ECE 552
Numerical Circuit Analysis

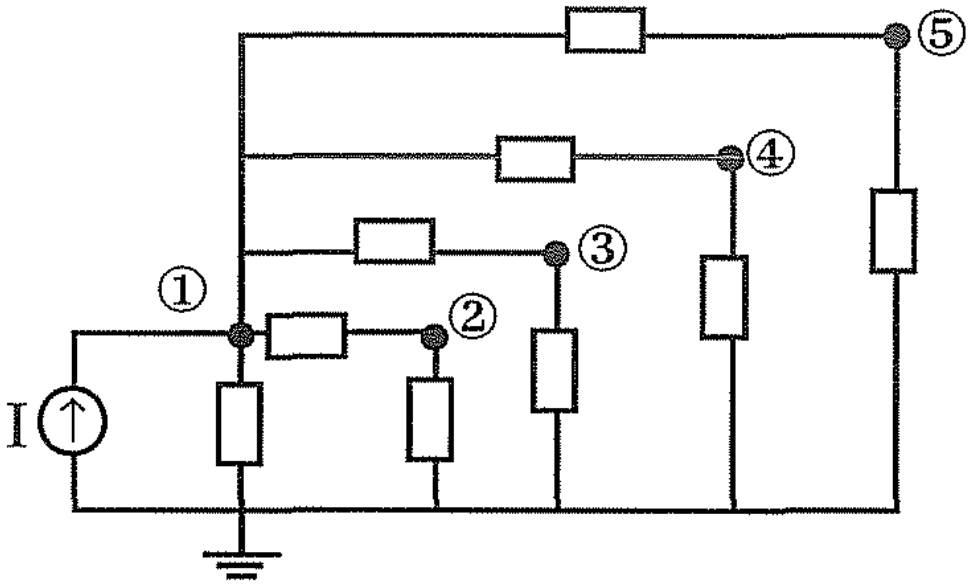
Chapter Four

SPARSE MATRIX SOLUTION TECHNIQUES

I. Hajj

Sparse Matrix Solution Techniques

Example



Sparse Matrix Solution Techniques

Case 1 (nodal equations)

$$\begin{bmatrix} * & x & x & x & x \\ x & * & F & F & F \\ x & F & * & F & F \\ x & F & F & * & F \\ x & F & F & F & * \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{bmatrix} = \begin{bmatrix} I \\ F \\ F \\ F \\ F \end{bmatrix}$$

F = Fills

- Number of operations required for LU factorization: 20+12+6+2 = 40
- Number of operations required for Forward and Backward Substitution: 25
- Total number of operations = 65
- Storage locations needed: 25 for matrix and its factors + 5 for input vector; total 30.

Sparse Matrix Solution Techniques

Case 2 (node reordering → diagonal pivoting)

$$\begin{bmatrix} x & & & & x \\ & x & & & x \\ & & x & & x \\ & & & x & x \\ x & x & x & x & x \end{bmatrix} \begin{bmatrix} v_5 \\ v_2 \\ v_3 \\ v_4 \\ v_1 \end{bmatrix} = \begin{bmatrix} F \\ F \\ F \\ F \\ I \end{bmatrix}$$

- Number of operations required for LU factorization: 2+2+2+2 = 8
- Number of operations required for Forward and Backward Substitution: 1 + 4= 5
- Total number of operations = 13
- Storage locations needed: 13 for matrix and its factors + 5 for input vector; total 18

Number of operations with sparsity considerations

(no operations on zeros)

Factorization:

$$\alpha = \sum_{k=1}^{n} |\mathbb{L}_{\cdot k}| \left(|\mathbb{U}_{k\cdot}| - 1 \right)$$

$|\mathbb{L}_{\cdot k}| \equiv$ # of nonzero elements in col. k of $\mathbb{L}$

$|\mathbb{U}_{k\cdot}| \equiv$ # of nonzero elements in row k of $\mathbb{U}$

Forward and Backward Substitutions

Assume **b** (rhs vector) is full

$$\beta = |\mathbb{L}| + |\mathbb{U}| - n \quad \text{(worst case)}$$

Also, β = storage for $\mathbb{L}$ and $\mathbb{U}$.

Reordering algorithms for sparsity

- Ideally one would want to minimize α + β. We will describe two schemes: one attempts to minimize α, and the second β.

  Scheme 1 **(***Markowitz*)**:** Minimize α (minimum operations, minimum degree)

- At each step in the reordering process, choose the pivot that requires the least number of operations.
- If more than one pivot candidate exists, choose any one of them.
- After a pivot is selected, check if any fills would be created by it.
- If fills would be created, modify the structure of the remaining submatrix by adding the fills.

Reordering algorithms for sparsity (cont.)

- **Variation:**

  If more than one pivot candidate exists, choose the one that creates the minimum number of fills. If more than one candidate satisfies this criterion, choose any one of them.

  Scheme 2: Minimum-Fill (attempts to minimize β)

- At each step in the reordering process, choose the pivot that creates the minimum number of fills.
- If more than one pivot satisfies the criterion, choose any one of them.
- After a pivot is selected, account for any fills it may create and modify the structure of the remaining submatrix to account for the fills.

Reordering algorithms for sparsity (cont.)

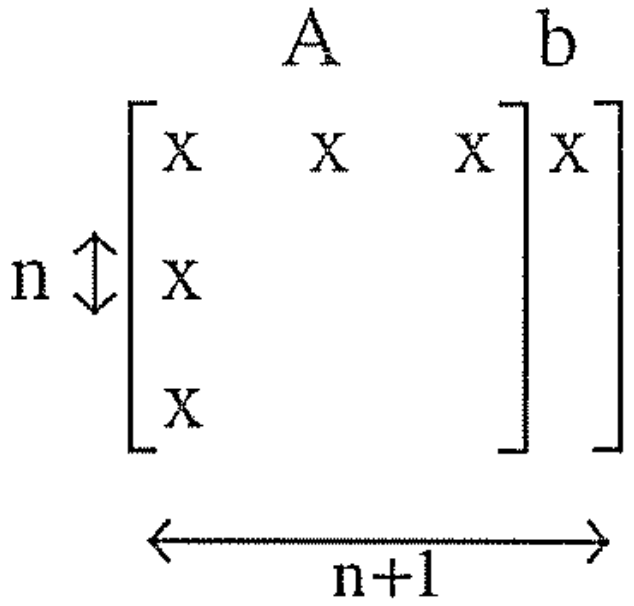
- **Variation 1**: If more than one pivot meets the criterion, choose one that requires minimum operations. If more than one pivot meets this additional criterion, choose any one of them.

  (This variation attempts to reduce o locally as a second priority.)

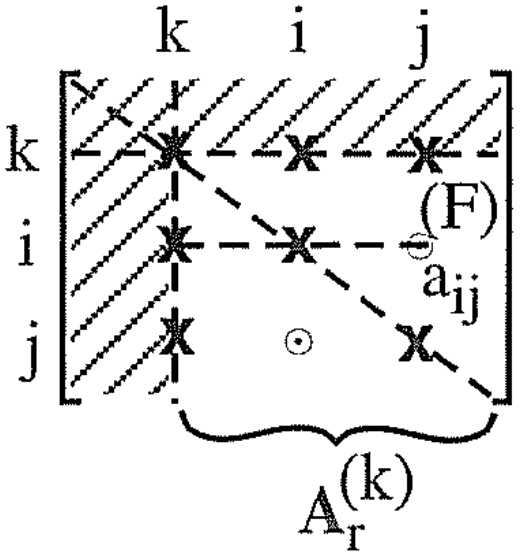
- **Variation 2:** If more than one pivot meets the criterion, choose one that has the maximum number of off-diagonal elements (in row plus column in reduced matrix). The idea is to get rid of as many nonzero elements as possible as early as possible. This may increase operation locally, but may decrease it later.

