minors of the $n \times n$ matrix A are non-singular, then A has an LU-decomposition.

Definition 2.1.2 Principal Minors

Let A be a square matrix of order n, and let $I = \{i_1, i_2, \ldots, i_k\}$ be a non-empty subset of $\{1, 2, \ldots, n\}$. Then, the principal minor of A corresponding to I, denoted by $A[I, I]$, is the square submatrix of order k obtained by selecting rows $i_1, i_2, \ldots, i_k$ from A and the corresponding columns $i_1, i_2, \ldots, i_k$.

Example

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Principal Minor $A[1, 1]$

This is the smallest possible principal minor, containing only the first element of A:

$$A = \begin{bmatrix} 1 \end{bmatrix}$$

Principal Minor $A[2, 2]$**:**

This minor includes the first two rows and columns of A:

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

Principal Minor $A[3, 3]$**:**

This is the entire matrix itself, as all rows and columns are selected:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Theorem 2.1.3

The LU Factorization resulting in a lower matrix with diagonal entries as 1 will yield a unique solution.

Proof

Let us assume that the LU factorization is not unique. So,

$$A = LU = LIU = LDD^{-1}U = (LD)(D^{-1}U)$$

Where $D \in R^{n \times n}$, $D \neq I_n$.

Since it's an LU decomposition, LD needs to be a lower matrix and $D^{-1}U$ needs to be an upper matrix. Therefore, D has to be a diagonal matrix. Now, the diagonal elements of D can't be equal to 1 since D is not an identity matrix.

$$(LD)_{ii} = \sum_{j=1} L_{ij} D_{ij} = L_{ii} D_{ii} + \sum_{j=1, j \neq i} L_{ij} \cdot 0 = 1 \cdot D_{ii} + 0 \neq 1$$

So, such D doesn't exist, and hence it will always have a unique solution in this format.

2.2 Applicability of LU Factorization

Since every matrix is not LU factorizable without the use of partial pivoting, there are special matrices that are always LU factorizable. This includes symmetric matrices and diagonally dominant matrices.

Definition 2.2.1: Diagonally Dominant Matrices

Let A be a square matrix of order n. A is said to be strictly diagonally dominant if and only if, for every row i $(1 \leq i \leq n)$, the following inequality holds:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad (1 \leq j \leq n)$$

Now, certain matrices can be converted to diagonally dominant form by following these steps.

Algorithm 2.2.2: Permutation Matrix for LU Process

Input: A matrix which can be made strictly diagonally dominant.
Output: A permutation matrix which makes it diagonally dominant.

1. Identify the largest element. For each row, scan the elements below the diagonal and find the element with the largest absolute value. If this element is already on the diagonal, no pivoting is needed.
2. Swap rows. If the largest element is not on the diagonal, swap the row containing it with the row containing the current diagonal element.
3. Update permutation matrix. Keep track of the row swaps in a separate permutation matrix. For each swap, record a swap operation in the corresponding rows of the permutation matrix.
4. Repeat steps 1-3. Continue iterating through the remaining rows, performing pivoting, and updating the permutation matrix until the entire matrix is upper triangular.

Note: In case of ties, a column can be skipped and revisited in the later steps.

Complexity Analysis of Algorithm 2.2.2

1. Step 1: Identifying the largest element in each row takes approximately $O(n-1)$ comparisons for a row of size n. This is repeated for n rows, making it $O(n^2)$.
2. Step 2: Swapping rows involves exchanging all elements between the two rows, which takes $O(n)$ operations. This happens only if a pivot is needed, so its cost is amortized over the total number of pivots.

3. Step 3: Updating the permutation matrix is a constant time operation ($O(1)$) for each pivot.

Total Complexity: $O(n^2 + n^2 + n) = O(2n^2 + n) \approx O(n^2)$

Drawbacks of this Algorithm

When encountering ties (elements with the same largest absolute value) in Step 1, the chosen tie-breaking rule can influence the success of the algorithm. Some rules might lead to unnecessary pivots or not guarantee a suitable pivot for further elimination, potentially stalling the process.

Theorem 2.2.3: Theorem on Preserving Diagonal Dominance

Gaussian elimination without pivoting preserves the diagonal dominance of a matrix.

