Sparse Matrices in Numerical Mathematics

Miroslav Tůma

Faculty of Mathematics and Physics

Charles University

mirektuma@karlin.mff.cuni.cz

Praha, December 21, 2021

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Introductory notes

- Created as a material supporting online lectures of NMNV533.
- Assuming basic knowledge of algebraic iterative (Krylov space) and direct (dense) solvers (elimination/factorization/solve)
- Many techniques can be formulated for both SPD and nonsymmetric cases with only slight algorithmic (but possibly strong theoretical) differences. Orientation in variants of Cholesky and LU decompositions is assumed.
- We will concentrate here on purely algebraic techniques which often serve as building blocks for more complex approaches.
- Some important techniques are not mentioned at all (MG/ML preconditioners, DD techniques, row projection techniques).
- Some ideas and techniques are only mentioned (block algorithms) Only preconditioning of real systems is considered here.

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

The problem

- Most of our activities around solving

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

- Direct methods
- Iterative methods
- Practical boundaries between them more and more fuzzy.
- But they are principially different.

Direct methods and algebraic preconditioners

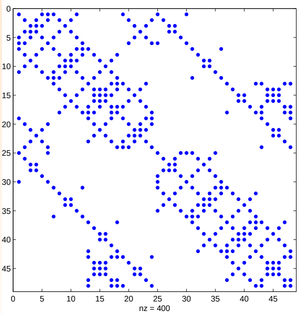
Direct methods

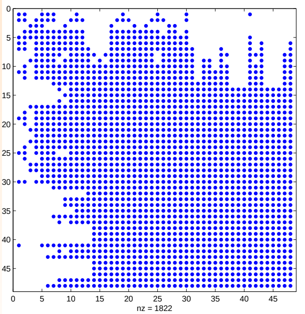
- Direct methods: the name traditionally used for the approach based on decomposition and subsequent substitutions
- The most simple case: $A \rightarrow LL^T$ or LDL^T or LU

  In principal = Gaussian elimination. Modern (decompositional) form based a lot on the work of Householder (end of 1950's)
  - Occasionally other decompositions
  - Most work is in the (Cholesky, indefinite, LU) decomposition.
  - But: It is the computer model (sequential, concurrent processors, multicore, GPU) which decides about the relative complexity of the two steps.
- The algorithms can be made more efficient/stable by the use of additional techniques used before, after or during the decomposition.
- In particular, solution can be made more precise by an auxiliary iterative method.

Sparse matrices

- Sparse matrices may look like as follows





Direct methods and algebraic preconditioners

Iterative methods

- $x^0, x^1, \ldots$
- Iterative method are usually accompanied by a problem transformation based on a direct method called preconditioner.
- Algebraic preconditioners are tools to convert the problem $Ax = b$ into the one which is easier to solve. They are typically expressed in matrix form as a transformation like:

$$MAx = Mb$$

- M can be then used to apply approximation to A^{-1} to vectors used in the iterative method.
- In practice, it can store approximation to A or A^{-1} (approximate inverse).
- The computation is often based on a relaxation of a direct method, but not always.

Sparsity: PR

SPARSITY!

Sparse decompositions

- Exact (direct) decompositions $A = \; LL^T$, LU (up to the floating-point model) → Direct methods

- Inexact processes able to provide approximation to A^{-1}
  - incomplete decompositions ($A \approx \; LL^T$, LU etc.)
  - incomplete inverse decompositions ($A^{-1} \approx ZZ^T$, WZ^T etc.)
- → Preconditioners

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. **Basic terminology**
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Basic Terminology

Definition

Size of a set X (number of its elements) will be denoted by $|X|$. **Set partitioning** of a set X will be called the set $\mathcal{P}_X = \{P_1, \ldots, P_p\}$ of a system of its nonempty and mutually disjoints subsets whose union is X. In the other words, it is $\mathcal{P}_X$ of subsets of X satisfying the following conditions

- $\emptyset \notin \mathcal{P}_X$,
- $X = \bigcup_{P_i \in \mathcal{P}_X} P_i$,
- $(P_1 \in \mathcal{P}_X \wedge P_2 \in \mathcal{P}_X) \wedge P_1 \neq P_2 \Rightarrow P_1 \cap P_2 = \emptyset$.

Elements $P \in \mathcal{P}_X$ are called classes of the partitioning $\mathcal{P}_X$.

Basic Terminology

- $$\binom{X}{2} = \{Y \subseteq X \mid |Y| = 2\}$$
- Vectors denoted by small letters as v, u, x, matrices by capital letters as $A, B, \ldots$
- $A \in \mathbb{R}^{n\times n}, 1 \leq i \leq n$, is nonsingular, the right-hand side vector $b \in \mathbb{R}^n$ is given and $x \in \mathbb{R}^n$ is the required solution vector. n is the **order** (or dimension) of A.
- $$A = (a_{ij}), \;\; 1 \leq i, j \leq n. \tag{1}$$

  Similarly for other matrices.
- Matlab notation for subvectors, submatrices as $A_{i_1:i_2, j_1:j_2}$
- Using mostly real objects from $\mathbf{R}^n$ or $\mathbf{R}^{n\times n}$.

Basic Terminology (continuation)

- An entry of A that has a non zero value or is treated as having a non zero value is called a **non zero**. Column j of A is denoted by A_{*j} and row i by A_{i*}.
- The matrix A is **symmetric** if, for all i and j,

$$a_{ij} = a_{ji}, \;\; 1 \leq i, j \leq n. \tag{2}$$

  Otherwise, the matrix is **nonsymmetric**.

- If A is symmetric then it is said to be **symmetric positive definite** if

$$x^T Ax > 0 \text{ for all non zero } x \in \mathbb{R}^n. \tag{3}$$

  Otherwise, A is **symmetric indefinite** .

Basic Terminology (continuation)

- The **sparsity pattern** $\mathcal{S}\{A\}$ of A is the set

$$\mathcal{S}\{A\} = \{(i,j) \,|\, a_{ij} \neq 0,\, i,j = 1,\ldots,n\}.$$

- The **number of non zeros** in A is $nz(A)$ (or $|\mathcal{S}\{A\}|$ or simply $|A|$). $\mathcal{S}\{A\}$ is symmetric if $a_{ij} \neq 0$ if and only if $a_{ji} \neq 0$ (the values of the two entries need not be the same).
- If $\mathcal{S}\{A\}$ is symmetric then A is said to be **structurally symmetric**. In some situations, sparse vectors (vectors that contain many zero entries) are considered. The sparsity pattern of $v \in \mathbb{R}^n$ is given by

$$\mathcal{S}\{v\} = \{i \,|\, v_i \neq 0\},$$

and $|\mathcal{S}\{v\}|$ is the **length** of v.

Basic Terminology (continuation)

- Block partitioning of A into $nb > 1$ blocks. We formally express the partitioning as

$$A = (A_{i,j}), \;\; A_{i,j} \in \mathbb{R}^{n_i \times n_j}, \;\; 1 \leq i, j \leq nb, \tag{4}$$

  that is,

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,nb} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,nb} \\ \vdots & \vdots & \ddots & \vdots \\ A_{nb,1} & A_{nb,2} & \cdots & A_{nb,nb} \end{pmatrix}. \tag{5}$$

  We assume the square blocks $A_{i,i}$ on the diagonal are nonsingular.

- A is **block diagonal** if $A_{i,j} = 0, i \neq j$, **lower block triangular** if $A_{i,j} = 0, i > j$ or **upper block triangular** if $A_{i,j} = 0, i < j$.

Basic Terminology

Algorithms, implementation

- Basic generally used terminology: if, endif, while, end while, begin, end
- Vague distinction between **algorithm** and **implementation**
- Exercises in Matlab - with some details belonging more to implementation

Basic Terminology (continued)

- The **computational complexity** of a numerical algorithm is typically based on estimating asymptotically the number of integer or floating-point operations or the memory usage.

Definition

A real function $f(k)$ of a nonnegative real k satisfies $f = O(g)$ if there exist positive constants c_u and k_0 such that

$$f(k) \leq c_u g(k) \text{ for all } k \geq k_0. \tag{6}$$

We say that $f = \Theta(g)$ if, additionally, there exists a positive constant c_l such that

$$0 \leq c_l \; g(k) \leq f(k) \leq c_u \; g(k) \text{ for all } k \geq k_0.$$

Basic Terminology (continued)

- While $O(g)$ bounds f asymptotically from above, $\Theta(g)$ represents an **asymptotically tight** bound.
- As a simple illustration, consider the quadratic function

  $$f(k) = \alpha * k^2 + \beta * k - \gamma.$$

  Provided $\alpha \neq 0$, $f(k) = \Theta(k^2)$ and the coefficient of the highest asymptotic term is α. Computational complexity can estimate quantities related to the worst-case behaviour of an algorithm (**worst-case complexity**), or it can express average behaviour (**average-case complexity**).
- Unit costs, Sparse matrix algorithms that are $\Theta(n^3)$ are considered to be computationally expensive.

Computer architectures

Computer architectures

- Need to understand their basic principles
- Von Neumann model: CPU, memory, communication system
- Nowadays, all of these components extremely structured and always in development
- CPU
  - Speed often measured by Mflops (millions of floating-point operations) - this can be misleading, often because of memory effects, but also:
  - **multiple functional units** (since early machines, up to now); **superscalar, threads**
  - **instruction pipelining** (segmentation of instructions)
  - **data pipelining** (segmentation of operations) - vectorization
  - **chaining** (connecting pipelines) to get supervector speed

Computer architectures II

- **Memory**
  - Registers (fastest), cache, main (central) memory, disc memory, local network, ...
  - Real and virtual memory
- All of this shows the importance of using vectors, blocks.
- Underlined by the development of BLAS (basic linear algebra subroutines)

procedure	*comm*	*ops*	ratio
BLAS 1: AXPY: $y = y + \alpha x$	$3n + 1$	$2n$	$2/3$
BLAS 2: GEMV: $y = Ax$	$n^2 + 2n$	$n(2n-1)$	2
BLAS 3: GEMM: $C = AB$	$3n^2$	$n^2(2n-1)$	$n/2$

Complexity of algorithms

Complexity here and in CS

- Because of the development in computations, MFLOPs may be misleading
- Still terminology $O(.)$ (bounding from above) or $\Theta(.)$ (bounding from both sides) sometimes relevant - consists in replacing the bound (bounds) by $constant \times simpler\ function$ (etalon).
- Simpler functions are, e.g., $n^2, n^3, \log n, \ldots$
- Distinguish **worst case** and **average case** analysis
- Inverse Ackermann function will be introduced in exercises
- In CS: polynomial complexity versus superpolynomial complexity. Our case: even n^3 may be too much.
- Decision problems, polynomial reduction, class $\mathcal{NP}$, etc.

Outline

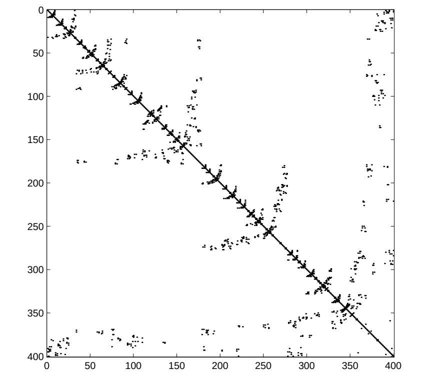
1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. **Sparsity**
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Sparsity

Sparsity: taking into account the structure of matrix nonzeros

Definition

Matrix $A \in \mathbb{R}^{m \times n}$ is said to be sparse if it has $O(\min\{m, n\})$ entries.



Sparsity

Definition

Matrix $A \in \mathbb{R}^{m \times n}$ is said to be sparse if it has row counts bounded by $r_{max} << n$ 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 we can exploit the fact that a part of its entries is equal to zero.

Sparsity

An example showing importance of small exponent γ for $n = 10^4$

γ	$n^{1+\gamma}$
0.1	25119
0.2	63096
0.3	158489
0.4	398107
0.5	1000000

Sparsity

Rough comparison of **dense** and **sparse** (dimension, storage, time for decomposition)

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

Sparsity

Sparsity importance

- Absolutely crucial for direct methods: complexity for generally dense matrices, sequential case: $O(n^3)$ factorization, $O(n^2)$ substitutions
- Useful for iterative methods as well: repeated multiplications
- sparse matrix: its combinatorial structure of zeros and nonzeros can be exploited
- complexity in the sparse case depends on the decomposition model (implementation, completeness/incompleteness)
- **Sparsity pattern**

$$\mathcal{S}(A) = \{(i,j) \,|\, a_{ij} \neq 0,\, i,j = 1,\ldots,n\}. \tag{7}$$

Sparsity

Structured sparsity

- There are special and important types of **structured sparsity**
- Consider, for example, a **block structured** matrix

$$A = \begin{pmatrix} A_{i_1,i_1} & A_{i_1,i_2} & \cdots & A_{i_1,i_{p-1}} \\ A_{i_2 i_1} & A_{i_2 i_2} & \cdots & A_{i_2 i_{p-1}} \\ \vdots & \vdots & \ddots & \vdots \\ A_{i_{p-1} i_1} & A_{i_{p-1} i_2} & \cdots & A_{i_{p-1} i_{p-1}} \end{pmatrix} \tag{8}$$

- special cases of structured sparsity: **block diagonal, block triangular** matrices
- we will introduce profile and banded structures later

Sparsity

Reducibility

Definition

Matrix $A \in \mathbf{R}^{n\times n}$ is **reducible**, if there is a permutation matrix P such that

$$P^T A P = \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix}, \tag{9}$$

where A_{11} and A_{22} are square nontrivial matrices (of dimension at least 1). If A is not reducible, it is called **irreducible**. Matrices of dimension 1 are always considered to be irreducible.

Remark

Symmetric reducible matrix is block diagonal.

Data structures for sparse matrices and vectors

Definition

List $list = (x_1, \ldots, x_k)$ represents a sequence of (arbitrary) elements. x_1 will be called **head (first element)** of a list, x_k is **tail (last element)** of a list. Generally, $list(i) = x_i$. Empty list: $list = ()$. List length is k.

Definition

List is called **queue**, if it enables efficient

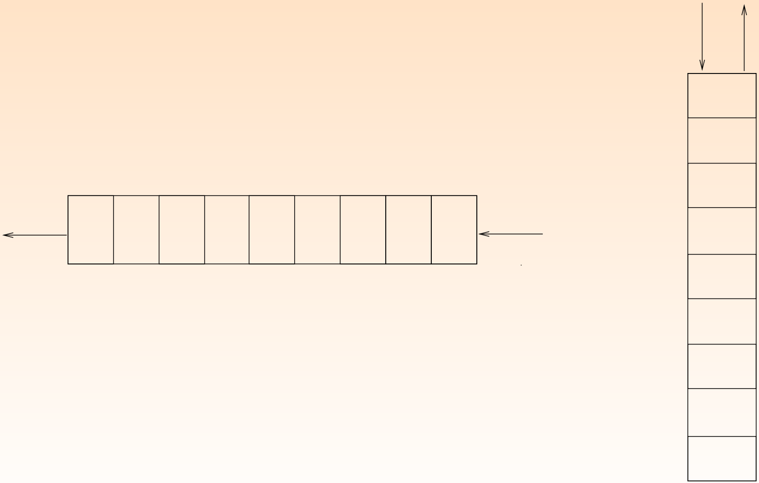
- access head of a list,
- deletion of the list head (pop) and
- adding an element behind the current tail.

List is called **stack**, if it enables efficient

- access head of a list,
- deletion of the list head (pop) and
- adding and element before the current head (push).

Data structures for sparse matrices and vectors

Queue and stack are schematically depicted below. The arrows represent efficient (easily implementable) operations.



Sparse vectors and matrices in a computer

- Consider a sparse vector.

Example

Consider the sparse row vector $v \in \mathbb{R}^8$

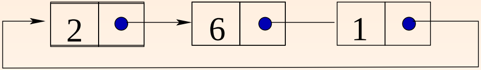
$$v = \begin{pmatrix} 1. & -2. & 0. & -3. & 0. & 5. & 3. & 0. \end{pmatrix}. \tag{10}$$

The real array `valV` that stores the non zero values and corresponding integer array `indV` are as follows.

Subscripts	1	2	3	4	5
`valV`	1.	-2.	-3.	5.	3.
`indV`	1	2	4	6	7

Sparse vectors and matrices in a computer

- Alternatively, a **linked list** can be used.
- **linked list - based** format: stores matrix rows/columns as items connected by pointers
- linked lists can be cyclic, one-way, two-way
- A figure for demonstration, only values (not their indices) are shown



- **rows/columns** embedded into a larger array: emulated dynamic behavior

Sparsity

Data structures for sparse matrices: dynamic data structures

- Explicit embedding into a large array.

Example

Two possible ways of storing the sparse vector using linked lists.

Subscripts	1	2	3	4	5
Values	1.	-2.	-3.	5.	3.
Indices	1	2	4	6	7
Links	2	3	4	5	0
Header	1				

Subscripts	1	2	3	4	5
Values	5.	3.	1.	-2.	-3.
Indices	6	3	1	2	4
Links	2	0	4	5	1
Header	3				

Sparse vectors and matrices in a computer

- **Reasons** for using linked lists: straightforward adds and removes.

Example

An entry -4 has been added to the sparse vector in position 5.

Subscripts	1	2	3	4	5	6
Values	1.	-2.	-3.	5.	3.	-4.
Indices	1	2	4	6	7	5
Links	2	3	4	5	**6**	**0**
Header	1					

Sparse vectors and matrices in a computer

- **Reasons** for using linked lists: straightforward adds and removes.

Example

An entry -2 in position 2 has been removed.

Subscripts	1	2	3	4	5
Values	1.	$*$	-3.	5.	3.
Indices	1	$*$	4	6	7
Links	**3**	$*$	4	5	0
Header	1				

Sparse vectors and matrices in a computer

- The vector data structures can be generalized to store sparse matrices. The simplest way to store a sparse matrix is using **coordinate** (or **triplet** format. The individual entries of A are held as triplets (i, j, a_{ij}), where i is the row index and j is the column index of the entry $a_{ij} \neq 0$.
- More commonly used is the **CSR (Compressed Sparse Row)** format. The column indices of the entries of A are held by rows in an integer array (which we will call `colindA`) of length $nz(A)$, with those in row 1 followed by those in row 2, and so on (with no space between rows). Often, within each row, the entries are held by increasing column index.
- **CSC (Compressed Sparse Columns)** is defined analogously by holding the entries by columns, rather than by rows.
- If A is **symmetric**, only the lower (or upper) triangular part is generally stored. If the matrix values are not stored, the arrays `rowptrA` and `colindA` represent the graph $\mathcal{G}(A)$.

Sparse vectors and matrices in a computer

- Consider the sparse matrix $A \in \mathbb{R}^{5\times 5}$

$$\begin{array}{cc} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{pmatrix} 3. & & & -2. & \\ & 1. & & & 4. \\ -1. & & 3. & & 1. \\ & & & 1. & \\ & 7. & & & 6. \end{pmatrix} \end{array}. \tag{11}$$

Coordinate format represents A as follows. Note that the entries are in no particular order.

Subscripts	1	2	3	4	5	6	7	8	9	10
`rowindA`	3	2	3	4	1	1	2	5	3	5
`colindA`	3	2	1	4	4	1	5	5	5	2
`valA`	3.	1.	-1.	1.	-2.	3.	4.	6.	1.	7.

Sparse vectors and matrices in a computer

- CSR format represents A as follows. Here the entries within each row are in order of increasing column index. This additional condition is often but not always used.

Subscripts	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.

- In our codes: ia: rowptrA, ja: colindA, aa: valindA

Sparse vectors and matrices in a computer

- Adding and deleting entries is possible if the sparse rows or columns are stored using linked lists.
  The matrix held as a collection of columns, each in a linked list, as follows. Here the array `colA_head` holds header pointers. Columns held in order but this is not a requirement.

Subscripts	1	2	3	4	5	6	7	8	9	10
`rowindA`	3	2	3	4	1	1	2	5	3	5
`valA`	3.	1.	-1.	1.	-2.	3.	4.	6.	1.	7.
`link`	0	10	0	0	4	3	9	0	8	0
`colA_head`	6	2	1	5	7					

  If we consider column 4, then `colA_head`$(4) = 5$, `rowindA`$(5) = 1$ and `valA`$(5) = -2.$, so the first entry in column 4 is $a_{1,4} = -2.$.
  Next, `link`$(5) = 4$, `rowindA`$(4) = 4$ and `valA`$(4) = 1.$, so the next entry in column 4 is $a_{4,4} = 1.$.

Sparse vectors and matrices in a computer

- A **disadvantage of linked list storage** is that it prohibits the fast access to rows (or columns) of the matrix that is needed for efficient processing on contemporary computers that use vectorization and/or work with matrix blocks. Consequently, CSR or CSC formats are commonly used in sparse direct methods.
- In some cases a storage format with some **additional elbow space** for new non zero entries of A is needed. This is often the case in **approximate factorizations** where new non zero entries can be added and/or removed and it is **hard to predict the necessary space** in advance. Such storage scheme that has some extra space to embed new non zeros is the **DS format**.
- Goal of the course is to show that in important cases **we are able to predict** ...

Sparse vectors and matrices in a computer

- Consider again the sparse matrix $A \in \mathbb{R}^{5\times 5}$ (11). The DS format represents A as follows.

Subscripts	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`rowptrA`	1	5	8	12	14										
`colindA`	1	4			2	5		1	3	5		4		2	5
`valAR`	3.	-2.			1.	4.		-1.	3.		1.		1.	7.	6.
`rowlength`	2	2	3	1	2										
`colptrA`	1	4	6	9	12										
`rowindA`	1	3		2	5	3			1	4		2	3	5	
`valAC`	3.	-1.		1.	7.	3.			-2.	1.		4.	1.	6.	
`collength`	2	2	1	2	3										

Sparse vectors and matrices in a computer

- When dealing with the DS format it can happen that the free space between row and/or column segments disappears throughout a computational algorithm. Then **the DS format must be reorganized**. In particular, a row segment can be moved to the end of the arrays `valAR` and `colindA` implying also a corresponding update in `rowptrA`. The space where the row i originally resided is then denoted as free. If there is no free space at the end of the arrays `valAR` and `colindA`, a **compression** of the row segments or full reallocation of the DS format should be done.
- While the DS format seems to be complicated, it can be **extremely useful in some cases**. It is surprisingly efficient if the amount of changes is reasonably limited as it often is in approximate factorizations.

Sparsity

Data structures for sparse matrices: static versus **dynamic** data structures: summary.

- dynamic data structures:
  - – more flexible but this flexibility might **not be needed**
  - – difficult to **vectorize**
  - – difficult to keep **spatial locality**
  - – used preferably for storing vectors
- static data structures:
  - – we need to avoid ad-hoc insertions/deletions
  - – much simpler to vectorize
  - – efficient access to rows/columns

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. **Graphs and their matrices**
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Undirected graphs: basic terminology

Definition

Simple undirected graph G is an ordered pair of sets (V, E), where $V = \{v_1, \ldots, v_n\}$ is called the set of **vertices** of G, $E = \{e_1, \ldots, e_m\}$ is called the set of **edges** satisfying

$$E \subseteq \binom{V}{2}$$

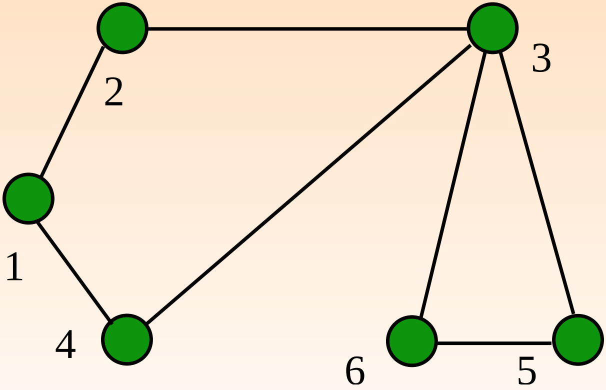
.

Remark

Formally the definition also **means** *that there are no multiple edges and no loops (edges that would belong to V only).*

Graphs and their matrices

Undirected graph: example

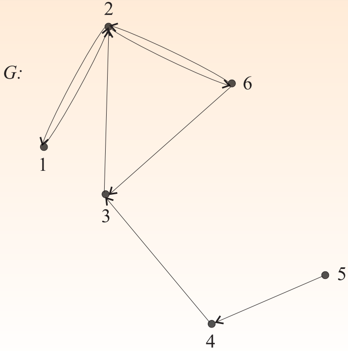


Simple undirected graph $G = (\{1,2,3,4,5,6\}, \{\{1,2\}, \{2,3\}, \{1,4\}, \{3,4\}, \{3,6\}, \{3,5\}, \{5,6\}\})$

Directed graphs: example

Definition

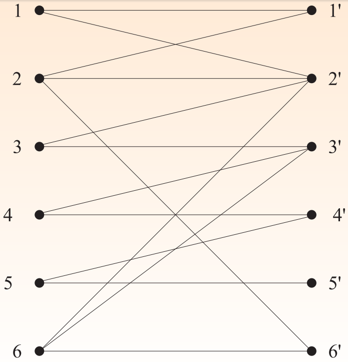
A simple directed graph is an ordered pair of sets (V, E) such that $E = \{(i, j) | i \in V, j \in V\}$. V is called the **vertex (node)** set and $E \subseteq V \times V$ is called the **edge (arc)** set.



Bipartite graphs: example

Definition

A simple bipartite graph is an ordered pair of sets (R, C, E) such that $E = \{\{i,j\}|i \in R, j \in C\}$. R is called the **row vertex** set, C is called the **column vertex** set and E is called the **edge** set.



Graphs and their matrices

Terminology of undirected graphs (repetition)

- neighbors, incidence, isolated vertices, subgraph, graph induced by a set of vertices / edges, clique
- adjacency set for a vertex set U (set of neighbors)

$$adj_G(U) = \{u \notin U | (\exists v \in U)(u \in adj_G(v))\} \tag{12}$$

- A (finite) **walk** is a sequence of edges and vertices that can be written as

$$\{i_0, i_1\}, \{i_1, i_2\}, \ldots, \{i_{t-1}, i_t\}$$

- A **trail** is a walk with all edges are distinct.
- A **path** is a walk with all vertices distinct. Denoted by $i_0 \xLeftrightarrow[\{1,\ldots,j\}]{G} i_t$
- **closed** walks, trails, paths modify the definitions by setting $i_0 \equiv i_t$.

Graphs and their matrices

Terminology of undirected graphs (repetition)

Definition

Two graphs G and H are **isomorphic** if there is a bijection between $V(G)$ and $V(H)$ $f\colon V(G) \to V(H) f\colon V(G) \to V(H)$ such that

$$(\forall u, v \in V(G))(\exists \{u, v\} \in E(G) \Leftrightarrow \exists \{f(u), f(v)\} \in E(H).$$

-

- Reachable vertices, Reachability set (undirected path between i_0 and k using intermediate vertices from S only.

$$Reach(i_0, S, G) = \{k \in V \mid i_0 \overset{G}{\underset{S}{\Leftrightarrow}} k\}$$

- Connected graph, components, coloring of vertices, cycle, acyclic graphs, tree (connected acyclic), forest (acyclic).

Graphs and their matrices

Adjacency matrix of an undirected graph

Definition

For a simple undirected graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ the **adjacency matrix** is the $(0, 1)$ matrix $A_G = (a_{ij})$ $(i, j \in V)$, where a_{ij} is 0 if there is an edge $\{i, j\}$ in E where $i, j \in V$ and zero otherwise.

$$\begin{pmatrix} & 1 & & 1 & & \\ 1 & & 1 & & & \\ & 1 & & 1 & 1 & 1 \\ 1 & & 1 & & & \\ & & 1 & & & 1 \\ & & 1 & & 1 & \end{pmatrix}$$

Adjacency matrix for the undirected graph above.

Graphs and their matrices

Incidence matrix of an undirected graph

Definition

For a simple undirected graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ and $E = \{1, \ldots, m\}$ the **incidence matrix** is the $(0, 1)$ matrix $A_G = (a_{ij})$ $(i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\})$, where $a_{ij} = 1$, if j is the vertex of the edge i and $a_{ij} = 0$ otherwise.

- Incidence matrix is generally rectangular.
- More slightly differing definitions of adjacency and incidence matrices.
- In particular, various generalizations.

Graphs and their matrices

Transfer between the classes of undirected and directed graphs

- **Symmetrization**: directed → undirected
  - Just considering edges from $V \times V$ as from $\binom{V}{2}$
- **Orientation**: undirected → directed
  - **Not unique**. Instead from an edge from $\binom{V}{2}$ we can have one or two edges from $V \times V$.
- In any case, **part of terminology is shared** between the classes of undirected and directed graphs

Graphs and their matrices

Terminology of directed graphs (additional)

- Walks, trails, paths can be considered as undirected or directed.
- For example, **directed path** is $(i_0, i_1), (i_1, i_2), \dots, (i_{t-1}, i_t)$.
- connectivity → **strong connectivity**.

Definition

Vertices x_1 and x_2 of a directed graph G are **strongly connected**, if

$$x_1 \overset{G}{\Rightarrow} x_2 \quad \wedge \quad x_2 \overset{G}{\Rightarrow} x_1. \tag{13}$$

- Strong connectivity is an equivalence on $V(G)$. Subgraphs **induced** by strong connectivity on $V(G)$ are called strong components of G.

Definition

Directed graph G is strongly connected iff it has only one strong component.

Acyclic graphs and rooted trees

Definition

Topological ordering of a (directed graph, vertices of a directed graph) is a mapping (numbering) $\alpha : V \to \{1, \dots, |V|\}$ of V, such that for all its edges (u, v), $u, v \in V$ we have

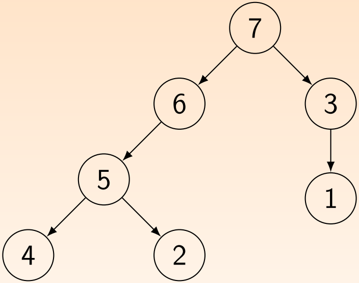
$$\alpha(u) > \alpha(v).$$

Theorem

Directed graph is **acyclic** *if and only if it can be topologically ordered (if there is such numbering α).*

Graphs and their matrices

Acyclic graphs and rooted trees: an example of a topologically ordered acyclic graph



Remark

Note that we define it here by an **exactly opposite** *inequality than it is* **typically** *defined.*

Graphs and their matrices

Definition

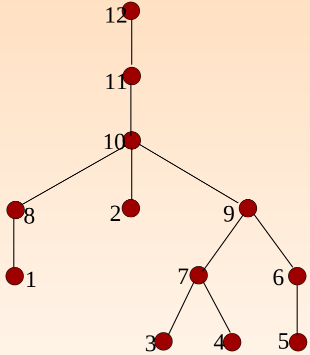
Rooted tree is an acyclic and connected graph (**tree**) having (in addition) one designated vertex r called the **root**.

- The root of a rooted tree determines a **partial ordering** of vertices.
- Designating the root also means **orientation** of edges of the rooted tree. We assume that the orientation is **uniquely** defined by having **directed paths from** r to all vertices of $V \setminus r$.

Terminology of rooted trees

- **Parent** u of a vertex $v \neq r$ is the closest vertex on the unique path $r \to v$ to v. r has no parent. We write $u = parent(v)$ and state that v is a **child** of u. A vertex of a rooted tree without children is its **leaf**.
- **Ancestors** of a vertex v of the rooted tree T are all vertices on the path $r \to v$ (including v) in T. Set of ancestors of v will be denoted by $anc(v)$ or $anc_T(v)$.
- For vertices of a rooted tree T: if u is an ancestor of v then v is called a **descendant** of u. Set of descendants of v will be denoted by $desc(v)$ or $desc_T(v)$.

An example of a rooted tree



- The root of this tree is 12
- Then, for example, 10 is an ancestor of vertices 2, 8 a 9. These vertices are descendants of 10. Set of ancestors of 10 is $anc(10) = \{10, 11, 12\}$. $parent(i)$ for vertices $1 \ldots 11$ is $8, 10, 7, 7, 6, 9, 9, 10, 10, 11, 12,$ **null**.

