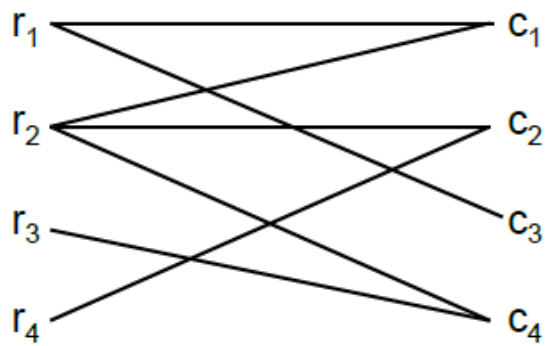
4.5.8. Perfect Matching Reordering

- Find row permutation of A such that elements with largest absolute values are located at diagonal.
- Advantage: No pivoting necessary (also in the indefinite case).

4.5.9. Perfect Matching Reordering

- Find row permutation of A such that elements with largest absolute values are located at diagonal.
- Advantage: No pivoting necessary (also in the indefinite case).
- Describe the sparsity pattern of A by bipartite graph $G = (V_r, V_c, E)$ with $V_r = \{r_1, \ldots, r_n\}$ and $V_c = \{c_1, \ldots, c_n\}$.
- Vertices r_i and c_j are connected by edge $\leftrightarrow$ $a_{i,j} \neq 0$:

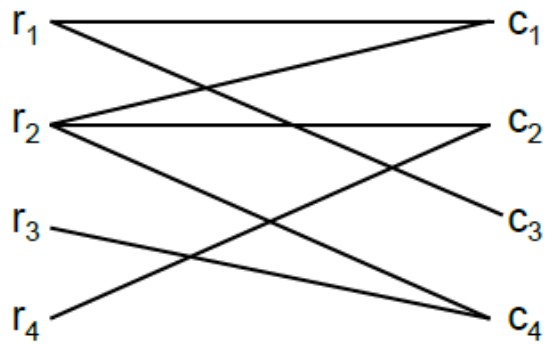
$$A = \begin{pmatrix} * & 0 & * & 0 \\ * & * & 0 & 0 \\ 0 & 0 & 0 & * \\ 0 & * & 0 & 0 \end{pmatrix}$$



4.5.10. Perfect Matching Reordering

- Find row permutation of A such that elements with largest absolute values are located at diagonal.
- Advantage: No pivoting necessary (also in the indefinite case).
- Describe the sparsity pattern of A by bipartite graph $G = (V_r, V_c, E)$ with $V_r = \{r_1, \ldots, r_n\}$ and $V_c = \{c_1, \ldots, c_n\}$.
- Vertices r_i and c_j are connected by edge $\leftrightarrow$ $a_{i,j} \neq 0$:

$$A = \begin{pmatrix} * & 0 & * & 0 \\ * & * & 0 & 0 \\ 0 & 0 & 0 & * \\ 0 & * & 0 & 0 \end{pmatrix}$$

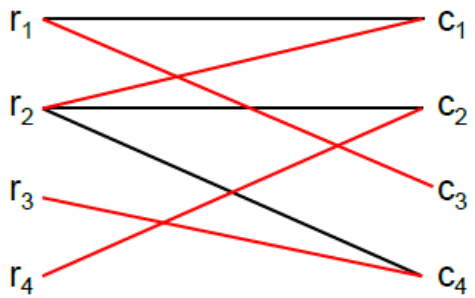


For each column we want to choose a large entry $\rightarrow$ diagonal.

Perfect Matching (cont.)

- Matching is subset of edges such that each row vertex is connected exactly to one column vertex and vice versa. Bijective mapping between V_r and V_c.

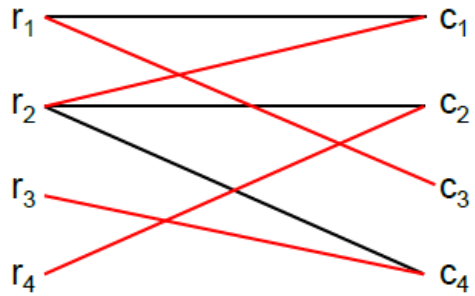
Matching in our example:



Perfect Matching (cont.)

- Matching is subset of edges such that each row vertex is connected exactly to one column vertex and vice versa. Bijective mapping between V_r and V_c.

Matching in our example:



- This matching induces row permutation to permute related entries $a_{1,3}$, $a_{2,1}$, $a_{3,4}$, and $a_{4,2}$ on the diagonal positions.

Permutation:
$1 \rightarrow 3, 2 \rightarrow 1, 3 \rightarrow 4, 4 \rightarrow 2$

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix}$$

$$(1 \quad 3 \quad 4 \quad 2)$$

$$A = \begin{pmatrix} * & 0 & * & 0 \\ * & * & 0 & * \\ 0 & 0 & 0 & * \\ 0 & * & 0 & 0 \end{pmatrix}$$

Perfect Matching (cont.)

Permutation: $1 \to 3, 2 \to 1, 3 \to 4, 4 \to 2$

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix}$$

$$(1 \quad 3 \quad 4 \quad 2)$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} x_2 \\ x_4 \\ x_1 \\ x_3 \end{pmatrix}$$

Perfect Matching (cont. 2)

- Add additional condition to matching problem: ”Find matching that maximizes a given function”
- Look for a subset of edges $M \leq E$:
  - where each vertex is incident to exactly one edge e in M and
  - where the matched edges maximize a weight function, e.g.

$$w(M) = \sum_{(i,j) \in M} c_{i,j} \qquad \text{with} \qquad c_{i,j} = \begin{cases} \log(|a_{i,j}|) & \text{for } a_{i,j} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Perfect Matching (cont. 2)

- Add additional condition to matching problem: ”Find matching that maximizes a given function”
- Look for a subset of edges $M \leq E$:
  - where each vertex is incident to exactly one edge e in M and
  - where the matched edges maximize a weight function, e.g.

$$w(M) = \sum_{(i,j) \in M} c_{i,j} \qquad \text{with} \qquad c_{i,j} = \begin{cases} \log(|a_{i,j}|) & \text{for } a_{i,j} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Solution of this perfect matching problem leads to permutation of A such that product of the new diagonal entries is maximized (therefore, all the diagonal entries should be large)
- Exact solution to costly, use heuristic approximate solutions!

Perfect Matching: Example

$$A = \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 1 & -2 & 0 & 0 & 110 \\ -1 & 0 & 0 & -120 & 0 \\ 0 & 1 & -80 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & & & & \\ & & & & 1 \\ & & 1 & & \\ & 1 & & & \\ & & & 1 & \end{pmatrix} \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 1 & -2 & 0 & 0 & 110 \\ -1 & 0 & 0 & -120 & 0 \\ 0 & 1 & -80 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \\ 0 & 1 & -80 & 0 & 0 \\ -1 & 0 & 0 & -120 & 0 \\ 1 & -2 & 0 & 0 & 110 \end{pmatrix}$$

Perfect Matching: Example

$$A = \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 1 & -2 & 0 & 0 & 110 \\ -1 & 0 & 0 & -120 & 0 \\ 0 & 1 & -80 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & & & & \\ & & & & 1 \\ & & 1 & & \\ & 1 & & & \\ & & & 1 & \end{pmatrix} \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 1 & -2 & 0 & 0 & 110 \\ -1 & 0 & 0 & -120 & 0 \\ 0 & 1 & -80 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \\ 0 & 1 & -80 & 0 & 0 \\ -1 & 0 & 0 & -120 & 0 \\ 1 & -2 & 0 & 0 & 110 \end{pmatrix}$$

$$A = \begin{pmatrix} 100 & 90 & -100 & 80 & 70 \\ 1 & -2 & 0 & 0 & 10 \\ -1 & 0 & 0 & -20 & 0 \\ 0 & 1 & -8 & 0 & 0 \\ 2 & 9 & -2 & 1 & 1 \end{pmatrix} \rightarrow \; ?$$

Perfect Matching for Symmetric *A*

- Given: Symmetric matrix *A*
- *s*: small entry, *l*: large entry
- Find symmetric reordering that maximizes the diagonal.

$$A = \begin{pmatrix} s & \boxed{l} & & l & & \\ l & & & \boxed{l} & & s \\ & & & & \boxed{l} & \\ \boxed{l} & l & & & s & \\ & & \boxed{l} & s & & \\ & s & & & & \boxed{l} \end{pmatrix}$$

Perfect Matching for Symmetric A (cont.)

