Solving large sparse linear systems and least squares

Miroslav Tůma

Faculty of Mathematics and Physics

Charles University

mirektuma@karlin.mff.cuni.cz

SNA'25, Ostrava, January 2025

Outline

Introductory notes: resources / history

The main text resources are

- Jennifer Scott and Miroslav Tůma: Algorithms for sparse linear systems, Birkhäuser- Springer, 2023, open access.
- Jennifer Scott and Miroslav Tůma: Solving large sparse linear least squares, Acta Numerica, 2025, to appear.
- Further resources mentioned in these two texts

Our problems

Our two problems

- Systems of linear algebraic equations

$$\boxed{\mathbf{Ax} = \mathbf{b}, A \in \mathbb{R}^{n \times n}}$$

Our problems

Our two problems

- Systems of linear algebraic equations

$$\boxed{\mathbf{Ax} = \mathbf{b}, A \in \mathbb{R}^{n \times n}}$$

- Solving the (overdetermined) linear least squares (LS) problems

$$\boxed{\textbf{Given } A \in \mathbb{R}^{m \times n} \textbf{ of rank } n, m \geq n \textbf{ and } b \in \mathbb{R}^m}$$

$$\boxed{\textbf{find } x \in \mathbb{R}^n \textbf{ that minimises } \|b - Ax\|_2.}$$

Theorem

x is a solution of this least squares (LS) problem $\Longleftrightarrow$ x satisfies the $n \times n$ normal equations

$$Cx = A^T b, \quad C = A^T A$$

- LS $\longrightarrow$ one linear system. Enables to discuss LE and LS jointly

Motivation

LE: two principially different classes of solution methods

- I. Direct methods: heirs of Gaussian elimination, formulated as a) factorization, b) solution by substitution steps
  - a) Cholesky $A \to LL^T$ (A SPD), $A \to LU$ (A factorizable), indefinite factorizations, QR.
  - b) the factorized matrix used to find the solution (by substitution)
- An example: $Ax = b$: $A = LU$, $y = L^{-1}b$, $x = U^{-1}y$

Motivation

LE: two principially different classes of solution methods

- I. Direct methods: heirs of Gaussian elimination, formulated as a) factorization, b) solution by substitution steps
  - a) Cholesky $A \to LL^T$ (A SPD), $A \to LU$ (A factorizable), indefinite factorizations, QR.
  - b) the factorized matrix used to find the solution (by substitution)
- An example: $Ax = b$: $A = LU$, $y = L^{-1}b$, $x = U^{-1}y$
- II. Iterative methods
- Compute a sequence of approximations $x^{(0)}, x^{(1)}, x^{(2)}, \ldots$ that hopefully converges to the solution x of the linear system.
- Various approaches
  - Stationary iterative methods (linearly convergent)
  - Krylov subspace methods (typically more efficient)
  - (Some) convergence theory for both classes of methods

Motivation

Direct methods and iterative methods once more

- Direct methods
  - Designed to solve the systems of equations.
  - Properly implemented: they are robust, often predictable accuracy.
  - They can be expensive, requiring large amounts of memory.
- Iterative methods
  - Designed to approximate (not solve)
  - This may be an advantage if only an approximate solution is needed
  - Can be terminated as soon as the required accuracy is achieved
  - But this may be also a disadvantage (if matrix properties prohibit achieving the required accuracy, stopping iterations)

Motivation

Which approach is better? Complexity?

- First idea: operation counts
- In direct methods it seems to us that most of the work is in the factorization, less in the substitution steps.
  - Fully populated A: n^2 entries
    - $1/3n^3 + O(n^2)$ complexity of Cholesky
    - $2/3n^3 + O(n^2)$ complexity of LU (factorizable A)
    - Substitution steps: only $O(n^2)$

Motivation

Which approach is better? Complexity?

- First idea: operation counts
- In direct methods it seems to us that most of the work is in the factorization, less in the substitution steps.
  - Fully populated A: n^2 entries
    - $1/3n^3 + O(n^2)$ complexity of Cholesky
    - $2/3n^3 + O(n^2)$ complexity of LU (factorizable A)
    - Substitution steps: only $O(n^2)$
- In iterative methods this can be less. Like $O(n^{5/2})$ in the fully-populated model cases (that fulfill an assumption on the convergence).
  - The issue of convergence of iterative methods is much more complicated, not discussed here

Motivation

Which approach is better? Complexity?

- The complexity issues are more involved: not only due to different matrix properties, but also due to hardware for computation and communication
  - Nowadays, nearly nothing is really sequential
  - CPU $\rightarrow$ a mixture of powerful processors, coprocessors, cores, GPUs, and so on.
  - Furthermore, arithmetic operations are much faster than communication-based operations. And can be even accelerated by less accurate computation.

Motivation

Which approach is better? Complexity?

- The complexity issues are more involved: not only due to different matrix properties, but also due to hardware for computation and communication
  - Nowadays, nearly nothing is really sequential
  - CPU $\rightarrow$ a mixture of powerful processors, coprocessors, cores, GPUs, and so on.
  - Furthermore, arithmetic operations are much faster than communication-based operations. And can be even accelerated by less accurate computation.
- All of this helps to push the research on.

The question which class is better to solve our problems is ill-posed

Motivation

Iterative methods complement approximate direct methods

- Direct methods may provide a less accurate solution due to possible relaxations.
- Making solution more accurate: use preprocessing or postprocessing by an auxiliary iterative method.

Motivation

Iterative methods complement approximate direct methods

- Direct methods may provide a less accurate solution due to possible relaxations.
- Making solution more accurate: use preprocessing or postprocessing by an auxiliary iterative method.

Approximate direct methods complement iterative methods

- Pure iterative methods converge typically poorly. Or may have a low final attainable accuracy.
- Should be accompanied by a problem transformation based on a preconditioner. As:

$$MAx = Mb \quad \text{or} \quad AMy = b, y = Mx$$

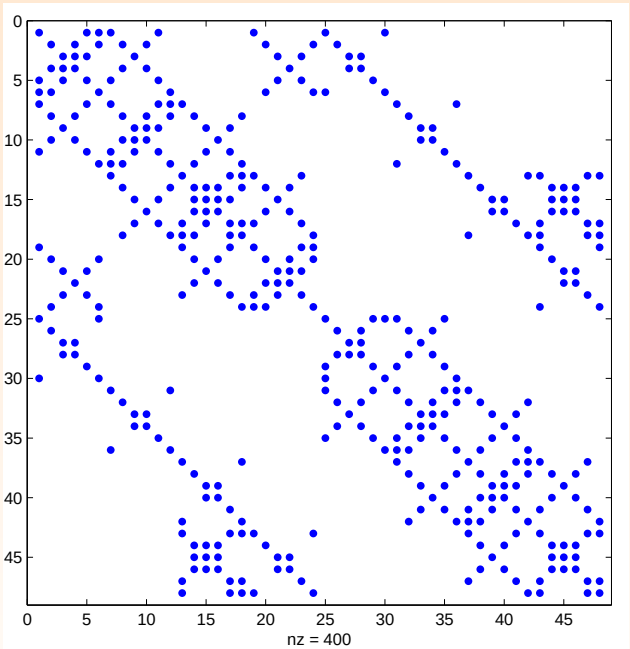
- Preconditioner M may approximate A or A^{-1}.

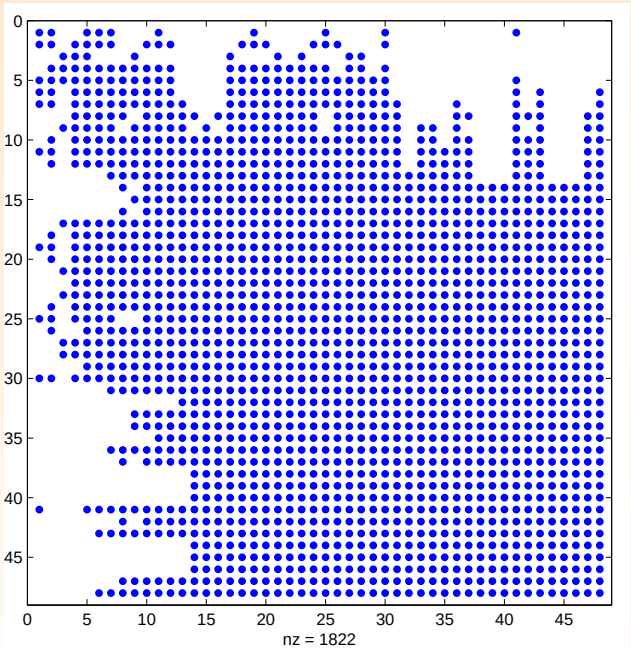
Borderline between the use of direct and iterative methods is fuzzy

Motivation

What else? The structure

- Matrix may contain a lot of zeros
- Nonzeros in A (left) and its factors (right) look like:

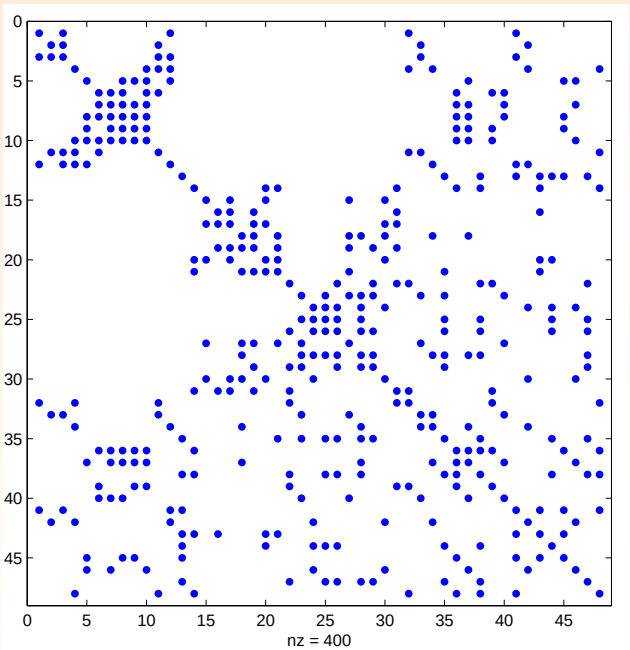


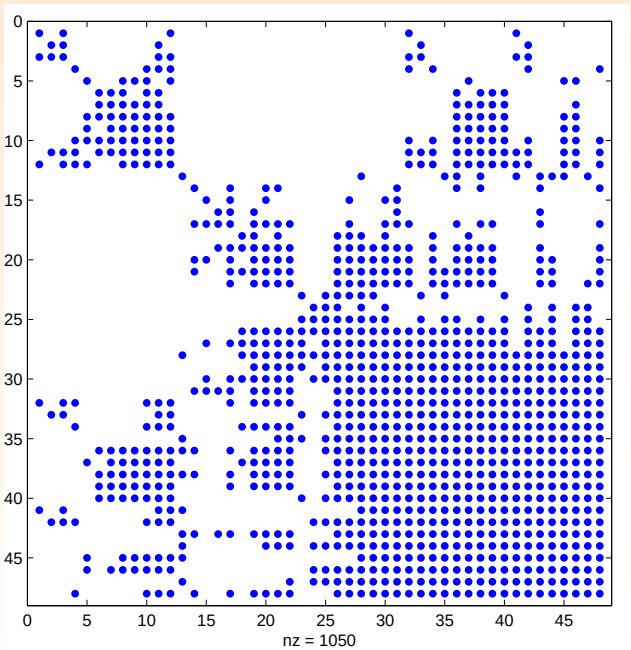


Motivation

What else? The structure

- Matrix may contain a lot of zeros.
- Nonzeros in A (left).
- Nonzeros in its factors (right) can look like much better if A was preprocessed by a reordering: $A \to PAP^T$, $A \to PAQ$.

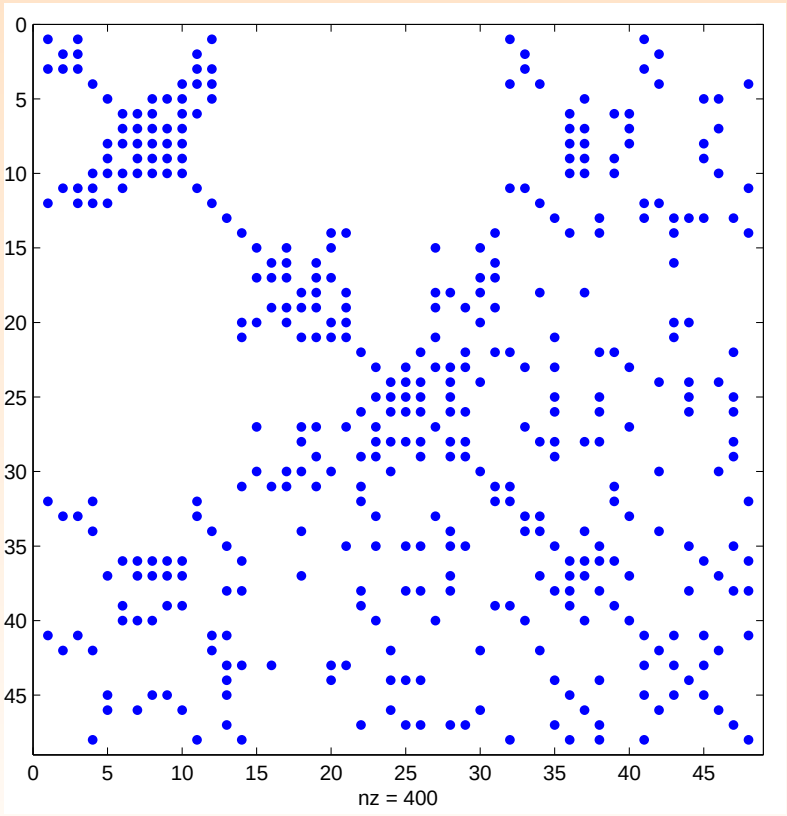




Basic Terminology: sparsity

Sparsity: so let us define it

- A is a sparse matrix if many of its entries are zero.



The sparsity pattern: $\mathcal{S}\{A\} = \{(i,j) \mid a_{ij} \neq 0,\ 1 \leq i,j \leq n\}$.

Basic Terminology: sparsity

Sparsity: more formally

- Attempts to formalize the sparsity more precisely like:

Definition

Matrix $A \in \mathbb{R}^{m \times n}$ is said to be sparse if it has $O(\min\{m, n\})$ nonzero entries. Another possibility: if A has row counts bounded by $r_{max} << n$ and/or column counts bounded by $c_{max} << m$.

Definition

Matrix $A \in \mathbb{R}^{m \times n}$ is said to be sparse if its number of nonzero entries is $O(n^{1+\gamma})$ for some $\gamma < 1$.

Basic Terminology: sparsity

Sparsity: more formally

- Attempts to formalize the sparsity more precisely like:

Definition

Matrix $A \in \mathbb{R}^{m \times n}$ is said to be sparse if it has $O(\min\{m, n\})$ nonzero entries. Another possibility: if A has row counts bounded by $r_{max} << n$ and/or column counts bounded by $c_{max} << m$.

Definition

Matrix $A \in \mathbb{R}^{m \times n}$ is said to be sparse if its number of nonzero entries is $O(n^{1+\gamma})$ for some $\gamma < 1$.

Definition

(pragmatic, application-based definition: J.H. Wilkinson) Matrix $A \in \mathbb{R}^{m \times n}$ is said to be sparse if the fact that a part of its entries is equal to zero can be (algorithmically) exploited.

Motivation

Is the (sparsity) structure really so important?

- Our claim is: yes, it is.
- But, should be used jointly with other computational concepts.
- Let us mention two new and interesting concepts: to show that the importance of exploiting sparsity is not disappearing.

Motivation

Is the (sparsity) structure really so important?

- Our claim is: yes, it is.
- But, should be used jointly with other computational concepts.
- Let us mention two new and interesting concepts: to show that the importance of exploiting sparsity is not disappearing.

Concept 1: low-rank approximation

- Blocks expressed as products of matrices of low-rank: $B \in \mathbb{R}^{k \times l} \rightarrow B = EF^T$ with $E \in \mathbb{R}^{k \times r}$ and $F \in \mathbb{R}^{l \times r}$
- The two factors of low rank a) may occupy less memory, b) may be cheaper in matrix-matrix products (matvecs).

Motivation

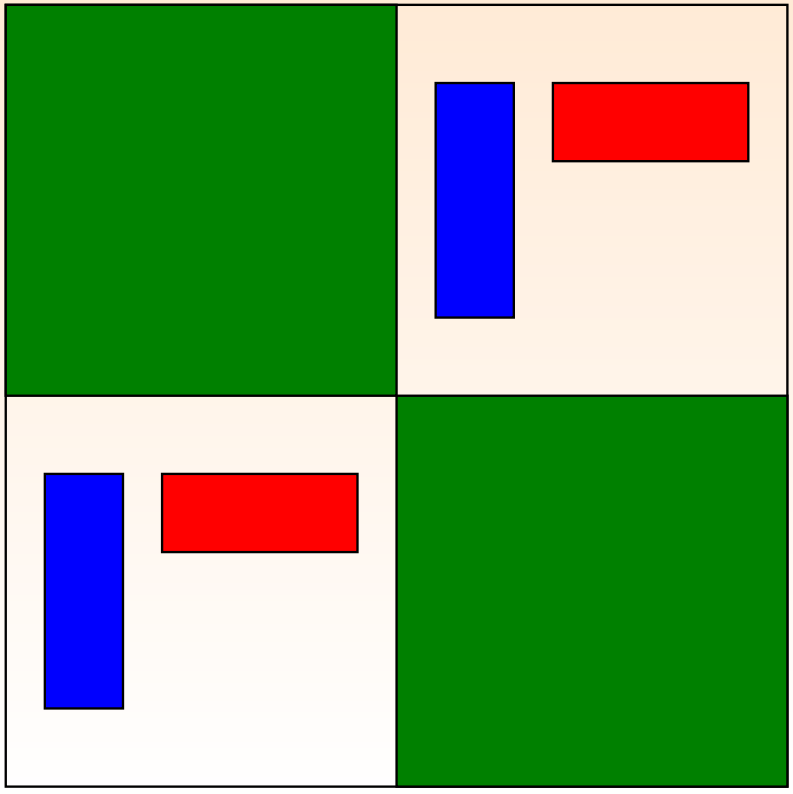
Accelerating by low-rank compression

- Often implied by appplications
- Like: panel clustering in BEM (Hackbusch, Nowak, 1989), the multipole method (Greengard, Rokhlin, 1997), mosaic-skeleton approximations (Tyrtyshnikov, 1996) etc.: an example of a hierarchical (data-sparse) matrix:

Motivation

Accelerating by low-rank compression

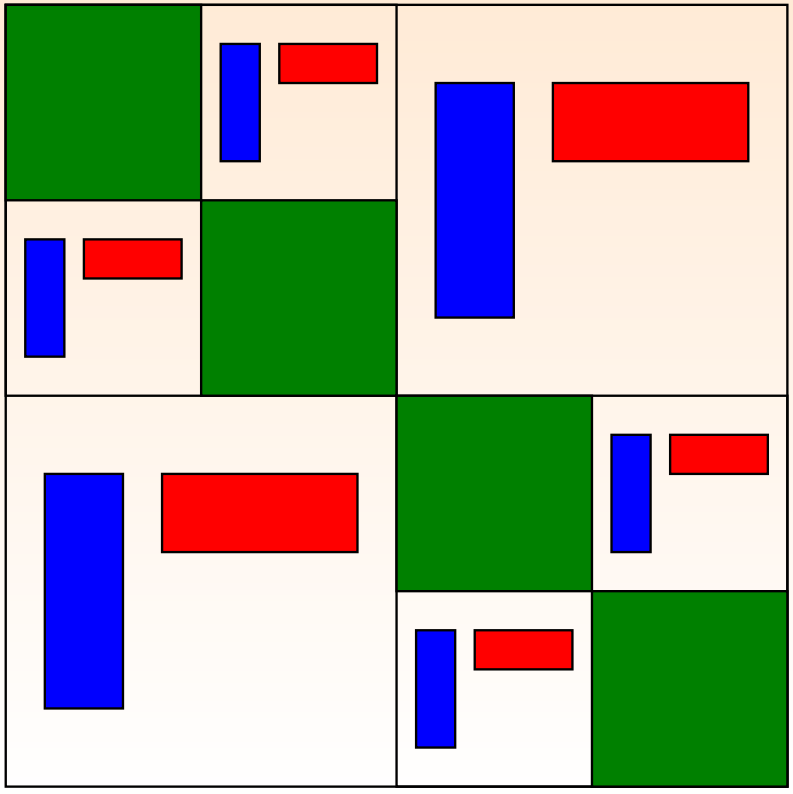
- Often implied by appplications
- Like: panel clustering in BEM (Hackbusch, Nowak, 1989), the multipole method (Greengard, Rokhlin, 1997), mosaic-skeleton approximations (Tyrtyshnikov, 1996) etc.: an example of a hierarchical (data-sparse) matrix:



Motivation

Accelerating by low-rank compression

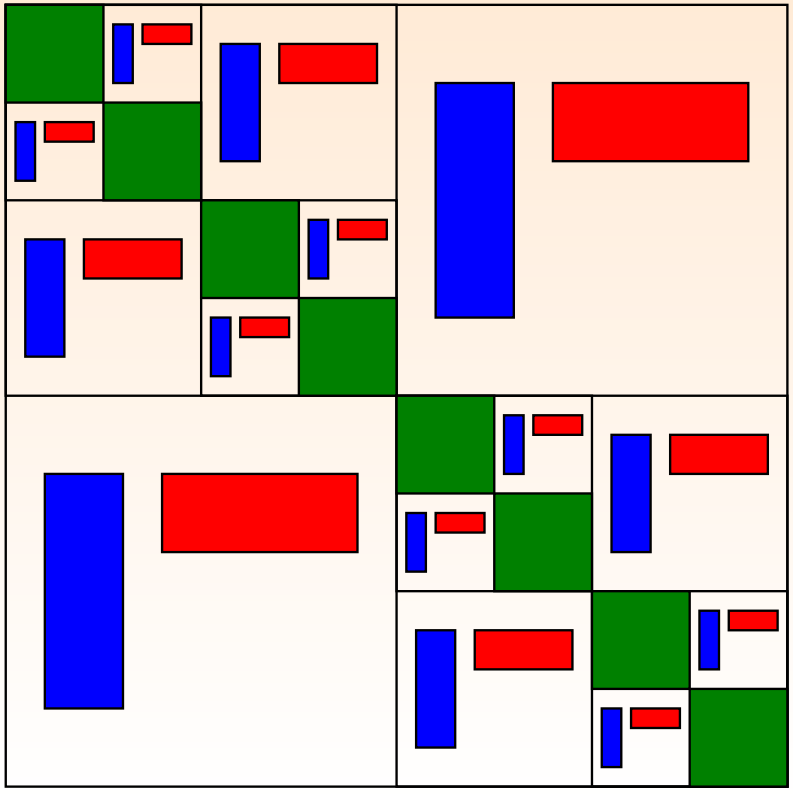
- Often implied by appplications
- Like: panel clustering in BEM (Hackbusch, Nowak, 1989), the multipole method (Greengard, Rokhlin, 1997), mosaic-skeleton approximations (Tyrtyshnikov, 1996) etc.: an example of a hierarchical (data-sparse) matrix:



Motivation

Accelerating by low-rank compression

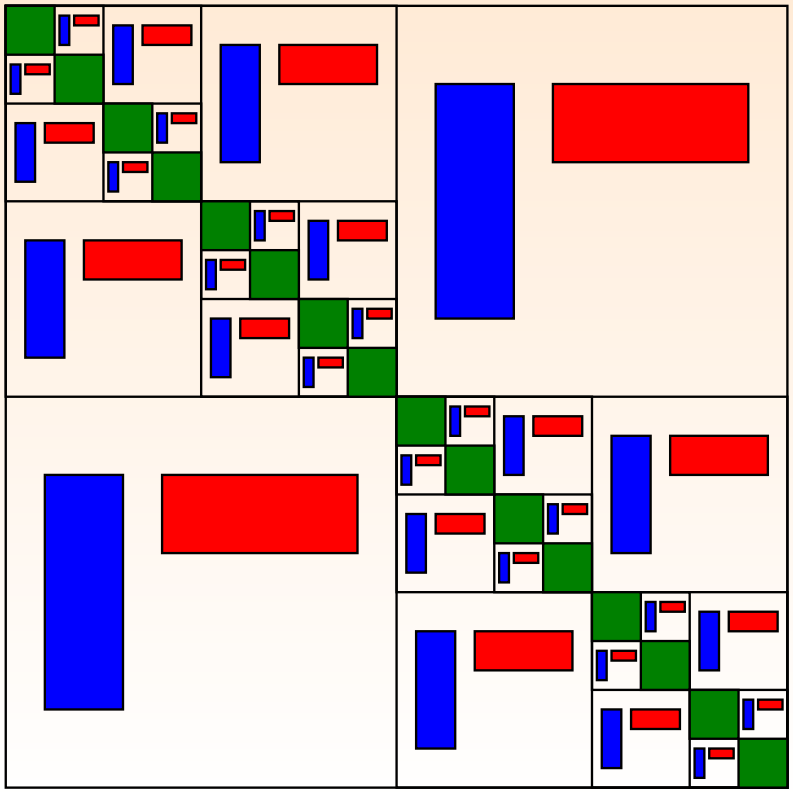
- Often implied by appplications
- Like: panel clustering in BEM (Hackbusch, Nowak, 1989), the multipole method (Greengard, Rokhlin, 1997), mosaic-skeleton approximations (Tyrtyshnikov, 1996) etc.: an example of a hierarchical (data-sparse) matrix:



Motivation

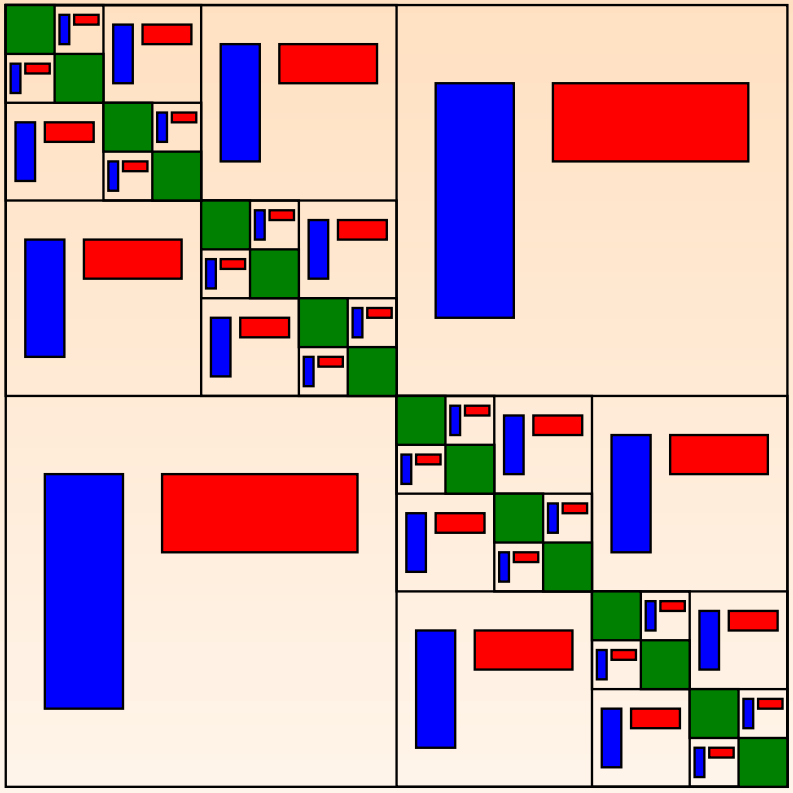
Accelerating by low-rank compression

- Often implied by appplications
- Like: panel clustering in BEM (Hackbusch, Nowak, 1989), the multipole method (Greengard, Rokhlin, 1997), mosaic-skeleton approximations (Tyrtyshnikov, 1996) etc.: an example of a hierarchical (data-sparse) matrix:



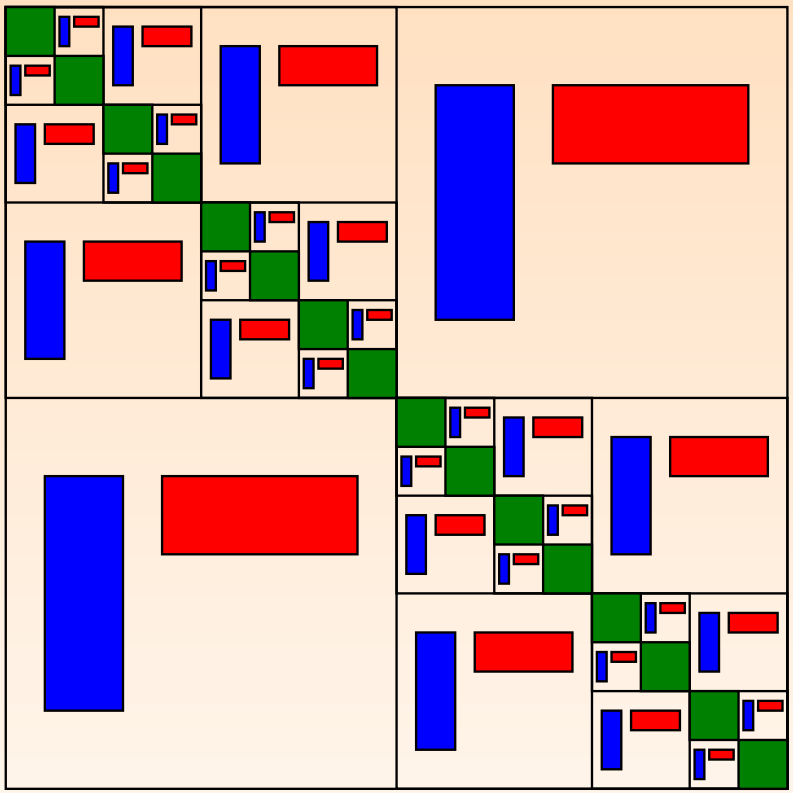
Motivation

Accelerating by low-rank compression: sparsity still needed



Motivation

Accelerating by low-rank compression: sparsity still needed



- More complex applications: algebraic variations needed.
- But then: generalized schemes need exploiting the classical (blockwise) sparsity outside a specific hierarchical scheme.

Motivation

Concept 2: Accelerating by low precision computation

- Traditionally: single precision (fp32) and double precision (fp64)
- Throughout 1990's, fp32 was not much faster than fp64.
- Real breakthrough: (SSE units, Intel, 1999): fp32 significantly accelerated
- Emergence of half precision (fp16) floating-point arithmetic: 2008 revision of the IEEE standard.

Motivation

Concept 2: Accelerating by low precision computation

- fp16 started as storage format, but soon in GPU accelerators. See discussions in Higham, 2017; Higham, Mary, 2022.
- BUT: fp16: limited range (largest positive number is 6.55×10^4); also bfloat16 (Google, tensor processing units, larger range)

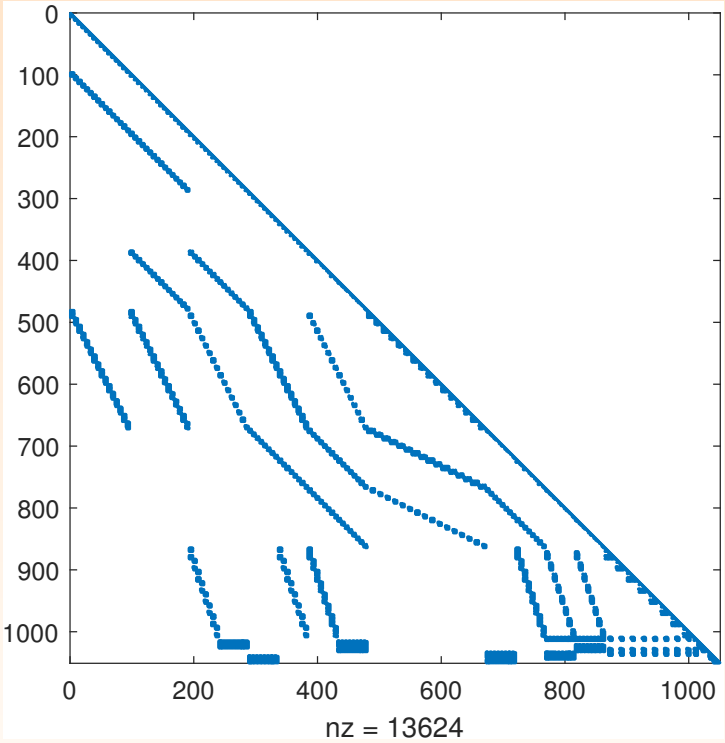
Table: Parameters for bfloat16, fp16, fp32, and fp64 arithmetic: the number of bits, u, smallest positive (subnormal) number x_{min}^s , smallest normalized positive number x_{min}, and largest finite number x_{max}.