Remark

- If the structure of the rhs vector **b** is ***fixed***, then include its structure as an extra column of matrix **A** for reordering purposes



How a fill is created



$$a_{ij}^{(k+1)} \leftarrow a_{ij}^{(k)} - a_{ik}^{(k+1)} \bullet a_{kj}^{(k+1)}$$

- A fill is created at location (i, j) in $A_r^{(k+1)}$ if:

$$a_{ij}^{(k)} = 0 \quad \text{and both } a_{ik}^{(k+1)} \neq 0 \text{ and } a_{kj}^{(k+1)} \neq 0$$

- In structurally symmetric matrices, fills are created in pairs: at (i, j) and at (j, i)

Example

$$\begin{bmatrix} a_{11} & 0 & a_{13} & 0 \\ a_{21} & a_{22} & 0 & a_{24} \\ a_{31} & 0 & a_{33} & a_{34} \\ a_{41} & a_{42} & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} x & 0 & x & 0 \\ x & x & 0 & x \\ x & 0 & x & x \\ x & x & 0 & 0 \end{bmatrix}$$

non-symmetric matrix structure

$$\begin{bmatrix} x & 0 & x & 0 \\ x & x & F & x \\ x & 0 & x & x \\ x & x & F & F \end{bmatrix} \quad \begin{matrix} 4 \\ 4 \\ 2 \\ 0 \\ \hline 10 \end{matrix}$$

Number of operations in factorization with the given order =10

Scheme 1

(Markowitz, minimum operations) Consider complete pivoting:

$$\rightarrow \underbrace{\begin{bmatrix} 4 & - & 2 & - \\ 8 & 4 & - & 4 \\ 8 & - & 4 & 4 \\ 4 & 2 & - & - \end{bmatrix}}_{\mathbf{U} \text{ has l's on Diag}} , \quad \underbrace{\begin{bmatrix} 6 & - & 2 & - \\ 9 & 3 & - & 3 \\ 9 & - & 3 & 3 \\ 6 & 2 & - & - \end{bmatrix}}_{\mathbf{L} \text{ has l's on Diag}}$$

Note: Number in matrix indicate # of operations taking each entry as a pivot at a time

Scheme 1

Continue with case where **U** has **I**’s on Diagonal

$$\begin{bmatrix} x & 0 & x & 0 \\ x & x & 0 & x \\ x & 0 & x & x \\ x & x & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} - & - & - & - \\ 6 & 4 & - & 4 \\ 3 & - & - & 2 \\ 3 & 2 & - & - \end{bmatrix}$$

$$\begin{bmatrix} x & 0 & x & 0 \\ x & x & 0 & x \\ x & 0 & x & x \\ x & x & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} - & - & - & - \\ 2 & - & - & 2 \\ 2 & - & - & 2 \\ - & - & - & - \end{bmatrix}$$

Scheme 1

Reordered Matrix

$$\begin{bmatrix} a_{13} & 0 & a_{11} & 0 \\ 0 & a_{42} & a_{41} & 0 \\ 0 & a_{22} & a_{21} & a_{23} \\ a_{33} & 0 & a_{31} & a_{34} \end{bmatrix}$$

No fills; total number of operations = 2 + 2 + 2 = 6
(was 10 operations)

Scheme 2 (minimum fills)

$$\begin{bmatrix} x & 0 & x & 0 \\ x & x & 0 & x \\ x & 0 & x & x \\ x & x & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & - & 0 & - \\ 4 & 1 & - & 1 \\ 4 & - & 1 & 1 \\ 2 & 0 & - & - \end{bmatrix}$$

Pivoting on a_{13} or on a_{42} produces no fills

Scheme 1 with Diagonal Pivoting

$$\begin{bmatrix} x & 0 & x & 0 \\ x & x & 0 & x \\ x & 0 & x & x \\ x & x & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 4 \\ 4 \\ 4 \\ - \end{bmatrix}, \quad \begin{bmatrix} 2 \\ 1 \\ 1 \\ - \end{bmatrix}$$

oper fill

$$\begin{bmatrix} x & 0 & x & 0 \\ x & x & 0 & x \\ x & 0 & x & x \\ x & x & 0 & F \end{bmatrix}, \quad \begin{bmatrix} 3 \\ - \\ 4 \\ 2 \end{bmatrix}$$

Scheme 1 with Diagonal Pivoting

$$\begin{bmatrix} x & 0 & x & 0 \\ x & x & 0 & x \\ x & 0 & x & x \\ x & x & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 2 \\ - \\ 2 \\ - \end{bmatrix}$$

$$\begin{bmatrix} a_{22} & a_{24} & a_{21} & 0 \\ a_{42} & F & a_{41} & 0 \\ 0 & 0 & a_{11} & a_{13} \\ 0 & a_{34} & a_{31} & a_{33} \end{bmatrix} \begin{matrix} 4 \\ 2 \\ \underline{2} \\ 8 \end{matrix}$$

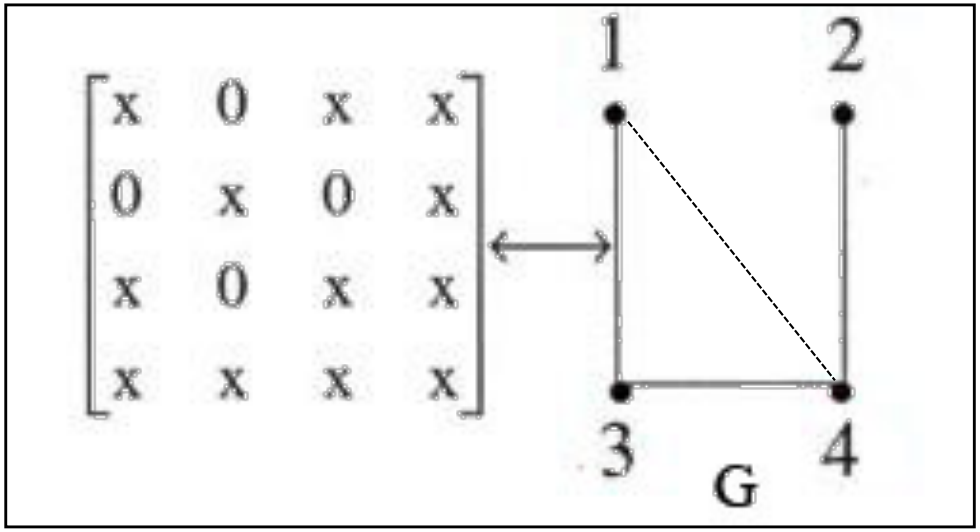
Final order

Graphical representation of matrices and graphical interpretation of Gaussian elimination

A **structurally** symmetric matrix can be represented by an associated (undirected) graph G:

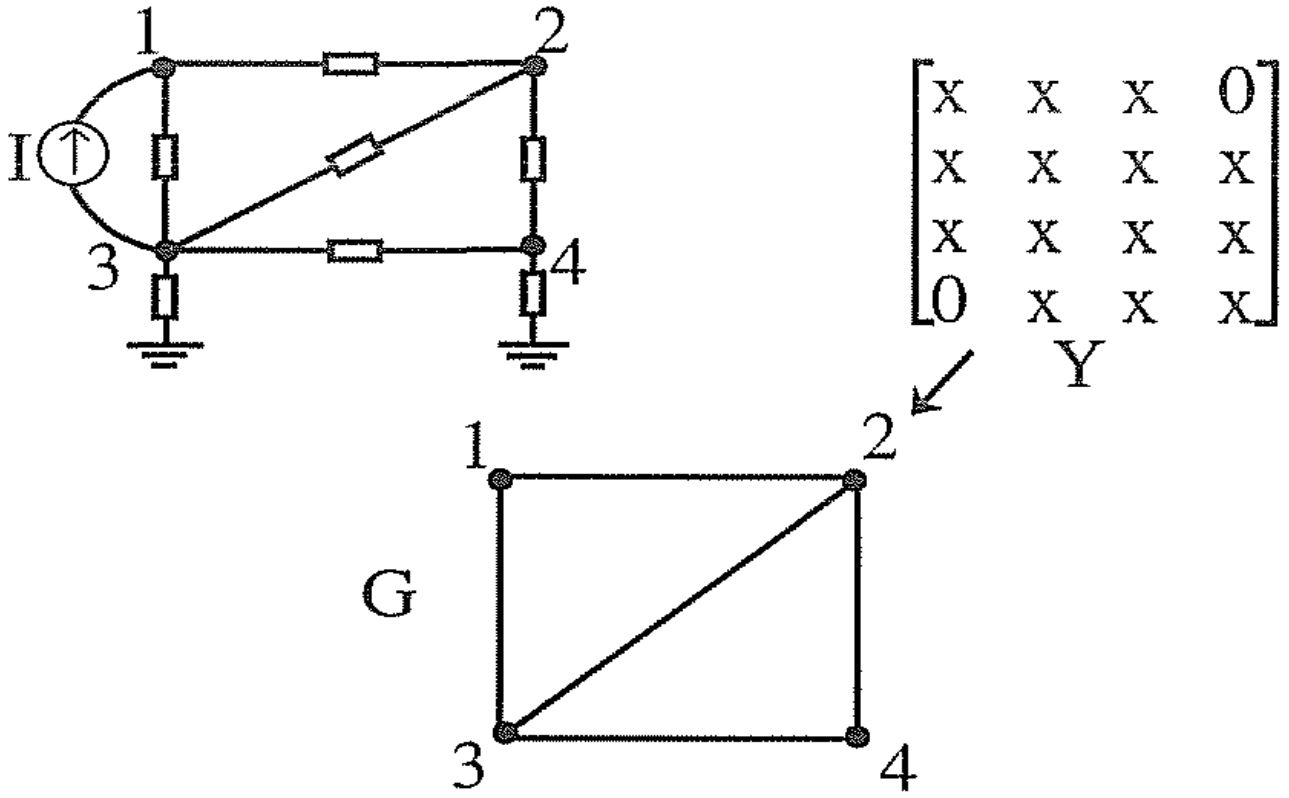
- Each diagonal entry in the matrix corresponds to a node in the graph G.
- Each nonzero off-diagonal pair in positions (i, j) and (j, i) in the matrix is represented by an edge connecting nodes i and j in G.