- For symmetric A row permutation would destroy symmetry! We need way to move large entries **near** the diagonal and permute rows and columns symmetrically!

Perfect Matching for Symmetric A (cont.)

- For symmetric A row permutation would destroy symmetry! We need way to move large entries **near** the diagonal and permute rows and columns symmetrically!
- Idea: Solve perfect matching unsymmetrically $\rightarrow$ gives permutation.
- Resulting permutation can be written as sequence of cyclic permutations, e.g., we can rewrite the permutation in the following way

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 5 & 1 & 3 & 6 \end{pmatrix} \Rightarrow (1 \quad 2 \quad 4)(3 \quad 5)(6)$$

Perfect Matching for Symmetric A (cont.)

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 5 & 1 & 3 & 6 \end{pmatrix} \Rightarrow (1 \quad 2 \quad 4)(3 \quad 5)(6)$$

Original row permutation gives

$$P := \begin{pmatrix} & 1 & & & & \\ & & & & 1 & \\ & & & 1 & & \\ 1 & & & & & \\ & & 1 & & & \\ & & & & & 1 \end{pmatrix}$$

Perfect Matching for Symmetric A (cont.)

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 5 & 1 & 3 & 6 \end{pmatrix} \Rightarrow (1 \quad 2 \quad 4)(3 \quad 5)(6)$$

Original row permutation gives

$$P := \begin{pmatrix} & 1 & & & & \\ & & & & 1 & \\ & & & 1 & & \\ 1 & & & & & \\ & & 1 & & & \\ & & & & & 1 \end{pmatrix}$$

Reordering of columns in view of cyclic representation gives

Reordering such that the indices inside a cycle are ordered sequentially: $3 \leftrightarrow 4$

$$P_c := \begin{pmatrix} & 1 & & & & \\ & & 1 & & & \\ 1 & & & & & \\ & & & & 1 & \\ & & & 1 & & \\ & & & & & 1 \end{pmatrix}$$

Perfect Matching for Symmetric A: Example

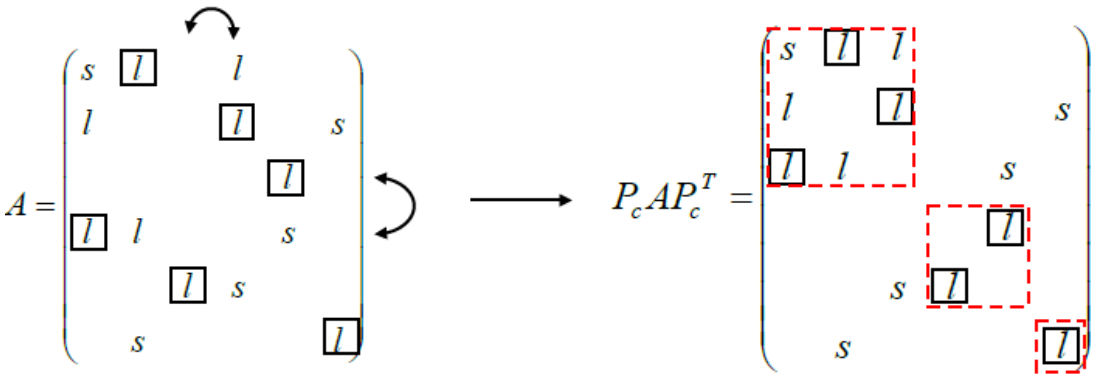
s: small entry, l: large entry, $\boxed{l}$: chosen entry

$$A = \begin{pmatrix} s & \boxed{l} & l & & \\ l & & \boxed{l} & & s \\ & & & \boxed{l} & \\ \boxed{l} & l & & s & \\ & & \boxed{l} & s & \\ & s & & & \boxed{l} \end{pmatrix}$$

- Nonsymmetric perfect matching $P = (1, 2, 4)(3, 5)(6)$
- Apply symmetric permutation that groups together the blocks in P: switch $3 \leftrightarrow 4$.
- Change numbering in A accordingly.
- Then, the nonsymmetric perfect matching would be $P = (1, 2, 3)(4, 5)(6)$
  $\rightarrow$, hence inside the blocks are the largest entries!

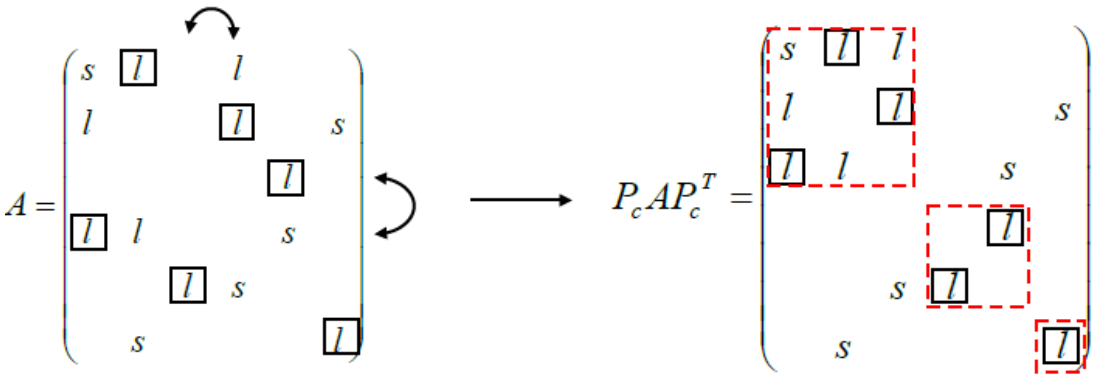
Perfect Matching for Symmetric A: Example (cont.)

P_c from P by changing the order of the columns: $P_c : 3 \leftrightarrow 4$

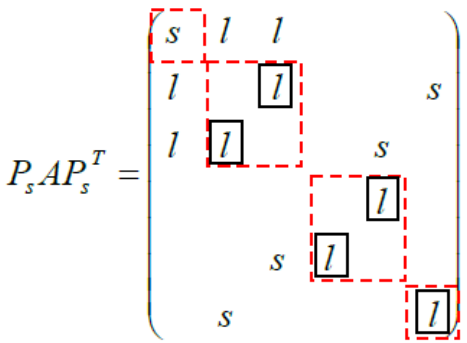


Perfect Matching for Symmetric A: Example (cont.)

P_c from P by changing the order of the columns: $P_c : 3 \leftrightarrow 4$



P_s from P_c by changing the interpretation of the blocks:



PARDISO: fastest parallel direct solver (uses perfect matching).

4.6. Gaussian Elimination for Sparse Matrices

4.6.1. Algebraic Pivoting in GE

- Numerical pivoting: for eliminating elements in column k choose large(st) entry in column/row/block k and permute this element on the diagonal position.
- Disadvantage: may lead to large fill in in the sparsity pattern of A.

