

Unit 1 Linear System Solutions

Numerical Analysis

EE/NTHU

Mar. 12, 2020

Linear Systems

- In using computers to solve numerical problems, one often encounters linear systems

$$\mathbf{Ax} = \mathbf{b}. \quad (1.1.1)$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}. \quad (1.1.2)$$

\mathbf{A} is the $n \times n$ **coefficient matrix**. \mathbf{b} is an n -vector, also known as the **right-hand-side vector**. \mathbf{x} , which is also an n -vector, is the **unknown vector** to be solved for.

- It is also assumed in this course that \mathbf{A} and \mathbf{b} are real though the techniques developed can be applied when they are complex.
- \mathbf{A} can also be expressed as

$$\mathbf{A} = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_n]. \quad (1.1.3)$$

where $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$ are n **column vectors** of matrix \mathbf{A} .

Theorem 1.1.1.

The equation (1.1.1) has a unique solution if one of the following conditions holds

1. \mathbf{A} is invertible,
2. $\text{rank}(\mathbf{A}) = n$,
3. the homogeneous system $\mathbf{Ax} = \mathbf{0}$ has only trivial solution of $\mathbf{x} = \mathbf{0}$.

- If the solution exists, then it can be found by Cramer's rule

$$x_i = \frac{\Delta_i}{\det(\mathbf{A})}, \quad i = 1, \dots, n. \quad (1.1.4)$$

where Δ_i is the determinant of the matrix obtained by replacing the i -th column of \mathbf{A} by the right-hand side vector \mathbf{b} .

- This formula is, however, too slow to be useful.

Matrix Inversion

- Solving the linear system is closely related to matrix inversion problem.
Given

$$\mathbf{Ax} = \mathbf{b},$$

- If \mathbf{A}^{-1} is known then

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}. \quad (1.1.5)$$

- Let

$$\mathbf{A}^{-1} = [\bar{\mathbf{a}}_1 \quad \bar{\mathbf{a}}_2 \quad \cdots \quad \bar{\mathbf{a}}_n], \quad (1.1.6)$$

since

$$\mathbf{AA}^{-1} = \mathbf{I}, \quad (1.1.7)$$

$$\mathbf{AA}^{-1} = [\mathbf{e}_1 \quad \mathbf{e}_2 \quad \cdots \quad \mathbf{e}_n], \quad (1.1.8)$$

then $\bar{\mathbf{a}}_i$ is the solution of the linear system

$$\mathbf{A}\bar{\mathbf{a}}_i = \mathbf{e}_i. \quad (1.1.9)$$

Thus, we can find the inverse of matrix \mathbf{A} if we know how to solve the linear system; and if we know how to solve the linear system we can find \mathbf{A}^{-1} using Eq. (1.1.9).

Gaussian Elimination

- The linear system Eq. (1.1.1) can be solved by a familiar method:
– Gaussian Elimination.

Example 1.1.2.

Find the solution to the following linear system

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

Solution. Assuming $a_{11} \neq 0$, from new row'_2 and row'_3 by

$$\begin{aligned} row'_2 &= row_2 - \frac{a_{21}}{a_{11}} \times row_1 \\ row'_3 &= row_3 - \frac{a_{31}}{a_{11}} \times row_1 \end{aligned}$$

The linear system then becomes

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & a'_{32} & a'_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ b'_3 \end{bmatrix}.$$

Gaussian Elimination, II

Assuming $a'_{22} \neq 0$, form new row''_3 again by

$$row''_3 = row'_3 - \frac{a'_{32}}{a'_{22}} \times row'_2$$

And the linear system becomes

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a''_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ b''_3 \end{bmatrix}.$$

And the solution can be found to be.

$$\begin{aligned} x_3 &= b''_3 / a''_{33} \\ x_2 &= (b'_2 - a'_{23}x_3) / a'_{22} \\ x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3) / a_{11} \end{aligned}$$

- This is the **Gaussian Elimination** method. The process is to transform the original matrix into an upper triangle matrix. Once that is done, **backward substitution** can be used to find the solution.
- Note also that the diagonal elements were assumed to be nonzero. If any of them is zero, **row pivoting** needs to be performed to avoid divide-by-zero error.