Graphs and their matrices

Remark

Topological numbering α of a directed acyclic graph $G = (V, E)$ has the following **transitivity** *property.*

$$\alpha(v) < \alpha(u) \text{ for } u \in anc(v),\ u \neq v. \tag{14}$$

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

From matrices to graphs

From matrices to graphs

Sparsity pattern $\mathcal{S}$ of a matrix (matrix structure, positions of nonzero entries) can be conveniently expressed by **graphs** (**graph models**)

$$\Downarrow$$

Different graph models for different purposes

- undirected graph
- directed graph
- bipartite graph

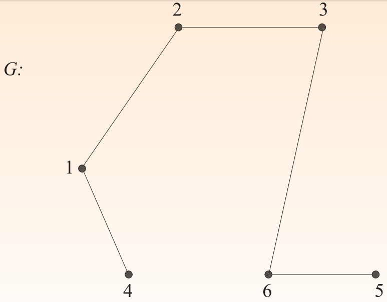
Values of matrix entries can be stored as graph (vertex, edge) **weights**

From matrices to graphs

Symmetric matrix and undirected graph of its sparsity pattern (structure)

Definition

$\{x, y\} \in E$ or $(x, y) \in E \Leftrightarrow$ vertices x and y are **adjacent**
$Adj(x) = \{y | y \text{ and } x \text{ are adjacent } \}$

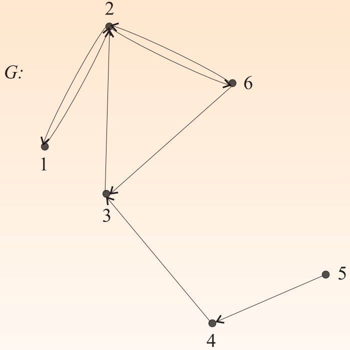


$$\begin{pmatrix} * & * & & * & & \\ * & * & * & & & \\ & * & * & & & * \\ * & & & * & & \\ & & & & * & * \\ & & * & & * & * \end{pmatrix}$$

From matrices to graphs

Nonsymmetric matrix and directed graph of its sparsity pattern (structure)

- Arrows express that the edges (arcs) are directed.



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

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Graph searches

- Many sparse matrix reordering algorithms involve searching the adjacency graph $\mathcal{G}(A) = (V, E)$. The sequence in which the vertices are visited can be used, for example, to reorder the graph and hence permute the matrix.
- Given a start vertex, a **graph search** (also called a **graph traversal**) performs a step-by-step exploration of the vertices and edges of $\mathcal{G}(A)$, generating sets of visited vertices and explored edges.
- Let V_v be the set of visited vertices and V_n be the set of vertices that have not yet been visited. Following some chosen rule, the search step selects an unexplored edge $e = (v, w) \in E$ such that one of its vertices belongs to V_v.

Graph searches:BFS

- Starting from a chosen start vertex s, a **breadth-first search** (BFS) explores all the vertices adjacent to s. It then explores all vertices whose shortest path from s is of length 2, and then length 3, and so on (that is, sibling vertices are visited before child vertices); a queue is used in its implementation.

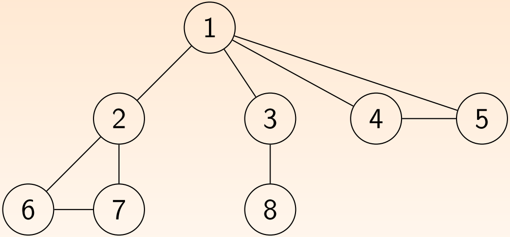


Figure: *An illustration of a BFS of a connected undirected graph, with the labels indicating the order in which the vertices are visited.*

Graph searches:DFS

- A **depth-first search** (DFS) visits the child vertices before visiting the sibling vertices; that is, it traverses the depth of a path before exploring its breadth. Starting from a chosen vertex s, the set of vertices that are visited are those vertices u for which a directed path from s to u exists in $\mathcal{G}$. The edges that are traversed form a tree $\mathcal{T}_s$, called the **depth-first-search tree** of $\mathcal{G}$ starting at the chosen root s, and the edges in this tree are called **tree edges**. The other edges can be divided into three categories:

  Back edges point from a vertex to one of its ancestors in the DFS tree.

  Forward edges point from a vertex to one of its descendants.

  Cross edges point from a vertex to a previously visited vertex that is neither an ancestor nor a descendant.

Graph searches:DFS

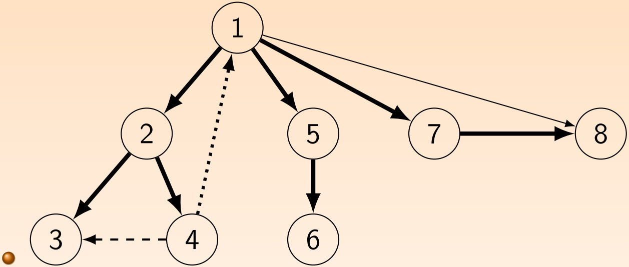


Figure: *An illustration of a DFS search of a connected directed graph. The labels indicate the order in which the vertices are visited and the classification of the edges. The tree edges are in bold;* $(1 \to 8)$ *is a forward edge;* $(4 \to 1)$ *is a back edge;* $(4 \to 3)$ *is a cross edge.*

Graph searches:DFS

Algorithm (Find preorder and postorder lists using a DFS)

```
Initialization: $V_v = \emptyset$, $preorder = ()$ and $postorder = ()$
for each $v \in V$ do
    if $v \notin V_v$ then
        $push(preorder, v)$                ▷ Add $v$ onto the preorder stack
        $V_v = V_v \cup \{v\}$             ▷ Add $v$ to the set of visited vertices
        dfs_step($v$)
    end if
end for
recursive function (dfs_step($v$))
    for each $(v \to w) \in E$ do
        $push(preorder, w)$                ▷ Add $w$ onto the preorder stack
        $V_v = V_v \cup \{w\}$             ▷ Add $w$ to the set of visited vertices
        if $w \notin V_v$ then
            dfs_step($w$)                  ▷ recursive search
        end if
    end for
    $push(postorder, v)$                   ▷ Add $v$ onto the postorder stack
end recursive function
```

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Matmats

1) CSR - CSC

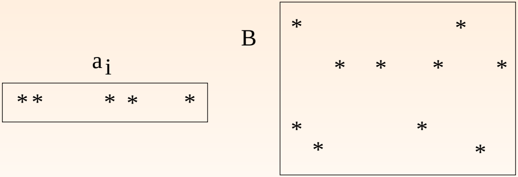
$$C = AB, A = \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix}, B = \begin{pmatrix} b_1, \ldots, b_n \end{pmatrix}, C = (c_{ij}) \tag{15}$$

- Each entry c_{ij} computed as a product of a compressed row of A and compressed column of B
- Not clear whether the result c_{ij} is nonzero
- Consequently: $O(n^3)$ operations, not useful for sparse matrices.

Matmats

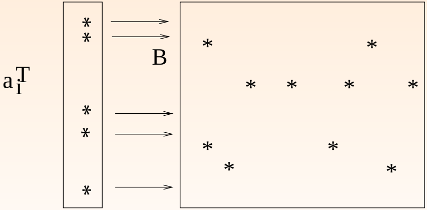
2) CSR - CSR

$$C = AB, A = \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix}, B = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}, C = (c_{ij}) \tag{16}$$



2) CSR - CSR

$$C = AB, A = \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix}, B = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}, C = (c_{ij}) \tag{17}$$



Matmats

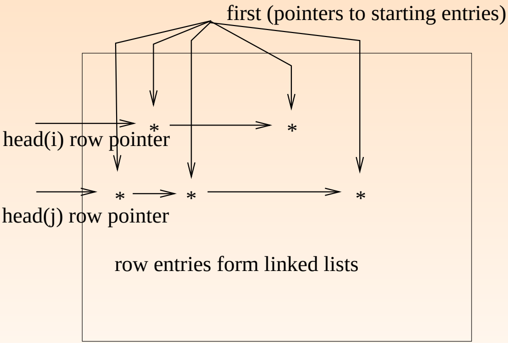
3) CSC - CSR

$$C = AB, A = \left(a_1, \ldots, a_m\right), B = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}, C = (c_{ij}) \tag{18}$$

- How one can store A by CSC and pass it by rows?
- **Pointers to first entries in columns**: (array $first$)
- First test: nonzero in the first row → move one step down, add next nonzero into the list $value(next)$
- Complexity: $O(nonzeros) + O(n)$

Matmats

3) CSC - CSR


first (pointers to starting entries)
*
*
head(i) row pointer
*
*
*
head(j) row pointer
row entries form linked lists

Based on forming **virtual rows** in A

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Schemes for solving systems of linear algebraic equations

The schemes: introduction

- methods based on solving $Ax = b$ by a matrix decomposition – variant of Gaussian elimination; typical decompositions:
- $A = LL^T, A = LDL^T$ (Cholesky decomposition, LDL^T decomposition for SPD matrices)
- $A = LU$ (LU decomposition for general nonsymmetric matrices)
- $A = LBL^T$ (symmetric indefinite / diagonal pivoting decomposition for A symmetric indefinite)

three steps of a (basic!) direct method:

1) $A \rightarrow LU$, 2) y from $Ly = b$, 3) x from $Ux = y$

Schemes for solving systems of linear algebraic equations

Definition

Matrix $A \in R^{n \times n}$ is **strongly regular** if all its leading principal minors are nonzero.

Simple matrices that are regular but not strongly regular.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{19}$$

Schemes for solving systems of linear algebraic equations

Theorem

Let $A \in R^{n \times n}$ be strongly regular. Then there exist unique unit lower triangular matrix L and upper triangular matrix U such that

$$A = LU. \tag{20}$$

Theorem

For each regular A there is a row permutation matrix P such that PA is strongly regular. This permutation matrix can be found on-the-fly during the factorization.

Triangular solves

- Gaussian elimination: sequence of operations that reduce the matrix A to an upper triangular form. This is an an equivalent linear system with an upper triangular matrix. The solution x is computed by solving the triangular systems

  $$Ly = b, \tag{21}$$

  and then

  $$Ux = y. \tag{22}$$

- Solves with the dense right-hand side vector (straightforward) The solution of (21) can be obtained by **forward subsitution** in which we obtain the component y_1 from the first equation, substitute it into the second equation to obtain y_2, and so on.

Triangular solves

Algorithm (Lower triangular solve $Ly = b$ with b dense)

Input: *Lower triangular matrix L with unit diagonal and dense b.*
Output: *Dense solution y.*

```
Initialise: $y = b$
for $j = 1 : n$ do
    for $i = j + 1 : n$ do
        if $l_{ij} \neq 0$ then
            $y_i = y_i - l_{ij} y_j$
        end if
    end for
end for
```

Triangular solves

- When b is sparse, the solution y is also sparse. In particular, if $y_k = 0$ then the outer loop with $j = k$ can be skipped.

Algorithm (Lower triangular solve $Ly = b$ with b sparse)

Input: *Lower triangular matrix L with unit diagonal, sparse b and the set $\mathcal{J}$.*

Output: *Sparse solution y.*

```
Initialise: $y = b$
for $j \in \mathcal{J}$ do
    for $i = j + 1 : n$ do
        if $l_{ij} \neq 0$ then
            $y_i = y_i - l_{ij} y_j$
        end if
    end for
end for
```

- How to get $\mathcal{J}$?

Schemes for solving systems of linear algebraic equations

One step of elimination

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ a_{21}/a_{11} & 1 & & & \\ a_{31}/a_{11} & & 1 & & \\ \vdots & & & 1 & \\ a_{n1}/a_{11} & & & & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & \dots & a_{3n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{pmatrix} \tag{23}$$

$$\equiv M_1 A^{(1)}.$$

Corresponding Schur complement

$$S = A_R^{(1)} = \begin{pmatrix} a_{22}^{(1)} & \dots \; \dots & a_{2n}^{(1)} \\ a_{32}^{(1)} & \dots \; \dots & a_{3n}^{(1)} \\ \vdots & \ddots & \vdots \\ a_{n2}^{(1)} & \dots \; \dots & a_{nn}^{(1)} \end{pmatrix} \in R^{(n-1)\times(n-1)} \tag{24}$$

Schemes for solving systems of linear algebraic equations

Second step of the elimination

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & \dots & a_{3n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & a_{32}/a_{11} & 1 & & \\ & \vdots & & 1 & \\ & a_{n2}/a_{11} & & & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ 0 & a_{22}^{(1)} & \dots & \dots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(1)} & \dots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(1)} & \dots & a_{nn}^{(1)} \end{pmatrix} \tag{25}$$

$$\equiv M_2 A^{(2)}.$$

Summarizing this

$$A = M_1 M_2 \dots M_{n-1} A^{(n-1)}. \tag{26}$$

Schemes for solving systems of linear algebraic equations

$$L = \begin{pmatrix} 1 & & & \\ a_{21}^{(0)}/a_{11}^{(0)} & 1 & & \\ a_{31}^{(0)}/a_{11}^{(0)} & & & \\ \vdots & \vdots & \ddots & \\ a_{n1}^{(0)}/a_{11}^{(0)} & & \dots & 1 \end{pmatrix} \begin{pmatrix} 1 & & & \\ & 1 & & \\ & a_{32}^{(1)}/a_{22}^{(1)} & & \\ \vdots & \vdots & \ddots & \\ & a_{n2}^{(1)}/a_{22}^{(1)} & \dots & 1 \end{pmatrix} \dots = \tag{27}$$

$$\begin{pmatrix} 1 & & & \\ a_{21}^{(0)}/a_{11}^{(0)} & 1 & & \\ a_{31}^{(0)}/a_{11}^{(0)} & a_{32}^{(1)}/a_{22}^{(1)} & & \\ \vdots & \vdots & \ddots & \\ a_{n1}^{(0)}/a_{11}^{(0)} & a_{n2}^{(1)}/a_{22}^{(1)} & \dots & 1 \end{pmatrix}$$

Remark

Subdiagonal entries that are explicitly in the i-th column of M_i, $i = 1, \dots, n-1$ are entries of the unit lower triangular matrix (factor) L.

Schemes for solving systems of linear algebraic equations

$$\begin{pmatrix} A & b \end{pmatrix} = M_1 \begin{pmatrix} A^{(1)} & M_1^{-1} b \end{pmatrix}. \tag{28}$$

and then

$$\begin{pmatrix} A & b \end{pmatrix} = M_1 M_2 \dots M_{n-1} \begin{pmatrix} A^{(n-1)} & L^{-1} b \end{pmatrix}. \tag{29}$$

Algorithm

Generic scheme of the LU factorization.

```
1.  for —————————-
        for —————————-
            for —————————-
                $a_{ij} = a_{ij} - l_{ik} a_{kj}$
            end
        end
    end
```

Submatrix factorization

$$A = \begin{pmatrix} a_{11} & u^T \\ v & C \end{pmatrix}, \tag{30}$$

where

$$v = a_{2:n,1},\, C = A_{2:n,2:n},\, u^T = a_{1,2:n} \tag{31}$$

leads to

$$A = M_1 \begin{pmatrix} a_{11} & u^T \\ & C - vu^T/a_{11} \end{pmatrix} = \begin{pmatrix} 1 & \\ v/a_{11} & I \end{pmatrix} \begin{pmatrix} a_{11} & u^T \\ & C - vu^T/a_{11} \end{pmatrix} \tag{32}$$

$$\equiv \begin{pmatrix} 1 & \\ v/a_{11} & I \end{pmatrix} \begin{pmatrix} a_{11} & u^T \\ & S \end{pmatrix}, \tag{33}$$

Schemes for solving systems of linear algebraic equations

Submatrix factorization: example

$$A = \begin{pmatrix} 4 & -1 & -5 & -11 & 6 \\ -1 & 4 & -2 & -1 & 3 \\ -1 & 0 & 4 & -1 & 5 \\ 6 & 7 & 8 & 10 & 2 \\ 8 & -1 & -1 & 4 & 7 \end{pmatrix}$$

Schemes for solving systems of linear algebraic equations

$$A = \begin{pmatrix} 4 & -1 & -5 & -11 & 6 \\ -0.25 & 3.75 & -3.25 & -3.75 & 4.5 \\ -0.25 & -0.25 & 2.75 & -3.75 & 6.5 \\ 1.5 & 8.5 & 15.5 & 26.5 & -7 \\ 2 & 1 & 9 & 26 & -5 \end{pmatrix}$$

Schemes for solving systems of linear algebraic equations

Submatrix factorization: example

$$A = \begin{pmatrix} 4 & -1 & -5 & -11 & 6 \\ -0.25 & 3.75 & -3.25 & -3.75 & 4.5 \\ -0.25 & -0.07 & 2.53 & -4 & 6.8 \\ 1.5 & 2.27 & 22.87 & 35 & -17.2 \\ 2 & 0.27 & 9.87 & 27 & -6.2 \end{pmatrix}$$

Schemes for solving systems of linear algebraic equations

Submatrix factorization: example

$$A = \begin{pmatrix} 4 & -1 & -5 & -11 & 6 \\ -0.25 & 3.75 & -3.25 & -3.75 & 4.5 \\ -0.25 & -0.07 & 2.53 & -4 & 6.8 \\ 1.5 & 2.27 & 9.03 & 71.1 & -78.6 \\ 2 & 0.27 & 3.89 & 42.6 & -32.7 \end{pmatrix}$$

Submatrix factorization: example

$$A = \begin{pmatrix} 4 & -1 & -5 & -11 & 6 \\ -0.25 & 3.75 & -3.25 & -3.75 & 4.5 \\ -0.25 & -0.07 & 2.53 & -4 & 6.8 \\ 1.5 & 2.27 & 9.03 & 71.1 & -78.6 \\ 2 & 0.27 & 3.89 & 0.6 & \color{red}{14.4} \end{pmatrix}$$

Schemes for solving systems of linear algebraic equations

Algorithm

kij *lu decomposition (row oriented submatrix dense algorithm)*

```
$l = I_n$
$u = O_n$
for k=1:n-1
    for i=k+1:n
        $l_{ik} = a_{ik}/a_{kk}$
        for j=k+1:n
            $a_{ij} = a_{ij} - l_{ik} * a_{kj}$
        end
    end
    $u_{kk:n} = a_{kk:n}$
end
$u_{nn} = a_{nn}$
```

Schemes for solving systems of linear algebraic equations

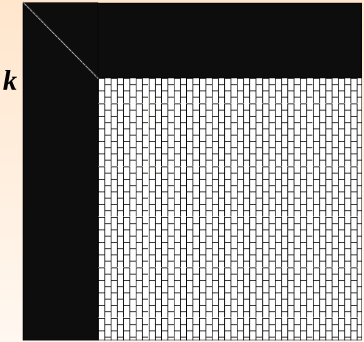
Algorithm

kji *lu decomposition (column oriented submatrix dense algorithm)*

```
$l = I_n$, $u = O_n$
for k=1:n-1
    for s=k+1:n
        $l_{sk} = a_{s,k}/a_{k,k}$
    end
    for j=k+1:n
        for i=k+1:n
            $a_{ij} = a_{ij} - l_{ik} * a_{kj}$
        end
    end
    $u_{kk:n} = a_{kk:n}$
end
$u_{nn} = a_{nn}$
```

Schemes for solving systems of linear algebraic equations

Depicting submatrix LU factorization



Schemes for solving systems of linear algebraic equations

Column algorithm

- Generic scheme with the outer index j
- For $j = 1$ clear since off-diagonal entries in the first column of L are entries of the first column of A divided by $u_{11} = a_{11}$.
- Assume we have computed for $1 < j < n$, in the first $j-1$ steps of factorization first $j-1$ columns of L and U of the matrix $A \in R^{n \times n}$.

$$\begin{pmatrix} L_{j-1} \\ L'_{j-1} \end{pmatrix} U_{j-1} = \begin{pmatrix} A_{1:j-1,1:j-1} \\ A_{j:n,1:j-1} \end{pmatrix}, \tag{34}$$

where

$$L_{j-1} \in R^{(j-1)\times(j-1)},\ L'_{j-1} \in R^{(n-j+1)\times(j-1)},\ U_{j-1} \in R^{(j-1)\times(j-1)} \tag{35}$$

Schemes for solving systems of linear algebraic equations

Column algorithm

- Next step for the j-th column l of L a j-th column z of U is

$$U_{1:j-1,j} = L_{j-1}^{-1} A_{1:j-1,j},\ u_{jj} = a_{j,j} - (L'_{j-1})_{j,1:j-1} U_{1:j-1,j}, \tag{36}$$

$$L_{j+1:n,j} = A_{j+1:n,j} - L'_{j-1} U_{1:j-1,j}. \tag{37}$$

- Next slide: additional row interchanges

Schemes for solving systems of linear algebraic equations

Column algorithm

Algorithm

Basic sparse column LU factorization

Output: *LU factorization $PA = LU$ where P is a row permutation matrix.*

```
Interchange rows of $A$ so that $a_{11} = max_{1 \leq i \leq n} a_{i1}$
Set $l_{11} = 1, u_{11} = a_{11}, L_{2:n,1} = A_{2:n,1}$
for $j = 2 : n$ do
    Solve $L_{1:j-1,1:j-1} U_{1:j-1,j} = A_{1:j-1,j}$
    Set $z = A_{j:n,j} - L_{j:n,1:j-1} U_{1:j-1,j}$        ▷ Vector $z$ is of dimension $n - j + 1$.
    Apply row interchanges to $z$, $A$ and $L$ such that $z_1 = max_{1 \leq i \leq n-j+1} z_i$.
    Set $u_{jj} = z_1$
    Set $L_{j:n,j} = z_{2:n-j+1} / u_{jj}$
end for
```

Schemes for solving systems of linear algebraic equations

Column algorithm: example

$$A = \begin{pmatrix} 4 & -1 & -5 & -11 & 6 \\ -1 & 4 & -2 & -1 & 3 \\ -1 & 0 & 4 & -1 & 5 \\ 6 & 7 & 8 & 10 & 2 \\ 8 & -1 & -1 & 4 & 7 \end{pmatrix}$$

Schemes for solving systems of linear algebraic equations

Column algorithm: example

$$A = \begin{pmatrix} 4 & -1 & -5 & -11 & 6 \\ -0.25 & 4 & -2 & -1 & 3 \\ -0.25 & 0 & 4 & -1 & 5 \\ 1.5 & 7 & 8 & 10 & 2 \\ 2 & -1 & -1 & 4 & 7 \end{pmatrix}$$

Schemes for solving systems of linear algebraic equations

Column algorithm: example

$$A = \begin{pmatrix} 4 & -1 & -5 & -11 & 6 \\ -0.25 & 3.75 & -2 & -1 & 3 \\ -0.25 & -0.07 & 4 & -1 & 5 \\ 1.5 & 2.27 & 8 & 10 & 2 \\ 2 & 0.27 & -1 & 4 & 7 \end{pmatrix}$$

Schemes for solving systems of linear algebraic equations

Column algorithm

- Two **completely different phases** of the column construction.
  - applying L_{j-1}^{-1} (as a forward substitution)
  - computing subdiagonal part of L as a linear combination of a column of A with previously computed columns of L
- They are separated in the Cholesky factorization
- Row algorithm is column algorithm for A^T

Schemes for solving systems of linear algebraic equations

Algorithm

jki *lu decomposition (delayed column dense algorithm)*

```
$l = I_n$, $u = O_n$, $u_{11} = a_{11}$
for j=2:n
    for s=j:n
        $l_{sj-1} = a_{sj-1}/a_{j-1j-1}$
    end
    for k=1:j-1
        for i=k+1:n
            $a_{ij} = a_{ij} - l_{ik} * a_{kj}$
        end
    end
    $u_{1:jj} = a_{1:jj}$
end
```

Schemes for solving systems of linear algebraic equations

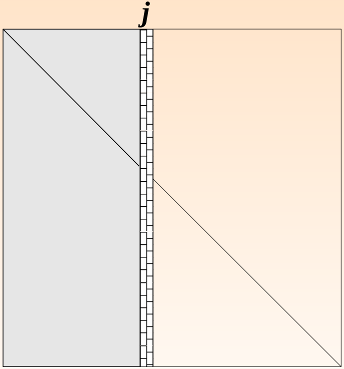
Algorithm

jik *lu decomposition (dot product - based column dense algorithm)*

```
$l = I_n$, $u_{11} = a_{11}$
for j=2:n
    for s=j:n
        $l_{sj-1} = a_{sj-1}/a_{j-1j-1}$
    end
    for i=2:j
        for k=1:i-1
            $a_{ij} = a_{ij} - l_{ik} * a_{kj}$
        end
    end
    for i=j+1:n
        for k=1:j-1
            $a_{ij} = a_{ij} - l_{ik} * a_{kj}$
        end
    end
```

Schemes for solving systems of linear algebraic equations

Depicting column LU factorization



Schemes for solving systems of linear algebraic equations

Algorithm

ikj *lu decomposition (delayed row dense algorithm)*

```
    $l = I_n$
    $u = O_n$
    $u_{11:n} = a_{1,1:n}$
    for i=2:n
        for k=1:i-1
            $l_{ik} = a_{ik}/a_{kk}$
            for j=k+1:n
                $a_{ij} = a_{ij} - l_{ik} * a_{kj}$
            end
        end
        $u_{ii:n} = a_{ii:n}$
    end
```

Schemes for solving systems of linear algebraic equations

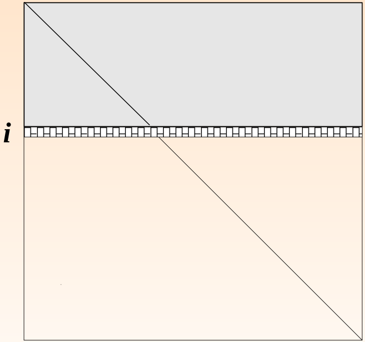
Algorithm

ijk *lu decomposition (dot product - based row dense algorithm)*

```
$l = I_n$, $u = O_n$, $u_{11:n} = a_{11:n}$
for i=2:n
    for j=2:i
        $l_{ij-1} = a_{ij-1}/a_{j-1j-1}$
        for k=1:j-1
            $a_{ij} = a_{ij} - l_{ik} * a_{kj}$
        end
    end
    for j=i+1:n
        for k=1:i-1
            $a_{ij} = a_{ij} - l_{ik} * a_{kj}$
        end
    end
    $u_{i,i:n} = a_{i,i:n}$
end
```

Schemes for solving systems of linear algebraic equations

Depicting row LU factorization



Schemes for solving systems of linear algebraic equations

Bordering algorithm

- Going outside the generic scheme determining i, j a k by

$$a_{i,j} = a_{i,j} - l_{i,k}a_{k,j} \quad \equiv \quad a_{i,j} = a_{i,j} - a_{i,k}a_{k,k}^{-1}a_{k,j}, \tag{38}$$

- Based on $A_{1:k,1:k} =$

$$\begin{pmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k} \\ A_{k,1:k-1} & a_{k,k} \end{pmatrix} = \begin{pmatrix} L_{1:k-1,1:k-1} & 0 \\ L_{k,1:k-1} & 1 \end{pmatrix} \begin{pmatrix} U_{1:k-1,1:k-1} & U_{1:k-1,k} \\ 0 & u_{k,k} \end{pmatrix} \tag{39}$$

- New row $L_{k,1:k-1}$ of L and new column $U_{1:k-1,k}$ from

$$\begin{aligned} L_{k,1:k-1}U_{1:k-1,1:k-1} &= A_{k,1:k-1}, \\ L_{1:k-1,1:k-1}U_{1:k-1,k} &= A_{1:k-1,k}. \end{aligned}$$

- Diagonal entry $u_{k,k}$ of U obtained from

$$u_{k,k} = a_{k,k} - L_{k,1:k-1}U_{1:k-1,k}.$$

Schemes for solving systems of linear algebraic equations

Cholesky factorization

- As expected – more variants as by LU
- In addition, more different forms - not distinguished here if not needed

$$LDL^T = \tilde{L}\tilde{L}^T,$$

- If A is not SPD, $\tilde{L}\tilde{L}^T$ does not need to exist
- LDL^T then can exist but it can be unstable
- An example of LDL^T factorization

$$\begin{pmatrix} -2 & -6 & 4 \\ -6 & -21 & 15 \\ 4 & 15 & -13 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -2 & -1 & 1 \end{pmatrix} \begin{pmatrix} -2 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} 1 & 3 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

Schemes for solving systems of linear algebraic equations

Lemma

Consider one step of the submatrix factorization of an SPD A. Schur complement of A with respect to (positive) $a_{1,1}$ is positive definite.

Proof.

For $\begin{pmatrix}\alpha & z^T\end{pmatrix}^T$ we have $x^T A x =$

$$\begin{pmatrix}\alpha & z^T\end{pmatrix} \begin{pmatrix} a_{1,1} & a_{1,2:n} \\ a_{2:n,1} & A_{2:n,2:n} \end{pmatrix} \begin{pmatrix}\alpha \\ z\end{pmatrix} = \tag{40}$$

$$\alpha^2 a_{1,1} + \alpha a_{1,2:n} z + \alpha z^T a_{2:n,1} + z^T A_{2:n,2:n} z = \tag{41}$$

$$(\alpha + a_{1,1}^{-1} a_{1,2:n} z)^T a_{1,1} (\alpha + a_{1,1}^{-1} a_{1,2:n} z) + z^T (A_{2:n,2:n} - a_{2:n,1} a_{1,1}^{-1} a_{1,2:n}) z \tag{42}$$

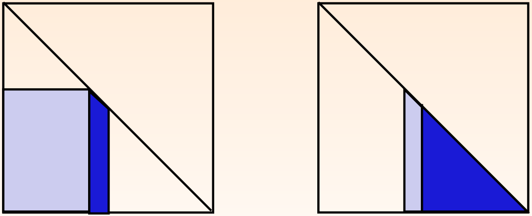
Choosing $z \neq 0$ and setting $\alpha = -a_{1,1}^{-1} a_{1,2:n} z$ we get

$$x^T A x = z^T S z \text{ where } S = A_{2:n,2:n} - a_{2:n,1} a_{1,1}^{-1} a_{1,2:n}.$$

Schemes for solving systems of linear algebraic equations

Schemes for Cholesky factorization

- Left-looking schemes (second part of the column LU)
- Right-looking schemes (just the submatrix scheme)
- The row scheme corresponds to the first part of the Cholesky algorithm.



Submatrix Cholesky

$$
\begin{aligned}
A &= \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3:n} \\ a_{2,1} & a_{2,2} & a_{2,3:n} \\ a_{3:n,1} & a_{3:n,2} & A_{3:n,3:n} \end{pmatrix} \\
&= \begin{pmatrix} \sqrt{a_{1,1}} & 0 & \\ \frac{a_{2,1}}{\sqrt{a_{1,1}}} & \sqrt{a^{(1)}_{2,2}} & \\ \frac{a_{3:n,1}}{\sqrt{a_{1,1}}} & \frac{a^{(1)}_{3:n,2}}{\sqrt{a^{(1)}_{2,2}}} & I_{n-2} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & A^{(2)}_{3:n,3:n} - \frac{a_{3:n,1}a_{1,3:n}}{a_{1,1}} - \frac{a^{(1)}_{3:n,2}a^{(1)}_{2,3:n}}{a^{(1)}_{2,2}} \end{pmatrix} \\
& \quad \begin{pmatrix} \sqrt{a_{1,1}} & \frac{a_{2,1}}{\sqrt{a_{1,1}}} & \frac{a_{1,3:n}}{\sqrt{a_{1,1}}} \\ 0 & \sqrt{a^{(1)}_{2,2}} & \frac{a^{(1)}_{2,3:n}}{\sqrt{a^{(1)}_{2,2}}} \\ & & I_{n-2} \end{pmatrix} \\
&= \begin{pmatrix} l_{1,1} & 0 & 0 \\ l_{2,1} & l_{2,2} & 0 \\ l_{3:n,1} & l_{3:n,2} & I_{n-2} \end{pmatrix} \begin{pmatrix} l_{1,1} & l_{2,1} & l_{1,3:n} \\ 0 & l_{2,2} & l_{2,3:n} \\ 0 & 0 & I_{n-2} \end{pmatrix}
\end{aligned}
$$

Column Cholesky

Algorithm

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

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

$$\begin{pmatrix} t_j \\ \vdots \\ t_n \end{pmatrix} = \begin{pmatrix} a_{jj} \\ \vdots \\ a_{nj} \end{pmatrix} - \sum_{\{k \mid l_{jk} \neq 0\}} l_{jk} \begin{pmatrix} l_{jk} \\ \vdots \\ l_{nk} \end{pmatrix} \tag{43}$$

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

$$\begin{pmatrix} l_{jj} \\ \vdots \\ l_{nj} \end{pmatrix} = \frac{1}{\sqrt{t_j}} \begin{pmatrix} t_j \\ \vdots \\ t_n \end{pmatrix} \tag{44}$$

4. **end** j

Schemes for solving systems of linear algebraic equations

Row Cholesky

Algorithm

Row Cholesky factorization: : $A \to$ **square-root factor** $L = (l_{ij})$**.** *1.* **for** $i = 1 : n$ **do**

2. Solve the triangular system

$$L_{1:i-1,1:i-1} \begin{pmatrix} l_{i1} \\ \vdots \\ l_{i,i-1} \end{pmatrix} = \begin{pmatrix} a_{i1} \\ \vdots \\ a_{i,i-1} \end{pmatrix} \tag{45}$$

3. Compute the diagonal entry $l_{ii} = \sqrt{\left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2\right)}$

4. **end** i

Schemes for solving systems of linear algebraic equations

Dense direct methods: elimination versus **decomposition**

- Householder (end of 1950's, beginning of 1960's): expressing Gaussian elimination as a decomposition
- Various reformulations of the same decomposition: different properties in
  - – sparse implementations
  - – vector processing
  - – parallel implementations
- Six algorithms as for LU, but there are others (bordering, Dongarra-Eisenstat)

Reducibility (again) and block triangular form

- Mentioned above

$$\begin{pmatrix} A_{q_1,q_1} & A_{q_1,q_2} \\ 0 & A_{q_2,q_2} \end{pmatrix}, \tag{46}$$

  where A_{q_1,q_1} and A_{q_2,q_2} are non trivial square matrix blocks (that is, they are of order at least 1).
- An irreducible matrix ($\mathcal{G}(A)$ is strongly connected): having the **strong Hall property**.
- The matrix A is said to have the **Hall property** if any set of its $k \geq 1$ columns has non zeros in at least k rows.
- The Hall property is a necessary condition for A to have full structural rank.
- A has the **strong Hall property** if any set of its $1 \leq k \leq n-1$ columns has non zeros in at least $k+1$ rows.
- An irreducible matrix ($\mathcal{G}(A)$ is strongly connected): having the **strong Hall property**.

