FEM and Sparse Linear System Solving

Lecture 6, October 27, 2017: Direct methods for sparse linear systems

http://people.inf.ethz.ch/arbenz/FEM17

Peter Arbenz
Computer Science Department, ETH Zürich
E-mail: arbenz@inf.ethz.ch

- The finite element method
- Direct solvers for sparse systems
  - LU and Cholesky decomposition
  - Sparse matrices
  - Fill-reducing orderings
- Iterative solvers for sparse systems

Recap of some linear algebra/factorization basics

LU factorization

- J. Demmel: *Applied Numerical Linear Algebra*, SIAM 1997.
- G. Golub & Ch. van Loan: *Matrix Computations*, 4th ed., Chapter 3, Johns Hopkins, 2013.

Gaussian elimination is the algorithm to compute the LU factorization.

- A fundamental theoretical and practical tool!
- One of the 'Top 10 Algorithms', see:
  G. W. Stewart: *The decompositional approach to matrix computation*, Computing in Science and Engineering 2(1), pp. 50–59, 2000.

Recap of some linear algebra/factorization basics (cont.)

Recent survey of one of the protagonists:

- Timothy A. Davis (+ S. Rajamanickam, W. M. Sid-Lakhdar): *A survey of direct methods for sparse linear systems*. Acta Numerica (2016), pp. 383–566. https://doi.org/10.1017/S0962492916000076

Linear systems: problem statement

We consider linear systems of equation of the form

$$\sum_{k=1}^{n} a_{ik} x_k = b_i, \qquad i = 1, \ldots, n,$$

or

$$A\,\boldsymbol{x} = \boldsymbol{b}.$$

The matrix elements a_{ik} and the right-hand side elements b_i are given. We are looking for the n *unknowns* x_k.

Direct vs. iterative methods

- We consider the problem of finding $\boldsymbol{x}$ which solves

$$A\,\boldsymbol{x} = \boldsymbol{b}.$$

  where A is a given real, square $(n \times n)$, nonsingular matrix and $\boldsymbol{b}$ is a given real vector.
- Such problems are ubiquitous in applications, and often the most time critical.
- *Two types of solution approaches*:
  1. Direct methods: yield exact solution in absence of roundoff error.
     - Variations of *Gaussian elimination*.
  2. Iterative methods: iterate in a similar fashion to what we do for nonlinear problems.
     - Use only when direct methods are ineffective.

Existence and uniqueness of LU decomposition

The decomposition of the matrix A into a unit lower triangular matrix L and an upper triangular matrix U, $A = LU$, is called LU decomposition or LU factorization. The process that computes the LU decomposition is called Gaussian elimination.

Theorem

The square matrix $A \in \mathbb{R}^{n\times n}$ has a unique decomposition $A = LU$ if and only if the leading principal submatrices $A_k := A(1:k, 1:k), k = 1, \ldots, n-1$, are nonsingular.

Existence and uniqueness of LU decomposition (cont.)

Theorem

If A is nonsingular, then one can find a row permutation P such that PA satisfies the conditions of the previous theorem, that is $PA = LU$ exists and is unique.

If $PA = LU$ is available, then instead of $A\boldsymbol{x} = \boldsymbol{b}$ one solves

$$P\,A\,\boldsymbol{x} = P\,\boldsymbol{b},$$

in two steps: $L\boldsymbol{y} = P\,\boldsymbol{b}$, forward substitution,

$U\boldsymbol{x} = \boldsymbol{y}$, backward substitution.

REMARK: L and U can be stored in A. (They overwrite A.)
P needs an extra vector.

Cholesky factorization

Theorem

If A is symmetric positive definite (SPD), then there is a lower triangular matrix L such that $A = LL^T$.

The factorization is unique if we request that the diagonal elements of L are positive.

If $A = LL^T$, then instead of

$$A\boldsymbol{x} = \boldsymbol{b},$$

one solves

$$LL^T\boldsymbol{x} = \boldsymbol{b} \quad \Longleftrightarrow \quad L\boldsymbol{y} = \boldsymbol{b}, \qquad L^T\boldsymbol{x} = \boldsymbol{y}.$$

REMARK: Only the lower (upper) triangle of A is accessed.

Gaussian elimination with partial pivoting

x_1	x_2	x_3	x_4	1
a_{11}	a_{12}	a_{13}	a_{14}	b_1
a_{21}	a_{22}	a_{23}	a_{24}	b_2
a_{31}	a_{32}	a_{33}	a_{34}	b_2
a_{41}	a_{42}	a_{43}	a_{44}	b_2

1. Permute rows $i = 1, \ldots, 4$ (if necessary) such that $a_{11} \neq 0$. This element is called the pivot.
2. Subtract multiples $l_{i1} = a_{i1}/a_{11}$ of row 1 from row i, $i = 2, \ldots, 4$.
3. Set $a'_{ik} = a_{ik} - l_{i1}a_{1k}$, $i, k = 2, \ldots, 4$.
4. Set $b'_i = b_i - l_{i1}b_1$, $i = 2, \ldots, 4$.

Gaussian elimination with partial pivoting (cont.)

x_1	x_2	x_3	x_4	1
a_{11}	a_{12}	a_{13}	a_{14}	b_1
0	a'_{22}	a'_{23}	a'_{24}	b'_2
0	a'_{32}	a'_{33}	a'_{34}	b'_2
0	a'_{42}	a'_{43}	a'_{44}	b'_2

1. Permute rows $i = 2, \ldots, 4$ (if necessary) such that $a'_{22} \neq 0$. This is the next pivot.
2. Subtract multiples $l'_{i2} = a'_{i2}/a'_{22}$ of row 2 from row i, $i = 3, \ldots, 4$.
3. Set $a''_{ik} = a'_{ik} - l'_{i2}a'_{2k}$, $i, k = 3, \ldots, 4$.
4. Set $b''_i = b'_i - l'_{i2}b'_2$, $i = 3, \ldots, 4$.

Gaussian elimination with partial pivoting (cont.)

x_1	x_2	x_3	x_4	1
a_{11}	a_{12}	a_{13}	a_{14}	b_1
0	a'_{22}	a'_{23}	a'_{24}	b'_2
0	0	a''_{33}	a''_{34}	b''_2
0	0	a''_{43}	a''_{44}	b''_2

1. Permute rows $i = 3, \ldots, 4$ (if necessary) such that $a''_{33} \neq 0$. This is the next pivot.
2. Subtract multiple $l''_{43} = a''_{43}/a''_{33}$ of row 3 from row 4.
3. Set $a'''_{44} = a''_{44} - l''_{43} a''_{34}$.
4. Set $b'''_4 = b''_4 - l''_{i1} b''_3$.

Gaussian elimination with partial pivoting (cont.)

x_1	x_2	x_3	x_4	1
u_{11}	u_{12}	u_{13}	u_{14}	c_1
0	u_{22}	u_{23}	u_{24}	c_2
0	0	u_{33}	u_{34}	c_2
0	0	0	u_{44}	c_2

$\Longleftrightarrow$

x_1	x_2	x_3	x_4	1
a_{11}	a_{12}	a_{13}	a_{14}	b_1
0	a'_{22}	a'_{23}	a'_{24}	b'_2
0	0	a''_{33}	a''_{34}	b''_2
0	0	0	a'''_{44}	b'''_2

Actual storage scheme

x_1	x_2	x_3	x_4	1
a_{11}	a_{12}	a_{13}	a_{14}	b_1
l_{21}	a'_{22}	a'_{23}	a'_{24}	b'_2
l_{31}	l'_{32}	a''_{33}	a''_{34}	b''_2
l_{41}	l'_{42}	l'_{43}	a'''_{44}	b'''_2

plus vector that stores info on permutations

Gaussian elimination for $Ax = b$ (matrix notation)

We stick with the 4×4 example. Let

$$L_1 = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ l_{31} & & 1 & \\ l_{41} & & & 1 \end{pmatrix} \quad L_2 = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & l'_{32} & 1 & \\ & l'_{42} & & 1 \end{pmatrix} \quad L_3 = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & l''_{43} & 1 \end{pmatrix}$$

Then, we executed the following

$$U = L_3^{-1} P_3 L_2^{-1} P_2 L_1^{-1} P_1 A$$

which can be interpreted as

$$U = \underbrace{L_3^{-1}(P_3 L_2^{-1} P_3^{-1})(P_3 P_2 L_1^{-1} P_2^{-1} P_3^{-1})}_{L^{-1}} \underbrace{(P_3 P_2 P_1)}_{P} A$$

Possible numerical difficulties

$$\begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & \\ 10^{20} & 1 \end{bmatrix} \begin{bmatrix} 10^{-20} & 1 \\ & 1 - 10^{20} \end{bmatrix}$$

Finite precision: 1.) align to same exponent, 2.) subtract.

In double precision (MATLAB):

```
>> a=1;
>> b=10^20;
>> a-b
ans =
     -1.000000000000000e+20
```

Relative error of subtraction ok, but:

$$\begin{bmatrix} 1 & \\ 10^{20} & 1 \end{bmatrix} \begin{bmatrix} 10^{-20} & 1 \\ & -10^{20} \end{bmatrix} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 0 \end{bmatrix}$$

$\Rightarrow$ Very different matrix, algorithm like this not *backward stable*

Sensitivity (Wilkinson, Higham)

Let A be nonsingular

$$\begin{aligned} Ax &= b, \\ (A+\Delta A)(\boldsymbol{x}+\boldsymbol{\Delta x}) &= (\boldsymbol{b}+\boldsymbol{\Delta b}). \end{aligned}$$

If $\|\Delta A\| \leq \varepsilon \|A\|$, $\|\boldsymbol{\Delta b}\| \leq \varepsilon \|\boldsymbol{b}\|$, then

$$\frac{\|\boldsymbol{\Delta x}\|}{\|\boldsymbol{x}\|} \leq \frac{2\varepsilon\kappa(A)}{1-\varepsilon\kappa(A)},$$

for $\varepsilon\kappa(A) < 1$.

Norm-wise condition number $\kappa(A) = \|A\|\|A^{-1}\|$.

A ill conditioned $:\Longleftrightarrow \kappa(A) \gg 1$

Condition Estimation

With the Singular Value Decomposition $A = U\Sigma V^T$, one has

$$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2 = \frac{\sigma_{\max}}{\sigma_{\min}}.$$

In practice, to compute the bound

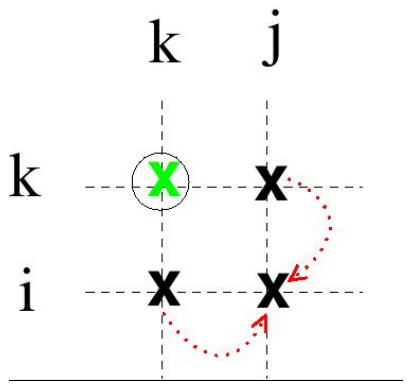
$$\frac{\|\Delta \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{2\varepsilon\kappa(A)}{1 - \varepsilon\kappa(A)},$$

one often does not care about the exact κ, an estimate is sufficient!

MATLAB: `condest(A)`. We will not talk about this here, see

- N. Higham: *Accuracy and Stability of Numerical Algorithms*, Chapter 15, SIAM 2002.

