Lecture 9.
Linear Solver:
LU Solver and Sparse Matrix

Guoyong Shi, PhD
shiguoyong@ic.sjtu.edu.cn
School of Microelectronics
Shanghai Jiao Tong University
Fall 2010

Outline

Part 1:

- **Gaussian Elimination**
- **LU Factorization**
- **Pivoting**
- **Doolittle method and Crout's Method**
- **Summary**

Part 2: Sparse Matrix

Motivation

- **Either in Sparse Tableau Analysis (STA) or in Modified Nodal Analysis (MNA), we have to solve linear system of equations:** $\mathbf{Ax = b}$

$$\begin{pmatrix} A & 0 & 0 \\ 0 & I & -A^T \\ K_i & K_v & 0 \end{pmatrix} \begin{pmatrix} i \\ v \\ e \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ S \end{pmatrix}$$

$$\begin{bmatrix} \frac{1}{R_1} + G_2 + \frac{1}{R_3} & -G_2 - \frac{1}{R_3} \\ -\frac{1}{R_3} & \frac{1}{R_4} + \frac{1}{R_3} \end{bmatrix} \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} 0 \\ I_{S5} \end{pmatrix}$$

Motivation

- **Even in nonlinear circuit analysis, after "linearization", again one has to solve a system of linear equations: $Ax = b$.**
- **Many other engineering problems require solving a system of linear equations.**

- **Typically, matrix size is of 1000s to millions.**
- **This needs to be solved 1000 to million times for one simulation cycle.**
- **That's why we'd like to have very efficient linear solvers!**

Problem Description

Problem:

Solve $Ax = b$

A: nxn (real, non-singular), x: nx1, b: nx1

Methods:

- **Direct Methods (this lecture)**

  Gaussian Elimination, LU Decomposition, Crout

- **Indirect, Iterative Methods (another lecture)**

  Gauss-Jacobi, Gauss-Seidel, Successive Over Relaxation (SOR), Krylov

Gaussian Elimination -- Example

$$\begin{cases} 2x + y = 5 \\ x + 2y = 4 \end{cases}$$

$$-\frac{1}{2} \left(\begin{array}{cc|c} 2 & 1 & 5 \\ 1 & 2 & 4 \end{array}\right) \rightarrow \left(\begin{array}{cc|c} 2 & 1 & 5 \\ 0 & \frac{3}{2} & \frac{3}{2} \end{array}\right) \rightarrow \begin{cases} x = 2 \\ y = 1 \end{cases}$$

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 0 & \frac{3}{2} \end{pmatrix}$$

LU factorization

Use of LU Factorization

L U

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 0 & \frac{3}{2} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \end{pmatrix}$$

Define

$$\begin{pmatrix} u \\ v \end{pmatrix} := \begin{pmatrix} 2 & 1 \\ 0 & \frac{3}{2} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$$

Solving the L-system:

$$\begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix}\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \end{pmatrix}$$

L

Solve

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 5 \\ \frac{3}{2} \end{pmatrix}$$

Triangle systems are easy to solve (by back-substitution.)

Use of LU Factorization

$$\begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \end{pmatrix} \Rightarrow \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 5 \\ \frac{3}{2} \end{pmatrix}$$

U

$$\begin{pmatrix} 2 & 1 \\ 0 & \frac{3}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix}$$

U

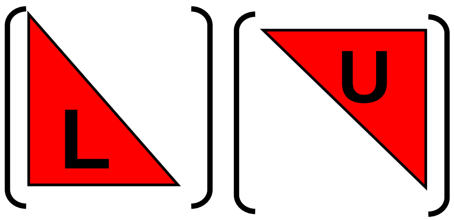
$$\begin{pmatrix} 2 & 1 \\ 0 & \frac{3}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ \frac{3}{2} \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Solving the U-system:

LU Factorization

$$A = LU =$$



The task of L & U factorization is to find the elements in matrices L and U.

$$Ax = L(Ux) = Ly = b$$

1. **Let $y = Ux$.**
2. **Solve y from $Ly = b$**
3. **Solve x from $Ux = y$**

Advantages of LU Factorization

- **When solving $Ax = b$ for multiple b, but the same A, then we only LU-factorize A only once.**
- **In circuit simulation, entries of A may change, but structure of A does not alter.**
  - **This factor can used to speed up repeated LU-factorization.**
  - **Implemented as symbolic factorization in the "sparse1.3" solver in Spice 3f4.**

Gaussian Elimination

- **Gaussian elimination is a process of row transformation**

$$Ax = b$$

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array}\right]$$

Eliminate the lower triangular part

Gaussian Elimination

$a_{11} \neq 0$

$$-\frac{a_{i1}}{a_{11}} \qquad \left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array}\right]$$

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ 0 & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array}\right]$$

Entries updated

Eliminating 1st Column

- Column elimination is equiv to row transformation

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -\frac{a_{21}}{a_{11}} & 1 & 0 & \cdots & 0 \\ -\frac{a_{31}}{a_{11}} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{a_{11}} & 0 & 0 & \cdots & 1 \end{bmatrix} \times \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{matrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{matrix}$$

$$L_1^{-1} A = A^{(2)}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix} \begin{matrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \\ \vdots \\ b_n^{(2)} \end{matrix}$$