Gaussian Elimination, III

Algorithm 1.1.3. Gaussian Elimination

```
01 void GE(double A[n][n], double b[n])
02 {
03     int i, j, k;
04     double y;
05
06     for (i = 0; i < n - 1; i++) {
07         for (j = i + 1; j <= n - 1; j++) {
08             y = A[j][i] / A[i][i];
09             for (k = i; k <= n - 1; k++) {
10                 A[j][k] -= y * A[i][k];
11             }
12             b[j] -= y * b[i];
13         }
14     }
15 }
```

- Number of division operations $\frac{n(n-1)}{2}$.
- Number of multiplication-subtraction operations $\frac{n^3 - n}{3}$.

Gaussian Elimination, IV

Algorithm 1.1.4. Backward Substitution

```
01 void BckSubst(double A[n][n], double b[n], double x[n])
02 {
03     int i, j;
04
05     for (i = n - 1; i >= 0; i--) {
06         x[i] = b[i];
07         for (j = i + 1; j <= n - 1; j++) {
08             x[i] -= A[i][j] * x[j];
09         }
10         x[i] /= A[i][i];
11     }
12 }
```

- Number of multiplication-subtraction operations: $\frac{n(n-1)}{2}$.
- Number of divisions: n .
- Solution complexity of Gaussian elimination method is dominated by the elimination process $\mathcal{O}(n^3)$.

Gaussian Elimination, Pivoting

- In Gaussian elimination process the diagonal elements need to be nonzero, otherwise the algorithm will fail.

- Example

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & a'_{32} & a'_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ b'_3 \end{bmatrix}.$$

- a'_{22} can be zero, even though \mathbf{A} is nonsingular.
- In this case, one can swap the 2nd and the 3rd row to form the equivalent system and then carry out the elimination process.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{32} & a'_{33} \\ 0 & a'_{22} & a'_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_3 \\ b'_2 \end{bmatrix}.$$

- In fact, for solution stability and accuracy it is desirable to select the element with the largest absolute value as the diagonal element (pivot).
- Gaussian elimination with pivoting.
 - Partial pivoting, row or column,
 - Full pivoting, row and column.

Gaussian Elimination with Partial Pivoting

Algorithm 1.1.5. Gaussian Elimination with Partial Pivoting

```
01 void GE_PP(double A[n][n], double b[n])
02 {
03     int i, j, k;
04     double y;
05
06     for (i = 0; i < n - 1; i++) {
07         y = fabs(A[i][i]);
08         for (k = i + 1; k <= n - 1; k++)
09             if (fabs(A[k][i]) > y) {
10                 y = fabs(A[k][i]); k = k;
11             }
12         if (i != k) {
13             for (j = i; j < n; j++) {
14                 y = A[i][j]; A[i][j] = A[k][j]; A[k][j] = y;
15             }
16             y = b[i]; b[i] = b[k]; b[k] = y;
17         }
18         for (j = i + 1; j <= n - 1; j++) {
19             y = A[j][i] / A[i][i];
20             for (k = i + 1; k <= n - 1; k++)
21                 A[j][k] -= y * A[i][k];
22             b[j] -= y * b[i];
23         }
24     }
25 }
```

LU Decomposition

- The preceding Gaussian elimination is robust in solving linear system of equations.
 - But when the right hand side vector \mathbf{b} is changed, the entire process needs to be carried out again.
 - LU decomposition can be more effective in solving the linear system with different \mathbf{b} vectors.
- LU decomposition assumes matrix \mathbf{A} can be factorized to be the product of two matrices \mathbf{L} and \mathbf{U} such that \mathbf{L} is a lower triangle matrix and \mathbf{U} is an upper triangle matrix.

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U} \quad (1.1.10)$$

- Example

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \quad (1.1.11)$$

- Note that we set $\ell_{ii} = 1, 1 \leq i \leq n$.

