Parallel Numerics, WT 2015/2016

4 Sparse Matrices

Contents

4.1. General Properties of Sparse Matrices

- Full $n \times n$-matrix: storage $\mathcal{O}(n^2)$, solution $\mathcal{O}(n^3) \rightarrow$ too costly for most applications, esp. for fine discretization (large n)

4.2. General Properties of Sparse Matrices

- Full $n \times n$-matrix: storage $\mathcal{O}(n^2)$, solution $\mathcal{O}(n^3) \rightarrow$ too costly for most applications, esp. for fine discretization (large n)
- Formulate given problem in clever way that leads to a linear system that is sparse: $\mathcal{O}(n)$, solution $\mathcal{O}(n)$?
  - (that is structured: storage $\mathcal{O}(n)$, solution $O(n\log(n))$, e.g., FFT)
  - (that is dense, but reduced from, e.g., 3D to 2D)
  - (based on sparse grids)
  - (based on tensor approximations)

4.3. General Properties of Sparse Matrices

- Full $n \times n$-matrix: storage $\mathcal{O}(n^2)$, solution $\mathcal{O}(n^3) \rightarrow$ too costly for most applications, esp. for fine discretization (large n)
- Formulate given problem in clever way that leads to a linear system that is sparse: $\mathcal{O}(n)$, solution $\mathcal{O}(n)$?
  - (that is structured: storage $\mathcal{O}(n)$, solution $O(n\log(n))$, e.g., FFT)
  - (that is dense, but reduced from, e.g., 3D to 2D)
  - (based on sparse grids)
  - (based on tensor approximations)
- Examples:
  - tridiagonal matrix
  - banded matrix
  - block band matrix

Sparse Matrix Example

$$\begin{pmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{pmatrix}$$

Additionally we need to store:

- the size of the matrix $n = 5$
- the number of nonzero entries $\text{nnz} = 12$

4.3.1. Storage in Coordinate Form

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

To store:

- n
- nnz
- $2 \cdot$ nnz integer for row and column indices in JR and JC
- nnz float in AA

4.3.2. Storage in Coordinate Form

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

To store:

- n
- nnz
- $2 \cdot$ nnz integer for row and column indices in JR and JC
- nnz float in AA

No sorting included. Redundant information.

Storage in Coordinate Form (cont.)

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

Storage in Coordinate Form (cont.)

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

Pseudocode for computing $c = A \cdot b$:

$c = 0;$
`for` $j = 1 \ : \ \text{nnz}(A)$
$\quad c_{JR(j)} = c_{JR(j)} + \underbrace{AA(j)}_{A_{JR(j),JC(j)}} * b_{JC(j)};$
`end`

Storage in Coordinate Form (cont.)

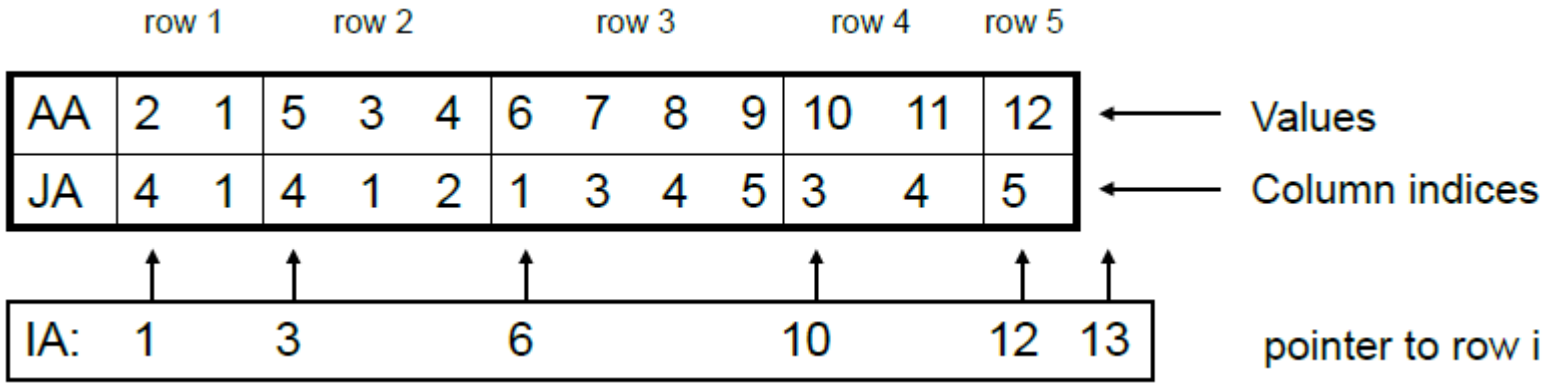
values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

Pseudocode for computing $c = A \cdot b$:

$c = 0;$
`for` $j = 1 \ : \ \text{nnz}(A)$
$\quad c_{JR(j)} = c_{JR(j)} + \underbrace{AA(j)}_{A_{JR(j),JC(j)}} * b_{JC(j)};$
`end`

- Disadvantage: Indirect addressing (indexing) in vector c and $b \rightarrow$ jumps in memory
- Advantage: No difference between columns and rows (A and A^T), simple.

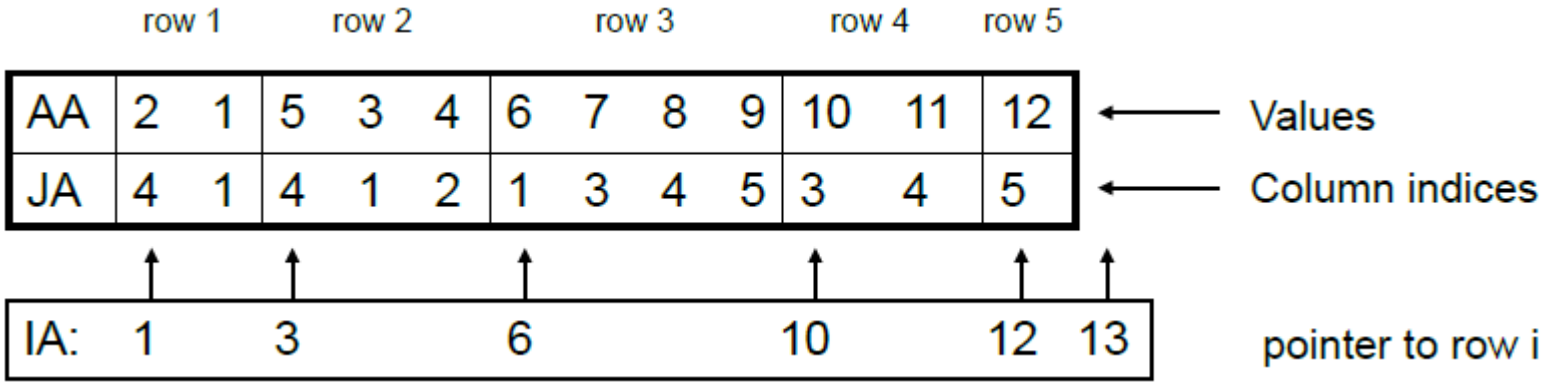
4.3.3. Compressed Sparse Row Format: CSR



