NUMERICAL LINEAR ALGEBRA

Prof. Paola Antonietti
MOX - Dipartimento di Matematica
Politecnico di Milano

https://antonietti.faculty.polimi.it
TA: Dr. Michele Botti

POLITECNICO MILANO 1863 | DEPARTMENT OF MATHEMATICS

P8: Direct Methods for Linear Systems

A short introduction to direct methods for linear systems of equations

LU factorization

Let $A_{p,i} \in \mathbb{R}^{i,i}, i = 1,\ldots,n$ be the principal submatrixes of A obtained by considering the first rows and columns

$$A_{p,i}$$

$$\begin{bmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1n} \\ \vdots & & & & \\ a_{i1} & \cdots & a_{ii} & & \vdots \\ & & & & \\ a_{n1} & \cdots & & \cdots & a_{nn} \end{bmatrix}$$

$$det(A_{p,i}) \neq 0 \ \forall i = 1,\ldots,n-1$$

LU factorization. Let A be a square matrix. An LU factorization refers to the factorization of A into two factors: a lower unitary triangular matrix $L$and an upper triangular matrix U: $A = LU$.

Theorem: If A is invertible, then it admits an LU factorization if and only if all its leading principal minors are nonzero.

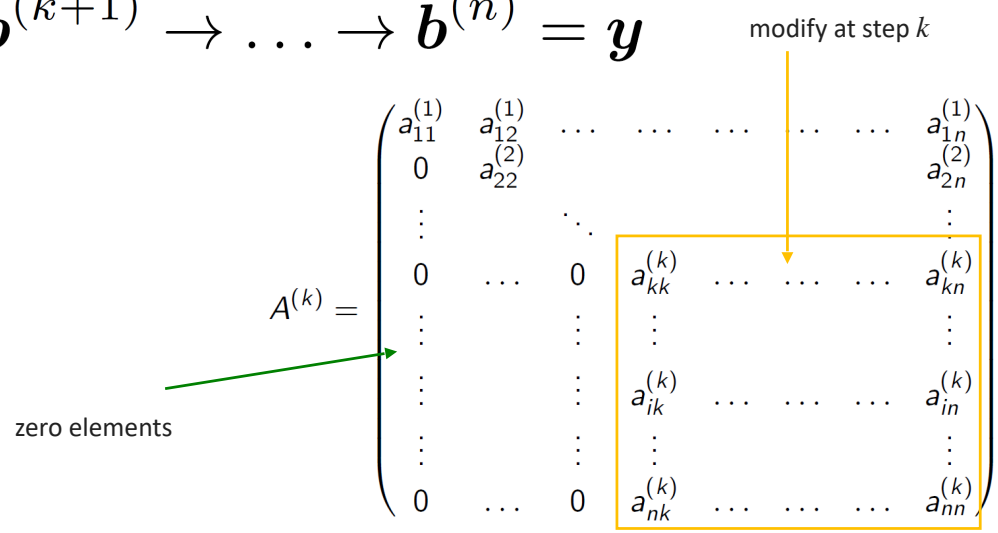
Gaussian elimination

It consists of a sequence of operations (swapping two rows, multiplying a row by a nonzero number, adding a multiple of one row to another row) performed on the corresponding matrix of coefficients, so as to "transform" A into an upper triangular matrix ($A^{(n)} = U$).

For $k = 1, \ldots, n$

$$A^{(1)} = A \rightarrow A^{(2)} \rightarrow \ldots \rightarrow A^{(k)} \rightarrow A^{(k+1)} \rightarrow \ldots \rightarrow A^{(n)} = U$$

$$\boldsymbol{b}^{(1)} = \boldsymbol{b} \rightarrow \boldsymbol{b}^{(2)} \rightarrow \ldots \rightarrow \boldsymbol{b}^{(k)} \rightarrow \boldsymbol{b}^{(k+1)} \rightarrow \ldots \rightarrow \boldsymbol{b}^{(n)} = \boldsymbol{y}$$



Gaussian elimination

To do this in practice, a multiple of row k is subtracted from the subsequent rows in order to cancel the desired elements

pivotal element (must be different from zero!)

For $k = 1, \ldots, n-1$

For $i = k+1, \ldots, n$

$$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$$

For $j = k+1, \ldots, n$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}$$

$$b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k^{(k)}$$

After n steps we have

$$A^{(n)} = U \qquad l_{ij} \to L \qquad \boldsymbol{b}^{(n)} = \boldsymbol{y}$$

A practical example

$$A = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} 11/6 \\ 13/12 \\ 47/60 \end{bmatrix}$$

Gaussian Elimination

$$\rightarrow A^{(2)} = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 0 & 1/12 & 1/12 \\ 0 & 1/12 & 4/45 \end{bmatrix} \qquad \boldsymbol{b}^{(2)} = \begin{bmatrix} 11/6 \\ 1/6 \\ 31/180 \end{bmatrix}$$

$$\rightarrow U = A^{(3)} = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 0 & 1/12 & 1/12 \\ 0 & 0 & 1/180 \end{bmatrix} \qquad \boldsymbol{y} = \boldsymbol{b}^{(3)} = \begin{bmatrix} 11/6 \\ 1/6 \\ 1/180 \end{bmatrix}$$

