LU decomposition

From Wikipedia, the free encyclopedia

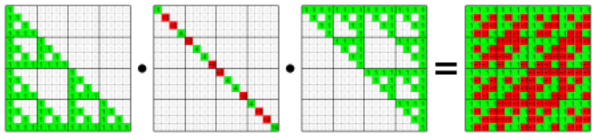
In numerical analysis and linear algebra, **lower–upper** (**LU**) **decomposition** or **factorization** factors a matrix as the product of a lower triangular matrix and an upper triangular matrix (see matrix decomposition). The product sometimes includes a permutation matrix as well. LU decomposition can be viewed as the matrix form of Gaussian elimination. Computers usually solve square systems of linear equations using LU decomposition, and it is also a key step when inverting a matrix or computing the determinant of a matrix. The LU decomposition was introduced by the Polish mathematician Tadeusz Banachiewicz in 1938.[1] It's also referred to as **LR** decomposition (factors into left and right triangular matrices).

Contents

Definitions

Let A be a square matrix. An **LU factorization** refers to the factorization of A, with proper row and/or column orderings or permutations, into two factors – a lower triangular matrix L and an upper triangular matrix U:

$$A = LU.$$



LDU decomposition of a Walsh matrix

In the lower triangular matrix all elements above the diagonal are zero, in the upper triangular matrix, all the elements below the diagonal are zero. For example, for a 3 × 3 matrix A, its LU decomposition looks like this:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

Without a proper ordering or permutations in the matrix, the factorization may fail to materialize. For example, it is easy to verify (by expanding the matrix multiplication) that $a_{11} = \ell_{11}u_{11}$. If $a_{11} = 0$, then at least one of ℓ_{11} and u_{11} has to be zero, which implies that either L or U is singular. This is impossible if A is nonsingular (invertible). This is a procedural problem. It can be removed by simply reordering the rows of A so that the first element of the permuted

matrix is nonzero. The same problem in subsequent factorization steps can be removed the same way; see the basic procedure below.

LU factorization with partial pivoting

It turns out that a proper permutation in rows (or columns) is sufficient for LU factorization. **LU factorization with partial pivoting** (LUP) refers often to LU factorization with row permutations only:

$$PA = LU,$$

where L and U are again lower and upper triangular matrices, and P is a permutation matrix, which, when left-multiplied to A, reorders the rows of A. It turns out that all square matrices can be factorized in this form,[2] and the factorization is numerically stable in practice.[3] This makes LUP decomposition a useful technique in practice.

LU factorization with full pivoting

An **LU factorization with full pivoting** involves both row and column permutations:

$$PAQ = LU,$$

where L, U and P are defined as before, and Q is a permutation matrix that reorders the columns of A.[4]

Lower-diagonal-upper (LDU) decomposition

A **Lower-diagonal-upper (LDU) decomposition** is a decomposition of the form

$$A = LDU,$$

where D is a diagonal matrix, and L and U are unitriangular matrices, meaning that all the entries on the diagonals of L and U are one.

Rectangular matrices

Above we required that A be a square matrix, but these decompositions can all be generalized to rectangular matrices as well.[5] In that case, L and D are square matrices both of which have the same number of rows as A, and U has exactly the same dimensions as A. *Upper triangular* should be interpreted as having only zero entries below the main diagonal, which starts at the upper left corner. Similarly, the more precise term for U is that it is the "row

echelon form" of the matrix A.

Example

We factor the following 2-by-2 matrix:

$$\begin{bmatrix} 4 & 3 \\ 6 & 3 \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 \\ \ell_{21} & \ell_{22} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}.$$

One way to find the LU decomposition of this simple matrix would be to simply solve the linear equations by inspection. Expanding the matrix multiplication gives

$$\begin{aligned} \ell_{11} \cdot u_{11} + 0 \cdot 0 &= 4 \\ \ell_{11} \cdot u_{12} + 0 \cdot u_{22} &= 3 \\ \ell_{21} \cdot u_{11} + \ell_{22} \cdot 0 &= 6 \\ \ell_{21} \cdot u_{12} + \ell_{22} \cdot u_{22} &= 3. \end{aligned}$$