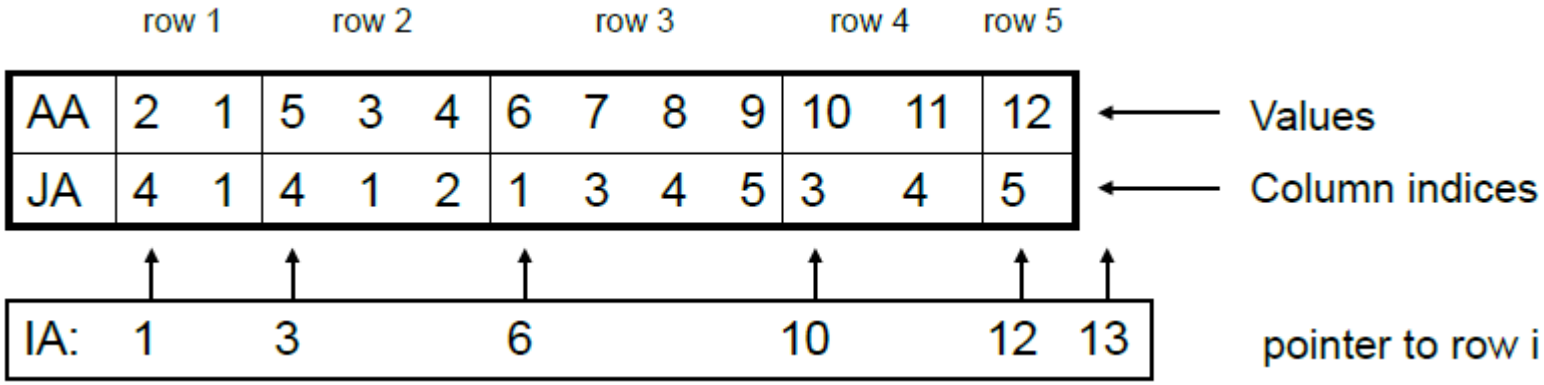
Storage:

- n and nnz
- $n + \text{nnz} + 1$ integer
- nnz float

Compressed Sparse Row: CSR (cont.)



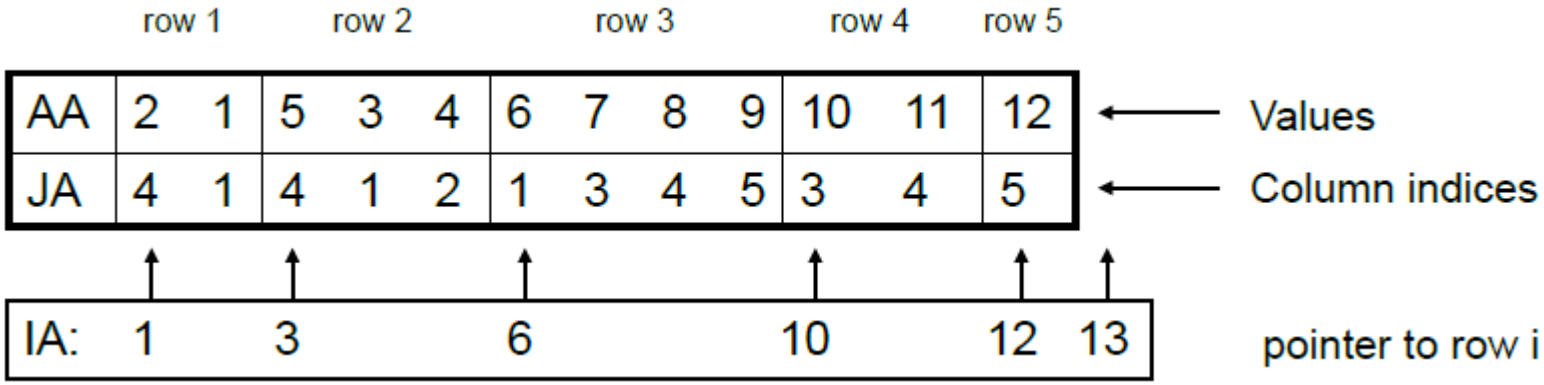
Compressed Sparse Row: CSR (cont.)



Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for i = 1 : n
   for j = IA(i) : IA(i + 1) − 1
      c_i = c_i + AA(j) * b_JA(j);
   end
end
```

Compressed Sparse Row: CSR (cont.)

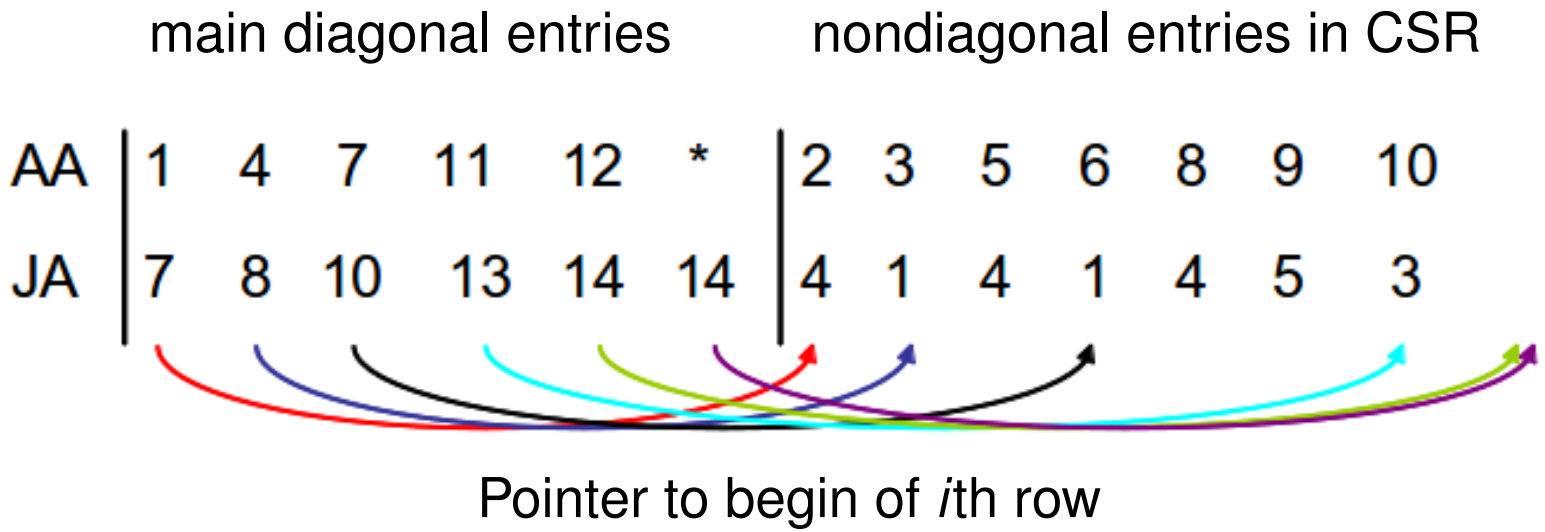


Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for i = 1 : n
   for j = IA(i) : IA(i+1) - 1
      c_i = c_i + AA(j) * b_JA(j);
   end
end
```

- Indirect addressing only in b.
- Columnwise $\rightarrow$ compressed sparse column format.