Element growth in the LU decomposition(Wilkinson)



$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)} \cdot a_{kj}^{(k)}}{a_{kk}^{(k)}}$$

The stability of an LU decomposition depends on the growth factor. Let $A = LU$ be nonsingular, and $\hat{\boldsymbol{x}}$ the computed solution of $A\boldsymbol{x} = \boldsymbol{b}$.

$$(A + \Delta A)\hat{\boldsymbol{x}} = \boldsymbol{b}, \quad \|\Delta A\| \sim \rho \cdot \|A\|, \quad \rho = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|}.$$

Thus the growth factor measures the growth of the elements during the elimination. Purpose of pivoting: obtain small element growth where possible. Remember:

$$\begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix} \text{ vs. } \begin{bmatrix} 1 & \\ 10^{20} & 1 \end{bmatrix} \begin{bmatrix} 10^{-20} & 1 \\ & 1 - 10^{20} \end{bmatrix}$$

Partial (column) pivoting $PA = LU$

Before elimination step k, exchange row
$i = \arg\max \left\{ |a_{ik}^{(k)}|, i \geq k \right\}$ with row $k \implies |l_{ik}| \leq 1$.

$$\begin{bmatrix} * & * & * & * \\ \times & \times & \times & \times \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \times & \times & \times & \times \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \times & \times & \times & \times \\ * & * & * & * \\ \times & \times & \times & \times \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \times & \times & \times \\ \mathbf{0} & \times & \times & \times \\ \mathbf{0} & \times & \times & \times \end{bmatrix} \Rightarrow \cdots$$

Notes:

- Leading columns of U are never affected by later permutations,
- Lower triangular part can be used to store L.

SPD matrices & Cholesky factorization

Conditions that are equivalent to $\boldsymbol{x}^T A\boldsymbol{x} > 0,\ \forall \boldsymbol{x} \in \mathbb{R}^n \backslash \{0\}$:

1. All eigenvalues of A are positive.
2. All leading principal matrices $A(1:k, 1:k)$ have positive determinants.
3. The Cholesky decomposition $A = LL^T$ exists.
   (Cheapest way to check.)

L is lower triangular, with positive entries on the diagonal (difference to LU decomposition: here, these are in general different from 1)

Connection to QR factorization:
Let $A = QR$ be nonsingular, then $A^T A = R^T Q^T QR = R^T R$.
R is the Cholesky factor of $A^T A$ if all diagonal entries are positive.

When pivoting is *not* needed

Definition

$A \in \mathbb{R}^{n \times n}$ is column-wise strictly diagonally dominant (SDD), if

$$|A_{i,i}| > \sum_{j \neq i} |A_{j,i}|.$$

A is Symmetric Positive Definite (SPD), if $A = A^T$ and

$$\boldsymbol{x}^T A \boldsymbol{x} > 0, \quad \forall \boldsymbol{x} \in \mathbb{R}^n \setminus \{\boldsymbol{0}\}.$$

Theorem

If A is column-wise SDD or SPD, then $A = LU$ exists and is unique.

Exercise, or see Golub & van Loan.

Cholesky vs. Gaussian Elimination

Q: Why prefer Cholesky to Gaussian Elimination?
A: Cholesky exploits symmetry and definiteness!

- Cholesky is always stable! Element growth is limited:

$$L = U\Sigma V^T \Rightarrow A = U\Sigma^2 U^T \Rightarrow \|L\|_2 = \sqrt{\|A\|_2},\ \|L^{-1}\|_2 = \sqrt{\|A^{-1}\|_2}.$$

- Only need to store half of the matrix and one triangular factor.
- Better complexity. For $A \in \mathbb{R}^{n\times n}$, we have

$$\text{Gauss: } \frac{2}{3}n^3 + \mathcal{O}\left(n^2\right), \quad \text{Cholesky: } \frac{1}{3}n^3 + \mathcal{O}\left(n^2\right).$$

floating point operations. Both need $\mathcal{O}(n^2)$ memory space.

Matlab

- MATLAB's backslash operator, `x=A\b`, provides Gaussian Elimination with partial pivoting to solve $Ax = b$.
- Explicit form for $LU = PA$ is

  `[U,L,P]=lu(A);` such that `x=U\(L\(P*b));`

- With

  `[U,L]=lu(A);` such that `x=U\(L\b);`

  we get a "psychologically lower triangular matrix" `L`, i.e., MATLAB actually returns $P^T L$ in `L`.
- The Cholesky decompositions is obtained by

  `L=chol(A,'lower');` `x=L'\(L\b);`

Introduction to sparse matrices

What is a sparse matrix?

Informal working definition by J.H. Wilkinson (1969):

> *Any matrix with enough zeros that it pays to take advantage of them.*

This is not a 'real' definition.

- Sparsity not so much a matter of fraction of nonzeros.
- 'It's the economy, ...': memory, operations, computing time.

Remark: More generally: *structurally sparse* matrices are matrices that can be represented with a 'few numbers'. E.g., the rank-1 matrix $H = \boldsymbol{u}\boldsymbol{v}^T$ is dense in general. However, one needs to store only two vectors to have all the information to do a matrix-vector product.

Introduction to sparse matrices (cont.)

Various practical aspects of sparsity:

- Storage of the sparse matrix
- Access of matrix elements
- Operations on the matrix
  (LU and Cholesky decomposition)
- Operations with the matrix
  (multiplication of a sparse matrix with a vector)

Note: The storage scheme depends on what you want to do with the matrix.

Sparse Matrix Storage Formats

MATLAB uses coordinate-based (COO) format to display sparse matrices. You can enter a sparse matrix using the COO format:

```
S = sparse ( i, j, s, m, n, nzmax );
```

- i: the row indices of the nonzero elements;
- j: the column indices of the nonzero elements;
- s: the values of the nonzero elements;
- m: the number of rows in the matrix;
- n: the number of columns in the matrix;
- nzmax: the maximum number of nonzero elements in the matrix; nzmax is commonly omitted because its value is taken from the length of s.

Sparse Matrix Storage Formats (cont.)

Simple! But can be improved regarding memory accesses.

```
i = [ 1, 1, 1, 2, 2, 3, 3, 3, 3 ];
j = [ 1, 2, 5, 2, 3, 1, 3, 4, 5 ];
val = [ 11, 12, 15, 22, 23, 31, 33, 34, 35 ];
m = 3;
n = 5;
nzmax = 9;
a = sparse ( i, j, val, m, n, nzmax );
```

$$\boldsymbol{a} = \begin{bmatrix} 11 & 12 & 0 & 0 & 15 \\ 0 & 22 & 23 & 0 & 0 \\ 31 & 0 & 33 & 34 & 35 \end{bmatrix}$$

Survey of Sparse Matrix Storage Formats

Some alternatives to COO:

- Compressed Row Storage (CRS). Most used for MatVec
- Compressed Column Storage (CCS), used by MATLAB, Rutherford-Boeing
- Block Compressed Row/Column Storage (BCRS/BCCS)
- ELLPACK / ELLPACK-R
- Compressed Diagonal Storage (CDS)
- Jagged Diagonal Storage (JDS)
- Skyline Storage (SKS), used by many FE codes
- Element-by-element (FEM)

All formats have advantages/disadvantages. The best choice depends on context (matrix structure) and algorithm.

Example: CRS format

$$A = \begin{bmatrix} 1 & & & & \\ & 2 & 5 & & \\ & 3 & 6 & & 9 \\ & 4 & & 8 & \\ & & 7 & & 10 \end{bmatrix}$$

CRS uses three vectors for storing nonzeros row-wise:

$$\texttt{vals} = [1, 2, 5, 3, 6, 9, 4, 8, 7, 10]$$
$$\texttt{col\_ind} = [1, 2, 3, 2, 3, 5, 2, 4, 3, 5]$$
$$\texttt{row\_ptr} = [1, 2, 4, 7, 9, 11]$$

The last element of `row_ptr` points *behind* last element of `vals`.

Sparse MatVec with CRS format

```
y = zeros(m,1);
for k=1:m,
   for i=row_ptr(k):row_ptr(k+1)-1
      y(k) = y(k) + vals(i)*x(col_ind(i));
   end
end
```

`vals`: set of arrays comprising the matrix entries
`col_ind`: set of arrays comprising the column indices

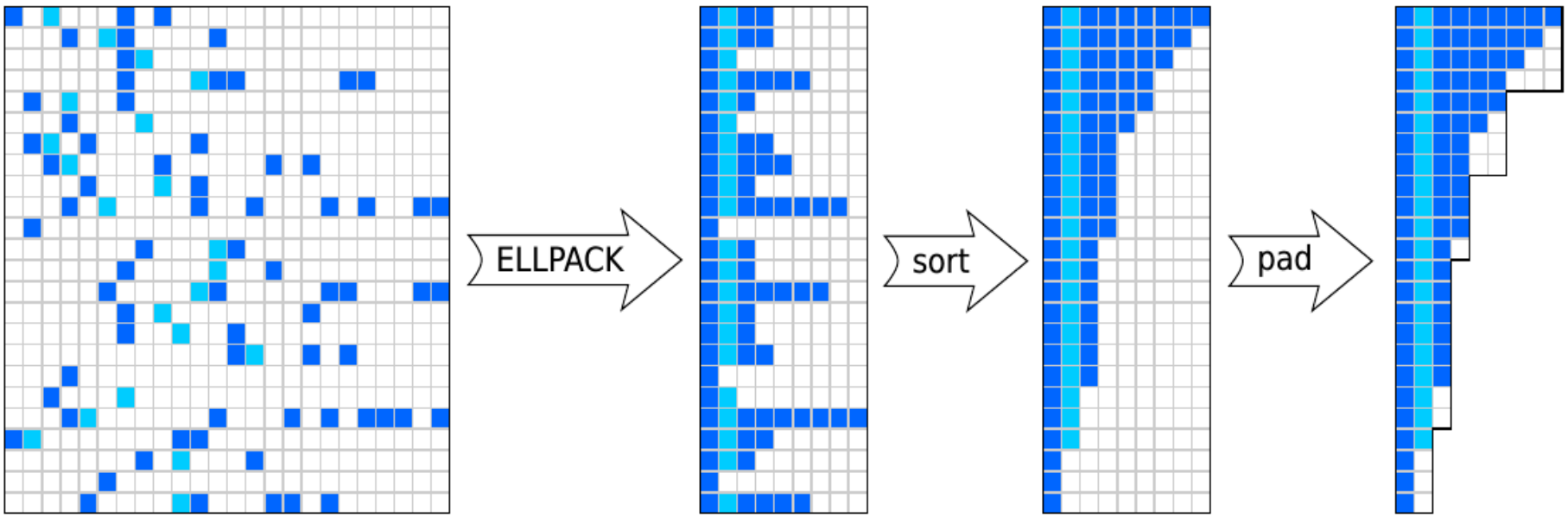
Should be used if no regular substructures can be exploited.

Sparse matrices in Matlab: CCS format

```
% Given a sparse matrix A stored as dense matrix

S = sparse(A);          % generate sparse matrix from A
[i,j,s] = find(S);      % extract generating information
[m,n] = size(S);
A = sparse(i,j,s,m,n);    % generate sparse matrix from
                          %    generating information
A = sparse(m,n);
A = spalloc(m,n,nnz);
A = spdiags(B,d,m,n); A = speye(n); A = spones(S);
spy(A)                    % look at nonzero structure
```

ELLPACK/ELLPACK-R



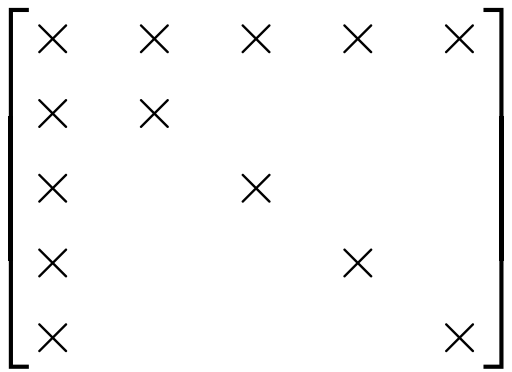
ELLPACK contains zero entries (white boxes in figure).