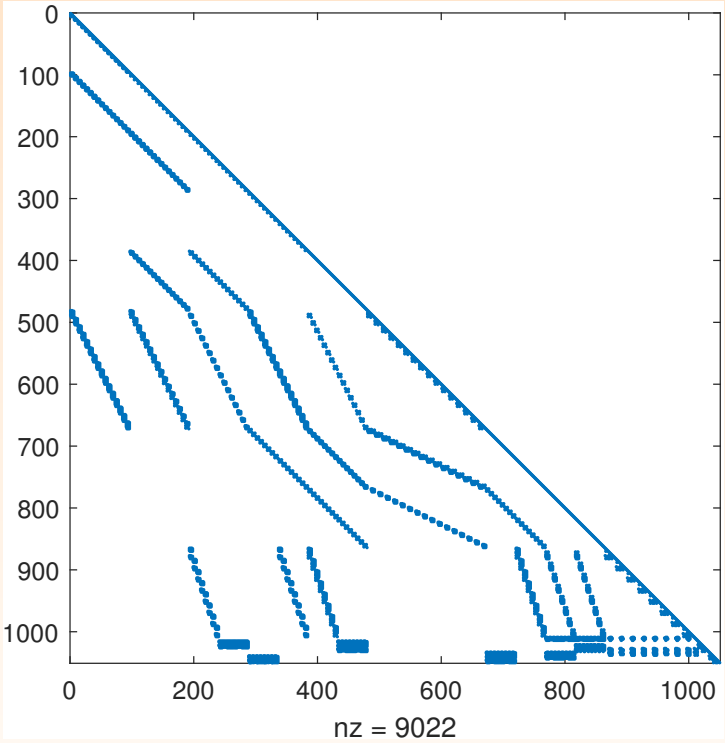
	Signif.	Exp.	u	x_{min}^s	x_{min}	x_{max}
fp16	11	5	4.88×10^{-4}	5.96×10^{-8}	6.10×10^{-5}	6.55×10^{4}
fp32	24	8	5.96×10^{-8}	1.40×10^{-45}	1.18×10^{-38}	3.40×10^{38}
fp64	53	11	1.11×10^{-16}	4.94×10^{-324}	2.22×10^{-308}	1.80×10^{308}
bfloat16	8	8	3.91×10^{-3}	not used	1.18×10^{-38}	3.39×10^{38}

Motivation

Concept 2: Accelerating by using low precision: Boeing/msc01050

- Left: A in standard double precision (fp64)
- Right: A in the half precision (fp16)



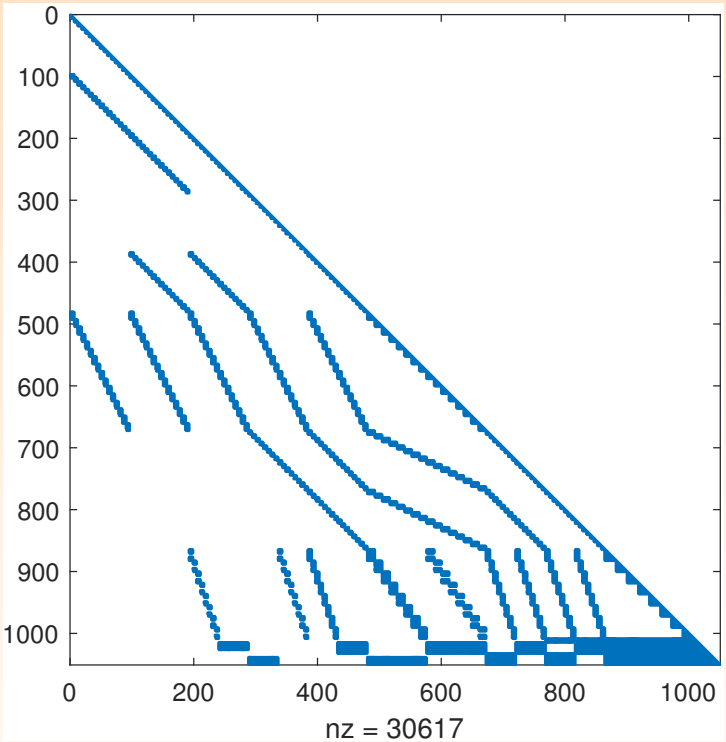


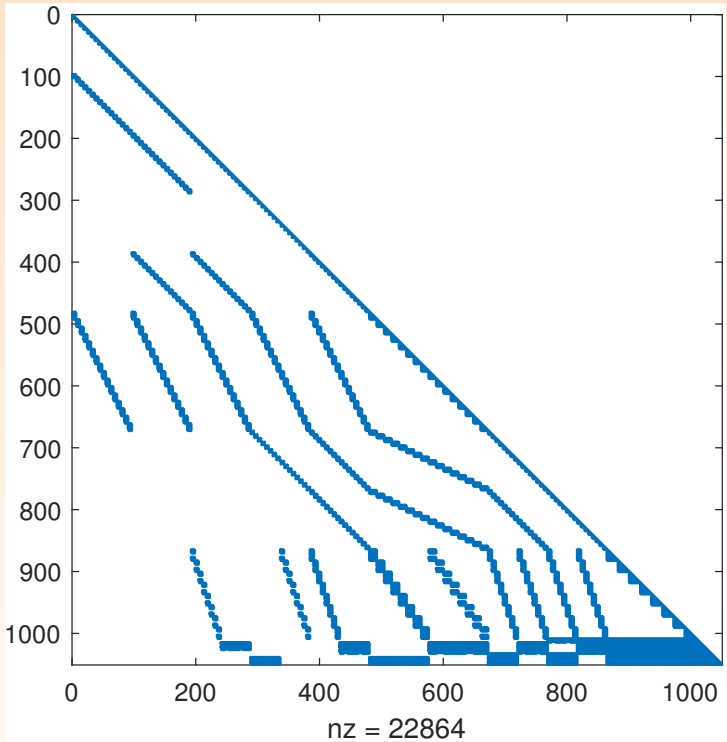
- In fp16, we get only an approximation

Motivation

Concept 2: Accelerating by using low precision: Boeing/msc01050

- Left: Cholesky factor L in standard double precision (fp64)
- Right: Cholesky factor L in the half precision (fp16)





- fp16: similar fill (ratio 2.247 versus 2.534 for fp16)
- low precision, but sparsity factorization problems are still here: they should be considered also here.

Basic Terminology: sparsity

Rough comparison of extreme cases of dense and sparse A

Dense matrix

dim	space	dec time (s)
3000	4.5M	5.72
4000	8M	14.1
5000	12.5M	27.5
6000	18M	47.8

Sparse matrix

dim	space	dec time (s)
10000	40k	0.02
90000	0.36M	0.5
1M	4M	16.6
2M	8M	49.8

Basic Terminology: sparsity

Rough comparison of extreme cases of dense and sparse A

Dense matrix		
dim	space	dec time (s)
3000	4.5M	5.72
4000	8M	14.1
5000	12.5M	27.5
6000	18M	47.8

Sparse matrix		
dim	space	dec time (s)
10000	40k	0.02
90000	0.36M	0.5
1M	4M	16.6
2M	8M	49.8

- Recall the pragmatic definition.
- The decision whether to use or not to use depends also on what we know about the computation.
- In sparse direct methods we know a lot.
- Clearly, exploiting sparsity is a must. But, the question is how.

Basic Terminology: blocks

Blocks: why we like them

Basic Terminology: blocks

Blocks: why we like them

- Contemporary terminology related to computations emphasizes the most limiting factor.
- Algorithms are compute-bound, memory-bound or latency-bound.

- Most chips are designed such that dense matrix-matrix multiply, which typically performs k^3 operations on k^2 data can run at full compute throughput

$$\Downarrow$$

BLOCKS

- We may use large (I.) or small (II.) blocks

Basic Terminology: blocks

I. Large blocks

- Connected to reducibility or input (application)
- $A \in \mathbf{R}^{n\times n}$ is reducible, if it can be permuted as

$$P^T A P = \begin{pmatrix} A_{11} & A_{12} \\ & A_{22} \end{pmatrix},$$

  A_{11} and A_{22} are square matrices of dimensions at least 1.
- If A is not reducible, it is called irreducible.
- A reducible: block factorization/substitution (permutation omitted).

$$\begin{pmatrix} A_{11} & A_{12} \\ & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \rightarrow x_2 = A_{22}^{-1} b_2, x_1 = A_{11}^{-1}(b_1 - A_{12}x_2)$$

Basic Terminology: blocks

I. Large blocks

- Connected to reducibility or input (application)
- $A \in \mathbf{R}^{n \times n}$ is reducible, if it can be permuted as

$$P^T A P = \begin{pmatrix} A_{11} & A_{12} \\ & A_{22} \end{pmatrix},$$

  A_{11} and A_{22} are square matrices of dimensions at least 1.
- If A is not reducible, it is called irreducible.
- A reducible: block factorization/substitution (permutation omitted).

$$\begin{pmatrix} A_{11} & A_{12} \\ & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \rightarrow x_2 = A_{22}^{-1} b_2, x_1 = A_{11}^{-1}(b_1 - A_{12} x_2)$$

- Large blocks can be also a result of a (saddle-point) input A, like:

$$A = \begin{pmatrix} B & E \\ F & C \end{pmatrix},$$

  A lot of specialized approaches. Solving subproblems is not principially different from standard sparse approaches.

Basic Terminology: blocks

II. small blocks (only symmetric variant mentioned)

- $$A = (A_{ib,\,jb}), \quad A_{ib,\,jb} \in \mathbb{R}^{n_i \times n_j}, \quad 1 \le ib, jb \le nb,$$

  that is,

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,nb} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,nb} \\ \vdots & \vdots & \ddots & \vdots \\ A_{nb,1} & A_{nb,2} & \cdots & A_{nb,nb} \end{pmatrix}.$$

- Assuming nonsingular square blocks $A_{jb,\,jb}$ on the diagonal.

Basic Terminology: blocks

II. small blocks (only symmetric variant mentioned)

- $$A = (A_{ib,\,jb}), \quad A_{ib,\,jb} \in \mathbb{R}^{n_i \times n_j}, \quad 1 \le ib, jb \le nb,$$

  that is,

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,nb} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,nb} \\ \vdots & \vdots & \ddots & \vdots \\ A_{nb,1} & A_{nb,2} & \cdots & A_{nb,nb} \end{pmatrix}.$$

- Assuming nonsingular square blocks $A_{jb,\,jb}$ on the diagonal.

Implications

- Large blocks: as we have seen above: only an additional hierarchical level.
- Small blocks: Pointwise factorizations can be formulated blockwise:
  entries $\rightarrow$ submatrices. Here not reminded, but expected.

Outline

Introduction to factorizations

- Traditional way: Gaussian elimination: systematic columnwise annihilation of entries in the lower triangular part of A.
- Formally a sequential multiplications by column elimination matrices (A factorizable) getting the elimination sequence:

$$A^{(1)} \to A^{(2)} = C_1 A^{(1)} \to A^{(3)} = C_2 C_1 A^{(1)} \to \ldots \to A^{(n)} = C_{n-1} \ldots C_1 A^{(1)}.$$

- Elementwise, ($a_{11} = a_{11}^{(1)} \neq 0$), the first step $C_1 A^{(1)} = A^{(2)}$ is

$$\begin{pmatrix} 1 & & & \\ -a_{21}^{(1)}/a_{11}^{(1)} & 1 & & \\ -a_{31}^{(1)}/a_{11}^{(1)} & & 1 & \\ \vdots & & & 1 \\ -a_{n1}^{(1)}/a_{11}^{(1)} & & & 1 \end{pmatrix} \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \ldots & a_{2n}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & \ldots & a_{3n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \ldots & a_{nn}^{(1)} \end{pmatrix} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ 0 & a_{32}^{(2)} & \ldots & a_{3n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \ldots & a_{nn}^{(2)} \end{pmatrix},$$

Introduction to factorizations

- The k-th partially eliminated matrix is $A^{(k)}$.
- The product of inverted column elimination matrices

$$\begin{pmatrix} 1 & & & & \\ a_{21}^{(1)}/a_{11}^{(1)} & 1 & & & \\ a_{31}^{(1)}/a_{11}^{(1)} & a_{32}^{(2)}/a_{22}^{(2)} & 1 & & \\ \vdots & \vdots & \vdots & 1 & \\ a_{n1}^{(1)}/a_{11}^{(1)} & a_{n2}^{(2)}/a_{22}^{(2)} & \vdots & \vdots & 1 \end{pmatrix}$$

- That is, we have the LU factorization

$$A = A^{(1)} = C_1^{-1} C_2^{-1} \dots C_{n-1}^{-1} A^{(n)} = LU.$$

- There are more ways differing in relative order of elimination steps that are the same even in finite precision arithmetic!

LU written in matrix/vector form: submatrix LU

- The first step ($k = 1$):

$$C_1 A = \begin{pmatrix} 1 & \\ -v/a_{11} & I \end{pmatrix} \begin{pmatrix} a_{11} & u^T \\ v & A_{2:n,2:n} \end{pmatrix} = \begin{pmatrix} a_{11} & u^T \\ & A_{2:n,2:n} - vu^T/a_{11} \end{pmatrix},$$

$$v = (a_{21}, \ldots, a_{n1})^T, \quad (l_{21}, \ldots, l_{n1})^T = v/a_{11}, \quad u^T = (a_{12}, \ldots, a_{1n}).$$

- The $(n-1) \times (n-1)$ active submatrix

$$A^{(2)} = S = A_{2:n,2:n} - vu^T/a_{11}$$

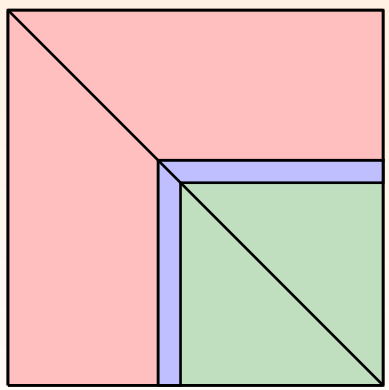
  is the Schur complement of A with respect to a_{11}.
- A is factorizable $\Rightarrow$ S is factorizable, and the process can be repeated.

Submatrix LU

- The elimination: sequence of rank-one updates applied to the Schur complements.
- After $k-1$ steps ($1 < k \leq n$):

$$S^{(k)} = \begin{pmatrix} a_{kk} & \dots & a_{kn} \\ \vdots & \ddots & \vdots \\ a_{nk} & \dots & a_{nn} \end{pmatrix} - \sum_{j=1}^{k-1} \begin{pmatrix} l_{kj} \\ \vdots \\ l_{nj} \end{pmatrix} \begin{pmatrix} u_{jk} & \dots & u_{jn} \end{pmatrix} = \begin{pmatrix} a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ \vdots & \ddots & \vdots \\ a_{nk}^{(k)} & \dots & a_{nn}^{(k)} \end{pmatrix} = A_{k:n,k:n}^{(k)}.$$

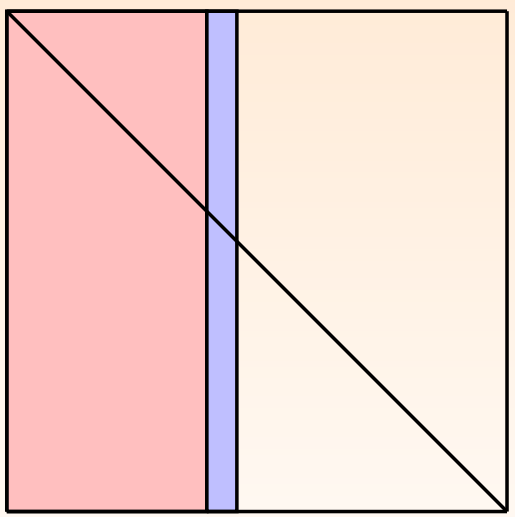
- Schematically:



Another relative order of operations: Column LU

- Consider first j columns of A: they must satisfy

$$\begin{pmatrix} A_{1:j-1,1:j-1} & A_{1:j-1,j} \\ A_{j:n,1:j-1} & A_{j:n,j} \end{pmatrix} \rightarrow \begin{pmatrix} L_{1:j-1,1:j-1} & \\ L_{j:n,1:j-1} & L_{j:n,j} \end{pmatrix} \begin{pmatrix} U_{1:j-1,1:j-1} & U_{1:j-1,j} \\ & u_{jj} \end{pmatrix}$$



Factorizations

Another relative order of operations: Column LU

- Consider first j columns of A: we must have

$$\begin{pmatrix} A_{1:j-1,1:j-1} & A_{1:j-1,j} \\ A_{j:n,1:j-1} & A_{j:n,j} \end{pmatrix} \to \begin{pmatrix} L_{1:j-1,1:j-1} & \\ L_{j:n,1:j-1} & L_{j:n,j} \end{pmatrix} \begin{pmatrix} U_{1:j-1,1:j-1} & U_{1:j-1,j} \\ & u_{jj} \end{pmatrix}$$

- This implies conditions for the two phases of computation (column of U and L):

$$U_{1:j-1,j} = L^{-1}_{1:j-1,1:j-1} A_{1:j-1,j}, \quad u_{jj} = a_{jj} - L_{j,1:j-1} U_{1:j-1,j},$$

$$l_{jj} = 1,\ L_{j+1:n,j} = (A_{j+1:n,j} - L_{j+1:n,1:j-1} U_{1:j-1,j})/u_{jj}.$$

- The factors can be computed column by column:

$$1 \to \ldots j \to \ldots n$$

- Easy embedding of the row permutation: $A \to PA$
- Scheme by rows: computing columns of A^T

Factorizations

Schemes described as a generic scheme of three nested loops

Algorithm (Generic LU factorization)

1: **for** ———— **do**
2: **for** ———— **do**
3: **for** ———— **do**
4: $l_{ik} = a_{ik}^{(k)} a_{kk}^{-1}$
5: $a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}$
6: **end for**
7: **end for**
8: **end for**

- The crucial pointwise operation:

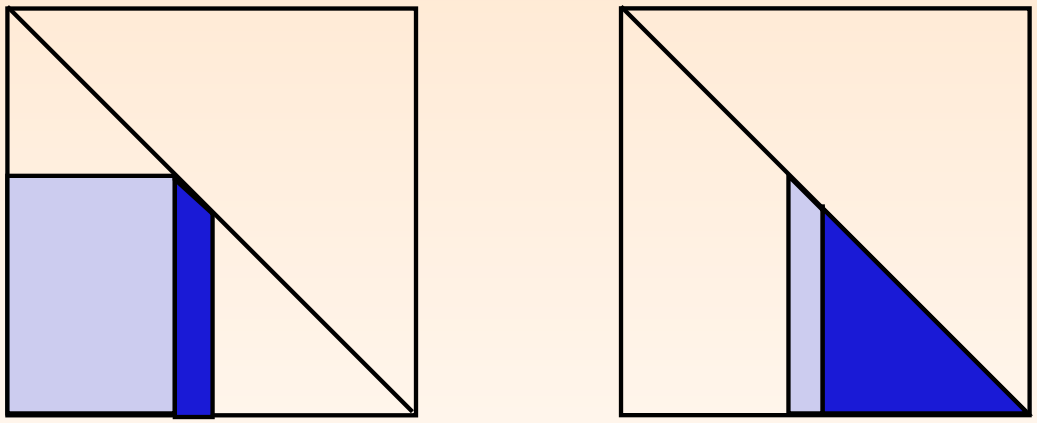
$$a_{ij} = a_{ij} - a_{ik} a_{kk}^{-1} a_{kj} \equiv a_{ij} = a_{ij} - l_{ik} a_{kj}$$

- Schemes differ by treating sparsity, vectorization etc.

Factorizations

Cholesky factorization: also three basic ways

- Left-looking schemes (second phase of the column LU)
- Right-looking schemes (submatrix scheme that computes only quantities in L)



- But there is also the row scheme based on the first phase (solve) of the column LU

Column (left-looking) Cholesky

Algorithm

Column Cholesky factorization: $A \rightarrow$ *square-root factor* $L = (l_{ij})$

1. *for* $j = 1 : n$ *do*
2. *Compute an auxiliary vector* $t_{j:n}$

$$\begin{pmatrix} t_j \\ \vdots \\ t_n \end{pmatrix} = \begin{pmatrix} a_{jj} \\ \vdots \\ a_{nj} \end{pmatrix} - \sum_{\{k|l_{jk}\neq 0\}} l_{jk} \begin{pmatrix} l_{jk} \\ \vdots \\ l_{nk} \end{pmatrix} \tag{1}$$

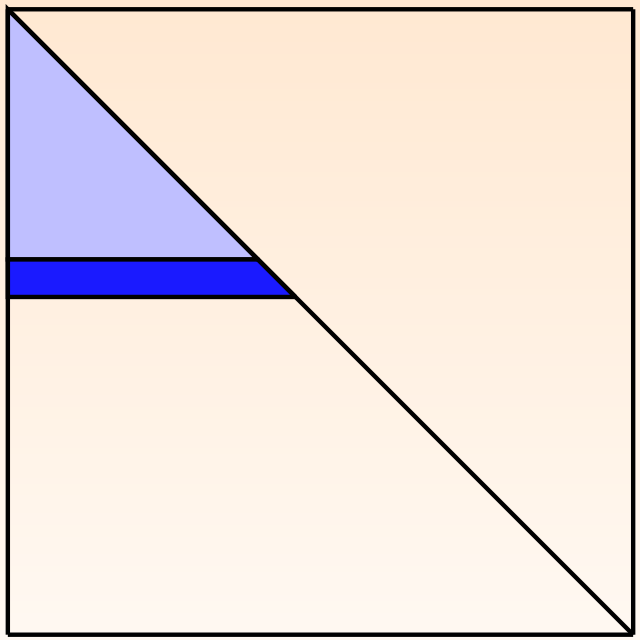
3. *Get a column of* L *by scaling* $t_{j:n}$

$$\begin{pmatrix} l_{jj} \\ \vdots \\ l_{nj} \end{pmatrix} = \frac{1}{\sqrt{t_j}} \begin{pmatrix} t_j \\ \vdots \\ t_n \end{pmatrix} \tag{2}$$

4. *end* j

Cholesky factorization: row scheme

- But there is also the row scheme.
- The row scheme is based on the first phase (solve) of the column LU



- Easy to implement column permutation $A \rightarrow AP$.

Factorizations and sparsity

- Factorizations of sparse matrices may create new nonzero entries outside $\mathcal{S}\{A\}$ called fill/fill-in/filled entries

$$\begin{pmatrix} * & * & & * & * \\ & * & & & \\ * & & * & & \\ * & * & & * & \\ * & & * & & * \end{pmatrix} \rightarrow \begin{pmatrix} * & * & & * & * \\ & * & & & \\ * & f & * & f & f \\ * & * & & * & f \\ * & f & * & f & * \end{pmatrix}$$

- Fill-in means more operations, more memory

Factorizations and sparsity

- Can we expect that some nonzeros become zeros due to cancellation?

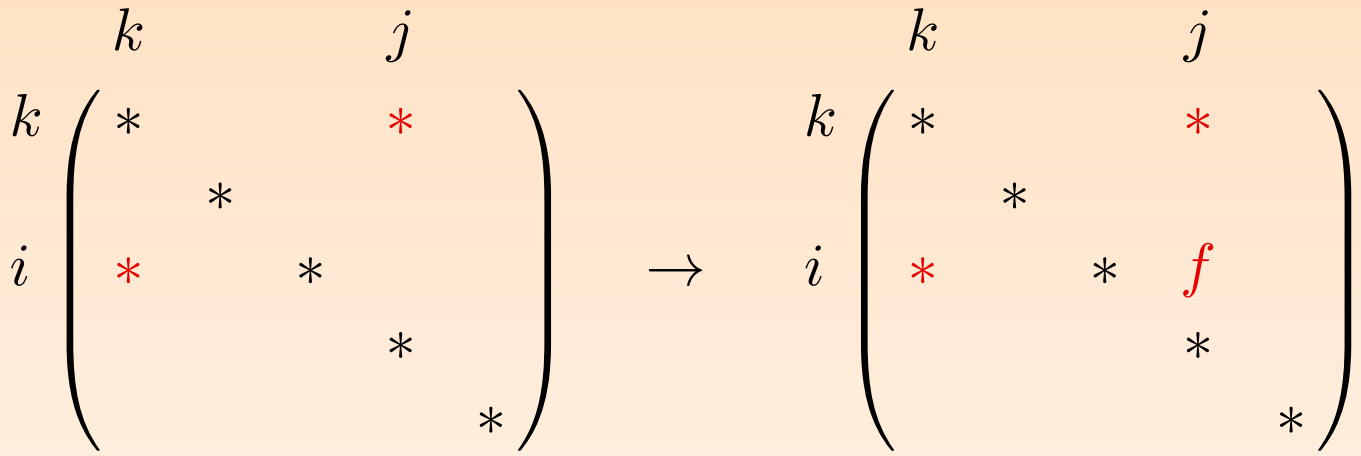
Factorizations and sparsity

- Can we expect that some nonzeros become zeros due to cancellation?
- Very rarely.
- We assume non-cancellation: the result of adding, subtracting or multiplying two nonzeros is nonzero again.
- This implies:

$$\mathcal{S}\{A\} \subseteq \mathcal{S}\{L+U\}.$$

- Non-cancellation implies a possibility to deal with the structure only using graphs to determine the fill-in (if factorizability guaranteed ☺)
- Let us go to see the fill-in results

Simple fill-in results: one step of factorization



- Summarized as the fill-in lemma: one step of the fill-in

Lemma

Let $i, j, k \in \{1, \ldots, n\}$, *step* $k < \min\{i, j\} \leq n$. *Then*

$$a_{ij}^{(k)} \neq 0 \iff a_{ij}^{(k-1)} \neq 0 \vee (a_{ik}^{(k-1)} \neq 0 \wedge a_{kj}^{(k-1)} \neq 0)$$

Fill-in during the factorization: more steps

- But we have the sequence (of Schur complements)

$$S^{(1)} \to S^{(2)} \to S^{(3)} \to \ldots \to S^{(n)} = a_{nn}^{(n-1)}.$$

- With sparsity structures $\mathcal{S}(S^{(i)})$ representing the elimination graphs

$$\mathcal{G}^1 \equiv \mathcal{G}(A), \mathcal{G}^2, \ldots, \mathcal{G}^n, \mathcal{G}^k = (\mathcal{V}^k, \mathcal{E}^k)$$

Fill-in during the factorization: more steps

- But we have the sequence (of Schur complements)

$$S^{(1)} \to S^{(2)} \to S^{(3)} \to \ldots \to S^{(n)} = a_{nn}^{(n-1)}.$$

- With sparsity structures $\mathcal{S}(S^{(i)})$ representing the elimination graphs

$$\mathcal{G}^1 \equiv \mathcal{G}(A), \mathcal{G}^2, \ldots, \mathcal{G}^n, \mathcal{G}^k = (\mathcal{V}^k, \mathcal{E}^k)$$

- The fill-in in the sequence is described by the Parter's rule:

  To obtain the elimination graph $\mathcal{G}^{k+1}$ from $\mathcal{G}^k$, delete vertex k and add all edges $(i \xrightarrow{\mathcal{G}^{k+1}} j)$ such that $(i \xrightarrow{\mathcal{G}^k} k)$ and $(k \xrightarrow{\mathcal{G}^k} j)$.

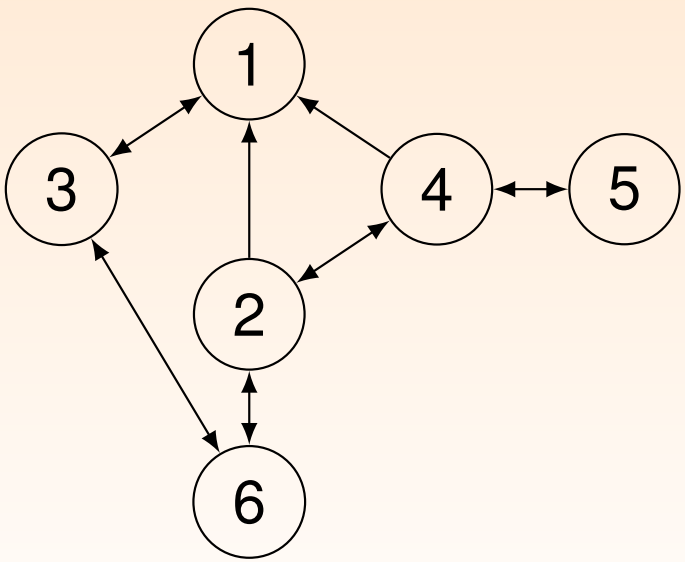
$$\mathcal{V}^{k+1} = \mathcal{V}^k \setminus \{k\},\ \mathcal{E}^{k+1} = \mathcal{E}^k \cup \{(i,j) \,|\, i,j \in adj_{\mathcal{G}^k}\{k\}\} \setminus \{(i,k) \,|\, i \in adj_{\mathcal{G}^k}\{k\}\}.$$

- The reason that graphs can be used: the non-cancellation assumption: once created fill-in remains ☺

Factorizations

- A (nonsymmetric) example

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} * & & * & & & \\ * & * & & * & & * \\ * & & * & & & * \\ * & * & & * & * & \\ & & & * & * & \\ & * & * & & & * \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} * & & * & & & \\ * & * & f & * & & * \\ * & & * & & & * \\ * & * & f & * & * & \\ & & & * & * & \\ & * & * & & & * \end{pmatrix} \end{array}$$



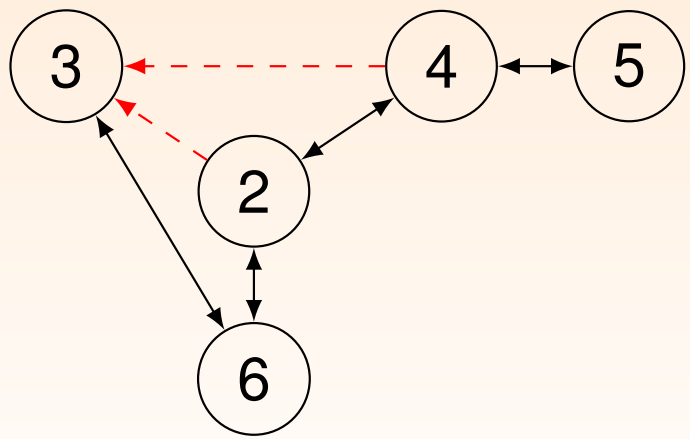


Figure: *The original digraph $\mathcal{G} = \mathcal{G}^1$ (left) and the directed elimination graph $\mathcal{G}^2$ (right). The red dashed lines denote fill edges.*

Factorizations

- $\mathcal{S}(A)$ symmetric: the adjacency set of vertex k forms a clique.

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{pmatrix}
* & * & * & * & & \\
* & * & & * & & * \\
* & & * & & & * \\
* & * & & * & * & \\
 & & & * & * & \\
 & * & * & & & *
\end{pmatrix}
\end{array}
\qquad
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{pmatrix}
* & * & * & * & & \\
* & * & f & * & & * \\
* & f & * & f & & * \\
* & * & f & * & * & \\
 & & & * & * & \\
 & * & * & & & *
\end{pmatrix}
\end{array}
$$

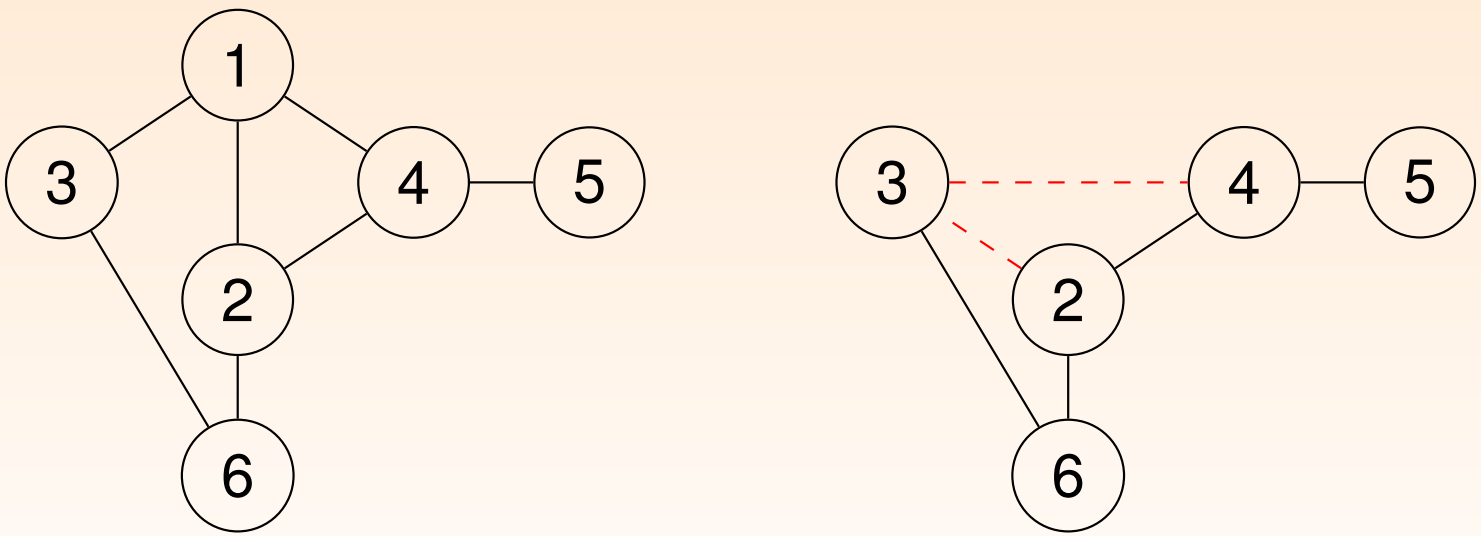


Figure: *The original undirected graph $\mathcal{G} = \mathcal{G}^1$ (left) and the obtained graph $\mathcal{G}^2$ (right). The red dashed lines denote fill edges. The vertices $\{2, 3, 4\}$ become a clique.*

Factorizations

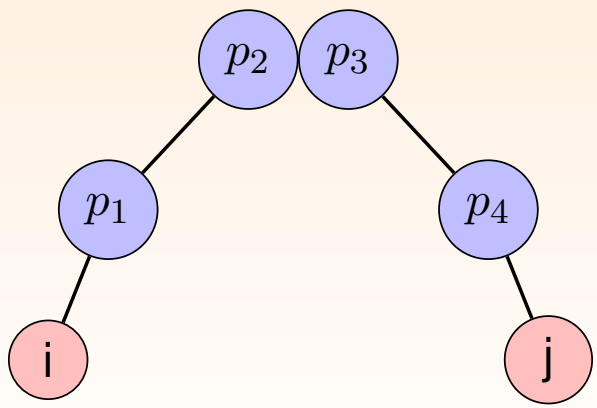
From the Parter's rule for factors to fill paths in $\mathcal{G}(A)$

- But the Parter's rule is only a local rule. The following theorem fully characterizes the fill-in in the factors.