Corollary 2.2.3.1 (1st Corollary on Diagonally Dominant Matrix)

Every diagonally dominant matrix is non-singular and has an LU-factorization.

Corollary 2.2.3.2 (2nd Corollary on Diagonally Dominant Matrix)

If the scaled row pivoting version of Gaussian elimination recomputes the scale array after each major step is applied to a diagonally dominant matrix, then the pivots will be the natural ones: $1, 2, \ldots, n$. Hence, the work of choosing the pivots can be omitted in this case.

Theorem 2.2.4: Every Symmetric Matrix is LU Factorizable

Suppose $A \in R^{n \times n}$ is non-singular. Then A has an LDL^T-factorization if and only if $A = A^T$ and A_k is non-singular for $k = 1, \ldots, n-1$ (Cholesky Factorization).

Proof

We can write A as

$$A = LDM^T$$

where $M^T = D^{-1}U$. M^T is unit upper triangular and

$$A^T = A$$

$$A^T = (LDM^T)^T = MDL^T = LU = A$$

Now $M(DL^T)$ and LU are two LU-factorizations of A, and by the uniqueness of the LU-factorization, we must have $M = L$. Thus, $A = LDM^T = LDL^T$ is an LDL^T-factorization of A (Theorem 2.1.2).

Conversely, if $A = LDL^T$ is an LDL^T-factorization of A, then A is symmetric since LDL^T is symmetric, and A has an LU-factorization with $U = DL^T$.

Algorithm 2.2.5: Convert an LU Factorizable Matrix to its LU Form

Input: A matrix of $2k$ order which is LU factorizable.
Output: LU Factorization of the given matrix.

Let a matrix be written in the form of four partition matrices, which will always be possible if A^{-1} exists. The existence of A^{-1} will always be possible in the case of an LU factorizable matrix, which will be discussed later.

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{CA}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{D} - \mathbf{CA}^{-1}\mathbf{B} \end{bmatrix}$$

Now calculate this matrix:

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{CA}^{-1} & \mathbf{I} \end{bmatrix}$$

Now pre-multiply this matrix with the original matrix:

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{CA}^{-1} & \mathbf{I} \end{bmatrix} \times \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

Since the determinant of the pre-multiplied matrix is 1:

$$\mathrm{Det}(AB) = \mathrm{Det}(A)\mathrm{Det}(B)$$

So the determinant of our original matrix will remain conserved. Now, the algorithm must be applied recursively. After multiplying, the matrix will be an upper triangular matrix whose determinant is easier to calculate.

Similar Example

1. Break the matrix in this format.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ l_{3,1} & l_{3,2} & 1 & 0 \\ l_{4,1} & l_{4,2} & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ 0 & 0 & u_{3,3} & u_{3,4} \\ 0 & 0 & u_{4,3} & u_{4,4} \end{bmatrix}$$

2. Finding Inverse of U matrix

$$U^{-1} = \begin{bmatrix} \frac{1}{a_{1,1}} & \frac{1}{a_{1,2}} & u'_{1,3} & u'_{1,4} \\ \frac{1}{a_{2,1}} & \frac{1}{a_{2,2}} & u'_{2,3} & u'_{2,4} \\ 0 & 0 & u'_{3,3} & u'_{3,4} \\ 0 & 0 & u'_{4,3} & u'_{4,4} \end{bmatrix}$$

3. Calculating AU^{-1}

$$AU^{-1} = L = \begin{bmatrix} l_{3,1} & l_{3,2} \\ l_{4,1} & l_{4,2} \end{bmatrix} = \begin{bmatrix} \frac{a_{3,1}}{a_{1,1}} + \frac{a_{3,2}}{a_{2,1}} & \frac{a_{3,1}}{a_{1,2}} + \frac{a_{3,2}}{a_{2,2}} \\ \frac{a_{4,1}}{a_{1,1}} + \frac{a_{4,2}}{a_{2,1}} & \frac{a_{4,1}}{a_{1,2}} + \frac{a_{4,2}}{a_{2,2}} \end{bmatrix}$$