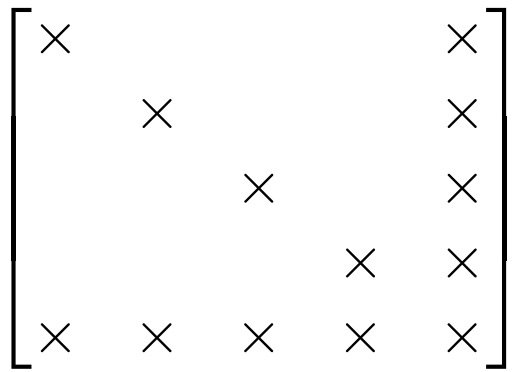
See Kreutzer *et al.*: Sparse matrix-vector multiplication on GPGPU clusters... IPDPS 2012.

Sparse factorizations and fill-in

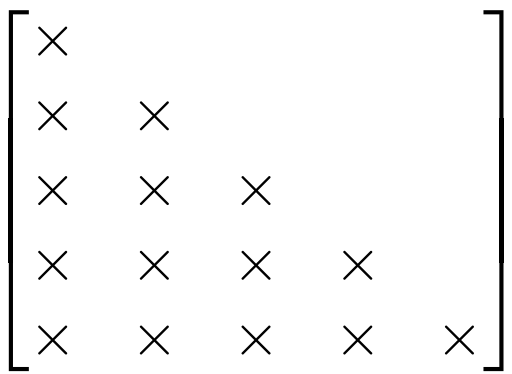
- We consider the LU (Cholesky) factorization of a sparse matrix A.
- The factors L and U are in general sparse again.
- The structure of the L and U factors, the locations of any non-zero entries, is different as compared to A. Matrix entries which are zero in A, but non-zero in L or U are called fill-ins.
- Triangular factors generally have more, even much more (!), nonzero entries than the original matrix.
- Fill-in requires a larger amount of memory and causes a more expensive factorization.

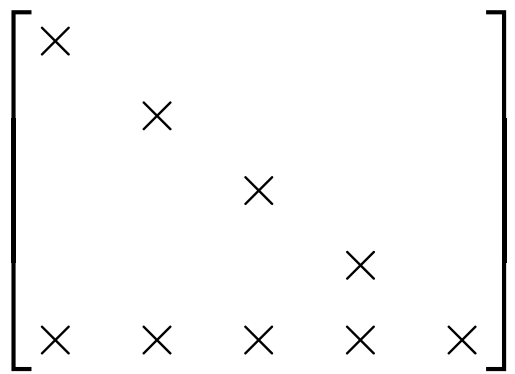
Fill-in and reordering





Corresponding lower triangular factors:





Sparse factorizations and fill-in (cont.)

- What can we do to obtain little or no fill-in?
- No perfect solution for this problem. Finding optimal permutations P, Q s.t. the factorization of PAQ has less fill-in than that of A is NP-hard problem $\longrightarrow$ heuristics needed.
- Occasionally, permuting rows and columns of A signifiantly helps in reducing fill-in.
- Leads to considerations involving graph theory.
- If no pivoting occurs, we can compute the structural fill-in symbolically!
- We represent the matrix as a graph and work on the graph.
- By doing this we do not consider the actual values of the matrix entries! (Therefore the word 'structural'.)

Double role of permutations

1. Guarantee stability, **and**
2. Avoid fill-in.

Potentially additional objectives:

1. Preserve symmetry if A symmetric ($Q = P^T$).
2. Obtain diagonal entries that are large in magnitude (avoid pivoting).
3. Block entries of dense submatrices to allow use of BLAS3 operations.
4. Allow for good parallelism.

Adjacency graph & incidence matrix

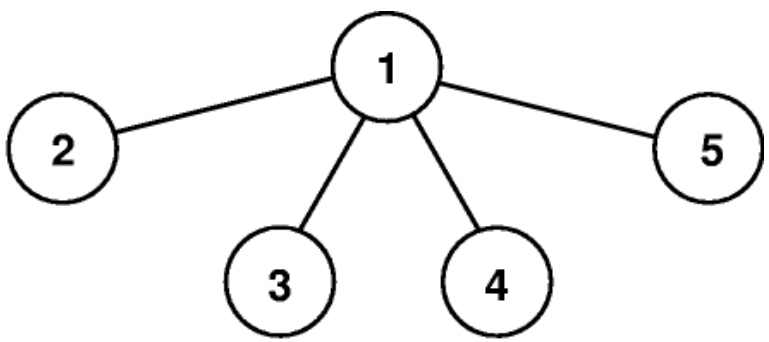
Definition

The adjacency graph $G(S) = (V, E)$ of the symmetric sparse matrix $S \in \mathcal{R}^{n \times n}$ consists of the vertices $V = \{1, \ldots, n\}$ and the edges $E = \{(i, j) : s_{ij} \neq 0\}$.

(Dual definition: incidence matrix for given undirected graph.)

Example: arrow matrix

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & & & \\ \times & & \times & & \\ \times & & & \times & \\ \times & & & & \times \end{bmatrix}$$

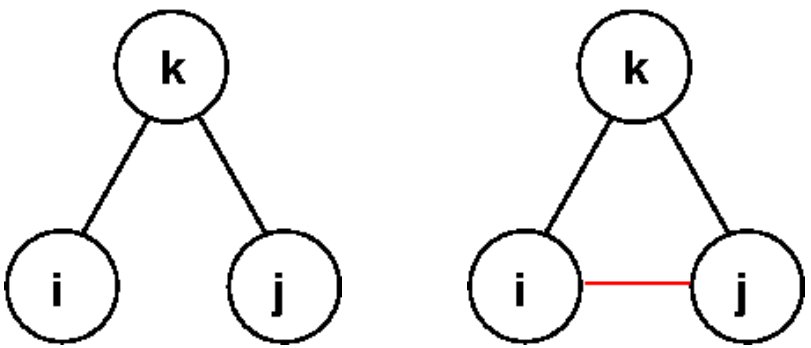


When does fill-in occur?

Let's consider the k-th step in Gaussian elimination.
Let us assume that $a_{ik}^{(k)} \cdot a_{kj}^{(k)} \neq 0$ but that $a_{ij}^{(k)} = 0$.
Then, according to

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)} \cdot a_{kj}^{(k)}}{a_{kk}^{(k)}}$$

there will be fill in.

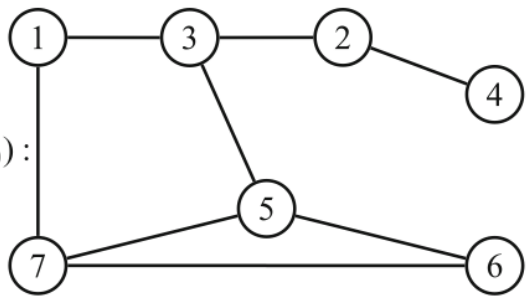


In the k-th elimination step all neighbors of node k are connected.

Example from Larson–Bengzon book

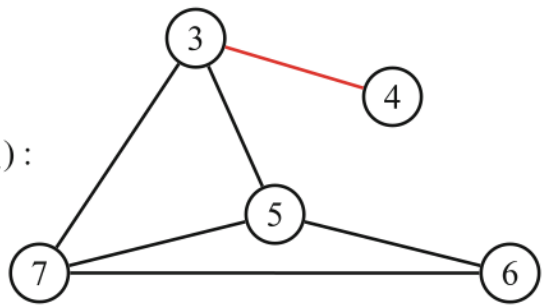
$$A_0 = \begin{bmatrix} 1 & & & & x \\ & 2 & x & x & \\ x & x & 3 & & x \\ & x & & 4 & \\ & & x & 5 & x & x \\ & & & x & 6 & x \\ x & & & x & x & 7 \end{bmatrix}$$

$G(A_0)$:



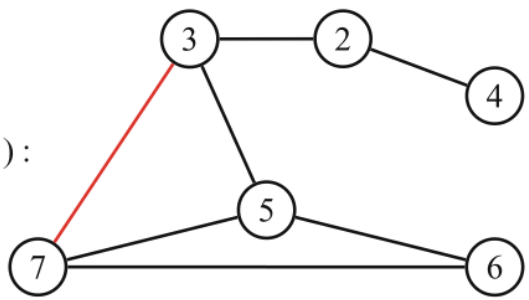
$$A_2 = \begin{bmatrix} 3 & x & x & & x \\ x & 4 & & & \\ x & & 5 & x & x \\ & & x & 6 & x \\ x & & x & x & 7 \end{bmatrix}$$

$G(A_2)$:



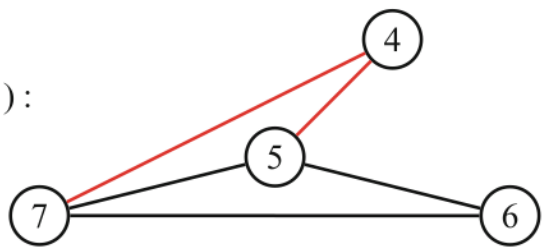
$$A_1 = \begin{bmatrix} 2 & x & x & & \\ x & 3 & & x & x \\ x & & 4 & & \\ & x & & 5 & x & x \\ & & & x & 6 & x \\ & x & & x & x & 7 \end{bmatrix}$$

$G(A_1)$:



$$A_3 = \begin{bmatrix} 4 & x & & x \\ x & 5 & x & x \\ & x & 6 & x \\ x & x & x & 7 \end{bmatrix}$$

$G(A_3)$:

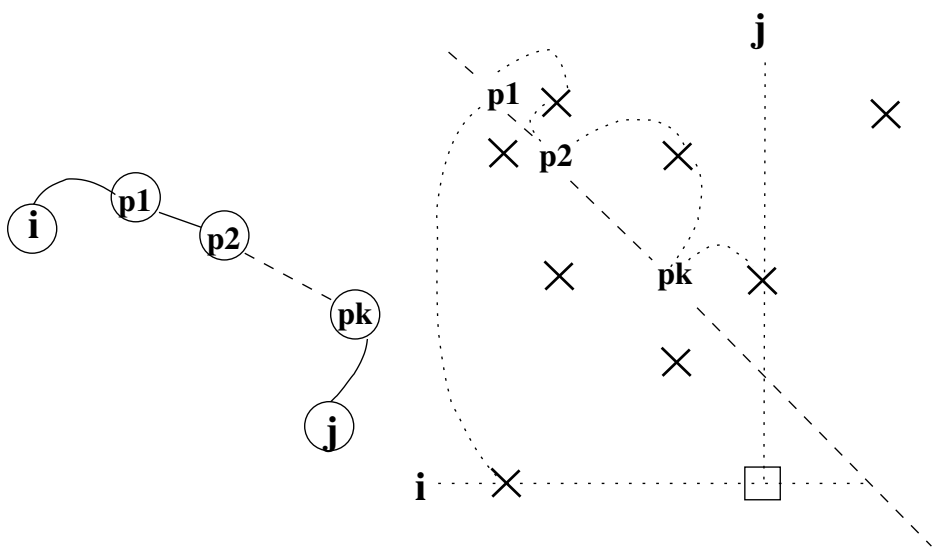


Cholesky fill-in path characterization

- Cholesky does not need pivoting
- Fill-in can be predicted *beforehand*!