Eliminating 2nd Column

$$
\begin{bmatrix}
1 & 0 & 0 & \cdots & 0 \\
0 & 1 & 0 & \cdots & 0 \\
 & -\frac{a_{32}^{(2)}}{a_{22}^{(2)}} & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & -\frac{a_{n2}^{(2)}}{a_{22}^{(2)}} & 0 & \cdots & 1
\end{bmatrix}
\times
\begin{bmatrix}
a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\
0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\
0 & a_{32}^{(2)} & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & a_{n2}^{(2)} & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)}
\end{bmatrix}
$$

$$a_{22}^{(2)} \neq 0$$

$$L_2^{-1} \; A^{(2)} = A^{(3)}$$

$$
\begin{bmatrix}
a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\
0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\
0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & a_{n3}^{(3)} & \cdots & a_{nn}^{(3)}
\end{bmatrix}
\quad
\begin{matrix}
b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_n^{(3)}
\end{matrix}
$$

Continue on Elimination

- **Suppose all diagonals are nonzero**

$$L_n^{-1} L_{n-1}^{-1} \cdots L_2^{-1} L_1^{-1}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{matrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{matrix}$$

Upper triangular

$$A^{(n)} x = b^{(n)}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix} \begin{matrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_n^{(n)} \end{matrix}$$

$A^{(n)}$

Triangular System

Gaussian elimination ends up with the following upper triangular system of equations

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_n^{(n)} \end{pmatrix}$$

Solve this system from bottom up: $\mathbf{x_n, x_{n-1}, \ldots, x_1}$

LU Factorization

- **Gaussian elimination leads to LU factorization**

$$(L_n^{-1} L_{n-1}^{-1} \cdots L_2^{-1} L_1^{-1}) \, A = U$$

$$A = (L_1 L_2 \cdots L_{n-1} L_n) \, U = LU$$

$$L = (L_1 L_2 \cdots L_{n-1} L_n)$$

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \frac{a_{21}}{a_{11}} & 1 & 0 & \cdots & 0 \\ \frac{a_{31}}{a_{11}} & \frac{a_{32}^{(2)}}{a_{22}^{(2)}} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{a_{n1}}{a_{11}} & \frac{a_{n2}^{(2)}}{a_{22}^{(2)}} & * & \cdots & 1 \end{bmatrix} \quad U = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix}$$

Complexity of LU

$a_{11} \neq 0$

$$
\begin{matrix}
 & a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\
-\frac{a_{21}}{a_{11}} & a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\
-\frac{a_{31}}{a_{11}} & a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\
 & \vdots & \vdots & \vdots & \ddots & \vdots \\
-\frac{a_{n1}}{a_{11}} & a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn}
\end{matrix}
$$

# of mul / div = (n-1)*n $\approx$ n^2

n >> 1

$$\sum_{i=1}^{n} i^2 = \frac{1}{6} n(n+1)(2n+1) \sim O(n^3)$$

Cost of Back-Substitution

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_n^{(n)} \end{pmatrix}$$

$$x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}} \qquad x_{n-1} = \frac{b_{n-1}^{(n-1)} - a_{n-1,n}^{(n-1)} x_n}{a_{n-1,n-1}^{(n-1)}}$$

Total # of mul / div = $\sum_{i=1}^{n} i = \frac{1}{2} n(n+1) \sim O(n^2)$

Zero Diagonal

Example 1: After two steps of Gaussian elimination:

$$
\begin{bmatrix}
\times & \times & 0 & 0 & 0 & 0 \\
0 & \times & 0 & 0 & \times & 0 \\
0 & 0 & \frac{1}{R} & -\frac{1}{R} & 1 & 0 \\
0 & 0 & -\frac{1}{R} & \frac{1}{R} & 0 & 1 \\
0 & 0 & \times & 0 & \times & 0 \\
0 & 0 & 0 & \times & \times & 0
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
\times & \times & 0 & 0 & 0 & 0 \\
0 & \times & 0 & 0 & \times & 0 \\
0 & 0 & \frac{1}{R} & -\frac{1}{R} & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & \times & \times & 0 \\
0 & 0 & 0 & \times & \times & 0
\end{bmatrix}
$$

Gaussian elimination cannot continue

Pivoting

Solution 1:

Interchange rows to bring a non-zero element into position (k, k):

$$\begin{bmatrix} 0 & 1 & 1 \\ \times & \times & 0 \\ \times & \times & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} x & x & 0 \\ 0 & 1 & 1 \\ x & x & 0 \end{bmatrix}$$

Solution 2: How about column exchange? Yes

Then the unknowns are re-ordered as well.

$$\begin{bmatrix} 0 & 1 & 1 \\ \times & \times & 0 \\ \times & \times & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 1 \\ x & x & 0 \\ x & x & 0 \end{bmatrix}$$

In general both rows and columns can be exchanged!

Small Diagonal

Example 2:

$$\begin{bmatrix} 1.25\times 10^{-4} & 1.25 \\ 12.5 & 12.5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6.25 \\ 75 \end{bmatrix}$$

Assume finite arithmetic: 3-digit floating point, we have

-10^5

pivoting

$$\begin{bmatrix} 1.25\times 10^{-4} & 1.25 \\ 12.5 & 12.5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6.25 \\ 75 \end{bmatrix}$$

$$\begin{bmatrix} 1.25\times 10^{-4} & 1.25 \\ 0 & -1.25\times 10^{5} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6.25 \\ -6.25\times 10^{5} \end{bmatrix}$$

12.5 rounded off

$$\begin{cases} x_2 = 5 \\ (1.25\times 10^{-4})x_1 + (1.25)x_2 = 6.25 \end{cases} \Rightarrow x_1 = 0$$

Unfortunately, (0, 5) is not the solution. Considering the 2nd equation: 12.5 * 0 + 12.5 * 5 = 62.5 ≠ 75.

