MATH7030 Advanced Numerical Methods

Chapter 2 Direct Methods for Linear Systems

BY YULIANG WANG

北京师范大学
香港浸会大学 联合国际学院

BEIJING NORMAL UNIVERSITY · HONG KONG BAPTIST UNIVERSITY
UNITED INTERNATIONAL COLLEGE

Email: yuliangwang@uic.edu.cn

1 Gaussian elimination

Example 1.1 Consider the linear system

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

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 0 & 2 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & -3 & -6 & -4 \\ 0 & -6 & -21 & -5 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & -3 & -6 & -4 \\ 0 & 0 & -9 & 3 \end{array}\right) \longrightarrow \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} -2 \\ 2 \\ -4 \end{pmatrix}$$

$$L_1=\begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -7 & 0 & 1 \end{pmatrix}, \quad L_2=\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix}$$

Gaussian elimination

$$A=\begin{pmatrix}1&2&3\\4&5&6\\7&8&0\end{pmatrix},\quad L_1A=\begin{pmatrix}1&2&3\\0&-3&-6\\0&-6&-21\end{pmatrix},\quad L_2L_1A=\begin{pmatrix}1&2&3\\0&-3&-6\\0&0&-9\end{pmatrix}:=U$$

$$L=L_2L_1=\begin{pmatrix}1&0&0\\-4&1&0\\1&-2&1\end{pmatrix},\quad L^{-1}=L_1^{-1}L_2^{-1}=\begin{pmatrix}1&0&0\\4&1&0\\7&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\0&2&1\end{pmatrix}=\begin{pmatrix}1&0&0\\4&1&0\\7&2&1\end{pmatrix}$$

$$A=LU$$

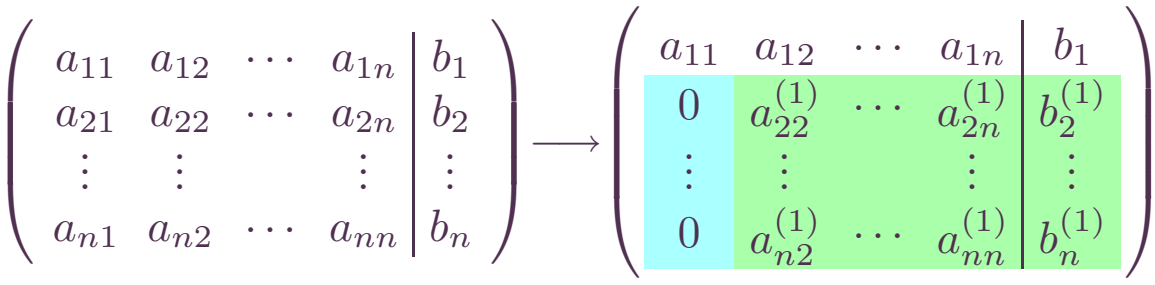
$$Ax=b\Rightarrow(LU)x=b\Rightarrow Ly=b\Rightarrow y=L^{-1}b\Rightarrow x=U^{-1}y$$

Exercise 1.1. Determine and verify a general rule for computing L and L^{-1}.

Gaussian elimination

```
% Step 1 of Gaussian elimination.
n = 3;
A = [1 2 3; 4 5 6; 7 8 0];
b = [1;0;2];
for i=2:n % Loop over rows below row 1.
    mult = A(i,1)/A(1,1);
    A(i,:) = A(i,:) - mult*A(1,:);
    b(i) = b(i) - mult*b(1);
end
```

Gaussian elimination



Repeat on the submatrix (green part)

```
% Gaussian elimination without pivoting
n = 3;
A = [1 2 3; 4 5 6; 7 8 0];
b = [1;0;2];
for j=1:n-1
    for i=j+1:n
        mult = A(i,j)/A(j,j);
        % A(i,:) = A(i,:) - mult*A(j,:); % less efficient
        A(i,j:n) = A(i,j:n) - mult*A(j,j:n); % more efficient
        b(i) = b(i) - mult*b(j);
    end
end
```

Operation counts

- operation counts (without improvement)

$$\sum_{j=1}^{n-1} \sum_{i=j+1}^{n} (1+2n+2) = \sum_{j=1}^{n-1} (n-j)(2n+3) = (2n+3)\frac{n(n-1)}{2} = n^3 + O(n^2)$$

- operation counts (with improvement)

$$\sum_{j=1}^{n-1} \sum_{i=j+1}^{n} (1+2(n-j+1)+2) = \frac{2}{3}n^3 + O(n^2)$$

- We can save the L and U factors so that they can be used to solve linear systems with the same coefficient matrix but different right-hand side vectors.
- We can use **in-place modification** to save memory.

