Solving linear systems of equations

3/29/11 L1

Our general strategy for solving a system $Ax=b$ will be to <u>transform</u> 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

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

is an upper triangular matrix.

Here is anexample, 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: $x_2 = 3$

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

We can write this procedure, formally, in pseudocode

<u>Back-substitution for Upper Triangular System</u>

for $j = n$ to 1

  if $u_{jj} = 0$ then <u>stop</u> [matrix is singular]

  $x_j \leftarrow b_j / u_{jj}$ (1)

  for $i = 1$ to $j-1$

    $b_i \leftarrow b_i - u_{ij} x_j$ (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 $L = \begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & 0 \\ \vdots & & & & \vdots \\ l_{n1} & \cdots & & \cdots & l_{nn} \end{bmatrix}$ is a lower triangular matrix, a similar procedure can be followed to solve $Lx = b$:

3/29/11

<u>Forward substitution for $Lx=b$</u>

for $j = 1$ to n
  if $l_{jj} = 0$ then stop   [matrix is singular]
  $x_j \leftarrow b_j / l_{jj}$
  for $i = j+1$ to n
    $b_i \leftarrow b_i - l_{ij} x_j$
  end
end

<u>Cost $= O(n^2)$ again.</u>

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

<u>Thm</u> If A is an $n \times n$ matrix, it can be (generally) written as a product:

$$A = LU.$$

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

<u>Algorithm</u> : LU factorization by Gaussian Elimination
[<u>Note</u>: This algorithm executes in-place. i.e. the matrix A is <u>replaced</u> 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 <- 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
end.
```

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

Cost $= O(n^3)$.

More specifically, this algoritm produces a factorization $A = LU$, where:

$$L = \begin{bmatrix} 1 & & & & & 0 \\ l_{21} & 1 & & & & \\ l_{31} & l_{32} & 1 & & & \\ l_{41} & l_{42} & l_{43} & 1 & & \\ \vdots & \vdots & \vdots & & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{n,n-1} & 1 \end{bmatrix}$$

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

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

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

3/29/11 LE

Existence/Uniqueness and pivoting

When computing the LU factorization, the algorithm will <u>halt</u> 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

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

largest

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_{kk} is selected to be nonzero. In this process, we can guarrantee 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 $l_{ii} = 1$, then the LU factorization exists and is unique!

$$PA = LU.$$

Solving $Ax = b$

- Without pivoting:

$$Ax = b$$

$$L\underbrace{Ux}_{=y} = b \implies$$

1. Solve $Ly = b$ (F.S).
2. Solve $Ux = y$ (B.S).

x is the solution.

In MATLAB

3/29/11

$[L, U] = lu(A) \Rightarrow$ Cost $O(n^3)$

$y = b \backslash L \Rightarrow$ Cost $O(n^2)$

$x = y \backslash L \Rightarrow$ Cost $O(n^2)$.

If have multiple systems $Ax_i = b_i \Rightarrow$ cost of LU only once.

- With pivoting: For $P^T P = I$

$Ax = b \Leftrightarrow PAx = Pb \Leftrightarrow \cancel{P^T P} Ax = \cancel{P^T P} b$

$L \underbrace{Ux}_{y} = Pb$

$Ly = Pb$

$Ux = y$

Matlab

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

$z = Pb;$ $\quad\rightarrow$ Cost $O(n)$

$y = L\backslash z;$ $\quad\rightarrow$ Cost $O(n^2)$

$x = U\backslash y;$ $\quad\rightarrow$ Cost $O(n^2)$

Final variant: <u>Full pivoting</u>

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

pivot position

e.g. $k=2$ and $Ax=b \Rightarrow$

$$\begin{bmatrix} 1 & 2 & 5 & -1 \\ 0 & \boxed{0} & 3 & 1 \\ 0 & 4 & 1 & \boxed{-8} \\ 0 & -6 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 8 \\ 2 \end{bmatrix}$$

submatrix $\qquad$ largest element.

In this case, we can bring (-8) to the pivot position a_{22} by permuting <u>both</u> rows 2-3 <u>AND</u> columns 2-4. Naturally, we will respectively swap rows 2-3 of the RHS, and rows 2-4 of the unknown vector.

Thus :

$$\begin{bmatrix} 1 & 2 & 5 & -1 \\ 0 & \boxed{-8} & 1 & 4 \\ 0 & 1 & 3 & 0 \\ 0 & 3 & 0 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_4 \\ x_3 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 7 \\ 2 \end{bmatrix}$$

is the equivalent system !

This process is encoded in the LU factorization using $\underline{\underline{2}}$ permutation matrices P & Q such that $\boxed{PAQ = LU}$

The solution is then computed via:

$$Ax = b \Rightarrow PA\underbrace{QQ^T}_{=I}x = Pb \Rightarrow (LU)(Q^Tx) = Pb$$

$$\Rightarrow L\underbrace{(UQ^Tx)}_{y} = Pb \Rightarrow y \text{ found via lower triangular solve}$$

$$U\underbrace{Q^Tx}_{z} = y \Rightarrow z \text{ found via upper triangular solve}$$

$$Q^Tx = z \Rightarrow QQ^Tx = Qz \Rightarrow \boxed{x = Qz}$$

<u>In Matlab</u> :

```
[L,U,P,Q] = lu(A);    -> Cost O(n^3)
  c = Pb;             -> Cost O(n)
  y = L\c;            -> Cost O(n^2)
  z = U\y;            -> Cost O(n^2)
  x = Q*u;            -> Cost O(n).
```