Accuracy Depends on Pivoting

$$-10^5 \quad \text{pivoting} \quad \begin{bmatrix} 1.25 \times 10^{-4} & 1.25 \\ 12.5 & 12.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6.25 \\ 75 \end{bmatrix}$$

Reason:

a_{11} (the pivot) is too small relative to the other numbers!

Solution: Don't choose small element to do elimination. Pick a large element by row / column interchanges.

Correct solution to 5 digit accuracy is

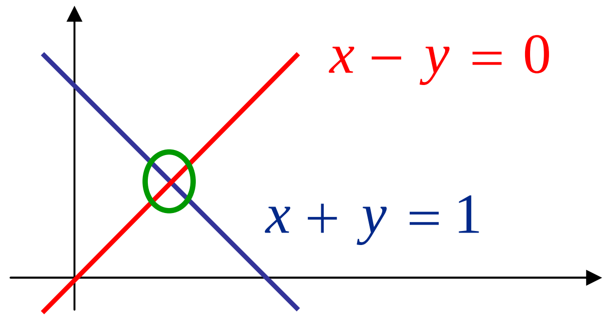
$x_1 = 1.0001$

$x_2 = 5.0000$

What causes accuracy problem?

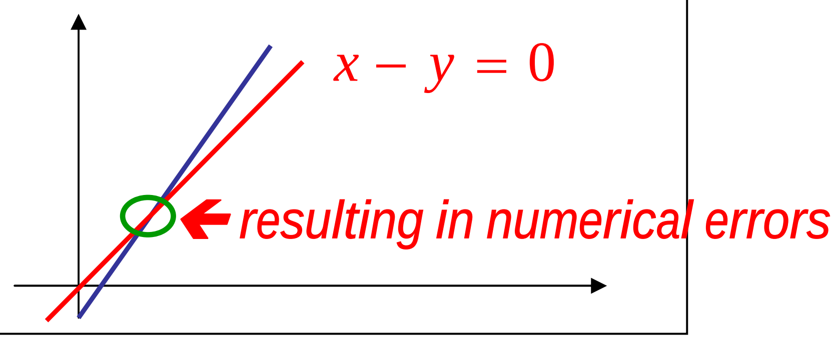
- <u>Ill Conditioning</u>: **The A matrix close to singular**
- <u>Round-off error</u>: **Relative magnitude too big**

$$x + y = 1$$
$$x - y = 0$$



$$x - y = 0$$
$$x - 1.01y = 0.01$$

ill-conditioned



Pivoting Strategies

1. **Partial Pivoting**

2. **Complete Pivoting**

3. **Threshold Pivoting**

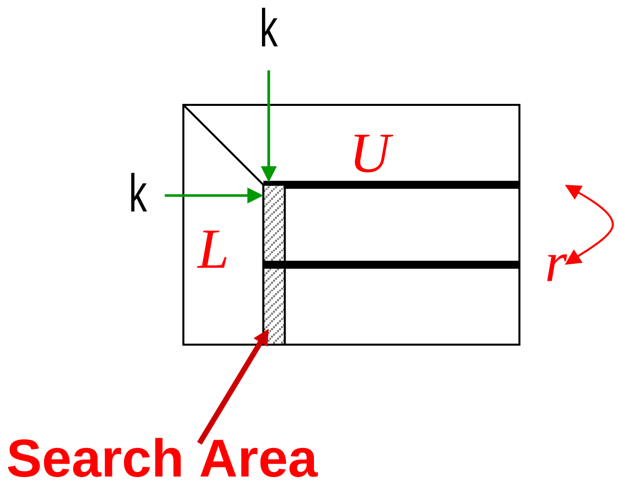
Pivoting Strategy 1

1. Partial Pivoting: (Row interchange only)

Choose *r* as the smallest integer such that:

$$\left| a_{rk}^{(k)} \right| = \max_{j=k,\ldots,n} \left| a_{jk}^{(k)} \right|$$

Rows k to n



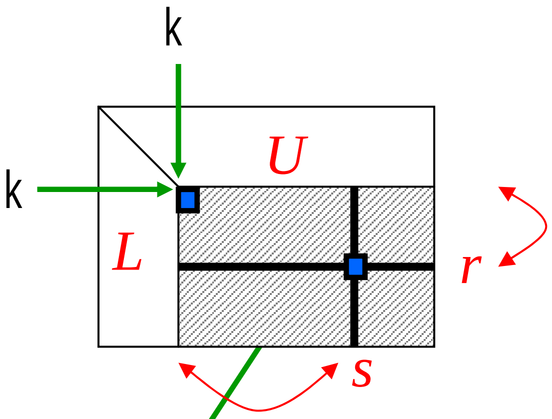
Pivoting Strategy 2

2. Complete Pivoting: (Row and column interchange)

Choose r and s as the smallest integer such that:

$$\left|a_{rs}^{(k)}\right| = \max_{\substack{i=k,\dots,n \\ j=k,\dots,n}} \left|a_{ij}^{(k)}\right|$$

rows k to n; cols k to n



Search Area

Pivoting Strategy 3

3. Threshold Pivoting:

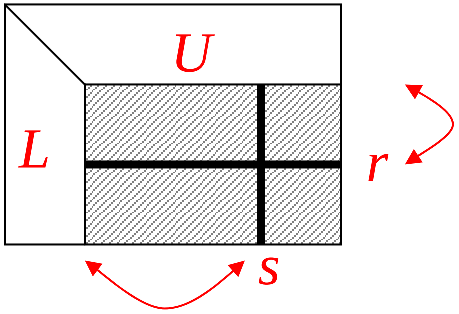
a. Apply partial pivoting only if $\left|a_{kk}^{(k)}\right| < \varepsilon_p \left|a_{rk}^{(k)}\right|$

b. Apply complete pivoting only if $\left|a_{kk}^{(k)}\right| < \varepsilon_p \left|a_{rs}^{(k)}\right|$

user specified

$$\left|a_{rk}^{(k)}\right| = \max_{j=k,\ldots,n} \left|a_{jk}^{(k)}\right|$$

$$\left|a_{rs}^{(k)}\right| = \max_{\substack{i=k,\ldots,n \\ j=k,\ldots,n}} \left|a_{ij}^{(k)}\right|$$