Theorem

Let $A = LU$. Then $\mathcal{S}(L+U)_{ij} \neq 0$ if and only if there is a fill-path $i \xrightarrow[min]{\mathcal{G}(A)} j$. The fill-paths may not be unique.

- Demonstrate this: starting with a path $(i, p_1, p_2, p_3, p_4, j)$ in $\mathcal{G}(A)$



$$p_2 < p_3 < p_1 < p_4 < \min(i, j)$$

Factorizations

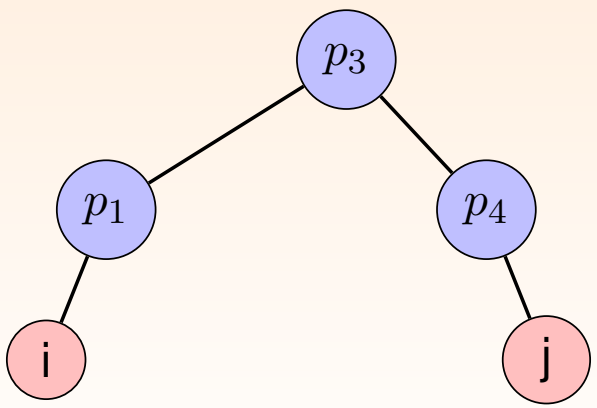
From the Parter's rule for factors to fill paths in $\mathcal{G}(A)$

- But the Parter's rule is only a local rule. The following theorem fully characterizes the fill-in in the factors.

Theorem

Let $A = LU$. Then $\mathcal{S}(L+U)_{ij} \neq 0$ if and only if there is a fill-path $i \xrightarrow[min]{\mathcal{G}(A)} j$. The fill-paths may not be unique.

- Demonstrate this: starting with a path $(i, p_1, p_2, p_3, p_4, j)$ in $\mathcal{G}(A)$



$$p_2 < p_3 < p_1 < p_4 < \min(i, j)$$

Factorizations

From the Parter's rule for factors to fill paths in $\mathcal{G}(A)$

- But the Parter's rule is only a local rule. The following theorem fully characterizes the fill-in in the factors.

Theorem

Let $A = LU$. Then $\mathcal{S}(L+U)_{ij} \neq 0$ if and only if there is a fill-path $i \xrightarrow[min]{\mathcal{G}(A)} j$. The fill-paths may not be unique.

- Demonstrate this: starting with a path $(i, p_1, p_2, p_3, p_4, j)$ in $\mathcal{G}(A)$

$$p_2 < p_3 < p_1 < p_4 < \min(i, j)$$

Factorizations

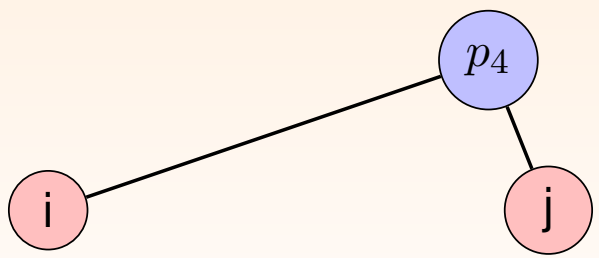
From the Parter's rule for factors to fill paths in $\mathcal{G}(A)$

- But the Parter's rule is only a local rule. The following theorem fully characterizes the fill-in in the factors.

Theorem

Let $A = LU$. Then $\mathcal{S}(L+U)_{ij} \neq 0$ if and only if there is a fill-path $i \xrightarrow[min]{\mathcal{G}(A)} j$. The fill-paths may not be unique.

- Demonstrate this: starting with a path $(i, p_1, p_2, p_3, p_4, j)$ in $\mathcal{G}(A)$



$$p_2 < p_3 < p_1 < p_4 < \min(i, j)$$

Factorizations

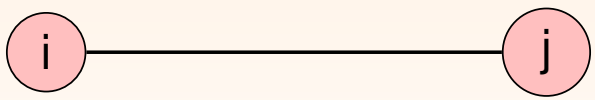
From the Parter's rule for factors to fill paths in $\mathcal{G}(A)$

- But the Parter's rule is only a local rule. The following theorem fully characterizes the fill-in in the factors.

Theorem

Let $A = LU$. Then $\mathcal{S}(L+U)_{ij} \neq 0$ if and only if there is a fill-path $i \xrightarrow[min]{\mathcal{G}(A)} j$. The fill-paths may not be unique.

- Demonstrate this: starting with a path $(i, p_1, p_2, p_3, p_4, j)$ in $\mathcal{G}(A)$



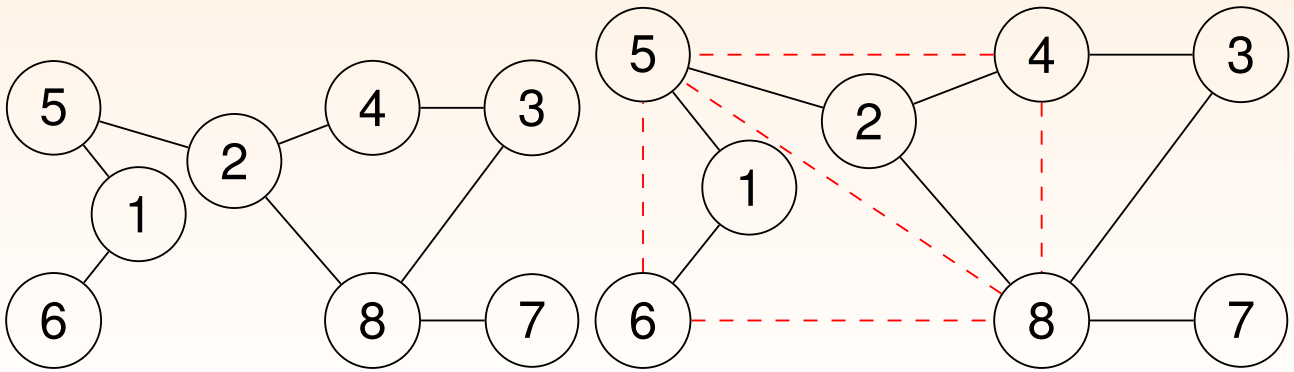
$$p_2 < p_3 < p_1 < p_4 < \min(i, j)$$

Factorizations

From the Parter's rule for factors to fill paths in $\mathcal{G}(A)$

- Symmetric $\mathcal{S}\{A\}$: a filled entry in position $(8,6)$ of L because of the fill-path $8 \overset{\mathcal{G}(A)}{\underset{min}{\Longleftrightarrow}} 6$: $8 \longleftrightarrow 2 \longleftrightarrow 5 \longleftrightarrow 1 \longleftrightarrow 6$.

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
* & & & & * & * & & \\
 & * & & * & * & & & * \\
 & & * & * & & & & * \\
 & * & * & * & & & & \\
* & * & & & * & & & \\
* & & & & & * & & \\
 & & & & & & * & * \\
 & * & * & & & & * & *
\end{pmatrix}
\end{array}
\qquad
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
* & & & & * & * & & \\
 & * & & * & * & & & * \\
 & & * & * & & & & * \\
 & * & * & * & f & & & f \\
* & * & & f & * & f & & f \\
* & & & & f & * & & f \\
 & & & & & & * & * \\
 & * & * & f & f & f & * & *
\end{pmatrix}
\end{array}
$$



Factorizations

So far, only implicit results on the fill-in

Factorizations

So far, only implicit results on the fill-in

- Too complicated to be exploited algorithmically

Factorizations

So far, only implicit results on the fill-in

- Too complicated to be exploited algorithmically
- Something that is even simpler than the fill paths needed.

Factorizations

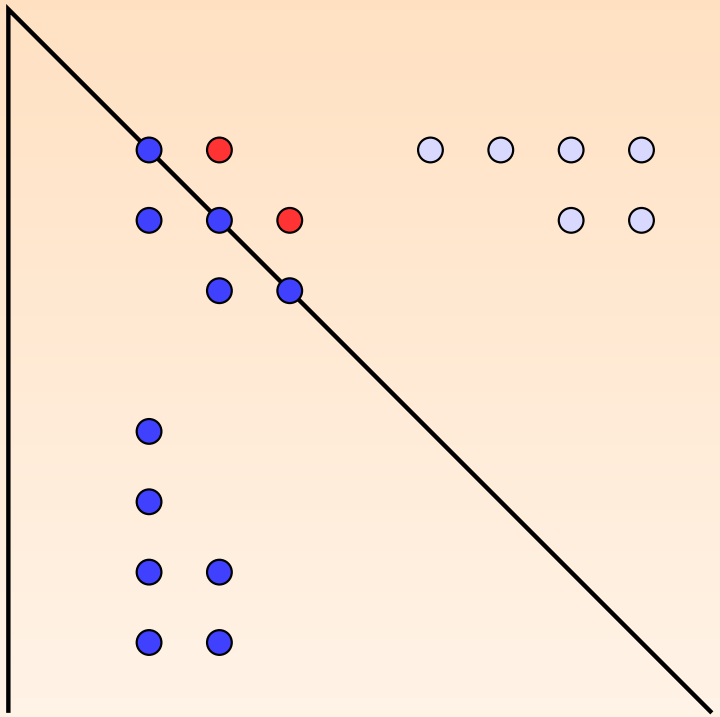
So far, only implicit results on the fill-in

- Too complicated to be exploited algorithmically
- Something that is even simpler than the fill paths needed.
- The symmetric case: SPD matrix is always factorizable $\rightarrow$ using graphs to model $A = LL^T$.
- The dependence: replication of entries among columns.

Outline

Symbolic Cholesky

Column replication: as a sequence

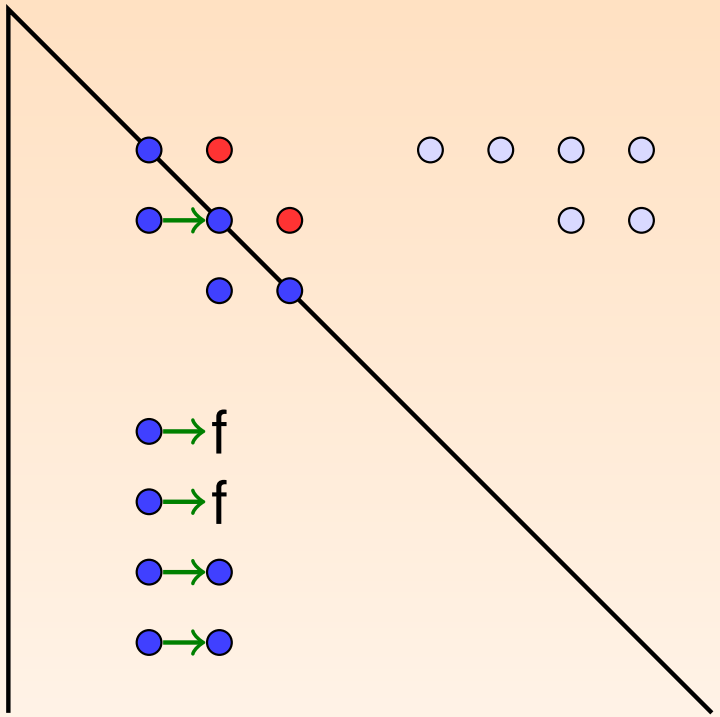


$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
* & * & & & & * & * & * \\
* & * & * & & & & * & * \\
 & * & * & & & & & \\
 & & & * & & & & \\
* & & & & * & & & \\
* & & & & & * & & \\
* & * & & & & & * & \\
* & * & & & & & & *
\end{pmatrix}
\end{array}
$$

Nonzero entries of the lower triangular part

Symbolic Cholesky

Column replication: as a sequence

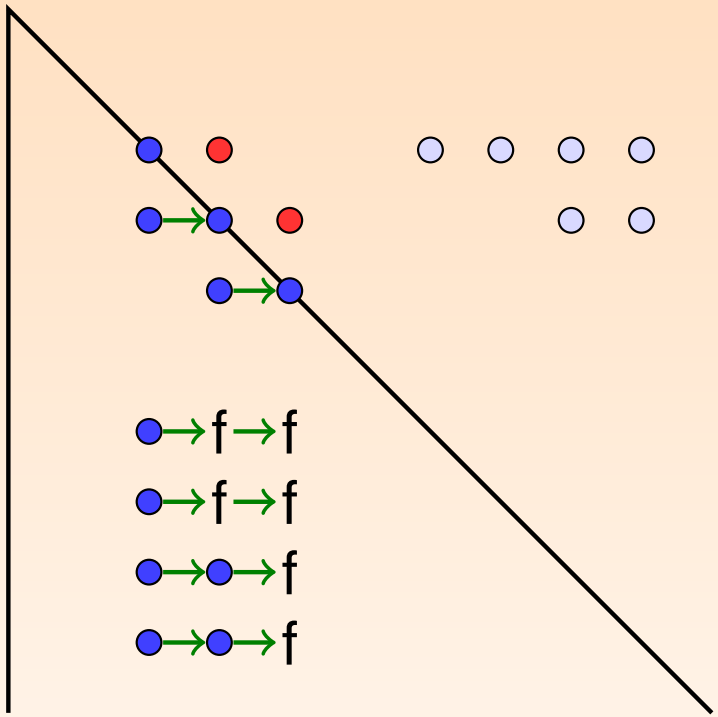


$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
* & * & & & & * & * & * \\
* & * & * & & f & f & * & * \\
 & * & * & & & & & \\
 & & & * & & & & \\
* & f & & & * & & & \\
* & f & & & & * & & \\
* & * & & & & & * & \\
* & * & & & & & & *
\end{pmatrix}
\end{array}
$$

Nonzero entries of the lower triangular part

Symbolic Cholesky

Column replication: as a sequence



$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
* & * & & & & * & * & * \\
* & * & * & & f & f & * & * \\
 & * & * & & f & f & f & f \\
 & & & * & & & & \\
* & f & f & & * & & & \\
* & f & f & & & * & & \\
* & * & f & & & & * & \\
* & * & f & & & & & *
\end{pmatrix}
\end{array}
$$

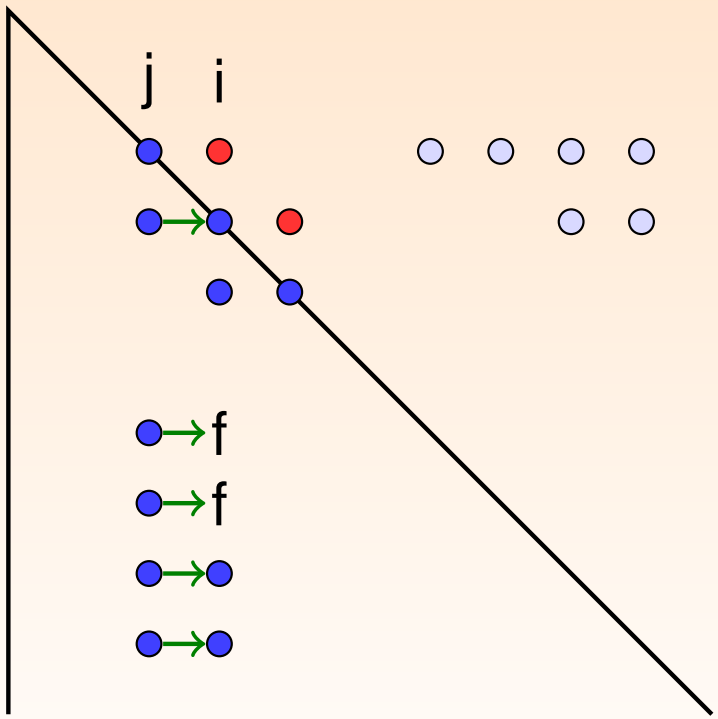
Nonzero entries of the lower triangular part

Column replication formally

- First observation:
  - For any $i > j \geq 1$ such that $l_{ij} \neq 0$

$$\mathcal{S}\{L_{i:n,j}\} \subseteq \mathcal{S}\{L_{i:n,i}\}. \tag{3}$$

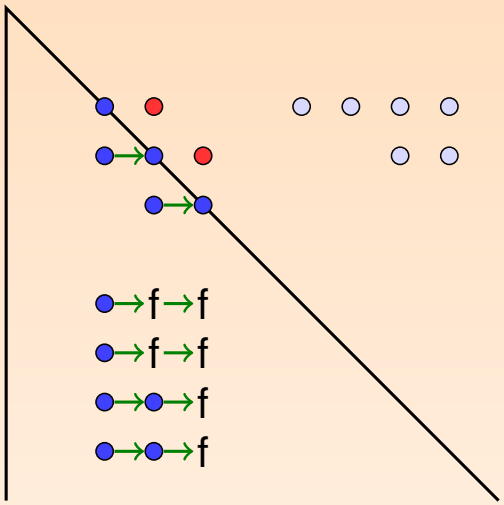
This is called the column replication principle.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} * & * & & & & * & * & * \\ * & * & * & & f & f & * & * \\ & * & * & & & & & \\ & & & * & & & & \\ * & f & & & * & & & \\ * & f & & & & * & & \\ * & * & & & & & * & \\ * & * & & & & & & * \end{pmatrix} \end{array}$$

Symbolic Cholesky

Column replication: as a sequence

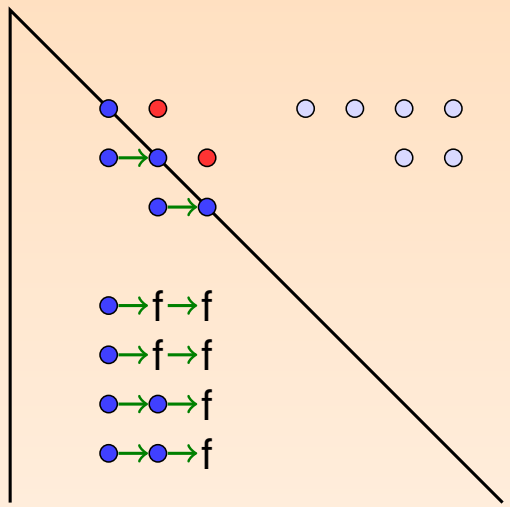


$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
* & * & & & & * & * & * \\
* & * & * & & f & f & * & * \\
 & * & * & & f & f & f & f \\
 & & & * & & & & \\
* & f & f & & * & & & \\
* & f & f & & & * & & \\
* & * & f & & & & * & \\
* & * & f & & & & & *
\end{pmatrix}
\end{array}
$$

- First row entries of L^T are sufficient to guarantee the replication.

Symbolic Cholesky

Column replication: as a sequence

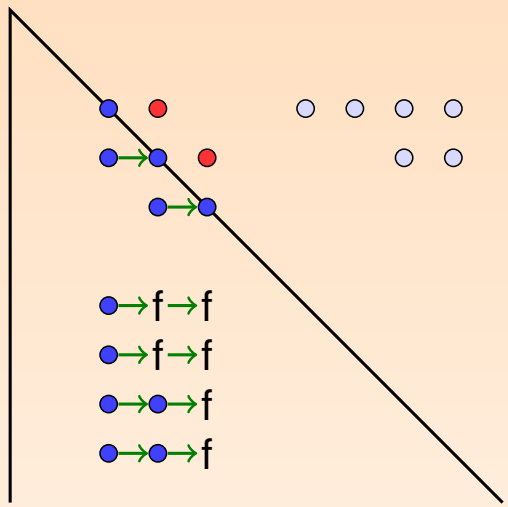


$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
* & * & & & & * & * & * \\
* & * & * & & f & f & * & * \\
 & * & * & & f & f & f & f \\
 & & & * & & & & \\
* & f & f & & * & & & \\
* & f & f & & & * & & \\
* & * & f & & & & * & \\
* & * & f & & & & & *
\end{pmatrix}
\end{array}
$$

- First row entries of L^T are sufficient to guarantee the replication.
- They represent a directed acyclic graph (DAG) $\mathcal{T}(A) \subseteq \mathcal{G}(L^T)$.
- $\mathcal{T}(A)$: a special case of the transitive reduction of $\mathcal{G}(L^T)$ (simplest DAG that preserves paths in $\mathcal{G}(L^T)$.

Symbolic Cholesky

Column replication: as a sequence

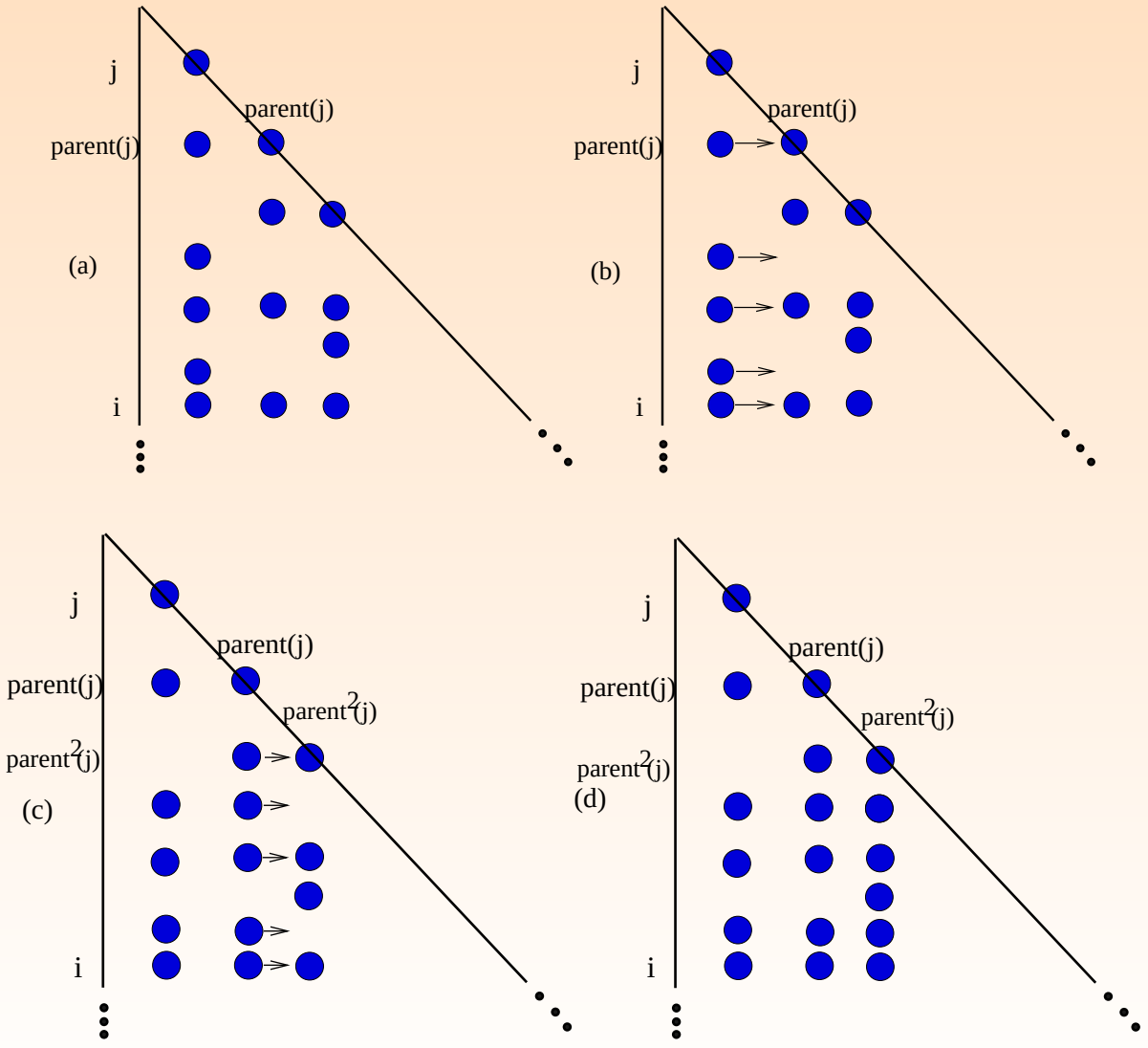


$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
* & * & & & & * & * & * \\
* & * & * & & f & f & * & * \\
 & * & * & & f & f & f & f \\
 & & & * & & & & \\
* & f & f & & * & & & \\
* & f & f & & & * & & \\
* & * & f & & & & * & \\
* & * & f & & & & & *
\end{pmatrix}
\end{array}
$$

- First row entries of L^T are sufficient to guarantee the replication.
- They represent a directed acyclic graph (DAG) $\mathcal{T}(A) \subseteq \mathcal{G}(L^T)$.
- $\mathcal{T}(A)$: a special case of the transitive reduction of $\mathcal{G}(L^T)$ (simplest DAG that preserves paths in $\mathcal{G}(L^T)$.
- Equivalently: edges of $\mathcal{T}(A) \leftrightarrow$ first subdiagonal entries of L, denoted $parent(j)$: $parent(j) = \min\{i \mid i > j,\ l_{ij} \neq 0\}$.

Symbolic Cholesky

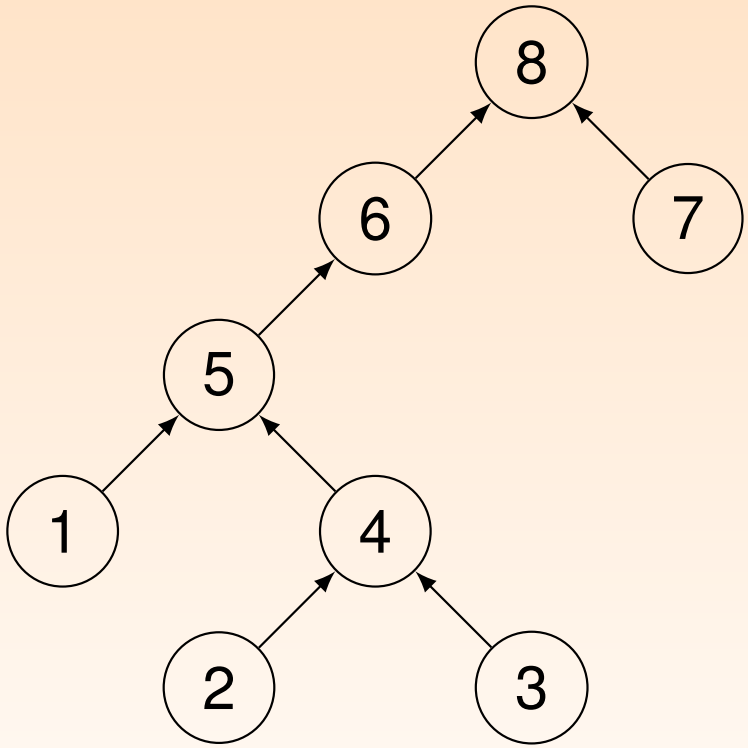
Replication of column structures once more



Symbolic Cholesky

That DAG is a tree or forest, called the elimination tree

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\left(\begin{array}{cccccccc}
* & & & & * & * & & \\
 & * & & * & * & & & * \\
 & & * & * & & & & * \\
 & * & * & * & f & & & f \\
* & * & & f & * & f & & f \\
* & & & & f & * & & f \\
 & & & & & & * & * \\
 & * & * & f & f & f & * & *
\end{array}\right)
\end{array}
$$

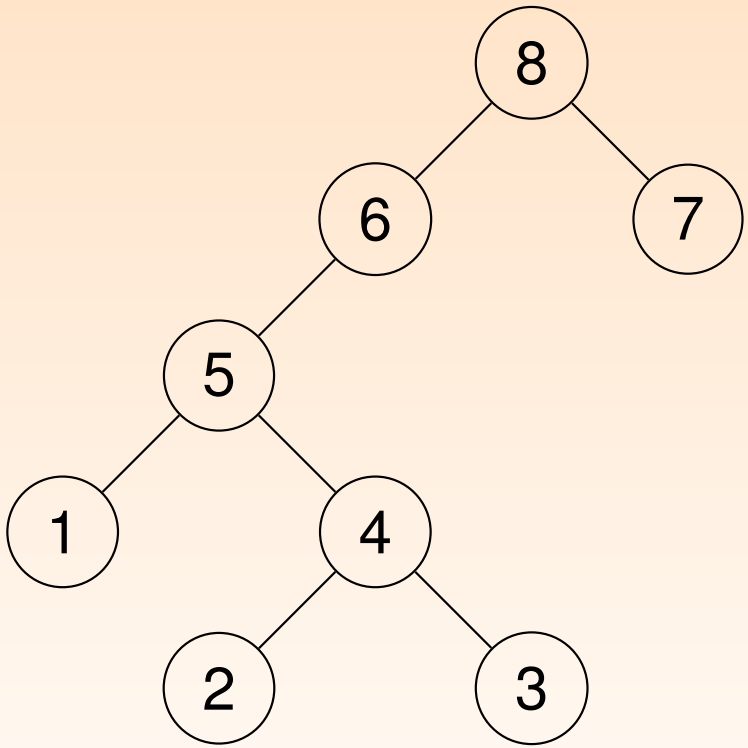


- Subtree $\mathcal{T}(5)$ includes vertices $1, 2, 3, 4, 5$; $|\mathcal{T}(5)| = 5$;

Symbolic Cholesky

That DAG is a tree or forest, called the elimination tree

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\left(\begin{array}{cccccccc}
* & & & & * & * & & \\
 & * & & * & * & & & * \\
 & & * & * & & & & * \\
 & * & * & * & f & & & f \\
* & * & & f & * & f & & f \\
* & & & & f & * & & f \\
 & & & & & & * & * \\
 & * & * & f & f & f & * & *
\end{array}\right)
\end{array}
$$

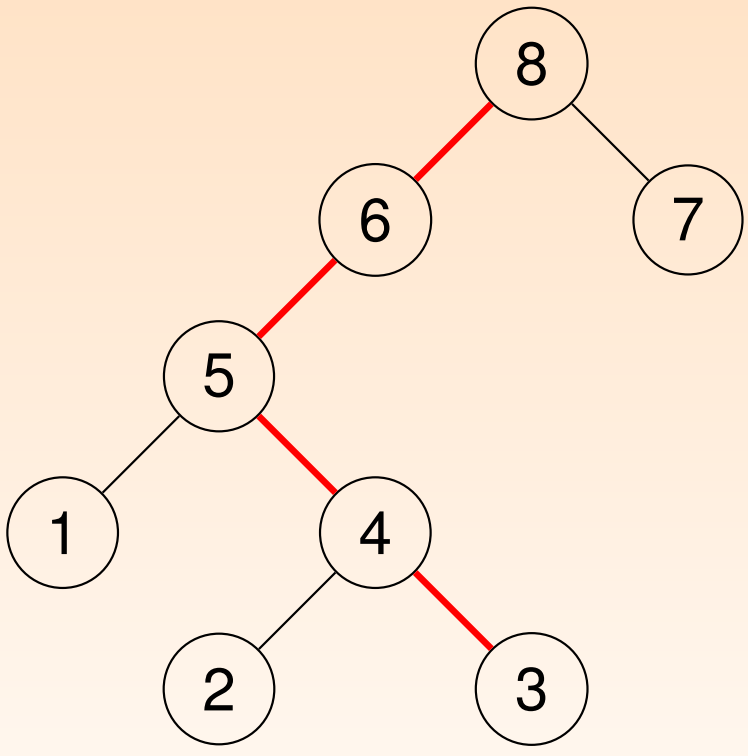


- No need to use arrows.

Symbolic Cholesky

Side-effect of column replication: row replication

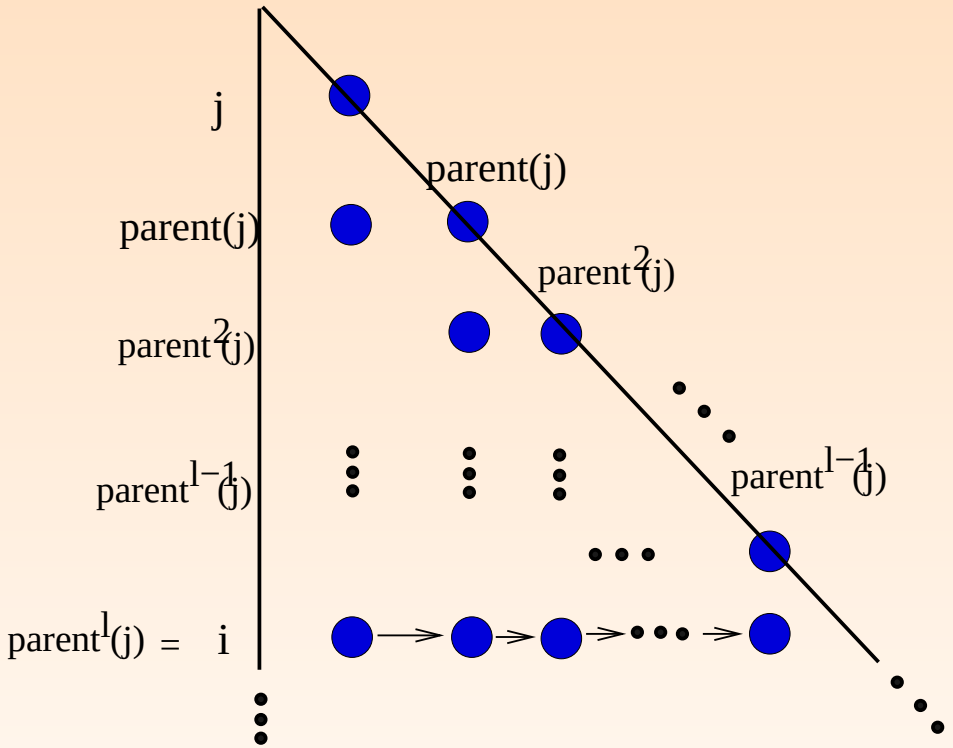
$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\left(\begin{array}{cccccccc}
* & & & & * & * & & \\
 & * & & * & * & & & * \\
 & & * & * & & & & * \\
 & * & * & * & f & & & f \\
* & * & & f & * & f & & f \\
* & & & & f & * & & f \\
 & & & & & & * & * \\
 & * & * & f & f & f & * & *
\end{array}\right)
\end{array}
$$



- $a_{8,3} \neq 0 \Rightarrow l_{8,4} \neq 0 \Rightarrow l_{8,5} \neq 0$ and so on
- This is equivalent to passing row fill up the tree due to $a_{8,3} \neq 0$.