Backward substitution

$$x_3 = \frac{1/180}{1/180} = 1 \rightarrow x_2 = \frac{1/6 - 1/12 \cdot 1}{1/12} = 1$$

$$\rightarrow x_1 = \frac{11/6 - 1/2 \cdot 1 - 1/3 \cdot 1}{1} = 1$$

Gaussian elimination: computational costs

For any
$k = 1,\ldots,n$

$n-k$ flops

$2(n-k)^2$ flops

$2(n-k)$ flops

For $k = 1,\ldots,n-1$

For $i = k+1,\ldots,n$

$$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}},$$

For $j = k+1,\ldots,n$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}a_{kj}^{(k)}$$

$$b_i^{(k+1)} = b_i^{(k)} - l_{ik}b_k^{(k)}$$

$$\sum_{k=1}^{n-1}\left(2(n-k)^2 + 3(n-k)\right) = \sum_{p=1}^{n-1}\left(2p^2+3p\right) =$$

$$= 2\frac{n(n-1)(2n-1)}{6} + 3\frac{n(n-1)}{2} \sim 2/3n^3$$

Sufficient conditions for Gaussian elimination

1) A is strictly diagonally dominant by rows / columns

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \;\; i = 1, \ldots, n,$$

$$|a_{ii}| > \sum_{j \neq i} |a_{ji}|, \;\; i = 1, \ldots, n$$

2) A is SPD

$$\forall \boldsymbol{z} \in \mathbb{R}^n, \; \boldsymbol{z} \neq 0, \; \rightarrow \;\; \boldsymbol{z}^T A \boldsymbol{z} > 0$$

Cholesky factorisation (A is SPD)

Lat A be a SPD matrix of order n. Then, there exists a unique upper triangular matrix R with real and positive diagonal entries such that $A = R^T R$

Algorithm: Let $r_{11} = \sqrt{a_{11}}$. For $j = 2, \ldots, n$

$$r_{ij} = \frac{1}{r_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj} \right), \; i = 1, \ldots, j-1$$

$$r_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2}$$

Computational cost $\approx n^3/3$

Example

$$A = \begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix} \qquad det(A) \neq 0$$

The pivotal element is zero. $\Longrightarrow$ $a_{22}^{(2)} = 0$

Gaussian Elimination stops since

$$det(A_{p,2}) = 0$$

$$A^{(2)} = \begin{bmatrix} 1 & 1 & 3 \\ 0 & 0 & -4 \\ 0 & 3 & -5 \end{bmatrix} \Longrightarrow$$

By swapping the second and third rows (also of $\mathbf{b}^{(2)}$!) the algorithm works

per $k = 1, \ldots, n-1$

per $i = k+1, \ldots, n$

$$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}},$$

per $j = k+1, \ldots, n$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}$$

$$b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k^{(k)}$$

PIVOTING

PIVOTING TECHINQUES

The idea

If at the step k of Gaussian Elimination the pivotal element $a_{kk}^{(k)} = 0$ then I switch row k with row $i > k$, where the index i is chosen so that $a_{ik}^{(k)} \neq 0$.

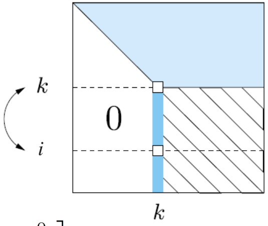
The pivoting strategy can be interpreted as a pre-multiplication of A (and b!) by a permutation matrix P. In our example

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \longrightarrow PA = LU$$

$$PA\boldsymbol{x} = P\boldsymbol{b} \longrightarrow LU\boldsymbol{x} = P\boldsymbol{b} \longrightarrow \begin{cases} L\boldsymbol{y} = P\boldsymbol{b} \\ U\boldsymbol{x} = \boldsymbol{y} \end{cases}$$

PIVOTING by rows

As a matter of fact , if at step k I swap row k with row $i > k$, I am pre-multiplying $A^{(k)}$ by a matrix obtained by exchanging the k-th and i-th rows in the identity matrix:



$$I = \begin{bmatrix} 1 & \dots & 0 & \dots & & & 0 \\ & \ddots & & & & & \\ 0 & \dots & 1 & 0 & 0 & 0 & 0 \\ & & & \ddots & & & \\ 0 & \dots & 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow P^{(k)} = \begin{bmatrix} 1 & \dots & 0 & \dots & & & 0 \\ & \ddots & & & & & \\ 0 & \dots & 0 & 0 & 1 & 0 & 0 \\ & & & \ddots & & & \\ 0 & \dots & 1 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} \\ \\ \leftarrow k \\ \\ \leftarrow i \\ \\ \end{matrix}$$

(columns marked: k, i)

$$A = A^{(1)} \to P^{(1)}A^{(1)} \to A^{(2)} \to P^{(2)}A^{(2)} \to A^{(3)} \to \dots \to A^{(n)}$$

$$P = P^{(n-1)}P^{(n-2)} \dots P^{(1)}$$

Pivoting by rows

The pivotal element should be as large as possible to "avoid" round-off errors.

In practice:

- do pivoting even when it is not strictly needed $(a_{kk}^{(k)} \neq 0)$
- Swap the row k whit the row $\bar{i}$, where