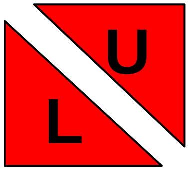


Implemented in Spice 3f4

Variants of LU Factorization

- **Doolittle Method**
- **Crout Method**
- **Motivated by directly filling in L/U elements in the storage space of the original matrix "A".**

$$A = LU =$$



$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

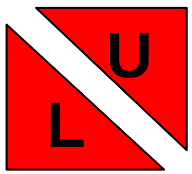
Reuse the storage

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}$$

Variants of LU Factorization

Hence we need a sequential method to process the rows and columns of A in certain order – processed rows / columns are not used in the later processing.

$$A = LU =$$



Reuse the storage

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix} \quad \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Doolittle Method – 1

Keep this row

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

First solve the 1st row of U, i.e., *U(1, :)*

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \end{pmatrix}$$

Doolittle Method – 2

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Then solve the 1st column of L, i.e., *L(2:n, 1)*

$$\begin{bmatrix} a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{bmatrix} = \begin{bmatrix} \ell_{21} \\ \ell_{31} \\ \vdots \\ \ell_{n1} \end{bmatrix} u_{11} \qquad u_{11} = a_{11}$$

Doolittle Method – 3

(1)

(2)

(3)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Solve the 2nd row of U, i.e., *U(2, 2:n)*

$$\ell_{21}\begin{pmatrix} u_{12} & u_{13} & \cdots & u_{1n} \end{pmatrix} + \begin{pmatrix} u_{22} & u_{23} & \cdots & u_{2n} \end{pmatrix}$$

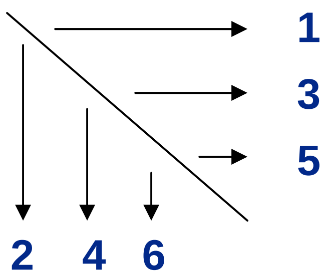
$$= \begin{pmatrix} a_{22} & a_{23} & \cdots & a_{2n} \end{pmatrix}$$

Doolittle Method – 4

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

The computation order of the Doolittle Method:

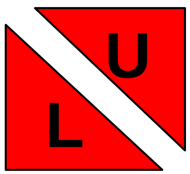
$$L \backslash U =$$



Crout Method

- **Similar to the Doolittle Method, but starts from the <u>1st column</u> (Doolittle starts from the <u>1st row</u>.)**

$$A = LU =$$

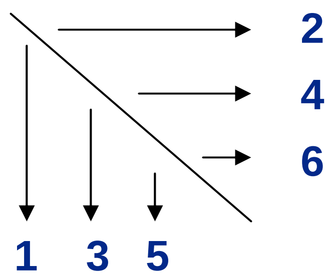


$$\begin{bmatrix} \ell_{11} & 0 & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & \ell_{nn} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & 1 & u_{23} & \cdots & u_{2n} \\ 0 & 0 & 1 & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

The computation order of the Crout Method:

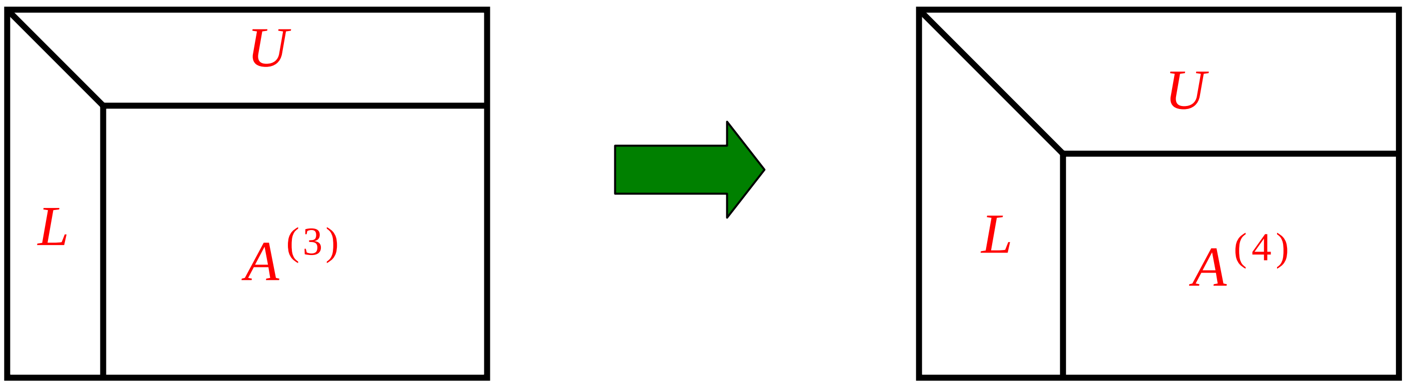
The diagonals of U are normalized !

$$L \setminus U =$$



Storage of LU Factorization

Using only one 2-dimensional array !