4.7. Gaussian Elimination for Sparse Matrices

4.7.1. Algebraic Pivoting in GE

- Numerical pivoting: for eliminating elements in column k choose large(st) entry in column/row/block k and permute this element on the diagonal position.
- Disadvantage: may lead to large fill in in the sparsity pattern of A.
- Idea: Choose pivot element according to minimum fill in! Note that for well-conditioned $A = A^T > 0$ no numerical pivoting is necessary.
- Heuristic: Choose pivot element according to the degree in graph $\rightarrow$ minimum degree reordering

Special Case $A = A^T$

- For elimination in the kth column of A:
  - Define $r_m :=$ number of nonzero entries in row m
  - Choose pivot index i by $r_i = \min_m r_m$
  - Do the pivot permutation and the elimination
  - Go to next column k

Special Case $A = A^T$

- For elimination in the kth column of A:
  - Define $r_m :=$ number of nonzero entries in row m
  - Choose pivot index i by $r_i = \min_m r_m$
  - Do the pivot permutation and the elimination
  - Go to next column k

- r_m is #nonzeros in the mth row = #vertices directly connected with vertex m
  Hence, pivot vertex is vertex with minimum degree in $G(A_k)$

Special Case $A = A^T$

- For elimination in the kth column of A:
  - Define $r_m :=$ number of nonzero entries in row m
  - Choose pivot index i by $r_i = \min_m r_m$
  - Do the pivot permutation and the elimination
  - Go to next column k

- r_m is #nonzeros in the mth row = #vertices directly connected with vertex m
  Hence, pivot vertex is vertex with minimum degree in $G(A_k)$

- Heuristics: few entries in mth row/column $\rightarrow$ few fill in because
  - only few elements to eliminate
  - the pivot row used in the elimination is very sparse

  $\rightarrow$ Multiple minimum degree reordering

Multiple Minimum Degree reordering MMD

- Often, many vertices may have the minimum degree. Which one should be chosen?
- Choose an independent set: Indices with the same minimum degree, but that are no neighbors.

Multiple Minimum Degree reordering MMD

- Often, many vertices may have the minimum degree. Which one should be chosen?
- Choose an independent set: Indices with the same minimum degree, but that are no neighbors.
  - Eliminating any will have no effect on the degree of the others, hence we can eliminate in Gaussian Elimination process in parallel (compare Multifrontal methods)
  - It reduces also the runtime because the outer loop is less than n.
  - It often leads to fewer nonzeros.

Multiple Minimum Degree reordering MMD

- Often, many vertices may have the minimum degree. Which one should be chosen?
- Choose an independent set: Indices with the same minimum degree, but that are no neighbors.
  - Eliminating any will have no effect on the degree of the others, hence we can eliminate in Gaussian Elimination process in parallel (compare Multifrontal methods)
  - It reduces also the runtime because the outer loop is less than n.
  - It often leads to fewer nonzeros.
- Approximative Minimum Degree Reordering (AMD) uses approximations of the degrees.

Generalization to Nonsymmetric Problems: Markowitz

- Define $r_m :=$ nnz in row m; $\quad c_p :=$ nnz in column p
- Choose pivot element with index pair (i, j) such that

$$(r_i - 1)(c_j - 1) = \min_{m,p}(r_m - 1)(c_p - 1)$$

Generalization to Nonsymmetric Problems: Markowitz

- Define $r_m :=$ nnz in row m; $\quad c_p :=$ nnz in column p
- Choose pivot element with index pair (i, j) such that

$$(r_i - 1)(c_j - 1) = \min_{m,p}(r_m - 1)(c_p - 1)$$

- Heuristics:
  - small c_j leads to few elimination steps
  - small r_i leads to sparse pivot row used in the elimination.
- Special case $r_i = 1$ or $c_j = 1$: no fill in.

Generalization to Nonsymmetric Problems: Markowitz

- Define $r_m :=$ nnz in row m; $\quad c_p :=$ nnz in column p
- Choose pivot element with index pair (i, j) such that

$$(r_i - 1)(c_j - 1) = \min_{m,p}(r_m - 1)(c_p - 1)$$

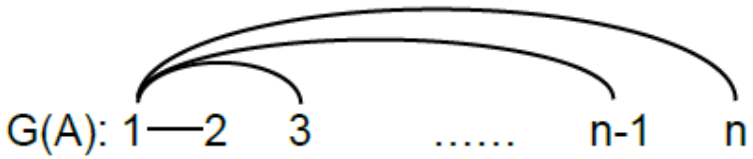
- Heuristics:
  - small c_j leads to few elimination steps
  - small r_i leads to sparse pivot row used in the elimination.
- Special case $r_i = 1$ or $c_j = 1$: no fill in.
- Include numerical pivoting by applying algebraic pivoting only on indices with absolute value that is not to small, e.g.,

$$|a_{i,j}| \geq 0.1 \cdot \max_{r,s} |a_{r,s}|$$

Comparison

Example:

$$A = \begin{pmatrix} * & * & * & \dots & * \\ * & * & & & \\ * & & * & & \\ \vdots & & & \ddots & \\ * & & & & * \end{pmatrix}$$

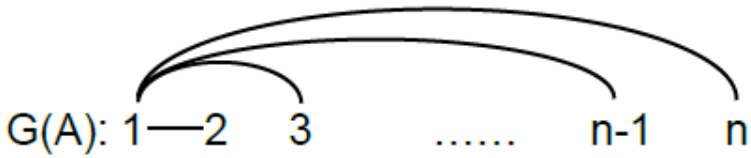


First elimination step leads to dense matrix!

$$\begin{pmatrix} * & * & * & \dots & * \\ 0 & * & * & \dots & * \\ 0 & * & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & * & \dots & * \end{pmatrix}$$

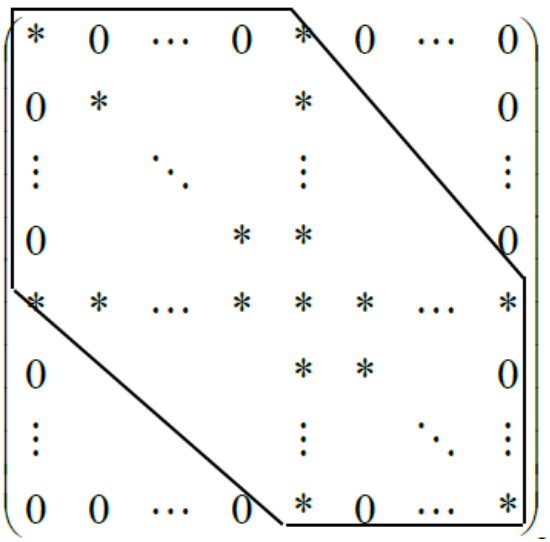
Comparison (cont.)

Cuthill McKee:



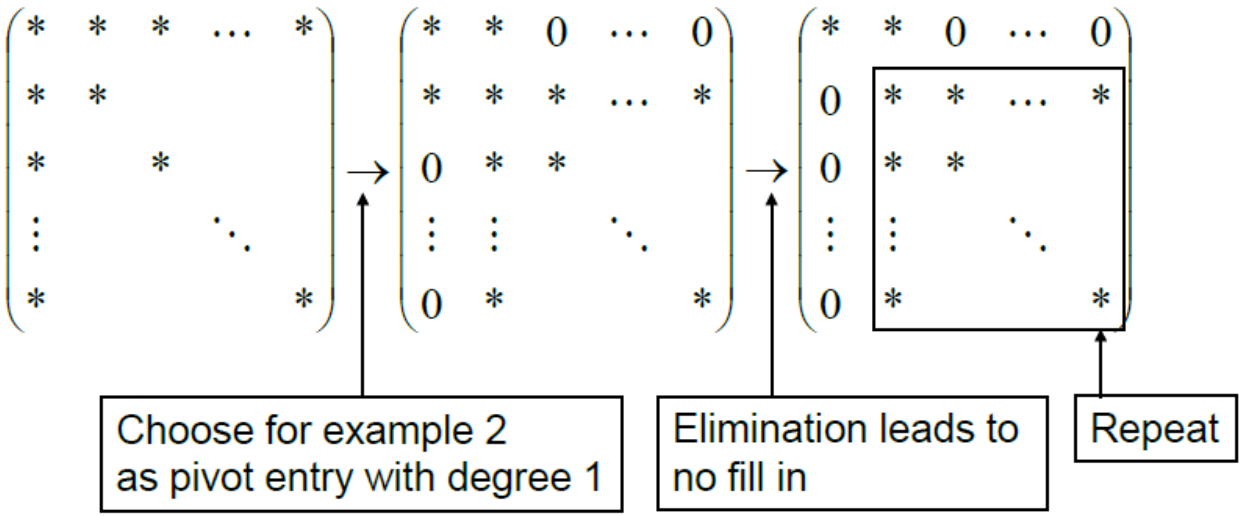
With starting vertex 1:
keeps numbers unchanged $\rightarrow$ no improvement