LU factorization

LU factorization (without pivoting)

```
% LU factorization without pivoting
function [L,U] = mylu_no_pivot(A)

[m, n] = size(A);
if m ~= n
    error('Input matrix must be square for LU factorization.');
end
for j = 1:n-1
    for i = j+1:n
        mult = A(i,j)/A(j,j);
        A(i,j+1:n) = A(i,j+1:n) - mult*A(j,j+1:n);
        A(i,j) = mult;
    end
end
% the strict lower part of A becomes the strict lower part of L
L = tril(A, -1)+eye(size(A));
% the upper part of A becomes U
U = triu(A);
```

LU factorization

The operation counts of the above algorithm is also $\frac{2}{3}n^3 + O(n^2)$.

$$Ax = b \Rightarrow (LU)x = b \Rightarrow Ly = b \Rightarrow y = L^{-1}b \Rightarrow x = U^{-1}y$$

First, solve the lower triangular system $Ly = b$

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{21} & 1 & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{2n} & \dots & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \Longrightarrow$$

$$\begin{aligned} y_1 &= b_1 \\ y_2 &= b_2 - \ell_{21}\, y_1 \\ &\vdots \\ y_i &= b_i - \sum_{j=1}^{i-1} \ell_{ij}\, y_j \end{aligned}$$

```
function y = lsolve(L,b)
% y = lsolve(L,b)
% Given a lower triangular matrix L with unit diagonal
% and a vector b, this routine solves Ly = b and returns the solution y.
n = length(b);
y = zeros(n,1);
for i = 1:n
    y(i) = b(i);
    for j = 1:i-1
        y(i) = y(i) - L(i,j)*y(j);
    end
end
```

operation counts: $n^2 + O(n)$.

Exercise 1.2. Write a code to solve the upper triangular system $Ux = y$ and count the operations.

Pivoting

- The Gaussian elimination or LU factorization fail if $a_{jj} = 0$ at the j-th step for some j.
- If $a_{kj} \neq 0$ for some $k > j$, we can interchange row j and k. This is called **pivoting**, and a_{kj} is called the **pivot element**.
- If $a_{kj} = 0$ for all $k \geqslant j$, then A is singular (verify).

Which k if there are multiple choices?

A tiny pivot element can cause troubles in finite-precision arithmetic.

Example 1.2 Consider

$$\begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} \frac{1}{10^{20}-1} \\ -\frac{1}{10^{20}-1} \end{pmatrix}.$$

Gaussian elimination in double precision: $(x_1, x_2) = (0, 1)$. Large relative error!

Interchange row 1 and 2 before GE: $(x_1, x_2) = (1, 1)$.

Partial pivoting

Partial pivoting: searches for the largest entry in the columnin absolute value and uses that as the pivot element.

extra cost: $O(n^2)$ operation counts + data movement

Example 1.3

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 0 & 2 \end{array}\right) \xrightarrow{\text{pivot}} \left(\begin{array}{ccc|c} 7 & 8 & 0 & 2 \\ 4 & 5 & 6 & 0 \\ 1 & 2 & 3 & 1 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 7 & 8 & 0 & 2 \\ 0 & 3/7 & 6 & -8/7 \\ 0 & 6/7 & 3 & 5/7 \end{array}\right)$$

$$\xrightarrow{\text{pivot}} \left(\begin{array}{ccc|c} 7 & 8 & 0 & 2 \\ 0 & 6/7 & 3 & 5/7 \\ 0 & 3/7 & 6 & -8/7 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 7 & 8 & 0 & 2 \\ 0 & 6/7 & 3 & 5/7 \\ 0 & 0 & 9/2 & -3/2 \end{array}\right)$$

Partial pivoting

```
% Gaussian elimination with partial pivoting
n = 2;
A = [1e-20 1; 1 1];
b = [1;2];
for j=1:n-1 % Loop over columns.
    [pivot,k] = max(abs(A(j:n,j)));
    if pivot==0
        error('Matrix is singular.')
    end
    temp = A(j,:);
    A(j,:) = A(k+j-1,:);
    A(k+j-1,:) = temp;
    tempb = b(j);
    b(j) = b(k+j-1);
    b(k+j-1) = tempb;
    for i=j+1:n % Loop over rows below j.
        mult = A(i,j)/A(j,j);
        A(i,j:n) = A(i,j:n) - mult*A(j,j:n);
        b(i) = b(i) - mult*b(j);
    end
end
```

Partial pivoting

Theorem 1.4 Every n by n nonsingular matrix A can be factored in the form $A = PLU$, where P is a **permutation matrix**, L is a unit lower triangular matrix, and U is an upper triangular matrix.

Exercise 1.3. Write a MATLAB code for the PLU factorization.

Partial pivoting

- For certain matrices, pivoting in Gaussian elimination is not required.
- For **symmetric positive definite** (SPD) matrices, Gaussian elimination can be performed stably without pivoting.
- If A is SPD, then we have the **Cholesky decomposition** $A = LL^T$. (Also consider the LDL decomposition $A = LDL^T$) project?
- Gaussian elimination without pivoting is also stable for **strictly diagonally dominant** matrices. A matrix A is strictly diagonally dominant if

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad \text{for all } i = 1, \ldots, n.$$

- A matrix A is called **banded** if $a_{ij} = 0$, if $|i - j| > m$, where $m \ll n$ is the half bandwidth.
- SPD and banded matrices often arise in the discretization of partial differential equations.

Implementation Considerations for High Performance

- **Memory references** (fetch and store) may be costly on today's supercomputers.
- Basic Linear Algebra Subprograms (BLAS) has its programs arranged in 3 levels.
  - $\mathbf{y} \leftarrow \mathbf{ax} + \mathbf{y}$
  - $\mathbf{y} \leftarrow A\mathbf{x} + \mathbf{y}$
  - $C \leftarrow AB + C$