Computational Methods
CMSC/AMSC/MAPL 460

Wrap up Linear Systems, LU Decomposition,

Ramani Duraiswami,
Dept. of Computer Science
Gaussian Elimination

- Zero elements of first column below 1st row
  - multiplying 1st row by 0.3 and add to 2nd row
  - multiplying 1st row by -0.5 and add to 3rd row
  - Results in

- Zero elements of first column below 2nd row
  - Swap rows
  - Multiply 2nd row by 0.04 and add to 3rd
Pivoting

- Every step involves dividing by diagonal in the current row.
- Algorithm – should work for general data.
- Remember: whenever an algorithm calls for division, we need to check if the entry being divided by can become zero (or almost zero).
- Consider
- Here the system has solution.
- Yet division by zero would occur!
- Fix: rearrange the system
- Element by which we divide is called “Pivot element”
- Changing the pivot element is called “pivoting”
- Partial Pivoting and Full Pivoting

\[
\begin{bmatrix}
0 & 4 & 4 \\
4 & 0 & 2 \\
2 & 0 & 2
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
=
\begin{bmatrix}
4 \\
2 \\
0
\end{bmatrix}
\]

\[
x = [1, 2, -1]^T
\]
Partial Pivoting

• Interchange rows below the current row to ensure that the largest element by magnitude is in the current row
• Also possible to do column interchanges in addition to row interchanges (called full pivoting)
Solution

- Start from last equation which can be solved by division
- Next substitute in the previous line and continue
- This describes the way to do the algorithm by hand
- How to represent it using matrices?
- Also, how do we solve another system that has the same matrix?
  - Upper triangular matrix we end up with will be the same, but the sequence of operations on the r.h.s needs to be repeated
\[ LU = PA \]

\[
L = \begin{pmatrix}
1 & 0 & 0 \\
0.5 & 1 & 0 \\
-0.3 & -0.04 & 1
\end{pmatrix}
\quad
U = \begin{pmatrix}
10 & -7 & 0 \\
0 & 2.5 & 5 \\
0 & 0 & 6.2
\end{pmatrix}
\quad
P = \begin{pmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{pmatrix}
\]

- Identify the elements of \( L \) and \( P \)?
- \( L \) has the multipliers we used in the elimination steps
- \( P \) has a record of the row swaps we did to avoid dividing by small numbers
- In fact we can write each step of Gaussian elimination in matrix form

\[
U = M_{n-1}P_{n-1} \cdots M_2P_2M_1P_1A
\]

\[
L_1L_2 \cdots L_{n-1}U = P_{n-1} \cdots P_2P_1A
\]
$$A = \begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} \quad L = L_1L_2 \cdots L_{n-1} \quad P = P_{n-1} \cdots P_2P_1$$

the matrices defined during the elimination are

$$P_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0.3 & 1 & 0 \\ -0.5 & 0 & 1 \end{pmatrix} \quad L_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.3 & 0 & 1 \end{pmatrix}$$

$$P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0.04 & 1 \end{pmatrix}, L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -0.04 & 1 \end{pmatrix},$$
Solving a system with the LU decomposition

\[ Ax = b \]

\[ LU = PA \]

\[ P^T LUx = b \]

\[ L[Ux] = Pb \]

Solve \( Ly = Pb \)

Then \( Ux = y \)
• Book keeping to account for changes in a permutation matrix
• (If we did full pivoting we need two permutation matrices to account in addition for column interchanges)
• We can represent the permutation matrix as a vector
  – Convenient in matlab
How good are the answers given by LU?

• In general we cannot determine the exact answer.
• Rather we will determine answer (possibly incorrect), and use it to find how poorly it does in predicting the right hand side
• Difference in r.h.s is the “residual” and is a measure of the error

\[ r \equiv b - Ax \]

\[
= \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ \delta & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\
= \begin{bmatrix} 0 \\ \delta \end{bmatrix}.
\]

Suppose \( \delta < 0.5 \epsilon_{mach} \).

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

; system without pivoting, we’ll get

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

\[ x_2 = 1, \ x_1 = 0. \]

True solution

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

With pivoting we get

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

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

\[ x_2 = 1, \ x_1 = -1. \]
• Recall “backward error analysis”
• Determine region in “problem space” where the problem we solved lies
• Here the problem we solved is $Ax = b-r$
• Theorem
  – Gauss elimination with partial pivoting produces small residuals
• Next question:
  – Does a small residue mean a small forward error?
  – Here it did
Example 2

- Compute via GE with partial pivoting
  Let's 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

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

- However true solution is \([1.000 \ -1.000]^t\)
- Residual was small, but error is large!
- Why?
- Recall condition number
Condition Number of a Matrix

A measure of how close a matrix is to singular

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

$$= \frac{\text{maximum stretch}}{\text{maximum shrink}} = \frac{\max \lambda_i}{\min \lambda_i}$$

- $\text{cond}(I) = 1$
- $\text{cond}(\text{singular matrix}) = \infty$
- So even though residual was small, error was multiplied by the condition number, and was significant
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,:).

• Initialize
  – Matrix size
  – Permutation vector

\[
\begin{align*}
  [n,n] &= \text{size}(A); \\
p &= (1:n),
\end{align*}
\]

\[
\text{for } k = 1:n-1
\]

% Find largest element below diagonal in k-th column
\[
[r,m] = \text{max}(\text{abs}(A(k:n,k)));
\]

\[
m = m + k - 1;
\]

% Skip elimination if column is zero
\[
\text{if } (A(m,k) \approx 0)
\]

% Swap pivot row
\[
\text{if } (m \approx k)
\]

\[
A([k \ m],:) = A([m \ k],:); \\
p([k \ m]) = p([m \ k]);
\]

end

• 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
  - $A(i,k)A(k,j)$ 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

% 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)) % 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
function x = forward(L, x)
% FORWARD. Forward elimination.
% For lower triangular $L$, $x = forward(L, b)$ solves $L \times x = b$.
[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 $U \times x = b$.
[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] = lutx(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:

\[
x = A \ \backslash \ b;
\]

- 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

\[
X = A \ \backslash \ B;
\]

**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.