This system of equations is underdetermined. In this case any two non-zero elements of L and U matrices are parameters of the solution and can be set arbitrarily to any non-zero value. Therefore, to find the unique LU decomposition, it is necessary to put some restriction on L and U matrices. For example, we can conveniently require the lower triangular matrix L to be a unit triangular matrix (i.e. set all the entries of its main diagonal to ones). Then the system of equations has the following solution:

$$\begin{aligned} \ell_{11} = \ell_{22} &= 1 \\ \ell_{21} &= 1.5 \\ u_{11} &= 4 \\ u_{12} &= 3 \\ u_{22} &= -1.5 \end{aligned}$$

Substituting these values into the LU decomposition above yields

$$\begin{bmatrix} 4 & 3 \\ 6 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1.5 & 1 \end{bmatrix} \begin{bmatrix} 4 & 3 \\ 0 & -1.5 \end{bmatrix}.$$

Existence and uniqueness

Square matrices

Any square matrix A admits *LUP* and *PLU* factorizations.[2] If A is invertible, then it admits an *LU* (or *LDU*) factorization if and only if all its leading principal minors[6] are nonzero[7] (for example $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ does not admit an *LU* or *LDU* factorization). If A is a singular matrix of rank k, then it admits an *LU* factorization if the first k leading principal minors are nonzero, although the converse is not true.[8]

If a square, invertible matrix has an *LDU* (factorization with all diagonal entries of L and U equal to 1), then the factorization is unique.[7] In that case, the *LU* factorization is also unique if we require that the diagonal of L (or U) consists of ones.

In general, any square matrix $A_{n\times n}$ could have one of the following:

1. a unique LU factorization (as mentioned above)
2. infinitely many LU factorizations if two or more of any first $(n-1)$ columns are linearly dependent or any of the first $(n-1)$ columns are 0, then A has infinitely many LU factorizations.
3. no LU factorization if the first $(n-1)$ columns are non-zero and linearly independent and at least one leading principal minor is zero.

In Case 3, one can approximate an LU factorization by changing a diagonal entry a_{jj} to $a_{jj} \pm \varepsilon$ to avoid a zero leading principal minor.[9]

Symmetric positive-definite matrices

If A is a symmetric (or Hermitian, if A is complex) positive-definite matrix, we can arrange matters so that U is the conjugate transpose of L. That is, we can write A as

$$A = LL^*.$$

This decomposition is called the Cholesky decomposition. The Cholesky decomposition always exists and is unique — provided the matrix is positive definite. Furthermore, computing the Cholesky decomposition is more efficient and numerically more stable than computing some other LU decompositions.

General matrices

For a (not necessarily invertible) matrix over any field, the exact necessary and sufficient conditions under which it has an LU factorization are known. The conditions are expressed in terms of the ranks of certain submatrices. The Gaussian elimination algorithm for obtaining LU decomposition has also been extended to this most general case.[10]

Algorithms

Closed formula

When an LDU factorization exists and is unique, there is a closed (explicit) formula for the elements of L, D, and U in terms of ratios of determinants of certain submatrices of the original matrix A.[11] In particular, $D_1 = A_{1,1}$, and for $i = 2, \ldots, n$, D_i is the ratio of the i-th principal submatrix to the $(i-1)$-th principal submatrix. Computation of the determinants is computationally expensive, so this explicit formula is not used in practice.

Using Gaussian elimination

The following algorithm is essentially a modified form of Gaussian elimination. Computing an LU decomposition using this algorithm requires $\frac{2}{3}n^3$ floating-point operations, ignoring lower-order terms. Partial pivoting adds only a quadratic term; this is not the case for full pivoting.[12]

Procedure

Given an $N \times N$ matrix $A = (a_{i,j})_{1 \le i,j \le N}$, define $A^{(0)}$ as the matrix A in which the necessary rows have been swapped to meet the desired conditions (such as partial pivoting) for the 1st column. The parenthetical superscript (e.g., (0)) of the matrix A is the version of the matrix. The matrix $A^{(n)}$ is the A matrix in which the elements below the main diagonal have already been eliminated to 0 through Gaussian elimination for the first n columns, and the necessary rows have been swapped to meet the desired conditions for the $(n+1)^{th}$ column.

