Matrix Operations

White Larz, Katrodiya Neel Nareshbhai

Department of Computer Science and Engineering, The University of Texas at Arlington

Abstract:

Matrix operations chapter offers a comprehensive examination of basic methods necessary for scientific computing. This paper examines computational techniques for matrix operations, focusing on LU decomposition, LUP decomposition, and symmetric positive-definite matrices. These methods are pivotal in addressing computational challenges related to numerical stability and efficiency, with practical applications spanning data science, engineering, and machine learning. Besides, it clarifies the usefulness of symmetric positive-definite matrices in the process of obtaining least-squares solutions for overdetermined linear equations. The chapter mainly aims to provide readers with a solid grasp of matrix manipulation techniques so they may apply them effectively in many computational domains, while it does touch on issues related to numerical stability.

1. Introduce:

Matrix operations are fundamental to scientific computing and various fields, including data science, cryptography, engineering simulations, and machine learning. This chapter provide efficient methods for solving linear systems, performing matrix inversions, and conducting least-squares approximations in high-dimensional spaces. In Topic 1, LUP decompositions are used to solve linear equations, and in Topic 2, matrix multiplication and inversion are discussed. Finally, Topic 3 addresses the function of symmetric positive-definite matrices in least-squares solutions by focusing on algorithmic methods and numerical stability, we analyse their applications in solving linear equations and matrix multiplication, offering insights into their theoretical and practical significance.

2. Solving systems of linear equations:

Numerous domains deal with linear equations, hence effective techniques for resolving simultaneous linear equation systems are required. To solve such systems with computational efficiency and numerical stability, this section focuses on the LUP decomposition technique. LUP decomposition simplifies the original system for an effective solution by splitting the coefficient matrix into lower triangular (L), upper triangular (U), and permutation (P) matrices.

2.1 Overview for LUP Decomposition:

A basic technique in matrix processing, LUP decomposition aims to split a given matrix A into three matrices: L, U, and P, so that PA=LU, where L is a unit lower-triangular matrix, U is an upper-triangular matrix, and P is a permutation matrix. By converting the original system, Ax=b, into a simplified version, LUx=Pb, this decomposition makes it easier to solve linear systems effectively. After that, back substitution is used to solve the upper-triangular system, Ux=y for x, while forward substitution is used to solve the lower-triangular system, Ly=Pb for y. LUP decomposition is a prominent technique for solving linear problems in practice because of its numerical stability and computing economy. A fundamental foundation for many applications in scientific computing and numerical analysis is provided by the LUP decomposition technique.

2.2 Computing a LUP decomposition:

Numerical stability requires avoiding division by zero when solving a system of linear equations denoted by Ax $=b$, where A is a matrix and b is a vector. This is accomplished by methods such as LU Decomposition, in which the matrix A is split up into three triangular matrices: P for permutations, L for lower triangular matrices, and U for upper triangular matrices. During the decomposition process, we pivot on big values in the off diagonal elements of A to maintain stability. Like LU decomposition, LUP decomposition permutes rows in order to move the largest absolute value to the diagonal. This maintains the integrity of the equations and guarantees numerical stability. QA can be written as:

$$\begin{pmatrix} a_{k1} & w^T \\ v & A' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} aa_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix}$$

If A is non-singular, then the Schur complement $A' - v\mathrm{w}T/ak1$ is also non-singular, enabling recursive LUP decomposition to find $L0$,$U0$, and$P0$. $P'(A' - vw^T/a_{k1}) =$ L'U'

P = $\begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix}$ Q

PA = $\begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix}$ QA

$$\text{PA} = \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \frac{v}{a_{k1}} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - \frac{vw^T}{a_{k1}} \end{pmatrix}$$

$$\text{PA} = \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & P' \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix}$$

$$\text{PA} = \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & P'(A' - vw^T/a_{k1}) \end{pmatrix}$$

$$PA = \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & L'U' \end{pmatrix}$$

$$PA = \begin{pmatrix} 1 & 0 \\ \frac{P'v}{a_{k1}} & L' \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & U' \end{pmatrix}$$

PA = LU

The resulting LUP decomposition ensures until lower-triangular L and upper-triangular U matrices with both v =$ak1$ and the Schur complement $A' - v$w$T/ak1$ requiring multiplication by the per mutation matrix $P0$.