- **In sparse matrix implementation, this type of storage requires increasing memory space because of fill-ins during the factorization.**

Summary

- **LU factorization has been used in virtually all circuit simulators**
  - **Good for multiple RHS and sensitivity calculation**
- **Pivoting is required to handle zero diagonals and to improve numerical accuracy**
  - **Partial pivoting (row exchange): tradeoff between accuracy and efficiency**
  - **Matrix condition number is used to analyze the effect of round-off errors and numerical stability**

Part 2.
Programming Techniques for Sparse Matrices

Outline

- **Why Sparse Matrix Techniques?**
- **Sparse Matrix Data Structure**
- **Markowitz Pivoting**
- **Diagonal Pivoting for MNA Matrices**
- **Modified Markowitz pivoting**
- **How to Handle Sparse RHS**
- **Summary**

Why Sparse Matrix?

Motivation:

- **$n = 10^3$ equations**
- **Complexity of Gaussian elimination ~ $O(n^3)$**
- **$n = 10^3$ ➔ ~ 10^9 flops operations**
  - **➔ (1 GHz computer) 10 sec**
  - **➔ storage 10^6 words**

Exploiting Sparsity

- **MNA ➔ 3 nonzeros / row**
- **Can reach complexity for Gaussian elimination**
  - **~ $O(n^{1.1})$ – $O(n^{1.5})$ (Empirical complexity)**

Sparse Matrix Programming

- **Use linked-list data structure**
  - **to avoid storing zeros**
  - **used to be hard before 1980s: in Fortran!**
- **Avoid trivial operations 0x = 0, 0+x = x**
- **Two kinds of zero**
  - **Structural zeros – always 0 independent of numerical operations**
  - **Numerical zeros – resulting from computation**
- **Avoid losing sparsity (very important!)**
  - **sparsity changes with pivoting**

Non-zero Fill-ins

- **Gaussian elimination causes nonzero fill-ins**

-1

3	4	5	0	6	0	0	1	2	8
3	0	0	0	-1	0	0	3	-2	4
5	0	-3	2	6	1	2	7	0	0
0	4	8	0	10	0	0	2	2	3
1	5	4	0	0	0	0	1	0	6

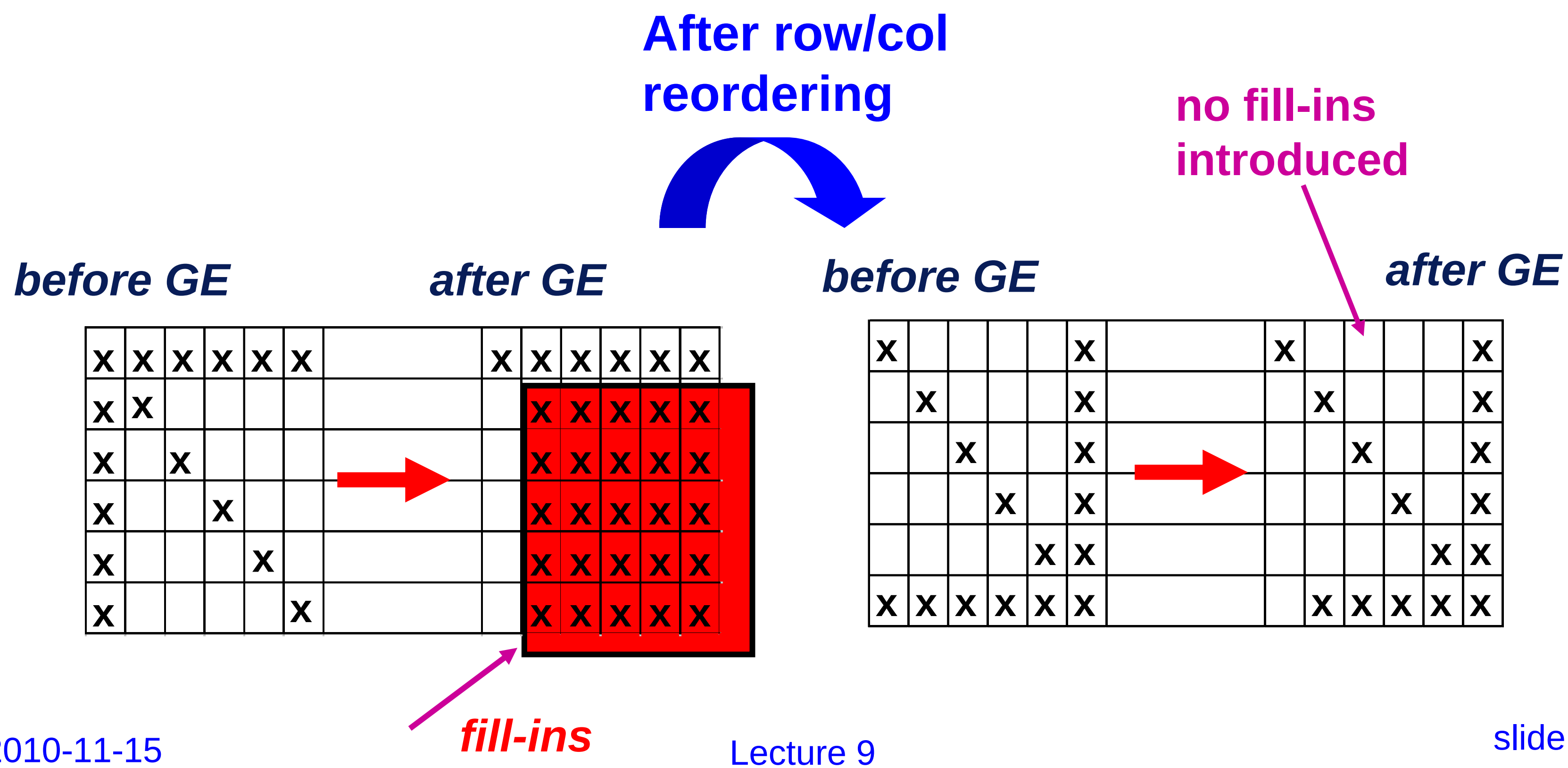
⇒

3	4	5	0	6	0	0	1	2	8
0	-4	-5	0	-7	0	0	2	-4	-4
5	0	-3	2	6	1	2	7	0	0
0	4	8	0	10	0	0	2	2	3
1	5	4	0	0	0	0	1	0	6

fill-ins

How to Maintain Sparsity

- **One should choose appropriate <u>pivoting</u> (during Gaussian Elimination, G.E.) to avoid large increment of fill-ins.**