Theorem (Fill path by Rose–Tarjan–Lueker)

Let the Cholesky factorization $A = LL^T$ exist. Then $l_{ij} \neq 0$ if and only if there is an edge path connecting the vertices $i, p_1, \ldots, p_k, j$ from $G(A)$, with $p_1, \ldots, p_k < \min(i, j)$.

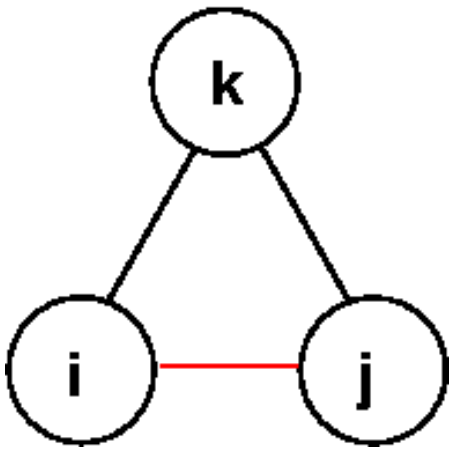


Fill-in in the Cholesky factorization

Let $k < i < j$.

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}^{(k)} \cdot l_{jk}^{(k)}$$

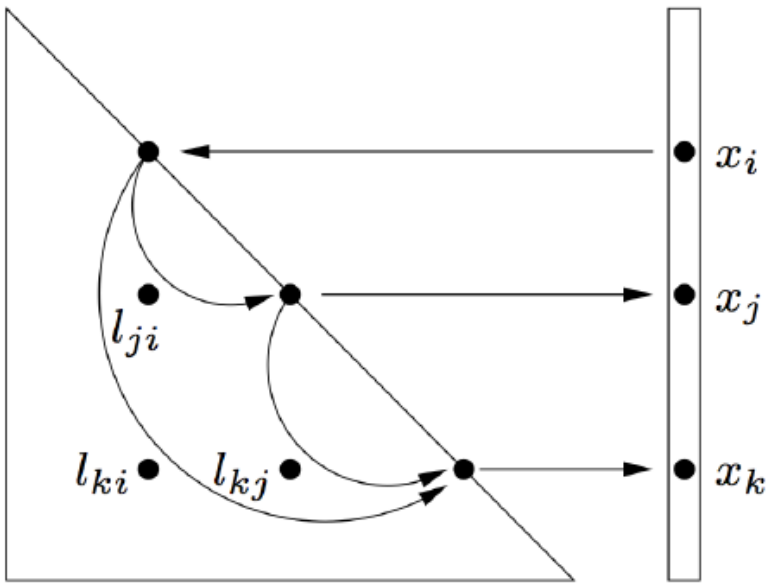
In the k-th elimination step there is fill if $a_{ij}^{(k)} = 0$ and $l_{ik}^{(k)} \cdot l_{jk}^{(k)} \neq 0$.



Any fill-in that is caused by l_{jk} (or the edge (j, k)) is recovered by l_{ik} and l_{ij} (or the path $j \to i \to k$).

Therefore, the first nonzero element of each column below the diagonal holds all the necessary information to compute the nonzeros of the Cholesky factorization.

Fill-in in the Cholesky factorization (cont.)



$a_{ik} \neq 0$ implies $l_{ki} \neq 0$

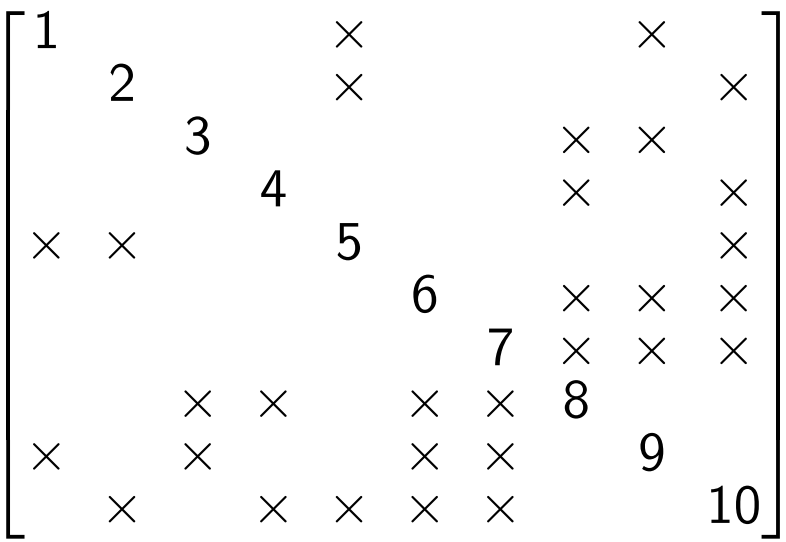
$l_{ji} \neq 0$ and $l_{ki} \neq 0$ implies $l_{kj} \neq 0$

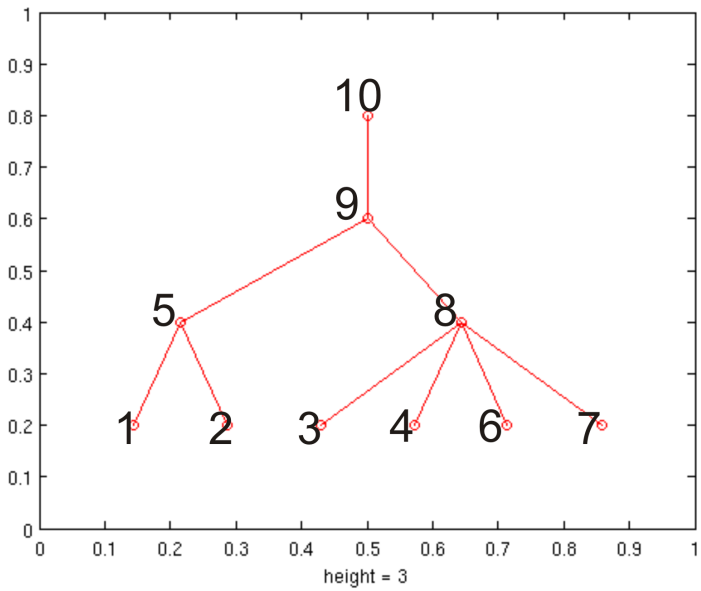
Thus l_{ki} redundant for $\mathcal{X} = \text{Reach}_L(i)$

From paper by Davis, Rajamanickam, Sid-Lakhdar (2016).

The elimination tree

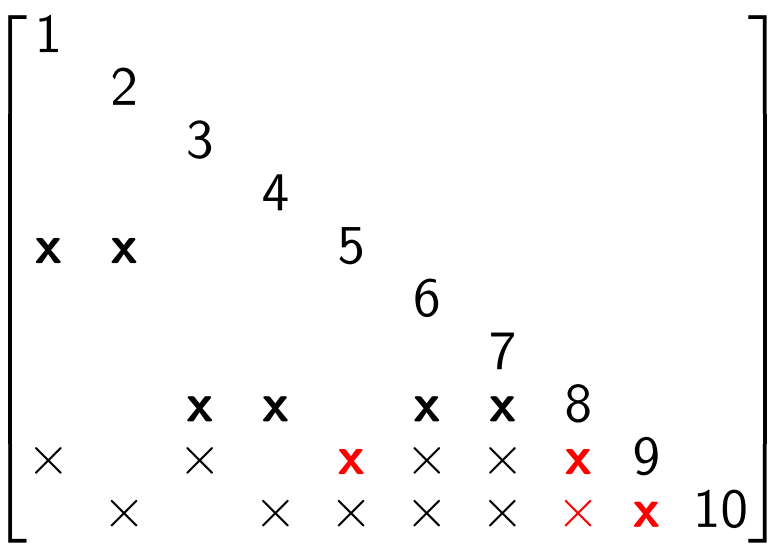
- Represents partial ordering for Cholesky factorization
- See Matlab's `etree`, `etreeplot`

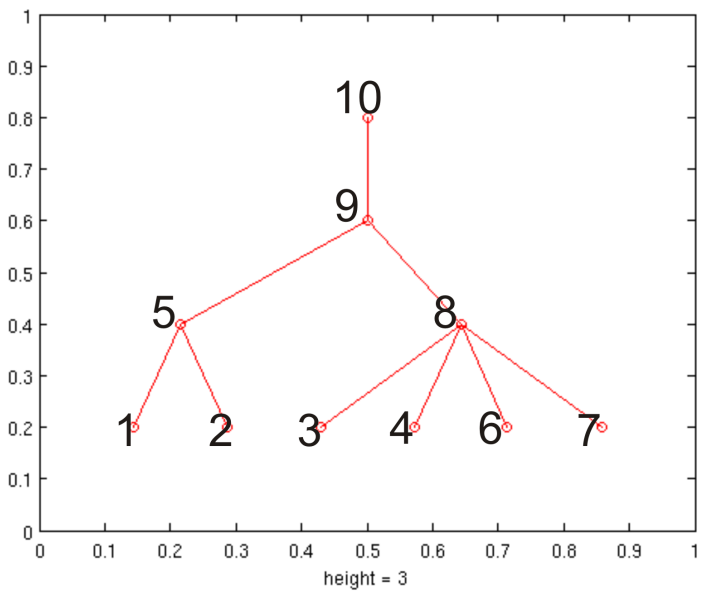




The elimination tree (cont.)

- Computation of elim. tree from symbolic Cholesky factor.
- Rule: *first* entry (bold) below diagonal determines ordering.





J. Liu: *The Role of Elimination Trees in Sparse Factorization*, SIAM J. Matrix Anal. **11**, 134–172 (1990).

The elimination tree (cont.)

$$\begin{bmatrix} L_{11} & \\ \boldsymbol{l}_{21} & l_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & \boldsymbol{l}_{21}^T \\ & l_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & \boldsymbol{a}_{12} \\ \boldsymbol{a}_{12}^T & a_{22} \end{bmatrix}$$

$\boldsymbol{l}_{21}$ is obtained by solving $L_{11}\boldsymbol{l}_{21}^T = \boldsymbol{a}_{12}$. Therefore, the structure of $\boldsymbol{l}_{21}$ depends on the structure of L_{11} and of $\boldsymbol{a}_{12}$.

- Elimination tree can be used to cheaply determine the location of the nonzeros of the Cholesky factor, row by row.
- The parent of node i in the tree is node $k > i$ where l_{ki} is the first nonzero off-diagonal element of L in column i.

LU fill-in

Pivoting makes LU more complicated than Cholesky!

Practical issues:

- Fill-in in general *not* predictable a-priori.
- Symbolic analysis is *not* enough, numerical values matter too!
- 'Dynamic' pivoting conflicts with a-priori estimated fill-in.
- A-priori estimates may be too small, may need to provide & manage additional memory during factorization.

To get an a-priori estimate on fill-in, commonly use $G(A + A^T)$. Without pivoting, this yields an upper bound.