4. Calculating L^{-1}

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{a_{3,1}}{a_{1,1}} + \frac{a_{3,2}}{a_{2,1}} & \frac{a_{3,1}}{a_{1,2}} + \frac{a_{3,2}}{a_{2,2}} & 1 & 0 \\ \frac{a_{4,1}}{a_{1,1}} + \frac{a_{4,2}}{a_{2,1}} & \frac{a_{4,1}}{a_{1,2}} + \frac{a_{4,2}}{a_{2,2}} & 0 & 1 \end{bmatrix}$$

5. Calculating $U = L^{-1}A$

$$U = L^{-1}A = \begin{bmatrix} u_{3,3} & u_{3,4} \\ u_{4,3} & u_{4,4} \end{bmatrix} =$$

$$\begin{bmatrix} \left(\frac{-a_{3,1}}{a_{1,1}} - \frac{a_{3,2}}{a_{2,1}}\right) * a_{1,3} + \left(\frac{-a_{3,1}}{a_{1,2}} - \frac{a_{3,2}}{a_{2,2}}\right) * a_{2,3} + a_{3,3} & \left(\frac{-a_{3,1}}{a_{1,1}} - \frac{a_{3,2}}{a_{2,1}}\right) * a_{1,4} + \left(\frac{-a_{3,1}}{a_{1,2}} - \frac{a_{3,2}}{a_{2,2}}\right) * a_{2,4} + a_{3,4} \\ \left(\frac{-a_{4,1}}{a_{1,1}} - \frac{a_{4,2}}{a_{2,1}}\right) * a_{1,3} + \left(\frac{-a_{4,1}}{a_{1,2}} - \frac{a_{4,2}}{a_{2,2}}\right) * a_{2,3} + a_{4,3} & \left(\frac{-a_{4,1}}{a_{1,1}} - \frac{a_{4,2}}{a_{2,1}}\right) * a_{1,4} + \left(\frac{-a_{4,1}}{a_{1,2}} - \frac{a_{4,2}}{a_{2,2}}\right) * a_{2,4} + a_{4,4} \end{bmatrix}$$

Now, we can see that in this, the first left and bottom right matrices are still not in upper matrix form. This algorithm can be used recursively to change them to this format. We will have to do this about $\log_2 n$ times, as each time we are dividing the matrix by 2. So, the final time complexity will be $O(n^{log7})$

Note Multiplying with a lower matrix recursively won't change the solution space of the matrix because we are technically doing elementary row operations, which do not change the solution space as well as the kernel of the matrix. Also, the calculation of U^{-1} need not be done, and instead, the lower matrix should be updated directly as per the partition matrix formula. It is shown in the algorithm for illustration.

Step Count for Algorithm 2.2.5

(a) Creation of Partition matrices as shown in Algorithm 2.2 $\mathrm{O(n^2)} as it requires n^2$ items to be updated and stored.

(b) Calculation of A^{-1}: Since our original matrix is of $n \times n$ dimensions, the matrix will be of $n/2 \times n/2$. The number of steps for its inverse according to Strassen's Algorithm ($\mathrm{O(n^{log_2 7})}$)

(c) Pre-multiplying by C: Since both A and C are of $n/2$, the number of steps via Strassen's Algorithm ($O(n^{log_2 7})$))

(d) Updating of Lower matrix by adding CA^{-1} terms. $O((n/2)^2)$ as $n/2$ items are being updated.

(e) Updating the $DCA^{-1}B$ terms in the upper matrix. The number of steps via Strassen's Algorithm $O(n^{log_2 7})$, where o is a constant. $O((n/2)^2)$ as $n/2$ items are being updated.

(f) Now calling this recursively for smaller multiplications (Step 1-Step 5 where the partition matrix is updated).

(g) Multiplying all the lower matrices achieved from the decomposition. This will be $O(n^{log_2 7})$ since the matrices are getting reduced by a factor of 2 in each recursion.

(h) After getting the final L matrix, L^{-1} can be calculated in $O(n^2)$ and then pre-multiplied to A. $O(n^{log_2 7})$
.

Total Steps $= O(n^{\log_2 7})$.

Theorem 2.2.6

An A^{-1} will always exist in the types of partition defined if it's an LU Factorizable matrix.