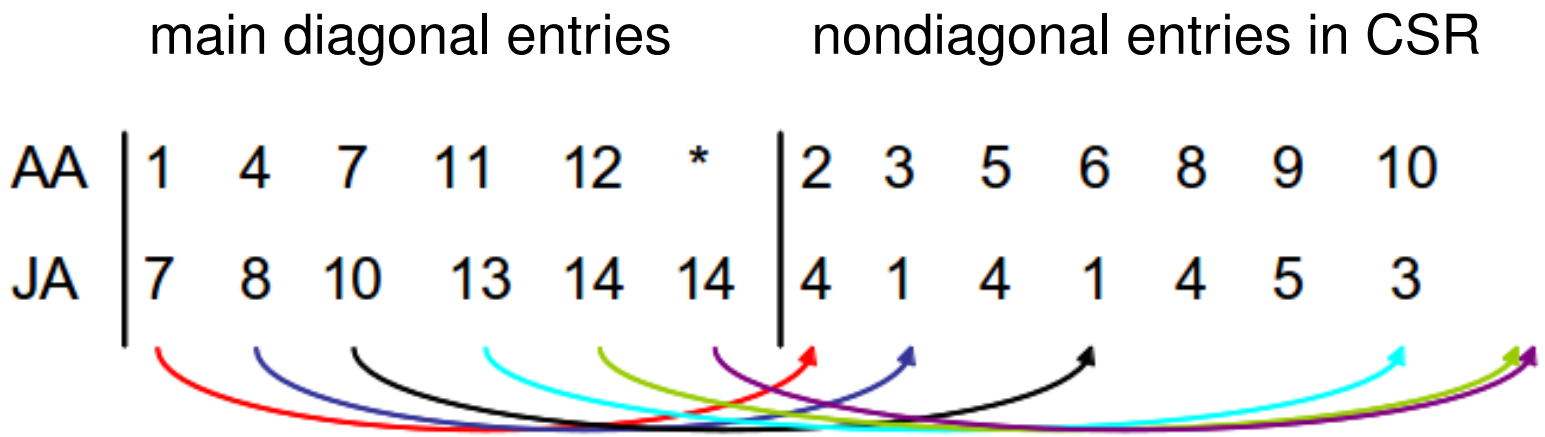
4.3.4. CSR with Extracted Main Diagonal



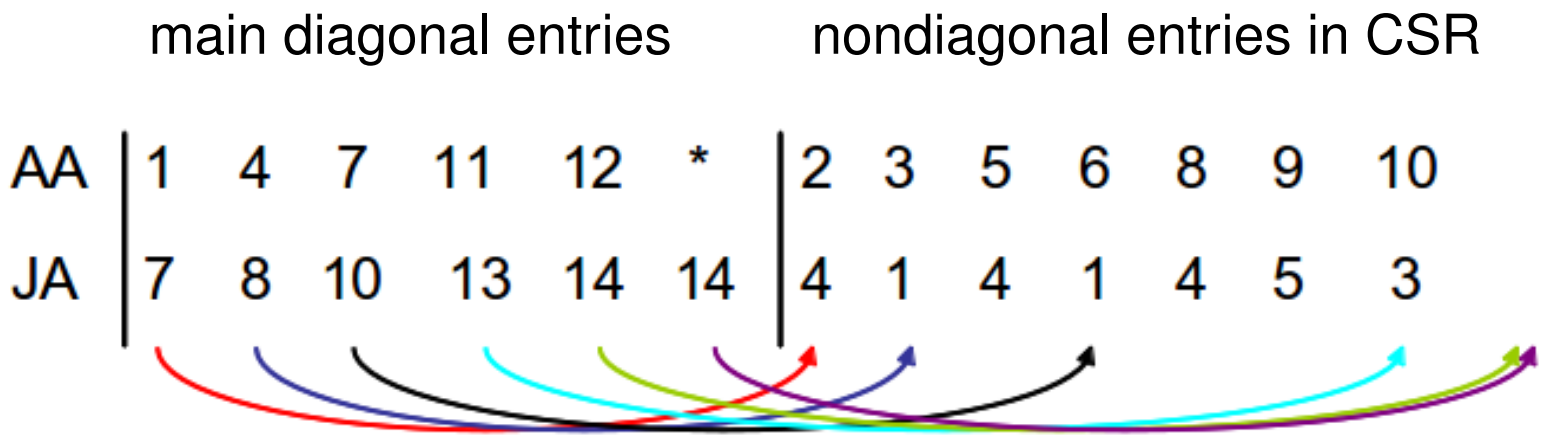
Storage:

- n and nnz
- nnz $+1$ integer
- nnz $+1$ float

CSR with Extracted Main Diagonal (cont.)



CSR with Extracted Main Diagonal (cont.)



Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for i = 1 : n
   c_i = AA_i * b_i;
   for j = JA(i) : JA(i+1) - 1
      c_i = c_i + AA_j * b_JA(j);
   end
end
```

4.3.5. Diagonalwise Storage

$$\begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix}$$

New matrix A!
Different matrix to slides before!

Diagonal numbers: $-1 \;\; 0 \;\; 2$

Values in:

$$\text{DIAG} = \begin{pmatrix} * & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & * \\ 11 & 12 & * \end{pmatrix}, \quad \text{IOFF} = (-1 \;\; 0 \;\; 2)$$

Storage: n, $nd := $ #diagonals, nd integers for IOFF and $n \cdot nd$ float

4.3.6. Rectangular Storage Scheme by Pressing from the Right

$$\left(\begin{array}{ccccc} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{array}\right| \leftarrow \text{pressing from right}$$

gives

$$\text{COEF} = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 0 \\ 11 & 12 & 0 \end{pmatrix} \qquad \text{JCOEF} = \begin{pmatrix} 1 & 3 & * \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 3 & 4 & * \\ 4 & 5 & * \end{pmatrix}$$

Storage: n, $n \cdot nl$ integer and float ($nl :=$ nnz of longest row)

Rectangular Storage Scheme by Pressing from the Right (cont.)

Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for i = 1 : n
   for j = 1 : nl
      c_i = c_i + COEF(i,j) * b(JCOEF(i,j));
   end
end
```

This format was used in ELLPACK (package of subroutines for elliptic PDEs).

4.3.7. Jagged Diagonal Form

Prestep: Sort rows after their length. Long rows first.

$$A = \begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix} \Rightarrow PA = \begin{pmatrix} \boxed{3} & \boxed{4} & 0 & \boxed{5} & 0 \\ 0 & \boxed{6} & \boxed{7} & 0 & \boxed{8} \\ \boxed{1} & 0 & \boxed{2} & 0 & 0 \\ 0 & 0 & \boxed{9} & \boxed{10} & 0 \\ 0 & 0 & 0 & \boxed{11} & \boxed{12} \end{pmatrix} \begin{matrix} \} \text{ Length 3} \\ \\ \} \text{ Length 2} \\ \\ \\ \end{matrix}$$

- Values of PA: DJ = (3 6 1 9 11 | 4 7 2 10 12 | 5 8|)
- Column indices: JDIAG = (1 2 1 3 4 | 2 3 3 4 5 | 4 5|)
- Pointer to beginning of jth diagonal: IDIAG = (1 6 11 13)

Jagged Diagonal Form (cont.)

NDIAG := number of jagged diagonals

Storage: n, NDIAG, nnz float, nnz + NDIAG integer

Jagged Diagonal Form (cont.)

NDIAG := number of jagged diagonals

Storage: n, NDIAG, nnz float, nnz + NDIAG integer

Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for j = 1 : NDIAG
   for i = 1 : IDIAG(j+1) - IDIAG(j)
      k = IDIAG(j) + i - 1;
      c_i = c_i + DJ(k) * b(JDIAG(k));
   end
end
```

Jagged Diagonal Form (cont.)

NDIAG := number of jagged diagonals

Storage: n, NDIAG, nnz float, nnz + NDIAG integer

Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for j = 1 : NDIAG
   for i = 1 : IDIAG(j+1) - IDIAG(j)
      k = IDIAG(j) + i - 1;
      c_i = c_i + DJ(k) * b(JDIAG(k));
   end
end
```

- Always start with row 1.
- More operations on neighboring data.
- Less indirect addressing.
- Pre-permutation changes only rows. Can be done implicitly.