$$\bar{i} \ : \ |a_{\bar{r}k}^{(k)}| = \max_{i>k} |a_{ik}^{(k)}|$$

per $k = 1, \ldots, n-1$

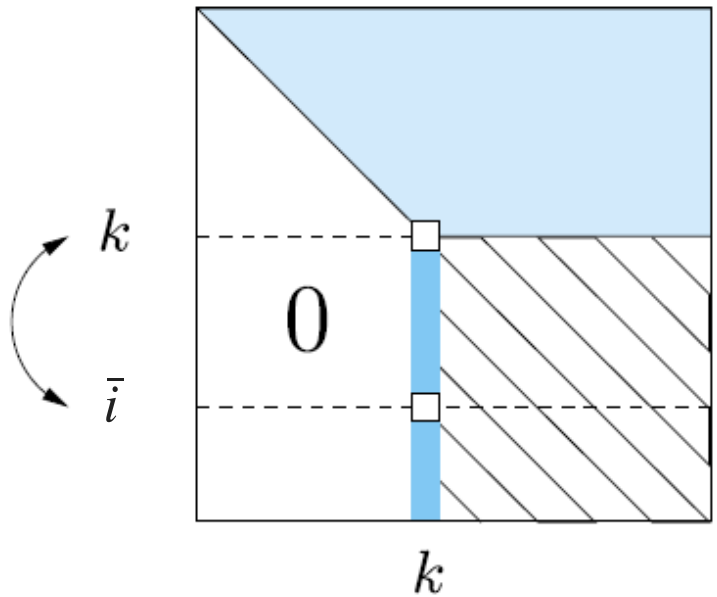
  per $i = k+1, \ldots, n$

$$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}},$$

    per $j = k+1, \ldots, n$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}$$

$$b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k^{(k)}$$



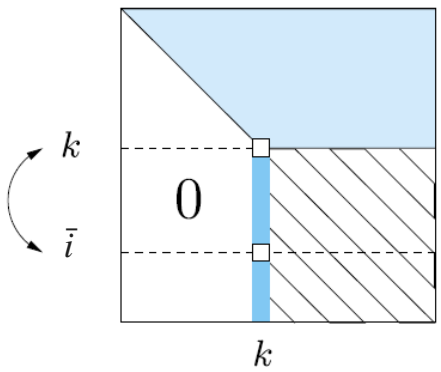
Complete pivoting

In the most general case, complete pivoting compares prospective pivots with all elements in the largest submatrix for which the prospective pivot is in the upper left position.

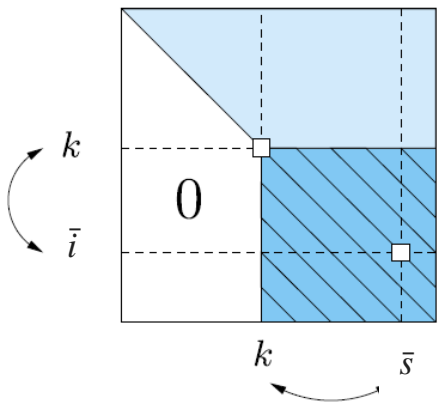
This is equivalent to introducing a second permutation matrix Q:

$$PAQ = LU$$

Partial pivoting (by rows)



Complete pivoting

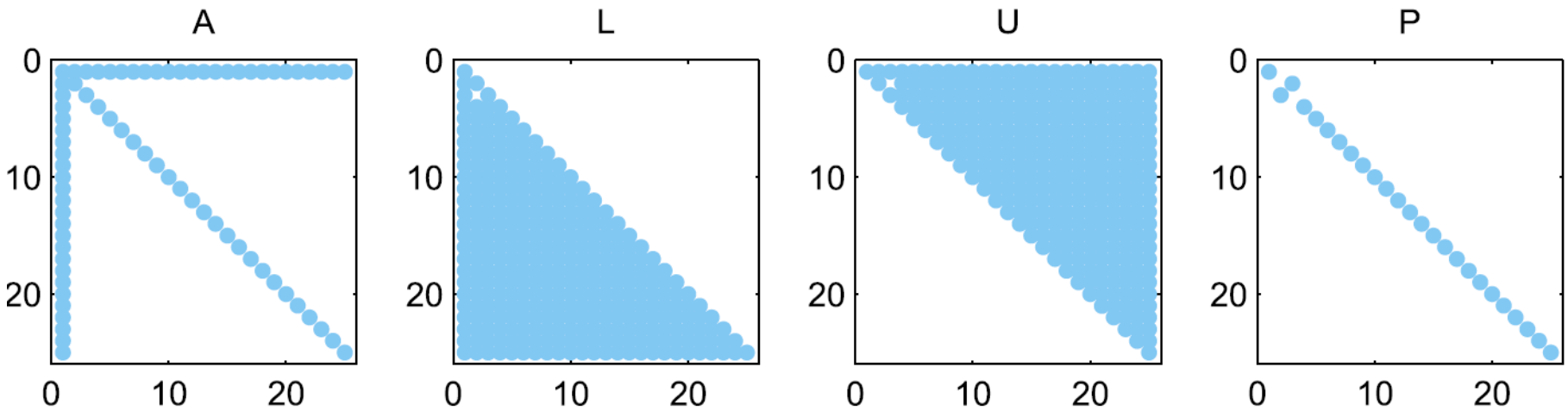


$$\mathrm{Ax} = \mathbf{b} \Leftrightarrow \underbrace{\mathrm{PAQ}}_{\mathrm{LU}} \underbrace{\mathrm{Q}^{-1}\mathbf{x}}_{\mathbf{x}^*} = \mathrm{Pb} \Longrightarrow \mathrm{Ly} = \mathrm{Pb} \qquad \mathrm{Ux}^* = \mathbf{y} \qquad \mathbf{x} = \mathrm{Qx}^*$$

FILL-IN

Fill-in

When A is sparse, LU decomposition results in L,U with nonzeros (called fill-in) at positions that were originally zero



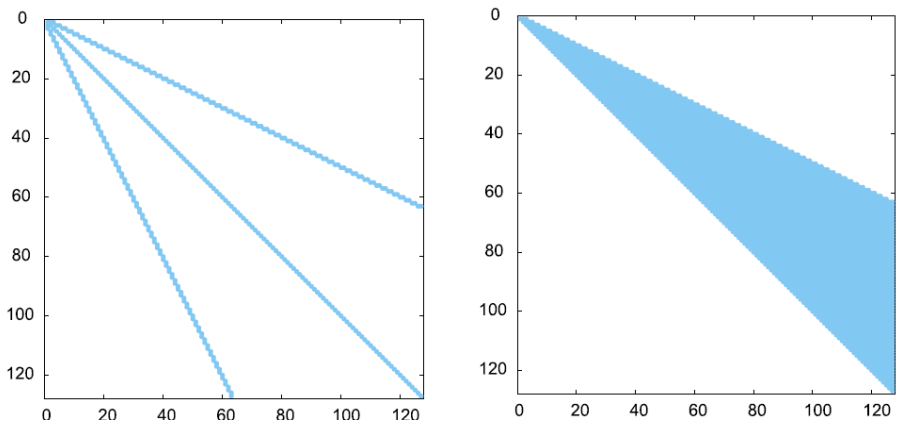
Gaussian Elimination (in its plain version) is made in such a way as to memorise L and U by overwriting the space allocated for A

⟹ LU factorisation is not suitable (from the point of view of memory occupation) for sparse matrices (fill-in reduction strategies, see later).