Optimal bandwidth is not satisfactory: bandwidht $\frac{n}{2}$



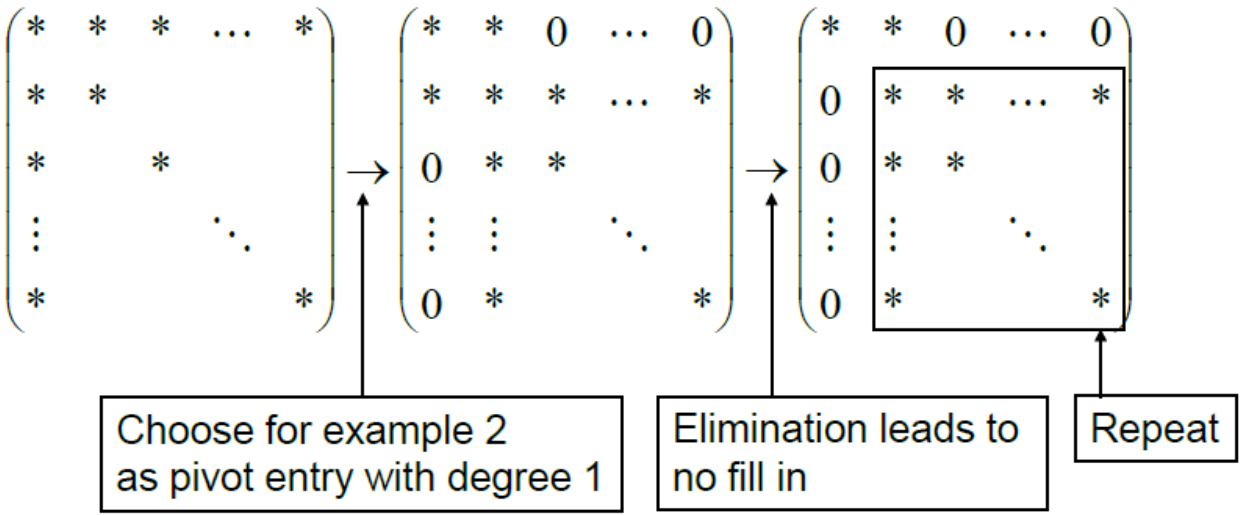
Comparison (cont. 2)

Minimum degree is efficient for this example (PAP^T):



Comparison (cont. 2)

Minimum degree is efficient for this example (PAP^T):

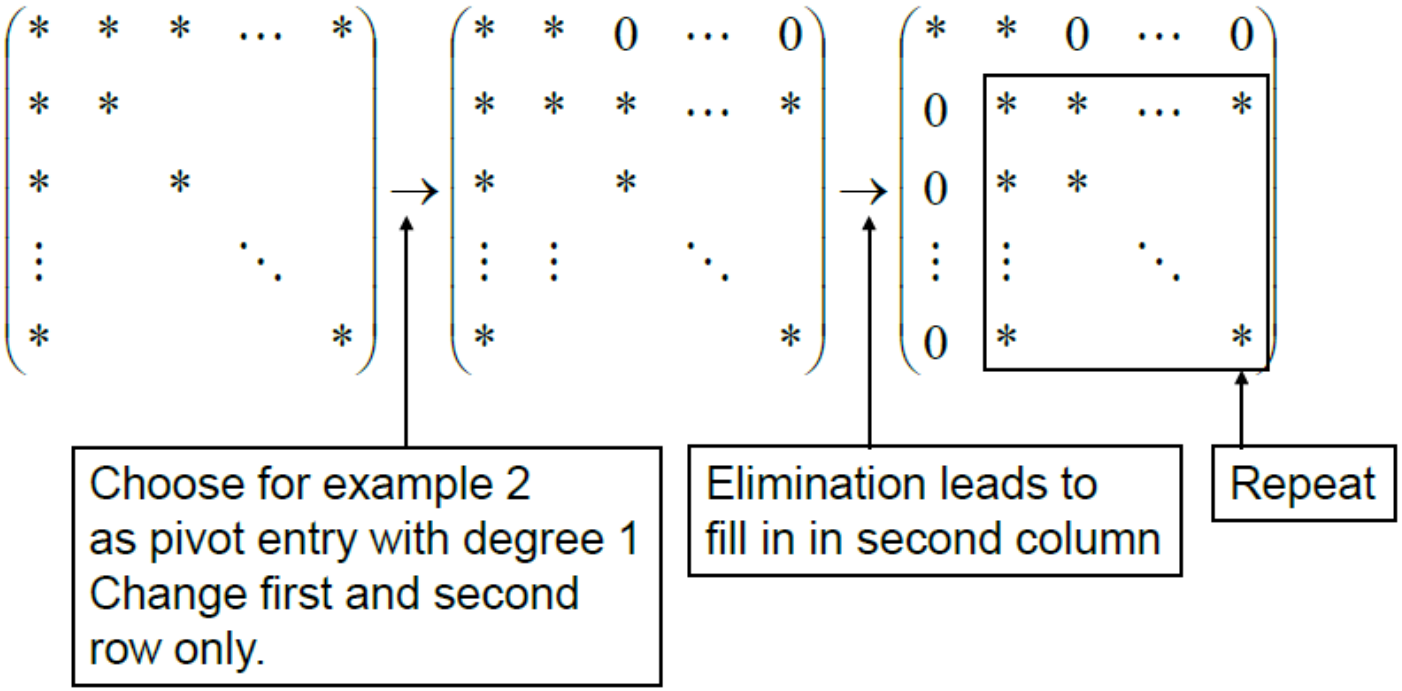


Optimal reordering in one step: $1 \leftrightarrow n$ Optimal costs: $\mathcal{O}(n)$.

$$\begin{pmatrix} * & 0 & 0 & \dots & * \\ 0 & * & & \dots & * \\ 0 & & * & & \vdots \\ \vdots & \vdots & \vdots & \ddots & * \\ * & * & \dots & * & * \end{pmatrix}$$

Example: Nonsymmetric Permutation

Costs: $\mathcal{O}(n^2)$

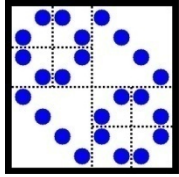


Test: MATLAB (symamd, colamd, amd, colperm, symrcm)

```
load('west0479.mat');a=west0479; s=a'*a; p=symamd(s); spy(s(p,p));
```

See matrix `'west0479.mat'` on Matrix Market:

http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/chemwest/west0479.html



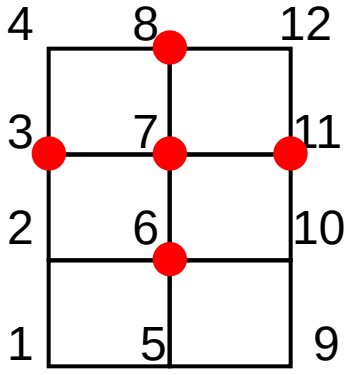
4.4 GE in Graph

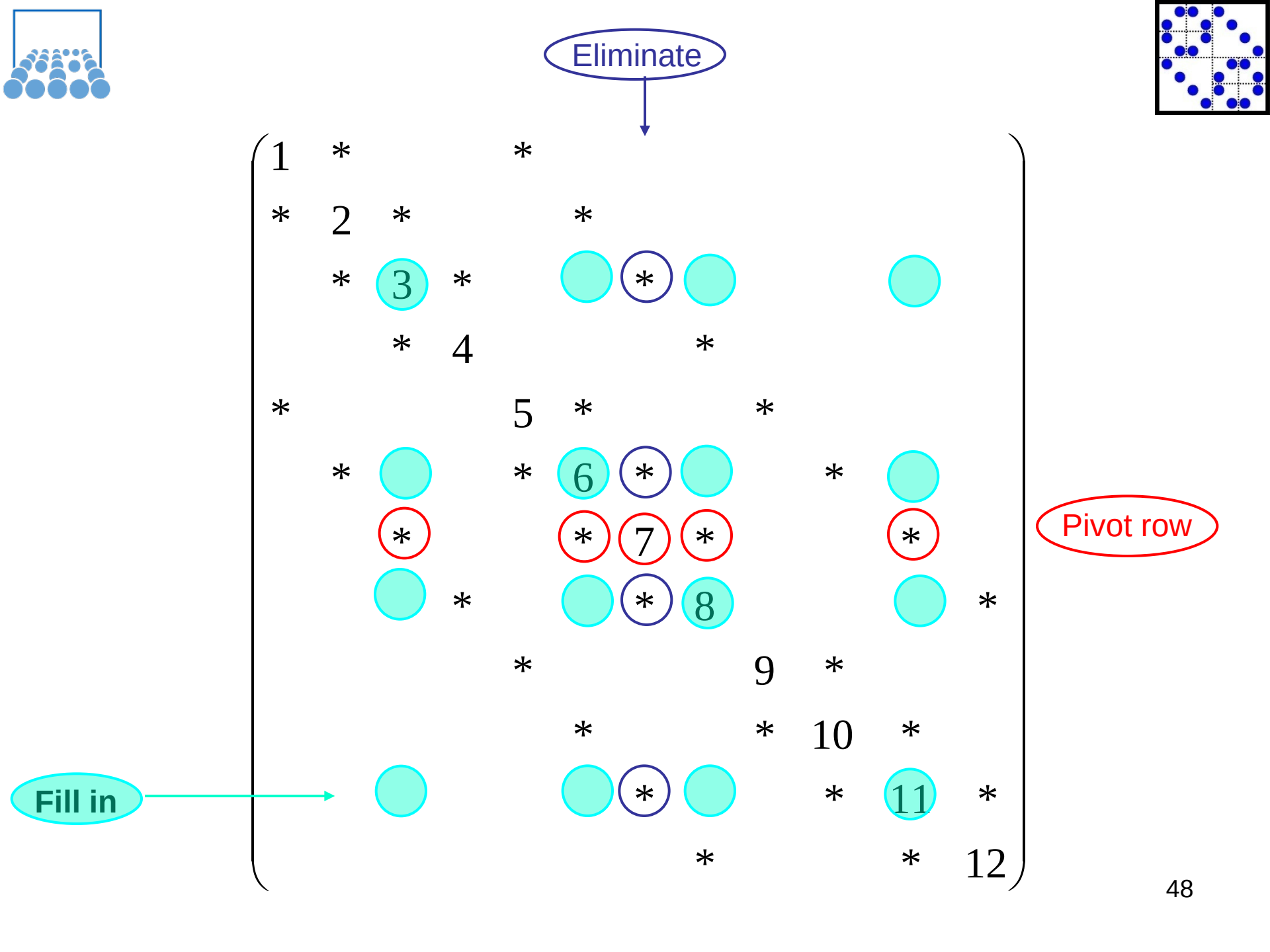
Assume $A = A^T > 0$ spd, symmetric positive definite
(no pivoting necessary)

Consider the graphs related to the Gaussian Elimination step by step.

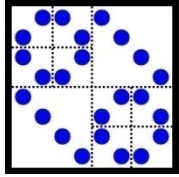
Example: Matrix A with graph G(A)

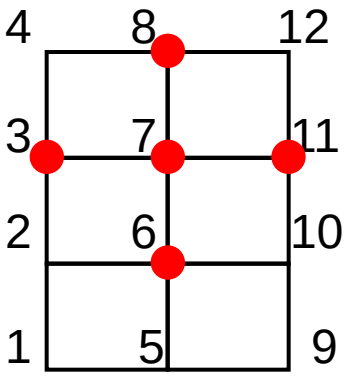
Choose 7 as pivot entry:

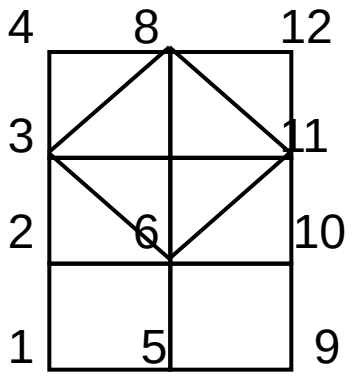



Eliminate
Pivot row
Fill in

Graph after first elimination step:



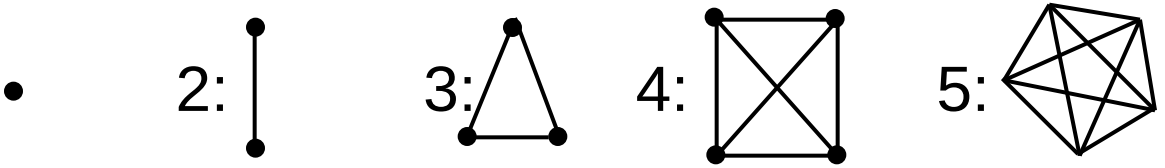




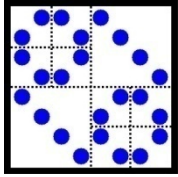
Remove vertex 7 and

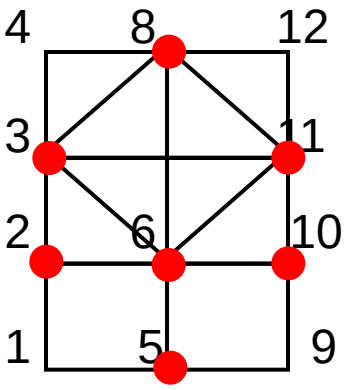
connect all former neighbors of 7 by edges.

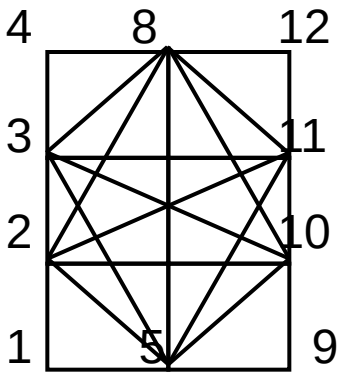
New subgraph with vertices 3,6,8, and 11 (former neighbors of 7) is a fully connected graph = clique = dense submatrix

Clique: 1: 

Next elimination step with pivot 6:





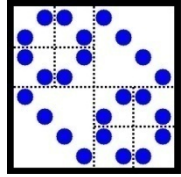


Remove 6

and connect all former neighbors of 6

New clique: 2,3,5,8,10,11

For graph algorithms: Use subroutine package METIS.



Result:

Gaussian elimination can be modelled without numerical computations only algebraically by computing the sequence of related graphs
(in terms of dense subgraphs (matrices) = clique)

Modification of GE:

1. Apply algebraic prestep for GE, determining the graphs related to the elimination matrices A_k in GE.
2. Based on these graphs we can decide
   - whether GE will lead to nearly dense matrices
     (don‘t use GE in this case!)
   - what additional entries will appear during GE
     (prepare the storage)

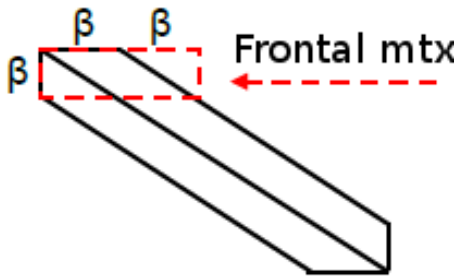
Algebraic prestep is cheap, can be implemented using cliques.

4.7.4. Gaussian Elimination in Graph

- Gaussian elimination can be modelled without numerical computations only algebraically by computing the sequence of related graphs (in terms of dense subgraphs (matrices) = clique)
- Modification of GE:
  1. Apply algebraic prestep for GE, determining the graphs related to the elimination matrices A_k in GE.
  2. Based on these graphs we can decide
     - whether GE will lead to nearly dense matrices (do not use GE in this case!)
     - what additional entries will appear during GE (prepare the storage)
- Algebraic prestep is cheap, can be implemented using cliques.

4.7.5. A Parallel Direct Solver

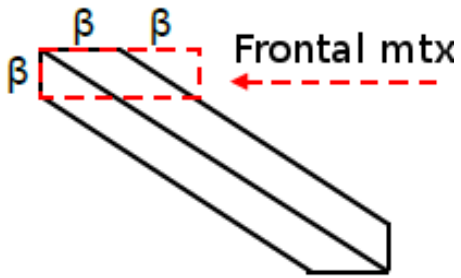
Frontal methods for band matrices (from PDE)



- Frontal dense matrix of size $(\beta + 1) \times (2\beta + 1)$
- Move frontal matrix to CPU and treat as dense matrix.