Markowitz Criterion

- **Markowitz criterion**
  - ***kth* pivot;**
  - **$A^{(k)}$ is the reduced matrix**
  - **NZ = nonzero**
  - **The num of nonzeros in a row (column) is also called the *row (column) degree*.**
  - **The column degrees can be used for column ordering.**

If chosen for pivoting

$A^{(k)}$ →

x	x	x		x	4
	x		x	x	3
x		x			2
	x			x	2
x			x		2
3	3	2	2	3	$c_i \setminus r_i$

← # NZ this row (row degrees)

NZ this col (column degrees)

Markowitz Product

- If Gaussian Elimination to pivot on (i, j)
- Markowitz product = $(r_i - 1)(c_j - 1)$

  = maximum possible number of fill-ins if pivoting at (i, j)

- Recommendations: (implemented in Sparse1.3)
  - Best with largest magnitude of pivot element and smallest Markowitz product
  - Try threshold test after choosing smallest Markowitz product (M.P.)
  - Break ties (if equal M.P.) by choosing element with largest magnitude

Sparse Matrix Data Structure

Example Matrix

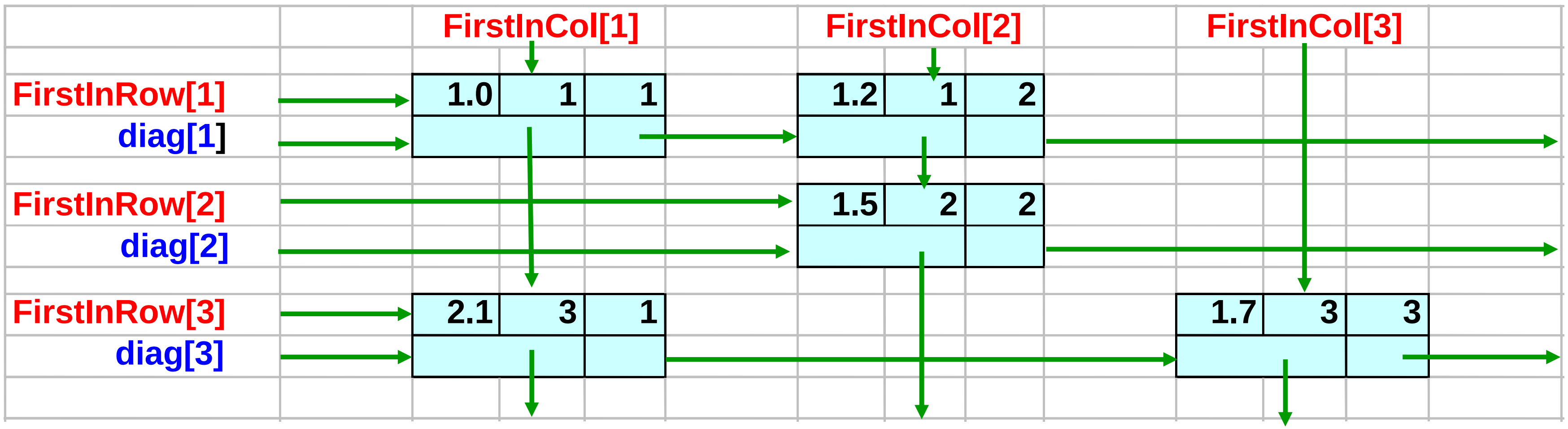
r \ c	1	2	3
1	1	1.2	0
2	0	1.5	0
3	2.1	0	1.7

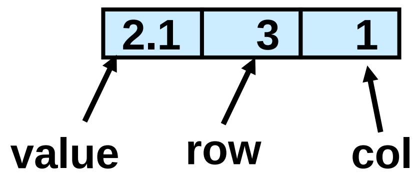
Matrix Element structure

```
struct elem{
        real          value;
        int           row;
        int           col;
        struct elem     *next_in_row;
        struct elem     *next_in_col;
} Element;
```

Data Structure in Sparse 1.3

- **Sparse 1.3 – Written by Ken Kundert, 1985~1988, then PhD student at Berkeley, later with Cadence Design Systems, Inc.**

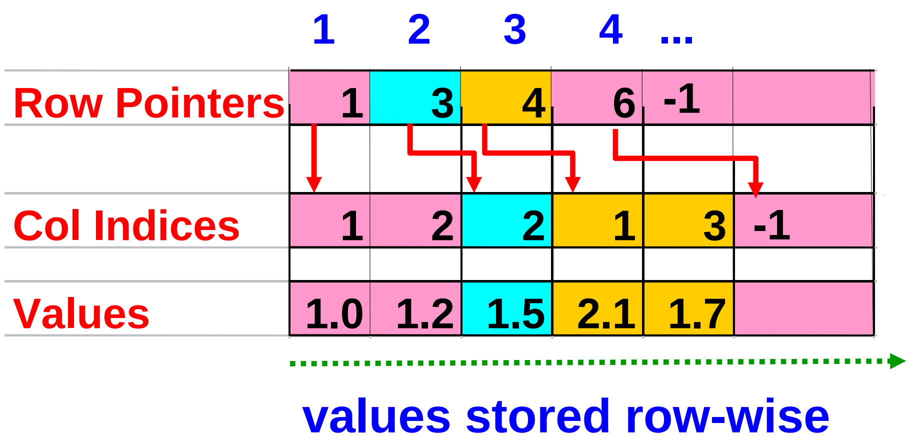




ASTAP Data Structure

- **ASTAP is an IBM simulator using STA (Sparse Tableau Analysis).**

r \ c	1	2	3
1	1	1.2	0
2	0	1.5	0
3	2.1	0	1.7



✓Row Pointers point to the beginning of Col Indices.

