

LU Factorisation*

Graeme Taylor

June 17, 2004

A significant proportion of the world's computing capacity and time is dedicated to the solution of systems of linear equations through manipulation of matrices, and as a result many methods have emerged to tackle the essential problem: finding a vector \mathbf{x} that satisfies the equation $A\mathbf{x} = \mathbf{b}$. A costly approach that is manageable for student problems (but rarely real-world ones) is to find the inverse of the matrix, as then pre-multiplication on each side yields $\mathbf{x} = A^{-1}\mathbf{b}$.

Indeed, there are many ways to go about finding the inverse, from the motivating but painfully slow example of finding adjugate matrices to row operations; but ultimately this approach is overkill. A system can be solved without finding the inverse by means of back substitution, a method that obtains the elements of \mathbf{x} based on eliminating successive rows in a triangular matrix. Of course, to apply back substitution you'll need a triangular matrix first, but if your initial A isn't of this type, all is not lost. A common means of obtaining a suitable matrix is to apply Gaussian elimination; but the LU factorisation offers another route which may save on computation, be easier to understand, or simpler to implement in code.

Definition

A pair of matrices $L, U \in \mathbb{R}^{N \times N}$ comprise a LU factorisation of $A \in \mathbb{R}^{N \times N}$ if the following conditions are met

- L is lower triangular (no entries above the main diagonal)
- $L_{ii} = 1 \forall i \in 1 \dots N$ (entries on the main diagonal are all 1)
- U is upper triangular (no entries below the main diagonal)
- $LU = A$ (L and U are actually factors!)

*First appeared on Everything2, at http://www.everything2.com/index.pl?node_id=1542931

Solving a system with LU factorisation

Given a matrix system $A\mathbf{x} = \mathbf{b}$, and an LU factorisation of A , proceed as follows:

- Use back substitution to solve $L\mathbf{y} = \mathbf{b}$
- Use back substitution to solve $U\mathbf{x} = \mathbf{y}$
- Then \mathbf{x} is precisely that desired, since $A\mathbf{x} = (LU)\mathbf{x} = L(U\mathbf{x}) = L\mathbf{y}$ by the second equation $= \mathbf{b}$ by the first equation.

Deriving an LU factorisation

We derive L and U iteratively, one row or column at a time. For the derivation, consider L as a series of columns L_1, \dots, L_N and U as a series of rows U_1, \dots, U_N .

To start the derivation, set L_1 equal to the first column of A , divided through by $a_{1,1}$ and U_1 equal to the first row.

Now construct a new matrix, $A^2 = A - L_1U_1$, where L_1U_1 denotes a vector product (i.e., a matrix where the i th row is U_1 multiplied by the i th entry of L_1 ; see the example if this is unclear!)

Now L_2 is the second column of A^2 divided by $a_{2,2}^2$; and U_2 is the second row of A^2 .

Continue in this vein to obtain the third column and row through use of A^3 etc. until all N rows and columns of L and U have been found (you should get that the final column of L is simply the elementary column vector e_N i.e. has 1 as the n th entry and zero elsewhere, due to the definition of L)

Complications and Complexity

It is worth noting that this procedure fails if any entry on the diagonal of A is zero, since then we are required to divide through by that zero to obtain a column of L , which is obviously invalid. The same scenario will also break Gaussian elimination, as it introduces a zero pivot. To avoid such problems, rows of the matrix would need to be reordered and the problem manipulated to compensate for these row operations. Testing for and resolving this issue adds to the overheads of an implementation of either method.

An efficient implementation of LU factorisation (using knowledge that many entries in L and U are zero, and the diagonal of L contains only ones, for instance) is of order $\frac{2N^3}{3}$ in terms of computing complexity- compare to the order N^2 calculations required for back substitution if we are fortunate enough to start with a triangular matrix. Although some savings can be made in special cases (See the Cholesky factorisation of SPD matrices), as N grows large such precise methods become unwieldy and iterative techniques need to be employed.

A sample LU factorisation

Let A be the 3X3 matrix

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

This gives $L_1 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} / 1$ and $U_1 = [1, 3, 2]$. We now construct A^2 :

$$\begin{pmatrix} 1 & 3 & 2 \\ 0 & 4 & 0 \\ -1 & 5 & 1 \end{pmatrix} - \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} [1, 3, 2] = \begin{pmatrix} 1 & 3 & 2 \\ 0 & 4 & 0 \\ -1 & 5 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 3 & 2 \\ 0 & 0 & 0 \\ -1 & -3 & -2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 8 & 3 \end{pmatrix}$$

This gives $L_2 = \begin{bmatrix} 0 \\ 4 \\ 8 \end{bmatrix} / 4$ and $U_2 = [0, 4, 0]$. We now construct A^3 :

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 8 & 3 \end{pmatrix} - \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} [0, 4, 0] = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 8 & 3 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 8 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

From which we get $L_3 = \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix} / 3$ and $U_3 = [0, 0, 3]$. This enables us to build our solution:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 2 & 1 \end{pmatrix} \text{ and } U = \begin{pmatrix} 1 & 3 & 2 \\ 0 & 4 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

You can multiply these out to check that we indeed obtain A .