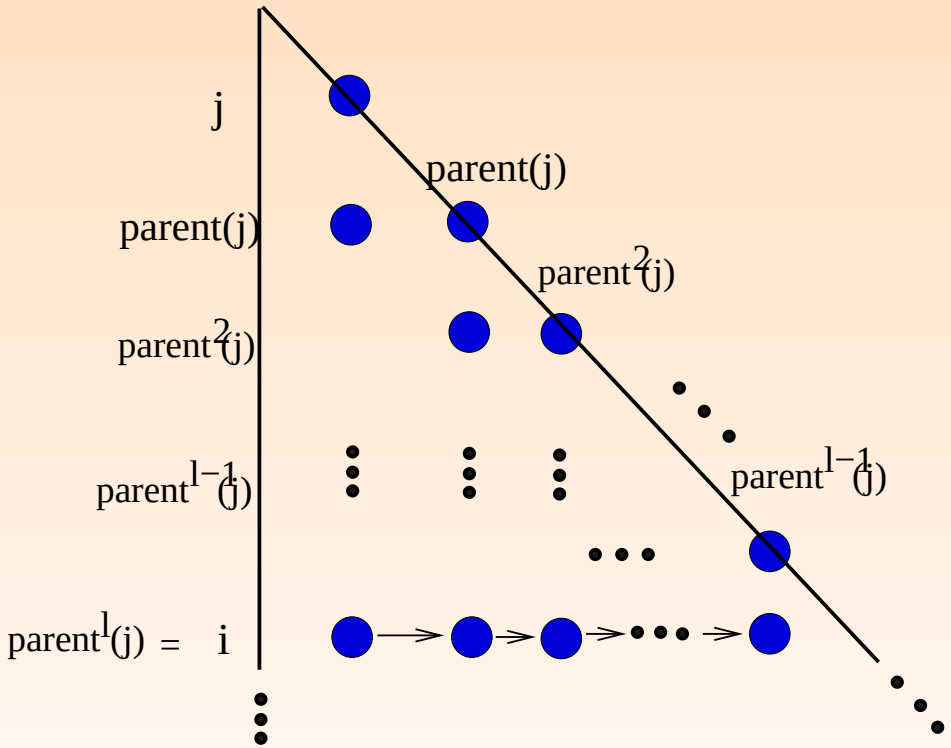
Symbolic Cholesky

Side-effect of column replication: row replication: shown again



- Replication of columns $\Rightarrow$ replication in a particular row.

Side-effect of column replication: row replication: shown again



- Replication of columns $\Rightarrow$ replication in a particular row.
- When such row replication starts? If the first entry belongs to A!

Symbolic Cholesky

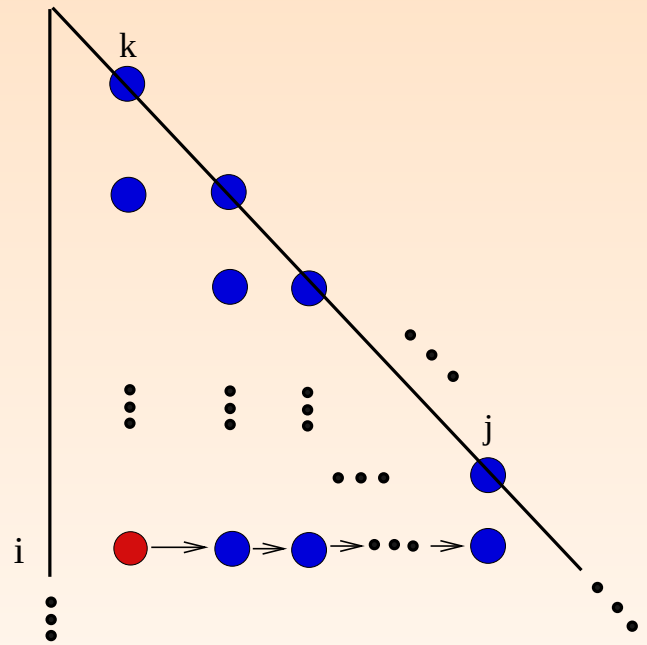
Necessary and sufficient condition for a fill-in entry

- No $k \geq 1$ with $a_{ik} \neq 0$, no replication of nonzeros in row i can start.

Symbolic Cholesky

Necessary and sufficient condition for a fill-in entry

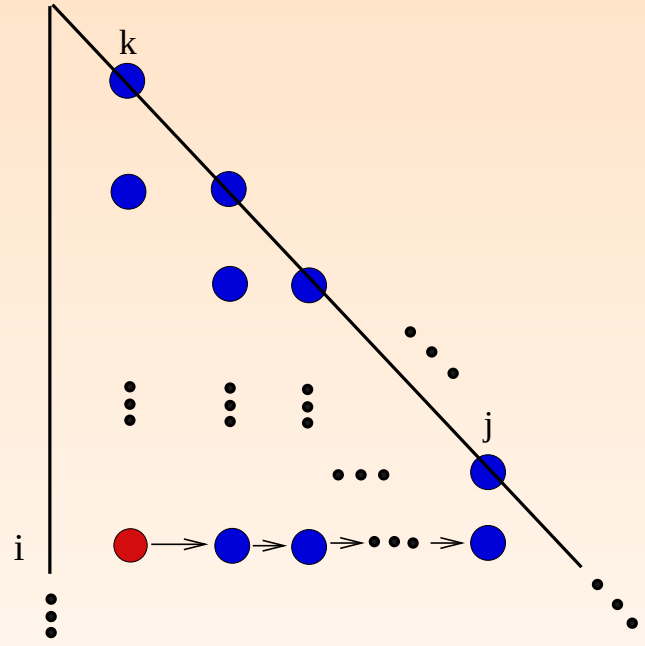
- No $k \geq 1$ with $a_{ik} \neq 0$, no replication of nonzeros in row i can start.
- Otherwise, there is a nonzero in A_{i*} that starts the row replication.



Symbolic Cholesky

Necessary and sufficient condition for a fill-in entry

- No $k \geq 1$ with $a_{ik} \neq 0$, no replication of nonzeros in row i can start.
- Otherwise, there is a nonzero in A_{i*} that starts the row replication.

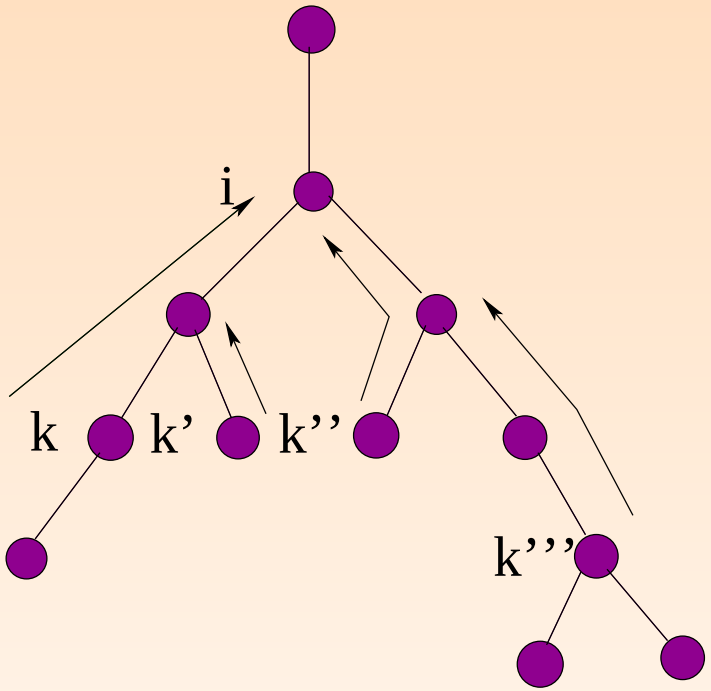


Theorem

Let A be SPD and let L be its Cholesky factor. If $a_{ij} = 0$ for some $1 \leq j < i \leq n$ then there is a filled entry $l_{ij} \neq 0$ if and only if there exists $k < j$ and $t \geq 1$ such that $a_{ik} \neq 0$ and $parent^t(k) = j$.

Symbolic Cholesky

Taking all replications in row i, we have its structure in L.



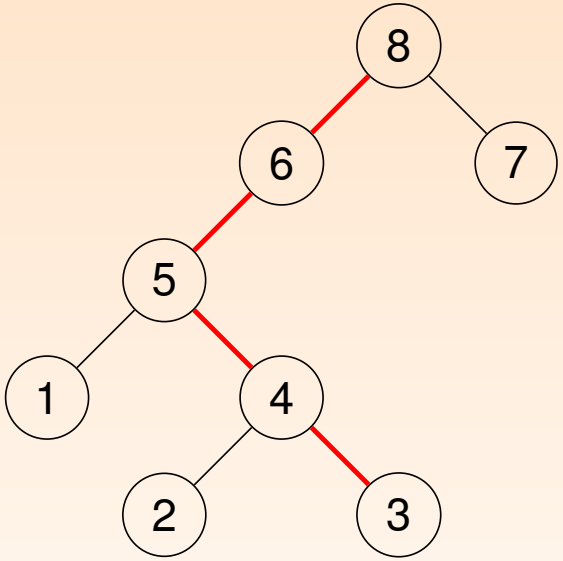
- The subgraph of $\mathcal{T}(A)$ determines it
  - Detached by k, k', k'' and k''' from below (corresponding to nonzeros $a_{i,k}$, $a_{i,k'}$, $a_{i,k''}$ and $a_{i,k'''}$), by i from above.
  - called the i-th row subtree of $\mathcal{T}(A)$.
- Its vertices precisely determine nonzeros in the i-th row of L.
- But, for factorization we may prefer to know column structure of L

Symbolic Cholesky

It would be nice to know column sparsity patterns of L as well

- Repetition: Row structures: going up $\mathcal{T}(A)$ from nonzeros of A (k, k', k'' and k''').

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\left(\begin{array}{cccccccc}
* & & & & * & * & & \\
 & * & & * & * & & & * \\
 & & * & * & & & & * \\
 & * & * & * & f & & & f \\
* & * & & f & * & f & & f \\
* & & & & f & * & & f \\
 & & & & & & * & * \\
 & * & * & f & f & f & * & *
\end{array}\right)
\end{array}
$$

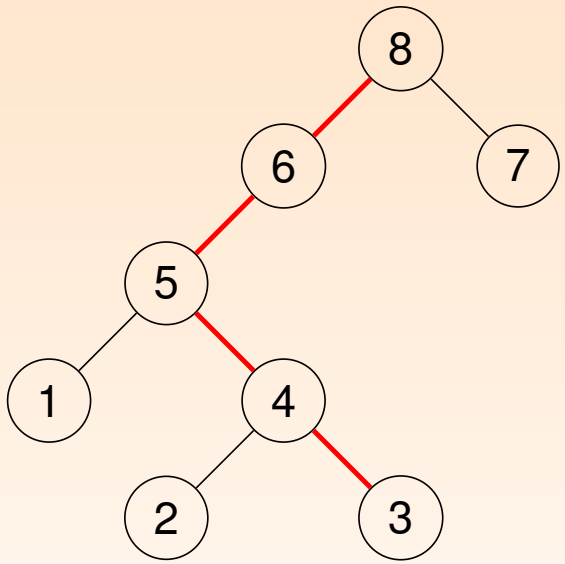


Symbolic Cholesky

It would be nice to know column sparsity patterns of L as well

- Repetition: Row structures: going up $\mathcal{T}(A)$ from nonzeros of A (k, k', k'' and k''').

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\left(\begin{array}{cccccccc}
* & & & & * & * & & \\
 & * & & * & * & & & * \\
 & & * & * & & & & * \\
 & * & * & * & f & & & f \\
* & * & & f & * & f & & f \\
* & & & & f & * & & f \\
 & & & & & & * & * \\
 & * & * & f & f & f & * & *
\end{array}\right)
\end{array}
$$

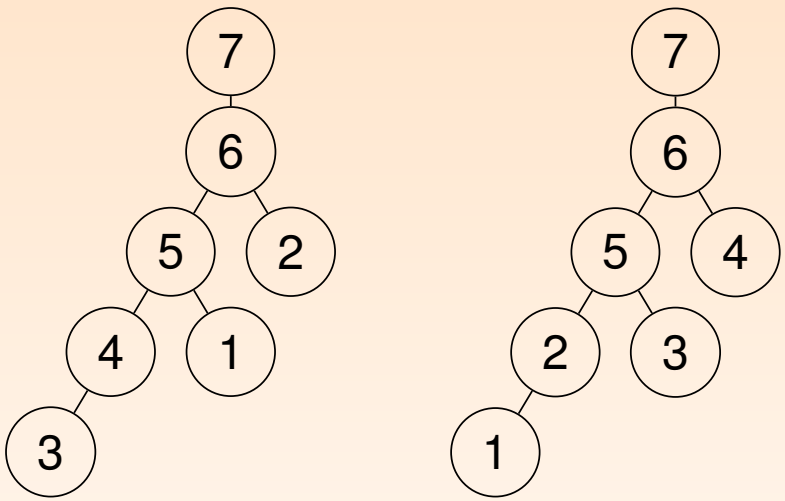


- Column structures: merging column lists: $col_L\{j\} = adj_{\mathcal{G}(A)}\{\mathcal{T}(j)\}$ e.g., $col_L\{5\} = adj_{\mathcal{G}(A)}(1) \cup adj_{\mathcal{G}(A)}(2) \cup adj_{\mathcal{G}(A)}(3) \cup adj_{\mathcal{G}(A)}(5)$
- Up the tree. This is clear, but implementation may be funny.

Symbolic Cholesky

Getting column structures more efficiently

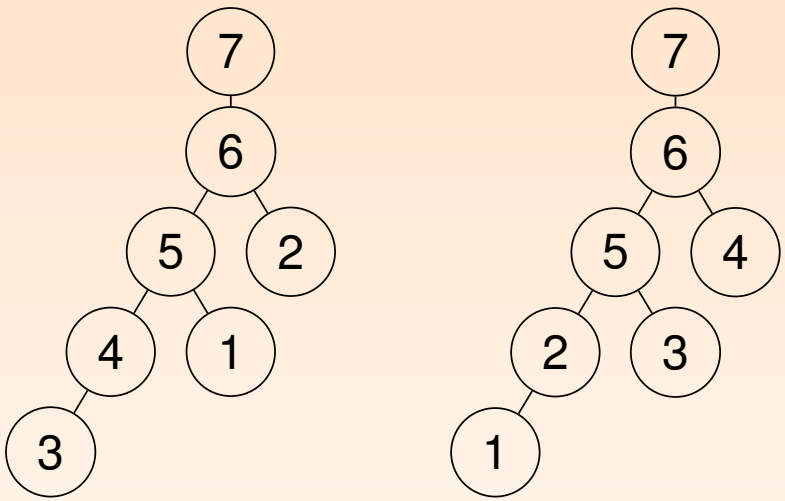
- First define: A labeling of the vertices of a tree (and, more generally, in a DAG) is a topological ordering if, for all i and j, $j \in desc_{\mathcal{T}}\{i\}$ implies $j < i$



Symbolic Cholesky

Getting column structures more efficiently

- First define: A labeling of the vertices of a tree (and, more generally, in a DAG) is a topological ordering if, for all i and j, $j \in desc_{\mathcal{T}}\{i\}$ implies $j < i$



- Apparently, the second labeling is better.
- Why? It localizes. $\rightarrow$
- $\mathcal{S}(L)$ by columns is obtained by merging columns, the merged columns should not wait too long to be merged again, in order to use small intermediate memory.

Symbolic Cholesky

All topological orderings are nice

Sparsity patterns of the Cholesky factors of A and PAP^T can be different, but the amount of fill-in is the same.

Theorem

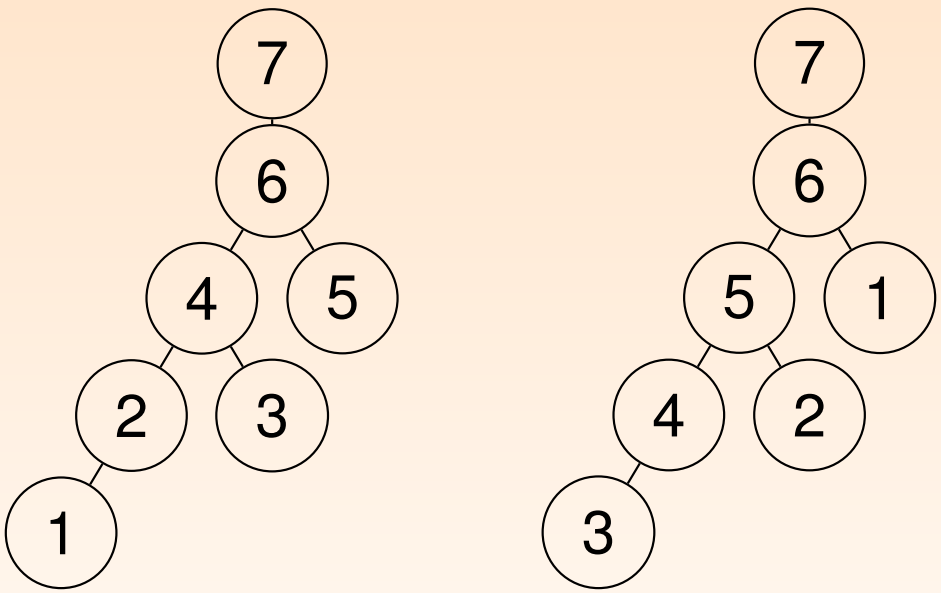
Let $\mathcal{S}\{A\}$ be symmetric. If P is the permutation matrix corresponding to a topological reordering of the elimination tree $\mathcal{T}$ of A then the filled graphs of A and PAP^T are isomorphic.

- Topological orderings do not change fill-in size
- In the other words, the amount of fill-in is invariant under the class of topological reorderings of the elimination tree.

Symbolic Cholesky

... are nice. But, some topological orderings are nicer: postordering

- A topological ordering of $\mathcal{T}$ is a postordering if the vertex set of any subtree $\mathcal{T}(i)$ ($i = 1, \ldots, n$) is a contiguous sublist of $1, \ldots, n$.

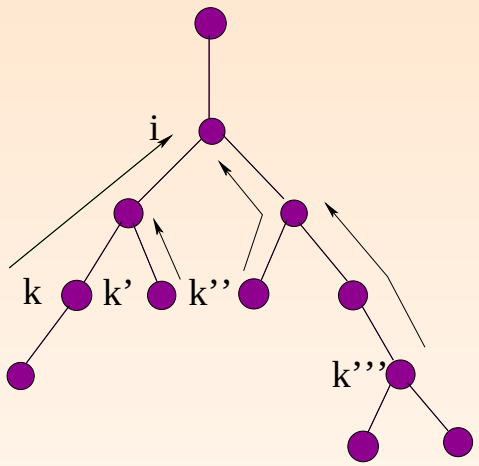


- Postordering is even more localizing labeling.
- Needed in fast algorithms.

Symbolic Cholesky

Ooops. We still do not have the elimination tree. How to get it?

- To find $\mathcal{T}(A)$, we just mimick the row replication: scan A by rows for $i = 1, \ldots, n-1$ and go up the constructed part of $\mathcal{T}(A)$ to attach i as a temporary root.



- During the search if the i-th row, vertex i is either put on the top of the current structure or added as an isolated vertex if not connected to the rest of $\mathcal{T}(A)$ yet.

Constructing elimination tree: complexity

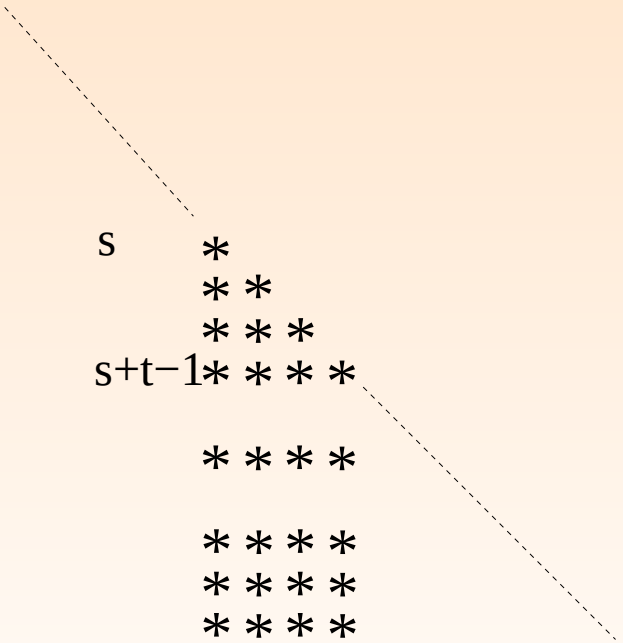
- A complexity problem

$$\begin{pmatrix} * & * & * & * & * & * \\ * & * & & & & \\ * & & * & & & \\ * & & & * & & \\ * & & & & * & \\ * & & & & & * \end{pmatrix} \qquad \begin{pmatrix} * & * & * & * & * & * \\ * & * & f & f & f & f \\ * & f & * & f & f & f \\ * & f & f & * & f & f \\ * & f & f & f & * & f \\ * & f & f & f & f & * \end{pmatrix}$$

- $\mathcal{T}(A)$: $parent(6) = 0$; $parent(i) = i + 1$, $i = 1, \ldots, 5$.
- For each i we start from a_{i1} and attach i at the top of the partial $\mathcal{T}$: $O(n^2)$ complexity
- But, improvements lead to the nearly linear complexity of getting $\mathcal{T}(A)$.

What else: blocks

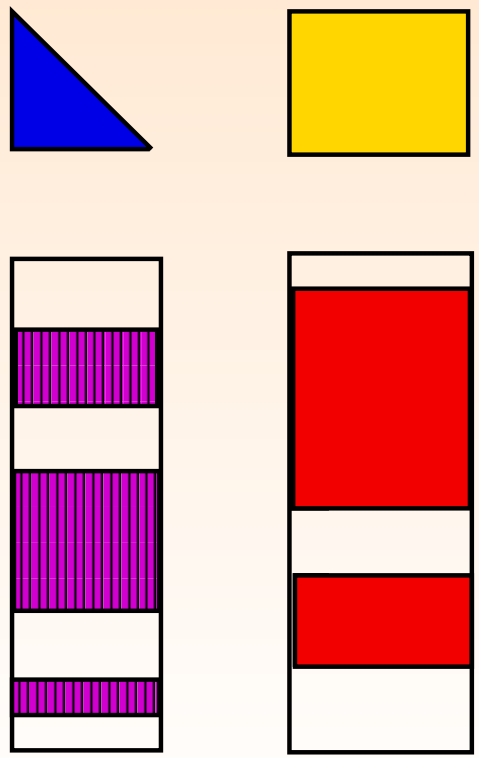
- They look like this. In L! They are called the supernodes.



- Replication principle increases their probability.

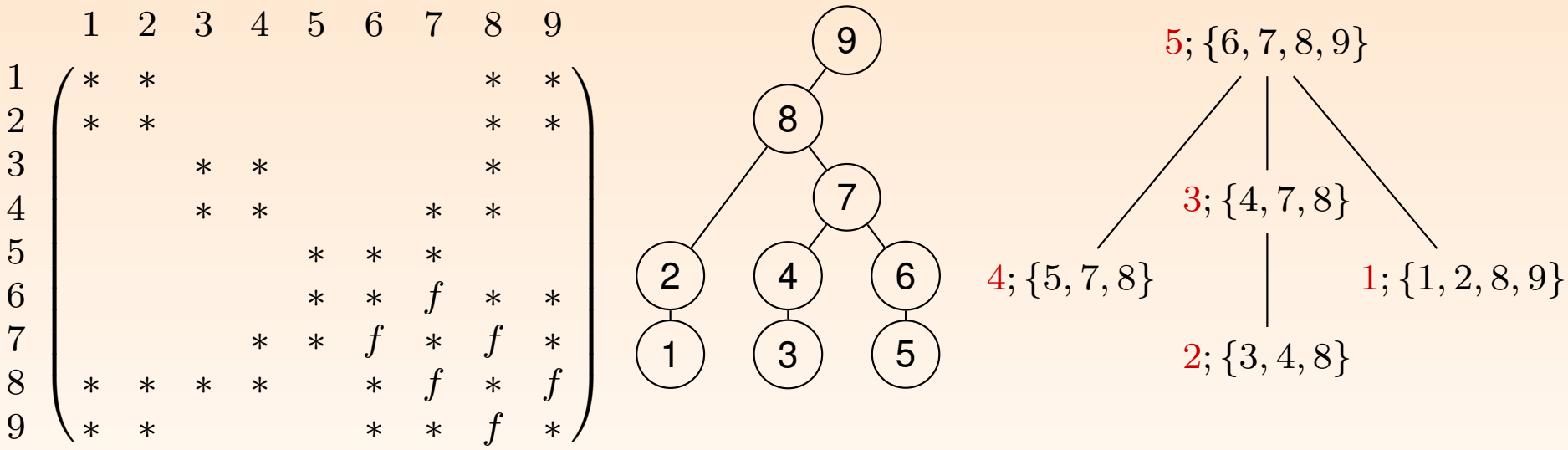
Supernodes and efficient computation

- the loop over columns of the updating supernode can be unrolled to save memory references (dense BLAS2)
- parts of the updating supernode can be used for blocks of updated supernode (dense BLAS3)



Supernodes: block-based elimination

- Supernodes imply the supernodal elimination (assembly) tree.



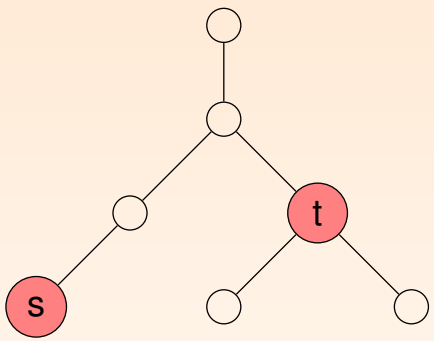
- Their important type can be found in a nearly linear complexity.

Symbolic Cholesky

Independence of subtrees: parallelism at hand

Theorem

Consider the elimination tree $\mathcal{T}$ and the Cholesky factor L of A. Let $\mathcal{T}(i)$ and $\mathcal{T}(j)$ be two vertex-disjoint subtrees of $\mathcal{T}$. Then for all $s \in \mathcal{T}(i)$ and $t \in \mathcal{T}(j)$ the entry l_{st} of L is zero.



- Of course, $l_{st} = 0$. Otherwise t would have to be ancestor of s or vice versa.
- Column structures and columns can be merged independently. Contradiction with row replication.

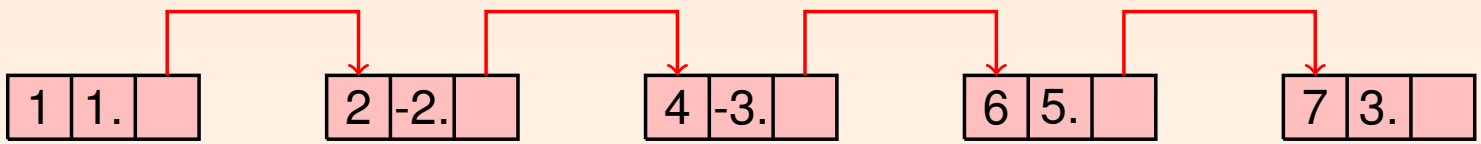
Outline

Sparse vectors and matrices in a computer

Sparse data (matrix/row/column) in a computer: I. dynamic formats

- a) Coordinate (or triplet format for data: individual entries of A held as triplets (i, j, a_{ij}), where i is the row index and j is the column index of the entry $a_{ij} \neq 0$; similar for vectors

- b) Linked list - based format: stores data as linked items



- Linked lists can be cyclic, one-way, two-way, etc., can be embedded into a larger array: emulated dynamic behavior

Sparse vectors and matrices in a computer

Sparse matrix storage: II. static formats

- CSR (Compressed Sparse Row) static format. The column indices compressed in the array `colindA` by rows. Sorted or unsorted. CSC: variant by columns.

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{pmatrix}
3. & & & -2. & \\
 & 1. & & & 4. \\
-1. & & 3. & & 1. \\
 & & & 1. & \\
 & 7. & & & 6.
\end{pmatrix}
\end{array}.
$$

Indices	1	2	3	4	5	6	7	8	9	10
`rowptrA`	1	3	5	8	9	11				
`colindA`	1	4	2	5	1	3	5	4	2	5
`valA`	3.	-2.	1.	4.	-1.	3.	1.	1.	7.	6.

- If A is symmetric, only the lower (or upper) triangular part stored.
- Possible to store only $\mathcal{S}\{A\}$ and not numerical values.
- Useful: static, theory helps to use them efficiently

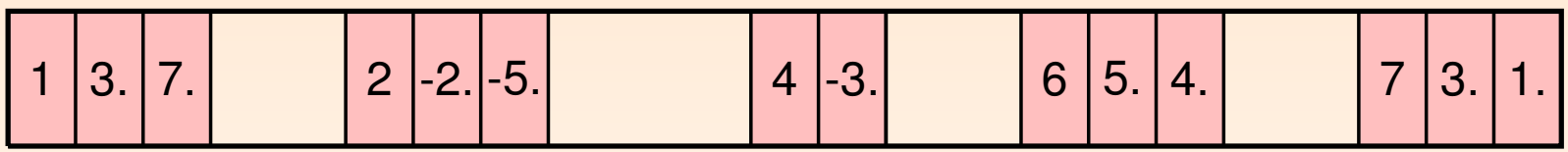
Sparse vectors and matrices in a computer

Sparse matrix storage: static versus dynamic formats

- dynamic data structures:
  - – more flexible but this flexibility might not be needed
  - – difficult to vectorize
  - – difficult to keep spatial locality of rows and columns
  - – used preferably for storing vectors
- static data structures:
  - – ad-hoc insertions/deletions should be avoided (better algorithms)
  - – much simpler to vectorize / utilize cache
  - – efficient access to rows/columns

Simulating dynamic storage formats by static ones

- Dynamic storage formats can be simulated by
  - adding to CSR/CSC an elbow space for fill-in entries



  - A mechanism to compress/extend such structure needed.
  - Useful for approximate factorization with limited fill-in.