LU Decomposition, II

- When Eq. (1.1.11) is multiplied out, we get

$$a_{11} = u_{11}$$

$$u_{11} = a_{11}$$

$$a_{12} = u_{12}$$

$$u_{12} = a_{12}$$

$$a_{13} = u_{13}$$

$$u_{13} = a_{13}$$

$$a_{21} = \ell_{21} \cdot u_{11}$$

$$\ell_{21} = a_{21} / u_{11}$$

$$a_{22} = \ell_{21} \cdot u_{12} + u_{22}$$

$$u_{22} = a_{22} - \ell_{21} \cdot u_{12}$$

$$a_{23} = \ell_{21} \cdot u_{13} + u_{23}$$

$$u_{23} = a_{23} - \ell_{21} \cdot u_{13}$$

$$a_{31} = \ell_{31} \cdot u_{11}$$

$$\ell_{31} = a_{31} / u_{11}$$

$$a_{32} = \ell_{31} \cdot u_{12} + \ell_{32} \cdot u_{22}$$

$$\ell_{32} = (a_{32} - \ell_{31} \cdot u_{12}) / u_{22}$$

$$a_{33} = \ell_{31} \cdot u_{13} + \ell_{32} \cdot u_{23} + u_{33}$$

$$u_{33} = a_{33} - \ell_{31} \cdot u_{13} - \ell_{32} \cdot u_{23}$$

- Note the order of evaluation is important.

LU Decomposition, III

- Or it can be rearranged as

$$u_{11} = a_{11}$$

$$u_{12} = a_{12}$$

$$u_{13} = a_{13}$$

$$\ell_{21} = a_{21}/u_{11}$$

$$\ell_{31} = a_{31}/u_{11}$$

$$u_{22} = a_{22} - \ell_{21} \cdot u_{12}$$

$$u_{23} = a_{23} - \ell_{21} \cdot u_{13}$$

$$\ell_{32} = (a_{32} - \ell_{31} \cdot u_{12})/u_{22}$$

$$u_{33} = a_{33} - \ell_{31} \cdot u_{13} - \ell_{32} \cdot u_{23}$$

- Or divide into 3 steps

$$u_{11} = a_{11}$$

$$u_{12} = a_{12}$$

$$u_{13} = a_{13}$$

$$\ell_{21} = a_{21}/u_{11}$$

$$\ell_{31} = a_{31}/u_{11}$$

$$a'_{22} = a_{22} - \ell_{21} \cdot u_{12}$$

$$a'_{23} = a_{23} - \ell_{21} \cdot u_{13}$$

$$a'_{32} = a_{32} - \ell_{31} \cdot u_{12}$$

$$a'_{33} = a_{33} - \ell_{31} \cdot u_{13}$$

$$u_{22} = a'_{22}$$

$$u_{23} = a'_{23}$$

$$\ell_{32} = a'_{32}/u_{22}$$

$$a''_{33} = a'_{33} - \ell_{32} \cdot u_{23}$$

$$u_{33} = a''_{33}$$

LU Decomposition, IV

- Note that since $\ell_{ii} = 1$ there are totally n^2 unknowns for ℓ_{ij} and u_{jk} , same number as a_{ij}
 - Thus, it is possible to store ℓ_{ij} and u_{jk} into the original \mathbf{A} matrix
 - In-place LU decomposition.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ \ell_{21} & u_{22} & u_{23} \\ \ell_{31} & \ell_{32} & u_{33} \end{bmatrix}$$

- Repeat three steps in LU decomposition
 - Form u_i row by copy a_{ij} to u_{ij}
 - Form ℓ_j column by divide a_{ij} by u_{ii}
 - Update lower-right submatrix of \mathbf{A}

- Example: LU decomposition steps

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ \ell_{21} & a'_{22} & a'_{23} \\ \ell_{31} & a'_{32} & a'_{33} \end{bmatrix} \quad \begin{aligned} u_{ij} &= a_{ij} \\ \ell_{ji} &= a_{ji}/u_{ii} \\ a'_{jk} &= a_{jk} - \ell_{ji} \cdot u_{ik} \end{aligned}$$

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ \ell_{21} & a'_{22} & a'_{23} \\ \ell_{31} & a'_{32} & a'_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ \ell_{21} & u_{22} & u_{23} \\ \ell_{31} & \ell_{32} & a''_{33} \end{bmatrix} \quad \begin{aligned} u_{ij} &= a_{ij} \\ \ell_{ji} &= a_{ji}/u_{ii} \\ a''_{jk} &= a'_{jk} - \ell_{ji} \cdot u_{ik} \end{aligned}$$