Survey on Sparse Storage Formats

Coordinate form and CSR traditional way to specify sparse matrix in MATLAB.

	global int	idx int	value floats
Coordinate	2	$2 \cdot \text{nnz}$	nnz
CSR	2	$n + \text{nnz} + 1$	nnz
CSR with Extract. Diag.	2	$\text{nnz} + 1$	$\text{nnz} + 1$
Diagonalwise	2	nd	$n \cdot nd$
Rectang. with Pressing	1	$n \cdot nl$	$n \cdot nl$
Jagged Diagonal	2	$\text{nnz} + nd$	nnz

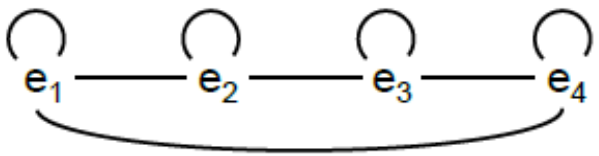
4.4. Sparse Matrices and Graphs

4.4.1. Graph $G(A)$ for symmetric positive definite (SPD)

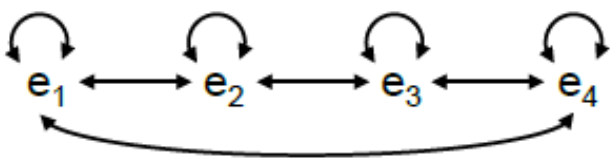
$A = A^T > 0$

$n \times n$-matrix: vertices $e_1, \ldots, e_n$ with edges (e_i, e_k) for $a_{ik} \neq 0$, undirected Graph

$$A = \begin{pmatrix} * & * & 0 & * \\ * & * & * & 0 \\ 0 & * & * & * \\ * & 0 & * & * \end{pmatrix} \rightarrow G(A) :$$



G(A) as directed graph:



Adjacency Matrix for $G(A)$ or A

- Can be obtained directly by replacing in A each nonzero by 1.

$$\mathcal{A}(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Adjacency Matrix for $G(A)$ or A

- Can be obtained directly by replacing in A each nonzero by 1.

$$\mathcal{A}(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

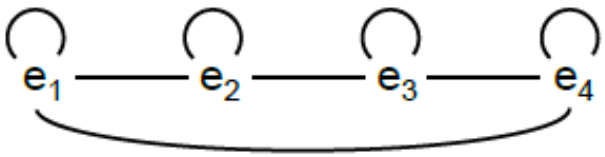
- Symmetric permutations of A in the form PAP^T change the ordering of the rows and columns of A simultaneously.
- Therefore, the graph of PAP^T can be obtained by the graph of A by renumbering the vertices.

Matrix A with graph G(A) (cont.)

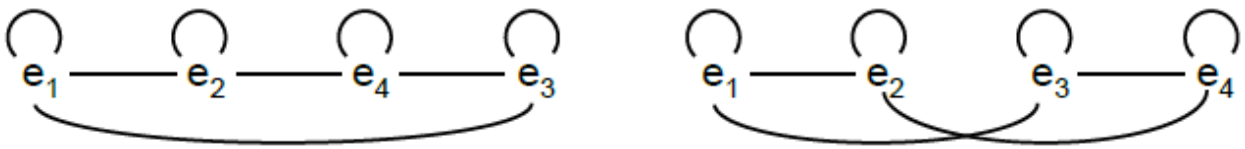
Symmetric permutation PAP^T with graph $G(PAP^T)$.

Example: P permutation that changes $3 \leftrightarrow 4$:

$G(A)$



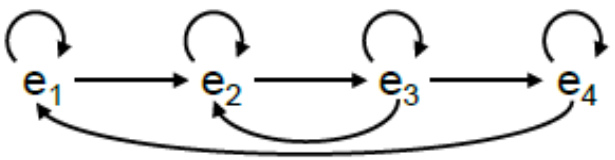
$G(PAP^T)$



4.4.2. Matrix A nonsymmetric, $G(A)$ directed

$$A = \begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & * & * & * \\ * & 0 & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

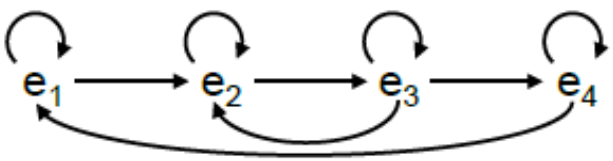
G(A):



4.4.3. Matrix A nonsymmetric, $G(A)$ directed

$$A = \begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & * & * & * \\ * & 0 & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

G(A):



Questions:
How can we characterize “good“ sparsity patterns?

”good”: Gaussian elimination can be reduced to smaller subproblems or produces no (or small) fill-in, or is easy to parallelize.

Block Diagonal Pattern

New matrix A!

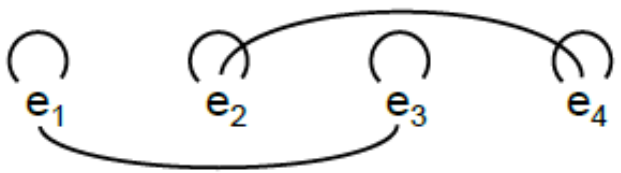
$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Block Diagonal Pattern

New matrix A!

$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

G(A):

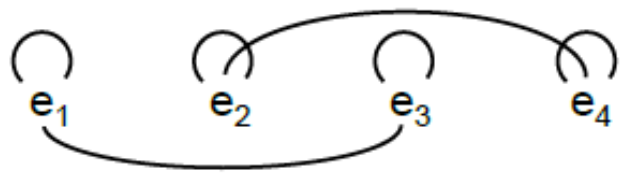


Block Diagonal Pattern

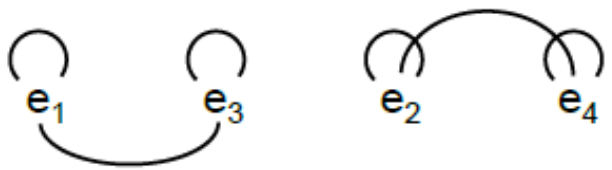
New matrix A!

$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

G(A):



$2 \leftrightarrow 3$:



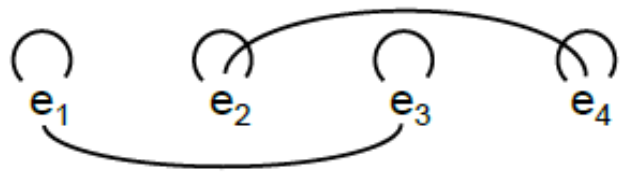
$$PAP^T = \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

Block Diagonal Pattern

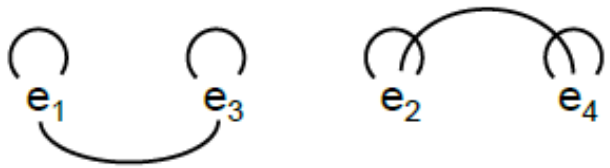
New matrix A!

