CS412: Lecture #17

Mridul Aanjaneya

March 19, 2015

Solving linear systems of equations

Consider a lower triangular matrix L:

$$L = \begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 \\ \vdots & & & & \vdots \\ l_{n1} & \dots & \dots & \dots & l_{nn} \end{bmatrix}$$

A procedure similar to that for upper triangular systems can be followed to solve $Lx = b$. The overall complexity is again $O(n^2)$.

Algorithm 1 Forward substitution for $Lx = b$:

```
for j = 1...n do
    if l_jj = 0 then
        return.                          ▷ matrix is singular
    end if
    x_j ← b_j / l_jj
    for i = j + 1...n do
        b_i ← b_i − l_ij x_j
    end for
end for
```

The forward and backward substitution processes can be used to solve a non-triangular system by virtue of the following factorization property.

Theorem 1. *If A is an $n \times n$ matrix, it can be (generally) written as a product:*

$$A = LU$$

where L is a lower triangular matrix and U is an upper triangular matrix. Furthermore, it is possible to construct L such that all diagonal elements $l_{ii} = 1$.

Algorithm 2 *LU* factorization by Gaussian Elimination

```
1: for k = 1...n-1 do
2:     if a_kk = 0 then
3:         return.
4:     end if
5:     for i = k+1...n do
6:         a_ik ← a_ik/a_kk
7:     end for
8:     for j = k+1...n do
9:         for i = k+1...n do
10:            a_ij ← a_ij − a_ik a_kj
11:        end for
12:    end for
13: end for
```

Note that this algorithm executes in-place, i.e., the matrix A is *replaced* by its LU factorization, in compact form. More specifically, this algorithm produces a factorization $A = LU$, where:

$$L = \begin{bmatrix} 1 & & & & & \\ l_{21} & 1 & & & O & \\ l_{31} & l_{32} & 1 & & & \\ l_{41} & l_{42} & l_{43} & 1 & & \\ \vdots & \vdots & \vdots & & \ddots & \\ l_{n1} & n_{n2} & l_{n3} & \dots & l_{n,n-1} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & u_{33} & \dots & u_{3n} \\ & O & & \ddots & u_{n-1,n} \\ & & & & u_{nn} \end{bmatrix}$$

After the in-place factorization algorithm completes, A is replaced by the following "'compacted" encoding of L and U together:

$$A = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ l_{21} & u_{22} & u_{23} & \dots & u_{2n} \\ l_{31} & l_{32} & u_{33} & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{n-1,n} & u_{nn} \end{bmatrix}$$

Here is a slightly different algorithm for Gaussian Elimination via *elimination matrices*. Define the basis vector e_k as:

$$e_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

where the 1 is in the kth row and the length of e_k is n. In order to perform Gaussian Elimination on the kth column a_k of A, we define the $n \times n$ elimination matrix $M_k = I - m_k e_k^T$ where

$$m_k = \frac{1}{a_{kk}} \cdot \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{k+1,k} \\ \vdots \\ a_{n,k} \end{pmatrix}$$

M_k adds multiples of row k to the rows $> k$ in order to create 0's. As an example, for $a_k = (2, 4, -2)^T$

$$M_1 a_k = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$$

Similarly,

$$M_2 a_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix}$$

The inverse of an elimination matrix is defined as $L_k = M_k^{-1} = I + m_k e_k^T$. For example,

$$L_1 = M_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, \text{ and } L_2 = M_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1/2 & 1 \end{bmatrix}$$

The algorithm now proceeds as follows. Consider the example:

$$\begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix}$$

First, we eliminate the lower triangular portion of A one column at a time using M_k to get $U = M_{n-1} \dots M_1 A$. Note that we also carry out the operations on b to get a new system of equations $M - 2M_1 Ax = M_2 M_1 b$ or $Ux = M_2 M_1 b$ which can be solved for via back substitution.

$$M_1 A = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix}$$

$$M_1 b = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix}$$

$$M_2 M_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix}$$

$$M_2 M_1 b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix}$$

Finally, solve the following system via back substitution.

$$\begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix}$$

Note that using the fact that the L matrices are inverses of the M matrices allows us to write $LU = (L_1 \dots L_{n-1})(M_{n-1} \dots M_1 A) = A$ where $L = L_1 \dots L_{n-1}$ can be formed trivially from the M_k to obtain:

$$L = L_1 L_2 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix}$$

And thus, although we never needed it to solve the equations, the LU factorization of A is

$$A = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} = LU$$

Existence/Uniqueness and pivoting

When computing the LU factorization, the algorithm will *halt* if the diagonal element $a_{kk} = 0$. This can be avoided by swapping rows of A prior to computing the LU factorization. This is done to always select the largest a_{kk} from the equations that follow. As an example, consider the matrix A and the action of the permutation matrix P on it.

$$A = \begin{bmatrix} 1 & 2 & 5 & -1 \\ 0 & 0 & 3 & 1 \\ 0 & 4 & 1 & -2 \\ 0 & -6 & 0 & 3 \end{bmatrix}, P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, PA = \begin{bmatrix} 1 & 2 & 5 & -1 \\ 0 & -6 & 0 & 3 \\ 0 & 4 & 1 & -2 \\ 0 & 0 & 3 & 1 \end{bmatrix}$$

This is pivoting: the pivot a_{kk} is selected to be non-zero. In this process, we can guarantee uniqueness and existence of LU:

Theorem 2. *If P is a permutation matrix such that all pivots in the Gaussian Elimination of PA are non-zero, and $l_{ii} = 1$, then the LU factorization exists and is unique!*

$$PA = LU$$