Fill-in reduction strategies

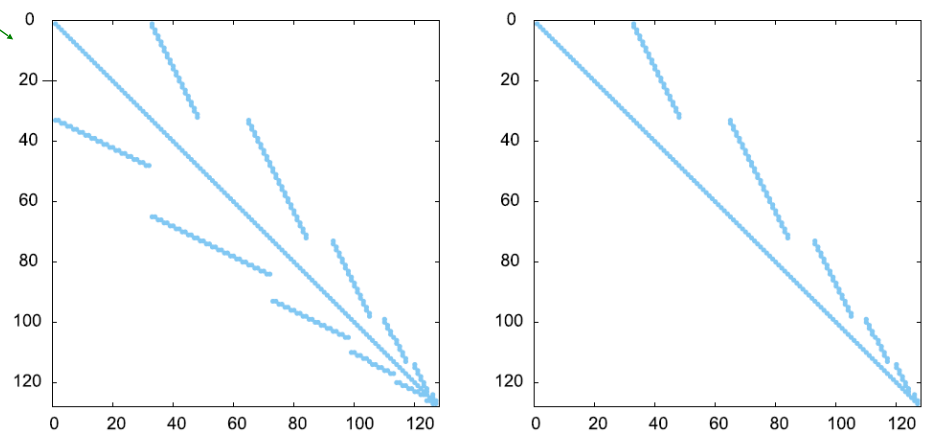
To reduce fill-in, it may be useful to employ reordering techniques, which consists in numbering the rows of A differently.

The aim is to reduce the number of nonzeros in L, U by permuting the nonzero structure of A into a special form and respecting this form when performing the reordering



A e U before re-ordering

A e U after re-ordering



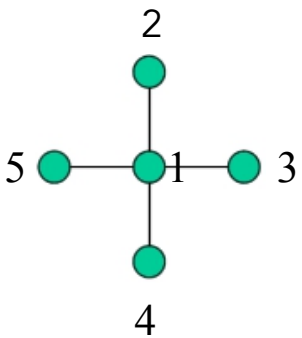
Direct Methods for Sparse Linear Systems

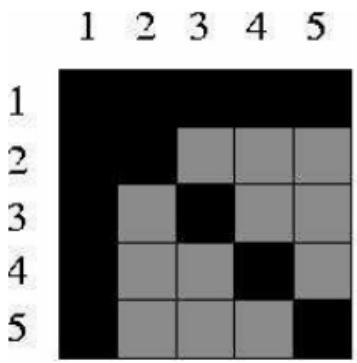
Sparse direct methods

Variable ordering has large impact on performance

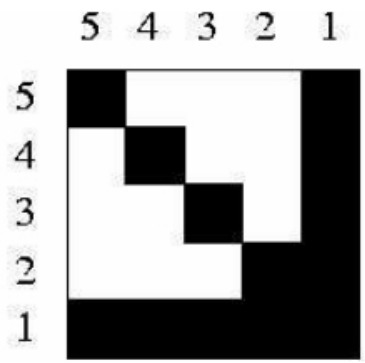
- Ordering impacts elimination tree, parallelism, computation, and memory use
- It easily pays off to spend time on analyzing the matrix before factorization
- However, finding the optimal ordering to minimize fill-in is an NP-hard problem

Use heuristics to study/optimize topology of the tree (number of nodes, sizes of nodes, …)

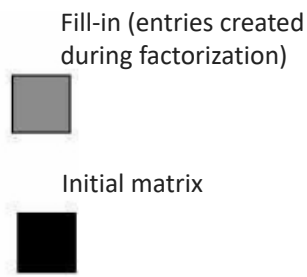




fill-in



no fill-in



Sparse direct methods

Three phases to solve $A\,\mathbf{x} = \mathbf{b}$:

1. Analysis of matrix A (**symbolic phase**):
   - Determine appropriate P and Q based on nonzero structure of A to minimize fill-in
   - Compute the elimination tree.
   - Prepare for parallel execution, map tree nodes onto processors.
   - Each processor prepares the local data structures.

2. Factorization of A (**numerical factorization**):
   - Compute the factors using the tree.
   - Possibly modify P and Q a-posteriori to ensure numerical stability.
3. Forward/backward substitution

The symbolic phase (idea)

1. Find a good **fill-reducing permutation**.
2. Once the ordering is found, the symbolic analysis finds the **elimination tree**, the nonzero pattern of the factorization or its key properties such as the number of non-zeros in each row and column of the factors.

The symbolic phase:

- is asymptotically faster than the numerical factorization phase
- allows the numerical phase to be more efficient in terms of time and memory.
- allows the numeric factorisation to be repeated for a sequence of matrices with identical nonzero pattern (it arises when solving non-linear and/or differential equations).

Fill-in reducing orderings

The fill-in minimization problem

The fill-in minimization problem can be stated as follows.
Given a matrix A, find a row and column permutation P and Q (with the added constraint that $Q = P^T$ for a sparse Cholesky factorization) such that the number of non-zeros in the factorization of PAQ (or the amount of work required to compute it) are minimized.

Computing an ordering for the minimum fill is NP-hard, in its many forms.