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ \ell_{21} & u_{22} & u_{23} \\ \ell_{31} & \ell_{32} & a''_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ \ell_{21} & u_{22} & u_{23} \\ \ell_{31} & \ell_{32} & u_{33} \end{bmatrix} \quad u_{ij} = a_{ij}$$

LU Decomposition Algorithm – Without Pivoting

- In-place LU decomposition algorithm without pivoting

Algorithm 1.1.6. LU Decomposition

```

01 void LU(double A[n][n])
02 {
03     int i, j, k;
04
05     for (i = 0; i < n; i++) {
06         // copy a[i][j] to u[i][j] needs no action due to in-place LU
07         for (j = i + 1; j < n; j++) { // form l[j][i]
08             a[j][i] /= a[i][i];
09         }
10         for (j = i + 1; j < n; j++) { // update lower submatrix
11             for (k = i + 1; k < n; k++) {
12                 a[j][k] -= a[j][i] * a[i][k];
13             }
14         }
15     }
16 }
    
```


Forward and Backward Substitutions

- Once LU factors are found, the solution to the linear system can be obtained using forward and backward substitutions

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{LUx} = \mathbf{b}$$

- Let $\mathbf{Ux} = \mathbf{y}$, then

$$\mathbf{Ly} = \mathbf{b}$$

- Once \mathbf{y} is obtained

$$\mathbf{Ux} = \mathbf{y}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$y_1 = b_1$$

$$y_2 = b_2 - \ell_{21} \cdot y_1$$

$$y_3 = b_3 - \ell_{31} \cdot y_1 - \ell_{32} \cdot y_2$$

$$x_3 = y_3 / u_{33}$$

$$x_2 = (y_2 - u_{23} \cdot x_3) / u_{22}$$

$$x_1 = (y_1 - u_{12} \cdot x_2 - u_{13} \cdot x_3) / u_{11}$$

- This is the forward substitution

- This is the backward substitution

Forward and Backward Substitutions, II

Algorithm 1.1.7. Forward Substitution

```
01 void fwdSubst(double A[n][n], double b[n], double y[n])
02 {
03     int i, j;
04     for (i = 0; i < n; i++) y[i] = b[i]; // initialize y to b
05     for (i = 0; i < n; i++)
06         for (j = i + 1; j < n; j++)
07             y[j] -= a[j][i] * y[i];
08 }
```

Algorithm 1.1.8. Backward Substitution

```
01 void bckSubst(double A[n][n], double y[n], double x[n])
02 {
03     int i, j, k;
04     for (i = 0; i < n; i++) x[i] = y[i]; // initialize x to y
05     for (i = n - 1; i >= 0; i--) {
06         x[i] /= a[i][i];
07         for (j = i - 1; j >= 0; j--)
08             x[j] -= a[j][i] * x[i];
09     }
10 }
```

- LU decomposition
 - The outer loop is carried out n times, $0 \leq i \leq n-1$
 - For each iteration
 - Division is performed $n-i-1$ times
 - Multiplication and subtraction are performed $(n-i-1)^2$ times
 - Overall $\mathcal{O}(n^3)$
 - Division is repeated

$$\sum_{i=0}^{n-1} n-i-1 = \sum_{j=0}^{n-1} j = \frac{n(n-1)}{2}. \quad (1.1.12)$$

- Subtraction and multiplication are repeated

$$\sum_{i=0}^{n-1} (n-i-1)^2 = \sum_{j=0}^{n-1} j^2 = \frac{n^3 - n}{3} \quad (1.1.13)$$

- Forward and backward substitutions have the computation complexity of $\mathcal{O}(n^2)$
- If multiple linear systems with the same \mathbf{A} but different \mathbf{b} , then only one LU decomposition is needed and multiple forward and backward substitutions can be done for different solutions
 - Much more efficient than Gaussian elimination