We perform the operation $row_i = row_i - (\ell_{i,n}) \cdot row_n$ for each row i with elements (labelled as $a_{i,n}^{(n-1)}$ where $i = n+1, \ldots, N$) below the main diagonal in the n-th column of $A^{(n-1)}$. For this operation,

$$\ell_{i,n} := \frac{a_{i,n}^{(n-1)}}{a_{n,n}^{(n-1)}}.$$

We perform these row operations to eliminate the elements $a_{i,n}^{(n-1)}$ to zero. Once we have subtracted these rows, we may swap rows to provide the desired conditions for the $(n+1)^{th}$ column. We may swap rows here to perform partial pivoting, or because the element $a_{n+1,n+1}$ on the main diagonal is zero (and therefore cannot be used to implement Gaussian elimination).

We define the final permutation matrix $\boldsymbol{P}$ as the identity matrix which has all the same rows swapped in the same order as the $\boldsymbol{A}$ matrix.

Once we have performed the row operations for the first $N-1$ columns, we have obtained an upper triangular matrix $A^{(N-1)}$ which is denoted by $\boldsymbol{U}$. Note, we can denote $A^{(N-1)}$ as $\boldsymbol{U}$ because the N-th column of $A^{(N-1)}$ has no conditions for which rows need to be swapped. We can also calculate the lower triangular matrix denoted as $\boldsymbol{L}$, such that $\boldsymbol{PA} = \boldsymbol{LU}$, by directly inputting the values of values of $\ell_{i,n}$ via the formula below.

$$L = \begin{pmatrix} 1 & & & & & \\ \ell_{2,1} & \ddots & & & & \\ & \ddots & 1 & & & \\ \vdots & \cdots & \ell_{n+1,n} & 1 & & \\ & & \vdots & \ddots & \ddots & \\ \ell_{N,1} & \cdots & \ell_{N,n} & \cdots & \ell_{N,N-1} & 1 \end{pmatrix}$$

Example

If we are given the matrix

$$A = \begin{pmatrix} 0 & 5 & \frac{22}{3} \\ 4 & 2 & 1 \end{pmatrix},$$

$$\begin{pmatrix} 2 & 7 & 9 \end{pmatrix}$$

we will chose to implement partial pivoting and thus swap the first and second row so that our matrix A and the first iteration of our P matrix respectively become

$$A^{(0)} = \begin{pmatrix} 4 & 2 & 1 \\ 0 & 5 & \frac{22}{3} \\ 2 & 7 & 9 \end{pmatrix}, \quad P^{(0)} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Once we have swapped the rows, we can eliminate the elements below the main diagonal on the first column by performing

$$row_2 = row_2 - (\ell_{2,1}) \cdot row_1$$
$$row_3 = row_3 - (\ell_{3,1}) \cdot row_1$$

such that,

$$\ell_{2,1} = \frac{0}{4} = 0$$
$$\ell_{3,1} = \frac{2}{4} = 0.5$$

Once these rows have been subtracted, we have derived from $A^{(0)}$ the matrix

$$\begin{pmatrix} 4 & 2 & 1 \\ 0 & 5 & \frac{22}{3} \\ 0 & 6 & 8.5 \end{pmatrix}.$$

Because we are implementing partial pivoting, we swap the second and third rows of our derived matrix and the current version of our P matrix respectively to obtain

$$A^{(1)} = \begin{pmatrix} 4 & 2 & 1 \\ 0 & 6 & 8.5 \\ 0 & 5 & \frac{22}{3} \end{pmatrix}, \quad P^{(1)} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

Now, we eliminate the elements below the main diagonal on the second column by performing $row_3 = row_3 - (\ell_{3,2}) \cdot row_2$ such that $\ell_{3,2} = \frac{5}{6}$. Because no non-zero elements exist below the main diagonal in our current iteration of A after this row subtraction, this row subtraction derives our final A matrix (denoted as U) and

final P matrix:

$$A^{(2)} = A^{(N-1)} = U = \begin{pmatrix} 4 & 2 & 1 \\ 0 & 6 & 8.5 \\ 0 & 0 & 0.25 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

Now we can obtain our final L matrix:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ \ell_{2,1} & 1 & 0 \\ \ell_{3,1} & \ell_{3,2} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & \frac{5}{6} & 1 \end{pmatrix}$$

Now these matrices have a relation such that $PA = LU$.