Example



The associated graph of the **nodal admittance matrix** of a "passive" RLC circuit is the circuit graph itself, with edges corresponding to independent (current) sources and edges connected to the ground node removed

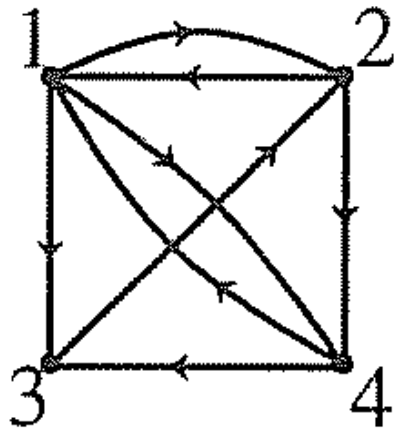
Example



Remark

- A structurally **unsymmetrical** matrix can be represented by a **directed** graph. Each nonzero off-diagonal entry in position (i, j) in the matrix is represented by a directed edge from node i to node j in the graph.

$$\begin{bmatrix} x & x & x & x \\ x & x & 0 & x \\ 0 & x & x & 0 \\ x & 0 & x & x \end{bmatrix}$$



Directed Graph

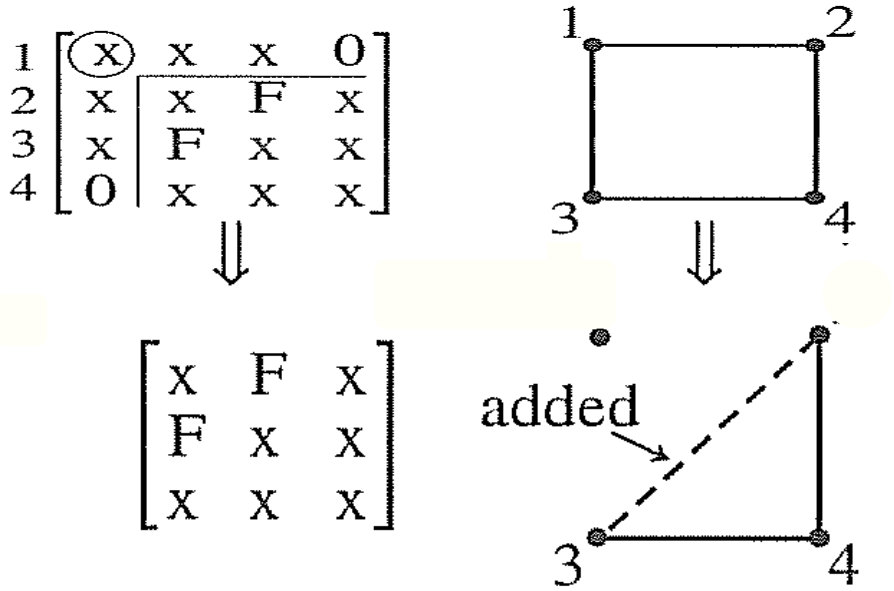
Consider structurally symmetric matrices

- Pivoting on a diagonal entry in the matrix is equivalent to removing the corresponding node in the associated graph, together with all edges incident at it.

- Any two neighboring nodes to the pivot node (i.e., nodes connected to it by edges) that are not connected to each other by an edge, will be connected together by a new edge in the reduced graph.

- The new added edges correspond to fills, and the reduced graph corresponds to the reduced matrix.

Example



- Reordering the sequence of pivoting along the diagonal of a structurally symmetric matrix is equivalent to reordering the sequence of node elimination in the associated graph.

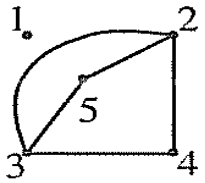
- **Diagonal reordering of structurally symmetrical matrices for sparsity using schemes 1 and 2 follows in the next slides**:

Scheme 1 (Diagonal Pivoting)

(1) $d_1 = 2, d_2 = 3, d_3 = 3, d_4 = 2, d_5 = 2$

(2) Choose 1

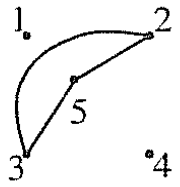
(2 fills)



(3) $d_2 = 3 - 1 + 1 = 3, d_3 = 3 - 1 + 1 = 3, d_4 = 2, d_5 = 2$

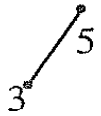
(4) Choose 4

(no fills)



(5) $d_2 = 3 - 1 = 2, d_3 = 3 - 1 = 2, d_5 = 2$

(6) Choose 2



(7) Choose 3, then 5.

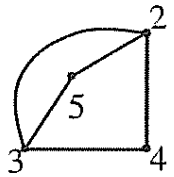
Scheme 2

(minimum fill)

($f \triangleq$ number of fills)

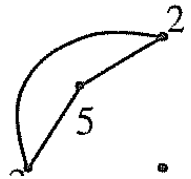
(1) $f_1 = 2, f_2 = 6, f_3 = 6, f_4 = 2, f_5 = 2$

(2) Choose 1



(3) $f_2 = 2, f_3 = 2, f_4 = 0, f_5 = 0$

(4) Choose 4
(no fills)



(5) $f_2 = 0, f_3 = 0, f_5 = 0$

(6) Choose 2, 3, 5.

Data Structures for Sparse Matrix Solution Techniques

- (1) Matrix Storage
- (2) Reordering Schemes

Data Structures for Storing and Operating on Sparse Matrices

- **Aim:** To store and operate only on nonzero entries, including fills.

  – Linear List
  – Orthogonal Linked-List

Example:

$$\begin{bmatrix} a_{11} & 0 & a_{13} & a_{14} \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} \\ a_{41} & a_{42} & 0 & a_{44} \end{bmatrix}$$ real or complex

Linear List (row-by-row)

$$\begin{array}{l} \quad\quad 1 \quad 2 \quad\; 3 \quad\; 4 \quad\; 5 \quad\; 6 \quad\; 7 \quad 8 \quad 9 \\ A = [a_{11}\ a_{13}\ a_{14} \vdots a_{22} \vdots a_{33}\ a_{34} \vdots a_{41}\ a_{42}\ a_{44}] \end{array}$$

$IB = [1 \;\; 4 \;\; 5 \;\; 7 \;\; 10]$ Pointers to rows in A

$JB = [1 \;\; 3 \;\; 4 \vdots 2 \vdots 3 \;\; 4 \vdots 1 \;\; 2 \;\; 4]$ Col. indices

- **How to locate an element a_{ij}** (e.g., a_{34} in row i = 3)
  k = IB (i = 3) = 5

- Number of nonzero elements in row 3:
  IB(4) - IB(3) = 7 - 5 = 2

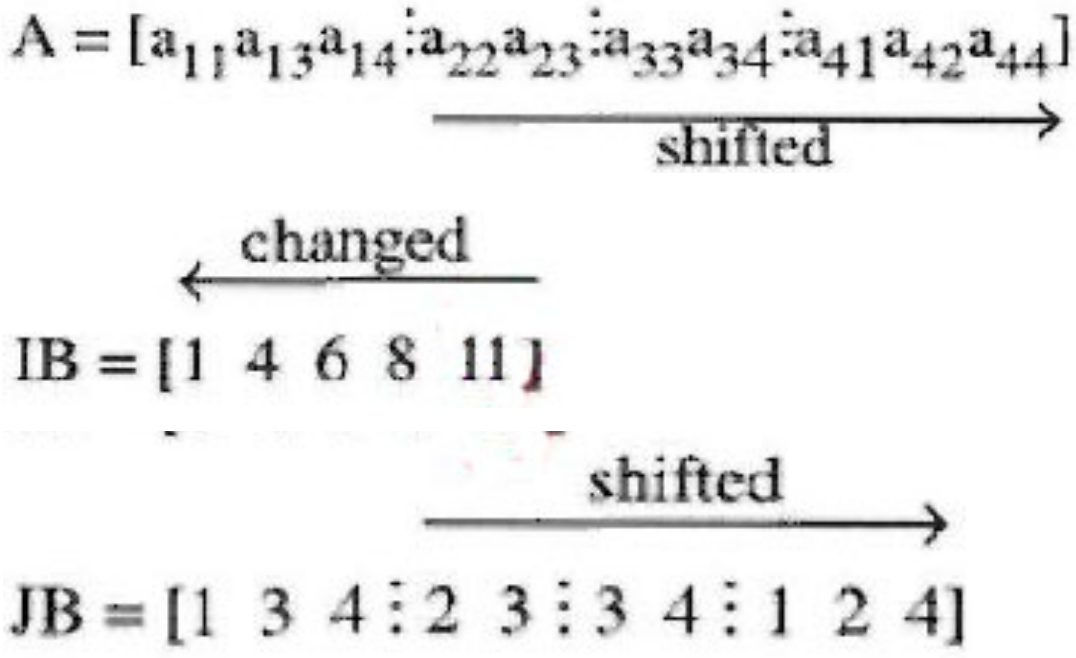
=> Column indices of all nonzero elements in row 3 are in locations 5 and 6 in JB and their values are in locations 5 and 6 in **A**:

$$A(5) = a_{33}$$

$$A(6) = a_{34}$$

Linear List (row-by-row) (cont.)