Approaches (1)

- Symmetric minimum degree. It is a widely-used heuristic for finding a permutation P so that $PA\,P^T$has fewer non-zeros in its factorization than A. It is a greedy method that selects the sparsest pivot row and column during the course of a right-looking sparse Cholesky factorization.

- Unsymmetric minimum degree

Approaches (2)

- Nested dissection. The goal of this ordering is the same as the minimum degree ordering; it is a heuristic for reducing fill-in. Consider the undirected graph of a matrix A with symmetric nonzero pattern. Nested dissection finds a vertex separator that splits the graph into two or more roughly equal-sized subgraphs (left and right), when the vertices in the separator (and their incident edges) are removed from the graph. The subgraphs are then ordered recursively.
- Permutations to block-triangular-form, and other special forms

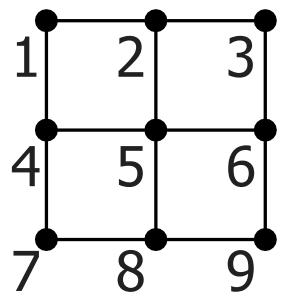
Sparse matrices and graphs

Symmetric sparse matrices and graphs

The structure of a square symmetric matrix A with nonzero diagonal can be represented by an undirected graph $G(A) = (V, E)$ with

- n vertices, one for each row/column of A
- an edge (i, j) for each nonzero $a_{ij}, i > j$

	1	2	3	4	5	6	7	8	9
1	X	X		X					
2	X	X	X		X				
3		X	X			X			
4	X			X	X		X		
5		X		X	X	X		X	
6			X		X	X			X
7				X			X	X	
8					X		X	X	X
9						X		X	X

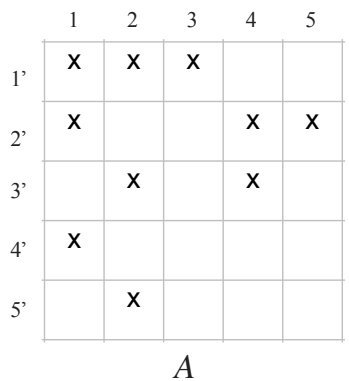


$G(A)$

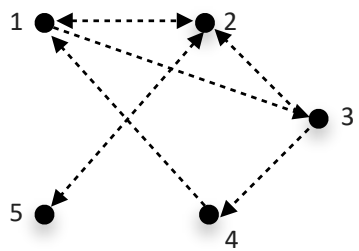
Non-symmetric sparse matrices and graphs

The structure of a non-symmetric matrix A of size $n \times n$ can be represented by

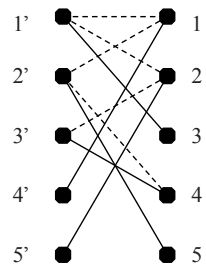
- a directed graph $G(A) = (V, E)$ with
  - n vertices, one for each column of A
  - an edge from i to j for each nonzero a_{ij}
- a bipartite graph $H(A) = (V, E)$ with
  - $2n$ vertices, for n rows and n columns of A
  - an edge (i', j) for each nonzero a_{ij}



A



$G(A)$



$H(A)$

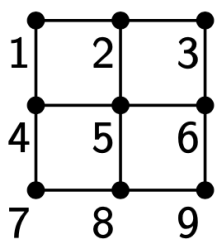
Filled graph $G^+(A)$, A SPD

- Given $G(A) = (V, E), G^+(A) = (V, E^+)$ is defined as follows there is an edge $(i, j) \in G^+(A)$ if and only if there is a path from i to j in $G(A)$ going through lower numbered vertices.
- The same definition holds also for directed graphs (LU factorisation).
- Useful remarks:
    - $G(R + R^T) = G^+(A)$ (ignoring cancellations).

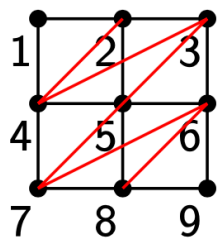
References: [Parter, 1961, Rose, 1970, Rose and Tarjan, 1978]

Filled graph $G^+(A)$, A SPD

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} x & x & & x & & & & & \\ x & x & x & & x & & & & \\ & x & x & & & x & & & \\ x & & & x & x & & x & & \\ & x & & x & x & x & & x & \\ & & x & & x & x & & & x \\ & & & x & & & x & x & \\ & & & & x & & x & x & x \\ & & & & & x & & x & x \end{pmatrix} \end{array} \qquad R + R^T = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} x & x & & x & & & & & \\ x & x & x & x & x & & & & \\ & x & x & x & x & x & & & \\ x & x & x & x & x & & x & & \\ & x & x & x & x & x & x & x & \\ & & x & & x & x & x & x & x \\ & & & x & x & x & x & x & \\ & & & & x & x & x & x & x \\ & & & & & x & & x & x \end{pmatrix} \end{array}$$



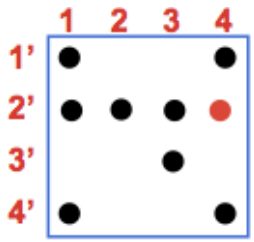
$G(A)$



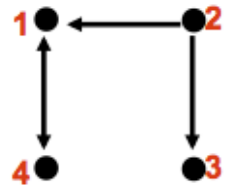
$G^+(A)$

Filled graph $G^+(A)$, A non-symmetric

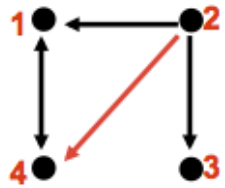
- A is square, non-symmetric (nonzero diagonal entries).
- Nonzero structure of L and U can be determined prior to the numerical factorisation from the structure of A
- Filled graph $G^+(A)$:
    - edges from rows to columns for all nonzeros of A ($G(A)$)
    - Add fill edge $i \rightarrow j$ if there is a path from i to j in $G(A)$ through lower numbered vertices.
- Remark: $G(L+U) = G^+(A)$ (ignoring cancellations)