Relations when no rows are swapped

If we did not swap rows at all during this process, we can perform the row operations simultaneously for each column n by setting $A^{(n)} := L_n A^{(n-1)}$, where L_n is the $N \times N$ identity matrix with its n-th column replaced by the transposed vector $\begin{pmatrix} 0 & \cdots & 0 & 1 & -\ell_{n+1,n} & \cdots & -\ell_{N,n} \end{pmatrix}^{\mathsf{T}}$. In other words, the lower triangular matrix

$$L_n = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\ell_{n+1,n} & & \\ & & \vdots & \ddots & \\ & & -\ell_{N,n} & & 1 \end{pmatrix}.$$

Performing all the row operations for the first $N-1$ columns using the $A^{(n)} := L_n A^{(n-1)}$ formula is equivalent to finding the decomposition

$$A = L_1^{-1} L_1 A^{(0)} = L_1^{-1} A^{(1)} = L_1^{-1} L_2^{-1} L_2 A^{(1)} = L_1^{-1} L_2^{-1} A^{(2)} = \cdots = L_1^{-1} \cdots L_{N-1}^{-1} A^{(N-1)}.$$

Denote $L = L_1^{-1} \cdots L_{N-1}^{-1}$ so that $A = LA^{(N-1)} = LU$.

Now let's compute the sequence of $L_1^{-1} \cdots L_{N-1}^{-1}$. We know that L_i^{-1} has the following formula.

$$L_n^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & \ell_{n+1,n} & & \\ & & \vdots & \ddots & \\ & & \ell_{N,n} & & 1 \end{pmatrix}$$

If there are two lower triangular matrices with 1s in the main diagonal, and neither have a non-zero item below the main diagonal in the same column as the other, then we can include all non-zero items at their same location in the product of the two matrices. For example:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 77 & 1 & 0 & 0 & 0 \\ 12 & 0 & 1 & 0 & 0 \\ 63 & 0 & 0 & 1 & 0 \\ 7 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 22 & 1 & 0 & 0 \\ 0 & 33 & 0 & 1 & 0 \\ 0 & 44 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 77 & 1 & 0 & 0 & 0 \\ 12 & 22 & 1 & 0 & 0 \\ 63 & 33 & 0 & 1 & 0 \\ 7 & 44 & 0 & 0 & 1 \end{pmatrix}$$

Finally, multiply L_i^{-1} together and generate the fused matrix denoted as L (as previously mentioned). Using the matrix L, we obtain $A = LU$.

It is clear that in order for this algorithm to work, one needs to have $a_{n,n}^{(n-1)} \neq 0$ at each step (see the definition of $\ell_{i,n}$). If this assumption fails at some point, one needs to interchange n-th row with another row below it before continuing. This is why an LU decomposition in general looks like $P^{-1}A = LU$.

LU Crout decomposition

Note that the decomposition obtained through this procedure is a *Doolittle decomposition*: the main diagonal of L is

composed solely of *1*s. If one would proceed by removing elements *above* the main diagonal by adding multiples of the *columns* (instead of removing elements *below* the diagonal by adding multiples of the *rows*), we would obtain a *Crout decomposition*, where the main diagonal of U is of *1*s.

Another (equivalent) way of producing a Crout decomposition of a given matrix A is to obtain a Doolittle decomposition of the transpose of A. Indeed, if $A^{\mathsf{T}} = L_0 U_0$ is the LU-decomposition obtained through the algorithm presented in this section, then by taking $L = U_0^{\mathsf{T}}$ and $U = L_0^{\mathsf{T}}$, we have that $A = LU$ is a Crout decomposition.

Through recursion

Cormen et al.[13] describe a recursive algorithm for LUP decomposition.

Given a matrix A, let P_1 be a permutation matrix such that

$$P_1 A = \left(\begin{array}{c|c} a & w^{\mathsf{T}} \\ \hline v & A' \end{array}\right),$$

where $a \neq 0$, if there is a nonzero entry in the first column of A; or take P_1 as the identity matrix otherwise. Now let $c = 1/a$, if $a \neq 0$; or $c = 0$ otherwise. We have

$$P_1 A = \left(\begin{array}{c|c} 1 & 0 \\ \hline cv & I_{n-1} \end{array}\right) \left(\begin{array}{c|c} a & w^{\mathsf{T}} \\ \hline 0 & A' - cvw^{\mathsf{T}} \end{array}\right).$$

Now we can recursively find an LUP decomposition $P' \left(A' - cvw^{\mathsf{T}}\right) = L'U'$. Let $v' = P'v$. Therefore

$$\left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & P' \end{array}\right) P_1 A = \left(\begin{array}{c|c} 1 & 0 \\ \hline c v' & L' \end{array}\right) \left(\begin{array}{c|c} a & w^{\mathsf{T}} \\ \hline 0 & U' \end{array}\right),$$

which is an LUP decomposition of A.