Reducibility (again) and block triangular form

Theorem

Given a nonsymmetric matrix A there exists a permutation matrix P such that

$$PAP^T = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,nb} \\ 0 & A_{2,2} & \cdots & A_{2,nb} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{nb,nb} \end{pmatrix}, \tag{47}$$

where the square blocks $A_{i,i}$ on the diagonal are irreducible. The set of blocks $\{A_{i,i}\}$ is uniquely determined (but they may appear on the diagonal in a different order). The order of the rows and columns within each block $A_{i,i}$ may not be unique.

- The **upper block triangular form** (47) is also known as the **Frobenius normal form**.

Reducibility (again) and block triangular form

An example of a matrix that can be symmetrically permuted to block triangular form with $nb = 2$ is given in Figure 3.

$$
\begin{array}{cc}
 & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} &
\left(\begin{array}{cccccc}
* & * &   & * & * &   \\
  & * &   &   & * &   \\
  & * & * &   & * & * \\
  & * &   & * &   &   \\
* &   &   &   & * &   \\
  & * & * &   &   & *
\end{array}\right)
\end{array}
\qquad
\begin{array}{cc}
 & \begin{array}{cccccc} 6 & 3 & 5 & 4 & 1 & 2 \end{array} \\
\begin{array}{c} 6 \\ 3 \\ 5 \\ 4 \\ 1 \\ 2 \end{array} &
\left(\begin{array}{cccccc}
* &   & * &   & * &   \\
* & * & * &   &   & * \\
  &   & * &   & * &   \\
  &   &   & * & * &   \\
  &   & * & * & * & * \\
  &   & * &   &   & *
\end{array}\right)
\end{array}
$$

Figure: *The sparsity patterns of a matrix A (left) and PAP^T in upper block triangular form with diagonal blocks of size 2 and 4 (right).*

- Block triangular form: only diagonal blocks needed to be factorized

Reducibility (again) and block triangular form

- $PAP^Ty = c$, $c = Pb$, $x = P^Ty$.

Algorithm

Solve a sparse linear system in a upper block triangular form
(right-hand side c, solution y)

```
for $i = 1 : nb$ do
    Compute the LU factorization  $A_{i,i} = L_iU_i$
end for
for $i = nb : 1$ do
    $s = c_i$
    for $j = i + 1 : nb$ do
        $s = s - A_{i,j}y_j$
    end for
    Solve $L_iU_iy_i = s$
end for
```

Reducibility (again) and block triangular form

- We say that a matrix has a **full** or **maximum transversal** when the diagonal is free from zeros.
- Any nonsingular matrix A can be permuted using suitable permutation matrices P and Q so that the nonsymmetric permutation PAQ has a full transversal. The converse is (of course) not true.
- If A has a full transversal, a permutation P can be found such that PAP^T has block triangular form and this is identical to finding the strongly connected components (SCCs) of $\mathcal{G}$.
- The **way to find the block triangular form**: next slide.

Reducibility (again) and block triangular form

- **Quotient graph**: assume the vertices V of the directed graph $\mathcal{G}$ are **partitioned** into non-empty subsets V_i in such a way that every vertex is included in exactly one subset. Each vertex i in a quotient graph corresponds to a subset V_i and there is an edge in the quotient graph with endpoints i and j if $\mathcal{G}$ has one or more edges with one endpoint in the subset V_i and the other in V_j.
- **Condensation** of a directed graph is the quotient graph where the strongly connected components (SCCs) form the subsets of the partition, that is, each SCC is contracted to a single vertex.

Theorem

The condensation of a directed graph is a DAG.

Reducibility (again) and block triangular form

We have a block triangular form

- Because any DAG can be topologically ordered, the condensation $\mathcal{G}_C = (\mathcal{V}, \mathcal{E})$ can be topologically ordered.
- Consider an ordering of A induced by this ordering of $\mathcal{G}_C$, that is, a symmetric permutation PAP^T such that any two vertices i and j from different SCCs s_i and s_j of $\mathcal{G}$ satisfy

$$(i, j) \in E(PAP^T) \Rightarrow (s_i, s_j) \in \mathcal{E}.$$

Then PAP^T is of block upper triangular form (47)

Reducibility (again) and block triangular form

Finding strong components

- Here the approach based on the DFS
- The modification with respect to the DFS involves a vector $lowlink$ defined for $u \in V$ as follows:

  $$lowlink(u) = min\{preorder(v) \,|\, (u \to v) \in E\}.$$

  The algorithm performs a DFS, keeping track of two properties for each vertex: when it was encountered (the $index$ (its preorder)) and the lowest index of any vertex reachable from this vertex (the $lowlink$). It pushes vertices onto a stack as it goes and outputs a strongly connected component when it finds a completely processed vertex whose $index$ and $lowlink$ are the same.
- $lowlink(v) = min(lowlink(v), preorder(w))$ (we will see what w is)
- The algorithm is linear in the number of edges and vertices.

Reducibility: Tarjan's SCC algorithm

Algorithm

```
Initialization: $V_v = \emptyset$, $S = ()$, $index = 0$
for each $v \in V$ do
    if $v \notin V_v$ then scomp_step($v$)
    end if
end for
recursive function (scomp_step($v$))
    $V_v = V_v \cup \{v\}$, $index = index + 1$              ▷ Add $v$ to the set of visited vertices
    $preorder(v) = index$, $lowlink(v) = index$, $push(S, v)$      ▷ Put $v$ on the stack
    Set $v = head(S)$                                         ▷ $v$ is the current head of $S$.
    for each $(v \to w) \in E$ do                             ▷ Look in the adjacency list of $v$
        if $w \notin V_v$ then                                ▷ $w$ not yet been visited; recurse on it
            scomp_step($w$)
            $lowlink(v) = min(lowlink(v), lowlink(w))$
        else if $w \in S$ then                                ▷ $w$ is on the stack and hence in current SCC
            $lowlink(v) = min(lowlink(v), preorder(w))$
        end if
    end for
    if $lowlink(v) = preorder(v)$ then
        pop all vertices down to $v$ from $S$ to obtain a new SSC
    end if
```

Reducibility (again) and block triangular form

Five SSCs: $\{s, p, q\}$, $\{t, u, v\}$, $\{x\}$, $\{y\}$ and $\{z\}$. The condensation DAG $\mathcal{G}_C$ also shown; Tarjan's algorithm computes a topological ordering for this graph.

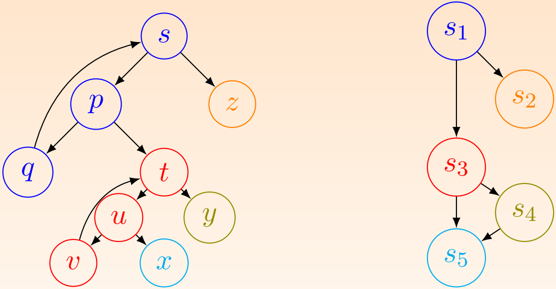


Figure: *A graph to illustrate finding strong components. On the left, the five SSCs are denoted using different colours and on the right is the condensation DAG $\mathcal{G}_C$ formed by the SSCs.*

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Definition of blocks

- matrix → graph
- **blocks are defined as follows**
  - 1: vertices that are connected to each other
  - 2: vertices that have the same sets of neighbors

Getting blocks: Approach by Ashcraft (1995)

The strategy of the first approach

1. Number graph nodes/vertices (use numbers as their labels)
2. Compute vertex checksums:

$$chksum = \sum_{\{u,v\} \in E} w$$

3. Sort vertices by their checksums: in O(|E|+|V|log(|V|)) time
4. Different checksum **means** different block
5. First **tie-breaking** rule: if $chksum(u) = chksum(v)$: compare $|adj(u)|$ and $|adj(v)|$
6. Second **tie-breaking** rule: compare adjacency sets of u and v (the most time consuming)

Getting blocks: Approach by Saad (2005)

The strategy of the second approach

- First idea: it is possible to use **different** (scaled) checksums (hash functions)
- Example where the previous approach is not the best: **some blocks** seem to be here

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

Getting blocks

The strategy of the second approach (continued)

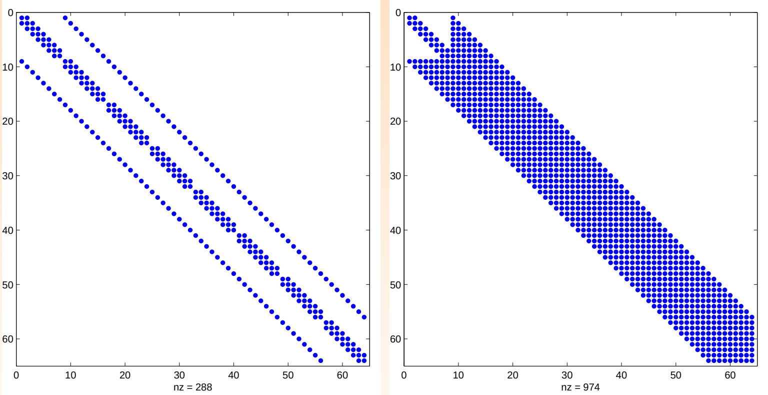
- Cosine algorithm: comparison of **angles** between matrix rows
- Getting them: multiply AA^T
- How it could be done cheaply?: see above: **apply our matmats**
- Then grouping according to these products!

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

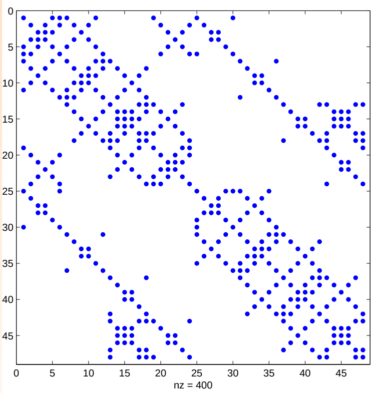
Sparse factorizations

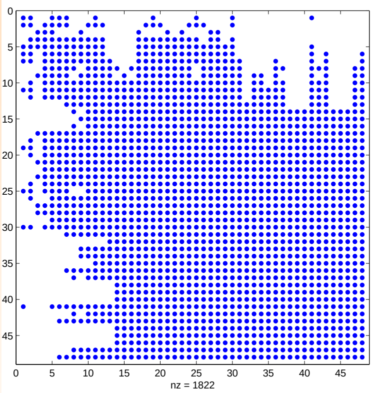
Direct decomposition may fill



Sparse factorizations

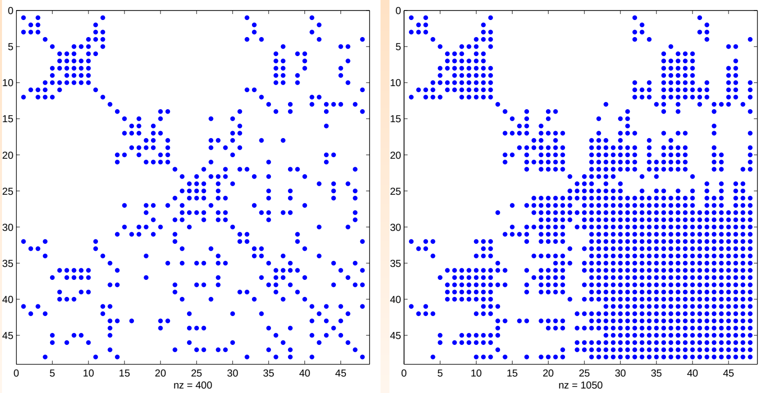
Direct decomposition may fill





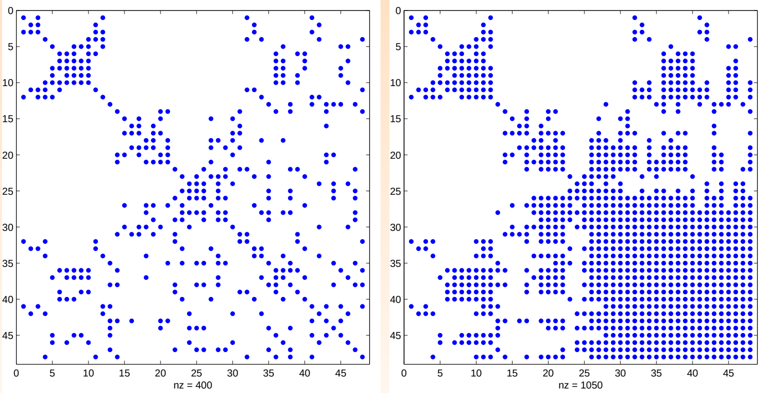
Sparse factorizations

Direct decomposition may fill



Sparse factorizations

Direct decomposition may fill



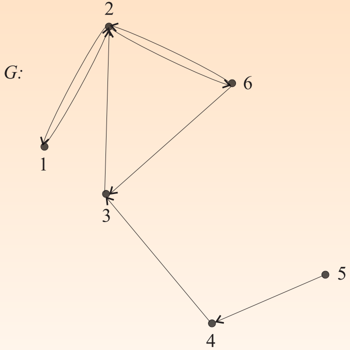
- Need to describe the fill-in: 1) describe it 2) avoid it
- Need to exploit the fill-in structure algorithmically
- Or ... we can cut the fill-in and perform an incomplete process ... later

Sparse factorizations

Repeat matrix → description

- Combinatorial structure of zeros and nonzeros → graphs

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



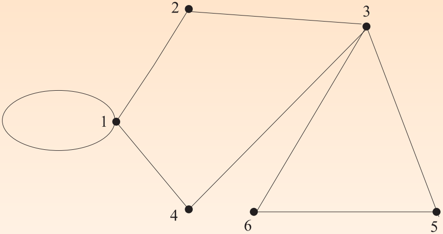
- Fill-in changes during the decomposition: dynamic description
- Data structures, implementation with respect to the architecture

Sparse factorizations

Repeat matrix → description

- Combinatorial structure of zeros and nonzeros → graphs

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



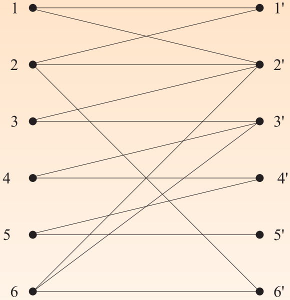
- Fill-in changes during the decomposition: dynamic description
- Data structures, implementation with respect to the architecture

Sparse factorizations

Repeat matrix → description

- Combinatorial structure of zeros and nonzeros → graphs

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



- Fill-in changes during the decomposition: dynamic description
- Data structures, implementation with respect to the architecture

Sparse factorizations

Sparsity structure changes during factorization

- Arrow matrix - original matrices: example showing how **bad** the fill-in problem can be

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

Sparse factorizations

Sparsity structure changes during factorization

- Arrow matrix - structure after elimination

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

Sparse factorizations

Sparsity structure changes during factorization

- Arrow matrix - structure after elimination

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

- Fill-in description and ways to avoid it must capture it dynamically!

Sparse factorizations

Dynamic development of the fill-in (**generally nonsymmetric**)

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

Sparse factorizations

Dynamic development of the fill-in (**generally nonsymmetric**)

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

elimination of the first row and column

Sparse factorizations

Dynamic development of the fill-in (**generally nonsymmetric**)

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

elimination of the second row and column

Sparse factorizations

Dynamic development of the fill-in (**generally nonsymmetric**)

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

elimination of the third row and column

Sparse factorizations

Dynamic development of the fill-in (**generally nonsymmetric**)

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

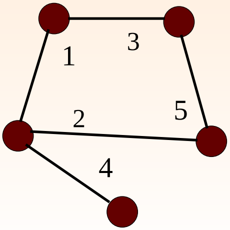
elimination of the third row and column

- Formal description: the sequence of **elimination matrices** $A^{(1)}, A^{(2)}, \ldots$
- And the sequence of their graphs

Sparse factorizations

Dynamic development of the fill-in (**symmetric**) - easier to track

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

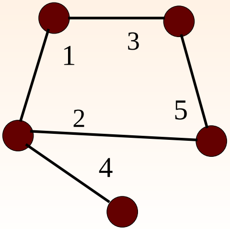


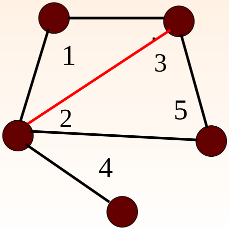
Sparse factorizations

Dynamic development of the fill-in (**symmetric**) - easier to track

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

elimination of the first row and column



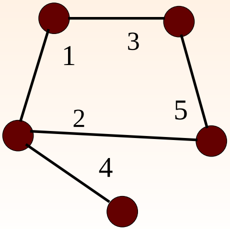


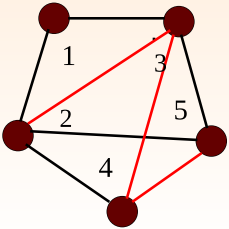
Sparse factorizations

Dynamic development of the fill-in (**symmetric**) - easier to track

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

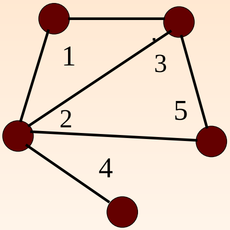
elimination of the second row and column



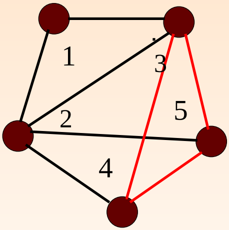


Sparse factorizations

Dynamic development of the fill-in (**symmetric**) - easier to track



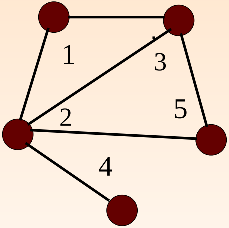
after 1st step



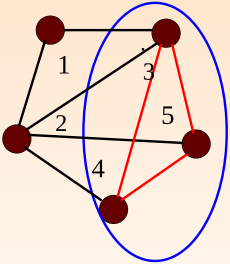
after second step

Sparse factorizations

Dynamic development of the fill-in (**symmetric**) - easier to track



after 1st step



after second step

Sparse factorizations

Another demonstration of the symmetric filling process

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & * & * & & * & * \\ 2 & * & * & * & & \\ 3 & & * & * & & * \\ 4 & * & & & * & \\ 5 & * & & * & & * \end{array} \quad \rightarrow \quad \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & * & * & & * & * \\ 2 & * & * & * & f & f \\ 3 & & * & * & & * \\ 4 & * & f & & * & f \\ 5 & * & f & * & f & * \end{array}$$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & * & * & & * & * \\ 2 & * & * & * & f & f \\ 3 & & * & * & f & * \\ 4 & * & f & f & * & f \\ 5 & * & f & * & f & * \end{array}$$

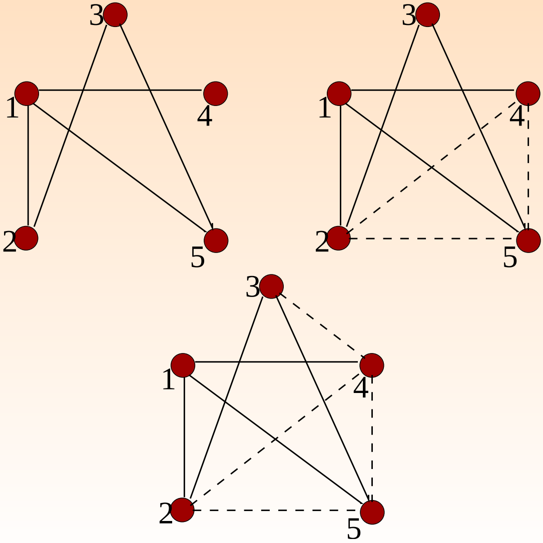
Sparse factorizations

Dynamic development of the fill-in: summary

- **Symmetric**: Elimination step induces a clique in the graph model
- **Nonsymmetric**: Elimination step induces a more general structure that can be considered as a fully dense bipartite subgraph (after permutation)

Local description of fill-in

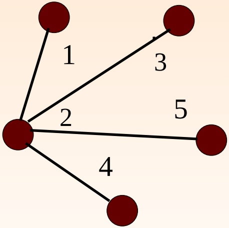
The filling process (in graph model)

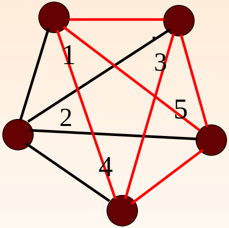


Sparse factorizations

Memory considerations

- How should be $\mathcal{E}$ stored in the computer?
- Storing clique (symmetric case) instead of a subgraph → complexity?
- A clique can be stored just implicitly - storing entries that caused it!
- Nonsymmetric case is more complicated (pivoting may be needed)
- **symmetric example**: Recursive storing of the cliques caused by the elimination needed. Go implicit!





{{1,3},{1,4},{1,5}, {3,4}, {3,5},{4,5}} → {1,3,4,5}

- Still: too local, row/column character of the decomposition not used

Local description of fill-in

- No-cancellation assumption (always assumed throughout) $\Rightarrow$

$$\mathcal{S}(A) \subseteq \mathcal{S}(L + L^T), \qquad \mathcal{S}(A) \subseteq \mathcal{S}(L + U) \tag{49}$$

- Where fill-in can be expected (works also for nonsymmetric A)

$$\begin{array}{cccc}
 & k & j & i \\
k & \ast & \ast & \ast \\
j & \ast & \ast & \bigcirc \\
i & \ast & \bigcirc & \ast
\end{array}$$

Local description of fill-in

Fill-in and the Schur complement ($k**$ schemes)

$$A^{(k)} = A^{(k-1)} - \begin{array}{cc} & \begin{array}{cccccc} 1 & \dots & k & k+1 & \dots & n \end{array} \\ \begin{array}{c} 1 \\ \vdots \\ k \\ k+1 \\ \vdots \\ n \end{array} & \left(\begin{array}{cccccc} & & & & & \\ & & & & & \\ & & & & & \\ & & & \frac{a_{k+1,k}^{k-1} a_{k,k+1}^{k-1}}{a_{k,k}^{k-1}} & \dots & \frac{a_{k+1,k}^{k-1} a_{k,n}^{k-1}}{a_{k,k}^{k-1}} \\ & & & \vdots & \ddots & \vdots \\ & & & \frac{a_{n,k}^{k-1} a_{k,k+1}^{k-1}}{a_{k,k}^{k-1}} & \dots & \frac{a_{n,n}^{k-1} a_{n,n}^{k-1}}{a_{k,k}^{k-1}} \end{array} \right) \end{array} \tag{50}$$

- Works also in the nonsymmetric case.

Local description of fill-in

Local description of the fill-in

(Notation uses elimination matrices $A \equiv A^{(0)}, A^{(1)}, \ldots$)

Lemma

(Fill-in Lemma) *Let* $i, j, k \in \{1, \ldots, n\}, k < \min\{i, j\} \leq n$. *Then*

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

Only sparsity pattern of $L + L^T$

Lemma

(Graph form of fill-in Lemma) *Let* $i, j \in \{1, \ldots, n\}, k < \min\{i, j\}$. *Then*

$$\{i, j\} \in E(F) \Longleftrightarrow \{i, j\} \in E(A) \vee (\{i, k\} \in E(F) \wedge \{k, j\} \in E(F)),$$

where $G(F) = (V, E(F))$ *represents graph of the filled matrix* $F = L + L^T$.

Global description of fill-in: Cholesky

Theorem

(Cholesky fill-in theorem) *Let* $i, j, k \in \{1, \ldots, n\}, k < \min\{i, j\} \leq n$*. Then* $a_{ij}^{(k)} \neq 0$ *iff there is in the undirected graph model* G *of* A *a path*

$$i \overset{G}{\Leftrightarrow} j,$$

such that all its vertices not equal to i *or* j *are smaller than* $\min\{i, j\}$ *(fill-in path).*

Theorem

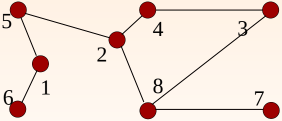
(Graph form of the Cholesky fill-in theorem) *Let* $i, j \in \{1, \ldots, n\}$*. Then* $l_{ij} \neq 0$ *(*$\{i, j\} \in E(F)$*) iff there is a* **fill-in** *path in* $G(A)$

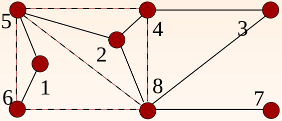
$$i \overset{G}{\Leftrightarrow} j.$$

Global description of fill-in: Cholesky

Demonstration of the fill-in theorem for Cholesky

$$
\begin{array}{cc}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} &
\begin{pmatrix}
* & & & & * & * & & \\
 & * & & * & * & & & * \\
 & & * & * & & & & * \\
 & * & * & * & & & & \\
* & * & & & * & & & \\
* & & & & & * & & \\
 & & & & & & * & * \\
 & * & * & & & & * & *
\end{pmatrix}
\end{array}
\qquad
\begin{array}{cc}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} &
\begin{pmatrix}
* & & & & * & * & & \\
 & * & & * & * & & & \\
 & & * & * & & & & * \\
 & * & * & * & f & & & f \\
* & * & & f & * & f & & f \\
* & & & & f & * & & f \\
 & & & & & & * & * \\
 & * & * & f & f & f & * & *
\end{pmatrix}
\end{array}
\tag{51}
$$





$$l_{86} :\ 8 \leftrightarrow 2 \leftrightarrow 5 \leftrightarrow 1 \leftrightarrow 6$$

Global description of fill-in: Cholesky and LU

Theorem

Let A have a **symmetric** *sparsity pattern and let $A = LDL^T$. Let $F = L + L^T$. Then $f_{ij} \neq 0$ $(i \neq j)$ if and only if there is a fill-path $i \underset{min}{\overset{\mathcal{G}}{\Longleftrightarrow}} j$.*

Let A have a **nonsymmetric** *sparsity pattern and let $A = LU$. Let $F = L + U$. Then $f_{ij} \neq 0$ $(i \neq j)$ if and only if there is a fill-path $i \underset{min}{\overset{\mathcal{G}}{\Longrightarrow}} j$. The fill-path may not be unique.*

- Still an implicit description.

Global description of fill-in: LU

Demonstration of the nonsymmetric fill-in theorem

$$\begin{array}{c@{}c}
 & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} &
\left(\begin{array}{cccccc}
* &   & * &   & * &   \\
* & * &   &   & * & * \\
  &   & * & * &   &   \\
* &   & * & * &   &   \\
  &   & * &   & * &   \\
* & * &   &   &   & *
\end{array}\right)
\end{array}
\qquad
\begin{array}{c@{}c}
 & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} &
\left(\begin{array}{cccccc}
* &   & * &   & * &   \\
* & * & f &   & * & * \\
  &   & * & * &   &   \\
* &   & * & * & f &   \\
  &   & * & f & * &   \\
* & * & f & f & f & *
\end{array}\right)
\end{array}
\tag{52}$$

For the nonsymmetric matrix in the figure, entry $(6, 4)$ of L is a fill-in entry because there is a fill-path $6 \xRightarrow[min]{\mathcal{G}} 4$ given by $6 \rightarrow 1 \rightarrow 3 \rightarrow 4$.

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Sparse Cholesky factorization - components

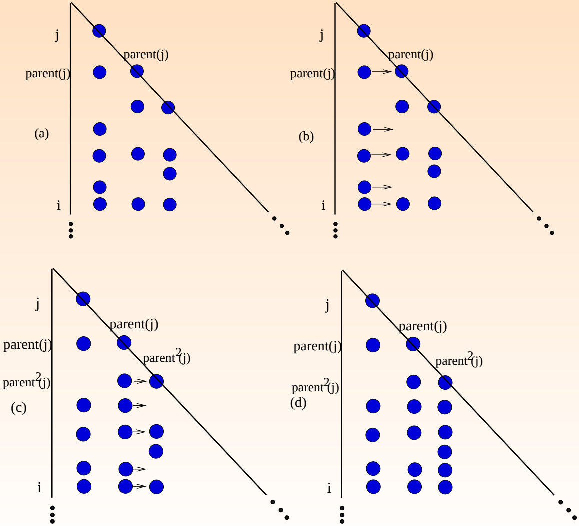
- Consider Cholesky factorization (symmetric, positive definite)
- **A way to see the fill-in: replication of column structures**
- Define terminology (of rooted trees) first

Definition

Denote by $parent(j)$ for $j \in \{1, \ldots, n\}$ the row index of the **first subdiagonal (nonzero) entry** in column j of L. If there is no such entry, set $parent(j) = 0$. Denote further $parent^2(j) \equiv parent(parent(j))$, $parent^3(j) \equiv parent(parent(parent(j)))$ etc.

Sparse Cholesky factorization - components

Fill-in as replication of column structures II



Sparse Cholesky factorization - components

Fill-in as replication of column structures IIa

- From the depiction we can see the following:

Observation

$$\mathcal{S}(L_{j+1:n,j}) \subseteq \mathcal{S}(L_{parent(j):n,parent(j)}).$$

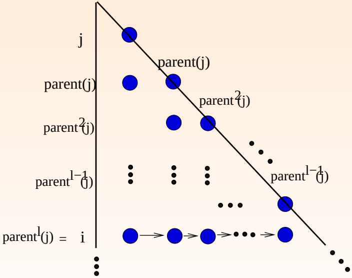
- Note that we use the non-cancellation assumption

Sparse Cholesky factorization - components

Fill-in as replication of column structures III

Theorem

Let $l_{ij} \neq 0$ for $j < i \leq n$. Then there is an integer $p > 0$ such that $parent^p(j) = i$. In addition, $l_{is} \neq 0$ for $s = j$, $parent(j)$, $parent^2(j) \ldots, parent^p(j)$.



- Replication of column structures → replication of nonzeros in a row.

Sparse Cholesky factorization - components

Fill-in as replication of column structures V

Theorem

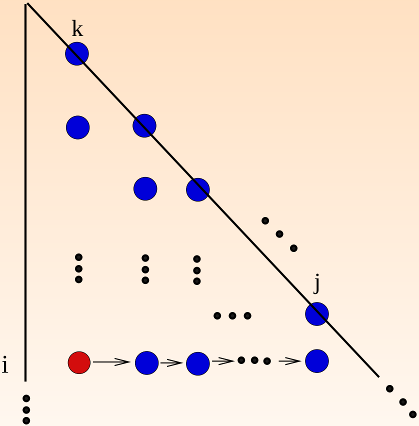
*(***Extended fill-in theorem using column structure replication***) Let $a_{ij} = 0$ for $j < i \le n$. Then $l_{ij} \ne 0$ iff there exists $k < j$, for which $a_{ik} \ne 0$ and there exists a $p > 0$ such that*

$$parent^p(k) = j \quad .$$

- The theorem says that at each **row** should be a starter: **row replication should start somewhere**
- This is a basis to easy determination of sparsity patterns of **rows** of L
- Next slide demonstrates the principle.

Sparse Cholesky factorization - components

Fill-in as replication of column structures VI



- Is there a simpler graph structure that describes the fill-in?
- Yes, the directed acyclic graph based on the mapping **parent**, called the **elimination tree**.

