Solving linear systems of equations

3/29/11

Our general strategy for solving a system $Ax=b$ will be to transform it to an equivalent, but easier to solve problem (or problems). An example of an easier sub-problem is a "triangular" system $Ux=b$ where

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

is an upper triangular matrix.

Here is an example, illustrating how such systems are easy to solve.

$$\begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -6 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -6 \\ 1 \end{bmatrix}$$

replace $x_3$ with $-1$

Solve: $x_3 = -1$

$\downarrow$

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & -4 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ -12 \\ -1 \end{bmatrix}$$

replace $x_2$ with $3$

Solve: $x_2 = 3$
\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
=egin{bmatrix}
-1 \\
3 \\
-1
\end{bmatrix}.
\]

We can write this procedure, formally, in pseudocode.

**Back-substitution for Upper Triangular System**

for \( j = n \) to 1

if \( u_{ij} = 0 \) then stop \[[matrix is singular]\]

\( x_{j} \leftarrow b_{j} / u_{jj} \) \hspace{1cm} (1)

for \( i = 1 \) to \( j - 1 \)

\( b_{i} \leftarrow b_{i} - u_{ij} x_{j} \) \hspace{1cm} (2)

end

end

By counting how many times the loops are executed, we see that \((n-1)\) divisions (line 1) are required, while line 2 is executed \( \frac{n(n-1)}{2} \) times. Overall, the cost of back substitution is proportional to \( O(n^2) \).
If \( L = \begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} \) is a lower triangular matrix, a similar procedure can be followed to solve \( Lx = b \):

**Forward substitution for \( Lx = b \)**

for \( j = 1 \) to \( n \)

if \( l_{jj} = 0 \) then stop \[ \text{[matrix is singular]} \]

\[ x_j \leftarrow \frac{b_j}{l_{jj}} \]

for \( i = j+1 \) to \( n \)

\[ b_i \leftarrow b_i - l_{ij} x_j \]

end

end

Cost = \( O(n^3) \) again.

The forward- and backward substitution processes can be used to solve a non-triangular system by virtue of the following factorization property:
Thm  If $A$ is an $n \times n$ matrix, it can be (generally) written as a product:

$$A = LU,$$

where $L = \text{lower triangular}$ and $U = \text{upper triangular}$.

Furthermore it is possible to construct $L$ such that all diagonal elements $l_{ii} = 1$

Algorithm: LU factorization by Gaussian Elimination

[Note: This algorithm executes in-place, i.e. the matrix $A$ is replaced by its LU factorization, in compact form]

for $k = 1$ to $n - 1$

if $a_{kk} = 0$ then stop

for $i = k + 1$ to $n$

$$a_{ik} \leftarrow a_{ik} / a_{kk}$$

end

for $j = k + 1$ to $n$

for $i = k + 1$ to $n$

$$a_{ij} = a_{ij} - a_{ik} a_{kj}$$

end

end

$A^{3} \approx \frac{n^{3}}{3}$ times

$\text{cost} = O(n^{3})$.
More specifically, this algorithm produces a factorization \( A = LU \), where:

\[
L = \begin{bmatrix}
1 \\
l_{21} & 1 \\
l_{31} & l_{32} & 1 \\
l_{41} & l_{42} & l_{43} & 1 \\
\vdots & \vdots & \ddots & \ddots \\
l_{n1} & l_{n2} & l_{n3} & \cdots & l_{n,n-1} & 1
\end{bmatrix}
\]

\[
U = \begin{bmatrix}
u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\
0 & u_{22} & u_{23} & \cdots & u_{2n} \\
0 & 0 & u_{33} & \cdots & u_{3n} \\
\vdots & \vdots & \vdots & \ddots & \ddots \\
0 & 0 & 0 & \cdots & u_{nn}
\end{bmatrix}
\]

After the in-place factorization algorithm completes, \( A \) is replaced by the following "compacted" encoding of \( LU \) together:

\[
A = \begin{bmatrix}
u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\
l_{21} & u_{22} & u_{23} & \cdots & u_{2n} \\
l_{31} & l_{32} & u_{33} & \cdots & u_{3n} \\
\vdots & \vdots & \vdots & \ddots & \ddots \\
l_{n1} & l_{n2} & l_{n3} & \cdots & u_{nn}
\end{bmatrix}
\]
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.