Randomized algorithm

It is possible to find a low rank approximation to an LU decomposition using a randomized algorithm. Given an input matrix $\boldsymbol{A}$ and a desired low rank $\boldsymbol{k}$, the randomized LU returns permutation matrices $\boldsymbol{P}, \boldsymbol{Q}$ and lower/upper trapezoidal matrices $\boldsymbol{L}, \boldsymbol{U}$ of size $\boldsymbol{m} \times \boldsymbol{k}$ and $\boldsymbol{k} \times \boldsymbol{n}$ respectively, such that with high probability $\|\boldsymbol{PAQ} - \boldsymbol{LU}\|_2 \leq \boldsymbol{C}\sigma_{\boldsymbol{k}+1}$, where $\boldsymbol{C}$ is a constant that depends on the parameters of the algorithm and $\sigma_{\boldsymbol{k}+1}$ is the $(\boldsymbol{k}+1)$-th singular value of the input matrix $\boldsymbol{A}$.[14]

Theoretical complexity

If two matrices of order n can be multiplied in time $M(n)$, where $M(n) \geq n^a$ for some $a > 2$, then an LU decomposition can be computed in time $O(M(n))$.[15] This means, for example, that an $O(n^{2.376})$ algorithm exists based on the Coppersmith–Winograd algorithm.

Sparse-matrix decomposition

Special algorithms have been developed for factorizing large sparse matrices. These algorithms attempt to find sparse factors L and U. Ideally, the cost of computation is determined by the number of nonzero entries, rather than by the size of the matrix.

These algorithms use the freedom to exchange rows and columns to minimize fill-in (entries that change from an initial zero to a non-zero value during the execution of an algorithm).

General treatment of orderings that minimize fill-in can be addressed using graph theory.

Applications

Solving linear equations

Given a system of linear equations in matrix form

$$A\mathbf{x} = \mathbf{b},$$

we want to solve the equation for $\mathbf{x}$, given A and $\mathbf{b}$. Suppose we have already obtained the LUP decomposition of A such that $PA = LU$, so $LU\mathbf{x} = P\mathbf{b}$.

In this case the solution is done in two logical steps:

1. First, we solve the equation $L\mathbf{y} = P\mathbf{b}$ for $\mathbf{y}$.
2. Second, we solve the equation $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$.

In both cases we are dealing with triangular matrices (L and U), which can be solved directly by forward and backward substitution without using the Gaussian elimination process (however we do need this process or equivalent to compute the LU decomposition itself).

The above procedure can be repeatedly applied to solve the equation multiple times for different $\mathbf{b}$. In this case it is faster (and more convenient) to do an LU decomposition of the matrix A once and then solve the triangular matrices for the different $\mathbf{b}$, rather than using Gaussian elimination each time. The matrices L and U could be thought to have "encoded" the Gaussian elimination process.

The cost of solving a system of linear equations is approximately $\frac{2}{3}n^3$ floating-point operations if the matrix A has size n. This makes it twice as fast as algorithms based on QR decomposition, which costs about $\frac{4}{3}n^3$ floating-point operations when Householder reflections are used. For this reason, LU decomposition is usually preferred.[16]

Inverting a matrix

When solving systems of equations, b is usually treated as a vector with a length equal to the height of matrix A. In matrix inversion however, instead of vector b, we have matrix B, where B is an n-by-p matrix, so that we are trying to find a matrix X (also a n-by-p matrix):

$$AX = LUX = B.$$

We can use the same algorithm presented earlier to solve for each column of matrix X. Now suppose that B is the identity matrix of size n. It would follow that the result X must be the inverse of A.[17]

Computing the determinant

Given the LUP decomposition $A = P^{-1}LU$ of a square matrix A, the determinant of A can be computed straightforwardly as

$$\det(A) = \det\left(P^{-1}\right)\det(L)\det(U) = (-1)^S \left(\prod_{i=1}^{n} l_{ii}\right)\left(\prod_{i=1}^{n} u_{ii}\right).$$

The second equation follows from the fact that the determinant of a triangular matrix is simply the product of its diagonal entries, and that the determinant of a permutation matrix is equal to $(-1)^S$ where S is the number of row exchanges in the decomposition.