$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

G(A):



$2 \leftrightarrow 3$:

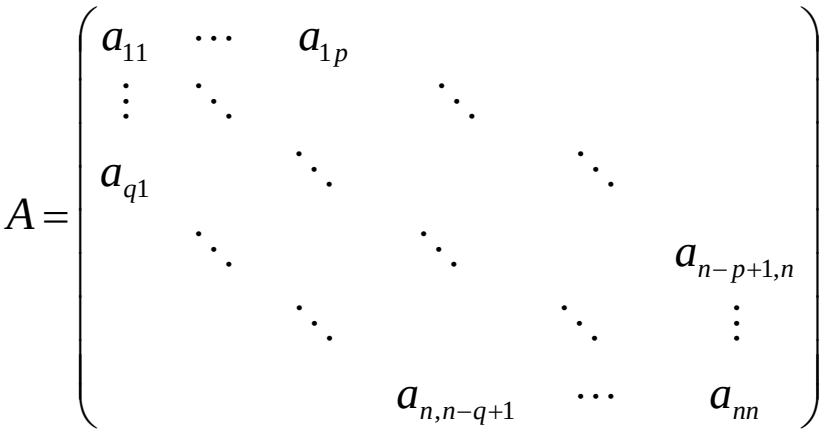


$$PAP^T = \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

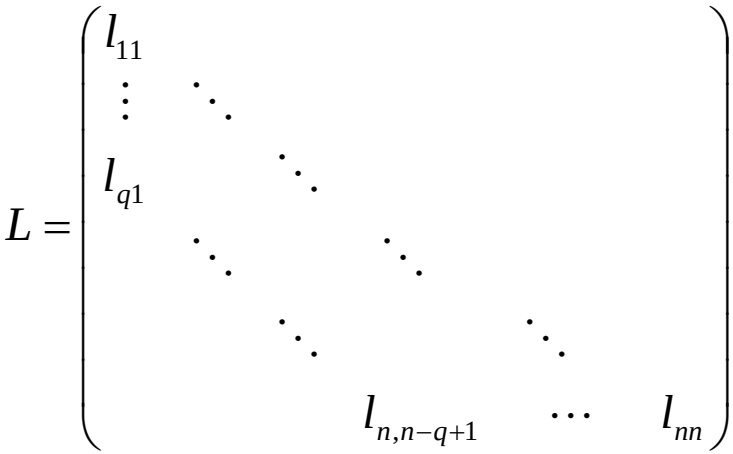
By this permutation, A can be transformed into block diagonal form $\rightarrow$ easy to solve, fully parallel!

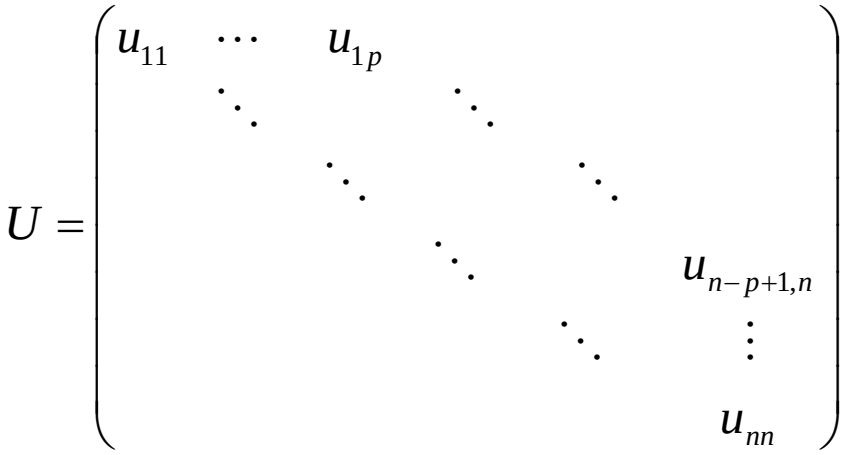
$$\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}^{-1} = \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix}$$

Banded Pattern



Gauss Elimination without pivoting preserves the sparsity pattern

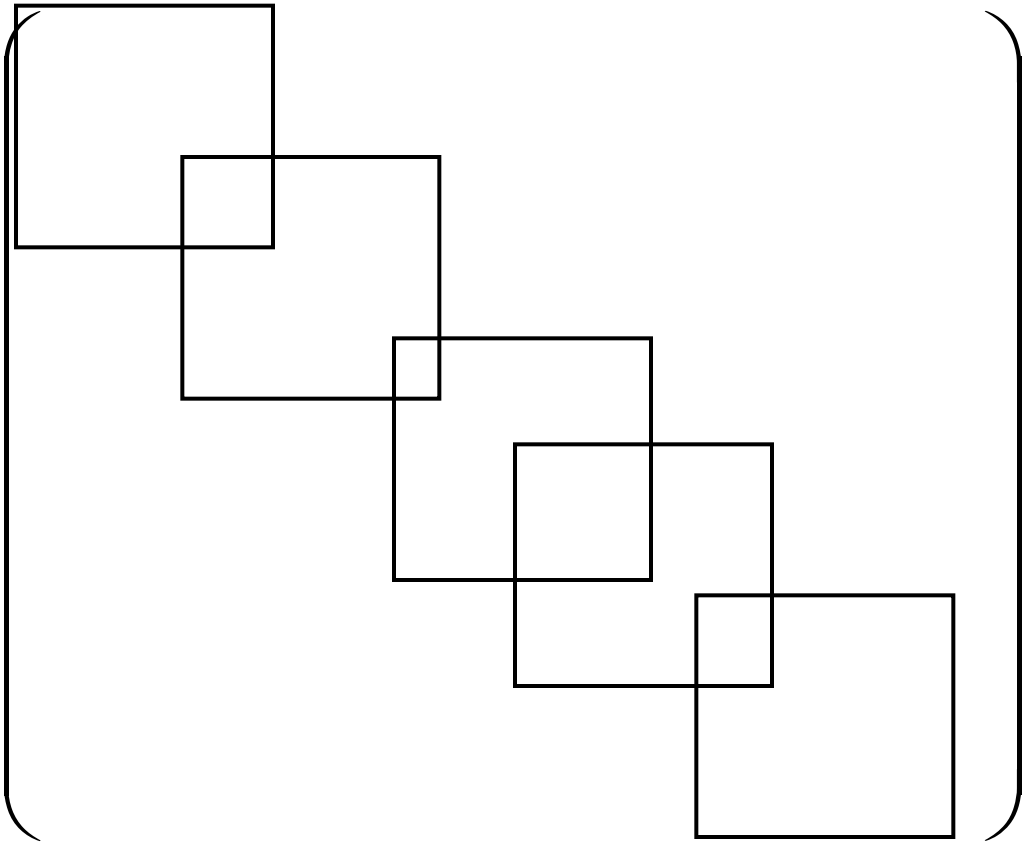




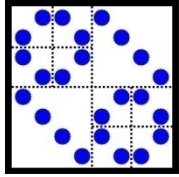
With pivoting the bandwidth in U grows, but remains <= p+q.

Overlapping Block Diagonal

$A =$



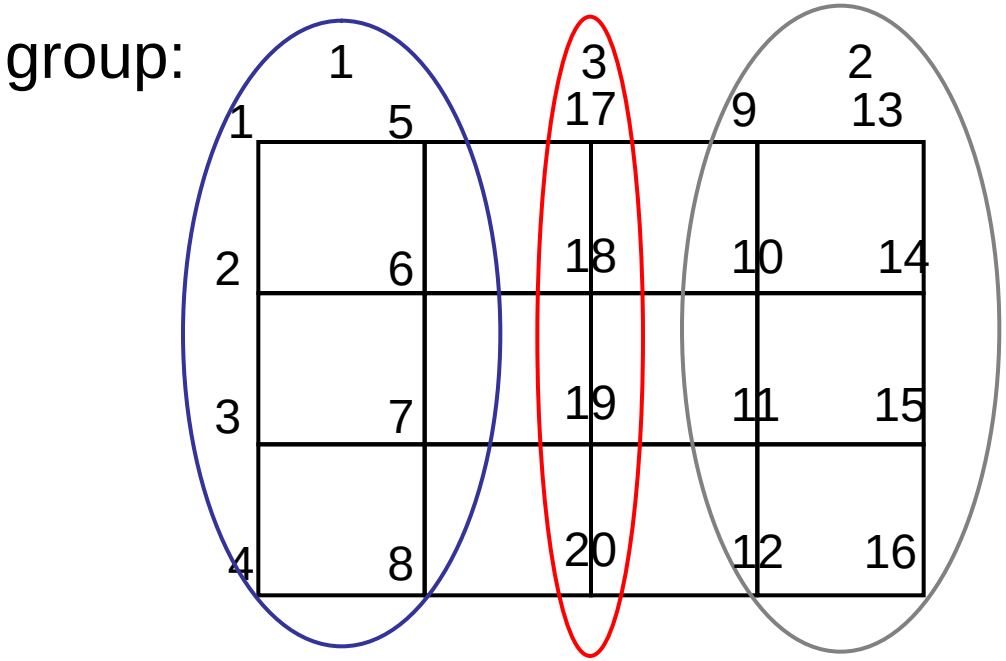
Pattern is preserved by Gaussian Elimination (in case of restricted pivoting).