Proof: Since the LU Factorizable matrix has all principal minors as non-zero (Lemma 2.1.1), the determinant of A will always be non-zero. A matrix in each recursion will be made by removing the last $n/2$ rows and last $n/2$ columns of the same index. Since we are removing the columns and rows with the same index numbers, this will result in a principal $n/2$-th minor which is non-zero, implying the determinant of that partition matrix is non-zero and hence invertible.

Example

Let us take this matrix.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix}$$

Lemma 2.1.1 implies that.

$$\text{Det} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \neq 0$$

$$\text{Det} \begin{bmatrix} a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \neq 0 \text{ and Det} \begin{bmatrix} a_{1,1} & a_{1,3} & a_{1,4} \\ a_{3,1} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,3} & a_{4,4} \end{bmatrix} \neq 0$$

$$\text{Det} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,4} \\ a_{4,1} & a_{4,2} & a_{4,4} \end{bmatrix} \neq 0 \text{ and Det} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \neq 0$$

Similarly, there will be a case:

$$\text{Det} \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \neq 0 \text{ (Condition 1)}$$

and

$$\text{Det} \begin{bmatrix} a_{1,1} \end{bmatrix} \neq 0 \text{ (Condition 2)}$$

Condition 1 and Condition 2 are required for the existence of A^{-1} in our case, and thus the Lemma already provides this condition.

Conclusion

So, it is possible to LU factorize a matrix if possible, in the same time complexity of matrix multiplication. Some drawbacks are that for expanding it to LPU factorization, partial pivoting might be required which makes it generalizable to all matrices and algorithm 2.2.5 can naively be used for symmetric matrices and matrices which can be converted to diagonally dominant or LU factorizable matrices.

Appendix

Example 1

Let us take the matrix S:

$$\begin{bmatrix} 12 & 06 \\ 18 & 05 \end{bmatrix}$$

Recursion 1

$$\begin{bmatrix} 12 & 06 \\ 18 & 05 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1.5 & 1 \end{bmatrix} \begin{bmatrix} 12 & 6 \\ 0 & -4 \end{bmatrix}$$

Step 1

Calculation of A^{-1}

$$\begin{bmatrix} 1/12 \end{bmatrix}$$

Step 2

Calculation of CA^{-1}

$$\begin{bmatrix} 18 \end{bmatrix} \begin{bmatrix} 1/12 \end{bmatrix} = \begin{bmatrix} 1.5 \end{bmatrix}$$

Step 3

Updating Lower Matrix

$$\begin{bmatrix} 1 & 0 \\ 1.5 & 1 \end{bmatrix}$$

Step 4

Updating Upper Matrix

$$\begin{bmatrix} 12 & 6 \\ 0 & -4 \end{bmatrix}$$

Example 2

Let us take the matrix:

$$\begin{bmatrix} 8 & 9 & 10 & 11 \\ 16 & 30 & 33 & 36 \\ 24 & 75 & 97 & 105 \\ 40 & 117 & 233 & 268 \end{bmatrix}$$

Recursion 1

$$\begin{bmatrix} 8 & 9 & 10 & 11 \\ 16 & 30 & 33 & 36 \\ 24 & 75 & 97 & 105 \\ 40 & 117 & 233 & 268 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -5 & 4 & 1 & 0 \\ -7 & 6 & 0 & 1 \end{bmatrix} \begin{bmatrix} 8 & 9 & 10 & 11 \\ 16 & 30 & 33 & 36 \\ 0 & 0 & 15 & 16 \\ 0 & 0 & 105 & 129 \end{bmatrix}$$

Step 1

Calculation of A^{-1}

$$\begin{bmatrix} 5/16 & -3/32 \\ -1/6 & 1/12 \end{bmatrix}$$

Step 2

Calculation of CA^{-1}

$$\begin{bmatrix} 25 & 75 \\ 40 & 117 \end{bmatrix} \begin{bmatrix} 5/16 & -3/32 \\ -1/6 & 1/12 \end{bmatrix} = \begin{bmatrix} -5 & 4 \\ -7 & 6 \end{bmatrix}$$

Step 3

Updating Lower Matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -5 & 4 & 1 & 0 \\ -7 & 6 & 0 & 1 \end{bmatrix}$$