Sparse Cholesky factorization - components

Elimination tree

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

Sparse Cholesky factorization - components

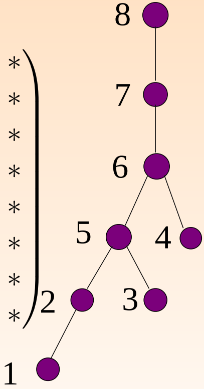
Elimination tree

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

Sparse Cholesky factorization - components

Elimination tree

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



- the elimination tree in computer: just one vector **parent**

Sparse Cholesky factorization - components

Definition

Elimination tree $T(A) = (V, E(A))$ of a symmetric matrix (not necessarily SPD) $A \in \Re^{n \times n}$ such that for some matrix values the Cholesky factorization $A = LDL^T$ exists is the graph induced by the edge set

$$E(A) = \{(i,j) \mid i = \min\{k \mid k > j \wedge l_{kj} \neq 0\}.$$

Lemma

If A is irreducible (not reducible), $T(A)$ is **tree***. That is, the symmetrization of $T(A)$ is connected and acyclic. Otherwise, elimination tree is a* **forest** *(not connected).*

Sparse Cholesky factorization - components

Elimination tree

- Remind: the elimination tree is defined via the filled graph (graph with fill-in, $G(F)$)
- In practice, it has to be computed from the original matrix A (we do not have the filled matrix)
- Elimination tree (or its variations) is one of the most fundamental tree structure connected to the Cholesky factorization.

Sparse Cholesky factorization - components

Some terminology related to the elimination tree

Definition

Reminder: **Leafs** of a rooted tree are its vertices that do not have descendants.

Definition

Subtree $T(j)$ of a directed acyclic graph T in vertex j is its subgraph induced by the set $desc_T(j)$ of all descendants of j in T. $T(j)$ is a rooted tree with the root j. **Size** of $T(j)$ denoted by $|T(j)|$ is the number of vertices of $T(j)$.

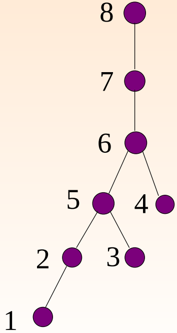
Sparse Cholesky factorization - components

Elimination tree: simple properties

Lemma

If i is an ancestor of $j \neq i$ in the elimination tree $T(A)$ then $i > j$.

- Straightforward to see from the definition



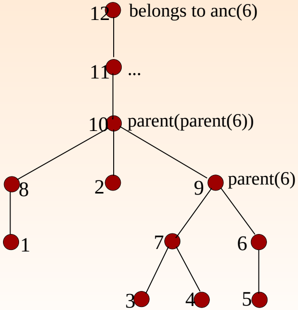
Sparse Cholesky factorization - components

Elimination tree: using the rooted tree terminology

Observation

For i, j, $i > j$ *of* $V(T)$ *and an integer* $p > 0$ *we have*

$$i = anc_T(j) \Leftrightarrow j \in desc_T(i) \Leftrightarrow (\exists p > 0)(parent^p(j) = i).$$



Sparse Cholesky factorization - components

Elimination tree: construction

```
for $i = 1$ to $n \equiv |V|$ do
    $parent(i) = 0$
    for $k$ such that $k \in adj(i) \wedge k < i$ do
        $j = k$
        while $(parent(j) \neq 0 \wedge parent(j) \neq i)$ do
            $j = parent(j)$
        end while
        if $parent(j) = 0$ then $parent(j) = i$
    end $k$
end $i$
```

Sparse Cholesky factorization - components

Lemma

Consider the construction and its i-th step for $i = 1, \ldots, n$. At its end we have the elimination tree of $T(A_{1:i,1:i})$.

- Outer loop of the construction: scanning rows
  - Either the first nonzero of a column is found and a new root of the component of the elimination tree is set up.
  - Or the replication principle induces **moving** current row index (vertex) **up the tree component** toward its ancestors.

Sparse Cholesky factorization - components

Elimination tree: construction

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

Sparse Cholesky factorization - components

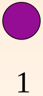
Elimination tree: construction

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

Sparse Cholesky factorization - components

Elimination tree: construction

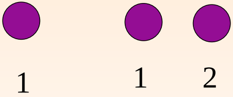
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$



Sparse Cholesky factorization - components

Elimination tree: construction

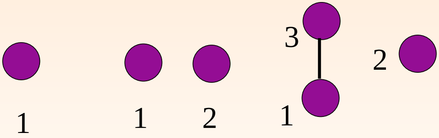
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$



Sparse Cholesky factorization - components

Elimination tree: construction

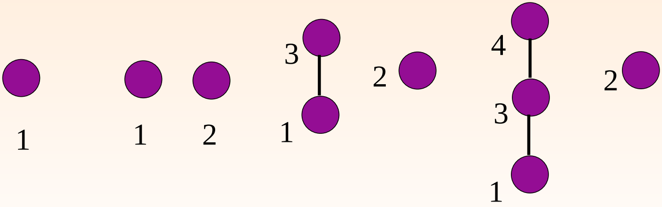
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$



Sparse Cholesky factorization - components

Elimination tree: construction

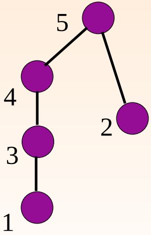
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & {\color{red}f} & * \\ * & * & * & {\color{red}f} & * & {\color{red}f} \\ & * & & * & {\color{red}f} & * \end{pmatrix}$$



Sparse Cholesky factorization - components

Elimination tree: construction

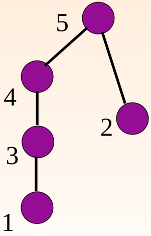
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$

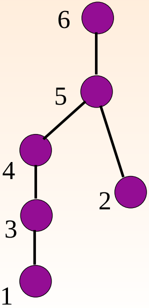


Sparse Cholesky factorization - components

Elimination tree: construction

$$\begin{pmatrix} \ast & & \ast & & \ast & \\ & \ast & & & \ast & \ast \\ \ast & & \ast & \ast & \ast & \\ & & \ast & \ast & & \ast \\ \ast & \ast & \ast & & \ast & \\ & \ast & & \ast & & \ast \end{pmatrix} \quad \begin{pmatrix} \ast & & \ast & & \ast & \\ & \ast & & & \ast & \ast \\ \ast & & \ast & \ast & \ast & \\ & & \ast & \ast & f & \ast \\ \ast & \ast & \ast & f & \ast & f \\ & \ast & & \ast & f & \ast \end{pmatrix}$$





Sparse Cholesky factorization - components

Elimination tree construction: problems with long dependency chains

$$\begin{array}{c|cccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & * & * & * & * & * & * & * & * \\
2 & * & * & & & & & & \\
3 & * & & * & & & & & \\
4 & * & & & * & & & & \\
5 & * & & & & * & & & \\
6 & * & & & & & * & & \\
7 & * & & & & & & * & \\
8 & * & & & & & & & *
\end{array} \tag{53}$$

- Elimination tree is determined by the directed edges

$$parent(i) = i + 1,\; i = 1, \ldots, n.$$

- But the search starts always from a leaf and goes up the tree: algorithm may not be efficient

Sparse Cholesky factorization - components

Elimination tree: an improved construction

```
for $i = 1$ to $n$ do
    $parent(i) = 0$; $ancestor(i) = 0$
    for $k$ such that $k \in adj(i) \wedge k < i$ do
        $j = k$
        while $(ancestor(j) \neq 0 \wedge ancestor(j) \neq i)$ do
            $j = ancestor(j)$; $ancestor(j) = i$; $j = t$
        end while
        if $ancestor(j) = 0$ then $parent(j) = i$; $ancestor(j) = i$
    end $k$

end $i$
```

- Complexity $O(|E(G(A))| \log_2 |V|)$. Can be further reduced by other **general tree techniques up to close $O(|E(G(A))|)$.**

Sparse Cholesky factorization - components

Theorem

*(***Locality property of the elimination tree***) Vertex i is an ancestor of j, $i > j$ in the elimination tree $T(A)$ iff there is in $G(\{1,\dots,i\})$ an undirected path*

$$j \overset{G(A)}{\underset{\{1,\dots,i\}}{\Longleftrightarrow}} i.$$

- How we can see this: replication in row i is always implied by vertices smaller than i

Sparse Cholesky factorization - components

Corollary

Ancestor i *of* j, $i > j$ *in elimination tree* $T(A)$ *is the father of* j *(*$i = parent(j)$*) iff* j *is the smallest vertex of the component of* $G(\{1,\ldots,i\}) \setminus \{i\}$ *that contains an undirected path*

$$j \overset{G(A)}{\underset{\{1,\ldots,i\}}{\Longleftrightarrow}} i.$$

Sparse Cholesky factorization - components

Elimination tree: let us repeat our motivation and goals

- How can be the fill-in described (and avoided ...)?
- How should be data structures set up?
- We have the elimination tree but this may be not enough.

Sparse Cholesky factorization - components

Row structures of L

Theorem

*(***Extended fill-in theorem formulated using the elimination tree***)* *Consider a computed Cholesky factor, no-cancellation. For i, j, $i > j$ we have $l_{ij} \neq 0$ iff j is an ancestor of some k in T for which $a_{ik} \neq 0$.*

- Theorem describes fill-in in the i-th row of L.
- Some k must precede i in the elimination tree (**starter**)
- Then all ancestors j correspond to nonzeros in the i-th row

Sparse Cholesky factorization - components

Pruned and row subtrees of $T(A)$

Definition

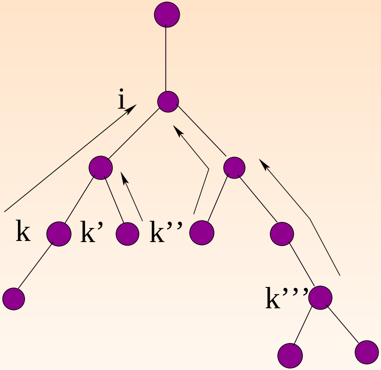
Subtree $T_p(i)$ of the elimination tree T is called the **pruned subtree** of T at the vertex i if it is a rooted tree with the root i, subset of vertices from $T(i)$ and such that for each leaf k of $T_p(i)$ all vertices on the directed path from i to k belong to $T_p(i)$. $T_p(i)$ is the subgraph of T (and also of $T(i)$) induced by this set of vertices.

Corollary

Let $n \geq i > j \geq 1$. Consider $T_p(i)$ of T where $j \in V(T_p(i))$ iff (i) either $a_{ij} \neq 0$, (ii) or $j \in anc_{T(i)}(k)$ for some k where $a_{ik} \neq 0$. This pruned subtree we will denote by $T_r(i)$ and call i-th **row subtree** *of T. Vertices $row_L(i) \cup \{i\}$ are then exactly vertices of $T_r(i)$.*

Sparse Cholesky factorization - components

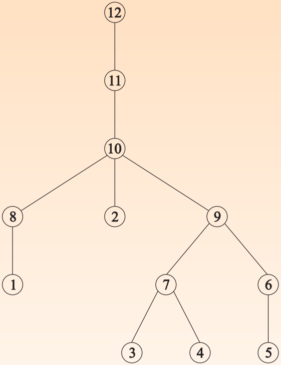
Row subtrees: demonstration



Sparse Cholesky factorization - components

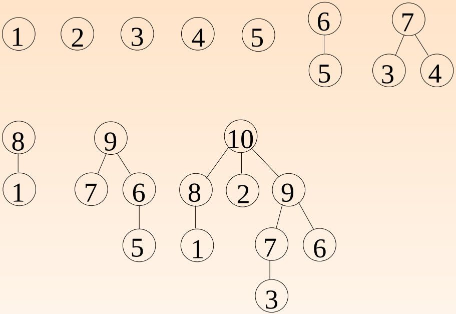
Row subtrees

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



Sparse Cholesky factorization - components

Row subtrees



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

Sparse Cholesky factorization - components

Row structures of L, row subtrees of $T(A)$

- **Just remind:** vertices in the row subtree rooted at i correspond to nonzeros in a **row** of L

$$row_L(j) \equiv \mathcal{S}(L_{j,1:j-1}) = \{k \mid k < j,\ l_{jk} \neq 0\},\ 1 \leq j \leq n. \tag{54}$$

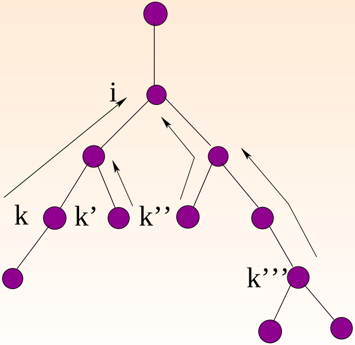
- Formally, we do not count diagonal nonzeros

Sparse Cholesky factorization - components

Row counts: simple algorithm

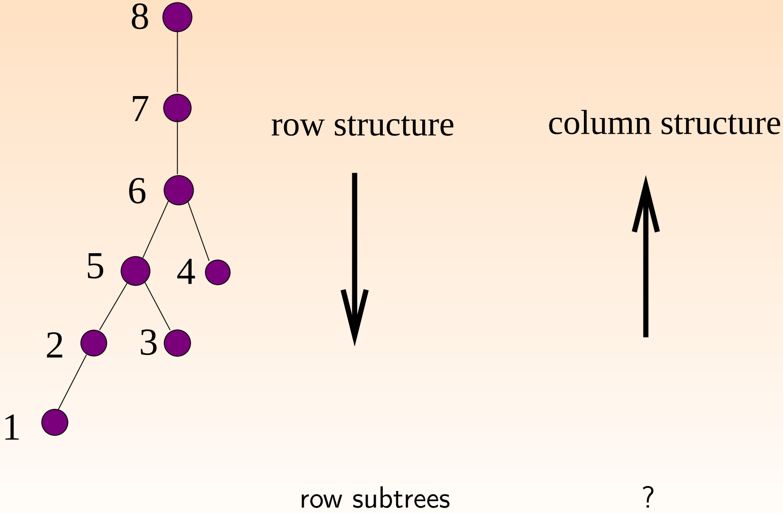
```
initialize all $colcounts$ to 1
for $i = 1$ to $n$ do
    $rowcount(i) = 1$
    $mark(i) = i$
    for $k$ such that $k < i \wedge a_{ik} \neq 0$ do
        $j = k$
        while $mark(j) \neq i$ do
            $rowcount(i) = rowcount(i) + 1$
            $colcount(j) = colcount(j) + 1$
            $mark(j) = i$
            $j = parent(j)$
        end while
    end $k$

end $i$
```



Sparse Cholesky factorization - components

It would be nice to know column structures of L as well



row structure

column structure

row subtrees

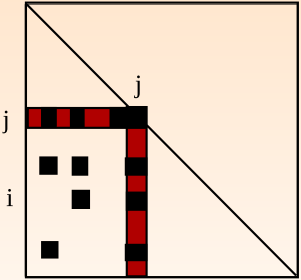
?

Sparse Cholesky factorization - components

Column structures of L

Lemma

Column j is updated by such columns k satisfying $l_{jk} \neq 0$.



Lemma

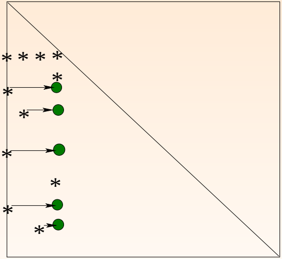
$Struct(L_{*j}) = Struct(A_{*j}) \ \cup \ \bigcup_{k, l_{jk} \neq 0} Struct(L_{*k}) \setminus \{1, \dots, j-1\}.$

Sparse Cholesky factorization - components

Column structures of L

Lemma

$Struct(L_{*j}) = Struct(A_{*j}) \ \cup \ \bigcup_{k, l_{jk} \neq 0} Struct(L_{*k}) \setminus \{1, \ldots, j-1\}.$

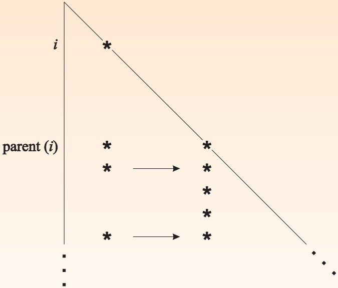


Sparse Cholesky factorization - components

Column structures of L

Lemma

$Struct(L_{*j}) \setminus \{j\} \subseteq Struct(L_{*parent(j)})$



$$Struct(L_{*j}) = Struct(A_{*j}) \cup \bigcup_{\{i|j=parent(i)\}} Struct(L_{*i}) \setminus \{1,\ldots,j-1\}.$$

Sparse Cholesky factorization - components

Column structures of L

$$col_L(j) \equiv \mathcal{S}(L_{j+1:n,j}) = \{k \mid k > j,\ l_{kj} \neq 0\},\ 1 < j \leq n. \tag{55}$$

- Rewritten replication

Lemma

$$col_L(j) \subseteq col_L(parent(j)) \cup \{parent(j)\}. \tag{56}$$

- Summarizing theorem with a **recursion behind**

Theorem

Sparsity structure of a column j of the Cholesky factor L of A is given by the **adjacency set** *of the vertices of $T(j)$ of the elimination tree T. In the other words,*

$$col_L(j) = adj_{G(A)}(T(j)). \tag{57}$$

Sparse Cholesky factorization - components

Column structures

Once more, formulated as a theorem.

Theorem

$$col_L(j) = \left(adj_{G(A)}(j) \cup \bigcup_{\{i|j=parent(i)\}} col_L(i) \right) \ \setminus \ \{1, \ldots, j\}. \tag{58}$$

Sparse Cholesky factorization - components

Finding column structures: (called symbolic factorization in a restricted sense

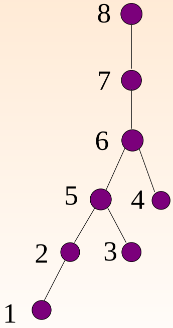
Algorithm

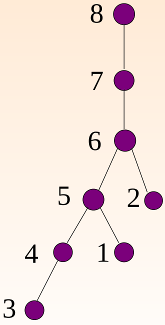
```
for $j = 1$ to $n$ do
    $son(j) = \emptyset$
end $j$
for $j = 1$ to $n$ do
    $col_L(j)$ = set of indices of $adj_G(j) \ \setminus \ \{1, \ldots, j-1\}$
    for $k \in son(j)$ do
        $col_L(j) = col_L(j) \cup col_L(k) \ \setminus \ \{j\}$
    end $k$
    if $col_L(j) \neq 0$ then
        $p = \min\{i \mid i \in col_L(j)\}$
        $son(p) = son(p) \cup \{j\}$
    end if
end $j$
```

Sparse Cholesky factorization - components

Reorderings/renumberings induced by the elimination tree

- Initial numbering of nodes of the elimination tree is **topological.**
- Observe two different topological renumberings. Is some of them preferable?





Sparse Cholesky factorization - components

Reorderings/renumberings induced by the elimination tree

- Topological reorderings are **equivalent from the point of view of fill-in**.

Theorem

Let A be a symmetric matrix and $T(A)$ is its elimination tree. Let P be a compatible permutation matrix where the relation between vertices of P^TAP and $T(A)$ is given by a topological renumbering $\alpha : V(A) \leftrightarrow V(P^TAP)$. Then the **filled** *graphs of Cholesky factorizations of A and P^TAP are isomorphic.*

Sparse Cholesky factorization - components

Reorderings/renumberings induced by the elimination tree

Definition of postordering: the importance of locality

Definition

Topological numbering α of a rooted tree $T = (V = \{1, \ldots, n\}, E)$ is called **postordering** if sets of vertices of **every its subtree**

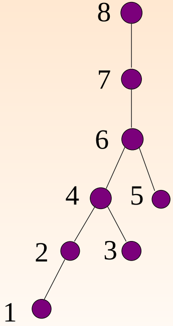
$$T(j), \quad \text{for} \quad j = 1, \ldots, n$$

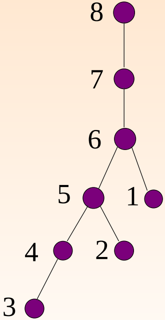
contain values of an **interval** of natural numbers from $\{1, \ldots, n\}$.

Sparse Cholesky factorization - components

Reorderings/renumberings induced by the elimination tree

- Two different postorderings





Sparse Cholesky factorization - components

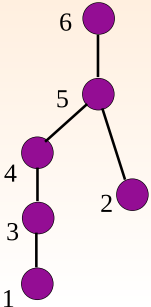
Again: postordering = topological + labels in subtrees form intervals

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

Sparse Cholesky factorization - components

Again: postordering = topological + labels in subtrees form intervals

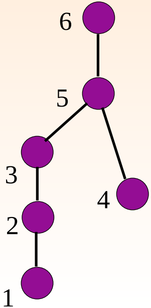
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & {\color{red}f} & * \\ * & * & * & {\color{red}f} & * & {\color{red}f} \\ & * & & * & {\color{red}f} & * \end{pmatrix}$$



Sparse Cholesky factorization - components

Again: postordering = topological + labels in subtrees form intervals

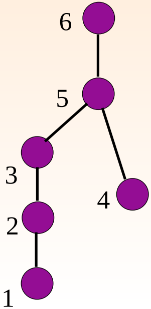
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$



Sparse Cholesky factorization - components

Again: postordering = topological + labels in subtrees form intervals

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

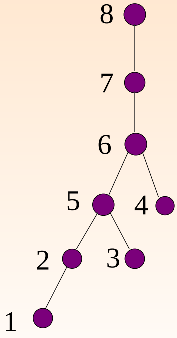


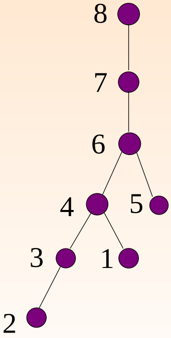
$$\rightarrow \begin{pmatrix} * & * & & & * & \\ * & * & * & & * & \\ & * & * & & f & * \\ & & & * & * & * \\ * & * & f & * & * & f \\ & & * & * & f & * \end{pmatrix}$$

Sparse Cholesky factorization - components

Postordering: why?

- Postordering is necessary for **locality**: efficient exploiting memory hierarchies, paging environment, crucial for multifrontal methods, efficient computation of factor row and column counts (different than mentioned) etc.

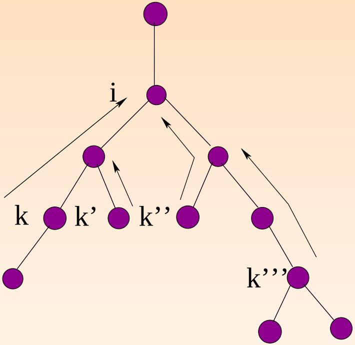




Postordered tree

Sparse Cholesky factorization - components

Row counts: more sophisticated algorithm: the idea



- Needed: fast algorithm to determine **junctions** of branches in the elimination tree,
- and fast algorithm to find **leaves** of the elimination tree.
- **Can be done** by traversing the postordered elimination tree.
- The complexity can be then nearly linear in m.
- Can be done similarly for column counts

Sparse Cholesky factorization - components

Elimination tree leaves

Theorem

Let the vertices of an elimination tree T of A are postordered. Consider a row index i, $1 < i \le n$. Denote also

$$adj_{G(A)}(i) \cap \{1, \ldots, i-1\} = \{c_1, \ldots, c_s\},\ 0 < c_1 < \ldots < c_s < i,\ s \ge 1.$$

Then c_t pro $t \in \{1, \ldots, s\}$ is a leaf of $T_r(i)$ of the elimination tree T iff either $t = 1$ or

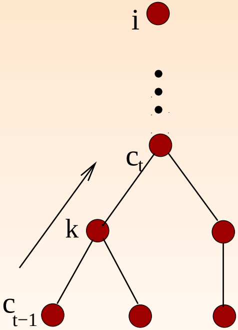
$$t > 1\ \wedge\ c_{t-1} \notin T(c_t). \tag{59}$$

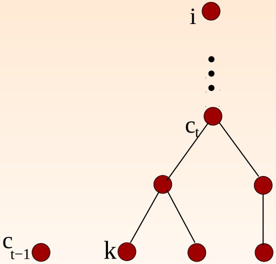
- Case $t = 1$ is clear
- $t > 1$: $c_{t-1} \in T(c_t) \wedge c_t$ is a leaf of $T_r(i) \Rightarrow l_{ik}$ for all ancestors k of c_{t-1}. Postordering implies that $c_{t-1} \le k < c_t$ and these ancestors are then also in $T_r(i)$. And this is a contradiction.
- Proof of the reverse implication similar.

Sparse Cholesky factorization - components

Elimination tree leaves: figures to demonstrate the situation

- Figure on the left demonstrates the described implication proof.





Sparse Cholesky factorization - components

Elimination tree leaves

- Corollary: closer to implementation

Corollary

Let the vertices of an elimination tree T of A are postordered. Consider a row index i, $1 < i \leq n$. Denote also

$$adj_{G(A)}(i) \cap \{1, \ldots, i-1\} = \{c_1, \ldots, c_s\},\ 0 < c_1 < \ldots < c_s < i,\ s \geq 1.$$

Then c_t pro $t \in \{1, \ldots, s\}$ is a leaf of $T_r(i)$ of the elimination tree T iff either

$$t = 1 \quad \text{nebo} \quad t > 1 \wedge c_{t-1} < c_t - |T(c_t)| + 1.$$

Sparse Cholesky factorization - components

Elimination tree leaves

Needed subtree sizes

Algorithm

```
for $i = 1 : n$ do
    $|T(i)| = 1$
end $i$
4. for $i = 1 : n - 1$ do
    $k = parent(i)$
    $|T(k)| = |T(k)| + |T(i)|$
end $i$
```

Sparse Cholesky factorization - components

Elimination tree leaves

- Auxiliary result: looking for the leaves by columns

Theorem

Vertex j is a leaf of some row subtree of the postordered elimination tree T iff there exists $i \in adj_{G(A)}(j), i > j$ such that $i \notin adj_{G(A)}(k)$ for all $k \in T(j) \setminus \{j\}, i > k$.

- That is: going through columns - we will **surely** recognize that it is a leaf
- Next algorithm is by columns - more useful because of: (1) combined with other tasks, (2) compatible with column factorization

Sparse Cholesky factorization - components

Elimination tree leaves

- Algorithm to find the leaves of row subtrees

Algorithm

```
for $j = 1 : n$ do
    $isleaf(j)$=false; $prev\_nonz(j)$=0; compute $|T(j)|$
end $j$
for $j = 1$ to $n$ do
    for $i$ such that $i > j \wedge a_{ij} \neq 0$ $(j \in adj_{G(A)}(i))$ do
        $k = prev\_nonz(i)$
        if $k < j - |T(j)| + 1$ then
            $isleaf(j)$=true
        end if
        $prev\_nonz(i) = j$
    end $i$
end $j$
```

Sparse Cholesky factorization - components

Recapitulation

- Fill-in described both by rows and columns.
- How to avoid it: reorderings: just keep in mind the arrow matrix example

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

But this is not enough. An efficient algorithm needs also blocks

Sparse Cholesky factorization - components

Blocks

- Blocks are absolutely crucial to compute efficiently on contemporary computers: we need as much data as possible for a unit of data transfer inside memory hierarchy.
- In BLAS terminology:

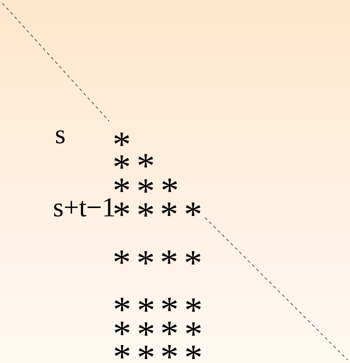
$$z = x + \alpha y \longrightarrow Z = X + \alpha Y \ (\ vector \ opes \ )$$

in general: saxpy $\longrightarrow$ dgemm

- But we have sparse matrices. It is not so straightforward to split their nonzeros into blocks.
- In fact, we need to reorder them in order to get blocks.
  - Application-based blocks in discretized systems.
  - Graph-based strategies which can be very fast.
  - But we need to optimize the block structure of L: supernodes.
  - Help: again our good friend, the elimination tree.

Sparse Cholesky factorization - components

Supernodes



Sparse Cholesky factorization - components

Supernodes

Definition

Let $s, t \in N$, $1 \leq s \leq n, 1 \leq t \leq n, s+t-1 \leq n$. We will say that a set of columns of L with indices

$$\{s, s+1, \ldots, s+t-1\}, \, \mathrm{kde} \, s, t \in N, \, 1 \leq s, t \leq n, s+t-1 \leq n, \quad (60)$$

is a **supernode** of L if this set cannot be increased by adding the column $s-1$ for $s > 1$ or the column $s+t$ for $s+t-1 < n$ a if, at the same time, column indices $s, s+1, \ldots, s+t-1$ satisfy

$$col_L(s) \cup \{s\} = col_L(s+t-1) \cup \{s, \ldots, s+t-1\}. \quad (61)$$

Vertex s and vertex $s+t-1$ (of the corresponding graph) will be called starting and ending vertex of the supernode, respectively. Supernode can be also a trivial set with one column (vertex) s only ($t=1$).

Sparse Cholesky factorization - components

Supernodes

Definition

(Reworded) Let $s, t \in \{1, \ldots, n\}$ such that $s + t - 1 \leq n$. Then the columns with indices $\{s, s+1, \ldots, s+t-1\}$ form a **supernode** if these columns satisfy $Struct(L_{*s}) = Struct(L_{*s+t-1}) \cup \{s, \ldots, s+t-2\}$, and the sequence is **maximal**.

Sparse Cholesky factorization - components

Supernodes

$$\begin{array}{c|ccccccccccc}
 & \cdots & \cdots & s & \cdots & \cdots & s' & \cdots & \cdots & \cdots & \cdots & \\
\hline
\vdots & \ddots & \cdots & & & & & & & & & \\
\vdots & \vdots & \ddots & & & & & & & & & \\
s & & & * & & & & & & & & \\
\vdots & & & * & \ddots & & & & & & & \\
\vdots & & & * & * & \ddots & & & & & & \\
s' & & & * & * & * & * & & & & & \\
\vdots & & & . & \cdots & \cdots & . & \ddots & & & & \\
\vdots & & & * & * & * & * & & \ddots & & & \\
\vdots & & & * & * & * & * & & & \ddots & & \\
\vdots & & & . & \cdots & \cdots & . & & & & \ddots & \\
\vdots & & & * & * & * & * & & & & & \ddots
\end{array}$$

Sparse Cholesky factorization - components

Supernodes

Theorem

Set of matrix column $S = \{s, s+1, \ldots, s+t-1\}$ is a **supernode** *of L iff it is maximal set of subsequent columns such that*

$$\text{vertex } s+k-1 \text{ is a son of vertex } s+k, \;\; k = 1, \ldots, t-1$$

provided

$$|\, col_L(s) \,| = |\, col_L(s+t-1) \,| + t - 1. \tag{62}$$

Sparse Cholesky factorization - components

Fundamental supernodes

Definition

Let $s, t \in N$, $1 \leq s \leq n, 1 \leq t \leq n, s+t-1 \leq n$ We say that columns $\{s, s+1, \ldots, s+t-1\}$ of L form a **fundamental supernode** of L if it is, at the same time, a maximal set of subsequent columns for which $s+i-1$ is the **only** son of $s+i$ in T for all $i = 1, \ldots, t-1$ provided

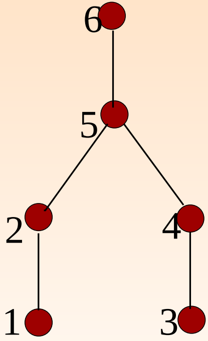
$$col_L(s) \cup \{s\} = col_L(s+t-1) \cup \{s, \ldots, s+t-1\}. \tag{63}$$

Sparse Cholesky factorization - components

Fundamental supernodes

- Example of a supernode that is not fundamental

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



Sparse Cholesky factorization - components

Fundamental supernodes

Theorem

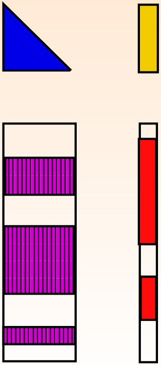
Column s, $1 \leq s \leq n$ is a starting column of a fundamental supernode iff s has either at least two sons in the elimination tree T or if s is a **leaf of a row subtree** *of T.*

- Fundamental supernodes can be found in a nearly linear time by traversing the **postordered elimination tree**

Sparse Cholesky factorization - components

Supernodes and efficient computation

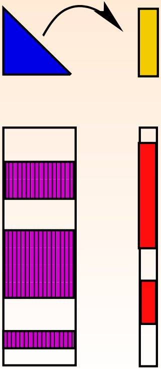
- the loop over rows has no indirect addressing: (dense BLAS1)



Sparse Cholesky factorization - components

Supernodes and efficient computation

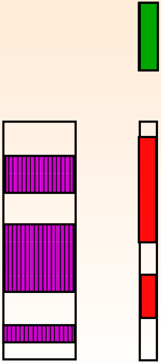
- the loop over rows has no indirect addressing: (dense BLAS1)



Sparse Cholesky factorization - components

Supernodes and efficient computation

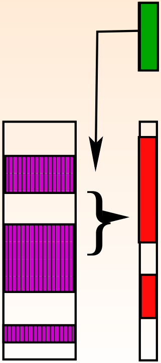
- the loop over rows has no indirect addressing: (dense BLAS1)
- the loop over columns of the updating supernode can be unrolled to save memory references (dense BLAS2)



Sparse Cholesky factorization - components

Supernodes and efficient computation

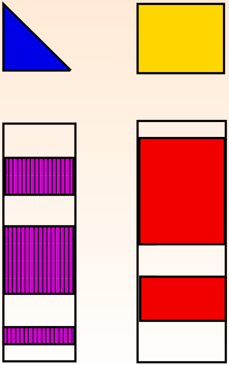
- the loop over rows has no indirect addressing: (dense BLAS1)
- the loop over columns of the updating supernode can be unrolled to save memory references (dense BLAS2)



Sparse Cholesky factorization - components

Supernodes and efficient computation

- the loop over rows has no indirect addressing: (dense BLAS1)
- 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)



Outline

1 Foreword
2 Direct methods and algebraic preconditioners
3 Basic terminology
4 Sparsity
5 Graphs and their matrices
6 From matrices to graphs
7 Graph searches
8 Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9 Schemes for solving systems of linear algebraic equations
10 Notes: on getting blocks in sparse matrices
11 Sparse factorization: fill-in
12 Sparse Cholesky factorization - components
13 Sparse Cholesky factorization - synthesis
14 Sparse LU factorization of generally nonsymmetric matrices
15 Sparse LU factorization of generally nonsymmetric matrices
16 Initial reordering
17 Stability of sparse factorizations
18 Symmetric indefinite factorization

Sparse Cholesky factorization - synthesis

Solvers: general strategy in the SPD case

Preprocessing

– prepares the matrix so that the fill-in amount is as small as possible

Symbolic factorization

– elimination tree, determines structures of columns of L (symbolic elimination in the strict sense). Consequently, L can be allocated and used for the actual decomposition

– the boundary between the first two steps is somewhat blurred due to many possible enhancements

Numeric factorization

– the actual decomposition to obtain numerical values of the factor L

Sparse Cholesky factorization - synthesis

Solvers: Preprocessing - an example why it is needed

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

$$A \rightarrow P^T A P$$

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

(our well-known arrow example)

Sparse Cholesky factorization - synthesis

Solvers

- 1) **Row Cholesky**. Based on substitution steps. They are difficult to be efficiently implemented on modern computers. We mention this approach only marginally
- 2) Block (supernodal) column algorithm. Sometimes called **left-looking**.
- 3) **Multifrontal** algorithm - efficient variant of the submatrix algorithm
- All sparse solvers may use **blocks**. More profound use of blocks in the two latter approaches. **Distinguish** blocks based on A and supernodes.