- **How to insert a new element in the matrix** (e.g., add a_{23}):



Linear List (row-by-row) (cont.)

Structurally Symmetric Matrices

$$\begin{bmatrix} a_{11} & 0 & a_{13} & a_{14} \\ 0 & a_{22} & 0 & a_{24} \\ a_{31} & 0 & a_{33} & 0 \\ a_{41} & a_{42} & 0 & a_{44} \end{bmatrix}$$

$$\begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ A = [& \underbrace{a_{11} a_{22} a_{33} a_{44}} & : & \underbrace{a_{13} a_{14} a_{24}} & : & \underbrace{a_{31} a_{41} a_{42}} &] \\ & \text{diag.} & & \text{rows} & & \text{cols.} & \end{matrix}$$

n =4

IB = [5 7 8 8] Number of nonzero elem. in upper part of A = 8-5 = 3

JB = [3 4 : 4]

Linear List (row-by-row) (cont.)

How to find a_{24} in Row 2

- k = 2
  IB(2) = 7
- IB(3) - IB(2) =1 Number of nonzero elements in off-diagonal row 2 of upper matrix
- JB(7-4) = JB(3) = 4 column index of element in row 2
- A(7) = a_{24}
- Find a_{42}: IB(2) + 3 = 7 + 3 = 10 => A(10) = a_{42}

Linear List (row-by-row) (cont.)

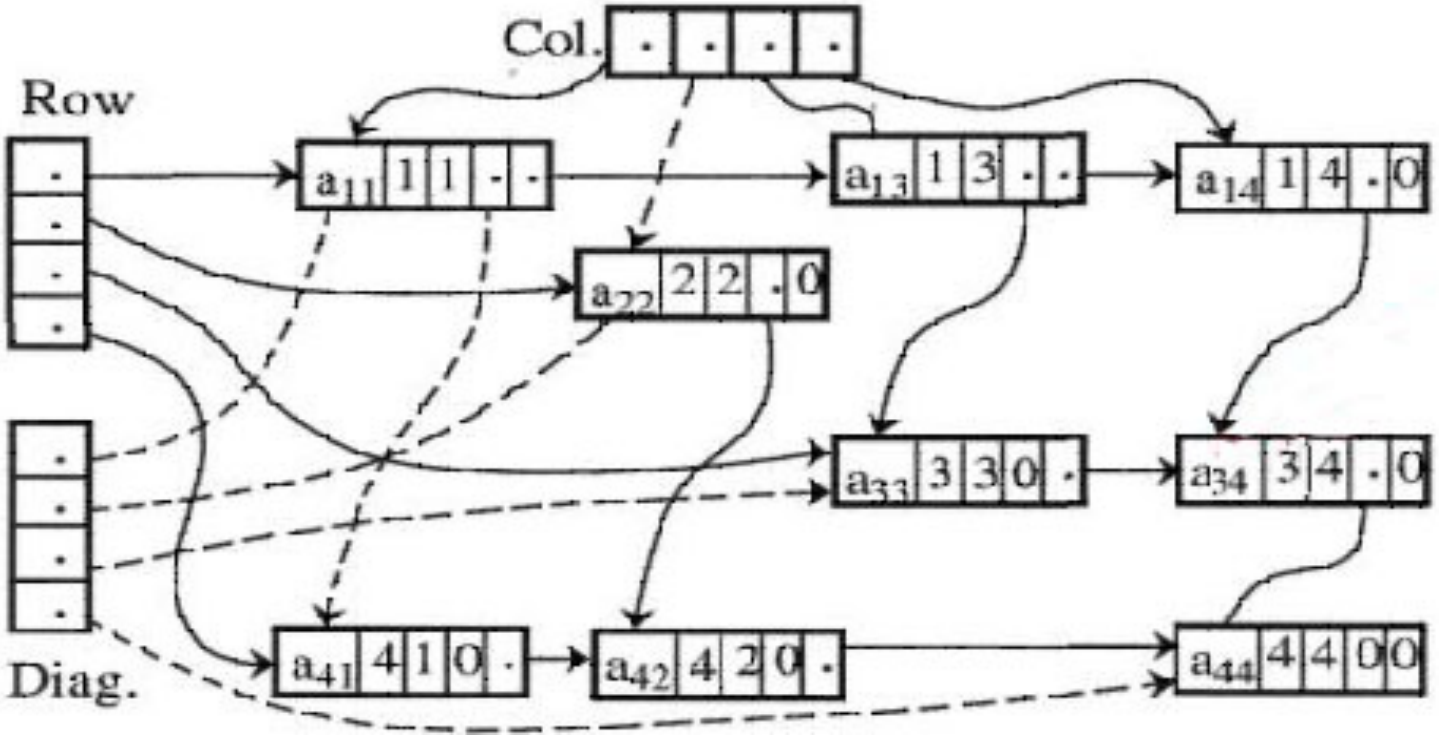
Numerically Symmetric Matrices

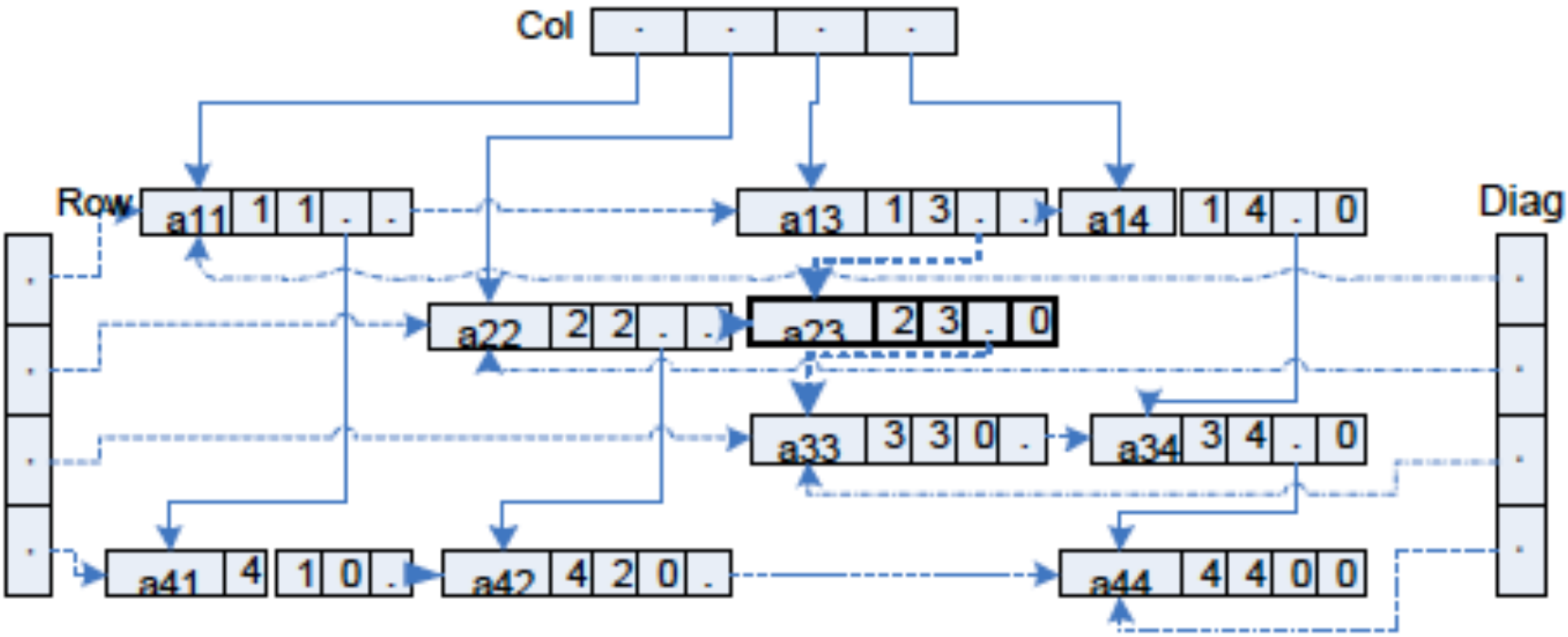
1 2 3 4 5 6 7

- n = 4
- IB [5 7 8 8]
- JB = [3 4 4]