Outline

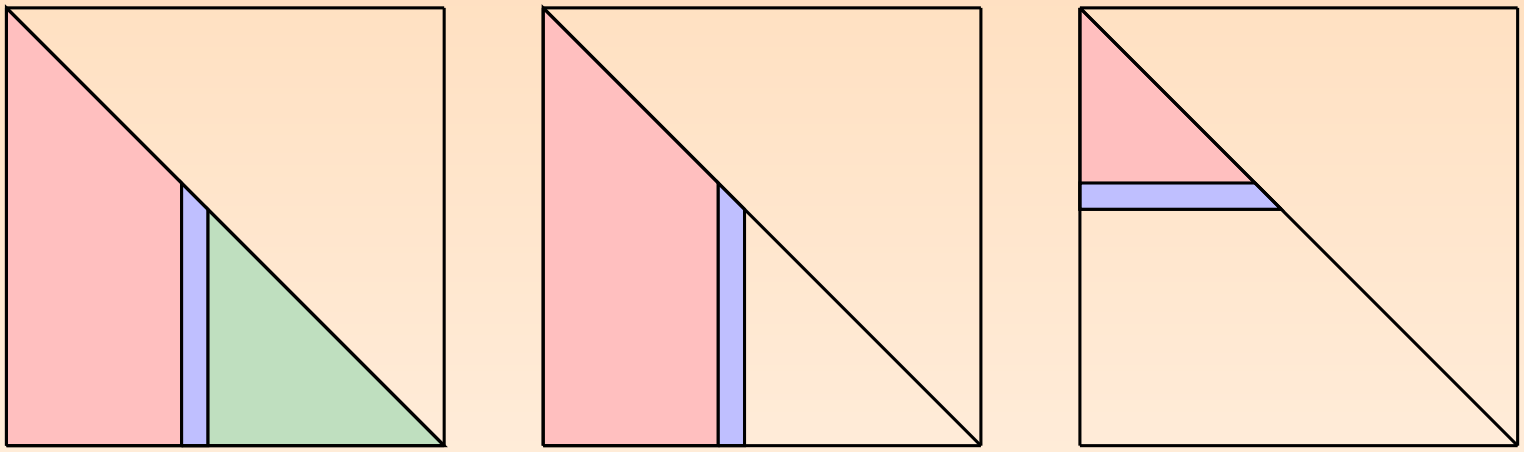
Cholesky Factorization

Sparse Cholesky factorization: conceptual comments

- Efficient symbolic phase based on $\mathcal{T}$: explained:
  - Row/column counts of L known $\rightarrow$ storage can be allocated
  - Postordering enables a lot of other efficient algorithms
  - Blocks: supernodes
  - Technical tricks as splitting large supernodes into smaller panels to embed them into computer caches
- Numerical factorization: new important feature $\rightarrow$ more communication: this is described by a communication graph: DAG (directed acyclic)

Cholesky Factorization

Numerical Cholesky factorization: from operations to tasks



a) **b)** **c)**

- $cdiv(k)$: scaling column k by the square root of the diagonal entry
- $cmod(j,k)$: column j modified by a multiple of column k

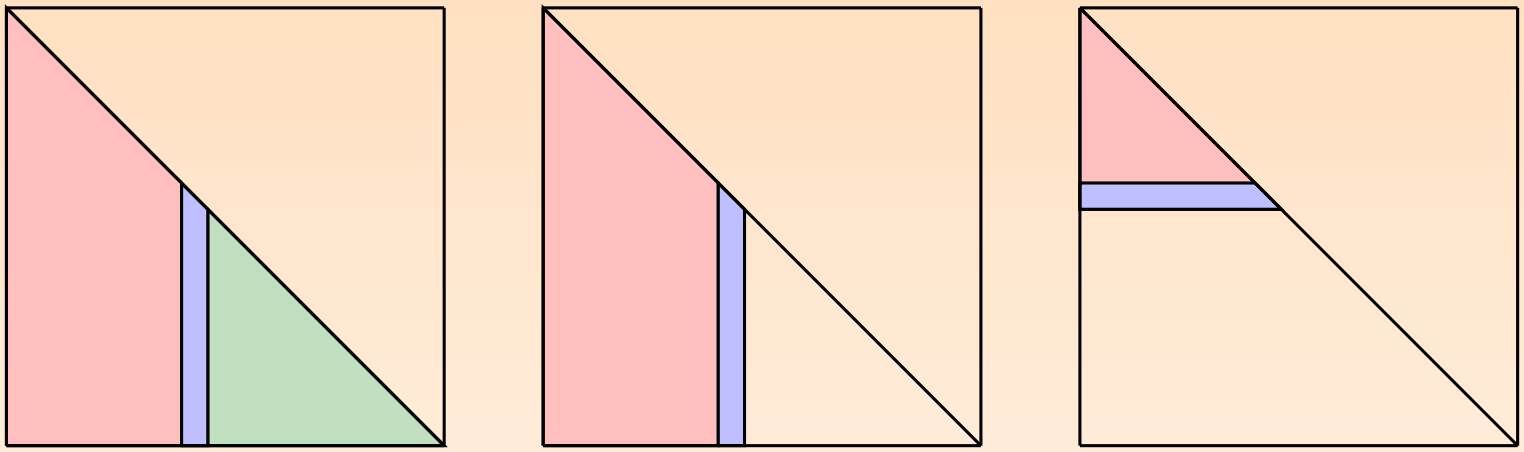
Algorithm

Sparse column (left-looking) Cholesky

1: ***for*** $j = 1 : n$ ***do***
2: ***for*** $k \in Struct(L_{j*})$ ***do*** ▷ *All of them !!!!!!!*
3: $cmod(j,k)$
4: ***end for***
5: $cdiv(j)$
6: ***end for***

Cholesky Factorization

Numerical Cholesky factorization: from operations to tasks



a) **b)** **c)**

- $cdiv(k)$: scaling column k by the square root of the diagonal entry
- $cmod(j,k)$: column j modified by a multiple of column k

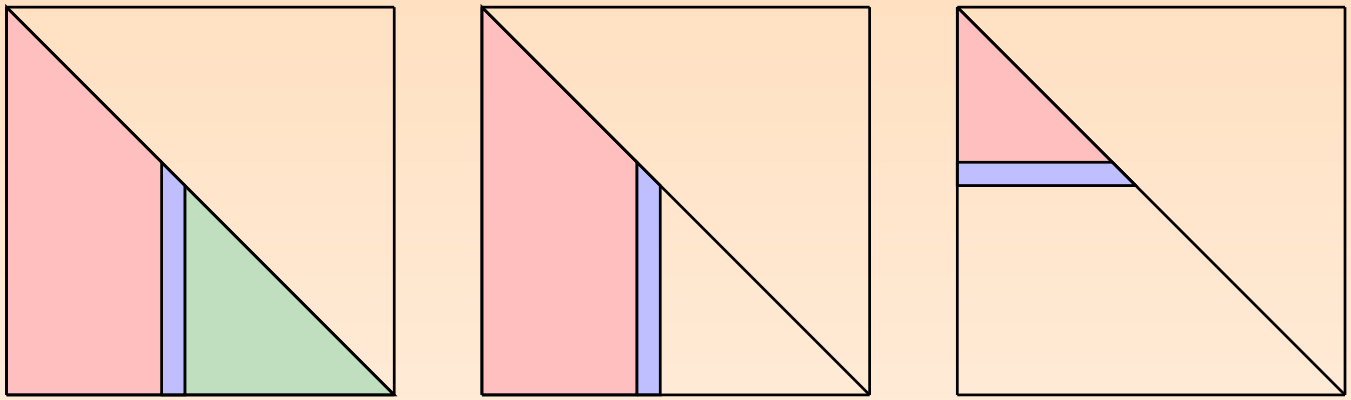
Algorithm

Sparse submatrix (right-looking) Cholesky

1: ***for*** $k = 1 : n$ ***do***
2: $\quad cdiv(k)$
3: $\quad$ ***for*** $j \in Struct(L_{*k})$ ***do***
4: $\quad\quad cmod(j,k)$
5: $\quad$ ***end for***
6: ***end for***

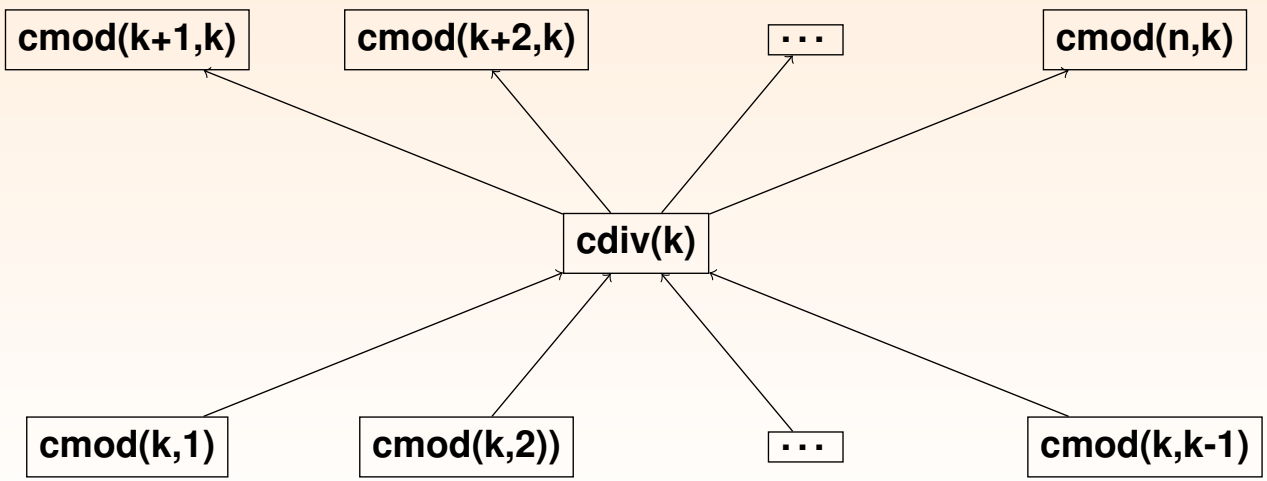
Cholesky Factorization

Splitting Cholesky factorization into tasks



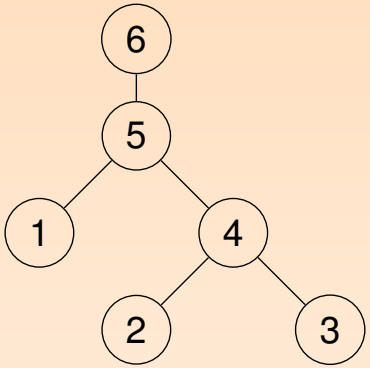
a) **b)** **c)**

- $cdiv(k)$: scaling column k by the square root of the diagonal entry
- $cmod(j,k)$: column j modified by a multiple of column k

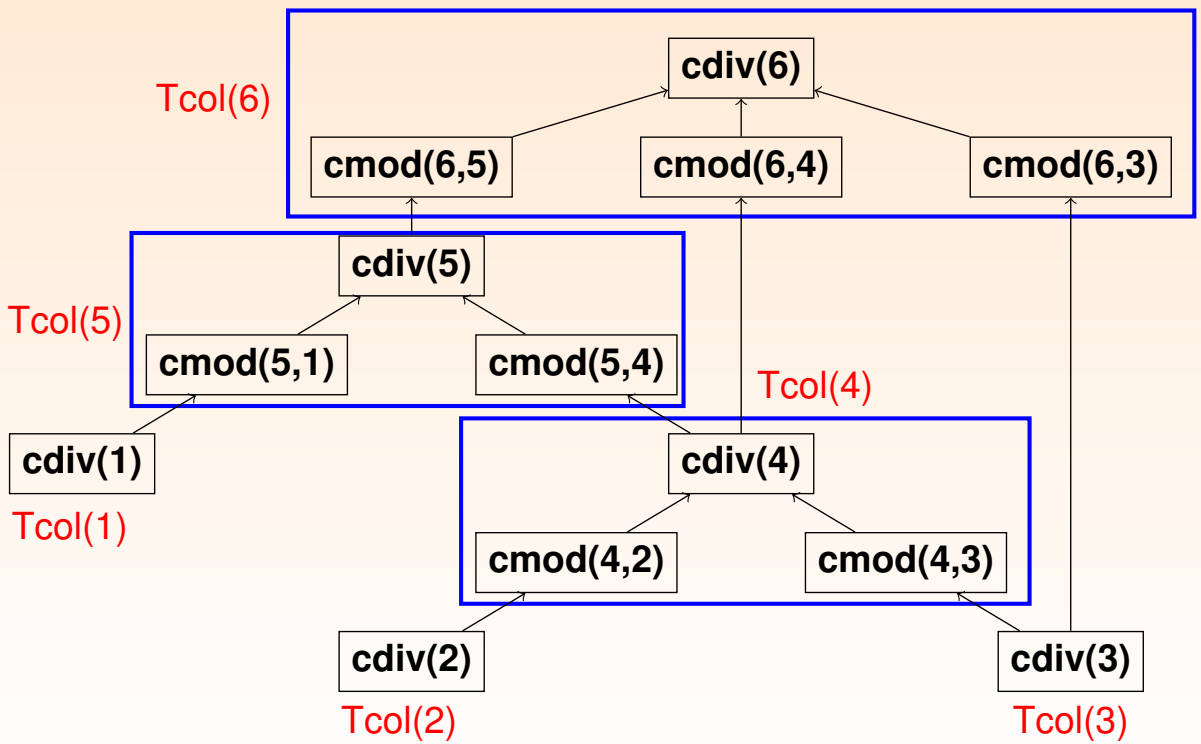


Cholesky Factorization

Large-grain column (left-looking) communication model

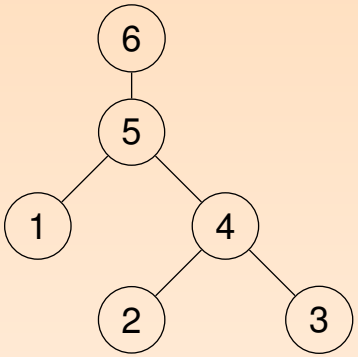


$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} * & & & & * & \\ & * & & * & & \\ & & * & * & & * \\ & * & * & * & * & f \\ * & & & * & * & * \\ & & * & f & * & * \end{pmatrix} \end{array}$$

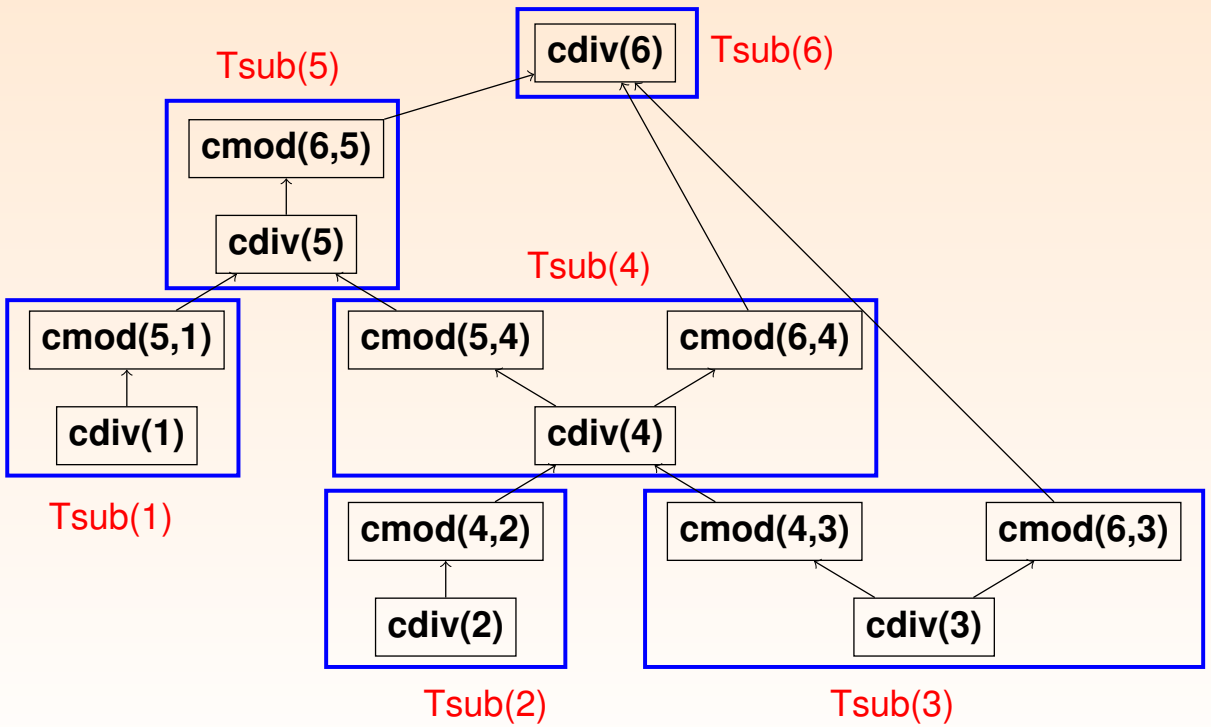


Cholesky Factorization

Large-grain submatrix (right-looking) communication model



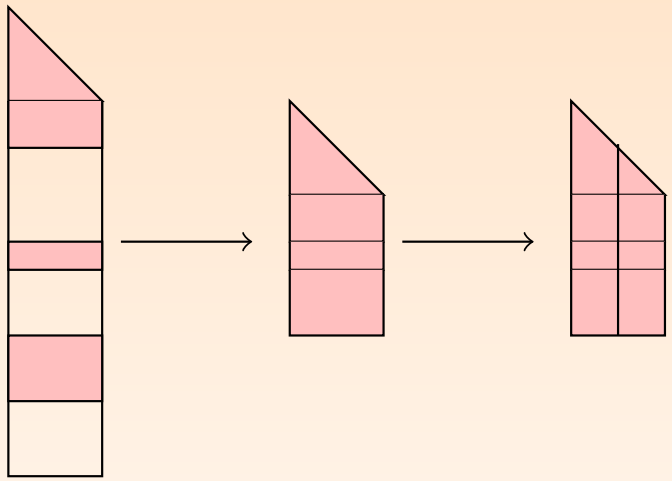
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} * & & & & * & \\ & * & & * & & \\ & & * & * & & * \\ & * & * & * & * & f \\ * & & & * & * & * \\ & & * & f & * & * \end{pmatrix} \end{array}$$



Cholesky Factorization

Using supernodes: enhancing parallel processing

- Arithmetic of dense trapezoidal matrices. Sophisticated mappings among them.



- Dependencies captured by the communication (dependency) DAG.
- The tree parallelism.
- Block arithmetic.

Cholesky Factorization

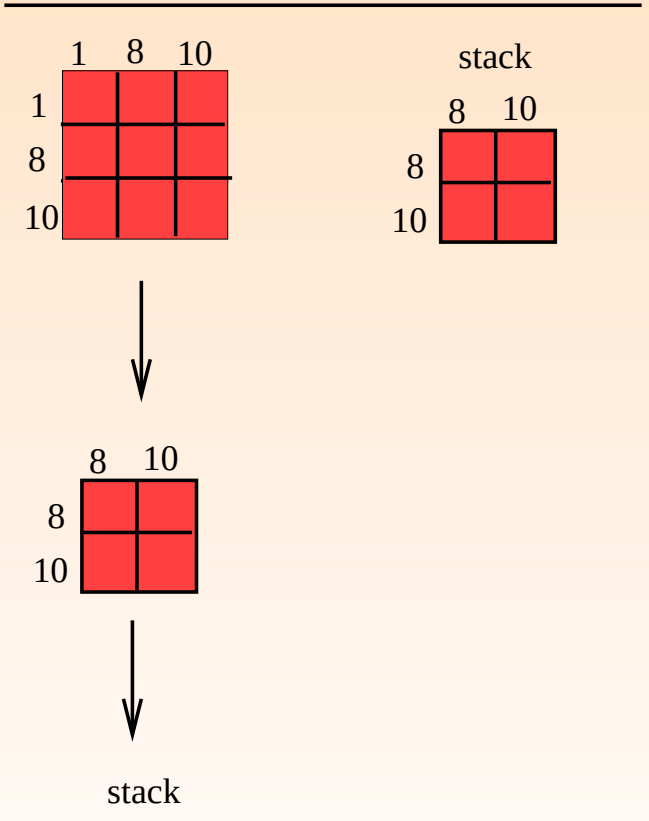
Variations of the Cholesky factorization: sparsity and supernodes

- Left-looking approach:
  - Dependency DAG
  - Block arithmetic.
- Right-looking approach:
  - Dependency DAG
  - A specific popular approach: uses the supernodal elimination tree for dependencies: the multifrontal method
  - High level of memory efficiency due to computational locality: contributions to the Schur complement kept aside in a stack

Cholesky Factorization

Multifrontal method: just sketching: updates put on a stack

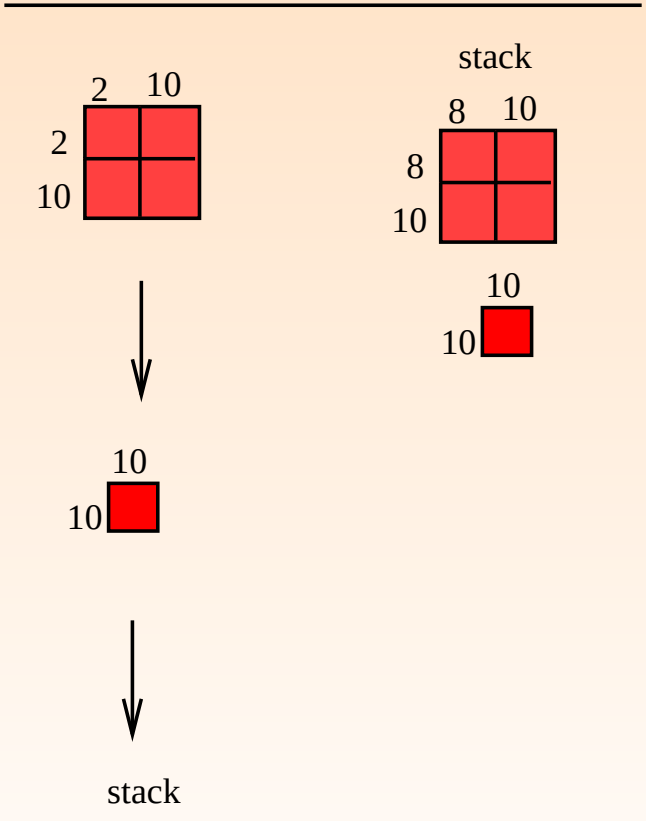
$$\begin{pmatrix}
* & & & & & & * & & * \\
 & * & & & & & & & * \\
 & & * & & & & * & & * \\
 & & & * & & & * & & \\
 & & & & * & * & & * & \\
 & & & & * & * & & f & * \\
 & & * & * & & & * & * & f \\
* & & & & & & * & & * \\
 & & & & * & f & * & * & f \\
* & * & * & & & * & f & * & f & * \\
\end{pmatrix}$$



Cholesky Factorization

Multifrontal method: just sketching: updates put on a stack

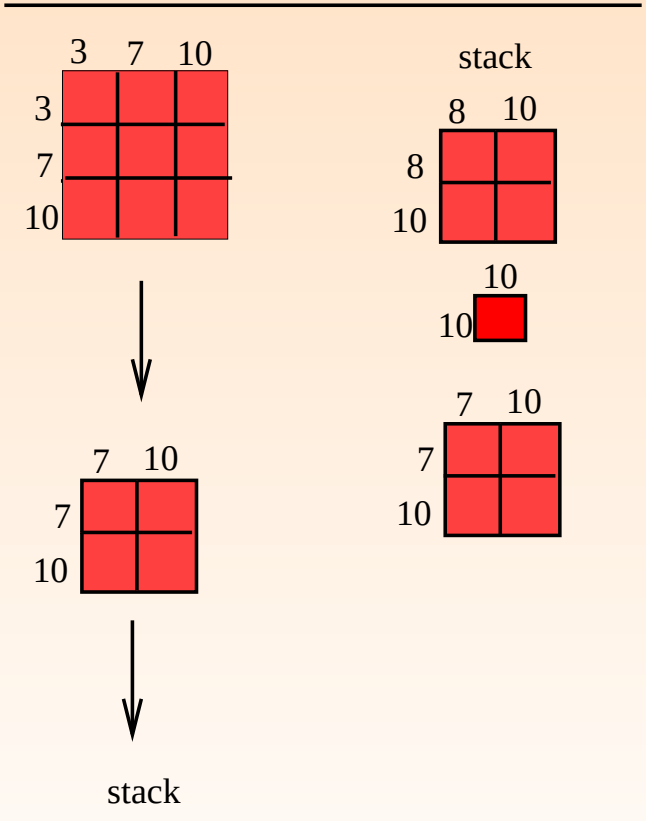
$$\begin{pmatrix}
* & & & & & & * & & * \\
 & * & & & & & & & * \\
 & & * & & & & * & & * \\
 & & & * & & & * & & \\
 & & & & * & * & & * & \\
 & & & & * & * & & f & * \\
 & & * & * & & & * & * & f \\
* & & & & & & * & & * \\
 & & & & * & f & * & * & f \\
* & * & * & & & * & f & * & f & * \\
\end{pmatrix}$$



Cholesky Factorization

Multifrontal method: just sketching: updates put on a stack

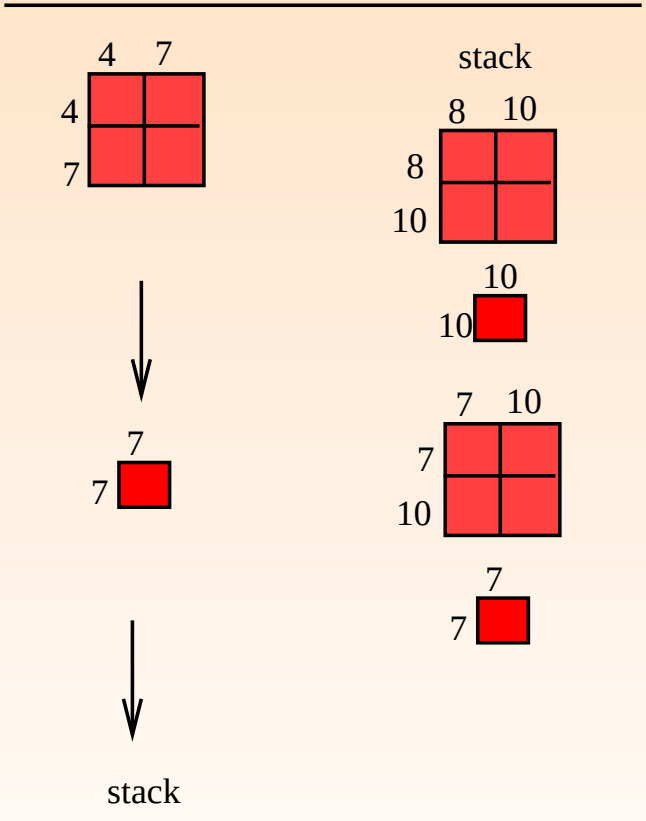
$$\begin{pmatrix}
* & & & & & & * & & * \\
 & * & & & & & & & * \\
 & & * & & & & * & & * \\
 & & & * & & & * & & \\
 & & & & * & * & & * & \\
 & & & & * & * & & f & * \\
 & & * & * & & & * & & * & f \\
* & & & & & & & * & & * \\
 & & & & * & f & * & & * & f \\
* & * & * & & & * & f & * & f & *
\end{pmatrix}$$



Direct methods: Multifrontal method

Multifrontal method: just sketching: updates put on a stack

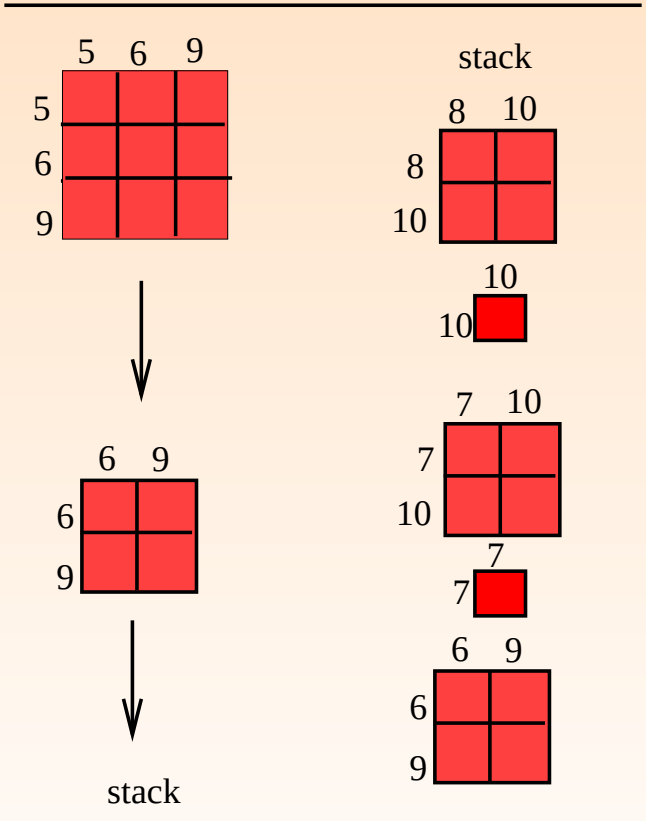
$$\begin{pmatrix}
* & & & & & & * & & * \\
 & * & & & & & & & * \\
 & & * & & & & * & & * \\
 & & & * & & & * & & \\
 & & & & * & * & & * & \\
 & & & & * & * & & f & * \\
 & & * & * & & & * & * & f \\
* & & & & & & * & & * \\
 & & & & * & f & * & & * & f \\
* & * & * & & & * & f & * & f & *
\end{pmatrix}$$



Cholesky Factorization

Multifrontal method: just sketching: updates put on a stack

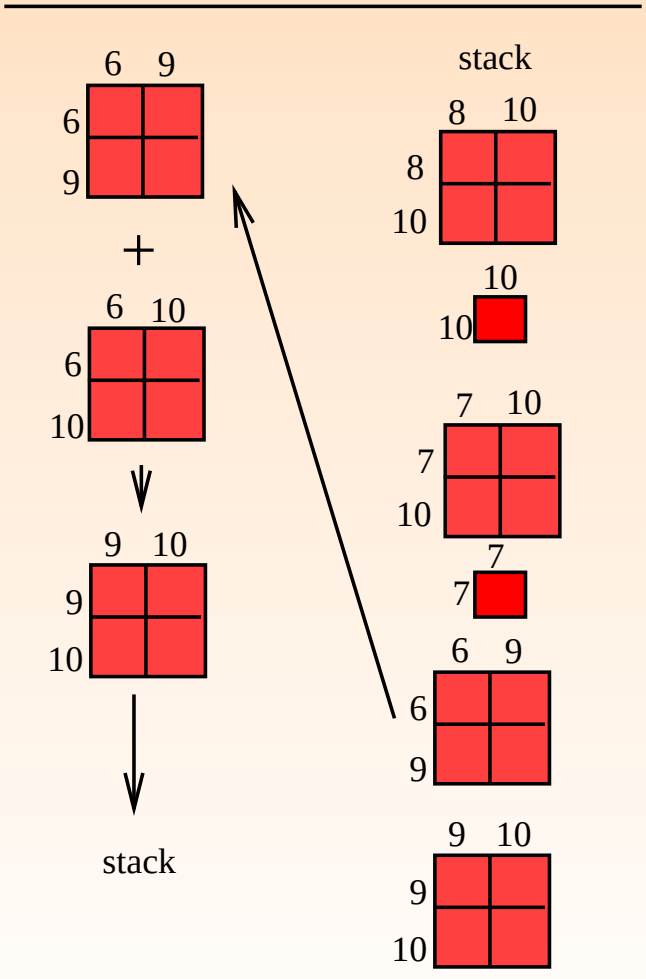
$$\begin{pmatrix}
* & & & & & & * & & * \\
 & * & & & & & & & * \\
 & & * & & & & * & & * \\
 & & & * & & & * & & \\
 & & & & * & * & & * & \\
 & & & & * & * & & f & * \\
 & & * & * & & & * & & * & f \\
* & & & & & & & * & & * \\
 & & & & * & f & * & & * & f \\
* & * & * & & & * & f & * & f & *
\end{pmatrix}$$



Cholesky Factorization

Multifrontal method: just sketching: updates put on a stack

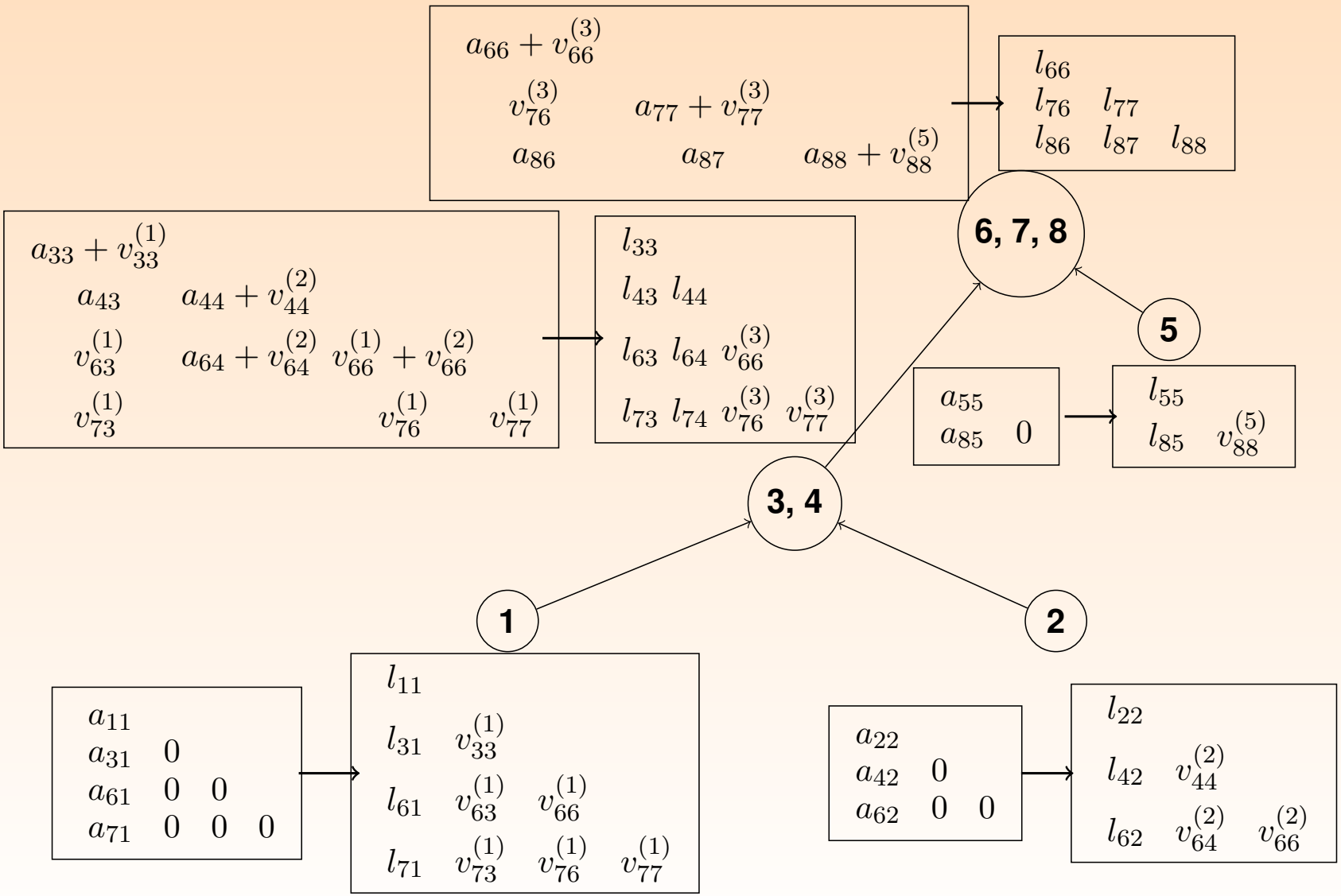
$$\begin{pmatrix}
* & & & & & & * & & * \\
 & * & & & & & & & * \\
 & & * & & & & * & & * \\
 & & & * & & & * & & \\
 & & & & * & * & & * & \\
 & & & & * & * & & f & * \\
 & & * & * & & & * & * & f \\
* & & & & & & & * & * \\
 & & & & * & f & * & * & f \\
* & * & * & & & * & f & * & f & * \\
\end{pmatrix}$$