Sparse Cholesky factorization - synthesis

Sparse row Cholesky

- See the scheme in Section **Schemes for solving systems - row Cholesky**
- (1) The scheme is based on repeated solution of systems with triangular matrices (see above the Cholesky scheme)
- (2) We may also need to find sparsity patterns of the rows as they are constructed

Sparse Cholesky factorization - synthesis

Sparse row Cholesky

Theorem

Let $L = (l_{ij})_{i,j \in \{1,\ldots,n\}} \in R^{n \times n}$ *is a regular lower triangular matrix and* $b = (b_i)_{i \in \{1,\ldots,n\}} \in R^n$*. Assuming (as always) non-cancelation in solving the system*

$$Lx = b$$

for $x = (x_i)_{i \in \{1,\ldots,n\}} \in R^n$ *we have* $x_i \neq 0$ *iff there is in* $G(L^T)$ *a path* $j \Rightarrow i$ *from* $j \in \{1,\ldots,n\}, j < i$ *such that* $b_j \neq 0$*.*

- Apply this prediction to the repeated solution mentioned on the previous slide.
- Similarly for an upper triangular matrix.

Sparse Cholesky factorization - synthesis

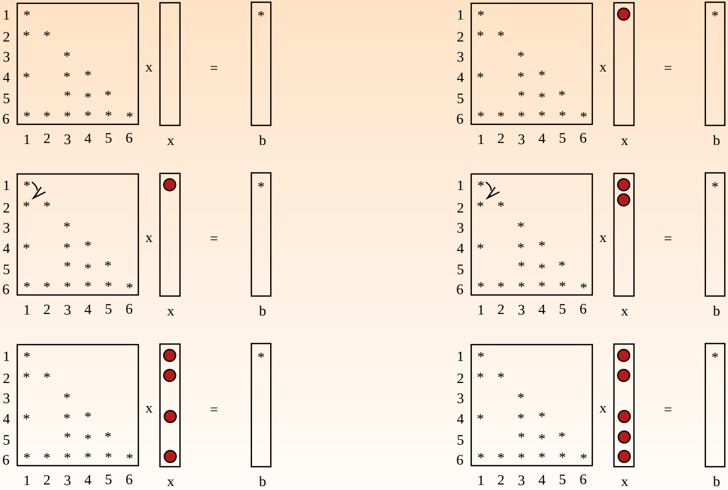
Sparse row Cholesky: example

$$\begin{pmatrix} * & & & & \\ & * & & & \\ * & & * & & \\ & & & * & \\ & & * & & * \end{pmatrix} \begin{pmatrix} \color{red}{*} \\ \\ \color{red}{*} \\ \\ \color{red}{*} \end{pmatrix} = \begin{pmatrix} * \\ \\ \\ \\ \\ \end{pmatrix} \tag{64}$$

- The only nonzero of the right-hand side implies the three nonzeros in the solution

Sparse Cholesky factorization - synthesis

Another example: subsequent development of fill-in



Sparse Cholesky factorization - synthesis

Block column (left-looking) algorithm

Algorithm

```
Find initial fill-in minimizing matrix reordering
Symbolic factorization:
    Find elimination tree
    Find its postordering
    Find column counts
    Optimize the postordering (more reasons, partially mentioned later)
    Find supernodes, optimize them, estimate the workspace, reorder again.
    Find supernodal elimination tree.
    Find column structures for the left-looking supernodal factorization
Supernodal numeric factorization
```

Sparse Cholesky factorization - synthesis

Block column (left-looking) algorithm: notes on implementation

- The following theorem shows that we need to go through **rows** of columns of L computed so far within a block algorithm.

Theorem

Let $j > k$. Numerical values of entries in L_{*j} depend on the values of entries in L_{*k} iff $l_{jk} \neq 0$.

- Note that to get the **sparsity patterns of columns** we need **less**.

Sparse Cholesky factorization - synthesis

Block left-looking algorithm: **notes on implementation**

- Construction columns (block columns) one by one.
- Going through the rows **can be simulated by linked lists** as we saw in **CSC-CSR matvec.**
- Plan to mention this again when discussing **approximate** factorizations.

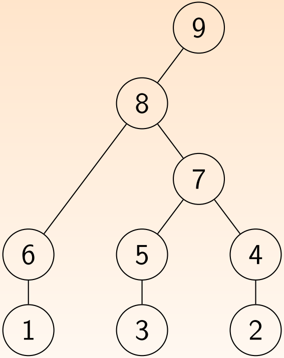
Sparse Cholesky factorization - example

Multifrontal method: (example 1)

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

Sparse Cholesky factorization - synthesis

Multifrontal method: (example 1)



Sparse Cholesky factorization - synthesis

Multifrontal method: (example 1)

$$F_1 = \begin{array}{c|cccc} & 1 & 6 & 8 & 9 \\ \hline 1 & * & * & * & * \\ 6 & * & & & \\ 8 & * & & & \\ 9 & * & & & \end{array}, \quad V_1 = \begin{array}{c|ccc} & 6 & 8 & 9 \\ \hline 6 & * & * & * \\ 8 & * & * & f \\ 9 & * & f & * \end{array}.$$

Here V_1 is dense and f denotes fill-in entries. Similarly, we have

$$F_2 = \begin{array}{c|ccc} & 2 & 4 & 7 \\ \hline 2 & * & * & * \\ 4 & * & & \\ 7 & * & & \end{array}, \quad V_2 = \begin{array}{c|cc} & 4 & 7 \\ \hline 4 & * & f \\ 7 & f & * \end{array}, \quad F_3 = \begin{array}{c|ccc} & 3 & 5 & 8 \\ \hline 3 & * & * & * \\ 5 & * & & \\ 8 & * & & \end{array}, V_3 = \begin{array}{c|cc} & 5 & 8 \\ \hline 5 & * & * \\ 8 & * & * \end{array}.$$

The sparsity pattern of the frontal matrix F_4 is then

$$F_4 = \begin{array}{c|ccc} & 4 & 8 & 9 \\ \hline 4 & * & * & * \\ 8 & * & & \\ 9 & * & & \end{array} \mathbin{\leftrightarrow\!\!\!\!\!\updownarrow} V_2 \; = \; \begin{array}{c|cccc} & 4 & 7 & 8 & 9 \\ \hline 4 & * & f & * & * \\ 7 & f & * & & \\ 8 & * & & & \\ 9 & * & & & \end{array}, \quad V_4 = \begin{array}{c|ccc} & 7 & 8 & 9 \\ \hline 7 & * & f & * \\ 8 & f & * & f \\ 9 & * & f & * \end{array},$$

Sparse Cholesky factorization - synthesis

Multifrontal method

Theorem

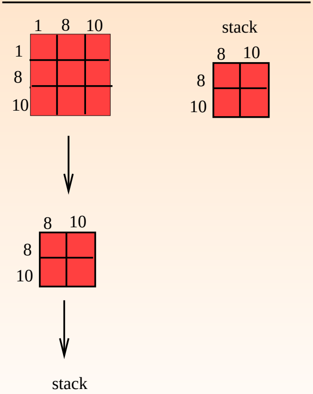
Let $\mathcal{T}$ be postordered. Assume each computed generated element (V_x) is pushed onto a stack. Then when constructing the (frontal matrix) F_j, the required generated elements are **on the top of the stack***. They can be directly popped from the stack and assembled into F_j.*

Proof sketch: 1) Vertices of each subtree of the postordered $\mathcal{T}$ form an interval. 2) Denote c_l, $l = 1, \ldots, s$ children of j in $\mathcal{T}$. 3) Each c_l is the root of a subtree $\mathcal{T}(c_l)$. 4) Once the frontal matrix F_{c_l} for a leaf of $\mathcal{T}(c_l)$ is constructed, all its children have been processed and the generated V_{c_l} is pushed onto the stack. 5) That is, all subtrees $\mathcal{T}_{c_l}$, $l = 1, \ldots, s$ are fully assembled into the generated elements before F_j can be constructed. 6) If F_j is ready to be assembled (step j), the s generated V_{c_l}, $l = 1, \ldots, s$ are on the top of the stack.

Sparse Cholesky factorization - synthesis

Multifrontal method: (example 2)

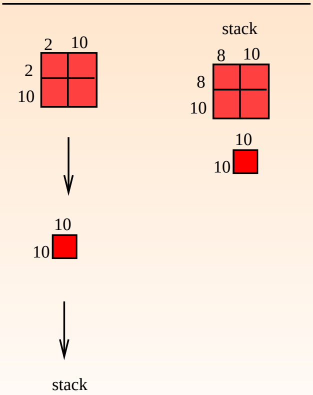
$$\begin{pmatrix}
* & & & & & & & * & & * \\
 & * & & & & & & & & * \\
 & & * & & & & * & & & * \\
 & & & * & & & * & & & \\
 & & & & * & * & & & * & \\
 & & & & * & * & & & f & * \\
 & & * & * & & & * & & * & f \\
* & & & & & & & * & & * \\
 & & & & * & f & * & & * & f \\
* & * & * & & & * & f & * &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



Sparse Cholesky factorization - synthesis

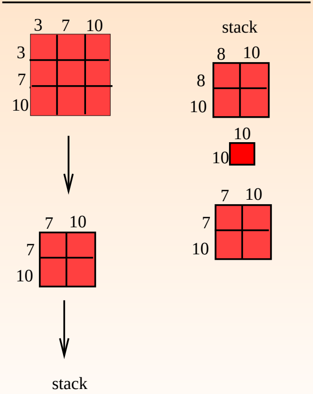
Multifrontal method: (example 2)

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



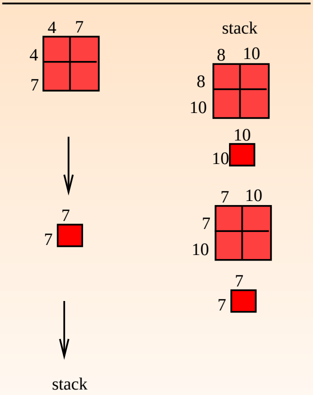
Multifrontal method: (example 2)

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



Direct methods: Multifrontal method

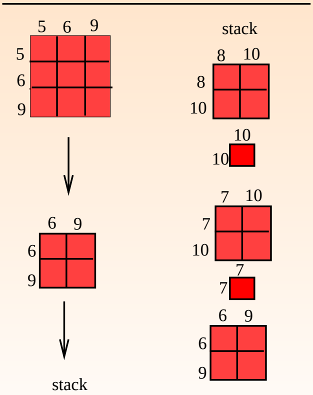
$$\begin{pmatrix}
* & & & & & & & * & & * \\
& * & & & & & & & & * \\
& & * & & & & * & & & * \\
& & & * & & & * & & & \\
& & & & * & * & & & * & \\
& & & & * & * & & & f & * \\
& & * & * & & & * & & * & f \\
* & & & & & & & * & & * \\
& & & & * & f & * & & * & f \\
* & * & * & & & * & f & * & f & *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



Sparse Cholesky factorization - synthesis

Multifrontal method: (example 2)

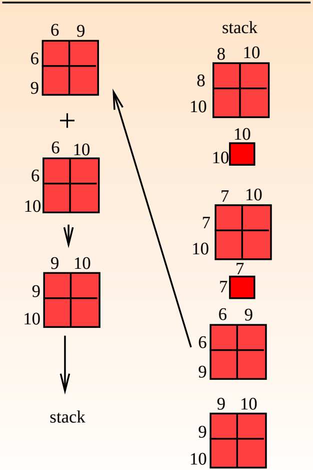
$$\begin{pmatrix}
* & & & & & & & * & & * \\
& * & & & & & & & & * \\
& & * & & & & * & & & * \\
& & & * & & & * & & & \\
& & & & * & * & & & * & \\
& & & & * & * & & & f & * \\
& & * & * & & & * & & * & f \\
* & & & & & & & * & & * \\
& & & & * & f & * & & * & f \\
* & * & * & & & * & f & * & f & *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



Sparse Cholesky factorization - synthesis

Multifrontal method: (example 2)

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



Sparse Cholesky factorization - synthesis

Multifrontal method: summary

- Right-looking (submatrix) method
- Does not form the Schur complement directly. Instead, the **updates** are moved to a **stack** as dense matrices and **used when needed**.
- The processing order is based on the **elimination tree**
- We will see that in order to have the needed **updates** at the stack top, postordering is needed.
- **Specific postorderings** used to minimize the needed amount of memory.
- Now example, properties repeated once more later.

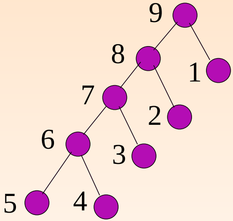
Sparse Cholesky factorization - synthesis

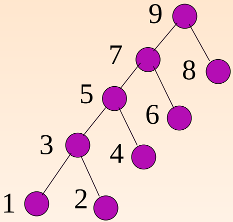
Multifrontal method: assumptions and properties

- We do need to have the entries from the stack readily available.
- → elimination tree should be postordered
- Arithmetic of dense matrices
- Connection with the frontal method (later) is relatively week.
- One of the most important methods for the sparse direct factorization.

Sparse Cholesky factorization - synthesis

Multifrontal method: postorderings memory issues





- First case: Maximum stack size may be $1 \times 1 + 2 \times 2 + 3 \times 3 + 4 \times 4$
- Second case: Maximum stack size may be 4×4
- Conclusion: **Even postorderings can be very different with respect to algorithmic/architectural needs**

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Sparse LU factorization of generally nonsymmetric matrices

Similarities with the SPD case

- Fill-in lemma - it was formulated generally, and it is valid for the nonsymmetric case as well. But, matrix **factorizability** should be assumed here.
- Fill-in theorem: next slide: uses directed paths instead of the undirected ones.
- Extended fill-in theorem: based on the interplay of **both** factors

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Sparse LU factorization of generally nonsymmetric matrices

Nonsymmetric fill-in theorem: statement

Theorem

(nonsymmetric fill-in theorem) *Let* $i, j, k \in \{1, \ldots, n\}$, $k < \min\{i, j\}$, $k \leq n-1$. *Then* $a_{ij}^{(k)} \neq 0$ *iff there is a directed path* $i \overset{G}{\Rightarrow} j$ *denoted as* $(i, p_1, \ldots, p_t, j)$ *v* $G(A)$, *such that*

$$p_l \leq k,\ 1 \leq l \leq k. \tag{65}$$

The set of intermediate vertices $\{p_1, \ldots, p_t\}$ *can be empty.*

- The theorem is a direct analogy of the one for symmetric factorizations. But it uses **directed** paths.

Sparse LU factorization of generally nonsymmetric matrices

Nonsymmetric fill-in theorem: example

$$\begin{array}{cc} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{pmatrix} \ast & & \ast & & \ast & \\ \ast & \ast & & & \ast & \ast \\ & & \ast & \ast & & \\ \ast & & \ast & \ast & & \\ & & \ast & & \ast & \\ \ast & \ast & & & & \ast \end{pmatrix} \end{array} \qquad \begin{array}{cc} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{pmatrix} \ast & & \ast & & \ast & \\ \ast & \ast & f & & \ast & \ast \\ & & \ast & \ast & & \\ \ast & & \ast & \ast & f & \\ & & \ast & f & \ast & \\ \ast & \ast & f & f & f & \ast \end{pmatrix} \end{array} \tag{66}$$

- For example, l_{64} implied by existence of the path

$$6 \Rightarrow 4 \;\equiv\; 6 \to 1 \to 3 \to 4, \tag{67}$$

or the path

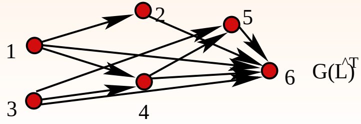
$$6 \Rightarrow 4 \;\equiv\; 6 \to 2 \to 1 \to 3 \to 4. \tag{68}$$

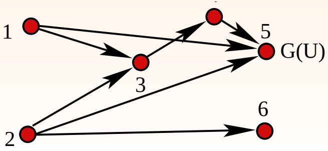
Sparse LU factorization of generally nonsymmetric matrices

Graph model 1: directed acyclic graphs

$$\begin{array}{ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & * & & * & & * & \\ 2 & * & * & & & * & * \\ 3 & & & * & * & & \\ 4 & * & & * & * & & \\ 5 & & & * & & * & \\ 6 & * & * & & & & * \end{array} \qquad \begin{array}{ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & * & & * & & * & \\ 2 & * & * & f & & * & * \\ 3 & & & * & * & & \\ 4 & * & & * & * & f & \\ 5 & & & * & f & * & \\ 6 & * & * & f & f & f & * \end{array} \tag{69}$$

- **directed acyclic graphs** capture structure of the factors. We use $G(L^T)$ (L by columns) and $G(U)$ (U by rows).





Sparse LU factorization of generally nonsymmetric matrices

Interplay of L and U

Theorem

Let $A = LU$. Consider graf $G(A)$ with $V = \{1, \ldots, n\}$. Let $i > j$. Then $l_{ij} \neq 0$ iff there is a $k,\ k \leq j$ such that $a_{ik} \neq 0$ and there is directed path $k \Rightarrow j$ in $G(U)$.

Let $A = LU$. Consider graf $G(A)$ with $V = \{1, \ldots, n\}$. Let $i < j$. Then $u_{ij} \neq 0$ iff there is a $k,\ k \leq i$ such that $a_{kj} \neq 0$ and there is a directed path $k \Rightarrow i$ in $G(L^T)$.

- That is: For LU factorization a starter plus a path in the **"opposite"** graph are needed.
- Clear hopefully even without the formal proof ☺

Sparse LU factorization of generally nonsymmetric matrices

Interplay of L and U: an example

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & * & & * & & * & \\ 2 & * & * & & & * & * \\ 3 & & & * & * & & \\ 4 & * & & * & * & & \\ 5 & & & * & & * & \\ 6 & * & * & & & & * \end{array} \qquad \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & * & & * & & * & \\ 2 & * & * & f & & * & * \\ 3 & & & * & * & & \\ 4 & * & & * & * & f & \\ 5 & & & * & f & * & \\ 6 & * & * & f & f & f & * \end{array} \tag{70}$$

- L fills at $(6, 4)$ since

$$a_{ij} \equiv a_{62} \neq 0$$

and in $G(U)$ there is a path

$$k \equiv 2 \to 3 \to 4 \equiv j.$$

Sparse LU factorization of generally nonsymmetric matrices

Column structures of L, row structures of U

Theorem

Let $A = LU$ and there is no cancellation. Then

$$\mathcal{S}(L_{*j}) = \mathcal{S}(A_{*j}) \cup \bigcup \{\mathcal{S}(L_{*k}) \mid k < j,\ u_{kj} \neq 0\} \setminus \{1, \ldots, j-1\} \quad (71)$$

Let $A = LU$ and there is no cancellation. Then

$$\mathcal{S}(U_{i*}) = \mathcal{S}(A_{i*}) \cup \bigcup \{\mathcal{S}(U_{k*}) \mid k < i,\ l_{ik} \neq 0\} \setminus \{1, \ldots, i-1\} \quad (72)$$

- Not a victory - too many entries to take into account

Sparse LU factorization of generally nonsymmetric matrices

Nonsymmetric replication

Observation

Let $1 \le \ldots \le k < j \le \ldots \le n$ a $(k, j) \in E(G(U))$. *Then*

$$\mathcal{S}(L_{*k}) \setminus \{1, \ldots, j-1\} \subseteq \mathcal{S}(L_{*j}). \tag{73}$$

- The structures can be possibly constructed **sequentially**. The construction should interleave updates of of columns of $\mathcal{S}(L)$ and rows of $\mathcal{S}(U)$.
- But: the **construction complexity** is a problem.
- Previous example:

$$\mathcal{S}(L_{*1}) \setminus \{2\} \subseteq \mathcal{S}(L_{*3}).$$

Sparse LU factorization of generally nonsymmetric matrices

How can be structures reduced: transitive reduction

Definition

Graph $G^0 = (V, E^0)$ is called a **transitive reduction** of the directed graph $G = (V, E)$ if

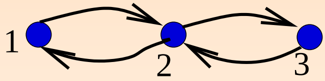
- For any $u, v \in V$ we have $u \overset{G^0}{\Longrightarrow} v$ iff $u \overset{G}{\Rightarrow} v$.
- There is no graph with the vertex set V with a smaller size $|E^0|$ satisfying this condition.

- Transitive reduction is **not** necessarily **unique**.
- Transitive reduction G^0 may not be a subgraph of G.

Sparse LU factorization of generally nonsymmetric matrices

Transitive reduction

- Transitive reduction is the loop interconnecting vertices $1, 2$ and 3.



Theorem

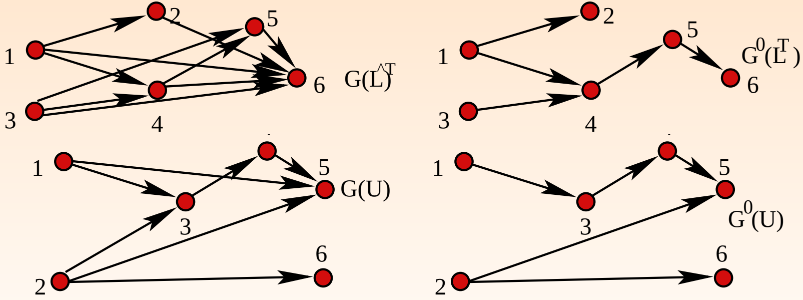
Transitive reduction of a **directed acyclic graph** *is unique and is a subgraph of G.*

Theorem

Transitive reduction of a directed acyclic graph $G(U) \equiv G(L^T)$ of a symmetric irreducible matrix is the elimination tree.

Sparse LU factorization of generally nonsymmetric matrices

Transitive reduction



Sparse LU factorization of generally nonsymmetric matrices

Using transitive reductions

- Transitive reduction does not change **vertex reachability**.

Theorem

Let $A = LU$ and there is no cancellation and let $l_{ij} \neq 0$, $i > j$. Then there is a directed path $i \xrightarrow{G^0(L)} j$.

- Nonzero in L (edge in $G(L^T)$) means a path in the transitive reduction $G^0(L^T)$.

Sparse LU factorization of generally nonsymmetric matrices

Using transitive reductions

Theorem

Let $A = LU$ *and there is no cancellation. Consider* $G(A)$ *with* $V = \{1, \ldots, n\}$ *and transitive reduction of* $G(U)$ $G^0(U) = (V, E_U^0)$*. Then*

$$\mathcal{S}(L_{*j}) = \mathcal{S}(A_{*j}) \cup \bigcup \{\mathcal{S}(L_{*k}) \mid (k, j) \in E_U^0\} \setminus \{1, \ldots, j-1\} \tag{74}$$

- This is a straightforward generalization of the **column structure formula** from the symmetric case.

Sparse LU factorization of generally nonsymmetric matrices

Strengthened interplay of L and U

Theorem

Let $A = LU$ and there is no cancellation. Consider $G(A)$ with $V = \{1, \ldots, n\}$.
Let $i > j$. Then $l_{ij} \neq 0$ iff there is a $k,\ k \leq j$ such that $a_{ik} \neq 0$ and there is a directed path $k \xRightarrow{G^0(U)} j$.
Let $i < j$. Then $u_{ij} \neq 0$ iff there is a $k,\ k \leq i$ such that $a_{kj} \neq 0$ and there is a directed path $i \xRightarrow{G^0(L)} k$.

- And that's it. ☺: We can use the transitive reductions.
- Or something **in between**.

Sparse LU factorization of generally nonsymmetric matrices

- We need to know how transitive reductions can be constructed.
- Next slide: symmetric pruning
- We can remove some edges of $G(L)$ (or $G(L^T)$) or $G(U)$.
- The next pattern can be easily exploited.

Sparse LU factorization of generally nonsymmetric matrices

One TR possibility: symmetric pruning

$$
\begin{array}{c@{}c}
 & \begin{array}{ccccccc} & j & & s & & k & \end{array} \\
\begin{array}{c} \\ j \\ \\ s \\ \\ k \\ \\ \end{array} &
\left(\begin{array}{ccccccc}
* & & & & & & \\
 & * & & * & & * & \\
 & & * & & & & \\
 & * & & * & & & \\
 & & & & * & & \\
 & * & & & & * & \\
 & & & & & & *
\end{array}\right)
\end{array}
\qquad
\begin{array}{c@{}c}
 & \begin{array}{ccccccc} & j & & s & & k & \end{array} \\
\begin{array}{c} \\ j \\ \\ s \\ \\ k \\ \\ \end{array} &
\left(\begin{array}{ccccccc}
* & & & & & & \\
 & * & & * & & * & \\
 & & * & & & & \\
 & * & & * & & f & \\
 & & & & * & & \\
 & * & & f & & * & \\
 & & & & & & *
\end{array}\right)
\end{array}
\tag{75}
$$

$$\downarrow$$

$$
\begin{array}{c@{}c}
 & \begin{array}{ccccccc} & j & & s & & k & \end{array} \\
\begin{array}{c} \\ j \\ \\ s \\ \\ k \\ \\ \end{array} &
\left(\begin{array}{ccccccc}
* & & & & & & \\
 & * & & * & & & \\
 & & * & & & & \\
 & * & & * & & f & \\
 & & & & * & & \\
 & & & f & & * & \\
 & & & & & & *
\end{array}\right)
\end{array}
\tag{76}
$$

Sparse LU factorization of generally nonsymmetric matrices

Problems of the approach based on directed acyclic graphs

- **Pivoting** is often needed.
- This problem is still partially shared by other approaches as well.
- But not by the next approach.

Sparse LU factorization of generally nonsymmetric matrices

Sparse LU based on the column elimination tree

Theorem

Assume that A has all diagonal entries nonzero. Furthermore assume that $A^TA = \tilde{L}\tilde{L}^T$ without cancellation. Then

$$\mathcal{S}(L+U) \subseteq \mathcal{S}((\tilde{L}) + (\tilde{L})^T)$$

for any permutation matrix P such that $PA = LU$.

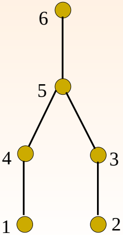
- $$(PA)^TPA = A^TP^TPA = A^TA$$

  Therefore, A^TA is independent of partial (row) pivoting.
- The problem is that A^TA can be **much denser.**

Sparse LU factorization of generally nonsymmetric matrices

Sparse LU based on the column elimination tree

$$
\underset{A}{\begin{pmatrix}
* & & & & & \\
 & * & & & * & \\
 & & & * & * & * \\
* & & & * & & * \\
 & * & * & & & \\
 & & * & & * &
\end{pmatrix}}
\quad
\underset{A^T A}{\begin{pmatrix}
* & & & * & & * \\
 & * & * & & * & \\
 & * & * & & & * \\
* & & & * & * & * \\
 & * & & * & * & * \\
* & & * & * & * & *
\end{pmatrix}}
\quad
\underset{\tilde{L}+\tilde{L}^T}{\begin{pmatrix}
* & & & * & & * \\
 & * & * & & * & \\
 & * & * & & f & * \\
* & & & * & * & * \\
 & * & f & * & * & * \\
* & & * & * & * & *
\end{pmatrix}}
\tag{77}
$$



Sparse LU factorization of generally nonsymmetric matrices

Sparse LU based on the column elimination tree

$$A : \begin{pmatrix} * & & & * & & & \\ * & * & & * & & & \\ & & * & & & * & \\ * & * & & * & & & * \\ & * & * & & * & * & * \\ & & * & & & * & \\ * & & & * & * & & * \end{pmatrix} \qquad A^T A : \begin{pmatrix} * & * & & * & * & & * \\ * & * & * & * & * & * & * \\ & * & * & & * & * & * \\ * & * & & * & * & & * \\ * & * & * & * & * & * & * \\ & * & * & & * & * & * \\ * & * & * & * & * & * & * \end{pmatrix}$$

$$L + U : \begin{pmatrix} * & & & * & & & \\ * & * & f & * & & & \\ & & * & & & * & \\ * & * & & * & f & & * \\ & * & * & f & * & * & * \\ & & * & & & * & \\ * & & & * & * & f & * \end{pmatrix} \qquad \tilde{L} + \tilde{L}^T : \begin{pmatrix} * & * & & * & * & & * \\ * & * & * & * & * & * & * \\ & * & * & f & * & * & * \\ * & * & f & * & * & f & * \\ * & * & * & * & * & * & * \\ & * & * & f & * & * & * \\ * & * & * & * & * & * & * \end{pmatrix}$$

Sparse LU factorization of generally nonsymmetric matrices

Sparse LU based on the nonsymmetric elimination tree

$$parent(k) = \min\{j \mid j > k \wedge j \xrightarrow{G(L)} k\}.$$

$$\downarrow$$

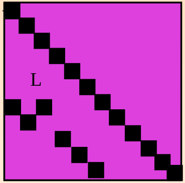
$$parent(k) = \min\{j \mid j > k \wedge j \xrightarrow{G(L)} k \xrightarrow{G(L^T)} j\}.$$

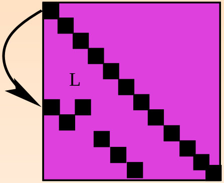
$$\downarrow$$

$$parent(k) =: \min\{j \mid j > k \wedge j \overset{G(L)}{\Longrightarrow} k \overset{G(U)}{\Longrightarrow} j\}. \tag{78}$$

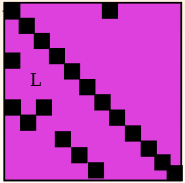
Sparse LU factorization of generally nonsymmetric matrices

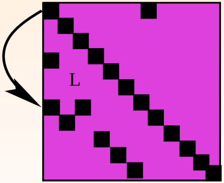
Sparse LU based on the nonsymmetric elimination tree





- Path $7 \xrightarrow{G(L)} 1 \xrightarrow{G(L^T)} 7$





- Path $7 \overset{G(L)}{\Longrightarrow} 1 \overset{G(L^T)}{\Longrightarrow} 7$

Sparse LU factorization of generally nonsymmetric matrices

Sparse LU based on the nonsymmetric elimination tree

$$\begin{array}{c|cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & * & & * & & & & & & & \\
2 & * & * & & & * & & & * & & * \\
3 & * & & * & * & & & & & & \\
4 & & & * & * & & & & & * & * \\
5 & & & & * & * & * & & * & & \\
6 & & * & & & & * & & & & \\
7 & & & & * & & & * & & & \\
8 & * & * & & & & & & * & & \\
9 & & & & & & & & & * & * \\
10 & & & & & * & & * & & & *
\end{array}
\qquad
\begin{array}{c|cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & * & & * & & & & & & & \\
2 & * & * & f & & * & & & * & & * \\
3 & * & & * & * & & & & & & \\
4 & & & * & * & & & & & * & * \\
5 & & & & * & * & * & & * & f & f \\
6 & & * & f & f & f & * & & f & f & f \\
7 & & & & * & & & * & & f & f \\
8 & * & * & f & f & f & f & & * & f & f \\
9 & & & & & & & & & * & * \\
10 & & & & & * & f & * & f & f & *
\end{array}$$

- $6 \xrightarrow{G(L)} 2 \xrightarrow{G(U)} 5 \xrightarrow{G(U)} 6$, $6 \xrightarrow{G(L)} 5 \xrightarrow{G(U)} 6$ implies that 6 is father of 5 and 2.