Scaling & equilibration

'Badly scaled systems' can look artificially hard! Compare:

$$\begin{bmatrix} 10 & 10^8 \\ 10^8 & 10^{16} \end{bmatrix} \quad \text{vs.} \quad \begin{bmatrix} 10 & 1 \\ 1 & 1 \end{bmatrix}$$

Idea: apply nonsingular diagonal scaling matrices D_r, D_c to equilibrate as much as possible:

- $A \Rightarrow D_r A D_c$, here: $D_r = D_c = \mathrm{diag}(1, 10^{-8})$,
- $Ax = b \Leftrightarrow (D_r A D_c) D_c^{-1} x = D_r b$

Scaling does influence pivoting (see above!), and thus also

- fill-in & complexity,
- condition number, accuracy.

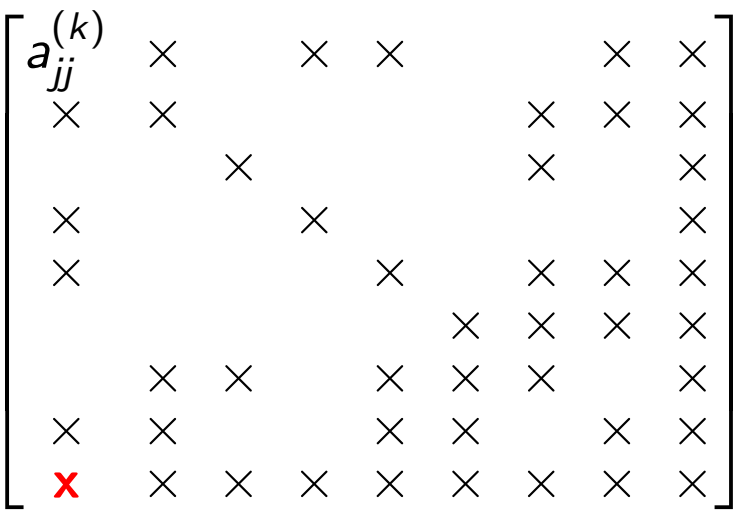
Dynamic Threshold-Pivoting (LU)

- Partial-Pivoting: good stability
- Threshold-Pivoting: tradeoff some stability for smaller fill-in

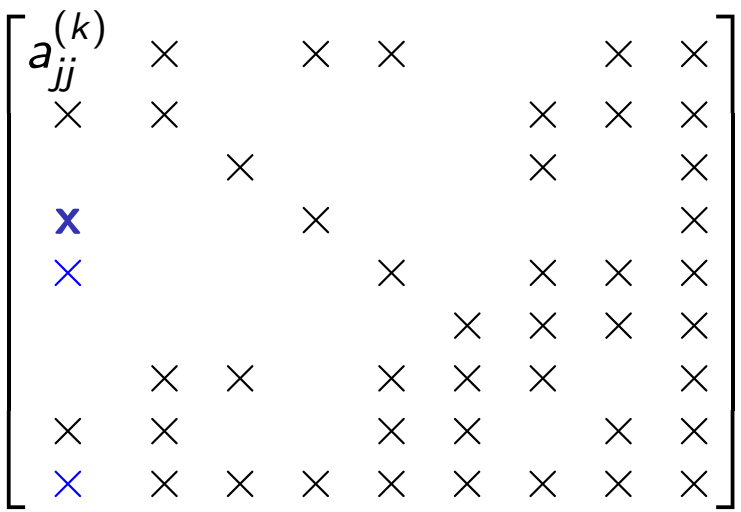
Select pivot with smallest fill-in from set

$$\{a_{ij}^{(k)} \ : \ |a_{ij}^{(k)}| \geq u \cdot \max_i |a_{ij}^{(k)}|\}, \quad 0 < u \leq 1. \ \ (e.g.\ u := 0.1)$$

Partial Pivoting:



Threshold Pivoting:



3 phases of direct linear solvers for sparse $Ax = b$

1. Analysis (or: symbolic factorization)
   - Analysis of the nonzero structure
   - Row- and column-permutations
   - Scaling to improve condition number
2. Numerical Factorization
   - $A = LU$ unsymmetric, with pivoting
   - $A = LL^T$ symmetric positive definite, no pivoting
3. Solve $Ax = b$ using triangular factors
   - Can also solve for multiple right-hand sides
   - If more accuracy necessary: Iterative Refinement

Band matrices

A is a $n \times n$ **band** matrix if all matrix elements are zero outside a diagonally bordered band

$$a_{i,j} = 0, \quad \text{if } j < i - k_1 \text{ or } j > i + k_2, \quad k_1, k_2 \geq 0.$$

A band matrix with

- $k_1 = k_2 = 0 \Rightarrow$ Diagonal Matrix
- $k_1 = k_2 = 1 \Rightarrow$ Tridiagonal Matrix
- $k_1 = k_2 = 2 \Rightarrow$ Pentadiagonal Matrix

The bandwidth of the matrix is $k_1 + k_2 + 1$.
Fill-in can occur only within the band. (The band structure can be consider a simplified version of the skyline format.)

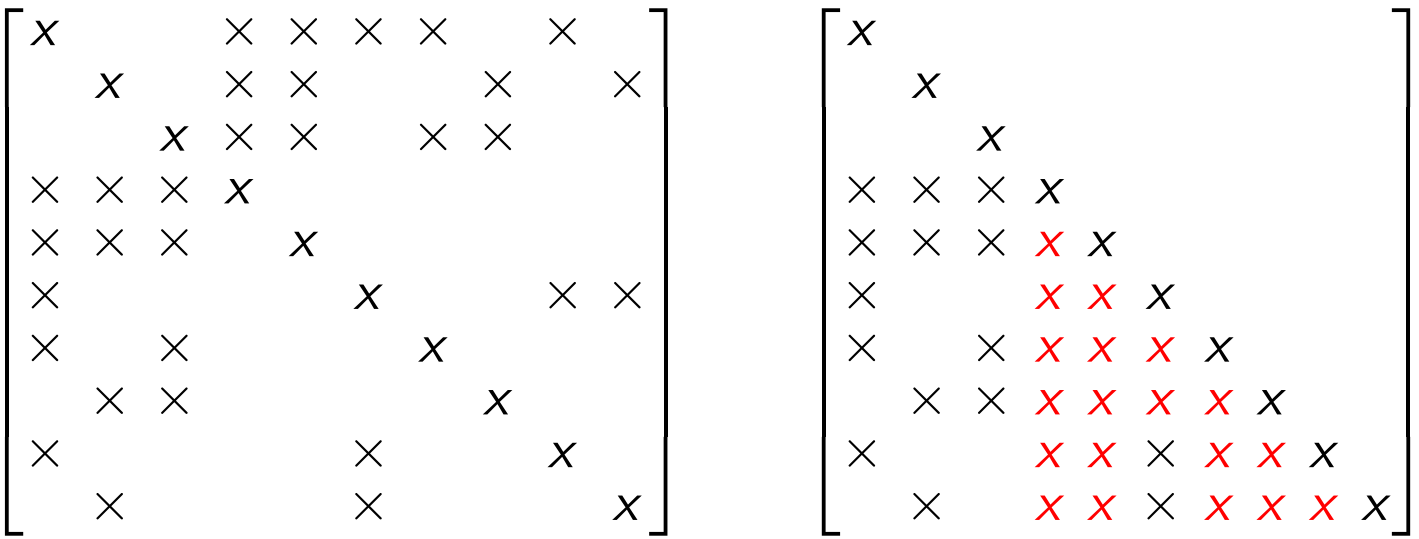
Heuristics for avoiding fill-in

1. General principles of sparse direct solvers
2. Heuristics for avoiding fill-in: Matrix Reorderings
3. Combination with Maximum-Transversal Permutation
4. Software

Heuristics for avoiding fill-in

1. For reducible matrices
   - DulmageMendelsohn (Matlab's DMPERM): permute A to block-triangular form (if possible)
2. Local Greedy-algorithms
   - Reverse Cuthill-McKee (MATLAB's `symrcm`): reduce bandwidth of A
   - Approximate Minimum Degree (MATLAB's `symamd`): local minimization of fill-in
3. Global, Domain-Decomposition inspired reordering
   - Nested Dissection: divide and conquer heuristic

A sample matrix & fill-in



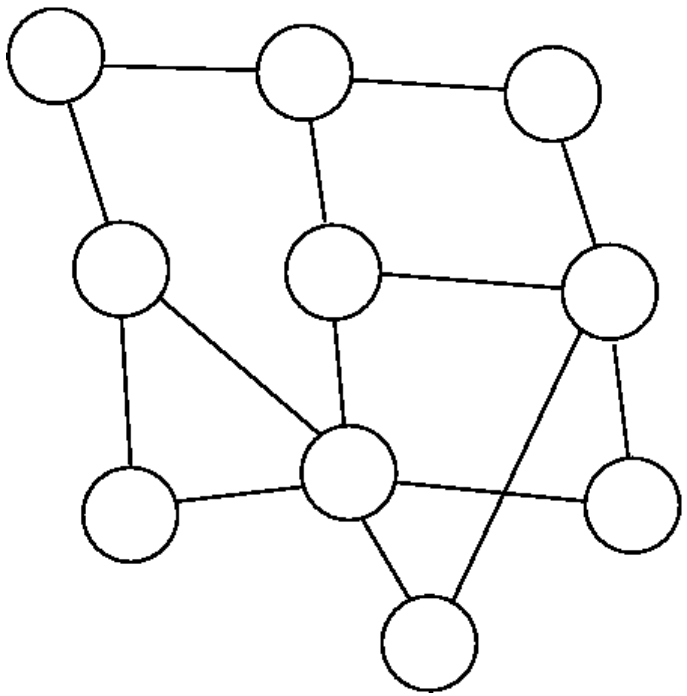
- Elimination of an unknown $\Rightarrow$ clique among neighbors
- Matrix looks like lots of 'up arrows' $\Rightarrow$ submatrix $A(4:10, 4:10)$ *completely filled in*!
- Can we do better than 19 fill-ins?

Reverse Cuthill-McKee algorithm

- The Cuthill-McKee algorithm was designed to reduce the bandwidth of sparse symmetric matrices.
  REF.: E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In: Proc. 24th Nat. Conf. ACM, pp. 157–172, 1969.
- The reverse Cuthill-McKee algorithm (RCM) is the same algorithm but with the resulting index numbers reversed. In general, a better solution (less fill-in).
- The reverse Cuthill-McKee algorithm algorithm is often (but need not be) used in connection with the skyline storage scheme. (Only the entries from the first nonzero entry to the last nonzero entry are stored.)
- MATLAB command `symrcm`.

Reverse Cuthill-McKee algorithm (cont.)

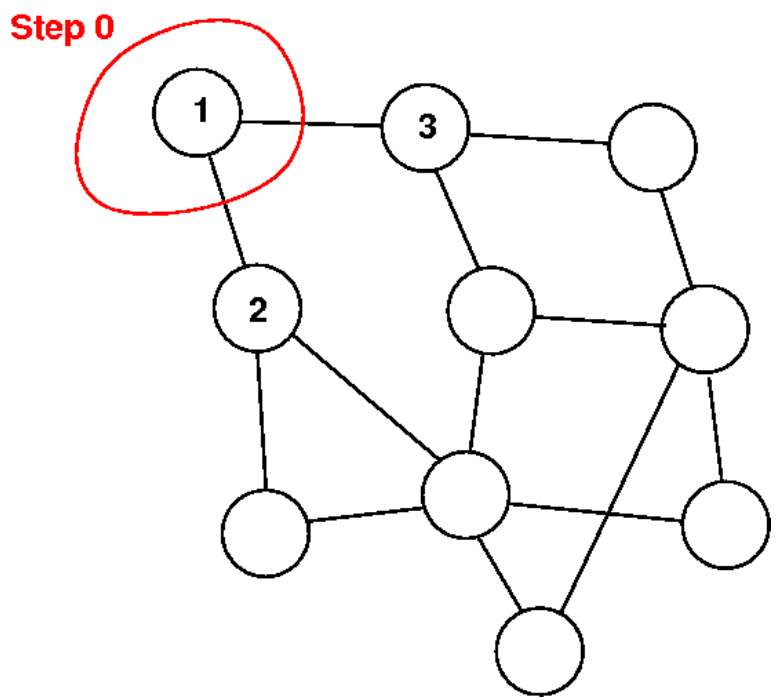
- **Step 0:** Choose a vertex with minimum degree
  ($\Rightarrow$ Greedy: corresponds often to locally minimum fill-in).