Multifrontal method: another example matrix

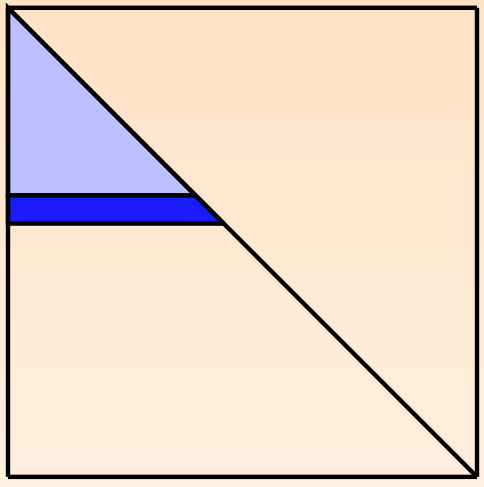
$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{pmatrix}
* & & & & & & & \\
 & * & & & & & & \\
* & & * & & & & & \\
 & * & * & * & & & & \\
 & & & & * & & & \\
* & * & f & * & & * & & \\
* & & f & f & & f & * & \\
 & & & & * & * & * & *
\end{pmatrix}
\end{array}
$$

Multifrontal method: details



Cholesky Factorization

Sparse Cholesky factorizations: up-looking factorization



- An alternative for sparse matrices is to compute L one row at a time. This is sometimes called an up-looking factorization.
- Asymptotically optimal, but difficult to incorporate high level BLAS.
- Also an efficient symbolic phase possible.
- High potential for approximate factorizations

Outline

Sparse LU: models and methods

LU factorization and graphs and methods

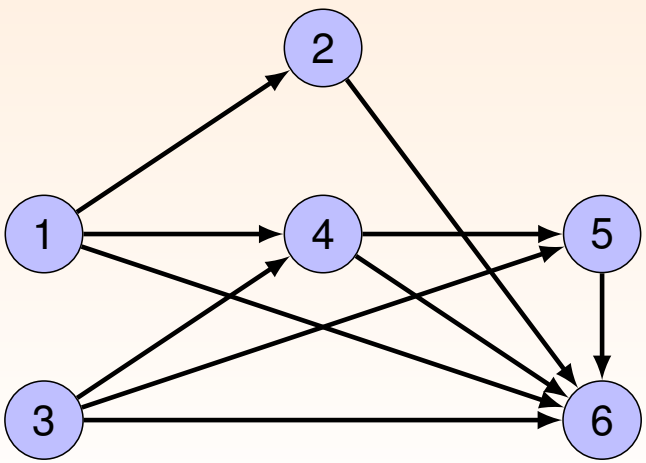
- Differences with respect to Cholesky (roughly):
  - Two factors: more general graph models (directed, bipartite) needed to describe A and the factors
  - Problems with factorizability: symbolic and numerical steps cannot be always separated
  - Due to this, sometimes stronger assumptions needed, sometimes on-the-fly changes: pivoting

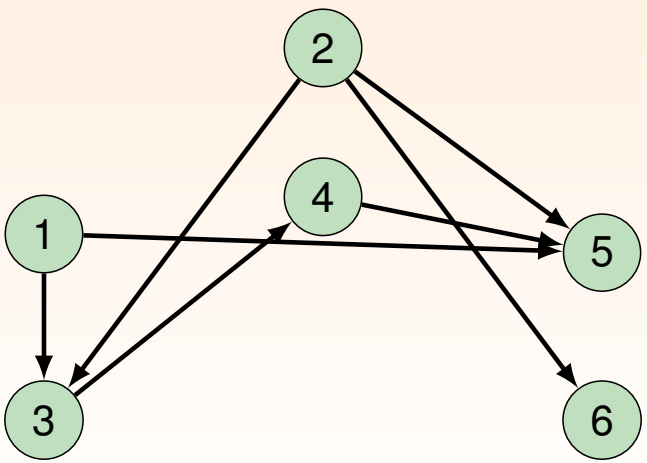
Sparse LU factorization of generally nonsymmetric matrices

LU factorization: first symbolic model: DAGs

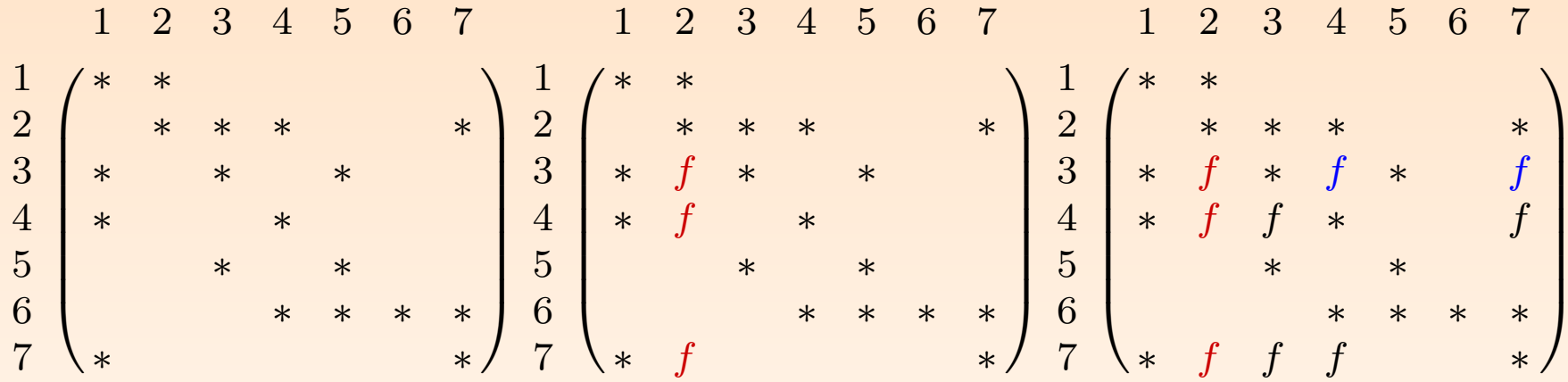
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} * & & * & & * & \\ * & * & & & * & * \\ & & * & * & & \\ * & & * & * & & \\ & & * & & * & \\ * & * & & & & * \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} * & & * & & * & \\ * & * & f & & * & * \\ & & * & * & & \\ * & & * & * & f & \\ & & * & f & * & \\ * & * & f & f & f & * \end{pmatrix} \end{array}$$

- Directed acyclic graphs for the factors capture their structure. We use $G(L^T)$ (L by columns, left) and $G(U)$ (U by rows, right).





LU factorization and DAGs: alternating replications

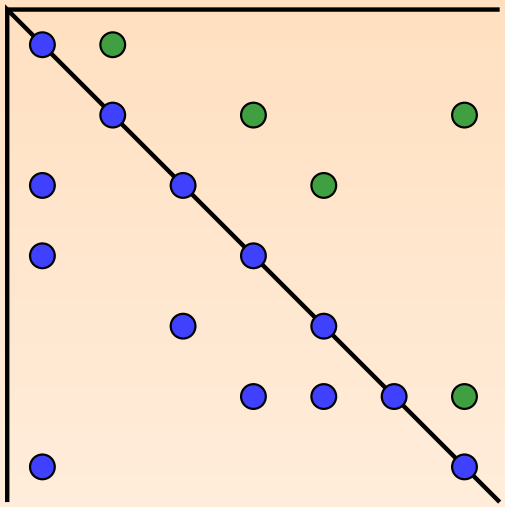


- Alternating column and row replication (in the submatrix model).
- Left: A. Centre: showing one column replication. Right: also a row replication.

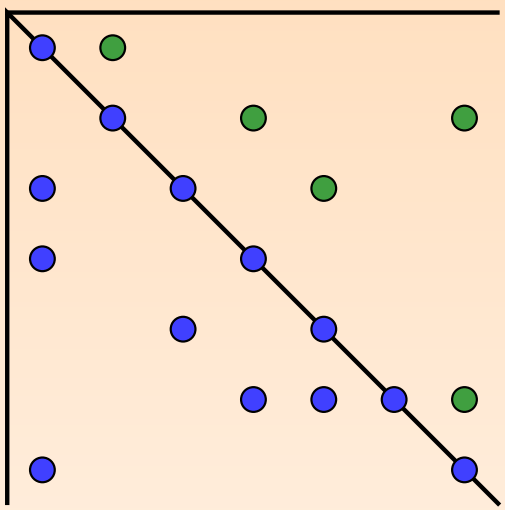
Recursive alternating replications

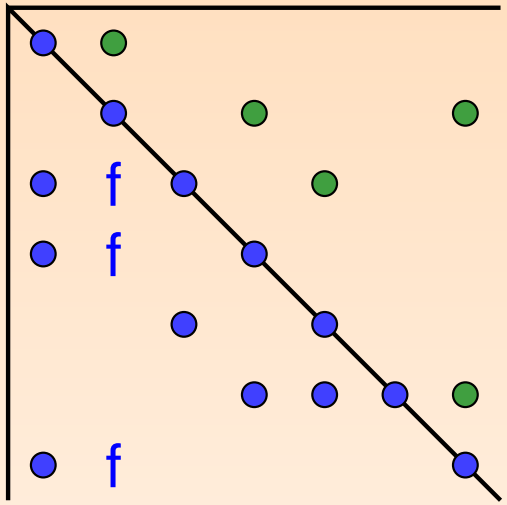
- Symmetric factorization: the recursive replications driven by the parents, subgraph of $G(L^T)$.
- In LU it is more interesting ☺:
  - for columns of L: directed paths in U are used
  - for rows of U: directed paths in $\mathcal{G}(L^T)$.

Column replication in LU: example

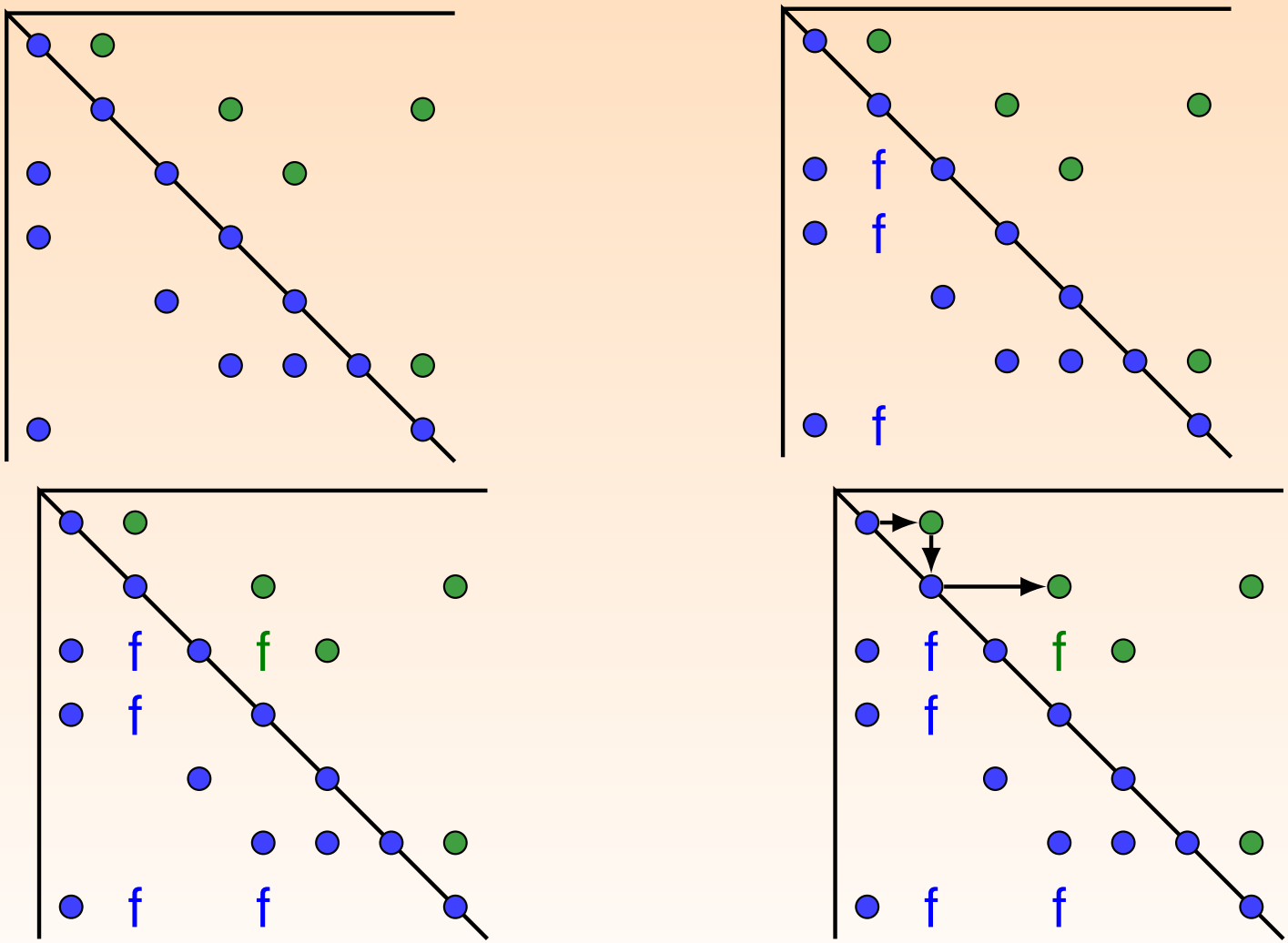


Column replication in LU: example





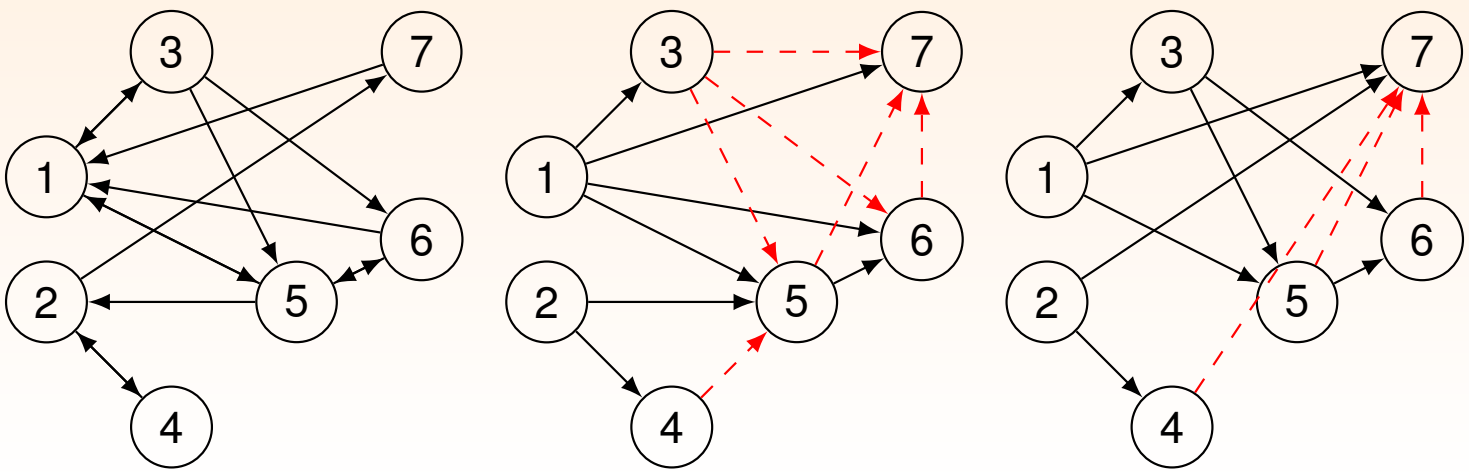
Column replication in LU: example



Sparse LU: replications: funny game to detect paths

- The path $1 \to 3 \to 5 \to 6$ in $\mathcal{G}(U)$. It implies the fill-in in L, first in column 3, then in columns 5 and 6.
- $2 \to 4 \to 5 \to 6$ in $\mathcal{G}(L^T) \Rightarrow$ fill-in at $(4,7)$, $(5,7)$ and $(6,7)$ in U.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} * & & * & & * & & \\ & * & & * & & & * \\ * & & * & & * & * & \\ & * & & * & & & \\ * & * & & & * & * & \\ * & & & & * & * & \\ * & & & & & & * \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} * & & * & & * & & \\ & * & & * & & & * \\ * & & * & & * & * & \\ & * & & * & & & f \\ * & * & f & f & * & * & f \\ * & & f & & * & * & f \\ * & & f & & f & f & * \end{pmatrix} \end{array}$$



Sparse LU: models and methods

Something other than bothering with paths needed

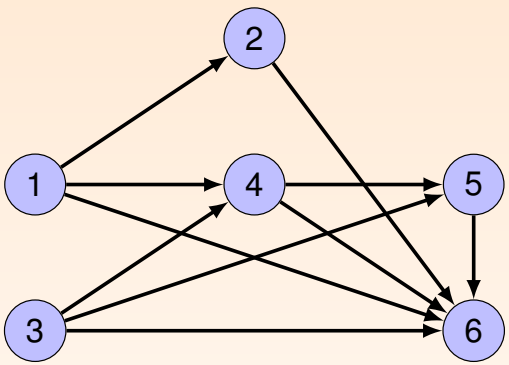
- To employ $\mathcal{G}(L^T)$ and $\mathcal{G}(U)$ in efficient algorithms, they need to be simplified.
- They must be sparser and preserve reachability (transitive reduction adds also the edge set minimality condition).
- Remind: the elimination tree $\mathcal{T}$ is a transitive reduction of $G(L^T)$.
- In LU, the analogy are transitive reductions $\mathcal{G}(L^T)$ and $\mathcal{G}(U)$.

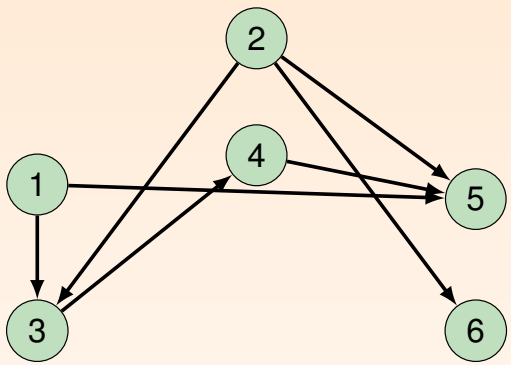
Sparse LU factorization of generally nonsymmetric matrices

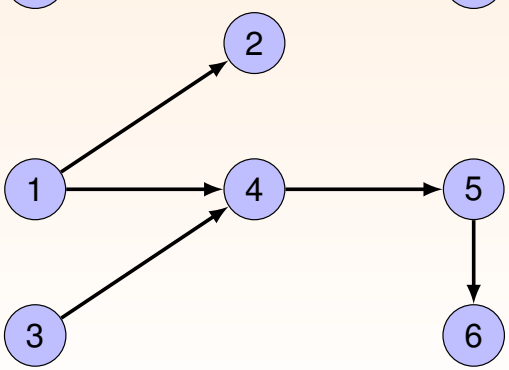
Transitive reductions of $\mathcal{G}(L^T)$ and $\mathcal{G}(U)$

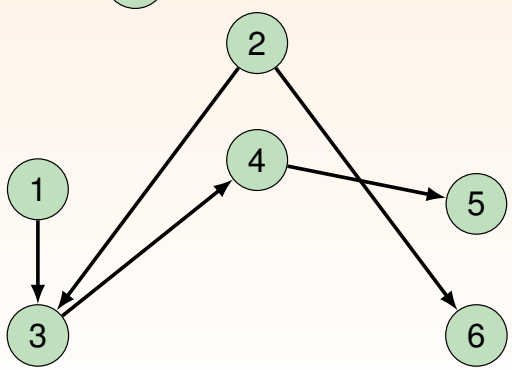
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} * & & * & & * & \\ * & * & & & * & * \\ & & * & * & & \\ * & & * & * & & \\ & & * & & * & \\ * & * & & & & * \end{pmatrix} \end{array}$$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} * & & * & & * & \\ * & * & f & & * & * \\ & & * & * & & \\ * & & * & * & f & \\ & & * & f & * & \\ * & * & f & f & f & * \end{pmatrix} \end{array}$$









Sparse LU: models and methods

Transitive reduction may be expensive to obtain

- Obtaining exact transitive reductions of $\mathcal{G}(L^T)$ and $\mathcal{G}(U)$ can be expensive on-the-fly due to the mutual dependency of the DAGs.

Sparse LU: models and methods

Transitive reduction may be expensive to obtain

- Obtaining exact transitive reductions of $\mathcal{G}(L^T)$ and $\mathcal{G}(U)$ can be expensive on-the-fly due to the mutual dependency of the DAGs.
- Instead, approximate reductions without the minimality condition may be computed. additional nonzeros do not make harm.
- We will call them equireachable DAGs (not fully transitively reduced DAGs).

Equireachability: example

Figure depicts $\mathcal{G}(U)$ and $\mathcal{G}'(U)$ for the matrix in Figure above.

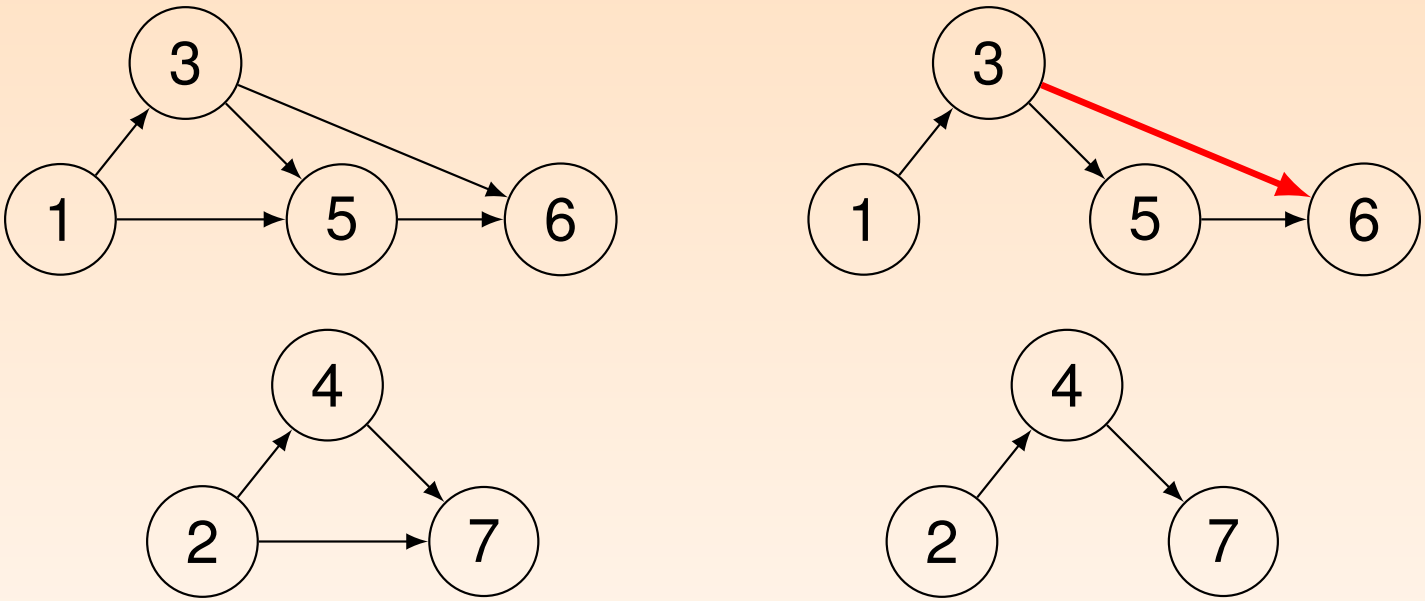


Figure: *The DAG $\mathcal{G}(U)$ (left) and $\mathcal{G}'(U)$ which is equireachable with $\mathcal{G}(U)$ (right).*

- The edge $(3, 6)$ is not in the transitive reduction.

Column sparsity patterns (for L)

- Schur complement description

$$\mathcal{S}\{L_{j:n,j}\} = \mathcal{S}\{A_{j:n,j}\} \bigcup_{k<j, u_{kj}\neq 0} \mathcal{S}\{L_{j:n,k}\}, \quad 1 \leq j \leq n.$$

- As in the symmetric case where the patterns are merged up $\mathcal{T}(A)$, not all the terms in this union are needed to get $\mathcal{S}\{L_{j:n,j}\}$. Theorem shows this merging formally:

Theorem

If $\mathcal{G}'(U)$ is equireachable with $\mathcal{G}(U)$ then

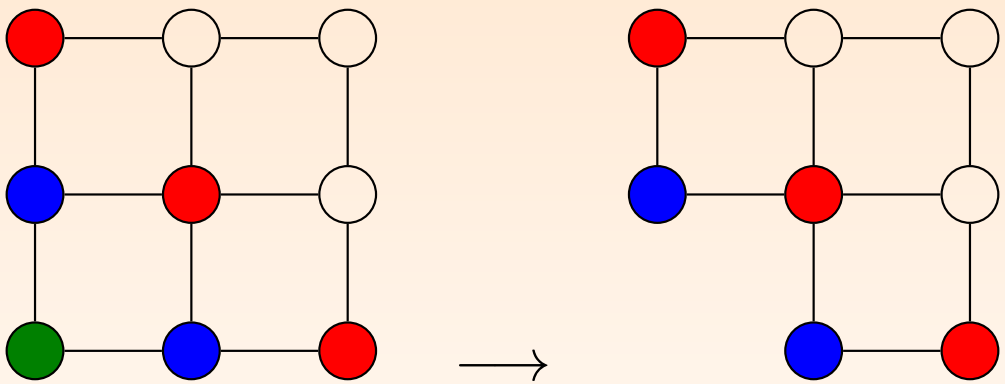
$$\mathcal{S}\{L_{j:n,j}\} = \mathcal{S}\{A_{j:n,j}\} \bigcup_{(k\to j)\in\mathcal{E}(\mathcal{G}'(U))} \mathcal{S}\{L_{j:n,k}\}, \quad 1 \leq j \leq n.$$

- Those in an equireachable graph are sufficient.
- Similarly for sparsity patterns of the rows of U.

Getting an equireachable DAG: pruning of the elimination DAGs

Theorem

If for some $j < s$ both $l_{sj} \neq 0$ and $u_{js} \neq 0$, then there are no edges $(j \to k)$ with $k > s$ in the transitive reductions of $\mathcal{G}(U)$ and $\mathcal{G}(L^T)$.



- Pruning in $\mathcal{G}(L^T)$: green and blue nodes represent edges.
- $l_{kj} \neq 0$ and $u_{js} \neq 0$ imply $l_{ks} \neq 0$: $k \to s \Rightarrow$ The green ones can be removed.

Another graph model: column elimination tree

- An attractive idea for constructing $\mathcal{S}\{L+U\}$ is based on using the column elimination tree $\mathcal{T}(A^TA)$.

Theorem

Assume all the diagonal entries of A are nonzero and let $\widehat{L}\widehat{L}^T$ be the Cholesky factorization of A^TA. Then for any row permutation matrix P such that $PA = LU$

$$\mathcal{S}\{L+U\} \subseteq \mathcal{S}\{\widehat{L}+\widehat{L}^T\}.$$

- Very strong result (theoretically).

The column elimination tree

- A potential problem with the column elimination tree is that:
- $\mathcal{S}\{A^T A\}$ can have significantly more nonzero entries than $\mathcal{S}\{L + U\}$.

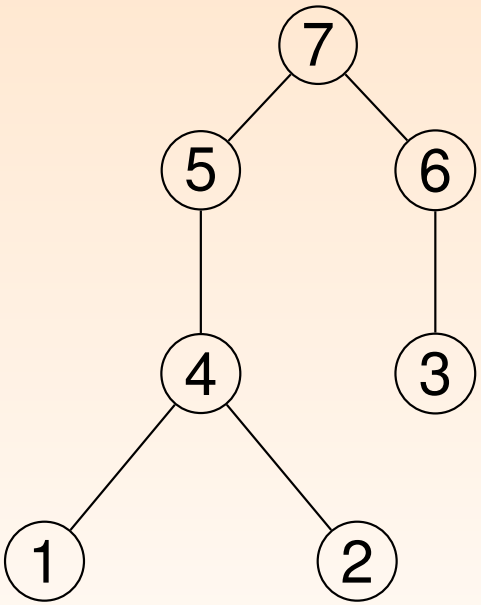
The column elimination tree

- A potential problem with the column elimination tree is that:
- $\mathcal{S}\{A^T A\}$ can have significantly more nonzero entries than $\mathcal{S}\{L+U\}$.
- An extreme example is when A has one or more dense rows because $A^T A$ is then fully dense.
- So, using it or not using it, it depends.

Column elimination tree: example

- Standard elimination tree $\mathcal{T}(A)$.

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{pmatrix}
* & & & * & & & \\
* & * & f & * & & & \\
 & & * & & & * & \\
* & * & & * & f & & * \\
 & * & * & f & * & * & * \\
 & & * & & & * & \\
* & & & * & * & f & *
\end{pmatrix}
\end{array}
$$



Sparse LU: models and methods

- The elimination tree $\mathcal{T}(A^T A)$: much more dependencies, much less parallelism.

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{pmatrix}
* & * & & * & * & & * \\
* & * & * & * & * & * & * \\
 & * & * & f & * & * & * \\
* & * & f & * & * & f & * \\
* & * & * & * & * & * & * \\
 & * & * & f & * & * & * \\
* & * & * & * & * & * & *
\end{pmatrix}
\end{array}
$$

Sparse LU: models and methods

Other related issues: similar to Cholesky

- We can define supernodes (supernodal structure in L and U). Some compatibility between the factors is needed. Not mentioning the danger of pivoting.
- We can use a modified multifrontal method
- Typically distinguishing A with a nearly symmetric pattern from other situations.
- Note that the factorizability is not generally guaranteed.

Sparse LU: preprocessing to get full transversal

Preprocessing for LU

- There exist preprocessing techniques that may alleviate problem of expensive LU.

I. Permuting nonzeros to the diagonal of A

- This can be achieved by a nonsymmetric permutation like $A \to AQ$
- Terminology: The set of the diagonal entries of A is called the transversal.
- If A is nonsingular (even structurally only) then it can be nonsymmetrically permuted to have the full transversal.

Sparse LU: preprocessing to get full transversal

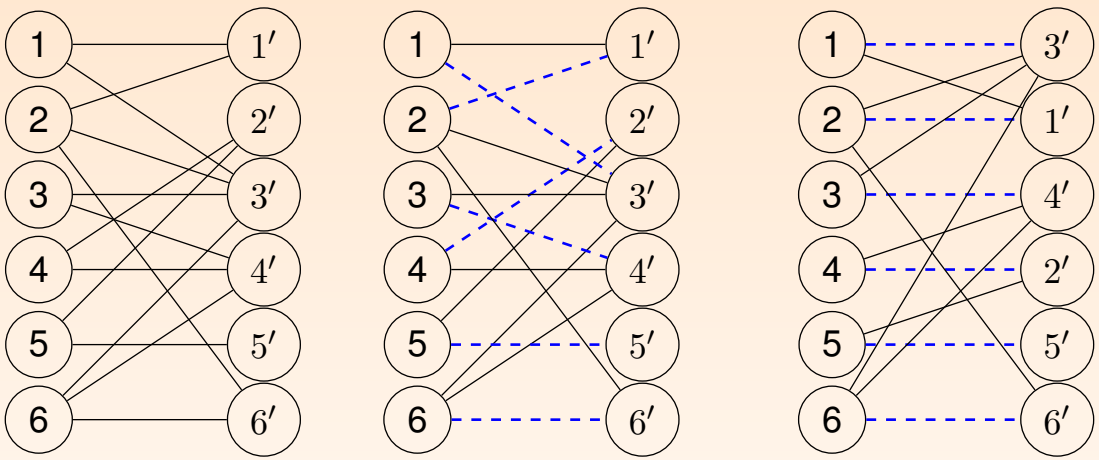
Preprocessing for LU: I. getting full transversal

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 1' & 2' & 3' & 4' & 5' & 6' \end{array} \\
\begin{pmatrix}
* & & * & & & \\
* & & * & & & * \\
 & & * & * & & \\
 & * & & * & & \\
 & * & & & * & \\
 & & * & * & & *
\end{pmatrix}
\end{array}
$$

Sparse LU: preprocessing to get full transversal

Preprocessing for LU: I. getting full transversal

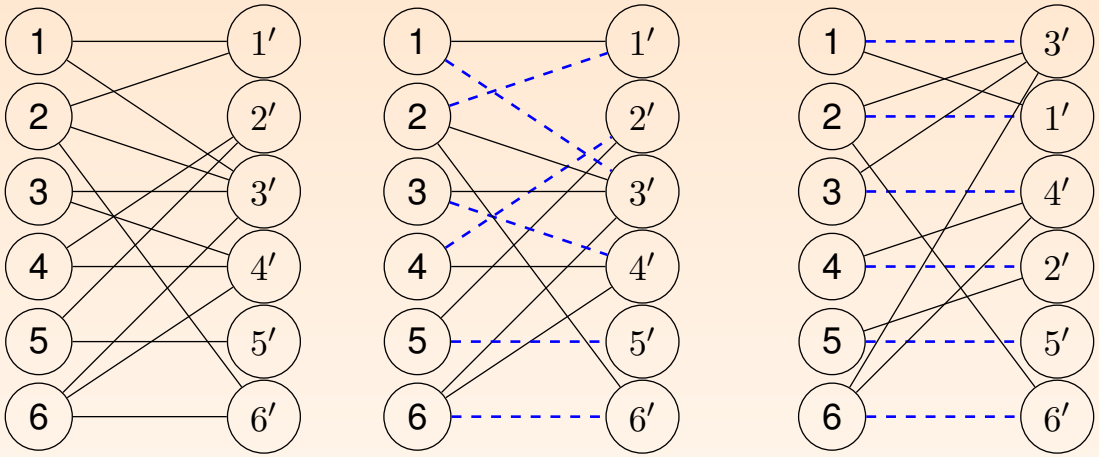
$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 1' & 2' & 3' & 4' & 5' & 6' \end{array} \\
\begin{pmatrix}
* & & * & & & \\
* & & * & & & * \\
 & & * & * & & \\
 & * & & * & & \\
 & * & & & * & \\
 & & * & * & & *
\end{pmatrix}
\end{array}
$$



Sparse LU: preprocessing to get full transversal

Preprocessing for LU: I. getting full transversal

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 1' & 2' & 3' & 4' & 5' & 6' \end{array} \\
\begin{pmatrix}
* & & * & & & \\
* & & * & & & * \\
 & & * & * & & \\
 & * & & * & & \\
 & * & & & * & \\
 & & * & * & & *
\end{pmatrix}
\end{array}
$$



- Just a column (or row) permutation is needed.
- An algorithm to be used: bipartite graph matching
- It can consider also sizes of nonzero values: still cheap.