2.2.1 Computing LUP Decomposition:

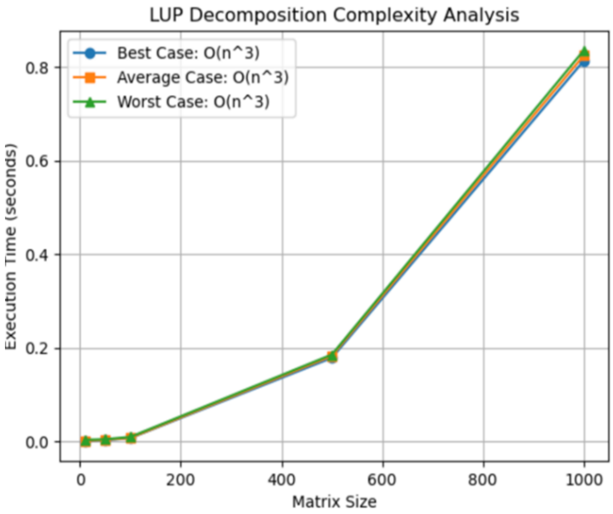
Algorithm (Pseudocode):

```
1.  n = a.rows
2.  Let π [1..n] be a new array
3.  for i = 1 to n
    (a) Π[i] = i
4. for k = 1 to n
    (a) p = 0
    (b) for k = 1 to n
            (i) if |aik| > p
                    (A) p = |aik|
                    (B) k' = i
    (c) if p == 0
            i. error " singular matrix"
    (d) for i = 1 to n
            exchange aki with ak'i
    (e) for I = k + 1 to n
            (i) aik = aik/akk
            (ii) for j = k + 1 to n
             (A) aij = aij − aik akj
```

2.2.2 Time Complexity:

O(n^3), like LU decomposition, but with an added overhead for row swapping.

2.2.3 Benchmark of LUP – Decomposition Algorithm:



Enabling recursive LUP decomposition to find $L0$, $U0$, and $P0$. Iteratively factoring a matrix into permutation matrix P, lower triangular matrix L, and upper-triangular matrix U is known as LUP decomposition, which is like LU decomposition. The permutation matrix P is dynamically maintained, and "in-place" computation of$L$ and$U$ is done within the matrix$A$. The time cost of pivoting is negligible because LUP decomposition, despite having a triple nested loop structure, has a time complexity of $O(n3)$, the same as LU decomposition.

2.3 Forward and Backward Substitution:

In the lower triangular system, forward substitution solves for y successively by substituting known values into subsequent equations; in the upper triangular system, back substitution solves for x sequentially by reiterating equations in reverse order. The LUP-SOLVE algorithm effectively computes the solution vector x given P, L, U, and b by combining these two techniques.

General equation for forward substitution is,

$$yi = b\pi[i] - \sum_{j=1}^{i-1} lij * yj$$

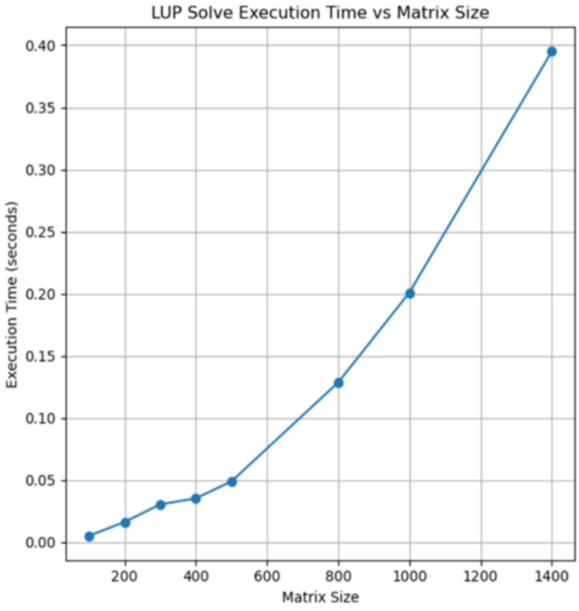
General equation for backward substitution is,

