LU Decomposition

CS 330

September 24, 2008

1 LU Decomposition

We can write an $N \times N$ matrix $\mathbf{A}$ as the product of a lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$ as follows (case $N = 4$):

$$\mathbf{LU} = \mathbf{A} \tag{1}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ \alpha_{10} & 1 & 0 & 0 \\ \alpha_{20} & \alpha_{21} & 1 & 0 \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & 1 \end{bmatrix} \begin{bmatrix} \beta_{00} & \beta_{01} & \beta_{02} & \beta_{03} \\ 0 & \beta_{11} & \beta_{12} & \beta_{13} \\ 0 & 0 & \beta_{22} & \beta_{23} \\ 0 & 0 & 0 & \beta_{33} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{2}$$

We then generate N^2 equations for the N^2 unknowns and order these equations according to the columns of $\mathbf{A}$ and underline the unknown to solve for as we progress:

$$\begin{aligned}
\underline{\beta_{00}} &= a_{00} \quad \text{column } j = 0 && (3)\\
\underline{\alpha_{10}}\beta_{00} &= a_{10} && (4)\\
\underline{\alpha_{20}}\beta_{00} &= a_{20} && (5)\\
\underline{\alpha_{30}}\beta_{00} &= a_{30} && (6)\\
\underline{\beta_{01}} &= a_{01} \quad \text{column } j = 1 && (7)\\
\alpha_{10}\beta_{01} + \underline{\beta_{11}} &= a_{11} && (8)\\
\alpha_{20}\beta_{01} + \underline{\alpha_{21}}\beta_{11} &= a_{21} && (9)\\
\alpha_{30}\beta_{01} + \underline{\alpha_{31}}\beta_{11} &= a_{31} && (10)\\
\underline{\beta_{02}} &= a_{02} \quad \text{column } j = 2 && (11)\\
\alpha_{10}\beta_{02} + \underline{\beta_{12}} &= a_{12} && (12)\\
\alpha_{20}\beta_{02} + \alpha_{21}\beta_{12} + \underline{\beta_{22}} &= a_{22} && (13)\\
\alpha_{30}\beta_{02} + \alpha_{31}\beta_{12} + \underline{\alpha_{32}}\beta_{22} &= a_{32} && (14)\\
\underline{\beta_{03}} &= a_{03} \quad \text{column } j = 3 && (15)\\
\alpha_{10}\beta_{03} + \underline{\beta_{13}} &= a_{13} && (16)\\
\alpha_{20}\beta_{03} + \alpha_{21}\beta_{13} + \underline{\beta_{23}} &= a_{23} && (17)\\
\alpha_{30}\beta_{03} + \alpha_{31}\beta_{13} + \alpha_{32}\beta_{23} + \underline{\beta_{33}} &= a_{33} && (18)
\end{aligned}$$

This reveals the direct method called *Crout's algorithm* or *Doolittle factorization* for solving for the unknowns:

for $j = 0 \ldots N-1$
    for $i = 0 \ldots j$
        $\beta_{ij} = a_{ij} - \sum_{k=0}^{i-1} \alpha_{ik}\beta_{kj}$
    for $i = j+1 \ldots N-1$
        $\alpha_{ij} = \frac{1}{\beta_{jj}} \left(a_{ij} - \sum_{k=0}^{j-1} \alpha_{ik}\beta_{kj} \right)$

We can store $\mathbf{L}$ and $\mathbf{U}$ in a single matrix since we do not need to explicitly store the zeroes:

$$\text{Combined } \mathbf{LU} \text{ matrix} = \begin{bmatrix} \beta_{00} & \beta_{01} & \beta_{02} & \beta_{03} \\ \alpha_{10} & \beta_{11} & \beta_{12} & \beta_{13} \\ \alpha_{20} & \alpha_{21} & \beta_{22} & \beta_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \beta_{33} \end{bmatrix}. \tag{19}$$

Furtherfore, each a_{ij} is referenced exactly once as we solve for each α_{ij} or β_{ij}, so we can replace $\mathbf{A}$ “in place” as we go. Crout’s “in place” modification algorithm is as follows:

1 **for** $j = 0 \dots N-1$
2   **for** $i = 0 \dots j$
3         $a_{ij} = a_{ij} - \sum_{k=0}^{i-1} a_{ik} a_{kj}$
4   **for** $i = j+1 \dots N-1$
5         $a_{ij} = \frac{1}{a_{jj}} \left(a_{ij} - \sum_{k=0}^{j-1} a_{ik} a_{kj} \right)$

1.1 Partial Pivoting

Line 5 of Crout’s algorithm has a problem when $\beta_{jj} \approx 0$. We can use *partial pivoting* (row swapping) to avoid this situation as much as possible. We will actually store $\mathbf{LU}$ for a row-wise permutation of $\mathbf{A}$ and record how $\mathbf{A}$ is permuted.

Line 3 computes the β_{ij} values on and above the diagonal ($i \leq j$). Line 5 computes the α_{ij} values below the diagonal ($i > j$). Note that the expression in parentheses on Line 5 is the same as the expression on the right hand side of Line 3 when $i = j$ (*i.e.,* on the diagonal). Therefore, we can put off the division by β_{jj} on Line 5 and wait to see if one of the α_{ij}’s below the diagonal would make a bettor pivot value; If so, we perform the row swap and go back and do the necessary divisions once the appropriate pivot value is in place. The array mutate$[0, \dots, N-1]$ records row permutations (*i.e.,* row i of $\mathbf{LU}$ equals row mutate$[i]$ of $\mathbf{A}$). The sign of d depends on the parity of the number of row exchanges (used for computing $|\mathbf{A}|$).

