Computational Methods
CMSC/AMSC/MAPL 460

Linear Systems, LU Decomposition,

Ramani Duraiswami,
Dept. of Computer Science
Look at LU code

%%%LU Triangular factorization
%%% [L,U,p] = lutx(A) produces a unit lower triangular
%%% matrix L, an upper triangular matrix U, and a
%%% permutation vector p, so that L*U = A(p,:).

\begin{verbatim}
[n,n] = size(A);
p = (1:n)';

for k = 1:n-1
    % Find largest element below diagonal in k-th column
    [r,m] = max(abs(A(k:n,k)));
    m = m+k-1;

    % Skip elimination if column is zero
    if (A(m,k) ~= 0)
        % Swap pivot row
        if (m ~= k)
            A([k m],:) = A([m k],:);
            p([k m]) = p([m k]);
        end
    end
\end{verbatim}

• Initialize
  – Matrix size
  – Permutation vector

• Second output argument to max is index of max element

• If max element is zero then we need not eliminate

• Exchange rows

• Update permutation vector
Look at LU code

- Multipliers for each row below diagonal
  - Note multipliers are stored in the lower triangular part of A
- Vectorized update
  - $A(i,k)\times A(k,j)$ multiplies column vector by row vector to produce a square, rank 1 matrix of order $n-k$.
  - Matrix is then subtracted from the submatrix of the same size in the bottom right corner of $A$.
  - In a programming language without vector and matrix operations, this update of a portion of $A$ would be done with doubly nested loops on $i$ and $j$.
  - Cost is $n^2$ and done $n$ times for a total cost of $n^3$
- Computes decomposition in the matrix $A$ itself
- Here they are separated, but when memory is important it can be left there

```matlab
%% Compute multipliers
i = k+1:n;
A(i,k) = A(i,k)/A(k,k);

%% Update the remainder of the matrix
j = k+1:n;
A(i,j) = A(i,j) - A(i,k)*A(k,j);
end
end

%% Separate result
L = tril(A,-1) + eye(n,n);
U = triu(A);
```
\[ [r, m] = \max (\abs{A(k:n, k)}); \]

\[ r = -1; \quad \%\% \text{ initialize } r \text{ and } m \]

\[ m = 1; \]

\[ \text{for } kk = k:n \]

\[ r_{\text{now}} = \max (\abs{A(kk, k)}, r); \]

\[ \text{if } (r_{\text{now}} \neq r) \]

\[ m = kk; \]

\[ r = r_{\text{now}}; \]

\[ \text{end} \]

\[ \text{end} \]
if (m ~= k)
    A([k m],:) = A([m k],:);
    p([k m]) = p([m k]);
end
if (m ~= k)
    temp1=A(k,kk);
    A(k,kk)=A(m,kk);
    A(m,kk)=temp1;
end
temp1=p(k);
p(k)=p(m);
p(m)=temp1;
end
% Compute multipliers
i = k+1:n;
A(i,k) = A(i,k)/A(k,k);

% Update the remainder of the matrix
j = k+1:n;
A(i,j) = A(i,j) - A(i,k)*A(k,j);

for i=k+1:n;
    A(i,k) = A(i,k)/A(k,k);
    for j=k+1:n
        A(i,j) = A(i,j) - A(i,k)*A(k,j);
    end
end
Code to solve linear system using LU

- In Matlab the backslash operator can be used to solve linear systems.
  
- For square matrices it employs LU or special variants
  - Lower triangular
  - Upper triangular
  - symmetric

- Symmetric LU is called Cholesky decomposition
  - $A=LL^T$
  - Upper and lower triangular are equal (transposes)
  - If matrix not positive-definite go to regular solution

```matlab
function x = bslashtx(A,b)
% BSLASHTX  Solve linear system (backslash)
% x = bslashtx(A,b) solves A*x = b

[n,n] = size(A);
if isequal(triu(A,1),zeros(n,n))
    % Lower triangular
    x = forward(A,b);
    return
elseif isequal(tril(A,-1),zeros(n,n))
    % Upper triangular
    x = backsubs(A,b);
    return
elseif isequal(A,A')
    [R,fail] = chol(A);
    if ~fail
        % Positive definite
        y = forward(R',b);
        x = backsubs(R,y);
        return
    end
end
end
```
Code continues

- Call LU
  - Solve $y = Lb$
  - Solve $x = Uy$

```matlab
% Triangular factorization
[L,U,p] = lutx(A);

% Permutation and forward elimination
y = forward(L,b(p));
x = backsubs(U,y);
```

function $x = forward(L,x)$

% FORWARD. Forward elimination.
% For lower triangular $L$, $x = forward(L,b)$ solves $L*x = b$.
[n,n] = size(L);
for $k = 1:n$
    $j = 1:k-1$;
    $x(k) = (x(k) - L(k,j)*x(j))/L(k,k)$;
end

function $x = backsubs(U,x)$

% BACKSUBS. Back substitution.
% For upper triangular $U$, $x = backsubs(U,b)$ solves $U*x = b$.
[n,n] = size(U);
for $k = n:-1:1$
    $j = k+1:n$;
    $x(k) = (x(k) - U(k,j)*x(j))/U(k,k)$;
end
Is pivoting necessary in LU?

- Consider
\[
\begin{bmatrix}
\delta & 1 \\
1 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
= 
\begin{bmatrix}
1 \\
0
\end{bmatrix}
\]

- Exact solution is
\[
x =
\begin{bmatrix}
-\frac{1}{1-\delta} \\
\frac{1}{1-\delta}
\end{bmatrix}
\]

- Let \( \delta < 0.5 \cdot \varepsilon \)

- Solution without pivoting gives
\[
\begin{bmatrix}
\delta & 1 \\
0 & -1/\delta
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
= 
\begin{bmatrix}
1 \\
-1/\delta
\end{bmatrix}
\]

\[
x_2 = 1, \quad x_1 = 0.
\]
Is pivoting necessary?

- With pivoting
  \[
  \begin{bmatrix}
  1 & 1 \\
  \delta & 1
  \end{bmatrix}
  \begin{bmatrix}
  x_1 \\
  x_2
  \end{bmatrix}
  =
  \begin{bmatrix}
  0 \\
  1
  \end{bmatrix}
  \]

- Elimination gives
  \[
  \begin{bmatrix}
  1 & 1 \\
  0 & 1
  \end{bmatrix}
  \begin{bmatrix}
  x_1 \\
  x_2
  \end{bmatrix}
  =
  \begin{bmatrix}
  0 \\
  1
  \end{bmatrix}
  \]

- With answers
  \[x_2 = 1, \quad x_1 = -1\]

- Close to exact