Step 4

Updating Upper Matrix (In this, it is not necessary to update all the rows (Highlighted ones which are outside are partition matrix in recursion 2) but only the matrix we have divided because it wouldn't matter in our recursive process and the correct matrix can be found by taking L^{-1} and pre-multiplying by A after all the recursions are over, which will take fewer steps)

$$\begin{bmatrix} 8 & 9 & 10 & 11 \\ 16 & 30 & 33 & 36 \\ 0 & 0 & 15 & 16 \\ 0 & 0 & 105 & 129 \end{bmatrix}$$

Recursion 2 for upper matrix

$$\begin{bmatrix} 8 & 9 & 10 & 11 \\ 16 & 30 & 33 & 36 \\ 0 & 0 & 82 & 89 \\ 0 & 0 & 128 & 139 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 8 & 9 & 10 & 11 \\ 0 & 12 & 13 & 14 \\ 0 & 0 & 15 & 16 \\ 0 & 0 & 105 & 129 \end{bmatrix}$$

Recursion 3 for lower matrix

$$\begin{bmatrix} 8 & 9 & 10 & 11 \\ 0 & 12 & 13 & 14 \\ 0 & 0 & 15 & 16 \\ 0 & 0 & 105 & 129 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 7 & 1 \end{bmatrix} \begin{bmatrix} 8 & 9 & 10 & 11 \\ 0 & 12 & 13 & 14 \\ 0 & 0 & 15 & 16 \\ 0 & 0 & 0 & 17 \end{bmatrix}$$

The Recursion will be reduced by a factor of 2 in every step so the sum of it will be like:

$$O(n)^w + 2O\left(\frac{n}{2}\right)^w + 4O\left(\frac{n}{4}\right)^w + \ldots \leq k(n)^w + 2k\left(\frac{n}{2}\right)^w + 4k\left(\frac{n}{4}\right)^w + \ldots$$

This will converge as it has a ratio of $\frac{1}{2^{w-1}}$ which is always less than one as the theoretical bound on $w > 2$. So, it will converge. This can also be seen as:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)^w$$

Multiplying all the Lower Matrices:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -5 & -4 & 1 & 0 \\ -7 & 6 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -5 & -4 & 1 & 0 \\ -7 & 6 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 7 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 5 & 6 & 7 & 1 \end{bmatrix}$$

The multiplication steps will take less time as it needs to multiply only matrices of lower dimensions which will be recursive and make an infinite GP that converges to $n^{log_2 7}$. It will be sort of this kind, and essentially only 1 multiplication is required, that is of $\frac{n}{2}$ order. The rest is equivalent to copying the matrix:

$$\begin{bmatrix} I & \mathbf{0} \\ A & I \end{bmatrix} \begin{bmatrix} B & \mathbf{0} \\ \mathbf{0} & I \end{bmatrix} = \begin{bmatrix} B & \mathbf{0} \\ AB & I \end{bmatrix}$$

The other multiplication of lower updated matrices won't matter as:

$$\begin{bmatrix} B & \mathbf{0} \\ AB & I \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ \mathbf{0} & C \end{bmatrix} = \begin{bmatrix} B & \mathbf{0} \\ AB & C \end{bmatrix}$$

So effectively, it is just joining the matrix. So effectively we need only 1 multiplication of $\frac{n}{2}$ order which reduces recursively by a factor of 2 and, hence forming an infinite GP.

Hence

$$\begin{bmatrix} 8 & 9 & 10 & 11 \\ 16 & 30 & 33 & 36 \\ 24 & 75 & 97 & 105 \\ 40 & 117 & 233 & 268 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 5 & 6 & 7 & 1 \end{bmatrix} \begin{bmatrix} 8 & 9 & 10 & 11 \\ 0 & 12 & 13 & 14 \\ 0 & 0 & 15 & 16 \\ 0 & 0 & 0 & 17 \end{bmatrix}$$

Example 3

Let us take a bigger 16×16 matrix to see how the GP terms come.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,16} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,16} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{16,1} & a_{16,2} & a_{16,3} & \cdots & a_{16,16} \end{bmatrix}$$