Sparse LU factorization of generally nonsymmetric matrices

Sparse LU based on the nonsymmetric elimination tree

Theorem

Let $A = LU$ and there is no cancellation. We have

$$j \overset{G(L)}{\Longrightarrow} k \iff j \underset{\{1,\ldots,j\}}{\overset{G(A)}{\Longrightarrow}} k, \tag{79}$$

where all the intermediate vertices on the considered paths are j at most.

- This is just an extension of the path theorem. Each L entry is replaced by such feasible path.

Sparse LU factorization of generally nonsymmetric matrices

Sparse LU based on the nonsymmetric elimination tree

Theorem

Vertex i is ancestor of j in the nonsymmetric elimination tree $T(A)$ of $G(A)$ iff $i > j$ and both the vertices j and i belong to the same **strong component** *of $G(\{1, \ldots, i\})$.*

- Based on this theorem we can construct the nonsymmetric elimination tree!
- The construction combines **finding strong components** (algorithm based on the depth-first search) and simple additional numbering.

Sparse LU factorization of generally nonsymmetric matrices

Sparse LU based on the nonsymmetric elimination tree

- A simple construction of the nonsymmetric elimination tree.

Algorithm

```
for $i = 1 : n$ do
    $parent(i) = n + 1$
end $i$
for $i = 1 : n$ do
    *Find the component $C_i$ that contains $i$.*
    for $j \in C_i,\ j \neq i$ do
        if $parent(j) > n$ set $parent(j) = i$
    end $j$
    $parent(i) = n + 1$
end $i$
```

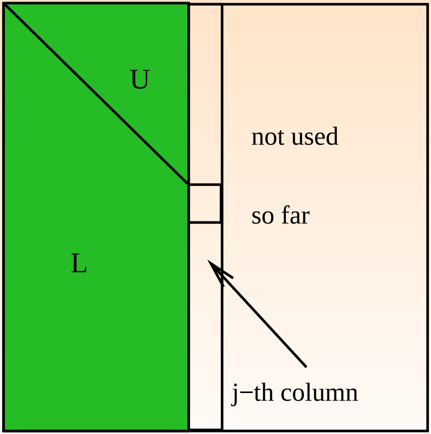
Sparse LU factorization of generally nonsymmetric matrices

Left-looking LU with partial pivoting

```
**for** $j = 1 : n$ **do**
    Compute $u_{1:j-1,j}$ from $L_{1:j-1,1:j-1}u_{1:j-1,j} = A_{1:j-1,j}$
    Set $\tilde{L}_{j:n,j} = A_{j:n,j} - L_{j:n,1:j-1}u_{1:j-1,j}$
    Find in $\tilde{L}_{j:n,j}$ a component $\ell$ of maximum value and **permute** it to diagonal.
    Set $U_{j,j} = \ell$
    Set $L_{j:n,j} = \tilde{L}_{j:n,j}/U_{jj}$
**end** $j$
```

Sparse LU factorization of generally nonsymmetric matrices

Left-looking LU with partial pivoting



Sparse LU factorization of generally nonsymmetric matrices

Preprocessing 1: Forcing diagonal dominance

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

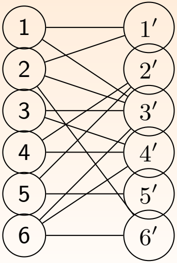


Figure: *A sparse matrix A and its bipartite graph $\mathcal{G}_b(A)$.*

Sparse LU factorization of generally nonsymmetric matrices

Diagonal dominance

Definition

Matching of a simple undirected graph $G = (V, E)$ is its subgraph $M = (V_M, E_M)$ induced by a set of its vertex disjoint edges $E_M, \ E_M \subseteq E$.

- In matrix terms, a matching corresponds to a set of non zero entries with no two belonging to the same row or column. A vertex is matched if there is an edge in the matching incident on the vertex, and is unmatched (or free) otherwise.
- The **cardinality** of a matching is the number of edges in it and is sometimes called the **structural rank** of A.
- A **maximum cardinality matching** (or **maximum matching**) is a matching of maximum cardinality. This is equal to n if A is structurally nonsingular and, in this case, the matching is said to be **perfect**.

Sparse LU factorization of generally nonsymmetric matrices

Preprocessing 1: Forcing diagonal dominance

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

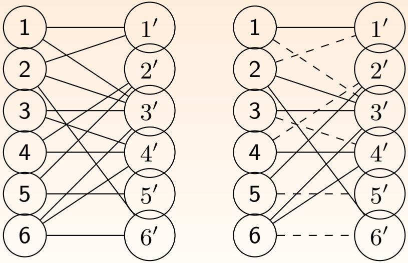


Figure: *A sparse matrix A and its bipartite graph $\mathcal{G}_b(A)$ (left). The edges that belong to the perfect matching in $\mathcal{G}_b(A)$ are given by the dashed lines (right).*

Sparse LU factorization of generally nonsymmetric matrices

Preprocessing 1: Forcing diagonal dominance

- Matching defines an $n \times n$ permutation matrix Q with entries q_{ij} given by
$$\begin{cases} q_{ij} = 1, & \text{if } (j \to i) \in \mathcal{M}, \\ q_{ij} = 0, & \text{otherwise.} \end{cases}$$
- Both QA and AQ have the matching entries on the (zero-free) diagonal. The column permuted matrix AQ is illustrated on the next slide.

Sparse LU factorization of generally nonsymmetric matrices

Preprocessing 1: Forcing diagonal dominance

$$Q = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & & 1 & & & & \\ 2 & & & & 1 & & \\ 3 & 1 & & & & & \\ 4 & & & 1 & & & \\ 5 & & & & & 1 & \\ 6 & & & & & & 1 \end{array} \qquad AQ = \begin{array}{c|cccccc} & 3' & 1' & 4' & 2' & 5' & 6' \\ \hline 1 & * & * & & & & \\ 2 & * & * & & & & * \\ 3 & * & & * & & & \\ 4 & & & * & * & & \\ 5 & & & & * & * & \\ 6 & * & & * & & & * \end{array}$$

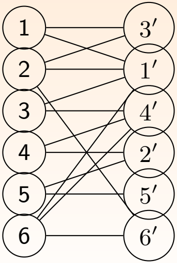


Figure: *The permutation matrix Q, the column permuted matrix AQ*

Preprocessing 1: Forcing diagonal dominance

- If a perfect matching exists, it can be found using **augmenting paths**.
- Recall that a path $\mathcal{P}$ in a graph is an ordered set of edges in which successive edges are incident to the same vertex. $\mathcal{P}$ is called an $\mathcal{M}$-**alternating path** if the edges of $\mathcal{P}$ are alternately in $\mathcal{M}$ and not in $\mathcal{M}$. An $\mathcal{M}$-alternating path is an $\mathcal{M}$-**augmenting path** if it connects an unmatched column node with an unmatched row node.
- Let $\mathcal{M}$ and $\mathcal{P}$ be subsets of E and define

  $$\mathcal{M} \oplus \mathcal{P} := (\mathcal{M} \setminus \mathcal{P}) \cup (\mathcal{P} \setminus \mathcal{M}).$$

  If $\mathcal{M}$ is a matching and $\mathcal{P}$ is an $\mathcal{M}$-augmenting path, then $\mathcal{M} \oplus \mathcal{P}$ is a matching with cardinality $|\mathcal{M}|+1$.
- Growing the matching in this way is called augmenting along $\mathcal{P}$ and can be used to find a perfect matching.

Sparse LU factorization of generally nonsymmetric matrices

Preprocessing 1: Forcing diagonal dominance

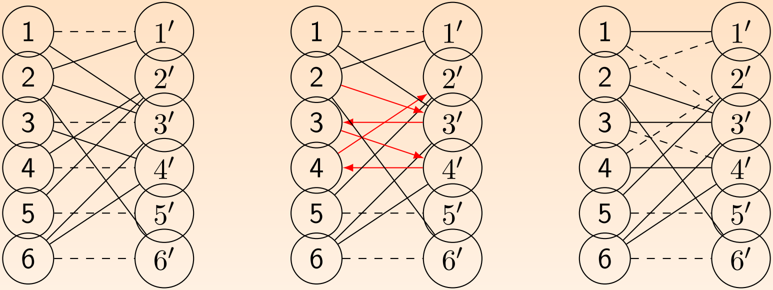


Figure: *An illustration of the search for a perfect matching using augmenting paths. On the left, the dashed lines are the initial matching. In the centre, the red line is an augmenting path with end vertices 2 and $2'$. On the right is a perfect matching obtained using the augmenting path.*

Sparse LU factorization of generally nonsymmetric matrices

Preprocessing 1: Forcing diagonal dominance: summary

Observation

If there exists a perfect matching of the bipartite graph model $G(A) = (R, C, E)$ of the matrix A, where $|R| = |C|$ and edge set $E_M = \{(i_k, j_k) \mid k = 1, \ldots, n\}$ then there are permutation matrices P and Q such that the diagonal entries of PAQ are

$$\{a_{i_k,j_k} \mid k = 1, \ldots, n\}.$$

- Hooray! We are able to move nonzeros to diagonal.
- But we would like to have at the diagonal **large** entries.
- Modified matching problems: **matchings weighted by various ways**.

Sparse LU factorization of generally nonsymmetric matrices

Preprocessing 2: Block triangular forms

- Nonsymmetric matrices enable nonsymmetric permutations. Having symmetric matrices, nonsymmetric permutations it may be possible but very often not advisable.

Definition

Let $G = (V, E)$ be a directed graph and let $V_1, \ldots, V_k$ be vertex sets of its strong components. Its **condensation (see above in the slides)** is the directed graph $G_C = (V', E')$ where $V' = \{V_1, \ldots, V_k\}$ and where

$$E' = \{(V_i, V_j)|\ (\exists x \in V_i)(\exists y \in V_j)((x, y) \in E)\}.$$

Sparse LU factorization of generally nonsymmetric matrices

Preprocessing 2: Block triangular forms (some repetition)

Theorem

Condensation G_C of G does not contain any directed walk and it is a directed acyclic graph.

Corollary

Vertices of condensation can be topologically ordered.

- Algorithmically, we are again by algorithms to find strong components

Sparse LU factorization of generally nonsymmetric matrices

Block triangular forms

Theorem

For a square A of dimension n exist permutation matrix P and a natural number $t \geq 1$ such that

$$PAP^T = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1t} \\ 0 & A_{22} & \dots & A_{2t} \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & A_{tt} \end{pmatrix}, \tag{80}$$

where $A_{11}, A_{22}, \dots, A_{tt}$ are square irreducible matrices. The matrices $A_{11}, A_{22}, \dots, A_{tt}$ are uniquely determined up to symmetric permutation of their rows and columns and up to their order on the diagonal of PAP^T.

Sparse LU factorization of generally nonsymmetric matrices

Preprocessing 2: Block triangular forms (an example)

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

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

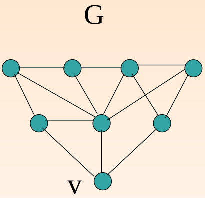
Initial reordering

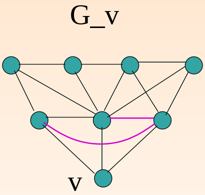
Initial reordering: description

- Needed to avoid high amount of fill-in
- Two basic types
  - local reorderings: based on a local greedy criterion
    - based on the vertex degree $deg(v) = |Adj_G(v)|$ **(minimum degree type)**
    - based on the expected fill-in **(minimum fill-in type)**
    - both of them can be **exact** or **approximate**
  - global reorderings: take into account the whole graph / matrix

Initial reordering

Initial local reordering: motivating example





- G_v is G after **symbolic** elimination on v with v removed
- Note that it is needed to eliminate (symbolically) **without** the theoretical support of Cholesky described above.

Initial reordering

Basic minimum degree algorithm

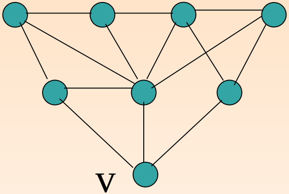
Algorithm

```
*Set the list* $V' = \emptyset$
**for** $i = 1 : n$ **do**
    *Find* $i$ *such that* $deg_G(i) = min_{j \in V(G)} deg_G(j)$
    *Add* $i$ *behind the current tail of the list* $V'$
    *Set* $G = G_i$
**end** $i$
```

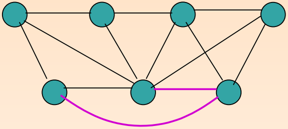
- Order of vertices in V' induces renumbering (matrix reordering)
- Obtained list V' is generally not unique.

Initial reordering

Basic minimum degree algorithm: notes



G



G_v

- The complexity hidden behind the elimination that repeatedly creates graphs of Schur complements.
- Ways to implement the minimum degree algorithm faster motivated by the following text.

Initial reordering

Multiple minimum degree algorithm: block elimination

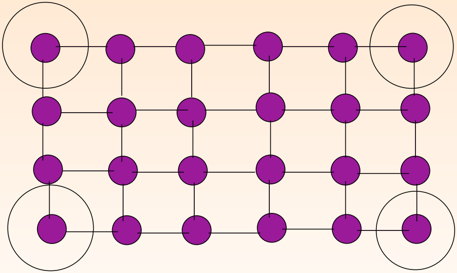
Algorithm

```
Set $V' = \emptyset$
while $G \neq \emptyset$ do
    Find all $v'_j$, $j = 1, \dots, s$ such that
      $deg_G(v'_j) = min_{v \in V(G)} deg_G(v)$ and $adj(v'_j) \cap adj(v'_k)$ for $j \neq k$
    Add all $v'_j$, $j = 1, \dots, s$ behind the current tail of $V'$
    for $j = 1 : s$ do
        $G = G_{v'_j}$
    end $i$
end while
```

- Again, order of vertices in V' induces renumbering (matrix reordering)

Initial reordering

Multiple minimum degree algorithm: demonstration



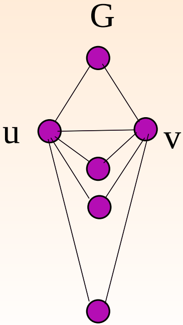
Initial reordering

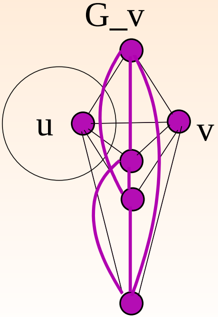
Minimum degree algorithm: indistinguishability

Definition

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

$$Adj_G(u) \cup \{u\} = Adj_G(v) \cup \{v\}. \tag{81}$$





Initial reordering

Minimum degree algorithm: indistinguishability

Lemma

Let vertices $\mathbf{u}$ *and* $\mathbf{v}$ *of* G *are indistinguishable Furthermore, let* $y \in V(G)$, $y \neq u, v$. *Then* $\mathbf{u}$ *and* $\mathbf{v}$ *are also indistinguishable in* G_y.

Proof.

Let $\mathbf{u} \notin adj_G(y)$. Then also $\mathbf{v} \notin adj_G(y)$ and the vertices are still indistinguishable since their neighbors were not influenced. Consider now $\mathbf{u}, \mathbf{v} \ \in adj_G(y)$. Then cardinalities of the adjacency sets are decreased by one but $\mathbf{u}$ and $\mathbf{v}$ still stay indistinguishable as defined above. □

Initial reordering

Minimum degree algorithm: indistinguishability

Lemma

Let the vertices **u** *and* **v** *are indistinguishable in* G*. Furthermore, let* $y \equiv \mathbf{u}$ *is a vertex of minimum degree in* G*. Then* **v** *is a vertex of minimum degree in* G_y*.*

- Consequently, two indistinguishable vertices can be eliminated **one after another**. In other words, the vertices may imply a matrix block used for joint elimination using group of two vertices.
- Easy to extend for larger groups getting the **quotient graph** (mass elimination).

Minimum degree algorithm: dominance

Definition

Vertex **v** is dominated in G by another vertex **u** of G if

$$Adj_G(u) \cup \{u\} \subseteq Adj_G(v) \cup \{v\}. \tag{82}$$

- $$\deg_G(u) \leq deg_G(v).$$

Lemma

Let **u** *dominates* **v** *in G. Furthermore, let $y \neq \mathbf{u}, \mathbf{v}$ is a minimum degree vertex in G. Then* **u** *dominates* **v** *in G_y.*

Initial reordering

Minimum degree algorithm: dominance

- To run MD we need to recompute vertex degrees:

$$v \ \notin Adj_G(y) \Rightarrow Adj_{G_y}(v) = Adj_G(v) \tag{83}$$

$$v \ \in Adj_G(y) \Rightarrow Adj_{G_y}(v) = (Adj_G(y) \cup Adj_G(v)) \setminus \{y\} \tag{84}$$

- If **u dominates v** then it is not necessary to update its degree in the sequence of Schur complement graphs until **u** is eliminated.
- Still the problem of implementation → partially eliminated graph **implicitly** expressed serves as an implementation model.

Initial reordering

Minimum degree algorithm: mass elimination model

Definition

Mass elimination graph Γ of the graph $G = (V, E)$ is a ordered triple $(\mathcal{S}, \mathcal{E}, E)$, where $\mathcal{S} \cup \mathcal{E} = V, \mathcal{S} \cap \mathcal{E} = \emptyset$ and $E \subseteq \binom{\mathcal{S}}{2} \cup \binom{\mathcal{E}(\Gamma)}{2}$ are its edges.

- Edge set $\mathcal{E}$ captures eliminated vertices.
- Edge set $\mathcal{S}$ captures non-eliminated vertices.
- Neighbors of non-eliminated vertices are found as **reachability** sets.
- Search through the reachability sets can be pruned: $\rightarrow$ **approximate minimum degree (AMD)** algorithm.

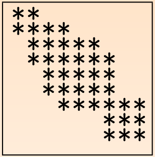
Initial reordering

Band and profile initial reorderings: shape pushers

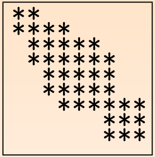
- Assume $A \in R^{n\times n}$, $i \in \{1,\ldots,n\}$ Define column positions of first nonzeros in rows by
$$r_i(A) = \min\{j|\ a_{ij} \neq 0\}. \tag{85}$$
- Analogically
$$c_j(A) = \min\{k|\ a_{kj} \neq 0\}.$$
- Define i-th **lower bandwidth**
$$\beta_i^L(A) = i - r_i(A)$$
- j-th **upper bandwidth**
$$\beta_j^U(A) = j - c_j(A)$$
- Bandwidth
$$\beta(A) = \max\{\beta_i^L(A)|\ 1 \leq i \leq n\} + \max\{\beta_i^U(A)|\ 1 \leq i \leq n\} + 1. \tag{86}$$

Initial reordering

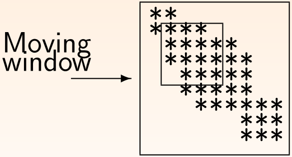
Classical local reorderings: shape pushers



Band



Profile



Frontal method - dynamic band

Initial reordering

Band and profile initial reorderings

$$\begin{pmatrix}
* & & & & & & & \\
* & * & * & & & & & \\
* & & * & & & * & & \\
& * & * & * & & & & \\
& & & * & * & * & & \\
& & & & * & * & & \\
& & & * & * & * & * & * \\
& & & & & * & * & *
\end{pmatrix} \tag{87}$$

- $r_1 = 1$, $r_2 = 1$, $r_3 = 1$, $c_1 = 1$, $c_2 = 2$, $c_3 = 2$, $\beta_1^L(A) = 0$, $\beta_2^L(A) = 1$, $\beta_3^L(A) = 2$, $\beta_7^L(A) = 3$, $\beta_6^U(A) = 2$, $\beta_7^U(A) = 0$.
- $\beta(A) = 6$.

Initial reordering

Band and profile initial reorderings

- Band

$$band(A) = \{(i,j) |\ 0 \geq i - j \leq \beta(A)\} \cup \{(j,i) |\ 0 < j - i \leq \beta(A)\}.$$

- Profile (envelope)

$$env(A) = \{(i,j) |\ 0 \geq i - j \leq \beta_i^L(A)\} \cup \{(j,i) |\ 0 < j - i \leq \beta_i^U(A)\}.$$

Initial reordering

Band and profile initial reorderings

Band Profile

$$
\begin{pmatrix}
\circledast & \circledast & \circledast & \circledast & & & & \\
\circledast & \circledast & \circledast & \circledast & \circledast & & & \\
\circledast & \circledast & \circledast & \circledast & \circledast & \circledast & & \\
\circledast & \circledast & \circledast & \circledast & \circledast & \circledast & \circledast & \\
 & \circledast & \circledast & \circledast & \circledast & \circledast & \circledast & \circledast \\
 & & \circledast & \circledast & \circledast & \circledast & \circledast & \circledast \\
 & & & \circledast & \circledast & \circledast & \circledast & \circledast \\
 & & & & \circledast & \circledast & \circledast & \circledast
\end{pmatrix}
\begin{pmatrix}
\circledast & & & & & & & \\
\circledast & \circledast & \circledast & & & & & \\
\circledast & \circledast & \circledast & & \circledast & & & \\
 & \circledast & \circledast & \circledast & \circledast & & & \\
 & & & \circledast & \circledast & \circledast & & \\
 & & & & \circledast & \circledast & & \\
 & & & \circledast & \circledast & \circledast & \circledast & \circledast \\
 & & & & & \circledast & \circledast & \circledast
\end{pmatrix}
\tag{88}
$$

Initial reordering

Band and profile initial reorderings: additional notation

Frontwidth

$$\omega_i^L(A) = |\{k | k \ > i \wedge (\exists l \leq i)(a_{kl} \neq 0)\}|. \tag{89}$$

Number of active rows at the i-th stepf in the factorization

Front

$$front(A) = \{(i,j)|\ 0 \geq i-j \leq \omega_i^L(A)\} \cup \{(j,i)|\ 0 < j-i \leq \omega_i^U(A)\}.$$

Initial reordering

Band and profile initial reorderings

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

- ω_i^L for $i = 1, \ldots, 5$ are $2, 1, 2, 1, 0$
- β_i^L for $i = 1, \ldots, 5$ are $0, 1, 0, 3, 2$.

Initial reordering

Band and profile initial reorderings

Theorem

$$Band(L + L^T) = Band(A)$$

$$Env(L + L^T) \equiv Env(F) = Env(A)$$

Initial reordering

Band and profile initial reorderings: CM algorithm

Algorithm

```
Find initial vertex $r$ and set $v_1 = r$, set $V' = \emptyset$ (queue)
Add $v_1$ as **unmarked** at the tail of $V'$
while $|V'| \neq n$ do
    Find first unmarked vertex $v_j$ at the head of $V'$
    Mark $v_j$
    Add all its neighbors not in $V'$ at its tail.
end while
```

- Only graph $G(A)$ needed → much cheaper than algorithms from MD family
- But: minimizing band/profile and **not the fill-in**.

Initial reordering

Band and profile initial reorderings: starting vertex r for CM

- Excentricity defined as

$$\epsilon(u) = \max_{v \in V} d(u, v) \tag{90}$$

- Here $d(u, v)$ is the length of the shortest path between u and v
- Maximum excentricity of a vertex in a graph is called graph **diameter**
- Good candidates for r are vertices with large excentricity.
- But finding excentricities would be expensive.

Initial reordering

Band and profile initial reorderings: starting vertex r for CM

- GPS algorithm

Definition

Level structure of graph $G = (V, E)$ is its vertex partitioning $\mathcal{L} = (L_0, \ldots, L_\lambda)$ for which $\lambda \in N,\ \lambda \geq 1$, $adj(L_i) \subseteq L_{i-1} \cup L_{i+1}$ for $i = 1, \ldots, \lambda - 1$, $adj(L_0) \subseteq L_1$ and $adj(L_\lambda) \subseteq L_{\lambda-1}$. Its **width** $\mathcal{L}$ is the number $w(\mathcal{L}) = \max_{0 \leq i \leq \lambda} |L_i|$

- Construction: Set $L_0(r) = \{r\}$, $L_i(r) = adj(\bigcup_{k=0}^{i-1} L_k(r))$ for $i = 1, \ldots, \lambda$, where $\lambda = \epsilon(r)$.
- Level structure is then $\mathcal{L}(r) = (L_0(r), \ldots, L_\lambda(r))$.

Initial reordering

Band and profile initial reorderings: CM algorithm

- Finding starting vertex r for the CM algorithm: GPS algorithm

Algorithm

```
Choose an arbitrary vertex $r$
do
    Find the level structure $\mathcal{L}(r) = (L_0(r), \ldots, L_{\lambda(r)}(r))$.
    Sort vertices in $x \in L_\lambda(r)$ by degrees non-decreasingly
    for all $x \in L_\lambda(r)$ in this order do
        Look for the level structure $\mathcal{L}(x)$ with the width $\lambda(x)$.
    end for all
    if $|\mathcal{L}(x)| > |\mathcal{L}(r)|$ then
        set $r = x$
    else
        exit do
    end if
end do
```

Initial reordering

Band and profile initial reorderings: properties

Lemma

Irreducible symmetric A ordered by the CM algorithm such that $r_i(A) < i$ for $1 < i \leq n$ has fully nonzero profile $env(F)$.

Lemma

Irreducible symmetric A ordered by the CM algorithm satisfies

$$front(A) \subseteq env(A). \tag{91}$$

Initial reordering

Band and profile initial reorderings: RCM algorithm

Algorithm

Find the list $V'' = (v_1, \ldots, v_n)$ *using CM algorithm*
Set new ordering by reversing the list items: $V' = (v_n, \ldots, v_1)$

- For CM reordered matrix A and RCM reordered matrix $\tilde{A}$ we have

$$|env(\tilde{A})| \leq |env(A)|.$$

Initial reordering

Global reorderings

Definition

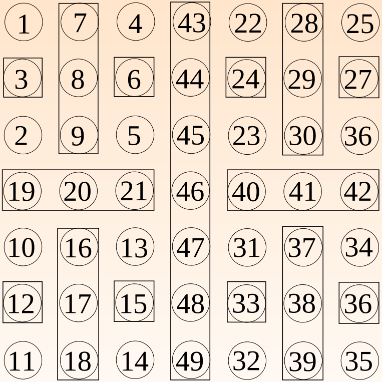
Vertex separator of an undirected $G = (V, E)$ is subset S of its vertices such that the subgraph induced by $V \setminus S$ has more components than G.

- Induced reordering

$$A = \begin{pmatrix} A_{11} & 0 & A_{31}^T \\ 0 & A_{22} & A_{32}^T \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \tag{92}$$

Initial reordering

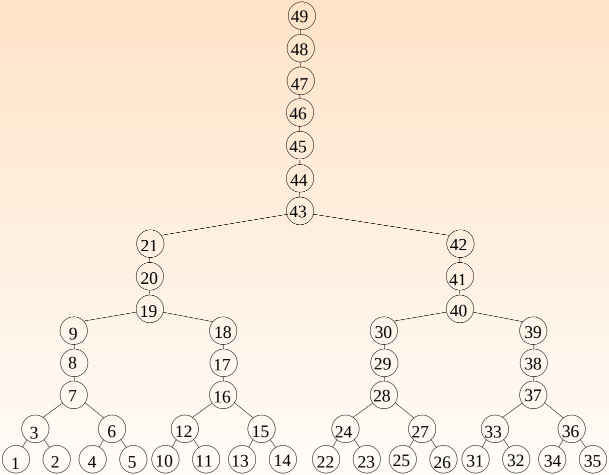
Global reorderings



1	7	4	43	22	28	25
3	8	6	44	24	29	27
2	9	5	45	23	30	36
19	20	21	46	40	41	42
10	16	13	47	31	37	34
12	17	15	48	33	38	36
11	18	14	49	32	39	35

Initial reordering

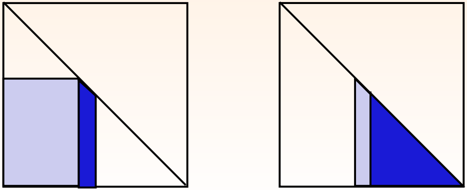
Global reorderings



Initial reordering

Complexity

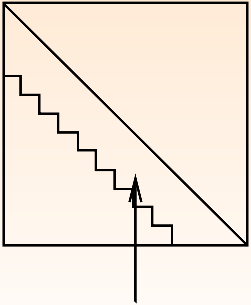
- Overall time dominated by time for the factorization
- **General dense matrices**
  - Space: $O(n^2)$
  - Time: $O(n^3)$
- **General sparse matrices**
  - Space: $\eta(L) = n + \sum_{i=1}^{n-1} (\eta(L_{*i}) - 1)$
  - Time in the i-th step: $\eta(L_{*i}) - 1$ divisions, $1/2(\eta(L_{*i}) - 1)\eta(L_{*i})$ multiple-add pairs
  - Time totally: $1/2 \sum_{i=1}^{n-1} (\eta(L_{*i}) - 1)(\eta(L_{*i}) + 2)$



Initial reordering

Complexity

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

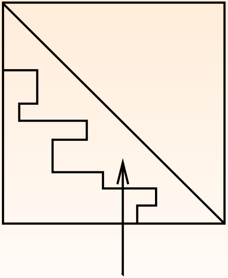


Band

Initial reordering

Complexity

- **Profile/envelope schemes**
  - Space: $\sum_{i=1}^{n} \beta_i$
  - Frontwidth: $\omega_i(A) = |\{k | k > i \wedge a_{kl} \neq 0 \ for \ some \ l \leq i\}|$
  - Time: $1/2 \sum_{i=1}^{n-1} \omega_i(A)(\omega_i(A) + 3)$

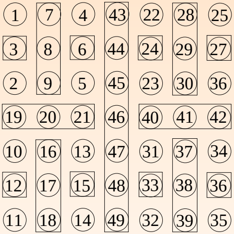


Profile (Envelope)

From direct to iterative methods

Complexity

- General sparse schemes can be analyzed in some cases
  - **Nested dissection**



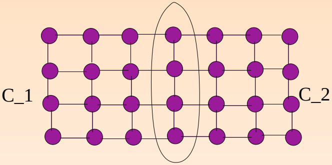
1	7	4	43	22	28	25
3	8	6	44	24	29	27
2	9	5	45	23	30	36
19	20	21	46	40	41	42
10	16	13	47	31	37	34
12	17	15	48	33	38	36
11	18	14	49	32	39	35

Definition

(α, σ) separation of a graph with n vertices: each its subgraph can be separated by a vertex separator S such that its size is of the order $O(n^{\sigma})$ and the separated subgraphs components have sizes $\leq \alpha n$,$1/2 \leq \alpha < 1$.

From direct to iterative methods

Complexity: Generalized nested dissection



Vertex separator S

- Planar graphs, 2D finite element graphs (bounded degree)
  - $\sigma = 1/2$, $\alpha = 2/3$
  - Space: $O(n \log n)$
  - Time: $O(n^{3/2})$
- 3D Finite element graphs
  - $\sigma = 2/3$
  - Space: $O(n^{4/3})$
  - Time: $O(n^2)$
- Lipton, Rose, Tarjan (1979), Teng (1997).