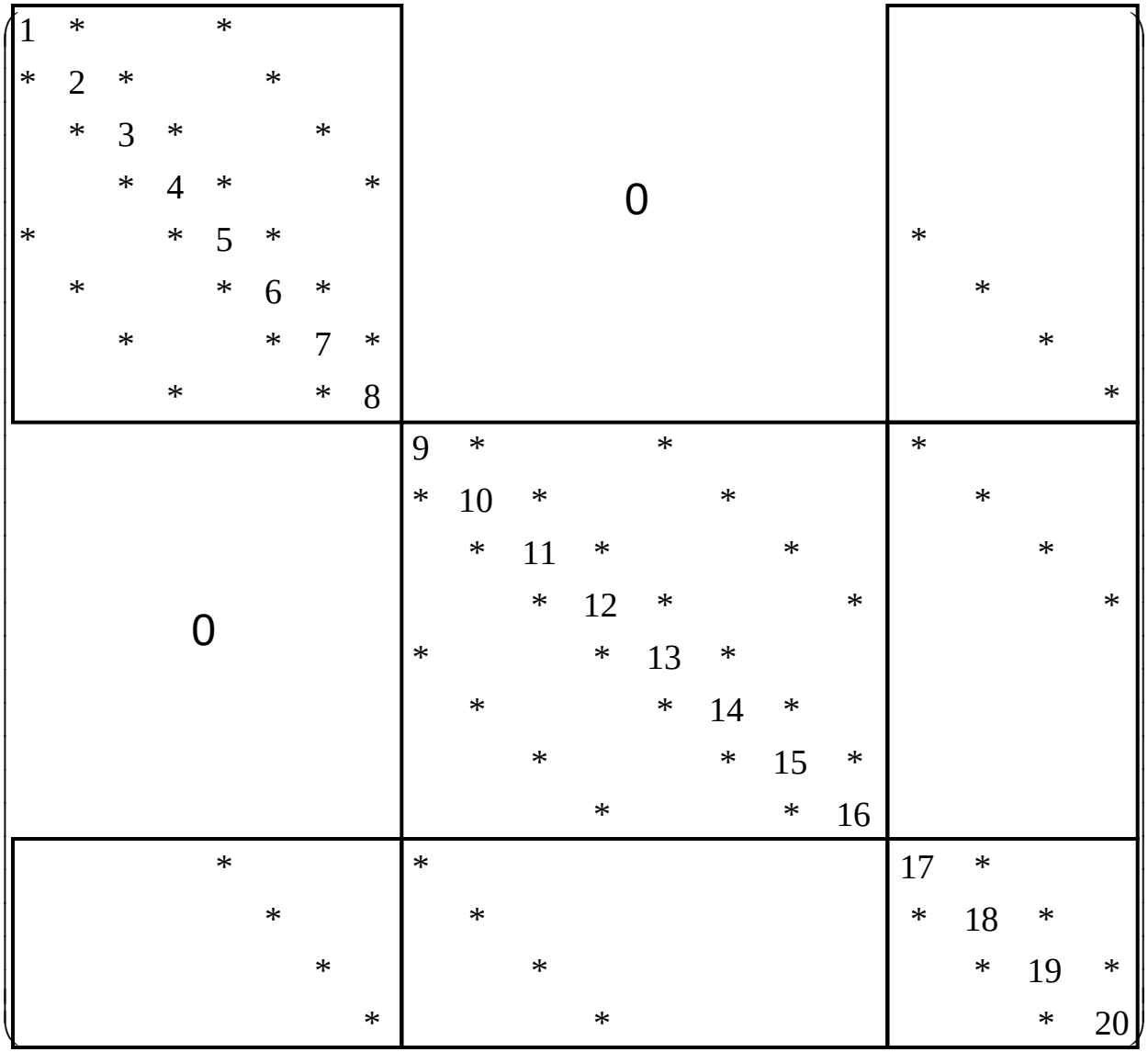
4.3.2 Dissection Reordering

Consider matrix A with graph G(A):

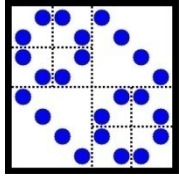
Numbering of unknows in two groups separated by third group



This numbering leads to sparsity pattern:



Leads automatically to Dissection form:



$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix}$$

Can be solved e.g. based on Schur Complement.

General idea:

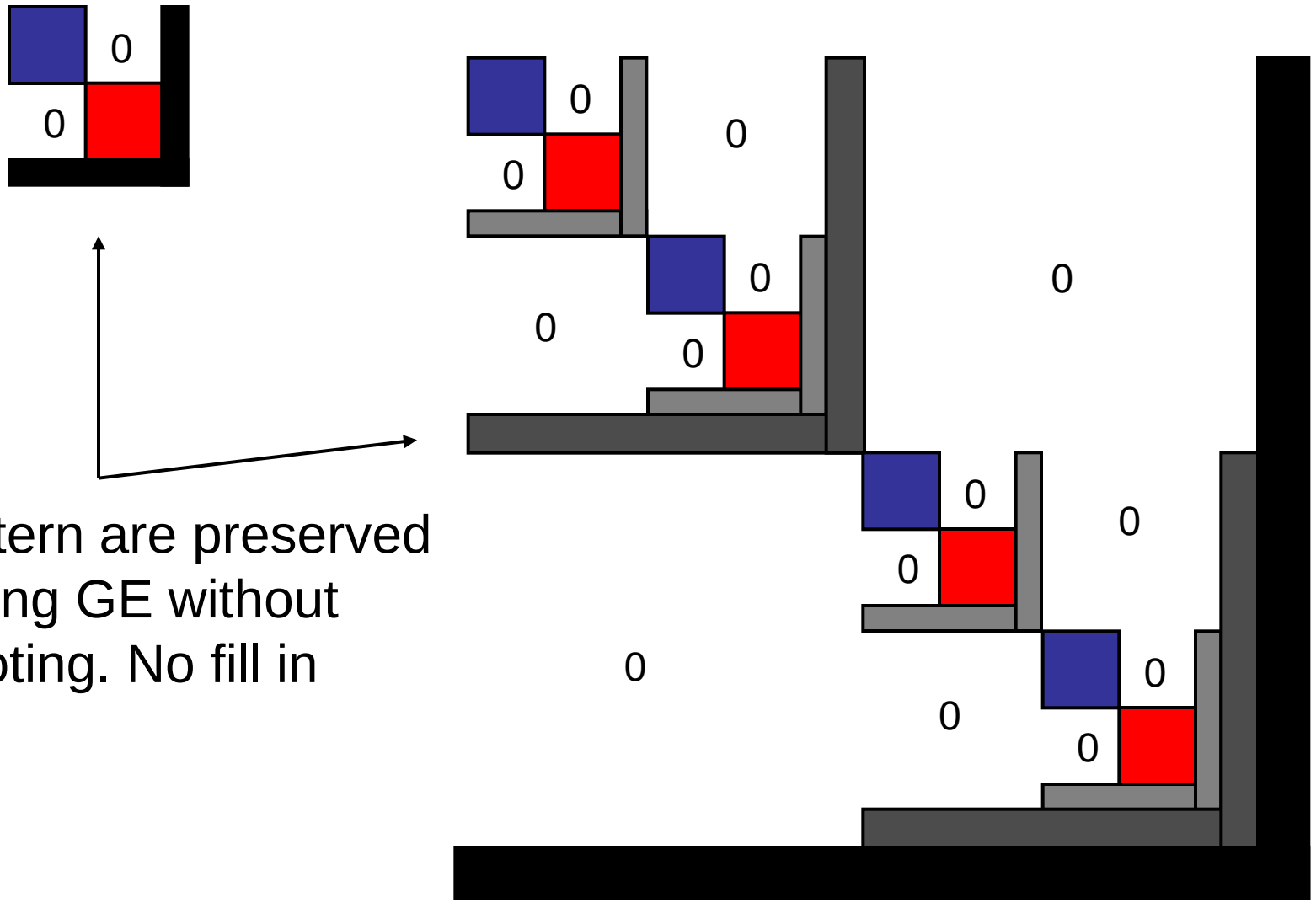
Cut G(A) by separator between unconnected subgraphs.
Separator is numbered last.