(2) Orthogonal Linked-List

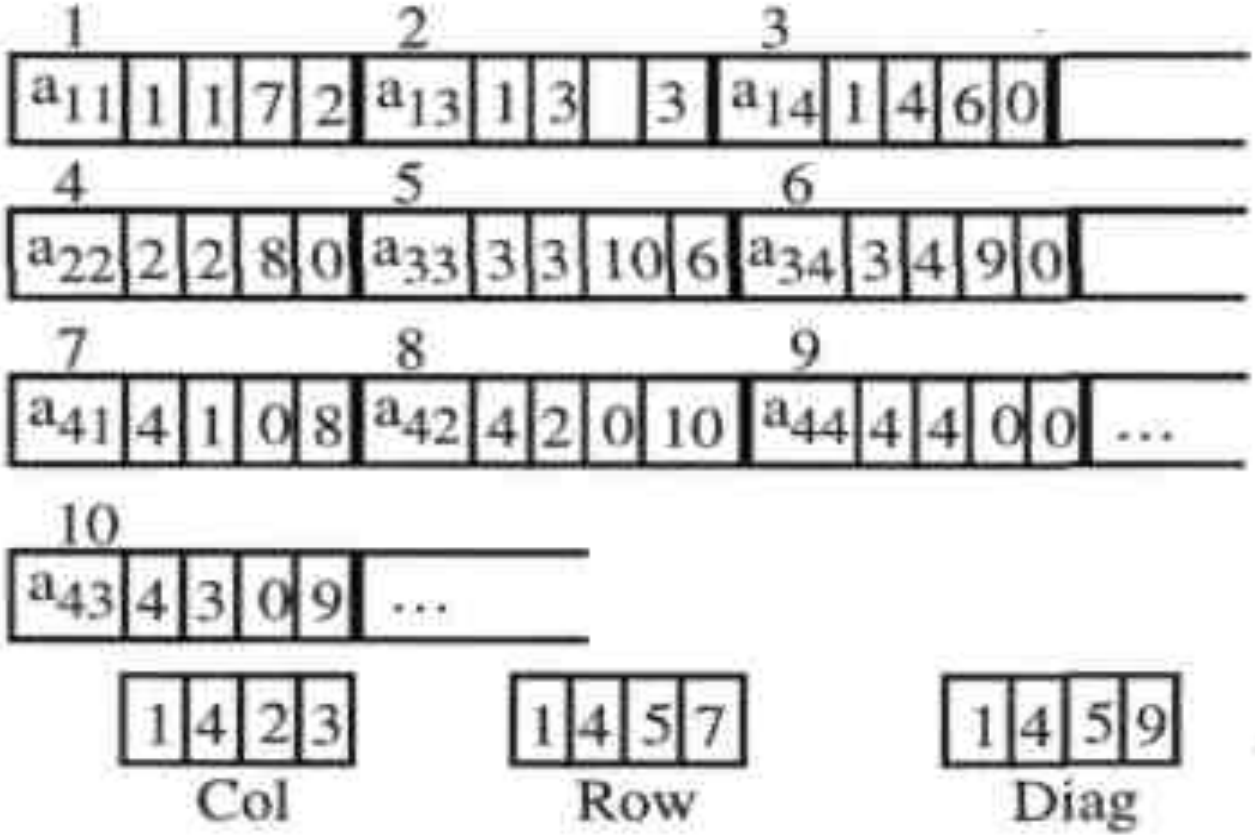
$$\begin{bmatrix} a_{11} & 0 & a_{13} & a_{14} \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} \\ a_{41} & a_{42} & 0 & a_{44} \end{bmatrix}$$




Col
Row
Diag
a11 1 1 . .
a13 1 3 . .
a14 1 4 . 0
a22 2 2 . .
a23 2 3 . 0
a33 3 3 0 .
a34 3 4 . 0
a41 4
1 0 .
a42 4 2 0 .
a44 4 4 0 0

(2) Orthogonal Linked-List (cont.)

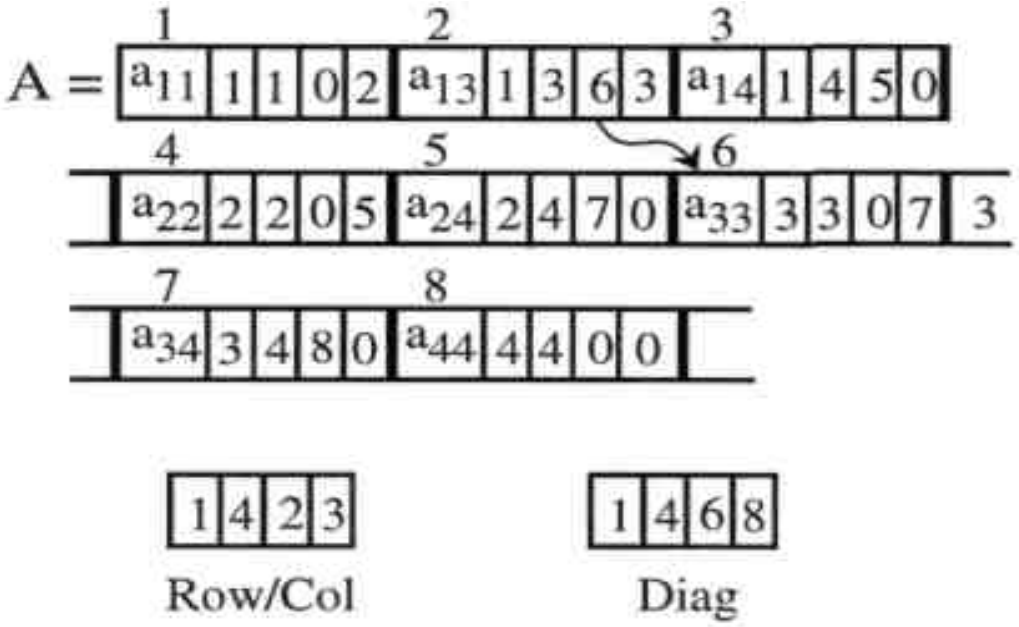
A and pointers are stored as a "linear vector"



(2) **Orthogonal Linked-List** (cont.)

Numerically Symmetric Matrices

$$\begin{bmatrix} a_{11} & 0 & a_{13} & a_{14} \\ 0 & a_{22} & 0 & a_{24} \\ a_{13} & 0 & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix}$$



Data Structure for Sparse Matrix Reordering

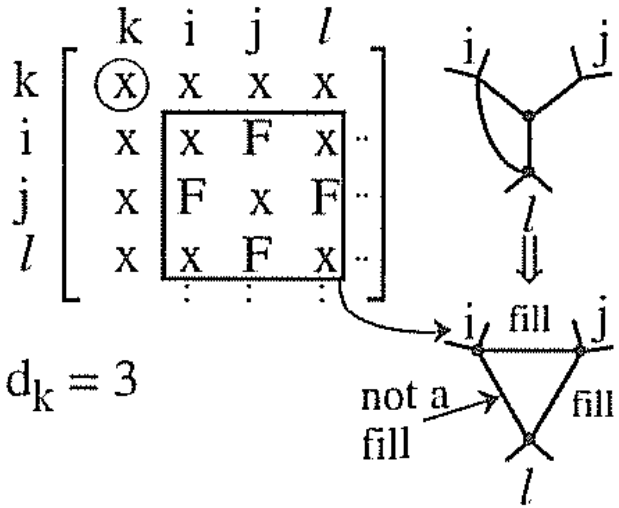
- For the sake of clarity, we'll assume structurally symmetric matrices and consider diagonal pivoting.
- These matrices can be represented by undirected elimination graphs
- Diagonal pivoting is equivalent to renumbering (or reordering) the nodes in the elimination graph.
- "Nearly" structurally symmetric matrices can be considered to be structurally symmetric with zero-valued entries in some (few) locations
- The # of operations and the # of fills can be determined from the adjacency sets of nodes (nodes ↔ diagonal pivots)

Data Structure for Sparse Matrix Reordering (cont.)

- The idea is to update the adjacency sets after a pivot is selected, rather than recomputing them.
- Adjacency set of node , Adj(k), is the set of adjacent nodes or neighbors of k:

$$\text{Adj}(k) = \text{set } \{i \mid a_{ki} \neq 0, i \neq k\}$$

- Define $d_k = |\text{Adj}(k)|$ = degree of k
  = Number of nonzero elements in row k, excluding the diagonal



Data Structure for Sparse Matrix Reordering (cont.)