4.7.6. A Parallel Direct Solver

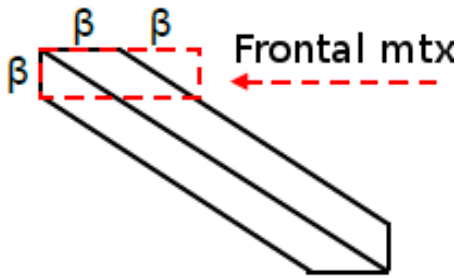
Frontal methods for band matrices (from PDE)



- Frontal dense matrix of size $(\beta + 1) \times (2\beta + 1)$
- Move frontal matrix to CPU and treat as dense matrix.
- Then move frontal matrix one entry down-right and do next elimination.
- Repeat until done.

4.7.7. A Parallel Direct Solver

Frontal methods for band matrices (from PDE)



- Frontal dense matrix of size $(\beta + 1) \times (2\beta + 1)$
- Move frontal matrix to CPU and treat as dense matrix.
- Then move frontal matrix one entry down-right and do next elimination.
- Repeat until done.
- No parallelism until now.

Multifrontal in Parallel

To make efficient use of parallelism search for “independent“ elimination steps!

$$A = \begin{pmatrix} * & 0 & * & * \\ 0 & * & * & * \\ * & * & * & 0 \\ * & * & 0 & * \end{pmatrix}$$

$A_{1,1}$ as first pivot element is related to a first frontal matrix that contains all information to eliminate the first column:

Multifrontal in Parallel

To make efficient use of parallelism search for "independent" elimination steps!

$$A = \begin{pmatrix} * & 0 & * & * \\ 0 & * & * & * \\ * & * & * & 0 \\ * & * & 0 & * \end{pmatrix}$$

$A_{1,1}$ as first pivot element is related to a first frontal matrix that contains all information to eliminate the first column:

Dense submatrix for $k = 1$:

$$\begin{array}{ccc} A_{1,1} & A_{1,3} & A_{1,4} \\ A_{3,1} & \frac{A_{3,1} \cdot A_{1,3}}{A_{1,1}} & \frac{A_{3,1} \cdot A_{1,4}}{A_{1,1}} \\ A_{4,1} & \frac{A_{4,1} \cdot A_{1,3}}{A_{1,1}} & \frac{A_{4,1} \cdot A_{1,4}}{A_{1,1}} \end{array}$$

Multifrontal in Parallel (cont.)

Because $A_{1,2} = A_{2,1} = 0$ we can in parallel consider already the frontal matrix related to $k = 2$, the second step:

$$\begin{array}{ccc} A_{2,2} & A_{2,3} & A_{2,4} \\ A_{3,2} & \frac{A_{3,2} \cdot A_{2,3}}{A_{2,2}} & \frac{A_{3,2} \cdot A_{2,4}}{A_{2,2}} \\ A_{4,2} & \frac{A_{4,2} \cdot A_{2,3}}{A_{2,2}} & \frac{A_{4,2} \cdot A_{2,4}}{A_{2,2}} \end{array}$$

$$A = \begin{pmatrix} * & 0 & * & * \\ 0 & * & * & * \\ * & * & * & 0 \\ * & * & 0 & * \end{pmatrix}$$

Multifrontal in Parallel (cont. 2)

The computations for $k = 1$ and $k = 2$ are independent and can be done in parallel (in the 2×2 block):

$k = 1$:

$$A_{i,j} \to A_{i,j} - \frac{A_{i,1} \cdot A_{1,j}}{A_{1,1}}$$

$k = 2$:

$$A_{i,j} \to A_{i,j} - \frac{A_{i,2} \cdot A_{2,j}}{A_{2,2}}$$

Number of frontal matrices that can be used in parallel depends on the sparsity pattern.

Parallel Numerics, WT 2015/2016

5 Iterative Methods for Sparse Linear Systems of Equations

Contents

- Disadvantages of direct methods (in parallel):
  - strongly sequential
  - may lead to dense matrices
  - sparsity pattern changes, additional entries necessary
  - indirect addressing
  - storage
  - computational effort

- Disadvantages of direct methods (in parallel):
  - strongly sequential
  - may lead to dense matrices
  - sparsity pattern changes, additional entries necessary
  - indirect addressing
  - storage
  - computational effort
- Iterative solver:
  - choose initial guess = starting vector $x^{(0)}$, e.g., $x^{(0)} = \mathbf{0}$
  - iteration function $x^{(k+1)} := \Phi(x^{(k)})$

- Disadvantages of direct methods (in parallel):
  - strongly sequential
  - may lead to dense matrices
  - sparsity pattern changes, additional entries necessary
  - indirect addressing
  - storage
  - computational effort
- Iterative solver:
  - choose initial guess = starting vector $x^{(0)}$, e.g., $x^{(0)} = \mathbf{0}$
  - iteration function $x^{(k+1)} := \Phi(x^{(k)})$
- Applied on solving a linear system:
  - Main part of Φ should be a matrix-vector multiplication Ax (matrix-free!?)
  - Easy to parallelize, no change in the pattern of A.

$$x^{(k)} \overset{k\to\infty}{\longrightarrow} \bar{x} = A^{-1}b$$

  - Main problem: Fast convergence!

5.1. Stationary Methods

5.1.1. Richardson Iteration

- Construct from $Ax = b$ an iteration process:

$$b = Ax = (\underbrace{A - I + I}_{\text{(artificial) splitting of } A})x = x - (I - A)x \quad \Rightarrow \quad x = b + (I - A)x$$

$$= b + Nx$$

5.2. Stationary Methods

5.2.1. Richardson Iteration

- Construct from $Ax = b$ an iteration process:

$$b = Ax = (\underbrace{A - I + I}_{\text{(artificial) splitting of } A})x = x - (I - A)x \quad \Rightarrow \quad x = b + (I - A)x$$

$$= b + Nx$$

- Leads to equation $x = \Phi(x)$ with $\Phi(x) := b + Nx$:

start: $x^{(0)}$;

$$x^{(k+1)} := \Phi(x^{(k)}) = b + Nx^{(k)} = b + (I - A)x^{(k)}$$

Richardson Iteration (cont.)

start: $x^{(0)}$;

$$x^{(k+1)} := \Phi(x^{(k)}) = b + Nx^{(k)} = b + (I - A)x^{(k)}$$

If $x^{(k)}$ is convergent, $x^{(k)} \to \tilde{x}$,

then

$$\tilde{x} = \Phi(\tilde{x}) = b + N\tilde{x} = b + (I - A)\tilde{x} \Rightarrow A\tilde{x} = b$$

and therefore it holds

$$x^{(k)} \to \tilde{x} = \bar{x} := A^{-1}b$$

Richardson Iteration (cont.)

start: $x^{(0)}$;

$$x^{(k+1)} := \Phi(x^{(k)}) = b + Nx^{(k)} = b + (I - A)x^{(k)}$$

If $x^{(k)}$ is convergent, $x^{(k)} \to \tilde{x}$,

then

$$\tilde{x} = \Phi(\tilde{x}) = b + N\tilde{x} = b + (I - A)\tilde{x} \Rightarrow A\tilde{x} = b$$

and therefore it holds

$$x^{(k)} \to \tilde{x} = \bar{x} := A^{-1}b$$

Residual-based formulation:

$$\begin{aligned} x^{(k+1)} &= \Phi(x^{(k)}) = b + (I - A)x^{(k)} = b + x^{(k)} - Ax^{(k)} \\ &= x^{(k)} + \underbrace{(b - Ax^{(k)})}_{r(x) = \text{residual}} = x^{(k)} + r(x^{(k)}) \end{aligned}$$