Repeat recursively → Nested Dissection form

Looking for partitioning of the graph with minimum connections!

Dissection Form

Nested (recursive) dissection:



Pattern are preserved during GE without pivoting. No fill in

Schur Complement Reduction

Write matrix B in terms of smaller submatrices:

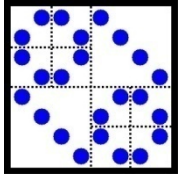
$$\begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \cdot \begin{pmatrix} B_1^{-1} & D \\ 0 & S^{-1} \end{pmatrix} = \begin{pmatrix} I & B_1 D + B_2 S^{-1} \\ B_3 B_1^{-1} & B_3 D + B_4 S^{-1} \end{pmatrix} \overset{!}{=} \begin{pmatrix} I & 0 \\ * & I \end{pmatrix}$$

To satisfy this equation we have to set:

$$B_1 D + B_2 S^{-1} = 0 \Rightarrow D = -B_1^{-1} B_2 S^{-1}$$

$$B_3 D + B_4 S^{-1} = I \Rightarrow I = -B_3 B_1^{-1} B_2 S^{-1} + B_4 S^{-1}$$

$$\Rightarrow \underbrace{S = B_4 - B_3 B_1^{-1} B_2}_{\text{S Schur Complement}} \quad and \quad D = -B_1^{-1} B_2 S^{-1}$$



$$B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} = \begin{pmatrix} I & 0 \\ B_3 B_1^{-1} & I \end{pmatrix} \cdot \begin{pmatrix} B_1 & B_2 \\ 0 & S \end{pmatrix}$$

Therefore, solving linear system in B is reduced to solving two smaller linear systems, one in B_1 and the other in the Schur complement S.

B sparse → B_1 also sparse, but S usually dense!

Example: Schur complement and dissection form:

$$A = \left(\begin{array}{cc|c} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ \hline G_1 & G_2 & A_3 \end{array}\right)$$

Schur complement:

$$S = A_3 - \begin{pmatrix} G_1 & G_2 \end{pmatrix} \cdot \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} =$$

$$= A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2$$

Direct derivation of Schur complement:

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \Rightarrow \begin{array}{lllll} A_1 x_1 + & & F_1 x_3 & = & b_1 \\ & A_2 x_2 + & F_2 x_3 & = & b_2 \\ G_1 x_1 + & G_2 x_2 + & A_3 x_3 & = & b_3 \end{array}$$

$$x_1 = A_1^{-1} b_1 - A_1^{-1} F_1 x_3$$

$$\Rightarrow$$

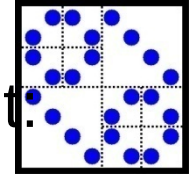
$$x_2 = A_2^{-1} b_2 - A_2^{-1} F_2 x_3$$

$$\Rightarrow (G_1 A_1^{-1} b_1 - G_1 A_1^{-1} F_1 x_3) + (G_2 A_2^{-1} b_2 - G_2 A_2^{-1} F_2 x_3) + A_3 x_3 = b_3$$

$$\Rightarrow (A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2) x_3 = b_3 - G_1 A_1^{-1} b_1 - G_2 A_2^{-1} b_2$$

$$\Rightarrow S x_3 = \tilde{b}_3$$

Algorithm for solving Ax=b based on Schur complement:



1. Compute S by using inv(A_1) and inv(A_2)

2. Solve $Sx_3 = \tilde{b}_3$

3. Compute x_1 and x_2 by using inv(A_1) and inv(A_2)

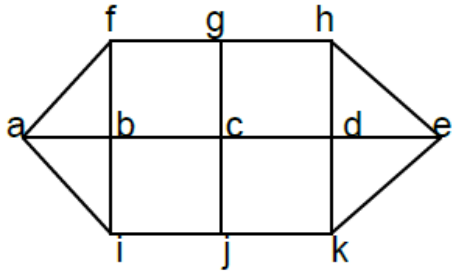
The explicit computation of S can be avoided by solving the linear System in S iteratively, e.g. Jacobi, pcg, ….
Then we need only part of S and in every iteration step we have to compute S * intermediate vector.

To achieve fast convergence, a preconditioner (approximation) for S has to be used!

Iterative methods and preconditioning will be subject of later chapters.

4.5. Reordering

Consider sparse matrix A with graph G(A):

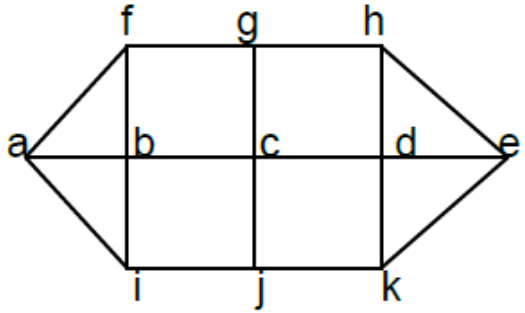


Consider following matrix (in tabular form) with bandwidth 9:

	a	b	c	d	e	f	g	h	i	j	k
a	a	∗				∗			∗		
b	∗	b	∗			∗			∗		
c		∗	c	∗			∗			∗	
d			∗	d	∗			∗			∗
e				∗	e			∗			∗
f	∗	∗				f	∗				
g			∗			∗	g	∗			
h				∗	∗		∗	h			
i	∗	∗							i	∗	
j			∗						∗	j	∗
k				∗	∗					∗	k

4.5.1. Cuthill McKee

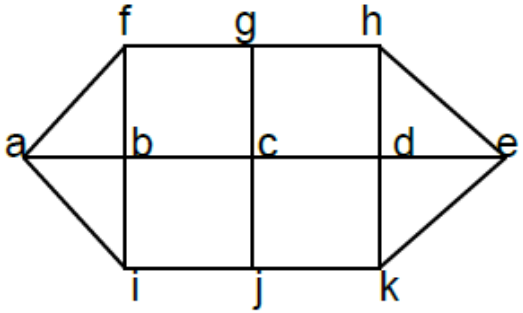
Graph G(A):



Optimal ordering that leads to small bandwidth?