A=LU

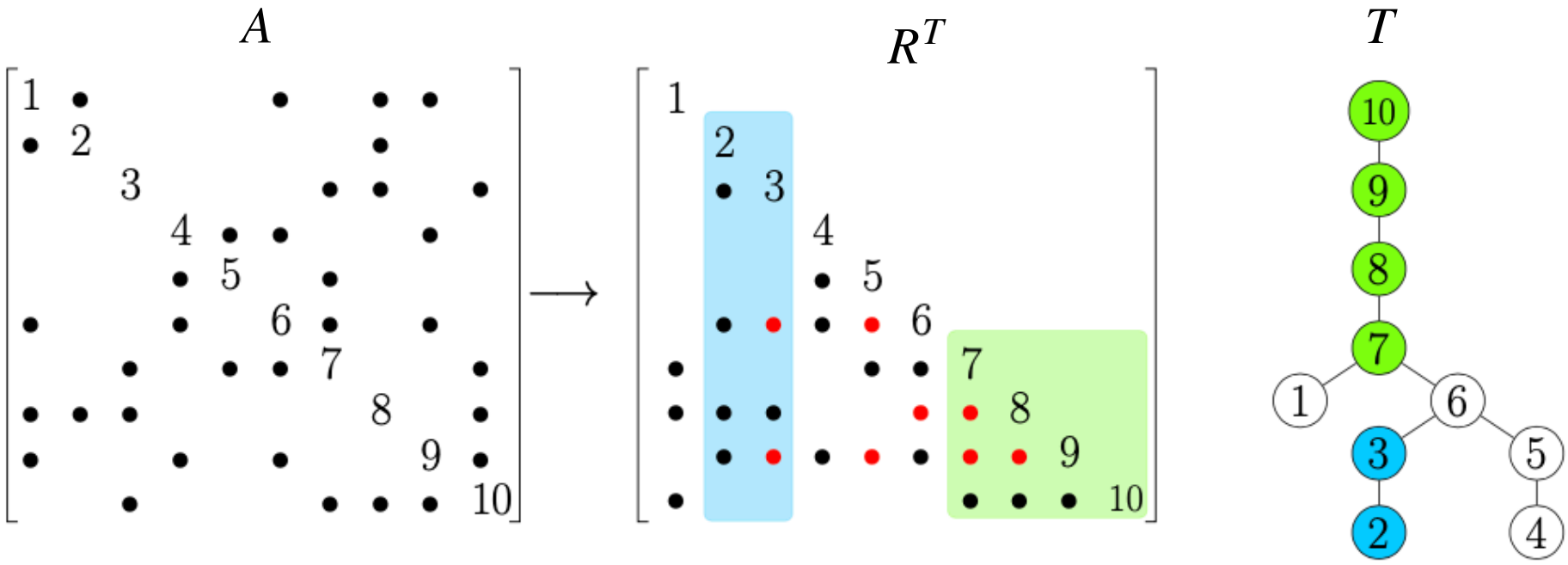


G(A)



$G^+(A)$

An example



The matrix A and its R factor from Cholesky factorization. The red nonzeros in R are fill-ins. The corresponding elimination tree (T) of A is also shown. Nodes in T and columns in R highlighted with the same color belong to the same supernode.

Steps of sparse Cholesky factorisation

1. Order rows and columns of A to reduce fill-in
2. Symbolic factorization (based on eliminaton trees)
    - Compute the elimination tree
    - Allocate data structure
    - Compute the nonzero structure of the factor R
3. Factorization
4. Triangular solve

Multifrontal direct methods

The multifrontal method organizes the operations that take place during the factorization of sparse matrices in such a way that the entire factorization is performed through partial factorizations of a sequence of dense and small submatrices.

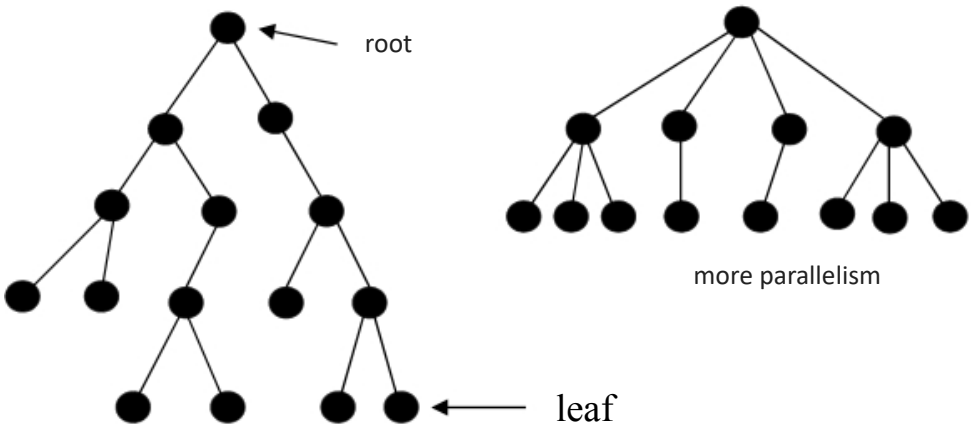
It is guided by a tree (elimination tree) that represents the dependencies between those partial factorizations.

Multifrontal direct methods

Basic idea: The factorization of a matrix A is driven by an elimination tree that is determined by the matrix structure and the variable ordering (permutations P and Q).

- A node in the tree represents a partial factorization of a (small) dense matrix.
- An edge in the tree represents data movement between dense matrices.
- The tree defines a partial ordering: a node can only be processed when all its children are processed.

