Computational Methods
CMSC/AMSC/MAPL 460

LU Decomposition, Sensitivity

Ramani Duraiswami,
Dept. of Computer Science
Solving a system with the LU decomposition

\[Ax = b\]
\[LU = PA\]
\[P^T LUx = b\]
\[L[Ux] = Pb\]
Solve \(Ly = Pb\)
Then \(Ux = y\)
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,:).

[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

• Initialize
  – Matrix size
  – Permutation vector

• Second argument is index with max element

• If max element is zero then we need not eliminate

• Exchange rows

• Change permut vector
Look at LU code

- Multipliers for each row below diagonal
  - Note multipliers are stored in the lower triangular part of A
- Vectorized update
  - $A(i,k)A(k,j)$ 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 $j$.
- Computes decomposition in the matrix $A$ itself
- Here they are separated, but when memory is important it can be left there

```
% 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);
```
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))
    x = forward(A,b);
    return
elseif isequal(tril(A,-1),zeros(n,n))
    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);
    end
end
```
Code continues

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

```matlab
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
```
% Triangular factorization
[L,U,p] = lutex(A);
% Permutation and forward elimination
y = forward(L,b(p));
x = backsubs(U,y);
LU Wrap up

- Operations count: $n^3/3$ multiplications.

- Matlab's `backslash` operator solves linear systems, using LU, without forming the inverse:

  ```
  x = A \ b;
  ```

- If you have $k$ right-hand sides involving the same matrix, store them as columns in a matrix $B$ of size $n \times k$ and then solve using, for example

  ```
  X = A \ B;
  ```

What about sparsity?

If $A$ has lots of zeros, we would like our algorithms to take advantage of this, and not to ruin the structure by introducing many nonzeros.

If $A$ is initialized as a sparse matrix in Matlab, then backslash and the `lu` algorithm both try to preserve sparsity.
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

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

- Let \( \delta < 0.5 \epsilon \)

- 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
Another example from the book

\[
\begin{pmatrix}
10 & -7 & 0 \\
-3 & 2.099 & 6 \\
5 & -1 & 5 \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
\end{pmatrix}
=
\begin{pmatrix}
7 \\
3.901 \\
6 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
10 & -7 & 0 \\
0 & -0.001 & 6 \\
0 & 2.5 & 5 \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
\end{pmatrix}
=
\begin{pmatrix}
7 \\
6.001 \\
2.5 \\
\end{pmatrix}
\]

\[(5 + (2.5 \cdot 10^3)(6))x_3 = (2.5 + (2.5 \cdot 10^3)(6.001))\]

\[(5 + 1.5000 \cdot 10^4)x_3 = (2.5 + 1.50025 \cdot 10^4)\]

\[1.5005 \cdot 10^4 x_3 = 1.5004 \cdot 10^4\]
Another example from the book

\[ x_3 = \frac{1.5004 \cdot 10^4}{1.5005 \cdot 10^4} = 0.99993 \]

\[-0.001x_2 + (6)(0.99993) = 6.001 \]

\[ x_2 = \frac{1.5 \cdot 10^{-3}}{-1.0 \cdot 10^{-3}} = -1.5 \]

\[ 10x_1 + (-7)(-1.5) = 7 \]

\[ x_1 = -0.35 \]

Correct answer is \((0, -1, 1)^T\)
How accurate are answers from LU?

- We solve the equation \( Ax = b \)
- Let true solution be \( x^* \)
- Let obtained solution be \( x \)
- Then error is \( e = x^* - x \)
  - Error is not computable (also called “Forward” error)
- New concept “residual” (also called “Backward error”)
  - Residual is the difference between the original right hand side and the right hand side obtained with the obtained solution \( r = b - Ax \)
- Guarantee: LU produces answers with small residuals
  - on computers with IEEE floating point
- Do small residuals mean small errors?
Return to our example

• Compute residual

\[ r \equiv b - Ax \]
\[ = \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ \delta & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \]
\[ = \begin{bmatrix} 0 \\ \delta \end{bmatrix}. \]

• We have exactly solved a nearby problem

\[ Ax = b - r \]
Another example

assume 3-digit decimal arithmetic.

\[
\begin{bmatrix}
.780 & .563 \\
.913 & .659
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} =
\begin{bmatrix}
.217 \\
.254
\end{bmatrix}
\]

If we compute the solution with pivoting, we obtain

\[
x = \begin{bmatrix}
-.443 \\
1.000
\end{bmatrix},
\quad r = \begin{bmatrix}
-.000460 \\
-.000541
\end{bmatrix}
\]

\[
x_{true} = \begin{bmatrix}
1.000 \\
-1.000
\end{bmatrix}
\]

• Solution has small residual but very large error
• In fact signs of the solution are opposite!
• Why?
  • Can condition numbers tell us what is going on?
Condition numbers

The first problem is **well-conditioned**; small changes in the data produce small changes in the answer.

The second problem is **ill-conditioned**; small changes in the data can produce large changes in the answer.

• Recall definition of condition number
Condition Number of a Matrix

A measure of how close a matrix is to singular

\[ \text{cond}(A) = \kappa(A) = \|A\| \cdot \|A^{-1}\| \]

\[ = \frac{\text{maximum stretch}}{\text{maximum shrink}} = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|} \]

- \( \text{cond}(I) = 1 \)
- \( \text{cond}(\text{singular matrix}) = \infty \)
- Norm can be any norm
- One norm is easy to compute
Relation between condition number and error

\[ Ax_{true} = b \quad \rightarrow \quad \|b\| = \|Ax_{true}\| \leq \|A\| \|x_{true}\| \]

\[ \|x_{true}\| \geq \frac{\|b\|}{\|A\|} \quad \rightarrow \quad \frac{1}{\|x_{true}\|} \leq \frac{\|A\|}{\|b\|} \]

\[ Ax = b - r \quad \rightarrow \quad A(x_{true} - x) = r \]

\[ (x_{true} - x) = A^{-1}r \quad \rightarrow \quad \|x_{true} - x\| \leq \|A^{-1}\| \|r\| \]

\[ \frac{\|x_{true} - x\|}{\|x_{true}\|} \leq \frac{\|r\|}{\|b\|} \|A\| \|A^{-1}\| = \frac{\|r\|}{\|b\|} \kappa(A). \]

- In words: relative error is smaller than norm of residual divided by norm of rhs times condition number
- So larger condition number means larger error
Properties of the condition number

Some properties:

- $\kappa(A) \geq 1$ for all matrices.
- $\kappa(A) = \infty$ for singular matrices.
- $\kappa(cA) = \kappa(A)$ for any nonzero scalar $c$.
- $\kappa(D) = \max |d_{ii}| / \min |d_{ii}|$ if $D$ is diagonal.
- $\kappa$ measures closeness to singularity better than the determinant.
Closing remarks

- Never compute matrix inverse
- Use a stable algorithm
- Check residual and condition number of problem
- If condition number is large, do not trust solution
  - Can problem be reformulated somehow?