Convergence Analysis via Neumann Series

$$
\begin{aligned}
x^{(k)} &= b + Nx^{(k-1)} = b + N(b + Nx^{(k-2)}) = b + Nb + N^2 x^{(k-2)} = \\
\ldots &= b + Nb + N^2 b + \cdots + N^{k-1} b + N^k x^{(0)} = \\
&= \sum_{j=0}^{k-1} N^j b + N^k x^{(0)} = \left(\sum_{j=0}^{k-1} N^j \right) b + N^k x^{(0)}
\end{aligned}
$$

Special case $x^{(0)} = 0$:

$$
x^{(k)} = \left(\sum_{j=0}^{k-1} N^j \right) b
$$

Convergence Analysis via Neumann Series

$$
\begin{aligned}
x^{(k)} &= b + Nx^{(k-1)} = b + N(b + Nx^{(k-2)}) = b + Nb + N^2 x^{(k-2)} = \\
\ldots &= b + Nb + N^2 b + \cdots + N^{k-1} b + N^k x^{(0)} = \\
&= \sum_{j=0}^{k-1} N^j b + N^k x^{(0)} = \left(\sum_{j=0}^{k-1} N^j \right) b + N^k x^{(0)}
\end{aligned}
$$

Special case $x^{(0)} = 0$:

$$
x^{(k)} = \left(\sum_{j=0}^{k-1} N^j \right) b
$$

$$
\begin{aligned}
\Rightarrow x^{(k)} \in \operatorname{span}\{b, Nb, N^2 b, \ldots, N^{k-1} b\} &= \operatorname{span}\{b, Ab, A^2 b, \ldots, A^{k-1} b\} \\
&= K_k(A, b)
\end{aligned}
$$

which is called the Krylov space to A and b.

Convergence Analysis via Neumann Series

$$\begin{aligned} x^{(k)} &= b + Nx^{(k-1)} = b + N(b + Nx^{(k-2)}) = b + Nb + N^2 x^{(k-2)} = \\ \ldots &= b + Nb + N^2 b + \cdots + N^{k-1} b + N^k x^{(0)} = \\ &= \sum_{j=0}^{k-1} N^j b + N^k x^{(0)} = \left(\sum_{j=0}^{k-1} N^j \right) b + N^k x^{(0)} \end{aligned}$$

Special case $x^{(0)} = 0$:

$$x^{(k)} = \left(\sum_{j=0}^{k-1} N^j \right) b$$

$$\begin{aligned} \Rightarrow x^{(k)} \in \operatorname{span}\{b, Nb, N^2 b, \ldots, N^{k-1} b\} &= \operatorname{span}\{b, Ab, A^2 b, \ldots, A^{k-1} b\} \\ &= K_k(A, b) \end{aligned}$$

which is called the Krylov space to A and b.

For $\|N\| < 1$ holds:

$$\sum_{j=0}^{k-1} N^j \to \sum_{j=0}^{\infty} N^j = (I - N)^{-1} = (I - (I - A))^{-1} = A^{-1}$$

Convergence Analysis via Neumann Series (cont.)

$$x^{(k)} \to \left(\sum_{j=0}^{\infty} N^j \right) b = (I - N)^{-1} b = A^{-1} b = \bar{x}$$

Richardson iteration is convergent for $\|N\| < 1$ or $A \approx I$.

Convergence Analysis via Neumann Series (cont.)

$$x^{(k)} \to \left(\sum_{j=0}^{\infty} N^j\right) b = (I - N)^{-1} b = A^{-1} b = \bar{x}$$

Richardson iteration is convergent for $\|N\| < 1$ or $A \approx I$.

Error analysis for $e^{(k)} := x^{(k)} - \bar{x}$:

$$\begin{aligned} e^{(k+1)} &= x^{(k+1)} - \bar{x} = \Phi(x^{(k)}) - \Phi(\bar{x}) = (b + Nx^{(k)}) - (b + N\bar{x}) = \\ &= N(x^{(k)} - \bar{x}) = Ne^{(k)} \end{aligned}$$

$$\|e^{(k)}\| \leq \|N\| \|e^{(k-1)}\| \leq \|N\|^2 \|e^{(k-2)}\| \leq \cdots \leq \|N\|^k \|e^{(0)}\|$$

$$\|N\| < 1 \Rightarrow \|N\|^k \overset{k\to\infty}{\longrightarrow} 0 \Rightarrow \|e^{(k)}\| \overset{k\to\infty}{\longrightarrow} 0$$

Convergence Analysis via Neumann Series (cont.)

$$x^{(k)} \to \left(\sum_{j=0}^{\infty} N^j\right) b = (I-N)^{-1} b = A^{-1} b = \bar{x}$$

Richardson iteration is convergent for $\|N\| < 1$ or $A \approx I$.

Error analysis for $e^{(k)} := x^{(k)} - \bar{x}$:

$$\begin{aligned} e^{(k+1)} &= x^{(k+1)} - \bar{x} = \Phi(x^{(k)}) - \Phi(\bar{x}) = (b + Nx^{(k)}) - (b + N\bar{x}) = \\ &= N(x^{(k)} - \bar{x}) = Ne^{(k)} \end{aligned}$$

$$\|e^{(k)}\| \leq \|N\| \|e^{(k-1)}\| \leq \|N\|^2 \|e^{(k-2)}\| \leq \cdots \leq \|N\|^k \|e^{(0)}\|$$

$$\|N\| < 1 \Rightarrow \|N\|^k \overset{k\to\infty}{\longrightarrow} 0 \Rightarrow \|e^{(k)}\| \overset{k\to\infty}{\longrightarrow} 0$$

- Convergence, if $\rho(N) = \rho(I-A) < 1$, where ρ is spectral radius

$$\rho(N) = |\lambda_{\max}| = \max_i(|\lambda_i|) \qquad (\lambda_i \text{ is eigenvalue of } N)$$

- Eigenvalues of A have to be all in circle around 1 with radius 1.

Splittings of A

- Convergence of Richardson only in very special cases!
  Try to improve the iteration for better convergence!
- Write A in form $A := M - N$

$$b = Ax = (M - N)x = Mx - Nx \Leftrightarrow x = M^{-1}b + M^{-1}Nx = \Phi(x)$$

$$\begin{aligned} \Phi(x) &= M^{-1}b + M^{-1}Nx = M^{-1}b + M^{-1}(M - A)x = \\ &= M^{-1}(b - Ax) + x = x + M^{-1}r(x) \end{aligned}$$

Splittings of A

- Convergence of Richardson only in very special cases!
  Try to improve the iteration for better convergence!
- Write A in form $A := M - N$

$$b = Ax = (M - N)x = Mx - Nx \Leftrightarrow x = M^{-1}b + M^{-1}Nx = \Phi(x)$$

$$\begin{aligned} \Phi(x) &= M^{-1}b + M^{-1}Nx = M^{-1}b + M^{-1}(M - A)x = \\ &= M^{-1}(b - Ax) + x = x + M^{-1}r(x) \end{aligned}$$

- N should be such that Ny can be evaluated efficiently.
- M should be such that $M^{-1}y$ can be evaluated efficiently.

$$x^{(k+1)} = x^{(k)} + M^{-1}r^{(k)}$$

- Iteration with splitting $M - N$ is equivalent to Richardson on

$$M^{-1}Ax = M^{-1}b$$

Convergence

- Iteration with splitting $A = M - N$ is convergent if

$$\rho(M^{-1}N) = \rho(I - M^{-1}A) < 1$$

- For fast convergence it should hold
  - $M^{-1}A \approx I$
  - $M^{-1}A$ should be better conditioned than A itself

Convergence

- Iteration with splitting $A = M - N$ is convergent if

$$\rho(M^{-1}N) = \rho(I - M^{-1}A) < 1$$

- For fast convergence it should hold
  - $M^{-1}A \approx I$
  - $M^{-1}A$ should be better conditioned than A itself

- Such a matrix M is called a preconditioner for A.
  Is used in other iterative methods to accelerate convergence.