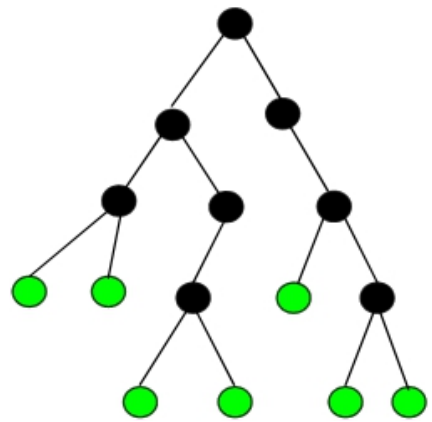
Leaves are processed first; root comes last. Nodes that are not ancestors of one another can be processed simultaneously (parallelism).



tridiagonal matrix with natural ordering; no parallelism

diagonal matrix; embarrassing parallel

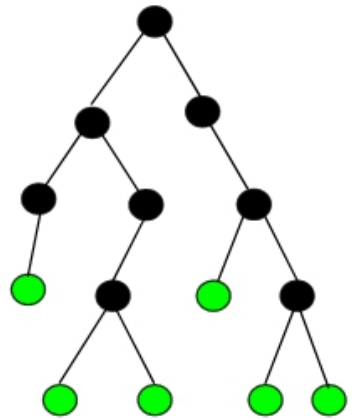
Multifrontal direct methods



7 nodes ready to be processed

8 nodes not ready to be processed

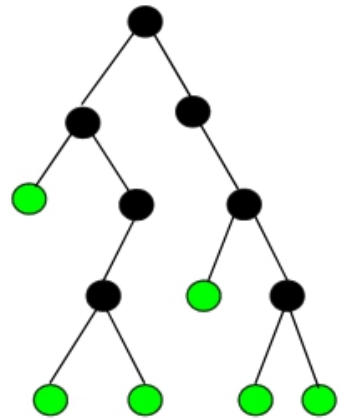
Multifrontal direct methods



6 nodes ready to be processed

8 nodes not ready to be processed

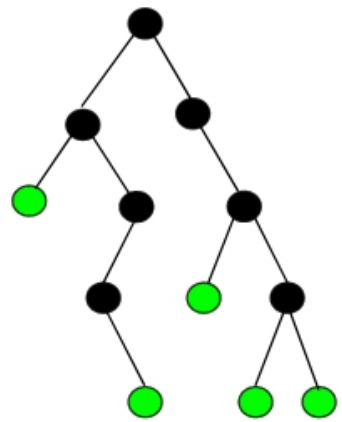
Multifrontal direct methods



6 nodes ready to be processed

7 nodes not ready to be processed

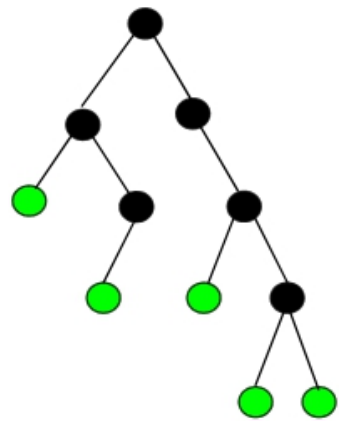
Multifrontal direct methods



5 nodes ready to be processed

7 nodes not ready to be processed

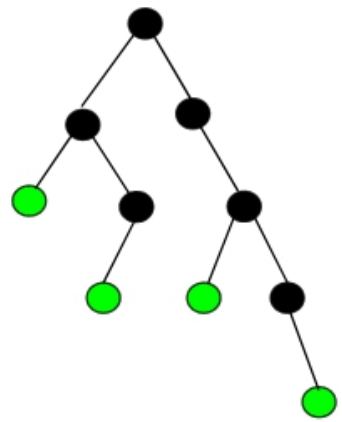
Multifrontal direct methods



5 nodes ready to be processed

6 nodes not ready to be processed

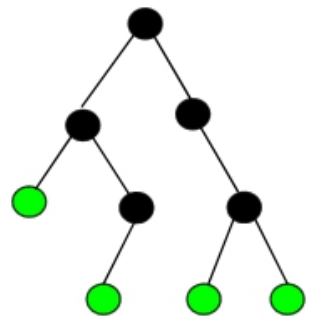
Multifrontal direct methods



4 nodes ready to be processed

6 nodes not ready to be processed

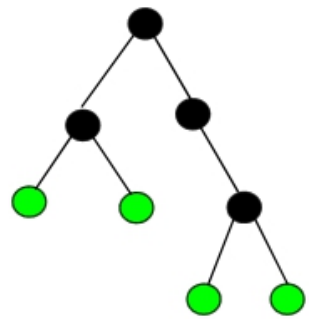
Multifrontal direct methods



4 nodes ready to be processed

5 nodes not ready to be processed

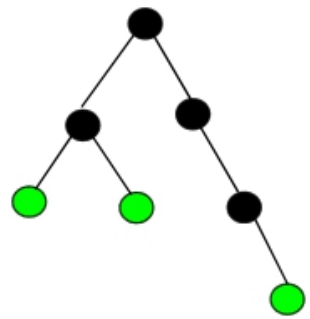
Multifrontal direct methods



4 nodes ready to be processed

4 nodes not ready to be processed

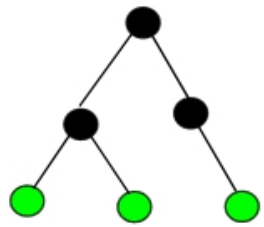