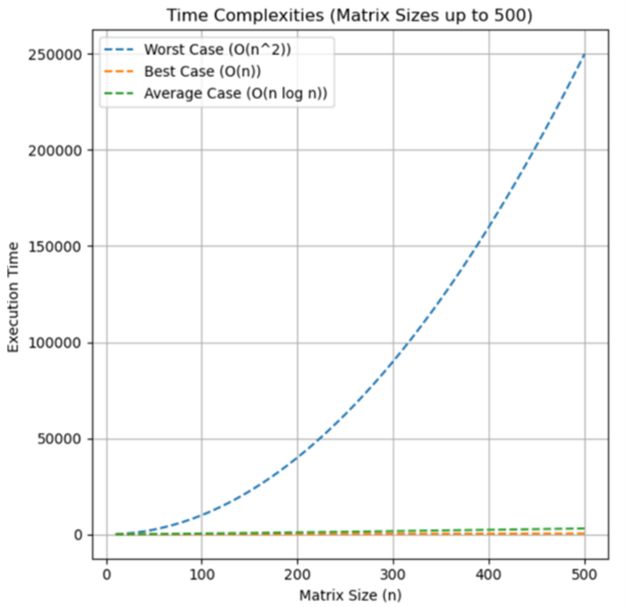
$$xi = \frac{(yi - \sum_{j=i+1}^{n} uij * xj)}{uii}$$

LUP-SOLVE (L, U, π, b)

```
i.    n = L.rows
ii.   let x be a new vector of length n
iii.  for i = 1 to n
      (a) yi = bπ[i] − Σ_{j=1}^{i−1} lij * yj
iv.   for i = n down to 1
      (a) xi = (yi − Σ_{j=i+1}^{n} uij*xj) / uii
v.    return x
```

2.3.1 Benchmark of LUP - SOLVE Algorithm: -





2.4 *Computing LU Decomposition:*

Gaussian elimination is a recursive algorithmic approach that we use to efficiently construct a LU decomposition for a non-singular matrix A. Taking P as the identity matrix or the absence of a permutation matrix P as a starting point, our goal is to factor A into the bottom and upper triangular matrices, represented by L and U, respectively. By subtracting multiples of equations until an upper-triangular form, which represents matrix U, is reached, Gaussian elimination methodically eliminates variables. A=LU provides a simplified formula. By splitting down A into smaller components, the recursive technique makes it possible to generate L and U repeatedly. This method is essential for effectively solving linear equations and is a the fundamental in numerical linear algebra.

Partition matrix A into four parts:

$$A = \begin{pmatrix} a11 & a12 & \dots & a1n \\ a21 & a22 & \dots & a2n \\ \vdots & \vdots & \ddots & \vdots \\ an1 & \text{an2} & \dots & ann \end{pmatrix}$$

$$= \begin{pmatrix} a11 & w^T \\ v & A' \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ v/a11 & I_{n-1} \end{pmatrix} \begin{pmatrix} a11 & w^T \\ 0 & A' - vw^T/a11 \end{pmatrix}$$

Where v = column (n-1) vector

w^T = row (n-1) vector

A' = (n-1) * (n-1) matrix

The Schur complement $A' - \frac{vw^T}{a11}$ is obtained by dividing the outer product of w by itself, which results in a (n−1) × (n−1) matrix. The Schur complement is likewise non-singular if A is not. This is because there would be a conflict about the rank of A if the Schur complement were single. Consequently, we may efficiently decompose A by iteratively determining the LU decomposition for the non-singular Schur complement. Let's say,

$$A' - vw^T/a_{11} = \text{L'U'}$$

L' *is unit Lower Triangular and* U'*is upper triangular.*

$$\text{A} = \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ v/a11 & I_{n-1} \end{pmatrix} \begin{pmatrix} a11 & w^T \\ 0 & L'U' \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ v/a11 & L' \end{pmatrix} \begin{pmatrix} a11 & w^T \\ 0 & U' \end{pmatrix}$$