The degree of a vertex in a graph is the number of connecting edges.

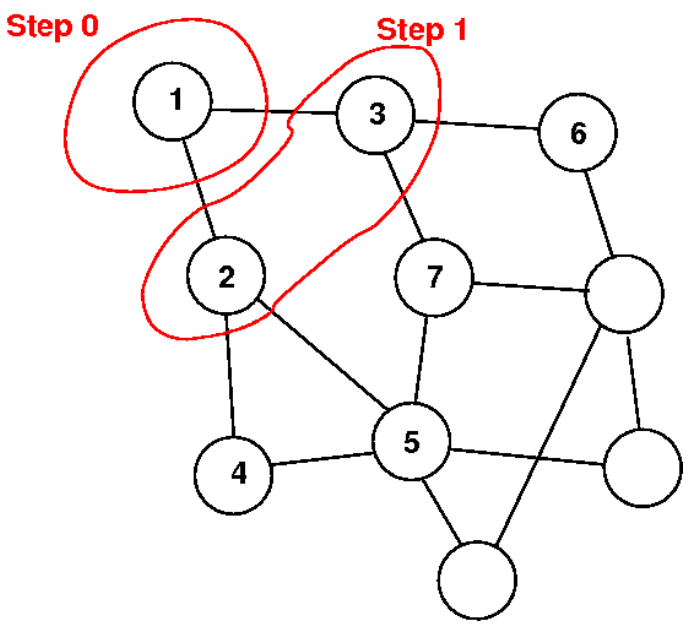
Reverse Cuthill-McKee algorithm (cont.)

- Step 0: Choose a vertex with minimum degree.
- **Step 1:** Number all neighbors of vertex 1 by increasing degree.



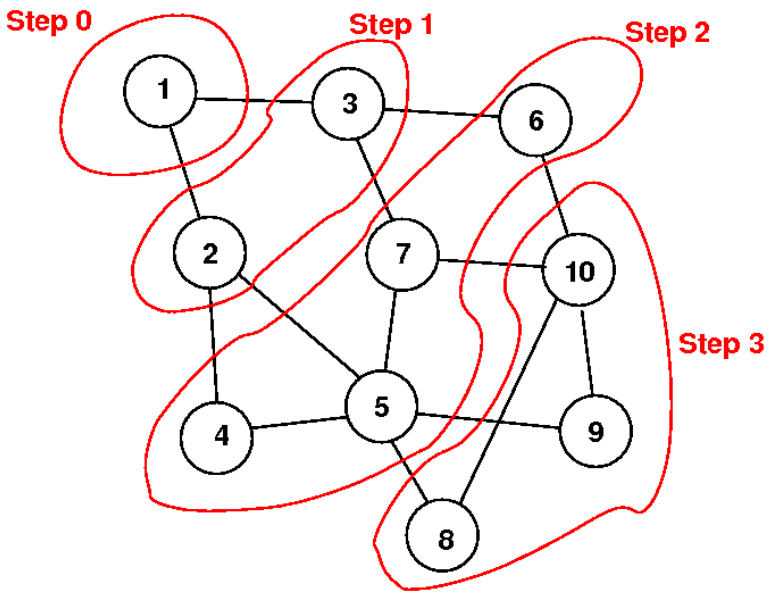
Reverse Cuthill-McKee algorithm (cont.)

- Step 0: Choose a vertex with minimum degree.
- Step 1: Number all neighbors of vertex 1 by increasing degree.
- **Step 2:** For *each* vertex from step 1, number all its remaining neighbors by increasing degree. etc.



Reverse Cuthill-McKee algorithm (cont.)

$$\begin{bmatrix}
1 & \times & \times & & & & & & & \\
\times & 2 & & \times & \times & & & & & \\
\times & & 3 & & & \times & \times & & & \\
 & \times & & 4 & \times & & & & & \\
 & \times & & \times & 5 & & \times & \times & \times & \\
 & & \times & & & 6 & & & & \times \\
 & & \times & & \times & & 7 & & & \times \\
 & & & & \times & & & 8 & & \times \\
 & & & & \times & & & & 9 & \times \\
 & & & & & \times & \times & \times & \times & 10
\end{bmatrix}$$



'Reverse' Cuthill-McKee (RCM, George 1971, Jennings 1977):
Empirical finding: bandwidth often better when order reversed

- What is the fill-in incurred in CM and RCM?
- Try to find out yourself!

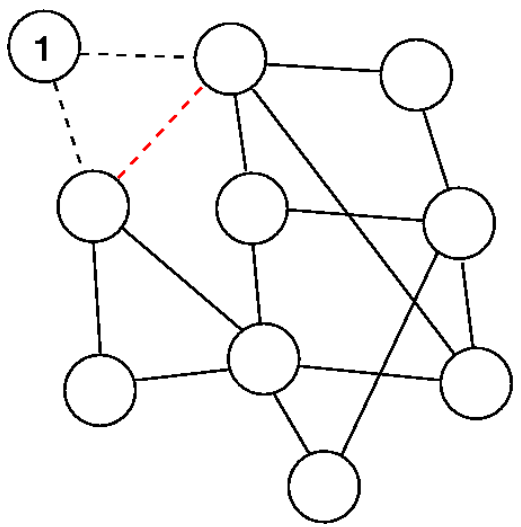
Minimum Degree Algorithm

- The Minimum Degree Algorithm is a heuristic for finding a permutation P such that PAP^T has fewer nonzeros in its factorization than A.
- The algorithm is greedy in that it trys to do the best in each iteration step. It selects the sparsest pivot row/column as the next pivot row.
- The symmetric approximate minimum degree algorithm is implemented in the MATLAB function `symamd`.
- The MATLAB function `amd` is applicable to all matrices and is claimed to be faster than `symamd`.

Minimum Degree Algorithm (cont.)

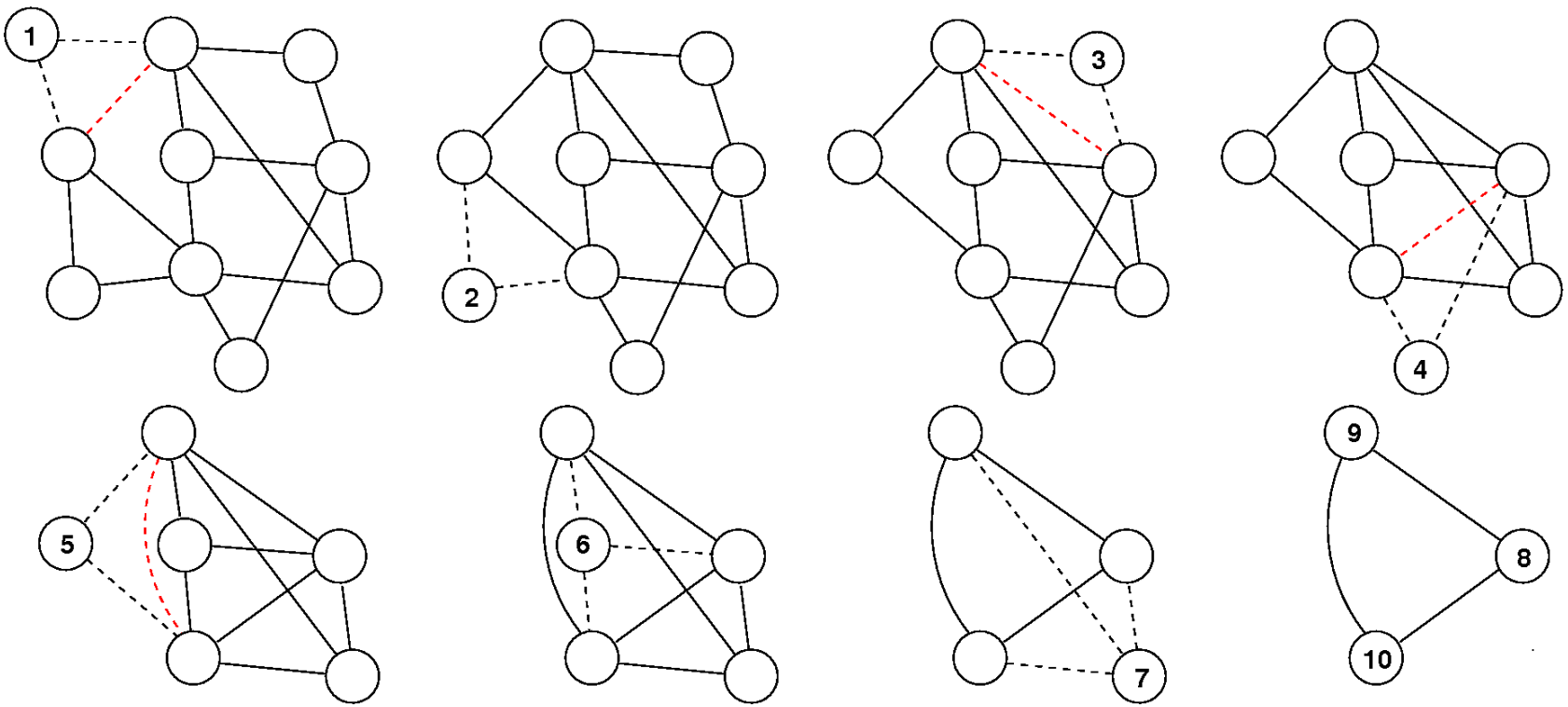
Include fill-in and update the vertex degree after elimination of a variable.

- Step 1: Choose among n vertices one with minimum degree as 1. Connect all neighbors of 1 with each other to a 'clique,' (fill-in). Remove vertex 1 and all its edges from graph.
- Step 2: repeat with reduced graph of $n-1$ vertices, etc.