- After pivot k is selected, the sub-block consisting of the nonzero entries in row/col k become a dense block — and all nodes adjacent to node k in the associated graph become pairwise adjacent (with fills created as necessary) to form a CLIQUE or SUPERNODE.

- Structural modifications (adjacency information updating) are required only for nodes adjacent to pivot node.

- Fill information updating is confined to nodes at distance 1 and 2 from the pivot node; where distance between two nodes k and *l* is defined as the mm. # of edges that must be traversed in going from k to *l*.

Reordering Algorithms Revisited

- Consider structurally symmetric matrices and diagonal pivoting

- Aim: Update operation count (and fill) after each pivot selection, rather than re-computing them.

Ref: VLACH & SINGHAL, Chap. 2

Reordering Algorithms Revisited (cont.)

- ***Scheme 1*** (Markowitz, minimum operations, or minimum degree)
- **Step** 1: Compute the degree of each diagonal pivot d_k = |Adj(k)|
  [# of operation required by pivot k is d_k (d_k + 1)]
- **Step** 2: Choose one pivot with minimum degree (say pivot k).
- **Step 3:** If d_k = 1, go to Step 5.
- **Step 4:** Scan the nodes {i, j, *l*} Є Adj(k). For each pair (i, j) ЄAdj(k). if *l* Є Adj(j), which implies j Є Adj(i)), add i to Adj(j) and j to Adj(i), and set
- **Step 5:** Since k is new deleted, remove k from the adjacent sets of all its adjacent nodes and set
- **Step 6:** Go to Step 2, until only two nodes are left.

Reordering Algorithms Revisited (cont.)

- ***Scheme 2*** (Minimum Fill)

- **Step 1:** Compute the number of fills each pivot would create as well as the degree of each pivot. This is done by scanning the adjacency set of each pivot.

- **Step 2:** Choose a pivot, say k, with minimum fill f_k.

- **Step 3:** If $f_k = 0$, go to Step 5.

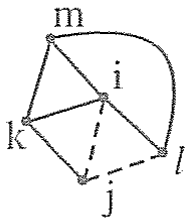
- **Step 4:** Scan the nodes in Adj(k). As each fill is found, say (i, j), add i to Adj(j) and j to Adj(i) and set

$$d_i \leftarrow d_i + 1 \text{ and } d_j \leftarrow d_j + 1$$

Fill information is updated as follows

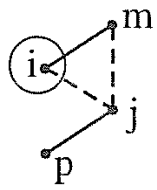
4(a): Each node *l* adjacent to both i and j was creating the same fill, therefore

$$\check{f}_l \leftarrow f_l - 1$$



4(b): If node i is selected next as a pivot, then it would create at i fill between j and each node in m Є to Adj(i) but Є Adj(j), and we set

$$f_i \leftarrow f_i + \alpha$$

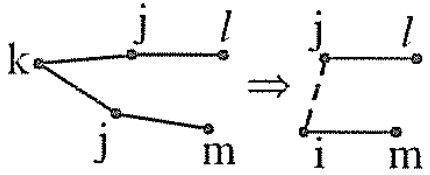


where α is the number of nodes in Adj(i) but not in Adj(j).

Cont.

Similarly, $f_j \leftarrow f_j + \beta$

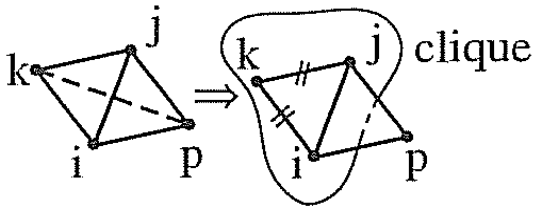
where β is the number of nodes in Adj(j) but not in Adj(i).



Step 5: Since k is now to be deleted, all nodes 2 Adj(k) can no longer create fills incident on k, and their fill information is modified as follows:

$$f_i \leftarrow f_i - (d_i - d_k)$$

where d_k and d_i are the degrees at k and i (after adding fills).



Cont.

- **Step 6:** Update the adjacency sets for all i Є Adj(k):

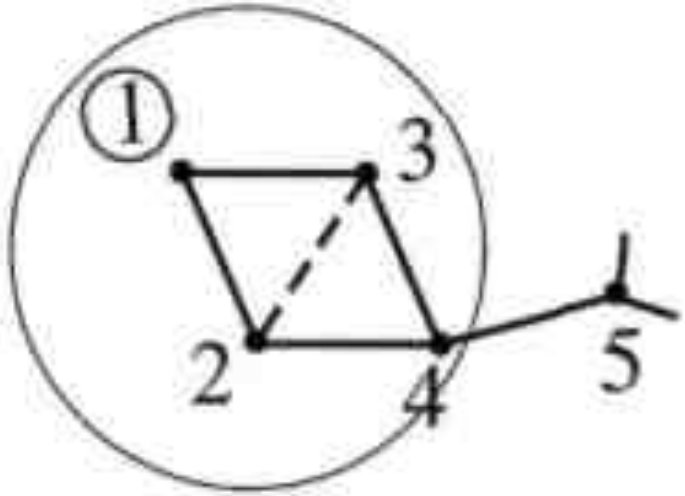
$$d_i \leftarrow d_i - 1$$

- **Step 7:** Go to Step 2, until two nodes are left.

Remarks

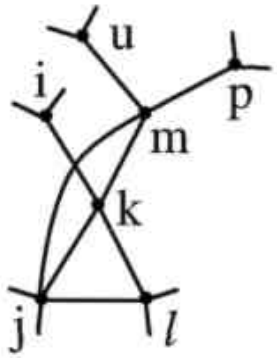
- Give priority in next pivot selection to neighbors. In many cases, after a pivot is selected, the fills of some of its neighbors are reduced to zero. Also, this creates clusters of full diagonal sub-blocks (I-nodes).

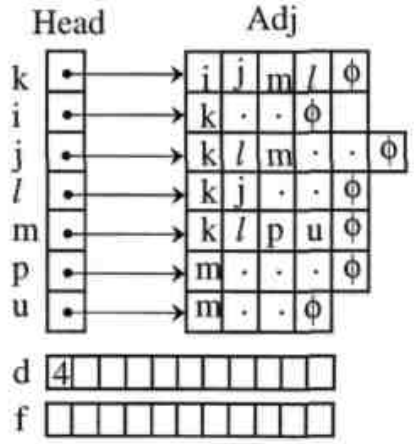
- In addition, neighbors are often numerically dependent on each other.



Data Structure for Storing and Updating Adj(●)

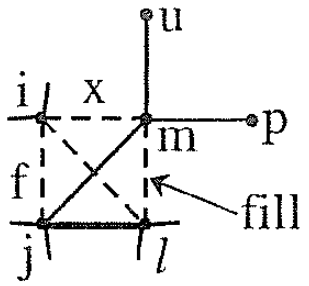
	k	i	j	*l*	m	p	u	
k	x	x	x	x	x			
i	x	x						⋯
j	x		x	x	x			⋯
l	x		x	x				
m	x		x		x	x	x	⋯
p					x	x		⋯
u					x		x	⋯
					⋮		⋮	

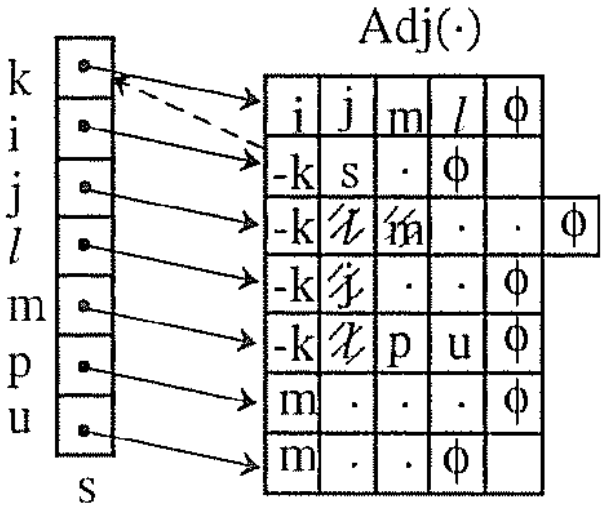




Data Structure for Storing and Updating Adj(●) (cont.)