✓Nonzeros in the same row are indexed by their col indexes continuously.

✓Used by many *iterative* sparse solvers

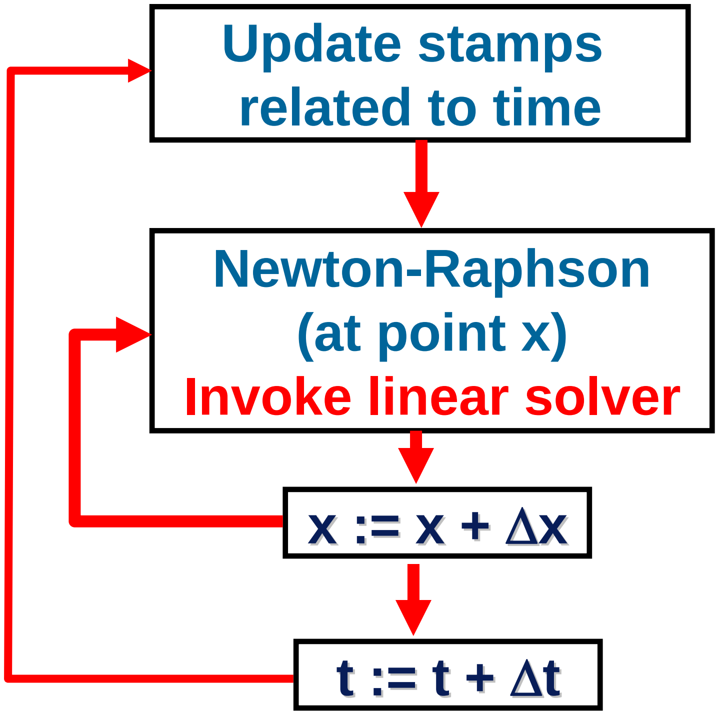
Key Loops in a SPICE Program

$$C\frac{dx}{dt} = f(x,t)$$

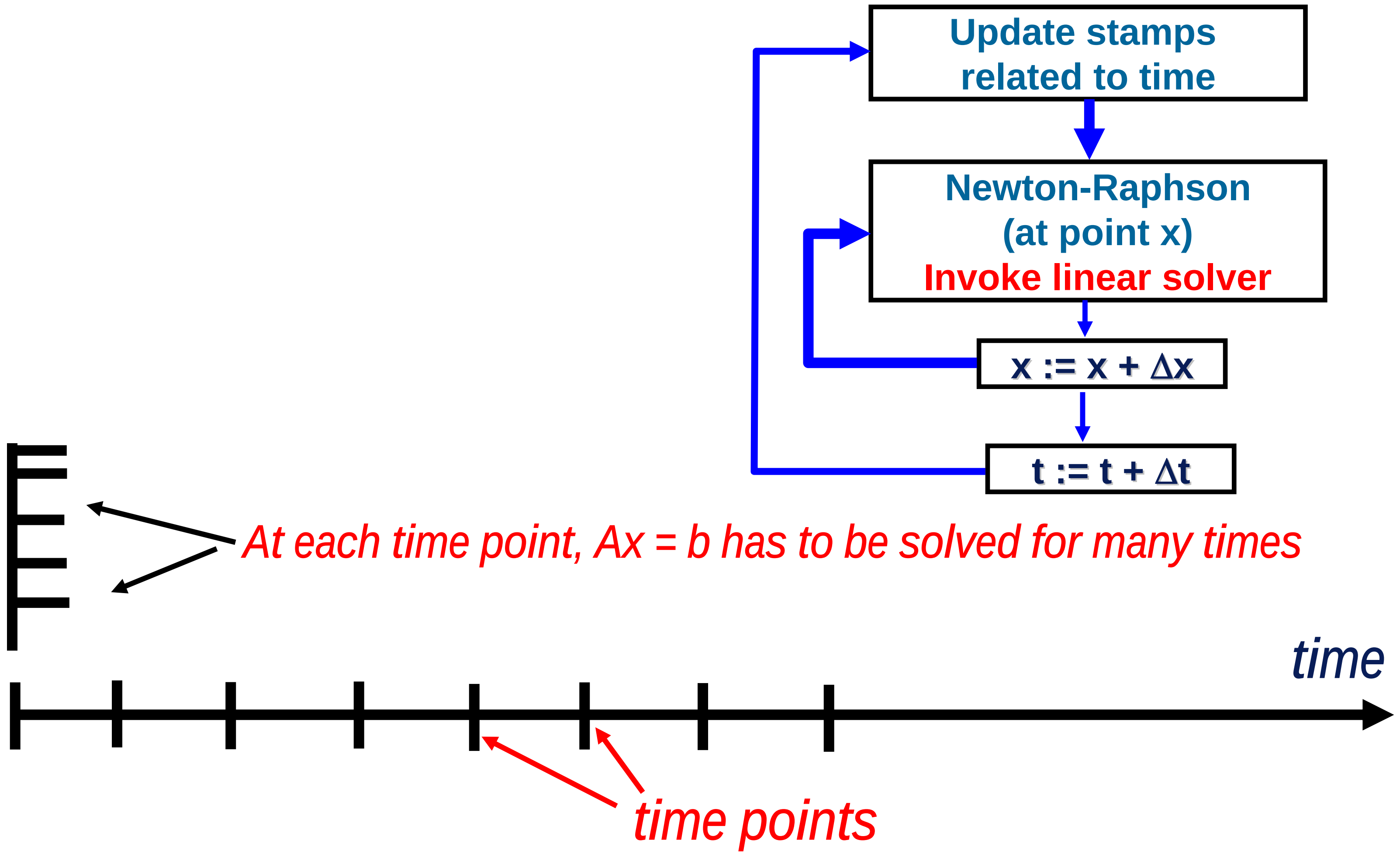
$$Cx_{n+1} = Cx_n + h \cdot f(x_{n+1}, t_{n+1}) + \cdots$$