\[
A = \begin{bmatrix}
1 & 2 & 5 & -1 \\
0 & \circ & 3 & 1 \\
0 & 4 & 1 & -2 \\
0 & -6 & 0 & 8
\end{bmatrix}
\]

\[
P = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \circ & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

I with swap columns

\[
PA = \begin{bmatrix}
1 & 2 & 5 & -1 \\
0 & -6 & 0 & 3 \\
0 & 4 & 1 & -2 \\
0 & 0 & 3 & 1
\end{bmatrix}
\]
This is pivoting: The pivot a_{k,k} is selected to be nonzero. In this process, we can guarantee uniqueness & existence of \(LU\):

**Thm** 
If \(P\) is a permutation matrix such that all pivots in the Gaussian Elimination of \(PA\) are non-zero, and \(l_{kk} = 1\), then the \(LU\) factorization exists and is unique!

\[
PA = LU.
\]

Solving \(Ax = b\)

- Without pivoting:

\[
A\ x = b
\]

\[
LU\ x = b \quad \Rightarrow \quad 1. \text{ Solve } \ Ly = b \quad (F.S).
\]

\[
= y
\]

\[
2. \text{ Solve } \ U\ x = y \quad (B.S).
\]

\(x\) is the solution.
$[L,U] = lu(A) = \text{cost } O(n^3)$

$y = b \backslash L \implies \text{cost } O(n^2)$

$x = y \backslash L \implies \text{cost } O(n^2).$

If have multiple systems $A_{x_i} = b_i = \text{cost of LU only once.}$

With pivoting: For $P^T \cdot P = I$

$A_{x} = b \iff PAX = Pb \iff P^T \cdot PAx = P^T \cdot Pb$

$LUX = Pb$

$Ly = Pb$

$UX = y$
\[ [L, U, P] = \text{lu}(A) \quad \rightarrow \quad \text{cost } O(n^3) \]
\[ z = Pb \quad \rightarrow \quad \text{cost } O(n) \]
\[ y = L \backslash z \quad \rightarrow \quad \text{cost } O(n^2) \]
\[ x = U \backslash y \quad \rightarrow \quad \text{cost } O(n^2) \]

**Final variant: Full pivoting**

In this case, when we are in the \( k \)-th step of the Gauss-Elimination / LU procedure, we pick the pivot element among the entire \( (n-k+1) \times (n-k+1) \) lower-rightmost submatrix of \( A \).

\[
\text{e.g. } k=2 \text{ and } Ax=b \Rightarrow \begin{bmatrix}
1 & 2 & 5 \\
0 & 0 & 3 \\
0 & 4 & 1 \\
0 & -6 & 0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix}
= \begin{bmatrix}
4 \\
7 \\
8 \\
2
\end{bmatrix}
\]

In this case, we can bring \((-8)\) to the pivot position \( a_{22} \) by permuting both rows 2-3 and columns 2-4. Naturally, we will respectively swap rows 2-3 of the RHS, and rows 2-4 of the unknown vector.
Thus: \[
\begin{bmatrix}
1 & 2 & 5 & -1 \\
0 & -8 & 1 & 4 \\
0 & 1 & 3 & 0 \\
0 & 3 & 0 & -6
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_4 \\
x_3 \\
x_2
\end{bmatrix}
=
\begin{bmatrix}
4 \\
8 \\
7 \\
2
\end{bmatrix}
\] is the equivalent system!

This process is encoded in the LU factorization using permutation matrices P & Q such that \[PAQ = LU\]

The solution is then computed via:

\[Ax = b \Rightarrow PAQQ^T x = Pb \Rightarrow (LU)(Q^T x) = Pb\]

\[\Rightarrow L(UQ^T x) = Pb \Rightarrow y \text{ found via lower triangular solve}\]

\[UQ^T x = y \Rightarrow z \text{ found via upper triangular solve}\]

\[Q^T x = z \Rightarrow QQ^T x = Qz \Rightarrow x = Qz\]

\[
\text{In Matlab:}
\]

\[
[L, U, P, Q] = lu(A); \quad \rightarrow \text{Cost } O(n^3)
\]

\[
c = Pb; \quad \rightarrow \text{Cost } O(n)
\]

\[
y = \text{L} \backslash c; \quad \rightarrow \text{Cost } O(n^2)
\]

\[
z = \text{U} \backslash y; \quad \rightarrow \text{Cost } O(n^2)
\]

\[
x = \text{Q} \times z; \quad \rightarrow \text{Cost } O(n)
\]