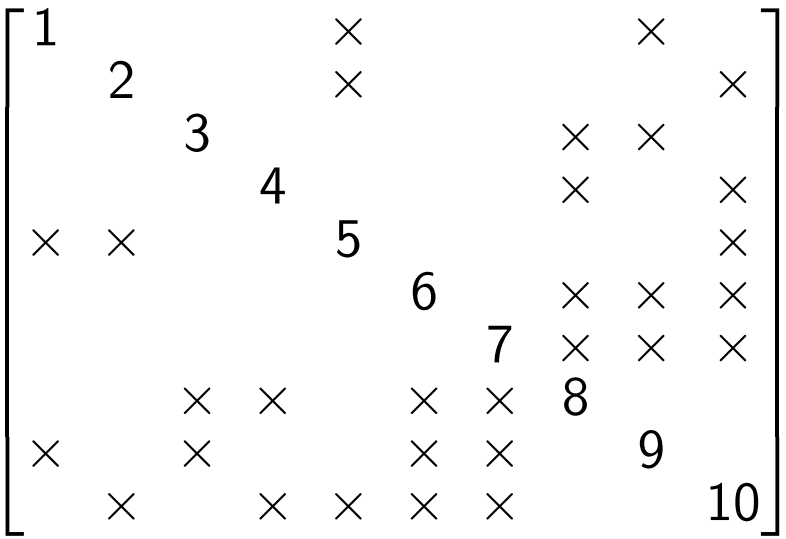


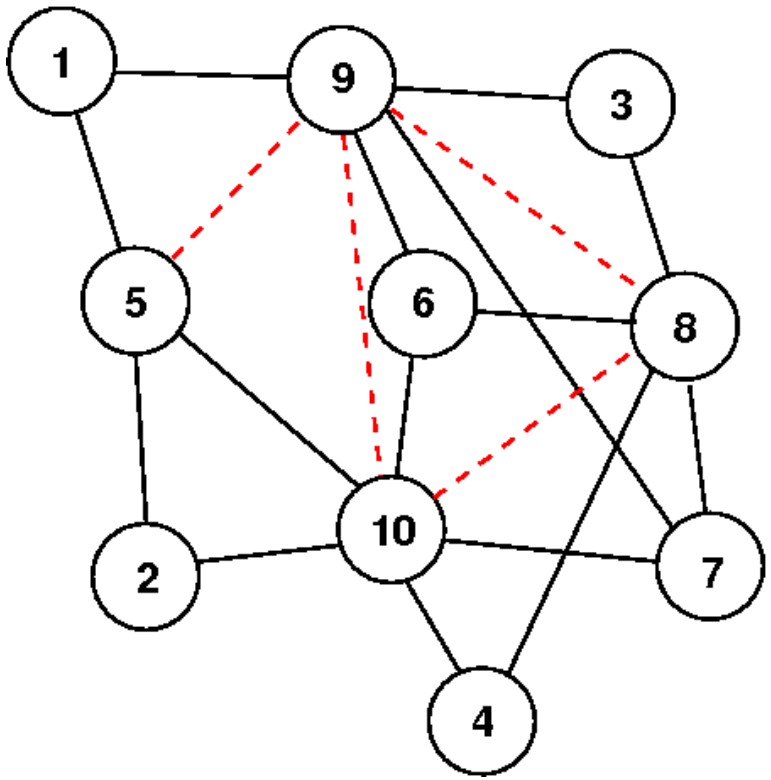
Minimum Degree Algorithm (cont.)

Complete execution for sample graph:



Minimum Degree Algorithm (cont.)

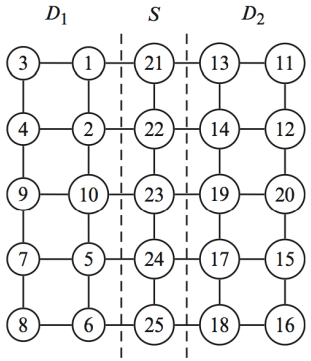




Nested Dissection

- The nested dissection matrix reordering strategy tries to reorder the matrix A, so that the fill-in is kept within certain matrix blocks in the factorisation.
- The graph of A is dissected into smaller subgraphs by separators,
- A separator S between two subgraph D_1 and D_2 is the set of vertices containing all paths between between D_1 and D_2.
- The rationale for making this dissection is that there can not be any fill-in L_{ij} with vertex i in D_1 and j in D_2.

Nested Dissection (cont.)



$$A = \begin{bmatrix} A_{11} & 0 & A_{1S} \\ 0 & A_{22} & A_{2S} \\ A_{S1} & A_{S2} & A_{SS} \end{bmatrix} = LL^T, \qquad L = \begin{bmatrix} L_{11} & 0 & 0 \\ 0 & L_{22} & 0 \\ L_{S1} & L_{S2} & L_{SS} \end{bmatrix},$$

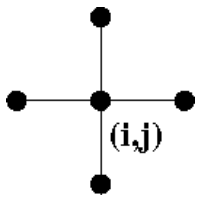
where the diagonal blocks $A_{ii}, i = 1, 2$, and A_{SS} stem from the vertex sets D_i and S, and the off-diagonal blocks A_{Si} stem from the edge intersection of these sets.

Due to the nested dissection any fill-in must occur in A_{ii} or A_{SS}.

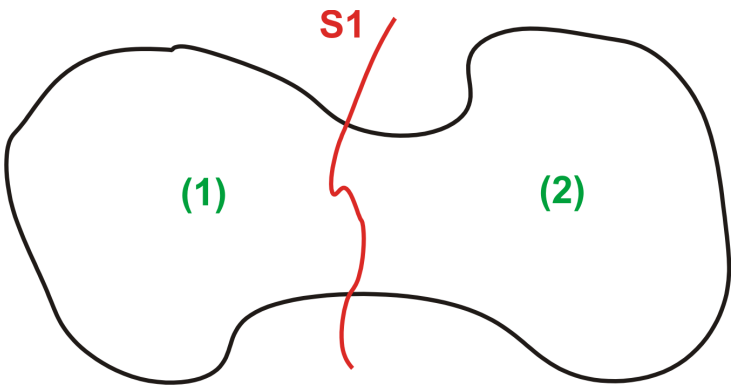
Domain Decomposition 1/3: Nested Dissection

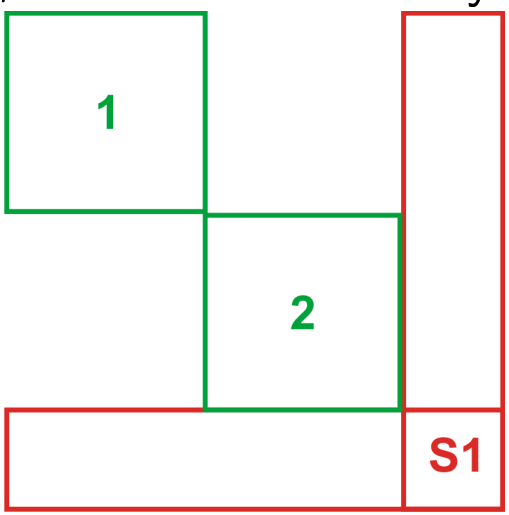
Approximation of 2D-Poisson equation using FD:

$$-\Delta u(\boldsymbol{x}_{i,j}) = f(\boldsymbol{x}_{i,j}),\ \ 0 < i,j < n{+}1$$



Numbering: first the two sub-domains, then the boundary points

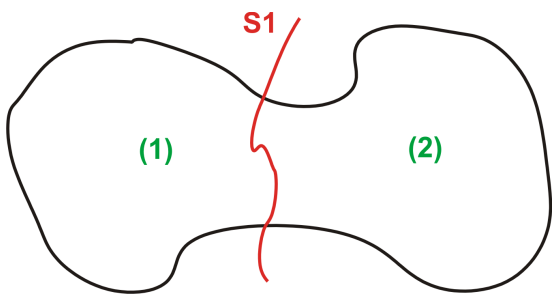


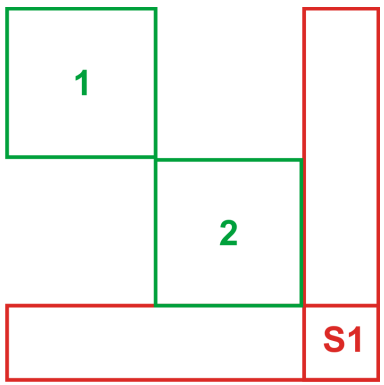


Only fill-in *within* the blocks of the 'block-bordered' matrix!

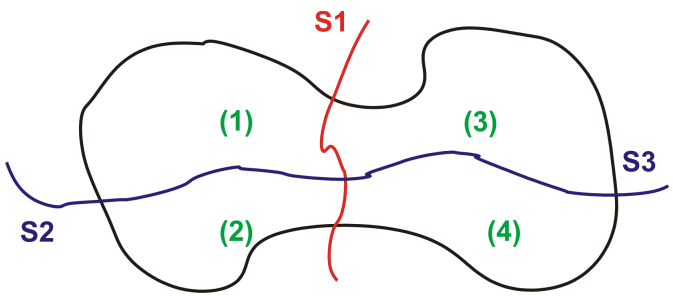
Domain Decomposition 2/3: Nested Dissection (recursive)

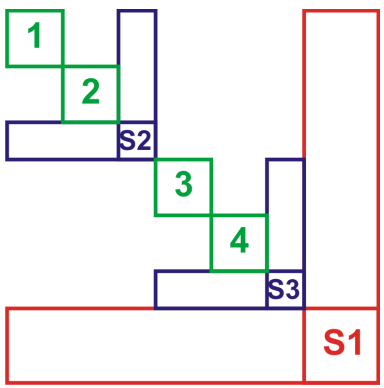
Numbering: first the two sub-domains, then the boundary points





Recursively:

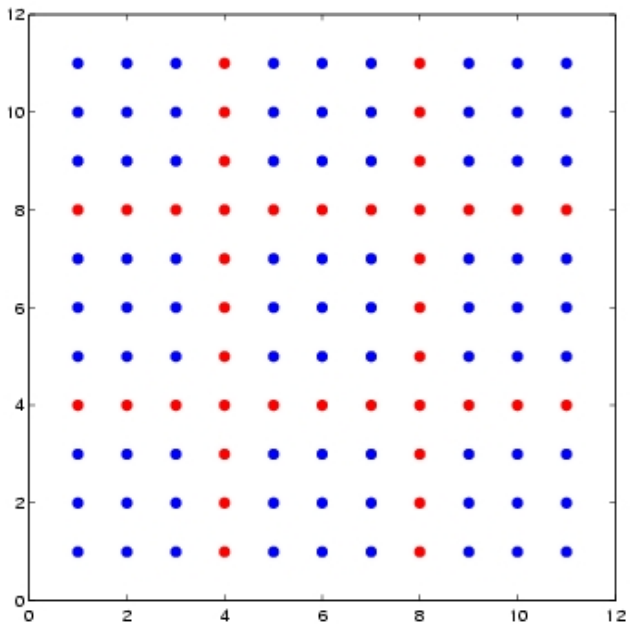


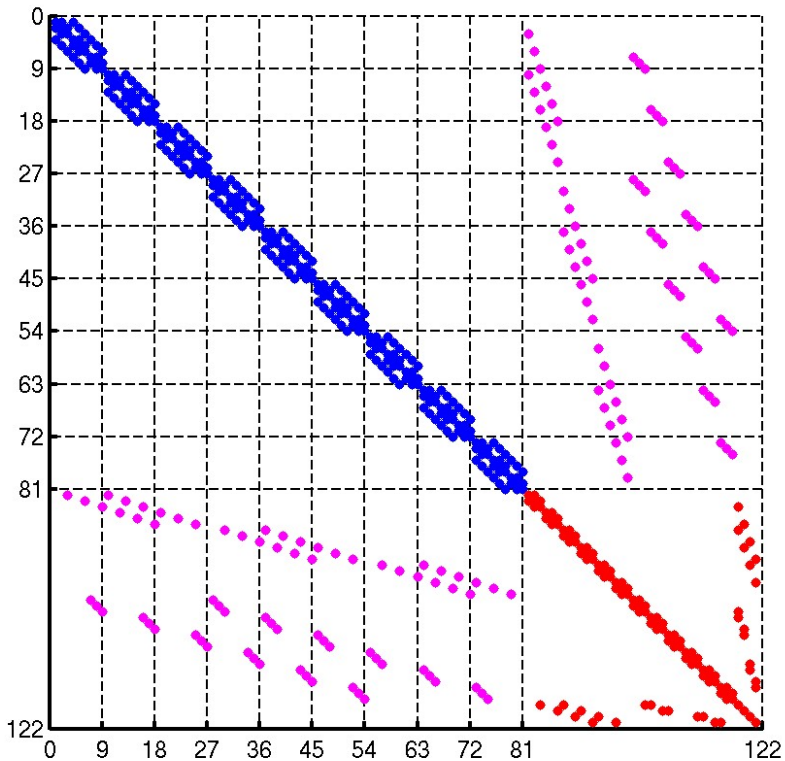


Only fill-in *within* the blocks of the 'block-bordered' matrix!