4.5.2. Cuthill McKee

Graph G(A):



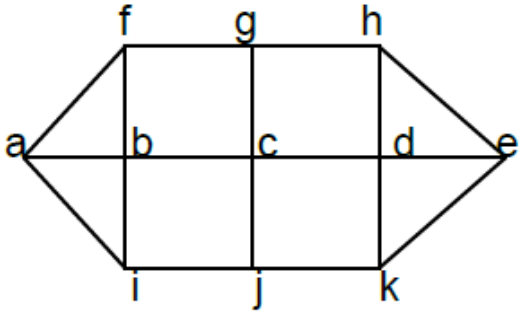
Optimal ordering that leads to small bandwidth?

Level sets, starting with:

$$S_1 := \{a\}$$

4.5.3. Cuthill McKee

Graph G(A):



Optimal ordering that leads to small bandwidth?

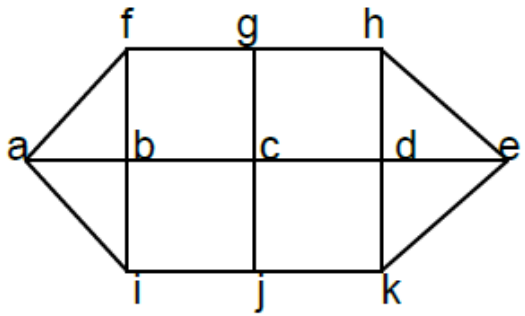
Level sets, starting with:

$S_1 := \{a\}$

$S_2 := \{f, b, i\}$, all vertices directly connected with S_1

4.5.4. Cuthill McKee

Graph G(A):



Optimal ordering that leads to small bandwidth?

Level sets, starting with:

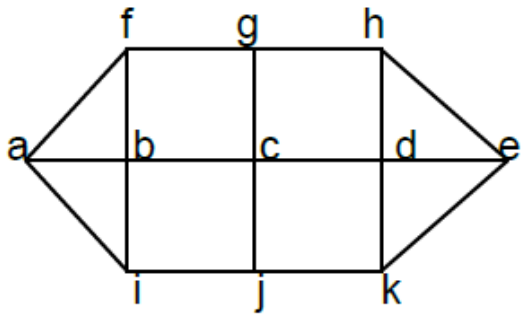
$S_1 := \{a\}$

$S_2 := \{f, b, i\}$, all vertices directly connected with S_1

$S_3 := \{g, c, j\}$, all vertices directly connected with S_2

4.5.5. Cuthill McKee

Graph G(A):



Optimal ordering that leads to small bandwidth?

Level sets, starting with:

$S_1 := \{a\}$

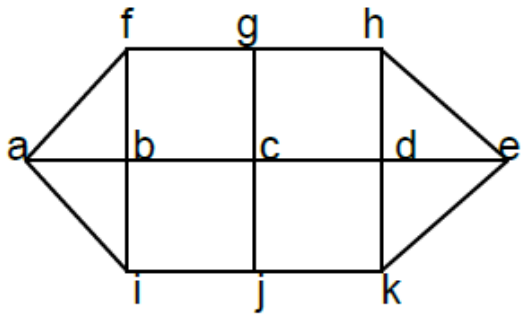
$S_2 := \{f, b, i\}$, all vertices directly connected with S_1

$S_3 := \{g, c, j\}$, all vertices directly connected with S_2

$S_4 := \{h, d, k\}$, all vertices directly connected with S_3

4.5.6. Cuthill McKee

Graph G(A):



Optimal ordering that leads to small bandwidth?

Level sets, starting with:

$S_1 := \{a\}$

$S_2 := \{f, b, i\}$, all vertices directly connected with S_1

$S_3 := \{g, c, j\}$, all vertices directly connected with S_2

$S_4 := \{h, d, k\}$, all vertices directly connected with S_3

$S_5 := \{e\}$

Cuthill McKee (cont.)

1. First ordering by level sets

Cuthill McKee (cont.)

1. First ordering by level sets
2. Inside level sets order the vertices such that first group of indices in S_i are neighbors of first vertex in S_{i-1}

Cuthill McKee (cont.)

1. First ordering by level sets
2. Inside level sets order the vertices such that first group of indices in S_i are neighbors of first vertex in S_{i-1}
3. If there is choice left: number indices with small degree first!

Cuthill McKee (cont.)

1. First ordering by level sets
2. Inside level sets order the vertices such that first group of indices in S_i are neighbors of first vertex in S_{i-1}
3. If there is choice left: number indices with small degree first!

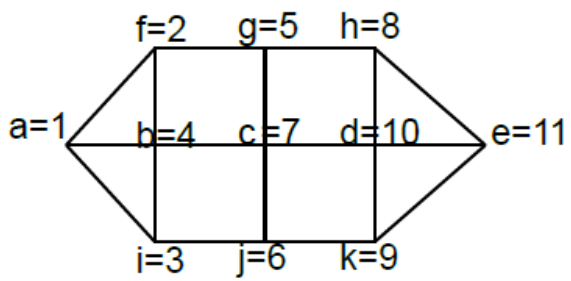
$S_1 := \{a\}$

$S_2 := \{f = 2, i = 3, b = 4\}$, (could also be different!)

$S_3 := \{g = 5, j = 6, c = 7\}$, (as we start with the neighbors of $f = 2$, then i, and then b)

$S_4 := \{h = 8, k = 9, d = 10\}$, (as we start with neighbors of g)

$S_5 := \{e = 11\}$



Cuthill McKee (cont. 2)

- Resulting matrix with small bandwidth 4:

$$\begin{pmatrix}
1 & * & * & * & & & & & & & \\
* & 2 & 0 & * & * & & & & & & \\
* & 0 & 3 & * & 0 & * & & & & & \\
* & * & * & 4 & 0 & 0 & * & & & & \\
 & * & 0 & 0 & 5 & 0 & * & * & & & \\
 & & * & 0 & 0 & 6 & * & 0 & * & & \\
 & & & * & * & * & 7 & 0 & 0 & * & \\
 & & & & * & 0 & 0 & 8 & 0 & * & * \\
 & & & & & * & 0 & 0 & 9 & * & * \\
 & & & & & & * & * & * & 10 & * \\
 & & & & & & & * & * & * & 11
\end{pmatrix}$$

Cuthill McKee (cont. 2)

- Resulting matrix with small bandwidth 4:

$$\begin{pmatrix}
1 & * & * & * & & & & & & & \\
* & 2 & 0 & * & * & & & & & & \\
* & 0 & 3 & * & 0 & * & & & & & \\
* & * & * & 4 & 0 & 0 & * & & & & \\
 & * & 0 & 0 & 5 & 0 & * & * & & & \\
 & & * & 0 & 0 & 6 & * & 0 & * & & \\
 & & & * & * & * & 7 & 0 & 0 & * & \\
 & & & & * & 0 & 0 & 8 & 0 & * & * \\
 & & & & & * & 0 & 0 & 9 & * & * \\
 & & & & & & * & * & * & 10 & * \\
 & & & & & & & * & * & * & 11
\end{pmatrix}$$

- Often „Reverse Cuthill Mckee“ gives slightly better results: reversing the ordering of Cuthill McKee: $11 \leftrightarrow 1$, $10 \leftrightarrow 2, \ldots$ Choose start as extreme vertex (diameter)!
- This method is far from being optimal and strongly heuristical.