Another preprocessing step: II. get a BTF shape

Preprocessing for LU: II. block triangular form

- When we can do this? If A is reducible.
- Remind that A is said to be reducible if there is a permutation matrix P such that

$$PAP^T = \begin{pmatrix} A_{p_1,p_1} & A_{p_1,p_2} \\ 0 & A_{p_2,p_2} \end{pmatrix},$$

where A_{p_1,p_1} and A_{p_2,p_2} are non trivial square matrices (that is, they are of order at least 1).

Another preprocessing step: II. get a BTF shape

Preprocessing for LU: II. block triangular form

- When we can do this? If A is reducible.
- Remind that A is said to be reducible if there is a permutation matrix P such that

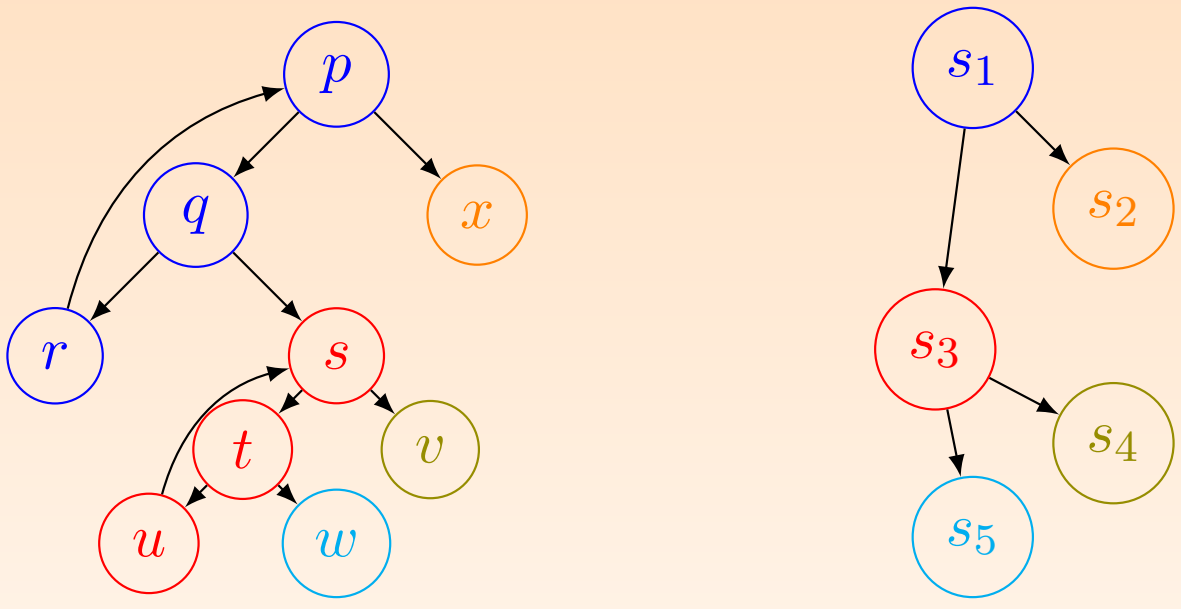
$$PAP^T = \begin{pmatrix} A_{p_1,p_1} & A_{p_1,p_2} \\ 0 & A_{p_2,p_2} \end{pmatrix},$$

  where A_{p_1,p_1} and A_{p_2,p_2} are non trivial square matrices (that is, they are of order at least 1).
- Why do we do this? To be more happy ☺:
  factorize only diagonal blocks $\Rightarrow$ do solves only with blocks.

Sparse LU: preprocessing to get BTF shape

Permutation to BTF: getting strong components

An example of five SCCs: $\{p, q, r\}$, $\{s, t, u\}$, $\{v\}$, $\{w\}$, $\{x\}$.



- Shrinking the strong components: DAG. And the DAG can be always ordered to provide a block upper triangular matrix (blocks correspond to the strong components)
- The transformation is a vertex relabelling. This is a symmetric permutation $A \rightarrow PAP^T$.

Outline

Stability and ill-conditioning

Backward stability and ill-conditioning: standard points

- Consider getting factors as $(L, U) = g(A)$. Two different notions.

Backward stability and ill-conditioning: standard points

- Consider getting factors as $(L, U) = g(A)$. Two different notions.
- Backward stable algorithm: the computed factors $(\hat{L}, \hat{U})$ are the exact solution of $(\hat{L}, \hat{U}) = g(A + \Delta A)$ and ΔA (the backward error) is "small" for all possible inputs A.

Stability and ill-conditioning

Backward stability and ill-conditioning: standard points

- Consider getting factors as $(L, U) = g(A)$. Two different notions.
- Backward stable algorithm: the computed factors $(\hat{L}, \hat{U})$ are the exact solution of $(\hat{L}, \hat{U}) = g(A + \Delta A)$ and ΔA (the backward error) is "small" for all possible inputs A.
- The problem $(L, U) = g(A)$ is ill-conditioned if small perturbations in A can lead to large changes in $(\hat{L}, \hat{U})$. The condition number then measures sensitivity of the output to the function input.

Stability and ill-conditioning

Backward stability and ill-conditioning: standard points

- Consider getting factors as $(L, U) = g(A)$. Two different notions.
- Backward stable algorithm: the computed factors $(\hat{L}, \hat{U})$ are the exact solution of $(\hat{L}, \hat{U}) = g(A + \Delta A)$ and ΔA (the backward error) is "small" for all possible inputs A.
- The problem $(L, U) = g(A)$ is ill-conditioned if small perturbations in A can lead to large changes in $(\hat{L}, \hat{U})$. The condition number then measures sensitivity of the output to the function input.

Observation

Backward stability is a property of the computational algorithm. To compute solutions with a small backward error we need to consider stable algorithms. Ill-conditioning is a property of input problem data. To suppress the ill-conditioning, we need to transform the problem (a priori or a posteriori)

Stability and ill-conditioning

Sidestep: using the inverse instead of factorization

- No stability results (in contrast to factorization and solve): The computed inverse is typically not the exact inverse of a nearby matrix $A + \Delta A$ for any small perturbation ΔA.
- Impractical to compute and store A^{-1}, regardless of how sparse A is: see below: the matrix sparsity strikes back.

Stability and ill-conditioning

Sidestep: using the inverse instead of factorization

- No stability results (in contrast to factorization and solve): The computed inverse is typically not the exact inverse of a nearby matrix $A + \Delta A$ for any small perturbation ΔA.
- Impractical to compute and store A^{-1}, regardless of how sparse A is: see below: the matrix sparsity strikes back.

Theorem

A irreducible $\Rightarrow$ the sparsity pattern $\mathcal{S}\{A^{-1}\}$ is fully dense.

- This is the reason why inverses of A are not much used.

Stability and ill-conditioning

Improving the backward stability (and forcing factorizability)

- At step k of LU, the computed $a_{kk}^{(k)}$ (pivot) $(1 \leq k < n)$ should be nonzero (to keep factorizability) and not of a small magnitude (to keep the growth in factors small).
- The growth can be measured by the growth factor:

$$\rho_{growth} = \max_{i,j,k} (|a_{ij}^{(k)}| \, / \, |a_{ij}|). \tag{4}$$

- Simple row interchanges: $A \to PA$ called partial pivoting ensures

$$|l_{ik}| \leq 1 \Longrightarrow \max_{i>k} |a_{ik}^{(k)}| \leq |a_{kk}^{(k)}|.$$

- Partial pivoting may not be sufficient. Complete pivoting is better, but it has much smaller potential for parallelization.

Stability and ill-conditioning

Pivoting possibilities

- Partial pivoting:
$$\rho_{growth} \leq 2^{n-1}.$$
- Complete pivoting choosing the pivot as an entry of the largest magnitude in the Schur complement.
$$\rho_{growth} \leq n^{1/2}(2.\, 3^{1/2}.\, 4^{1/3} \ldots n^{1/(n-1)})^{1/2}.$$
- Rook pivoting: the largest magnitude in its row *and* its column:
$$\rho_{growth} \leq 1.5\, n^{(3/4)\log n}.$$
- Taking sparsity into account: threshold partial pivoting
$$\max_{i>k} |a_{ik}^{(k)}| \leq \gamma^{-1} |a_{kk}^{(k)}|,$$
where $\gamma \in (0, 1]$ is a chosen threshold parameter.
- Even complete pivoting can be mixed with sparsity considerations: Markowitz pivoting

Outline

Symmetric indefinite matrix: example

- Consider
$$A = \begin{pmatrix} \delta & 1 \\ 1 & 0 \end{pmatrix}.$$
- $\delta = 0 \Rightarrow$ LDLT with D diagonal does not exist.
- $\delta \ll 1 \Rightarrow$ LDLT with D diagonal is not stable since $\rho_{growth} = 1/\delta$.
- LDLT factorization generalized to allow D with 1×1 and $2 \times 2 \Rightarrow$ blocks. It preserves symmetry and is nearly as stable as the LU factorization.

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = LDL^T.$$

Here D has one 1×1 block and one 2×2 block.

Stability and ill-conditioning

Symmetric indefinite: balancing 1×1 pivots and 2×2 pivots

- Small growth for 1×1 pivot if $|a_{kk}|$ (a diagonal entry) is large.
- If such pivot not found, consider large off-diagonals
- Consider the inverse of the 2×2 block

$$\begin{pmatrix} a & b \\ b & d \end{pmatrix}^{-1} = \frac{1}{ad - b^2} \begin{pmatrix} d & -b \\ -b & a \end{pmatrix}$$

- $\Rightarrow$ if $|a|, |d|$ small with respect to $|b|$, 2×2 pivot may be used.
- The standard rule balancing the pivots: based on requiring the same potential maximal growth in a 2×2 pivot versus two consecutive 1×1 pivots.
- This implies an appropriate parameter $(1 + \sqrt{17})/8$ to choose between the pivots (see the next slide)
- $$\rho_{growth} < 3n\sqrt{2\, 3^{1/2} 4^{1/3} \ldots n^{1/(n-1)}},$$

Indefinite factorization: full pivoting

Algorithm (One step of full indefinite pivoting)

Set $\alpha = (1+\sqrt{17})/8 \approx 0.64$
Find a_{kk}*: diagonal entry of maximum size*
Find a_{ij}*: off-diagonal entry of maximum size* $(i < j)$
if $|a_{kk}| \geq \alpha |a_{ij}|$ ***then***
    use a_{kk} *as* 1×1 *pivot (ready for* $a_{kk} = 0$*)*
else
    use $\begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix}$ *as* 2×2 *the pivot*
end if

- But sparsity must be also considered!

Stability and ill-conditioning

Indefinite factorization: classical scheme of symmetric partial pivoting

- The following scheme shows entries sufficient to be checked

$$\begin{pmatrix} d & . & . & \lambda & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ \lambda & . & . & c & . & \sigma & . \\ . & . & . & . & . & . & . \\ . & . & . & \sigma & . & . & . \\ . & . & . & . & . & . & . \end{pmatrix}$$

- λ, σ: maximum absolute value in its row and column, respectively.
- That is: only two rows and columns of A searched.
- Less searches: slightly larger growth factor bound than in LU
- There are stable schemes and threshold extensions.

Solving ill-conditioned problems

- a) Preprocessing by diagonal scaling:

$$S_r\, A\, S_c\, y = S_r\, b, \qquad y = S_c^{-1}\, x.$$

Theorem

Let the matrix A be SPD and let D_A be the diagonal matrix with entries a_{ii} $(1 \le i \le n)$. Then for all diagonal matrices D with positive entries

$$\kappa(D_A^{-1/2}\, A\, D_A^{-1/2}) \le nz_{rmax}\, \kappa(D^{-1/2}\, A\, D^{-1/2}),$$

where nz_{rmax} is the maximum number of entries in a row of A.

Stability and ill-conditioning

Solving ill-conditioned problems

- a) Preprocessing by diagonal scaling:

$$S_r\, A\, S_c\, y = S_r\, b, \qquad y = S_c^{-1}\, x.$$

Theorem

Let the matrix A be SPD and let D_A be the diagonal matrix with entries a_{ii} $(1 \le i \le n)$. Then for all diagonal matrices D with positive entries

$$\kappa(D_A^{-1/2}\, A\, D_A^{-1/2}) \le nz_{rmax}\, \kappa(D^{-1/2}\, A\, D^{-1/2}),$$

where nz_{rmax} is the maximum number of entries in a row of A.

- b) postprocessing: various iterative refinements (IR) like

Algorithm (IR of the solution x of $Ax = b$)

1: *Solve* $Ax^{(0)} = b$ ▷ $x^{(0)}$ *is the initial computed solution*
2: ***for*** $k = 0, 1, \ldots$ ***do***
3: *Compute* $r^{(k)} = b - Ax^{(k)}$ ▷ *Residual on iteration* k
4: *Solve* $A\,\delta x^{(k)} = r^{(k)}$ ▷ *Solve correction equation: using a factorization*
5: $x^{(k+1)} = x^{(k)} + \delta x^{(k)}$
6: ***end for***

Outline

Least squares: factorizations

- Direct methods are relevant even for LS: regular LS: normal equations: Cholesky
- Another formulation for LS:
- The normal equations are equivalent to the linear equations $A^T r = 0$, and $r = b - Ax$ that can be expressed as the $(m+n) \times (m+n)$ augmented system ($z = r$ and $c = 0$).

$$K \begin{pmatrix} z \\ x \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix} \quad \text{with} \quad K = \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix},$$

Least squares: factorizations

- Direct methods are relevant even for LS: regular LS: normal equations: Cholesky
- Another formulation for LS:
- The normal equations are equivalent to the linear equations $A^T r = 0$, and $r = b - Ax$ that can be expressed as the $(m+n) \times (m+n)$ augmented system ($z = r$ and $c = 0$).

$$K \begin{pmatrix} z \\ x \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix} \quad \text{with} \quad K = \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix},$$

- The symmetric indefinite matrix K is non singular if and only if $rank(A) = n$: general indefinite solvers

Least squares: factorizations

- Direct methods are relevant even for LS: regular LS: normal equations: Cholesky
- Another formulation for LS:
- The normal equations are equivalent to the linear equations $A^T r = 0$, and $r = b - Ax$ that can be expressed as the $(m+n) \times (m+n)$ augmented system ($z = r$ and $c = 0$).

$$K \begin{pmatrix} z \\ x \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix} \quad \text{with} \quad K = \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix},$$

- The symmetric indefinite matrix K is non singular if and only if $rank(A) = n$: general indefinite solvers
- To be more general, consider the regularized LS

$$\min_{x \in \mathbb{R}^n} \left(\|b - Ax\|_2^2 + \gamma^2 \|x\|_2^2 \right) = \min_{x \in \mathbb{R}^n} \left\| \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} A \\ \gamma I \end{pmatrix} x \right\|_2.$$

Sparse Least Squares and QR factorization

Least squares: two solution approaches so far

- 1. SPD (Cholesky) factorization of $A^T A$
- If $\gamma > \sigma_{min}(A)$, we have $\kappa(A^T A + \gamma^2 I) \approx (\|A\|_2/\gamma)^2$
- Not a big progress since γ should be kept small.

Sparse Least Squares and QR factorization

Least squares: two solution approaches so far

- 1. SPD (Cholesky) factorization of $A^T A$
- If $\gamma > \sigma_{min}(A)$, we have $\kappa(A^T A + \gamma^2 I) \approx (\|A\|_2/\gamma)^2$
- Not a big progress since γ should be kept small.

- 2. Symmetric indefinite factorization of K

$$\begin{pmatrix} I & A \\ A^T & -\gamma^2 I \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} \text{ or } K_\gamma \begin{pmatrix} s \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}, \; K_\gamma = \begin{pmatrix} \gamma I & A \\ A^T & -\gamma I \end{pmatrix}, \; r = \gamma s.$$

- If $\gamma > \sigma_{min}(A)$, we have $\kappa(K_\gamma) \approx \|A\|_2/\gamma$.
- Seems to be better, but indefiniteness.

Least squares: an additional solution approach

- Another solution strategy: using another (QR) factorization.
- $$A = (Q_1 \;\; Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R,$$
- $Q = (Q_1 \;\; Q_2)$ is orthogonal, $R \in \mathbb{R}^{m \times n}$ is upper triangular.

Sparse Least Squares and QR factorization

Least squares: an additional solution approach

- Another solution strategy: using another (QR) factorization.
- $$A = (Q_1 \;\; Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R,$$
- $Q = (Q_1 \;\; Q_2)$ is orthogonal, $R \in \mathbb{R}^{m \times n}$ is upper triangular.
- There are more ways to orthogonalize A
  - Givens rotations
  - Householder reflections
  - Gram-Schmidt orthogonalization
- All of them should get the same Q (modulo signs of the diagonal entries of R)
- Let us proceed to a sparse A, to see that the fill-in can be overestimated.

Contemporary sparse QR: symbolic phase

- Consider a symbolic phase predicting R or Q.
- A Givens rotation $G(i,j)$ applied to $A_{i,i:n}$ and $A_{j,i:n}$ of A:

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} A_{i,i:n} \\ A_{j,i:n} \end{pmatrix} = \begin{pmatrix} A'_{i,i:n} \\ A'_{j,i:n} \end{pmatrix}, \; A'_{j,i} = 0.$$

- An example that emphasizes sparsity patterns:

$$\begin{pmatrix} A_{i,i:n} \\ A_{j,i:n} \end{pmatrix} = \begin{pmatrix} * & * & * & & * & & * \\ * & & & * & & & \end{pmatrix}.$$

Applying $G(i,j)$ gives

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} A_{i,i:n} \\ A_{j,i:n} \end{pmatrix} = \begin{pmatrix} * & * & * & * & * & & * \\ & * & * & * & * & & * \end{pmatrix} = \begin{pmatrix} A'_{i,i:n} \\ A'_{j,i:n} \end{pmatrix}.$$

Sparse Least Squares and QR factorization

Contemporary sparse QR: symbolic phase

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} A_{i,i:n} \\ A_{j,i:n} \end{pmatrix} = \begin{pmatrix} * & * & * & * & * & * \\ & * & * & * & * & * \end{pmatrix} = \begin{pmatrix} A'_{i,i:n} \\ A'_{j,i:n} \end{pmatrix}.$$

- The $(1,1)$ entry $A'_{i,i}$ seems to remain nonzero (it is the Euclidean norm of the vector $(A_{ii}\ A_{ji})^T$) and the sparsity patterns of columns 2 to n satisfy

$$\mathcal{S}(A'_{i,i+1:n}) = \mathcal{S}(A_{i,i+1:n}) \cup \mathcal{S}(A_{j,i+1:n}), 1 \leq i \leq n-1.$$

- This is the row merge rule. Apparently, significantly more fill-in than in LU/LL^T

Contemporary sparse QR: symbolic phase

- However, the fill-in can be overestimated. Consider $a, b \neq 0$

$$\begin{pmatrix} * & a & b & & \\ * & & & * & * \\ * & & & * & * \end{pmatrix} \rightarrow \begin{pmatrix} * & c'ca & c'cb & & \\ & sa & sb & * & * \\ & s'ca & s'cb & * & * \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} * & a & b & & \\ & c''sa - s''s'ca & c''sb - s''s'cb & * & * \\ & s''sa + c''s'ca & s''sb + c''s'cb & * & * \end{pmatrix}.$$

- Steps: apply $G(2,1)$ with c, s to eliminate the $(2,1)$ entry;apply $G(3,1)$ with c', s' to eliminate the $(3,1)$ entry; eliminate the fill-in at $(3,2)$ by rotation with c'', s''.
- We have $s''sa + c''s'ca = 0$.
- But this a nonzero multiple of the entry $s''sb + c''s'cb$ at $(3,3)$.
- The row merge rule is not able to predict that the $(3,3)$ entry also always becomes zero.

Contemporary sparse QR: another possibility for symbolic QR

Lemma

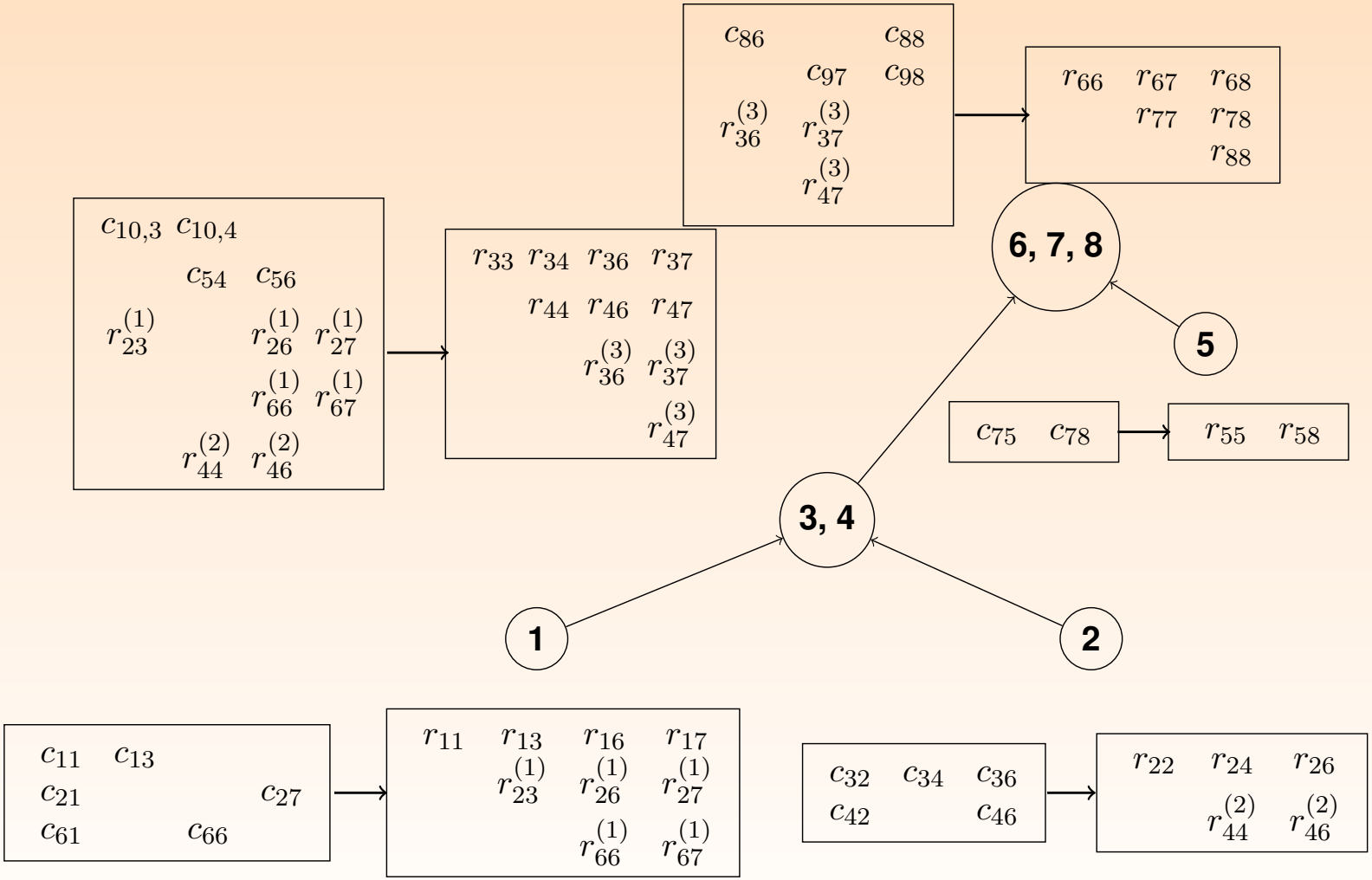
$\mathcal{S}(R) \subseteq \{$prediction of $\mathcal{S}(R)$ based on row merge rule $\} \subseteq \{$prediction of $\mathcal{S}(R)$ based on $A^TA\}$.

- This surprising behavior can be suppressed by considering structural properties of A, this is not a problem.
- But still, the QR may not be a progress, structurally. To feel this, consider

$$A = QR \Rightarrow A^TA = R^TQ^TQR = R^TR$$

- And we have Cholesky of A^TA. See our concerns above.
- As for the Lemma, a practical sparse QR solver may be based on the pattern of A^TA.
- For example, the multifrontal method that uses $C = A^TA$ implicitly.

Contemporary sparse QR: multifrontal QR factorization



Outline

Minimizing the fill-in: reorderings

- Key problem of factorizations: minimizing the fill-in. Remind:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} * & * & * & * & * \\ * & * & & & \\ * & & * & & \\ * & & & * & \\ * & & & & * \end{pmatrix} \end{array} \quad \rightarrow \quad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} * & & & & * \\ & * & & & * \\ & & * & & * \\ & & & * & * \\ * & * & * & * & * \end{pmatrix} \end{array}$$

- Our tools: symmetric permutations: $A \rightarrow PAP^T$
- Finding a permutation minimizing fill-in is NP complete: heuristics called fill-reducing orderings.
- No stability concerns: only sparsity pattern $\mathcal{S}\{A\}$ needed
- Otherwise: further permutations of A to force factorizability needed.

Reorderings

A. Local (greedy) reorderings

- Two basic greedy heuristics are the minimum degree (MD) criterion and the minimum fill (MF) criteria.

A.I. Minimum fill-in (MF) criterion

- Select as the next vertex of $\mathcal{G}(A)$ the one that introduce the least fill-in in $\mathcal{G}^k$. Or do it approximatively (AMF).
- High quality, but the cost of MF can be prohibitive: needed to check neighbors of neighbors.

A.II. Minimum degree (MD) criterion

- Select as the next vertex a vertex of minimum degree in $\mathcal{G}^k$. Or do it approximatively (AMD).
- MD is the most widely-used local heuristic. Less expensive than MF.

A. Local (greedy) reorderings: MD algorithm example

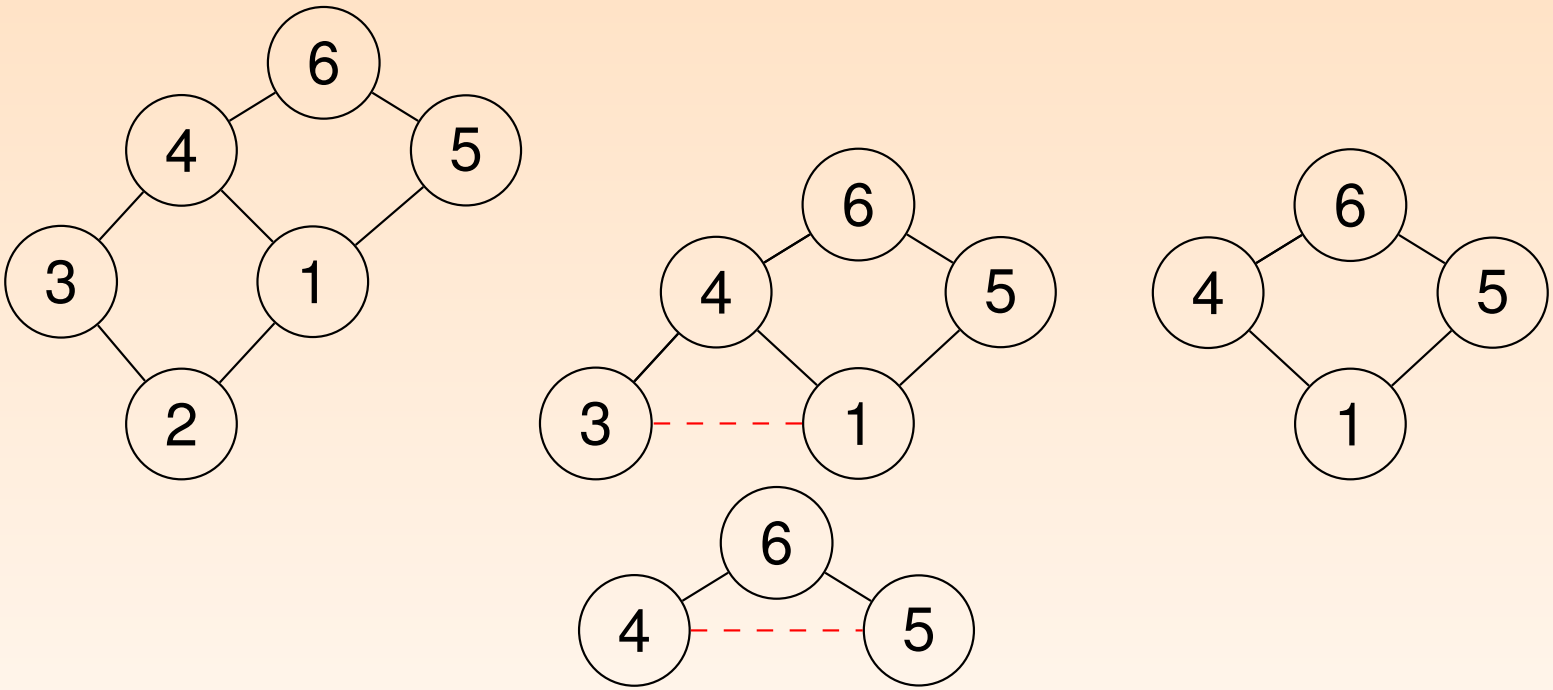
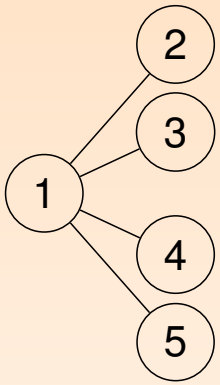


Figure: *An illustration of three steps of the MD algorithm. Elimination order:* $\mathcal{G}^2$, $\mathcal{G}^3$ *and* $\mathcal{G}^4$.

Reorderings

A. Local reorderings: storing and using the fill-in

- A clique with m vertices has $m(m-1)/2$ edges. This cannot be stored explicitly in the initial space!. $\mathcal{G}_k$ must be stored implicitly.



- 4 vertices instead of 6 edges if the clique stored implicitly
- The cliques stored as lists of neighbors. As the elimination process progresses, cliques grow and can be merged.
- If vertices not merged (as blocks) $\Rightarrow \mathcal{E}_k$ (changed according to the Parter's rule) expressed as reachable sets in modified elimination graphs.

A. Local reorderings: From Parter's rule to reachable sets

- Figure: graph $\mathcal{G}(A)$. The adjacency sets of the vertices in $\mathcal{G}^4$ that result from eliminating vertices $\mathcal{V}^4 = \{1, 2, 3\}$ are
  $adj_{\mathcal{G}^4}\{4\} = \mathcal{R}each(4, \mathcal{V}^4) = \{5\}$,
  $adj_{\mathcal{G}^4}\{5\} = \mathcal{R}each(5, \mathcal{V}^4) = \{4, 6, 7\}$,
  $adj_{\mathcal{G}^4}\{6\} = \mathcal{R}each(6, \mathcal{V}^4) = \{5, 7\}$,
  $adj_{\mathcal{G}^4}\{7\} = \mathcal{R}each(7, \mathcal{V}^4) = \{5, 6, 8\}$,
  $adj_{\mathcal{G}^4}\{8\} = \mathcal{R}each(8, \mathcal{V}^4) = \{7\}$.

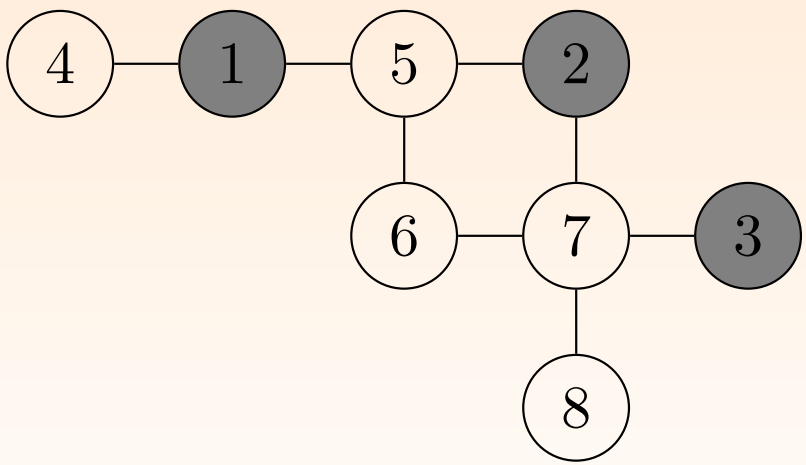


Figure: *1, 2, and 3 eliminated in the first three elimination steps* ($\mathcal{V}^4 = \{1, 2, 3\}$).