- As k is numbered, all nodes that are adjacent to it become pairwi adjacent, with fills created as necessary:





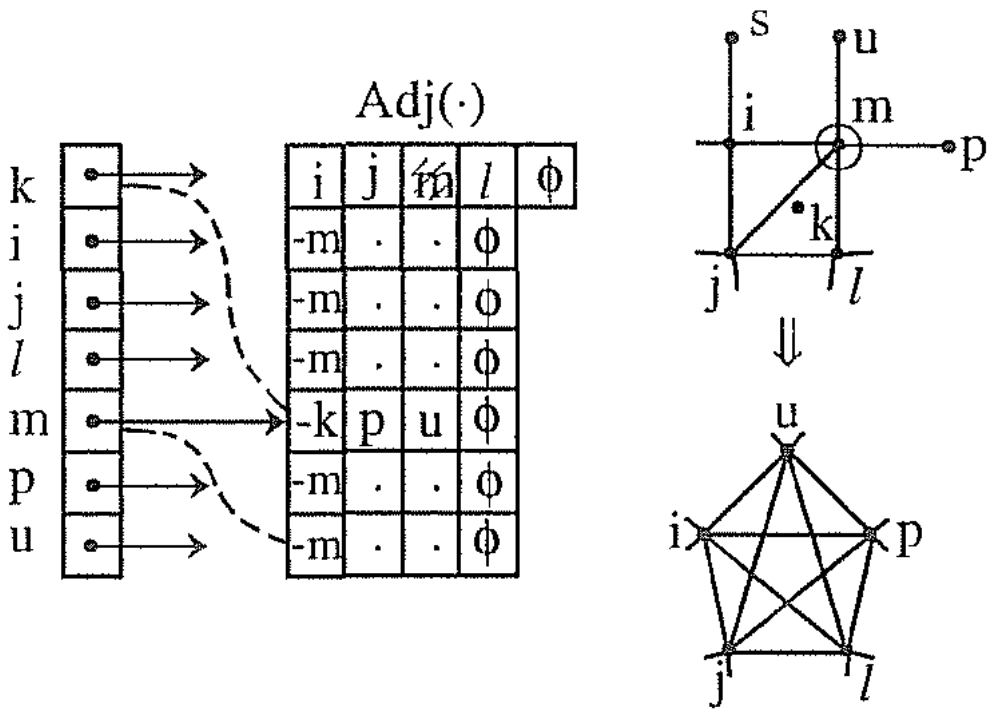
Data Structure for Storing and Updating Adj(●) (cont.)

- Pivoting on k: replace k by –k
- Edges that connect nodes in Adj(k) are deleted to free storage space
- The adjacency set of any node i is found as the union of ordinary nodes (with positive values) and the adjacencies of the E-nodes (those with negative values) in Adj(i).
- Self loops are ignored.
- Suppose node m is chosen next. Then all neighbors of k and m become pairwise adjacent.
- A common E-node combining k and m is formed and labeled m.

Data Structure for Storing and Updating Adj(●) (cont.)

- This is done by replacing all instances of m and -k with -m in array Adj (other than in Adj(m)).

- Self loops and duplicate entries are deleted.



Sparse Matrix Solution Program Outline

1. **Ordering routine** which numbers the equations and variables: This routine is usually executed only once based on a specific zero-nonzero structure of matrix **A**.

   The ordering routine is often independent of the actual values of **A**.

2. **A symbolic factorization routine,** which determines the zero-nonzero structure of **L** and **U**.

   – In some cases a machine code is created that will perform the factorization of matrices with specific structures.

   – The symbolic factorization routine is executed once for a given structure of **A** after ordering.

Sparse Matrix Solution Program Outline (cont.)

3. **Symbolic Solution Routine**:
   This routine analyzes the zero-nonzero structure of the rhs vectors and determines the variables that must be found in order to generate the ‘outputs.’ This routine also generates the forward and backward substitution code.

4. **Numeric Factorization** routine which is invoked each time the values in A are modified.

5. A **Numeric Solution** routine which is executed each time the rhs vector is changed.

Numerical considerations

in sparse matrix solution

Pivoting for Sparsity and for Numerical Accuracy

- **Often in conflict:**
  Sparse matrix reordering is often based on matrix structure, with no regard to pivot values.

- During the numerical factorization phase, pivot values could become zero or small, leading to numerical instability.

- We will describe ***four*** methods used to handle this problem.

- **Remember:** If the matrix is diagonally dominant or symmetric positive definite, pivoting for numerical stability is not necessary provided diagonal pivoting is used in reordering for scarcity.

Method 1

- If the *entries* of the matrix are fixed (i.e., factorization is done only once), then during the sparse matrix reordering process, choose a pivot only if its numerical value is larger (in absolute value) than some threshold value.

  - In this case, *numerical elimination of a pivot* is carried out as soon as the pivot is selected so that the latest updated values of the next pivot candidates are available.

Method 2

- If the values of the matrix entries change from one solution to the next, while the structure remains the same, then during the numerical factorization phase, if a pivot's absolute value is *smaller than a threshold value*, reorder the remaining part of the matrix for both numerical stability and sacristy. (Expensive)

Method 3

Solution Updating Method

- During the numerical factorization phase, if a *pivot* is encountered that is smaller in absolute value than a threshold value, add to a constant (say, +1.0 if it is positive, or a -1.0 if it is negative).
- Suppose p such pivots are modified. Then the resulting **LU** factors will be the factors of a modified matrix, $\mathbf{B = L_B U_B}$, where the original matrix **A** is related to **B** by

- $\mathbf{A = B - PDQ^T,\ \ Q = P}$

- where **D** is a diagonal matrix of dimension p.

Method 3 (cont.)

- **Remember:** Sherman-Morrison-Woodbury (Householder) Formula that relates the inverses of matrices **A** and **B,**, where $\mathbf{A = B \pm PDQ^T}$, (**A** and **B** are nonsigular)

- In our case: $\mathbf{A = B - PQ^T}$, $\mathbf{D = I}$, then

$$\mathbf{A^{-1} = B^{-1} + B^{-1}P(I- Q^TB^{-1}P)^{-1} Q^T B^{-1}}$$

Solution Updating Algorithm:

(1) Solve $\mathbf{B}\hat{\mathbf{x}} = \mathbf{b} \leftarrow \mathbf{L_B U_B}\ \hat{\mathbf{x}} = \mathbf{b}$ (modified system)

(2) Solve **BV = P,** requires p forward and backward substitutions

(3) Form $\mathbf{H} = \mathbf{Q}^{\mathbf{T}}\mathbf{V}, \mathbf{y} = \mathbf{Q}^{\mathbf{T}}\hat{\mathbf{x}}$

(4) Solve $\mathbf{(D^{-1} - H)z = -y}$ (p × p system of equations)

(5) x = $\hat{\mathbf{x}}$ **- Vz** (solution of original system)

(In our case: $\mathbf{Q^T = P^T}$, $\mathbf{D^{-1} = I}$)

Method 3 (cont.)

For example, suppose two such pivots, k and ℓ, are modified, then

$$\mathbf{A} = \mathbf{B} - \begin{array}{c} \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cc} \mathrm{k} & \ell \end{array} \\ \begin{bmatrix} 0 & & & \mathrm{O} \\ & +1 & & \\ & & +1 & \\ \mathrm{O} & & & 0 \end{bmatrix} \end{array}$$

$$= \mathbf{B} - \begin{array}{c} \\ \mathrm{k} \\ \ell \\ \\ \end{array} \begin{array}{c} \begin{array}{cc} 0 & 0 \end{array} \\ \begin{bmatrix} 1 & \vdots \\ \vdots & 1 \end{bmatrix} \\ \begin{array}{cc} 0 & 0 \end{array} \end{array} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathrm{D}} \underbrace{\begin{array}{c} \begin{array}{cccc} & \mathrm{k} & & \ell \end{array} \\ \begin{bmatrix} 0 & 1 & \cdots & 0 \\ 0 & \cdots & 1 & 0 \end{bmatrix} \end{array}}_{\mathrm{P}^{\mathrm{T}}}$$

$$\underbrace{\phantom{\begin{bmatrix} 0 & 0 \\ 1 & \vdots \\ \vdots & 1 \\ 0 & 0 \end{bmatrix}}}_{\mathrm{P}}$$

Example (cont.)