In the case of LU decomposition with full pivoting, $\det(A)$ also equals the right-hand side of the above equation, if we let S be the total number of row and column exchanges.

The same method readily applies to LU decomposition by setting P equal to the identity matrix.

Code examples

C code example

```c
/* INPUT: A - array of pointers to rows of a square matrix having dimension N
 *        Tol - small tolerance number to detect failure when the matrix is near degenerate
 * OUTPUT: Matrix A is changed, it contains a copy of both matrices L-E and U as A=(L-E)+U such that P*A=L*U.
 *        The permutation matrix is not stored as a matrix, but in an integer vector P of size N+1
 *        containing column indexes where the permutation matrix has "1". The last element P[N]=S+N,
 *        where S is the number of row exchanges needed for determinant computation, det(P)=(-1)^S
 */
int LUPDecompose(double **A, int N, double Tol, int *P) {

    int i, j, k, imax;
    double maxA, *ptr, absA;

    for (i = 0; i <= N; i++)
        P[i] = i; //Unit permutation matrix, P[N] initialized with N

    for (i = 0; i < N; i++) {
        maxA = 0.0;
        imax = i;

        for (k = i; k < N; k++)
```

```
            if ((absA = fabs(A[k][i])) > maxA) {
                maxA = absA;
                imax = k;
            }

        if (maxA < Tol) return 0; //failure, matrix is degenerate

        if (imax != i) {
            //pivoting P
            j = P[i];
            P[i] = P[imax];
            P[imax] = j;

            //pivoting rows of A
            ptr = A[i];
            A[i] = A[imax];
            A[imax] = ptr;

            //counting pivots starting from N (for determinant)
            P[N]++;
        }

        for (j = i + 1; j < N; j++) {
            A[j][i] /= A[i][i];

            for (k = i + 1; k < N; k++)
                A[j][k] -= A[j][i] * A[i][k];
        }
    }

    return 1;  //decomposition done
}

/* INPUT: A,P filled in LUPDecompose; b - rhs vector; N - dimension
 * OUTPUT: x - solution vector of A*x=b
 */
void LUPSolve(double **A, int *P, double *b, int N, double *x) {

    for (int i = 0; i < N; i++) {
        x[i] = b[P[i]];

        for (int k = 0; k < i; k++)
            x[i] -= A[i][k] * x[k];
    }

    for (int i = N - 1; i >= 0; i--) {
        for (int k = i + 1; k < N; k++)
            x[i] -= A[i][k] * x[k];

        x[i] /= A[i][i];
    }
}

/* INPUT: A,P filled in LUPDecompose; N - dimension
```

```
 * OUTPUT: IA is the inverse of the initial matrix
 */
void LUPInvert(double **A, int *P, int N, double **IA) {

    for (int j = 0; j < N; j++) {
        for (int i = 0; i < N; i++) {
            IA[i][j] = P[i] == j ? 1.0 : 0.0;

            for (int k = 0; k < i; k++)
                IA[i][j] -= A[i][k] * IA[k][j];
        }

        for (int i = N - 1; i >= 0; i--) {
            for (int k = i + 1; k < N; k++)
                IA[i][j] -= A[i][k] * IA[k][j];

            IA[i][j] /= A[i][i];
        }
    }
}

/* INPUT: A,P filled in LUPDecompose; N - dimension.
 * OUTPUT: Function returns the determinant of the initial matrix
 */
double LUPDeterminant(double **A, int *P, int N) {

    double det = A[0][0];

    for (int i = 1; i < N; i++)
        det *= A[i][i];

    return (P[N] - N) % 2 == 0 ? det : -det;
}
```