- Condition number:

$$\kappa(A) = \|A^{-1}\| \|A\|, \quad \left|\frac{\lambda_{\max}}{\lambda_{\min}}\right|, \quad \text{or} \quad \frac{\sigma_{\max}}{\sigma_{\min}}$$

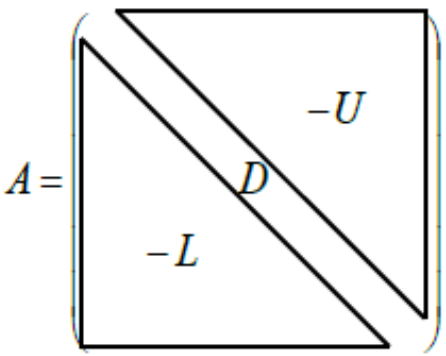
5.2.2. Jacobi (Diagonal) Splitting

Choose $A = M - N = D - (L + U)$ with

$D = \mathrm{diag}(A)$

L the lower triangular part of A, and

U the upper triangular part.



$$
\begin{aligned}
x^{(k+1)} &= D^{-1}b + D^{-1}(L+U)x^{(k)} = \\
&= D^{-1}b + D^{-1}(D-A)x^{(k)} = x^{(k)} + D^{-1}r^{(k)}
\end{aligned}
$$

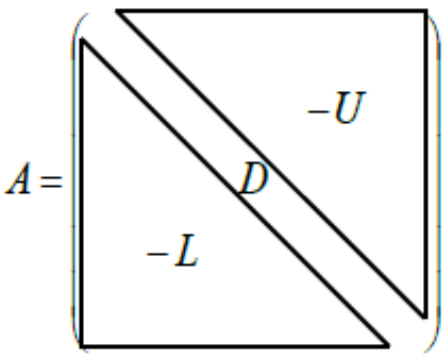
5.2.3. Jacobi (Diagonal) Splitting

Choose $A = M - N = D - (L + U)$ with

$D = \text{diag}(A)$

L the lower triangular part of A, and

U the upper triangular part.



$$\begin{aligned} x^{(k+1)} &= D^{-1}b + D^{-1}(L+U)x^{(k)} = \\ &= D^{-1}b + D^{-1}(D-A)x^{(k)} = x^{(k)} + D^{-1}r^{(k)} \end{aligned}$$

Convergent for $A \approx \text{diag}(A)$ or diagonal dominant matrices:

$$\rho(D^{-1}N) = \rho(I - D^{-1}A) < 1$$

Jacobi (Diagonal) Splitting (cont.)

Iteration process written elementwise:

$$x^{(k+1)} = D^{-1}(b - (A - D)x^{(k)}) \Rightarrow x_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{m=1, m \neq j}^{n} a_{j,m} x_m^{(k)} \right)$$

$$a_{jj} x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{j,m} x_m^{(k)} - \sum_{m=j+1}^{n} a_{j,m} x_m^{(k)}$$

Jacobi (Diagonal) Splitting (cont.)

Iteration process written elementwise:

$$x^{(k+1)} = D^{-1}(b - (A - D)x^{(k)}) \Rightarrow x_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{m=1, m \neq j}^{n} a_{j,m} x_m^{(k)} \right)$$

$$a_{jj} x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{j,m} x_m^{(k)} - \sum_{m=j+1}^{n} a_{j,m} x_m^{(k)}$$

- Damping or relaxation for improving convergence
- Idea: Iterative method as correction of last iterate in search direction.

Jacobi (Diagonal) Splitting (cont.)

Iteration process written elementwise:

$$x^{(k+1)} = D^{-1}(b - (A - D)x^{(k)}) \Rightarrow x_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{m=1, m \neq j}^{n} a_{j,m} x_m^{(k)} \right)$$

$$a_{jj} x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{j,m} x_m^{(k)} - \sum_{m=j+1}^{n} a_{j,m} x_m^{(k)}$$

- Damping or relaxation for improving convergence
- Idea: Iterative method as correction of last iterate in search direction.
- Introduce step length for this correction step:

  $$x^{(k+1)} = x^{(k)} + D^{-1} r^{(k)} \qquad \rightarrow \qquad x^{(k+1)} = x^{(k)} + \omega D^{-1} r^{(k)}$$

  with additional damping parameter ω.
- Damped Jacobi iteration:

  $$x_{\text{damped}}^{(k+1)} = (\omega + 1 - \omega) x^{(k)} + \omega D^{-1} r^{(k)} = \omega x^{(k+1)} + (1 - \omega) x^{(k)}$$

Damped Jacobi Iteration

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \omega D^{-1} r^{(k)} = x^{(k)} + \omega D^{-1}(b - Ax^{(k)}) = \\ &= \dots \\ &= \omega D^{-1} b + [(1-\omega)I + \omega D^{-1}(L+U)]x^{(k)} \end{aligned}$$

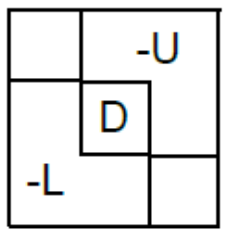
is convergent for

$$\rho(\underbrace{[(1-\omega)I + \omega D^{-1}(L+U)]}_{\overset{\omega \to 0}{\longrightarrow} I}) < 1$$

Look for optimal ω with best convergence (add. degree of freedom).

Parallelism in the Jacobi Iteration

- Jacobi method is easy to parallelize: only Ax and $D^{-1}x$.
- But often too slow convergence!
- Improvement: block Jacobi iteration



5.2.4. Gauss-Seidel Iteration

Always use newest information available!

Jacobi iteration:

$$a_{jj}x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{j,m} \underbrace{x_m^{(k)}}_{\text{already computed}} - \sum_{m=j+1}^{n} a_{j,m}x_m^{(k)}$$

Gauss-Seidel iteration:

$$a_{jj}x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{j,m} \underbrace{x_m^{(k+1)}}_{\text{already computed}} - \sum_{m=j+1}^{n} a_{j,m}x_m^{(k)}$$

Gauss-Seidel Iteration (cont.)

- Compare dependancy graphs for general iterative algorithms. Here:

$$x = f(x) = D^{-1}(b + (D - A)x) = D^{-1}(b - (L + U)x)$$

to splitting $A = (D - L) - U = M - N$

$$\begin{aligned} x^{(k+1)} &= (D-L)^{-1}b + (D-L)^{-1}Ux^{(k)} = \\ &= (D-L)^{-1}b + (D-L)^{-1}(D-L-A)x^{(k)} = \\ &= x^{(k)} + (D-L)^{-1}r^{(k)} \end{aligned}$$

- Convergence depends on spectral radius $\rho(I - (D - L)^{-1}A) < 1$

Parallelism in the Gauss-Seidel Iteration

- Linear system in $D - L$ is easy to solve because $D - L$ is lower triangular but
- strongly sequential!
- Use red-black ordering or graph colouring for compromise:

$$\text{parallel} \leftrightarrow \text{convergence}$$

Successive Over Relaxation (SOR)

- Damping or relaxation:

$$x^{(k+1)} = x^{(k)} + \omega(D-L)^{-1} r^{(k)} = \omega(D-L)^{-1} b + [(1-\omega) + \omega(D-L)^{-1} U] x^{(k)}$$

- Convergence depends on spectral radius of iteration matrix

$$(1-\omega) + \omega(D-L)^{-1} U$$

- Parallelization of SOR == parallelization of GS

Stationary Methods (in General)

- Can always be written in the two normal forms

  $$x^{(k+1)} = c + Bx^{(k)}$$

  with convergence depending on $\rho(B)$ and

  $$x^{(k+1)} = x^{(k)} + Fr^{(k)}$$

  with preconditioner F, $B = I - FA$

Stationary Methods (in General)

- Can always be written in the two normal forms

  $$x^{(k+1)} = c + Bx^{(k)}$$

  with convergence depending on $\rho(B)$ and

  $$x^{(k+1)} = x^{(k)} + Fr^{(k)}$$

  with preconditioner F, $B = I - FA$

- For $x^{(0)} = \mathbf{0}$:

  $$x^{(k+1)} \subseteq K_k(B, c),$$

  which is the Krylov space with respect to matrix B and vector c.

- Slow convergence (but good smoothing properties! $\rightarrow$ multigrid)