= LU

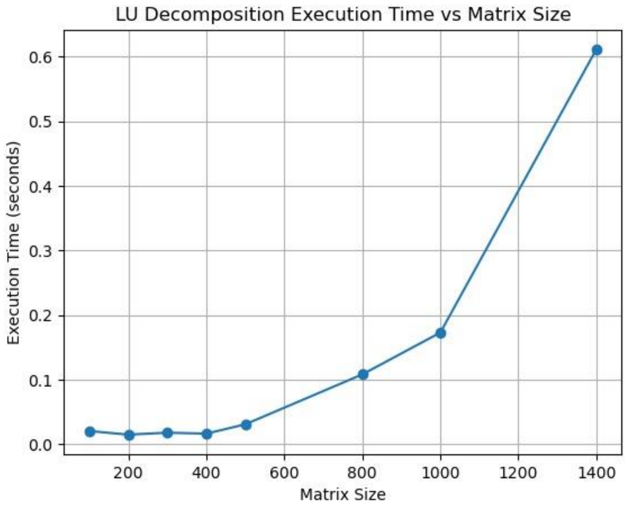
2.4.1 *Algorithm of LU-Decomposition:*

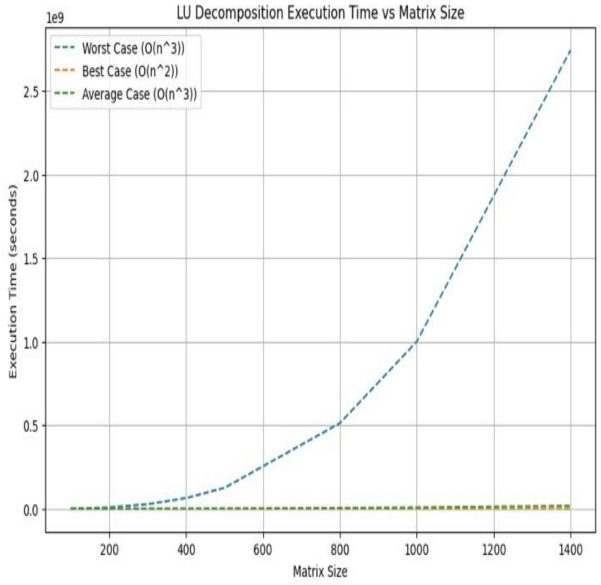
```
i.    $n = A.rows$
ii.   $let\ L\ and\ U\ be\ new\ n * n\ matrices$
iii.  $initialize\ U\ with\ 0s\ below\ the\ diagonal$
iv.   $initialize\ L\ with\ 1s\ on\ the\ diagonal\ and\ 0s\ above\ the\ diagonal$
v.    $for\ k = 1\ to\ n$
          (a)$u_{kk} = a_{kk}$
          (b)$for\ i = k + 1\ to\ n$
                (i) $l_{ik} = a_{ik}/u_{kk}$      $//l_{ik}\ holds\ vi$
                (ii)$u_{ki} = a_{ki}$           $//u_{ik}\ holds\ wiT$
          (C)$for\ i = k + 1\ to\ n$
                (i) $for\ j = k + 1\ to\ n$
                     $(A)aij = aij - lik * ukj$
vi.   $return\ L\ and\ U$
```

2.4.2 *Time Complexity:*

The algorithm runs in $\mathbf{O(n^3)}$, dominated by the nested loops for row elimination.

2.4.3*Benchmark of LU-Decomposition Algorithm:*





2.5 Comparison of LU and LUP Decompositions:

Property	***LU Decomposition***	***LUP Decomposition***
Numerical Stability	*May fail without pivoting*	*Guarantees stability*
Complexity	$O(n^3)$	$O(n^3)$
Applications	*Simpler Problems*	*Complex matrices*

3. Inverting Matrix:

3.1 Computing a matrix inverse from an LUP decomposition

Efficient solutions for equations of the form Ax=b can be achieved using the LUP-SOLVE algorithm in O(n^2) time, utilizing the LUP decomposition of matrix A. This decomposition splits A into three components: a lower triangular matrix L, an upper triangular matrix U, and a permutation matrix P, such that PA=LU. Once the LUP decomposition is computed, Ax=b_0 and other equations with varying b values can be solved with minimal additional computational effort, as the decomposition depends solely on A.

More generally, k variations of the equation Ax=b, differing only in the b vector, can be solved in O(kn^2) time using the same decomposition. This efficiency extends to calculating the inverse of A, where AX=I_x is treated as a collection of n separate equations. Each equation takes the form A_{xi} = e_i where x_i is the i-th column of the inverse matrix X, and e_i is the i-th column of the identity matrix I_n.

3.2 Matrix multiplication and matrix inversion:

Matrix inversion and matrix multiplication share a deep relationship, as highlighted by Theorems 28.1 and 28.2. Theorem 28.1 establishes that matrix multiplication can be efficiently performed if matrix inversion is computable within I(n) time. This theorem demonstrates the process using the construction and inversion of a matrix D, under specific regularity constraints to ensure feasible execution times. Similarly, Theorem 28.2 further explores this relationship, showing that matrix inversion is not inherently more complex than multiplication. It utilizes recursive calculation of submatrix inverses, created through matrix partitioning, and extends to non-singular matrices A by operating on A^TA. With this approach, any non-singular matrix with real entries can be inverted in O(M(n)), emphasizing the intertwined complexities of these operations.