Domain Decomposition 3/3: Multisection

Generalization of Nested Dissection:





Numbering: first all sub-domains and then the boundary points

Combination with Maximum-Transversal Permutation

- So far: symmetric permutations to avoid fill-in
- Observation: A and PAP^T have the *same* entries on their diagonals (only in different positions)
- Strategy (for unsymmetric A):
  1. Unsymmetric permutation $A' = AQ$ to obtain large diagonal entries (Koster & Duff 1999)
  2. Symmetric permutation $A'' = PA'P^T$ for small fill-in

Idea: the larger the diagonal ('transversal'), the less one (probably) has to resort to pivoting

Remember: no pivoting if A is strictly diagonally dominant!

Algorithm: Matching of columns with rows (Koster & Duff)

Matching algorithm for nonsymmetric matrices

- Let $A \in \mathbb{R}^{n \times n}$ be a general matrix.
- The nonzero elements of A define a graph with edges $\mathcal{E} = \{(i,j) : a_{ij} \neq 0\}$ of ordered pairs of row and column indices.
- A subset $\mathcal{M} \subset \mathcal{E}$ is called a matching, or transversal, if every row index i and every column index j appears at most once in $\mathcal{M}$.
- A matching is called perfect if its cardinality is n.
- For a nonsingular matrix, at least one perfect matching exists and can be found with known algorithms.

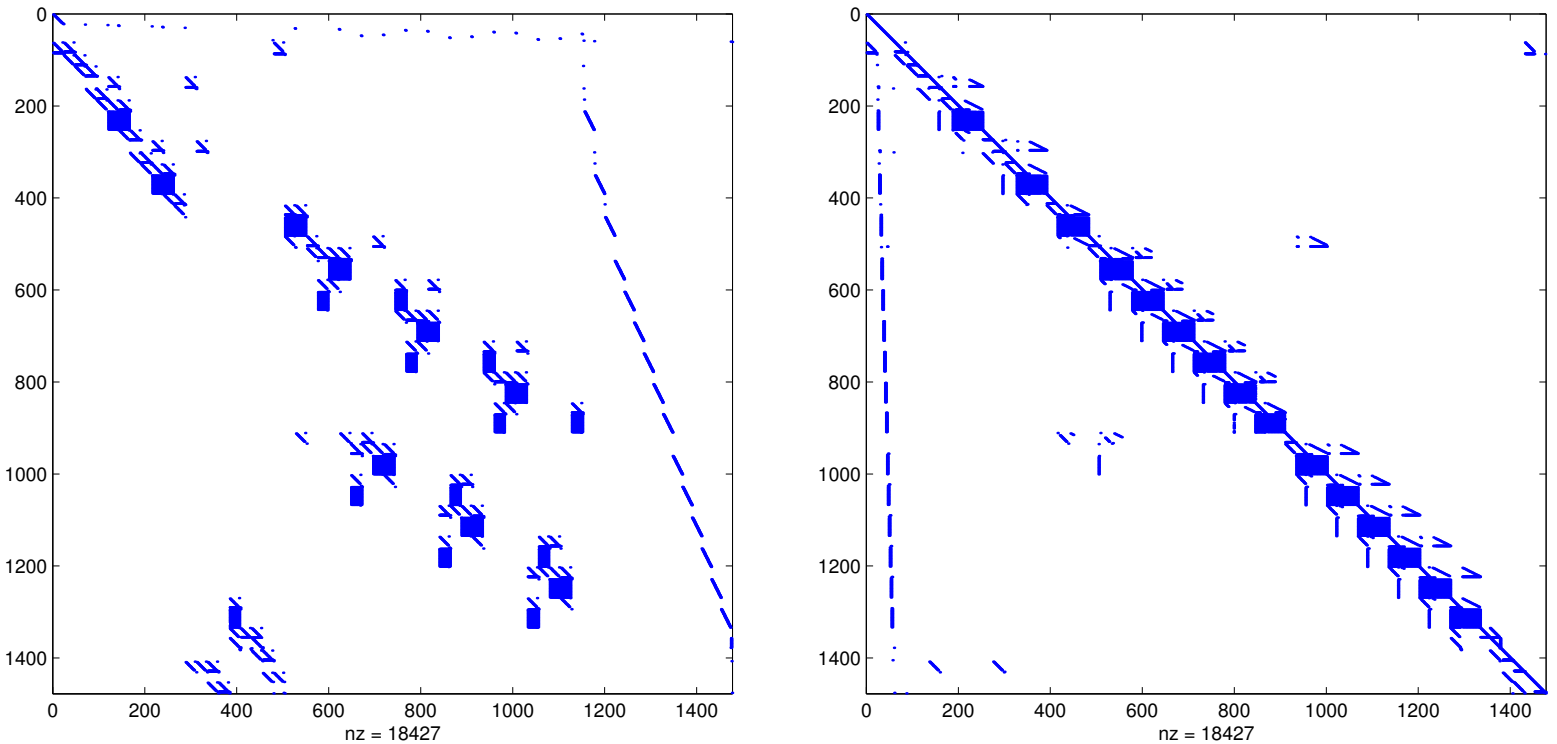
Matching algorithm for nonsymmetric matrices (cont.)

- With a perfect matching $\mathcal{M}$, it is possible to define a permutation matrix $P_{\mathcal{M}} = (p_{ij})$ with

$$p_{ij} = \begin{cases} 1 & (j,i) \in \mathcal{M}, \\ 0 & \textit{otherwise}. \end{cases}$$

  By consequence, $P_{\mathcal{M}}A$ has nonzero elements on its diagonal.
- This takes only the nonzero structure of the matrix into account.
- Other approaches maximize the diagonal values in some sense.
- One could try to find a matching such that the product of the diagonal values of $P_{\mathcal{M}}A$ is maximal.
- Problem in combinatorial optimization.

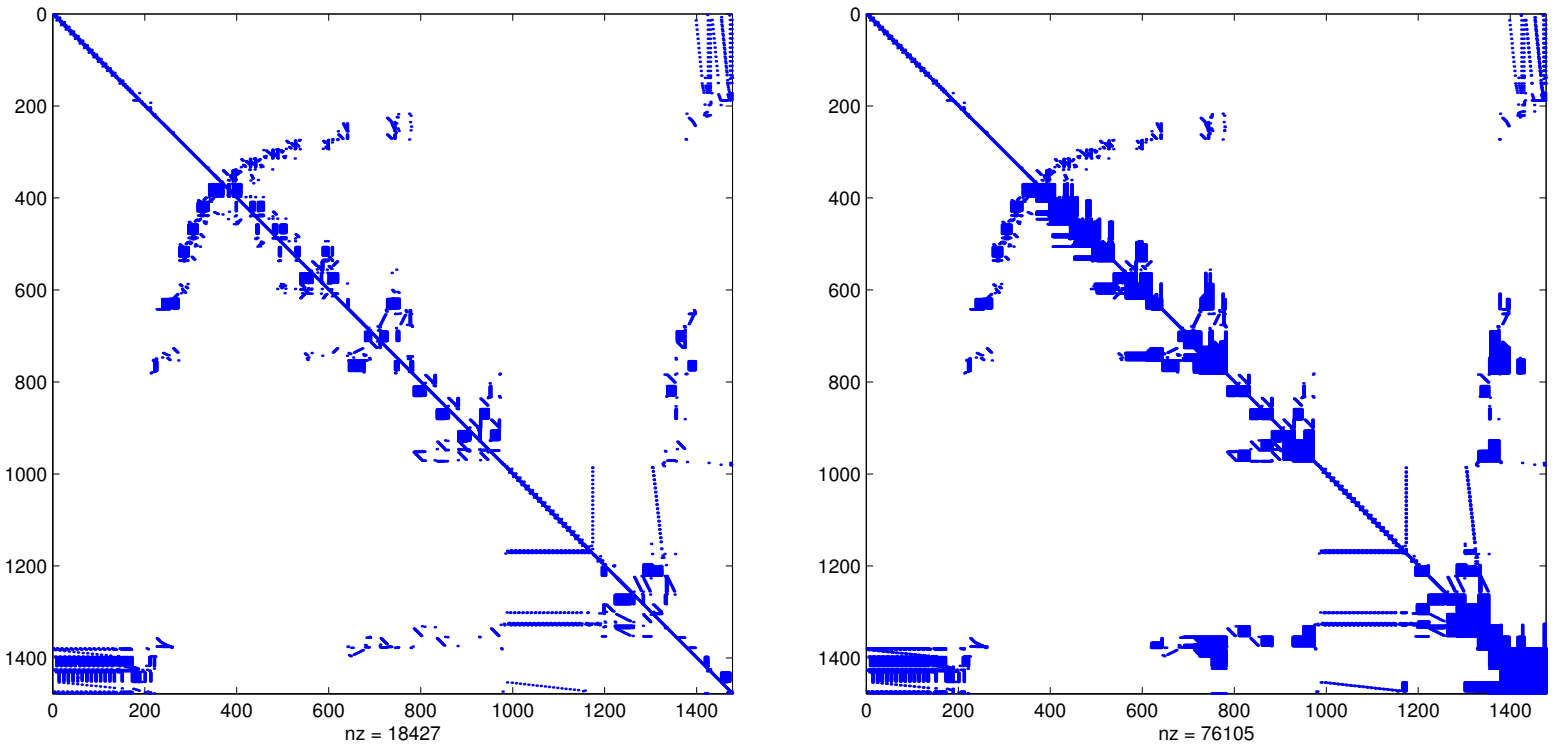
Practical effect of Maximum-Transversal Permutation



- Left: Original A
- Right: Column-permuted version $A' = AQ$

(Data from: Amestoy & L'Excellent: *Direct methods for sparse linear algebra*, CEA-EDF-INRIA School on High Performance Scientific Computing, 2006)

Symmetric reordering (AMD) and factorization



- Left: symmetric permutation $A'' = PA'P^T$
- Right: factorization $A'' = LU$

Available Software

- Sequential:
  - `UMFPACK` (Davis, U. Florida/Texas A&M): indep. / MATLAB / FreeFem++
  - `SuperLU` (Li, Berkeley)
- Parallel:
  - `SuperLU_DIST` (Li, Berkeley): distributed memory
  - `MUMPS` (Amestoy, L'Excellent, France): distributed memory

References

- T. Davis: *Summary of available software for sparse direct methods*. SIAM 2006.
- N. Gould, J. Scott, Y. Hu: *A Numerical Evaluation of Sparse Direct Solvers for [..] Symmetric Linear Systems of Equations*, 2007.

Exercise 6:

http://people.inf.ethz.ch/arbenz/FEM16/pdfs/ex6.pdf