C# code example

```
public class SystemOfLinearEquations
{
    public double[] SolveUsingLU(double[,] matrix, double[] rightPart, int n)
    {
        // decomposition of matrix
        double[,] lu = new double[n, n];
        double sum = 0;
        for (int i = 0; i < n; i++)
        {
            for (int j = i; j < n; j++)
            {
                sum = 0;
                for (int k = 0; k < i; k++)
                    sum += lu[i, k] * lu[k, j];
```

```
                lu[i, j] = matrix[i, j] - sum;
            }
            for (int j = i + 1; j < n; j++)
            {
                sum = 0;
                for (int k = 0; k < i; k++)
                    sum += lu[j, k] * lu[k, i];
                lu[j, i] = (1 / lu[i, i]) * (matrix[j, i] - sum);
            }
        }

        // lu = L+U-I
        // find solution of Ly = b
        double[] y = new double[n];
        for (int i = 0; i < n; i++)
        {
            sum = 0;
            for (int k = 0; k < i; k++)
                sum += lu[i, k] * y[k];
            y[i] = rightPart[i] - sum;
        }
        // find solution of Ux = y
        double[] x = new double[n];
        for (int i = n - 1; i >= 0; i--)
        {
            sum = 0;
            for (int k = i + 1; k < n; k++)
                sum += lu[i, k] * x[k];
            x[i] = (1 / lu[i, i]) * (y[i] - sum);
        }
        return x;
    }
}
```

MATLAB code example

```
function LU = LUDecompDoolittle(A)
    n = length(A);
    LU = A;
    % decomposition of matrix, Doolittle's Method
    for i = 1:1:n
        for j = 1:(i - 1)
            LU(i,j) = (LU(i,j) - LU(i,1:(j - 1))*LU(1:(j - 1),j)) / LU(j,j);
        end
        j = i:n;
        LU(i,j) = LU(i,j) - LU(i,1:(i - 1))*LU(1:(i - 1),j);
    end
    %LU = L+U-I
end
```

```
function x = SolveLinearSystem(LU, B)
    n = length(LU);
    y = zeros(size(B));
    % find solution of Ly = B
    for i = 1:n
        y(i,:) = B(i,:) - LU(i,1:i)*y(1:i,:);
    end
    % find solution of Ux = y
    x = zeros(size(B));
    for i = n:(-1):1
        x(i,:) = (y(i,:) - LU(i,(i + 1):n)*x((i + 1):n,:))/LU(i, i);
    end
end

A = [ 4 3 3; 6 3 3; 3 4 3 ]
LU = LUDecompDoolittle(A)
B = [ 1 2 3; 4 5 6; 7 8 9; 10 11 12 ]'
x = SolveLinearSystem(LU, B)
A * x
```

See also

- Block LU decomposition
- Bruhat decomposition
- Cholesky decomposition
- Crout matrix decomposition
- Incomplete LU factorization
- LU Reduction
- Matrix decomposition
- QR decomposition

Notes

1. Schwarzenberg-Czerny, A. (1995). "On matrix factorization and efficient least squares solution". *Astronomy and Astrophysics Supplement Series*. **110**: 405. Bibcode:1995A&AS..110..405S (https://ui.adsabs.harvard.edu/abs/1995A&AS..110..405S).
2. Okunev & Johnson (1997), Corollary 3.
3. Trefethen & Bau (1997), p. 166.
4. Trefethen & Bau (1997), p. 161.

5. Lay, David C. (2016). *Linear algebra and its applications* (https://www.worldcat.org/oclc/920463015). Steven R. Lay, Judith McDonald (Fifth ed.). Harlow. p. 142. ISBN 1-292-09223-8. OCLC 920463015 (https://www.worldcat.org/oclc/920463015).
6. Rigotti (2001), Leading Principle Minor
7. Horn & Johnson (1985), Corollary 3.5.5
8. Horn & Johnson (1985), Theorem 3.5.2
9. Nhiayi, Ly; Phan-Yamada, Tuyetdong (2021). "Examining Possible LU Decomposition". *North American GeoGebra Journal*. **9** (1).
10. Okunev & Johnson (1997)
11. Householder (1975)
12. Golub & Van Loan (1996), p. 112, 119.
13. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). *Introduction to Algorithms*. MIT Press and McGraw-Hill. ISBN 978-0-262-03293-3.
14. Shabat, Gil; Shmueli, Yaniv; Aizenbud, Yariv; Averbuch, Amir (2016). "Randomized LU Decomposition". *Applied and Computational Harmonic Analysis*. **44** (2): 246–272. arXiv:1310.7202 (https://arxiv.org/abs/1310.7202). doi:10.1016/j.acha.2016.04.006 (https://doi.org/10.1016%2Fj.acha.2016.04.006). S2CID 1900701 (https://api.semanticscholar.org/CorpusID:1900701).
15. Bunch & Hopcroft (1974)
16. Trefethen & Bau (1997), p. 152.
17. Golub & Van Loan (1996), p. 121