LU Decomposition – Doolittle's Algorithm

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \quad (1.1.14)$$

$$\begin{array}{ll} u_{11} = a_{11} & u_{11} = a_{11} \\ u_{12} = a_{12} & u_{12} = a_{12} \\ u_{13} = a_{13} & u_{13} = a_{13} \\ \ell_{21} = a_{21}/u_{11} & \ell_{21} = a_{21}/u_{11} \\ \ell_{31} = a_{31}/u_{11} & u_{22} = a_{22} - \ell_{21}u_{12} \\ u_{22} = a_{22} - \ell_{21}u_{12} & u_{23} = a_{23} - \ell_{21}u_{13} \\ u_{23} = a_{23} - \ell_{21}u_{13} & \ell_{31} = a_{31}/u_{11} \\ \ell_{32} = (a_{32} - \ell_{31}u_{12})/u_{22} & \ell_{32} = (a_{32} - \ell_{31}u_{12})/u_{22} \\ u_{33} = a_{33} - \ell_{31}u_{13} - \ell_{32}u_{23} & u_{33} = a_{33} - \ell_{31}u_{13} - \ell_{32}u_{23} \end{array}$$

- Doolittle's algorithm is row based
- Exercise: write a C function to perform Doolittle's algorithm

LU Decomposition – Crout's Algorithm

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} \quad (1.1.15)$$

- Crout's algorithm assumes 1 on the diagonal of **U** matrix
- When performing LU decomposition, the roles of **L** and **U** matrices are reversed
 - **L**-column is formed first, instead of **U**-row
 - **U**-row is then formed by dividing $\ell_{i,i}$
 - Lower-right submatrix of **A** is then updated
- Forward substitution involves dividing $\ell_{i,i}$
- Backward substitution is simpler
- Different forms of LU decomposition have the same computational complexity of $\mathcal{O}(n^3)$

LU Factors and Matrix Inversion

- Once the LU factors are obtained, the inverse matrix can also be constructed

$$\begin{aligned} \mathbf{A}\mathbf{A}^{-1} &= \mathbf{I} \\ \mathbf{L}\mathbf{U}\mathbf{A}^{-1} &= \mathbf{I} = [\mathbf{e}_1 \quad \mathbf{e}_2 \quad \cdots \quad \mathbf{e}_n] \\ \mathbf{L}\mathbf{U} [\bar{\mathbf{a}}_1 \quad \bar{\mathbf{a}}_2 \quad \cdots \quad \bar{\mathbf{a}}_n] &= [\mathbf{e}_1 \quad \mathbf{e}_2 \quad \cdots \quad \mathbf{e}_n] \\ \mathbf{L}\mathbf{U}\bar{\mathbf{a}}_i &= \mathbf{e}_i, \quad 1 \leq i \leq n. \end{aligned}$$

- Thus, each $\bar{\mathbf{a}}_i$ can be found by forward and backward substitutions. And then the entire \mathbf{A}^{-1} is obtained.
- LU decomposition is carried once, $\mathcal{O}(n^3)$
- Forward and backward substitutions are carried out n times, $\mathcal{O}(n^3)$
- Thus, the overall matrix inversion is $\mathcal{O}(n^3)$

Computer Round-Off Errors

- Computer arithmetic usually employs finite number of bits to represent real numbers and to perform calculations
- This finite precision can cause computation errors
- For example, assuming a machine is using **4-digit** decimal number system to solve

$$\begin{bmatrix} 0.001 & 2.42 \\ 1 & 1.58 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5.2 \\ 4.57 \end{bmatrix}$$

- LU decomposition on matrix **A**

$$\begin{bmatrix} 0.001 & 2.42 \\ 1 & 1.58 \end{bmatrix}$$

$$\begin{bmatrix} 0.001 & 2.42 \\ 1000 & -2418 \end{bmatrix}$$

This is due to

$$1.58 - 1000 \times 2.42 = 1.58 - 2420 = -2418.$$

Computer Round-Off Errors, II

$$\mathbf{LU} = \begin{bmatrix} 0.001 & 2.42 \\ 1000 & -2418 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 5.2 \\ 4.57 \end{bmatrix}$$

- After forward substitution, RHS is

$$\begin{bmatrix} 5.2 \\ -5195 \end{bmatrix}$$

Because $4.57 - 1000 \times 5.2 = 4.57 - 5200 = -5195$.