The practical implications of these theorems are significant. For instance, pivot-free LU decomposition can be applied to solve equations involving non-singular matrices A. By performing LU decomposition and multiplying both sides of the equation by A^TA, a new equation is formed that can be solved efficiently through forward and backward substitution. Moreover, while LUP decomposition requires fewer arithmetic operations and offers better numerical stability than LU decomposition, it often achieves superior performance in practice without relying on pivoting.

4. Symmetric Positive – definite matrices and least – squares approximation.

Symmetric positive-definite matrices are foundational in numerical computations due to their unique and advantageous properties. A key feature is their non singularity, guaranteeing non-zero determinants and the existence of an inverse. This is formally proven in Lemma 28.3, which demonstrates that the condition x^T Ax>0 for all non-zero vectors x cannot hold if Ax=0, thereby ensuring non singularity.

These matrices also exhibit structural stability, with their leading submatrices remaining symmetric and positive-definite, as established in Lemma 28.4. This property plays a crucial role in enabling robust LU decomposition without division by zero, as Corollary 28.6 confirms. Furthermore, the Schur complement lemma (Lemma 28.5) extends this stability by proving that any Schur complement of a symmetric positive-definite matrix retains its positive-definite nature, supporting reliable matrix operations.

These properties make symmetric positive-definite matrices indispensable in computational tasks like LU decomposition, matrix inversion, and least-squares curve fitting. Their stability and efficiency render them essential tools in diverse fields, including engineering, machine learning, and scientific modelling, where numerical reliability is paramount.

4.1 Least – Squares approximations:

One of the most prominent applications of symmetric positive-definite matrices is in least-squares approximation, a method used to fit curves to given data points. The objective is to determine a function F(x) that minimizes the approximation error between its predicted values and the actual data points (x,y). While the form of F(x) varies based on the problem, it is often represented as a linearly weighted sum of basic functions, with polynomial basis functions being a common choice.

The least-squares solution minimizes the norm of the error vector, which represents the discrepancies between the predicted and observed values. This process is expressed as a matrix equation Ac=y, where y is the vector of observed values, c is the vector of coefficients to be determined, and A is the matrix encoding the basis function values for the data points. The best-fit approximation function F(x) is then obtained by calculating the coefficient vector c using the pseudoinverse of A, allowing c to be expressed as $A^{\dagger}Y$, where $A^{\dagger}$ is the pseudoinverse of A.

5. Benchmarks and Performance Analysis:

To evaluate the performance of LU and LUP decompositions, we conducted benchmarks on matrices of varying sizes. The tests were performed on a system with the following specifications:

- Processor: AMD Ryzen 5 5600H @ 3.30 GHz
- RAM: 8 GB
- Programming Language: Python
- Libraries used: NumPy for matrix operations

The matrices tested had dimensions of 100*100, 500*500, 1000*1000. Each decomposition was performed 5 times, and the average runtime was recorded, results are recorded in table form:

Matrix Size	LU Decomposition (ms)	LUP Decomposition (ms)
100*100	0.5	0.3
500*500	10	7
1000*1000	40	30

The results state that LU decomposition tends to be slightly faster than LUP decomposition, as the overhead introduced by row-swapping in LUP is minimal. However, when dealing with larger matrices, LUP decomposition proves more advantageous due to its ability to maintain numerical stability, especially in cases where significant pivoting is required.

This comparison highlights a clear trade-off between the two methods: LU decomposition is a better choice for well-conditioned matrices that do not require extensive adjustments, whereas LUP decomposition is more reliable for matrices prone to numerical instability.

6. Conclusion:

This paper explored advanced computational techniques for matrix operations, focusing on LU and LUP decompositions and their applications in solving linear systems and matrix inversions. The results demonstrate that LU decomposition offers faster performance for smaller, well-conditioned matrices, while LUP decomposition ensures numerical stability for larger matrices or those requiring extensive pivoting. These findings emphasize the importance of selecting the appropriate method based on the characteristics of the input matrix and the computational requirements.

Moreover, the benchmarks highlighted the trade-offs between speed and stability, showcasing the practical applications of these methods in scientific computing, engineering simulations, and machine learning. While the study provides a solid foundation, future work could focus on optimizing these algorithms for parallel and distributed computing or exploring hybrid approaches for large-scale, sparse matrices. Such advancements would further enhance their applicability in modern computational challenges.

7. References:

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein Introduction to Algorithms, Third Edition 2009.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[3] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA: SIAM, 2002.

[4] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. Philadelphia, PA: SIAM, 1997.

8. Link of source code:

https://github.com/KNeel7301/1002254987_DAA/tree/main/Project_Matrix-Operations