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Stability of sparse factorizations

Stability of LU factorization

- finite precision, $\hat{L}$ and $\hat{U}$ are computed LU factors for $A \in R^{n \times n}$, ϵ is machine precision
- Let
$$A + E = \hat{L}\hat{U}$$
- Then this factorization backward error is bounded as
$$||E||_\infty \le 2n\epsilon ||\hat{L}||_\infty ||\hat{U}||_\infty + \mathcal{O}(\epsilon^2). \tag{93}$$
- If
$$n\epsilon \ll 1 \tag{94}$$
and $\hat{x}$ is computed solution of $(A + \Delta A)\hat{x} = b$ then
$$||\Delta A||_\infty \le 6n\epsilon ||\hat{L}||_\infty ||\hat{U}||_\infty + \mathcal{O}(\epsilon^2). \tag{95}$$
- Can be formulated in other matrix norms

Stability of sparse factorizations

Stability of LU factorization

- Growth factor $\rho_{growth} \rightarrow$ conditional backward stability

$$\frac{||\hat{U}||_\infty}{||\hat{A}||_\infty}, \tag{96}$$

Stability of sparse factorizations

Stability of Cholesky factorization

- $$A + E = \hat{L}\hat{L}^T$$
- $$||E||_F \le \left(\frac{2n^{3/2}}{1 - 2n^{3/2}\epsilon} \right) \epsilon ||A||_F + \mathcal{O}(\epsilon^2). \tag{97}$$

Stability of sparse factorizations

Pivoting

- Partial pivoting: still possible the exponential growth factor

$$\rho_{growth} \le 2^{n-1} \max_{i,j} |a_{ij}|. \tag{98}$$

- Complete pivoting

$$\rho_{growth} \le \sqrt{n2\ 3^{1/2}4^{1/3}\dots n^{1/(n-1)}}\max_{i,j} |a_{ij}|. \tag{99}$$

- In addition, sparsity needs to be taken into account

Stability of sparse factorizations

Pivoting

- Markowitz number: generalization of the concept of degree using **row and column counts** in A

$$M_{ij} = (r_i - 1)(c_j - 1),\ i, j = 1, \ldots, n \tag{100}$$

$$\begin{pmatrix} 1 & 2 & 3 & & 4 \\ & 5 & & 6 & \\ 7 & & 8 & & 9 \\ & 10 & 11 & 12 & \\ 13 & 14 & & & 15 \end{pmatrix}$$

- Smallest M: $(2, 4) : (M = 1)$ and $(4, 4) : (M = 2)$
- Choosing an entry at the position $(4, 4)$ because of its magnitude.
- But the full choice can be very expensive

Stability of sparse factorizations

Threshold (partial) pivoting in LU: search in columns only

- Satisfying
$$|a_{ij}^{(k)}| \geq \mu \; max_l |a_{lj}^{(k)}| \tag{101}$$
for some $0 < \mu \leq 1$ that maximized among them the Markowitz number.
- It is possible to show that
$$max_i |a_{ij}^{(k+1)}| \leq (1 + 1/\mu) max_i |a_{ij}^{(k)}|.$$
- Modifying diagonal entries. An example is modify small diagonal entries by
$$||A|| \sqrt{\epsilon}, \tag{102}$$
- This is a way towards approximate factorizations useful to be combined with iterations.

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Symmetric indefinite factorization

- $$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$
- Standard symmetric factorization of the following matrix is unstable for small ϵ
  $$\begin{pmatrix} \epsilon & 1 \\ 1 & 0 \end{pmatrix}$$

Symmetric indefinite factorization

- An example: diagonal blocks of size 1 and 2.
- $$\begin{pmatrix}1 & 1 & 0\\ 1 & 1 & 1\\ 0 & 1 & 0\end{pmatrix} = \begin{pmatrix}1 & 0 & 0\\ 1 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} \begin{pmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{pmatrix} \begin{pmatrix}1 & 1 & 0\\ 0 & 1 & 1\\ 0 & 0 & 1\end{pmatrix}.$$

Symmetric indefinite factorization

- Why blocks of the size up to 2 and not larger?
- Either ad or $ad - bc$ has large magnitude:
- $$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = 1/(ad - bc) \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Symmetric indefinite factorization

Algorithm

One step of full indefinite pivoting by Bunch and Parlett (1971)

```
    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 $1 \times 1$ pivot (ready for $a_{kk} = 0$)
    else
        use $\begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix}$ as $2 \times 2$ pivot
    end if
```

- Full pivoting: choosing entries of largest magnitudes: can be expensive.
- But a growth factor bound can be derived (slightly worse than for LU)

Symmetric indefinite factorization: Bunch-Kaufmann

Algorithm

```
$\alpha = (1+\sqrt{17})/8 \approx 0.64, i = 1$ (possible an initial preprocessing)
Find $j \neq i$ such that $a_{ji} = \max\{|a_{ki}|, k \neq i\} =: \lambda$
if $|a_{ii}| \geq \alpha\lambda$ then
    use $a_{ii}$ as $1 \times 1$ pivot
else
    $\sigma = \max\{|a_{kj}|, k \neq j\}$
    if $|a_{ii}|\sigma \geq \alpha\lambda^2$ then
        use $a_{ii}$ as $1 \times 1$ pivot
    else if $|a_{jj}| \geq \alpha\sigma$ then
        use $a_{jj}$ as a $1 \times 1$ pivot
    else
        use $\begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix}$ as $2 \times 2$ pivot
    end if
end if
```

Symmetric indefinite factorization

- This scheme shows why Bunch-Kaufmann is useful to factorize sparse matrices
- The price for less searches expressed theoretically by large growth factor bound

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

Symmetric indefinite factorization

- What if only f is used to decide such that $|f| \geq \tau|\lambda|$?
- Again weaker bounds
- This can be practical if the processed matrix is not available in a current step (this can happen in multifrontal approaches). Implies rules like

Algorithm

```
if $|d| \geq \alpha|\lambda|$ use $d$ as $1 \times 1$ pivot
if $|d\gamma| \geq \alpha|\lambda|^2$: use $d$ as $1 \times 1$ pivot
if $|e| \geq \alpha|\gamma|$: use $e$ as $1 \times 1$ pivot
else
use $\begin{pmatrix} d & f \\ f & e \end{pmatrix}$ as $2 \times 2$ pivot
```

Symmetric indefinite factorization

$$\begin{pmatrix} d & . & f & \lambda & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ f & . & e & . & . & \gamma & . \\ \lambda & . & . & . & . & . & . \\ . & . & \gamma & . & . & . & . \\ . & . & . & . & . & . & . \end{pmatrix}$$

Symmetric indefinite factorization: rook pivoting

Algorithm

```
$\alpha = (1+\sqrt{17})/8 \approx 0.64$, $i = 1$
Find $j \neq i$ such that $a_{ji} = \max\{|a_{pi}|, p \neq i\}$
if $|a_{ii}| \geq \alpha|a_{ji}|$ a $a_{ii} \neq 0$ then
    use $a_{ii}$ as $1 \times 1$ pivot
else
    repeat
        Find $k \neq j$ such that $|a_{kj}| = \max\{|a_{pj}|, p \neq j\}$
        if $|a_{jj}|\sigma \geq \alpha|a_{kj}|$ a $a_{jj} \neq 0$ then
            use $a_{jj}$ as $1 \times 1$ pivot
        else if $|a_{ij}| = |a_{kj}|$ then
            use $\begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix}$ as $2 \times 2$ pivot
        else
            Set $i = j$ and $j = k$
        end
```

Symmetric indefinite factorization

Algorithm

Stable tridiagonal pivoting: approach by Bunch

```
    $\alpha = (\sqrt{5}-1)/2 \approx 0.62$
    Find $\sigma$: entry of maximum magnitude of $A$
    if $|a_{11}|\sigma \geq \alpha |a_{21}|^2$ then
        use $a_{11}$ as $1 \times 1$ pivot
    else
        use $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ as $2 \times 2$ pivot
   end
end
```

- Increased probability of a 1×1 pivot
- Useful to keep tridiagonal structure

Symmetric indefinite factorization

Algorithm

Pivoting Bunch-Marcia

```
$\alpha = (\sqrt{5} - 1)/2 \approx 0.62$
$\Delta = a_{11}a_{12} - a_{21}^2$
if $|\Delta| \leq \alpha |a_{11}a_{32}|$ or $|a_{21}\Delta| \leq \alpha |a_{11}^2 a_{32}|$ or $|a_{11}a_{22}| \geq \alpha a_{21}^2$ then
    use $a_{11}$ as $1 \times 1$ pivot
else
    use $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ as $2 \times 2$ pivot
end if
```

- Useful when matrix is not known in advance (Lanczos)

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Approximate factorizations, splitting and preconditioning

Why approximate factorizations?

- Direct factorizations may not be feasible (data structures and pivoting, operation counts, stability)
- Improving solution when using less accurate arithmetic (smaller ϵ) – see also modification metnioned above
- Improving solution after relaxation (e.g., in parallel computational environment)
- $\longrightarrow$ Simple iterative procedure: **iterative improvement**.
- But there are other possibilities (Krylov methods).

Approximate factorizations, splitting and preconditioning

Iterative improvement

- B is the computed factorization, $Ax^* = b$, x is a current solution
- $Bx^* = (B - A)x^* - b$
- Iterative procedure: $x^+ = (I - B^{-1}A)x + B^{-1}b$
- $\rho(I - B^{-1}A) < 1$ sufficient for the convergence

Theorem

One step *of single precision iterative refinement enough for obtaining componentwise relative backward error to the order of $O(\epsilon)$ under weaker assumptions.*
Strong bound for the **error norm** *using double precision iterative refinement.*

Approximate factorizations, splitting and preconditioning

Definition

Linear onestep stationary iterative method is a process where the relation between the two subsequent iterates $x, x^+ \in R^n$ is expressed as

$$x^+ = Sx + M^{-1}b. \tag{103}$$

$S, M \in R^{n \times n}$; M regular. Matrix S is called the **iteration matrix**.

- Briefly called **stationary iterative methods**.
- **Consistence** of an iterative method is expressed by

$$x^* = Sx^* + M^{-1}Ax^*,$$

- This implies

$$S = I - M^{-1}A,$$

  where x^* is a solution of $Ax = b$.

Approximate factorizations, splitting and preconditioning

- Another expression

$$x^+ = x - M^{-1}Ax + M^{-1}b \equiv (I - M^{-1}A)x + M^{-1}b. \tag{104}$$

- Or

$$M(x - x^+) = Ax - b. \tag{105}$$

- Different choices of M imply different iterative methods.
- Choosing M from

$$A = M - R \equiv M - (M - A) \tag{106}$$

  for some $R \in R^n$ is called a choice by **splitting** of A.
- The choice $M = I$ is sometimes called **simple iteration**.
- Matrix M can be called a **preconditioning** of the simple iteration.

Approximate factorizations, splitting and preconditioning

Definition

Stationary iterative method for solving

$$Ax = b,\, A \in R^{n\times n},\, x \in R^n,\, b \in R^n \tag{107}$$

is **convergent** if the sequence of its iterates converges to the problem solution x^* independently of the choice of the initial approximation x_0.

- Remind that the **spectral radius** of $S \in R^{n\times n}$ is given as

$$\lim_{k\to\infty} \parallel S^k \parallel^{1/k}, \tag{108}$$

- Another equivalent expression:

$$\rho(S) = \max\{|\lambda_i|\,|\,\lambda \in \sigma(A)\}, \tag{109}$$

Approximate factorizations, splitting and preconditioning

Theorem

Stationary iterative method (103) with iteration matrix S is **convergent** *iff*

$$\rho(S) < 1,$$

where $\rho(S)$ is the spectral radius of S.

Approximate factorizations, splitting and preconditioning

Preconditioning as a general transformation

- $Ax = b$, M regular
- $$M^{-1}Ax = M^{-1}b. \tag{110}$$
- $$x^+ + M^{-1}Ax = x + M^{-1}b. \tag{111}$$
- $$x^+ = (I - M^{-1}A)x + M^{-1}b, \tag{112}$$

Approximate factorizations, splitting and preconditioning

- Construct $M^{-1}A$ or not?

Desirable properties of preconditioning

- small

$$\| M - A \|$$

- small

$$\| I - M^{-1}A \| .$$

  Note that these norms may be very different
- Stable application of composed preconditioners as $M = M_1 M_2$
- Useful for the specific target computer architecture.

Approximate factorizations, splitting and preconditioning

Left, right or split preconditioning

$$\begin{aligned} M^{-1}Ax &= M^{-1}b \\ AM^{-1}y &= b,\, x = My \\ M_1^{-1}AM_2^{-1}y &= M_1^{-1}b,\, x = M_2y,\, M = M_1M_2 \end{aligned}$$

Theorem

Let ϵ and Δ are positive numbers. Then for every $n \geq 2$ there are regular matrices $A \in R^n$ and $X \in R^n$ such that all entries of $XA - I$ have magnitudes less than ϵ and all entries of $AX - I$ have magnitudes larger than Δ.

Approximate factorizations, splitting and preconditioning

Let A be SPD. Then the system preconditioned from both sides

$$L_M^{-1} A L_M^{-T} y = L_M^{-1} b,\ x = L_M^T y \tag{113}$$

where $M = L_M L_M^T$ has SPD system matrix $L_M^{-1} A L_M^{-T}$ and can be solved by the CG method.

Theorem

Consider solving $Ax = b$ with SPD preconditioning matrix M. Then

- $M^{-1}A$ *is self-adjoint in the dot product* $(.,.)_M = (M.,.)$.
- AM^{-1} *is self-adjoint in the dot product* $(.,.)_{M^{-1}} = (M^{-1}.,.)$.

Approximate factorizations, splitting and preconditioning

Proof.

$$
\begin{aligned}
(M^{-1}Ax, y)_M &= (Ax, y) \\
&= (x, Ay) \\
&= (x, MM^{-1}Ay) \\
&= (Mx, M^{-1}Ay) \\
&= (x, M^{-1}Ay)_M
\end{aligned}
$$

$$(AM^{-1}x, y)_{M^{-1}} = (AM^{-1}x, M^{-1}y) = (M^{-1}x, AM^{-1}y) = (x, AM^{-1}y)_{M^{-1}} \tag{114}$$

□

Approximate factorizations, splitting and preconditioning

Corollary

CG method preconditioned from the left based on the dot product $(.,.)_M$, *CG method preconditioned from the right based on the dot product* $(.,.)_{M^{-1}}$ *and CG method using standard dot product and preconditioned from both sides as above (113) provide in the exact arithmetic the same iterates.*

Preconditioning of a simple iteration

$$x^{+} = (I - A)x + b \tag{115}$$

- Richardson method
$$M = (1/\theta)\, I, \tag{116}$$
- Jacobi method
$$M = D_A; A = D_A - L_A - U_A$$
- Gauss-Seidel method
$$M = D_A - L_A$$

Approximate factorizations, splitting and preconditioning

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.

Approximate factorizations, splitting and preconditioning

- Still theoretical assumptions are rather strong.
- Concept of special matrices

Theorem

Matrix A is called a **regular M-matrix** *if $a_{ij} \leq 0, i \neq j$, is regular and $A^{-1} \geq 0$.*

Theorem

A is a **H-matrix** *if $B = |D_A| - |A - D_A|$ is an M-matrix.*

- Many equivalent definitions

Approximate factorizations, splitting and preconditioning

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

Approximate factorizations, splitting and preconditioning

Another approach: via simple diagonal preconditioning

Theorem

Let A be SPD. Then

$$\kappa(D_A^{-1/2} A D_A^{-1/2}) \le p\, min_{\{D \,|\, D_{ij}=0 \ pro \ i \ne j\}} \kappa(D^{-1/2} A D^{-1/2}), \qquad (117)$$

where $D_A = diag(A)$ and where p bounds row counts of A.

- Towards incomplete factorizations

Approximate factorizations, splitting and preconditioning

Algorithm (**Crout incomplete LU factorization**)

Input: *Matrix A, target sparsity pattern $\mathcal{S}\{\widetilde{F}\}$.*
Output: *Incomplete LU factorization $A \approx \widetilde{L}\widetilde{U}$.*

```
for $j = 1 : n$ do
    $\widetilde{L}_{j+1:n,j} = A_{j+1:n,j}$, $\tilde{l}_{jj} = 1$.
    $\widetilde{U}_{j,j:n} = A_{j,j:n}$
    for $k = 1 : j - 1$ such that $(j, k) \in \mathcal{S}\{\widetilde{L}\}$ do        ▷ Sparse linear combination
        $\widetilde{U}_{j,j:n} = \widetilde{U}_{j,j:n} - \tilde{l}_{jk}\widetilde{U}_{k,j:n}$
    end for
    sparsify $\widetilde{U}_{j+1,j:n}$                                              ▷ Drop entries from row $j$ of $\widetilde{U}$
    for $k = 1 : j - 1$ such that $(k, j) \in \mathcal{S}\{\widetilde{U}\}$ do        ▷ Sparse linear combination
        $\widetilde{L}_{j+1:n,j} = \widetilde{L}_{j+1:n,j} - \tilde{u}_{kj}\widetilde{L}_{j+1:n,k}$
    end for
    sparsify $\widetilde{L}_{j+1:n,j}$                                              ▷ Drop entries from column $j$ of $\widetilde{L}$
    $\widetilde{L}_{j+1:n} = \widetilde{L}_{j+1:n}\tilde{u}_{jj}^{-1}$
end for
```

Approximate factorizations, splitting and preconditioning

Algorithm (Row incomplete LU factorization)

Input: *Matrix A, target sparsity pattern $\mathcal{S}\{\widetilde{F}\}$.*
Output: *Incomplete LU factorization $A \approx \widetilde{L}\widetilde{U}$.*

```
for $i = 1 : n$ do
    $\widetilde{U}_{i,1:n} = A_{i,1:n}$
    $\widetilde{L}_{i,1:i-1} = A_{i,1:i-1}$, $\tilde{l}_{ii} = 1$.
    for $k = 1 : i-1$ such that $(i,k) \in \mathcal{S}\{\widetilde{L}\}$ do
        $\tilde{l}_{ik} = \tilde{l}_{ik} u_{kk}^{-1}$
        for $j = k+1 : i-1$ such that $(k,j) \in \mathcal{S}\{\widetilde{U}\}$ do
            $\tilde{l}_{ij} = \tilde{l}_{ij} - \tilde{l}_{ik} \tilde{u}_{kj}$
        end for
        sparsify $\widetilde{L}_{i,k+1:i-1}$
        for $j = i : n$ such that $(k,j) \in \mathcal{S}\{\widetilde{U}\}$ do
            $\tilde{u}_{ij} = \tilde{u}_{ij} - \tilde{l}_{ik} \tilde{u}_{kj}$
        end for
        sparsify $\widetilde{U}_{i,i+1:n}$
    end for
end for
```

Approximate factorizations, splitting and preconditioning

Crucial observation

Theorem

$$A = \widetilde{L}\widetilde{U} - E,$$

$$(i, j) \in \mathcal{S}(\widetilde{L} + \widetilde{U}) \Rightarrow e_{ij} = 0. \tag{118}$$

Approximate factorizations, splitting and preconditioning

Various ways to use approximations for preconditioning

- Incomplete factorizations developing the ILU (IC for SPD matrices) given above (see MILU below)
  - The sparsity pattern $\mathcal{S}$ can be given a priori
  - The sparsity pattern $\mathcal{S}$ can be found throughout
  - Additional modifications
- Incomplete inverse factorizations: direct approximation of the inverse: **avoiding substitution steps**
  - Factorized inverses (often reasonably sparse)
  - Non-factorized (dense for irreducible matrices)
- Polynomial preconditioners (M as a polynomial in A; feasibility follows for example, from Cayley-Hamilton theorem.
- More complex approaches: algebraic multigrid, domain decomposition

Approximate factorizations, splitting and preconditioning

Level-based incomplete factorization

- Define

$$level(i,j) = \min_{1\le k<\min\{i,j\}} (level(i,k) + level(k,j) + 1). \qquad (119)$$

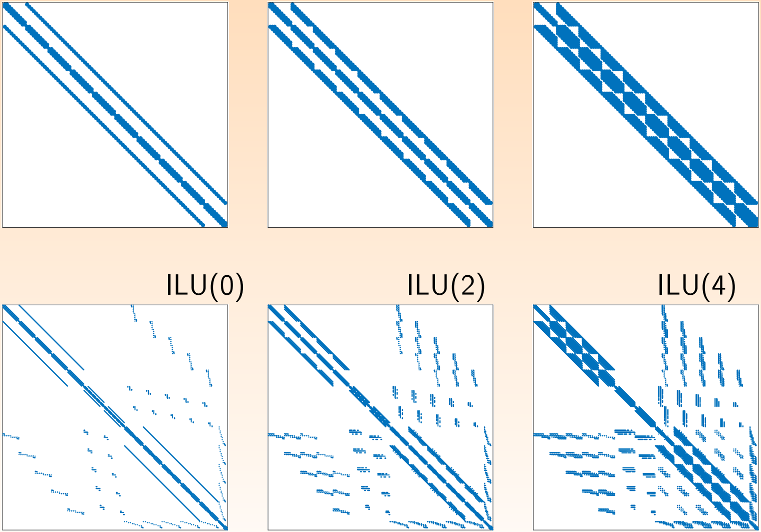
- Given $\ell \geq 0$, fill-in is permitted in the incomplete factors at position (i,j) provided $level(i,j) \leq l$. The resulting incomplete factorization is denoted by ILU(ℓ) and is called a **level-based** incomplete factorization.
- Diagonal entries of A are treated as nonzero while $level$ is set to $n+1$ for all off-diagonal zero entries in A. Levels of all other entries are set to zero.

Approximate factorizations, splitting and preconditioning

Algorithm (Level-based row incomplete LU factorization)

```
Initialize $level$ to 0 for nonzeros and diagonal entries of $A$ and to $n+1$ otherwise.
for $i = 1 : n$ do
    $\widetilde{U}_{i,i:n} = A_{i,i:n}$
    $\widetilde{L}_{i,1:i-1} = A_{i,1:i-1}$, $\tilde{l}_{ii} = 1$.
    for $k = 1 : i-1$ such that $level(i,k) \leq \ell$ do
        $\tilde{l}_{ik} = \tilde{l}_{ik} u_{kk}^{-1}$
        for $j = k+1 : i-1$ do
            $\tilde{l}_{ij} = \tilde{l}_{ij} - \tilde{l}_{ik} \tilde{u}_{kj}$
            $level(i,j) = \min(level(i,j), level(i,k) + level(k,j) + 1)$
        end for
        sparsify $\widetilde{L}_{i,k+1:i-1}$
        for $j = i : n$ do
            $\tilde{u}_{ij} = \tilde{u}_{ij} - \tilde{l}_{ik} \tilde{u}_{kj}$
            $level(i,j) = \min(level(i,j), level(i,k) + level(k,j) + 1)$
        end for
        sparsify $\widetilde{U}_{i,i:n}$
    end for
    for $k = 1 : i-1$ do
        if $level(i,k) > \ell$ then
            $l_{ik} = 0$
```

Approximate factorizations, splitting and preconditioning



ILU(0) ILU(2) ILU(4)

ILU(0) ILU(2) ILU(4)

Figure: Sparsity patterns for $ILU(\ell)$ factorizations for discretized Poi

Approximate factorizations, splitting and preconditioning

Theorem

Consider the ILU(ℓ) factorization $A \approx \widetilde{L}\widetilde{U}$. $level(i, j) = k$ for some $k \leq \ell$ if and only if there is a shortest fill path $i \Rightarrow j$ of length $k + 1$ in $\mathcal{G}(A)$.

- Theorem can be used to find the sparsity patterns of $\widetilde{L}$ and $\widetilde{U}$.

Approximate factorizations, splitting and preconditioning

Algorithm (Find the sparsity pattern of row i of $\widetilde{U}$.)

Input: *Matrix A. Level parameter $\ell \geq 0$. Row index i.*
Output: *Sparsity pattern $\widetilde{U}_{i,i:n}$ of the $ILU(\ell)$ factorization $A = \widetilde{L}\widetilde{U}$.*

```
Set $Q$ to be an empty queue
$inject(Q, i)$                                                     ▷ add $i$ to the queue
Initialise: set $length(i) = 0$, $visited(i) = i$ ▷ $length$ and $visited$ are arrays of length $n$
while $Q$ is not empty do
    $pop(Q, h)$                                                    ▷ take $h$ from the queue
    for $t \in adj_{\mathcal{G}(A)}(h)$ with $visited(t) \neq i$ do
        Set $visited(t) = i$
        if $t < i$ and $length(h) < \ell$ then
            $inject(Q, t)$                                         ▷ add $t$ to the queue
            $length(t) = length(h) + 1$
        else if $t > i$ then
            insert $t$ into the sparsity pattern of set of $\widetilde{U}_{i,i:n}$
        end if
    end for
end while
```

Approximate factorizations, splitting and preconditioning

The idea behind **modified incomplete factorizations** (MILU) is to maintain equality between the row sums of A and $\widetilde{L}\widetilde{U}$, that is, $\widetilde{L}\widetilde{U}e = Ae$, where e is the vector of all ones.

Algorithm (Submatrix formulation of MILU)

```
Set $\widetilde{L}$ to $I$ plus the strictly lower triangular part of $A$ and $\widetilde{U}$ to the upper triangular part of $A$
for $k = 1 : n - 1$ do
    for $i = k + 1 : n$ such that $(i, k) \in \mathcal{S}\{\widetilde{L}\}$ do
        $\tilde{l}_{ik} = a_{ik}\tilde{u}_{kk}^{-1}$
        for $j = i : n$ such that $(k, j) \in \mathcal{S}\{\widetilde{F}\}$ do
            if $(i, j) \in \mathcal{S}\{\widetilde{F}\}$ then
                $\tilde{u}_{ij} = \tilde{u}_{ij} - \tilde{l}_{ik}\tilde{u}_{kj}$
            else
                $\tilde{u}_{ii} = \tilde{u}_{ii} - \tilde{l}_{ik}\tilde{u}_{kj}$          ▷ Modify diagonal instead of creating fill-in
            end if
        end for
        for $j = k + 1 : i - 1$ such that $(k, j) \in \mathcal{S}\{\widetilde{F}\}$ do
            if $(i, j) \in \mathcal{S}\{\widetilde{F}\}$ then
                $\tilde{l}_{ij} = \tilde{l}_{ij} - \tilde{l}_{ik}\tilde{u}_{kj}$
            else
                $\tilde{u}_{ii} = \tilde{u}_{ii} - \tilde{l}_{ik}\tilde{u}_{kj}$          ▷ Modify diagonal instead of creating fill-in
            end if
        end for
```

Approximate factorizations, splitting and preconditioning

Theorem

Consider the MILU factorization $A + E = \widetilde{L}\widetilde{U}$ *with sparsity pattern* $\mathcal{S}\{\widetilde{F}\}$*. Then the off-diagonal entries* a_{ij} *of* A *for which* $(i, j) \in \mathcal{S}\{\widetilde{F}\}$ *are exactly approximated by the entries of* $\widetilde{L}\widetilde{U}$*.*

Theorem

Let A *be from a discretized Poisson problem on a uniform two-dimensional rectangular grid with the Dirichlet boundary conditions and discretization parameter* h*. Then the condition number* $\kappa((\widetilde{L}\widetilde{U})^{-1}A)$ *for the level-based MIC(0) preconditioner is* $O(h^{-1})$*.*

Approximate factorizations, splitting and preconditioning

Next algorithm describes the row ILU factorization with dynamic modification that uses a parameter ρ_{aj} computed as the ratio $\sqrt{u_{ii}/u_{jj}}$. In practice ρ_{aj} can be chosen more generally from $0 \leq \rho_{aj} \leq 1$.

Approximate factorizations, splitting and preconditioning

Algorithm (Row incomplete LU with dynamic modification)

```
Set $\widetilde{L}$ to $I$ plus the strictly lower triangular part of $A$
Set $\widetilde{U}$ to the upper triangular part of $A$
for $k = 1 : n - 1$ do
    for $i = k + 1 : n$ such that $(i, k) \in \mathcal{S}\{\widetilde{L}\}$ do
        $\tilde{l}_{ik} = a_{ik}\tilde{u}_{kk}^{-1}$
        if $(k, j) \in \mathcal{S}\{\widetilde{F}\}$ then
            for $j = i : n$ do
                if $(i, j) \in \mathcal{S}\{\widetilde{F}\}$ then
                    $\tilde{u}_{ij} = \tilde{u}_{ij} - \tilde{l}_{ik}\tilde{u}_{kj}$
                else
                    $\rho_{aj} = \sqrt{\tilde{u}_{ii}/\tilde{u}_{jj}}$, $\tilde{u}_{ii} = \tilde{u}_{ii} + \rho_{aj}|\tilde{u}_{ij}|$, $\tilde{u}_{jj} = \tilde{u}_{jj} + |\tilde{u}_{ij}|/\rho_{aj}$, $\tilde{u}_{ij} = 0$.
                end if
            end for
        end if
        if $(k, j) \in \mathcal{S}\{\widetilde{F}\}$ then
            for $j = k + 1 : i - 1$ do
                if $(i, j) \in \mathcal{S}\{\widetilde{F}\}$ then
                    $\tilde{l}_{ij} = \tilde{l}_{ij} - \tilde{l}_{ik}\tilde{u}_{kj}$
```

Approximate factorizations, splitting and preconditioning

- A related modification called **diagonally compensated reduction** can be performed in advance.
- As it is known that the incomplete LU factorization does not break down for M-matrices, it is possible to modify A in advance by setting its positive off-diagonal entries to zero and adding their values to the diagonal of A.
- If A has been originally symmetric and positive definite, then the resulting matrix is a symmetric M-matrix since the transformation can be considered as adding a positive semidefinite matrix to A.

Approximate factorizations, splitting and preconditioning

Incomplete factorizations with intermediate memory

Algorithm (Recursively constructed $\mathcal{S}\{\tilde{F}\}$ for the incomplete LU factorization)

Input: *Matrix A.*
Output: *Target sparsity pattern $\mathcal{S}\{\widetilde{F}\}$.*

```
Set $\mathcal{S}\{\widetilde{F}\} = \mathcal{S}\{A\}$.
for $k = 1 : n-1$ do
    Denote $\mathcal{S}_k = \{i \,|\, i > k, (i,k) \in \mathcal{S}\{\widetilde{F}\}\}$
    Choose arbitrarily $S_k \subseteq \mathcal{S}_k$
    Set $\bar{S}_k = \mathcal{S}_k \setminus S_k$
    Set $\mathcal{S}\{\widetilde{F}\} = \mathcal{S}\{\widetilde{F}\} \setminus (\mathcal{S}_k \cup \mathcal{S}_k^T) \cup S_k \times S_k \cup \bar{S}_k \times S_k \cup \bar{S}_k \times S_k$.
end for
```

Approximate factorizations, splitting and preconditioning

Note that the constructed sparsity pattern is symmetric. The following Theorem 118 shows that if the matrix is symmetric and positive definite, the resulting incomplete Cholesky factorization of A is breakdown free.

Theorem

Consider the incomplete LU factorization with the target sparsity pattern determined by Algorithm 19.7 without any additional modification. Then the factorization is breakdown-free.

Approximate factorizations, splitting and preconditioning

Reordering and pivoting in incomplete factorizations

- If dropping rules involve **upper bounds** on row/column count allowed in the incomplete factors, like $lsize$ and $rsize$ then the minimum degree type **may not be a suitable choice**. In such cases, specification of nonzero counts in rows/columns of the incomplete factorization should vary among the rows.
- Incomplete factorizations allowing a **small amount of fill-in**, as the level-based factorizations with low ℓ may not be so efficient as preconditioners since the fill-in in A reordered by an ordering of the minimum degree type is typically generated irregularly throughout the steps of the complete factorization: initially less, later steps use a lot of updates.

Approximate factorizations, splitting and preconditioning

Combining complete and incomplete factorizations

- A posteriori dropping
- Mimicking row/column counts in complete factorization
- Row and column updates based on an auxiliary elimination tree

Approximate factorizations, splitting and preconditioning

Approximate inverses

Theorem

Consider an entry $\alpha_{ij}, i, j = 1, \ldots, n$ of the inverse of the matrix A. It is nonzero if and only if there is a path $i \xrightarrow{\mathcal{G}(A)} j$.

- In the other words, the nonzeros in the matrix inverse of are implied by all paths in $\mathcal{G}(A)$.
- In contrast, nonzeros in factors of the LU factorization of A are implied only by fill paths.
- In the case of the incomplete factorization based on levels are implied by fill-paths of limited lengths.

Approximate factorizations, splitting and preconditioning

Other motivations

Approximate factorizations, splitting and preconditioning

Approximate inverses based on Frobenius norm minimization

- A symmetric and positive definite.
-
$$(A+E)^{-1} = \widetilde{Z}\widetilde{Z}^T \tag{120}$$
- Constrained optimization problem

$$\min_{\mathcal{S}\{\widetilde{Z}\}\subseteq\mathcal{S}_{\widetilde{Z}}} F_W(\widetilde{Z}) = \|I-\widetilde{Z}^T L\|_W^2 = tr\left[(I-\widetilde{Z}^T L)W(I-\widetilde{Z}^T L)^T\right], \tag{121}$$

  W positive definite matrix of weights of the dimension n, L is the Cholesky factor of A and tr denotes the matrix trace operator.