- After backward substitution, RHS is

$$\begin{bmatrix} 2 \\ 2.148 \end{bmatrix}$$

Due to

$$-5195 / -2418 = 2.148$$

$$(5.2 - 2.42 \times 2.148) / 0.001 = 0.002 / 0.001 = 2$$

- Thus, we have

$$\mathbf{x} = \begin{bmatrix} 2 \\ 2.148 \end{bmatrix}$$

- Substitute back to the original system

$$\begin{bmatrix} 0.001 & 2.42 \\ 1 & 1.58 \end{bmatrix} \begin{bmatrix} 2 \\ 2.148 \end{bmatrix} = \begin{bmatrix} 5.2 \\ 5.394 \end{bmatrix}$$

- Compared to the original system

$$\begin{bmatrix} 0.001 & 2.42 \\ 1 & 1.58 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5.2 \\ 4.57 \end{bmatrix}$$

- Significant error was obtained
- Round-off error is a fundamental error in digital computer systems
- Should use as many digits as possible to reduce round-off errors
 - float: ~ 7 digits
 - double: ~ 14 digits

Round-off Errors and Pivoting

- Instead of solving

$$\begin{bmatrix} 0.001 & 2.42 \\ 1 & 1.58 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5.2 \\ 4.57 \end{bmatrix}$$

- We solve

$$\begin{bmatrix} 1 & 1.58 \\ 0.001 & 2.42 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4.57 \\ 5.2 \end{bmatrix}$$

- The LU factors can be shown to be

$$\begin{bmatrix} 1 & 1.58 \\ 0.001 & 2.418 \end{bmatrix}$$

- After forward substitution, we have

$$\begin{bmatrix} 4.57 \\ 5.195 \end{bmatrix}$$

- And after backward substitution

$$\begin{bmatrix} 1.176 \\ 2.148 \end{bmatrix}$$

Round-off Errors and Pivoting, II

- Substitute back to the original system

$$\begin{bmatrix} 0.001 & 2.42 \\ 1 & 1.58 \end{bmatrix} \begin{bmatrix} 1.176 \\ 2.148 \end{bmatrix} = \begin{bmatrix} 5.199 \\ 4.57 \end{bmatrix}$$

- We get a good solution even with 4-digit computer
- Matrix ordering can affect solution accuracy
- Selecting the right diagonal element (together with the corresponding matrix ordering) is the strategy of **pivoting**
- It can be shown that selecting element with the largest absolute value as the pivot (diagonal element) can improve the accuracy, and stability, of the linear system solution.
- Diagonal dominant** matrices can be solved accurately

$$|a_{i,i}| \geq \sum_{j \neq i} |a_{i,j}|, \quad 1 \leq i \leq n \quad (1.1.16)$$

- Most finite difference matrices have this property
- Circuit matrices may not have this property

Matrix Pivoting

- Before pivoting

$$\begin{bmatrix} \dots & \dots & \dots & \dots & \dots \\ \dots & a_{i,i} & a_{i,i+1} & a_{i,i+2} & \dots \\ \dots & a_{i+1,i} & a_{i+1,i+1} & a_{i+1,i+2} & \dots \\ \dots & a_{i+2,i} & a_{i+2,i+1} & a_{i+2,i+2} & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \dots \\ x_i \\ x_{i+1} \\ x_{i+2} \\ \dots \end{bmatrix} = \begin{bmatrix} \dots \\ b_i \\ b_{i+1} \\ b_{i+2} \\ \dots \end{bmatrix}$$

- Row pivoting

- Swapping with the selected row – diagonal element changed
- RHS is also swapped

$$\begin{bmatrix} \dots & \dots & \dots & \dots & \dots \\ \dots & a_{i+2,i} & a_{i+2,i+1} & a_{i+2,i+2} & \dots \\ \dots & a_{i+1,i} & a_{i+1,i+1} & a_{i+1,i+2} & \dots \\ \dots & a_{i,i} & a_{i,i+1} & a_{i,i+2} & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \dots \\ x_i \\ x_{i+1} \\ x_{i+2} \\ \dots \end{bmatrix} = \begin{bmatrix} \dots \\ b_{i+2} \\ b_{i+1} \\ b_i \\ \dots \end{bmatrix} \quad (1.1.17)$$