It will have a lower matrix in this format.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ l_{2,1} & 1 & 0 & 0 & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & 1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ l_{16,1} & l_{16,2} & l_{16,3} & \cdots & l_{16,14} & l_{16,15} & 1 \end{bmatrix}$$

Now Recursion 1 will take time $k\left(\frac{n}{2}\right)^w$ since n is 16 and we are multiplying matrices of order $\left(\frac{n}{2}\right)$. For 4×4 matrices time will be $2k\left(\frac{n}{4}\right)^w$. For 2×2 matrices time will be $4k\left(\frac{n}{8}\right)^w$. For 1×1 matrices time will be $8k\left(\frac{n}{16}\right)^w$. As we can see, it's an infinite GP with r as $\frac{1}{2w-1}$ so it converges.

Note - This algorithm is defined for 2^k order matrices, but any matrix can be converted to that order by adding 0 to the right of the present rows and for new rows an identity-type matrix can be made.

Example 4

Let us take matrix A

$$A = \begin{bmatrix} 8 & 9 & 10 \\ 16 & 30 & 33 \\ 24 & 75 & 97 \end{bmatrix}$$

This is equivalent to solving.

$$\begin{bmatrix} 8 & 9 & 10 & 0 \\ 16 & 30 & 33 & 0 \\ 24 & 75 & 97 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Recursion 1

$$\begin{bmatrix} 8 & 9 & 10 & 0 \\ 16 & 30 & 33 & 0 \\ 24 & 75 & 97 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -5 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 8 & 9 & 10 & 0 \\ 16 & 30 & 33 & 0 \\ 0 & 0 & 15 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Recursion 2

$$\begin{bmatrix} 8 & 9 & 10 & 0 \\ 16 & 30 & 33 & 0 \\ 0 & 0 & 15 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 8 & 9 & 10 & 0 \\ 0 & 12 & 13 & 0 \\ 0 & 0 & 15 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplication of Lower Matrices

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -5 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now the extra rows and columns can be removed.

$$A = \begin{bmatrix} 8 & 9 & 10 \\ 16 & 30 & 33 \\ 24 & 75 & 97 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix} \begin{bmatrix} 8 & 9 & 10 \\ 0 & 12 & 13 \\ 0 & 0 & 15 \end{bmatrix}$$

References

[1] Strassen, V. (1969). *Gaussian elimination is not optimal.* Numerische Mathematik, 13(4), 354–356. https://doi.org/10.1007/bf02165411

[2] Coppersmith, D., & Winograd, S. (1990). *Matrix multiplication via arithmetic progressions.* Journal of Symbolic Computation, 9(3), 251–280. https://doi.org/10.1016/s0747-7171(08)80013-2

[3] Stothers, A. (n.d.). *On the complexity of matrix multiplication.* School of Mathematics, University of Edinburgh. https://www.maths.ed.ac.uk/sites/default/files/atoms/files/stothers.pdf

[4] Williams, V. (n.d.). *Multiplying matrices in* $O(N^{2.373})$ *time.* Massachusetts Institute of Technology. https://people.csail.mit.edu/virgi/matrixmult-f.pdf

[5] Le Gall, F. (2014). *Powers of tensors and fast matrix multiplication.* Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation. https://doi.org/10.1145/2608628.2608664

[6] Davie, A. M., & Stothers, A. J. (2013). *Improved bound for complexity of matrix multiplication.* Proceedings of the Royal Society of Edinburgh: Section A Mathematics, 143(2), 351–369. https://doi.org/10.1017/s0308210511001648

[7] Rote, G. (2001). *Division-free algorithms for the determinant and the Pfaffian: Algebraic and combinatorial approaches.* Computational Discrete Mathematics, 119–135. https://doi.org/10.1007/3-540-45506-x_9

[8] Bareiss, E. H. (1968). *Sylvester's identity and multistep integer-preserving gaussian elimination.* Mathematics of Computation, 22(103), 565–578. https://doi.org/10.1090/s0025-5718-1968-0226829-0

[9] Baur, W., & Strassen, V. (1983). *The complexity of partial derivatives.* Theoretical Computer Science, 22(3), 317–330. https://doi.org/10.1016/0304-3975(83)90110-x