Approximate factorizations, splitting and preconditioning

Approximate inverses based on Frobenius norm minimization

The objective function $F_W(\widetilde{Z})$ can be transformed

$$\begin{aligned} F_W(\widetilde{Z}) &= tr\left[(I-\widetilde{Z}^T L)W(I-\widetilde{Z}^T L)^T\right] \\ &= tr(W) - tr(\widetilde{Z}^T LW) - tr(WL^T\widetilde{Z}) + tr(\widetilde{Z}^T LWL^T\widetilde{Z}) \\ &= tr(W) - \sum_{i,j} \tilde{z}_{ji}\left[(LW)_{ji} + (WL^T)_{ij}\right] + tr(\widetilde{Z}^T LWL^T\widetilde{Z}), \end{aligned}$$

where we have used that traces of any two matrices $C=(c_{ij}), D=(d_{ij}), i,j=1,\ldots,n$ satisfy

$$tr(CD) = \sum_i \sum_j c_{ij}d_{ji}$$

. Taking into account the minimality condition to be satisfied

$$\frac{\partial F_W(\widetilde{Z})}{\partial \tilde{z}_{ji}} = 0, \ (j,i) \in \mathcal{S}_{\widetilde{Z}} \tag{122}$$

Approximate factorizations, splitting and preconditioning

Approximate inverses based on Frobenius norm minimization

we get

$$-(LW)_{ji} - (WL^T)_{ij} + (LWL^T\widetilde{Z})_{ji} + (\widetilde{Z}^T LWL^T)_{ij} = 0,\ (i,j) \in \mathcal{S}_{\widetilde{Z}^T}. \tag{123}$$

This implies

$$(\widetilde{Z}^T LWL^T)_{ij} = (WL^T)_{ij},\ (i,j) \in \mathcal{S}_{\widetilde{Z}^T}. \tag{124}$$

Setting $W = I$ we get the condition

$$(\widetilde{Z}^T A)_{ij} = (L^T)_{ij} \text{ for } (i,j) \in \mathcal{S}_{\widetilde{Z}^T}, \tag{125}$$

that has to be satisfied by the solution $F_I(\widetilde{Z})$ of (121).

- FSAI approximated inverse

Approximate factorizations, splitting and preconditioning

Algorithm (FSAI algorithm to find and approximate factorization $\widetilde{Z}^T\widetilde{Z} \approx A^{-1}$)

Input: *Symmetric and positive definite matrix A.*
Output: *An upper triangular matrix $\widetilde{Z}$.*

```
1: Solve for $\widehat{Z}$ the system
       $$(\widehat{Z}^T A)_{ij} = \delta_{ij} \text{ for } (i,j) \in \mathcal{S}_{\widetilde{Z}^T} \qquad (126)$$
2: Find $D_Z$ such that $(D_Z \widehat{Z}^T A \widehat{Z} D_Z)_{ii} = 1, i = 1, \ldots, n$    ▷ Symmetric Jacobi scaling
3: Set $\widetilde{Z} = \widehat{Z} D_Z$
```

Approximate factorizations, splitting and preconditioning

Define the reduced symmetric matrices $\hat{A}_i = A(\mathcal{J}_i, \mathcal{J}_i)$ of the dimension $|\mathcal{J}_i|$ by their entries as follows

$$(\hat{A}_i)_{ij} = \begin{cases} a_{ij} & \text{if } i = j \text{ or } (i,j) \in \mathcal{S}_{\widehat{Z}} \text{ or } (j,i) \in \mathcal{S}_{\widehat{Z}} \\ 0 & \text{otherwise} \end{cases} \tag{127}$$

Solution of the system above decouples into solving the following n systems of linear equations independently for each column $\tilde{z}_i, i = 1, \ldots, n$ of $\widetilde{Z}^T$.

$$\widehat{A}_i \tilde{z}_i = \hat{e}_{|\mathcal{J}_i|}, \tag{128}$$

Here, $\hat{e}_{|\mathcal{J}_i|}$ is the $|\mathcal{J}_i|$-th unit vector of the dimension $|\mathcal{J}_i|$ for

$$\mathcal{J}_i = \{ j \mid (j,i) \in \mathcal{S}_{\widetilde{Z}} \cup (i,i) \} \tag{129}$$

The following theorem states an existence and uniqueness of the computed $\widetilde{Z}$ exists under a weak assumption. It is instructive to present its short proof as well.

Approximate factorizations, splitting and preconditioning

Theorem

Let A be symmetric and positive definite. Let the $\mathcal{S}_{\widetilde{Z}}$ includes all diagonal positions. Then $\widetilde{Z}$ exists and it is unique.

The reduced systems (128) are symmetric and positive definite. Moreover, for any $i \in \{1, \ldots, n\}$ we get

$$(\tilde{Z}^T A \tilde{Z})_{ii} = \sum_{j \in \mathcal{J}_i} \delta_{ij} \tilde{Z}_{ji} = (\tilde{Z})_{ii} = \left[\hat{A}_i^{-1} \right]_{ii}. \qquad (130)$$

Approximate factorizations, splitting and preconditioning

Theorem

Let A be symmetric and positive definite with LL^T be its Cholesky factorization. Moreover, let $\mathcal{S}_{\widetilde{Z}}$ be an upper triangular pattern which contains positions of diagonal entries and let $\widetilde{Z}$ be obtained from FSAI algorithm with $\mathcal{S}_{\widetilde{Z}}$. Then any upper triangular matrix $T \in R^{n\times n}$ with the sparsity pattern included in $\mathcal{S}_{\widetilde{Z}}$ such that $(T^TAT)_{ii} = 1$ for $i = 1, \ldots, n$ satisfies

$$||I - \widetilde{Z}^T L||_F \leq ||I - T^T L||_F. \tag{131}$$

Approximate factorizations, splitting and preconditioning

Theorem

Assume that we have two different upper triangular patterns $\mathcal{S}_{\widetilde{Z}'}$ *and* $\mathcal{S}_{\widetilde{Z}''}$ *which both contain positions of diagonal entries. Let* $A \in R^{n \times n}$ *be a symmetric and positive definite matrix with the Cholesky factorization* LL^T*. Assume further* $S_{\widetilde{Z}'} \supseteq S_{\widetilde{Z}''}$*. Let* $\widetilde{Z}'$ *and* $\widetilde{Z}''$ *be the FSAI approximate inverses for these two patterns, respectively. Then*

$$F_I(\widetilde{Z}') \leq F_I(\widetilde{Z}''). \tag{132}$$

Approximate factorizations, splitting and preconditioning

Let A be a generally nonsymmetric and assume the factorized approximate inverse be provided in the form

$$A \approx (A+E)^{-1} = \widetilde{Z}\widetilde{D}^{-1}\widetilde{W}^T \tag{133}$$

$$(\widetilde{Z}^T A)_{ij} = \delta_{ij} \text{ for } (i,j) \in \mathcal{S}_{\widetilde{Z}^T} \tag{134}$$

$$(A\widetilde{W})_{ij} = \delta_{ij} \text{ for } (i,j) \in \mathcal{S}_{\widetilde{W}} \tag{135}$$

Then, finally, $\widetilde{D}$ is obtained as the inverse of the diagonal of the matrix

$$\widetilde{W}^T A \widetilde{Z}$$

Approximate factorizations, splitting and preconditioning

Nonfactorized approximate inverses based on Frobenius norm minimization

$$\| \, I - MA \, \| \quad \rightarrow \; \min \qquad \text{or} \qquad \| \, I - AM \, \| \quad \rightarrow \; \min \, . \tag{136}$$

$$\|I - AM\|_F^2 = \sum_{i=1}^{n} \|e_i - Am_i\|_2^2, \tag{137}$$

$$\min(\mathcal{J}_i) \equiv \min_{\hat{m}_i} \|\hat{e}_i - A(1:n, \mathcal{J}_i)\hat{m}_i\|_2 \tag{138}$$

Approximate factorizations, splitting and preconditioning

Problems above can be solved, for instance, using the dense QR factorization of $A(1:n, \mathcal{J}_i)$ which can be written as

$$A(1:n, \mathcal{J}_i) = \hat{Q} \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix}, \tag{139}$$

Then

$$m_i = \hat{R}^{-1} \hat{Q}^T e_i. \tag{140}$$

Approximate factorizations, splitting and preconditioning

Guessing $\mathcal{S}(M)$ such that the computed M well approximates the inverse of A and leads to an efficient preconditioner, is a difficult problem. A successful variant of the minimization-based approach in the Frobenius norm improves an initial guess of $\mathcal{S}(M)$ iteratively.

- Denote by $\mathcal{I}_i$ the set of nonzero rows of $A_{1:n,\mathcal{J}_i}$. Consider the residual r_i of the least squares problem (138) given by

$$r_i = e_i - A_{1:n,\mathcal{J}_i}\hat{m}_i.$$

- Denote by $\mathcal{I}_i$ the set of nonzero rows of $A_{1:n,\mathcal{J}_i}$. Consider the residual r_i of the least squares problem (138) given by

$$r_i = e_i - A_{1:n,\mathcal{J}_i}\hat{m}_i.$$

- Consider nonzero residual components of r_i with row indices outside the set $\mathcal{I}_i$ and denote their union by $\hat{\mathcal{I}}_i$. That is

$$\hat{\mathcal{I}}_i = \bigcup_{\{k \,|\, (r_i)_k \neq 0\}} k$$

$$\hat{\mathcal{J}}_i = \{l \,|\, a_{jl} \neq 0 \text{ and } j \in \hat{\mathcal{I}}_i\} \setminus \mathcal{J}_i.$$

Approximate factorizations, splitting and preconditioning

- We will call this set the set of candidate columns.
- If $\mathcal{J}_i'$ denotes the sparsity pattern $\mathcal{J}_i$ extended by an index k of a candidate column from $\hat{\mathcal{J}}_i$, then we get a minimization problem $\min(\mathcal{J}_i \cup \{k\})$ for the solution $\hat{m}_i$ extended by μ_{ki} with the column sparsity pattern $\mathcal{J}_i$ extended by k.
- In this way, performing a sequence of such minimization steps for each column of M, the final Frobenius norm may be significantly reduced. Note that in practice, solution of the extended problem $\min(\mathcal{J}_i \cup \{k\})$ does not need to be computed from scratch. Instead, it can be computed as an update of the problem $\min(\mathcal{J}_i)$.

Approximate factorizations, splitting and preconditioning

Let us discuss the ways to choose a suitable candidate column k from $\hat{\mathcal{J}}_i$ to extend $\mathcal{J}_i$.

$$\min_{\hat{m}_i,\mu_{ki}} ||e_i - A(1:n,\mathcal{J}_i)\hat{m}_i - A(1:n),k)\mu_{ki}||_2. \tag{141}$$

One possibility is to keep the computed $\hat{m}_i$ fixed and minimize only with respect to μ_{ki}. In this case, the solution of the underlying one-dimensional minimization problem is achieved for

$$\mu_{ki} = r_i^T A(1:n,k)/A(1:n,k)^T A(1:n,k)$$

and the square of the norm under minimization is given by

$$\rho_{ik} = \|r_i\|^2 - \frac{(r_i^T A(1:n,k))^2}{A(1:n,k)^T A(1:n,k)}.$$

Clearly, column k that minimizes ρ_{ik}^2 is a good candidate k to extend $\mathcal{J}_i$. Note that if $r_i \neq 0$, there is at least one such candidate.

Approximate factorizations, splitting and preconditioning

Another possibility to evaluate candidate columns that is more accurate and not much less efficient to compute is based on the full minimization in (141) without fixing the value of $\hat{m}_i$. Square norm resulting from this full minimization obtained by explicit solving of (141) is

$$\bar{\rho}_{ik} = ||r_i||_2^2 - \frac{(r_i^T A(1:n,k))^2}{||\mathbf{P}_{\mathcal{J}_i} A(1:n,k)||_2^2}, \tag{142}$$

where $\mathbf{P}_{\mathcal{J}_i}$ is an orthogonal projector onto the null space of the matrix $A(1:n,\mathcal{J}_i)^T$. Note that for the chosen i-th column of M with given sparsity pattern and a column k from the set of candidate columns we have

$$\bar{\rho}_{ik} \le \rho_{ik} \le \|r_i\|^2.$$

Approximate factorizations, splitting and preconditioning

Algorithm (SPAI algorithm to find its right approximate inverse $M \approx A^{-1}$)

Input: *Generally nonsymmetric matrix A.*
Output: *Right approximate inverse of A.*

```
for i=1:n do
    Set the initial column sparsity pattern $\mathcal{J}_i = \{j|(j,i) \in \mathcal{S}\{A\}\}$
    while Stopping criterion is not satisfied do
        Set the row sparsity pattern $\mathcal{I}_i = \{j|A(j,\mathcal{J}_i) \neq 0\}$
        Set Find the solution $\hat{m}_i$ to $\min_{\hat{m}_i} ||\hat{e}_i - \hat{A}_i\hat{m}_i||_2$
        Update the column sparsity pattern $\mathcal{J}_i$ to $\mathcal{J}_i'$ and set $\mathcal{J}_i' = \mathcal{J}_i$.
    end while
    Extend $\hat{m}_i$ of the dimension $|\mathcal{J}_i|$ to the dimension $n$ by zeros at the remaining positions
end for
Set $M = [m_1, \ldots, m_n]$
```

Approximate factorizations, splitting and preconditioning

Factorized approximate inverses based on biconjugation

- A symmetric and positive definite
- The factorized approximate inverse algorithm for A called AINV is based on orthogonalization in the inner product $\langle .,. \rangle_A$. The approach constructs the columns $z_1, \ldots, z_n$ of the unit upper triangular matrix

$$Z = [z_1, z_2, \ldots, z_n] \tag{143}$$

and the diagonal matrix D with nonzero diagonal entries $d_1, \ldots, d_n$ such that

$$Z^T A Z = D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix}. \tag{144}$$

- That is, the columns of Z for $1 \le i, j \le n$ satisfy

$$z_i^T A z_j = \begin{cases} 0 & i \ne j \\ d_i & i = j \end{cases}.$$

Approximate factorizations, splitting and preconditioning

- Since A is symmetric and positive definite, such matrices Z and D exist. The relation (144) directly reveals the inverse of A. It is easy to see that

$$A^{-1} = ZD^{-1}Z^T = \sum_{i=1}^{n} \frac{z_i z_i^T}{d_i} \tag{145}$$

Approximate factorizations, splitting and preconditioning

- Algorithmically, Z is constructed orthogonalizing in the inner product $\langle .\,, . \rangle_A$ the set of linearly independent starting vectors.
- The orthogonalization algorithm thus generalizes standard Gram-Schmidt algorithm.
- Choosing $e_1, \ldots, e_n$ as the starting vectors, the Z resulting is unit upper triangular and due to the uniqueness of the factorization $A = Z^{-T} D Z^{-1}$ we have
$$Z = L^{-T}, \tag{146}$$
where L is the lower triangular factor of the square root-free Cholesky factorization of A and D is the diagonal matrix of this factorization.

Approximate factorizations, splitting and preconditioning

Algorithm (SPD left-looking AINV algorithm to find a factorized approximate inverse of A)

Input: *Symmetric and positive definite matrix A.*
Output: *A unit upper triangular matrix $\widetilde{Z}$ and diagonal matrix $\widetilde{D}$ such that $A^{-1} \approx \widetilde{Z}^T \widetilde{D}^{-1} \widetilde{Z}$.*

```
Initiate $\widetilde{Z} = [\tilde{z}_1^{(0)}, \dots, \tilde{z}_n^{(0)}] = I$
for $j = 1 : n$ do
    for $k = 1 : j-1$ do
        Set $\tilde{\alpha}_{kj} = A_{k,1:n} \tilde{z}_j^{(k-1)} \tilde{d}_k^{-1}$
        $\tilde{z}_j^{(k)} = \tilde{z}_j^{(k-1)} - \tilde{\alpha}_{kj} \tilde{z}_k^{(k-1)}$
        sparsify $\tilde{z}_j^{(k)}$                ▷ Drop entries from the column $\tilde{z}_j^{(k)}$
    end for
    $\tilde{d}_j = A_{j,1:n} \tilde{z}_j^{(j-1)}$
end for
Set $\widetilde{Z} = [\tilde{z}_1, \dots, \tilde{z}_n] = [\tilde{z}_1^{(0)}, \dots, \tilde{z}_n^{(n-1)}]$
```

Approximate factorizations, splitting and preconditioning

Algorithm (SPD right-looking AINV algorithm to find a factorized approximate inverse of A)

Input: *Symmetric and positive definite matrix A.*
Output: *A unit upper triangular matrix $\widetilde{Z}$ and diagonal matrix $\widetilde{D}$ such that $A^{-1} \approx \widetilde{Z}^T \widetilde{D}^{-1} \widetilde{Z}$.*

```
Initiate $\widetilde{Z} = [\tilde{z}_1^{(0)}, \ldots, \tilde{z}_n^{(0)}] = I$
for $j = 1 : n$ do
    Set $\tilde{d}_j = A_{j,1:n} \tilde{z}_j^{(j-1)}$
    for $k = j + 1 : n$ do
        Set $\tilde{\alpha}_{jk} = A^T_{j,1:n} \tilde{z}_k^{(j-1)} \tilde{d}_j^{-1}$
        $\tilde{z}_k^{(j)} = \tilde{z}_k^{(j-1)} - \tilde{\alpha}_{jk} \tilde{z}_j^{(j-1)}$
        sparsify $\tilde{z}_k^{(j)}$                    ▷ Drop entries from the column $\tilde{z}_k^{(j)}$
    end for
end for
Set $\widetilde{Z} = [\tilde{z}_1, \ldots, \tilde{z}_n] = [\tilde{z}_1^{(0)}, \ldots, \tilde{z}_n^{(n-1)}]$
```

Approximate factorizations, splitting and preconditioning

Assume now that A is nonsymmetric and consider a more general AINV approach that computes the unit upper triangular matrices Z and W and the diagonal matrix D. Columns of the computed factors $z_1, \ldots, z_n$ and $w_1, \ldots, w_n$ of Z and W as well as the nonzero diagonal entries of D should satisfy

$$W^T A Z = D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix}. \tag{147}$$

Rewriting this we demand that the columns of Z and W satisfy

$$w_i^T A z_j = \left\{ \begin{array}{ll} 0 & i \neq j \\ d_i & i = j \end{array} \right. .$$

If such matrices Z and W exist, they are nonsingular and we have

$$A^{-1} = Z D^{-1} W^T = \sum_{i=1}^{n} \frac{z_i w_i^T}{d_i}. \tag{148}$$

Approximate factorizations, splitting and preconditioning

Algorithm (Nonsymmetric right-looking AINV algorithm to find a factorized approximate inverse of A)

Input: *Symmetric and positive definite matrix A.*
Output: *A unit upper triangular matrix $\widetilde{W}$, $\widetilde{Z}$ and diagonal matrix $\widetilde{D}$ such that $A^{-1} \approx \widetilde{W}^T \widetilde{D}^{-1} \widetilde{Z}$.*

```
Initiate $\widetilde{Z} = [\tilde{z}_1^{(0)}, \ldots, \tilde{z}_n^{(0)}] = I$, $\widetilde{W} = [\tilde{w}_1^{(0)}, \ldots, \tilde{w}_n^{(0)}] = I$
for $j = 1 : n$ do
    Set $\tilde{d}_j = A_{j,1:n} \tilde{z}_j^{(j-1)}$  or  $\tilde{d}_j = A_{1:n,j}^T \tilde{w}_j^{(j-1)}$
    for $k = j + 1 : n$ do
        Set $\tilde{\alpha}_{jk} = A_{j,1:n} \tilde{z}_k^{(j-1)} \tilde{d}_j^{-1}$
        $\tilde{z}_k^{(j)} = \tilde{z}_k^{(j-1)} - \tilde{\alpha}_{jk} \tilde{z}_j^{(j-1)}$
        sparsify $\tilde{z}_k^{(j)}$                    ▷ Drop entries from the column $\tilde{z}_k^{(j)}$
        $\tilde{\beta}_{jk} = A_{1:n,j}^T \tilde{w}_k^{(j-1)} \tilde{d}_j^{-1}$
        $\tilde{w}_k^{(j)} = \tilde{w}_k^{(j-1)} - \tilde{\beta}_{jk} \tilde{w}_j^{(j-1)}$
        sparsify $\tilde{w}_k^{(j)}$                    ▷ Drop entries from the column $\tilde{w}_k^{(j)}$
    end for
end for
Set $\widetilde{Z} = [\tilde{z}_1, \ldots, \tilde{z}_n] = [\tilde{z}_1^{(0)}, \ldots, \tilde{z}_n^{(n-1)}]$, $\widetilde{W} = [\tilde{w}_1, \ldots, \tilde{w}_n] = [\tilde{w}_1^{(0)}, \ldots, \tilde{w}_n^{(n-1)}]$
```

Approximate factorizations, splitting and preconditioning

Theorem

Consider Algorithm 19.11 without sparsification, that is, with the exactly computed quantities. The following identities are valid for the exactly computed quantities inside the algorithm.

$$A_{j,1:n} z_k^{(j-1)} \equiv e_j^T A z_k^{(j-1)} = (z_j^{(j-1)})^T A z_k^{(j-1)} \text{ for } \; n \geq k \geq j \geq 1. \quad (149)$$

Approximate factorizations, splitting and preconditioning

Algorithm (SPD right-looking SAINV algorithm to find a factorized approximate inverse of A)

Input: *Symmetric and positive definite matrix A.*
Output: *A unit upper triangular matrix $\widetilde{Z}$ and diagonal matrix $\widetilde{D}$ such that $A^{-1} \approx \widetilde{Z}^T \widetilde{D}^{-1} \widetilde{Z}$.*

```
Initiate $\widetilde{Z} = [\tilde{z}_1^{(0)}, \dots, \tilde{z}_n^{(0)}] = I$
for $j = 1 : n$ do
    Set $\tilde{d}_j = (\tilde{z}_j^{(j-1)})^T A \tilde{z}_j^{(j-1)}$
    for $k = j + 1 : n$ do
        Set $\tilde{\alpha}_{jk} = (\tilde{z}_j^{(j-1)})^T A \tilde{z}_k^{(j-1)} \tilde{d}_j^{-1}$
        $\tilde{z}_k^{(j)} = \tilde{z}_k^{(j-1)} - \tilde{\alpha}_{jk} \tilde{z}_j^{(j-1)}$
        sparsify $\tilde{z}_k^{(j)}$                ▷ Drop entries from the column $\tilde{z}_k^{(j)}$
    end for
end for
Set $\widetilde{Z} = [\tilde{z}_1, \dots, \tilde{z}_n] = [\tilde{z}_1^{(0)}, \dots, \tilde{z}_n^{(n-1)}]$
```

Approximate factorizations, splitting and preconditioning

Theorem

The following identities are valid for the exactly computed quantities after the k-th major step of Algorithm 19.12 was finished

$$A_{j,1:n}z_k^{(j-1)} \equiv e_j^T A z_k^{(j-1)} = (w_j^{(j-1)})^T A z_k^{(j-1)} \text{ for } \; n \geq k \geq j \geq 1. \quad (150)$$

$$A_{1:n,j}^T w_k^{(j-1)} \equiv e_j^T A^T w_k^{(j-1)} = (z_j^{(j-1)})^T A^T w_k^{(j-1)} \text{ for } \; n \geq k \geq j \geq 1. \quad (151)$$

Alternative computation of the Cholesky factorization

Consider the AINV algorithm without dropping and the LDLT factorization $A = LDL^T$ of A symmetric and positive definite with L unit lower triangular and D diagonal. The factor L and the inverse factor Z satisfy

$$AZ = LD \quad \text{or} \quad L = AZD^{-1},$$

where D is the diagonal matrix containing the pivots. Since $d_j = (z_j^{(j-1)})^T A z_j^{(j-1)}$, by equating corresponding entries of AZD^{-1} and L we find that

$$L_{kj} = \frac{(z_j^{(j-1)})^T A z_k^{(j-1)}}{(z_j^{(j-1)})^T A z_j^{(j-1)}}, \quad n \geq k \geq j \geq 1. \tag{152}$$

Approximate factorizations, splitting and preconditioning

Factorized sparse inverses can be influenced by reorderings more deeply than the nonfactored approximate inverses as SPAI, that are largely insensitive to reorderings. The following theorem describes the fill-in in the inverse of a triangular factor of the Cholesky factorization.

Theorem

Assume that A is symmetric and positive definite and LL^T is its Cholesky factorization. Sparsity pattern $\mathcal{S}(L^{-1})$ is a union of all positions (i, j) such that i is an ancestor of j in the elimination tree $\mathcal{T}(A)$.

Approximate factorizations, splitting and preconditioning

Approximate inverse by global iterations

Consider one-dimensional Newton-Raphson iterations to find a scalar value p which is the root of a given function f, that is

$$f(p) = 0.$$

The method approaches p by a sequence of approximations $p_0, p_1, \ldots$. Consider a tangent of f at p_k for some integer $k \geq 0$ in the following form

$$y = f'(p_k)p_k + b. \tag{153}$$

The tangent crosses $(p_k, f(p_k))$ and this can be put down as

$$f(p_k) = f'(p_k)p_k + b. \tag{154}$$

This implies

$$b = f(p_k) - f'(p_k)p_k \tag{155}$$

and we get a function of x given by

$$y = f'(x)x + f(p_k) - f'(p_k)p_k. \tag{156}$$

Approximate factorizations, splitting and preconditioning

Assume that the root is achieved at p_{k+1}. Then

$$0 = f'(p_{k+1})p_{k+1} + f(p_k) - f'(p_k)p_k \tag{157}$$

and therefore

$$p_{k+1} = p_k - \frac{f(p_k)}{f'(p_k)}. \tag{158}$$

For f beeing the function of the inverse given by

$$f(x) = 1/x - a$$

we have

$$p_{k+1} = p_k - \frac{1/p_k - a}{-1/p_k^2} = p_k(2 - ap_k). \tag{159}$$

Matrix generalization for finding the matrix inverse in case it is well-defined is then given by the following iterative scheme

$$G_{i+1} = G_i(2I - AG_i),\ i = 1, \ldots$$

for the sequence of non-factorized approximate inverses $G_0, \ldots$. The main problem with this approach is that the G is for irreducible A fully dense and it may be very difficult to find useful sparsity patterns for the iterates

Approximate factorizations, splitting and preconditioning

The bordering method

- Based on the following **bordering identity**
- Consider the factorization $A^{-1} = ZD^{-1}W^T$, where W and Z are unit upper triangular matrices and D is the diagonal matrix.

Then we can write in exact arithmetic for the factorization of $W^T_{1:j,1:j}A_{1:j,1:j}Z_{1:j,1:j}$

$$\begin{pmatrix} W^T_{1:j-1,1:j-1} & 0 \\ w_j^T & 1 \end{pmatrix} \begin{pmatrix} A_{1:j-1,1:j-1} & A_{1:j-1,j} \\ A_{j,1:j-1} & A_{jj} \end{pmatrix} \begin{pmatrix} Z_{1:j-1,1:j-1} & z_j \\ 0 & 1 \end{pmatrix} = \left(\begin{matrix} D_{1:j-1,} \\ 0 \end{matrix}\right. \tag{160}$$

Approximate factorizations, splitting and preconditioning

Algorithm (Nonsymmetric inverse bordering algorithm)

Input: *Generally nonsymmetric* A.
Output: *A unit upper triangular matrix* $\widetilde{Z}$ *and diagonal matrix* $\widetilde{D}$ *such that* $A^{-1} \approx \widetilde{Z}^T \widetilde{D}^{-1} \widetilde{W}^T$.

```
Set $\widetilde{Z}_1 = (1)$, $\widetilde{W}_1 = (1)$, $\widetilde{D}_1 = (a_{11})$.
for $j = 2 : n$ do
    Set $\tilde{z}_j = -\widetilde{Z}_{1:j-1,1:j-1}\widetilde{D}^{-1}_{1:j-1,1:j-1}\widetilde{W}^T_{1:j-1,1:j-1}A_{1:j-1,j}$
    Set $\tilde{w}_j = -\widetilde{W}_{1:j-1,1:j-1}\widetilde{D}^{-1}_{1:j-1,1:j-1}\widetilde{Z}^T_{1:j-1,1:j-1}A^T_{j,1:j-1}$
    Set $\tilde{d}_j = A_{jj} + A^T_{1:j-1,j}\tilde{w}_j + A_{j,1:j-1}\tilde{z}_j + \tilde{w}_j^T A_{j-1}\tilde{z}_j$
    Set $\widetilde{Z}_j = \begin{pmatrix} \widetilde{Z}_{1:j-1,1:j-1} & \tilde{z}_j \\ 0 & 1 \end{pmatrix}$
    Set $\widetilde{W}_j = \begin{pmatrix} \widetilde{W}_{1:j-1,1:j-1} & \tilde{w}_j \\ 0 & 1 \end{pmatrix}$
end for
Set $\widetilde{Z} = \widetilde{Z}_n, \widetilde{W} = \widetilde{W}_n, \widetilde{D} = \widetilde{D}_n$
```

Approximate factorizations, splitting and preconditioning

Consider the computation of the j-th diagonal entry and assume the exactly computed quantities. The computation from the formula

$$d_j = A_{jj} + A^T_{1:j-1,j} w_j + A_{j,1:j-1} z_j + w_j^T A_{j-1} z_j \tag{161}$$

can be easily replaced by the mathematically equivalent formula which we used in the algorithms using biconjugation computation:

$$d_j = A_{j,1:j-1} z_j \text{ or } d_j = A^T_{1:j-1,j} w_j. \tag{162}$$

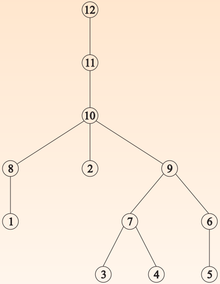
Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Basic terminology
4. Sparsity
5. Graphs and their matrices
6. From matrices to graphs
7. Graph searches
8. Notes: on matmats (matrix-matrix multiplications) in CSR/CSC formats
9. Schemes for solving systems of linear algebraic equations
10. Notes: on getting blocks in sparse matrices
11. Sparse factorization: fill-in
12. Sparse Cholesky factorization - components
13. Sparse Cholesky factorization - synthesis
14. Sparse LU factorization of generally nonsymmetric matrices
15. Sparse LU factorization of generally nonsymmetric matrices
16. Initial reordering
17. Stability of sparse factorizations
18. Symmetric indefinite factorization

Decomposition and computer architectures: Parallelism

1. Shared memory computers

- 1st level of parallelism: tree structure of the decomposition.
- 2nd level of parallelism: local node parallel enhancements.



- Both may/should be coordinated.
- **Tree parallelism** potential decreases towards its root.
- Potential for the local parallelism (larger dense matrices) increases towards the root.

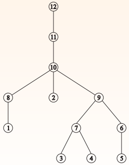
Decomposition and computer architectures: 1st level of parallelism

Two basic possibilities for the tree parallelism

- **Dynamic** task scheduling on shared memory computers
- Direct **static** mapping: subtree to subcube

1. Dynamic task scheduling on shared memory computers

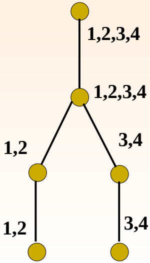
- Dynamic scheduling of the tasks
- Each processor selects a task
- Again, problem of **elimination tree reordering**
- Not easy to optimize memory, e.g., in the multifrontal method



Decomposition and computer architectures: 1st level of parallelism: II

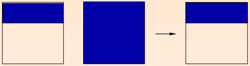
2. Direct static mapping: subtree to subcube

- Recursively map processors to the tree parts from the top
- Various ways of mapping.
- Note: In the SPD (non-pivoting) case the **arithmetic work can be computed and considered**
- Localized communication
- More difficult to share the work among processors in more complex models

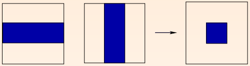


Decomposition and computer architectures: 2nd level of parallelism

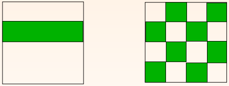
- Block Cholesky/LU factorization
- BLAS / parallel BLAS operations



1D partitioning



2D partitioning



1D and 2D block cyclic distribution

(Only illustrative figures for the talk!)

Decomposition and computer architectures: Distributed memory parallelism

Basic classical parallelization approaches (consider Cholesky)

- **Fan-in approach**
  - Demand-driven column-based algorithm
  - Required data are aggregated updates asked from previous columns
- bf Fan-out approach
  - Data-driven column-based algorithm
  - Updates are broadcasted once computed and aggregated
  - Historically the first approach; greater interprocessor communication than fan-in
- Multifrontal approach
  - Example: MUMPS