0	mutate$[] = \{0, \dots, N-1\}$	// Initialize row permutation array (no row exchanges yet).
1	$d = +1$	// Initialize row swap parity value.
2	**for** $j = 0 \dots N-1$	// We replace $\mathbf{A}$ with $\mathbf{LU}$ column by column…
3	**for** $i = 0 \dots j$	// Compute $a_{ij} \leftarrow \beta_{ij}$ on and above diagonal.
4	$a_{ij} = a_{ij} - \sum_{k=0}^{i-1} a_{ik} a_{kj}$	// (*Note:* If $i = 0$ then sum $= 0$.)
5	$p = \lvert a_{jj} \rvert$	// $p =$ initial pivot value
6	$n = j$	// $n =$ initial pivot row
7	**for** $i = j+1 \dots N-1$	// Compute $a_{ij} \leftarrow \alpha_{ij}$ below diagonal.
8	$a_{ij} = a_{ij} - \sum_{k=0}^{j-1} a_{ik} a_{kj}$	
9	**if** $\lvert a_{ij} \rvert > p$	// If better pivot found…
10	$p = \lvert a_{ij} \rvert$	// …then record new pivot.
11	$n = i$	
12	**if** $p = 0$ abort!	// Singular matrix! If $p \approx 0$ we may have problems.
13	**if** $n \neq j$	// If best pivot off diagonal…
14	swap rows n and j of $\mathbf{A}$	// …(*Note:* previous pivots unaltered)
15	swap mutate$[n]$ and mutate$[j]$	// …record row exchange
16	$d = -d$	// …flip parity
17	**for** $i = j+1 \dots N-1$	// perform divisions below diagonal
18	$a_{ij} = a_{ij} / a_{jj}$	

2 Applications

2.1 Solving $\mathbf{Ax} = \mathbf{b}$ for multiple right-hand sides

We can use the LU decomposition to solve for multiple systems of the form $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ remains the same but $\mathbf{b}$ changes. In fact, the $\mathbf{b}$'s do not need to known ahead of time. Given $\mathbf{A} = \mathbf{LU}$ we have

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} && (20) \\ (\mathbf{LU})\mathbf{x} = \mathbf{L}(\mathbf{Ux}) &= \mathbf{b}. && (21) \end{aligned}$$

We first solve

$$\mathbf{Ly} = \mathbf{b} \tag{22}$$

for $\mathbf{y}$ and then solve

$$\mathbf{Ux} = \mathbf{y} \tag{23}$$

for $\mathbf{x}$. Each triangular system can be easily solved. We solve Equation 22 via *forward substitution* (note that we must first permute $\mathbf{b}$ to account for row exchanges):

$$\begin{aligned} y_0 &= b_{\text{mutate}[0]}, \\ y_i &= b_{\text{mutate}[i]} - \sum_{j=0}^{i-1} \alpha_{ij} y_j \quad i = 1, \ldots, N-1. \end{aligned}$$

Equation 23 is solved by *back substitution*:

$$\begin{aligned} x_{N-1} &= \frac{y_{N-1}}{\beta_{N-1,N-1}}, \\ x_i &= \frac{1}{\beta_{ii}} \left(y_i - \sum_{j=i+1}^{N-1} \beta_{ij} x_j \right) \quad i = N-2, \ldots, 0. \end{aligned}$$

Note that the solution $\mathbf{x}$ does not need to be permuted since it represents a linear combination of the *columns* of $\mathbf{A}$, but we only performed *row* exchanges (partial pivoting).

Crout's algorithm requires $O(N^3)$ multiplications to perform LU decomposition. Forward and backsolving takes another $O(N^2)$ multiplications for each right hand side. Gaussian elimination and back-solving requires $O(N^3)$ operations. Solving a linear system via LU decomposition requires about a third of the operations needed via Gaussian elimination. In addition, we only need another $O(N^2)$ operations to solve using a different $\mathbf{b}$ vector!

2.1.1 Iterative Improvement

Given that $\mathbf{x}$ is the *exact* solution to Equation 20, the above procedure yields only an approximate solution $\hat{\mathbf{x}} = \mathbf{x} + \delta\mathbf{x}$ due to limited precision arithmetic. If we multiply $\mathbf{A}$ by our approximate we have

$$\begin{aligned} \mathbf{A}\hat{\mathbf{x}} &= \hat{\mathbf{b}} && (24) \\ \mathbf{A}(\mathbf{x} + \delta\mathbf{x}) &= \mathbf{b} + \delta\mathbf{b} && (25) \\ \mathbf{A}\delta\mathbf{x} &= \delta\mathbf{b}. && (26) \end{aligned}$$

Since we know $\mathbf{b}$ and we can compute $\hat{\mathbf{b}} = \mathbf{A}\hat{\mathbf{x}}$, we can determine $\delta\mathbf{b}$ as follows:

$$\delta\mathbf{b} = \mathbf{A}\hat{\mathbf{x}} - \mathbf{b} \tag{27}$$

(note that right-hand side should be computed with higher precision). Then we can solve Equation 26 for $\delta\mathbf{x}$ (using our LU decomposition). Our refined solution is then

$$\mathbf{x} = \hat{\mathbf{x}} - \delta\mathbf{x} \tag{28}$$

Lather, rinse, repeat as often as desired. Since we have already performed $O(N^3)$ operations computing $\mathbf{x}$, why not spend another $O(N^2)$ operations improving the solution?

2.2 Matrix Inversion

In this case, you have N right hand sides:

$$\mathbf{AX} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{29}$$

2.3 Determinant

$$|\mathbf{A}| = \prod_{j=0}^{N-1} \beta_{jj} \tag{30}$$

Note that we must scale the result by d (which is ± 1) to account for row exchanges.