References

- Bunch, James R.; Hopcroft, John (1974), "Triangular factorization and inversion by fast matrix multiplication", *Mathematics of Computation*, **28** (125): 231–236, doi:10.2307/2005828 (https://doi.org/10.2307%2F2005828), ISSN 0025-5718 (https://www.worldcat.org/issn/0025-5718), JSTOR 2005828 (https://www.jstor.org/stable/2005828).
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), *Introduction to Algorithms*, MIT Press and McGraw-Hill, ISBN 978-0-262-03293-3.
- Golub, Gene H.; Van Loan, Charles F. (1996), *Matrix Computations* (3rd ed.), Baltimore: Johns Hopkins, ISBN 978-0-8018-5414-9.
- Horn, Roger A.; Johnson, Charles R. (1985), *Matrix Analysis*, Cambridge University Press, ISBN 978-0-521-38632-6. See Section 3.5. $N - 1$
- Householder, Alston S. (1975), *The Theory of Matrices in Numerical Analysis*, New York: Dover Publications, MR 0378371 (https://mathscinet.ams.org/mathscinet-getitem?mr=0378371).
- Okunev, Pavel; Johnson, Charles R. (1997), *Necessary And Sufficient Conditions For Existence of the LU Factorization of an Arbitrary Matrix*, arXiv:math.NA/0506382 (https://arxiv.org/abs/math.NA/0506382).

- Poole, David (2006), *Linear Algebra: A Modern Introduction* (2nd ed.), Canada: Thomson Brooks/Cole, ISBN 978-0-534-99845-5.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007), "Section 2.3" (http://apps.nrbook.com/empanel/index.html?pg=48), *Numerical Recipes: The Art of Scientific Computing* (3rd ed.), New York: Cambridge University Press, ISBN 978-0-521-88068-8.
- Trefethen, Lloyd N.; Bau, David (1997), *Numerical linear algebra*, Philadelphia: Society for Industrial and Applied Mathematics, ISBN 978-0-89871-361-9.
- Rigotti, Luca (2001), *ECON 2001 - Introduction to Mathematical Methods, Lecture 8* (https://sites.pitt.edu/~luca/ECON2001/)

External links

References

- LU decomposition (http://mathworld.wolfram.com/LUDecomposition.html) on *MathWorld*.
- LU decomposition (http://www.math-linux.com/spip.php?article51) on *Math-Linux*.
- LU decomposition (http://numericalmethods.eng.usf.edu/topics/lu_decomposition.html) at *Holistic Numerical Methods Institute*
- LU matrix factorization (http://www.mathworks.com/help/techdoc/ref/lu.html). MATLAB reference.

Computer code

- LAPACK (http://www.netlib.org/lapack/) is a collection of FORTRAN subroutines for solving dense linear algebra problems
- ALGLIB (http://www.alglib.net/) includes a partial port of the LAPACK to C++, C#, Delphi, etc.
- C++ code (https://web.archive.org/web/20120301111350/http://www.johnloomis.org/ece538/notes/Matrix/ludcmp.html), Prof. J. Loomis, University of Dayton
- C code (http://www.mymathlib.com/matrices/linearsystems/doolittle.html), Mathematics Source Library
- Rust code (https://github.com/NJdevPro/Rmatrix/blob/main/src/rmatrix.rs)
- LU in X10 (https://web.archive.org/web/20100429160954/http://docs.codehaus.org/display/XTENLANG/LU)

Online resources

- WebApp descriptively solving systems of linear equations with LU Decomposition (https://archive.today/20110425223706/http://sole.ooz.ie/)
- Matrix Calculator with steps, including LU decomposition (https://matrixcalc.org/),
- LU Decomposition Tool (https://web.archive.org/web/20081020061504/http://www.uni-bonn.de/~manfear/matrix

_lu.php), uni-bonn.de
- LU Decomposition (http://demonstrations.wolfram.com/LUDecomposition/) by Ed Pegg, Jr., The Wolfram Demonstrations Project, 2007.

Retrieved from "https://en.wikipedia.org/w/index.php?title=LU_decomposition&oldid=1137430688"