Multifrontal direct methods



3 nodes ready to be processed

4 nodes not ready to be processed

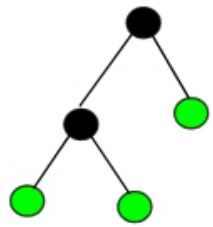
Multifrontal direct methods



3 nodes ready to be processed

3 nodes not ready to be processed

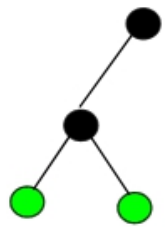
Multifrontal direct methods



3 nodes ready to be processed

2 nodes not ready to be processed

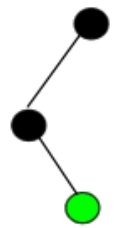
Multifrontal direct methods



2 nodes ready to be processed

2 nodes not ready to be processed

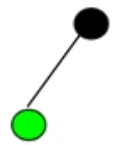
Multifrontal direct methods



1 nodes ready to be processed

2 nodes not ready to be processed

Multifrontal direct methods

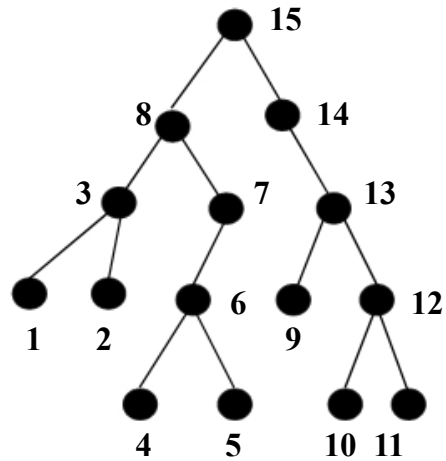


1 nodes ready to be processed

1 nodes not ready to be processed

Multifrontal direct methods

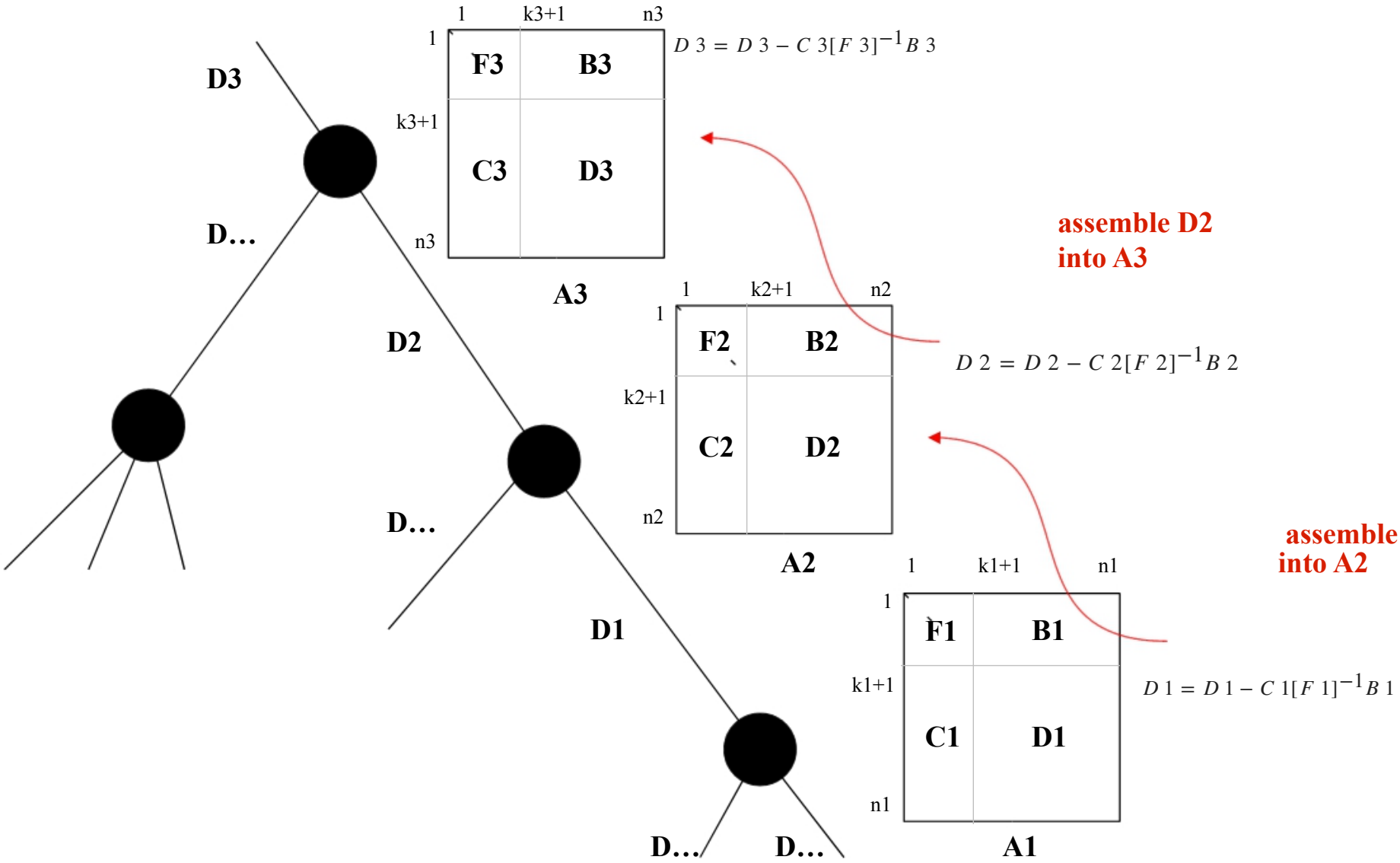
(DFS numbering)



1 node ready to be processed

A depth-first search (DFS) order allows the frontal matrices to be stored on a stack.

Multifrontal direct methods



Examples of direct solvers

An incomplete list of solvers and their characteristics:

- PSPASES: for SPD matrices, distributed memory.
- UMFPACK / SuiteSparse (Matlab, Google Ceres) - symmetric/non-symmetric, LU, QR, multicores/GPUs.
- SuperLU: non-symmetric matrices, shared/distributed memory.
- MUMPS: symmetric/non-symmetric, distributed memory.
- Pardiso (Intel MKL): symmetric/unsymmetric, shared/distributed memory.