(1) Obtain $\mathbf{B} = \mathbf{L_B U_B}$ and solve $\mathbf{L_B U_B}\,\hat{\mathbf{x}} = \mathbf{b}$

(2) Solve $\mathbf{L_B U_B}$ $\begin{bmatrix} 0 & 0 \\ 1 & \vdots \\ & 1 \\ 0 & 0 \end{bmatrix} \Rightarrow \mathbf{V} = [\mathbf{v}_1 \ \mathbf{v}_2]$ V

(3)

$$\mathbf{H} = \begin{bmatrix} 0 & \cdots & 1 & \cdots & 0 \\ 0 & \cdots & \cdots & 1 & 0 \end{bmatrix} \mathbf{V} = \begin{bmatrix} \mathbf{v}_{1,k} & \mathbf{v}_{2,k} \\ \mathbf{v}_{1,\ell} & \mathbf{v}_{2,\ell} \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 0 & \cdots & 1 & \cdots & 0 \\ 0 & \cdots & \cdots & 1 & 0 \end{bmatrix} \hat{\mathbf{x}} = \begin{bmatrix} \hat{\mathbf{x}}_k \\ \hat{\mathbf{x}}_\ell \end{bmatrix}$$

(4) Solve $\mathbf{[-1 + H]z = y}$ (2 × 2 system)

(5) $\mathbf{x = x - [v_1 \ v_2]z}$

Method 4

Iterative Refinement

- Original Equation **Ax = b**
- Modified Equation **Bx = b**
- Let the solution of **Bx** = **b** be $\mathbf{x}^{(1)}$

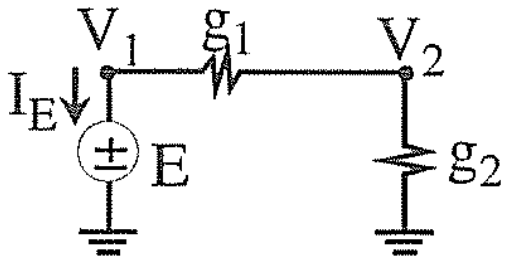
Algorithm (Need to save A)

- Let **k=1**
- Find $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$ "error" in solution
- Solve $\mathbf{B}\,\Delta\mathbf{x}^{(k)} = \mathbf{r}^{(k)}$
- Put $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}$
- Repeat until $||\Delta\mathbf{x}^{(k)}|| < \epsilon$
- **Recommendation:** Use **Method 3** (**Solution Updating**) when the number of pivot changes is small; otherwise, use **Method 4** (**Iterative Refinement**)

Modified Nodal Equations Reordering

Modified Nodal Equations Revisited

$$\begin{bmatrix} g_1 & -g_1 & 1 \\ -g_1 & g_1+g_2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ I_E \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ E \end{bmatrix}$$



- Symmetric
- zero on the diagonal: Careful in diagonal pivoting
- zero on diagonal gets filled after node to which voltage source is connected to is eliminated
  — **at the cost of sacrificing sparsity.**

Modified Nodal Equations Revisited

⇒ Apply row pivoting:

$$\begin{array}{r} \text{El. Char.} \\ \\ \text{KCL} \rightarrow \end{array} \begin{bmatrix} 1 & 0 & 0 \\ g_1 & -g_1 & 1 \\ -g_1 & g_1+g_2 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ I_E \end{bmatrix} = \begin{bmatrix} E \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} \leftarrow \text{El. Char.} \\ \\ \leftarrow \text{KCL at } V_2 \end{array}$$

Followed by **column pivoting**:

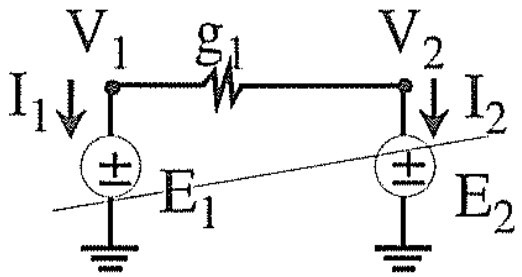
$$\begin{bmatrix} 1 & 0 & 0 \\ g_1 & 1 & -g_1 \\ -g_1 & 0 & g_1+g_2 \end{bmatrix} \begin{bmatrix} V_1 \\ I_E \\ V_2 \end{bmatrix} = \begin{bmatrix} E \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} \leftarrow \text{El. Char. of E} \\ \leftarrow \text{KCL at } V_1 \\ \\ \end{array}$$

Matrix now has no zero on diagonal and can be factorized without creating fills.

Another simple example (Cutset of current variables)

singular

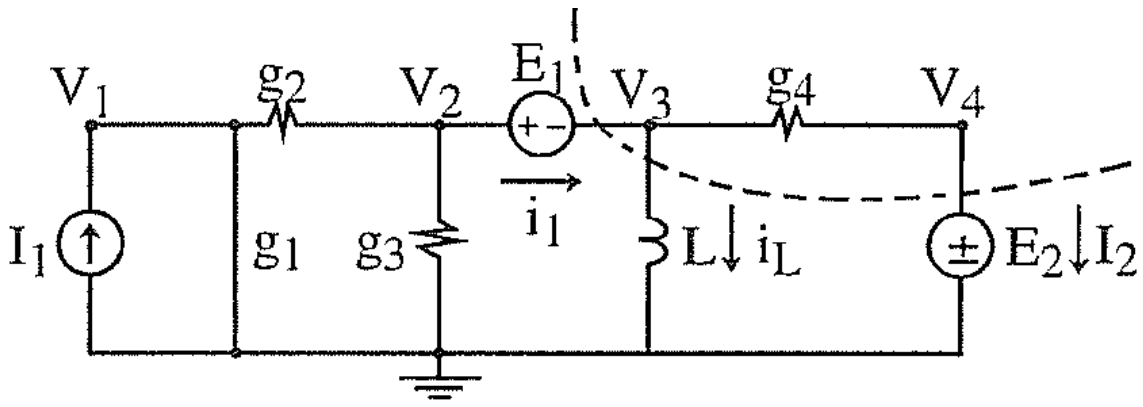
$$\left[\begin{array}{cc:cc} g_1 & -g_1 & 1 & 0 \\ -g_1 & g_1 & F & 1 \\ \hdashline 1 & F & F & 0 \\ 0 & 1 & 0 & F \end{array}\right]\left[\begin{array}{c} V_1 \\ V_2 \\ I_1 \\ I_2 \end{array}\right]=\left[\begin{array}{c} 0 \\ 0 \\ E_1 \\ E_2 \end{array}\right]$$



exchange rows:

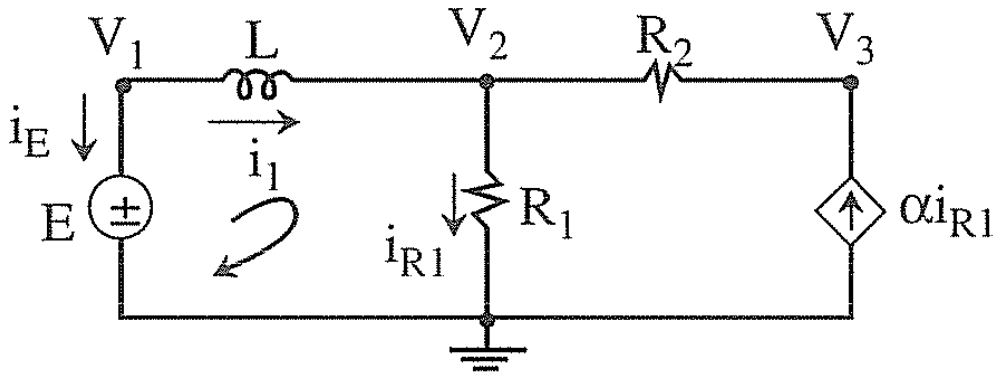
$$\left[\begin{array}{cccc} 1 & & & O \\ & 1 & & \\ g_1 & -g_1 & 1 & \\ -g_1 & g_1 & & 1 \end{array}\right]\left[\begin{array}{c} V_1 \\ V_2 \\ I_1 \\ I_2 \end{array}\right]=\left[\begin{array}{c} E_1 \\ E_2 \\ 0 \\ 0 \end{array}\right]$$

Example



$$
\begin{array}{r}
\text{El.} \\
\text{Char.} \\
\\
V_4 \\
\text{KCL } V_3 \\
V_2 \\
\text{KCL } V_1
\end{array}
\begin{bmatrix}
1 & & & & & & \\
 & 1 & & & -j\omega L & & \\
 & -1 & 1 & & & & \\
g_4 & -g_4 & & 1 & & & \\
-g_4 & g_4 & & & 1 & -1 & \\
 & & g_2+g_3 & & & 1 & -g_2 \\
 & & -g_2 & & & & g_1+g_2
\end{bmatrix}
\begin{bmatrix} V_4 \\ V_3 \\ V_2 \\ i_2 \\ i_L \\ i_1 \\ V_1 \end{bmatrix}
=
\begin{bmatrix} E_2 \\ 0 \\ E_1 \\ \\ \\ \\ I_1 \end{bmatrix}
$$

Example:
Loop of current variables



$R_1 \neq 0$

$$
\begin{array}{lll}
\text{El.} & & \\
\text{Char.} & & \\
\text{KCL} & V_1 & \\
 & V_2 & \\
\text{KCL} & V_3 & \\
\text{El.} & \text{Char.} &
\end{array}
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
+1 & -1 & 0 & -j\omega L & 0 & 0 \\
\hdashline
 & & +1 & +1 & & \\
 & +g_2 & & -1 & -g_2 & 1 \\
\hdashline
 & -g_2 & & & +g_2 & -\alpha \\
 & 1 & & & & -R_1
\end{bmatrix}
\begin{bmatrix} V_1 \\ V_2 \\ i_E \\ i_L \\ V_3 \\ i_{R1} \end{bmatrix}
=
\begin{bmatrix} E \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
$$