$$Cx_{n+1}^{(k)} = \left[\frac{\partial f}{\partial x}\right]\left[x_{n+1}^{(k)} - x_{n+1}^{(k-1)}\right] + \cdots$$

$$A = \frac{\partial f\left(x_{n+1}^{(k-1)}, t_{n+1}\right)}{\partial x}$$



Linear Solves in Simulation



Structure of Matrix Stamps

- **In circuit simulation, matrix being solved repeatedly is of the <u>same structur</u>;**
- **only <u>some entries vary</u> at different frequency or time points.**

Typical matrix structure

	T		C			
		X			T	
A =		T	C	X		X
				T		
	C	X			C	
			C			X

C = Constant

T = Time varying

X = Nonlinear (varying even at the same time point)

Strategies for Efficiency

- **Utilizing the structural information can greatly improve the solving efficiency.**

- **Strategies:**
  - **Weighted Markowitz Product**
  - **Reuse the LU factorization**
  - **Iterative solver (by conditioning)**
  - **...**

A Good (Sparse) LU Solver

Properties of a good LU solver:

- Should have a **good** column ordering algorithm.
- With a good column ordering, **partial (row) pivoting** would be enough !
- Should have an ordering/elimination **separated design**:
  - **i.e., ordering** is separated from **elimination**.
  - **SuperLU** does this,
  - but **Sparse1.3** doesn't.

Optimal Ordering is NP-hard

- **The ordering has a significant impact on the memory and computational requirements for the latter stages.**
- **However, finding the optimal ordering for *A* (in the sense of minimizing fill-in) has been proven to be NP-complete.**
- **Heuristics must be used for all but simple (or specially structured) cases.**

M.R. Garey and D.S. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness* W.H. Freeman, New York, 1979.

Column Ordering

Why Important ?

- A good column ordering greatly reduces the **number of fill-ins**, resulting in a vast speedup.
- However, searching a pivot with minimum degree at each step (in Sparse 1.3) is **not efficient**.
- Best to get a good ordering **before** elimination (e.g. SuperLU), but not easy!

Available Ordering Algorithms

SuperLU uses the following algorithms:

- Multiple Minimum Degree (**MMD**) applied to the structure of (A^TA).
  - **Mostly good**
- Multiple Minimum Degree (**MMD**) applied to the structure of (A^T+A).
  - **Mostly good**
- Column Approximate Minimum Degree (**COLAMD**).
  - **Mostly not good!**

Summary

- Exploiting sparsity reduces CPU time and memory
- Markowitz algorithm reflects a good tradeoff between overhead (computation of MP) and savings (less fill-ins)
- Use weighted Markowitz to account for different types of element stamps in nonlinear dynamic circuit simulation
- Consider sparse RHS and selective unknowns for speedup

No-turn-in Exercise

- Spice3f4 contains a solver called Sparse 1.3 (in src/lib/sparse)
- This is a independent solver that can be used outside Spice3f4.
- Download the sparse package from the course web page (sparse.tar.gz) (or ask TA).
- Find the test program called "spTest.c".
- Modify this program if necessary so that you can run the solver.
- Create some test matrices to test the sparse solver.
- Compare the solved results to that by MATLAB.

Software

- Sparse1.3 is in C and was programmed by Dr. Ken Kundert (fellow of Cadence; architect of Spectre).
- Source code is available from
  http://www.netlib.org/sparse/
- SparseLib++ is in C++ and comes from NIST. The authors are J. Dongarra, A. Loumsdaine, R. Pozo, K. Remington.
- See``A Sparse Matrix Library in C++ for High Performance Architectures", Proc. of the Second Object Oriented Numerics Conference, pp. 214-218, 1994.
- The paper and the C++ source code are available from http://math.nist.gov/sparselib%2b%2b/

References

1. G. Dahlquist and A. Bjorck, *Numerical Methods* (translated by N. Anderson), Prentice Hall, Inc. Englewood Cliffs, New Jersey, 1974.
2. W. J. McCalla, *Fundamentals of Computer-Aided Circuit Simulation*, Kluwer Academic Publishers.
   1. Chapter 3, “Sparse Matrix Methods”
3. Albert Ruehli (Ed.), “Circuit Analysis, Simulation and Design”, North-Holland, 1986.
   1. K. Kundert, “Sparse Matrix Techniques”
4. J. Dongarra, A. Loumsdaine, R. Pozo, K. Remington, ``A Sparse Matrix Library in C++ for High Performance Architectures,” Proc. of the Second Object Oriented Numerics Conference, pp. 214-218, 1994.