- Before pivoting

$$\begin{bmatrix} \dots & \dots & \dots & \dots & \dots \\ \dots & a_{i,i} & a_{i,i+1} & a_{i,i+2} & \dots \\ \dots & a_{i+1,i} & a_{i+1,i+1} & a_{i+1,i+2} & \dots \\ \dots & a_{i+2,i} & a_{i+2,i+1} & a_{i+2,i+2} & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \dots \\ x_i \\ x_{i+1} \\ x_{i+2} \\ \dots \end{bmatrix} = \begin{bmatrix} \dots \\ b_i \\ b_{i+1} \\ b_{i+2} \\ \dots \end{bmatrix}$$

- Column pivoting

- Swapping with the selected column – diagonal element changed
- Unknown variables swapped
- RHS unchanged

$$\begin{bmatrix} \dots & \dots & \dots & \dots & \dots \\ \dots & a_{i,i+2} & a_{i,i+1} & a_{i,i} & \dots \\ \dots & a_{i+1,i+2} & a_{i+1,i+1} & a_{i+1,i} & \dots \\ \dots & a_{i+2,i+2} & a_{i+2,i+1} & a_{i+2,i} & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \dots \\ x_{i+2} \\ x_{i+1} \\ x_i \\ \dots \end{bmatrix} = \begin{bmatrix} \dots \\ b_i \\ b_{i+1} \\ b_{i+2} \\ \dots \end{bmatrix} \quad (1.1.18)$$

LU Decomposition with Partial Pivoting

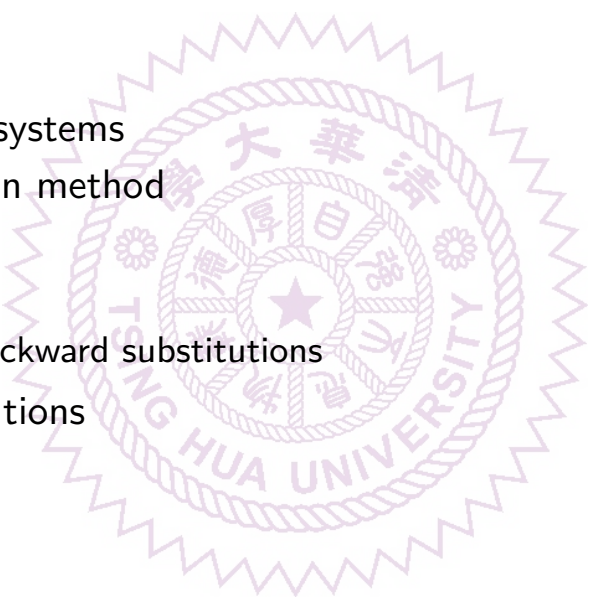
- In LU decomposition, we need to divide the column by the diagonal element.
- If $a_{ii} = 0$, for any i , then LU decomposition fails even the original matrix is nonsingular.
- Pivoting can solve this problem
 - Pivoting can also improve the stability of the linear system solution
 - If $a_{ii} = 0$, select j such that $|a_{ji}|$ is the maximum
 - Swap rows i and j
 - Then carry out the LU decomposition process.
 - Let \mathbf{P} be an identity matrix initially
 - When swapping of rows i and j of matrix \mathbf{A} is performed, the same operation is also performed on \mathbf{P} ,
 - Then effectively, the LU decomposition is carried out on \mathbf{PA} , i.e.,

$$\mathbf{LU} = \mathbf{PA}. \quad (1.1.19)$$

- To get the correct solution, the vector \mathbf{b} needs to premultiply matrix \mathbf{P} since

$$\mathbf{LUx} = \mathbf{PAx} = \mathbf{Pb} \quad (1.1.20)$$

- Note that with partial pivoting, the number of divisions and multiplications do not change, hence the computational complexity remains the same.

- Solutions of linear systems
 - Gaussian elimination method
 - Pivoting
 - LU decomposition
 - Forward and backward substitutions
 - Errors in linear solutions
 - Matrix pivoting
- 
- A large, faint, circular watermark seal of Tsinghua University is centered in the background. It features a star in the center, surrounded by Chinese characters and the English text 'TSINGHUA UNIVERSITY'.