Reorderings

A. Local reorderings: tricks

- The construction of $\mathcal{G}^{k+1}$ needs some tricks to make it cheaper.
- Replication $\Rightarrow$ accumulation of information:
  finding and exploiting analogies to the supernodes needed
- In fact, we must find supernodes without the efficient tools like the elimination tree.

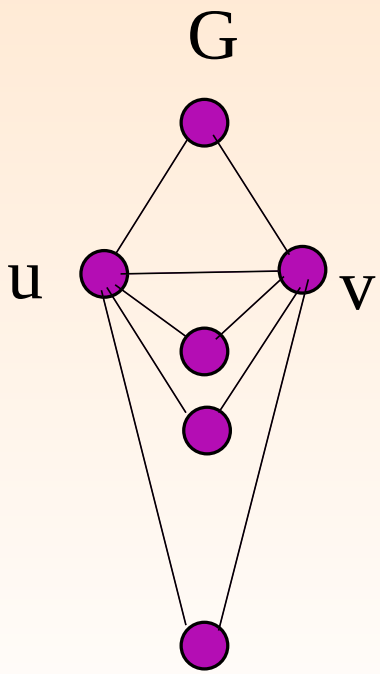
Reorderings

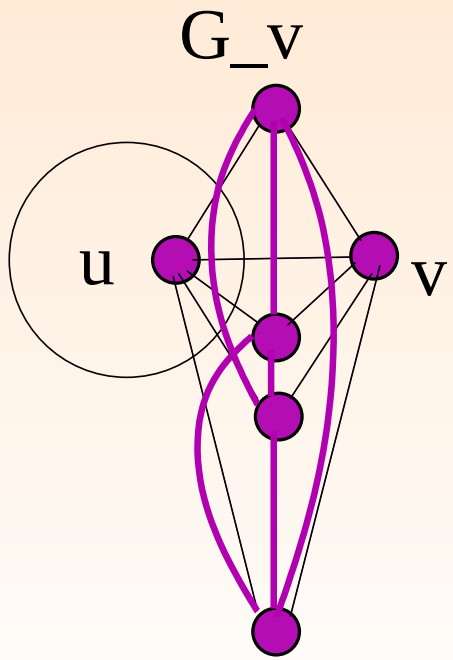
A. The first acceleration trick: indistinuishability

Definition

Two different vertices **u** and **v** of G are called indistinguishable if

$$Adj_G(u) \cup \{u\} = Adj_G(v) \cup \{v\}. \tag{5}$$





- Correspond to supernodes: can be eliminated in any mutual order.

A. Second acceleration trick: degree outmatching

- Vertex w is said to be outmatched by vertex u if

$$adj_{\mathcal{G}}\{u\} \cup \{u\} \subseteq adj_{\mathcal{G}}\{w\} \cup \{w\}.$$

- It follows: $deg_{\mathcal{G}}(u) \leq deg_{\mathcal{G}}(w)$, preserved in $\mathcal{G}_v$ for $v, v \neq u, w$

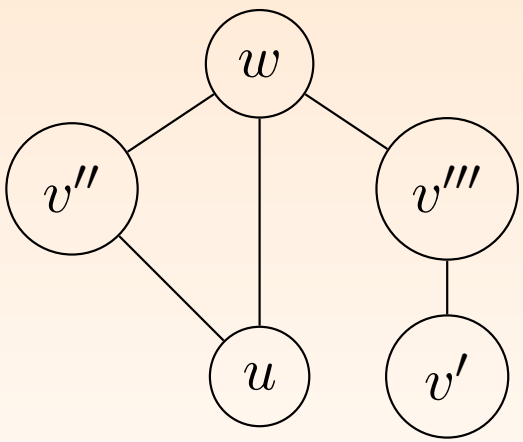
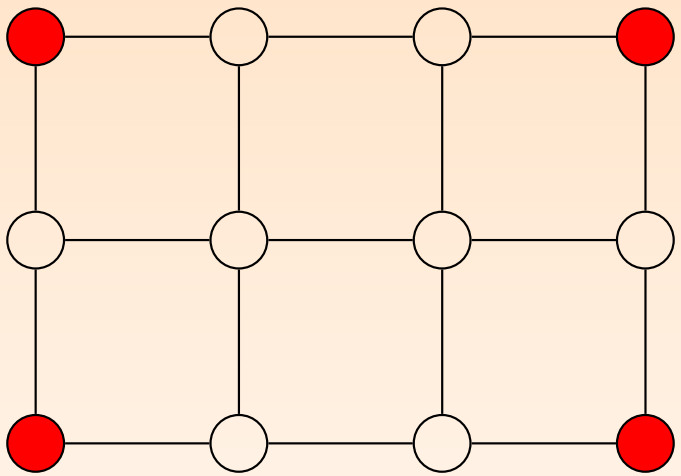


Figure: *An example $\mathcal{G}$ in which vertex w is outmatched by vertex u.*

A. Third acceleration trick: Multiple minimum degree (MMD)



- The mutually non-adjacent can be eliminated at the same time.

A. Local reorderings: complexity

- The complexity of the MD algorithms is $O(nz(A)n^2)$.
- The tricks do not change the worst-case bound.
- Additional trick: limit the search length in reachable sets: AMD (approximate minimum degree).
- The complexity of AMD is $O(nz(A)n)$.
- In practice, runtime of AMD is typically significantly smaller than that of the MD and MMD approaches.

Reorderings

B. Global (nested dissection) orderings

- Identify a small subset of vertices: vertex separator

Definition

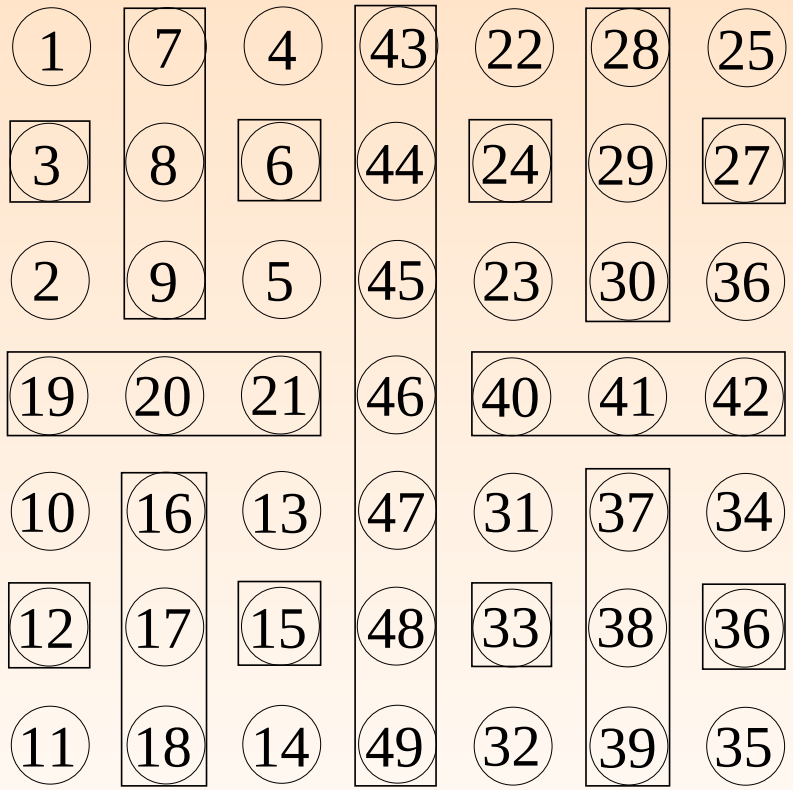
Vertex separator of an undirected $G = (V, E)$ is subset S of its vertices such that the subgraph induced by $V \setminus S$ has more components than G.

- Order it last, then the separated parts.
  - Induced reordering

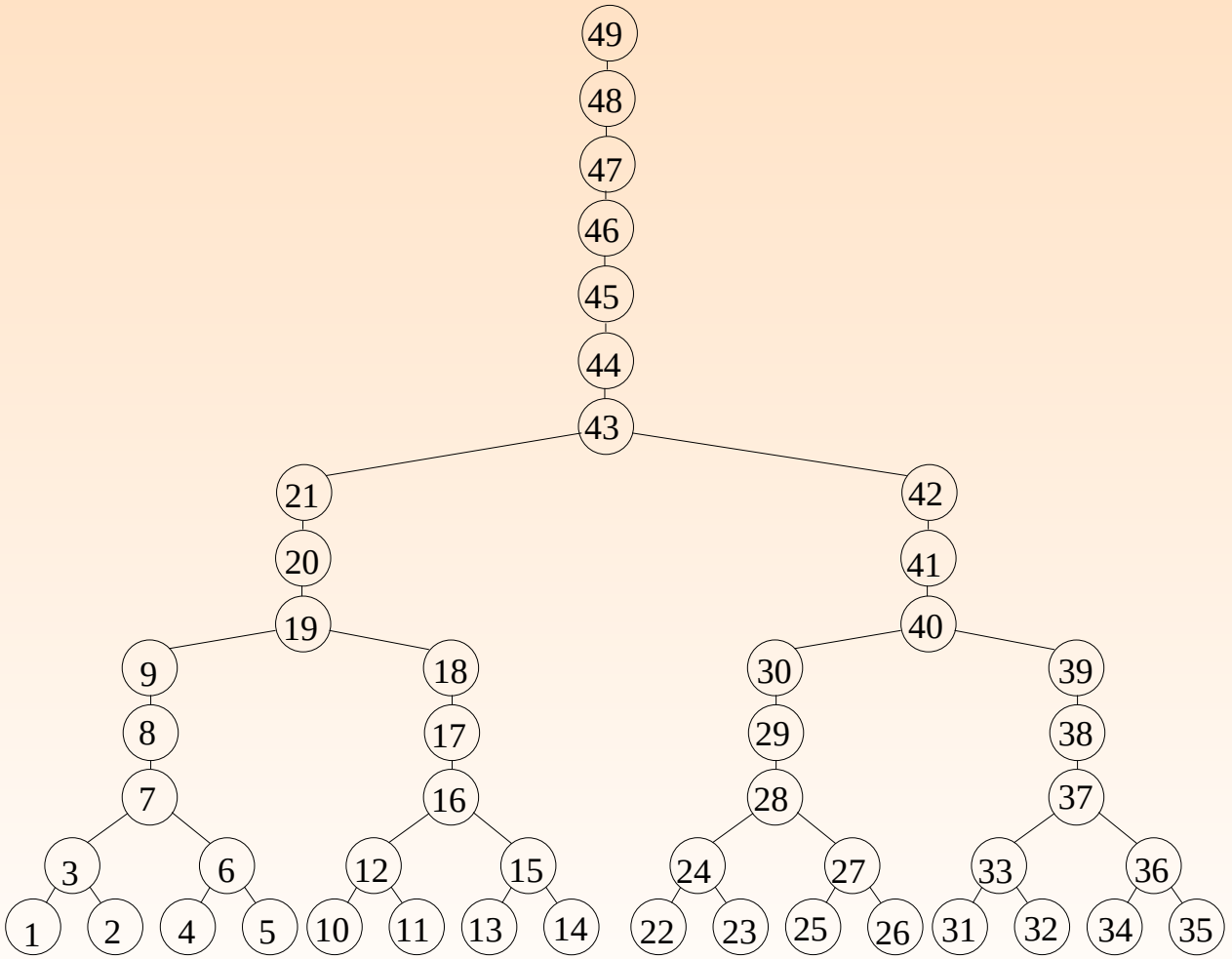
$$A = \begin{pmatrix} A_{11} & 0 & A_{31}^T \\ 0 & A_{22} & A_{32}^T \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \tag{6}$$

- Do it recursively

B. Global (nested dissection) orderings

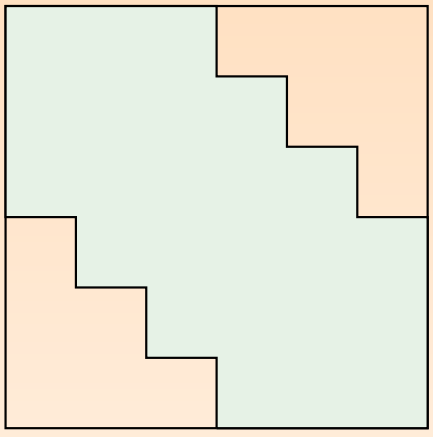


B. Global (nested dissection) orderings

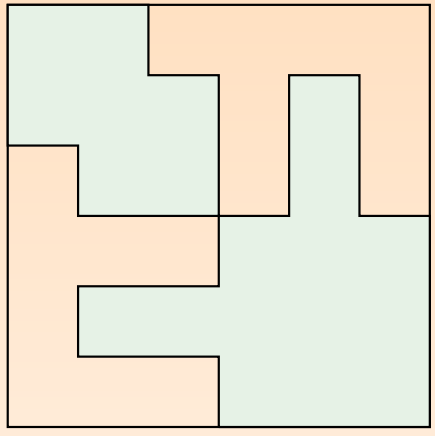


Reorderings

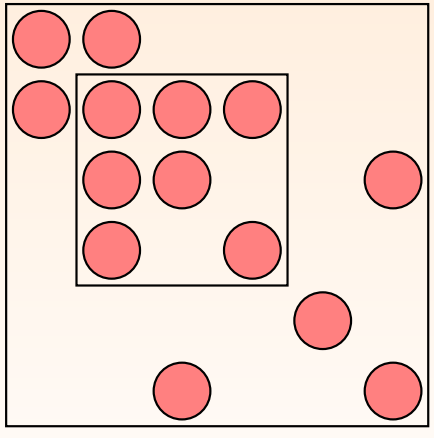
C. Minimizing fill-in + getting a favourable shape



Band



Profile (Envelope)



Frontal method : moving window determines ordering

Reorderings

C. Minimizing fill-in + getting a favourable shape

- Why do we do this?
- Static structures! Motivated by the following theorem:

Theorem

If L is the Cholesky factor of A then

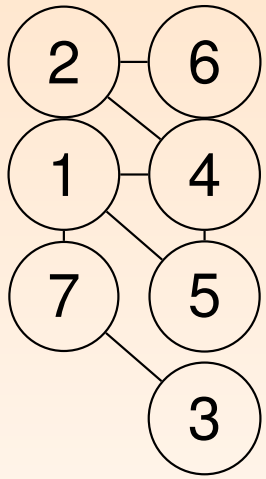
$$envelope(A) = envelope(L),\ band(A) = band(L).$$

- How to get such shape? Finding a permutation!
  - In advance: band, profile (envelope) methods
  - On-the fly: frontal method

Reorderings

C. Minimizing fill-in + getting a favourable shape

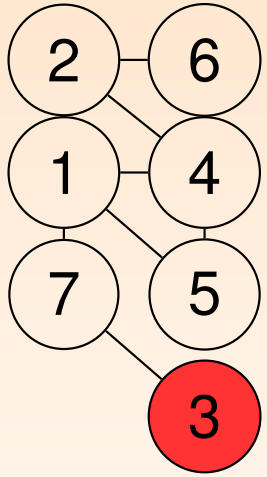
- Getting the permutation in advance: a modified breadth-first search called CM and RCM (CM plus reversing the permutation). Both: the same bandwidth, RCM can decrease the envelope.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} * & * & & * & * & & * \\ * & * & & * & & * & \\ & & * & & & & * \\ * & * & & * & * & & \\ * & & & * & * & & \\ & * & & & & * & \\ * & & * & & & & * \end{pmatrix} \end{array}, \quad \begin{array}{c} \\ 3 \\ 7 \\ 1 \\ 5 \\ 2 \\ 4 \\ 6 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 3 & 7 & 1 & 5 & 2 & 4 & 6 \end{array} \\ \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & * & * & \\ & & * & * & & * & \\ & & * & & * & * & * \\ & & * & * & * & * & \\ & & & & * & & * \end{pmatrix} \end{array}, \quad \begin{array}{c} \\ 6 \\ 4 \\ 2 \\ 5 \\ 1 \\ 7 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 6 & 4 & 2 & 5 & 1 & 7 & 3 \end{array} \\ \begin{pmatrix} * & & * & & & & \\ & * & * & * & * & & \\ * & * & * & & * & & \\ & * & & * & * & & \\ & * & * & * & * & * & \\ & & & & * & * & * \\ & & & & & * & * \end{pmatrix} \end{array},$$

C. Minimizing fill-in + getting a favourable shape

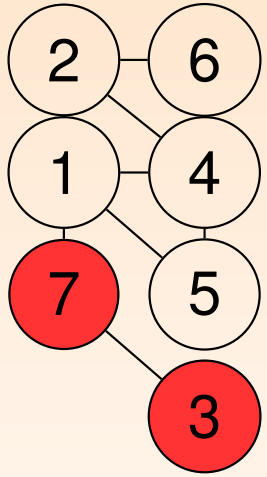
- Getting the permutation in advance: a modified breadth-first search called CM and RCM (CM plus reversing the permutation). Both: the same bandwidth, RCM can decrease the envelope.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} * & * & & * & * & & * \\ * & * & & * & & * & \\ & & * & & & & * \\ * & * & & * & * & & \\ * & & & * & * & & \\ & * & & & & * & \\ * & & * & & & & * \end{pmatrix} \end{array}, \quad \begin{array}{c} \\ 3 \\ 7 \\ 1 \\ 5 \\ 2 \\ 4 \\ 6 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 3 & 7 & 1 & 5 & 2 & 4 & 6 \end{array} \\ \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & * & * & \\ & & * & * & & * & \\ & & * & & * & * & * \\ & & * & * & * & * & \\ & & & & * & & * \end{pmatrix} \end{array}, \quad \begin{array}{c} \\ 6 \\ 4 \\ 2 \\ 5 \\ 1 \\ 7 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 6 & 4 & 2 & 5 & 1 & 7 & 3 \end{array} \\ \begin{pmatrix} * & & * & & & & \\ & * & * & * & * & & \\ * & * & * & & * & & \\ & * & & * & * & & \\ & * & * & * & * & * & \\ & & & & * & * & * \\ & & & & & * & * \end{pmatrix} \end{array},$$

C. Minimizing fill-in + getting a favourable shape

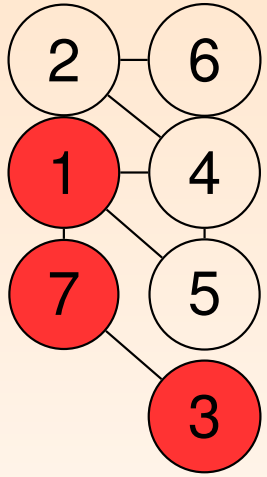
- Getting the permutation in advance: a modified breadth-first search called CM and RCM (CM plus reversing the permutation). Both: the same bandwidth, RCM can decrease the envelope.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} * & * & & * & * & & * \\ * & * & & * & & * & \\ & & * & & & & * \\ * & * & & * & * & & \\ * & & & * & * & & \\ & * & & & & * & \\ * & & * & & & & * \end{pmatrix} \end{array}, \quad \begin{array}{c} \\ 3 \\ 7 \\ 1 \\ 5 \\ 2 \\ 4 \\ 6 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 3 & 7 & 1 & 5 & 2 & 4 & 6 \end{array} \\ \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & * & * & \\ & & * & * & & * & \\ & & * & & * & * & * \\ & & * & * & * & * & \\ & & & & * & & * \end{pmatrix} \end{array}, \quad \begin{array}{c} \\ 6 \\ 4 \\ 2 \\ 5 \\ 1 \\ 7 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 6 & 4 & 2 & 5 & 1 & 7 & 3 \end{array} \\ \begin{pmatrix} * & & * & & & & \\ & * & * & * & * & & \\ * & * & * & & * & & \\ & * & & * & * & & \\ & * & * & * & * & * & \\ & & & & * & * & * \\ & & & & & * & * \end{pmatrix} \end{array},$$

C. Minimizing fill-in + getting a favourable shape

- Getting the permutation in advance: a modified breadth-first search called CM and RCM (CM plus reversing the permutation). Both: the same bandwidth, RCM can decrease the envelope.

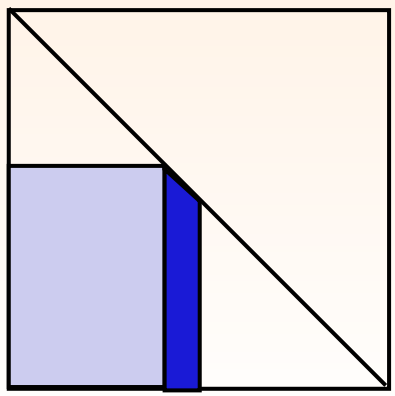


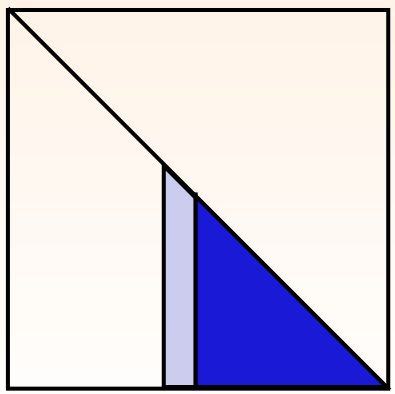
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} * & * & & * & * & & * \\ * & * & & * & & * & \\ & & * & & & & * \\ * & * & & * & * & & \\ * & & & * & * & & \\ & * & & & & * & \\ * & & * & & & & * \end{pmatrix} \end{array}, \quad \begin{array}{c} \\ 3 \\ 7 \\ 1 \\ 5 \\ 2 \\ 4 \\ 6 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 3 & 7 & 1 & 5 & 2 & 4 & 6 \end{array} \\ \begin{pmatrix} * & * & & & & & \\ * & * & * & & & & \\ & * & * & * & * & * & \\ & & * & * & & * & \\ & & * & & * & * & * \\ & & * & * & * & * & \\ & & & & * & & * \end{pmatrix} \end{array}, \quad \begin{array}{c} \\ 6 \\ 4 \\ 2 \\ 5 \\ 1 \\ 7 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} 6 & 4 & 2 & 5 & 1 & 7 & 3 \end{array} \\ \begin{pmatrix} * & & * & & & & \\ & * & * & * & * & & \\ * & * & * & & * & & \\ & * & & * & * & & \\ & * & * & * & * & * & \\ & & & & * & * & * \\ & & & & & * & * \end{pmatrix} \end{array},$$

Factorization: complexity

Complexity of (some) factorizations

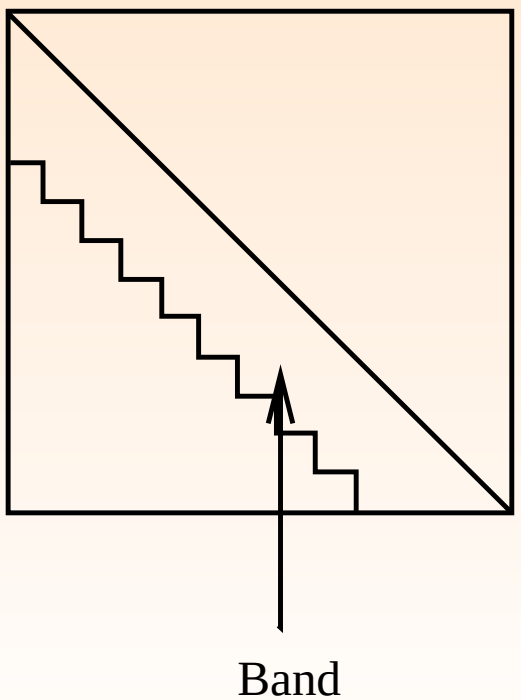
- Sequential complexity dominated by the factorization, but see our comments on parallel costs
- General dense matrices
  - Space: $O(n^2)$
  - Time: $O(n^3)$
- General sparse matrices
  - Space: $\eta(L) = n + \sum_{i=1}^{n-1}(\eta(L_{*i}) - 1)$
  - The i-th step: $\eta(L_{*i}) - 1$ div, $1/2(\eta(L_{*i}) - 1)\eta(L_{*i})$ multiple-add
  - Time totally: $1/2\sum_{i=1}^{n-1}(\eta(L_{*i}) - 1)(\eta(L_{*i}) + 2)$





Complexity

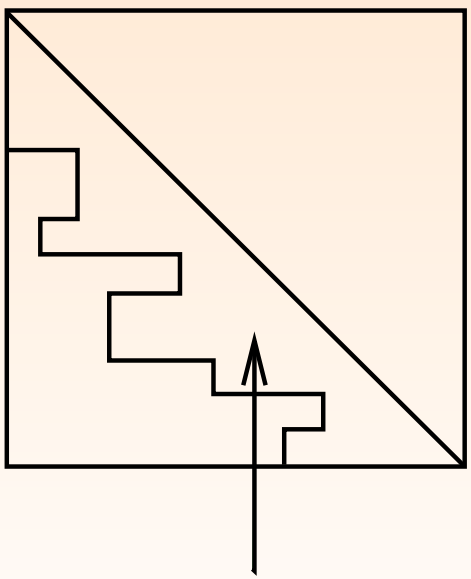
- Band schemes ($\beta << n$)
  - Space: $O(\beta n)$
  - Time: $O(\beta^2 n)$



Factorization: complexity

Complexity of (some) factorizations

- Profile/envelope schemes
  - Space: $\sum_{i=1}^{n} \beta_i$
  - β_i: lengths of row segments containing their nonzeros
  - Complexity can expressed similarly.



Factorization: complexity

Complexity of (some) factorizations

- Nested dissection
- Planar graphs, 2D finite element graphs (bounded degree)
  - Space: $O(n \log n)$
  - Time: $O(n^{3/2})$
- 3D Finite element graphs
  - Space: $O(n^{4/3})$
  - Time: $O(n^2)$

Outline

Algebraic preconditioning

- Finite precision fp64 arithmetic: computed factors are not exact.
- Lower precision arithmetic: even less accuracy
- Parallelism: sometimes hard to get complete factorization, the effort to obtain more accurate results can lead to complex coding and unavoidable inefficiencies: further approximation
- What about even a stronger relaxation: intentional relaxation of factorizations

Algebraic preconditioning

Two basic possibilities

- Approximate factorizing of A can be interpreted as a splitting of A

$$A = M - E,$$

Algebraic preconditioning

Two basic possibilities

- Approximate factorizing of A can be interpreted as a splitting of A

$$A = M - E,$$

- The matrix M nonsingular and (easy to invert, we like to invert ☺); E is the error matrix. The iterations are then

$$x^{(k+1)} = M^{-1}Ex^{(k)} + M^{-1}b, \quad k = 0, 1, \ldots; \text{ provided } x^{(0)}$$

This can be rewritten as

  - stationary iterations

$$x^{(k+1)} = x^{(k)} + M^{-1}(b - Ax^{(k)}) = x^{(k)} + M^{-1}\, r^{(k)}, \quad k = 0, 1, \ldots$$

  - considered as system transformation, often used with Krylov space methods

$$x^{(approx)} = x^{(approx)} + M^{-1}(b - Ax^{(approx)}).$$

Splitting rewritten as stationary iterations

Theorem

For any initial $x^{(0)}$ and vector b, the stationary iteration converges if and only if the spectral radius of $(I - M^{-1}A)$ is less than unity.

- $A = D_A + L_A + U_A$: more classical choices for M
  - Richardson method: $M = \omega^{-1} I$,
  - Jacobi and damped Jacobi methods: $M = D_A$ and $M = \omega^{-1} D_A$,
  - Gauss-Seidel and SOR methods: $M = D_A + L_A$ and $M = \omega^{-1} D_A + L_A$ ($\omega > 0$).
  - Linear convergence, its guarantees.

Algebraic preconditioning

Splitting rewritten as stationary iterations: convergence: just reminder

Theorem

If $A \in \mathbb{R}^{n \times n}$ is strongly diagonally dominant then Jacobi method and Gauss-Seidel method are convergent.

Theorem

If $A \in \mathbb{R}^{n \times n}$ is symmetric with positive diagonal D_A then the Jacobi method is convergent iff A and $2D_A - A$ are positive definite.

Theorem

If $A \in \mathbb{R}^{n \times n}$ is symmetric and positive definite then the Gauss-Seidel method is convergent.

Algebraic preconditioning

Preconditioning as a system transformation

- Consider the preconditioned linear system

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

  Here M^{-1} is applied to A from the left.
- The linear system can be also preconditioned from the right

$$AM^{-1}y = b, \qquad x = M^{-1}y.$$

Algebraic preconditioning

Preconditioning as a system transformation

- Consider the preconditioned linear system

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

  Here M^{-1} is applied to A from the left.
- The linear system can be also preconditioned from the right

$$AM^{-1}y = b, \qquad x = M^{-1}y.$$

- Is one of them better? No.

Theorem

Let δ and Δ be positive numbers. Then for any $n \geq 3$ there exist nonsingular $n \times n$ matrices A and M such that all the entries of $M^{-1}A - I$ have absolute value less than δ and all the entries of $AM^{-1} - I$ have absolute values greater than Δ.

Algebraic preconditioning

Preconditioning as a system transformation

- Consider the preconditioned linear system

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

  Here M^{-1} is applied to A from the left.
- The linear system can be also preconditioned from the right

$$AM^{-1}y = b, \qquad x = M^{-1}y.$$

- Is one of them better? No.

Theorem

Let δ and Δ be positive numbers. Then for any $n \geq 3$ there exist nonsingular $n \times n$ matrices A and M such that all the entries of $M^{-1}A - I$ have absolute value less than δ and all the entries of $AM^{-1} - I$ have absolute values greater than Δ.

- Left/right: to be compatible with the Krylov space accelerator ☺
- Generally cheaper to apply M^{-1} and A separately.

Algebraic preconditioning

From direct methods to preconditioning

- Zoological garden of approaches: structure-based, threshold-based, memory-based. Algorithms may modify the standard LU/LDL^T scheme.

Algebraic preconditioning

From direct methods to preconditioning

- Zoological garden of approaches: structure-based, threshold-based, memory-based. Algorithms may modify the standard LU/LDL^T scheme.
- Holy grail for prescribing $\mathcal{S}(A)$?

Theorem

Consider the incomplete LU factorization $A + E = \widetilde{L}\widetilde{U}$ with sparsity pattern $\mathcal{S}\{\widetilde{L} + \widetilde{U}\}$. The entries of the error matrix E are zero at positions $(i, j) \in \mathcal{S}\{\widetilde{L} + \widetilde{U}\}$.

Algebraic preconditioning

From direct methods to preconditioning

- Zoological garden of approaches: structure-based, threshold-based, memory-based. Algorithms may modify the standard LU/LDL^T scheme.
- Holy grail for prescribing $\mathcal{S}(A)$?

Theorem

Consider the incomplete LU factorization $A + E = \widetilde{L}\widetilde{U}$ with sparsity pattern $\mathcal{S}\{\widetilde{L} + \widetilde{U}\}$. The entries of the error matrix E are zero at positions $(i, j) \in \mathcal{S}\{\widetilde{L} + \widetilde{U}\}$.

- No. Improvement from an increase of $\mathcal{S}\{\widetilde{L} + \widetilde{U}\}$ are typically very slow.
- So, what to do?

From direct methods to preconditioning

- Two general theoretical directions
  - I. Avoiding breakdowns, possible for special matrices like: (M-matrix)

$$A = \begin{pmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{pmatrix}.$$

I. Avoiding breakdowns

- But factorization may breakdown even in case of Cholesky and/or low precision

$$A = \begin{pmatrix} 3 & -2 & & 2 \\ -2 & 3 & -2 & \\ & -2 & 3 & -2 \\ 2 & & -2 & 8 \end{pmatrix}, \; L = \begin{pmatrix} 1 & & & \\ -2/3 & 1 & & \\ & -6/5 & 1 & \\ 2/3 & 4/5 & -2/3 & 1 \end{pmatrix}, \; D = \begin{pmatrix} 3 & & & \\ & 5/3 & & \\ & & 3/5 & \\ & & & 16/3 \end{pmatrix}.$$

$$\widetilde{L} = \begin{pmatrix} 1 & & & \\ -2/3 & 1 & & \\ & -6/5 & 1 & \\ 2/3 & & -10/3 & 1 \end{pmatrix}, \; \widetilde{D} = \begin{pmatrix} 3 & & & \\ & 5/3 & & \\ & & 3/5 & \\ & & & 0 \end{pmatrix}.$$

Algorithm (Trial-and-error global shifted incomplete factorization)

1: ***for*** $k = 0, 1, 2, \ldots$ ***do***
2: $A + \alpha^{(k)} I \approx \widetilde{L}\widetilde{U}$ ▷ *Perform incomplete factorization*
3: *If successful,* $\alpha = \alpha^{(k)}$ *and return*
4: $\alpha^{(k+1)} = 2\alpha^{(k)}$
5: ***end for***

Incomplete factorizations

II. Increasing hope for fast convergence.

- II. Such hope indicated for model problems by $\kappa(M^{-1}A)$
  - For example, it is possible to go from $O(h^{-2})$ to $O(h^{-1})$ by special constructions and/or reorderings
  - For model problems ☺
- Generally, no royal way to efficient preconditioning based on relaxed factorizations
- But, still a field with great potential for research.

Thank you for your attention!